Appeared at LICS'04

Deciding Quantifier-Free Presburger Formulas Using Parameterized Solution Bounds

Sanjit A. Seshia Randal E. Bryant
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213, USA
{Sanjit.Seshia, Randy.Bryant}@cs.cmu.edu

Abstract

Given a formula Φ in quantifier-free Presburger arithmetic, it is well known that, if there is a satisfying solution to Φ, there is one whose size, measured in bits, is polynomially bounded in the size of Φ. In this paper, we consider a special class of quantifier-free Presburger formulas in which most linear constraints are separation (difference-bound) constraints, and the non-separation constraints are sparse. This class has been observed to commonly occur in software verification problems. We derive a new solution bound in terms of parameters characterizing the sparseness of linear constraints and the number of non-separation constraints, in addition to traditional measures of formula size. In particular, the number of bits needed per integer variable is linear in the number of non-separation constraints and logarithmic in the number and size of non-zero coefficients in them, but is otherwise independent of the total number of linear constraints in the formula. The derived bound can be used in a decision procedure based on instantiating integer variables over a finite domain and translating the input quantifier-free Presburger formula to an equi-satisfiable Boolean formula, which is then checked using a Boolean satisfiability solver. We present empirical evidence indicating that this method can greatly outperform other decision procedures.

1. Introduction

Presburger arithmetic [30] is defined as the first-order theory of the structure $\langle \mathbb{N}, 0, 1, \leqslant, + \rangle$, where $\mathbb{N}$ denotes the set of natural numbers. The satisfiability problem for Presburger arithmetic is decidable, but of super-exponential worst-case complexity [14]. Fortunately, for many applications, such as in program analysis (e.g., [31]) and hardware verification (e.g., [8]), the quantifier-free fragment suffices.

A formula Φ in quantifier-free Presburger arithmetic (QFP) is constructed by combining linear constraints with Boolean operators ($\wedge$, $\vee$, $\neg$). Formally, the ith constraint is of the form $\sum_{j=1}^{n} a_{i,j} x_j \geq b_i$, where the coefficients and the constant terms are integer constants and the variables $x_1, x_2, \ldots, x_n$ are integer-valued[1]. In this paper, we are concerned with the satisfiability problem for QFP, viz., that of finding a valuation of the variables such that Φ evaluates to **true**. The NP-hardness of this problem follows from a straightforward encoding of the 3SAT problem as a 0-1 integer linear program. That it is moreover in NP, and hence NP-complete, can be concluded from the result that integer linear programming is in NP [6, 36, 19, 26].

Thus, if there is a satisfying solution to a QFP formula, there is one whose size, measured in bits, is polynomially bounded in the problem size. Problem size is traditionally measured in terms of the parameters m, n, $\log a_{\max}$, and $\log b_{\max}$, where m is the total number of constraints in the formula, n is the number of variables, and $a_{\max} = \max_{(i,j)} |a_{i,j}|$ and $b_{\max} = \max_i |b_i|$ are upper bounds on the absolute values of coefficients and constant terms respectively.

The above result suggests the following approach to checking the satisfiability of a QFP formula Φ:

1. Compute the polynomial bound S on solution size.
2. Search for a satisfying solution to Φ in the bounded

1 While Presburger arithmetic is defined over $\mathbb{N}$, we interpret the variables over $\mathbb{Z}$ as it is general and more suitable for applications. It is straightforward to translate a formula with integer variables to one where variables are interpreted over $\mathbb{N}$, and vice-versa, by adding (linearly many) additional variables or constraints.

space $\{0, 1, \ldots, 2^S - 1\}^n$.

This approach has been successfully applied to highly restricted sub-classes of QFP, such as *equality logic* [28] and *separation logic* [9], and is termed as *finite instantiation*. The basic idea is to translate Φ to a Boolean formula by encoding each integer variable as a vector of Boolean variables (a "symbolic bit-vector") of length S. The resulting Boolean formula is checked using a Boolean satisfiability (SAT) solver. This approach leverages the dramatic advances in SAT solving made in recent years (e.g., [24, 16]). It is straightforward to extend the approach to additionally handle the theory of uninterpreted functions and equality, by using, for example, Ackermann's technique of eliminating function applications [1].

However, a naïve implementation of a decision procedure based on finite instantiation fails for QFP formulas encountered in practice. The problem is that the bound on solution size, S, is $O(\log m + \log b_{\max} + m[\log m + \log a_{\max}])$. In particular, the presence of the $m \log m$ term means that, for practical problems involving hundreds of linear constraints, the Boolean formulas generated are likely to be too large to be decided by present-day SAT solvers.

In this paper, we explore the above finite instantiation-based approach to deciding QFP formulas, but with a focus on formulas generated in software verification. It has been observed, by us and others, that formulas from this domain have:

1. *Mainly Separation Constraints:* Of the m constraints, $m - k$ are *separation* constraints, where $k \ll m$. Separation constraints, also called *difference-bound constraints*, are of the form $x_i - x_j \bowtie b_t$ or $x_i \bowtie b_t$, where b_t is an integer constant, and $\bowtie \in \{>, \geq, =, <, \leq\}$.
2. *Sparse Structure:* The k non-separation constraints are sparse, with at most w variables per constraint, where w is "small". We will refer to w as the *width* of the constraint.

