Computing Optimized Representations for Non-convex Polyhedra by Detection and Removal of Redundant Linear Constraints

Christoph Scholl, Stefan Disch, Florian Pigorsch, and Stefan Kupferschmid

Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 51, 79110 Freiburg, Germany

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 (Satisfiability Modulo Theories) solver and then removes the redundant constraints based on Craig interpolation. The approach is motivated by applications in the context of model checking for Linear Hybrid Automata. Basically, it can be seen as an optimization method for formulas including arbitrary boolean combinations of linear constraints and boolean variables. We show that our method provides an essential step making quantifier elimination for linear arithmetic much more efficient. Experimental results clearly show the advantages of our approach in comparison to state-of-the-art solvers.

1 Introduction

In this paper we present an approach which uses SMT (Satisfiability Modulo Theories) solvers and Craig interpolation [1] 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 [2,3] consider unions of convex polyhedra (i.e. unions of conjunctions of linear constraints) together with an explicit representation of discrete states, in [4,5] 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

S. Kowalewski and A. Philippou (Eds.): TACAS 2009, LNCS 5505, pp. 383–397, 2009.

constraints. In that sense our method significantly extends results for eliminating redundant linear constraints from convex polyhedra used by Frehse [3] and Wang [6].[1] In previous work [5] we already made the observation that a major obstacle to the application of sequences of quantifier eliminations in the context of model checking for hybrid systems is formed by the growth of the number of linear constraints in state set representations during Weispfennig–Loos quantifier elimination [7]. For that reason removing redundant linear constraints from non-convex polyhedra plays an essential role 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.
- 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* [1,8,9].

In a comparison with existing tools we consider formulas consisting of arbitrary boolean combinations of linear constraints and boolean variables, 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 [10], the computer algebra system REDUCE/REDLOG [11,12] (which also solve both problems mentioned above) and to the results of state-of-the-art SMT solvers Yices [13] and CVC3 [14] (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 the results of SMT solvers restricted to quantifier-free satisfiability solving.[2] Our

[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.

[2] In our implementation we use Yices [13] and MathSAT [15] for this task.

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 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 α_i and $c_i \in C$. Predicates are based on 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 [4,5]. LinAIGs are And-Inverter-Graphs (AIGs) enriched by linear constraints. The structure of LinAIGs is illustrated in Fig. 1.

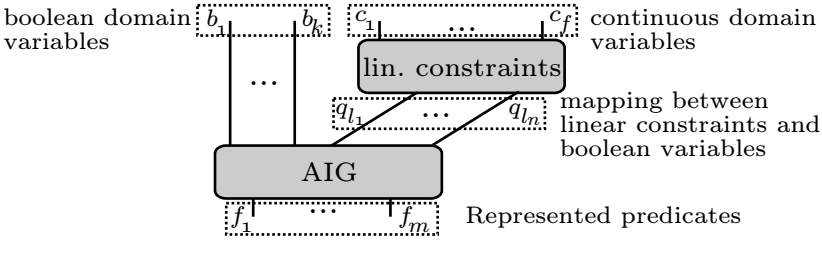

Fig. 1. Structure of LinAIGs

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.

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 $q_{\ell_i} \in Q$. For keeping the representation as compact as possible we use a multitude of methods; e.g., inserting different nodes representing the same predicate is avoided using an SMT (SAT modulo theories) solver which combines DPLL with linear programming as a decision procedure [4,5].

2.2 Quantifier Elimination

In [5] Loos's and Weispfenning's test point method [7] was adapted to the LinAIG data structure described above. The method eliminates universal quantifiers for real-valued variables 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 in 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 in the original formula as small as possible. Moreover, after elimination of one quantifier (starting with the innermost), it is also important to remove redundant linear constraints generated as a result of the test point method. For this reason we developed an algorithm which computes representations depending on a minimal set of linear constraints (see 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. Note that our approach also *works for the generalized case of arbitrary boolean combinations of linear constraints and additional boolean variables* (as represented by LinAIGs, e.g.), but here we confine ourselves to non-convex polyhedra in order to keep the exposition more compact and readable.

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.

