Quantifier elimination by lazy model enumeration[*]

David Monniaux
CNRS / VERIMAG [†]

April 15, 2010

Abstract

We propose a quantifier elimination scheme based on nested lazy model enumeration through SMT-solving, and projections. This scheme may be applied to any logic that fulfills certain conditions; we illustrate it for linear real arithmetic. The quantifier elimination problem for linear real arithmetic is doubly exponential in the worst case, and so is our method. We have implemented it and benchmarked it against other methods from the literature.

1 Introduction

Quantifier elimination consists in transforming a quantified formula into an equivalent quantifier-free formula. For instance, the formulas $\forall y\ (y - z \geq x \Rightarrow x + z \geq 1)$ and $x \geq 1 - z$ are equivalent (they have the same models for (x, z)), whether considered over the reals or integers. Quantifier elimination subsumes both satisfiability testing for quantifier-free formulas, and the decision of quantified formulas without free variables. In program analysis, quantifier elimination has been applied to obtain optimal invariants and optimal abstract transformers [21, 20], and to obtain preconditions for modular assertion checking [22].

Unfortunately, quantifier elimination tends to be slow; as recalled in §4, worst-case complexities for useful theories tend to be towers of exponentials. Yet, high worst-case complexity does not preclude exploring procedures that perform fast on most examples, as shown by the high success of SAT solving. This motivates our work on new quantifier elimination algorithms.

Many interesting mathematical theories admit quantifier elimination. In order to introduce better elimination schemes, we shall first describe a naive, but inefficient algorithm (§2.2) which works by calling a *projection* operator,

[*]This work was partially funded by ANR project "ASOPT".

[†]VERIMAG is a joint laboratory of CNRS, Université Joseph Fourier and Grenoble-INP.

that is, an algorithm taking as an input a conjunction C of literals of the theory, and a list of variables $x_1, \ldots, x_n$, and outputting a formula equivalent to $\exists x_1, \ldots, x_n\ C$. Examples of theories with projection operators include nonlinear complex arithmetic (also known as the theory of algebraically closed fields), using Gröbner bases [9]; linear real arithmetic, using Fourier-Motzkin elimination [15, §5.4] or more advanced polyhedral projection techniques; and linear integer arithmetic, also known as Presburger arithmetic, using the Omega test [23].

Nonlinear integer arithmetic, also known as Peano arithmetic, is undecidable. However, nonlinear (polynomial) real arithmetic[1] admits quantifier elimination. The best known general algorithms construct a *cylindrical algebraic decomposition* of the polynomials present in the atoms of the formula; once this costly decomposition is obtained, the quantifier elimination is trivial [6, 2]. We therefore exclude these two theories from our study.

This article provides two contributions. First, it describe an algorithm that uses both projection and satisfiability testing modulo the chosen theory, and illustrate it with linear real arithmetic. This algorithm performs nested satisfiability tests, with lazy generation of constraints. Second, we improve on the worst case complexity bounds for an earlier algorithm [19], which are also valid for the new one.

In §2, we give a short introduction to quantifier elimination techniques over linear real arithmetic, and the idea of lazy constraint generation. In §3, we describe our algorithm, and we prove that its complexity is at most doubly exponential in §4. Finally, in §5 we provide benchmarks.

2 Previous State of the Art

Let us first recall some vocabulary on formulas. We shall then summarize previous work on quantifier elimination on linear real arithmetic, most notably our eager projection method (§2.2). We propose a lazy projection method using ideas of *lazy constraint generation*; §2.3 gives examples of such techniques in other applications.

2.1 Formulas

We consider quantifier-free formulas written using $\wedge$, $\vee$ and $\neg$ connectors, as well as literals (atoms or negation thereof). A formula written without $\neg$ except just around an atom is said to be in *negation normal form* (NNF), a formula consisting in a disjunction of conjunctions of literals is in *disjunctive normal form* (DNF), a formula consisting in a conjunction of disjunctions of literals is in *conjunctive normal form* (CNF).

[1]Also known as the theory of real closed fields.

We shall mostly focus on the case where the atoms are of the form $\sum_i a_i x_i \leq b$ where the a_i and b are constant rational numbers and the x_i are real variables. A model of a formula F is an assignment to the x_i such that the formula is satisfied; we then note $(x_1, \ldots, x_n) \models F$. We say that two formulas F and G are equivalent, noted $F \equiv G$, if they have the same models. We say that F implies G, noted $F \Rrightarrow G$, if all models of F are models of G. Formulas without free variables are equivalent to true or false. A *decision procedure* provides this truth value given such a formula. Obviously, a quantifier elimination procedure may be used as a decision procedure, since it will turn any formula into an equivalent formula without quantifiers or variables, thus trivially checkable.

We add to this language the $\forall$ and $\exists$ quantifiers. The definitions for models, equivalence and implication are the same as above, except that models only assign values to the free variables of the formula. Quantifier elimination consists in obtaining an equivalent quantifier-free formula from a quantified formula.

There exist two major classes of algorithms for quantifier elimination over arithmetic. One is based on substitution: an infinite disjunction $\exists x\ F$ is shown to be equivalent to a finite disjunction $F[x_1/x] \vee \cdots \vee F[x_n/x]$ where the x_i are functions of the free variables in F constructed by examination of the atoms in F. For linear real arithmetic, Ferrante and Rackoff's method [12] and Loos and Weispfenning's method [16] belong to this class, and so does Cooper's method for Presburger arithmetic [7]. Other methods, more geometrical in kind, project conjunctions of atoms and thus need some form of conversion to DNF; such is the case of Fourier-Mozkin elimination for linear real arithmetic, and of Pugh's Omega test for Presburger arithmetic [23]. Our methods belong to that latter class.

2.2 Eager Model Enumeration Algorithm

It is easy to see that if there is an algorithm π for eliminating quantifiers from formulas of the form $\exists x_1, \ldots, x_n\ F$ where C is a quantifier-free conjunction of literals, then there is an algorithm, albeit an inefficient one, for eliminating quantifiers from any formula.

