Preprint from Proceedings of 13th International Symposium on Automated Technology for Verification and Analysis (ATVA), October 2015, Shanghai, China

Improving Interpolants for Linear Arithmetic

Ernst Althaus[1,2], Björn Beber[1(✉)], Joschka Kupilas[1], and Christoph Scholl[3]

[1] Max Planck Institute for Informatics, Campus E 14, 66123 Saarbrücken, Germany
bbeber@mpi-inf.mpg.de
[2] Johannes Gutenberg University, Staudinger Weg 9, 55128 Mainz, Germany
[3] Albert-Ludwigs-Universität, Georges Köhler Allee, Gebäude 51, 79110 Freiburg im Breisgau, Germany

Abstract. Craig interpolation for satisfiability modulo theory formulas have come more into focus for applications of formal verification. In this paper we, introduce a method to reduce the size of linear constraints used in the description of already computed interpolant in the theory of linear arithmetic with respect to the number of linear constraints. We successfully improve interpolants by combining satisfiability modulo theory and linear programming in a local search heuristic. Our experimental results suggest a lower running time and a larger reduction compared to other methods from the literature.

Keywords: Craig-interpolation · Linear arithmetic · Satisfiability modulo theory · Linear programming

1 Introduction

First efficients methods were introduced for computing such interpolants for Boolean systems out of the resolution proofs from DPLL-based SAT solvers [1,2]. McMillan [2] introduced interpolants to formal verification as over-approximations of state sets.

Due to the fact that such interpolants are in general not unique, many researchers try to find *good* interpolants. The measurement for interpolants differs depending on the context. In Boolean formulas it is often the size of the representation of the interpolant, i.e. the size of the And-Inverter-Graph representing the interpolant. In the theory of linear arithmetic with rational coefficients $\mathcal{LA}(\mathbb{Q})$ the measurement of the number of linear constraints in the representation is often used, e.g. in [3] the authors showed that this measurement is beneficial for an improved generation of program invariants in software verification. This measurement is motivated by the verification of hybrid systems where operations are applied on state sets whose complexity strongly depends on the number of linear constraints, e.g. quantifier elimination in the worst-case leads to a quadratic blow-up of linear constraints after the elimination. Damm et al. showed [4] that such a measurement is the key for avoiding an explosion in the complexity of the state set representation.

© Springer International Publishing Switzerland 2015
B. Finkbeiner et al. (Eds.): ATVA 2015, LNCS 9364, pp. 48–63, 2015.
DOI: 10.1007/978-3-319-24953-7_5

In this paper we will only consider interpolants for the theory $\mathcal{LA}(\mathbb{Q})$. Therefore our measurement for *good* interpolants is the number of linear constraints. The problem of finding the best interpolant in this sense is NP-hard, which can be shown by a reduction from *k-Polyhedral Separability* [5], i.e. given two sets of points P and Q and an integer k, recognize whether there exist k hyperplanes that separate the sets P and Q through a Boolean formula. Therefore, we will here present a heuristic approach that simplifies a given interpolant. Additionally, there is no known algorithm that can compute reasonable lower bounds for this problem to measure the quality of an interpolant, so benchmarks exist where the initial interpolant is optimal, without a chance of verifying this situation.

Most approaches in this case were done by constructing an interpolant out of the resolution proof and then simplifying the proof by removing or combining theory lemmas [4,6], but maintaining the resolution proof correctly. Our approach differs in the way that we do not try to maintain the resolution proof. In fact, we only derive the set of linear constraints used in the interpolant by such proofs and then try to reduce this set in a local search heuristic. The invariant we preserve is based on an extension of Proposition 3 in [5], i.e. our set of interpolant constraints L fulfill that for every pair of points $a \in A$ and $b \in B$, there exists a linear constraint $l \in L$ such that l separates the points a and b. The proposition states that then there exists a Boolean formula, i.e. the interpolant, that separates A and B.

The paper is structured as follows: In Sect. 2 we will briefly describe previous methods to construct interpolants and the data-structure used in our benchmarks. Then we present our approach in Sect. 3, and some optimizations in Sect. 4. After that we present the evaluation of our method compared to three different other approaches in Sect. 5.

2 Preliminaries

2.1 Notations

For a point $p \in \mathbb{B}^n \times \mathbb{R}^m$, we denote the first n components, i.e. the Boolean components, by $p_{\mathbb{B}}$ and the last m, i.e. the real components, by $p_{\mathbb{R}}$. We describe a single linear constraint l by a formula $d^T x \leq d_0$ with rational coefficients $d_0, d_1, \ldots, d_m \in \mathbb{Q}^{m+1}$ over rational variables $x_1, \ldots, x_m$. For a point $p \in \mathbb{B}^n \times \mathbb{R}^m$ and a linear constraint l, we associate with $l(p)$ the Boolean variable that is true, if $d^T p_{\mathbb{R}} \leq d_0$ and false if $d^T p_{\mathbb{R}} > d_0$. We write $Dx \leq d_0$ for a conjunction of u linear constraints over rational variables $(x_1, \ldots, x_m)^T = x$ with $D \in \mathbb{Q}^{u \times m}$ and $d_0 \in \mathbb{Q}^u$.

2.2 Computing Interpolants by Resolution Proofs

A (quantifier-free) formula F in the theory of linear arithmetic with rational coefficients ($\mathcal{LA}(\mathbb{Q})$) is a Boolean combination of *atoms*. Each atom is potentially a linear constraint with rational coefficients d over a fixed set of rational variables

$x_1, \ldots, x_m$, denoted by $d^T x \leq d_0$, or a Boolean variable of a fixed set $y_1, \ldots, y_n$. A *literal* κ is either an atom or the negation of an atom. A *clause* is a disjunction of literals $\kappa_1 \vee \cdots \vee \kappa_s$. For a set of literals C and a formula F, we denote by $C \setminus F$ the set containing the literals in C by removing all atoms occurring in F, and by $C \downarrow F$ the set of literals created from C by removing all atoms *not* occurring in F.

