Preprint from Proceedings of 6th International Workshop on Satisfiability Modulo Theories, July 2008, Princeton, USA

# Using an SMT Solver and Craig Interpolation to Detect and Remove Redundant Linear Constraints in Representations of Non-Convex Polyhedra

Christoph Scholl
University of Freiburg
Georges-Köhler-Allee 51
D 79110 Freiburg, Germany
scholl@informatik.uni-freiburg.de

Stefan Disch
University of Freiburg
Georges-Köhler-Allee 51
D 79110 Freiburg, Germany
disch@informatik.uni-freiburg.de

Florian Pigorsch
University of Freiburg
Georges-Köhler-Allee 51
D 79110 Freiburg, Germany
pigorsch@informatik.uni-freiburg.de

Stefan Kupferschmid
University of Freiburg
Georges-Köhler-Allee 51
D 79110 Freiburg, Germany
skupfers@informatik.uni-freiburg.de

## ABSTRACT

We present a method which computes optimized representations for non-convex polyhedra. Our method detects so-called redundant linear constraints in these representations by using an incremental SMT solver and then removes the redundant constraints based on Craig interpolation. The approach is evaluated both for formulas from the model checking context including boolean combinations of linear constraints and boolean variables and for random trees composed of quantifiers, AND-, OR-, NOT-operators, and linear constraints produced by a generator. The results clearly show the advantages of our approach in comparison to state-of-the-art solvers.

## Categories and Subject Descriptors

I.1.1 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation—*Expressions and Their Representation, Simplification of expressions*; G.4 [**Mathematics of Computing**]: Mathematical Software—*Verification*

## Keywords

Satisfiability Modulo Theories, linear arithmetic, non-convex polyhedra, Craig interpolation, hybrid system verification

## 1. INTRODUCTION

In this paper we present an approach which uses SMT (Satisfiability Modulo Theories) solvers and Craig interpolation [3] for optimizing representations of non-convex polyhedra. Non-convex polyhedra are formed by arbitrary boolean combinations (including conjunction, disjunction and negation) of linear constraints. Non-convex polyhedra have been used to represent sets of states of hybrid systems. Whereas approaches like [12, 11] consider unions of convex polyhedra (i.e. unions of conjunctions of linear constraints) together with an explicit representation of discrete states, in [5, 4] a data structure called LinAIGs was used as a single symbolic representation for sets of states of hybrid systems with large discrete state spaces (in the context of model checking by backward analysis). LinAIGs in turn represent an extension of non-convex polyhedra by additional boolean variables, i.e. they represent arbitrary boolean combinations of boolean variables and linear constraints.

In particular, our optimization methods for non-convex polyhedra remove so-called *redundant linear constraints* from our representations. A linear constraint is called redundant for a non-convex polyhedron if and only if the non-convex polyhedron can be described without using this linear constraint. Note that an alternative representation of the polyhedron without using the redundant linear constraint may require a completely different boolean combination of linear constraints. In that sense our method significantly extends results for eliminating redundant linear constraints from convex polyhedra used by Frehse [11] and Wang [22].[1] Removing redundant linear constraints from non-convex polyhedra plays an important role especially during the elimination of quantifiers for real-valued variables in the context of model checking for hybrid systems. Previous work [4] demonstrated how already a simple preliminary version of redundancy removal can be used during Weispfennig–Loos quantifier elimination [14]: Based on

- the fact that the size of the formula produced by Weis-

[1]For convex polyhedra redundancy of linear constraints reduces to the question whether the linear constraint can be omitted in the conjunction of linear constraints without changing the represented set.

pfennig–Loos quantifier elimination of *one* real variable may grow by a factor which is linear in the number of constraints in the original formula and

- the observation that a large number of the new linear constraints which were generated during quantifier elimination was in fact redundant

we were able to show that it is essential to make use of redundancy removal (keeping the number of linear constraints in our representations as small as possible) in order to enable sequences of quantifier eliminations during model checking of non-trivial examples.

Our paper makes the following contributions:

- We present an algorithm for detecting a maximal number of linear constraints which can be removed *simultaneously*. The algorithm is based on sets of don't cares which result from inconsistent assignments of truth values to linear constraints. We show how the *detection* of sets of redundant constraints can be performed using an SMT solver. In particular we show how to use *incremental* SMT solving for detecting larger and larger sets of redundant constraints until a maximal set is obtained.

- Based on the don't care sets mentioned above we provide a detailed proof showing the correctness of the algorithm.

- We show how the information needed for *removing* redundant linear constraints can be extracted from the conflict clauses of an SMT solver. Finally, we present a novel method really performing the removal of redundant linear constraints based on this information. The method is based on Craig interpolation [3, 18, 15].

It is important to note that our method using redundancy elimination is not only applicable in the context of model checking for hybrid systems, but it provides a general method making quantifier elimination for linear arithmetic more efficient. Therefore our experimental evaluation is not only done for formulas generated during runs of the model checker from [4], but also for formulas from [9, 2] (in SMT-LIB format [19]) which consist of arbitrary boolean combinations of linear constraints, combined with quantifications of real-valued variables. For such formulas we solve two problems: First, we compute whether the resulting formula is satisfiable by any assignment of values to the free variables and secondly we do even more, we also compute a predicate over the free variables which is true for all satisfying assignments of the formula. We compare our results to the results of the automata-based tool LIRA [9, 2] (which also solves both problems mentioned above) and to the results of state-of-the-art SMT solvers Yices [7] and CVC3 [20] (which solve the first problem of checking whether the formula is satisfiable). Whereas these solvers are not restricted to the subclass of formulas we consider in this paper (and are not optimized for this subclass in the case of Yices and CVC3), our experiments show that for the subclass of formulas considered here our method is much more effective. Our results are obtained by an elaborate scheme combining several methods for keeping representations of intermediate results compact with redundancy removal as an essential component. Internally, these methods make heavy use of

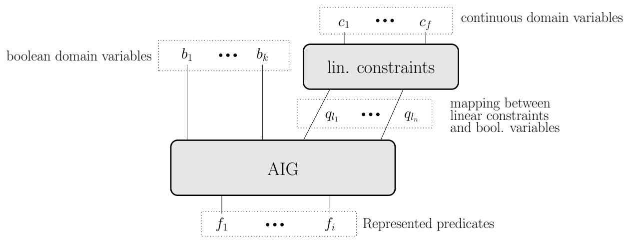


**Figure 1: The LinAIG structure**

the results of SMT solvers restricted to quantifier-free satisfiability solving.[2] Our results suggest to make use of our approach, if the formula at hand belongs to the subclass of linear arithmetic with quantification over reals and moreover, even for more general formulas, one can imagine to use our method as a fast preprocessor for simplifying subformulas from this subclass.

