Mixed and Integer Linear Programming Using Automata Techniques*

Mia Minnes
Cornell University
Ithaca, NY
minnes@math.cornell.edu

ABSTRACT

We present a comprehensive overview of automata techniques for deciding first order logical theories. These techniques are useful in Integer Linear Programming and Mixed Integer Linear Programming, which in turn have wide applications in diverse areas of computer science and engineering.

We have several goals in this paper. The first is to solidify the theory underpinning the automata techniques. Since much of the previous literature in this subject was published only in the form of Extended Abstracts, many of the proofs to key theorems were missing. We fill in these gaps.

Another objective is to explore extending the automata approach to address questions beyond satisfiability. A key problem for the (M)ILP community has been the enumeration of solutions to systems. We present a way of addressing this question within the automata framework.

Finally, we consider alternate approaches to (M)ILP and discuss their relative advantages and disadvantages as compared to the automata formulation.

Keywords

Integer Linear Programming, Mixed Integer Linear Programming, Presburger Arithmetic, Real Linear Arithmetic, Satisfiability, Finite Automata, Büchi Automata, Real Vector Automata

1. INTRODUCTION

Examples abound of applications requiring solutions to systems of constraint equations. One usual formulation of such systems is by Integer Linear Programming (ILP). ILP has been used in discrete optimization problems and control theory, in modern compilers for such tasks as dependence analysis for loop transformation [22], and for verification of hardware design [10], [13]. Moreover, extending our attention to Mixed Integer Linear Programming (MILP), where variables may range over either real numbers or the integers, leads to applications in hybrid systems and timed systems.

(M)ILP solvers must be able to answer several questions. The first is: given a system of linear equations and inequalities, is there a solution to the system? If so, it must be possible to exhibit such a solution. And finally, it is often desirable (especially in compiler applications) to enumerate all possible solutions, or give bounds on their range.

(M)ILP may be represented as a fragment of particular first order logical theories. Hence, a decision procedure for the satisfiability question of these theories satisfies to answer the first question above. Moreover, if this decision procedure is constructive (i.e. yields an example if the formula is satisfiable) then the second question is answered as well. An advantage to treating the subject of ILP via logic techniques is that it is likely that the resulting decision procedure will be extendible to more complicated systems.

2. ILP AND PRESBURGER ARITHMETIC

Presburger arithmetic is the first order theory of

$$(\mathbb{Z}, +, \leqslant, 0, 1).$$

The atomic formulae are linear equations and inequalities of the form

$$a_1x_1 + \cdots + a_nx_n = c$$

or

$$a_1x_1 + \cdots + a_nx_n \leqslant c$$

where $(a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$. Note that scalar multiplication is allowed since, for example, a_1x_1 is a notational abbreviation for $x_1 + \cdots + x_1$ (a_1 times). Moreover, note that by negation, multiplication by -1, or subtraction from c by 1, any formula of the form $a_1x_1 + \cdots + a_nx_n \mathscr{R} c$, for $\mathscr{R} \in \{=, \leqslant, \geqslant, <, >\}$ is representable in Presburger arithmtetic.

Observe that Presburger Arithmetic is a stronger system than that needed to represent ILP. In fact, ILP formulations consist exactly of disjunctions of conjunctions of atomic formulae and their negations. Hence, we need only focus on the quantifier free fragment of Presburger Arithmetic to implement an ILP solver. In 1927, Presburger gave a decision procedure for the full theory [21], which was improved by Cooper [14] and by Reddy and Loveland in 1978 [23]. Each

*Final Project for CS 612
Also see `http://www.math.cornell.edu/~minnes/Automata`

of these is based on the standard logic technique of quantifier elimination for formulae in prenex normal form. This approach can be made fairly efficient for deciding the satisfaction of single formulas. However, it neither gives a sample vector satisfying the formula, nor enumerates the solutions. Moreover, while the procedure is elegant mathematically, it does not readily lend itself to implementation and automation. Hence, to use Presburger arithmetic as the domain of the ILP satisfaction problem, we will need a different decision procedure.

2.1 An Automata Theoretic Formulation

The main idea is the following: Given a formula $\varphi(x_1, \ldots, x_n)$ in Presburger Arithmetic, generate a finite automaton A_φ accepting the set $\{(x_1, \ldots, x_n) \in \mathbb{Z}^n : (x_1, \ldots, x_n) \vDash \varphi\}$. Then, use automata techniques to ask questions about the satisifiability of the formula and its solutions. We include definitions of the basic terminology of finite automata to introduce the notation used hereafter.

Definition 1. A **Finite Automaton** on finite words (FA) is a tuple $(S, \Sigma, \delta, I, F)$ such that S is a finite set of states, Σ is a finite set of symbols (the alphabet), $\delta \subset (S \times \Sigma) \times S$ is the transition relation, $I \subset S$ is the non-empty set of initial states, and $F \subset S$ is the set of accepting states. For $\mathcal{A}$ a FA, a **run** of $\mathcal{A}$ on input $w = \sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{n}} \in \Sigma^*$ is a sequence of states $s_0 \ldots s_{n+1}$ such that $s_0 \in I$ and for each $0 \leqslant i \leqslant n$, $(s_i, \sigma_{\mathbf{i}}, s_{i+1}) \in \delta$. A run of $\mathcal{A}$ is successful if the last state of the run is in F. A word $w \in \Sigma^*$ is **accepted** by $\mathcal{A}$ if there is some run of $\mathcal{A}$ on w which is successful. $\mathcal{L}(\mathcal{A}) \subset \Sigma^*$ is the set of words accepted by $\mathcal{A}$. A FA is called **deterministic** if I is a singleton set and δ is a (possibly partial) function $S \times \Sigma \to S$.

2.1.1 Translation

The following algorithm is that of [25, 26], extending that of [9]. The translation is performed inductively, corresponding to the recursive definition of formulae. Hence, we first describe the automata corresponding to atomic formulae, and then those for conjuctions, disjunctions, negations, and quanitified formulae.

First, some technical points. Given $\varphi(x_1, \ldots, x_n)$ a formula in Presburger arithmetic with free variables $x_1, \ldots, x_n$, vectors satisfying φ must lie in $\mathbb{Z}^n$. Hence, the language of $\mathcal{A}_\varphi$, must be a subset of $\mathbb{Z}^n$. For implementation purposes, vectors of integers are represented as vectors of binary encodings where each component has the same encoding length. Hence, the alphabet of $\mathcal{A}_\varphi$ is $\Sigma = \{0, 1\}^n$, so that each unit of input gives one more bit to the encoding of each component in the vector. The encodings have the MSB first and use 2's complement for negative numbers. Finally, we require that any automaton accepting some encoding of a vector accepts all valid encodings of the vector (i.e., that we allow for arbitrarily many repetitions of the sign bit). Note that the translation described below is easily generalizable to base r.

2.1.2 Automata for Equations

Let φ be $a_1 x_1 + \cdots + a_n x_n = c$ for $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$. Abbreviating the scalar product as $\mathbf{a} \cdot \mathbf{x}$, φ may be written as $\mathbf{a} \cdot \mathbf{x} = c$.

The intuition behind the definition for A_φ is that each state will represent the (integer value of the) computation $\mathbf{a} \cdot \mathbf{x}$ for the currently known value of $\mathbf{x}$. Transitions between states will then reflect the contribution of the most recently "learned"bit of $\mathbf{x}$ to the computation.

Therefore, we define $\mathcal{A}^1_\varphi = (\mathbb{Z} \cup \{s_i\}, \{0, 1\}^n, \delta^1, \{s_i\}, \{c\})$, where

$$\delta^1(s, \mathbf{b}) = \begin{cases} -\mathbf{a} \cdot \mathbf{b} & \text{if } s = s_i \\ 2s + \mathbf{a} \cdot \mathbf{b} & \text{if } s \in \mathbb{Z} \end{cases}$$

The transition function corresponds to a 2's complement interpretation of the first bit[1], and corresponds to shifting the current value of the computation to the left and adding the newest contribution for non-sign bits.

Observe that $\mathcal{A}_\varphi$ is deterministic. Hence, there is exactly one run of $\mathcal{A}_\varphi$ for any input $\mathbf{w} \in \Sigma^*$.

THEOREM 1. [2] *For any word* $\mathbf{w} \in \Sigma^*$, $\mathbf{w} \in \mathcal{L}(\mathcal{A}^1_\varphi)$ *iff* $\mathbf{w}$ *encodes a solution* $\mathbf{x}$ *to* $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$.

PROOF. We prove the stronger statement: for any word $\mathbf{w} \in \Sigma^*$ and any $s \in S$, there is a run of $\mathcal{A}^1_\varphi$ on $\mathbf{w}$ ending at state s iff $\mathbf{w}$ encodes a solution $\mathbf{x}$ to $\mathbf{a} \cdot \mathbf{x} = s$. The proof is by induction on the length of words. Suppose $\mathbf{w} = \sigma_{\mathbf{0}}$, encoding the vector $\mathbf{x} = (-\sigma_{0,1}, \ldots, -\sigma_{0,n})$. Then the run of $\mathcal{A}^1_\varphi$ is $s_i s_1$ where $s_1 = -\mathbf{a} \cdot \sigma_{\mathbf{0}}$. Thus, $s_1 = -\mathbf{a} \cdot (-\sigma_{0,1}, \ldots, -\sigma_{0,n}) = \mathbf{a} \cdot \mathbf{x}$, as required.