Definition 1 (Craig Interpolant [7]). *Let A and B be two formulas, such that $A \wedge B \vDash \bot$. A* Craig interpolant *I is a formula such that $A \vDash I$, $I \wedge B \vDash \bot$ and the uninterpreted symbols in I occur both in A and B, the free variabels in I can occur freely both in A and B.*

Typically an SMT-Solver solves such formulas stepwise, first handling the non Boolean atoms like Boolean variables and solving this formula with a SAT approach. After finding a satisfiable assignment for the Boolean variables, e.g. $\kappa_1 \wedge \cdots \wedge \kappa_s$, the solver starts a decision procedure for the rational variables, i.e. computes if there exists one assignment of x such that all inequalities in the assignment simultaneously imply their given Boolean assignment. If this is not possible, we call $\eta = \{\kappa_1, \ldots, \kappa_s\}$ a $\mathcal{LA}(\mathbb{Q})$-*conflict*, the SMT-solver then adds a subset of $\kappa_1, \wedge \cdots \wedge \kappa_s$ described as a clause called $\mathcal{LA}(\mathbb{Q})$-*lemma*, to its set of clauses and starts backtracking, this will provide the procedure from selecting the invalid assignment again. These subsets are often called *minimal infeasible subsets* and are optimized in the way that the number of literals is minimized but still maintaining the $\mathcal{LA}(\mathbb{Q})$-unsatisfiability.

Definition 2 (Resolution Proof). *Let $S = \{c_1, \ldots, c_t\}$ be a set of clauses. $P = (V_P, E_P)$ is a directed acyclic graph partitioned in inner nodes and leaves. P is a resolution proof of the unsatisfiability of $c_1 \wedge \cdots \wedge c_t$ in $\mathcal{LA}(\mathbb{Q})$, if*

1. *each leaf in P is either a clause in S or a $\mathcal{LA}(\mathbb{Q})$-lemma (corresponding to some $\mathcal{LA}(\mathbb{Q})$-conflict η);*
2. *each inner node v in P has exactly two parents v^R and v^L, such that v^R and v^L share a common variable p (*pivot *variable) in the way that $p \in v^L$ and $\neg p \in v^R$. We derive v by computing the conjunction of $v^R \wedge v^L$;*
3. *the unique root node r in P is the empty clause.*

Let $S = \{c_1, \ldots, c_t\}$ be a $\mathcal{LA}(\mathbb{Q})$-unsatisfiable set of clauses, (A, B) a disjoint partition of S, and P a proof for the unsatisfiability of S in $\mathcal{LA}(\mathbb{Q})$. Then an interpolant I for (A, B) can be constructed by the following procedure [8]:

Initiate I as a copy of P, we then change the association of the nodes accordingly.

1. For every leaf $v_P \in P$ associated with a clause in S,
   set $v_I = v_P \downarrow B$, if $v_P \in A$, and set $v_I = \top$, if $v_P \in B$.
2. For every leaf $v_P \in P$ associated with a $\mathcal{LA}(\mathbb{Q})$-conflict η,
   set $v_I = \texttt{Interpolant}(\eta \setminus B, \eta \downarrow B)$.
3. For every inner node $v_P \in P$,
   set $v_I = v_I^L \vee v_I^R$ if $v_P \notin B$, and set $v_I = v_I^L \wedge v_I^R$ if $v_P \in B$.

The unique root node r_I then represents the formula of an interpolant of A and B. Several methods are known to construct $\mathcal{LA}(\mathbb{Q})$-interpolants, e.g. [9], all have in common that they construct a single linear constraint based on the convex regions given to the function. One basic idea is to use Farkas Lemma to construct a linear constraint separating these convex regions computed by linear programming. If those calls are handled separately, this will add for each $\mathcal{LA}(\mathbb{Q})$-conflict in the proof a linear constraint with a fixed normal [6] to the interpolant, and therefore will lead to a substantial blow up in the complexity. To find *simpler* interpolants the authors in [6] try to combine those calls by enlarging their degrees of freedom, i.e. extending the $\mathcal{LA}(\mathbb{Q})$-lemmas and relaxing specific constraints in the linear program to find shared interpolants for different theory lemmas in one proof.

2.3 Computing Interpolants by DC-Removability Checks

In the description of a formula F we could ask whether it is possible to remove redundant constraints. This is not as straightforward as in the situation of convex polyhedra. A linear constraint l is redundant for a formula F if there exists a formula G that depends on the same linear constraints except of l and F and G represents the same predicates. To achieve this we introduce a "don't care set" DC for F, which consists of all Boolean configurations which are not $\mathcal{LA}(\mathbb{Q})$-sufficient, and then efficiently construct G, which holds $F = G$ for every configuration not in DC. This method was introduced by [10] and expanded in [11] to compute an interpolant of A and B. Assume we have two disjoint formulas A, B, i.e. $A \wedge B$ is unsatisfiable. We then set $F = A$ and extend the set DC from above by the set $\neg A \wedge \neg B$. By the usage of the same construction now the new formula G can differ in all regions that are not specified by either A or B. This turns the method of redundancy removal in a method of interpolation.