We reduce ourselves to the case of eliminating the existential quantifier from $\exists x_1, \ldots, x_n\ F$ where F is quantifier-free. We handle an existentially quantified formula $\exists x_1, \ldots, x_n\ F$ as follows: convert F to DNF $F_1 \vee \cdots \vee F_m$; the formula is then equivalent to $(\exists x_1, \ldots, x_n\ F_1) \vee \cdots \vee (\exists x_1, \ldots, x_n\ F_m)$, and thus to $\pi(\exists x_1, \ldots, x_n\ F_1) \vee \cdots \vee \pi(\exists x_1, \ldots, x_n\ F_m)$. In the simplest form, π can be performed by the Fourier-Motzkin algorithm, and conversion to DNF by repeat application of the distributivity of $\wedge$ over $\vee$.

Can we do better? A more efficient way to convert to DNF is to use an "all-SAT" approach within a *satisfiability modulo theory* (SMT) solver. Given a quantifier-free formula F over linear real arithmetic, an SMT-solver will either answer “unsat”, in which case F is unsatisfiable, or provide a model for F — that is, a valuation for all the free variables such that F is satisfied. Equivalently, an SMT-solver may provide truth values for all atoms in F such that all valuations of the free variables of F for which the atoms in F have these truth values are models; in other words, it provides a conjunction C of literals from the atoms of F such that C implies F.

In order to convert a formula F to DNF, we run an SMT-solver over it. If it answers “unsat”, we are done. Otherwise, it provides a conjunction of literals C_1 such that $C_1 \Rightarrow F$. Run the SMT-solver over $F \wedge \neg C_1$. If it answers “unsat”, we are done. Otherwise, it provides a conjunction of literals C_2 such that $C_2 \Rightarrow F$. Run the SMT-solver over $F \wedge \neg C_1 \wedge \neg C_2$, etc. This algorithm terminates, because there is a finite number of atoms and thus a finite number of conjunctions C_i that can be built out of them, and the same conjunction cannot occur twice. At the end $C_1 \vee C_2 \vee \ldots$ is a DNF form for F.

There is still room for improvement. Consider the formula defining the vertices of a n-dimensional hypercube $F \triangleq (x_1 = 0 \vee x_1 = 1) \wedge \cdots \wedge (x_n = 0 \vee x_n = 1)$ and compare with the result of the quantifier elimination $\exists x_2, \ldots, x_n\ F \equiv x_1 = 0 \vee x_1 = 1$. Certainly it seems excessive to enumerate the 2^{n-1} disjuncts of the DNF of F whereas the final result only has 2 disjuncts.

We therefore suggested another improvement [19]. Instead of adding $\neg C_i$ to the constraints of the system, we add $\neg\pi(C_i)$. With this method, the number of calls to the SMT-solver is not the size of the DNF of F, but the size of the DNF for the eliminated form of $\exists v_1, \ldots, v_m\ F$.

This algorithm has a weakness: when applied to nested quantifiers, for instance, $\exists x_1 \forall x_2 \exists x_3\ F$, it will compute a full DNF for $\exists x_3\ F$, then a full CNF for $\forall x_2 \exists x_3\ F$, prior to computing the DNF for the full formula, and it will do so even if most conjuncts or disjuncts are actually useless. Consider for instance the following example:

$$F \triangleq \exists x \forall y \exists z\ z \geq 0 \wedge (x \geq z \wedge y \geq z \vee y \leq 1 - z) \tag{1}$$

This formula was produced by adding an extra z to $(x \geq 0 \wedge y \geq 0) \vee y \leq 1$, which is equivalent to $x \geq 0 \vee y \leq 1$.

Let us see how the eager algorithm performs on F. First, $\exists z\ z \geq 0 (x \geq z \wedge y \geq z \vee x \geq z - 1 \wedge y \geq 1 - z)$ is turned to DNF: $F_2 \triangleq (x \geq 0 \wedge y \geq 0) \vee y \leq 1$ or, perhaps with some better algorithm, $x \geq 0 \vee y \leq 1$. Then, $\forall y\ F_2$ is turned to CNF, that is, $F_1 \triangleq x \geq 0$, and then $\exists x\ F_1$ is turned into true. Now consider that instead of F_2, we had taken $F_2' \triangleq x \geq 0$; clearly $F_2' \Rightarrow F_2$. $\forall y\ F_2'$ is then $x \geq 0$. In short, instead of computing a full DNF for F_2' we could have simply computed one term of it. This motivates our lazy algorithm.

2.3 Lazy Constraint Generation in Other Contexts

In short, when looking for a model for variable x of $\forall y \exists z\ F$, each disjunct in the DNF of $\exists z\ F$ is an additional constraint over x, but we do not wish to generate the full list of these constraints because some of them may not be actually needed. The idea of our algorithm is to try to solve the already known constraints, find a tentative solution, and find if this solution violates some yet unknown constraint; if so, we add this constraint to the system and resume our search for a solution. Before describing a formal version of the algorithm, we wish to note that lazy constraint generation approaches are already used in other contexts, in order to better convey the intuition of the method.

In operational research, it is not uncommon for constrained optimization problems to be specified using a very large number of constraints, so large that explicitly taking them all into account at once would be impractical. New constraints are "discovered" when the proposed solution violates them. In linear programming, such technique is known as *delayed column generation.* As early as 1954, it was observed that it was possible to solve large instances of the traveling salesman problem by dynamically generating the inequalities that a solution should satisfy, the full set of inequalities being "astronomically" large [10].