Pratt [29] observed that most inequalities generated in program verification are separation constraints. More recently, the authors of the theorem prover Simplify observed in the context of the Extended Static Checker for Java (ESC/Java) project that "the inequalities that occur in program checking rarely involve more than two or three terms" [13]. We have performed a study of formulas generated in various recent software verification projects: the Blast project at Berkeley [17], the Magic project at CMU [10], the Wisconsin Safety Analyzer (WiSA) project [37], and the software upgrade checking project at MIT [22]. The results of this study, indicated in Table 1, support the afore-mentioned observations regarding the "sparse, mostly separation" nature of constraints in QFP formulas. To our knowledge, no previous decision procedure for QFP has attempted to exploit this problem structure.

We make the following novel contributions in this paper:

- We derive bounds on solutions for QFP formulas, not only in terms of the traditional parameters m, n, $a_{\max}$, and $b_{\max}$, but also in terms of k and w. In particular, we show that the worst-case number of bits required per integer variable is linear in k, but only logarithmic in w. Unlike previously derived bounds, ours is not in terms of the total number of constraints m.
- We use the derived bounds in a sound and complete decision procedure for QFP based on finite instantiation, and present empirical evidence that our method can greatly outperform other decision procedures.

Related Work. There has been much work on deciding quantifier-free Presburger arithmetic; we present a brief discussion here and refer the reader to a recent survey [15] for more details. Recent techniques fall into three categories:

- The first class comprises procedures targeted towards solving conjunctions of constraints, with disjunctions handled by enumerating terms in a disjunctive normal form (DNF). Examples include the Omega test [31] and solvers based on other integer linear programming techniques. The drawback of these methods is the need to enumerate the potentially exponentially many terms in the DNF representation.
- The second set of methods attempt to remedy this problem by instead relying on modern SAT solving strategies. The approach works as follows. A Boolean abstraction of the QFP formula Φ is generated by replacing each linear constraint with a corresponding Boolean variable. If the abstraction is unsatisfiable, then so is Φ. If not, the satisfying assignment (model) is checked for consistency with the theory of quantifier-free Presburger arithmetic, using a ground decision procedure for conjunctions of linear constraints. Assignments that are inconsistent are excluded from later consideration by adding a "lemma" to the Boolean abstraction. The process continues until either a consistent assignment is found, or all (exponentially many) assignments have been explored. Examples of decision procedures in this class that have some support for QFP include CVC [2, 3] and ICS [12]. The ground decision procedures used by provers in this class employ a combination framework such as the Nelson-Oppen architecture for cooperating decision procedures [25] or a Shostak-like combination method [34, 33]. These methods are only defined for combining disjoint theories. In order to exploit the mostly-separation struc-

Project	Maximum Fraction of Non-Separation Constraints	Maximum Width of a Non-Separation Constraint
Blast	0.0276	6
Magic	0.0032	2
MIT	0.0087	3
WiSA	0.0054	4

Table 1. Linear Arithmetic Constraints in Software Verification are Mostly Separation Constraints. For each software verification project, the maximum fraction of non-separation constraints is shown, as well as the maximum width of a non-separation constraint, where the maximum is taken over all formulas in the set. The Blast formulas were generated from device drivers written in C, the Magic formulas from an implementation of `openssl` written in C, the MIT formulas from Java programs, and the WiSA formulas were generated in the checking of format string vulnerabilities.

ture of a formula, one approach could be to combine a decision procedure for a theory of separation constraints with one for a theory of non-separation constraints, but this needs an extension of the combination methods that applies to these non-disjoint theories.

- The final class of methods are based on finite automata theory (e.g., [38, 15]). The basic idea is to construct a finite automaton corresponding to the input QFP formula Φ, such that language accepted by the automaton consists of the binary encodings of satisfying solutions of Φ. According to a recent experimental evaluation with other methods [15], these techniques are better than others at solving formulas with very large coefficients, but do not scale well with the number of variables and constraints.[2]

The approach we present in this paper is distinct from the categories mentioned above. In particular, the following unique features differentiate it from previous methods:

- It is the first finite instantiation method and the first procedure based on translating a QFP formula to SAT in a single step. The clear separation between the translation and the SAT solving allows us to leverage future advances in SAT solving far more easily than other SAT-based procedures.
- It is the first technique, to the best of our knowledge, that exploits the structure of formulas commonly encountered in software verification.

Outline of the paper. The rest of this paper is organized as follows. In Section 2, we discuss background material on bounds on satisfying solutions of integer linear programs. An integer linear program (ILP) is a conjunction of linear constraints, and hence is a special kind of QFP formula. The bounds for QFP follow directly from those for ILPs. Our main theoretical results are presented in Sections 3–5. Section 3 gives bounds for ILPs for the case of $k = 0$, when all constraints are separation constraints. In Section 4, we compute a bound for ILPs for arbitrary k. In Section 5, we show how our results extend to arbitrary QFP formulas. We report on experimental results in Section 6, and conclude in Section 7.

2 Note that automata-based techniques can handle full Presburger arithmetic, not just the quantifier-free fragment.

2. Background

In this section, we define the integer linear programming problem formally and state the previous results on bounding satisfying solutions of ILPs. A more detailed discussion on the steps outlined in Section 2.1 can be found in reference books on ILP (e.g. [32, 27]).

