A Quantifier Elimination Algorithm for Linear Real Arithmetic

David Monniaux

A Quantifier Elimination Algorithm for Linear Real Arithmetic

David Monniaux

CNRS / VERIMAG

Abstract. We propose a new quantifier elimination algorithm for the theory of linear real arithmetic. This algorithm uses as subroutine satisfiability modulo this theory, a problem for which there are several implementations available. The quantifier elimination algorithm presented in the paper is compared, on examples arising from program analysis problems, to several other implementations, all of which cannot solve some of the examples that our algorithm solves easily.

1 Introduction

Consider a logic formula F, possibly with quantifiers, whose variables lay within a certain set S and whose atomic predicates are relations over S. The models of this formula are assignments of values in S for the free variables of F. *Quantifier elimination* is the act of providing another formula F', without quantifiers, such that F and F' are *equivalent*, that is, have exactly the same models. If F has no free variables, then F' is a ground (quantifier-free, variable-free) formula. In most practical cases such formulas can be easily decided to be true or false; quantifier elimination thus provides a *decision procedure* for quantified formulas.

In this paper, we only consider relations of the form $L(x, y, z, \dots) \geq 0$ where L is a linear affine expression (an arithmetic expression where multiplication is allowed only by a constant factor). We can thus deal with any formula over linear equalities or inequalities. Our algorithm transforms any formula of the form $\exists x_1, \dots, x_n \; F$, where F has no quantifiers, into a quantifier-free formula F' in disjunctive normal form. Nested quantifiers are dealt with by syntactic induction. Universal quantifiers are converted to existential ones ($\forall x_1, \dots, x_n \; F \equiv \neg\exists x_1, \dots, x_n \; \neg F$), yet our algorithm avoids the combinatorial explosion over negations that hinders some other methods.

Our method can be understood as an improvement over the approach of converting to DNF through ALL-SAT and performing projection. We compared our implementation with commercial and noncommercial quantifier elimination procedures over some examples arising from practical program analysis cases, and ours was the only one capable of processing them without exhausting memory or time, or failing altogether due to the impossibility of handling large coefficients.

2 The Algorithm

We first describe the datatypes on which our algorithm operates, then the off-the-shelf subroutines that it uses, then the algorithm and its correctness proof, then possible alterations.

2.1 Generalities

We operate on unquantified formulas built using $\wedge$, $\vee$, $\Rightarrow$, $\neg$ or other logical connectives such as exclusive-or (the exact set of connectives allowed depends on the satisfiability tester being used, see below; in this paper we shall only use $\wedge$, $\vee$ and $\neg$), and on quantified formulas built with the same connectives and the existential ($\exists$) and universal ($\forall$) quantifiers. It is possible to quantify not only on a single variable but also on a set of variables, represented as a vector $\boldsymbol{v}$; the algorithm is sensitive to the ordering of the variables in this vector. The atoms are linear inequalities, that is, formulas of the form $c + c_x x + c_y y + c_z z \cdots \geq 0$ where $c \in \mathbb{Q}$ is the *constant coefficient* and $c_v \in \mathbb{Q}$ is the coefficient associated with variable v. It is trivially possible to represent equalities or strict inequalities using this formula language.

The *models* of a formula F are assignments a of rational numbers to the free variables of F such that a satisfies F (written $a \models F$). F is said to be *satisfiable* if a model exists for it. If F has no free variables, then F is said to be *true* if F is satisfiable, *false* otherwise. Two formulas A and B are said to be *equivalent*, noted $A \equiv B$, if they have the same models. Formula A is said to *imply* formula B, noted $\Rrightarrow B$, if any model of A is a model of B.

Algorithm 1 GENERALIZE1(a, F): Generalize a model a of a formula F to a conjunction

Require: $a \models F$
  $M \leftarrow \mathsf{true}$
  for all $P \in$ ATOMICPREDICATES(F) **do**
    if $a \models P$ **then**
      $M \leftarrow M \wedge P$
    else
      $M \leftarrow M \wedge \neg P$
    end if
  end for
Ensure: $M \Rrightarrow F$

Consider a quantifier-free formula F, whose atomic predicates are linear inequalities, and variables $x_1, \ldots, x_n$. We wish to obtain a quantifier-free formula F' equivalent to $\exists x_1, \ldots, x_n\ F$. Let us temporarily forget about efficiency. F can be put into disjunctive normal form (DNF) $C_1 \vee \cdots \vee C_m$ (by recursive application of distributivity), and $\exists x_1, \ldots, x_n\ F$ is thus equivalent to

Algorithm 2 GENERALIZE2(G, M): Remove useless constraints from conjunction M so that $G \wedge M \equiv \mathsf{false}$