The paper is organized as follows: In Sect. 2 we give a brief review of our representations of non-convex polyhedra, Craig interpolation, and Weispfennig-Loos quantifier elimination. In Sect. 3 we give a definition of redundant linear constraints and present methods for detecting and removing them from representations of non-convex polyhedra. After presenting our encouraging experimental results in Sect. 4 we conclude the paper in Sect. 5.

# 2. PRELIMINARIES

## 2.1 Representation of Non-Convex Polyhedra

We assume disjoint sets of variables $C$ and $B$. The elements of $C = \{c_1, \ldots, c_f\}$ are continuous variables, which are interpreted over the reals $\mathbb{R}$. The elements of $B = \{b_1, \ldots, b_k\}$ are boolean variables and range over the domain $\mathbb{B} = \{0, 1\}$. When we consider logic formulas over $B \cup C$, we restrict terms over $C$ to the class of linear terms of the form $\sum \alpha_i c_i + \alpha_0$ with rational constants $\alpha_i$ and $c_i \in C$. Predicates are given by the set $\mathcal{L}(C)$ of linear constraints, they have the form $t \sim 0$, where $\sim \in \{=, <, \leq\}$ and $t$ is a linear term. $\mathcal{P}(C)$ is the set of all boolean combinations of linear constraints over $C$, the formulas from $\mathcal{P}(C)$ represent *non-convex polyhedra* over $\mathbb{R}^f$. In this paper we consider the class of formulas from $\mathcal{P}(B, C)$ which is the set of all boolean combinations of boolean variables from $B$ and linear constraints over $C$.

As a underlying data structure for our method we use representations of formulas from $\mathcal{P}(B, C)$ by LinAIGs [5, 4]. LinAIGs are And-Inverter-Graphs (AIGs) enriched by linear constraints. The structure of LinAIGs is illustrated in Fig. 1.

The component of LinAIGs representing boolean formulas consists in a variant of AIGs, the so–called Functionally Reduced AND-Inverter Graphs (FRAIGs) [16, 17]. AIGs enjoy a widespread application in combinational equivalence checking and Bounded Model Checking (BMC). They are basically boolean circuits consisting only of AND gates and inverters. In [17] FRAIGs were tailored towards the representation and manipulation of sets of states in symbolic model checking, replacing BDDs as a compact representation of large discrete state spaces.

[2]In our implementation we use Yices [7] and HySAT [10] for this task.

In LinAIGs (see Fig. 1) we use a set of new (boolean) *constraint variables* $Q$ as encodings for the linear constraints, where each occurring $\ell_i \in \mathcal{L}(C)$ is encoded by some $q_{\ell_i} \in Q$. In order to keep the representation compact, we avoid to represent equivalent predicates by different LinAIG nodes. Basically, this could be achieved by checking all pairs of nodes for equivalence (taking the interpretation of constraint variables $q_{\ell_i}$ by the corresponding linear constraints $\ell_i$ into account). This check can be performed by an SMT (SAT modulo theories) solver which combines DPLL with linear programming as a decision procedure. Instead of using SMT solver calls for all pairs of nodes, we make use of a carefully designed and tested strategy which avoids SMT solver calls when non-equivalence can be shown using test vectors with valuations $\mathbf{c} \in \mathbb{R}^f$ or when equivalence can be proven already for the boolean abstraction without referring to the definition of the constraint variables. (In this context boolean reasoning is supported by (approximate) knowledge on linear constraints like implications between constraints.)

## 2.2 Craig interpolation

As we will describe in Sect. 3.3 we remove linear constraints which were found to be redundant using Craig interpolation [3, 18]. Recently, Craig interpolation was applied by McMillan for generating an overapproximated image operator to be used in connection with Bounded Model Checking [15] or by Lee et al. for computing a so-called dependency function in logic synthesis algorithms [13]. A Craig interpolant is computed for a boolean formula $F$ in Conjunctive Normal Form (CNF) (i.e. for a conjunction of disjunctions of boolean variables) which is partitioned into two parts $A$ and $B$ with $F = A \wedge B$. When $F = A \wedge B$ is unsatisfiable, a Craig interpolant for the pair $(A, B)$ is a boolean formula $P$ with the following properties:

- $A$ implies $P$,
- $P \wedge B$ is unsatisfiable, and
- $P$ depends only on common variables of $A$ and $B$.

An appropriate Craig interpolant $P$ can be computed based on a proof by resolution that $F$ is unsatisfiable (time and space for this are linear in the size of the proof) [18, 15]. Proofs of unsatisfiability can be computed by any modern SAT solver with proof logging turned on.

## 2.3 Quantifier elimination

In [4] Loos's and Weispfenning's test point method [14] was adapted to the LinAIG data structure described above. The method eliminates universal quantifiers by converting them into finite conjunctions and existential quantifiers by converting them into finite disjunctions. The subformulas to be combined by conjunction (or disjunction in case of existential quantification) are obtained from the original formula by replacing real-valued variables by appropriate 'test points' arriving again at formulas in linear arithmetic. The test point method is well-suited for our LinAIG data structure, since substitutions and disjunctions / conjunctions can be performed efficiently in the LinAIG package and the method does not need (potentially costly) conversions of the original formula into CNF / DNF before applying quantifier elimination as the Fourier-Motzkin algorithm, e.g..

The number of test points needed depends linearly on the number of linear constraints in the original formula. Thus, during elimination of one real-valued variable, the number of linear constraints may grow quadratically with the number of linear constraints in the original formula. For sequences of quantifier eliminations it is therefore important to keep the number of linear constraints as small as possible. For this reason we developed an algorithm which computes representations depending on a minimal set of linear constraints. The method is presented in Sect. 3. Experimental results in Sect. 4 show that the method is indeed essential in order to enable sequences of quantifier eliminations.

# 3. REDUNDANT LINEAR CONSTRAINTS

In this section we present our methods to detect and remove redundant linear constraints from non-convex polyhedra.

For illustration of redundant linear constraints see Fig. 2 and 3, which show a typical example stemming from a model checking application. It represents a small state set based on two real variables: Lines in Figures 2 and 3 represent linear constraints, and the gray shaded area represents the space defined by some boolean combination of these constraints. Whereas the representation depicted in Fig. 2 contains 24 linear constraints, a closer analysis shows that an optimized representation can be found using only 15 linear constraints as depicted in Fig. 3.