2.1. Preliminaries

Consider a system of m linear constraints in n integer-valued variables:

$$A\mathbf{x} \geq b \tag{1}$$

Here A is an $m \times n$ matrix with integral entries, b is a $m \times 1$ vector of integral entries, and $\mathbf{x}$ is a $n \times 1$ vector of integer-valued variables. A satisfying solution to system (1) is an evaluation of $\mathbf{x}$ that satisfies (1).

In system (1), the entries in $\mathbf{x}$ can be negative. We can constrain the variables to be non-negative by adding a dummy variable x_0 that refers to the "zero value," replacing each original variable x_i by $x'_i - x_0$, and then adjusting the coefficients in the matrix A to get a new constraint matrix A'

and the following system:[3]

$$\begin{aligned} A'\mathbf{x}' &\geq b \\ \mathbf{x}' &\geq 0 \end{aligned} \tag{2}$$

Here the system has $n' = n + 1$ variables, and $\mathbf{x}' = [x'_1, x'_2, \ldots, x'_n, x_0]^T$. A' has the structure that $a'_{i,j} = a_{i,j}$ for $j = 1, 2, \ldots, n$ and $a'_{i,n+1} = -\sum_{j=1}^{n} a_{i,j}$. Note that the last column of A' is a linear combination of the previous n columns. It is easy to show that system (1) has a solution if and only if system (2) has one.

Finally, adding surplus variables to the system, we can rewrite system (2) as follows:

$$\begin{aligned} A''\mathbf{x}'' &= b \\ \mathbf{x}'' &\geq 0 \end{aligned} \tag{3}$$

where $A'' = [A| - I_m]$ is an $m \times (n' + m)$ integer matrix formed by concatenating A with the negation of the $m \times m$ identity matrix I_m.

For convenience we will drop the primes, referring to A'' and $\mathbf{x}''$ simply as $\mathbf{A}$ and $\mathbf{x}$. Rewriting system (3) thus, we get

$$\begin{aligned} A\mathbf{x} &= b \\ \mathbf{x} &\geq 0 \end{aligned} \tag{4}$$

Hereafter we will use the definition in (4). Let $a_{\max} = \max_{(i,j)} |a_{i,j}|$ and $b_{\max} = \max_t |b_t|$ be upper bounds on the absolute values of entries of A and b respectively.

2.2. Previous Results

The results of this paper build on results obtained by Borosh, Treybig, and Flahive [6, 5] on bounding the solution of systems of the form (4). We state their result in the following theorem:

Theorem 1 *Consider the augmented matrix $[A|b]$ of dimension $m\times(n'+m+1)$. Let Δ be the maximum of the absolute values of all minors of this augmented matrix. Then, the system (4) has a satisfying solution if and only if it has one with all entries bounded by $(n + 2)\Delta$.*

However, note that the determinant of a matrix can be more than exponential in the dimension of the matrix [7]. In the case of the Borosh-Flahive-Treybig result, it means that Δ can be as large as $\frac{\mu^m (m+1)^{(m+1)/2}}{2^m}$, where $\mu = \max(a_{\max}, b_{\max})$.

Papadimitriou [26, 27] also gives a bound of similar size, stated in the following theorem:

Theorem 2 *If the ILP of (4) has a satisfying solution, then it has a satisfying solution where all entries in the solution vector are bounded by $(n' + m)(1 + b_{\max})(m\, a_{\max})^{2m+3}$.*

Papadimitriou's bound implies that we need $O(\log m + \log b_{\max} + m[\log m + \log a_{\max}])$ bits to encode each variable (assuming $n' = O(m)$). The Borosh-Flahive-Treybig bound implies needing $O(m[\log m + \log \mu])$ bits per variable, which is of the same order.

3. Bounds for a System of Separation Constraints

Let us first consider computing solution bounds for an ILP for the case where $k = 0$, i.e., system (4) comprises only of separation constraints.

In this case, the left-hand side of each equation comprises exactly three variables: two variables x_i and x_j where $0 \leq i, j \leq n$ and one surplus variable x_l where $n + 1 \leq l \leq n + m$. The t^{th} equation in the system is of the form $x_i - x_j - x_l = b_t$.

As we noted in Section 2.1, the matrix A can be written as $[A_o|-I_m]$ where A_o comprises the first $n' = n+1$ columns, and I_m is the $m \times m$ identity matrix.

The important property of A_o is that each row has exactly one $+1$ entry and exactly one -1 entry, with all other entries 0. Thus, A_o^T can be interpreted as the node-arc incidence matrix of a directed graph. Therefore, A_o^T is *totally unimodular* (TUM), i.e., every square submatrix of A_o^T has determinant in $\{0, -1, +1\}$ [27]. Therefore, A_o is TUM, and so is $A = [A_o| - I_m]$.

Now, let us consider using the Borosh-Flahive-Treybig bound stated in Theorem 1. This bound is stated in terms of the minors of the matrix $[A|b]$. For the special case of this section, we have the following bound on the size of any minor:

Theorem 3 *The absolute value of any minor of $[A|b]$ is bounded above by $s\, b_{\max}$, where $s = \min(n + 1, m)$.*

Proof:

Consider any minor M of $[A|b]$. Let r be the order of M.

If the minor is obtained by deleting the last column (corresponding to b), then it is a minor of A, and its value is in $\{0, -1, +1\}$ since A is TUM. Thus, the bound of $s\, b_{\max}$ is attained for any non-trivial minor with $s \geq 1$ and $b_{\max} \geq 1$.

