A heuristic prover for real inequalities

Jeremy Avigad, Robert Y. Lewis, and Cody Roux

Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract. We describe a general method for verifying inequalities between real-valued expressions, especially the kinds of straightforward inferences that arise in interactive theorem proving. In contrast to approaches that aim to be complete with respect to a particular language or class of formulas, our method establishes claims that require heterogeneous forms of reasoning, relying on a Nelson-Oppen-style architecture in which special-purpose modules collaborate and share information. The framework is thus modular and extensible. A prototype implementation shows that the method is promising, complementing techniques that are used by contemporary interactive provers.

1 Introduction

Comparing measurements is fundamental to the sciences, and so it is not surprising that ordering, bounding, and optimizing real-valued expressions is central to mathematics. A host of computational methods have been developed to support such reasoning, using symbolic or numeric methods, or both. For example, there are well-developed methods of determining the satisfiability or unsatisfiability of linear inequalities [29,30], polynomial inequalities [5], nonlinear inequalities involving functions that can be approximated numerically [23], and inequalities involving convex functions [8]. The "satisfiability modulo theories" framework [25], [31] provides one way of integrating such methods with ordinary logical reasoning and proof search; integration with resolution theorem proving methods has also been explored [1], [28]. Interactive theorem provers like Isabelle [26] and HOL Light [17] now incorporate various such methods, either constructing correctness proofs along the way, or reconstructing them from appropriate certificates. (For a small sample, see [7], [9], [18], [21].) Such systems provide powerful tools to support interactive theorem proving. But, frustratingly, they often fail when it comes to fairly routine calculations, leaving users to carry out explicit calculations painstakingly by hand. Consider, for example, the following valid implication:

$$0 < x < y,\ u < v \ \Rightarrow\ 2u + exp(1 + x + x^4) < 2v + exp(1 + y + y^4)$$

The inference is not contained in linear arithmetic or even the theory of real-closed fields. The inference is tight, so symbolic or numeric approximations to the exponential function are of no use. Backchaining using monotonicity properties of addition, multiplication, and exponentiation might suggest reducing the goal to subgoals $2u < 2v$ and $exp(1 + x + x^4) < exp(1 + y + y^4)$, but this introduces some unsettling nondeterminism. After all, one could just as well reduce the goal to

- $2u < exp(1 + y + y^4)$ and $exp(1 + x + x^4) < 2v$, or
- $2u + exp(1 + x + x^4) < 2v$ and $0 < exp(1 + y + y^4)$, or even
- $2u < 2v + 7$ and $exp(1 + x + x^4) < exp(1 + y + y^4) - 7$.

And yet, the inference is entirely straightforward. With the hypothesis $u < v$ in mind, you probably noticed right away that the terms $2u$ and $2v$ can be compared; similarly, the comparison between x and y leads to comparisons between x^4 and y^4, then $1 + x + x^4$ and $1 + y + y^4$, and so on.

The method we propose is based on such heuristically guided forward reasoning, using properties of addition, multiplication, and the function symbols involved. As is common for resolution theorem proving, we try to establish the theorem above by negating the conclusion and deriving a contradiction. We then proceed as follows:

- Put all terms involved into a canonical normal form. This enables us to recognize terms that are the same up to a scalar multiple, and up to associativity and commutativity of addition and multiplication.
- Iteratively call specialized modules to learn new comparisons between subterms, and add these new comparisons to a common "blackboard" structure, which can be accessed by all modules.

The theorem is verified when any given module derives a contradiction using this common information. The procedure fails, on the other hand, when none of the modules can learn anything new. We will see in Section 6 that the method is far from complete, and may not even terminate. On the other hand, it is flexible and extensible, and easily verifies a number of inferences that are not obtained using more principled methods. As a result, it provides a useful complement to more conventional approaches.

We have thus far designed and implemented modules to learn comparisons from the additive and multiplicative structure of terms, as well as a module to instantiate axioms involving general functions. The additive and multiplicative modules have two different implementations, with different characteristic strengths and weaknesses. The first uses a natural but naive Fourier-Motzkin elimination; the second uses more refined geometric techniques. Our prototype implementation, written in Python, is available online:

`https://github.com/avigad/polya`

We have named the system "Polya," after George Pólya, in recognition of his work on inequalities as well as his thoughtful studies of heuristic methods in mathematics (e.g. [16], [27]).

The general idea of deriving inequalities by putting terms in a normal form and combining specialized modules is found in Avigad and Friedman [3], which examines what happens when the additive and multiplicative fragments of real arithmetic are combined. This is analogous to the situation handled by SMT solvers, with the added twist that the languages in question share more than just the equality symbol. Avigad and Friedman show that the universal fragment remains decidable even if both theories include multiplication by rational constants, while the full first-order theory is undecidable. The former decidability result, however, is entirely impractical, for reasons discussed there. Rather, it is the general framework for combining decision procedures and the use of canonical normal forms that we make use of here.

2 The Framework

2.1 Terms and Canonical Forms

We wish to consider terms, such as $3(5x + 3y + 4xy)^2 f(u + v)^{-1}$, that are built up from variables and rational constants using addition, multiplication, integer powers, and function application. To account for the associativity of addition and multiplication, we view sums and products as multi-arity rather than binary operations. We account for commutativity by imposing an arbitrary ordering on terms, and ordering the arguments accordingly.

Importantly, we would also like to easily identify the relationship between terms t and t' where $t = c \cdot t'$, for a nonzero rational constant c. For example, we would like to keep track of the fact that $4y + 2x$ is twice $x + 2y$. Towards that end, we distinguish between "terms" and "scaled terms": a scaled term is just an expression of the form $c \cdot t$, where t is a term and c is a rational constant. We refer to "scaled terms" as "s-terms" for brevity.