For both the basic approach of constructing an interpolant from a resolution proof and the reduncancy removal, the fundamental problem is that there is no freedom in the choice of linear constraints. The approach introduced in [6] is more flexible, but lacks the fact that it does not notice if a linear constraint is redundant.

2.4 Description of a State Set

A formula A can be interpreted as a specific subset of $\mathbb{B}^n \times \mathbb{R}^m$, e.g. a representation of a state set. There are several possibilities to describe state sets, e.g. the LinAIG data structure [11] where such formulas are saved as an extension of a classical And-Inverter-Graph. An And-Inverter-Graphs is a directed, acyclic graph partitioned in inner nodes and leaves. Each inner node has exactly two predecessors, representing a logical conjunction. Each leaf represents a Boolean variable $y \in \mathbb{B}^n$ or in LinAIGs a lesser-or-equal constraint over the rational variables $x \in \mathbb{Q}^m$. Additionally, the edges can contain markers indicating a logical negation. The formula A is then the association of the unique root.

This representation is also often used as a data structure for resolution proofs.

Our method does not depend on the way the state sets are described, but we assume that we can negate and intersect state sets and that we can compute whether a state set is non-empty and if so, obtain a point within the state set. All these operations are executable by common SMT solvers. Furthermore, we can easily create a set of all linear constraints used in the description of the state set.

Note that the state sets are not in disjunctive normal form, where convex regions would be easily accessible, and a transformation would not be practicable due to a potentially exponential blow-up in the description.

3 The General Algorithm

The problem we are solving can be briefly described as follows. Let two state sets A, B and an interpolant of these be given. Try to minimize the number of linear constraints used in the interpolant.

Example 1. Figure 1(*a*) shows two state sets A, B, with no Boolean complexity, i.e. $n = 0$. One representation of those sets is $A = (l_1 \wedge l_2) \vee l_3 \vee l_4$ and $B = (l_5 \wedge l_6) \vee (l_7 \wedge l_8)$. In this case, A itself is an interpolant of A and B.

Our algorithm is based on the following extension of Proposition 3 in [5].

Proposition 1. *The set of (linear) constraints $L = \{L_1, \ldots, L_h\}$ separates the state sets A and B through some Boolean formula if and only if for every pair of points $p \in A$ and $q \in B$ with $p_{\mathbb{B}} = q_{\mathbb{B}}$ there exists an i $(1 \leq i \leq h)$ such that $L_i(p) \neq L_i(q)$.*

The proof of the extension is basically the same as the one given in [5] despite two differences. Firstly, we have a finite set of (convex) regions instead of points. But since we also used L to construct these regions they behave like points, i.e. the points in such a (convex) region are constant under the associated Boolean variables. Secondly, in the extension this only holds for pairs of points in the same Boolean space, since we can distinguish the Boolean spaces by m other Boolean variables easily in a formula easily. We use this proposition in our algorithm as follows.

The algorithm iteratively improves the set L by replacing two linear constraints by one, while preserving the invariant that L satisfy Proposition 1. The linear constraints in L are called *interpolant constraints*. Every new constraint added to L can be described separately by a linear combination of constraints in A and B. This lead to the fact that if the initial L only contains constraints described over local variables of A and B, the output of our algorithm only contains such constraints, for details we refer to the linear constraints of the LP (4) and (5) in Sect. 3.4. Hence, our algorithm is a local search heuristic. In this section, we describe the test whether two linear constraints can be replaced by one and in Sect. 4, we describe some techniques to improve the performance. In particular, we describe our approach to reduce the number of pairs of linear constraints that are tested. Most approaches are of a heuristic nature.

The interpolant itself is constructed by using the method described in Sect. 2.3. We therefore deliver a sufficient set of constraints to construct the interpolant, so in the constraint minimization the part of finding redundant constraints can be skipped. This together with the fact that all interpolant constraints are defined over local variables of A and B lead to the fact that the resulting interpolant is in fact an Craig-Interpolant.

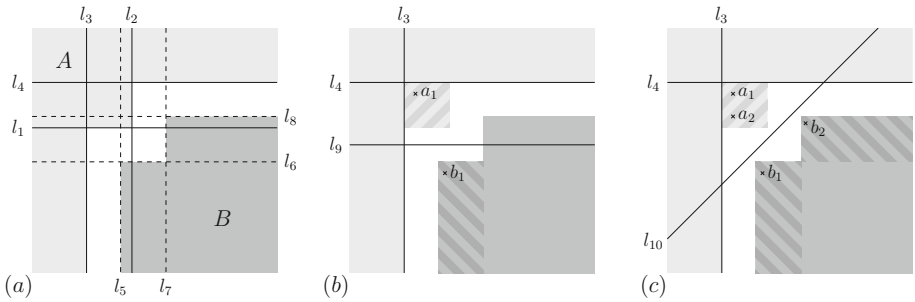

Fig. 1. Running example for our algorithm

3.1 Test Whether One Additional Linear Constraint Is Sufficient

This is the core part of the heuristic, where we determine if we can reduce the size of the interpolant. It will test if it is possible to substitute two given interpolant constraints by one new linear constraint l^*. Notice, that when removing two interpolant constraints, there will be pairs $a \in A, b \in B$ of points with $a_{\mathbb{B}} = b_{\mathbb{B}}$ that can not be distinguished by the remaining interpolant constraints L, i.e. $l(a) = l(b)$ for all $l \in L$. We will test whether all those pairs of points can be distinguished by a single new linear constraint.

Basically, we iteratively collect such pairs of points and construct a linear constraint l^* separating all pairs of points already found, until either all pairs of points can be distinguished with the additional help of l^* or no such linear constraint can be found. Figure 2 gives a sketch of the algorithm.