## 3.1 Redundancy Detection and Removal for Convex Polyhedra

Note that our redundancy detection and removal approach works for representations of *non-convex* polyhedra. Therefore the task is not as straightforward as for other approaches such as [12, 11] which represent sets of *convex* polyhedra, i. e., sets of conjunctions $\ell_1 \wedge \ldots \wedge \ell_n$ of linear constraints. If one is restricted to convex polyhedra, the question whether a linear constraint $\ell_1$ is redundant in the representation reduces to the question whether $\ell_2 \wedge \ldots \wedge \ell_n$ represents the same polyhedron as $\ell_1 \wedge \ldots \wedge \ell_n$, or equivalently, whether $\overline{\ell_1} \wedge \ell_2 \wedge \ldots \wedge \ell_n$ represents the empty set. This question can simply be answered by a linear program solver.

## 3.2 Detection of Redundant Constraints for Non-convex Polyhedra

Now we consider the case of non-convex polyhedra. To be more precise, we actually consider the slightly generalized case of boolean combinations of linear constraints and additional boolean variables instead of non-convex polyhedra. Our approach works (regardless of the boolean variables) for predicates $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$ where $\ell_1, \ldots, \ell_n$ are linear constraints over $C$, $b_1, \ldots, b_k$ are boolean variables, and $F$ is an arbitrary boolean function. Such predicates may be represented by LinAIGs, e.g..

*Definition 1.* The linear constraints $\ell_1, \ldots, \ell_r$ $(1 \le r \le n)$ are called *redundant* in the representation of $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$ iff there is a boolean function $G$ with the property that $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$ and $G(\ell_{r+1}, \ldots, \ell_n, b_1, \ldots, b_k)$ represent the same predicates.

In the following we will first prove a theorem which gives a necessary and sufficient condition for a subset $\{\ell_1, \ldots, \ell_r\}$ of linear constraints to be redundant, then we will present an algorithm based on incremental SMT solving which constructs a maximal set of redundant constraints, and finally,

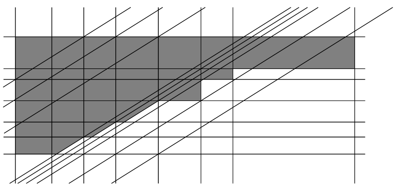

**Figure 2: Before redundancy removal**

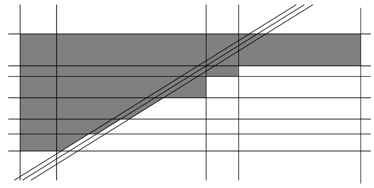

**Figure 3: After redundancy removal**

we develop a method really computing a representation not depending on $\{\ell_1, \ldots, \ell_r\}$ assuming that the redundancy check was successful.

In order to be able to check for redundancy, we assume a disjoint copy $C' = \{c'_1, \ldots, c'_f\}$ of the continuous variables $C = \{c_1, \ldots, c_f\}$. Moreover, for each linear constraint $\ell_i$ ($1 \le i \le n$) we introduce a corresponding linear constraint $\ell'_i$ which coincides with $\ell_i$ up to replacement of variables $c_j \in C$ by variables $c'_j \in C'$. Our check for redundancy is based on the following theorem:

THEOREM 1 (REDUNDANCY CHECK).
*Let $\ell_1, \ldots, \ell_r$ be linear constraints over variables from $\{c_1, \ldots, c_f\}$ and let $\ell'_1, \ldots, \ell'_r$ be identical linear constraints over a disjoint set of variables $\{c'_1, \ldots, c'_f\}$. The linear constraints $\ell_1, \ldots, \ell_r$ ($1 \le r \le n$) are redundant in the representation of $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$ if and only if the predicate*

$$F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k) \oplus F(\ell'_1, \ldots, \ell'_n, b_1, \ldots, b_k) \\ \wedge \bigwedge_{i=r+1}^{n} (\ell_i \equiv \ell'_i) \quad (1)$$

*(where $\oplus$ denotes exclusive-or and $\equiv$ denotes boolean equivalence) is not satisfiable by any assignment of real values to the variables $c_1, \ldots, c_f$, $c'_1, \ldots, c'_f$ and of boolean values to $b_1, \ldots, b_k$.*

Note that the check from Thm. 1 can be performed by a (conventional) SMT solver.

PROOF THM. 1, ONLY-IF-PART.
For the proof of the 'only-if-part' of Thm. 1 we assume that the predicate from formula (1) is satisfiable and under this assumption we prove that it cannot be the case that all linear constraints $\ell_1, \ldots, \ell_r$ are redundant, i.e., that there is no boolean function $G$ such that $G(\ell_{r+1}, \ldots, \ell_n, b_1, \ldots, b_k)$ and $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$ represent the same predicates.

Now consider some satisfying assignment to the predicate from formula (1) as follows: For the real variables $c_1 := v_{c_1}, \ldots, c_f := v_{c_f}$ with $(v_{c_1}, \ldots, v_{c_f}) \in \mathbb{R}^f$, for the copied real variables $c'_1 := v_{c'_1}, \ldots, c'_f := v_{c'_f}$ with $(v_{c'_1}, \ldots, v_{c'_f}) \in \mathbb{R}^f$, and for the boolean variables $b_1 := v_{b_1}, \ldots, b_k := v_{b_k}$ with $(v_{b_1}, \ldots, v_{b_k}) \in \{0,1\}^k$.

This satisfying assignment implies a corresponding truth assignment to the linear constraints by $\ell_i(v_{c_1}, \ldots, v_{c_f}) = v_{\ell_i}$ ($1 \le i \le n$) with $v_{\ell_i} \in \{0,1\}$ and to the copied linear constraints by $\ell'_i(v_{c'_1}, \ldots, v_{c'_f}) = v_{\ell'_i}$ ($1 \le i \le n$) with $v_{\ell'_i} \in \{0,1\}$.

Since the assignment satisfies formula (1), it holds that

$$\begin{array}{ll} F(v_{\ell_1}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) & \\ \quad \neq F(v_{\ell'_1}, \ldots, v_{\ell'_n}, v_{b_1}, \ldots, v_{b_k}), & (a) \\ \quad v_{\ell_i} = v_{\ell'_i} \text{ for all } r+1 \le i \le n. & (b) \end{array}$$

(Part (a) holds because of the first part of formula (1), i.e. $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n) \oplus F(b_1, \ldots, b_k, \ell'_1, \ldots, \ell'_n)$, and part (b) holds because of the second part $\bigwedge_{i=r+1}^{n} (\ell_i \equiv \ell'_i)$.)