For the inductive step, suppose that for any encoding $\sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}}$ of $\mathbf{y}$, $\mathbf{y}$ satisfies $\mathbf{a} \cdot \mathbf{y} = s$ iff the run of $\mathcal{A}^1_\varphi$ on the encoding ends at s. Let $\mathbf{w} = \sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}} \sigma_{\mathbf{m+1}}$. Then $\mathbf{w}$ encodes the vector $\mathbf{x} = 2\mathbf{y} + \sigma_{\mathbf{m+1}}$. The run of $\mathcal{A}^1_\varphi$ on $\mathbf{w}$ is the result of appending $s \to s'$ to the run on $\sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}}$, where by the definition of the transition function,

$$s' = 2s + \mathbf{a} \cdot \sigma_{\mathbf{m+1}} = 2(\mathbf{a} \cdot \mathbf{y}) + \mathbf{a} \cdot \sigma_{\mathbf{m+1}} = \mathbf{a} \cdot (2\mathbf{y} + \sigma_{\mathbf{m+1}}) = \mathbf{a} \cdot \mathbf{x}$$

as required. □

An immediate problem with this definition is the infinite state space. However, we may observe that once the automaton is in a state whose aboslute value is sufficiently large, the accepting state is not reachable. Formally, let $||\mathbf{a}||_1 = \sum_{i=1}^n |a_i|$ and suppose $|s| > \max(|c|, ||\mathbf{a}||_1)$. Then note that for any $\mathbf{b} \in \Sigma$,

$$|2s + \mathbf{a} \cdot \mathbf{b}| \leqslant 2|s| + |\mathbf{a} \cdot \mathbf{b}| \leqslant 2|s| + ||\mathbf{a}||_1.$$

Therefore, any transition from state s satisfying the above conditions leads to a state s' such that $|s'| > |c|$. In turn, s' satisfies the same magnitude conditions as s and hence transitions from it also move away (in absolute value) from c. Thus, for any state s such that $|s| > \max(|c|, ||\mathbf{a}||_1)$, there

[1] Recall that a word $b_1 \ldots b_n$ in 2's complement encodes the value

$$-b_1 2^{n-1} + b_2 2^{n-2} + \cdots + b_{n-1} 2^1 + b_n.$$

[2] This is one of the cases in which the correctness proof was missing from the literature.

is no path from s to c. In light of this, define

$$\mathcal{A}^2_\varphi = ((\mathbb{Z} \cap \{s : |s| \leqslant \max(|c|, ||\mathbf{a}||_1)\}) \cup \{s_i, s_+, s_-\}, \{0,1\}^n, \delta^2, \{s_i\}, \{c\})$$

where,

$$\delta^2(s, \mathbf{b}) = \begin{cases} -\mathbf{a}\cdot\mathbf{b} & \text{if } s = s_i, \\ s' & \text{if } s' = 2s + \mathbf{a}\cdot\mathbf{b}, s' \leqslant \max(|c|, ||\mathbf{a}||_1), \\ s_+ & \text{if } 2s + \mathbf{a}\cdot\mathbf{b} > \max(|c|, ||\mathbf{a}||_1), \\ s_- & \text{if } 2s + \mathbf{a}\cdot\mathbf{b} < -\max(|c|, ||\mathbf{a}||_1). \end{cases}$$

Note that the result of Theorem 1 still holds for $\mathcal{L}(\mathcal{A}^2_\varphi)$ since modifying the transition function as above does not affect which words are accepted.

The automaton A^2_φ is deterministic and gives a correct translation but is not minimal. To eliminate those states from which the accepting state is not reachable, we construct the automaton backwards and only include the necessary states. The algorithm is

1. Create a table H for the set of states and a list L for the active states. Initialize H to $\{s_i, c, s_+, s_-\}$ and L to $\{c\}$.
2. Repeat until $L = \emptyset$: Remove a state s from L. For every $\mathbf{b} \in \{0,1\}^n$,
   - If $s_0 = \frac{s - \mathbf{a}\cdot\mathbf{b}}{2} \in \mathbb{Z}$, then if s_0 is not already in H, add s_0 to H and L, and add a transition labelled by $\mathbf{b}$ from s_0 to s.
   - If $s = -\mathbf{a}\cdot\mathbf{b}$ then add a transition labelled by $\mathbf{b}$ from s_i to s.
3. Put $I = \{s_i\}$ and $F = \{c\}$.
4. Complete the automaton by directing all missing transitions to either s_+ or s_-.

Observe that the automaton resulting from this construction, which we call $\mathcal{A}_\varphi$, is a subset of $\mathcal{A}^2_\varphi$ and hence, is deterministic. Moreover, $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\mathcal{A}^2_\varphi)$.

2.1.3 Automata for Inequalities

The algorithms described in this section are based on [26].

Let φ be $a_1x_1 + \cdots + a_nx_n \leqslant c$ for $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$ As before, φ may be written as $\mathbf{a}\cdot\mathbf{x} = c$.

We apply the same intuition for representing inequalities as automata as we did in the equation case. Moreover, we may use the same reasoning for pruning the state space. Hence, define

$$\mathcal{A}^{in_1}_\varphi = ((\mathbb{Z} \cap \{s : |s| \leqslant \max(|c|, ||\mathbf{a}||_1)\}) \cup \{s_i, s_+, s_-\}, \{0,1\}^n, \delta^{in_1}, \{s_i\}, \{c\})$$

However, the transition function and/ or the set of accepting states must be defined a little differently. One possibility is to leave the transition function as is but to define $F = \{s \in S : s \in \mathbb{Z} \text{ and } s \leqslant c\}$. However, as we will see later, it will be helpful to have only one accepting state. To that end, we might define

$$\delta^{in_1} = \delta^2 \cup \{(s, \mathbf{b}, s') : \text{there is some } s'' \leqslant s \text{ such that } \delta^2(s, \mathbf{b}) = s''\}.$$

Thus, $\mathcal{A}^{in_1}_\varphi$ has the same states, set of initial states, and set of accepting states as the FA for the equation with the same coefficients, but has more transitions.[3]

It is possible to prune the FA as we did in the case of automata representing equations. The algorithm in this case is:

1. Create a table H for the set of states and a list L for the active states. Initialize H to $\{s_i, c, s_+, s_-\}$ and L to $\{c\}$.
2. Repeat until $L = \emptyset$: Remove a state s from L. For every $\mathbf{b} \in \{0,1\}^n$:
   - Let $s_0 = \lfloor \frac{s - \mathbf{a}\cdot\mathbf{b}}{2} \rfloor$. If s_0 is not already in H, add s_0 to H and L, and add a transition labelled by $\mathbf{b}$ from s_0 to s.
   - If $s = -\mathbf{a}\cdot\mathbf{b}$ then add a transition labelled by $\mathbf{b}$ from s_i to s.
3. Put $S = H$, $\delta^{in'}$ the transitions defined in step 2, $I = \{s_i\}$ and $F = \{s \in S : s \leqslant c\} \cup \{s_-\}$.
4. Complete the automaton by adding, for each state $s \in H$, input $\mathbf{b} \in \{0,1\}^n$

$$\delta^{in}(s, \mathbf{b}) = \delta^{in'}(s, \mathbf{b}) \cup \min\{s' \in S : s' \geqslant 2s + \mathbf{a}\cdot\mathbf{b}\}^4.$$

Note that $\mathcal{A}^{in}_\varphi = \left(S, \{0,1\}^n, \delta^{in}, I, F\right)$ is not deterministic. However, deterministic automata are much easier to implement and manipulate. In general, determinizing a FA comes at an exponential cost in the size of the state space (since the subset construction is used). However, the automata corresponding to linear inequalities are of a particular kind.

Definition 2. [26] Given a non-deterministic FA $\mathcal{A} = (S, \Sigma, \delta, \{s_0\}, F)$, for each $s \in S$ let $\mathcal{A}_s = (S, \Sigma, \delta, \{s\}, F)$. Then $\mathcal{A}$ is said to be **ordered** if there is a constant-time decidable strict total order $\prec$ on S (or $S \setminus \{s_i\}$) such that for any pair of states $s_1 \prec s_2$, $\mathcal{L}(\mathcal{A}_{s_1}) \subsetneq \mathcal{L}(\mathcal{A}_{s_2})$.

THEOREM 2. *[26] A non-deterministic ordered FA can be determinized in linear time and with no penalty in the number of states.*

PROOF. (Sketch of proof in [26]) Without loss of generality, we can remove transitions from a state on a given input to all but the $\prec$-greatest one, since the set of words accepted from all other states is a subset of those accepted

[3] This version of the forward construction is not included in [26].

[4] Note that we consider s_+ as greater than all other states, and s_- as less than all other states. There is little discussion of s_+, s_- in [26].

from it. Since this algorithm merely removes transitions at each state, it takes linear time in the size of the automaton, and produces a deterministic FA with the same set of states as the original non-deterministic FA. □

THEOREM 3. *[26] The FA $\mathcal{A}^{in}_{\varphi}$ defined above is ordered under the total order on the states which is the inverse of the arithmetic ordering on the integers.*

PROOF. (Sketch of proof in [26]) This follows from the interpretation of the states as labels for the right hand side of an inequality whose left hand side is $\mathbf{a} \cdot \mathbf{x}$ and analysing the relationship of words starting at p, q for $p < q$. □

Hence, following the proof in [26], let

$$\delta^{in}_{det}(s, \mathbf{b}) = \min\{\delta^{in}(s, \mathbf{b})\}.$$

The FA $\mathcal{A}^{in}_{\varphi,det}$ where δ^{in} is replaced by δ^{in}_{det} is a deterministic FA satisfying the property that its language is exactly binary encodings of vectors in $\mathbb{Z}^n$ satisfying φ.

