LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-312

HIERARCHICAL INEQUALITY REASONING

ELISHA P. SACKS

FEBRUARY 1987

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Hierarchical Inequality Reasoning

Elisha Sacks

MIT Laboratory for Computer Science
545 Technology Square, Room 370
Cambridge, MA 02139
U.S.A.
Tel: (617) 253-3447
Net: `ELISHA@ZERMATT.LCS.MIT.EDU`

Abstract

This paper describes a program called BOUNDER that proves inequalities between elementary functions over finite sets of constraints. Previous inequality algorithms perform well on some subset of the elementary functions, but poorly elsewhere. Although complex algorithms perform better than simple ones for most functions, exceptions exist. To overcome these problems, BOUNDER maintains a hierarchy of increasingly complex algorithms. When one fails to resolve an inequality, it tries the next. This strategy resolves more inequalities than any single algorithm. It also performs well on hard problems without wasting time on easier ones. The current hierarchy consists of four algorithms: bounds propagation, substitution, derivative inspection, and iterative approximation. Propagation is an extension of interval arithmetic that takes linear time, but ignores constraints between variables and multiple occurrences of variables. The remaining algorithms consider these factors, but require exponential time. Substitution is a new, provably correct, algorithm for utilizing constraints between variables. An earlier attempt by Brooks does not terminate on all inputs and exploits fewer constraints. The final two algorithms analyze constraints between variables. Inspection examines the signs of partial derivatives. Iteration is based on several earlier algorithms from interval arithmetic.

Keywords: inequality proving, theorem proving, symbolic reasoning

This research was supported (in part) by National Institutes of Health Grant No. R01 LM04493 from the National Library of Medicine and National Institutes of Health Grant No. R24 RR01320 from the Division of Research Resources.

Contents

1 Introduction

This paper describes a program called BOUNDER that proves inequalities between functions over all points satisfying a finite set of constraints: equalities and inequalities between functions. It manipulates *extended elementary functions:* polynomials and compositions of exponentials, logarithms, trigonometric functions, inverse trigonometric functions, absolute values, maxima, and minima. No program can prove all inequalities between arbitrary extended elementary functions, since Richardson [13] proves this problem undecidable. Programs that restrict their attention to a decidable subset, such as the linear functions, are often inadequate, since many important problems fall outside all such subsets. BOUNDER takes a different tack: it sacrifices completeness rather than generality.

BOUNDER tests whether a set of constraints, S, implies an inequality $a \leq b$ between the extended elementary functions a and b by calculating upper and lower bounds for $a-b$ over all points satisfying S. It proves the inequality when the upper bound is negative or zero, refutes it when the lower bound is positive, and fails otherwise. Previous bounding algorithms perform well on some subset of the extended elementary functions, but poorly elsewhere. For this reason, BOUNDER maintains a hierarchy of increasingly complex bounding algorithms. When one fails to resolve an inequality, it tries the next. Although complex algorithms derive tighter bounds than simple ones for most functions, exceptions exist. Hence, BOUNDER's hierarchy of algorithms derives tighter bounds than even its most powerful component. It also performs well on hard problems without wasting time on easier ones.

The purpose of BOUNDER is to resolve inequalities that arise in realistic modeling problems efficiently, not to derive deep theoretical results. It is an engineering utility, rather than a theorem-prover for pure mathematics. For this reason, it only addresses universally quantified inequalities, which make up the majority of practical problems, while ignoring the complexities of arbitrary quantification. BOUNDER enhances the performance of QMR [14], a program that derives the *qualitative prop-*

erties of parameterized functions: signs of the first and second derivatives, discontinuities, singularities, and asymptotes. It also helps explore the qualitative behavior of dynamic systems, such as stability and periodicity. For example, suppose a linear system contains symbolic parameters. Given constraints on the parameters, one can use BOUNDER to reason about the locations of the system's poles and zeroes.

The next section outlines BOUNDER's architecture. Details appear in the two subsequent sections. The final two sections contain a review of literature and conclusions. I argue that most current inequality provers are weak, brittle, or inefficient because they process all inputs uniformly. BOUNDER avoids these shortcomings with its hierarchical strategy.

2 System Architecture

BOUNDER consists of an inequality prover, a context manager, and four bounding algorithms: bounds propagation, substitution, derivative inspection, and iterative approximation. The prover tests whether an inequality between extended elementary functions follows from a set of constraints S. It reduces the original inequality to an equivalent but simpler one by canceling common terms and replacing monotonic functions with their arguments. For example, $x+1 \leq y+1$ simplifies to $x \leq y$, $-x \leq -y$ to $x \geq y$, and $e^x \leq e^y$ to $x \leq y$. It only cancels multiplicands whose signs it can determine by bounds propagation. For a simplified inequality of the form $a \leq b$, it applies each bounding algorithm to the pair $\langle a-b, S\rangle$ in turn. It proves the original inequality if the upper bound is ever negative or zero, refutes it if the lower bound is ever positive, and fails otherwise. The context manager organizes constraint sets in the format required by the bounding algorithms. The bounding algorithms derive upper and lower bounds for a function over all points satisfying a constraint set. I describe the context manager in the next section and the bounding algorithms in the following four sections.

BOUNDER distinguishes between strict and non-strict inequalities. This mechanism consists mainly of careful bookkeeping, based on the properties of continuous functions. For instance, a sum can only attain its upper (lower) bound if both addends can attain theirs. From here on, we will speak only of non-strict inequalities, since BOUNDER handles the strict case analogously.

3 The Context Manager

The context manager (CM) manages constraint sets for the other components. The simplest constraints, called *relational constraints*, are equalities and inequalities between extended elementary functions. CM derives an upper (lower) bound for a variable x from an inequality $L \leq R$ by reformulating it as $x \leq U$ ($x \geq U$) with U free of x. It derives upper and lower bounds for x from an equality $L = R$ by reformulating it as $x = U$. Inequality manipulation may depend on the signs of the expressions involved. For example, the constraint $ax \leq b$ can imply $x \leq b/a$ or $x \geq b/a$ depending on the sign of a. In such cases, CM attempts to derive the relevant signs from other members of the constraint set using bounds propagation. If it fails, it ignores the constraint. (Another possibility would be to create a disjunctive constraint, explained below.) Constraints whose variables cannot be isolated, such as $x \leq 2^x$, are ignored as well. CM gathers the bounds that it derives from a set of relational constraints into a *simple context.* The number of variables in a constraint is linear in its length and each variable requires linear time to isolate. Isolation may require deriving the signs of all the subexpressions in the constraint. Theorem 1 implies that this process takes linear time. All told, processing each constraint requires quadratic time in its length. Subsequent complexity results exclude this time.