Definition 1. *We define the set of* terms $\mathcal{T}$ *and* s-terms $\mathcal{S}$ *by mutual recursion:*

$$\begin{aligned} t, t_i \in \mathcal{T} \quad &:= \quad 1 \mid x \mid \textstyle\sum_i s_i \mid \prod_i t_i^{n_i} \mid f(s_1, \ldots, s_n) \\ s, s_i \in \mathcal{S} \quad &:= \quad c \cdot t \ . \end{aligned}$$

Here x ranges over a set of variables*, f ranges over a set of* function symbols*, $c \in \mathbb{Q}$, and $n_i \in \mathbb{Z}$.*

Thus we view $3(5x+3y+4xy)^2 f(u+v)^{-1}$ as an s-term of the form $3 \cdot t$, where t is the product $t_1^2 t_2^{-1}$, t_1 is a sum of three s-terms, and t_2 is the result of applying f to the single s-term $1 \cdot (u + v)$.

Note that there is an ambiguity, in that we can also view the coefficient 3 as the s-term $3 \cdot 1$. This ambiguity will be eliminated when we define a notion of *normal form* for terms. The notion extends to s-terms: an s-term is then in normal form when it is of the form $c \cdot t$, where t is a term in normal form. (In the special case where $c = 0$, we require t to be the term 1.) We also refer to terms in normal form as *canonical*, and similarly for s-terms.

To define the notion of normal form for terms, we fix an ordering $\prec$ on variables and function symbols, and extend that to an ordering on terms and s-terms. For example, we can arbitrarily set the term 1 to be minimal in the ordering, then variables, then products, then sums, and finally function applications, recursively using lexicographic ordering on the list of arguments (and the function symbol) within the latter three categories. The set of terms in normal form is then defined inductively as follows:

- $1, x, y, z, \ldots$ are terms in normal form.
- $\sum_{i=1 \ldots n} c_i \cdot t_i$ is in normal form provided $c_1 = 1$, each t_i is in normal form, and $t_1 \prec t_2 \prec \ldots \prec t_n$.
- $\prod_i t_i^{n_i}$ is in normal form provided each t_i is in normal form, and $1 \neq t_1 \prec t_2 \prec \ldots \prec t_n$.
- $f(s_1, \ldots, s_n)$ is in normal form if each s_i is.

The details are spelled out in Avigad and Friedman [3]. That paper provides an explicit first-order theory, T, expressing commutativity and associativity of addition and multiplication, distributivity of constants over sums, and so on, such that the following two properties hold:

1. For every term t, there is a unique s-term s in canonical form, such that T proves $t = s$.
2. Two terms t_1 and t_2 have the same canonical normal form if and only if T proves $t_1 = t_2$.

For example, the term $3(5x + 3y + 4xy)^2 f(u + v)^{-1}$ is expressed canonically as $75 \cdot (x + (3/5) \cdot y + (4/5) \cdot xy)^2 f(u + v)^{-1}$, where the constant in the additive term $5x + 3y + 4xy$ has been factored so that the result is in normal form.

The two clauses above provide an axiomatic characterization of what it means for terms to have the same canonical form. As discussed in Section 7, extending the reach of our methods requires extending the notion of a canonical form to include additional common operations.

2.2 The Blackboard

We now turn to the blackboard architecture, which allows modules to share information in a common language. To the addition module, multiplication is a black box; thus it can only make sense of additive information in the shared pool of knowledge. Conversely, the multiplication module cannot make sense of addition. But both modules can make sense of information in the form $t_1 < c \cdot t_2$, where t_1 and t_2 are subterms occurring in the problem. The blackboard enables modules to communicate facts of this shape.

When the user asserts a comparison $t > 0$ to the blackboard, t is first put in canonical form, and names $t_0, t_1, t_2, \ldots$ are introduced for each subterm. It is convenient to assume that t_0 denotes the canonical term 1. Given the example in the last section, the method could go on to define

$$t_1 := x, \quad t_2 := y, \quad t_3 := t_1 t_2, \quad t_4 := t_1 + (3/5) \cdot t_2 + (4/5) \cdot t_3,$$
$$t_5 := u, \quad t_6 := v, \quad t_7 := t_5 + t_6, \quad t_8 = f(t_7), \quad t_9 := t_4^2 t_8^{-1}$$

In that case, $75 \cdot t_9$ represents $3(5x+3y+4xy)^2 f(u+v)^{-1}$. Any subterm common to more than one term is represented by the same name. Separating terms in this way ensures that each module can focus on only those definitions that are meaningful to it, and otherwise treat subterms as uninterpreted constants.

Now any comparison $s \bowtie s'$ between canonical s-terms, where $\bowtie$ denotes any of $<, \leq, >, \geq, =$, or $\neq$, translates to a comparison $c_i t_i \bowtie c_j t_j$, where t_i and t_j name canonical terms. But this, in turn, can always be expressed in one of the following ways:

– $t_i \bowtie 0$ or $t_j \bowtie 0$, or
– $t_i \bowtie c \cdot t_j$, where $c \neq 0$ and $i < j$.

The blackboard therefore maintains the following data:

– a defining equation for each t_i, and
– comparisons between named terms, as above.

Note that this means that, *a priori*, modules can only look for and report comparisons between terms that have been "declared" to the blackboard. This is a central feature of our method: the search is deliberately constrained to focus on a small number of terms of interest. The architecture is flexible enough, however, that modules can heuristically expand that list of terms at any point in the search. For example, our addition and multiplication modules do not consider distributivity of multiplication over addition, beyond multiplication of rational scalars. But if a term $x(y+z)$ appears in the problem, a module could heuristically add the identity $x(y+z) = xy + xz$, adding names for the new terms as needed.