Require: $G \wedge M$ is not satisfiable
  for all c conjunct in M **do**
    if $(G \setminus \{c\}) \wedge M$ is not satisfiable (call SMT) **then** $\{G \setminus \{c\}$ denotes G with the conjunct c taken out$\}$
      remove c from M
    end if
  end for
Ensure: $G \wedge M$ is not satisfiable

Algorithm 3 EXISTELIM: Existential quantifier elimination

  $H \leftarrow F$
  $G \leftarrow \neg F$
  $O \leftarrow \mathsf{false}$
  while H is satisfiable (call SMT) **do** $\{(\exists \boldsymbol{v}\ F) \equiv (O \vee \exists \boldsymbol{v}\ H)$ and $G \equiv \neg(F \vee O)$ and $H \wedge O \equiv \mathsf{false}$ and O does not mention variables from $\boldsymbol{v}\}$
    $a \leftarrow$ a model of H $\{a \models H\}$
    $M_1 \leftarrow$ GENERALIZE1(H, a) $\{M_1 \Rrightarrow H\}$
    $M_2 \leftarrow$ GENERALIZE2(G, M_1) $\{\neg(M_2 \wedge G)\}$
    $\pi \leftarrow$ PROJECT$(M_2, \boldsymbol{v})$ $\{\pi \equiv \exists \boldsymbol{v}\ M_2\}$
    $O \leftarrow O \vee \pi$
    $H \leftarrow H \wedge \neg\pi$
    $G \leftarrow G \wedge \neg\pi$
  end while
Ensure: $O \equiv \exists \boldsymbol{v}\ F$

$(\exists x_1, \ldots, x_n\ C_1) \vee \cdots \vee (\exists x_1, \ldots, x_n\ C_m)$. Various methods exist for finding a conjunction C'_i equivalent to $\exists x_1, \ldots, x_n\ C_i$, among which Fourier-Motzkin elimination (see § 3.1). We therefore obtain F' in DNF. For a universal quantifier, through De Morgan's laws, we obtain a formula in conjunctive normal form (CNF).

Such a naive algorithm suffers from an obvious inefficiency, particularly if applied recursively to formulas with alternating quantifiers. Consider $\exists x \forall y\ F$. The algorithm will compute a CNF formula equivalent to $\forall y\ F$, then convert this formula to DNF. Conversion from CNF to DNF through the application of distributivity of $\wedge$ over $\vee$ is extremely inefficient, even on propositional formulas. Furthermore, many conjunctions in the DNF are likely to be contradictory; that is, they will express incompatible linear constraints. It is therefore a waste of time and space to generate them. Finally, the DNF form obtained by distributivity may be needlessly complex; for instance, $(x < 0 \wedge x \geq 0) \wedge y > 0$ gets turned into $(x < 0 \wedge y > 0) \vee (x \geq 0 \wedge y > 0)$ whereas one should have merged both conjuncts into the more general $y > 0$.

The basic ideas of our algorithm are: to only generate conjunctions that are actually useful: no contradictory conjunctions, and no conjunctions that add

nothing to the already computed DNF; and to generalize the conjunctions before further processing.

2.2 Building blocks

If one has propositional formulas with a large number of variables, one never converts formulas naively from CNF to DNF, but one uses techniques such as propositional satisfiability (SAT) solving. Even though SAT is NP-complete, there now exist algorithms and implementations that can deal efficiently with many large problems arising from program verification. In our case, we apply SAT modulo the theory of linear real inequalities (SMT), a problem for which there also exist algorithms, implementations, standard benchmarks and even a competition. Likewise SAT, SAT modulo linear inequalities is NP-complete. A SMT solver takes as an input a formula F where the literals are linear equalities or inequalities, and answers either "not satisfiable", or a model of F, assigning a rational number to each variable in F. We assume we have such an algorithm SMT at our disposal as a building block

Another needed building block is quantifier elimination over conjunctions, named PROJECT$(C, \boldsymbol{v})$: given a conjunction C a conjunction C of linear inequalities over variables $\boldsymbol{v} = v_1, \ldots, v_N$, obtain a conjunction C' equivalent to $\exists v_1, \ldots, v_n\ C$. For efficiency reasons, it is better if C' is minimal (no conjunct can be removed without adding more models), or at least "small". Fourier-Motzkin elimination is a simple algorithm, yet, when it eliminates a single variable, the output conjunction can have a quadratic number of conjuncts compared to the input conjunction, thus a pass of simplification would be needed for practical efficiency (our algorithm GENERALIZE2 provides such simplification). For our implementations, we rather used libraries implementing geometrical transformations: C defines a convex polyhedron[1] in $\mathbb{Q}^N$, and finding C' amounts to computing the inequalities defining the projection of this polyhedron into $\mathbb{Q}^{N-n}$. This can be achieved by computing the set of generators of the polyhedron defined by C using Chernikova's algorithm, projecting these generators and thus obtaining generators for the projected polyhedron, and then computing minimal constraints from these generators.