The context manager also handles *propositional constraints,* boolean combinations of relational constraints. A set of propositional constraints is equivalent to the conjunction of its members. CM eliminates negation from this conjunction by replacing every instance of $a \neq b$ with $a < b \vee b < a$, every instance of $\neg(a \leq b)$ with

x	VAR-LB$_S(x)$	VAR-UB$_S(x)$	LOWER$_S(x)$	UPPER$_S(x)$
a	1	∞	1	$-4/b$
b	-2	0	$\max\{-2, -4/a, c\}$	$\min\{0, c\}$
c	$-\infty$	∞	b	b

Table 1: Sample bounds for $S = \{a \geq 1,\ b \leq 0,\ b \geq -2,\ ab \geq -4,\ c = b\}$

$b < a$, and so on. It recasts the resulting formula in disjunctive normal form as a disjunction of conjunctions of relational constraints. CM creates a simple context for each disjunct and collects the results into a *disjunctive context.*

The bounding algorithms and the inequality prover reduce disjunctive contexts to simple ones. The upper bound of a function in a disjunctive context is the maximum of its upper bounds in all disjuncts and the lower bound is the minimum of its lower bounds. For example, the constraint $x \leq -1 \vee x \geq 2$ implies a lower bound of 1 for x^2. Similarly, an inequality holds in a disjunctive context iff it holds in all disjuncts. The remainder of this paper deals solely with relational constraints and simple contexts, hereafter abbreviated to constraints and contexts.

Two pairs of functions form the interface between the context manager and the inequality prover and bounding algorithms. Given a variable x and a constraint set S, the functions VAR-LB$_S(x)$ and VAR-UB$_S(x)$ return the maximum of x's numeric lower bounds in S and the minimum of its numeric upper bounds. The functions LOWER$_S(x)$ and UPPER$_S(x)$ return the maximum over all lower bounds, both symbolic and numeric, and the minimum over all upper bounds. Both VAR-LB and LOWER derive lower bounds for x, whereas both VAR-UB and UPPER derive upper bounds. However, LOWER and UPPER produce tighter bounds then VAR-LB and VAR-UB because they take symbolic constraints into account. Examples of these functions appear in Table 1. All four functions run in constant time once the contexts are constructed.

4 Bounding Algorithms

This section contains the details of the bounding algorithms. Each algorithm derives tighter bounds than its predecessor, but takes more time. Each invokes all of its predecessors for subtasks, except that derivative inspection never calls bounds propagation. The bounding algorithms define the extended elementary functions on the extended real numbers in the standard fashion, that is $1/\pm\infty = 0$, $\log 0 = -\infty$, $2^\infty = \infty$, and so on. Throughout this paper, "number" refers to an extended real number.

4.1 Bounds Propagation

The bounds propagation algorithm bounds a compound function by bounding its components recursively and combining the results. For example, the upper bound of a sum is the sum of the upper bounds of its addends. The recursion terminates when it reaches numbers and variables. Numbers are their own bounds, whereas VAR-LB and VAR-UB bound variables. Figure 1 contains the bounds propagation algorithm, $\text{BP}_S(e)$, for a function e over a set of constraints, S. One can represent e as an expression in its variables $x_1, \ldots, x_n$ or as a function $e(x)$ of the vector $x = (x_1, \ldots, x_n)$. From here on, these forms are used interchangeably. The notations lb_e and ub_e abbreviate $\text{LB}_S(e)$ and $\text{UB}_S(e)$. The bounds for exponentials appear in Figure 2. Figure 3 contains the upper bounds for trigonometric functions. The lower bounds are obtained by replacing max with min, ∞ with $-\infty$, TRIG-UB with TRIG-LB, and line 1.1 with

$$1.1' \quad \exists n \in \mathcal{Z}\ lb_a < (2n - \tfrac{1}{2})\pi < ub_a \quad -1$$

and by interchanging all instances of lb and ub that appear in the second column.

The correctness and complexity of BP are summarized in the theorem:

	e is	lb_e	ub_e
1	a number	e	e
2	a variable	$\text{VAR-LB}_S(e)$	$\text{VAR-UB}_S(e)$
3	$a+b$	$lb_a + lb_b$	$ub_a + ub_b$
4	ab	$\min\{lb_a lb_b,\ lb_a ub_b,\ ub_a lb_b,\ ub_a ub_b\}$	$\max\{lb_a lb_b,\ lb_a ub_b,\ ub_a lb_b,\ ub_a ub_b\}$
5	a^b	$\text{EXPT-LB}_S(a,b)$	$\text{EXPT-UB}_S(a,b)$
6	$\min\{a,b\}$	$\min\{lb_a, lb_b\}$	$\min\{ub_a, ub_b\}$
7	$\max\{a,b\}$	$\max\{lb_a, lb_b\}$	$\max\{ub_a, ub_b\}$
8	$\log a$	$\log lb_a$	$\log ub_a$
9	$\|a\|$		
	9.1 $lb_a < 0 < ub_a$	0	$\max\{-lb_a, ub_a\}$
	9.2 else	$\min\{\|lb_a\|, \|ub_a\|\}$	$\max\{\|lb_a\|, \|ub_a\|\}$
10	trigonometric	$\text{TRIG-LB}_S(e)$	$\text{TRIG-UB}_S(e)$

Figure 1: The $\text{BP}_S(e)$ Algorithm

	Case	$\text{EXPT-LB}_S(a,b)$	$\text{EXPT-UB}_S(a,b)$
1	$lb_a > 0$	$e^{lb_b \log a}$	$e^{ub_b \log a}$
2	$b = \frac{p}{q}$ with p, q integers		
	2.1 p, q odd and positive	$[lb_a]^b$	$[ub_a]^b$
	2.2 p, q odd and $ub_a < 0$	$[ub_a]^b$	$[lb_a]^b$
	2.3 p even	$e^{lb_b \log \|a\|}$	$e^{ub_b \log \|a\|}$
	2.4 else	$-\infty$	∞
3	else	$-\infty$	∞

Figure 2: Bounding Algorithms for Exponentials

	e is	TRIG-UB$_{S}(e)$
1	$\sin a$	
	1.1 $\exists n \in \mathcal{Z}\ lb_a < (2n + \frac{1}{2})\pi < ub_a$	1
	1.2 else	$\max\{\sin lb_a, \sin ub_a\}$
2	$\cos a$	TRIG-UB$_{S}(\sin(a + \frac{\pi}{2}))$
3	$\tan a$	
	3.1 $\exists n \in \mathcal{Z}\ lb_a < (n + \frac{1}{2})\pi < ub_a$	∞
	3.2 else	$\tan ub_a$
4	$\arcsin a$	
	4.1 $lb_a \geq -1 \wedge ub_a \leq 1$	$\arcsin ub_a$
	4.2 else	∞
5	$\arccos a$	
	5.1 $lb_a \geq -1 \wedge ub_a \leq 1$	$\arccos lb_a$
	5.2 else	∞
6	$\arctan a$	$\arctan ub_a$

Figure 3: Upper bounds for Trigonometric Functions

Theorem 1 *For any extended elementary function $e(x)$ and set of constraints S, bounds propagation derives numbers lb_e and ub_e satisfying*

$$\forall x.\mathit{satisfies}(x, S) \Rightarrow lb_e \le e(x) \le ub_e \qquad (1)$$

in time proportional to e's length.