Suppose the b column is not deleted.

First, note that the matrix A is of the form $[A_o| - I_m]$ where the rank of A_o is at most $s' = \min(n, m)$. This is because A_o has dimensions $m \times n + 1$, and the last column of A_o,

[3] Note that this procedure can increase the width of a constraint by 1. The statistics in Table 1 shows the width before this procedure is applied, computed from constraints as they appear in the original formulas.

corresponding to the variable x_0, is a linear combination of the previous n columns. (Refer to the construction of system (2) from system (1).)

Next, suppose the sub-matrix corresponding to M comprises p columns from the $-I_m$ part, $r-p-1$ columns from the A_o part, and the one column corresponding to b. Since permuting the rows and columns of M does not change its absolute value, we can permute the rows of M and the columns corresponding to the $-I_m$ part to get the corresponding sub-matrix in the following form:

$$\left[\begin{array}{c|cccc|c} & 0 & \ldots & 0 & -1 & b_{t_1} \\ & 0 & \ldots & -1 & 0 & b_{t_2} \\ A_o & \vdots & \ldots & \vdots & \vdots & \vdots \\ \text{part} & -1 & \ldots & 0 & 0 & b_{t_p} \\ & 0 & \ldots & 0 & 0 & b_{t_{p+1}} \\ & \vdots & \ldots & \vdots & \vdots & \vdots \\ & 0 & \ldots & 0 & 0 & b_{t_r} \end{array}\right]$$

Expanding M along the last column, we get

$$|M| = |b_{t_1} M_1 - b_{t_2} M_2 + b_{t_3} M_3 - \ldots (-1)^{r-1} b_{t_r} M_r|$$

where each M_i is a minor corresponding to a submatrix of A.

However, notice that $M_i = 0$ for all $1 \leq i \leq p$, since each of those minors have an entire column (from the $-I_m$ part) equal to 0. Therefore, we can reduce the right-hand side to the sum of $r-p$ terms:

$$|M| \leq |b_{t_{p+1}} M_{p+1}| + |b_{t_{p+2}} M_{p+2}| + \ldots |b_{t_r} M_r|$$

Notice that, so far, we have not made use of the special structure of A.

Now, observing that A is TUM, $|M_i| \leq 1$ for all i.

$$|M| \leq |b_{t_{p+1}}| + |b_{t_{p+2}}| + \ldots + |b_{t_r}|$$

For all i, $|b_{t_i}| \leq b_{\max}$. Further, since each non-zero M_i can be of order at most s', $r-p \leq s = min(s'+1, m)$.[4] Therefore, we get

$$|M| \leq s\, b_{\max}$$

□

Using the terminology of Theorem 1, we have $\Delta \leq s\, b_{\max}$. Thus, the bound in this case is $(n+2)\, s\, b_{\max}$.

Thus, S, the bound on the number of bits per variable, is

$$\lceil \log(n+2) + \log s + \log b_{\max} \rceil$$

4 We use $s'+1$ and not s' to account for the case where $p = 0$. The minimum with m is taken because $s'+1$ can exceed m but b has only m elements.

Formulas generated from verification problems tend to be overconstrained, so we assume $n < m$. Thus, $s = n+1$, and the bound reduces to $O(\log n + \log b_{\max})$ bits per variable.

Remark. The only property of the A matrix that the proof of Theorem 3 relies on is the totally unimodular (TUM) property. Thus, Theorem 3 would also apply to any system of linear constraints whose coefficient matrix is TUM. Examples of such matrices include *interval* matrices, or more generally *network* matrices. Note that the TUM property can be tested for in polynomial time [32].

4. Bounds for a Sparse System of Mainly Separation Constraints

We now consider the general case for ILPs, where we have k non-separation constraints, each referring to at most w variables.

Without loss of generality, we can reorder the rows of matrix A so that the k non-separation constraints are the top k rows, and the separation constraints are the bottom $m-k$ rows. Reordering the rows of A can only change the sign of any minor of $[A|b]$, not the absolute value. Thus, the matrix $[A|b]$ can be put into the following form:

$$\left[\begin{array}{c|c|c} A_1 & & b_1 \\ \cline{1-1} & -I_m & b_2 \\ A_2 & & \vdots \\ & & b_m \end{array}\right]$$

Here, A_1 is a $k \times n+1$ dimensional matrix corresponding to the non-separation constraints, A_2 is a $m-k \times n+1$ dimensional matrix with the separation constraints, I_m is the $m \times m$ identity corresponding to the surplus variables, and the last column is the vector b.

The matrix comprised of A_1 and A_2 will be referred to, as before, as A_o. Note that each row of A_1 has at most w non-zero entries, and each row of A_2 has exactly one $+1$ and one -1 with the remaining entries 0. Thus, A_2 is TUM.

We prove the following theorem:

Theorem 4 *The absolute value of any minor of $[A|b]$ is bounded above by $s\, b_{\max}\, (a_{\max}\, w)^k$, where $s = \min(n+1, m)$.*

Proof:

Consider any minor M of $[A|b]$, and let r be its order.

As in Theorem 3, if M includes p columns from the $-I_m$ part of A, then we can infer that $r-p \leq s$. (Our proof of this property in Theorem 3 made no assumptions on the form of A_o.)