2.1.4 Automata for General Formulas

We recall the standard procedures for Boolean operations on (complete) deterministic FA. Given $\mathcal{A}$ accepting $\mathcal{L}(\mathcal{A})$, an automaton $\neg\mathcal{A}$ accepting $\Sigma^* \setminus \mathcal{L}(\mathcal{A})$ is that resulting from flipping the accepting/ nonaccepting status of each state (i.e. by putting $F' = S \setminus F$).

Given $\mathcal{A}_1, \mathcal{A}_2$ deterministic FA whose languages are respectively $\mathcal{L}(\mathcal{A}_1)$, $\mathcal{L}(\mathcal{A}_2)$. The automaton $\mathcal{A}_1 \cap \mathcal{A}_2$ accepting $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ is defined via the product construction (which preserves the deterministic nature), putting

$$F_{\cap} = \{(s_1, s_2) \in S_1 \times S_2 : s_1 \in F_1 \text{ and } s_2 \in F_2\}.$$

Similarly, the automaton $\mathcal{A}_1 \cup \mathcal{A}_2$ accepting $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ is defined via the product construction with

$$F_{\cup} = \{(s_1, s_2) \in S_1 \times S_2 : s_1 \in F_1 \text{ or } s_2 \in F_2\}.$$

Now suppose $\varphi \equiv \neg\psi$ where ψ is a formula for which we have an equivalent FA, $\mathcal{A}_{\psi}$. The set of free variables in φ is the same as the set of free variables in ψ. Assume without loss of generality there are n free variables. Then,

$$\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x} \text{ satisfies } \psi\} = \mathbb{Z}^n \setminus \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x} \text{ does not satisfy } \psi\}.$$

Similarly:

$$\{\mathbf{w} \in (\{0,1\}^n)^* : \mathbf{w} \text{ encodes a vector satisfying } \psi\} = (\{0,1\}^n)^* \setminus \{\mathbf{w} \in (\{0,1\}^n)^* : \mathbf{w} \text{ does not encode a vector satisfying } \psi\}$$

since any word over $\{0,1\}^n$ encodes some vector of integers. Hence, putting $\mathcal{A}_{\varphi} = \neg\mathcal{A}_{\psi}$ gives an automaton accepting any and all encodings of vectors satisfying φ.

Following similar arguments, we conclude that if $\varphi \equiv \psi_1 \wedge \psi_2$, we put $\mathcal{A}_{\varphi} = \mathcal{A}_{\psi_1} \cap \mathcal{A}_{\psi_2}$. If $\varphi \equiv \psi_1 \vee \psi_2$, then we define $\mathcal{A}_{\varphi} = \mathcal{A}_{\psi_1} \cup \mathcal{A}_{\psi_2}$.

In the above, we have defined FA corresponding to any quantifier free formula of Presburger arithmetic, and hence have a translation for any formula representing an ILP system. To demonstrate the natural extendibility of the translation scheme, we examine automata for quantified formulae.

Let $\varphi \equiv \exists x_i \psi$, let $\mathcal{A}_{\psi}$ be the FA corresponding to ψ, and assume that ψ has n free variables (including x_i). Then,

$$\mathcal{L}(\mathcal{A}_{\psi}) = \{\mathbf{w} \in (\{0,1\}^n)^* : \mathbf{w} \text{ encodes a vector satisfying } \psi\}.$$

We would like to define an automaton over $\{0,1\}^{(n-1)}$ whose language is

$$\{\mathbf{w} \in \left(\{0,1\}^{(n-1)}\right)^* : \text{for each } 0 \leqslant j \leqslant, \text{length}(\mathbf{w}) \text{ there is } \sigma_j \in \{0,1\} \text{ such that } \mathbf{w}' \text{ defined by inserting these bits in the } i\text{th positions of each component is in } \mathcal{L}(\mathcal{A}_{\psi})\}$$

But, this is (almost) exactly the projection operation applied to finite automata[5]. The slight modification we must make is that the FA resulting from projection may no longer accept all encodings of vectors satisfying φ. To illustrate this, consider the case where $(10, 2)$ is encoded in the language of $\mathcal{A}_{\psi}$, i.e. as $(0^m 01010, 0^m 00010)$. After projecting out the first variable, only encodings of 2 with at least three leading zeroes are included in the language. To ensure that the automata conforms to our specifications, we can apply the following modification:

`For each` $\mathbf{b} \in \{0,1\}^{(n-1)}$`, add to` $\delta(s_i, \mathbf{b})$ `any states which are reachable from` s_i `by` $\mathbf{b}^k$ `for some finite` k.

Note that the resulting FA is non-deterministic. Hence, in order for the induction to go through (since our algorithm for complementation requires that the FA be deterministic), we must determinize the resulting FA (this time possibly incurring the exponential growth in the size of the set of states).

For universally quantified formulas, we first convert to the equivalent existential formula ($\forall x(\varphi) \equiv \neg\exists x(\neg\varphi)$) and then apply automata transformations as above.

This translation mechanism has been implemented in [7].[6]

2.1.5 Satisfiability and Solutions

Now that we have a translation procedure yielding for each formula φ a FA $\mathcal{A}_{\varphi}$ which accepts exactly the encodings of vectors satisfying φ, we can hope that we will be able to exploit automata techniques to easily look for a solution to the satisfiability question.

In fact, this is the case. Recall that a formula is satisfiable if and only if there is some vector which gives a true interpretation to the formula. Hence, φ is satisfiable if and only if there is some word accepted by $\mathcal{A}_{\varphi}$. We have therefore reduced satisfiability in Presburger arithmetic to the emptiness question for FA.

[5]For more on FA operations, see e.g. [17].

[6]In fact, a modification of this algorithm is used where input is read in sequentialized rather than n bits at a time. The algorithm is very much in the same spirit, but with a few details changed.

Moreover, the emptiness question is efficiently decidable. To see this, recall that we can abstract away most of the structure of any FA and obtain a directed graph (digraph). The question can then be stated as "Is there a path from some state in I to some state in F?". A linear time (in the size of the digraph) algorithm to answer this question is given in [4] and included below:

- `function accepting-path(state` s`)`

  `var` (s_1, a, s_2)`;`
  `if` $s \in F$ `do return` $True$`;`
  `for each` $(s_1, a, s_2) \in \delta$ `such that` $s_1 = s$ `do`
  `if accepting-path(`s_2`) then return` $True$`;`
  `return` $False$`;`

- `function isEmpty()`

  `for each` $s \in I$ `do`
  `if accepting-path(`s`) then return` $False$`;`
  `return` $True$`;`

An advantage of this approach to satisfiability is that if $\mathcal{L}(\mathcal{A}_\varphi)$ is not empty, the algorithm for determining this gives an example of a word accepted $\mathcal{A}_\varphi$. Then, simple translations from binary notation give a vector of integers satisfying φ. This is helpful for debugging when ILP is used to formulate model checking problems.

However, an even more useful property would be to enumerate the set of vectors satisfying φ. The main application of enumeration is in generating loop bounds for loop nests under affine loop transformations. Given $A\mathbf{i} \leqslant \mathbf{b}$, a description of the original loop bounds, and an invertible linear transformation $\mathbf{u} = T\mathbf{i}$ on the indices of the loops, we get the system: $AT^{-1}\mathbf{u} \leqslant \mathbf{b}$. This is a system of linear inequalities. If we could enumerate the solutions to this system, i.e. give bounds on the values of the components of U, we would have the bounds for the transformed loop nest.

Note that if this question is definable in the first order theory of Presburger arithmetic, then it is decidable by the techniques we have discussed above. Consider[7], only those cases where

$$\varphi(x_1, \ldots, x_n) \equiv (a_{1,1}x_1 + \cdots + a_{1,n}x_n \leqslant b_1) \wedge \cdots \wedge (a_{m,1}x_1 + \cdots + a_{m,n}x_n \leqslant b_m).$$

We abbreviate this as $A\mathbf{x} \leqslant \mathbf{b}$ and apply the following procedure: To get bounds on x_1, we define the following two formulae

$$\psi_1(x_1) \equiv \exists x_2 \ldots x_n(\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(y, x_2, \ldots, x_n) \rightarrow x_1 \leqslant y))$$
$$\psi_1'(x_1) \equiv \exists x_2 \ldots x_n(\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(y, x_2, \ldots, x_n) \rightarrow y \leqslant x_1)).$$

[7]We make this simplication since it corresponds to the application in which we are interested.

These are first order formulae in Presburger arithmetic with one free variable, whose semantics are that the value satisfying them is the minimum (resp. maximum) value for x_1 which might satisfy φ. By the translation to automata discussed above, we can build $\mathcal{A}_{\psi_1}$, $\mathcal{A}_{\psi_1'}$ and check for satisfiability. If they are satisfied, then we explicitly get (from the accepting path through the automaton) bounds for the first variable. Let m_1, M_1 be these bounds. For any subsequent variable, we define similar formulae ψ_k, ψ_k' which incorporate the bounds on previously considered variables:

$$\psi_k(x_k) \equiv \exists x_{k+1} \ldots x_n \forall x_1 \ldots x_{k-1}([m_1 \leqslant x_1 \leqslant M_1 \wedge \cdots \wedge m_{k-1} \leqslant x_{k-1} \leqslant M_{k-1}] \rightarrow [\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) \rightarrow x_k \leqslant y)])$$
$$\psi_k'(x_k) \equiv \exists x_{k+1} \ldots x_n \forall x_1 \ldots x_{k-1}([m_1 \leqslant x_1 \leqslant M_1 \wedge \cdots \wedge m_{k-1} \leqslant x_{k-1} \leqslant M_{k-1}] \rightarrow [\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) \rightarrow y \leqslant x_k)])$$

Again, we get formulae in one free variable that represent the bounds on the current variable. Once these formulae are converted to equivalent automata, we can quickly find an encoding for this bound, and hence its value. Thus, the ease with which the automata formulation handles quantified statements allows it to generate bounds for vectors satisfying constraint equations.