Proof: The proof is by induction on e's length. First, consider correctness. Numbers and variables are the only inputs of length 1. Numbers satisfy condition (1) identically and variables satisfy it by the definitions of VAR-LB and VAR-UB. Suppose condition (1) holds for inputs of length less than n and let e be of length n. BP bounds e by combining the bounds of its components, a and b, in one of steps 3–10. The correctness of the bounds on a and b follows from the inductive hypothesis, since they have length less than n. It remains to verify that the combination rules yield valid bounds. Straightforward algebraic manipulations, performed in full by Moore [11], confirm steps 3,4,6,7 and 9. Step 8 follows from the fact that $\log x$ increases monotonically in x, with the provision that undefined LB values represent $-\infty$ and undefined UB values, ∞. The remaining steps, 5 and 10, call the exponential and trigonometric bounding algorithms respectively.

Step 1 of the exponential bounding algorithm uses the identity $a^b = e^{b \log a}$ for $a \ge 0$ along with the monotonicity of the exponential function. The inductive hypothesis does not establish the correctness of the bounds on $b \log a$ because it has length n. Instead, we must apply the arguments for steps 4 and 8 of BP to the bounds of a and b. Steps 2.1 and 2.2 are valid because the function a^b respectively increases and decreases monotonically in a for those choices of b. In step 2.3 the equality $a^b = |a|^b$ holds, so the proof of step 1 applies. Finally, steps 2.4 and 3 yield valid bounds vacuously.

The correctness of the trigonometric bounding algorithms follows from the properties of piecewise monotonic functions. Suppose an interval A partitions into a set of subintervals D on which a function f decreases monotonically and a set I on which f increases monotonically. The maximum (minimum) of f on A occurs at the right (left) endpoint of some interval in I or at the left (right) endpoint of some interval in D. When specialized to the individual trigonometric functions, this result implies the correctness of their bounds. This completes the correctness proof.

Next, consider the time-complexity of BP, $t(n)$. Inputs of length 1, numbers and variables, take constant time. A unary function of length n takes time $t(n-1)$ to calculate the bounds of its argument plus constant overhead. A binary function of length n takes time $t(i)+t(n-i-1)$ to calculate the bounds of its arguments, with i the length of its first argument, plus constant overhead. Induction proves that $t(n)$ is of order kn, with k the maximum overhead required for any step of BP. ∎

BP achieves linear time-complexity by ignoring constraints among variables or multiple occurrences of a variable in an expression. It derives excessively loose bounds when these factors prevent all the constituents of an expression from varying independently over their ranges. For instance, the constraint $a \leq b$ implies that $a-b$ cannot be positive. Yet given only this constraint, BP derives an upper bound of ∞ for $a-b$ by adding the upper bounds of a and $-b$, both ∞. As another example, when no constraints exist, the joint occurrence of x in the constituents of x^2+x implies a global minimum of $-1/4$. Yet BP deduces a lower bound of $-\infty$ by adding the lower bounds of x^2 and x, 0 and $-\infty$. Subsequent bounding algorithms derive optimal bounds for these examples. Substitution analyzes constraints among variables and the final two algorithms handle multiple occurrences of variables. All three obtain better results than BP, but pay an exponential time-complexity price.

4.2 Substitution

The substitution algorithm constructs bounds for an expression by replacing some of its variables with their bounds in terms of the other variables. Substitution exploits all solvable constraints, whereas bounds propagation limits itself to constraints between variables and numbers. In our previous example, substitution derives an upper bound of 0 for $a-b$ from the constraint $a \leq b$ by bounding a from above with b, that is $a-b \leq b-b=0$. Substitution is performed by the algorithms $\textsc{sup}_S(e, H)$ and $\textsc{inf}_S(e, H)$, which calculate upper and lower bounds on e over the constraint set S in terms of the variable set H. When H is empty, the bounds reduce to numbers.

Figures 4 and 5 contain the SUP function and its auxiliary, SUPP. One obtains the INF function from Figure 4 by interchanging all occurrences of SUP and INF and replacing SUPP with INFF, UPPER with LOWER, EXPT-SUP with EXPT-INF, TRIG-SUP with TRIG-INF, and max with min. Also, step 9 is replaced with:

```
9'   $|a|$
     9.1  $\textsc{inf}_S(a,H) < 0 < \textsc{sup}_S(a,H)$    $0$
     9.2  else                                               $\min\{|\textsc{inf}_S(a,H)|, |\textsc{sup}_S(a,H)|\}$
```

The INFF function is obtained from Figure 5 by replacing SUPP with INFF and UB with LB. The auxiliary functions EXPT-SUP, EXPT-INF, TRIG-SUP, and TRIG-INF are derived from the exponential and trigonometric bounding algorithms (Figures 2 and 3) by replacing $\textsc{ub}_S(a)$ with $\textsc{sup}_S(a, H)$, replacing $\textsc{lb}_S(a)$ with $\textsc{inf}_S(a, H)$, and so on for b. The expression $v(e)$ denotes the variables contained in e. In the remainder of this section, we will focus on SUP. INF is analogous.

In step 1, SUP calculates the upper bounds of numbers and of variables included in H. It analyzes a variable, x, not in H by constructing an intermediate bound

$$B = \textsc{sup}_S(\textsc{upper}_S(x), H \cup \{x\}) \tag{2}$$

for x and calling SUPP to derive a final bound. If possible, SUPP derives an upper bound for x in H directly from the inequality $x \leq B$. Otherwise, it applies bounds propagation to B. For instance, the inequality $x \leq 1-x$ yields a bound of $1/2$, but $x \leq x^2-1$ does not provide an upper bound, so SUPP returns $\textsc{ub}_S(x^2-1)$.