If M includes the last column b, then as in the proof of Theorem 3, we can conclude that

$$|M| \;\leq\; (r-p)\, b_{\max}\, [\max_{j=1}^{r} |M_j|] \tag{5}$$

where M_j is a minor of A_o.

If M does not include b, then it is a minor of A. Without loss of generality, we can assume that M does not include a column from the $-I_m$ part of A, since such columns only contribute to the sign of the determinant.

So, let us consider bounding a minor M_j of A_o of order r (or $r-1$, if M includes the b column).

Since $A_o = \left[\frac{A_1}{A_2}\right]$, consider expanding M_j, using the standard determinant expansion by minors along the top k rows corresponding to non-separation constraints. Each term in the expansion is (up to a sign) the product of at most k entries from the A_1 portion, one from each row, and a minor from A_2. Since A_2 is TUM, each product term is bounded in absolute value by $a_{\max}^k$. Furthermore, there can be at most w^k non-zero terms in the expansion, since each non-zero product term is obtained by choosing one non-zero element from each of the rows of the A_1 portion of M_j, and this can be done in at most w^k ways.

Therefore, $|M_j|$ is bounded by $(a_{\max}\, w)^k$. Combining this with the inequality (5), and since $r-p \leq s$, we get

$$|M| \;\leq\; s\, b_{\max}\, (a_{\max}\, w)^k$$

which is what we set out to prove. □

Thus, we conclude that $\Delta \leq s\, b_{\max}(a_{\max}\, w)^k$, where $s = \min(n+1, m)$. From Theorems 1 and 4, the solution bound is $(n+2)\Delta$. Thus, S is

$$\lceil \log(n+2) + \log s + \log b_{\max} + k(\log a_{\max} + \log w) \rceil$$

We make the following observations about the bound derived above, assuming as before, that $n < m$, and so $s = n+1$:

- *Dependence on Parameters:* We observe that the bound is linear in k, logarithmic in $a_{\max}$, w, n, and $b_{\max}$. In particular, the bound is not in terms of the total number of linear constraints, m.

- *Worst-case Asymptotic Growth*: In the worst case, $k = m$, $w = n+1$, and $n = O(m)$, and we get the $O(\log m + \log b_{\max} + m[\log m + \log a_{\max}])$ bound of Papadimitriou.

- *Typical-case Asymptotic Growth:* As observed in Section 1, w is typically bounded by a small constant, so the number of bits needed per variable is $O(\log n + \log b_{\max} + k \log a_{\max} + k)$. In many cases, $a_{\max}$ and k are also bounded by a small constant. Thus, S is typically $O(\log n + \log b_{\max})$. This reduces the search space by an exponential factor over using the bound expressed in terms of m.

- *Representing Non-separation Constraints:* There are many ways to represent non-separation constraints and these have an impact on the bound we derive. In particular, it is possible to transform a system of non-separation constraints to one with at most three variables per constraint. For example, the linear constraint $x_1 + x_2 + x_3 + x_4 = x_5$ can be rewritten as:

$$\begin{aligned} x_1 + x_1' &= x_5 \\ x_2 + x_2' &= x_1' \\ x_3 + x_4 &= x_2' \end{aligned}$$

  For the original representation, $k = 1$ and $w = 5$, while for the new representation $k = 3$ and $w = 3$. Since our bound is linear in k and logarithmic in w, the original representation would yield a tighter bound.

  Similarly, one can eliminate variables with coefficients greater than 1 in absolute value by introducing new variables; e.g., $2x$ is represented as $x + x'$ with an additional separation constraint $x = x'$. This can be used to adjust w, $a_{\max}$, and n so that the overall bound is reduced.

The derived bound only yields benefits in the case when the system has few non-separation constraints which themselves are sparse. In this case, we can instantiate variables over a finite domain that is much smaller than that obtained without making any assumptions on the structure of the system.

5. Bounds for Arbitrary Quantifier-Free Presburger Formulas

We now return to the original goal of this paper, that of finding a solution bound for an arbitrary QFP formula Φ. Suppose that Φ has m linear constraints $\phi_1, \phi_2, \ldots, \phi_m$, of which $m-k$ are separation constraints, and n variables $x_1, x_2, \ldots, x_n$. As before, we assume that each non-separation constraint has at most w variables, $a_{\max}$ is the maximum over the absolute values of coefficients $a_{i,j}$ of variables, and $b_{\max}$ is the maximum over the absolute values of constants b_i appearing in the constraints.

We prove the following theorem.

Theorem 5 *If Φ is satisfiable, there is a solution to Φ that is bounded by $(n+2)\Delta$ where*

$$\Delta = s\,(b_{\max}+1)\,(a_{\max}\, w)^k$$

and $s = \min(n+1, m)$.

Proof: Let σ be a (concrete) model of Φ. Let m' constraints, $\phi_{i_1}, \phi_{i_2}, \ldots, \phi_{i_{m'}}$, evaluate to $\mathbf{true}$ under σ, the rest evaluating to **false**. Let $A' = [a_{i,j}]$ be a $m' \times n$ matrix in which each row comprises the coefficients of variables $x_1, x_2, \ldots, x_n$ in a constraint ϕ_{i_k}, $1 \le k \le m'$. Thus, $A' = [a_{i,j}]$ where $i \in \{i_1, \ldots, i_{m'}\}$.