2.3 Algorithm and Correctness Proof

Consider a quantifier-free formula F. The truth value of F on an assignment a of its variables only depends on the truth value of the atomic predicates of F over

[1] A good bibliography on convex polyhedra and the associated algorithms can be found in the documentation of the Parma Polyhedra Library. [1] By *convex polyhedron*, we mean, in a finite-dimension affine linear real space, an intersection of a finite number of half-spaces each delimited by a linear inequality, that is, the set of solutions of a finite system of linear inequalities. In particular, such a polyhedron can be unbounded. In the rest of the paper, the words "polyhedron" must be understood to mean "convex polyhedron" with that definition.

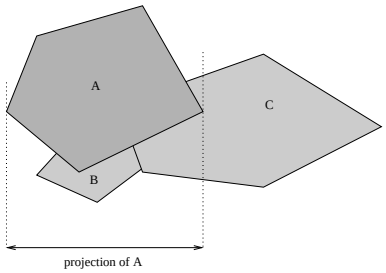

Fig. 1. Subsumption of one generalized model by another

a. Let us note $N_F = |\text{AtomicPredicates}(F)|$, where $|X|$ denotes the cardinal of the set X. These truth assignments therefore define at most 2^{N_F} equivalence classes. There can be fewer than 2^{N_F} equivalence classes, because some truth assignments can be contradictory (for instance, $x \geq 1$ assigned to true and $x \geq 0$ assigned to false). One can immediately generalize a model of a formula to its equivalence class, which motivates our algorithm Generalize1. Its output is a conjunction of literals from F.

This conjunction may itself be insufficiently general. Consider the formula $F = (x \geq 0 \wedge y \geq 0) \vee (\neg x \geq 0 \wedge y \geq 0)$. $x \mapsto 0, y \mapsto 0$ is a model of F. Generalize1 will output the conjunction $x \geq 0 \wedge y \geq 0$. Yet, the first conjunct could be safely removed. Generalize2$(\neg(F \vee O), M)$ will remove unnecessary conjuncts from M while preserving the property that $M \Rrightarrow F \vee O$.

Generalize2(G, M), where M is a conjunction such that $G \wedge M$ is unsatisfiable, works as follows:

- It attempts removing the first conjunct from M (thus relaxing the M constraint). If $G \wedge M$ stays unsatisfiable, the conjunct is removed. If it becomes satisfiable, then the conjunct is necessary and is kept.
- The process is continued with the following conjuncts.

Note that the results of this process depend on the order of the conjuncts inside the conjunction M, and that some orders may perform better than others; the resulting set of conjuncts is minimal with respect to inclusion, but not necessarily with respect to cardinality.

This is the case even if we consider a purely propositional case. As an example, consider $F = A \vee (B \wedge C)$. $M = A \wedge B \wedge C \Rrightarrow F$, otherwise said $M \wedge \neg F$ is not satisfiable. If one first relaxes the constraint A, one gets the conjunction $B \wedge C$, which still implies F; this conjunction has two propositional models ($A \wedge B \wedge C$ and $\neg A \wedge B \wedge C$). Yet, one could have chosen to relax B and obtain $A \wedge C$, and then to relax C and obtain A (which still implies F); this formula has four propositional models. This dependency on propositional variable ordering is similar to that of binary decision diagrams. In terms of formulas where the atoms are linear inequalities, the propositional models that are satisfiable with respect to linear arithmetic correspond to equivalence classes of models over the

reals. Informally, this means that depending on the order by which the conjuncts inside M are ordered, a smaller or bigger "chunk" of state space may be obtained.

The main algorithm is ExistElim$(F, \boldsymbol{v})$ which computes a DNF formula equivalent to $\exists \boldsymbol{v}\ F$. $\boldsymbol{v}$ is a vector of variables. $\boldsymbol{v}$ can be empty, and then the algorithm simply computes a "simple" DNF form for F. The algorithm computes generalized models of F and projects them one by one, until exhaustion. It maintains three formulas G, H and O. O is a DNF formula containing the projections of the models processed so far. H contains the models yet to be processed; it is initially equal to F. G is maintained so that it is equivalent to $\neg(F \vee O)$. For each generalized model M, its projection π is added to O and removed from H.