SUP exploits constraints among variables to improve its bounds on sums and products. If b contains variables that a lacks, but which have bounds in a's variables,

	e is	$\mathrm{SUP}_S(e, H)$
1	$v(e) \subseteq H$	e
2	a variable	$\mathrm{SUPP}_S(e, \mathrm{SUP}_S(\mathrm{UPPER}_S(e), H \cup \{e\}))$
3	$a + b$	
	3.1 $v(b) - v(a) \subseteq H$	$\mathrm{SUP}_S(a, H) + \mathrm{SUP}_S(b, H)$
	3.2 else	$\mathrm{SUP}_S\,(a + \mathrm{sup}_S(b, H \cup v(a)), H)$
4	ab	
	4.1 $\mathrm{LB}_S(a) \geq 0$	
	4.1.1 $v(b) - v(a) \subseteq H$	$\max\{\mathrm{SUP}_S(a, H)\mathrm{SUP}_S(b, H), \mathrm{INF}_S(a, H)\mathrm{SUP}_S(b, H)\}$
	4.1.2 else	$\mathrm{SUP}_S(a\,\mathrm{SUP}_S(b, H \cup v(a)), H)$
	4.2 $\mathrm{UB}_S(a) \leq 0$	
	4.2.1 $v(b) - v(a) \subseteq H$	$\max\{\mathrm{SUP}_S(a, H)\mathrm{INF}_S(b, H), \mathrm{INF}_S(a, H)\mathrm{INF}_S(b, H)\}$
	4.2.2 else	$\mathrm{SUP}_S(a\,\mathrm{INF}_S(b, H \cup v(a)), H)$
	4.3 else	$\max\{\mathrm{SUP}_S(a, H)\mathrm{SUP}_S(b, H), \mathrm{SUP}_S(a, H)\mathrm{INF}_S(b, H), \mathrm{INF}_S(a, H)\mathrm{SUP}_S(b, H), \mathrm{INF}_S(a, H)\mathrm{INF}_S(b, H)\}$
5	a^b	$\mathrm{EXPT\text{-}SUP}_S(a, b, H)$
6	$\min\{a, b\}$	$\min\{\mathrm{SUP}_S(a, H), \mathrm{SUP}_S(b, H)\}$
7	$\max\{a, b\}$	$\max\{\mathrm{SUP}_S(a, H), \mathrm{SUP}_S(b, H)\}$
8	$\log a$	$\log \mathrm{SUP}_S(a, H)$
9	$\lvert a \rvert$	$\max\{\lvert\mathrm{INF}_S(a, H)\rvert, \lvert\mathrm{SUP}_S(a, H)\rvert\}$
10	trigonometric	$\mathrm{TRIG\text{-}SUP}_S(e, H)$

Figure 4: The $\mathrm{SUP}_S(e, H)$ Algorithm

	case	$\text{SUPP}_S(x, B)$
1	$x \notin v(B)$	B
2	$B = rx + A;\ r \in \Re,\ v \notin v(A)$	
	2.1 $r \geq 1$	∞
	2.2 $r < 1$	$\frac{A}{1-r}$
3	$B = \min\{C, D\}$	$\min\{\text{SUPP}_S(x, C), \text{SUPP}_S(x, D)\}$
4	$B = \max\{C, D\}$	$\max\{\text{SUPP}_S(x, C), \text{SUPP}_S(x, D)\}$
5	else	$\text{UB}_S(B)$

Figure 5: The $\text{SUPP}_S(x, B)$ Algorithm

SUP constructs an intermediate upper bound, U, for $a+b$ or ab by replacing b with these bounds. A recursive application of SUP to U produces a final upper bound. (Although not indicated explicitly in Figure 4, these steps are symmetric in a and b.) For example, given the constraints $c \geq 1$, $d \geq 1$, and $cd \leq 4$, SUP derives an intermediate bound of $3c/4$ for $c - 1/d$ by replacing $-1/d$ with $-c/4$, its upper bound in c. This bound is derived as follows:

$$\text{SUP}(-\frac{1}{d}, \{c\}) = -\text{INF}(\frac{1}{d}, \{c\}) = -\frac{1}{\text{SUP}(d, \{c\})} = -\frac{1}{4/c} = -\frac{c}{4} \qquad (3)$$

SUP uses the recursive call

$$\text{SUP}(c, \{\}) = \text{SUPP}(c, \text{SUP}(\frac{4}{d}, \{c\})) = \text{SUPP}(c, \frac{4}{\text{INF}(d, \{c\})}) = \text{SUPP}(c, 4) = 4 \qquad (4)$$

to derive a final bound of 3 for $c - 1/d$. If a and b have the same variables, SUP bounds $a + b$ and ab by recursively bounding a and b and applying bounds propagation to the results.

Substitution produces valid bounds for any input, variable set, and constraints. Although the proof requires a joint induction on both SUP and INF, I will present only the SUP half, since the INF half is completely analogous. At first glance, it seems possible that SUP fails to terminate on some inputs. Step 2 bounds a variable by invoking SUP recursively on a more complex expression, its UPPER. Similarly, steps 3.1, 4.1.2, and 4.2.2 bound their inputs by recursing on intermediate values of unknown complexity. The following lemma rules out infinite recursion and allows us to proceed to the correctness theorem:

Lemma 2 *The algorithms* $\text{SUP}_S(e, H)$ *and* $\text{INF}_S(e, H)$ *terminate for any input* e*, variable set* H*, and constraint set* S*.*

Proof: The auxiliary functions EXPT-SUP and TRIG-SUP terminate if SUP does, since they call it at most twice and apply a few extended elementary functions to the results. Irregardless of SUP, SUPP always terminates because each recursive call, in steps 2 and 3, decreases the length of b, while the other steps terminate directly. It remains to show that SUP makes only finitely many recursive calls on any input. Let T be the invocation tree of recursive calls made by SUP for some e, H and S. Each node n is labeled with the input e_n and variable set H_n of its corresponding SUP call. A node n *takes step* k if e_n matches case k of SUP. Let V denote the union of e's variables with those appearing in S. The variables of every e_n form a subset of V. Consider some path p through T. Each node l of p that takes step 2 adds some member of V to the set H_m of its successor m. No step removes elements from H, so this variable appears in all subsequent H_n. At most $|V|$ nodes of p can take step 2 before some node takes step 1 and terminates the path.