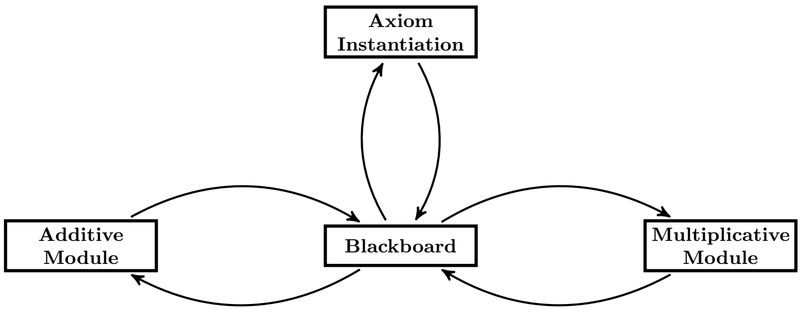


Fig. 1: The Blackboard Architecture

To verify an implication, the user asserts the hypotheses to the blackboard, together with the negation of the conclusion. Individual modules then take turns learning new comparisons from the data, and asserting them to the blackboard as well, until a contradiction is obtained, or no further conclusions can be drawn.

The setup is illustrated by Figure 1. Notice that this is essentially the Nelson-Oppen architecture [25], [31], in which (disjoint) theories communicate by means of a shared logical symbol, typically equality. Here, the shared language is instead assumed to contain the list of comparisons $<, \leq, >, \geq, =, \neq$, and multiplication by rational constants.

Now suppose a module asserts an inequality like $t_3 < 4t_5$ to the blackboard. It is the task of the central blackboard module to check whether the assertion provides new information, and, if so, to update its database accordingly. The task is not entirely straightforward: for example, the blackboard may already contain the inequality $t_3 < 2t_5$, but absent sign information on t_3 or t_5, this does not imply $t_3 < 4t_5$, nor does the converse hold. However, if the blackboard includes the inequalities $t_3 < 2t_5$ and $t_3 \leq 7t_5$, the new assertion is redundant. If, instead, the blackboard includes the inequalities $t_3 < 2t_5$ and $t_3 \leq 3t_5$, the new inequality should replace the second of these. A moment's reflection shows that at most two such inequalities need to be stored for each pair t_i and t_j (geometrically, each represents a half-plane through the origin), but comparisons between t_i or t_j and 0 should be counted among these.

There are additional subtleties: a weak inequality such as $t_3 \leq 4t_5$ paired with a disequality $t_3 \neq 4t_5$ results in a strong inequality; a pair of weak inequalities $t_3 \leq 4t_5$ and $t_3 \geq 4t_5$ should be replaced by an equality; and, conversely, a new equality can subsume previously known inequalities.

3 Fourier-Motzkin

The Fourier-Motzkin algorithm is a quantifier-elimination procedure for the theory of the structure $\langle \mathbb{R}, 0, +, < \rangle)$, that is, the real numbers as an additive ordered group. Nothing changes essentially if we add to the language of that theory the constant 1 and scalar multiplication by c, for each rational c. Here we see that the method can be used to infer comparisons between variables from additive data, and that this can be transported to the multiplicative setting as well.

3.1 The Fourier-Motzkin Additive Module

The Fourier-Motzkin additive module begins with the comparisons $t_i \bowtie c \cdot t_j$ stored in the blackboard, where $\bowtie$ is one of $\leq, <, \geq, >, =$ (disequalities are not used). It also makes use of comparisons $t_i \bowtie 0$, and all definitions $t_i = \sum_j c_j t_{k_j}$ in which the right-hand side is a sum. The goal is to learn new comparisons of the form $t_i \bowtie c \cdot t_j$ or $t_i \bowtie 0$. The idea is simple: to learn comparisons between t_i and t_j, we need only eliminate all the other variables. For example, suppose, after substituting equations, we have the following three inequalities:

$$\begin{aligned} 3t_1 + 2t_2 - t_3 &> 0 \\ 4t_1 + t_2 + t_3 &\geq 0 \\ 2t_1 - t_2 - 2t_3 &\geq 0 \end{aligned}$$

Eliminating t_3 from the first two equations we obtain $7t_1 + 3t_2 > 0$, from which we can conclude $t_1 > (-3/7)t_2$. Eliminating t_3 from the last two equations we obtain $10t_1 + t_2 \geq 0$, from which we can conclude $t_1 \geq (-1/10)t_2$. More generally, eliminating all the variables other than t_i and t_j gives the projection of the convex region determined by the constraints onto the i, j plane, which determines the strongest comparisons for t_i and t_j that are implied by the data.

Constants can be represented using the special variable $t_0 = 1$, which can be treated as any other variable. Thus eliminating all variables except for t_i and t_0 yields all comparisons between t_i and a constant.

The additive module simply carries out the elimination for each pair i, j. In general, though, Fourier-Motzkin elimination can require doubly-exponential time in the number of variables. With a bit of cleverness, one can use previous eliminations to save some work, but for a problem with n subterms, one is still left with $O(n^2)$-many instances of Fourier-Motzkin with up to n variables in each. It is interesting to note that for the examples described in Section 6, the algorithm performs reasonably well. In Section 4, however, we describe a more efficient approach.