Almost all SMT-solving systems proceed by Boolean relaxation: in order to decide whether a formula F is satisfiable, they first replace all non-propositional atoms by propositional variables, using a dictionary, then solve the resulting system using SAT. If the resulting propositional system is unsatisfiable, then so is the original problem. If it is satisfiable, it is possible that the Boolean solution is absurd with respect to the theory: for instance, if it assigns `true` to the propositional variables corresponding to the atoms $x > 1$, $y > 3$, and $x + y < 0$. If this is the case, an additional Boolean constraint is added to the problem, excluding this inconsistent assignment (or, for better efficiency, an inconsistent generalization of this assignment). In short, we generate on demand, or *lazily*, the theory of the atoms of F (the absurd conjunctions of atoms of F), because an eager approach would generate an exponential number of Boolean constraints [15, §11.2].

Some recent proposals for SMT-solving over linear real arithmetic [17, 8] do not use Boolean relaxation. Instead, they try solving the formula directly for the real variables: for a problem over x, y and z, if they realize that after choosing $x = x_0$ and $y = y_0$, there is no suitable z (once x and y are chosen, the solution set for z can be computed as a intervals), they deduce a constraint on x and y that excludes (x_0, y_0). When solving for x, y, constraints on x may be obtained. This approach has similarities to what we would obtain by applying the ideas of §3 to a $\exists x_1 \exists x_2 \dots \exists x_n\ F$ formula.

In quantified Boolean solving for formulas of the form $\forall b_1, \dots, b_m \exists c_1, \dots, c_n\ F$ (2QBF), some proposed approaches [24, Alg. I] use two successive layers of

SAT solvers, with the inner solver solving for $b_1, \ldots, b_m, c_1, \ldots, c_n$, initially for formula F, and the outer solver for $c_1, \ldots, c_n$, initially for formula true, with new constraints being lazily generated and accumulated into the outer solver. The algorithm we present in §3 can be understood as a generalization of this algorithm to arbitrary theories and arbitrary quantification depths.

3 Lazy Model Enumeration Algorithm

We shall now describe our lazy algorithm, instantiated on linear real arithmetic (first, the generalization sub-algorithm, then the main algorithm), and prove its correctness. Then we shall briefly investigate possibilities of extension.

3.1 Generalization Algorithm

Algorithm 1 GENERALIZE(C_0, *test*): given a set S_0 and a monotonic Boolean function *test* such that $test(S_0)$ is true, return $S \subseteq S_0$ such that $test(S)$ is true and S is minimal.

Require: $(S_0, test)$
Require: *test* is a function that takes as input a set S of literals and answers true or false. It is required to be monotonic: if $S_1 \subseteq S_2$, and $test(S_1)$ is true, then so is $test(S_2)$.
Require: S_0 is a set of literals such that $test(S_0)$ is true.
  $S := S_0$
  while $S \neq \emptyset$ **do**
    Choose a literal c in S
    if $test(S \setminus \{c\})$ is true **then**
      $S := S \setminus \{c\}$
    end if
  end while
Ensure: $test(S)$ is true
Ensure: $S \subseteq S_0$

This description intentionally omits a precise criterion for the choice of c, since the algorithm is correct regardless.

At some point during the course of the main algorithm, we shall generate a conjunction $C_1 \wedge \cdots \wedge C_n$ that implies a formula F, but for efficiency we would prefer a conjunction of fewer terms $C_{i_1} \wedge \cdots \wedge C_{i_n}$ that still implies F. In other words, we wish to generalize the conjunction $C_1 \wedge \cdots \wedge C_n$ under the condition that $C_1 \wedge \cdots \wedge C_n \Rrightarrow F$. Ideally, we wish the subset $\{C_{i_1}, \ldots, C_{i_n}\}$ to be minimal.

Our condition is equivalent to $C_1 \wedge \cdots \wedge C_n \wedge \neg F$ being a contradiction; thus, from a set of constraints $\{C_1, \ldots, C_n, \neg F\}$ we wish to extract a minimal contradictory set, or, in the terminology of operation research, a *irreducible infeasible subset*. The simplest algorithm for doing so is the *deletion filter* [4, 5]. A difference is that in operation research contexts, all constraints are inequalities, while in our case, formula F is in general complex, with disjunctions. In fact, we do not even want to explicitly write formula F — this is the difference with our earlier eager algorithm. Instead, we use a function *test* that takes as input a set $S \triangleq \{C_1, \dots, C_n\}$ of literals, and answers "true" if and only if $\neg F \wedge C_1 \wedge \cdots \wedge C_n$ is unsatisfiable, thus leading to Alg. 1. This procedure merely relies on *test* being monotonic as a function from the sets of literals, ordered by inclusion, to the Booleans.

Procedure Generalize can be replaced by a more clever, "divide-and-conquer" approach, as in the min function in [3] (or the equivalent QuickXPlain from [14]). While this procedure is theoretically better, making fewer calls to *test*, it performs worse in practice (§5). In our case, *test* is a complex procedure, possibly making use of multiple layers of quantifier elimination and SMT-solving, all of which use caches; thus, the cost of multiple calls does not depend solely on the number of calls but also on the relationships between the calls.

3.2 Main Algorithm

If B is a set $\{v_1, \dots, v_n\}$ of variables, we denote by $\forall_B F$ the formula $\forall v_1 \dots \forall v_n\ F$ (and respectively for $\exists_B F$). In all our algorithms and reasoning, the order of the bound variables inside these "block quantifiers" will not matter, thus the notation is justified. For technical reasons, we allow empty quantifier blocks ($\forall_\emptyset$ and $\exists_\emptyset$). We note $\neg^n F$ the formula F if n is even, $\neg F$ if n is odd. We note $\mathrm{FV}(F)$ the set of free variables of formula F.

We consider a formula F_0 in prenex form, with alternating quantifier blocks: $\forall_{B_0} \exists_{B_1} \forall_{B_2} \dots \neg^n F_n$. Without loss of generality, we can suppose that the B_i have pairwise empty intersection. Any quantified formula can be converted to this form with $\forall i > 0\ B_i \neq \emptyset$, but possibly with $B_0 = \emptyset$. More precisely, we note, for $0 \le i < n$, $F_i = \forall_{B_i} \neg F_{i+1}$.