Thus the satisfying assignment produces two assignments to the inputs of $F$ which may differ only in the first $r$ values, whereas the function values of $F$ for these two assignments differ. However, any boolean function $G$ not depending on the first $r$ inputs cannot see the difference between these two assignments and thus, it cannot produce different outputs for these two assignments as $F$. Thus, it is clear that any $G$ not depending on the first $r$ inputs cannot represent the same predicate as $F$. ☐

For the 'if-part' of Thm. 1 it remains to be shown that an appropriate function $G$ can be constructed, if formula (1) is unsatisfiable.

When constructing $G$, we need the notion of the *don't care set DC induced by linear constraints*:

*Definition 2.* The *don't care set DC induced by linear constraints* $\ell_1, \ldots, \ell_n$ is defined as $DC := \{(v_{\ell_1}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \mid \nexists (v_{c_1}, \ldots, v_{c_f}) \in \mathbb{R}^f$ with $\ell_i(v_{c_1}, \ldots, v_{c_f}) = v_{\ell_i} \forall 1 \le i \le n$, $(v_{b_1}, \ldots, v_{b_k}) \in \{0,1\}^k\}$.

This don't care set contains all assignments of truth values to linear constraints which are inconsistent in the sense that the corresponding linear constraints cannot hold these truth values at the same time.

Based on the set $DC$, an appropriate boolean function $G$ can be constructed with $G(\ell_{r+1}, \ldots, \ell_n, b_1, \ldots, b_k)$ and $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$ representing the same predicates, if there is no satisfying assignment to formula (1). This proves the if-part of the proof for Thm. 1. However, we omit this proof here, since we will give an alternative constructive proof for the if-part in Sect.3.3. This constructive proof makes use of a subset $DC'$ of $DC$ which is computed by an SMT solver during the proof of unsatisfiability for formula (1).

### *Overall algorithm for redundancy detection.*

Now we can present our overall algorithm detecting a maximal set of linear constraints which can be removed from the representation *at the same time.* We start with a small example demonstrating the effect that it is not enough to consider redundancy of single linear constraints and to construct larger sets of redundant constraints simply as unions of smaller sets.

*Example 1.* Consider the predicate $F(c_1, c_2) = (c_1 \ge 0) \wedge (c_2 \ge 0) \wedge \neg(c_1 + c_2 \le 0) \wedge \neg(2c_1 + c_2 \le 0)$. It is easy to see that both the third and the forth linear constraint in the conjunction have the effect of 'removing the value $(c_1, c_2) = (0, 0)$ from the predicate $F'(c_1, c_2) = (c_1 \ge 0) \wedge (c_2 \ge 0)$'. Therefore both $\ell_3 = (c_1 + c_2 \le 0)$ and $\ell_4 = (2c_1 + c_2 \le 0)$ are obviously redundant linear constraints in $F$. However, it is easy to see that $\ell_3$ and $\ell_4$ are not redundant in the representation of $F$ **at the same time**, i.e., only $\neg(c_1+c_2 \le$

0) **or** $\neg(2c_1 + c_2 \leq 0)$ can be omitted in the representation for $F$.

This observation motivates the following overall algorithm to detect a maximal set of redundant linear constraints:

**Input:** Predicate $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$
**Output:** Maximal set $S$ of redundant linear constraints
$S := \emptyset$
**for** $i := 1$ to $n$ **do**
  **if** redundant($F$, $S \cup \{\ell_i\}$) **then**
    $S := S \cup \{\ell_i\}$
  **end if**
**end for**

*redundant*$(F, S \cup \{l_i\})$ implements the check from Thm. 1 by using an SMT solver. It is important to note that the $n$ SMT problems to be solved in the above loop share almost all of their clauses. For that reason we make use of an *incremental* SMT solver to solve this series of problems. An incremental SMT solver is able to profit from the similarity of the problems by transferring learned knowledge from one SMT solver call to the next (by means of learned conflict clauses). Experimental results in Sect. 4 indeed show the advantage of using an incremental SMT solver.

## 3.3 Removal of Redundant Linear Constraints for Non-convex Polyhedra

Suppose that the linear constraints $\ell_1, \ldots, \ell_r$ are redundant in $F(\ell_1, \ldots, \ell_n, b_1, \ldots, b_k)$. Now we are looking for an efficient procedure to compute a boolean function $G$ which is appropriate in the sense of Def. 1 and does not depend on the first $r$ inputs. As already mentioned in Sect. 3.2 it is possible to define a method which in principle computes such a function $G$ using the don't care set $DC$ (according to Def. 2). However, an efficient realization of this method would certainly need a compact representation of the don't care set $DC$. Fortunately, a closer look at the problem reveals the following two interesting observations which turn the basic idea into a feasible approach:

1. In general, we do not need the complete set $DC$ for the computation of the boolean function $G$.

2. A representation of a sufficient subset of $DC$ which is needed for removing the redundant constraints $\ell_1, \ldots, \ell_r$ is already computed by an SMT solver when checking the satisfiability of formula (1), if one assumes that the SMT solver uses the option of minimizing conflict clauses.

In order to explain how an appropriate subset of $DC$ is computed by the SMT solver (when checking the satisfiability of formula (1)) we need to have a closer look at the functionality of an SMT solver:

An SMT solver introduces constraint variables $q_{\ell_i}$ for linear constraints $\ell_i$ (just as in LinAIGs as shown in Fig. 1). First, the SMT solver looks for satisfying assignments to the boolean variables (including the constraint variables). Whenever the SMT solver detects a satisfying assignment to the boolean variables, it checks whether the assignment to the constraint variables is consistent, i. e., whether it can be produced by replacing real-valued variables by reals in the linear constraints. This task is performed by a linear program solver. If the assignment is consistent, then the SMT solver has found a satisfying assignment, otherwise it continues searching for satisfying assignments to the boolean variables. If some assignment $\epsilon_1, \ldots, \epsilon_m$ to constraint variables $q_{\ell_{i_1}}, \ldots, q_{\ell_{i_m}}$ was found to be inconsistent, then the boolean 'conflict clause' $(q_{\ell_{i_1}}^{\overline{\epsilon_1}} + \ldots + q_{\ell_{i_m}}^{\overline{\epsilon_m}})$ is added to the set of clauses in the SMT solver to avoid running into the same conflict again. The negation of this conflict clause describes a set of don't cares due to an inconsistency of linear constraints.

