MetiTarski: An Automatic Theorem Prover for Real-Valued Special Functions

Behzad Akbarpour and Lawrence C. Paulson

Abstract Many theorems involving special functions such as ln, exp and sin can be proved automatically by MetiTarski: a resolution theorem prover modified to call a decision procedure for the theory of real closed fields. Special functions are approximated by upper and lower bounds, which are typically rational functions derived from Taylor or continued fraction expansions. The decision procedure simplifies clauses by deleting literals that are inconsistent with other algebraic facts. MetiTarski simplifies arithmetic expressions by conversion to a recursive representation, followed by flattening of nested quotients. Applications include verifying hybrid and control systems.

1 Introduction

Many branches of mathematics, engineering and science require reasoning about *special functions*: logarithms, sines, cosines and dozens of others. Few techniques are known for automatically proving statements involving such functions. We have implemented a theorem prover that works by eliminating special functions, substituting rational function upper or lower bounds, transforming parts of the problem into polynomial inequalities, and finally applying a decision procedure for the theory of real closed fields.

The theory of *real closed fields* (RCF) concerns equalities and inequalities involving addition, subtraction and multiplication. (We call logical formulas in this theory *algebraic*.) A field F is *real closed* if every positive number has a square root in F and every odd degree univariate polynomial with coefficients in F has a root in F. The decision procedure works by eliminating quantifiers from the supplied formula; for example, $\exists x.\, ax^2 + bx + c = 0$ reduces to

$$(a \neq 0 \wedge b^2 - 4ac \geq 0) \vee (a = 0 \wedge b \neq 0) \vee (a = b = c = 0).$$

Both universal and existential quantifiers can be eliminated, but our current experiments only require the decision procedure to refute purely existential formulas.

Tarski proved the decidability of RCF in the 1930s, but his procedure was impractical [19]. McLaughlin and Harrison [34] recently implemented a more efficient

Computer Laboratory, University of Cambridge, England
E-mail: {ba265,lp15}@cl.cam.ac.uk

procedure credited to Hörmander [27] and Cohen. We used it in earlier work [2, 4], but unfortunately it fails to terminate if applied to a polynomial of degree greater than six or so. QEPCAD-B [11, 26] is an advanced implementation of *cylindrical algebraic decomposition* (CAD), which is the best available decision procedure for the complete theory of RCF [19]. CAD is still doubly exponential in the number of variables, but it is polynomial in other parameters such as size of the input formula, the maximum degree of polynomials, the maximum coefficient length and so forth [11]. In our experience, QEPCAD usually returns quickly if the formula has only a few variables. We run it as a separate process.

Our approach [2–4] to proving inequalities involving special functions is to replace function occurrences one by one with appropriate upper or lower bounds. Once we have also eliminated occurrences of division, we can call QEPCAD and if it is successful, simplify the problem. Daumas et al. [17] present families of upper and lower bounds for square roots, trigonometric functions, logarithms and exponentials; in fact, virtually all functions of interest to engineering can be approximated by a power series or a continued fraction [14]. Each approximation is typically an upper or lower bound of the desired function on some good-sized interval. A small modification, such as including the next term of the power series, frequently transforms a lower bound into an upper bound or vice versa. We can use a variety of approximations in order to obtain coverage over wider intervals, in many cases infinite intervals.

Our approach requires a full first-order theorem prover even to prove simple inequalities. The bounds typically have side conditions that must be proved. Case analysis is necessary when eliminating division and often when substituting bounds, for example when combining intervals. We chose to modify a resolution theorem prover rather than implementing a theorem prover from scratch. Impressive examples of the latter approach include Analytica [13] and Weierstrass [9], both of which implement a form of sequent calculus. However, we felt that writing an entire prover would require more effort than modifying a resolution prover, while delivering inferior results. We were also inspired by SPASS+T [38], which effectively combines the resolution theorem prover SPASS with various SMT solvers. For the resolution prover, we chose Hurd's Metis [28]. Compared with leading provers, it is slow (being coded in Standard ML rather than C) and it lacks many refinements (such as advanced data structures for indexing). However, it implements the superposition calculus [7] and its code is extremely clear.

MetiTarski outputs proofs in the standard TSTP format [43]. These are machine-readable and can also, with perseverance, be checked by humans. Most proof steps involve standard resolution inference rules. To these we add specialist inference rules for arithmetic simplification, decision procedure calls and other steps. It should be straightforward to build an independent tool for checking MetiTarski proofs.

Paper outline. We begin with a general architectural overview (§2). We then discuss (§3) the upper and lower bounds we use, and other aspects of the axiom system. We proceed to describe (§4) how we modified the resolution prover Metis. We finally present a table of new results (§5) along with brief conclusions (§6).

2 Overview of MetiTarski

We work in first-order logic with equality. Detailed technical definitions can be found in standard reference works [7]. The following summary highlights the specific points that pertain to MetiTarski.

2.1 Definitions

Our universe of discourse is the set of real numbers. All variables range over the reals. A *term* is a variable, a constant or an n-ary function applied to an n-tuple of terms. Our language includes constants for the integers and the arithmetic functions $+$, $-$, $\times$ and $/$. Below we use familiar mathematical notation, for example writing xy or $x \cdot y$ instead of $x \times y$.

MetiTarski simplifies arithmetic expressions based on the assumptions outlined above. We are particularly interested in real-valued functions such as sin and cos, and their properties must be defined axiomatically. For each function of interest, axioms must be provided that express upper or lower bounds, or properties such as monotonicity. Needless to say, invalid axioms will yield invalid proofs.

An *atomic formula* P, Q, ... has the form $t = u$ or $t \leq u$, where t and u are terms.

A *literal* is an atomic formula or its negation. We regard $t < u$ as abbreviating the literal $\neg(u \leq t)$ and accept the familiar abbreviations $t \neq u$, $t \geq u$ and $t > u$.

A *clause* is a finite set of literals, interpreted as a disjunction. We typically write clauses as logical formulas such as $\neg P_1 \vee \neg P_2 \vee Q_1 \vee Q_2$ or $P_1 \wedge P_2 \rightarrow Q_1 \vee Q_2$ instead of sets such as $\{\neg P_1, \neg P_2, Q_1, Q_2\}$. The empty clause denotes the formula $\bot$, contradiction. A set of clauses is interpreted as a conjunction.

A *ground* term, literal or clause is one containing no variables.

2.2 The Resolution Loop

A resolution prover [7] represents and works on a problem as a set of clauses, which can be seen as a formula in *conjunctive normal form*. The conjecture is typically supplied as a first-order formula; it is negated and conjoined with axioms appropriate to the problem domain. The resolution procedure attempts to deduce the empty clause, thereby proving the original conjecture by contradiction.

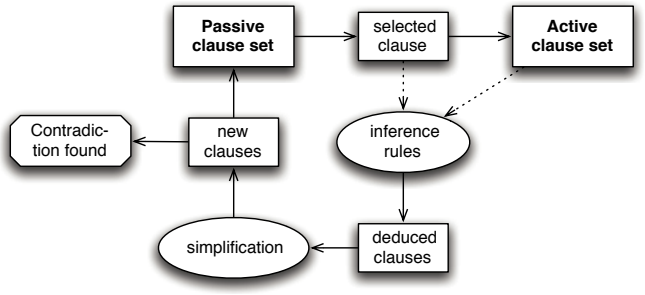

Fig. 1 The main loop of resolution

Each resolution inference combines two clauses and yields a new clause as follows:

$$\frac{P \vee A \quad \neg Q \vee B}{(A \vee B)\sigma}$$

Here P and Q are atomic formulas, A and B are sets of literals, and σ is the most general unifier of P and Q. In general, the new clause will be longer than the original ones, and most such steps are fruitless. The key to successful theorem proving is to choose the right literals, P and $\neg Q$, to resolve upon. Resolution provers use a *term ordering* to select the most appropriate literal; with the right ordering (since resolution deletes the selected literals), this process gradually eliminates all occurrences of special functions. We assist resolution by allowing a decision procedure (QEPCAD) to delete ground algebraic literals that it determines to be inconsistent with their context (such as adjacent literals).

A resolution prover's main loop (Fig. 1) manages two sets of clauses, *Active* and *Passive* [33]. The Active set consists of clauses that have been resolved with every other Active clause, while the Passive set consists of clauses waiting to be processed. At the start, all clauses belong to Passive. At each iteration, the following 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. Those that are not redundant are added to the Passive set.

MetiTarski modifies this procedure in several respects. The simplification phase converts arithmetic formulas to a canonical form and attempts to isolate special functions. It can also delete ground algebraic literals that it determines to be inconsistent with their context. The built-in term ordering is modified to ensure that special function occurrences are eliminated despite the existence of apparently more complex algebraic formulas. Clauses that are obviously trivial, such as $t \leq 0 \vee t \geq 0$, are automatically discarded. Every aspect of this architecture has been designed to ensure good performance for an axiom system of a specific form. Despite the modifications outlined above, MetiTarski remains a resolution theorem prover, with a control flow as shown in Fig. 1.

2.3 On Case Splitting

Newcomers to resolution may be surprised to learn that the procedure works by saturation rather than by posing subgoals or by case splitting on variables. It processes a single pool of assertions until it succeeds by detecting a contradiction or fails by running out of clauses to process (or running out of memory). To prove a formula of the form $P \vee Q \rightarrow R$, a subgoaling approach would attempt to prove two separate problems, $P \rightarrow R$ and $Q \rightarrow R$. Resolution expresses the problem $P \vee Q \rightarrow R$ as a pair of clauses derived from its negation: $P \vee Q$ and $\neg R$. Rather than proving the formula $P \rightarrow R$, resolution may succeed (with the help of other axioms present in the problem) in deducing Q from $P \vee Q$ and $\neg R$. It could complete the proof by finding a contradiction between Q and $\neg R$. The two cases are actually proved sequentially. Because every clause having two or more literals is a disjunction, every resolution proof can be regarded as consisting of a series of case analyses.

3 Axiom System

We distribute MetiTarski with several axiom files that users can insert into problems. MetiTarski requires axioms that specify the properties of the $\leq$ relation. Few of these

axioms are general; they typically concern the special functions or division. The most important axioms provide bounds for special functions. These bounds must be correct, accurate and not too complicated. They also need to be valid over a good-sized interval. We have sought reasonably accurate bounds for the purposes of our experiments, but these can always be improved upon.