Let q be the sub-path of p beginning with the successor of the last node that takes step 2. By the argument above, p is finite iff q is finite. Let the *free set* of a node n denote the variables of e_n not in H_n. Suppose n's predecessor, m, takes step 3.2, 4.1.2, or 4.2.2 and n corresponds to the outer SUP of that step. The expression e_m has the form $a + b$ or ab with b containing some variable not in $v(a) \cup H_m$, whereas e_n contains only variables from $v(a) \cup H_n$. This implies that n's free set is a proper subset of m's, since H_m equals H_n. None of SUP's steps increases the free set of a node over that of its predecessor, except 2. Path q contains no instances of step 2, so it can contain only finitely many such successors of nodes that take steps 3.2, 4.1.2, and 4.2.2 before it reaches a node with an empty free set and terminates at step 1. Let r be the sub-path of q beginning with the successor of the last node that takes one of these three steps. Each of r's nodes takes a step that reduces the size of its input, so r must terminate. This proves q, and hence p, finite. ∎

The following theorem establishes the correctness of substitution:

Theorem 3 *For every extended elementary function e, variable set H, and constraint set S, the expressions $i = \text{INF}_S(e, H)$ and $s = \text{SUP}_S(e, H)$ satisfy the conditions:*

$$i \textit{ and } s \textit{ are expressions in } H \tag{5}$$

$$\forall x.\textit{satisfies}(x, S) \Rightarrow i(x) \leq e(x) \leq s(x) \tag{6}$$

Proof: I will prove that every finite invocation tree for SUP satisfies conditions (5) and (6) by induction on tree depth. This proves the theorem, since both algorithms always produce finite trees by Lemma 2. Trees of depth 1 return bounds of e, which satisfy condition (5) trivially and (6) identically. Suppose the theorem holds for trees of depth less than n and consider a SUP tree T of depth $n > 1$. The correctness of all recursive calls follows from the inductive hypothesis. It remains to prove that the root node satisfies both conditions.

Step 1 cannot occur since n is greater than 1. By inductive hypothesis and by the definition of the UPPER function, the second argument to SUPP in step 2 is an expression in $H \cup \{e\}$ that bounds e. This implies directly that step 1 of SUPP satisfies conditions (5) and (6). Step 2.1 satisfies them by elementary algebra, step 2.2 satisfies them vacuously, and step 5, by Theorem 1. Since the other steps satisfy the conditions, the definitions of min and max guarantee that steps 3 and 4 do too.

Steps 3.2, 4.1.2, and 4.2.2 of SUP generate an intermediate upper bound, u, by bounding one of e's constituents. Their final result, $\text{SUP}(u, H)$, satisfies conditions 5 and 6 by the inductive hypothesis. The validity of u follows from the facts

$$\begin{array}{lrcl} \text{(step 3.2)} & k \leq l & \Rightarrow & a+k \leq a+l \\ \text{(step 4.1.2)} & a \geq 0 \wedge k \leq l & \Rightarrow & ak \leq al \\ \text{(step 4.2.2)} & a \leq 0 \wedge k \geq l & \Rightarrow & ak \leq al \end{array}$$

and the inductive hypothesis. SUP's remaining steps, including the auxiliary functions EXPT-SUP and TRIG-SUP, apply the combination rules of BP, appearing in Figures 1–3, to the SUP and INF of e's constituents. Their results satisfy condition (5) because the constituents do. Condition (6) holds by Theorem 1. ∎

Substitution utilizes constraints among variables to improve on the bounds of BP, but ignores constraints among multiple occurrences of variables. It performs identically to BP on the example of $x^2 + x$, deriving a lower bound of $-\infty$. Yet that bound is overly pessimistic because no value of x minimizes both addends simultaneously. The last two bounding algorithms address this shortcoming.

4.3 Derivative Inspection

The derivative inspection algorithm, DI, calculates bounds for a function, f, from the signs of its partial derivatives. If the partial derivative of f with respect to x is non-negative over an interval $[l, r]$, then f's minimum and maximum on the interval, for any choice of its other variables, occur at l and r respectively. If the partial derivative is non-positive, the maximum occurs at l and the minimum at r. In the example from the previous section, the derivative of $x^2 + x$ is non-positive on $[-\infty, -1/2]$ and non-negative on $[-1/2, \infty]$. This information enables DI to derive an optimal lower bound of $-1/4$ for $x^2 + x$, as opposed to INF's bound of $-\infty$.

Before describing DI in detail, I will introduce some notation. Let S be a set of constraints and x a vector $(x_1, \ldots x_n)$ of variables. The *range* of x_i in S is the interval

$$X_i = [\text{INF}_S(x_i, \{\}), \text{SUP}_S(x_i, \{\})] \tag{7}$$

and the *range* of x in S is the Cartesian product $X = X_1 \times \cdots \times X_n$ of its components' ranges. Theorem 3 implies that

$$\forall x.\mathit{satisfies}(x, S) \Rightarrow x \in X \tag{8}$$

One can split X in dimension i by choosing a point p_i in X_i and forming two supersets of S, one with the additional constraint $x_i \le p_i$ and the other with $x_i \ge p_i$. One can collapse X to a point in dimension i by adding the constraint $x_i = p_i$ to S. If X is collapsed in all directions, it reduces to a point.

DI bounds a function $f(x)$ by partitioning X into subregions on which $f(x)$ is monotonic. The maximum upper bound and minimum lower bound over all subregions bound $f(x)$ from above and below on X. These bounds are valid over the set of points whose components satisfy S, since all such points belong to X by equation 8. DI splits X in each dimension i by dividing X_i into maximal intervals of the following types:

$$\begin{array}{ll} \mathit{decreasing} & \mathrm{SUP}(\frac{\partial f(x)}{\partial x_i}, \{\}) \le 0 \\ \mathit{increasing} & \mathrm{INF}(\frac{\partial f(x)}{\partial x_i}, \{\}) \ge 0 \\ \mathit{unknown} & \text{neither of the above.} \end{array} \tag{9}$$

If $f(x)$ increases in x_i on a subregion, it must attain its minimum over that subregion when x_i equals its left endpoint and its maximum when x_i equals its right endpoint. If $f(x)$ decreases, the endpoints are reversed. Either way, DI can collapse the subregion to a point in dimension i. Figure 6 shows the results of this procedure for the function $x^2 - y^2$ on the region $([-1, 1], [-1, 1])$. The upper and lower bounds, u_i and l_i, are found directly for each interval because no intervals are *unknown*.

