Representing Arithmetic Constraints with Finite Automata: An Overview*

Bernard Boigelot and Pierre Wolper

Université de Liège,
Institut Montefiore, B28,
4000 Liège, Belgium
{boigelot,pw}@montefiore.ulg.ac.be,
http://www.montefiore.ulg.ac.be/~{boigelot,pw}/

Abstract. Linear numerical constraints and their first-order theory, whether defined over the reals or the integers, are basic tools that appear in many areas of Computer Science. This paper overviews a set of techniques based on finite automata that lead to decision procedures and other useful algorithms, as well as to a normal form, for the first-order linear theory of the integers, of the reals, and of the integers and reals combined. This approach has led to an implemented tool, which has the so far unique capability of handling the linear first-order theory of the integers and reals combined.

1 Introduction

Linear numerical constraints, i.e. constraints involving only addition or multiplication by constants, are a basic tool used in many areas of Computer Science and other disciplines. There is thus an abundance of algorithms and tools dealing with linear constraints, which mostly are geared to efficiently solving consistency and optimization problems. The power of linear constraints can be significantly enhanced if they are incorporated in a first-order theory allowing Boolean operations and quantification. But, this comes at the price of higher complexity, and tools handling the full first-order theory are less common, especially when the constraints are defined over the integers, the latter case corresponding to Presburger arithmetic, a decidable but two exponential-space complete theory. This limited number of tools was, until the approach described in this paper, even a complete absence when moving to the first-order theory of linear constraints over the reals and integers *combined*, i.e. involving both variables ranging over the reals and variables ranging over the integers.

The work overviewed here was motivated by problems related to the symbolic exploration of infinite state spaces [WB98], for which handling nonconvex and periodic constraints over the integers was essential. A general Presburger tool

* This work was partially funded by a grant of the "Communauté française de Belgique - Direction de la recherche scientifique - Actions de recherche concertées" and by the European IST-FET project ADVANCE (IST-1999-29082).

(for instance [Pug92]) was in principle sufficient for the task. However, the need to very frequently check implication of formulas, as well as the success of Binary Decision Diagrams (BDDs) [Bry86] for similarly dealing with Boolean formulas in the symbolic exploration of finite state spaces [BCM$^+$90], prompted us to search for a related representation of arithmetic formulas. The long known fact that by encoding integers as binary (or in general r-ary) strings, Presburger arithmetic can be handled by finite automata [Büc60], naturally pointed to finite automata as a tool for dealing with arithmetic, and to minimized finite automata as a normal form akin to BDDs. From a theoretical point of view this was all pretty obvious, what still needed to be done was to turn this idea into a working technology. The first step towards this was a careful choice of the coding of numbers by strings, for instance using r's complement for negative numbers and sequentializing the bits of vectors (see Section 3). Second, the development of specific algorithms, e.g. for generating automata directly from equations and inequations [BC96,WB00] was very helpful. Finally, an efficient package for dealing with finite automata, the LASH tool [LAS] was developed. Besides providing a Presburger tool, LASH also included some specific components to deal with state-space exploration, for instance an algorithm for computing (when possible) the effect of iterating a linear transformation [Boi98].

If automata on finite words can handle integer arithmetic, it follows almost immediately that automata on infinite words [Tho90] can handle real arithmetic. This has also been long known, but turning this attractive idea into a technology was substantially more difficult since manipulating automata on infinite words requires less efficient and harder to implement algorithms than for automata on finite words. However, it turns out that handling real linear arithmetic with automata does not require the full power of infinite-word automata, but can be done with a very restrictive class, namely deterministic weak automata [Sta83,MSS86,MS97]. This was shown using topological arguments in [BJW01] with two important consequences: algorithms very similar to those used for finite-word automata can be used for manipulating this class, and it admits a easily computable reduced normal form [Löd01]. Now, since it is very easy to express that a number is an integer with an automaton (its fractional part is 0), the automata-theoretic approach to handling real arithmetic can also cope with the theory in which both real and integer variables are allowed. This makes it for example possible to represent an infinite periodic set of dense intervals, something that is beyond the linear first-order theory of the reals alone. Potential applications include, for instance, the analysis of some classes of hybrid systems [BBR97].