π_i is a quantifier elimination procedure for conjunctions: from a conjunction C it returns another conjunction C' such that $C' \equiv \exists_{B_i} C$; for linear real arithmetic, Fourier-Motzkin elimination or more clever methods of polyhedral projection are suitable.[2] It is not necessary that C' be a conjunction for

[2] Fourier-Motzkin elimination directly obtains a set of inequalities defining the projected polyhedron, but it may create many unnecessary ones and it is thus often necessary to run tests for removing useless ones. Such tests are emptyness tests for polyhedra defined by constraints; these can be performed using the simplex algorithm implemented in exact rational arithmetic. Alternatively, methods based on the "double representation" of polyhedra first compute the set of vertices of the polyhedron (which may be exponential, for instance for a hypercube $[0, 1]^n$), project them (a trivial operation) then compute the facets of the resulting polyhedron. See [1] for a bibliography on polyhedral algorithms. Our implementation uses an off-the-shelf polyhedron library based on double representation;

most of our algorithm, except for the GENERALIZE procedure (this restriction will be lifted in §3.5).

The main algorithm is the function Q-TEST(i, C). It tests whether $F_i \wedge C$ is satisfiable, and if it is, it proposes a conjunction C' of literals such that $\mathrm{FV}(C') \subseteq \mathrm{FV}(F_i)$, $C' \Rrightarrow F_i$, and $C \wedge C'$ is satisfiable. It is defined by induction over $n - i$ for $0 \leq i \leq n$.

The case $i = n$ corresponds to is merely SMT-solving and generalization. We note SMT-TEST(C, F) the SMT-solver function, which takes as inputs two formulas C and F and returns a couple (b, C'). b is a Boolean, which states whether $C \wedge F$ is satisfiable. If b is true, C' is an "extended model": a conjunction of literals of F such that $C' \Rrightarrow F$ and $C \wedge C'$ is satisfiable. Such a function can be obtained from an ordinary SMT-solver providing satisfiability models as follows: get a model $M \vDash C \wedge F$, then set C' as the conjunction of all the atoms a of F such that $M \vDash a$ and of the negation of all the atoms a of F such that $M \nvDash a$; alternatively, some SMT-solvers directly output such a conjunction.

The recursive case for $i < n$ is defined by calling the recursive case for $i+1$. Let us begin by some intuition of the workings of the algorithm. Recall that $F_i \triangleq \forall_{B_i} \neg F_{i+1}$; thus, if we had a DNF $D_1 \vee \cdots \vee D_l$ of F_{i+1}, we could turn it immediately into a CNF of F_i: $(\neg \pi_i(D_1)) \wedge \cdots \wedge (\neg \pi_1(D_l))$. Our goal is to test whether $C \wedge F_i$ is satisfiable, which is equivalent to testing whether the set of constraints $\{C, \neg \pi_i(D_1), \ldots, \neg \pi_1(D_l)\}$ is satisfiable. Instead of computing all these constraints, then solving them, which is what our eager algorithm does, we wish to compute them "as needed".

The constraints that have already been computed at level i are stored as a current formula M_i (in practice, the current constraint state of an SMT-solver), initialized to `true`. Each of these formulas, for $0 < i < n$, satisfies two invariants: $\mathrm{FV}(M_i) \subseteq \mathrm{FV}(F_i)$ and $F_i \Rrightarrow M_i$. Intuitively, if the output of π_i is always a conjunction, M_i is a "partial CNF" for F_i. At worst, the algorithm completes it into a full CNF for F_i.

The algorithm at level $i < n$ works as follows. It first tests satisfiability with respect to the already computed constraints: whether $C \wedge M_i$ is satisfiable; if it is not, then *a fortiori* $C \wedge F_i$ is not and the answer is immediate. Otherwise, we obtain an extended model C' of $C \wedge M_i$. We however do not know yet whether it is an extended model of $C \wedge F_i$; this is the case if and only if $C' \wedge F_{i+1}$ is unsatisfiable. We thus perform a recursive call to Q-TEST at level $i + 1$:

- If this call answers that $C' \wedge F_{i+1}$ is unsatisfiable, we could immediately return C' as a correct generalized model. Yet, C' might not be general enough: we would prefer to extract from it a minimal conjunction C'_{min} such that $C'_{\mathrm{min}} \wedge F_{i+1}$ is still unsatisfiable; thus the call to GENERALIZE. GENERALIZE has to test the satisfiability of various formulas of the

profiling has shown that the choice of the projection algorithm did not matter much [19].

Algorithm 2 Q-$\textsc{Test}(i, C)$: satisfiability testing for $F_i \wedge C$

Require: (i, C) such that $0 \le i \le n$, $\mathrm{FV}(C) \subseteq \mathrm{FV}(F_i)$
  if $i = n$ **then**
    $(b', C') :=$ SMT-$\textsc{Test}(C, F_n)$
    if b' is false **then**
      return $(\mathsf{false}, \mathsf{false})$
    else
      return $(\mathsf{true}, \textsc{Generalize}(C', K \mapsto \neg \mathit{first}(\text{SMT-}\textsc{Test}(K, F_n))))$
    end if
  else
    while true do
      $(b', C') :=$ SMT-$\textsc{Test}(C, M_i)$
      if b' is false **then**
        return $(\mathsf{false}, \mathsf{false})$ {Since $F_i \Rrightarrow M_i$, then $C \wedge F_i$ is unsatisfiable too}
      else
        $(b'', C'') :=$ Q-$\textsc{Test}(i + 1, C')$
        if b'' is false **then**
          {$C' \wedge F_{i+1}$ is unsatisfiable}
          return $(\mathsf{true}, \textsc{Generalize}(C', K \mapsto \neg \mathit{first}(\text{Q-}\textsc{Test}(i+1, K))))$
        else
          {C'' is such that $\mathrm{FV}(C'') \subseteq \mathrm{FV}(F_{i+1})$, $C'' \Rrightarrow F_{i+1}$ and $C' \wedge C''$ satisfiable. Thus $\exists_{B_i} C'' \Rrightarrow \exists_{B_i} F_{i+1}$, whence $F_i = \forall_{B_i} \neg F_{i+1} \Rrightarrow \neg \exists_{B_i} C'' \equiv \neg \pi_i(C'')$}
          $M_i := M_i \wedge \neg \pi_i(C'')$
        end if
      end if
    end while
  end if

Ensure: The return value is a pair (b, C'). b is a Boolean stating whether $F_i \wedge C$ is satisfiable. If b is true, then C' is a conjunction of literals such that $\mathrm{FV}(C') \subseteq \mathrm{FV}(F_i)$, $C' \Rrightarrow F_i$, and $C \wedge C'$ is satisfiable.