Now consider formula (1) which has to be solved by an SMT solver and suppose that the solver introduces boolean constraint variables $q_{\ell_i}$ for linear constraints $\ell_i$ and $q_{\ell'_i}$ for $\ell'_i$ $(1 \leq i \leq n)$. Since linear constraints $\ell_1, \ldots, \ell_r$ are redundant, formula (1) is unsatisfiable (see Thm. 1). This means that whenever there is some satisfying assignment to boolean variables (including constraint variables) in the SMT solver, it will be necessarily shown to be inconsistent. The most important observation is now that the negations of conflict clauses due to these inconsistencies include the don't cares needed to compute an appropriate boolean function $G$.

In order to see this, we define for arbitrary values $(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \in \{0,1\}^{n-r+k}$ the sets $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) := \{(v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \mid (v_{\ell_1}, \ldots, v_{\ell_r}) \in \{0,1\}^r\}$.

If there is an orbit $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ containing two different elements $v^{(1)} := (v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ and $v^{(2)} := (v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ with $F(v^{(1)}) \neq F(v^{(2)})$, then the following assignment to the boolean variables obviously satisfies the boolean abstraction of formula (1) in the SMT solver: $q_{\ell_1} := v_{\ell_1}, \ldots, q_{\ell_r} := v_{\ell_r}, q_{\ell'_1} := v'_{\ell_1}, \ldots, q_{\ell'_r} := v'_{\ell_r}$, $q_{\ell_{r+1}} := q_{\ell'_{r+1}} := v_{\ell_{r+1}}, \ldots, q_{\ell_n} := q_{\ell'_n} := v_{\ell_n}$, $b_1 := v_{b_1}, \ldots, b_k := v_{b_k}$.

Since we assumed that formula (1) is not satisfiable, this assignment cannot be consistent wrt. the interpretation of constraint variables by linear constraints. So there must be an inconsistency in the truth assignment to some linear constraints $\ell_1, \ldots, \ell_n$, $\ell'_1, \ldots, \ell'_n$. Since the linear constraints $\ell_i$ and $\ell'_i$ are based on disjoint sets of real variables $C = \{c_1, \ldots, c_f\}$ and $C' = \{c'_1, \ldots, c'_f\}$, respectively, it is easy to see that a minimal number of assignments which are already inconsistent contains *either* only assignments to a subset of $\ell_1, \ldots, \ell_n$ *or* to a subset of $\ell'_1, \ldots, \ell'_n$.[3] When using the option of minimizing conflict clauses, the SMT solver will thus learn a conflict clause whose negation either contains the don't care $v^{(1)} = (v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ or the don't care $v^{(2)} = (v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$. Since this consideration holds for all pairs of elements in some orbit $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ for which $F$ produces different values, this means for the subset $DC' \subseteq DC$ of don't cares detected during the run of the SMT solver: If $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ is not completely contained in $DC'$, then $|F(orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \setminus DC')| = 1$ (or in other words: the elements of $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ which are not in $DC'$ are either all mapped by $F$ to 0 or are all mapped by $F$ to 1).

Now we define the function value of $G$ for each $(v_{\ell_{r+1}}, \ldots,$

[3] For our purposes, it does not matter whether the inconsistency is given in terms of linear constraints $\ell_1, \ldots, \ell_n$ or $\ell'_1, \ldots, \ell'_n$. We are only interested in assignments of boolean values to linear constraints leading to inconsistencies; of course, the same inconsistencies will hold both for $\ell_1, \ldots, \ell_n$ and their copies $\ell'_1, \ldots, \ell'_n$.

$v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \in \{0,1\}^{n-r+k}$:

1. If $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \subseteq DC'$, then $G(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ is chosen arbitrarily.

2. Otherwise $G(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) = \delta$ with $F(orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \setminus DC') = \{\delta\}$, $\delta \in \{0,1\}$.

It is easy to see that $G$ does not depend on variables $q_{\ell_1}, \ldots, q_{\ell_r}$ and that $G$ is well-defined (this follows from $|F(orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) \setminus DC')| = 1$), i.e. $G$ is a possible solution according to Def. 1. This consideration also provides a proof for the if-part of Thm. 1. Note that according to case 1. of the definition above there may be several possible choices fulfilling the definition of $G$.

A predicate $dc$ which describes the don't cares in $DC'$ may be extracted from the SMT solver as a disjunction of negated conflict clauses which record inconsistencies between linear constraints.

### 3.3.1 Redundancy Removal by Existential Quantification

A straightforward way of computing an appropriate function $G$ relies on existential quantification:

- At first by $G' = F \wedge \overline{dc}$ all don't cares represented by $dc$ are mapped to the function value 0.

- Secondly, we perform an existential quantification of the variables $q_{\ell_1}, \ldots, q_{\ell_r}$ in $G'$: $G = \exists q_{\ell_1}, \ldots, q_{\ell_r} G'$. This existential quantification maps all elements of an orbit $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ to 1, whenever the orbit contains an element $\epsilon$ with $dc(\epsilon) = 0$ and $F(\epsilon) = 1$. Since due to the argumentation above there is no other element $\delta$ in such an orbit with $dc(\delta) = 0$ and $F(\delta) = 0$, $G$ eventually differs from $F$ only for don't cares defined by $dc$ and it certainly does not depend on variables $q_{\ell_1}, \ldots, q_{\ell_r}$, i.e. existential quantification computes one possible solution for $G$ according to Def. 1 (more precisely it computes exactly the solution for $G$ which maps a minimum number of elements of $\{0,1\}^{n-r+k}$ to 1).

### 3.3.2 Redundancy Removal with Craig Interpolants

Although our implementation of LinAIGs supports quantification of boolean variables by a series of methods like avoiding the insertion of equivalent nodes (see Sect. 2.1), quantifier scheduling, BDD sweeping and node selection heuristics (see [17]), there remains the risk of doubling the representation size by quantifying a single boolean variable.[4] Therefore the computation of $G$ by $G = \exists q_{\ell_1}, \ldots, q_{\ell_r} G'$ as shown above may potentially lead to large LinAIG representations (although it reduces the number of linear constraints).

On the other hand, this choice for $G$ is only one of many other possible choices. Motivated by these facts we looked for an alternative solution. Here we present a solution which needs only one application of Craig interpolation [3, 18] (see Sect. 2.2) instead of a series of existential quantifications of boolean variables. Note that in this context Craig interpolation leads to an exact result (as one of several possible choices) and not to an approximation as in [15].