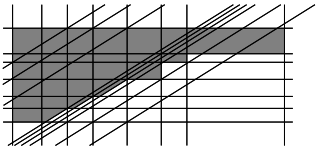

Fig. 2. Before redundancy removal

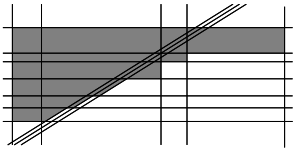

Fig. 3. After redundancy removal

3.1 Redundancy Detection and Removal for Convex Polyhedra

The task of detecting and removing redundant constraints in *non-convex* polyhedra is not as straightforward as for other approaches such as [2,3] 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 ℓ_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

Definition 1 (Redundancy of linear constraints). *Let F be a boolean function and let $\ell_1, \ldots, \ell_n$ be linear constraints over real-valued variables $C = \{c_1, \ldots, c_f\}$. The linear constraints $\ell_1, \ldots, \ell_r$ $(1 \leq r \leq n)$ are called redundant in the representation of $F(\ell_1, \ldots, \ell_n)$ iff there is a boolean function G with the property that $F(\ell_1, \ldots, \ell_n)$ and $G(\ell_{r+1}, \ldots, \ell_n)$ represent the same predicates.*

Our check for redundancy is based on the following theorem [5]:

Theorem 1 (Redundancy check). *For all $1 \leq i \leq n$ let ℓ_i be a linear constraint over real-valued variables $\{c_1, \ldots, c_f\}$ and ℓ'_i* **exactly the same** *linear constraint as ℓ_i, but now over a* **disjoint copy** *$\{c'_1, \ldots, c'_f\}$ of the variables. Let $\oplus$ denote exclusive-or and $\equiv$ denote boolean equivalence. The linear constraints $\ell_1, \ldots, \ell_r$ $(1 \leq r \leq n)$ are redundant in the representation of $F(\ell_1, \ldots, \ell_n)$ if and only if the predicate*

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

is not satisfiable by any assignment of real values to the variables $c_1, \ldots, c_f$ and $c'_1, \ldots, c'_f$.

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

A sketch of the proof for the 'only-if-part' of Thm. 1 was already given in [5]. In this paper (Sect. 3.3) we present a constructive proof for the 'if-part' of the theorem in order to provide an efficient procedure to compute an appropriate function G whenever formula (1) is unsatisfiable.