In our earlier work [2–4], we relied almost exclusively on Daumas et al. [17], who provide bounds for a few well-known functions. Those bounds, however, were intended for a different application: to decide constant formulas like $e > 2\sqrt{2}$ using interval arithmetic. For each function, they supplied a family of increasingly accurate bounds. Each bound included *range reduction*: scaling to ensure accuracy for function arguments of arbitrary magnitude. Their software selected appropriate parameters for the problem at hand. In effect, each bound was an infinite family indexed in two dimensions (accuracy and range). Resolution provers require a finite and preferably small axiom system.

We have addressed these difficulties in a variety of ways. We have moved away from Taylor expansions, which tend to be accurate only on narrow intervals and then veer away wildly, in favour of continued fractions. For some functions, in particular logarithm and arctan, the continued fraction approximation gives excellent accuracy over wide intervals. We have eliminated arbitrary range reduction and choosen a few fixed ranges of accuracy. These simplifications are adequate for our experiments, allowing us to focus on crucial issues such as the search space and the treatment of complex expressions. The original bounds were only claimed [17] to hold over narrow intervals; these could often be relaxed. In other cases, we sought new bounds that were valid over wider intervals. Relaxing the range restrictions allows inequalities to be proved over infinite intervals. The resolution procedure can perform case analyses (in the sense of Sect. 2.3) in order to join proofs involving bounds valid over different intervals, but it can only consider finitely many cases.

Our problems demand a wide range of accuracies. Those of mathematical origin sometimes require razor-sharp bounds while some derived from real-world problems [42] can be solved with crude bounds. The most accurate bounds are only necessary for a few problems, so we keep them in separate axiom files because the presence of many bounds for a particular function can greatly increase the search space. One of our continued fraction bounds for the exponential function is accurate to 1.06×10^{-12} on the interval $[-2, 0]$, according to the computer algebra system Maple.

The continued fraction expansions come from a standard reference book [14]. In order to avoid introducing specialised notation, we do not present the continued fractions but instead the specific instances ("approximants") that we require. We have computed these approximants using Maple and library code obtained from the book's associated website [8]. We also omit the proofs that these approximants are upper or lower bounds for the functions they approximate. To present those proofs would require us to develop the theory of continued fractions at length. Continued fractions are not essential to MetiTarski: our previous paper [3] does not use them at all.

Remark: we use $\overline{f}(x)$ and $\underline{f}(x)$ to stand for upper or lower bounds of $f(x)$ often in the absence of specific definitions; when we write say $\overline{\ln}(x)$, we often refer to a different function than do Daumas et al. [17].

3.1 The Square Root Function

In early work, we manually replaced square roots by new variables, replacing $\sqrt{t}$ by y such that $y \geq 0$ and $y^2 = t$. Provided the term t contains no special functions, we obtain algebraic constraints that QEPCAD can accept. However, this approach can only be applied to algebraic terms, and increasing the number of variables is inadvisable when the decision procedure is doubly exponential in that number. MetiTarski can now support the square root function directly, by means of upper and lower bounds.

Daumas et al. [17] base their bounds for $\sqrt{x}$ on Newton's method. We have improved their accuracy, making them exact when $x = 1$. Our versions are defined as follows:

$$\begin{aligned}
\overline{\mathrm{sqrt}}(x, 0) &= \frac{x+1}{2} \\
\overline{\mathrm{sqrt}}(x, n) &= \frac{y + x/y}{2} \quad n \geq 1, \text{where } y = \overline{\mathrm{sqrt}}(x, n-1) \\
\underline{\mathrm{sqrt}}(x, n) &= \frac{x}{\overline{\mathrm{sqrt}}(x, n)}
\end{aligned}$$

Their complexity increases rapidly. For example,

$$\overline{\mathrm{sqrt}}(x, 2) = \frac{x^4 + 28x^3 + 70x^2 + 28x + 1}{8(x+1)(x^2+6x+1)}$$

$$\overline{\mathrm{sqrt}}(x, 3) = \frac{x^8 + 120x^7 + 1820x^6 + 8008x^5 + +12870x^4 + 8008x^3 + 1820x^2 + 120x + 1}{16(x+1)(x^2+6x+1)(x^4+28x^3+70x^2+28x+1)}$$

We supply MetiTarski with axioms that assert

$$\underline{\mathrm{sqrt}}(x, n) \leq \sqrt{x} \leq \overline{\mathrm{sqrt}}(x, n)$$

subject to the condition that $x \geq 0$. As remarked above, we must choose a finite number of these. If we plot these functions (Fig. 2), the graphs suggest that the versions with higher values of n are more accurate everywhere, and so the best approach should be to choose one reasonably large n. Experiments demonstrate however that proofs are frequently found faster if versions with lower values of n are also present, presumably because the formulas are simpler. We therefore include instances of the axioms for $n = 0, \ldots, 4$, a total of 10 axioms.

Their correctness is easy to demonstrate. Below, let n denote a non-negative integer.

Lemma 1 *If $x \geq 0$ then $\overline{\mathrm{sqrt}}(x, n) > 0$.*

Proof. Immediate, by induction on n. □

Proposition 1 *If $x \geq 0$ then $\overline{\mathrm{sqrt}}(x, n) \geq \sqrt{x}$.*

Proof. By the previous lemma, it suffices to show $\left(\overline{\mathrm{sqrt}}(x, n)\right)^2 \geq x$. Regardless of whether n is zero or nonzero, $\overline{\mathrm{sqrt}}(x, n)$ can be written in the form $\frac{1}{2}(y + x/y)$ for some $y > 0$. A simple calculation reveals that

$$\left(\frac{y + x/y}{2}\right)^2 - x = \frac{y^2}{4} + \frac{x}{2} + \frac{x^2}{4y^2} - x = \frac{y^2}{4} - \frac{x}{2} + \frac{x^2}{4y^2} = \left(\frac{y - x/y}{2}\right)^2 \geq 0,$$

from which we conclude $\left(\frac{1}{2}(y + x/y)\right)^2 \geq x$. □

Proposition 2 *If $x \geq 0$ then $\underline{\mathrm{sqrt}}(x, n) \leq \sqrt{x}$.*

Proof. Immediate, by the previous two propositions. □

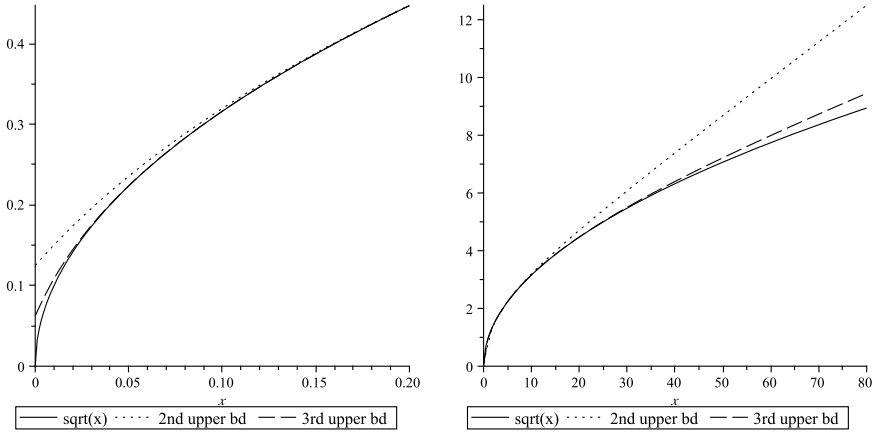

Fig. 2 Square Root Upper Bounds: $\overline{\mathrm{sqrt}}(x, 2)$, $\overline{\mathrm{sqrt}}(x, 3)$

3.2 The Logarithm Function

Daumas et al. [17] derive bounds for $\ln x$ from Taylor approximations,

$$\sum_{i=1}^{n} (-1)^{i+1} \frac{(x-1)^i}{i},$$

for the range $1 < x \leq 2$. With this series, even values of n yield lower bounds while odd values of n yield upper bounds. Unfortunately, these approximations become wildly inaccurate as x increases because their leading term involves x^n.

Our upper bounds are finite approximants of the continued fraction expansion (11.2.1) of Cuyt et al. [14, p. 196]. Odd approximants yield upper bounds for $\ln x$ for $x > 0$. Lower bounds are obtained by the identity $\ln x = -\ln(1/x)$: we can define $\underline{\ln}(x) = -\overline{\ln}(1/x)$ because the change of sign reverses the inequality. Here are the first four pairs of bounds:

$$\begin{aligned}
\frac{x-1}{x} &\leq \ln x \leq x - 1 \\
\frac{(1+5x)(x-1)}{2x(2+x)} &\leq \ln x \leq \frac{(x+5)(x-1)}{2(2x+1)} \\
\frac{(1+19x+10x^2)(x-1)}{3x(3+6x+x^2)} &\leq \ln x \leq \frac{(x^2+19x+10)(x-1)}{3(3x^2+6x+1)} \\
\frac{(47x^3+239x^2+131x+3)(x-1)}{12x(x^3+12x^2+18x+4)} &\leq \ln x \leq \frac{(3x^3+131x^2+239x+47)(x-1)}{12(4x^3+18x^2+12x+1)}
\end{aligned}$$

They are valid for $x > 0$ and are reasonably accurate: Fig. 3 portrays the second and third bounds shown above for the intervals $(0, 1]$ and $[1, 8]$. Their correctness can be proved by a straightforward argument, appealing to general theorems concerning the monotonicity properties of continued fraction tails [15].

The superiority of continued fractions over Taylor series is evident in Fig. 4. It compares one of our simplest upper bounds with a Taylor formula, which is inferior near zero and from $x > 2$ zooms into the stratosphere (its limiting value is $x^5/5$).

