Arithmetic Decision Procedures: a simple introduction

Michael Norrish

Abstract

Fourier-Motzkin variable elimination is introduced as a complete method for deciding linear arithmetic inequalities over $\mathbb{R}$. It is then shown how this method can be extended to also work over $\mathbb{Z}$, giving the Omega Test [2].

1 Introduction

All of the techniques described in this note are examples of a technique known as *quantifier elimination*. This technique assumes that the truth or falsity of formulas with no quantifiers is easy to determine. For example, $2 < 3 \times 4$ is true, while $\frac{2}{3} < \frac{1}{4} \wedge 2 + 2 = 4$ is false, and these facts are easy to check. If it is possible to transform a formula with quantifiers ($\forall$, $\exists$) into an *equivalent* formula containing fewer quantifiers, then it is clear that this process can be repeated until there are no quantifiers left. The result of this transformation will have an easily ascertained truth value, and this will also be the truth value (i.e., whether or not it is valid) of the input formula.

All of the formulas to be discussed are of the form given in Figure 1. Note that formulas can not include multiplication of a variable by another variable. Some of this syntax is redundant and can be expressed in terms of other formulas. For example, $\geq$ and $>$ will always be replaced with equivalent formulas involving $\leq$ and $<$ respectively. Similarly, $x = y$ might be replaced with $x \leq y \wedge y \leq z$ (sometimes it makes more sense to leave equalities intact). Finally, $\forall x.\ P(x)$ will always be replaced by $\neg\exists x.\ \neg P(x)$.

The quantifier elimination procedures to be described here work by eliminating the quantifier in a formula of the form $\exists x.\ P(x)$, where $P(x)$ does not include any other quantifiers, but may include free variables. Thus, the procedure works by starting with the innermost quantifiers and gradually working its way out and up through the formula.

$$\begin{array}{lcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \ \mid\ \textit{formula} \vee \textit{formula} \ \mid \\
 & & \neg\textit{formula} \ \mid\ \exists \textit{var}.\, \textit{formula} \ \mid\ \forall \textit{var}.\, \textit{formula} \ \mid \\
 & & \textit{term relop term} \\
\textit{term} & ::= & \textit{numeral} \ \mid\ \textit{term} + \textit{term} \ \mid\ -\textit{term} \ \mid\ \textit{numeral} * \textit{term} \ \mid\ \textit{var} \\
\textit{relop} & ::= & < \ \mid\ \leq \ \mid\ = \ \mid\ \geq \ \mid\ > \\
\textit{var} & ::= & x \ \mid\ y \ \mid\ z \ldots
\end{array}$$

Figure 1: Grammar defining decidable formulas. Valid values for *numeral* will depend on the domain. For formulas over $\mathbb{R}$, allow any real number; for formulas over $\mathbb{Z}$, allow any integer.

2 Fourier-Motzkin Variable Elimination

Over the real numbers $\mathbb{R}$, with a and b positive, non-zero coefficients, the following are true:

$$\begin{array}{lcl}
(\exists x.\, c \leq ax \wedge bx \leq d) & \equiv & bc \leq ad \\
(\exists x.\, c < ax \wedge bx \leq d) & \equiv & bc < ad \\
(\exists x.\, c \leq ax \wedge bx < d) & \equiv & bc < ad \\
(\exists x.\, c < ax \wedge bx < d) & \equiv & bc < ad
\end{array}$$

Proof of the second equivalence (proofs of the other three equivalences are similar): Left $\Rightarrow$ Right: assume there is an x such that $c < ax \wedge bx \leq d$. Then $cb < abx$ and $abx \leq ad$. By transitivity, $cb < ad$. Right $\Rightarrow$ Left: assume $bc < ad$. Then $c < a(\frac{d}{b})$ and $b(\frac{d}{b}) \leq d$. So, $\frac{d}{b}$ can be taken as the x that satisfies the LHS.

Next, extend the results above so that the bodies of the existential formulas need not consist of only two constraints:

$$\begin{array}{c}
\exists x.\, (\bigwedge_h c_h \leq a_h x) \wedge (\bigwedge_i c_i < a_i x) \wedge (\bigwedge_j b_j x \leq d_j) \wedge (\bigwedge_k b_k x < d_k) \\
\equiv \\
(\bigwedge_{h,j} b_j c_h \leq a_h d_j) \wedge (\bigwedge_{h,k} b_k c_h < a_h d_k) \wedge \\
(\bigwedge_{i,j} b_j c_i < a_i d_j) \wedge (\bigwedge_{i,k} b_k c_i < a_i d_k)
\end{array} \tag{1}$$

This is the process of taking all possible pairings of lower bounds ($c < ax$, $c \leq ax$) with upper bounds and calculating their consequences according to the first set of four equivalences. The proof of this equivalence is by induction. (The base case is a formula with one inequality of each type. Successive inductions establish that it is possible to have any finite number of constraints of a given type.) Note that if there are no upper bounds, or no lower bounds, then the formula on the right simplifies to true.