$x \mapsto v$ denotes the function mapping x to v; *first* denotes the function mapping a couple to its first element.

form $C'_s \wedge F_{i+1}$; we therefore supply it with the $K \mapsto \neg \mathit{first}(\text{Q-Test}(i+1, K))$ function, which answers whether $K \wedge F_{i+1}$ is unsatisfiable.

- If $C' \wedge F_{i+1}$ is satisfiable, we obtain an extended model C'': $C'' \Rrightarrow F_{i+1}$. We therefore add $\neg \pi_i(C'')$ as a new constraint in M_i and retry solving.

3.3 Correctness

The correctness of Generalize is obvious. The partial correctness of Q-Test algorithm is proved by induction over $n - i$: we show it is correct for levels $i = n$ down to $i = 0$. For $i = n$, its correctness reduces to that of SMT-solving and Generalize. The interesting case is $i < n$.

As noted in the algorithm description, we maintain the invariant $F_i \Rrightarrow M_i$. If $C \wedge M_i$ is unsatisfiable, then a fortiori $C \wedge F_i$ is unsatisfiable and the $(\mathsf{false}, \mathsf{false})$ answer is correct. Assume now the induction hypothesis: the correctness of Q-Test$(i+1, C)$, which answers whether $C' \wedge F_{i+1}$ is unsatisfiable. If it is so, then $C' \Rrightarrow \forall_{B_i} \neg F_{i+1}$; we then generalize C' and answer the generalized version. Otherwise, we obtain $C'' \Rrightarrow F_{i+1}$; therefore $F_i \Rrightarrow \neg \pi_i(C'')$ and we can add $\neg \pi_i(C'')$ as a constraint in M_i.

Total correctness is ensured by the fact that the number of C' that can be generated at a given level i is finite, which is proved, again, by induction from $i = n - 1$ down to $i = 0$. At level $n - 1$, all the C' that we obtain are conjunctions of literals built from atoms of M_i. M_i is a conjunction of negations of projections of conjunctions of atoms of M_{i+1}. By the induction hypothesis, only a finite number of atoms can accumulate into M_{i+1}, thus only a finite number of constraints can accumulate into M_i, and the induction is proved. §4.2 provides complexity bounds.

3.4 Example

Recall the formula from Eq. 1: $\exists x \forall y \exists z \; z \geq 0 \wedge (x \geq z \wedge y \geq z \vee y \leq 1 - z)$. Its truth value is equivalent to the satisfiability of F_0: We therefore have

$$F_0 \triangleq \forall y \exists z \; z \geq 0 \wedge (x \geq z \wedge y \geq z \vee y \leq 1 - z) \tag{2}$$

$$F_1 \triangleq \forall z \; \neg(z \geq 0 \wedge (x \geq z \wedge y \geq z \vee y \leq 1 - z)) \tag{3}$$

$$F_2 \triangleq z \geq 0 \wedge (x \geq z \wedge y \geq z \vee y \leq 1 - z) \tag{4}$$

We initialize $M_0 = M_1 = \mathsf{true}$. Consider the call Q-Test$(0, \mathsf{true})$. SMT-Test(true, M_0) returns $(\mathsf{true}, \mathsf{true})$. Q-Test$(1, \mathsf{true})$ is then called.

SMT-Test(true, M_1) returns $(\mathsf{true}, \mathsf{true})$. Q-Test$(2, \mathsf{true})$ is then called. This results in SMT-Test(F_2, true) being called. SMT-solving of F_2 results in a "satisfiable" answer with a solution, for instance, $(x = 0, y = 0, z = 0)$; thus SMT-Test(F_2, true) is $(\mathsf{true}, z \geq 0 \wedge x \geq z \wedge y \geq z \wedge y \leq 1 - z)$. Generalize yields the simpler conjunction $z \geq 0 \wedge x \geq z \wedge y \geq z$, which still implies F_2. The projection of this conjunction parallel to z is $x \geq 0 \wedge y \geq 0$; we add its negation $x < 0 \vee y < 0$ to M_1. We again run SMT-TEST(true, M_1), which returns $(\mathsf{true}, x < 0)$. Q-TEST$(2, x < 0)$ is then called. This results in SMT-TEST$(F_2, x < 0)$ being called. SMT-solving of $F_2 \wedge x < 0$ results in a "satisfiable" answer with a solution, for instance, $(x = -1, y = 0, z = 0)$; thus SMT-TEST$(F_2, x < 0)$ is $(\mathsf{true}, z \geq 0 \wedge y \leq 1 - z)$. GENERALIZE does not simplify this conjunction. The projection of this conjunction parallel to z is $y \leq 1$; we add its negation $y > 1$ to M_1, which is then $y > 1 \wedge (x < 0 \vee y < 0)$. We again run SMT-TEST$(\mathsf{true}, M_1)$, which returns $(\mathsf{true}, x < 0 \wedge y > 1)$. Q-TEST$(2, x < 0 \wedge y > 1)$ is then called. This results in SMT-TEST$(F_2, x < 0 \wedge y > 1)$ being called, with answer $(\mathsf{false}, \mathsf{false})$. We then know that $x < 0 \wedge y > 1 \Rrightarrow F_1$. Q-TEST$(1, \mathsf{true})$ then returns $(\mathsf{true}, x < 0 \wedge y > 1)$.