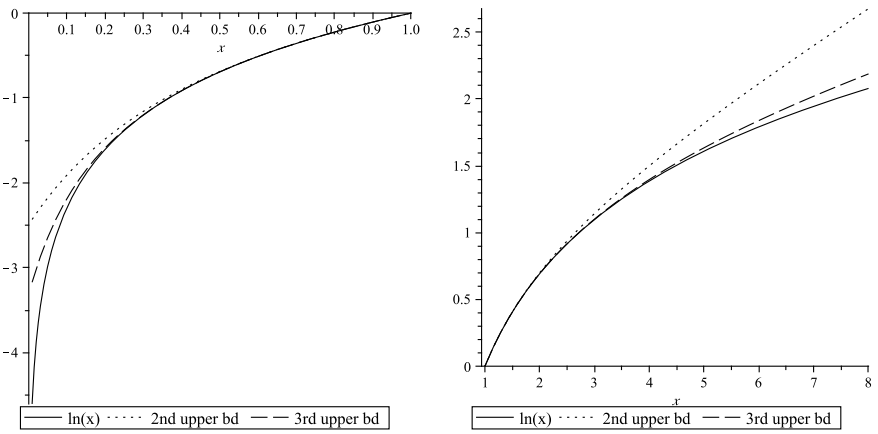

Fig. 3 Logarithm Upper Bounds: $\overline{\ln}(x,2)$, $\overline{\ln}(x,3)$

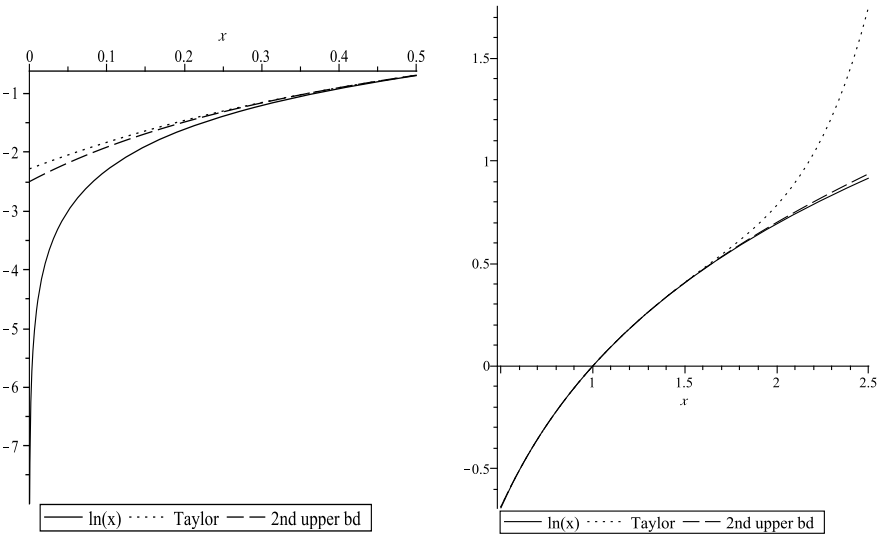

Fig. 4 Logarithm Upper Bounds: $(x-1)(12x^4-63x^3+137x^2-163x+137)/60$, $\overline{\ln}(x,2)$

3.3 The Exponential Function

Daumas et al. [17] derive bounds for $\exp x$ from its Taylor expansion, but only for $-1 \le x < 0$. They use a complicated system of transformations, first covering the negative numbers in separate intervals of the form $[k-1,k)$ for integer $k < 0$. For $x > 0$, they use the identity

$$\exp(-x) = \frac{1}{\exp x.} \tag{1}$$

The rapid growth of the exponential function necessitates the use range reduction to scale down large arguments. The variety of scaling possibilities yields a multiplicity

of bounds, but we can use only finitely many. Nevertheless, we have managed to find simpler bounds valid over wide ranges. We complement these Taylor series bounds with others based on continued fractions, specifically expansion (11.1.2) of Cuyt et al. [14, p. 194].

We use a crucial fact about the Taylor expansion [12, p. 83].

Proposition 3 *If n is odd and $x \neq 0$ then*

$$\exp x > \sum_{i=0}^{n} \frac{x^i}{i!}.$$

If n is even then this inequality holds if $x > 0$, while the opposite inequality holds if $x < 0$. Obviously we have equality when $x = 0$.

This opposite inequality yields upper bounds for $x \leq 0$. The bound using $n = 4$ is already poor when $x < -2$, but they are valid for all $x \leq 0$. Prop. 3 with odd n yields lower bounds for $x \geq 0$. Using (1), we define $\overline{\exp}(-x) = 1/\underline{\exp}(x)$: dividing by a positive lower bound yields an upper bound. Obviously the exponential function is not bounded by any rational function for $x > 0$, and one might imagine that the exponential function overtakes its bound after a certain point. In fact, our upper bounds are never overtaken, but reach a singularity as the denominator goes to zero. With $n = 3$, the upper bound is $6/(6-6x+3x^2-x^3)$; its denominator is a cubic equation with one real root at $x \approx 1.60$. We extend this upper limit using range reduction, via the identity $\exp x = \exp(x/k)^k$, for $k = 2$, 4. With $k = 4$ and $n = 3$, the denominator of the upper bound becomes a 12th degree polynomial; QEPCAD easily copes with such high degrees. As always, we can employ only finitely many cases of range reduction.

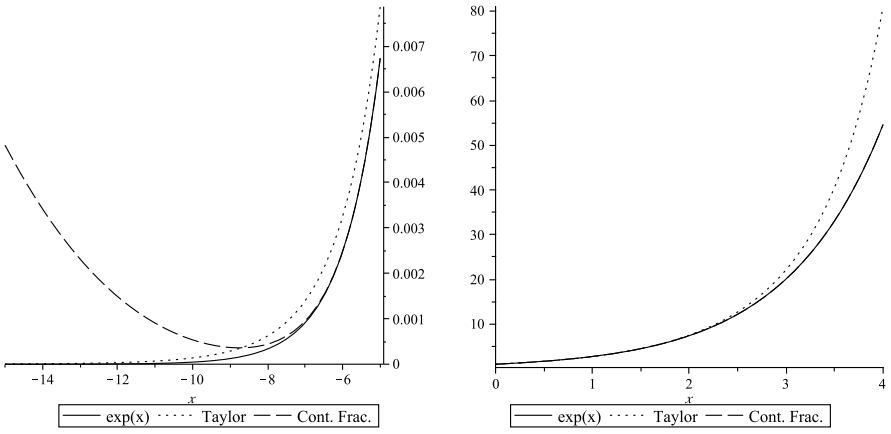

Fig. 5 Upper Bounds for the Exponential Function

Continued fraction bounds have advantages and disadvantages compared with Taylor series bounds. A Taylor series bound is valid for both positive and negative arguments, subject to the singularity mentioned above. With continued fractions, we must

treat the positive and negative cases separately.[1] On the other hand, a continued fraction bound is typically much more accurate (measured as absolute difference) than a similarly complex Taylor series bound, and is good over a wider interval. The Taylor bounds we use are reasonably accurate near zero but go wildly astray as $x \to -\infty$. Some of these points can be seen in Fig. 5, which compares a range-reduced Taylor upper bound with the fifth (for $x \geq 0$) and sixth (for $x \leq 0$) continued fraction approximants. We computed these approximants, as before, using Maple code from a continued fractions library [8].

$$-\frac{x^5 + 30x^4 + 420x^3 + 3360x^2 + 15120x + 30240}{x^5 - 30x^4 + 420x^3 - 3360x^2 + 15120x - 30240} \quad \text{(5th approximant)}$$

$$\frac{x^6 + 42x^5 + 840x^4 + 10080x^3 + 75600x^2 + 332640x + 665280}{x^6 - 42x^5 + 840x^4 - 10080x^3 + 75600x^2 - 332640x + 665280} \quad \text{(6th approximant)}$$

$$21743271936\left(-x^3 + 12x^2 - 96x + 384\right)^{-4} \quad \text{(Taylor)}$$

For $x \geq 0$, the continued fraction upper bounds are valid over wider ranges than the Taylor series bounds, even without range reduction. However, they all reach a singularity eventually. For example, the denominator of the fifth approximant goes to zero at $x \approx 7.29$. Obviously, a bound is not valid beyond such a singularity.

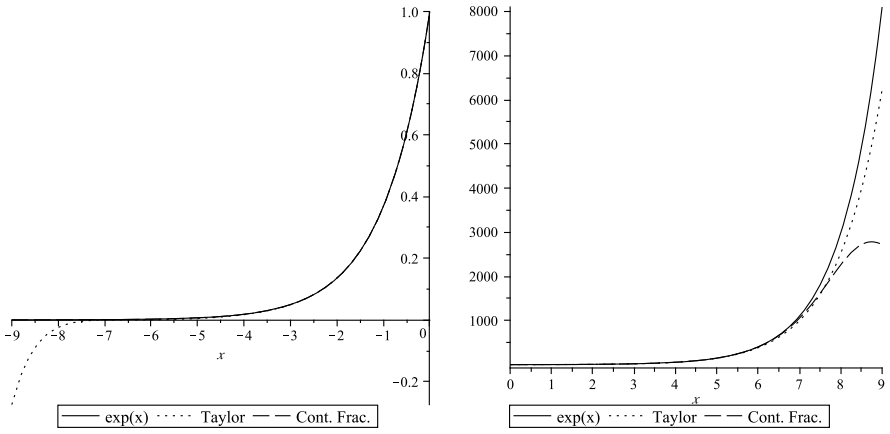

Fig. 6 Lower Bounds for the Exponential Function

Lower bounds are plentiful. By Prop. 3, the truncated Taylor expansion of $\exp x$ for odd n is a lower bound over the entire real line. Unfortunately, for negative arguments they are only accurate near zero. We can again use $\exp x = \exp(x/k)^k$ as a means of range reduction, but only for odd k. In order to use $\underline{\exp}(x/k)^k$ as another lower bound for $\exp x$, it suffices to deduce

$$\underline{\exp}(x/k)^k \leq \exp(x/k)^k = \exp x$$

[1] For $x \geq 0$, odd approximants yield upper bounds of $\exp x$ while even ones yield lower bounds; for $x \leq 0$, the situation is reversed.

from $\underline{\exp}(x/k) \leq \exp(x/k)$, for which we need k to be odd because $\underline{\exp}(x/k)$ could be negative. We use the Taylor expansion with $n = 5$, performing range reduction as described above with $k = 3$; this bound has degree 15 and gives an acceptable fit for $-6 \leq x \leq 6$. We include the lower bound $1 + x$ (the Taylor expansion with $n = 1$) because of its simplicity. We also use continued fraction approximants: the third (for $x \leq 0$) and second (for $x \geq 0$), going right up to the seventh when we need high accuracy. Finally, since the exponential function is always positive, we include zero as a lower bound; the other lower bounds are insufficient to prove $\exp x > 0$.

Figure 6 presents some of our lower bounds. It compares a range-reduced Taylor lower bound, namely

$$\left(1 + \frac{x}{3} + \frac{x^2}{18} + \frac{x^3}{162} + \frac{x^4}{1944} + \frac{x^5}{29160}\right)^3,$$

with the fifth (for $x \leq 0$) and sixth (for $x \geq 0$) continued fraction approximants. Note that the continued fraction lower bound peters out when $x > 8$, while the Taylor bound continues to follow the function upwards.