2.2 Other Approaches to the ILP Problem

As already mentioned, it is not necessary to formulate the ILP question as satisfaction of formulas in Presburger arithmetic. In fact, the two current standards for solving ILP are not implemented as decision procedures to particular logic theories. These are the Simplex Method, and Fourier-Motzkin Elimination (aka the Omega Test).

2.2.1 Simplex Method

The Simplex method for ILP refers to an approach which recognizes that each linear inequality characterizes a half-space and hence the feasibility region is a convex region in $\mathbb{Z}^n$. Then, solutions must exist within this convex region, and optimal solutions exist at vertices. This was one of the first methods for solving ILP problems and yields helpful geometric intuitions. It has the nice feature that optimal (with respect to some linear objective function) solutions are relatively easy to find if they exist. However, the enumeration problem is not handled well by the Simplex method.

2.2.2 Omega

The Omega test [22] is an extension of Fourier-Motzkin elimination of variables. Given a system of linear equalities and inequalities over integer variables, the method first uses Gaussian elimination and the gcd test to get solutions for the equations (if they exist). If there are no solutions, then we know the whole system is unsatisfiable and are done. However, if there are solutions to the equations in the system, the variables in the inequalities are parametrized by these, and then Fourier-Motzkin elimination is used to project each variable out of the remaining inequalities. The projected

variable is expressed in terms of maxima and minima involving expressions in the still-free variables. At the end, we remain with inequalities involving a single variable and integers. This is easy to simplify, and then ripple the effect to all the variables.

An advantage of the Omega approach to the ILP problem is that it provides a method to enumerate the solutions of a system of inequalities (which may represent the loop bounds of a loop nest)[1]. This is done, again, by projecting out variables one at a time and hence getting bounds for each variable in terms of other variables. The resulting formulations corresponds exactly to loop bounds for nested loops.

Note that a central disadvantage in implementations of both these methods is that they are conservative. In other words, it is possible that they will have false positives in that they claim a solution exists when none does. Moreover, by the nature of both the Omega method and the Simplex method, eliminating this conservativity is hard. On the other hand, false positives never happen with the automata approach since the decision procedure for non-emptiness is constructive.

2.3 Results and Evaluation

The automata approach to ILP via Presburger arithmetic may be evaluated by various metrics. We might ask how large the automaton need to be for an arbitrary formula. Since the emptiness problem for automata may be solved in time linear in the size of the automata, the space complexity plays a significant role in the potential performance of this approach. More concretely, experimental results for standard test questions in the area give an idea about the average case performance of this scheme as compared to other techniques.

2.3.1 Complexity Results

In the paper introducing the automata algorithm [26], Wolper and Boigelot argue that there is a nonelementary upper bound on the size of the FA generated by a Presburger formula, and hence a nonelementary upper bound on the time for solving the satisfiability question.

More recently, [18] presents the worst case complexity for any minimal deterministic FA of Presburger arithmetic formulae. In this paper, Klaedtke proposes some optimizations to the translation algorithm in [26] and proves that the tight worst case upper bound on the size of such FA for Presburger formulae is triply exponential.

However, we comment here is that the results above pay no heed to the time required to construct the FA. Indeed, the complexity bounds here reflect the size of the automata alone. Hence, these results are most important if the FA are used as many times as possible so that the size of the FA has the highest bearing on the performance of the algorithm, rather than the time required to construct it.

2.3.2 Experimental Results

As a result of considerations like the above, experimental study in addition to theoretical analysis is helpful in determining the efficacy of the automata approach.

A preliminary study is presented in [16]. It is interesting to note that in this study, the authors take the opposite approach to ours and consider ILP as a possible approach to deciding satisifiability of quantifier-free Presburger formulae. Since this decision problem is NP-complete, the authors of the study focussed on how the performance of the tools varied for different classes of formulae. The authors compare the performance of Boigelot's automata-based tool [7] and another automata based tool (SMV, introduced in [16]) with LP_SOLVE (a simplex-based open source tool), CPLEX (a commercial simplex-based linear programming tool) and OMEGA (a tool based on the Fourier-Motzkin alogirthm, [22]). Tests were run on randomly generated relatively small quantifier-free Presburger formulae. The parameters in the tests were the number of variables in the formulae, the number of atomic formulae, and the maximum value of the coefficients. The results are summarized below.

- *ILP* [8] tools were able to successfully complete runs for formulae with up to 20 atomic formulae. In general, no increase in run-time was noticed when the number of atomic formulae or the number of variables increased. However, once the coefficients became larger than approximately 10^7, many failures and overflows occurred.
- *OMEGA* exhibited similar performance to the ILP methods but incurred more segmentation faults when the values of the coefficients approached the limits of integer or float representation in the computer used.
- *Automata* techniques successfully completed runs for formulae with up to 20 atomic formulae[9]. However, they exhibited an exponential increase in run-time with an increase in the number of variables. Formulae with coefficients up to 2^{30} were handled successfully with little change in run-time[10].

An important observation of this study is that the automata techniques outperform both the ILP tools and OMEGA for formulae which have real vector solutions but whose integer vector solutions are either non-existent or sparse. A statistical analysis of ILP problems used in practice could shed light on which class of problems ILP toolsets should optimize for. An alternate approach would be to use some of the highly-tuned classes of test problems instead of randomly generating cases. A source for such problems is [3].

However, the run-time results of this study do not necessarily paint a full picture. OMEGA and the ILP tools use the native computer arithmetic for their calculations. Hence, their algorithm can run much more quickly but may have significant round-off errors or overflows. These sorts of errors may be hard to detect. Hence, systems which run ILP

[8] LP_SOLVE and CPLEX had similar performance records and hence are included under a single heading.

[9] LASH could handle formulae with 20 atomic formulae, whereas SMV could not complete runs with fewer atomic formulae.

[10] SMV performed much better than LASH in this case, perhaps due to differences in implementing the transition relations.

problems and are safety critical may not be willing to risk such faults.

3. MILP

Mixed integer linear programming is a framework in which we have constraint equations with variables that may vary over the real numbers or over the integers. As mentioned earlier, these types of systems are used to represent hybrid systems (in which a continuous system is interacting with a discrete program, inducing both continuous and logical constraints).

Analogous to the role Presburger arithmetic played in expressing ILP systems, the quantifier-free fragment of the first order theory of the real numbers with integers as a distinguished subset and with addition and order (hereafter denoted as $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$) suffices to formulate MILP questions. It has been shown that this theory is decidable (via model theoretic techniques by Zakon and Robinson, and by quantifier elimination by Weispfenning[24]). Note that atomic formulae in this theory are of the form

$$a_1 x_1 + \cdots + a_n x_n = c$$

or

$$a_1 x_1 + \cdots + a_n x_n \leqslant c$$

where $(a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$. Other inequality relations may be expressed as complements of these atomic formulae, perhaps after multiplying each of the constants by -1.

3.1 Automata Techniques for MILP

There are several challenges to be tackled when extending the automata techniques to the first order theory of $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$. We must formulate a tractable encoding scheme to allow for representation of real vectors. Then, we must describe the automata which will be capable of recognizing infinite words. Finally, we define the translation for formulas in the mixed linear arithmetic theory to automata and explore the graph theoretic algorithm for emptiness. The following algorithm follows closely the one presented in [5], [8].

3.1.1 Binary Encodings of Real Vectors

Real vectors with n components are encoded via their infinite binary representation such that

$$\mathbf{w} = \mathbf{w_I} * \mathbf{w_F}$$

where $\mathbf{w_I} \in (\{0,1\}^n)^*$ and $\mathbf{w_F} \in (\{0,1\}^n)^\omega$. Note that the integer part length of the encodings of each component of the vector must be the same. Negative numbers are represented with 2's complement notation.

Observe that there are infinitely many such encodings for each vector. One cause for the multiple representations is the possibility of repeating the sign bit arbitrarily many times. Another is the fact that any fraction which may be represented with finitely many bits has two encodings: one which has infinitely many zeroes as its tail, and one which has infinitely many ones as its tail. As in the finite words case, this algorithm generalizes easily to base r.

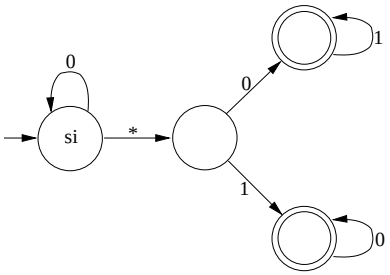

Figure 1: A RVA representing $\{\frac{1}{2}\}$.

3.1.2 Automata on Infinite Words

Since we encode real vectors as infinite words, the automata which serve as translations for arithmetic formulae must be able to accept or reject infinite strings.

Definition 3. A **Büchi automaton** (BA) [11], [12] is a tuple

$$(S, \Sigma, \delta, I, F)$$

where S is a finite set of states, Σ a finite alphabet, $\delta \subset S \times \Sigma \times S$ the transition relation, $I \subset S$ the initial states, $F \subset S$ the accepting states. For $\mathcal{A}$ a BA, a **run** of $\mathcal{A}$ on input $w \in \Sigma^\omega$ is a sequence of states $s_0, s_1, s_2 \ldots$ such that $s_0 \in I$ and for each i, $(s_i, \sigma_\mathbf{i}, s_{i+1}) \in \delta$. A run of $\mathcal{A}$ is successful run visits F infinitely many times. A word $w \in \Sigma^\omega$ is **accepted** by $\mathcal{A}$ if there is some run of $\mathcal{A}$ on w which is successful. $\mathcal{L}(\mathcal{A}) \subset \Sigma^\omega$ is the set of words accepted by $\mathcal{A}$. A BA is called **deterministic** if I is a singleton set and δ is a (possibly partial) function $S \times \Sigma \to S$.