Now consider a constraint ϕ_{i_k} where $k > m'$, that evaluates to **false** under σ. ϕ_{i_k} is the inequality

$$\sum_{j=1}^{n} a_{i_k,j} x_j \ge b_{i_k}$$

Then σ satisfies $\neg\phi_{i_k}$ which is the inequality

$$\sum_{j=1}^{n} a_{i_k,j} x_j < b_{i_k}$$

or equivalently,

$$\sum_{j=1}^{n} -a_{i_k,j} x_j \ge -b_{i_k} + 1$$

Let A'' be a $(m - m') \times n$ matrix corresponding to the coefficients of variables in constraints $\neg\phi_{i_{m'+1}}, \neg\phi_{i_{m'+2}}, \ldots, \neg\phi_{i_m}$. Thus, $A'' = [-a_{i,j}]$ where $i \in \{i_{m'+1}, \ldots, i_m\}$.

Finally, let $b = [b_{i_1}, b_{i_2}, \ldots, b_{i_{m'}}, -b_{i_{m'+1}} + 1, -b_{i_{m'+2}} + 1, \ldots, -b_{i_m} + 1]^T$

Clearly, σ is a satisfying solution to the ILP given by

$$\left[\frac{A'}{A''}\right] \mathbf{x} \ge b \qquad (6)$$

Also, if the system (6) has a satisfying solution then Φ is satisfied by that solution. Thus, Φ and the system (6) are equi-satisfiable, for every possible system (6) we construct in the manner described above.

By Theorems 1 and 4, we can conclude that if system (6) has a satisfying solution, it has one bounded by $(n+2)\Delta$ where

$$\Delta = s\,(b_{\max} + 1)\,(a_{\max}\, w)^k$$

and $s = \min(n+1, m)$. Moreover, this bound works for every possible system (6).

Therefore, if Φ has a satisfying solution, it has one bounded by $(n+2)\Delta$. $\Box$

Thus, to generate the Boolean encoding of the starting QFP formula, we must encode each integer variable as a symbolic bit-vector of length $S = \lceil \log[(n+2)\Delta] \rceil = \lceil \log(n+2) + \log s + \log(b_{\max}+1) + k(\log a_{\max} + \log w) \rceil$.

Remark. In the preceding discussion, we have used a single bit-vector length for all integer variables appearing in the formula Φ. This is conservative. In general, we can partition the set of variables into classes such that two variables are placed in the same class if there is a constraint in which they both appear with non-zero coefficients. For each class, we separately compute parameters n, k, $b_{\max}$, $a_{\max}$, and w, resulting in a separately computed bit-vector length for each class. The correctness of this partitioning optimization follows from a reduction to ILP as performed in the proof of Theorem 5, and the observation that a satisfying solution to a system of ILPs, no two of which share a variable, can be obtained by solving them independently and concatenating the solutions.

6. Implementation and Experimental Results

We used the bound derived in the previous section to implement a decision procedure based on finite instantiation. Integer variables in the QFP formula are encoded as symbolic bit-vectors large enough to express any integer value within the bound. Arithmetic operators are implemented as arbitrary-precision bit-vector arithmetic operations. Equalities and inequalities over integer expressions are translated to corresponding relations over bit-vector expressions. The resulting Boolean formula is passed as input to a SAT solver.

We implemented our procedure as part of UCLID [35], which is written in Moscow ML [23]. In our implementation we used the zChaff SAT solver [39] version 2003.7.22. We compared UCLID's performance with that of the SAT-based provers ICS [18] (the latest version 2.0) and and CVC-Lite [11] (version 1.1.0),[5] and the automata-based procedure LASH [21]. While CVC-Lite and LASH are sound and complete for QFP, ICS 2.0 is incomplete; i.e., it can report a formula to be satisfiable when it is not. The ground decision procedure ICS uses is the Simplex linear programming algorithm with some additional heuristics to deal with integer variables. However, in our experiments, both UCLID and ICS returned the same answer whenever they both terminated within the timeout. CVC-Lite's ground decision procedure is a proof-producing variant of the Omega test [3].

For benchmarks, we used several formulas from the Wisconsin Safety Analyzer project on checking format string vulnerabilities. The benchmarks include both satisfiable and unsatisfiable formulas in an extension of QFP with uninterpreted functions. Uninterpreted functions were first eliminated using Ackermann's technique [1], and the decision procedures were run on the resulting QFP formula. Some characteristics of the formulas are displayed in Table 2. For each formula, we indicate whether it is satisfiable or

5 The comparison with CVC-Lite is an addition to the paper that appeared at LICS'04. At the time of writing the final conference version, CVC-Lite's implementation was too unstable to be able to make useful comparisons.

not, and also give the values of parameters n, m, k, w, $a_{\max}$ and $b_{\max}$ corresponding to the variable class for which $S = \lceil \log[(n+2)\Delta] \rceil$ is largest, i.e, for which we need the largest number of bits per variable. Note that the *total* numbers of variables and constraints, for all variable classes, are larger: For example, for the benchmark xs-30-40, the formula has 115 variables and 2610 constraints in all. The formulas involve the combination of linear constraints by arbitrary Boolean operators ($\wedge$, $\vee$, $\neg$). The key characteristics of formulas generated in this class of problems is that they vary in n, m, and $b_{\max}$, but the values of k, w, and $a_{\max}$ are fixed at a small value.

Experiments were performed on a Pentium-IV 2 GHz machine with 1 GB of RAM running Linux. A timeout of 3600 seconds was imposed on each run.

