Extending a Resolution Prover for Inequalities on Elementary Functions

Behzad Akbarpour and Lawrence C. Paulson

Computer Laboratory, University of Cambridge, England
{ba265,lcp}@cl.cam.ac.uk

Abstract. Experiments show that many inequalities involving exponentials and logarithms can be proved automatically by combining a resolution theorem prover with a decision procedure for the theory of real closed fields (RCF). The method should be applicable to any functions for which polynomial upper and lower bounds are known. Most bounds only hold for specific argument ranges, but resolution can automatically perform the necessary case analyses. The system consists of a superposition prover (Metis) combined with John Harrison's RCF solver and a small amount of code to simplify literals with respect to the RCF theory.

1 Introduction

Despite the enormous progress that has been made in the development of decision procedures, many important problems are not decidable. In previous work [1], we have sketched ideas for solving inequalities over the elementary functions, such as exponentials, logarithms, sines and cosines. Our approach involves reducing them to inequalities in the theory of real closed fields (RCF), which is decidable. We argued that merely replacing occurrences of elementary functions by polynomial upper or lower bounds sufficed in many cases. However, we offered no implementation. In the present paper, we demonstrate that the method can be implemented by combining an RCF decision procedure with a resolution theorem prover.

The alternative approach would be to build a bespoke theorem prover that called theory-specific code. Examples include Analytica [8] and Weierstrass [6], both of which have achieved impressive results. However, building a new system from scratch will require more effort than building on existing technology. Moreover, the outcome might well be worse. For example, despite the well-known limitations of the sequent calculus, both Analytica and Weierstrass rely on it for logical reasoning. Also, it is difficult for other researchers to learn from and build upon a bespoke system. In contrast, Verifun [10] introduced the idea of combining existing SAT solvers with existing decision procedures; other researchers grasped the general concept and now SMT (SAT Modulo Theories) has become a well-known system architecture.

Our work is related to SPASS+T [17], which combines the resolution theorem prover SPASS with a number of SMT tools. However, there are some differences

between the two approaches. SPASS+T extends the resolution's test for unsatisfiability by allowing the SMT solver to declare the clauses inconsistent, and its objective is to improve the handling of quantification in SMT problems. We augment the resolution calculus to simplify clauses with respect to a theory, and our objective is to solve problems in this theory.

Our work can therefore be seen to address two separate questions.

- Can inequalities over the elementary functions be solved effectively?
- Can a modified resolution calculus serve as the basis for reasoning in a highly specialized theory?

At present we only have a small body of evidence, but the answer to both questions appears to be *yes*. The combination of resolution with a decision procedure for RCF can prove many theorems where the necessary case analyses and other reasoning steps are found automatically. An advantage of this approach is that further knowledge about the problem domain can be added declaratively (as axioms) rather than procedurally (as code). We achieve a principled integration of two technologies by using one (RCF) in the simplification phase of the other (resolution).

We eventually intend to output proofs where at least the main steps are justified. Claims would then not have to be taken on trust, and such a system could be integrated with an interactive prover such as Isabelle [16]. The tools we have combined are both designed for precisely such an integration [13, 15].

Paper outline. We begin (§2) by presenting the background for this work, including specific upper and lower bounds for the logarithm and exponential functions. We then describe our methods (§3): which axioms we used and how we modified the automatic prover. We work through a particular example (§4), indicating how our combined resolution/RCF solver proves it. We present a table of results (§5) and finally give brief conclusions (§6).

2 Background

The initial stimulus for our project was Avigad's formalization, using Isabelle, of the Prime Number Theorem [2]. This theorem concerns the logarithm function, and Avigad found that many obvious properties of logarithms were tedious and time-consuming to prove. We expect that proofs involving other so-called elementary functions, such as exponentials, sines and cosines, would be equally difficult. Avigad, along with Friedman, made an extensive investigation [3] into new ideas for combining decision procedures over the reals. Their approach provides insights into how mathematicians think. They outline the leading decision procedures and point out how easily they perform needless work. They present the example of proving

$$\frac{1+x^2}{(2+y)^{17}} < \frac{1+y^2}{(2+x)^{10}}$$

from the assumption $0 < x < y$: the argument is obvious by monotonicity, while mechanical procedures are likely to expand out the exponents. To preclude this

possibility, Avigad and Friedman have formalized theories of the real numbers in which the distributivity axioms are restricted to multiplication by constants. As computer scientists, we do not see how this sort of theory could lead to practical tools or be applied to the particular problem of logarithms. We prefer to use existing technology, augmented with search and proof heuristics to this problem domain. We have no interest in completeness—these problems tend to be undecidable anyway—and do not require the generated proofs to be elegant.