Overall algorithm for redundancy detection. First of all, we 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 $F(c_1, c_2) = (c_1 \geq 0) \wedge (c_2 \geq 0) \wedge \neg(c_1 + c_2 \leq 0) \wedge \neg(2c_1 + c_2 \leq 0)$. 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 $F'(c_1, c_2) = (c_1 \geq 0) \wedge (c_2 \geq 0)$'. Therefore both $\ell_3 = (c_1 + c_2 \leq 0)$ and $\ell_4 = (2c_1 + c_2 \leq 0)$ are obviously redundant linear constraints in F. However, it is also easy to see that ℓ_3 and ℓ_4 are not redundant in the representation of F *at the same time*, i.e., only $\neg(c_1 + c_2 \leq 0)$ *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(ℓ_1, ..., ℓ_n)
Output: S: Maximal set of redundant linear constraints
begin
    S := ∅;
    for i := 1 to n do
        if redundant(F, S ∪ {ℓ_i}) then
            S := S ∪ {ℓ_i};
    return S;
end
```

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

Suppose that formula (1) of Thm. 1 is unsatisfiable. Now we are looking for an efficient procedure to compute a boolean function G such that $G(\ell_{r+1}, \ldots, \ell_n)$

and $F(\ell_1, \ldots, \ell_n)$ represent the same predicates. Obviously, the boolean functions F and G do not need to be identical in order to achieve this objective; they are allowed to differ for 'inconsistent' arguments which can not be produced by evaluating the linear constraints with real values. The set of these arguments is described by the following set DC:

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

As we will see in the following, it is possible to compute a function G as needed by making use of the don't care set DC. However, an efficient realization 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 our 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 DC' 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 DC' of DC is computed by the SMT solver (when checking the satisfiability of formula (1)) we start with a brief review of the functionality of an SMT solver:[3]

An SMT solver introduces constraint variables q_{ℓ_i} for linear constraints ℓ_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_{ℓ_i} for linear constraints ℓ_i and $q_{\ell_i'}$ for ℓ_i' $(1 \leq i \leq n)$. Whenever there is some satisfying assignment to boolean variables (including constraint variables) in the SMT solver, it will be necessarily shown to be inconsistent, since formula (1) is unsatisfiable.

[3] Here we refer to the *lazy* approach to SMT solving, see [18], e.g., for an overview.

In order to define an appropriate function G we introduce the concept of so-called orbits: For an arbitrary value $(v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \in \{0,1\}^{n-r}$ the corresponding orbit is defined by

$$\mathit{orbit}(v_{\ell_{r+1}}, \ldots, v_{\ell_n}) := \{(v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \mid (v_{\ell_1}, \ldots, v_{\ell_r}) \in \{0,1\}^r\}.$$

Now the following essential observations result from the unsatisfiability of formula (1): If some orbit $\mathit{orbit}(v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ contains two different elements $v^{(1)} := (v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ and $v^{(2)} := (v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ with $F(v^{(1)}) \neq F(v^{(2)})$, then

(a) $v^{(1)} \in DC$ or $v^{(2)} \in DC$ and
(b) the SMT solver detects and records this don't care when solving formula (1).

In order to show fact (a), we consider the following assignment to the boolean abstraction variables in formula (1): Let $q_{\ell_1} := v_{\ell_1}, \ldots, q_{\ell_r} := v_{\ell_r}, q_{\ell'_1} := v'_{\ell_1}, \ldots, q_{\ell'_r} := v'_{\ell_r}$, $q_{\ell_{r+1}} := q_{\ell'_{r+1}} := v_{\ell_{r+1}}, \ldots, q_{\ell_n} := q_{\ell'_n} := v_{\ell_n}$. (Thus $v^{(1)}$ is assigned to the abstraction variables for $\ell_1, \ldots, \ell_n$ and $v^{(2)}$ to the abstraction variables for $\ell'_1, \ldots, \ell'_n$.) It is easy to see that this assignment satisfies the boolean abstraction of formula (1). Since formula (1) is unsatisfiable, the assignment has to be inconsistent 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 ℓ_i and ℓ'_i are based on disjoint sets of real variables $C = \{c_1, \ldots, c_f\}$ and $C' = \{c'_1, \ldots, c'_f\}$, already the partial assignment to $\ell_1, \ldots, \ell_n$ or the partial assignment to $\ell'_1, \ldots, \ell'_n$ has to be inconsistent, i.e., $v^{(1)} \in DC$ or $v^{(2)} \in DC$.

Fact (b) follows from the simple observation that the SMT solver has to detect and record the inconsistency of the assignment mentioned above in order to prove unsatisfiability of formula (1) and with minimization of conflict clauses it detects only conflicts which are confined either to $\ell_1, \ldots, \ell_n$ or to $\ell'_1, \ldots, \ell'_n$.[4]

Altogether this means that the elements of some $\mathit{orbit}(v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ which are not in the subset DC' of DC computed by the SMT solver are either all mapped by F to 0 or are all mapped by F to 1. Thus, we can define an appropriate function G by don't care assignment as follows:

1. If $\mathit{orbit}(v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \subseteq DC'$, then $G(v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ is chosen arbitrarily.