A well-known fact [11], [12] is that the set of Büchi automata recognizable languages is closed under union, intersection, projection, and complementation. However, unlike in the case of automata on finite words, this set of languages does not coincide with the languages recognizable by deterministic Büchi automata. Fortunately, we will see later that our construction stays within a class of automata which can be determinized.

However, since we would like these automata to "concentrate"on encodings of real vectors, we must restrict our attention to a particular class of Büchi automata.

Definition 4. A **Real Vector Automaton** (RVA) [5] for vectors in $\mathbb{R}^n$ is a Büchi automaton over $\Sigma = (\{0,1\} \cup \{*\})^n$ such that

1. Every word $\mathbf{w}$ accepted by the automaton is of the form $\mathbf{w_I} * \mathbf{w_F}$ where $\mathbf{w_I} \in (\{0,1\}^n)^*$ and $\mathbf{w_F} \in (\{0,1\}^n)^\omega$.
2. For every vector $\mathbf{x} \in \mathbb{R}^n$, either all encodings of $\mathbf{x}$ in binary are accepted, or none are.

An example of a RVA is included as Figure 1.

An interesting feature of the definition is that it is easy to construct a RVA for $\mathbb{Z}^n$. We include the RVA representing

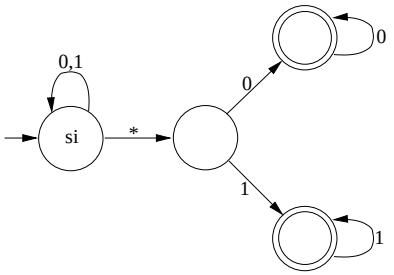

Figure 2: RVA representing $\mathbb{Z}$.

$\mathbb{Z}$ as an example. This is helpful since instead of worrying about a two-sorted system where variables range over either the reals or the integers, we can assume all variables are real-valued, and then use the RVA recognizing integers to constrain certain variables to be integers.

3.1.3 Translation

The goal of our translation algorithm is: given a formula in the first order theory of $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$, construct a RVA representing the set of all vectors in $\mathbb{R}^n$ satisfying the formula.

As in the Presburger arithmetic case, we will proceed inductively. Our main goal will be to build the RVA in two parts: one accepting the integer parts of solutions, and one accepting the fractional parts.

3.1.4 RVA for Equations

Suppose $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$ for $\mathbf{a}, c$ as before. Write $\mathbf{x} = \mathbf{x_I} + \mathbf{x_F}$. Then for any encoding $\mathbf{w}$ of $\mathbf{x}$, $\mathbf{w} = \mathbf{w_I} * \mathbf{w_F}$ and $\mathbf{w_I} * \mathbf{0}^\omega$ is an encoding for $\mathbf{x_I}$, $\mathbf{0} * \mathbf{w_F}$ is an encoding for $\mathbf{x_F}$. Define $\alpha = \Sigma_{a_i<0} a_i$, $\alpha' = \Sigma_{a_i>0} a_i$. Then since each component of the vector $\mathbf{x_F}$ is in the interval $[0, 1]$,

$$\begin{aligned} \mathbf{a} \cdot \mathbf{x_I} + \mathbf{a} \cdot \mathbf{x_F} &= c \\ \alpha \leqslant \mathbf{a} \cdot \mathbf{x_F} &\leqslant \alpha' \\ c - \alpha' \leqslant \mathbf{a} \cdot \mathbf{x_I} &\leqslant c - \alpha \end{aligned}$$

Moreover, by the gcd test for Diophantine equations,

$$\gcd(a_1, \ldots, a_n) | \mathbf{a} \cdot \mathbf{x_I}$$

Hence, the language of all encodings of all elements of the set of vectors satisfying φ may be written as

$$\mathcal{L} = \bigcup_{\beta : \chi(\beta)} (\{\mathbf{w_I} \in \Sigma^* : \mathbf{a} \cdot [\mathbf{w_I} * \mathbf{0}^\omega]_2 = \beta\} \cdot \{*^n\} \cdot \{\mathbf{w_F} \in \Sigma^\omega : \mathbf{a} \cdot [\mathbf{0} * \mathbf{w_F}]_2 = c - \beta\})$$

where

$$\chi(\beta) \equiv (\big[c - \alpha' \leqslant \beta \leqslant c - \alpha\big] \wedge (\exists m \in \mathbb{Z})(\beta = m \gcd(a_1, \ldots, a_n))).$$

Hence, the RVA representing the set of vectors satisfying φ can be decomposed as at most $(\alpha' + \alpha)$ RVA, each of which consists of

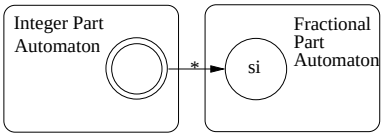

Figure 3: The decomposition of RVA.

- An automaton on finite words over $\Sigma = \{0, 1\}^n$ accepting all $\mathbf{w_I} \in \Sigma^*$ such that $\mathbf{a} \cdot [\mathbf{w_I} * \mathbf{0}^\omega]_2$ is a solution to a given linear equation.
- A transition between the two parts corresponding to reading the words containing the vector of fractional separator symbols.
- A Büchi automaton over $\Sigma = \{0, 1\}^n$ accepting all $\mathbf{w_F} \in \Sigma^\omega$ such that $\mathbf{a} \cdot [\mathbf{0} * \mathbf{w_F}]_2$ is a solution to a given linear equation.

For the integer part automaton, the techniques presented in the ILP section suffice. So, it remains to consider the fractional part automaton recognizing all solutions $\mathbf{x} \in [0, 1]^n$ to $\mathbf{a} \cdot \mathbf{x} = c - \beta$ for some β. For convenience, we write $\gamma = c - \beta$. The intuition behind the automaton we construct will be that each state represents some integer,s, with the property that any vector, $\mathbf{x} \in [0, 1]^n$, encoded by a path starting at that state and continuing infinitely long, satisfies $\mathbf{a} \cdot \mathbf{x} = s$.

Formally: since we are looking for fractional solutions only, the only states necessary are those labelled by integers in $S = [\alpha, \alpha']$. Also, according to our intended interpretation of the labels for the state, the initial state will be that labelled by γ. To completely specify the RVA, it remains only to define the transition relation. For $s \in S, \mathbf{d} \in \{0, 1\}^n$, define $s' = 2s - \mathbf{a} \cdot \mathbf{d}$ and put

$$\delta(s, \mathbf{d}) = \begin{cases} s' & \text{if } s' \in S \\ \emptyset & \text{otherwise.} \end{cases}$$

THEOREM 4. [11] *$\mathcal{A}_\varphi = (S, \Sigma, \delta, I = \{\gamma\}, F = S)$ is a Büchi automaton representing the set of solutions to φ in $[0, 1]^n$.*

PROOF. We will show that any encoding of a vector of fractional numbers satisfying φ is accepted by $\mathcal{A}_\varphi$, and that any accepting run of $\mathcal{A}_\varphi$ corresponds to a solution of φ.

For the forward direction, suppose that $\mathbf{w} \in ([0, 1]^n)^\omega$ is an encoding of a solution $\mathbf{x} \in [0, 1]^n$ to φ. Then, we will show that there is an infinite run of $\mathcal{A}_\varphi$ on $\mathbf{w}$. Observe that it suffices to prove that at each state of the run (where the current state is s and the current input is $\mathbf{w_i}$), $2s - \mathbf{a} \cdot \mathbf{w_i} \in [\alpha, \alpha']$. This is enough since it is exactly in this case that the transition function is defined. The proof is by (strong) induction and we prove the stronger statement that

1. At each state of the run, there is a transition out of this state via the current input,

[11] This theorem is mentioned in neither [5] not [8] but as it is very important for the soundness of this translation algorithm, we work it through and include it here.

2. and at each state s of the run, if s' is the next state we transition to (by above) and if $\mathbf{y}, \mathbf{z}$ are encoded by paths labelled by bits in $\mathbf{w}$ starting from s, s' (resp.), then if $\mathbf{a} \cdot \mathbf{y} = s$, $\mathbf{a} \cdot \mathbf{z} = s'$.

In the base case, the current state is γ and the current input is the MSB, $\mathbf{w_0}$, of an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = \gamma$. So,

$$\frac{1}{2}\mathbf{a} \cdot \mathbf{w_0} + \mathbf{a} \cdot [\mathbf{0} * \mathbf{0}\mathbf{w}^{tail}]_2 = \gamma.$$

Note that the maximum value of the scalar product of $\mathbf{a}$ and the vector containing all bits other than the MSB of $\mathbf{w}$ (i.e. the tail) occurs when the the components corresponding to positive elements in $\mathbf{a}$ have maximum value, and components corresponding to non-positive elements in $\mathbf{a}$ have zero value. Similarly, the minimum value of the scalar product occurs when components of the tail corresponding to negative elements in $\mathbf{a}$ have maximum value, and all other components have zero value. And, the maximum value of a binary fraction with zero MSB is $\frac{1}{2}$. Hence,

$$2\gamma - \mathbf{a} \cdot \mathbf{w_0} = 2\mathbf{a} \cdot [\mathbf{0} * \mathbf{0}\mathbf{w}^{tail}]_2$$
$$\alpha \leqslant 2\gamma - \mathbf{a} \cdot \mathbf{w_0} \leqslant \alpha'.$$

Therefore, there is a transition out of the initial state on the MSB of w.