The partial correctness of the algorithm ensues from the loop condition and loop invariants given in the description of the algorithm. The only nontrivial invariant condition is $(\exists \boldsymbol{v}\ F) \equiv O \vee (\exists \boldsymbol{v}\ H)$. Let us write $O' = O \vee \pi$ and $H' = H \wedge \neg\pi$. An important property is that both π (and thus $\neg\pi$) and O do not depend on $\boldsymbol{v}$, therefore for any X, $\exists \boldsymbol{v}(\pi \wedge X) \equiv (\exists \boldsymbol{v}\ X) \wedge \pi$ (respectively for $\neg\pi$ and O). $O' \vee \exists \boldsymbol{v}\ H' = \pi \vee O \vee \exists \boldsymbol{v}(H \wedge \neg\pi) \equiv \pi \vee O \vee (\exists \boldsymbol{v}\ H) \wedge \neg\pi \equiv \pi \vee O \vee (\exists \boldsymbol{v}\ H)$. $\mathsf{false} \equiv M_2 \wedge G \equiv M_2 \wedge (F \vee O)$, otherwise said $M_2 \Rrightarrow F \vee O$, thus $\pi = (\exists \boldsymbol{v} M_2) \Rrightarrow (\exists \boldsymbol{v} F) \vee O$. It follows that $O' \vee (\exists \boldsymbol{v}\ H') \equiv (\exists \boldsymbol{v}\ F)$.

Given a formula ϕ, we note $W(\phi)$ the number of equivalence of classes induced by the atomic predicates of F with nonempty intersection with the models of ϕ. Termination is ensured because $W(H)$ decreases by at least one at each iteration: M_1 defines exactly one equivalence class, M_2 defines a union of equivalence classes which includes the one defined by M_1, and the models of π include those of M_2 thus also at least one equivalence class. The number of iterations is thus at most 2^{N_F}. Note that Generalize2 is needed neither for correctness nor for termination, but only for efficiency: otherwise, the number of iterations would always be the number of nonempty equivalence classes, which can be huge.

2.4 Possible Changes and Extensions

Certain SMT solvers provide, in addition to an unsatisfiability result, an *unsatisfiable core* (from $A_1 \wedge \cdots \wedge A_n$ extract an unsatisfiable subset, minimal with respect to inclusion). This procedure can replace Generalize2.

The algorithm would still be correct if M was removed from H instead of π. It then becomes equivalent to performing ALL-SAT (obtaining all satisfying assignments) then projection. On the one hand, with this modified algorithm (mod1) set of atomic formulas of H would stay stay included in that of F throughout the iterations, while this set can grow larger with the original algorithm since the set of atomic formulas of the projection of F can be much larger than the set of atomic formulas in F (see §3.1). On the other hand, the original algorithm may need fewer iterations because π may subsume several generalized models, as shown by Fig. 1 : A is the first generalized model being generated, and its projection subsumes B; thus, the original algorithm will not have to generate B, while the modified algorithm will generate B. Our experience is that the original algorithm performs better in practice than the modified algorithm.

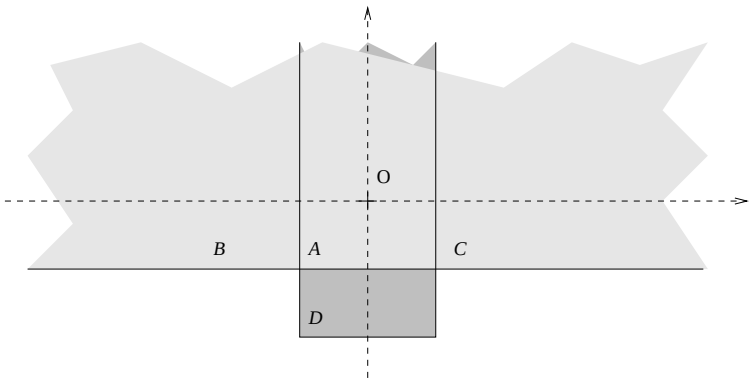

Fig. 2. The gray area is the set of points matched by formula $F = y \geq -1 \vee (y \geq -2 \wedge x \geq -1 \wedge x \leq 1)$. Point $\mathrm{O} = (0, 0)$ is found as a model. This model is first generalized to $y \geq -1 \wedge y \geq -2 \wedge x \geq -1 \wedge x \leq 1$ according to its valuations on the atomic boolean formulas. Depending on whether one first tries to relax $x \geq -1$ or $y \geq -1$, one gets either a half plane (one conjunct) or a vertical band (three conjuncts); the former is "simpler" than the second. The simplicity of the formula output by GENERALIZE2 thus depends on the ordering of the input conjuncts.