2. Otherwise $G(v_{\ell_{r+1}}, \ldots, v_{\ell_n}) = \delta$ with $F(\mathit{orbit}(v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \setminus DC') = \{\delta\}$, $\delta \in \{0,1\}$.

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 $|F(\mathit{orbit}(v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \setminus DC')| = 1$), i.e., G is a possible solution according to Def. 1. This consideration also provides a proof for the 'if-part' of Thm. 1.

[4] 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 $\ell'_1, \ldots, \ell'_n$.

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.

Note that according to case 1. of the definition above there may be several possible choices fulfilling the definition of G.

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 $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ to 1, whenever the orbit contains an element ϵ with $dc(\epsilon) = 0$ and $F(\epsilon) = 1$. Since due to the argumentation above there is no other element δ 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 $\{0,1\}^{n-r}$ to 1).

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, 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.[5] 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 [1,8,9,19] 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 [9].

Don't cares can be assigned arbitrarily in order to make G independent from $q_{\ell_1}, \ldots, q_{\ell_r}$, thus our task is to find a boolean function $G(q_{\ell_{r+1}}, \ldots, q_{\ell_n})$ with

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

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

Now let $A(q_{\ell_1}, \ldots, q_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n}, h_1, \ldots, h_l)$ represent the CNF for a Tseitin transformation [20] of $(F \wedge \overline{dc})(q_{\ell_1}, \ldots, q_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n})$ (with new auxiliary

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

variables $h_1, \ldots, h_l$). Likewise, let $B(q'_{\ell_1}, \ldots, q'_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n}, h'_1, \ldots, h'_{l'})$ be the CNF for a Tseitin transformation of $(\overline{F} \wedge \overline{dc})(q'_{\ell_1}, \ldots, q'_{\ell_r}, q_{\ell_{r+1}}, \ldots, q_{\ell_n})$ (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 '$A \wedge B = 0$' for Craig interpolation [1,8]:

Suppose that there is a satisfying assignment to $A \wedge B$ given by $q_{\ell_1} := v_{\ell_1}, \ldots, q_{\ell_r} := v_{\ell_r}, q'_{\ell_1} := v'_{\ell_1}, \ldots, q'_{\ell_r} := v'_{\ell_r}$, $q_{\ell_{r+1}} := v_{\ell_{r+1}}, \ldots, q_{\ell_n} := v_{\ell_n}$, 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 $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ would contain two elements $(v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ and $(v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ which do not belong to the don't care set DC' and which fulfill $F(v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) = 1$ and $F(v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) = 0$. This is a contradiction to the property shown above that the elements of $orbit(v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ 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 has the following properties [1,8]:

- It depends only on common variables $q_{\ell_{r+1}}, \ldots, q_{\ell_n}$ of A and B,
- $A \Longrightarrow G$, i.e., G fulfills equation (2), and
- $G \wedge B$ is unsatisfiable, or equivalently, $G \Longrightarrow \overline{B}$, 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. According to [8,9] a Craig interpolant can be computed in linear time based on a proof by resolution that a formula in CNF (in our case $A \wedge B$ as defined above) is unsatisfiable. Such proofs can be computed by any modern SAT solver with proof logging turned on.

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 [13] 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 ([4,5], 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 MathSAT [15], since it provides a method for extracting conflict clauses due to inconsistent assignments to linear constraints. The computation of the Craig interpolants is done with MiniSAT [21], 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.

Comparison of the LinAIG evolution with and without redundancy removal. In Fig. 4 we present a comparison of two typical runs of the model checker from [5]. 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

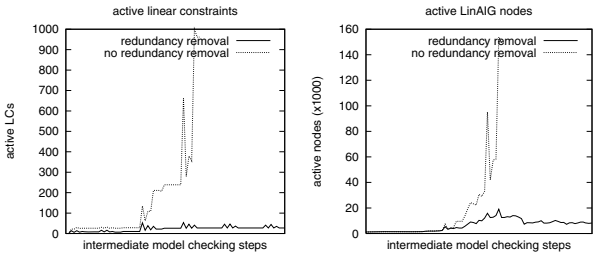

Fig. 4. Comparison of the LinAIG evolution with and without redundancy removal

redundancy removal, the number of linear constraint is quickly increasing up to 1000 and more, and the number of LinAIG 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 behavior has an immediate effect on the run times of the model checker: Whereas model checking finished within 15 minutes with redundancy removal, the version without redundancy removal did not finish within a timeout of 24 hours. 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.