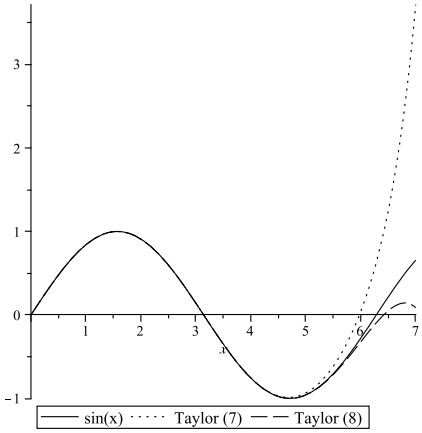

Fig. 7 Taylor Approximations for the Sine Function

3.4 Trigonometric Functions

For $\sin x$ and $\cos x$, we follow Daumas et al. [17] and rely on the Taylor expansions:

$$\sin x = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

$$\cos x = \sum_{i=0}^{n} \frac{(-1)^i}{(2i)!} x^{2i}$$

For $x > 0$, these yield upper bounds when n is even and lower bounds when n is odd; for $x < 0$, the situation is reversed.[2] We illustrate their behaviour by plotting $\sin x$ against its Taylor expansions of seven and eight terms (Fig. 7), the most accurate ones we use; they deteriorate badly when $|x| > 5$.

Daumas et al. [17] also suggest families of bounds that converge to π, but we are using a fixed set of axioms and have simply chosen the pair $3.1415926 < \pi < 3.1415927$.

For the inverse tangent, Daumas et al. [17] use the Taylor expansion once again:

$$\tan^{-1} x = \sum_{i=0}^{n} \frac{(-1)^i}{2i+1} x^{2i+1}$$

Unfortunately, this series is notorious for its slow convergence, especially when $|x| \approx 1$. We adopted it for our early experiments [3], but even 25 terms of the series yielded poor accuracy. We have since adopted bounds derived from its continued fraction representation (11.4.8) [14, p. 207]. They are extremely accurate when $|x| \leq 1$, and otherwise we can transform them by the identities $\tan^{-1} x = \pi/2 - \tan^{-1}(1/x)$ and $\tan^{-1}(-x) = -\tan^{-1} x$. We thereby obtain excellent bounds for the entire real line, at the expense of having separate bounds for the cases $x < -1$, $x \leq 0$, $x \geq 0$, $x > 1$. Note that there is no need to make the ranges of coverage disjoint.

Here are the first four approximants yielded by the continued fraction:

$$x, \quad \frac{3x}{x^2+3}, \quad \frac{(4x^2+15)x}{3(3x^2+5)}, \quad \frac{(11x^2+21)5x}{3(3x^4+30x^2+35)}$$

By Lemma 1 of Cuyt et al. [16], the odd-numbered approximants are upper bounds while the even-numbered ones are lower bounds.

The first two approximants are not very accurate but they yield simple bounds, which benefits the more complicated proofs (those involving nested functions, for instance).

$$\begin{aligned}
&\tan^{-1} x \leq -\frac{\pi}{2} - \frac{1}{x} && (x < -1)\\
&\tan^{-1} x \leq \frac{3x}{x^2+3} && (x < 0)\\
&\tan^{-1} x \leq x && (x > 0)\\
&\tan^{-1} x \leq \frac{\pi}{2} - \frac{3x}{1+3x^2} && (x > 1)
\end{aligned}$$

We also use accurate bounds derived from the fifth and sixth approximants.

$$\begin{aligned}
&\tan^{-1} x \leq -\frac{\pi}{2} - \frac{64+735x^2+945x^4}{15x(15+70x^2+63x^4)} && (x < -1)\\
&\tan^{-1} x \leq \frac{(33x^4+170x^2+165)7x}{5(5x^6+105x^4+315x^2+231)} && (x < 0)\\
&\tan^{-1} x \leq \frac{(64x^4+735x^2+945)x}{15(15x^4+70x^2+63)} && (x > 0)\\
&\tan^{-1} x \leq \frac{\pi}{2} - \frac{(33+170x^2+165x^4)7x}{5(5+105x^2+315x^4+231x^6)} && (x > 1)
\end{aligned}$$

[2] For the proofs of these statements, we refer to Daumas et al. [17], who have formally verified all the bounds in their paper using PVS.

The bounds that we show as valid for $x < -1$ are actually valid for $x < 0$, and those that we show as valid for $x > 1$ are actually valid for all x, but with poor accuracy. We find that restricting them as shown reduces the search space and improves performance.

We omit the lower bounds due to lack of space, but they are easily obtained from those shown. We define $\underline{\tan^{-1}}(x) = -\overline{\tan^{-1}}(-x)$, since

$$\underline{\tan^{-1}}(x) = -\overline{\tan^{-1}}(-x) \leq -\tan^{-1}(-x) = \tan^{-1} x.$$

3.5 Other Functions

Many familiar mathematical functions are normally defined in terms of other functions. Examples include the following:

$$x^y = \exp(y \ln x) \qquad \tan x = \frac{\sin x}{\cos x} \qquad \sinh x = \frac{\exp x - \exp(-x)}{2}$$

It seems natural to use these definitions directly rather than to seek upper and lower bounds from first principles. The prover will then use the approximations it has for the functions on the right-hand side.

Metis, like other modern resolution theorem provers, supports equality reasoning. Given an equality axiom of the form $t = u$, it can replace instances of the term t by corresponding instances of the term u, or vice versa. We can therefore insert such definitions as equality axioms and let resolution do the rest. However, MetiTarski usually gives better results if we regard the definitions as absolutely precise upper and lower bounds,

$$\frac{\sin x}{\cos x} \leq \tan x \leq \frac{\sin x}{\cos x},$$

which we formalise as discussed in Sect. 3.6 below. This use of inequalities has the advantage of postponing the introduction of the definiens (right-hand side) until the definiendum (left-hand side) becomes outermost in a term. Regardless of whether we regard a function's definition as an equality or as a pair of bounds, it is essential to modify the term ordering (Sect. 4.5) to ensure that the definiendum is replaced by the definiens and not the other way round.

Since the difficulty of a problem rises sharply with the number of function occurrences it contains, it may be preferable to specify functions through upper and lower bounds rather than in terms of other special functions. Reference works such as Cuyt et al. [14] present approximations for dozens of special functions, which could serve as the basis for upper and lower bounds. This approach requires more work initially: we have to identify new upper and lower bounds rather than reusing the ones we already have. It may yield better results in the long run.

We specify the *absolute value* function by a pair of clauses:

$$\neg(0 \leq x) \vee |x| = x \qquad 0 \leq x \vee |x| = -x$$

The theorem prover uses these axioms to replace $|t|$ by t or $-t$ through case analysis on the sign of t. This occurs via the paramodulation rule, which is the standard inference rule for dealing with equality. If we have a clause of the form

$$P(|t|) \vee C$$

then paramodulation generates two clauses:

$$P(t) \vee \neg(0 \leq t) \vee C$$
$$P(-t) \vee 0 \leq t \vee C$$

The new literals have the form $\neg(0 \leq t)$ or $0 \leq t$ and they are therefore complementary. To destroy this property by strengthening the second clause of the absolute value function to $0 < x \vee |x| = -x$ would harm performance.

3.6 Axioms

The guiding principle behind our axiom system is to avoid all use of general properties of orderings, such as transitivity, antisymmetry, and monotonicity of addition and multiplication. Such properties are notorious for blowing up the search space. Necessary instances of these properties are built into other axioms, built into simplification or left to the decision procedure. To limit the problem size and search space, we only include axioms that are relevant to the functions that appear in the problem. It is often obvious by inspection whether upper or lower bounds are required. At present the user has to identify the required sets of axioms, although this step would be straightforward to automate. On the other hand, analysis of a problem to determine the required accuracy and range of bounds is difficult; we have identified good general-purpose bounds, but no fixed set can be appropriate for all problems.

A significant change from our earlier work [2] is that the less-than relation no longer exists. We have only one primitive ordering relation, $\leq$. The equivalence $X < Y \iff \neg(Y \leq X)$, formerly a pair of clauses, is built into the parser. When we write $t < u$ in a clause, as just below for example, it actually abbreviates $\neg(u \leq t)$.

To illustrate our formalization of bounds, consider the fact $1 + x \leq \exp x$. We could combine it with transitivity for $\leq$ and $<$ by asserting two axioms:

$$\neg(Y \leq 1 + X) \vee Y \leq \exp(X)$$
$$\neg(Y < 1 + X) \vee Y < \exp(X)$$

However, writing each bound twice would be inconvenient and could result in user errors. Instead we introduce a generalized less-than relation. Its first argument indicates which relation it designates. We express the following two equivalences using the obvious four axiom clauses:

$$\mathrm{lgen}(0, X, Y) \iff X \leq Y$$
$$\mathrm{lgen}(1, X, Y) \iff X < Y$$

Now, the lower bound axiom for $\leq$ and $<$ can be expressed by a single clause:

$$\neg(\mathrm{lgen}(R, Y, 1 + X)) \vee \mathrm{lgen}(R, Y, \exp(X)).$$

The theorem prover will then generate the two clauses shown above.

We modify the term ordering (used in resolution, Sect. 4.5) to ensure that the exp literals are selected. The lower bound clauses combine with literals of the form $\neg(t \leq \exp(u))$ or $\neg(t < \exp(u))$, respectively, and therefore can resolve with a fact of the form $\exp(u) < t$ yielding the new fact $1 + u < t$, and similarly for $\leq$. Since resolution works by negating the conjecture to be proved, these inferences can be

regarded as reducing a conjecture of the form $t \leq \exp(u)$ to $t \leq 1 + u$, and similarly for $<$.

As before [2], we include axioms concerning division in order to encourage its replacement by multiplication. The term ordering is set up (see Sect. 4.5) to ensure that these axioms do not instead replace multiplication by division.

$$\neg(X \leq Y \cdot Z) \vee X/Z \leq Y \vee Z \leq 0$$
$$\neg(X \leq Y/Z) \vee X \cdot Z \leq Y \vee Z \leq 0$$
$$\neg(X \cdot Z \leq Y) \vee X \leq Y/Z \vee Z \leq 0$$
$$\neg(X/Z \leq Y) \vee X \leq Y \cdot Z \vee Z \leq 0$$

Because simplification flattens nested quotients (Sect. 4.1), these axioms are reasonably effective in removing division from a problem. Those shown include the literal $Z \leq 0$, which means they concern the case when $Z > 0$; we also include the analogous axioms for when $Z < 0$.