Our previous paper [1] presented families of upper and lower bounds for the exponential and logarithm functions. These families, indexed by natural numbers, converge to their target functions. The examples described below use some members of these families which are fairly loose bounds, but adequate for many problems.

$$\frac{x-1}{x} \le \ln x \le \frac{3x^2-4x+1}{2x^2} \qquad \left(\frac{1}{2} \le x \le 1\right)$$

$$\frac{-x^2+4x-3}{2} \le \ln x \le x-1 \qquad (1 \le x \le 2)$$

$$\frac{-x^2+8x-8}{8} \le \ln x \le \frac{x}{2} \qquad (2 \le x \le 4)$$

Fig. 1. Upper and Lower Bounds for Logarithms

$$\frac{x^3+3x^2+6x+6}{6} \le \exp x \le \frac{x^2+2x+2}{2} \qquad (-1 \le x \le 0)$$

$$\frac{2}{x^2-2x+2} \le \exp x \le \frac{6}{-x^3+3x^2-6x+6} \qquad (0 \le x \le 1)$$

Fig. 2. Upper and Lower Bounds for Exponentials

Figure 1 presents the upper and lower bounds for logarithms used in this paper. Note that each of these bounds constrains the argument to a closed interval. This particular family of bounds is only useful for finite intervals, and proofs involving unbounded intervals must refer to other properties of logarithms, such as monotonicity. Figure 2 presents upper and lower bounds for exponentials, which are again constrained to narrow intervals. Such constraints are necessary: obviously there exists no polynomial upper bound for $\exp x$ for unbounded x, and a bound like $\ln x \le x - 1$ is probably too loose to be useful for large x. Tighter constraints on the argument allow tighter bounds, but at the cost of requiring case analysis on x, which complicates proofs.

Our approach to proving inequalities is to replace occurrences of functions such as ln by suitable bounds, and then to prove the resulting algebraic inequal-

ity. Our previous paper walked through a proof of

$$-\frac{1}{2} \leq x \leq 3 \Longrightarrow \ln(1+x) \leq x. \tag{1}$$

One of the cases reduced to the following problem:

$$\frac{x}{1+x} + \frac{1}{2}\left(\frac{-x}{1+x}\right)^2 \leq x$$

As our paper shows, this problem is still non-trivial, but fortunately it belongs to a decidable theory. We have relied on two readable histories of this subject [9, 15]. Tarski proved the decidability of the theory of Real Closed Fields (RCF) in the 1930s: quantifiers can be eliminated from any inequality over the reals involving only the operations of addition, subtraction and multiplication. It is inefficient: the most sophisticated decision procedure, *cylindrical algebraic decomposition* (CAD), can be doubly exponential. We use a simpler procedure, implemented by McLaughlin and Harrison [15], who in their turn credit Hörmander [12] and others. For the RCF problems that we generate, the decision procedure usually returns quickly: as Table 1 shows, most inequalities are proved in less than one second, and each proof involves a dozen or more RCF calls.

Although quantifier elimination is hyper-exponential, the critical parameters are the degrees of the polynomials and the number of variables in the formula. The length of the formula appears to be unimportant. At present, all of our problems involve one variable, but the simplest way to eliminate quotients and roots involves introducing new variables. We do encounter situations where RCF does not return.

Our idea of replacing function occurrences by upper or lower bounds involves numerous complications. In particular, most bounds are only valid for limited argument ranges, so proofs typically require case splits to cover the full range of possible inputs. For example, three separate upper bounds are required to prove equation (1). Another criticism is that bounds alone cannot prove the trivial theorem

$$0 < x \leq y \Longrightarrow \ln x \leq \ln y,$$

which follows by the monotonicity of the function ln. Special properties such as monotonicity must somehow be built into the algorithm. Search will be necessary, since some proof attempts will fail. If functions are nested, the approach has to be applied recursively. We could have written code to perform all of these tasks, but it seems natural to see whether we can add an RCF solver to an existing theorem prover instead.

For the automatic theorem prover, we chose Metis [13], developed by Joe Hurd. It is a clean, straightforward implementation of the superposition calculus [4]. Metis, though not well known, is ideal at this stage in our research. It is natural to start with a simple prover, especially considering that the RCF decision procedure is more likely to cause difficulties.