Existential quantification vs. Craig interpolation. Here we evaluate the two different approaches to removal of redundant constraints as presented in Sect. 3.3. The first approach 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 from [5] during three runs with different model checking problems (the first group of runs was taken from an industrial case study considering a flap controller of an aircraft [5], the following two groups from the verification of collision avoidance protocols for train applications [22]). All these problems also contain boolean variables.

The results are given in Table 1. The numbers of 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 'Δ 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

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. Δ nodes	RR exist. quant. time (s)	RR Craig interp. Δ nodes	RR Craig interp. time (s)
stateset_1-1	1459	41	22	0.22	0.35	-541	7.19	-814	0.74
stateset_1-2	1716	74	27	0.51	0.71	-313	13.14	-1047	0.68
stateset_1-3	2340	105	22	1.89	2.41	459	25.35	-515	2.32
stateset_1-4	3500	142	28	8.02	5.53	1642	75.49	1062	10.08
stateset_1-5	2837	123	13	4.61	4.54	-230	12.34	1595	23.10
stateset_2-1	824	29	8	0.12	0.19	747	2.55	-142	0.43
stateset_2-2	1424	37	10	0.36	0.53	1104	3.45	233	1.19
stateset_2-3	3048	52	11	2.45	2.50	1996	10.13	171	4.22
stateset_2-4	1848	37	14	0.57	0.90	852	4.03	-149	1.34
stateset_3-1	495	74	44	0.10	0.14	656	4.55	-365	0.13
stateset_3-2	1775	297	228	1.86	1.74		>7200	-1453	1.60
stateset_3-3	6703	1281	1143	105.69	23.70		>7200	-5805	14.12
stateset_3-4	32021	5943	5706	2774.10	1012.78		>7200	-24633	126.19

to see that using incremental SMT solving techniques it was in many cases possible to detect large sets of redundant linear constraints in very short times. As shown in the previous experiment this pays off also in subsequent steps of model checking when quantifier elimination works on a representation with a smaller number of linear constraints. Considering column 6 we observe that run times for the generation of don't care sets by MathSAT are comparable to the run times of redundancy *detection*.[6]

Comparison of the LinAIG based quantifier elimination with other solvers. In a last experiment we compared our approach to quantifier elimination with several existing tools: REDLOG [12] is an extension to the computer algebra system REDUCE [11] and uses the quantifier elimination algorithm of Loos and Weispfenning [7], too, LIRA 1.1.2 [10] is an automata-based tool capable of representing sets of states over real, integer, and boolean variables, and CVC3 1.2.1 [14] as well as Yices 1.0.11 [13] are state-of-the-art SMT solvers.

The benchmarks formulas used in this experiment contain linear constraints and boolean variables, together with AND operators, negations, and quantifiers over real-valued variables. The formulas were extracted from the model checker [5] and represent continuous pre-image computations for state sets. All formulas contain two quantified variables, one is existentially quantified and the other one is universally quantified. All formulas are given in the SMT-LIB format [23] and are publicly available.[7] Since the SMT-LIB format only supports flat formulas (instead of shared graph structures), we had to confine ourselves to state set representations with moderate sizes.

For the SMT solvers we interpret free variables as implicitly existentially quantified and decide satisfiability, since they do not compute predicates representing all

[6] 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 MathSAT in order to be able to extract conflict clauses.

[7] `http://abs.informatik.uni-freiburg.de/tacas09bench/`

Table 2. Comparison of Solvers

Benchmark					LinAIG					REDUCE/REDLOG				LIRA		Yices		CVC3	
Name	AND	LC	B	R	N	AND	LC	B	Time	AND	LC	B	Time	Res.	Time	Res.	Time	Res.	Time
pre1	1K	22	5	4	27	30	12	5	1.48	830	26	5	0.14	SAT	27.71	?	0.03	?	0.22
pre2	2K	20	5	4	15	15	2	4	1.23	1133	35	5	0.21	SAT	67.98	?	0.05	?	0.25
pre3	5K	26	5	4	55	71	16	5	2.03	1918	39	5	0.43	SAT	443.66	?	0.15	?	0.40
pre4	160K	52	4	4	33	35	12	3	4.48	196483	224	4	38.06	timeout		?	3.94	?	4.65
pre5	188K	27	20	5	31	59	13	4	4.52	35356	42	16	14.67	memout		?	4.41	?	4.84
pre6	1396K	31	20	5	21	29	9	3	15.15	396887	68	20	98.89	timeout		?	32.88	?	22.81
pre7	3894K	30	20	5	32	111	8	4	46.58	memout				memout		?	148.47	?	90.97
pre8	6730K	44	20	5	186	545	14	12	68.95	memout				memout		?	239.20	?	111.56
pre9	9931K	52	8	4	555	8034	20	8	96.19	memout				memout		?	191.67	?	132.06

satisfying assignments. Our LinAIG based tool, REDLOG/REDUCE and LIRA additionally compute representations for predicates representing all satisfying assignments. Again, we used a time limit of 7200 CPU seconds and a memory limit of 4 GB for our experiments.

Table 2 shows the results. The first section 'Benchmark' shows details on the input formulas. The column 'AND' lists the number of AND operators in the formula, 'LC' lists the number of linear constraints, 'B' the number of boolean variables, and 'R' the number of real variables. The LinAIG section shows for the resulting predicates the numbers of AIG nodes ('N'), the numbers of AND operators in the corresponding flat formula ('AND'), the numbers of linear constraints ('LC'), boolean variables ('B'), and run times ('Time'). The run times include all CPU times necessary for reading the formulas, constructing LinAIGs, eliminating quantifiers, and removing redundant linear constraints. For the REDLOG tool we report the number of AND operators, linear constraints, and boolean variables in the resulting formula, as well as the run times needed for the computation. For LIRA, Yices and CVC3 run times are given together with the result whether the formula is proven to be satisfiable ('SAT') or unsatisfiable ('UNSAT'). The two SMT solvers Yices and CVC3 are not using a complete method and therefore may also report 'unknown' which is marked by '?'.