The projection of this conjunction parallel to y is $x < 0$; we add its negation $x \geq 0$ to M_0. SMT-TEST(true, M_0) returns $(\mathsf{true}, x \geq 0)$. Q-TEST$(1, x \geq 0)$ is then called. SMT-TEST$(x \geq 0, M_1)$ then returns $(\mathsf{false}, \mathsf{false})$. Q-TEST$(0, \mathsf{true})$ then returns $(\mathsf{true}, x \geq 0)$.

3.5 Generalizations

The above algorithm tests the satisfiability of a quantified formula and provides a generalized model if there is one. It can be turned into a quantifier elimination procedure by model enumeration: run Q-TEST$(0, \mathsf{true})$, obtain a model $C_1 \Rrightarrow F$, run Q-TEST$(0, \neg C_1)$, obtain a model $C_2 \Rrightarrow F$, run Q-TEST$(0, \neg V_1 \wedge \neg C_2)$ etc. until Q-TEST returns $(\mathsf{false}, \mathsf{false})$, then $C_1 \vee C_2$ is a DNF for F. This loop terminates for the same reason as Q-TEST: the number of C that can be generated is less than the 2^a where a is the number of possible atoms for M_0.

The algorithm can be generalized to any theory for which there are an SMT-solving algorithm and a projection operator. Obviously, propositional logic is suitable, though specialized QBF solvers are likely to be more efficient. Suitable theories include Presburger arithmetic: current SMT solvers implement integer arithmetic by relaxation to real numbers and branch'n'cut or Gomory cuts, and projection can be done using Omega [23].

One problem is that Omega outputs a disjunction: the results from the "dark shadow", plus a finite number of results from the "gray shadow". The simple generalization scheme in GENERALIZE is then unsuitable. Recall that this algorithm attempts generalizing a conjunction C by removing each conjunct and checking whether the resulting conjunction is still suitable (using the *test* oracle for suitability). Alternatively, one may see this method as replacing atoms by true inside the formula and checking whether the resulting formula is still suitable — which is a correct method for generalizing any formula in negation normal form.

For the sake of simplicity, we have required that the formula be in prenex form. It is possible to generalize the algorithm as follows: given formulas F and C, answer whether $C \wedge F$ is satisfiable and, if so, provide C' such that $C \wedge C'$ is satisfiable and $C' \Rrightarrow F$. Such an algorithm can be defined by induction over F: our Q-Test algorithm implements the case where F contains no quantifier, or is of the form $\forall \neg F$. The case for $\exists F$ is just the case for F followed by projection. The case for $F_1 \vee F_2$ first tests F_1 then, if unsuccessful, F_2. The case for $(F_1 \wedge F_2, C)$ where F_1 has no quantifier (if necessary, eliminate them) reduces to that of $(F_2, F_1 \wedge C)$.

4 Complexity

We shall now prove that the algorithms of §2.2 and §3.2 are at most doubly exponential.

4.1 Number of Faces in Projected Polyhedra

A *polytope* in dimension d is the convex hull of a finite number of points of $\mathbb{R}^d$. We recall the usual definitions of faces [18, §2.1, p. 39]: a *vertex* is a 0-face, an *edge* a 1-face,a *facet* a $(d-1)$-face. If one considers conventional tridimensional geometry, then these definitions fit the usual ones for vertices, edges and sides respectively.

The number of k-faces of a polytope with v vertices in a d-dimensional space can be bounded [18, ch. 4, 5]:

Theorem 1 (McMullen)**.** *The maximal number of k-faces for a polytope with v vertices in a d-dimensional space is obtained for cyclic polytopes and thus is $f_k(v, d)$.*

The number of k-faces in a cyclic polytope (a particular kind of polytope whose definition [18, p. 82] is unimportant for our purposes) can be explicitly computed [18, Prop. 19, p. 86]:

Proposition 2. *The number $f_k(v, d)$ of k-faces of a cyclic polytope with v vertices in a d-dimensional space ($k < d$) is given by:*

$$f_k(v, 2n) = \sum_{j=1}^{n} \frac{v}{v-j} \binom{v-j}{j} \binom{j}{k+1-j} \tag{5}$$

$$f_k(v, 2n+1) = \sum_{j=0}^{n} \frac{k+2}{v-j} \binom{v-j}{j+1} \binom{j+1}{k+1-j} \tag{6}$$

Observe that $f_k(v, 2n)$, as a polynomial in v, has at most degree n, and that its coefficient of degree n, if $k+1 \geq n$, is $\binom{n}{k+1-n} . \frac{1}{n!} = \frac{1}{(k+1-n)!(2n-k-1)!} \leq$

1. $f_k(v, 2n+1)$, as a polynomial in v, also has at most degree n, and its coefficient of degree n, if $k+1 \geq n$, is $\frac{k+2}{(n+1)!}\binom{n+1}{k+1-n} = \frac{k+2}{(k-n+1)!(2n-k)!} \leq 3$. It follows that when $v \to \infty$, $f_k(v, d) = O\left(v^{\lfloor d/2 \rfloor}\right)$. Furthermore, if $k+1 < \lfloor d/2 \rfloor$, then $f_k(v, d)$ has leading monomial $v^{k+1}/(k+1)!$.