3.2 The Fourier-Motzkin Multiplicative Module

The Fourier-Motzkin multiplication module works analogously: given comparisons $t_i \bowtie c \cdot t_j$ or $t_i \bowtie 0$ and definitions of the form $t_i = \prod_j t_{k_j}^{n_j}$, the module aims to learn comparisons of the first two forms. The use of Fourier-Motzkin here is based on the observation that the structure $\langle \mathbb{R}, 0, +, < \rangle$ is isomorphic to the structure $\langle \mathbb{R}^+, 1, \times, < \rangle$ under the map $x \mapsto e^x$. With some translation, the usual procedure works to eliminate variables in the multiplicative setting as well. In the multiplicative setting, however, several new issues arise.

First, the multiplicative module only makes use of terms t_i which are known to be strictly positive or strictly negative. The multiplicative module thus executes a preprocessing stage which tries to infer new sign information from the available data.

Second, the inequalities that are handled by the multiplicative module are different from those handled by the additive module, in that terms can have a rational coefficient. For example, we may have an inequality $3t_2^2 t_5 > 1$; here, the multiplicative constant 3 would correspond to an additive term of $\ln 3$ in the additive procedure. This difference makes it difficult to share code between the additive and multiplicative modules, but the rational coefficients are easy to handle. In order to use exact rational arithmetic in the elimination phase, the elimination procedures are designed to use only integer powers.

Finally, the multiplicative elimination may produce information that cannot be asserted directly to the blackboard, such as a comparison $t_i^2 < 3t_j^2$ or $t_i^3 < 2t_j^2$. In that case, we have to play careful attention to the signs of t_i and t_j and their relation to ± 1 to determine which facts of the form $t_i \bowtie c \cdot t_j$ can be inferred. We compute exact roots of rational numbers when possible, so a comparison $t_i^2 < 9t_j^2$ translates to $t_i < 3t_j$ when t_i and t_j are known to be positive. As a last resort, faced with a comparison like $t_i^2 < 2t_j^2$, we use a rational approximation of $\sqrt{2}$ to try to salvage useful information.

4 Geometric Methods

Although the Fourier-Motzkin modules perform reasonably well on small problems, they are unlikely to scale well. The problem is that many of the inequalities that are produced when a single variable is eliminated are redundant, or subsumed by the others. Thus, by the end of the elimination, the algorithm may be left with hundreds or thousands of comparisons of the form $t_i > c \cdot t_j$, for different values of c. Some optimizations are possible, such as using simplex based methods (e.g. [13]) to filter out some of the redundancies. In this section, however, we show how methods of computational geometry can be used to address the problem more directly. On many problems in our test suite, performance is roughly the same. But on some problems of moderate complexity we have found our implementation of the geometric approach to be faster than the Fourier-Motzkin approach by a factor of five, with the most notable improvement occurring in problems where the ratio of learned comparisons to number of variables is high.

4.1 The Geometric Additive Module

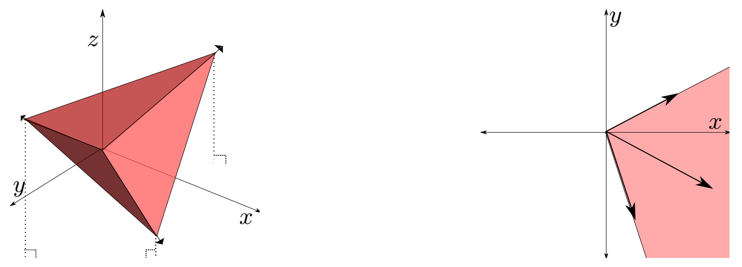
Geometric methods provide an alternative perspective on the task of eliminating variables. A linear inequality $c \leq \sum_{i=0}^{k} c_i \cdot t_i$ determines a half-space in $\mathbb{R}^k$; when $c = 0$, as in the homogenized inequalities in our current problem, the defining hyperplane of the half-space contains the origin. A set of n homogeneous inequalities determines an unbounded pyramidal polyhedron in $\mathbb{R}^k$ with vertex at the origin, called a "polyhedral cone." (Equalities, represented as $(k-1)$-dimensional hyperplanes, simply reduce the dimension of the polyhedron.) The points inside this polyhedron represent solutions to the inequalities. The problem of determining the strongest comparisons between t_i and t_j then reduces to finding the maximal and minimal ratios of the i-th and j-th coordinates of points inside the polyhedron.

We use the following well-known theorem of computational geometry (see [32, Section 1.1]):

Theorem 1. *A subset $P \subseteq \mathbb{R}^k$ is a sum of a convex hull of a finite point set and a conical combination of vectors (a $\mathcal{V}$-*polyhedron*) if and only if it is an intersection of closed half-spaces (an $\mathcal{H}$-*polyhedron*).*

A description of a $\mathcal{V}$-polyhedron is said to be a *$\mathcal{V}$-representation* of the polyhedron, and similarly for $\mathcal{H}$-polyhedrons; there are a number of effective methods to convert between representations.

The comparisons and additive equalities stored in the central blackboard essentially describe an $\mathcal{H}$-representation of a polyhedron. After constructing the corresponding $\mathcal{V}$-representation, it is easy to pick out the implied comparisons as follows. For every pair of variables t_i and t_j, project the set of vertices to



(a) A polyhedral cone in $\mathbb{R}^3$, defined by three half-spaces

(b) Projected to the xy plane, the polyhedron implies $x > 2y$ and $x > -\frac{1}{3}y$

Fig. 2: Variable elimination by geometric projection

the $t_i t_j$ plane. If there is anything to be learned, all (nonzero) vertices must fall in the same halfplane; find the two outermost points (as in Figure 2b) and compute their slopes to the origin. These slopes determine the coefficients c in two comparisons $t_i \bowtie c \cdot t_j$, and the relative position of the two vertices determine the inequality symbols in place of $\bowtie$.