4 Modifications to the Resolution Procedure

The axioms presented above are sufficient to reduce a problem involving special functions to one involving division and finally to one that is purely algebraic. The remaining reasoning takes place in the theory of real closed fields. To accomplish this reasoning, we modify the resolution procedure so that it interacts with a decision procedure for RCF. Specifically, our modifications are as follows:

- The integer constants are available, and the input file can express fractions in decimal notation; for example, 1.2 denotes $\frac{6}{5}$. The prover can perform rational arithmetic on such fractions. We never use floating point arithmetic.
- Arithmetic expressions are simplified in order to identify redundant forms and to isolate the special functions.
- Ground algebraic literals that are inconsistent with existing algebraic facts are deleted from every new clause. This brings us closer to the empty clause.
- New clauses that follow in RCF from existing algebraic facts are regarded as redundant and deleted. This reduces the use of space and time.
- The built-in term ordering supports *subterm coefficients* [32]. This encourages the replacement of functions by bounds, even when they superficially appear to be more complex.

4.1 Arithmetic Simplification

MetiTarski uses a recursive representation of polynomials. We map all variants of an expression to a unique canonical form. To focus the proof search, we isolate occurrences of special functions and flatten nested divisions.

4.1.1 Horner normal form

All terms built up using constants, negation, addition, subtraction, and multiplication can be considered as multivariate polynomials. Following Grégoire and Mahboubi [22],

we transform them to Horner normal form, also called the *recursive* representation. This representation is *canonical*: distinct representations imply that the original polynomials are not equal.

Any univariate polynomial $a_n x^n + \cdots + a_1 x + a_0$ can be rewritten in recursive form as

$$a_0 + x(a_1 + \cdots x(a_{n-1} + x a_n)).$$

We can consider a multivariate polynomial as a polynomial in one variable whose coefficients are themselves a canonical polynomial in the remaining variables.

For example, we can represent the polynomial $3xy^2 + 2xyz + 4y + yz/2 + 5$ as

$$(5 + y(4 + y(0 + x3))) + z(0 + y(\frac{1}{2} + x2)).$$

It is a polynomial in z whose coefficients are polynomials in y and then x. (Our Horner normal form makes the constant term explicit even if it is zero.) Integer constants denote themselves, while rational numbers are expressed using the binary function symbol "rational", which is distinct from the general division operator.

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 variables x and y are the same, then we just compute $(c + xp) + (d + xq) = (c + d) + x(p + q)$, returning simply $c + d$ if $p + q = 0$. If variable x is smaller than y in the ordering, then $c + xp$ is recursively added to d, yielding some d', and the result is $d' + yq$. The remaining case (y smaller than x) is handled by symmetry. 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 $p \cdot d$ and $p \cdot q$ recursively:

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

However, for $0 + y(p \cdot q)$ 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 computed by repeated multiplication. Our implementation of the canonical form algorithm is based on a preprint of John Harrison's recent book [24].

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 its canonical form Z and returning the equivalent formula $Z \leq 0$. We simplify $1 + x \leq -4$ to $5 + x \leq 0$, 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.

4.1.2 The Treatment of Division

Our normal form supports the operations of addition, subtraction, and multiplication. Division by an integer or rational does not present a problem, since a coefficient can be a rational number: the divisor is recursively supplied to the normal form conversion. Other occurrences of division must somehow be removed from a formula before we can give it to the RCF decision procedure.

Division can occur deep inside expressions as a proof develops. Without special treatment, such occurrences will be difficult to eliminate using resolution alone. Accordingly, we transform an expression containing division into a rational function according to the following rules. (We identify E with $\frac{E}{1}$ if necessary.)

$$\frac{x_1}{y_1} + \frac{x_2}{y_2} = \frac{x_1y_2 + x_2y_1}{y_1y_2} \qquad \frac{x_1}{y_1} \cdot \frac{x_2}{y_2} = \frac{x_1x_2}{y_1y_2}$$
$$\frac{x_1}{y_1} - \frac{x_2}{y_2} = \frac{x_1y_2 - x_2y_1}{y_1y_2} \qquad \frac{x_1}{y_1} \div \frac{x_2}{y_2} = \frac{x_1y_2}{y_1x_2}$$

Thus we replace nested divisions by one single division, which as the outermost symbol can be eliminated by one proof step using an appropriate division axiom (see Sect. 3.6). In this example, three divisions are replaced by one.

$$\left(\frac{x}{y}\right) \frac{1}{\left(x + \frac{1}{x}\right)} = \frac{x^2}{y(x^2+1)}$$

We add literals to the resulting clause to account for the possibility of division by zero. In particular, if we simplify $x_1/y_1 + x_2/y_2$ then we make the resulting clause conditional on $y_1 \neq 0$ and $y_2 \neq 0$. However, for $(x_1/y_1) \cdot (x_2/y_2)$ and $(x_1/y_1) \div (x_2/y_2)$, no such conditions are necessary. That is because we define $x/0 = 0$. It is trivial to see that $(x_1/y_1) \cdot (x_2/y_2) = 0$ if and only if any of x_1, x_2, y_1, y_2, are zero, and in this they agree with the corresponding right-hand side.

On division by zero. The Isabelle and HOL communities are comfortable with a formalised mathematics that defines $x/0 = 0$ on many numerical domains. They appreciate that it simplifies deductions by making certain identities unconditional, such as $(x \cdot y)^{-1} = x^{-1} \cdot y^{-1}$. But some mathematicians view the idea with suspicion. To simplify the discussion, let us restrict ourselves to fields, taking x/y as an abbreviation for $x \cdot y^{-1}$ and restricting the issue to the status of 0^{-1}.

In first-order logic, all functions are total. There are no models of the field axioms in which 0^{-1} is undefined: 0^{-1} must denote something. Because the field axioms do not constrain the value of 0^{-1}, the axiom $0^{-1} = 0$ is consistent with them; assuming it cannot allow anything to be proved that contradicts the other axioms. If we augment the real number system with an undefined value ∞ and augment the field axioms with axioms to propagate undefinedness, then we can derive $\infty = 0 \cdot \infty = 0$; that approach just replaces the issue of 0^{-1} with the equally vexing issue of $0 \cdot \infty$.

Bergstra and Tucker [10] have recently given an equational specification of the rational numbers from which $0^{-1} = 0$ is deducible, which they readily accept. Their article presents the issue in historical perspective.

4.1.3 Isolating Function Occurrences

We attempt to isolate occurrences of special functions. In the Horner normal form transformation (Sect. 4.1.1), we regard any non-algebraic term (preferably a special function occurrence) as a variable. We order the variables, taken in this general sense, using Metis's built-in Knuth-Bendix ordering. This ensures that one of the function occurrences will appear as the outermost "variable." If we detect this situation, we leave this term by itself on one side of the inequality, for example as $\ln t \leq t'$. We even

divide both sides by any constant coefficient, so that $-2 \ln t \leq t'$ becomes $-t'/2 \leq \ln t$. This transformation is built into arithmetic simplification.

Isolating the function is more difficult in cases such as $(\ln t)u \leq t'$, where a special function is multiplied by an arbitrary term u. It is natural to divide both sides of the inequality by this term, but we cannot do so unless we know the sign of u. Our solution is to generate a case analysis. This step cannot be integrated with simplification; it is, in fact, implemented as a separate rule of inference. Its logical justification is trivial, by the properties of division.

If we have a clause of the form $tu \leq t' \vee C$ (the analogous inference is available for $t' \leq tu \vee C$), then we create four new clauses:

$$\begin{aligned} t \leq t'/u \vee u \leq 0 \vee C \\ 0 \leq t' \vee u \neq 0 \vee C \\ t'/u \leq t \vee u \geq 0 \vee C \\ u < 0 \vee u = 0 \vee u > 0 \end{aligned}$$

The first three clauses describe the situation when $u > 0$, $u = 0$ or $u < 0$, respectively. The final clause expresses case analysis on the sign of u. We could express this trichotomy axiom in full generality, but its effect on the search space would be catastrophic. It is interesting to note that few of our examples require the fourth clause; it is only necessary when u is not algebraic (otherwise QEPCAD could perform the necessary reasoning) and cannot be proved to have one uniform sign.

4.2 Algebraic Literal Deletion

Literal deletion [2] simplifies new clauses that emerge from inference rules. For each ground algebraic literal in such a clause, we conjoin it with the negations of all ground algebraic literals in that clause (its context) and with all ground algebraic clauses known to the prover. Taking as variables all Skolem constants present in this conjunction, we proceed to form its existential closure. If the RCF solver (QEPCAD) reduces this existential formula to **false**, then the literal under consideration is deleted. This is the primary mechanism by which the decision procedure contributes to deduction.

As a small example, suppose we are trying to prove

$$\forall x \; [-3 < x < 1 \rightarrow \ln(1 - x) \leq -x]$$

with the help of a range-restricted polynomial upper bound f_2,

$$\forall x \; [2 \leq x \leq 4 \rightarrow \ln x \leq f_2(x)] \, .$$

Skolemization of the conjecture will yield three clauses, with u a Skolem constant:

$$-3 < u \qquad u < 1 \qquad \neg[\ln(1 - u) \leq -u].$$

MetiTarski maintains a list consisting of all ground algebraic clauses (regardless of whether they are active or passive). At the start of the proof, $-3 < u$ and $u < 1$ will be the only elements of this list. As the proof proceeds, a resolution step will eventually substitute our upper bound, yielding the following unsimplified clause:

$$f_2(1 - u) \leq -u \vee 2 > 1 - u \vee 1 - u > 4. \tag{2}$$

Ordinary arithmetic simplification can reduce $2 > 1 - u$ to $u > -1$, and $1 - u > 4$ to $-3 > u$, but if $f_2(1-u)$ is a complicated polynomial, then only QEPCAD can achieve a real simplification: we give it the formula

$$\exists u \left[f_2(1-u) \leq -u \wedge \underbrace{u \leq -1 \wedge -3 \leq u}_{\text{negated literals}} \wedge \underbrace{-3 < u \wedge u < 1}_{\text{algebraic clauses}} \right].$$

Provided f_2 is a sufficiently tight bound, the result will be **false** and the literal can be deleted from clause (2). The literal $u > -1$ turns out to be consistent with its context, so it is preserved. Then we call QEPCAD for $-3 > u$:

$$\exists u \left[-3 > u \wedge u \leq -1 \wedge -3 < u \wedge u < 1 \right].$$

This again is **false**, and the final simplified clause is

$$u > -1.$$

It will be added to our list of ground algebraic clauses. We have tightened the range of u to $-1 < u < 1$; if it becomes empty, then we have reached a contradiction.