Let $s_1 = \delta(\gamma, \mathbf{w_0})$. We observe that any path from s_1 is labelled by a word $\mathbf{u}$ such that if $\mathbf{v}$ labels a path starting at γ and going to s_1, $\mathbf{v} = \mathbf{w_0}\mathbf{u}$. Then if we treat $\mathbf{v}, \mathbf{u}$ as fractional binary representations of $\mathbf{y}, \mathbf{z}$ respectively, $\mathbf{y} = \frac{1}{2}(\mathbf{w_0} + \mathbf{z})$. If $\mathbf{a} \cdot \mathbf{y} = \gamma$,

$$\mathbf{a} \cdot \mathbf{z} = 2\mathbf{a} \cdot \mathbf{y} - \mathbf{a} \cdot \mathbf{w_0} = 2\gamma - \mathbf{a} \cdot \mathbf{w_0} = s_1.$$

Thus, if $\mathbf{y}$ is encoded by a path starting at γ and is a solution to $\mathbf{a} \cdot \mathbf{y} = \gamma$ and if $\mathbf{z}$ is encoded by the same path but starting at s_1 (i.e. truncating its first bit), then $\mathbf{z}$ satisfies $\mathbf{a} \cdot \mathbf{z} = s_1$.

For the inductive step, we suppose that the current state of $\mathcal{A}_\varphi$ is some s_m and the current input bit is $\mathbf{w_m}$. Then we assume that at each previous state there was a transition out of the state on the input bit. Hence we have a sequence $\gamma, s_1, s_2, \ldots, s_m$ where the path through this sequence is labelled by the first m bits of $\mathbf{w}$. Moreover, at each previous state, we assume (for the induction) that there is a relationship between words starting at those paths and satisfiability of a linear equation whose left hand side is $\mathbf{a} \cdot \mathbf{x}$. The current input is the $(m+1)$-st bit, $\mathbf{w_m}$, of an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = \gamma$. By the inductive hypothesis, it is also the MSB of an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = s_m$. Once this is realized, the same reasoning as in the base case show that $\alpha \leqslant 2s_m - \mathbf{a} \cdot \mathbf{w_m} \leqslant \alpha'$ and therefore there is a transition out of s_m allowed by δ. Let $s_{m+1} = \delta(s, \mathbf{w_m})$. Again, we apply analogous reasoning to the argument in the base case to get the second part of the inductive claim.

Hence, the induction holds and we have shown that if $\mathbf{w}$ is an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = \gamma$ then there is a run of $\mathcal{A}_\varphi$ on $\mathbf{w}$ which is successful.

Conversely, suppose $r = \gamma, s_1, s_2, \ldots$ is a run of $\mathcal{A}_\varphi$ on $\mathbf{w} \in (\{0,1\}^n)^\omega$ which is successful. We would like to prove that $\mathbf{w}$ encodes a solution to $\mathbf{a} \cdot \mathbf{x} = \gamma$. By definition of success of a run on a Büchi automaton and since the set accepting states is finite, there is some state which is visited infinitely many times during r.

A lemma will be helpful here: If there is a path labelled by $\mathbf{d_0}, \ldots, \mathbf{d_m}$ between states s, s', then if $\mathbf{d}, \mathbf{d}'$ are words labelling paths starting at s, s' (resp.) such that $\mathbf{d} = \mathbf{d_0} \ldots \mathbf{d_n}\mathbf{d}'$ and if $\mathbf{y}, \mathbf{z}$ are the vectors represented by $\mathbf{d}, \mathbf{d}'$ (resp.) then

$$(\mathbf{a} \cdot \mathbf{y} - s) = \frac{\mathbf{a} \cdot \mathbf{z} - s'}{2^{,+1}}.$$

Again, we prove the lemma by induction. For $m = 0$, then $\mathbf{y} = \frac{1}{2}(\mathbf{z} + \mathbf{d_0})$ so $\mathbf{a} \cdot \mathbf{y} = \frac{1}{2}(\mathbf{a} \cdot \mathbf{z} + \mathbf{a} \cdot \mathbf{d_0})$. Moreover, by definition of the transition relation, $s' = 2s - \mathbf{a} \cdot d_0$. Hence, $\mathbf{a} \cdot \mathbf{y} - s = \frac{1}{2}(\mathbf{a} \cdot \mathbf{z} - s')$. For the inductive step, suppose that for all $k \leqslant m$, if $\mathbf{d_0}, \ldots, \mathbf{d_k}$ is a path between states s'', s' and if $\mathbf{d}'', \mathbf{d}'$ are words labelling paths starting at s'', s' (resp.) such that $\mathbf{d}'' = \mathbf{d_0} \ldots \mathbf{d_k}\mathbf{d}'$ and if $\mathbf{y}, \mathbf{t}$ are the vectors represented by $\mathbf{d}'', \mathbf{d}'$ (resp.) then

$$(\mathbf{a} \cdot \mathbf{y} - s'') = \frac{\mathbf{a} \cdot \mathbf{t} - s'}{2^{k+1}}.$$

Then consider a path labelled by $\mathbf{d_0}, \ldots, \mathbf{d_{m+1}}$ between states s, s'. Let d, d' be words labelling paths starting at s, s' (resp.) such that $\mathbf{d} = \mathbf{d_0} \ldots \mathbf{d_{n+1}}\mathbf{d}'$ and let $\mathbf{y}, \mathbf{z}$ be the vectors represented by d, d' (resp.). Let $\mathbf{d}'' = \mathbf{d_{m+1}}\mathbf{d}'$ be a path starting at state s'' and let $\mathbf{t}$ be the vector represented by $\mathbf{d}''$. Then $\mathbf{t} = \frac{1}{2}(\mathbf{d_{m+1}} + \mathbf{z})$. The definition of the transition relation gives that $\mathbf{a} \cdot \mathbf{d_{m+1}} = 2s'' - s'$. Hence, $\mathbf{a} \cdot \mathbf{t} - s'' = \frac{1}{2}(\mathbf{a} \cdot \mathbf{z} - s')$. Since s'' lies on a path of length m from s, we apply the inductive hypothesis to get

$$\mathbf{a} \cdot \mathbf{z} - s' = 2(\mathbf{a} \cdot \mathbf{t} - s'') = 2(2^{m+1}(\mathbf{a} \cdot \mathbf{y} - s))$$

as required.

Why did we prove this lemma? Well, let s' be the state in S which is visited infinitely many times during r. Then there are infinitely many indices m_i for which $s' = s_{m_i}$. But, by the lemma, for each such m_i, if we let y, z be the vectors represented by the path of the run starting at γ, s_{m_i} (resp.)

$$(\mathbf{a} \cdot \mathbf{y} - \gamma) = \frac{\mathbf{a} \cdot \mathbf{z} - s'}{2^{m_i+1}}.$$

Since m_i gets arbitrarily large and

$$|\mathbf{a} \cdot \mathbf{z} - s'| \leqslant |\mathbf{a} \cdot \mathbf{z}| + |s'| \leqslant 2\alpha'$$

it must be the case that $\mathbf{a} \cdot \mathbf{y} - \gamma = 0$, i.e. that $\mathbf{y}$, the vector encoded by the run r is a solution to φ, as required. □

So, we conclude that the algorithm presented in [5] correctly yields Büchi automata corresponding to linear equations with real variables.

There are some design details involved in implementing this RVA. Since the body of the automaton (the set of states and the transition relation) depends only on the left hand side of φ and hence is independent of β, the states and relations for each of the integer and fractional parts of the automaton corresponding to different values of β can be shared. Then, the transition on the fractional separator connects those parts of the automaton that correspond to β being the integer part

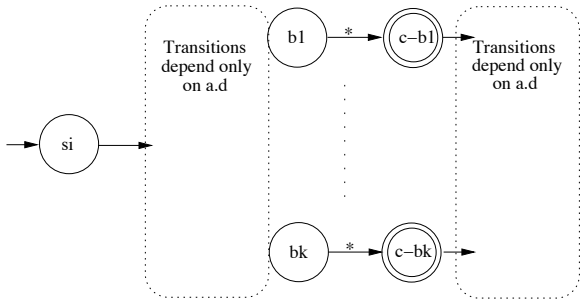

Figure 4: Shared states for automata corresponding to different values of the right hand side.

and $c - \beta$ being the fractional part. The following figure is a picture of how the sharing of states works.

A final note about efficiently implementing the above algorithm: since the gcd algorithm can be quite time-expensive, it may not be wise to compute it. Hence, we could define

$$\chi(\beta) \equiv \left(c - \alpha' \leqslant \beta \leqslant c - \alpha\right)$$

and have perhaps more transitions between the integer and fractional part. The benefit of such an optimization would depend on the size of the coefficients $a_1, \ldots, a_n, c$.