The goal of this paper is to present an overview of the theory and pragmatics of handling integer and real arithmetic with automata. Since the case of pure integer arithmetic is simply the restriction of the real case to finite words, only the more general latter case is presented, simplifications that occur in the pure integer case being mentioned. After a section recalling the necessary definitions about automata on infinite words, the encoding scheme by which a set of real vectors can be represented by a finite automaton accepting a set of infinite words is presented. We then give the algorithms for directly constructing automata from linear equations and inequations. Next, the automata-theoretic operations corresponding to the first-order logical constructs are reviewed and the corresponding algorithms are described. Finally, a series of experimental results obtained with the LASH tool are presented and some conclusions are given.

2 Logical and Automata-Theoretic Background

In this section we recall some logical and automata-theoretic concepts that are used in the paper.

2.1 Theories of the Integers and Reals

The main theory we will consider in this paper is the first-order theory of the structure $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, where $+$ represents the predicate $x + y = z$. Since any linear equality or order constraint can be encoded into this theory, we refer to it as additive or linear arithmetic over the reals and integers. We will often refer to its restriction to integer variables as Presburger arithmetic, though the theory originally defined by Presburger was defined over the natural numbers.

2.2 Automata on Infinite Words

An infinite word (or ω-word) w over an alphabet Σ is a mapping $w : \mathbb{N} \to \Sigma$ from the natural numbers to Σ. A Büchi automaton on infinite words is a five-tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where

- Q is a finite set of states;
- Σ is the input alphabet;
- δ is the transition function and is of the form $\delta : Q \times \Sigma \to 2^Q$ if the automaton is nondeterministic and of the form $\delta : Q \times \Sigma \to Q$ if the automaton is deterministic;
- $Q_0 \subseteq Q$ is a set of initial states (a singleton for deterministic automata);
- F is a set of accepting states.

A run π of a Büchi automaton $A = (Q, \Sigma, \delta, q_0, F)$ on an ω-word w is a mapping $\pi : \mathbb{N} \to Q$ that satisfies the following conditions:

- $\pi(0) \in Q_0$, i.e. the run starts in an initial state;
- For all $i \geq 0$, $\pi(i + 1) \in \delta(\pi(i), w(i))$ (nondeterministic automata) or $\pi(i + 1) = \delta(\pi(i), w(i))$ (deterministic automata), i.e. the run respects the transition function.

Let $\mathit{inf}(\pi)$ be the set of states that occur infinitely often in a run π. A run π is said to be accepting if $\mathit{inf}(\pi) \cap F \neq \emptyset$. An ω-word w is accepted by a Büchi automaton if that automaton has some accepting run on w. The language $L_\omega(A)$ of infinite words defined by a Büchi automaton A is the set of ω-words it accepts.

A co-Büchi automaton is defined exactly as a Büchi automaton except that its accepting runs are those for which $\mathit{inf}(\pi) \cap F = \emptyset$.

We will also use the notion of *weak* automata [MSS86]. For a Büchi automaton $A = (Q, \Sigma, \delta, Q_0, F)$ to be *weak*, there has to be a partition of its state set Q into disjoint subsets $Q_1, \ldots, Q_m$ such that

- for each of the Q_i either $Q_i \subseteq F$ or $Q_i \cap F = \emptyset$; and
- there is a partial order $\leq$ on the sets $Q_1, \ldots, Q_m$ such that for every $q \in Q_i$ and $q' \in Q_j$ for which, for some $a \in \Sigma$, $q' \in \delta(q, a)$ ($q' = \delta(q, a)$ in the deterministic case), $Q_j \leq Q_i$.

For more details, a survey of automata on infinite words can be found in [Tho90].

3 Representing Sets of Integers and Reals with Finite Automata

In order to use a finite automaton for recognizing numbers, one needs to establish a mapping between these and words. Our encoding scheme corresponds to the usual notation for reals and relies on an arbitrary integer base $r > 1$. We encode a number x in base r, most significant digit first, by words of the form $w_I \star w_F$, where w_I encodes the integer part x_I of x as a finite word over $\{0, \ldots, r-1\}$, the special symbol "$\star$" is a separator, and w_F encodes the fractional part x_F of x as an infinite word over $\{0, \ldots, r-1\}$. Negative numbers are represented by their r's complement. In this notation, a number $d_k d_{k-1} \ldots d_1 d_0 \star d_{-1} d_{-2} \ldots$ written in base r and of integer length $k+1$ is positive if it starts with 0 and negative if it starts with $r-1$, in which case its value is $-r^{k+1} + \sum_{-\infty < i \leq k} d_i r^i$. Since we are dealing with encodings of variable length, we need to make sure that the number of digits used for the integer part is sufficient for the leading digit always to be 0 for positive numbers and $r-1$ for negative numbers. This is done by requiring that the number of digits $k+1$ of the integer part of a number x be such that $-r^k \leq x < r^k$ ($-r^k \leq x \leq r^k$ if the encoding of the fractional part of x is an infinite sequence of digits $r-1$). Note that repeating the leading digit has no impact on the value of the number since it is 0 for positive numbers and since, for negative numbers, we have that $-r^k + (r-1)r^{k-1} = -r^{k-1}$.