Derivative inspection takes time proportional to the number of regions into which f's domain splits. For this reason, it only applies to functions whose partial derivatives all have finitely many zeroes in X. When every X_i consists solely of *increasing* and *decreasing* intervals, derivative inspection yields optimal bounds directly, since all regions reduce to points. Otherwise, one must use another bounding algorithm

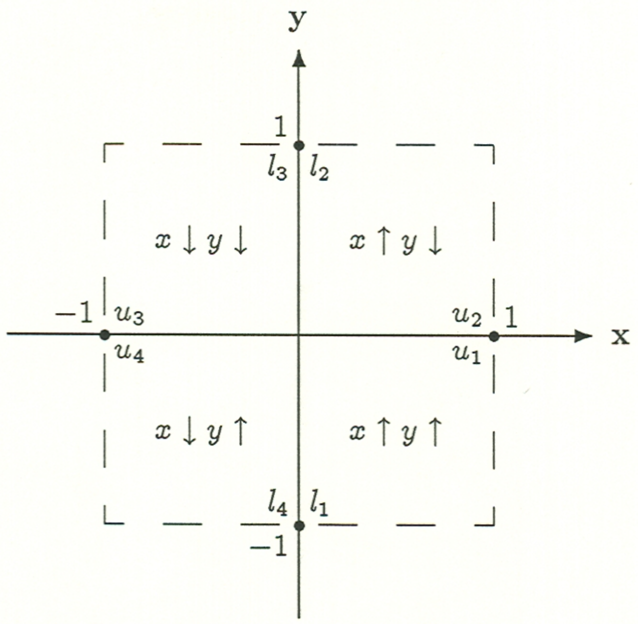
y
1
l_3
l_2
$x \downarrow y \downarrow$
$x \uparrow y \downarrow$
-1
u_3
u_2
1
x
u_4
u_1
$x \downarrow y \uparrow$
$x \uparrow y \uparrow$
l_4
l_1
-1

Figure 6: Derivative inspection for $x^2 - y^2$

to calculate bounds on the non-trivial subregions. This two-step approach generally yields tighter bounds than applying the second algorithm directly on f's entire domain, since the subregions are smaller and often reduce to points along some dimensions.

4.4 Iterative Approximation

Iterative approximation, like derivative inspection, reduces the errors in bounds propagation and substitution caused by multiple occurrences of variables. Instead of bounding a function over its entire range directly, it subdivides the regions under consideration and combines the results. Intuitively, BP's choice of multiple worst case values for a variable causes less damage on smaller regions because all these values are less far apart. Figure 7 illustrates this idea for the function $x^2 - x$ on the interval $[0, 1]$. Part (a) demonstrates that BP derives an overly pessimistic lower bound on $[0, 1]$ because it minimizes both $-x$ and x^2 independently. Part (b) shows that this factor is less significant on smaller intervals: the maximum of the two

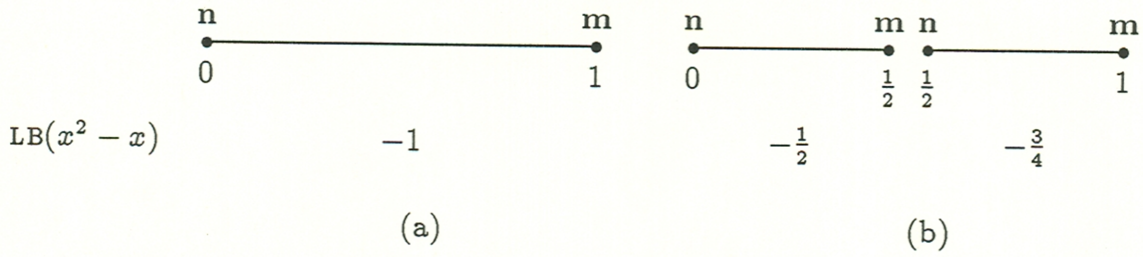

Figure 7: Illustration of iterative approximation for $x^2 - x$ on $[0, 1]$. The symbols m and n mark the values of x that minimize $-x$ and x^2 respectively.

lower bounds, $-3/4$, is a tighter bound for $x^2 - x$ on $[0, 1]$ than that of part (a). One can obtain arbitrarily tight bounds by constructing sufficiently fine partitions.

Iterative approximation generalizes interval subdivision to multivariate functions and increases its efficiency, using ideas from Moore [11] and Asaithambi et at. [1]. It converges to the true bounds of any continuously differentiable function on a bounded domain. Let X_0 denote the range of the vector x in constraint set S, as defined in the previous section. The IA algorithm, shown in Figure 8, calculates an upper bound for a continuously differentiable function, $f(x)$, on a finite region, X_0, by iteratively dividing and shrinking X_0. It derives a lower bound by negating the upper bound of $-f(x)$. By Theorem 3, these bounds are valid over the set of points whose components satisfy S.

Let c denote the collection of subsets of X_0 produced by DI on input $f(x)$ and S. The least upper bound, LUB, of f on X_0 equals the maximum LUB of f over c, as explained in the previous section. IA uses a modified version of the UB function, MUB, to estimate the LUB on regions. Initially, it pairs each member of c with its MUB value and sorts the pairs in decreasing order of MUB. The first MUB value in the list is an upper bound for f on X_0. At each iteration, IA splits the first region, Z, by bisecting an *unknown* component (defined in equation (9)) and inserting the resulting subregions into the list in proper order.

It would be inefficient to split Z in an *increasing* or *decreasing* direction, i, since the optimal value of x_i is known. In fact, the range of x_i consists of its optimal value. Derivative inspection imposes this condition on the initial partition and IA maintains it by collapsing each subregion of Z to its left endpoint in every *decreasing* direction and to its right endpoint in every *increasing* direction. Also, it would be pointless to consider regions on which f cannot attain its maximum. One such case occurs when the MUB value, b', of an entry $\langle Z', b' \rangle$ is less than f's value at the middle of Z_1 or Z_2. To improve efficiency, IA deletes these entries.

The MUB function is defined as follows:

$$\text{MUB}(X) = f(m(X)) + \text{UB}_X \left(\sum_{i \in U(X)} (x_i - m_i(X)) \frac{\partial f(x)}{\partial x_i} \right) \tag{10}$$

The set $U(X)$ contains all dimensions in which $f(x)$ has *unknown* direction in X and the vector $m(X)$ denotes the middle of X, whose component $m_i(X)$ equals the midpoint of X_i. The MLB function is defined analogously, with LB replacing UB in equation (10). Moore [11] proves that these functions bound f on all subsets of X_0, that is

$$\forall X \subseteq X_0 \, \forall x \in X . \text{MLB}(X) \leq f(x) \leq \text{MUB}(X) \tag{11}$$

This implies that the LUB of f on X lies between its MUB and MLB.