3.1.5 RVA for Inequalities

The construction in this case is very similar to what we saw above for equalities. Let $\varphi \equiv a_1 x_1 + \cdots + a_n x_n \leqslant c$, which we abbreviate as $\mathbf{a} \cdot \mathbf{x} \leqslant c$. We decompose the set of all encodings of solutions in the same way:

$$\mathcal{L} = \bigcup_{\beta : \chi(\beta)} (\{\mathbf{w_I} \in \Sigma^* : \mathbf{a} \cdot [\mathbf{w_I} * \mathbf{0}^\omega]_2 \leqslant \beta\} \cdot \{*^n\} \cdot \{w_F \in \Sigma^\omega : \mathbf{a} \cdot [\mathbf{0} * \mathbf{w_F}]_2 \leqslant c - \beta\})$$

where

$$\chi(\beta) \equiv \left(c - \alpha' \leqslant \beta \leqslant c - \alpha\right).$$

Again, we can construct the RVA by computing finite automata on finite words accepting encodings of integer solutions to some linear inequality, and concatenating with Büchi automata representing sets of vectors in $[0,1]^n$ which satisfy some linear inequality. The finite automata on finite words were discussed in an earlier section. It suffices to modify the construction of Büchi automata for linear equalities to get the Büchi automata accepting the fractional parts. Define $\mathcal{A}_\varphi$ as the Büchi automaton representing solutions to $\mathbf{a} \cdot \mathbf{x} = \gamma$ in $[0,1]^n$ by

$$\mathcal{A}_\varphi = (S = [\alpha, \alpha'] \cap \mathbb{Z}, \{0,1\}^n, \delta, \{\gamma\}, S)$$

where for each $s \in S, \mathbf{d} \in \{0,1\}^n$, put $s' = 2s - \mathbf{a} \cdot \mathbf{d}$ and

$$\delta(s, \mathbf{d}) = \begin{cases} s' & \text{if } s' \in S \text{ or } s' > \alpha' \\ \emptyset & \text{otherwise.} \end{cases}$$

A partial proof of the correctness of this automaton is below:[12] Given $s, s' \in S$ connected by transition on $\mathbf{d}$, let $\mathbf{w}, \mathbf{w}'$ be words labelling paths starting at s, s' (resp.) and $\mathbf{x}, \mathbf{x}'$ the vectors encoded by $\mathbf{w}, \mathbf{w}'$ (resp.). Then $\mathbf{w} = \mathbf{d}\mathbf{w}'$ and hence $\mathbf{x} = \frac{1}{2}(\mathbf{d} + \mathbf{x}')$. So if $\mathbf{a} \cdot \mathbf{x} \leqslant s$,

$$\begin{aligned} \frac{1}{2}(\mathbf{d} + \mathbf{x}') &= \mathbf{a} \cdot \mathbf{x} \leqslant s \\ \frac{1}{2}(2s - s' + \mathbf{x}') &\leqslant s \\ \mathbf{a} \cdot \mathbf{x}' &\leqslant s'. \end{aligned}$$

This supports the intuition that for each $s \in S$, the vectors encoded by infinite paths starting from s satisfy $\mathbf{a} \cdot \mathbf{x} \leqslant s$. For $s' > \alpha'$, note that since for $\mathbf{x} \in [0,1]^n$, the set of solutions to $\mathbf{a} \cdot \mathbf{x} \leqslant s'$ is the same as the set of solutions to $\mathbf{a} \cdot \mathbf{x} \leqslant \alpha'$. A full proof of correctness is similar to that for equations given above. Hence, it is not included here.

Once we have defined the integer and fractional part of the RVA for inequalities, we form the full RVA by concatenating the two part (with a transition labelled by the fractional separator), and allowing multiple use of states and transitions as before.

3.1.6 Weak Büchi Automata and RVA

Before we generalize the constructions above to arbitrary formulae in $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$, let us examine the structure of the RVA so far.

Definition 5. A Büchi automaton is **weak** if there is a partition of its state set S into disjoint subsets $Q_1, \ldots, Q_m$ such that

1. for each Q_i either $Q_i \subset F$ or $Q_i \cap F = \emptyset$, and
2. there is a partial order on $\{Q_1, \ldots, Q_m\}$ such that $Q_j \leqslant Q_i$ if Q_j is reachable from Q_i (i.e. if there is a path in the underlying digraph of the Büchi automaton from some state in Q_i to some state in Q_j).

THEOREM 5. *[6][13] The RVA constructed for equalities and inequalities with real-valued free variables are weak Büchi automata.*

PROOF. Partition the states S into strongly connected components[14]. Note that each strongly connected component will be either entirely within the integer part of the RVA or entirely within the fractional part of the RVA. By definition, all nodes in the integer part are non-accepting and all nodes in the accepting part are accepting. Hence, the first requirement is met. To fulfill the second condition, consider the reachability relation on the strongly connected components. By definition of strongly connected, this relation is reflexive, anti-symmetric, and transitive. Hence, it is a partial order and respects reachability. □

Therefore, we may now exploit beneficial features of weak Büchi automata.

[12] Again, the authors in [5] did not provide any discussion of correctness.

[13] This theorem was stated but not proved in [6].

[14] Strongly connected components are maximal subgraphs in which each node is reachable from any other node. We allow reachability if paths are of zero length, and hence all nodes are in some strongly connected component.

Definition 6. A **co-Büchi automaton** is a finite state automaton on infinite strings whose runs are successful if they infinitely often avoid the set of accepting states.

Fact: A weak Büchi automaton can be represented as an equivalent (i.e. accepting the same set of words) weak co-Büchi automaton. This is done by flipping the accepting/ non-accepting status of each state.

THEOREM 6. *Weak Büchi automata can be determined by a "breakpoint"construction. ([20], [19])*

PROOF. Let $\mathcal{A} = (S, \Sigma, \delta, I, F)$ be a weak Büchi automaton and let $\mathcal{A}' = (S, \Sigma, \delta, I, F' = S \setminus F)$ be the equivalent weak co-Büchi automaton. Define $\mathcal{A}'' = (S'', \Sigma, \delta'', I'', F'')$ to be

- $S'' = 2^S \times 2^S$, $I'' = \{(I, \emptyset)\}$, $F'' = 2^S \times \{\emptyset\}$
- For $(Q, \emptyset) \in S''$, $a \in \Sigma$: $\delta''((Q, \emptyset), a) = (T, T \setminus F')$. For $(Q, R) \in S''$ with $R \neq \emptyset$, $a \in \Sigma$: $\delta''((Q, R), a) = (T, U \setminus F')$. Where,

$$T = \{p \in S : \exists q \in Q\,(p \in \delta(q, a))\}$$

$$U = \{p \in S : \exists r \in R\,(p \in \delta(r, a))\}.$$

Note that this is a deterministic automaton. Also, when $\mathcal{A}''$ is in state (Q, R), R is the set of states of $\mathcal{A}$ which are reachable in $\mathcal{A}'$ by a run whose corresponding run of $\mathcal{A}''$ hasn't passed through a state in F' since the last breakpoint (a state of the form $(Q, \emptyset)$) .

It now follows readily (by tracing runs of corresponding automata and recalling definitions) that $\mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. □

Unfortunately, the procedure illustrated in the theorem above does not in general yield a weak automaton. However, we have the following theorem from [6]:

THEOREM 7. *Every deterministic RVA representing a set definable in $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$ is inherently weak, i.e. has no reachable strongly connected components of its transition graph which have both accepting and non-accepting cycles.*

PROOF. (Sketch of proof in [6]) Given a deterministic automaton representing a set definable in the theory above, call the set it represents S. Then $S \in F_\sigma \cap G_\delta$ in the Euclidean topology. It follows that the set of all words encoding S, $(L)(S)$ is in the same topological class over the ω-word topology. Finally, since the automaton is deterministic and its language is in this particular topological class, it is inherently weak. □

Recall that our goal is to give automata representing sets of vectors satisfying first order formulae in $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$. Hence, the theorem above applies for all the automata we construct. Moreover, given a deterministic inherently weak Büchi automaton, it is easy to transform it to a weak Büchi automaton: for each state, if it is in a strongly connected component with at least one accepting state, add it to the set of accepting states.

3.1.7 RVA for General Formulae

Equipped with the theoretical tools of the previous section, we can generalize our construction of RVA to arbitrary formulae [8]. For atomic formulae, we use the explicit constructions discussed earlier to get weak RVA. For the Boolean operations, we assume by induction that we have constructed the RVA accepting the sets of vectors satisfying the subformulae and hence need only worry about how to combine these RVA to represent the main connective. For conjunctions and disjunctions, we need RVA recognizing the intersection and union (resp.) of sets of real vectors. Note that it suffices to define RVA recognizing the intersection and union of the sets of encodings of real vectors. This may be done by the product construction[15], which preserves the weak nature of the automata.

For formulae whose main connective is negation, we need a RVA recognizing the complement of a set of real vectors. The first step is to determinize the given RVA using the "breakpoint construction". By the topological theorem in the previous section, the resulting automaton is inherently weak and hence can easily be made weak. Then we complement the deterministic weak automaton by switching the accepting / non-accepting status of each state[16]. This ensures that no encodings of vectors which were earlier accepted are now accepted. However, the automaton may accept words which do note encode any vector. Hence, to yield a RVA we intersect the automaton with the RVA for $\mathbb{R}^n$.

Last, we consider formulas whose main operation is quantification. As in the Presburger arithmetic case, if the quantifier is universal, we replace the formula with its equivalent negated existential form. For an existential formula, we first remove from each transition label the symbol corresponding to the variable to be projected out. This may (and probably will) destroy any determinism the automaton has, but will not affect its weak nature. Then (again, as in the finite words case), we modify the resulting automaton to ensure that it accepts all encodings of any vector it accepts. Recall that this amounts to adding transitions from the initial state to some of the states already reachable from it. Hence, the resulting RVA remains weak.

[7] also includes a (sequentialized) implementation of the algorithm translation formulae of $(\mathbb{R}, \mathbb{Z}, +, \leqslant, 0, 1)$ to RVA.

3.1.8 Satisfiability and Solutions

Given a RVA representing a set of real vectors, we can use techniques for checking emptiness of the language of the RVA to check if there are any vectors in the set. Since the

[15]The simple product construction does not work in general for Büchi automata, but since weak automata are very similar to finite automata on finite words, it is correct in this case.