In order to guarantee termination, it does not satisfy to collect pairs of points, as there can be an infinite number. Therefore, we construct convex regions C_a and C_b around the points a and b described only by constraints known to the system, i.e. for C_a we only use linear constraints used in the description of A and additionally all remaining interpolant constraints. The convex sets are contained in the respective sets, i.e. $C_a \subseteq A$ and $C_b \subseteq B$. Since there are only a finite number of linear constraints in the description of the state sets and interpolant constraints, there are only a finite number of possible convex sets describable by these constraints. This lead to a termination of this part of the algorithm in a finite number of steps.

Hence, we need three sub-algorithms. One for finding pairs of points that can not be distinguished, one for constructing the convex regions around them, and one for constructing a linear constraint separating a given set of pairs of convex

regions. An algorithm for the first problem which is based on SMT is given in Sect. 3.2. A simple solution for the second problem is given in Sect. 3.3. Finally, we give a solution for the third problem based on linear programming (LP) in Sect. 3.4.

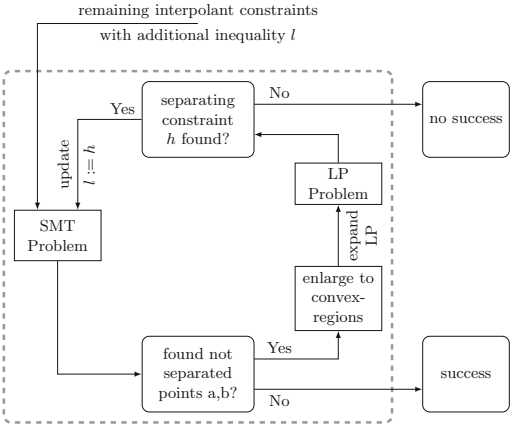

Fig. 2. Sketch of the core part of the heuristic

Example 2. As mentioned in the previous part the set of linear constraints $L = \{l_1, \ldots, l_4\}$ of A is a valid set of interpolant constraints, i.e. satisfy Proposition 1. Figure 1(*b*) and (*c*) show the solutions of the three sub-algorithms from the test, whether l_1 and l_2 can be substituted by a single new constraint. In the first iteration, shown in (*b*) we find a pair of points a_1, b_1 that are indistinguishable, once $L = \{l_3, l_4\}$, after that we compute convex regions around those points, shown as highlighted areas, and finally compute a linear constraint l_9 that separates this pair of convex regions. $L \cup \{l_9\}$ do not satisfy Proposition 1, therefore the algorithm starts a second iteration, shown in (*c*), and finally computes a linear constraint l_{10}, that simultaneously separates both pairs of convex regions found in the first and the current iteration. Finally $L \cup \{l_{10}\}$ satisfy Proposition 1 and therefore one constraint less is needed in the interpolant.

3.2 Finding Pairs of Indistinguishable Points with SMT

Notice, that we are in the situation that we have to test whether our tentative new linear constraints l^* together with the remaining linear constraints L of the interpolant satisfies to construct an interpolant between state sets A and B. The linear constraints are not sufficient if and only if we can find points $a \in A, b \in B$ with $a_{\mathbb{B}} = b_{\mathbb{B}}$ that are not distinguishable, i.e. for all $l \in L \cup \{l^*\} l(a) = l(b)$ holds.

To solve this by an SMT solver we use the following formula:

$$(a \in A) \wedge (b \in B) \wedge (a_{\mathbb{B}} = b_{\mathbb{B}}) \wedge \left(\bigwedge_{l \in L \cup \{l^*\}} l(a) = l(b) \right) \tag{1}$$

Either we found a valid solution and therefore two points a, b that are not separable by any linear constraint in $L \cup \{l^*\}$ or we found that $L \cup \{l^*\}$ is a valid set of interpolant constraints. The problem has only minor changes in every iteration, because we only substitute the old condition $l^*(p) = l^*(q)$ with an updated linear constraint l^*. This gives us the opportunity to use the advantage of incremental SMT.

3.3 Enlarge Pairs of Indistinguishable Points to Convex Regions

If we have found a pair of points a, b as described in the previous section, we want to find a set of points $C_a \subseteq A$ around a that is preferably large, such that no point in C_a can be distinguished from b and vice versa.

We achieve this by computing convex regions around a, such that all points within C_a are equal with respect to all linear constraints in $L \setminus \{l^*\}$ and all linear constraints used in the description of A.

For computing C_a test for every linear constraint l that is used in the description of A, if a satisfies this linear constraint, i.e. we compute $l(a)$. We therefore collect linear constraints in a set C. We add l to C when $l(a)$ is *true*, or $\neg l$ if $l(a)$ is *false*. After evaluating that for every linear constraint in A, we compute the same for every interpolant constraint. We do not use the current l^* in the description of the convex region, since we change this constraint in the next step, where we search for a new candidate. C_a is then computed as a conjunction of all constraints in C.

3.4 Finding a Linear Constraint Separating Pairs of Convex Regions with LP

We try to find a solution to this problem by constructing a linear program whose solution represents a separating linear constraint. The LP does not solve the problem in general, as it will fix all convex regions of A on one side and the convex regions of B on the other side, which is not necessarily required. Furthermore, it assumes that all inequalities in A and B are non-strict, i.e. convex regions are enlarged by their boundary. Both deficiencies are handled heuristically later. We use an LP-solver that handles rational arithmetic as errors in the coefficients prevent the algorithm from termination since we could be forced to separate the same pair of convex region multiple times.

The construction of the LP is similar to the one that computes the linear constraints in resolution proofs, hence based on Farkas' Lemma. We expanded the approach to separate multiple pairs of convex regions.