Let us define the width of the interval $[a, b]$ as $b - a$ and the width of the region X, $w(X)$, as the maximum over the widths of its components. Moore proves that

$$\forall X \subseteq X_0 . \text{MUB}(X) - \text{MLB}(X) \leq Lw(X) \tag{12}$$

with L a constant. This result enables us to prove that IA terminates.

Lemma 4 *Let $x = (x_1, \ldots, x_n)$ have finite range X in S. For every positive ϵ and function $f(x)$ continuously differentiable on X,* IA *terminates within*

$$M = \left(\frac{2L}{\epsilon} \right)^n \prod_{i=1}^{n} w(X_i) \tag{13}$$

iterations, where L is the constant of equation (12).

```
1. Apply DI to $f$ and $S$, pair each resulting region with its MUB value, sort the
   pairs in decreasing order of MUB, and set *list* to the sorted list.
2. $\langle Z, b\rangle \leftarrow head(list)$; $list \leftarrow tail(list)$   {The MUB of $b$ is $Z$.}
3. If $U(X) = \emptyset$ or $b - \mathrm{MLB}(Z) \leq \epsilon$ return $b$.
4. Choose an $i$ in $U(Z)$ which maximizes $w(Z_i)$.
5. Calculate $Z_1$ and $Z_2$ by bisecting $Z$ in dimension $i$ and collapsing the
   results in *increasing* and *decreasing* directions.
6. For $j = 1, 2$ do
   6.1 Delete from *list* every pair $\langle Z', b'\rangle$ for which $b' < f(m(Z_j))$.
   6.2 Insert $\langle Z_j, \mathrm{MUB}(Z_j)\rangle$ into *list* in proper order.
7. Go to step 2.
```

Figure 8: Algorithm $\mathrm{IA}(f(x), S, \epsilon)$

Proof: First, suppose *list* contains only the region X after step 1. Each interval X_k can be decomposed by a sequence of bisections into at most

$$\frac{2Lw(X_k)}{\epsilon} \tag{14}$$

subintervals without splitting some interval narrower than ϵ/L. All told, X can be decomposed into at most M regions without splitting such an interval. By the choice of i in step 4, IA can only split an interval narrower than ϵ/L if the region Z is narrower than ϵ/L. This never happens because such a Z satisfies the second disjunct of step 3, terminating the iteration, by equation (12). Hence, at most M iterations can occur.

Next, suppose *list* contains regions $X^1, \ldots, X^k$ after step 1. Applying the argument above to each region shows that IA terminates within

$$\left(\frac{2L}{\epsilon}\right)^n \sum_{j=1}^{k} \prod_{i=1}^{n} w(X_i^j) \tag{15}$$

iterations. The jth addend in this equation equals the volume of X^j, whereas the product in equation (13) equals the volume of X. The sum cannot exceed the product because the X^j are pairwise disjoint subsets of X. ∎

The following theorem establishes the correctness of iterative approximation:

Theorem 5 *Let x have finite range in S. For every positive ϵ and continuously differentiable function $f(x)$, iterative approximation calculates a number, b, satisfying*

$$\forall x.satisfies(x, S) \Rightarrow 0 \leq b - f(x) \leq \epsilon \tag{16}$$

Proof: It follows from Lemma 4 that IA terminates at step 3. If it halts because $U(Z)$ is empty, b is the LUB of f on Z because Z reduces to a point. If IA halts because $b - \text{MLB}(Z)$ is less than ϵ, then b is at most ϵ greater than the LUB by equation (11). Either way, b approximates the LUB of f on Z, which equals the LUB on X, within ϵ. This implies condition (16) by equation (8). ∎

For some applications and functions, other termination tests perform better than step 3 of Figure 8. Asaithambi et al. [1] prove that the test

$$\max\{\text{MUB}(Z_1), \text{MUB}(Z_2)\} \leq b \tag{17}$$

generates optimal bounds for rational functions. If one only needs to establish that f is not greater than a particular bound, b_0, then $b \leq b_0$ is a sufficient condition for termination. This case arises when the inequality prover proves that $c \leq d$ by establishing that $c - d$ has an upper bound of 0.

5 Related Work

In this section, I discuss, in order of increasing generality, existing programs that derive bounds and prove inequalities. As one would expect, the broader the domain of functions and constraints, the slower the program. The first class of systems bounds linear functions subject to linear constraints. Valdés-Pérez [18] analyzes sets of *simple linear inequalities* of the form $x - y \geq n$ with x and y variables and n a number. He uses graph search to test their consistency in cv time for c constraints and v variables. Malik and Binford [10] and Bledsoe [2] check sets of general linear constraints for consistency and calculate bounds on linear functions over consistent sets of constraints. Both methods require exponential time.[1] The former uses the Simplex algorithm, whereas the latter introduces preliminary versions of BOUNDER's substitution algorithms. Bledsoe defines SUP, SUPP, INF, and INFF for linear functions and constraints and proves the linear versions of Lemma 2 and Theorem 3. In fact, these algorithms produce *exact* bounds, as Shostak [15] proves.

The next class of systems bounds nonlinear functions, but allows only range constraints. All resemble BOUNDER's bounds propagation and all stem from Moore's [11] *interval arithmetic*. Moore introduces the rules for bounding elementary functions on finite domains by combining the bounds of their constituents. His algorithm takes linear time in the length of its input. Bundy [7] implements an *interval package* that resembles BP closely. It generalizes the combination rules of interval arithmetic to any function that has a finite number of extrema. If the user specifies the sign of a function's derivative over its domain, Bundy's program can perform interval arithmetic on it. Unlike BOUNDER's derivative inspection algorithm, it cannot derive this information for itself. Many other implementations of interval arithmetic exist, some in hardware.

Moore also proposes a simple form of iterative approximation, which Skelboe [17], Asaithambi et al. [1], and Ratschek and Rokne [12, ch. 4] improve. BOUNDER's iterative approximation algorithm draws on all these sources.

Simmons [16] handles functions and constraints containing numbers, variables, and the four arithmetic operators. He augments interval arithmetic with simple algebraic simplification and inequality information. For example, suppose x lies in the interval $[-1, 1]$. Simmons simplifies $x - x$ to 0, whereas interval arithmetic

[1]The simplex algorithm often performs better in practice. Also, a polynomial alternative exists.

produces the range $[-2, 2]$. He also deduces that $x \leq z$ from the constraints $x \leq y$ and $y \leq z$ by finding a path from x to z in the graph of known inequalities. The algorithm is linear in the total number of constraints. Although more powerful than BOUNDER's bounds propagation, Simmons's program is weaker than substitution. For example, it cannot deduce that $x^2 \geq y^2$ from the constraints $x \geq y$ and $y \geq 0$.