3 Method

Most resolution theorem provers implement some variant of the inference loop described by McCune and Wos [14]. There are two sets of clauses, *Active* and *Passive*. The Active set enjoys the invariant that every one of its elements has been resolved with every other, while the Passive set consists of clauses waiting to be processed. At each iteration, these steps take place:

- An element of the Passive set (called the *given* clause) is selected and moved to the Active set.
- The given clause is resolved with every member of the Active set.
- Newly inferred clauses are first simplified, for example by rewriting, then added to the Passive set. (They can also simplify the Active and Passive sets by subsumption, an important point but not relevant to this paper.)

Resolution uses purely syntactical unification: no theory unification is involved. Our integration involves modifying the simplification phase to take account of the RCF theory. Algebraic terms are simplified and put into a canonical form. Literals are deleted if the RCF solver finds them to be inconsistent with algebraic facts present in the clauses. Both simplifications are essential. The canonical form eliminates a huge amount of redundant representations, for example the $n!$ permutations of the terms of $x_1 + \cdots + x_n$. Literal deletion generates the empty clause if a new clause is inconsistent with existing algebraic facts, and more generally it eliminates much redundancy from clauses.

To summarize, we propose the following combination method:

1. Negate the problem and Skolemize it, finally converting the result into conjunctive normal form (CNF) represented by a list of conjecture clauses.
2. Combine the conjecture clauses with a set of axioms and make a problem file in TPTP format, for input to the resolution prover.
3. Apply the resolution procedure to the clauses. Simplify new clauses as described below before adding them to the Passive set.
4. If a contradiction is reached, we have refuted the negated formula.

3.1 Polynomial Simplification

All terms built up using constants, negation, addition, subtraction, and multiplication can be considered as multivariate polynomials. Following Grégoire and Mahboubi [11], we have chosen a canonical form for them: Horner normal form, also called the *recursive* representation.[1] An alternative is the *distributed* representation, a flat sum with an ordering of the monomials; however, our approach is often adopted and is easy to implement.

Any univariate polynomial can be rewritten in recursive form as

$$p(x) = a_n x^n + \cdots + a_1 x + a_0 = a_0 + x(a_1 + x(a_2 + \cdots x(a_{n-1} + x a_n))$$

[1] A representation is called *canonical* if two different representations always correspond to two different objects.

We can consider a multivariate polynomial as a polynomial in one variable whose coefficients are themselves a canonical polynomial in the remaining variables. We maintain a list with the innermost variable at the head, and this will determine the arrangement of variables in the canonical form. We adopt a *sparse* representation: zero terms are omitted.

For example, if variables from the inside out are x, y and z, then we represent the polynomial $3xy^2 + 2x^2yz + zx + 3yz$ as

$$[y(z3)] + x([z1 + y(y3)] + x[y(z2)]),$$

where the terms in square brackets are considered as coefficients. Note that we have added numeric literals to Metis: the constant named 3, for example, denotes that number, and $3 + 2$ simplifies to 5.

We define arithmetic operations on canonical polynomials, subject to a fixed variable ordering. For addition, our task is to add $c + xp$ and $d + yq$. If x and y are different, one or other is recursively added to the constant coefficient of the other. Otherwise, we just compute $(c + xp) + (d + xq) = (c + d) + x(p + q)$, returning simply $c + d$ if $p + q = 0$. For negation, we recursively negate the coefficients, while subtraction is an easy combination of addition and negation.

We can base a recursive definition of polynomial multiplication on the following equation, solving the simpler sub-problems $p \times d$ and $p \times q$ recursively:

$$p \times (d + yq) = (p \times d) + (0 + y(p \times q))$$

However, for $0 + y(p \times q)$ to be in canonical form we need y to be the topmost variable overall, with p having no variables strictly earlier in the list. Hence, we first check which polynomial has the earlier topmost variable and exchange the operands if necessary. Powers p^n (for fixed n) are just repeated multiplication.

Any algebraic term can now be translated into canonical form by transforming constants and variables, then recursively applying the appropriate canonical form operations. We simplify a formula of the form $X \leq Y$ by converting $X - Y$ to canonical form, finally generating an equivalent form $X' \leq Y'$ with X' and Y' both canonical polynomials with their coefficients all positive. We simplify $1 + x \leq 4$ to $x \leq 3$, for example. Any fixed format can harm completeness, but note that the literal deletion strategy described below is indifferent to the particular representation of a formula.