[4]Basically, existential quantification of a boolean variable is reduced to a disjunction of both cofactors wrt. 0 and wrt. 1.

Our task is to find a boolean function $G(q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k)$ with

$$(F \wedge \overline{dc})(q_{\ell_1}, \ldots, q_{\ell_n}, b_1, \ldots, b_k) \implies G(q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k), \tag{2}$$

$$G(q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k) \implies (F \vee dc)(q_{\ell_1}, \ldots, q_{\ell_n}, b_1, \ldots, b_k). \tag{3}$$

Now let $A(q_{\ell_1}, \ldots, q_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k, h_1, \ldots, h_l)$ represent the CNF for a Tseitin transformation [21] of $(F \wedge \overline{dc})(q_{\ell_1}, \ldots, q_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k)$ (with new auxiliary variables $h_1, \ldots, h_l$).

Likewise, let $B(q'_{\ell_1}, \ldots, q'_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k, h'_1, \ldots, h'_{l'})$ be the CNF for a Tseitin transformation of $(\overline{F} \wedge \overline{dc})(q'_{\ell_1}, \ldots, q'_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k)$ (with new auxiliary variables $h'_1, \ldots, h'_{l'}$ and new copies $q'_{\ell_1}, \ldots, q'_{\ell_r}$ of the variables $q_{\ell_1}, \ldots, q_{\ell_r}$). Then $A$ and $B$ fulfill the precondition for Craig interpolation as given in Sect. 2.2, i.e., $A \wedge B = 0$:

Suppose that there is a satisfying assignment to $A \wedge B$ given by $q_{\ell_1} := v_{\ell_1}, \ldots, q_{\ell_r} := v_{\ell_r}, q'_{\ell_1} := v'_{\ell_1}, \ldots, q'_{\ell_r} := v'_{\ell_r}$, $q_{\ell_{r+1}} := v_{\ell_{r+1}}, \ldots, q_{\ell_n} := v_{\ell_n}$, $b_1 := v_{b_1}, \ldots, b_k := v_{b_k}$ and the corresponding assignments to auxiliary variables $h_1, \ldots, h_l$ and $h'_1, \ldots, h'_{l'}$ which are implied by these assignments. According to the definition of $A$ and $B$ this would mean that the set $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ would contain two elements $(v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ and $(v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ which do not belong to the don't care set $DC'$ and which fulfill $F(v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) = 1$ and $F(v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k}) = 0$. This is a contradiction to the property shown above that the elements of $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n}, v_{b_1}, \ldots, v_{b_k})$ which are not in $DC'$ are either all mapped by $F$ to 0 or are all mapped by $F$ to 1.

A Craig interpolant $G$ computed for $A$ and $B$ (e.g. according to [18]) has the following properties:

- It depends only on common variables $q_{\ell_{r+1}}, \ldots, q_{\ell_n}, b_1, \ldots, b_k$ of $A$ and $B$,

- $A \implies G$, i.e., $G$ fulfills equation (2), and

- $G \wedge B$ is unsatisfiable, or equivalently, $G \implies \overline{B}$, i.e., $G$ fulfills equation (3).

This shows that a Craig interpolant for $(A, B)$ is exactly one of the possible solutions for $G$ which we were looking for.

## 4. EXPERIMENTAL RESULTS

We implemented redundancy detection by incremental SMT solving and redundancy removal by Craig interpolation in the framework of LinAIGs. The implementation uses two SMT solvers via API calls. Yices [7] is used for all SMT solver calls except the generation of the don't care set. This means that Yices performs all equivalence checks needed for LinAIG compaction as described in Sect. 2.1 and moreover, it is also used for the redundancy detection algorithm described in Sect. 3 in an incremental way. For the computation of the don't care set required for redundancy removal we use HySAT [10], since we needed an SMT solver where we could modify the source code in order to be able to extract conflict clauses. The computation of the Craig interpolants is done with MiniSAT [8], where we made an extension to the proof logging version. All experiments were performed on an AMD Opteron with 2.6 GHz and 16 GB RAM under Linux.

## 4.1 Comparison of the LinAIG evolution with and without redundancy removal

In Fig. 4 we present a comparison of two runs of the model checker from [4]. The left diagram shows the evolution of the linear constraints over time and the right diagram shows the evolution of node counts. When we do not use redundancy removal, the number of linear constraint is quickly increasing up to 1000 and more, and the number of AIG nodes is exploding up to a value of 150,000 . On the other hand, when using redundancy removal the number of linear constraints and the number of AIG nodes show only a moderate growth rate. This gives a strong evidence that redundancy removal is absolutely necessary when using quantifier elimination to keep the data structure compact in our model checking environment.

## 4.2 Elimination of redundant constraints: Existential quantification vs. Craig interpolation

In a second experiment we compared two different approaches to the removal of redundant constraints as presented in Sect. 3.3. The first one uses existential quantification to eliminate redundant constraints, the second one uses our approach based on Craig interpolation. The benchmarks represent state sets extracted from the model checker in [4] during three runs with different model checking problems, in each case after the elimination of quantifiers over real variables. These problems also contain boolean variables.

The results are given in Table 1. The number of the AIG nodes and linear constraints before redundancy removal are shown in columns 2 and 3. In column 4 the detected number of redundant linear constraints is given. The times for the detection of redundancy and the don't care set generation are given in columns 5 and 6. Note that these values are the same for both approaches, because the difference lies only in the way linear constraints are actually removed. In the last four columns the results of the two algorithms are shown, where '$\Delta$ nodes' denotes the difference between the number of AIG nodes before and after the removal step and 'time' is the CPU time needed for this step. We used a timeout of 7200 seconds and a memory limit of 4 GB.

The results clearly show that wrt. runtime the redundancy removal based on Craig interpolation outperforms the approach with existential quantification by far. Especially when the benchmarks are more complex and show a large number of redundant linear constraints, the difference between the two methods is substantial. Moreover, also the resulting AIG is often much smaller. It is interesting to see that using incremental SMT solving techniques it was in many cases really possible to detect large sets of redundant linear constraints in very short times. As shown in the previous experiment this pays off also in later steps of model checking when quantifier elimination works on a representation with a smaller number of linear constraints. Considering column 6 we observe that runtimes for the generation of don't care sets by HySAT often dominate the overall runtime.[5] For the future we plan to replace HySAT (which is tuned for BMC problems and is clearly outperformed by Yices in our experiments on non-BMC problems) by a state-of-the-art SMT solver allowing the extraction of conflict clauses.[6] This is an issue where we see much room for improvement.