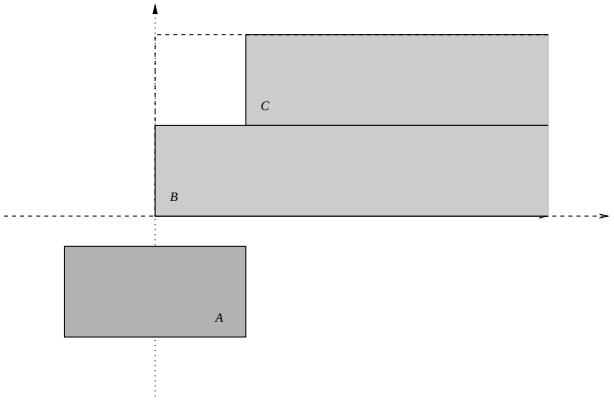

Fig. 3. A is the first generalized model selected. If $G_0 \stackrel{\text{def}}{=} \neg F$, the initial value of G, is replaced at the next iteration by $G_1 \stackrel{\text{def}}{=} \neg F \wedge \neg \pi_0$ where π_0 is the projection of A, then it is possible to generate a single generalized model encompassing both B and C (for instance $x \geq -1 \wedge y \geq 0 \wedge y \leq 2$. If G stays constant, then the $x \geq 1$ constraint defining the left edge of C cannot be relaxed.

Another possible change (mod2) is to leave $G = \neg F$ constant across iterations. This allows less generalization of models than the original algorithm, as

shown by Fig. 3. The modified algorithm tries to generalize M to a conjunction that implies F, but in fact this is too strict a condition: the original algorithm tries to generalize F to a conjunction that implies $F \vee O$. If at least one variable is projected out, and F actually depends on that variable, then the models of F are strictly included in those of the final value of O, which is equivalent to $\exists \boldsymbol{v}\ F$.

The algorithm can be easily extended to quantifier elimination modulo an assumption T on the free variables of F. All definitions stay the same except that $\Rrightarrow$ is replaced by $\Rrightarrow_T$, defined as $P \Rrightarrow_T Q \stackrel{\text{def}}{=} (P \wedge T) \Rrightarrow (Q \wedge T)$ and $\equiv$ is replaced by $\equiv_T$, defined as $(P \equiv_T Q) \stackrel{\text{def}}{=} (P \wedge T \equiv Q \wedge T)$. ExistElim is modified by replacing the initialization of G and H by $\neg F \wedge T$ and $F \wedge T$ respectively. Intuitively, T defines a universe of validity such that values outside of the models T are irrelevant to the problem being studied.

3 Comparison with Other Algorithms

3.1 Complexity bounds

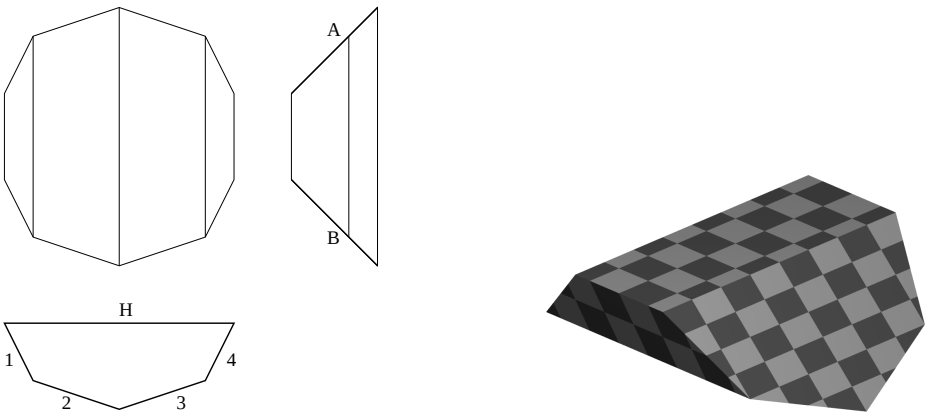

Fig. 4. This 7-face polyhedron in tridimensional space, shown using technical drawing conventions, projects to a 10-edge polygon.

Projecting a convex polyhedron from and to a representation using constraints (linear inequalities defining the facets of the polyhedron) is a hard problem; the number of facets can grow exponentially with the number of dimensions being projected out.

Lemma 1. *There exist a family of polyhedra in tridimensional space, indexed by n, with $n+3$ facets such that their projection onto a plan has $2n+2$ facets.*

Proof. The construction is shown in Fig. 4 for $n = 4$. One first draws a $n+1$ edge convex polygon in the (x, z) plane (shown in bold at bottom of the figure:

edges H, $1, \ldots, n$), including a $z = 0$ edge and other edges more or less in a semicircular shape (one can for instance take half a $2n$-edge regular polygon). This polygon is extruded along the y axis, and the resulting infinite prism is cut at at angle at both ends (the bold lines A and B at the right of the figure), thus producing a polyhedron with $n + 3$ facets. Its projection on the (x, y) plane has $2n + 2$ facets, as shown on the figure.