3.2 Literal Deletion

We can distinguish a literal L in a clause by writing the clause as $A \vee L \vee B$, where A and B are disjunctions of zero or more literals. Now if $\neg A \wedge \neg B \wedge L$ is inconsistent, then the clause is equivalent to $A \vee B$; therefore, L can be deleted. We take the formula $\neg A \wedge \neg B \wedge L$ to be inconsistent if the RCF solver reduces it to **false**.

For example, consider the clause $x \leq 3 \vee x = 3$. (Such redundancies arise frequently.) We focus on the first literal and note that $x \neq 3 \wedge x \leq 3$ is consistent, so that literal is preserved. We focus on the second literal and note that

$\neg(x \leq 3) \wedge x = 3$ is inconsistent, so that literal is deleted: we simplify the clause to $x \leq 3$.

This example illustrates a subtle point concerning variables and quantifiers. In the clause $x \leq 3 \vee x = 3$, the symbol x is a constant, typically a Skolem constant originating in a existentially quantified variable in the negated conjecture. Before calling RCF, we again convert x to an existentially quantified variable. Here is a precise justification of this practice. In the semantics of first-order logic [5], a *structure* for a first-order language L comprises a non-empty domain D and interprets the constant, function and relation symbols of L as corresponding elements, functions and relations over D. The clause $x \leq 3 \vee x = 3$ is satisfied by any structure that gives its symbols their usual meanings over the real numbers and also maps x to 3; it is, however, falsified if the structure instead maps x to 4. To prove the inconsistency of say $\neg(x \leq 3) \wedge x = 3$, we give the RCF solver a closed formula, $\exists x\, [\neg(x \leq 3) \wedge x = 3]$; if this formula is equivalent to **false**, then there exists no interpretation of x satisfying the original formula. More generally, we can prove the inconsistency of a ground formula by existentially quantifying over its uninterpreted constants before calling RCF.

We strengthen this test by taking account of all algebraic facts known to the prover. (At present, we consider only algebraic facts that are ground.) Whenever the automatic prover asserts new clauses, we extract those that concern only addition, subtraction and multiplication. We thus accumulate a list of algebraic clauses that we include in the formula that is delivered to the RCF solver. For example, consider proving $-\frac{1}{2} \leq u \leq 3 \Longrightarrow \ln(u+1) \leq u$. Upon termination, it has produced the following list of algebraic clauses:

```
F, ~(1 <= u), ~(0 <= u) \/ ~(u <= 1), 0 <= 1 + u * 2, u <= 3
```

This list is built from right to left. It begins with $u \leq 3$ from the problem statement, while $0 \leq 1 + u \times 2$ is soon derived from $-\frac{1}{2} \leq u$. The other clauses arise during the proof, which terminates with **false**.

To summarize, literal deletion works as follows, where C denotes the conjunction of all ground algebraic clauses known to the prover.

1. Identify a candidate literal L in a clause by writing the clause as $A \vee L \vee B$. Note that L must be algebraic.
2. Form the existential closure of the formula $\neg A \wedge \neg B \wedge C \wedge L$, retaining only the algebraic literals of A and B.
3. If RCF quantifier elimination reduces this formula to **false**, then delete L from the clause.

The formulas given to RCF contain no universal quantifiers, because at present they are constructed from ground clauses. As mentioned above, every uninterpreted constant contained in the formula becomes an existentially quantified variable.

3.3 Axioms

We have sought to find a general set of axioms describing the real numbers, and in particular their ordering. We combine these general axioms with upper

and lower bounds for the functions of interest. The resulting axiom set replaces inequalities concerning those functions by algebraic inequalities. These, in turn, will be simplified by the RCF solver.

We include axioms that relate division to multiplication, since RCF solvers do not accept division.

$$\begin{aligned} X \le Y \times Z &\implies X/Z \le Y \vee Z \le 0 \\ X \le Y/Z &\implies X \times Z \le Y \vee Z \le 0 \\ X \times Z \le Y &\implies X \le Y/Z \vee Z \le 0 \\ X/Z \le Y &\implies X \le Y \times Z \vee Z \le 0 \end{aligned}$$

We include similar axioms for equality.

$$\begin{aligned} Y/Z = X &\implies Z = 0 \vee Y = X \times Z \\ X = Y/Z &\implies Z = 0 \vee X \times Z = Y \end{aligned}$$

These axioms simplify inequalities between quotients.

$$\begin{aligned} X/Y \le W/Z &\implies Y \le 0 \vee Z \le 0 \vee X \times Z \le Y \times W \\ Y \times W \le X \times Z &\implies W/Z \le X/Y \vee Y \le 0 \vee Z \le 0 \end{aligned}$$

Our use of a canonical polynomial representation eliminates the need for the usual axioms for addition and multiplication, such as commutative laws.