In this example, the constraints that accumulate are linear, but in general they could relate arbitrary polynomials. QEPCAD can detect inconsistencies among nonlinear constraints.

4.3 Algebraic Subsumption

Resolution theorem provers generate many redundant clauses. To conserve space, they typically delete any clause that is a syntactic instance of another. The redundancy test is applied just before new clauses are added to the passive clause set (recall Fig. 1). We generalize this redundancy criterion, known as *subsumption*, by performing an analogous redundancy check in the RCF theory.

When a new clause is generated, we identify its ground algebraic literals and form their disjunction. If this disjunction is an algebraic consequence of the existing ground algebraic clauses, then we ignore the new clause; in future QEPCAD calls, it could only contribute redundant information. This technique can even improve the performance of some failing proofs so that they terminate (reporting failure) rather than running forever. The resulting performance improvement depends on other aspects of the formalization; at present, around four percent of our problems are proved significantly faster when this technique is enabled. Note that if we applied this idea to standard resolution inferences, then all new clauses would be ignored, because all clauses are logical consequences of the previous clauses.

Recall our previous example, where the ground algebraic clauses included $-1 < u$ and $u < 1$. Suppose that a resolution step yields the following clause:

$$\ln(1-u) \leq u^2 \vee u^2 < 2 \vee 4u > 3.$$

Algebraic subsumption will call QEPCAD with the formula

$$\exists u. \underbrace{u^2 \geq 2 \wedge 4u \leq 3}_{\text{negated literals}} \wedge \underbrace{-1 < u \wedge u < 1}_{\text{algebraic clauses}}.$$

QEPCAD returns **false**, indicating that the algebraic part of the clause follows from $-1 < u < 1$. The clause is discarded.

4.4 Removal of Arithmetic Tautologies

The division axioms presented above (Sect. 3.6) give rise to many fruitless deductions. These yield clauses such as $X \leq 0 \vee X \geq 0 \vee \ldots$. Note that the slightly different clause $X \leq 0 \vee X > 0$ is a propositional tautology, because $X > 0$ abbreviates $\neg(X \leq 0)$. It is therefore natural to interpret $X \leq 0 \vee X \geq 0$ as a tautology also. Generalising this idea, we examine each literal of the form $t \leq 0$ or $t \geq 0$ in a newly deduced clause in order to determine whether it is tautologous in the context of the neighbouring literals. If the term has the form $t_1 \cdot t_2$ or t_1/t_2, then we recursively perform the obvious sign computation on t_1 and t_2. If the computed sign for t logically implies the literal, then the entire clause is deleted as tautologous.

For example, consider the clause $X \leq 0 \vee Y \leq 0 \vee X \cdot Y > 0$. It is logically equivalent to

$$X > 0 \wedge Y > 0 \rightarrow X \cdot Y > 0,$$

so when we examine the literal $X \cdot Y > 0$, we can assume $X > 0$ and $Y > 0$. The sign computation concludes that $X \cdot Y$ is positive: the literal $X \cdot Y > 0$ is implied by its context, so the clause is deleted. The effect is to reduce the search space by ignoring deductions that cannot lead to a contradiction.

This procedure is the sole exception to our principle that properties of the special functions are specified by axioms rather than being built into the code. Our sign computation gives terms of the form $\exp x$ and $\cosh x$ a positive sign; for example, the clause $X \leq 0 \vee X \exp Y > 0$ will be deleted.

4.5 Modified Knuth-Bendix Ordering

The execution of a modern resolution prover is governed by a term ordering [7]. This ordering serves several purposes:

- to orient equations appropriately,
- to eliminate redundant combinations of inferences (those that could never produce essentially new results), and
- to draw the prover's attention to literals of high priority.

Metis follows most resolution theorem provers in providing the Knuth-Bendix ordering (KBO) [6, p. 124]. Its advantages include computational efficiency and a tendency to prefer simpler terms. The latter property, however, can be a drawback.

It gives an equation like $X - X = 0$ the obvious orientation, but others can be difficult to orient. For example, consider the equation $u = \frac{1}{2}$, where u is a Skolem constant. The default ordering will consider the fraction $\frac{1}{2}$ (which is a function application) to be more complex than u, and will therefore choose the perverse orientation $\frac{1}{2} = u$. We can solve this issue by assigning suitable *weights*. Weights (typically positive integers) can be assigned to all function symbols; the sum of the weights in a term is a key measure in the ordering. By giving Skolem constants a weight of 5, while integers and the function "rational" have a weight of 1, MetiTarski ensures that the orientation $u = \frac{1}{2}$ is chosen. Weights can similarly ensure that the equation $\tan x = \sin x / \cos x$ is oriented from left to right, so that it eliminates rather than introduces the tangent function.

Ordered resolution can implement our strategy of replacing special functions by rational functions and then division by multiplication, provided we adopt a suitable ordering. We are concerned with clauses such as the following:

$$\neg 0 < X \vee \neg[(X+5)(X-1)/(4X+2) \leq Y] \vee \ln X \leq Y.$$

This combines the upper bound property $\ln X \leq (X+5)(X-1)/(4X+2)$ with transitivity, allowing $\ln X$ to be replaced by its bound. We would like resolution to select the literal $\ln X \leq Y$ in order to eliminate an occurrence of $\ln t$ from another clause. Unfortunately, standard KBO will want to select $\neg[(X+5)(X-1)/(4X+2) \leq Y]$ because it is syntactically larger than $\ln X \leq Y$. We can attempt to force the issue by assigning ln a very high weight. Then $\ln X \leq Y$ will be selected, but the second literal will continue to be selected as well: KBO takes into account the number of variable occurrences in the terms being compared, and the upper bound for $\ln X$ contains multiple occurrences of X while $\ln X$ contains only one occurrence. Therefore both literals are maximal. Selecting multiple literals for resolution is normal, but in this case it needlessly expands the search space.

Ludwig and Waldmann [32] provide a solution to this difficulty. They give precise definitions of useful extensions to KBO, along with theory and implementation advice. We have modified Metis's built-in ordering so that a function can have not only a weight, but also a *subterm coefficient*. For example, if ln has a subterm coefficient of 10, then each occurrence of a variable in $\ln t$ is equivalent to 10 occurrences of that variable in t; then $\ln X$ is regarded as containing 10 occurrences of X while $(X+5)(X-1)/(4X+2)$ continues to contain only three occurrences. If we make a function's subterm coefficient large enough (greater than the number of occurrences of the variable in any bound), we can ensure that a literal containing that function is selected every time. This modification to Metis yields dramatic reductions in solution times for the great majority of problems.

A further detail is extremely important. Ordered resolution frequently employs a heuristic entitled *negative selection*: a literal's sign is taken into account, in addition to its rank in the ordering. Specifically, only maximal negative literals can be selected for resolution. Metis employs negative selection by default but also offers[3] unsigned literal selection. With this option, 76 percent of our problems are proved; with negative selection, only 10 percent are proved; with no ordering whatever (all literals selected), 42 percent are proved.[4] The terrible result with negative selection, where 58 percent of the proof attempts quickly terminate with a result of "countersatisfiable" is strange: negative selection should be complete, but clearly not with our heuristics! MetiTarski uses unsigned literal selection in order to eliminate occurrences of special functions regardless of their sign.

4.6 Waiting Queue Parameters

A resolution prover manages the unprocessed clauses in a priority queue. Other things being equal, the following priorities apply:

1. A simple clause will have priority over one containing large expressions.

[3] via a simple change to its source code, see file `Clause.sml`

[4] Tests were run on a 2.66 GHz Mac Pro allowing 10 seconds per problem.

2. A clause containing few literals will have priority over one containing many literals.
3. Older clauses have priority over younger ones.

The last point above ensures that every clause will eventually be processed, which is essential for completeness. The Metis prover, by default, gives equal weight to the first two points and a very low weight to the third. MetiTarski modifies this basic framework.

The complexity of an expression is generally referred to as its *weight*, but note that there is no connection between this concept of weight and that mentioned in the previous section. The weight of an expression is typically the sum of the weights of its constituent variables, constants and function symbols. Our upper and lower bounds produce large expressions. We therefore assign low weights to the algebraic operators (addition, subtraction and multiplication), a slightly higher weight to division, much higher weights to the special functions, and a very high weight to the absolute value function. All constants have a weight of zero. Variables receive an unusual treatment: the first occurrence of a variable in a literal is assigned a high weight, but subsequent occurrences are assigned a low weight. This penalises clauses that contain many literals containing variables. For ground literals, we have almost eliminated the penalty assessed on the number of literals in a clause. Many of our problems require extensive case analysis, which means that their proofs will require clauses having many literals. We have determined these values[5] by extensive empirical testing. They give much better results than the default settings.

5 Results and Related Work

As of this writing, we have nearly 400 problems, of which 79 percent are proved in under 40 seconds. For this paper, we present (Table 1) a small sample of the more interesting and difficult problems. The runtimes were measured on a 2.66 GHz Mac Pro running Poly/ML.

MetiTarski can prove problems involving square roots, but each square root $\sqrt{E}$ in the problems presented here has been manually replaced by a new variable y such that $y \geq 0$ and $y^2 = E$. This transformation encodes square roots as algebraic constraints and can easily be automated. Obviously, it is only useful if E is algebraic.

Our problems come from a variety of sources. Some were suggested by colleagues; many others come from mathematical textbooks and reference works [1,29,30,35,36]. We have recently been applying MetiTarski to problems in hybrid systems and control theory [5] and to analogue circuit verification [18]. For several problems obtained from the HSolver website [42], MetiTarski performs better than HSolver [41] itself, as we describe elsewhere [5].

Hybrid system problems are frequently expressed using linear differential equations, which can be solved using a computer algebra system such as Maple. The result is an inequality, typically involving the exponential, sine and cosine functions, which MetiTarski can often prove. The following formula arises from a collision avoidance

[5] Currently 10 for variables but 450 on the first occurrence; 7 for algebraic operators, 40 for division, 100 for special functions and 900 for the absolute value function.