Therefore, we define the variables $d \in \mathbb{Q}^m$ and $d_0 \in \mathbb{Q}$ that describe the new linear constraint l^* in the form $d^T x \leq d_0$. Pairs of convex sets (A^i, B^i) for $i \in \{1, \ldots, k\}$, which are present in the k-th iteration of the LP-problem for finding a new constraint for one test described in Sect. 3.1, all constructed by the enlargement of points to convex regions described in Sect. 3.3. The constraint $d^T x \leq d_0$ is implied by A^i, if there is a non-negative linear combination of the inequalities of A^i leading to $d^T x \leq d_0$. Similarly, the constraint $a^T x > b$ is

implied by B^i, if there is a positive ε such that there is a non-positive linear combination of the inequalities of B^i leading to $d^T x \geq d_0 + \varepsilon$. All constraints can easily be formulated as linear constraints.

A detailed description of the LP is given in the appendix. If the LP is solvable and $\varepsilon > 0$ the computed linear constraint l^* separates each pair of convex regions.

Extension to Non-Closed Polyhedra. Obviously, the former statement is also true in the case that some of the inequalities are strict. When the LP is solvable but $\varepsilon = 0$, at least one of the convex regions of each set touches the inequality if all inequalities are non-strict. Hence, the LP solution does not give a linear constraint in this case, as the negation of the non-strict inequality is not implied by one convex set of B and its strict version is not implied by one convex set of A. If there are strict inequalities in the description of the convex regions, we test whether the computed linear constraint still separates the convex regions either in the non-strict or in the strict version. This is done by computing if the non-strict or the strict version is a valid choice for each pair. Only if a solution is valid for every pair it is a solution for the problem. Details are given in Section B in the appendix. Alternatively, a linear program that is forced to use a strict inequality of the right-hand side of the linear combination evaluates if d_0 can be used.

Greedy Approach. The LP is trying to separate all regions by always forcing A^i on one side of l^* and all B^i on the other side of l^*. This is more than we actually need in order to satisfy Proposition 1. So we expand our LP problem by a greedy approach as follows. Assume we find a linear constraint separating the pairs $\left(A^i, B^i\right)$ for the first $k-1$ convex pairs, but the LP does not find a solution when trying to set A^k on the one side and B^k on the other. Then we try to switch the sides of A^k and B^k, i.e. we modify the variable bounds and the constraints concerning the last added pair of regions, such that the convex regions will change the sides on l^*.

This is a greedy approach, as we only change the positions of the convex regions of the last iteration.

The LP and Proposition 2 given in Section B in the appendix can be adopted easily for this greedy approach.

4 Optimizations

Our main goal with the optimizations is to reduce the number of tested pairs of linear constraints to a reasonable amount. For this goal, we first introduce the concept of Non-Redundancy-Certificate-points (NRC-points) for interpolant constraints, which are then used to choose interesting candidates of pairs of linear constraints. The idea behind this heuristic is, that it is more likely to combine constraints when they are needed to separate the same regions. There will be situations where we will not choose the correct pair. To check the potency of the

heuristic choice of specific pairs and the overall heuristical algorithm we computed all benchmarks in Sect. 5 with the heuristical choice and by testing every pair of interpolant constraints. Furthermore, we give some other optimizations to the general approach.

NRC-Points. An NRC-Point (a, b) is a pair of points, where $a \in A$ and $b \in B$ are only distinguishable by one interpolant constraint h and are indistinguishable for every other interpolant constraint, we call (a, b) a NRC-point of h. Formally, an NRC-point (a, b) of h is a solution of the formula

$$(a \in A) \wedge (b \in B) \wedge (a_{\mathbb{B}} = b_{\mathbb{B}}) \wedge \left(\bigwedge_{l \in L \setminus h} l(a) = l(b) \right). \tag{2}$$

Since there can be more than one NRC-point of h, we first solve (2), then we compute convex regions C_a, C_b around a and b, with the method described in Sect. 3.3. After this, we solve (2) with the additional conjunction, that $(a \notin C_a) \wedge (b \notin C_b)$. This can be done multiple times, and with the advantage of an incremental SMT since we only add clauses to the formula.

It is worth mentioning that the search for the NRC-points detects if an interpolant constraint is redundant, i.e. it is not needed to fulfill Proposition 1. This is the case, when (2) is not satisfiable in the first iteration. When this occurs, we delete the inequality from the interpolant constraints. To keep the computational effort low, we only compute a maximum of three NRC-points for each interpolant constraint, this is motivated by experimental results.

Example 3. Consider Example 2, we want to compute all NRC-points for l_3. Therefore we search for points (a_3, b_3), that are a solution to the problem $(a_3 \in A) \wedge (b_3 \in B) \wedge (\bigwedge_{i \in \{1,2,4\}} l_i(a_3) = l_i(b_3))$. The pair (a_3, b_3) is shown in Fig. 3(a). The highlighted areas are again the convex regions built around the points a_3, b_3. There is no other solution that is not in the convex regions, therefore (a_3, b_3) is the only NRC-point for l_3. Additionally it is easy to see in Fig. 1, that the pair (a_1, b_1) is an NRC-point of l_1, and (a_2, b_2) an NRC-point of l_2.

The Choice of Interesting Pairs of Inequalities. After we calculated the NRC-points for every interpolant constraint, we compare the constructed convex regions around these points. A pair of inequalities (s, t) is chosen by our heuristic, if there exist two NRC-points (a_s, b_s) and (a_t, b_t) for interpolant constraints s and t, such that either a_s and a_t or b_s and b_t are equal in respect to all $l \in L \setminus \{s, t\}$.