We have experimented with axioms defining the standard properties of a linear ordering:

$$X \le X \tag{2}$$

$$X \le Y \vee Y \le X \tag{3}$$

$$X \le Y \wedge Y \le X \implies X = Y \tag{4}$$

$$X \le Y \wedge Y \le Z \implies X \le Z \tag{5}$$

Note that inequalities are formalized using $\le$. We formalize $<$ by the equivalence $X < Y \iff \neg(Y \le X)$. This eliminates the need to have, for example, four versions of transitivity.

$$\begin{aligned} \neg(X < Y) \vee \neg(Y \le X) \\ (X < Y) \vee (Y \le X) \end{aligned}$$

However, the experiments reported below do not use axioms (2)–(5). Transitivity, in particular, blows up the search space. When transitivity is omitted, the lower and upper bound axioms must be modified in the obvious way; for example, the axiom $\phi \implies \ln X \le e$ must become $\phi \wedge e \le Y \implies \ln X \le Y$. We intend to do more work to find the best treatment of ordering properties.

These axioms are rather general. We can influence the way they are applied by means of *weights*, which influence the selection of literals in ordered resolution. Giving high weights (500 000) to the functions ln and exp encourages the prover

to eliminate them. We also give division a high weight (50), encouraging its replacement by multiplication. It is obvious that occurrences of certain functions must be discouraged, but the effect of adding weights was more powerful than we expected.

4 Worked Example

In order to see how this approach works, let us follow its proof of the formula

$$\forall X \left[-1/2 \le X \wedge X \le 3 \Longrightarrow \ln(1+X) \le X\right].$$

Below, for the sake of readability, we write

$$L_1 \wedge \ldots \wedge L_n \implies \textbf{false}$$

rather than

$$\neg L_1 \vee \ldots \vee \neg L_n.$$

We also use standard mathematical notation rather than Horner canonical form. The input file appears as Appendix A.

After negation and Skolemization, our problem consists of three conjecture clauses:

$$-1/2 \le u$$
$$u \le 3$$
$$\neg(\ln(1+u) \le u)$$

The first conjecture clause resolves with one of the divisibility axioms and yields

$$-1 \le u \times 2 \vee 2 \le 0,$$

which simplifies to $0 \le 1 + u \times 2$.

The high weight of ln will ensure that literals containing it are selected. Therefore, the negative literal above will combine with complementary literals in the axiom clauses specifying upper bounds of $\ln x$: that is, those shown in Fig. 1. One of these (combined with transitivity as described in §3.3) is

$$1 \le X \wedge X \le 2 \wedge X - 1 \le Y \Longrightarrow \ln X \le Y.$$

Resolution of the third conjecture clause with this axiom yields

$$1 \le 1+u \wedge 1+u \le 2 \wedge 1+u-1 \le u \implies \textbf{false} \tag{6}$$

Performing the obvious simplifications, we get

$$0 \le u \wedge u \le 1 \implies \textbf{false}.$$

We have now deduced $u < 0 \vee u > 1$.

Another upper bound axiom (combined with transitivity) is

$$2 \leq X \wedge X \leq 4 \wedge \frac{X}{2} \leq Y \Longrightarrow \ln X \leq Y$$

when resolution with the third conjecture clause yields

$$2 \leq 1 + u \wedge 1 + u \leq 4 \wedge \frac{1+u}{2} \leq u \implies \textbf{false}.$$

The simplified clause (RCF deletes $u \leq 3$) is

$$1 \leq u \wedge \frac{1+u}{2} \leq u \implies \textbf{false}. \tag{7}$$

Resolution of this with the appropriate division axiom produces

$$1 \leq u \wedge 1 + u \leq u \times 2 \implies 2 \leq 0,$$

which simplifies to

$$1 \leq u \implies \textbf{false},$$

so we have deduced $u < 1$. Indeed (since $u < 0 \vee u > 1$) we have $u < 0$, and RCF will notice this.

Resolution of the third conjecture clause with the third upper bound axiom and the division axiom yields

$$1/2 \leq 1 + u \wedge 1 + u \leq 1 \wedge 3(1+u)^2 - 4(1+u) + 1 \leq 2u(1+u)^2 \implies 2u^2 \leq 0.$$

The obvious simplifications yield

$$-1/2 \leq u \wedge u \leq 0 \wedge 0 \leq u^2 + u^3 \implies 2u^2 \leq 0,$$

but given the facts $u < 0$ and $0 \leq 1 + u \times 2$, RCF further simplifies it to

false.