Corollary 1. *For any $0 < \alpha < 2$ there exists a family of polyhedra indexed by k in $3k$-dimensional space, with a number f_k of facets, such that there exists a choice of dimensions such that their projection on $2k$-dimensional space has more than $f_k \alpha^k$ facets.*

Proof. In a $3k$-dimensional space, consider the product of k polyhedra of tridimensional space with $n + 3$ facets as produced by the preceding lemma. This product is a polyhedron with $(n + 3)^k$ facets. Its projection, leaving one every third dimension, has $(2n + 2)^k$ facets. With n large enough, $\frac{(2n+2)^k}{(n+3)^k} > \alpha^k$.

Methods for eliminating variables from systems of linear inequalities (that is, eliminating an existential quantifier from a conjunction of linear inequalities) have long been studied. The simplest method is Fourier-Motzkin elimination. Consider a system S of wide or strict linear inequalities, from which we wish to eliminate variable x (meaning that we wish to eliminate the quantifier from $\exists x\ S$). There exist three kind of inequalities in S: those equivalent to inequalities of the form $x \geq L_i^+(y, z, \ldots)$ or $x > L_i^+(y, z, \ldots)$ (where x does not appear in L), which we group in a sub-system S_+; those equivalent to inequalities of the form $x \leq L_j^-(y, z, \ldots)$ or $x < L_j^-(y, z, \ldots)$ (where x does not appear in L), grouped as S_-; and those in which x does not appear, grouped as S_0.

Assume for a moment, for the sake of simplicity, that S only contains wide inequalities; the algorithm for mixed strict and wide inequalities is almost the same. S_+ is equivalent to $x \geq \max(L_1^+(y, z, \ldots), \ldots, L_{|S_+|}^+(y, z, \ldots))$ and S_- is equivalent to $x \leq \max(L_1^-(y, z, \ldots), \ldots, L_{|S_-|}^-(y, z, \ldots))$. $\exists x\ S$ is thus equivalent to

$$S_0 \wedge \exists x\ \ \max(L_1^+(y, z, \ldots), \ldots, L_{|S_+|}^+(y, z, \ldots)) \leq x \leq \min(L_1^-(y, z, \ldots), \ldots, L_{|S_-|}^-(y, z, \ldots)),$$

thus equivalent to

$$S_0 \wedge \max(L_1^+(y, z, \ldots), \ldots, L_{|S_+|}^+(y, z, \ldots)) \leq \min(L_1^-(y, z, \ldots), \ldots, L_{|S_-|}^-(y, z, \ldots)),$$

thus in the end equivalent to S_0 and all inequalities of the form $L_i^+(y, z, \ldots) \leq L_j^-(y, z, \ldots)$. The number of such inequalities can be as large as $|S|^2/4$ if S is split evenly between S_+ and S_-. Thus, with the Fourier-Motzkin method without a simplification step, *the size of the system of inequalities can grow quadratically*

for each variable being eliminated, thus a complexity bound of $2^{2^{cn}}$ where n is the size of the original formula. The size of the coefficients of the inequalities can double. We have not been able to find results that close the gap between our single exponential lower bound obtained from Cor. 1 and the double exponential upper bound obtained from Fourier-Motzkin. However, this is of little practical importance, as discussed later.

The "classical" algorithm for quantifier elimination over real or rational arithmetic is Ferrante and Rackoff's method [2][3, §7.3]. This algorithm can be understood as an extension of Fourier-Motzkin's algorithm to formulas with disjunctions: in addition to checks for unbounded intervals, one looks at all couples (L_i^+, L_j^-), defining intervals, and checks that the middle point (or, for the matter, any point of the inside) verifies the formula. The complexity is, again, $2^{2^{cn}}$. Note that Ferrante and Rackoff's algorithm never simplifies formulas.