Brooks [6, sec. 3] extends Bundy's SUP and INF to nonlinear functions and argues informally that Lemma 2 and Theorem 3 hold for his algorithms. This argument must be faulty because his version of $\text{SUP}_H(e, \{\})$ recurses infinitely when e equals $x + 1/x$ or $x + x^2$, for instance. Brooks's program only exploits constraints among the variables of sums $rx + B$ and of products $x^n B$ with r real, x a variable of known sign, B an expression free of x, and n an integer. In other cases, it adds or multiplies the bounds of constituents, as in steps 3.1, 4.1.1, 4.2.1, and 4.3 of BOUNDER's SUP (Figure 4). These overly restrictive conditions rule out legitimate substitutions that steps 3.2, 4.1.2, and 4.2.2 permit. For example, BOUNDER can deduce that $1/x - 1/y \geq 0$ from the constraints $y \geq x$ and $x \geq 1$, but Brooks's algorithm cannot. On some functions and non-empty sets H, his algorithm makes recursive calls with H empty. This produces needlessly loose bounds and sometimes causes an infinite recursion.

Bundy and Welham [9, sec. 4] derive upper bounds for a variable x from an inequality $L \leq R$ by reformulating it as $x \leq U$ with U free of x. If U contains a single variable, they try to find its global maximum, M, by inspecting the sign of its second derivative at the zeroes of its first derivative. When successful, they bound x from above with M. Lower bounds and strict inequalities are treated analogously. Bundy and Welham use a modified version of the PRESS equation solver [9,8] to isolate x. As discussed in section 3, inequality manipulation depends on the signs of the expressions involved. When this information is required, they use Bundy's interval package to try to derive it. The complexity of this algorithm is unclear, since PRESS can apply its simplification rules repeatedly, possibly producing large intermediate expressions. BOUNDER contains both steps of Bundy and Welham's bounding algorithm: its context manager derives bounds on variables from constraints, while its derivative inspection algorithm generalizes theirs to multivariate functions. PRESS may be able to exploit some constraints that BOUNDER ignores because it contains a stronger equation solver than does BOUNDER.

The final class of systems consists of theorem provers for predicate calculus that treat inequalities specially. These systems focus on general theorem proving, rather than problem-solving. They handle more logical connectives than BOUNDER, including disjunction and existential quantification, but fewer functions, typically just addition. Bledsoe and Hines [4] derive a restricted form of resolution that contains a theory of dense linear orders without endpoints. Bledsoe et al. [5] prove this form of resolution complete. Finally, Bledsoe et al. [3] extend a natural deduction system with rules for inequalities. Although none of these authors discuss complexity, all their algorithms must be at least exponential.

6 Conclusions

Current inequality reasoners are weak, brittle, or inefficient because they treat all inputs uniformly. Interval arithmetic systems, such as Bundy's and Simmons's, run quickly, but generate exceedingly pessimistic bounds when dependencies exist among components of functions. These dependencies are caused by constraints among variables or multiple occurrences of a variable, as discussed in Section 4.1. The upper bound of $a - b$ given $a \leq b$ demonstrates the first type, while the lower bound of $x^2 + x$ given no constraints demonstrates the second. Each of the remaining systems is brittle because it takes only one type of dependency into account. Iterative approximation, suggested by Moore, and derivative inspection, performed in the univariate case by Bundy and Welham, address the second type of dependency, but ignore the first. Conversely, substitution, used (in a limited form) by Brooks and Simmons, exploits constraints among variables, while ignoring multiple occurrences of variables. All these systems are inefficient because they apply a complex algorithm to every input without trying a simple one first.

BOUNDER overcomes the limitations of current inequality reasoners with its hierarchical strategy. It uses substitution to analyze dependencies among variables and derivative inspection and iterative approximation to analyze multiple occurrences of variables. Together, these techniques cover far more cases than any single-algorithm system. Yet unlike those systems, BOUNDER does not waste time applying overly powerful methods to simple problems. It tries bounds propagation, which has linear time-complexity, before resorting to its other methods. An inequality reasoner like BOUNDER should be an important component of future general-purpose symbolic algebra packages.

References

[1] N. S. Asaithambi, Shen Zuhe, and R. E. Moore.
On computing the range of values.
Computing, 28:225–237, 1982.

[2] W. W. Bledsoe.
A new method for proving certain Presburger formulas.
In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 15–21, 1975.

[3] W. W. Bledsoe, Peter Bruell, and Robert Shostak.
A prover for general inequalities.
In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 66–69, 1979.

[4] W. W. Bledsoe and Larry M. Hines.
Variable elimination and chaining in a resolution-based prover for inequalities.
In *Proceeding of the fifth conference on automated deduction*, Springer-Verlag, Les Arcs, France, July 1980.

[5] W. W. Bledsoe, K. Kunen, and R. Shostak.
Completeness results for inequality provers.
ATP 65, University of Texas, 1983.

[6] Rodney A. Brooks.
Symbolic reasoning among 3-d models and 2-d images.

Artificial Intelligence, 17:285–348, 1981.

[7] Alan Bundy.
A generalized interval package and its use for semantic checking.
ACM Transactions on Mathematical Software, 10(4):397–409, December 1984.

[8] Alan Bundy and Bob Welham.
Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation.
Artificial Intelligence, 16(2):189–211, May 1981.

[9] Alan Bundy and Bob Welham.
Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation.
D.A.I. Working Paper 55, University of Edinburgh, Depatment of Artificial Intelligence, May 1979.

[10] J. Malik and T. Binford.
Reasoning in time and space.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 343–345, August 1983.

[11] Ramon E. Moore.
Methods and Applications of Interval Analysis.
SIAM Studies in Applied Mathematics, SIAM, Philadelphia, 1979.

[12] H. Ratschek and J. Rokne.
Computer Methods for the Range of Functions.
Halsted Press: a division of John Wiley and Sons, New York, 1984.

[13] D. Richardson.
Some unsolvable problems involving elementary functions of a real variable.
Journal of Symbolic Logic, 33:511–520, 1968.

[14] Elisha P. Sacks.
Qualitative mathematical reasoning.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 137–139, 1985.

[15] Robert E. Shostak.
On the SUP-INF method for proving Presburger formulas.
Journal of the ACM, 24:529–543, 1977.

[16] Reid Simmons.
"Commonsense" arithmetic reasoning.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 118–124, American Association for Artificial Intelligence, August 1986.

[17] S. Skelboe.
Computation of rational functions.
BIT, 14:87–95, 1974.

[18] Raúl Valdés-Pérez.
Spatio-temporal reasoning and linear inequalities.
AIM 875, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1986.