As mentioned earlier, this proof generates the following series of algebraic clauses, from right to left.

```
F, ~(1 <= u), ~(0 <= u) \/ ~(u <= 1), 0 <= 1 + u * 2, u <= 3
```

5 Results and Discussion

We have run only a few dozen examples, but the results are promising (Table 1). We are aware of no other system that can solve such problems, so we present the table merely to give an impression of what can be solved, rather than as a basis for comparison. Most of these problems are proved in under three seconds on a 3GHz Pentium D.

Table 1. Problems and Runtimes

problem	seconds
$1/2 \le x \le 4 \implies \ln x \le x - 1$	0.608
$1 \le x \le 4 \implies \ln x \le x^2 - x$	0.060
$1/2 \le x \le 3/2 \implies \ln x \le 2x^2 - 3x + 1$	0.643
$1/2 \le x \le 1 \implies \ln x \le (3 - 3x)/2$	0.779
$-1/2 \le x \le 3 \implies \ln(1 + x) \le x$	0.527
$0 \le x \le 3 \implies \ln(1 + x) \le x + x^2$	0.060
$-1/2 \le x \le 1/2 \implies \ln(1 + x) \le x + 2x^2$	2.500
$-1/2 \le x \le 0 \implies \ln(1 + x) \le (-3x)/2$	2.457
$-3 \le x \le 1/2 \implies \ln(1 - x) \le -x$	1.929
$-3 \le x \le 0 \implies \ln(1 - x) \le x^2 - x$	0.219
$-1/2 \le x \le 1/2 \implies \ln(1 - x) \le 2x^2 - x$	4.256
$0 \le x \le 1/2 \implies \ln(1 - x) \le (3x)/2$	3.303
$1/2 \le x \le 4 \implies (x - 1)/x \le \ln x$	0.272
$1 \le x \le 4 \implies -x^2 + 3x - 2 \le \ln x$	0.305
$1/2 \le x \le 3/2 \implies -2x^2 + 5x - 3 \le \ln x$	0.258
$-1/2 \le x \le 3 \implies x/(1 + x) \le \ln(1 + x)$	2.592
$0 \le x \le 3 \implies x - x^2 \le \ln(1 + x)$	0.723
$-1/2 \le x \le 1/2 \implies x - 2x^2 \le \ln(1 + x)$	1.643
$-3 \le x \le 1/2 \implies -x/(1 - x) \le \ln(1 - x)$	0.836
$-3 \le x \le 0 \implies -x - x^2 \le \ln(1 - x)$	0.779
$-1/2 \le x \le 1/2 \implies -x - 2x^2 \le \ln(1 - x)$	1.133
$-1 \le x \le 0 \implies \exp x \le (2 + x)/2$	0.307
$-1 \le x \le 0 \implies \exp x \le (4 + x)/4$	0.363
$0 \le x \le 1 \implies \exp x \le 1 + x + x^2$	0.781
$-1 \le x < 1 \implies \exp x \le 1/(1 - x)$	0.800
$0 \le x \le 1 \implies \exp(-x) \le (2 - x)/2$	0.319
$-1 < x \le 1 \implies \exp(-x) \le 1/(1 + x)$	0.614
$-1 \le x \le 1 \implies 1 + x \le \exp x$	0.611
$0 \le x \le 1 \implies (4 + x)/4 \le \exp x$	0.926
$-1 \le x \le 0 \implies (4 + 7x)/4 \le \exp x$	0.613
$-1 \le x \le 1 \implies 1 - x \le \exp(-x)$	0.993

We clearly need to broaden our range of problems. Our examples all take the same form: that a basic inequality holds over a specific interval. The potential strength of a combination of resolution and RCF is that we might be able to solve such problems when they occur indirectly buried in some more complicated goals, perhaps resulting from the unification of other variables.