A comparison of UCLID versus ICS is displayed in Table 2. CVC-Lite and LASH did not complete on any benchmark within the timeout. For LASH, we attribute this to the relatively large number of variables and constraints in our formulas, and note that Ganesh et al. obtained similar results in their study [15]. For CVC-Lite, we suspect that the time is being mainly spent in the ground decision procedure based on the Omega test, but have been unable to obtain detailed statistics.

From Table 2, we observe that UCLID outperforms ICS on all benchmarks, terminating within a minute on each one.
The reason for UCLID's superior performance is the formula structure, where k, w, and $a_{\max}$ remain fixed at a low value while m, n, and $b_{\max}$ increase. Thus, the maximum number of bits per variable is only moderately large (about 40), even as m increases substantially, and the resulting SAT problem is within the capacity of zChaff. Also, we note that the SAT time is almost always the larger portion of UCLID's run-time; this is not surprising since the time to compute the parameter values and generate the SAT-encoding is polynomial in the input size.
For ICS, we note that the run-time is dominated by the time taken by the ground decision procedure. We observe that the number of inconsistent Boolean assignments alone is not a precise indicator of total run-time, which also depends on the time taken by the ground decision procedure in ruling out a single Boolean assignment.

7. Conclusions and Future Work

In this paper, we have presented a formal approach to exploiting the "sparse, mainly separation constraint" nature of quantifier-free Presburger formulas encountered in software verification. Our approach is based on deriving a new parameterized bound on satisfying solutions to QFP formulas. Experimental results show the benefits of using the derived bound in a SAT-based decision procedure based on finite instantiation.

Note that the bounds we have derived and used in our experiments are conservative. First, the size of minors in a particular problem instance might be far smaller than the bounds we have computed. It is unclear how this can be exploited, since the number of minors grows exponentially with the dimensions of the constraint matrix. Second, for certain special cases, one can improve the $(n+2)\Delta$ bound. For example, if all the constraints are originally equalities and the system of constraints has full rank, a bound of Δ suffices [4]. Thirdly, in cases where the value of $b_{\max}$ is very large due to the presence of a single large constant, one might want to use a less conservative analysis than is performed in the proof of Theorem 4. For instance, the $s\, b_{\max}$ term can be replaced by $\sum_{j=1}^{s} |b_{i_j}|$, where $b_{i_1}, b_{i_2}, \ldots, b_{i_s}$ are the s largest elements of b in absolute value.

In our implementation, we translate a QFP formula to a Boolean formula in a single step. An alternative approach is to perform this transformation *lazily*, increasing the bit-vector size "on demand". This lazy encoding approach works, in brief, as follows. (Details can be found in [20].) We start with an encoding size for each integer variable that is smaller than that prescribed by the bound. If the resulting Boolean formula is satisfiable, so is the original QFP formula. If not, the proof of unsatisfiability generated by the SAT solver is used to generate a sound abstraction of the original formula, which can be checked with a sound and complete decision procedure for QFP (such as the one proposed in this paper). If this decision procedure concludes that the abstraction is unsatisfiable, so is the original formula, but if not, it provides a counterexample which indicates the necessary increase in the encoding size, and the procedure repeats. The advantage of this lazy approach is twofold: (1) It avoids using the conservative bounds we have derived in this paper, and (2) if the generated abstractions are small, the sound and complete decision procedure used by this approach will run much faster than if it were fed the original formula. The bound S that we derive in this paper implies an upper bound nS on the number of iterations of this lazy encoding procedure; thus the lazy encoding procedure needs only polynomially many iterations before it terminates with the correct answer. Using the decision procedure proposed in this paper with the above lazy encoding approach is an interesting avenue for future work.

Acknowledgments

We are grateful to Joël Ouaknine for his inputs and a careful reading of the proofs, and to Ofer Strichman and K. Subramani for valuable discussions. We thank Sagar Chaki, Michael Ernst, Vinod Ganapathy, Somesh Jha, Ranjit Jhala,

Formula	Ans.	Max. Parameters							UCLID Time (sec.)			ICS		
		n	m	k	w	$a_{\max}$	$b_{\max}$	S	Enc.	SAT	Total	#(Inc. assn.)	Time (sec.) Gnd.	Total
s-20-20	SAT	28	437	6	5	4	21	41	4.11	8.18	12.29	904	23.32	23.76
s-20-30	SAT	28	437	6	5	4	30	41	4.15	9.05	13.30	1887	51.68	52.29
s-20-40	UNS	28	437	6	5	4	40	41	4.27	2.03	6.30	25776	658.01	669.99
s-30-30	SAT	38	792	6	5	4	31	42	6.42	22.82	29.24	2286	268.21	269.42
s-30-40	SAT	38	792	6	5	4	40	42	6.45	20.45	26.90	14604	1621.27	1625.15
xs-20-20	SAT	49	668	6	5	4	21	42	5.49	30.95	36.44	2307	97.21	98.32
xs-20-30	SAT	49	668	6	5	4	30	43	5.51	24.48	29.99	33103	1519.77	1540.27
xs-20-40	UNS	49	668	6	5	4	40	43	5.55	14.30	19.85	97427	3468.91	*
xs-30-40	SAT	69	1288	6	5	4	40	44	9.90	36.61	46.51	33754	3082.34	*

Table 2. Benchmark characteristics and experimental results. For UCLID, we list the time taken to decide the formula including a breakup into the encoding time ("Enc.") and the time taken by the SAT solver ("SAT"). For ICS, we give the total time, the number of inconsistent Boolean assignments analyzed by the ground decision procedure ("#(Inc. assn.)"), as well as the overall time taken by the ground decision procedure ("Gnd."). A "$*$" indicates that the decision procedure timed out after 3600 sec. CVC-Lite and LASH did not complete within the timeout on any formula.

and Stephen McCamant for providing us with benchmark formulas. We also thank Leonardo de Moura and Louis Latour for help with ICS and LASH respectively. This research was supported by ARO grant DAAD19-01-1-0485.