In this case, our heuristic chooses the pair (s, t) as a pair that is promising, and therefore will be tested by the method described in Sect. 3.1.

If it was possible to improve the interpolant by testing the pair (s, t), all other constraints that where chosen to be tested with either s or t are then tested with the newly found interpolant constraint.

Example 4. Consider Example 3, we already computed NRC-points for l_1, l_2 and l_3. The heuristic for interesting pairs now compares these points, i.e. the

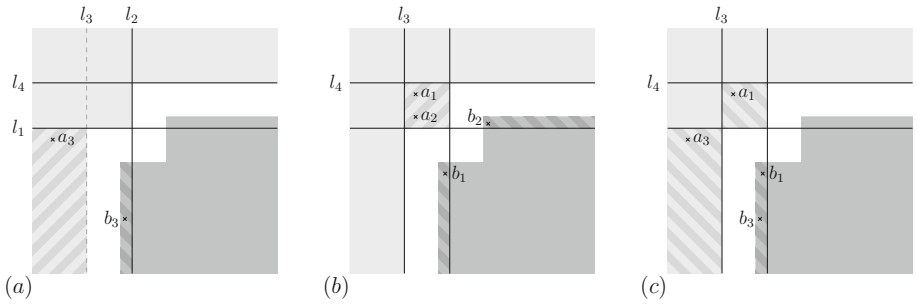

Fig. 3. Example of NRC-points and the heuristic choice of pairs.

convex regions around these points only build by interpolant constraints. In Fig. 3(*b*), we see that the convex regions around a_1 and a_2 are the same, the heuristic therefore chooses the pair l_1, l_2 as a promising candidate. The heuristic does not choose the pair l_1, l_3 since none of the convex regions are equal as we can see in Fig. 3(*c*).

Using NRC-Points to Save SMT-Calls. Assume we want to replace the interpolant constraints l_1 and l_2 by a new linear constraint l^*. The algorithm in Sect. 3.1 would first compute a pair of points, that is now not separable. With the precomputation of NRC-points, we can skip this since we already have at least two pairs of convex regions at hand that need to be distinguished by l^*.

5 Experimental Results

We implemented our algorithm in the C++ language, using Yices [12] to solve the SMT problems and QSopt-Exact [13] to solve the rational LP problems. All computations were done on a single core of an Intel I7 with 3.20 GHz, and a memory limit of 2 GB. The maximal time used for a computation of a single interpolant for a method was $\tilde{1}$5minutes, the average overall was around 2 s. The benchmarks are intermediate state sets of over 150 different model checker runs, mainly two different categories of models. We tested 62 models of the category *Flap/Slat System (FS)*, during take-off (and landing) flaps and slats of the aircraft are extended to generate more lift at low velocity and have to be retracted in time as they are not robust enough for high velocities. The models have different numbers of flaps/slats, explained in more detail in [15]. Another 90 models of the category of *ACC*, where a controllers objective is to set the acceleration of the controlled car to make it reach the goal velocity in a distance equal to or greater than the goal distance. Additionally, we tested 15 other models of other categories. Damm et al. [4] provided the model checker and models, which created the intermediate state sets. Independent of the model which is tested the intermediate state sets are divided in two different classes. In the first class, formula A describes a state set and formula B describes the

negation of a bloated version of A. This is used in the model checker, if the descriptions of the state sets become too difficult. Hence, we look for a state set with a simpler description that is slightly larger than the given state set. Due to the bloating factor this will lead to different degrees of freedom. In all our benchmarks the bloating factor for each linear constraint was set to 10 %, so every linear constraint of the bloated state set is pushed by 10 % of the total variable range of all variables used in the constraint, which is applicable, since all benchmarks are bounded in every variable. The second class comes from the so-called abstraction refinement, where specific points are excluded in a previous computed interpolant computation. In this set of benchmarks the two state sets A and B "touch" each other, i.e. they use the same linear constraint in different orientations.

All benchmarks are compared with *Constraint Minimization* [4], which basically removes redundant constraints out of the description, a version of *Simple Interpolants* [6], which creates shared constraints in the proof, and the standard interpolation method of *MathSat* [14]. To compare the quality of the solutions of *Simple interpolants* and *MathSat* we additionally executed a Constraint Minimization at the end of their computation. We were not able to compare our approach with the method described in [3], as their description of the state sets differs from ours.

We implemented four types of our approach. Approaches *g_all* and *n_all* test all pairs of linear constraints, while *g_h* and *n_h* only test pairs of interpolant constraints selected by our heuristic described in Sect. 3.4. Further *g_all* and *g_h* use the greedy approach described in Sect. 3.4, while *n_all* and *n_h* do not use it. A computed interpolant obtained by *Constraint Minimization* was used to compute the initial set of interpolant constraints L.

Table 1 shows a comparison of $g_{all}, n_{all}, g_h, n_h s$, *Simple Interpolants* (SI), *MathSat* (MS), and *Constraint Minimization* (CM).

The benchmarks are all sorted by categories (FS, ACC, other models) and classes (bloated, refinement). The key specifics of each benchmarks are given with the number instances, the number of Boolean variables n, and the number of real variables m in the table. To compare the approaches, we state the average number of linear constraints (# LC) and its variance (var. # LC). Then the relative size of the interpolant (rel. # LC) is compared to the approach of Constraint Minimization, since this method only removes redundant constraints. Additionally, we state the number of instances where the method improved the interpolant (# better), again compared to the Constraint Minimization. In the cell of the Constraint Minimization "# better" states the number of instances where no method constructed an interpolant better than the one computed by the Constraint Minimization. At last, the average runtime (time) of each method is stated.