We chose to use Avis' *lrs* implementation of the reverse-search algorithm [4] to carry out the geometric computations. Vertex enumeration algorithms typically assume convexity of the polyhedron: that is, all inequalities are taken to be weak. As it is essential for us to distinguish between $>$ and $\geq$, we use a trick we taken from Dutertre and de Moura [13, Section 5]. Namely, given a set of strict inequalities $\{0 < \sum_{i=0}^{k} c_i^m \cdot t_i : 0 \leq m \leq n\}$, we introduce the new variable $\Delta = \max\{\delta : 0 \leq m \leq n \implies \delta \leq \sum_{i=0}^{k} c_i^m \cdot t_i\}$. We then generate the polyhedron using the latter set of inequalities and the inequality $0 \leq \Delta$. If, in the vertex representation, every vertex has a 0 Δ-coordinate, then the inequalities are only satisfiable when $\Delta = 0$, a contradiction if there are any strict inequalities in the problem. Otherwise, a comparison $t_i \bowtie c \cdot t_j$ is weak if and only if any vertex on the hyperplane $t_i = c \cdot t_j$ has a nonzero Δ coordinate.

4.2 The Geometric Multiplicative Module

As with the Fourier-Motzkin method, multiplicative comparisons $1 \leq \prod_{i=1}^{k} t_i^{e_i}$ can be handled in a similar manner, by restricting to terms with known sign information and taking logarithms. Once again, there is a crucial difference from the additive setting: taking the logarithm of a comparison $c \cdot t_i \cdot t_j^{-1} \bowtie 1$ with $c \neq 1$, one is left with an irrational constant $\log(c)$, and the standard computational methods for vertex enumerations cannot perform exact computations with these terms.

To handle this situation we introduce new variables to represent the logarithms of the prime numbers occurring in these constant terms. Let $p_1, \ldots, p_l$ represent the prime factors of all constant coefficients in such a problem, and for each $1 \leq i \leq l$, let q_i be a variable representing $\log p_i$. We can then rewrite each $c \cdot t_i \cdot t_j^{-1} \bowtie 1$ as $p_1^{d_0} \cdot \ldots \cdot p_l^{d_l} \cdot t_i \cdot t_j^{-1} \bowtie 1$. Taking logarithms of all such inequalities produces a set of additive inequalities in $k + l$ variables.

In order to find the strongest comparisons between t_i and t_j, we can no longer project to the ij plane, but instead look at the $ijq_1 \ldots q_l$ hyperplane. The simple arithmetical comparisons to find the two strongest comparisons are no longer applicable; we face the harder problem of converting the vertex representation of a polyhedron to a half-space representation. This problem is dual to the conversion in the opposite direction, and the same computational packages are equipped to solve it. Experimentally, we have found Fukuda's *cdd* implementation of Motzkin's double description method [14] to be faster than *lrs* at this procedure.

5 Arbitrary Function Symbols

The inferences captured by the addition and multiplication modules constitute a fragment of the theory of real-closed fields, roughly, that theory "minus" the distributivity of multiplication over addition [3]. Recall, however, that we have alo included arbitrary function symbols in the language. An advantage to our framework is that we do not have to treat function terms as uninterpreted constants; rather, we can seamlessly add modules that (partially) interpret these symbols and learn relevant inequalities concerning them.

To start with, a user may wish to add axioms asserting that a particular function f is nonnegative, monotone, or convex. For example, the following axiom expresses that f is nondecreasing:

$$\forall x, y.\ x \leq y \rightarrow f(x) \leq f(y)$$

Given such an axiom, Polya's function module searches for useful instantiations during the course of a search, and may thus learn useful information.

Specifically, given a list of universal axioms in variables $v_1 \ldots v_n$, the instantiation module searches for relevant assignments $v_i \mapsto c_i \cdot t_{j_i}$, where each c_i is a constant and each t_{j_i} is a subterm in the given problem. Each axiom is then instantiated with these assignments, and added to the central blackboard as a set of disjunctive clauses. As the search progresses, elements of these clauses are refuted; when only one remains, it is added to the blackboard, as a new piece of information available to all the modules.

The task is a variant of the classic matching problem, but there are at least three aspects of our framework that present complications. First, given that we consider terms with a rational scalar multiplicative constant, the algorithm has to determine those values. So, in the example above, x and y can be instantiated to an s-term $c \cdot t_i$ for any c, when such an instantiation provides useful information. Second, we need to take into account the associativity and commutativity of operations like addition and multiplication, so, for example, a term $f(x+y)$ can be unified with a term $f(t_i + t_j + t_k)$ found in the blackboard in multiple ways. Finally, although the framework is built around the idea of restricting attention to subterms occurring in the original problem, at times it is useful to consider new terms. For example, given the axiom

$$\forall x, y.\ f(x+y) \leq f(x) + f(y)\ ,$$

it is clearly a good idea to instantiate x and y to t_i and t_j, respectively, whenever $f(t_i + t_j)$, $f(t_i)$, and $f(t_j)$ all appear in the blackboard, even if the term $f(t_i) + f(t_j)$ doesn't.

In short, we wish to avoid difficult calculations of rational constants, expensive matching up to associativity and commutativity (see e.g. [10]), and unrestrained creation of new terms, while at the same time making use of potentially useful instantiations. The solution we adopted is to use function terms to trigger and constrain the matching process, an idea commonly used by SMT solvers [25] [11]. Given a universal axiom $(\forall v_1 \ldots v_n) F(v_1, \ldots, v_n)$, F is first converted into clausal normal form, and each clause F' is treated separately. We take the *trigger set* of F' be the set of all functional subterms contained in F. A straightforward unification procedure finds all assignments that map each trigger element to a (constant multiple of a) problem term, and these assignments are used to instantiate the full clause F'. The instantiated clause is asserted to the central blackboard, which checks for satisfied and falsified literals.