This equivalence (1) is the basis of a quantifier elimination procedure over $\mathbb{R}$:

1. For any formula $\exists x.\, P(x)$, where $P(x)$ is quantifier-free, convert $P(x)$ to disjunctive normal form.

2. Using the equivalence $(\exists x. P(x) \vee Q(x)) \equiv (\exists x.\ P(x)) \vee (\exists x.\ Q(x))$ "push" the quantifier down over all of the disjunctions.
3. Each quantifier now has scope over a set of conjunctions. If any of these conjuncts doesn't mention the bound variable x "move" it to the side by applying the equivalence $(\exists x. P(x) \wedge Q) \equiv (\exists x. P(x)) \wedge Q$.
4. Convert all of the relational operators under each quantifier so that only $<$ and $\leq$ remain.
5. Isolate x in each conjunct and ensure it has a positive coefficient.
6. Apply the elimination theorem above to each quantifier.

2.1 Efficiency

This procedure is not efficient. Having to convert to DNF introduces potentially exponential cost, and the presence of alternating quantifiers will require repeated conversions. Consider, $\forall x. \exists y. P(x,y)$. After eliminating y, the formula will be $\forall x. P'(x)$, and $P'(x)$ will be in DNF. To then eliminate x, the formula will be converted to $\neg\exists x. \neg P'(x)$. Converting $\neg P'(x)$ to DNF will exhibit worst case behaviour.

Ignoring the conversion to DNF (as is reasonable if the problem only involves one sort of quantifier, and comes already converted to DNF), eliminating a quantifier can turn n constraints into $\frac{n^2}{4}$ constraints. For a problem of m quantifiers, and n constraints, the final number of constraints to be checked could be as many as

$$\frac{n^{2^m}}{4^m}$$

2.2 Applications

The method as it stands is easy to adjust so that it returns a satisfying assignment (if one exists) for a purely existential problem (i.e., an input of the form $\exists\vec{x}.\ P$, where P includes no other quantifiers). When the problem has been reduced to one of just one variable, a satisfying value for this variable is anything between the greatest of its lower bounds and the least of its upper bounds. This value can then be substituted back into the formula that held at the previous stage of the process (when there were two variables). After this substitution, the formula is of just one variable and a value can be calculated for that variable also. In this way, all of the existentially bound variables can have values found for them.

This much provides a simplistic constraint satisfaction solver for linear arithmetic problems. Simple optimisation problems can be solved by using alternating quantifiers. For example, if the problem is to find a satisfying assignment for $P(\vec{x},z)$ while maximising z, proceed as follows:

- Check $\exists\vec{x}, z.\ P(\vec{x},z)$. If this is not valid, then there is no solution at all.

- Otherwise, check $\exists z.\ (\exists\vec{x}.\,P(\vec{x},z)) \wedge (\forall z'.\,z' \leq z \vee \neg\exists\vec{x}.\,P(\vec{x},z'))$. If this is invalid, then there is no maximum value for z (it can be made as large as desired). If it is valid, then the same process that finds satisfying assignments for variables in purely existential goals will find one for the outermost z in this formula.
- With a maximal value for z found, substitute this into the orignal existential formula, and solve for the remaining $\vec{x}$.

The use of alternating quantifiers ensures that this method will be extremely inefficient. This method is not recommended as a solution for optimisation problems. Rather it is a demonstration of the expressive power of a language with alternating quantifiers.

3 Quantifier Elimination for $\mathbb{Z}$

Over the integers, things are slightly simpler because $x < y$ is equivalent to $x + 1 \leq y$, so one need only consider one type of relational operator. Unfortunately, the core theorem (1) for eliminating existential quantifiers is not a theorem over $\mathbb{Z}$. This arises because of the discreteness of the integers. Consider

$$\begin{array}{rl} & \exists x : \mathbb{Z}.\ 5 \leq 2x < 6 \\ \equiv & \exists x.\ 5 \leq 2x \wedge 2x + 1 \leq 6 \\ \equiv & \exists x.\ 5 \leq 2x \leq 5 \\ \not\equiv & 10 \leq 10 \end{array}$$

The proofs of the elimination equivalences fail because the RHSes do not imply the LHSes. The fact that the LHSes do still imply the RHSes (these arguments were a consequence only of transitivity) can be used to implement an incomplete check for unsatisfiability.

This procedure is for purely existential formulas: act as if the core quantifier elimination theorem were true, eliminating quantifiers and reducing the input formula to true or false. If the final result is false, then the original formula must have been invalid. (If the final result is true, then no conclusion can be drawn.) Because of the negations introduced when converting universal quantifiers to existentials, this method can also be used to show universal formulas valid.

This method is easy to implement and is used in a number of interactive theorem-proving systems, such as ACL2, Coq, HOL and Isabelle. This method is also the first phase of the *Omega Test* [2].