[16]Again, this mirrors the complementation algorithm for deterministic automata on finite words.

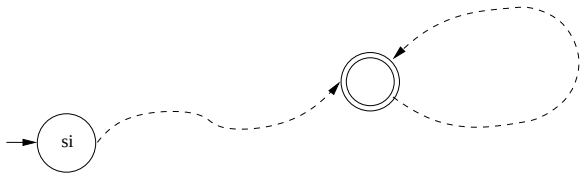

Figure 5: A lasso in a Büchi automaton.

RVA is a Büchi automaton, we can use Büchi's characterization theorem [12] which says that the language of a Büchi automaton is not empty if and only if there is an (I, F)*lasso* in the underlying digraph.

Definition 7. For a directed graph $G = (V, E)$ and for $A, B \subset V$, an (A, B) **lasso** is a path from some node $a \in A$ to a node $b \in B$ and then a loop from b to itself.

In the context of automata, an (I, F) lasso means that there is an infinite path allowed by the transition relation and starting from some initial state which visits an element in the accepting set infinitely often.

Checking for the existence of an (I, F) lasso can be done efficiently. An algorithm is:

- `function accepting-path(state` s`)`

  `var` (s_1, a, s_2)`;`
  `var` $list$ `=` $null$`;`
  `if` $s \in F$ `do return` s`;`
  `for each` $(s_1, a, s_2) \in \delta$ `such that` $s_1 = s$ `do`
  `if (accepting-path(`s_2`)!=` $null$`)`
  `then do add (accepting-path(`s_2`),` $list$`);`
  `return` $list$`;`

- `function accepting-loop(state` s`)`

  `var` (s_1, a, s_2)`;`
  `if` $s \notin F$ `do return` $False$`;`
  `for each` $(s_1, a, s_2) \in \delta$ `such that` $s_1 = s$ `do`
  `if (`$s_1 \in$ `accepting-path(`s_2`))`
  `then do return` $True$`);`
  `return` $False$`;`

- `function hasLasso()`

  `for each` $s \in I$ `do`
  `if (accepting-path(`s`)!=null) then {`
  `for each` $s' \in$ `accepting-path(`s`) do`
  `if accepting-loop(`s'`) return` $True$`;`
  `}`
  `return` $False$`;`

Moreover, the algorithm to check for such a lasso gives an example of an infinite word accepted by the automaton, if one exists. In fact, this infinite word encodes a rational solution to the formula[17].

The enumeration problem which was so important in the integer case has a different flavour for MILP. Regardless, bounds on feasible values of the variables can still be found using similar techniques to what we did earlier.

3.2 Other Approaches to MILP

The projection and elimination method has recently been extended to the mixed-integer case [2]. This approach uses a predicate to indicate whether a variable ranges over integers only and then combines the Fourier-Motzkin and Omega tests to project variables out one-by-one. One implementation of this algorithm is integrated into the Cooperating Validity Checker [15]. This particular implementation includes the feature of proof-production: if the checker claims there is no solution to a particular system, it gives a proof of this claim which can be externally verified. Such proofs help in identifying false positive replies of the decision procedure, but still do not give an example of a solution.

3.3 Results and Evaluation

The compact representation that RVA give to the set of solutions of MILP formulae is attractive. However, there have been no comparative studies of the automata approach to other MILP solvers (partially because not many have been implemented). Such a study could go a long way towards understanding the strengths and weaknesses to the various approaches in solving the question of satisfiability of MILP problems. Barring large scale performance differences, the automata approach seems to have the upper hand in being constructive.

4. CONCLUSIONS

We have presented an automata solution to problems arising in Integer (and Mixed-Integer) Linear Programming. In fact, we worked in greater generality and saw that there is an algorithm for translating any formula in Presburger arithmetic (resp. mixed real-integer linear arithmetic) into an automaton accepting all and only encodings of vectors satisfying the formula. We proved that the algorithm is correct and discussed some optimizations for its implementation.

Moreover, we saw that the nature of automata easily leads to constructive solutions for the satisfiability question. In the integer case, we used this to address the problem of enumerating the set of solutions to a formula. This algorithm had not been presented before, so it would be intereting to implement it and compare its performance with, for instance, the Omega system.

Finally, we considered other approaches to (M)ILP. It seems that different methodologies perform better for different classes of formulae. In particular, the automata approach appears to win out for sparse feasibility regions. However, these results are preliminary and further study should be done.

[17]A linear formula with integer coefficients always has some rational solution so long as the free variables are allowed to take values in a superset of the integers.

5. REFERENCES

[1] C. Ancourt and F. Irigoin. Scanning polyhedra with do loops. In *Proceedings of the third ACM SIGPLAN symposium on principles and practice of parallel programming*. ACM SIGPLAN Notices, April 1991.

[2] S. Berezin, V. Ganesh, and D. L. Dill. Online proof-producing decision procedure for mixed-integer linear arithmetic. In *Proceedings of 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, April 2003.

[3] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. Miplib 3.0: Pure and mixed integer programs. *Available at* `www.caam.rice.edu/~bixby/miplib/miplib.html`.

[4] B. Boigelot. Symbolic methods for exploring infinite state spaces, phd thesis. *Collection des Publications de la Faculté des Sciences Appliquées de l'Université de Liège*, 189, 1999.

[5] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems (extended abstract). In *Proceedings of 9th International Conference on Computer-Aided Verification*, pages 167–177. Lecture Notes in Computer Science, June 1997.

[6] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proceedings of 1st International Joint Conference on Automated Reasoning*, pages 611–625. Lecture Notes in Artificial Intelligence, June 2001.

[7] B. Boigelot, L. Latour, and A. Legay. Lash: The liège automata-based symbolic handler. *Available at* `www.montefiore.ulg.ac.be/~boigelot/research/lash`.

[8] B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata: An overview. In *Proceedings of 18th International Conference on Logic Programming*, pages 1–19. Lecture Notes in Computer Science, July 2002.

[9] A. Boudet and H. Comon. Diophantine equations, presburger arithmetic and finite automata. In *Proceedings of Colloquium on Trees in Algebra and Programming*, pages 30–43. Lecture Notes in Computer Science, 1996.

[10] R. Brinkmann and R. Drechsler. Rtl-datapath verification using integer linear programming. In *Proceedings of the 2002 conference on Asia South Pacific design automation/ VLSI design*, January 2002.

[11] J. Buchi. Weak second-order arithmetic and finite automata. *Zeitschrift Math. Logik und Grundlagen det Mathematik*, pages 66–92, 1960.

[12] J. Buchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress in Logic, Method and Philosophical Sciences, 1960*, pages 1–12, 1962.

[13] K. Chakrabarty. Design on system-on-a-chip test access architectures using integer linear programming. In *Proceedings of the 18th IEEE VLSI Test Symposium*, April 2000.

[14] D. Cooper. Theorem-proving in arithmetic without multiplication. *Machine Intelligence 7*, 1972.

[15] D. L. Dill, S. Berezin, and C. Barrett. Cvc lite: Cooperating validity checker. *Available at* `verify.stanford.edu/CVC`.

[16] V. Ganesh, S. Berezin, and D. L. Dill. Deciding presburger arithmetic by model checking and comparisons with other methods. In *Proceedings of 3rd Formal Methods of Computer-Aided Design*. Lecture Notes In Computer Science, November 2002.

[17] B. Khoussainov and A. Nerode. *Automata Theory and its Applications*. Birkhauser, Boston, Massachusetts, 2001.

[18] F. Klaedtke. On the automata size for presburger arithmetic. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 110–119. IEEE Computer Society Press, 2004.

[19] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *Israeli Symposium on Theory of Computing and Systems*, pages 111–131. ACM Transactions on Computational Logic, 1997.

[20] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[21] M. Presburger. Uber die vollstandigkeit eines gewissen systems der arithmetic ganzer zahlen, in welchem die addition als einzige operation hervortritt. *Compte-Rendus des Congres des Math. des pays Slavs*, 1929.

[22] W. Pugh. The omega test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, pages 102–114, August 1992.

[23] C. R. Reddy and D. W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of 10th annual ACM symposium on Theory of Computing*. ACM, May 1978.

[24] V. Weispfenning. Mixed real-integer linear quantifier elimination (extended version). In *Proceedings of International Symposium on Symbolic and Algebraic Computation '99*, pages 129–136. ACM Press, 1999.

[25] P. Wolper and B. Boigelot. An automata-theoretic approach to presburger arithmetic constraints (extended abstract). In *Proceedings of 2nd Static Analysis Symposium*, pages 21–32. Lecture Notes in Computer Science, September 1995.

[26] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–19. Lecture Notes in Computer Science, March 2000.

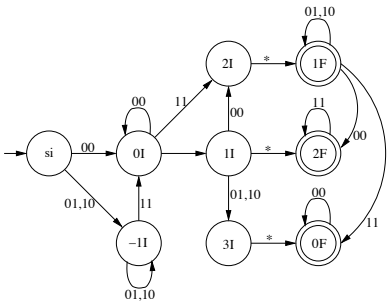

Figure 6: A RVA representing the set of solutions to $x + y = 3$.

6. ACKNOWLEDGMENTS

I thank my advisor, Anil Nerode, for suggesting this line of study. Also, thanks to Keshav Pingali for his insight on current and relevant applications to compiler research. Thanks as well to Todd Kemp for his help with the graphics.

APPENDIX

We include (in Figure 6) a fully worked example of a RVA representing the equation $x+y = 3$. Note the decomposition into integer and fractional parts, and the reuse of states and transitions.