According to this scheme, each number has an infinite number of encodings, given that the integer-part length can be increased unboundedly. In addition, the rational numbers whose denominator has only prime factors that are also factors of r have two distinct encodings with the same integer-part length. For example, in base 10, the number $11/2$ has the encodings $005 \star 5(0)^\omega$ and $005 \star 4(9)^\omega$, "ω" denoting infinite repetition.

To encode a vector of real numbers, we represent each of its components by words of identical integer-part length. This length can be chosen arbitrarily, provided that it is sufficient for encoding the vector component with the highest magnitude. An encoding of a vector $\boldsymbol{x} \in \mathbb{R}^n$ can indifferently be viewed either as a n-tuple of words of identical integer-part length over the alphabet $\{0, \ldots, r-1, \star\}$, or as a single word w over the alphabet $\{0, \ldots, r-1\}^n \cup \{\star\}$.

Example 1. In base 2, the vector $(-2, 12.3)$ can be encoded as

$$(11110 \star 0^\omega, 01100 \star 01[1001]^\omega)$$

or as the word

$$(1,0)(1,1)(1,1)(1,0)(0,0) \star (0,0)(0,1)[(0,1)(0,0)(0,0)(0,1)]^\omega.$$

Using an alphabet of size r^n is clearly going to be problematic as soon as n starts to grow. The solution is to read the digits of the various components of the vector serially, in a round robin way, thus reducing the alphabet size to the perfectly manageable r. We will refer to this scheme as the *serial encoding* as opposed to the *simultaneous encoding* in which the alphabet consists of tuples of digits.

Example 2. Using the serial encoding, the vector $(-2, 12.3)$ can be encoded in base 2 as

$$1011111000 \star 0001[01000001]^\omega.$$

Real vectors being encoded by infinite words, a set of vectors can be represented by an infinite-word automaton accepting the corresponding encodings. However, since a real vector has an infinite number of possible encodings, we have to choose which of these the automata will recognize. A natural choice is to accept all encodings. This leads to the following definition.

Definition 1. *Let $n > 0$ and $r > 1$ be integers. A base-r n-dimension serial* Real Vector Automaton (RVA) *is a Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ automaton over the alphabet $\Sigma = \{0, \ldots, r-1\} \cup \{\star\}$, such that*

- *Every word accepted by $\mathcal{A}$ is a serial encoding in base r of a vector in $\mathbb{R}^n$, and*
- *For every vector $\boldsymbol{x} \in \mathbb{R}^n$, $\mathcal{A}$ accepts either all the encodings of $\boldsymbol{x}$ in base r, or none of them.*

An RVA is said to *represent* the set of vectors encoded by the words that belong to its accepted language.

From a theoretical point of view, there is no difference between the serial and simultaneous encodings, and it is easy to move from one to the other. From an implementation point of view, using the serial encoding is clearly preferable. Notice also, that in the context of minimized deterministic automata, the serial encoding is essentially equivalent to using the simultaneous scheme, while representing the transitions from a state to another by a BDD (an r-ary Decision Diagram for bases r other than 2). The expressive power of RVAs has been studied in [BRW98] and corresponds exactly to linear arithmetic over the reals and integers, extended with a special base-dependent predicate that can check the value of the digit appearing in a given position. If this special predicate is not used, RVAs can always be constructed to be weak automata [BJW01], which we will always assume to be the case in what follows.

4 Constructing Automata from Linear Relations

To construct automata corresponding to the formulas of linear arithmetic, we start with automata corresponding to linear equalities and inequalities as basic building blocks. It would of course also be possible to start just with automata for addition and order comparison, but there are simple and easy to implement constructions [BC96,BRW98,WB00] that directly produce close to optimal deterministic automata for linear relations, which explains our choice. This section describes these constructions in the case of real variables and the serial encoding of vectors. If both integer and real variables are involved in a relation, integerhood can be imposed by forcing the fractional part of the integer variables to be either 0^ω or $(r-1)^\omega$. This can be done by a simple adaptation of the constructions below, or by imposing integerhood as a separate constraint for which a specific automaton is constructed.