References

[1] W. Ackermann. *Solvable Cases of the Decision Problem.* North-Holland, Amsterdam, 1954.

[2] C. Barrett, D. L. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In E. Brinksma and K. G. Larsen, editors, *Proc. 14th Intl. Conference on Computer-Aided Verification (CAV'02)*, LNCS 2404, pages 236–249. Springer-Verlag, July 2002.

[3] S. Berezin, V. Ganesh, and D. L. Dill. An online proof-producing decision procedure for mixed-integer linear arithmetic. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, LNCS 2619, pages 521–536, April 2003.

[4] I. Borosh, M. Flahive, D. Rubin, and L. B. Treybig. A sharp bound for solutions of linear Diophantine equations. *Proceedings of the American Mathematical Society*, 105(4):844–846, April 1989.

[5] I. Borosh, M. Flahive, and L. B. Treybig. Small solutions of linear Diophantine equations. *Discrete Mathematics*, 58:215–220, 1986.

[6] I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, March 1976.

[7] J. Brenner and L. Cummings. The Hadamard maximum determinant problem. *American Mathematical Monthly*, 79:626–630, June-July 1972.

[8] R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *Proceedings of the IEEE VLSI Design Conference*, pages 741–746, 2002.

[9] R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In E. Brinksma and K. G. Larsen, editors, *Proc. Computer-Aided Verification (CAV'02)*, LNCS 2404, pages 78–92, July 2002.

[10] S. Chaki, E. M. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in C. In *Proc. 25th International Conference on Software Engineering (ICSE)*, pages 385–395, 2003.

[11] CVC-Lite: Cooperating Validity Checker. Available at `http://verify.stanford.edu/CVCL/`.

[12] L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Proc. 18th International Conference on Automated Deduction (CADE)*, pages 438–455, 2002.

[13] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Laboratories Palo Alto, 2003.

[14] M. J. Fischer and M. O. Rabin. Super-exponential complexity of Presburger arithmetic. *Proceedings of SIAM-AMS*, 7:27–41, 1974.

[15] V. Ganesh, S. Berezin, and D. L. Dill. Deciding Presburger arithmetic by model checking and comparisons with other methods. In *Formal Methods in Computer-Aided Design (FMCAD '02)*, LNCS 2517, pages 171–186. Springer-Verlag, November 2002.

[16] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver. In *Design Automation and Test in Europe (DATE) 2002*, pages 142–149, 2002.

[17] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Proc. 29th ACM Symposium on Principles of Programming Languages*, pages 58–70, 2002.

[18] ICS: Integrated Canonizer and Solver. Available at `http://www.icansolve.com`.

[19] R. Kannan and C. L. Monma. On the computational complexity of integer programming problems. In *Optimisation and Operations Research*, volume 157 of *Lecture Notes in Economics and Mathematical Systems*, pages 161–172. Springer-Verlag, 1978.

[20] D. Kroening, J. Ouaknine, S. A. Seshia, and O. Strichman. Abstraction-based satisfiability solving of Presburger arithmetic. In *Proc. 16th International Conference on Computer-Aided Verification (CAV)*, pages 308–320, July 2004.

[21] LASH Toolset. Available at `http://www. montefiore.ulg.ac.be/~boigelot/research/lash`.

[22] S. McCamant and M. D. Ernst. Predicting problems caused by component upgrades. In *Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE)*, pages 287–296, 2003.

[23] Moscow ML. Available at `http://www.dina.dk/~sestoft/mosml.html`.

[24] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[25] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[26] C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.

[27] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, chapter 13. Prentice-Hall, 1982.

[28] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small-domain instantiations. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469. Springer-Verlag, July 1999.

[29] V. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, 1977. Cambridge, MA.

[30] M. Preßburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes-rendus du Premier Congrès des Mathématiciens des Pays Slaves*, 395:92–101, 1929.

[31] W. Pugh. The omega test: A fast and practical integer programming algorithm for dependence analysis. In *Supercomputing*, pages 4–13, 1991.

[32] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.

[33] N. Shankar and H. Rueß. Combining Shostak theories. In S. Tison, editor, *Proc. Rewriting Techniques and Applications*, LNCS 2378, pages 1–18. Springer-Verlag, July 2002.

[34] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, 1984.

[35] UCLID Verification System. Available at `http://www.cs.cmu.edu/~uclid`.

[36] J. von zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, October 1978.

[37] Wisconsin Safety Analyzer Project. `http://www.cs.wisc.edu/wisa`.

[38] P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proc. Static Analysis Symposium*, LNCS 983, pages 21–32, September 1995.

[39] zChaff Boolean Satisfiability Solver. Available at `http://ee.princeton.edu/~chaff/zchaff.php`.