From the table we can see that most of the benchmarks in the refinement context independent of the model could usually not be improved by any method. The distinct test where this can be seen is *ACC - refinement*. In this set of benchmarks for every of the 180 instances all methods computed an equal interpolant. We assume that this is the result of the fact that the state sets "touch" each

Table 1. Experimental results

	g_{all}	n_{all}	g_h	n_h	SI	MS	CM
FS - bloated (2255 instances, $n \in \{0,\ldots,34\}$, $m \in \{1,\ldots,3\}$)							
# LC	6.18	6.30	6.76	6.79	7.33	7.39	7.43
var. # LC	7.72	8.51	10.434	10.66	9.78	9.70	9.97
rel. # LC	0.83	0.85	0.90	0.90	0.99	1.00	1
# better	1533	1453	969	938	336	196	717
time	3.14 s	2.35 s	0.92 s	0.82 s	1.78 s	0.83 s	0.37 s
FS - refinement (915 instances, $n \in \{0,\ldots,36\}$, $m \in \{2,\ldots,3\}$)							
# LC	8.00	8.01	8.12	8.12	8.22	8.23	8.22
var. # LC	11.28	11.32	11.19	11.21	11.22	11.23	11.13
rel. # LC	0.97	0.97	0.98	0.99	1.00	1.00	1
# better	138	134	81	79	15	10	776
time	2.59 s	2.10 s	0.69 s	0.64 s	0.95 s	0.62 s	0.30 s
ACC - bloated (1575 instances, $n \in \{0,\ldots,7\}$, $m \in \{3,\ldots,5\}$)							
# LC	2.28	2.28	2.51	2.51	4.41	5.71	5.54
var. # LC	0.38	0.38	0.25	0.25	2.19	11.84	6.06
rel. # LC	0.51	0.51	0.55	0.55	0.87	1.01	1
# better	1305	1305	1305	1305	724	258	270
time	2.60 s	2.09 s	1.33 s	1.12 s	0.35 s	0.24 s	0.24 s
ACC - refinement (180 instances, $n \in \{4,\ldots,7\}$, $m \in \{7\}$)							
# LC	3	3	3	3	3	3	3
var. # LC	0	0	0	0	0	0	0
rel. # LC	1	1	1	1	1	1	1
# better	0	0	0	0	0	0	180
time	0.36 s	0.29 s	0.34 s	0.26 s	0.06 s	0.04 s	0.06 s
other models - bloated (740 instances, $n \in \{0,\ldots,31\}$, $m \in \{1,\ldots,5\}$)							
# LC	7.32	7.43	8.13	8.16	8.28	8.47	8.50
var. # LC	13.84	14.83	18.32	18.72	19.78	20.54	20.16
rel. # LC	0.87	0.88	0.95	0.95	1.00	1.00	1
# better	464	435	209	196	64	64	276
time	7.77 s	6.30 s	3.48 s	3.40 s	11.33 s	8.36 s	1.83 s
other models - refinement (96 instances, $n \in \{4,\ldots,24\}$, $m \in \{2,\ldots,3\}$)							
# LC	12.09	12.11	12.11	12.13	12.31	12.34	12.19
var. # LC	21.62	21.79	21.87	21.84	22.32	22.61	21.73
rel. # LC	0.99	0.99	0.99	0.99	1.01	1.01	1
# better	9	7	7	6	2	2	86
time	8.30 s	6.65 s	2.37 s	2.30 s	11.78 s	8.73 s	1.23 s

other and therefore reduce the degree of freedom. On the other hand, all of our algorithms were able to improve most of the bloated benchmarks. Again the obvious test where this can be seen is in the *ACC* models. There, all of our methods could improve 1305 of 1575 instances, with an overall relative improvement of 49 % for the algorithms that tested every pair of interpolant constraints, and 45 % for the two algorithms that tested only *interesting* pairs computed by our heuristic.

The experiments also indicate that our greedy approach for the LP problem is not often helpful in improving interpolants, i.e. there were only 162 of total 5761 instances where the greedy algorithm (g_{all}, g_{tps}) was better than the appropriate normal algorithm (n_{all}, n_{tps}). Overall, our algorithm n_{tps} achieves the best ratio in improving interpolants compared to the time effort, with a overall factor of $\sim$4.71 in exceeded running time and an improvement of around 20 %.

6 Conclusion and Further Research

In this paper, we showed how the number of linear constraints in interpolants for linear arithmetic can be reduced by a fair amount. The experiments showed that in the context of intermediate state sets in hybrid model checking the success of our algorithm closely related to the model in which this problem occurs. Further, we plan to improve our running times by replacing the rational LP solver by a state of the art LP solver and use rational arithmetic only to verify the feasibility of the solutions. Additionally, we want to improve our heuristic for the choice of "interesting" pairs of interpolant constraints. Furthermore, we will try to compute lower bounds for the amount of interpolant constraints needed in this context.