Our LinAIG based approach is able to eliminate the quantifiers of all formulas within a short runtime and moreover, returns formulas which are much more compact than the formulas produced by REDLOG/REDUCE, both in terms of AND operators and in terms of linear constraints. For mid-size examples the run times of REDUCE/REDLOG are outperformed by our tool, whereas the larger examples could not be solved by REDUCE/REDLOG. LIRA was able to solve only 3 out of 9 instances within the time or memory limit and it needs much more run time. The SMT solvers Yices and CVC3 were not able to solve any of the examples. 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 [24] and for handling formulas with quantifiers they make use of heuristics based on E-matching [25] which are not tuned to problems that contain only linear arithmetic.

In summary, the experiments clearly demonstrate that for the subclass of formulas considered in this paper we were able to provide an efficient method both wrt. run times and wrt. the sizes of the resulting formulas.

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. The method is based on an elaborate scheme for keeping graph-based representations of intermediate results as compact as possible, with redundancy removal as an essential component. 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 J. Eisinger from the University of Freiburg for providing the formula parser used in our experiments and A. Griggio from University of Trento for his support enabling the integration of MathSAT into our tool.

References

1. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. Journal on Symbolic Logic 22(3), 269–285 (1957)
2. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer 1(1–2), 110–122 (1997)
3. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005)
4. Damm, W., Disch, S., Hungar, H., Pang, J., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Automatic verification of hybrid systems with large discrete state space. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 276–291. Springer, Heidelberg (2006)
5. 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)

6. Wang, F.: Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. IEEE Trans. on Software Engineering 31(1), 38–52 (2005)
7. Loos, R., Weispfenning, V.: Applying linear quantifier elimination. The Computer Journal 36(5), 450–462 (1993)
8. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. Journal on Symbolic Logic 62(3), 981–998 (1997)
9. 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)
10. Eisinger, J., Klaedtke, F.: Don't care words with an application to the automata-based approach for real addition. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 67–80. Springer, Heidelberg (2006)
11. Griss, M.L.: The reduce system for computer algebra. In: ACM 1975: Proceedings of the 1975 annual conference, pp. 261–262. ACM Press, New York (1975)
12. Dolzmann, A., Sturm, T.: Redlog: computer algebra meets computer logic. SIGSAM Bull. 31(2), 2–9 (1997)
13. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
14. Stump, A., Barrett, C.W., Dill, D.L.: CVC: A cooperating validity checker. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 500–504. Springer, Heidelberg (2002)
15. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MATH-SAT 4 SMT solver. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 299–303. Springer, Heidelberg (2008)
16. Mishchenko, A., Chatterjee, S., Jiang, R., Brayton, R.K.: FRAIGs: A unifying representation for logic synthesis and verification. Technical report, EECS Dept., UC Berkeley (2005)
17. Pigorsch, F., Scholl, C., Disch, S.: Advanced unbounded model checking by using AIGs, BDD sweeping and quantifier scheduling. In: FMCAD, pp. 89–96 (2006)
18. Sebastiani, R.: Lazy satisability modulo theories. JSAT 3, 141–224 (2007)
19. Lee, C.C., Jiang, J.H.R., Huang, C.Y., Mishchenko, A.: Scalable exploration of functional dependency by interpolation and incremental SAT solving. In: ICCAD, pp. 227–233 (2007)
20. Tseitin, G.: On the complexity of derivations in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logics (1968)
21. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 541–638. Springer, Heidelberg (2004)
22. Damm, W., Mikschl, A., Oehlerking, J., Olderog, E.-R., Pang, J., Platzer, A., Segelken, M., Wirtz, B.: Automating verification of cooperation, control, and design in traffic applications. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) Formal Methods and Hybrid Real-Time Systems. LNCS, vol. 4700, pp. 115–169. Springer, Heidelberg (2007)
23. Ranise, S., Tinelli, C.: The SMT-LIB Standard: Version 1.2 (2006), `http://combination.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf`
24. Barrett, C., Deters, M., Oliveras, A., Stump, A.: Satisfiability Modulo Theories Competition (SMT-COMP) 2008: Rules and Precedures (2008), `http://smtcomp.org/rules08.pdf`
25. Detlefs, D., Nelson, G., Saxe, J.: Simplify: A theorem prover for program checking. J. ACM 52(3), 365–473 (2005)