[5]As already mentioned above, for technical reasons in our implementation we have to repeat the last step of redundancy detection (which actually was already performed by Yices) using HySAT in order to be able to extract conflict clauses.

## 4.3 Comparison of the LinAIG based quantifier elimination vs. other solvers

In order to evaluate our ideas in a more general domain we compared our approach to quantifier elimination with LIRA 1.1.2 [9, 2] which is an automata-based tool capable of representing sets of states over real, integer, and boolean variables and both CVC3 1.2.1 [20] and Yices 1.0.11 [7] which are state-of-the-art SMT solvers. We ran the solvers on three sets of formulas from the class of quantified linear real arithmetic:

1. model_X: These formulas are representing problems occurring in the model-checker [4] when computing a continuous pre-image of the state set. All formulas of this set contain two quantified variables, one is existentially quantified and the other is universally quantified.

2. RND: These formulas are random trees composed of quantifiers, AND-, OR-, NOT-operators, and linear inequations. The quantifiers are randomly distributed over the whole formula tree. We varied the number of quantified variables and the depth of the trees to get formulas with different difficulty levels. In all cases there was an additional free variable left in the formula. The random benchmarks were generated with the tool also used in [9, 2].

3. RNDPRE: These formulas are similar to the RND set, except that the formulas all consist of a prefix of alternating quantifiers and a quantifier free inner part.

All formulas are given in the SMT-LIB format [19] and are publically available[7].

Since the SMT solvers decide satisfiability of formulas instead of computing predicates representing all satisfying assignments, we interpret free variables as implicitly existentially quantified and decide satisfiability. Both our LinAIG based tool and LIRA additionally compute representations for predicates representing all satisfying assignments. We used a time limit of 1200 CPU seconds and a memory limit of 4 GB.

Table 2 shows the results. The column 'Benchmark' lists the benchmark sets, 'Quantified' lists the number of quantified variables in each formula of the set, 'Instances' shows the number of instances in the set. The columns labeled 'SAT' and 'UNSAT' give the numbers of instances for which the solver returned 'satisfiable' and 'unsatisfiable'. The numbers of instances where the solver returned 'unknown', ran out of memory, or violated the time limit, are listed in the columns 'Unknown', 'Memout', 'Timeout'. Column 'Time (s)' shows the total run times (in CPU seconds) of the solver for the formula set[8], and finally column 'Solved' lists the total numbers of solved instances of the set. The results for

[6]This would include also a handling of don't cares which do not occur in the set of (negated) conflict clauses due to 'theory propagation'.

[7]`http://abs.informatik.uni-freiburg.de/smtbench/`

[8]Unsolved instances (i. e. 'Unknown', 'Memout', and 'Timeout') are considered to contribute 1200 CPU seconds (the time limit)

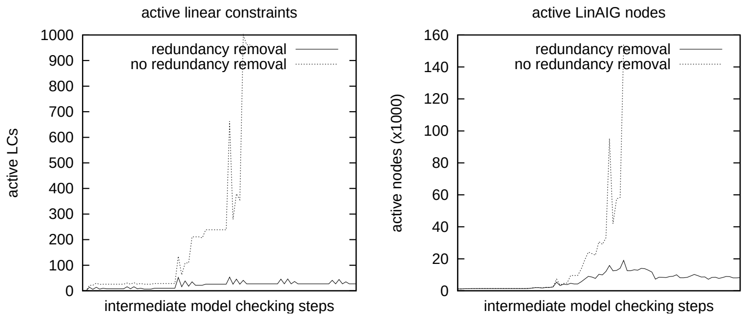


**Figure 4: Comparison of the LinAIG evolution with and without redundancy removal**

**Table 1: Comparison of redundancy removal: existential quantification vs. Craig interpolants**

| Benchmark | # AIG nodes | # linear constr. | # redundant lin. constr. | redundancy detection (s) | dc set creation (s) | RR exist. quant. | | RR Craig interp. | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Δ nodes | time (s) | Δ nodes | time (s) |
| model_1-1 | 1459 | 41 | 22 | 0.22 | 1.10 | -541 | 7.19 | -814 | 0.74 |
| model_1-2 | 1716 | 74 | 27 | 0.51 | 1.79 | -313 | 13.14 | -1047 | 0.68 |
| model_1-3 | 2340 | 105 | 22 | 1.89 | 8.48 | 459 | 25.35 | -515 | 2.32 |
| model_1-4 | 3500 | 142 | 28 | 8.02 | 41.72 | 1642 | 75.49 | 1062 | 10.08 |
| model_1-5 | 2837 | 123 | 13 | 4.61 | 29.02 | -230 | 12.34 | 1595 | 23.10 |
| model_2-1 | 824 | 29 | 8 | 0.12 | 0.27 | 747 | 2.55 | -142 | 0.43 |
| model_2-2 | 1424 | 37 | 10 | 0.36 | 0.80 | 1104 | 3.45 | 233 | 1.19 |
| model_2-3 | 3048 | 52 | 11 | 2.45 | 3.09 | 1996 | 10.13 | 171 | 4.22 |
| model_2-4 | 1848 | 37 | 14 | 0.57 | 1.02 | 852 | 4.03 | -149 | 1.34 |
| model_3-1 | 1775 | 297 | 228 | 1.86 | 49.28 | | >7200 | -1453 | 1.60 |
| model_3-2 | 6703 | 1281 | 1143 | 105.69 | 2113.20 | | >7200 | -5805 | 14.12 |

CVC3, Yices, LIRA, and our LinAIG based solver using redundancy removal are shown in the column groups labeled 'CVC3', 'Yices','LIRA', and 'LinAIG'.

CVC3 is able to solve 34 out of 380 instances, Yices solves 13 instances. Note however that these solvers are not restricted to the subclass of formulas we consider in this paper. They are able to handle the more general AUFLIRA class of formulas [1] and for handling formulas with quantifiers they make use of heuristics based on E-matching [6] which are not tuned to problems that contain only arithmetic.

The automata-based tool LIRA solves 95 out of 380 instances.

Our experiments show that for the subclass of formulas considered here our method is much more effective: The LinAIG based solver is able to solve 352 out of 380 instances.