Table 1 Problems and Runtimes

problem	seconds
$-1 < x \Longrightarrow 2\lvert x\rvert/(2+x) \leq \lvert\ln(1+x)\rvert$	0.372
$\lvert x\rvert < 1 \Longrightarrow \lvert\ln(1+x)\rvert \leq -\ln(1-\lvert x\rvert)$	0.100
$\lvert x\rvert < 1 \Longrightarrow \lvert x\rvert/(1+\lvert x\rvert) \leq \lvert\ln(1+x)\rvert$	0.318
$\lvert x\rvert < 1 \Longrightarrow \lvert\ln(1+x)\rvert \leq \lvert x\rvert(1+\lvert x\rvert)/\lvert 1+x\rvert$	0.611
$\lvert x\rvert < 1 \Longrightarrow \lvert x\rvert/4 < \lvert\exp x - 1\rvert$	0.230
$0 < \lvert x\rvert < 1 \Longrightarrow \lvert\exp x - 1\rvert < 7\lvert x\rvert/4$	0.262
$\lvert\exp x - 1\rvert \leq \exp(\lvert x\rvert) - 1$	0.174
$\lvert\exp x - (1+x)\rvert \leq \lvert\exp(\lvert x\rvert) - (1+\lvert x\rvert)\rvert$	0.327
$\lvert\exp x - (1+x/2)^2\rvert \leq \lvert\exp(\lvert x\rvert) - (1+\lvert x\rvert/2)^2\rvert$	0.368
$0 \leq x \Longrightarrow 2x/(2+x) \leq \ln(1+x)$	0.110
$-1/3 \leq x \leq 0 \Longrightarrow x/\sqrt{1+x} \leq \ln(1+x)$	0.121
$1/3 \leq x \Longrightarrow \ln((1+x)/x) \leq (12x^2+12x+1)/(12x^3+18x^2+6x)$	0.211
$1/3 \leq x \Longrightarrow \ln((1+x)/x) \leq 1/\sqrt{x^2+x}$	0.109
$0 < y < x \Longrightarrow (1/2)\ln(x/y) > (x-y)/(x+y)$	0.181
$0 \leq x \leq 1 \Longrightarrow \exp(x-x^2) \leq 1+x$	0.125
$x \leq 1/2 \Longrightarrow \exp(-x/(1-x)) \leq 1-x$	0.386
$\lvert x\rvert < 1 \Longrightarrow \lvert\sin(x)\rvert \leq 6/5\lvert x\rvert$	0.116
$0 < x < 100/201 \Longrightarrow \cos(\pi x) > 1-2x$	0.296
$\cos(x) - 1 + x^2/2 \geq 0$	0.005
$x > 0 \Longrightarrow \tan^{-1} x > 8\sqrt{3}\,x/(3\sqrt{3}+\sqrt{75+80x^2})$	285.200
$x > 0 \Longrightarrow (x+1/x)\tan^{-1} x > 1$	0.103
$x > 0 \Longrightarrow \tan^{-1} x > 3x/(1+2\sqrt{1+x^2})$	3.299
$0 < x \leq \pi \Longrightarrow \cos(x) \leq \sin(x)/x$	0.143
$0 < x < \pi/2 \Longrightarrow \cos x < \sin^2 x/x^2$	0.324
$\pi/3 \leq x \leq 2\pi/3 \Longrightarrow \sin x/3 + \sin(3x)/6 > 0$	1.368

system:

$$0 \leq x \leq 2 \Longrightarrow$$
$$12 - 14.2\exp(-0.318x) + \big[3.25\cos(1.16x) - 0.155\sin(1.16x)\big]\exp(-1.34x) > 0.$$

MetiTarski can prove this formula in 25 seconds. MetiTarski can also prove a wide variety of problems derived from the verification of Nichols plots [21, 23]. These typically involve the arctangent, logarithm and square root functions; many of the proofs take under one second.

Proofs require surprisingly few steps. Of our current set of problems, 314 can be proved with a limit of 300 seconds. Of these 314 proofs, the longest is 482 steps. The median proof is 50 steps long, so 50 per cent of the proofs are no longer than that. Because many of the proofs contain large formulas, we also examined their length in tokens (counted using the UNIX utility wc). The largest proof is 6586 tokens long and the median proof is 352 tokens long.

5.1 Limitations

Limitations of our approach can be seen in the facts that cannot be proved. We can prove $\cos(\pi x) > 1 - 2x$ under the assumption $0 < x < 100/201$ but unfortunately not

under the assumption $0 < x < 1/2$: our approximation to π is fixed and some precision is inevitably lost. If MetiTarski cannot prove a formula (whether it is a theorem or not), it typically runs forever rather than reporting this fact.

Our approach is inherently incapable of proving non-algebraic equalities. The idea of reducing $f(x) = g(x)$ to a pair of inequalities must fail, as we cannot hope to prove $f(x) \leq g(x)$ after we have substituted an upper bound for f and a lower bound for g. At best we can prove $|f(x) - g(x)| < \epsilon$ for some positive ϵ, whose value will depend on the accuracy of our bounds.

QEPCAD is hyperexponential in the number of variables, and it often fails to terminate if the problem involves more than three variables. Eliminating this serious limitation requires the development of more efficient RCF decision procedures. QEPCAD is usually to blame when a proof takes a long time; the proof in Table 1 that takes 285 seconds spends 284 seconds in QEPCAD. We do not require the full power of QEPCAD, which handles both existential and universal quantifiers; a decision procedure that can prove purely universal formulas would suffice.

5.2 Related Work

We are not aware of much related work. SPASS+T [38] combines a resolution theorem prover (SPASS) with an arbitrary SMT (satisfiability modulo theories) procedure. The objective is to combine resolution's power to handle quantification with SMT's ability to cope with huge propositional formulas whose atoms involve linear arithmetic or other decidable theories. It terminates successfully if either SPASS or the SMT procedure detect a contradiction. There are clear parallels with our project, but our method of integration is different (we use the decision procedure to simplify individual clauses) and we focus on a different problem domain, namely that of the real-valued special functions.

Interval-based arithmetic constraint solving is a general method for handling problems that involve non-linear formulas over the reals. In combination with an SMT solver, it has the potential to solve large problems in many variables. It generally yields a decision procedure, while theorem provers often fail to terminate. However, interval arithmetic also has drawbacks. It does not deliver proofs of its claims. Floating point arithmetic is typically used, rounding conservatively to ensure correctness. Interval arithmetic can lose precision rapidly in certain situations. Many of our examples are proved over unbounded intervals, which is not possible with interval arithmetic.

RSolver [39] is a constraint solver based on interval arithmetic. Termination is only guaranteed for problems that are *robust* "in the sense that their truth value ... does not change under small perturbations of the occurring constants." For example, the theorem $\forall x\ [1 + x \leq \exp(x)]$ is not robust because $\forall x\ [(1 + \epsilon) + x \leq \exp(x)]$ fails for all $\epsilon > 0$. More generally, any theorem of the form $\forall x \in I\ [f(x) \leq g(x)]$ is not robust if $f(x) = g(x)$ for some $x \in I$. The robustness requirement is natural in engineering applications but it does not hold for many of the problems in Table 1. HSolver [41] is a program for verifying of hybrid systems based upon RSolver; it works by generating a discrete approximation of the continuous state space. HySAT [20] combines SAT solving techniques with interval-based arithmetic constraint solving. Both HySAT [25] and RSolver [40] can reason about some transcendental functions but neither supports the division operator and therefore cannot express many of our problems.

Tangential to our approach, but possibly of interest, is the work of Lafferriere et al. [31] concerning the reachability problem for systems of linear differential equations. In general, closed form solutions to such systems involve transcendental functions, but they can be reduced to polynomial problems in certain highly specialised cases. For example, occasionally all occurrences of x belong to the expression $\exp(x)$, so a simple change of variable eliminates the exponential function. QEPCAD and similar decision procedures are then applicable. This work can be used to verify hybrid systems.

6 Conclusions

MetiTarski, which combines a resolution theorem prover with specialized simplification and a decision procedure, can prove numerous facts involving special functions automatically. By further refining our techniques, we expect to prove increasingly difficult theorems. The approach is flexible, and should work with any well-behaved functions.

MetiTarski delivers machine-readable proofs that could be checked, in principle, by a separate tool. Checking a proof is much easier than finding a proof because it requires no search, and proofs are fairly short. A complicating factor is that QEPCAD performs lengthy computations using sophisticated algorithms. A proof verifier could check its results by calling an alternative RCF decision procedure, but it would be better not to trust any decision procedure. We do not require the full power of QEPCAD, since we only seek to refute $\exists$-formulas (equivalently, to prove $\forall$-formulas). We could therefore consider decision procedures that yield independently checkable certificates, for example by transforming a polynomial into a sum of squares [37]. The transformation is easily verified using elementary algebra, and it is trivial to check that a sum of squares is nonnegative. Such an approach may yield both better performance and more trustworthy proof reconstruction.

Our choice of Metis over more advanced resolution provers has been successful. We have obtained acceptable performance and the source code has been easy to modify. ML's garbage collector has certainly simplified our task.

Resolution is traditionally regarded first as a formal calculus and only second as an implementation. A resolution calculus is first developed, then proved to be complete, before an implementation is contemplated. Our results demonstrate that modifying an implementation can deliver proofs and insights. Our modifications are sympathetic to the overall architecture of resolution: we modify its notions of simplification and subsumption and its ordering. We ignore completeness because proving something is better than proving nothing. Nonetheless, we welcome suggestions for achieving completeness under particular circumstances.

Acknowledgements The research was supported by the Engineering and Physical Sciences Research Council [grant number EP/C013409/1]. Joe Hurd offered much help with his Metis prover. Christopher W. Brown and Ian Grant provided support for QEPCAD. Annie Cuyt helped us with continued fractions. John Harrison, David Lester, César Muñoz , Thomas Türk and Uwe Waldmann answered various queries. Jeremy Avigad commented on a draft of this paper and he also provided several problems. The referees scrutinised the paper carefully and made numerous comments and suggestions.