Note that a facet of the projection of a polytope P along p coordinates necessarily corresponds to a $(d-1-p)$-face of P (as an example, the edges of a polygon obtained by projecting a tridimensional polyhedron correspond to some of the edges of the original polyhedron). By polytope duality [18, §2.2, p. 61], they correspond to p-faces of the dual polytope of P, whose vertices are the facets of P. Therefore:

Lemma 3. *The number of facets of the projection of a polytope with f facets in $\mathbb{R}^d$ into $\mathbb{R}^{d-p}$, along p coordinates, is at most $f_p(f, d)$.*

The results above are valid for bounded polytopes, whereas our algorithms operate on unbounded polyhedra. By adding at most $2n$ constraints of the form $x_i \leq \pm C$ with C a large enough constant, we can obtain a bounded polytope P' out of an unbounded polyhedron P. The facets of the projection of P are found among the facets of the projection of P'. From the above results we deduce:

Lemma 4. *The number of facets of the projection of a polyhedron with f facets in $\mathbb{R}^d$ into $\mathbb{R}^{d-p}$, along p coordinates, is $O(f^{\lfloor d/2 \rfloor})$ as $f \to \infty$. Furthermore, if $p+1 < \lfloor d \rfloor 2$, then it is $O(f^{p+1})$.*

4.2 Application to our Algorithms

Consider a formula F in prenex form with n atoms and m variables, from which p are quantified. We can immediately exclude trivial atoms (those equivalent to true or false) and simplify the formula accordingly. In the remaining formula, each atom delimits a half-space. The number of distinct polyhedra that can be constructed from these half-space is at most 2^n. At all levels of our algorithms, we work with facets from projections of these polyhedra. Applying Lemma 4, the number of projections along p given coordinates of these facets is $O(2^{n(p+1)})$ as $n \to \infty$. The model enumeration algorithm, at a given level, can enumerate at most 2^a choices of truth values for $a = O(2^{n(m+1)})$ atoms. Each choice represents one run of SMT-solving, whose cost is $O(2^{a+m})$. To summarize, the overall costs are in $O\left(2^{2^{cnm}}\right)$ for some constant c, thus $O\left(2^{2^{c|F|^2}}\right)$ where $|F|$ is the size of the formula.

In comparison, the substitution-based elimination procedures have complexity $2^{2^{c|F|}}$ [12, 16], and this is a lower bound for real quantifier elimination [11]. Also, any *nondeterministic* decision procedure for quantified real arithmetic has at least exponential complexity in the worst case [13]; so restricting ourselves to decision problems in lieu of quantifier elimination in general is not likely to help much.

However, when it comes to doubly exponential complexities, all that matters from practical purposes is practical complexity: an algorithm that performs well in practice is preferrable to an algorithm with better theoretical bounds, but that tends to reach its theoretical complexity. This is why we implemented the various methods and performed benchmarks, as seen in the next section.

5 Implementation and Benchmarks

We implemented the algorithms of Ferrante and Rackoff [12], Loos and Weispfenning [16], our eager algorithm [19], and our lazy algorithm for linear real arithmetic (§3.2) into our Mjollnir tool.[3]

Since algorithmic costs are sensitive to the kind of formula output (CNF, DNF or unconstrained), we preferred to test these procedures only on decision problems — those without free variables, for which the output is true or false. We generated random benchmarks in blocks of 300, of various kinds:

1. B_1 consists in formulas with 10 variables, with sparse atoms, of the form $\forall v_9 \exists v_8 \forall v_7 \exists v_6 \forall v_5 \exists v_4 \forall v_3 \exists v_2 \forall v_1 \exists v_0\ F$.
2. B_2 consists in formulas with 12 variables, with sparse atoms, of the form $\forall v_{11} \exists v_{10} \forall v_9 \exists v_8 \forall v_7 \exists v_6 \forall v_5 \exists v_4 \forall v_3 \exists v_2 \forall v_1 \exists v_0\ F$.
3. B_3 consists in formulas with 12 variables, with sparse atoms, of the form $\forall v_{11}, v_{10}, v_9, v_8 \exists v_7, v_6, v_5, v_4 \forall v_3, v_2, v_1, v_0\ F$.
4. B_4 consists in formula of the same form as B_3 but with a more complex Boolean structure in F.
5. B_5 consists in formulas with 18 variables, with sparse atoms, of the form $\forall v_{17}, v_{16}, v_{15}, v_{14}, v_{13}, v_{12} \exists v_{11}, v_{10}, v_9, v_8, v_7, v_6 \forall v_5, v_4, v_3, v_2, v_1, v_0\ F$.
6. B_6 consists in formula of the same form as B_5 but with a more complex Boolean structure in F and denser atoms.

Results are provided in Tab.1. Generally speaking, our model enumeration algorithms fail due to timeout (set at 300 s) while the substitution methods fail to out-of-memory (maximal memory 1 GiB); also, the lazy model enumeration algorithm tends to perform better than the eager algorithm, and Loos and Weispfenning's algorithm better than Ferrante and Rackoff's. Comparing the substitution methods to the model enumeration algorithms is difficult: depending on how the benchmarks are generated, one class of algorithms may perform significantly better than the other.

[3] The current version of Mjollnir is available from `http://www-verimag.imag.fr/~monniaux/download/`, as well as the benchmarks formulas.

	B_1	B_2	B_3	B_4	B_5	B_6
True formulas	115	80	220	285	219	230
False formulas	159	134	23	0	32	4
Ferrante-Rackoff (FR) solves	199	150	203	53	178	185
Loos-Weispfenning (LW) solves	250	220	244	80	247	249
Monniaux eager (M1) solves	241	128	218	260	231	112
Monniaux lazy (M2) solves	276	187	238	285	248	143
M2 solves but LW does not	32	9	26	209	28	1
LW solves but M2 does not	6	42	32	4	27	107
M2 solves but M1 does not	35	59	26	25	23	33
M1 solves but M2 does not	0	0	6	0	6	2
FR solves but LW does not	11	11	8	34	7	6
LW solves but FR does not	62	81	49	61	76	70

Table 1: Number of decision problems solved. Each block of 300 formulas $B_1, \ldots, B_6$ was randomly generated; we provide the number of formulas that at least one of the methods proved to be true or false. Maximal memory allowed was 1 GiB and maximal time 300 s.

On some $\forall\exists$ formulas produced by the minimization step of [20], the lazy procedure performs somewhat more slowly (10–40%) than the eager procedure. This seems to indicate that on examples where it is actually necessary to enumerate all items of the eliminated form of the subformulas, it is faster to do it eagerly rather than do it lazily — which tends to apply to any comparison of eager and lazy approaches.