For u a term containing unification variables $\{v_i\}$ and σ an assignment mapping $v_i \mapsto c_i \cdot t_{j_i}$, the problem of matching $\sigma(u)$ to a problem term t_j is nontrivial: the matching must be done modulo equalities stored in the blackboard. For example, if $t_1 = t_2 + t_3$, $t_4 = 2t_3 - t_5$, and $t_6 = f(t_1 + t_4)$, then given the assignment $\{v_1 \mapsto t_2 - t_5, v_2 \mapsto 3t_3\}$, the term $u = f(v_1 + v_2)$ should be matched to t_6. We thus combine a standard unification algorithm, which suggests candidate assignments to the variables occurring in an axiom, with Gaussian elimination over additive and multiplicative equations, to find the relevant matching substitutions.

The function module thus provides a general and natural scheme for incorporating axioms. It is flexible enough to integrate other methods for matching axioms and triggering the creation of new problem terms. We expect to achieve better performance for special function symbols, however, by refining the notion of canonical normal form. For example, we can handle the associative-commutative function $\min(c_1 \cdot t_1, \ldots, c_n \cdot t_n)$ by sorting the arguments according to the ordering on terms and scaling so that $c_1 = \pm 1$. Similarly, we can assume that in any subterm $|c \cdot t|$, we have $c = 1$. We intend to explore use for normal forms for arbitrary powers and logarithms, making use of ideas from [24]. Moreover, special-purpose modules can be used to contribute more refined information and inferences. For example, we expect that such a special-purpose module is be needed to effectively manage the additive and multiplicative nature of the exponentiation, e.g. under the identity $exp(\sum_i c_i \cdot t_i) = \prod_i exp(t_i)^{c_i}$.

6 Examples

The current distribution of Polya includes a number of examples that are designed to illustrate the method's strengths, as well as some of its weaknesses.

For comparison, we verified a number of these examples in Isabelle, trying to use Isabelle's automated tools as much as possible. These include "auto," an internal tableau theorem prover which also invokes a simplifier and arithmetic reasoning methods, and Sledgehammer [22], [7], which heuristically selects a body of facts from the local context and background library, and exports it to various provers. We also sent some of the inferences directly to the SMT solver Z3 [12].

To start with, Polya handles inferences involving linear real inequalities, which are verified automatically by many interactive theorem proving systems. It can also handle purely multiplicative inequalities such as

$$0 < u < v < 1,\ 2 \leq x \leq y \ \Rightarrow\ 2u^2x < vy^2, \tag{1}$$

which are not often handled automatically. It can solve problems that combine the two, like these:

$$x > 1 \ \Rightarrow\ (1+y^2)x > 1+y^2 \tag{2}$$

$$0 < x < 1 \ \Rightarrow\ 1/(1-x) > 1/(1-x^2) \tag{3}$$

$$0 < u,\ u < v,\ 0 < z,\ z+1 < w \ \Rightarrow\ (u+v+z)^3 < (u+v+w)^5 \tag{4}$$

It also handles inferences that combine such reasoning with axiomatic properties of functions, such as:

$$(\forall x, y.\ x \leq y \rightarrow f(x) \leq f(y)),\ u < v,\ x < y \ \Rightarrow\ u + f(x) < v + f(y) \tag{5}$$

$$(\forall x.\ f(x) \leq 1),\ u < v,\ 0 < w \ \Rightarrow\ u + w \cdot f(x) < v + w \tag{6}$$

Isabelle's auto and Sledgehammer fail on all of these but (5) and (6), which are proved by resolution theorem provers. Sledgehammer can verify more complicated variants of (5) and (6) by sending them to Z3, but fails on only slightly more complicated examples, such as:

$$\begin{aligned}(\forall x, y.\ x \leq y \rightarrow f(x) \leq f(y)),\ u < v,\ 1 < w,\ 2 < s,&\\ (w+s)/3 < v,\ x \leq y \ \Rightarrow\ u + f(x) \leq v + f(y)&\end{aligned} \tag{7}$$

$$(\forall x.\ f(x) \leq 2),\ u < v,\ 0 < w \ \Rightarrow\ u + w \cdot (f(x) - 1) < v + w \tag{8}$$

Z3 gets most of these when called directly, but also fails on (8), and the following slight variant of (5):

$$(\forall x, y.\ x \leq y \rightarrow f(x) \leq f(y)),\ u < v,\ 1 < v,\ x \leq y \ \Rightarrow\ u + f(x) \leq v^2 + f(y) \tag{9}$$

Moreover, when handling nonlinear equations, Z3 "flattens" polynomials, which makes a problem like (4) extremely difficult. It takes Z3 a couple of minutes when the exponents 3 and 5 in that problem are replaced by 9 and 19, respectively. Polya verifies all of these problems in a fraction of a second, and is insensitive to the exponents in (4). It is also unfazed if any of the variables above are replaced by more complex terms.

Polya has no problem with examples such as

$$0 < x < y,\ u < v \ \Rightarrow\ 2u + \mathit{exp}(1 + x + x^4) < 2v + \mathit{exp}(1 + y + y^4), \tag{10}$$

mentioned in the introduction. Sledgehammer verifies this using resolution, and slightly more complicated examples by calling Z3 with the monotonicity of *exp*. Sledgehammer restricts Z3 to linear arithmetic so that it can reconstruct proofs in Isabelle, so to verify (10) it provides Z3 with the monotonicity of the power function as well. When called directly on this problem with this same information, however, Z3 resorts to nonlinear mode, and fails.