Acknowledgment. The results presented in this paper were developed in the context of the Transregional Collaborative Research Center 'Automatic Verification and Analysis of Complex Systems' (SFB/TR 14 AVACS) supported by the German Research Council (DFG). We worked in close coorperation with our colleagues from the 'First Order Model Checking Team' within the subproject H3 and we would like to thank W. Damm, B. Wirtz, W. Hagemann, and A. Rakow from the University of Oldenburg, U. Waldmann from the Max Planck Institute for Informatics at Saarbrücken and S. Disch from the University of Freiburg for numerous ideas and discussions

A Detailed Description of the Linear Program

Recall the variables given in Sect. 3.4. Let A^i, B^i be the convex sets of the i-th iteration, constructed by s_{A^i}, respectively s_{B^i}, conjunctions of linear constraints. Then A^i is formally defined by $A^i = \left\{x \in \mathbb{R}^m \middle| \mathcal{A}^i x \leq \alpha^i\right\}$, with $\mathcal{A}^i \in \mathbb{Q}^{m \times s_{A^i}}$ and $B^i = \left\{x \in \mathbb{R}^m \middle| \mathcal{B}^i x \leq \beta^i\right\}$, with $\mathcal{B}^i \in \mathbb{Q}^{m \times s_{B^i}}$. We additionally introduce s_{A^i} variables λ^i and s_{B^i} variables μ^i for every iteration $i \in \{1, \ldots, k\}$.

We look for an inequality that maximizes a simple measure of the distance of the constructed inequality to the convex regions. We do this by subtracting

the ε to the positive convex combination of the inequalities from A^i for l, i.e. the convex combination leads to $d^T x \leq d_0 - \varepsilon$. As we can scale any LP-solution by an arbitrary positive scalar so far, we have to normalize the solution. Therefore, we restrict the linear combination of one region to be a convex combination.

Hence, we obtain the following LP, where all linear constraints except (6) and (11) are introduced for all $i \in \{1, \ldots, k\}$:

$$\begin{aligned}
\max \quad & \varepsilon && (3)\\
\text{s.t.} \quad & (\mathcal{A}^i)^T \lambda^i = d && (4)\\
& (\mathcal{B}^i)^T \mu^i = d && (5)\\
& \sum \lambda^1 = 1 && (6)\\
& (\alpha^i)^T \lambda^i \leq d_0 - \varepsilon && (7)\\
& (\beta^i)^T \mu^i \geq d_0 + \varepsilon && (8)\\
& \lambda^i \geq 0 && (9)\\
& \mu^i \leq 0 && (10)\\
& \varepsilon \geq 0 && (11)
\end{aligned}$$

Constraints (4) and (5) force that the direction of the new constraint, described by d, is representable by the linear constraint of the convex regions. Conditions (7–11) verify that convex regions are on the right side of l^*. Condition (6) normalizes the solutions.

B Detailed Distinction for Non-Closed Polyhedra

The following proposition states when we have found a separating constraint in case of $\varepsilon = 0$.

Proposition 2. *Assume the LP (4–11) has optimal value* 0 *and let* $(\bar{d}, \bar{d_0})$ *be the solution of the LP for the variables* d *and* d_0.

1. *If for all* $i \in \{1, \ldots, k\}$ *either* $(\beta^i)^T \mu^i - d_0 > 0$ *or there exists a strict inequality* $s \neq \mathbf{0}$ *in* $\mathcal{B}^i$ *with variable* $(\mu^i)_s$ *such that* $(\mu^i)_s < 0$, *then* $\bar{a}^T x \leq \bar{d_0}$ *separates the regions.*
2. *If for all* $i \in \{1, \ldots, k\}$ *either* $(\alpha^i)^T \lambda^i - d_0 < 0$ *or there exists a strict inequality* $s \neq \mathbf{0}$ *in* $\mathcal{A}^i$ *with variable* $(\lambda^i)_s$ *such that* $(\lambda^i)_s > 0$, *then* $\bar{a}^T x < \bar{d_0}$ *separates the regions.*

The proof for this proposition is straight forward and will not be given in the paper.

References

1. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. J. Symbolic Logic **62**(3), 981–998 (1997)
2. McMillan, K.L.: Interpolation and sat-based model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
3. Albarghouthi, A., McMillan, K.L.: Beautiful interpolants. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 313–329. Springer, Heidelberg (2013)
4. Damm, W., Dierks, H., Disch, S., Hagemann, W., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. Sci. Comput. Program. **77**(10–11), 1122–1150 (2012)
5. Megiddo, N.: On the complexity of polyhedral separability. Discrete Comput. Geom. **3**(1), 325–337 (1988)
6. Scholl, C., Pigorsch, F., Disch, S., Althaus, E.: Simple interpolants for linear arithmetic. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1–6. IEEE (2014)
7. William, C.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. Symbolic Logic **22**(03), 269–285 (1957)
8. McMillan, K.L.: An interpolating theorem prover. Theoret. Comput. Sci. **345**(1), 101–121 (2005)
9. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 346–362. Springer, Heidelberg (2007)
10. Scholl, C., Disch, S., Pigorsch, F., Kupferschmid, S.: Using an SMT solver and craig interpolation to detect and remove redundant linear constraints in representations of non-convex polyhedra. In: Proceedings of the Joint Workshops of the 6th International Workshop on Satisfiability Modulo Theories and 1st International Workshop on Bit-Precise Reasoning, pp. 18–26. ACM (2008)
11. Damm, W., Disch, S., Hungar, H., Jacobs, S., Pang, J., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Exact state set representations in the verification of linear hybrid systems with large discrete state space. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 425–440. Springer, Heidelberg (2007)
12. Dutertre, B., De Moura, L.: The yices SMT solver (2006). http://yices.csl.sri.com/tool-paper.pdf
13. Applegate, D.L., Cook, W., Dash, S., Espinoza, D.G.: Exact solutions to linear programming problems. Oper. Res. Lett. **35**(6), 693–699 (2007)
14. Griggio, A.: A practical approach to satisfiability modulo linear integer arithmetic. JSAT **8**, 1–27 (2012)
15. Rakow, A.: Flap/Slat System. http://www.avacs.org/fileadmin/Benchmarks/Open/FlapSlatSystem.pdf