Limitations of our approach cause it to fail on some problems. Our bounds shown in Figs. 1 and 2 are sometimes too loose. For example, to prove $0 \le x \le 1/2 \implies -3x/2 \le \ln(1 - x)$, we have to use a tighter logarithmic lower bound:

$$\frac{11x^3 - 18x^2 + 9x - 2}{6x^3} \le \ln x \qquad \left(\frac{1}{2} \le x \le 1\right)$$

Similarly, proving $0 \le x \le 1 \implies \exp x \le (4+7x)/4$ requires using a tighter exponential upper bound:

$$\exp x \le 120/(-x^5 + 5x^4 - 20x^3 + 60x^2 - 120x + 120) \qquad \Big(0 \le x \le 1\Big)$$

Some problems cannot be solved because the RCF decision procedure runs forever. One example is $0 \le x \le 1 \implies \exp(x - x^2) \le 1 + x$, which calls RCF on the following existentially quantified formula:

```
exists u. 0 <= u /\ u <= 1 /\
          ~ (u * u * u * u * (1 + u * u * 2) <=
             u * u * (3 + u * (2 + u * u * (3 + u * u))))
```

Introducing new variables allows some problems to be solved, but increases the danger that the decision procedure will loop. The problem $-1 < x \implies \exp(x/(1+x)) \le 1+x$ is easily proved if we modify it, replacing the quotient by an extra variable y such that $(1+x)y = x$. As a second example, the univariate problem $-1/2 \le x \le 0 \implies x/\sqrt{1+x} \le \ln(1+x)$ is first converted to a problem with two variables to avoid the square root: $-1/2 \le x \le 0 \wedge 0 \le y \wedge y^2 = 1+x \implies x/y \le \ln(1+x)$. Attempting the new problem will generate the following formula, which RCF cannot handle:

```
exists u v. 0 <= 1 + u * 2 /\ u <= 0 /\ 0 <= v /\ 1 + u = v * v /\
            ~ (u * ((2 + v * 2) + (u * ((4 + v * 3)+ u * 2))) <= 0)
```

These examples are too hard for a simple algorithm like Cohen-Hörmander. Eliminating two quantifiers from formulas involving nonlinear polynomials is not trivial. The first example is more marginal, but the doubly exponential complexity of Hörmander's algorithm seems to become noticeable when the degree of the polynomial exceeds 5. These examples suggest that we use a more powerful procedure, such as QEPCAD-B [7].

6 Conclusions

Inequalities concerning the elementary functions, such as exp and ln, can be proved by a simple combination of a resolution theorem prover and an RCF decision procedure. The architecture is simple and principled: it merely involves modifying resolution's simplification phase to take account of the RCF theory.

The axiom system requires further development and testing. It could contain a greater variety of upper and lower bounds. For example, the very loose bound $\ln x \le x - 1$ might be useful when x can be arbitrarily large. Use of bounds such as $\ln x \le 2(\sqrt{x} - 1)$ requires a means of eliminating the square root operator, through a translation such as $\exists y\, [y^2 = x \to \cdots]$. Most of the bounds are infinite families of axioms, so we must develop a preprocessing phase that inserts required instances of these axioms into the problem automatically. The general ordering axioms, such as transitivity, greatly expand the search space; we need to explore other methods of handling ordering properties.

Also, we need to consider a wider range of problems, including difficult features such as nested applications of elementary functions or sums and products of them. We need to consider equalities as well as inequalities. Introducing new variables to eliminate roots and quotients can cause the RCF procedure to run forever, so we intend to try using QEPCAD-B [7] instead.

Acknowledgements. The research was funded by the EPSRC grant EP/C013409/1, *Beyond Linear Arithmetic: Automatic Proof Procedures for the Reals*. John Harrison contributed his code for the RCF decision procedure, which Amine Chaieb helped to port from Ocaml to Standard ML. Joe Hurd offered much help with his Metis prover. Christoph Benzmüller and the referees commented on this paper.

References

1. Behzad Akbarpour and Lawrence C. Paulson. Towards automatic proofs of inequalities involving elementary functions. In Byron Cook and Roberto Sebastiani, editors, *PDPAR: Pragmatics of Decision Procedures in Automated Reasoning*, pages 27–37, 2006.
2. Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic*, in press.
3. Jeremy Avigad and Harvey Friedman. Combining decision procedures for the reals. *Logical Methods in Computer Science*, 2(4), 2006.
4. Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.
5. J. Barwise. An introduction to first-order logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 5–46. North-Holland, 1977.
6. Michael Beeson. Automatic generation of a proof of the irrationality of e. *JSC*, 32(4):333–349, 2001.
7. Christopher W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *SIGSAM Bulletin*, 37(4):97–108, 2003.
8. Edmund Clarke and Xudong Zhao. Analytica: A theorem prover for Mathematica. *Mathematica Journal*, 3(1):56–71, 1993.
9. Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real quantifier elimination in practice. Technical Report MIP-9720, Universität Passau, D-94030, Germany, 1997.
10. Cormac Flanagan, Rajeev Joshi, Xinming Ou, and James B. Saxe. Theorem proving using lazy proof explication. In Warren A. Hunt and Fabio Somenzi, editors, *Computer Aided Verification: 15th International Conference, CAV 2003*, LNCS 2725, pages 355–367. Springer, 2003.
11. Benjamin Grégoire and Assia Mahboubi. Proving equalities in a commutative ring done right in coq. In Joe Hurd and Tom Melham, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2005*, LNCS 3603, pages 98–113. Springer, 2005.
12. Lars Hörmander. *The Analysis of Linear Partial Differential Operators II: Differential Operators with Constant Coefficient*. Springer, 2006. First published in 1983; cited by Mclaughlin and Harrison [15].