Sledgehammer fails on an example that arose in connection with a formalization of the Prime Number Theorem, discussed in [2]:

$$0 \le n,\ n < (K/2)x,\ 0 < C,\ 0 < \varepsilon < 1 \ \Rightarrow\ \left(1 + \frac{\varepsilon}{3(C+3)}\right) \cdot n < Kx \tag{11}$$

Z3 verifies it when called directly. Sledgehammer also fails on these [3]:

$$0 < x < y \ \Rightarrow\ (1+x^2)/(2+y)^{17} < (1+y^2)/(2+x)^{10} \tag{12}$$

$$\begin{aligned}&(\forall x, y.\ x < y \rightarrow \mathit{exp}(x) < \mathit{exp}(y)),\\ &\qquad 0 < x < y \ \Rightarrow\ (1+x^2)/(2+\mathit{exp}(y)) \ge (2+y^2)/(1+\mathit{exp}(x))\ .\end{aligned} \tag{13}$$

Z3 gets (12) but not (13). Neither Sledgehammer nor Z3 get these:

$$(\forall x, y.\ f(x+y) = f(x)f(y)),\ a > 2,\ b > 2 \ \Rightarrow\ f(a+b) > 4 \tag{14}$$

$$(\forall x, y.\ f(x+y) = f(x)f(y)),\ a+b > 2,\ c+d > 2 \ \Rightarrow\ f(a+b+c+d) > 4 \tag{15}$$

Polya verifies all of the above easily.

Let us consider two examples that have come up in recent Isabelle formalizations by the first author. Billingsley [6, page 334] shows that if f is any function from a measure space to the real numbers, the set of continuity points of f is Borel. Formalizing the proof involved verifying the following inequality:

$$\begin{aligned}&i \ge 0,\ |f(y) - f(x)| < 1/(2(i+1)),\\ &\qquad |f(z) - f(y)| < 1/(2(i+1)) \ \Rightarrow\ |f(x) - f(y)| < 1/(i+1)\ .\end{aligned} \tag{16}$$

Sledgehammer and Z3 fail on this, while Polya verifies it given only the triangle inequality for the absolute value.

The second example involves the construction of a sequence $f(m)$ in an interval (a, b) with the property that for every $m > 0$, $f(m) < a + (b-a)/m$. The proof required showing that $f(m)$ approaches a from the right, in the sense that for every $x > a$, $f(m) < x$ for m sufficiently large. A little calculation shows that $m \ge (b-a)/(x-a)$ is sufficient. We can implicitly restrict the domain of f to the integers by considering only arguments $\lceil m \rceil$; thus the required inference is

$$\begin{aligned}&(\forall m.\ m > 0 \rightarrow f(\lceil m \rceil) < a + (b-a)/\lceil m \rceil),\\ &\qquad a < b,\ x > a,\ m \ge (b-a)/(x-a) \ \Rightarrow\ f(\lceil m \rceil) < x\ .\end{aligned} \tag{17}$$

Sledgehammer and Z3 do not capture this inference, and the Isabelle formalization was tedious. Polya verifies it immediately using only the information $\lceil x \rceil \ge x$ for every x.

When restricted to problems involving linear arithmetic and axioms for function symbols, the behavior of Z3 and Polya is similar, although Z3 is vastly more efficient. As the examples above show, Polya's advantages show up in problems that combine multiplicative properties with either linear arithmetic or axioms. In particular, Z3 procedures for handling nonlinear problems do not incorporate axioms for function symbols.

Of course, Polya fails on wide classes of problems where other methods succeed. It is much less efficient than the best linear solvers, for example, and so should not be expected to scale to large industrial problems. Because the multiplicative module only takes advantage of equations where the signs of all terms are known, Polya fails disappointingly on the trivial inference

$$x > 0,\ y < z \ \Rightarrow\ xy < xz\ . \tag{18}$$

But the problem is easily solved given a mechanism for splitting on the signs of y and z (see the discussion in the next section). Another shortcoming, in contrast to methods which begin by flattening polynomials, is that Polya does not, *a priori*, make use of distributivity at all, beyond the distributivity of multiplication by a rational constant over addition. Any reasonable theorem prover for the theory of real closed fields can easily establish

$$x^2 + 2x + 1 \geq 0, \tag{19}$$

which can also be obtained simply by writing the left-hand side as $(x+1)^2$. But, as pointed out by Avigad and Friedman [3], the method implemented by Polya is, in fact, nonterminating on this example.

We also tried a number of these problems with MetiTarski [1], which combines resolution theorem proving with procedures for real-closed fields as well as symbolic approximations to transcendental function. We found that MetiTarski does well on problems in the language of real-closed fields, but not with axioms for interpreted functions, nor with the examples with *exp* above.

For problems like these, time constraints are not a serious problem. Polya solves a suite of 51 problems, including all the ones above, in about 2 seconds on an ordinary desktop using the polytope packages, and in about 5.5 seconds using Fourier-Motzkin. Test files for Isabelle, Z3, and MetiTarski, as well as more precise benchmark results, can be found in the distribution.

7 Conclusions and Future Work

One advantage of the method described here is that it should not be difficult to generate proof certificates that can be verified independently, and used to construct formal derivations within client theorem provers. For procedures using real closed fields, this is much more difficult; see [21] [18].

Another interesting heuristic method, implemented in ACL2, is described in [19]. We have not carried out a detailed comparison, but the method is considerably different from ours. (For example, it flattens polynomial terms.)

We envision numerous extensions to our method. One possibility is to implement case splitting and conflict-driven clause learning (CDCL) search, as do contemporary SMT solvers. For example, recall that the multiplicative routines only work insofar as the signs of subterms are known. It is often advantageous, therefore, to split on the signs on subterms. More generally, we need to implement mechanisms for backtracking, and also make the addition and multiplication modules incremental.