In the model enumeration algorithms, most of the time is spent in the SMT-solver, not in the polyhedral projection.

We investigated alternatives to the Generalize function: the min function from [3], and variants of the order that Generalize and min follow when considering atoms (randomly shuffled, atoms with the variables quantified at the innermost level first, same with outermost level). Surprisingly, min tended to perform worse.

6 Conclusion

We have described a quantifier elimination algorithm for linear real arithmetic that uses nested SMT-solver calls and polyhedral projection, in order to lazily enumerate models. This algorithm is different from those commonly applied for this problem, which are based on replacing existential quantification by a finite disjunction, substituting well-chosen witnesses for the value of the quantified variable. Both kinds of algorithms have doubly exponential complexity in the worst case, which is unavoidable for this problem.

For practical purposes, these two kinds of algorithms behave differently: substitution methods occasionally attempt to construct very large intermediate formulas and finish with out-of-memory, while model enumeration methods occasionally run into high computation times. We have experimented both kinds of methods on various classes of formulas, and, depending on the quantification and Boolean structures of the formulas, one method is favored over the other. There is therefore no clear winner.

Acknowledgments We wish to thank Scott Cotton and Goran Frehse as well as the anonymous referees for helpful comments and suggestions.

References

[1] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. *The Parma Polyhedra Library, version 0.9.* URL `http://www.cs.unipr.it/ppl`.

[2] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry.* Algorithms and computation in mathematics. Springer, 2 edition, 2006. ISBN 3-540-33098-4. URL `http://perso.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html`.

[3] Aaron R. Bradley and Zohar Manna. Property-directed incremental invariant generation. *Formal Aspects of Computing*, 20(4–5):379–405, July 2008. ISSN 0934-5043. doi: 10.1007/s00165-008-0080-9.

[4] John W. Chinneck. Finding a useful subset of constraints for analysis in an infeasible linear program. *INFORMS J. on Computing*, 9(2), 1997.

[5] John W. Chinneck. *Feasibility and infeasibility in optimization.* Springer, 2008. ISBN 978-0-387-74931-0.

[6] George Collins. Quantifier elimination for real closed fields by cylindric algebraic decomposition. In *Automata theory and formal languages (2nd GI conference)*, volume 33 of *LNCS*, pages 134–183. Springer, 1975. ISBN 0-387-07407-4.

[7] D. C. Cooper. Theorem proving in arithmetic without multiplication. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 7*, pages 91–100. Edinburgh University Press, 1972. ISBN 0-85224-234-4.

[8] Scott Cotton. *On Some Problems in Satisfiability Solving.* PhD thesis, Université Joseph Fourier, Grenoble, France, June 2009.

[9] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer, 3rd edition, 2007. ISBN 0-387-35650-9.

[10] George B. Dantzig, Delbert R. Fulkerson, and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. In *50 Years of Integer Programming 1958–2008*, chapter 1, pages 7–28. Springer, 2009. ISBN 978-3-540-86398-4. doi: 10.1007/978-3-540-68279-0_1.

[11] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *J. Symb. Comput.*, 5(1-2):29–35, 1988. ISSN 0747-7171. doi: 10.1016/S0747-7171(88)80004-X.

[12] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. on Computing*, 4(1): 69–76, March 1975. ISSN 0097-5397. doi: 10.1137/0204006.

[13] Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In Richard Karp, editor, *Complexity of Computation*, number 7 in SIAM–AMS proceedings, pages 27–42. American Mathematical Society, 1974. ISBN 0-8218-1327-7.

[14] Ulrich Junker. QuickXplain: Conflict detection for arbitrary constraint propagation algorithms. IJCAI '01 workshop CONS-1, 2001.

[15] Daniel Kroening and Ofer Strichman. *Decision procedures*. Springer, 2008. ISBN 978-3-540-74104-6.

[16] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993. Special issue on computational quantifier elimination.

[17] Kenneth L. McMillan, Andreas Kuehlmann, and Mooly Sagiv. Generalizing DPLL to richer logics. In Ahmed Bouajjani and Oded Maler, editors, *Computer-aided verification (CAV)*, volume 5643 of *LNCS*, pages 462–476. Springer, 2009. ISBN 978-3-642-02657-7. doi: 10.1007/978-3-642-02658-4_35.

[18] Peter McMullen and Geoffrey C. Shepard. *Convex polytopes and the upper bound conjecture*, volume 3 of *London Mathematical Society lecture note series*. Cambridge University Press, 1971. ISBN 0-521-08017-7.

[19] David Monniaux. A quantifier elimination algorithm for linear real arithmetic. In *LPAR (Logic for Programming Artificial Intelligence and Reasoning)*, volume 5330 of *LNCS*, pages 243–257. Springer, 2008. doi: 10.1007/978-3-540-89439-1_18.

[20] David Monniaux. Automatic modular abstractions for linear constraints. In *POPL (Principles of programming languages)*. ACM, 2009.

[21] David Monniaux. Optimal abstraction on real-valued programs. In Hanne Riis Nielson and Gilberto Filé, editors, *Static analysis*

(SAS), volume 4634 of *LNCS*, pages 104–120. Springer, 2007. ISBN 978-3-540-74060-5. doi: 10.1007/978-3-540-74061-2_7.

[22] Yannick Moy. Sufficient preconditions for modular assertion checking. In Francesco Logozzo, Doron Peled, and Lenore D. Zuck, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 4905 of *LNCS*, pages 188–202. Springer, 2008. ISBN 978-3-540-78162-2. doi: 10.1007/978-3-540-78163-9_18.

[23] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Supercomputing '91*, pages 4–13. ACM, 1991. ISBN 0-89791-459-7. doi: 10.1145/125826.125848.

[24] Darsh P. Ranjan, Daijue Tang, and Sharad Malik. A comparative study of 2QBF algorithms. In *Theory and Applications of Satisfiability Testing (SAT)*, May 2004.