References

1. M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Wiley, 1972.
2. B. Akbarpour and L. Paulson. Extending a resolution prover for inequalities on elementary functions. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 47–61, 2007.
3. B. Akbarpour and L. Paulson. MetiTarski: An automatic prover for the elementary functions. In S. Autexier et al., editors, *Intelligent Computer Mathematics*, LNCS 5144, pages 217–231. Springer, 2008.
4. B. Akbarpour and L. C. Paulson. Towards automatic proofs of inequalities involving elementary functions. In B. Cook and R. Sebastiani, editors, *PDPAR: Pragmatics of Decision Procedures in Automated Reasoning*, pages 27–37, 2006.
5. B. Akbarpour and L. C. Paulson. Applications of MetiTarski in the verification of control and hybrid systems. In R. Majumdar and P. Tabuada, editors, *Hybrid Systems: Computation and Control*, LNCS 5469, pages 1–15. Springer, 2009.
6. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
7. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.
8. F. Backeljauw, S. Becuwe, M. Colman, A. Cuyt, and T. Docx. Special functions: continued fraction and series representations, 2008. On the Internet at `http://www.cfhblive.ua.ac.be/`.
9. M. Beeson. Automatic generation of a proof of the irrationality of e. *JSC*, 32(4):333–349, 2001.
10. J. A. Bergstra and J. V. Tucker. The rational numbers as an abstract data type. *J. ACM*, 54(2):Article No. 7, 2007.
11. C. W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *SIGSAM Bulletin*, 37(4):97–108, 2003.
12. P. S. Bullen. *A Dictionary of Inequalities*. Longman, 1998.
13. E. Clarke and X. Zhao. Analytica: A theorem prover for Mathematica. *Mathematica Journal*, 3(1):56–71, 1993.
14. A. Cuyt, V. Petersen, B. Verdonk, H. Waadeland, and W. Jones. *Handbook of Continued Fractions for Special Functions*. Springer, 2008.
15. A. A. M. Cuyt. Upper/lower bounds. E-mail dated 7 September 2008.
16. A. A. M. Cuyt, B. Verdonk, and H. Waadeland. Efficient and reliable multiprecision implementation of elementary and special functions. *SIAM J. Scientific Computing*, 28(4):1437–1462, 2006.
17. M. Daumas, C. Muñoz, and D. Lester. Verified real number calculations: A library for integer arithmetic. *IEEE Trans. Computers*, 58(2):226–237, 2009.
18. W. Denman, B. Akbarpour, S. Tahar, M. Zaki, and L. C. Paulson. Automated formal verification of analog designs using MetiTarski. In A. Biere and C. Pixley, editors, *Formal Methods in Computer Aided Design*, 2009. Accepted for publication.
19. A. Dolzmann, T. Sturm, and V. Weispfenning. Real quantifier elimination in practice. Technical Report MIP-9720, Universität Passau, D-94030, Germany, 1997.
20. M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.
21. H. Gottliebsen, R. Hardy, O. Lightfoot, and U. Martin. Applications of real number theorem proving in PVS. Preprint, 2007.
22. B. Grégoire and A. Mahboubi. Proving equalities in a commutative ring done right in Coq. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2005*, LNCS 3603, pages 98–113. Springer, 2005.
23. R. Hardy. *Formal Methods for Control Engineering: A Validated Decision Procedure for Nichols Plot Analysis*. PhD thesis, University of St Andrews, 2006.
24. J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.
25. C. Herde. *HySAT Quick Start Guide*. University of Oldenburg, 2008. On the Internet at `http://hysat.informatik.uni-oldenburg.de/user_guide/hysat-user-guide.pdf`.

26. H. Hong. QEPCAD — quantifier elimination by partial cylindrical algebraic decomposition. Sources and documentation are on the Internet at `http://www.cs.usna.edu/~qepcad/B/QEPCAD.html`.
27. L. 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 [34].
28. J. Hurd. Metis first order prover. `http://gilith.com/software/metis/`, 2007.
29. W. J. Kaczor and M. T. Nowak. *Problems in Mathematical Analysis II: Continuity and Differentiation*. American Mathematical Society, 2001.
30. P. P. Korovkin. *Inequalities*. Pergamon, 1961.
31. G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.
32. M. Ludwig and U. Waldmann. An extension of the Knuth-Bendix ordering with LPO-like properties. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2007*, volume 4790 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 2007.
33. W. McCune and L. Wos. Otter: The CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.
34. S. McLaughlin and J. Harrison. A proof-producing decision procedure for real arithmetic. In R. Nieuwenhuis, editor, *Automated Deduction — CADE-20 International Conference*, LNAI 3632, pages 295–314. Springer, 2005.
35. D. S. Mitrinović and P. M. Vasić. *Analytic Inequalities*. Springer, 1970.
36. J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhäuser, 2nd edition, 2006.
37. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003.
38. V. Prevosto and U. Waldmann. SPASS+T. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *FLoC'06 Workshop on Empirically Successful Computerized Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 18–33, 2006.
39. S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Logic*, 7(4):723–748, 2006.
40. S. Ratschan. *RSolver User Manual*. Academy of Sciences of the Czech Republic, 2007. On the Internet at `http://rsolver.sourceforge.net/documentation/manual.pdf`.
41. S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6(1), 2007.
42. S. Ratschan and Z. She. Benchmarks for safety verification of hybrid systems, 2008. `http://hsolver.sourceforge.net/benchmarks/`.
43. G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP data-exchange formats for automated theorem proving tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, number 112 in Frontiers in Artificial Intelligence and Applications, pages 201–215. IOS Press, 2004.

A Sample Proof

We present the proof of the formula

$$x > 0 \Longrightarrow (x + 1/x) \tan^{-1} x > 1.$$

In order to save space, we have manually removed applications of the substitution rule, which most resolution provers would omit anyway. Otherwise, the output is precisely as it was generated. Without examining the proof in detail, we can see which axioms were used and which facts were deduced. In line 9, we can see how the arctan function was isolated by dividing both sides of the previous line by $1 + x^2$, which QEPCAD proves in line 11 to be positive. By line 12, MetiTarski is trying to derive a contradiction from $\tan^{-1} x \leq x/(1 + x^2)$. By line 26, MetiTarski has derived $2x^3 \leq 0 \vee x^2 \leq -3$, which quickly leads to the desired contradiction.

```
SZS output start CNFRefutation for atan-problem-9.tptp
fof(atan_problem_9, conjecture,
    (! [X] : (~ X <= 0 => ~ (X + 1 / X) * arctan(X) <= 1))).

cnf(leq_right_mul_divide_pos, axiom, (~ X / Z <= Y | X <= Y * Z | Z <= 0)).

cnf(0, plain,
    (~ skoX * ((1 + skoX * skoX) * 3) / (skoX ^ 2 + 3) <= skoX |
     skoX * ((1 + skoX * skoX) * 3) <= skoX * (skoX ^ 2 + 3) |
     skoX ^ 2 + 3 <= 0), inference(subst, [leq_right_mul_divide_pos])).

cnf(leq_left_mul_divide_pos, axiom, (~ X <= Y / Z | Z <= 0 | X * Z <= Y)).

cnf(2, plain, ((skoX + 1 / skoX) * arctan(skoX) <= 1),
    inference(fof_to_cnf, [], [atan_problem_9])).

cnf(3, plain, (arctan(skoX) / skoX <= 1 + arctan(skoX) * -skoX),
    inference(arith, [2])).

cnf(5, plain,
    (arctan(skoX) <= (1 + arctan(skoX) * -skoX) * skoX | skoX <= 0),
    inference(resolve, [3, leq_right_mul_divide_pos])).

cnf(6, plain, (arctan(skoX) * (1 + skoX * skoX) <= skoX | skoX <= 0),
    inference(arith, [5])).

cnf(7, plain, (~ skoX <= 0), inference(fof_to_cnf, [], [atan_problem_9])).

cnf(8, plain, (arctan(skoX) * (1 + skoX * skoX) <= skoX),
    inference(resolve, [6, 7])).

cnf(9, plain,
    (1 + skoX * skoX <= 0 | arctan(skoX) <= skoX / (1 + skoX * skoX)),
    inference(split, [8])).

cnf(10, plain,
    (skoX * skoX <= -1 | arctan(skoX) <= skoX / (1 + skoX * skoX)),
    inference(arith, [9])).

cnf(11, plain, (~ skoX * skoX <= -1), inference(decision, [7])).

cnf(12, plain, (arctan(skoX) <= skoX / (1 + skoX * skoX)),
    inference(resolve, [10, 11])).

cnf(lgen_less_neg, axiom, (~ Y <= X | ~ lgen(1, X, Y))).

cnf(atan_lower_bound_case_13, axiom,
    (~ 0 <= X | ~ lgen(R, Y, 3 * X / (X ^ 2 + 3)) |
     lgen(R, Y, arctan(X)))).

cnf(15, plain,
    (~ 0 <= X0 | ~ arctan(X0) <= X1 |
     ~ lgen(1, X1, 3 * X0 / (X0 ^ 2 + 3))),
     inference(resolve, [atan_lower_bound_case_13, lgen_less_neg])).

cnf(16, plain,
    (~ 0 <= X0 | ~ arctan(X0) <= X1 | 3 * X0 / (X0 ^ 2 + 3) <= X1),
    inference(arith, [15])).

cnf(18, plain,
```

```
    (~ 0 <= skoX | 3 * skoX / (skoX ^ 2 + 3) <= skoX / (1 + skoX * skoX)),
    inference(resolve, [12, 16])).

cnf(19, plain,
    (~ 0 <= skoX | skoX * 3 / (skoX ^ 2 + 3) <= skoX / (1 + skoX * skoX)),
    inference(arith, [18])).

cnf(20, plain, (0 <= skoX), inference(decision, [7])).

cnf(21, plain, (skoX * 3 / (skoX ^ 2 + 3) <= skoX / (1 + skoX * skoX)),
    inference(resolve, [20, 19])).

cnf(22, plain,
    (skoX * 3 / (skoX ^ 2 + 3) * (1 + skoX * skoX) <= skoX |
     1 + skoX * skoX <= 0),
     inference(resolve, [21, leq_left_mul_divide_pos])).

cnf(23, plain,
    (skoX * skoX <= -1 |
     skoX * ((1 + skoX * skoX) * 3) / (skoX ^ 2 + 3) <= skoX),
    inference(arith, [22])).

cnf(24, plain, (skoX * ((1 + skoX * skoX) * 3) / (skoX ^ 2 + 3) <= skoX),
    inference(resolve, [23, 11])).

cnf(25, plain,
    (skoX * ((1 + skoX * skoX) * 3) <= skoX * (skoX ^ 2 + 3) |
     skoX ^ 2 + 3 <= 0), inference(resolve, [24, 0])).

cnf(26, plain, (skoX * (skoX * (skoX * 2)) <= 0 | skoX * skoX <= -3),
    inference(arith, [25])).

cnf(27, plain, (~ skoX * (skoX * (skoX * 2)) <= 0 | skoX * skoX <= -3),
    inference(decision, [7])).

cnf(28, plain, (skoX * skoX <= -3), inference(resolve, [26, 27])).

cnf(29, plain, (~ skoX * skoX <= -3), inference(decision, [7])).

cnf(30, plain, ($false), inference(resolve, [28, 29])).
SZS output end CNFRefutation for atan-problem-9.tptp
```