There are many ways our implementation could be optimized, and, of course, we would gain efficiency by moving from Python to a compiled language like C++. We find it encouraging, however, that even our unoptimized prototype performs well on interesting examples. It seems to us to be more important, therefore, to explore extensions of these methods, and try to capture wider classes of inequalities. This includes reasoning with exponents and logarithms; reasoning about the integers as a subset of the reals; reasoning about common functions, such as trigonometric functions, and heuristically allowing other natural moves in the search, such as flattening or factoring polynomials, when helpful. We would also like to handle second-order operators like integrals and sums, and integrate better with external theorem proving methods.

We emphasize again that this method is not designed to replace conventional methods for proving linear and nonlinear inequalities, which are typically much more powerful and efficient in their intended domains of application. Rather, our method is intended to complement these, capturing natural but heterogeneous patterns of reasoning that would otherwise fall through the cracks. What makes the method so promising is that it is open-ended and extensible. A good deal of experimentation is needed to determine how well the method scales and where the hard limitations lie.

Acknowledgment. We are grateful to Leonardo de Moura for helpful information and suggestions.

References

1. B. Akbarpour and L. Paulson. MetiTarski: An Automatic Prover for the Elementary Functions. In S. Autexier et al., editors, *AISC/MKM/Calculemus 2008*, pages 217–231. Springer, 2008.
2. J. Avigad, K. Donnelly, D. Gray, P. Raff. A formally verified proof of the prime number theorem. *ACM Trans. Comput. Logic*, 9(1):2, 2007.
3. J. Avigad and H. Friedman. Combining decision procedures for the reals. *Log. Methods Comput. Sci.*, 2(4):4:4, 42, 2006.
4. D. Avis. Living with lrs. In *Discrete and computational geometry (Tokyo, 1998)*, pages 47–56. Springer, 2000.
5. S. Basu, R. Pollack, M. Roy. *Algorithms in real algebraic geometry.* Springer, 2003.
6. P. Billingsley. *Probability and measure.* John Wiley & Sons Inc., 3rd edition, 1995.
7. J. Blanchette, S. Böhme, L. Paulson. Extending Sledgehammer with SMT solvers. *Automated Deduction–CADE-23*, pages 116–130, 2011.
8. S. Boyd, L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

9. A. Chaieb and T. Nipkow. Proof synthesis and reflection for linear arithmetic. *J. Autom. Reasoning*, 41(1):33–59, 2008.
10. E. Contejean. A Certified AC Matching Algorithm. In van Oostrom, ed., *Rewriting Techniques and Applications*, pages 70–84. Springer, 2004.
11. L. de Moura, N. Bjørner. Efficient E-Matching for SMT Solvers. In *CADE*, 183–198, 2007.
12. L. de Moura, N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, 337–340, 2008.
13. B. Dutertre and L. de Moura. A fast linear-arithmetic solver for dpll(t). In T. Ball and R. Jones, editors, *CAV 2006*, pages 81–94. Springer, 2006.
14. K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and computer science (Brest, 1995)*, pages 91–111. Springer, 1996.
15. D. J. H. Garling. *Inequalities: a journey into linear analysis.* Cambridge University Press, Cambridge, 2007.
16. G. H. Hardy, J. E. Littlewood, G. Pólya. *Inequalities.* Cambridge University Press, Cambridge, 1988. Reprint of the 1952 edition.
17. J. Harrison. HOL light: a tutorial introduction. In M. Srivas and A. Camilleri, editors, *FMCAD*, pages 265–269, Springer, 1996.
18. J. Harrison. Verifying Nonlinear Real Formulas Via Sums of Squares. In K. Schneider and J. Brandt, editors, *TPHOLs*, pages 102–118. Springer, 2007.
19. W. A. Hunt, R. B. Krug, J. Moore. Linear and nonlinear arithmetic in ACL2. In D. Geist and E. Tronci, eds., Correct Hardware Design and Verification Methods, pages 319–333. Springer, 2003.
20. C. N. Jones, E. C. Kerrigan, J. M. Maciejowski. Equality set projection: A new algorithm for the projection of polytopes in halfspace representation. Technical report, Department of Engineering, University of Cambridge, March 2004.
21. S. McLaughlin and J. Harrison. A proof producing decision procedure for real arithmetic. In R. Nieuwenhuis, editor, *Automated Deduction – CADE-20*, pages 295–314, Springer, 2005.
22. J. Meng and L. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.
23. R. Moore, R. Kearfott, M. Cloud. *Introduction to interval analysis.* Society for Industrial and Applied Mathematics (SIAM), 2009.
24. J. Moses. Algebraic simplification: A guide for the perplexed. *Communications of the ACM*, 14:527–537, 1971.
25. G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions of Programming Languages and Systems*, 1:245–257, 1979.
26. T. Nipkow, L. Paulson, M. Wenzel. *Isabelle/HOL. A proof assistant for higher-order logic.* Springer, 2002.
27. G. Polya. *How to solve it.* Princeton University Press, Princeton, NJ, 2004.
28. V. Prevosto and U. Waldmann. SPASS+T. In G. Sutcliffe et al., editors, *ES-CoR: Empirically Successful Computerized Reasoning 2006*, pages 18–33. CEUR Workshop Proceedings, 2006.
29. W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:4–13, 1992.
30. A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1986.
31. C. Tinelli and M. Harandi. A New Correctness Proof of the Nelson-Oppen Combination Procedure. In F. Baader and K. Schulz, editors, *Frontiers of Combining Systems (FroCos)*, pages 103–119. Kluwer Academic Publishers, 1996.
32. G. Ziegler. *Lectures on polytopes.* Springer, 1995.