Our algorithm uses satisfiability testing over the theory of rational linear inequalities, which is an NP-complete problem. It is NP-hard because any boolean satisfiability problem can be straightforwardly converted to a linear equality problem by replacing each boolean variable by a real variable with values constrained in $\{0, 1\}$. It is NP because one can solve this problem by looking nondeterministically for a boolean truth assignment for the atomic formulas, then check that this truth assignment is consistent. Each truth assignment determines a conjunction of linear inequalities, and checking whether such a conjunction defines a nonempty set of solutions is a linear programming problem with a null objective function. Linear programming can be solved in polynomial time. This result only applies to unquantified formulas; the decision problem for quantified formulas over rational linear inequalities requires at least exponential time. [3, §7.4][4, Th. 3]

For each quantifier elimination step, the number of atoms in the formula can grow at most quadratically, following the Fourier-Motzkin bounds. The final number of atomic formulas is thus at most $2^{2^{cn}}$. The number of iterations of the final quantifier elimination loop is at most exponential in the number of these atoms, as is the cost of running the satisfiability tests. The overall complexity is thus bounded by $2^{2^{2^{cn}}}$.

One could at first assume that the complexity bounds for our algorithm are asymptotically worse than Ferrante and Rackoff's (triple exponential compared to double exponential). Our algorithm, however, outputs results in CNF or DNF form, while Ferrante and Rackoff's algorithm does not. If we add a step of transformation to CNF or DNF to their algorithm, then we also obtain a triple exponential.

Our opinion is that when it comes to complexity in towers of exponential, comparisons of upper bounds on the worst case are of no practical importance. If an algorithm really exhibits such complexity in practice, it is useless, regardless of whether the bound has two or three exponentials.

3.2 Practical results

We implemented our method twice: first as a proof-of-concept using a quickly assembled SMT-solver, then, as the MJOLLNIR tool, using the state-of-the-art YICES solver[2] and the APRON[3] polyhedron library. We investigated examples produced from problems of program analysis following our method for the parametric computation of least invariants. [5] To summarize, each formula expresses the fact that a set of program states (such as a product of intervals for the numerical variables) is the least invariant of a program, or the strongest postcondition if there is no fixed point involved. Most of the examples, being extracted by hand from simple subprograms, are easily solved, but one of them, defining the least invariant of a rate limiter, proved to be tougher to solve, and we selected it as a benchmark. We have two versions of this example: the first for a rate limiter operating over real numbers[4], the second over floating-point numbers, abstracted using real numbers. The floating-point version is considerably tougher to process than the real example. We also tested our system on examples procured from the LIRA designers.

We compared our proof-of-concept implementation with the home-made SMT-solver, the MJOLLNIR tool, LIRA[5], a tool based on Büchi automata, and two symbolic algebra packages, MATHEMATICA[6] and REDUCE 3.8[7] + REDLOG[8], see Tab. 1. Profiling shows that most of the time is spent in the SMT-solver and only a few percents in the projection algorithm. The fact that the proof-of-concept implementation, with a very naive SMT-solver, performs decently on an example where other algorithms exhaust memory shows that the performance of our algorithm cannot be solely explained by the good quality of YICES.

4 Conclusion and Future Work

We have proposed a new quantifier elimination algorithm for the theory of linear inequalities over the real or rational numbers. Our motivation was the practical application of a recent result of ours on program analysis, stating that formulas for computing the least invariants of certain kinds of systems can be obtained through quantifier elimination [5]. This algorithm is efficient on examples obtained from this program analysis technique, as well as other examples, whereas earlier published algorithms, as well as several commercial packages, all exhaust time or memory resources. Our algorithm leverages the recent progresses on satisfiability modulo theory solvers (SMT) and, contrary to older algorithms,

[2] `http://yices.csl.sri.com/`
[3] `http://apron.cri.ensmp.fr/library/`
[4] Available in the LIRA input syntax from `http://www-verimag.imag.fr/~monniaux/download/rlim_nofloat_instanciated.lira`.
[5] `http://lira.gforge.avacs.org/`
[6] `http://www.wolfram.com/`
[7] `http://www.uni-koeln.de/REDUCE/`
[8] `http://www.algebra.fim.uni-passau.de/~redlog/`

Benchmark	r. lim. $\mathbb{R}$	r. lim. float	`prsb23`	`blowup5`
MJOLLNIR	1.5	34	0.07	negligible
proof-of-concept	n/a	823	n/a	n/a
MJOLLNIR (mod1)[a]	1.6	77[b]	0.06	negligible
MJOLLNIR (mod2)[c]	1.4	17	0.06	negligible
MJOLLNIR Ferrante-Rackoff	o-o-m	o-o-m	o-o-m	negligible
LIRA	o-o-m	o-o-m	8.1	0.6
REDUCE `rlqe`	182	o-o-m	1.4	negligible
REDUCE `rlqe`+`rldnf`	o-o-m	o-o-m	n/a	n/a
MATHEMATICA `Reduce`	(> 12000)	o-o-m	(> 780)	7.36

[a] `-no-block-projected-model` in MJOLLNIR
[b] Memory consumption grows to 1.1 GiB.
[c] `-no-add-blocking-to-g` in MJOLLNIR

Table 1. Timings (in seconds, on an AMD Turion TL-58 64-bit Linux system) for eliminating quantifiers from our benchmarks. The first line is the algorithm described in this paper, the two following linear variants from §2.4, then other packages. REDUCE has `rlqe` (quantifier elimination) and `rlqe`+`rldnf` (same, followed by conversion to DNF). ($> t$) means that the computation was killed after t seconds because it was running too long. The `prsb23` and following are decision problems, the output is `true` or `false`, thus DNF form does not matter. Out-of-memory is noted "o-o-m".

performs on-the-fly simplifications of formulas that keep formula sizes manageable.