Phase 1 of the Omega Test exploits the theorem

$$(\exists x : \mathbb{Z}.\ c \leq ax \wedge bx \leq d) \Rightarrow bc \leq ad$$

Phase 2 exploits the similar (imagine a or b equal to 1)

$$(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x : \mathbb{Z}.\ c \leq ax \wedge bx \leq d) \qquad (2)$$

This phase uses the theorem above to repeatedly "eliminate" quantifiers (extended, as before, to handle multiple constraints at once). If the eventual result of the eliminations is true, then the original formula is valid.

The proof of (2) is by contradiction. If the conclusion of the implication is false, then there are no multiples of ab occurring between bc and ad. The antecedent of the implication implies that $bc \leq ad$ (a and b are both positive). Let j be the greatest integer such that $abj < bc$. Then, $abj < bc \leq ad < ab(j+1)$.

Dividing a out of the top constraint leaves $d < b(j+1)$. This is the same as $1 \leq b(j+1) - d$. Multiplying through by a again, conclude $a \leq ab(j+1) - ad$. Similarly, $b \leq bc - abj$. Summing the constraints, obtain $a + b \leq ab - ad + bc$, equivalently $ad - bc \leq ab - a - b$. But this contradicts the antecedent, $(a-1)(b-1) \leq ad - bc$.

Phase 1 can show an existential formula to be false, and Phase 2 can show it to be true. With both phases working in concert, many problems can be decided (Pugh [2] claims most of his didn't need more than these two phases). But the combination of the two phases is not complete. It can't handle alternating quantifier problems at all (both phases rely on being able to eliminate quantifiers all the way to true or false), and even purely existential formulas may slip through both phases.

In the worst case, the following theorem may need to be used:

Theorem 1 (Pugh, 1992) *Let $L(x)$ be a conjunction of lower bounds on x, indexed by i, of the form $c_i \leq a_i x$, with a_i positive. Similarly, let $U(x)$ be a set of upper bounds on x, indexed by j, of the form $b_j x \leq d_j$, with b_j positive. Let m be the maximum of all the b_js. Then*

$$\begin{array}{rcc} (\exists x.\, L(x) \wedge U(x)) & \equiv & (\bigwedge_{i,j} (a_i - 1)(b_j - 1) \leq a_i d_j - b_j c_i) \\ & & \vee \\ & & \bigvee_i \bigvee_{k=0}^{\left\lfloor \frac{m a_i - a_i - m}{m} \right\rfloor} \exists x.\, (a_i x = c_i + k) \wedge L(x) \wedge U(x) \end{array}$$

To prove the equivalence, it suffices to show that:

- the first disjunct on the right implies the LHS (done above, as this is the same as the formula used in Phase 2 of the test);
- the other disjuncts also imply the LHS (trivial, as any x satisfying a disjunct on the right will also satisfy the original formula); and that
- $(\exists x.\, L(x) \wedge U(x)) \wedge \neg(\bigwedge_{i,j} (a_i - 1)(b_j - 1) \leq a_i d_j - b_j c_i) \;\Rightarrow$ $\bigvee_i \bigvee_{k=0}^{\left\lfloor \frac{m a_i - a_i - m}{m} \right\rfloor} \exists x.\, (a_i x = a_i + k) \wedge L(x) \wedge U(x)$
  Let x be the witness to the first assumption. The second assumption means that there exist a, b, c and d such that

  $$ad - bc \leq ab - b - a \qquad (3)$$

  These values occur in constraints from L and U, so $bx \leq d$ and $c \leq ax$. Multiplying the former through by a gives $abx \leq ad$, so in conjunction with (3)

  $$\begin{array}{rlcl} & abx & \leq & cb + ab - b - a \\ \Rightarrow & b(ax - c) & \leq & ab - b - a \\ \Rightarrow & ax - c & \leq & \left\lfloor \frac{ab-b-a}{b} \right\rfloor \end{array}$$

All of the b coefficients are $\leq m$, so

$$\left\lfloor \frac{ab - b - a}{b} \right\rfloor \leq \left\lfloor \frac{ma - a - m}{m} \right\rfloor$$

There is now enough information to pick the appropriate disjunct from the RHS. The a_i is a and k is $ax - a$.

Though all except the first of the disjuncts on the RHS of Theorem 1 have an existential quantifier, this quantifier can be eliminated immediately, thanks to the presence of the additional equality constraint (see [2] for details). Thus, Theorem 1 really does represent a quantifier elimination result.

4 Final Remark

The decidability of linear arithmetic over $\mathbb{Z}$ was proved by Presburger in 1929. Formulas of the form given in Figure 1 constitute the language often called Presburger Arithmetic. Presburger's proof of decidability provides another procedure for deciding problems of the sort discussed above. Cooper's algorithm [1] is a version of his method more suited to actual implementation (and one which does not require conversion to DNF.)

References

[1] D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99, New York, 1972. American Elsevier.

[2] William Pugh. The Omega Test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, August 1992.