## 5. CONCLUSIONS AND FUTURE WORK

We presented an approach for optimizing non-convex polyhedra based on the removal of redundant constraints. Our experimental results show that our approach can be successfully applied to solving quantified formulas including linear real arithmetic and boolean formulas. Since our method does not only solve satisfiability of formulas, but constructs predicates of all satisfying assignments to the free variables in the formula, our results may suggest to use the presented method in the future also as a fast preprocessor for more general formulas by simplifying subformulas from the subclass considered in this paper. Moreover, it will be interesting to apply the methods to underlying theories different from linear real arithmetic, too.

## Acknowledgements

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 cooperation with our colleagues from the 'First Order Model Checking team' within subproject H3 and we would like to thank W. Damm, H. Hungar, J. Pang, and B. Wirtz from the University of Oldenburg, and S. Jacobs and U. Waldmann from the Max Planck Institute for Computer Science at Saarbrücken for numerous ideas and helpful discussions. Moreover, we would like to thank Jochen Eisinger from the University of Freiburg for providing the formula generator used in our experiments.

## 6. REFERENCES

[1] C. Barrett, M. Deters, A. Oliveras, and A. Stump. Satisfiability Modulo Theories Competition (SMT-COMP) 2008: Rules and Precedures, 2008. `http://smtcomp.org/rules08.pdf`.

[2] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke. LIRA: Handling constraints of linear arithmetics over the integers and the reals. In *Proc. of the 19th International Conference on Computer Aided Verification*, LNCS, pages 312–315. Springer, 2007.

[3] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal on Symbolic Logic*, 22(3):269–285, 1957.

[4] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space.

**Table 2: Comparison of Solvers**

| | | | CVC3 | | | | | | | YICES | | | | | LIRA | | | | | | LinAIG | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Benchmark | Quantified | Instances | SAT | UNSAT | Unknown | Memout | Timeout | Time (s) | **Solved** | SAT | UNSAT | Unknown | Time (s) | **Solved** | SAT | UNSAT | Memout | Timeout | Time (s) | **Solved** | SAT | UNSAT | Timeout | Time (s) | **Solved** |
| model_4 | 2 | 6 | 0 | 0 | 6 | 0 | 0 | 7K | **0** | 0 | 0 | 6 | 7K | **0** | 2 | 0 | 4 | 0 | 5K | **2** | 6 | 0 | 0 | 11 | **6** |
| model_5 | 2 | 64 | 0 | 0 | 64 | 0 | 0 | 77K | **0** | 0 | 0 | 64 | 77K | **0** | 39 | 0 | 12 | 13 | 39K | **39** | 64 | 0 | 0 | 282 | **64** |
| model_6 | 2 | 80 | 0 | 0 | 80 | 0 | 0 | 96K | **0** | 0 | 0 | 80 | 96K | **0** | 27 | 0 | 46 | 7 | 69K | **27** | 80 | 0 | 0 | 251 | **80** |
| RND | 3 | 30 | 1 | 7 | 21 | 1 | 0 | 26K | **8** | 3 | 3 | 24 | 29K | **6** | 4 | 5 | 21 | 0 | 25K | **9** | 17 | 13 | 0 | 1K | **30** |
| RND | 4 | 30 | 0 | 2 | 27 | 0 | 1 | 34K | **2** | 1 | 0 | 29 | 35K | **1** | 1 | 1 | 28 | 0 | 34K | **2** | 19 | 10 | 1 | 2K | **29** |
| RND | 6 | 40 | 0 | 0 | 31 | 7 | 2 | 48K | **0** | 0 | 4 | 36 | 43K | **4** | 1 | 0 | 39 | 0 | 47K | **1** | 18 | 9 | 13 | 16K | **27** |
| RNDPRE | 3 | 60 | 0 | 24 | 31 | 4 | 1 | 43K | **24** | 0 | 1 | 59 | 71K | **1** | 6 | 8 | 45 | 1 | 57K | **14** | 34 | 26 | 0 | 2K | **60** |
| RNDPRE | 4 | 70 | 0 | 0 | 66 | 4 | 0 | 84K | **0** | 1 | 0 | 69 | 83K | **1** | 1 | 0 | 67 | 2 | 83K | **1** | 28 | 28 | 14 | 21K | **56** |
| Total | | 380 | 1 | 33 | 326 | 16 | 4 | 415K | **34** | 5 | 8 | 367 | 440K | **13** | 81 | 14 | 262 | 23 | 358K | **95** | 266 | 86 | 28 | 43K | **352** |


In *5th International Symposium on Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 425–440. Springer, 2007.

[5] W. Damm, S. Disch, H. Hungar, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Automatic verification of hybrid systems with large discrete state space. In *4th Symposium on Automated Technology for Verification and Analysis*, volume 4218 of *LNCS*, pages 276–291, 2006.

[6] D. Detlefs, G. Nelson, and J. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Systems Research Center, 2003.

[7] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *18th Conference on Computer Aided Verification*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.

[8] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th InternationalConference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8,2003 Selected Revised Papers*, volume 2919 of *LNCS*, pages 541–638. Springer, 2003.

[9] J. Eisinger and F. Klaedtke. Don't care words with application to the automata-based approach for real addition. In *Proc. of the 18th International Conference on Computer Aided Verification*, LNCS, pages 67–80. Springer, 2006.

[10] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[11] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *8th Workshop on Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.

[12] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[13] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko. Scalable exploration of functional dependency by interpolation and incremental SAT solving. In G. G. E. Gielen, editor, *ICCAD*, pages 227–233. IEEE, 2007.

[14] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.

[15] K. L. McMillan. Interpolation and SAT-based model checking. In *15th Conference on Computer Aided Verification*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003.

[16] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, EECS Dept., UC Berkeley, 2005.

[17] F. Pigorsch, C. Scholl, and S. Disch. Advanced unbounded model checking by using AIGs, BDD sweeping and quantifier scheduling. In *6th Conference on Formal Methods in Computer Aided Design*, pages 89–96. IEEE Press, 2006.

[18] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal on Symbolic Logic*, 62(3):981–998, 1997.

[19] S. Ranise and C. Tinelli. The SMT-LIB Standard: Version 1.2, 2006. `http://combination.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf`.

[20] A. Stump, C. W. Barrett, and D. L. Dill. CVC: A cooperating validity checker. In *Proceedings of the $14^{th}$ International Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 500–504. Springer, 2002.

[21] G. Tseitin. On the complexity of derivations in propositional calculus. In A. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logics*. 1968.

[22] F. Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Transactions on Software Engineering*, 31(1):38–52, 2005.