Our algorithm is described for rational or real linear arithmetic, but it can be extended to any theory for which there is an efficient satisfiability testing algorithm for unquantified formulas and a reasonably efficient projection algorithm for conjunctions. Among extensions that could be interesting from a practical point of view would be on the one hand the nonlinear case for real arithmetic (polynomials), and on the other hand the mixed integer / real problems. Of course, nonlinear integer arithmetic cannot be considered, since Peano arithmetic is undecidable.

Tarski showed that the theory of the real closed fields (inequalities of polynomial expressions) admits quantifier elimination, [6] however his algorithm had impractical (non-elementary) complexity. Later, the *cylindrical algebraic decomposition* (CAD) [7, Ch. 11] method was introduced, with doubly exponential complexity, which is unavoidable in the worst case [7, §11.4]. Our experiments with both MATHEMATICA and QEPCAD, both of which implement CAD, as well as with REDUCE/REDLOG, which implement various algorithms for quantifier elimination, showed us that combinatorial blowup occurs very quickly. For such techniques to be interesting in practice, practical complexity should be lowered. Perhaps our technique could help. There are, however, significant difficulties in that respect. Our technique starts with some single model of the target formula over the rational numbers; but a system of nonlinear inequalities needs not have

rational models when it is not full-dimensional (for instance, $X^2 = 2$). Our technique reduces the geometrical computations to computations on conjunctions; but in the nonlinear case, single inequalities can be reduced to disjunctions. As an example, $X^2 \geq 4$ is reduced to $X \leq -2 \vee X \geq 2$. Most importantly, our technique relies at several steps on the availability of a decision procedure that stays efficient even when the answer is negative.

Regarding the mixed integer / real problems, the LIRA tool implements quantifier elimination using a weak form of Büchi automata matching the b-ary expression of the integers or reals, where b is an arbitrary base. [8] The output of the process is an automaton and not a readable formula. While it is possible to decide a closed formula, and to obtain one model from a satisfiable non-closed formula, it is an open problem how to efficiently reconstruct a quantifier-free formula from the resulting automaton. The automaton construct is unsuitable for large coefficients (as our examples obtained from the analysis of floating-point programs). Even on examples with small coefficients, the tool was unable to complete quantifier elimination without blowing up. We think therefore that it would be interesting to be able to apply our technique to the mixed integer / real problems, but there are difficulties: the algorithms on integer polyhedra is considerably more complex than on rational polyhedra.

A classical objection to automatic program analysis tools meant to prove the absence of bugs is that these tools could themselves contain bugs. Our method uses complex algorithms (SMT-solving, polyhedron projection) as subprocedures. We consider developing techniques so that the algorithm outputs easily-checkable proofs or "proof witnesses" of the correctness of its computation.

References

1. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library, version 0.9. available from `http://www.cs.unipr.it/ppl`.
2. Ferrante, J., Rackoff, C.: A decision procedure for the first order theory of real addition with order. SIAM Journal of Computation **4**(1) (March 1975) 69–76
3. Bradley, A.R., Manna, Z.: The Calculus of Computation: Decision Procedures with Applications to Verification. Springer (October 2007)
4. Fischer, M.J., Rabin, M.O.: Super-exponential complexity of presburger arithmetic. In Karp, R., ed.: Complexity of Computation. Number 7 in SIAM–AMS proceedings, American Mathematical Society (1974) 27–42
5. Monniaux, D.: Optimal abstraction on real-valued programs. In Filé, G., Nielson, H.R., eds.: Static analysis (SAS '07). Number 4634 in LNCS, Springer (2007) 104–120
6. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press (1951)
7. Basu, S., Pollack, R., Roy, M.F.: Algorithms in real algebraic geometry. Algorithms and computation in mathematics. Springer (2003)
8. Becker B., Dax C., E.J., F., K.: LIRA: handling constraints of linear arithmetics over the integers and the reals. In: Computer Aided Verification (CAV). Number 4590 in LNCS (2005) 312–315