13. Joe Hurd. Metis first order prover. http://gilith.com/software/metis/, 2007.
14. William McCune and Larry Wos. Otter: The CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.
15. Sean McLaughlin and John Harrison. A proof-producing decision procedure for real arithmetic. In Robert Nieuwenhuis, editor, *Automated Deduction — CADE-20 International Conference*, LNAI 3632, pages 295–314. Springer, 2005.
16. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. LNCS Tutorial 2283.
17. Virgile Prevosto and Uwe Waldmann. SPASS+T. In Geoff Sutcliffe, Renate Schmidt, and Schulz Schulz, editors, *FLoC'06 Workshop on Empirically Successful Computerized Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 18–33, 2006.

A Input File for the Sample Problem

```
cnf(leq_left_divide_mul,axiom,
    ( ~ less_equal(X,multiply(Y,Z))
    | less_equal(divide(X,Z),Y)
    | less_equal(Z,0) )).

cnf(leq_left_mul_divide,axiom,
    ( ~ less_equal(X,divide(Y,Z))
    | less_equal(multiply(X,Z),Y)
    | less_equal(Z,0) )).

cnf(leq_right_divide_mul,axiom,
    ( ~ less_equal(multiply(X,Z),Y)
    | less_equal(X,divide(Y,Z))
    | less_equal(Z,0) )).

cnf(leq_right_mul_divide,axiom,
    ( ~ less_equal(divide(X,Z),Y)
    | less_equal(X,multiply(Y,Z))
    | less_equal(Z,0) )).

cnf(eq_left_divide_mul,axiom,
    ( ~ equal(divide(Y,Z),X)
    | equal(Z,0)
    | equal(Y,multiply(X,Z)) )).

cnf(eq_right_divide_mul,axiom,
    ( ~ equal(X,divide(Y,Z))
    | equal(Z,0)
    | equal(multiply(X,Z),Y) )).

cnf(leq_double_divide_mul,axiom,
    ( ~ less_equal(divide(X,Y),divide(W,Z))
    | less_equal(Y,0)
    | less_equal(Z,0)
```

```
    | less_equal(multiply(X,Z),multiply(Y,W)) )).

cnf(leq_double_mul_divide,axiom,
    ( ~ less_equal(multiply(Y,W),multiply(X,Z))
    | less_equal(divide(W,Z),divide(X,Y))
    | less_equal(Y,0)
    | less_equal(Z,0) )).

cnf(log_upper_bound_case_1,axiom,
    ( ~ less_equal(divide(1,2),X)
    | ~ less_equal(X,1)
    | ~ less_equal(divide(add(multiply(3,power(X,2)),
                  add(neg(multiply(4,X)),1)),multiply(2,power(X,2))),Y)
    | less_equal(ln(X),Y) )).

cnf(log_upper_bound_case_2,axiom,
    ( ~ less_equal(1,X)
    | ~ less_equal(X,2)
    | ~ less_equal(subtract(X,1),Y)
    | less_equal(ln(X),Y) )).

cnf(log_upper_bound_case_3,axiom,
    ( ~ less_equal(2,X)
    | ~ less_equal(X,4)
    | ~ less_equal(divide(X,2),Y)
    | less_equal(ln(X),Y) )).

cnf(log_upper_bound_problem_5_1,negated_conjecture,
    (less_equal(divide(neg(1),2),u) )).

cnf(log_upper_bound_problem_5_2,negated_conjecture,
    (less_equal(u,3) )).

cnf(log_upper_bound_problem_5_3,negated_conjecture,
    (~ less_equal(ln(add(1,u)),u) )).
```