4.1 Linear Equations

The problem addressed consists of constructing an RVA that represents the set S of all the solutions $\boldsymbol{x} \in \mathbb{R}^n$ of an equation of the form $\boldsymbol{a}.\boldsymbol{x} = b$, given $n \geq 0$, $\boldsymbol{a} \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$.

A Decomposition of the Problem The basic idea is to build the automaton corresponding to a linear equation in two parts : one that accepts the integer part of solutions of the equation and one that accepts the part of the solution that belongs to $[0,1]^n$. For convenience, in what follows the n-dimension real vector represented by a word w interpreted as a serial encoding in base r is denoted $[w]^n_r$.

More precisely, let $\boldsymbol{x} \in S$, and let $w_I \star w_F$ be a serial encoding of $\boldsymbol{x}$ in a base $r > 1$, with $w_I \in \Sigma^+$, $w_F \in \Sigma^\omega$, and $\Sigma = \{0, \ldots, r-1\}$. The vectors $\boldsymbol{x}_I$ and $\boldsymbol{x}_F$, respectively encoded by the words $w_I \star 0^\omega$ and $0^n \star w_F$, are such that $\boldsymbol{x}_I \in \mathbb{Z}^n$, $\boldsymbol{x}_F \in [0,1]^n$, and $\boldsymbol{x} = \boldsymbol{x}_I + \boldsymbol{x}_F$. Since $\boldsymbol{a}.\boldsymbol{x} = b$, we have $\boldsymbol{a}.\boldsymbol{x}_I + \boldsymbol{a}.\boldsymbol{x}_F = b$. Moreover, writing $\boldsymbol{a}$ as $(a_1, \ldots, a_n)$, we have $\alpha \leq \boldsymbol{a}.\boldsymbol{x}_F \leq \alpha'$, where $\alpha = \sum_{a_i<0} a_i$ and $\alpha' = \sum_{a_i>0} a_i$, which implies $b - \alpha' \leq \boldsymbol{a}.\boldsymbol{x}_I \leq b - \alpha$. Another immediate property of interest is that $\boldsymbol{a}.\boldsymbol{x}_I$ is divisible by $\gcd(a_1, \ldots, a_n)$.

From those results, we obtain that the language L of the encodings of all the elements of S satisfies

$$L = \bigcup_{\varphi(\beta)} \{w_I \in \Sigma^+ \mid \boldsymbol{a}.[w_I \star 0^\omega]^n_r = \beta\} \cdot \{\star\} \cdot \{w_F \in \Sigma^\omega \mid \boldsymbol{a}.[0^n \star w_F]^n_r = b - \beta\},$$

where "$\cdot$" denotes concatenation and $\varphi(\beta)$ stands for $b - \alpha' \leq \beta \leq b - \alpha \wedge (\exists m \in \mathbb{Z})(\beta = m \gcd(a_1, \ldots, a_n))$.

This decomposition of L reduces the computation of an RVA representing S to the following problems:

- building an automaton on finite words accepting all the words $w_I \in \Sigma^+$ such that $[w_I \star \mathbf{0}^\omega]^n_r$ is a solution of a given linear equation;

– building a Büchi automaton accepting all the words $w_F \in \Sigma^\omega$ such that $[0^n \star w_F]_r^n$ is a solution of a given linear equation.

These problems are addressed in the two following sections.

Recognizing Integer Solutions Our goal is, given an equation $\boldsymbol{a}.\boldsymbol{x} = b$ where $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$, to construct a finite automaton $\mathcal{A}_{\boldsymbol{a},b}$ that accepts all the finite words serially encoding in a given base r the integer solutions of that equation.

The construction proceeds as follows. Except for the unique initial state s_0 and the states reached from there while reading the first n digits of the vector encoding (the sign digits), the states s of $\mathcal{A}_{\boldsymbol{a},b}$ are in one-to-one correspondence with a pair (γ, i), where γ is an integer and $0 \leq i \leq n-1$ denotes a position in the serial reading of the vector digits. A state s of the form $(\gamma, 0)$, corresponding to the situation in which the number of digits read is the same for each vector component, has the property that the vectors $\boldsymbol{x} \in \mathbb{Z}^n$ accepted by the paths leading from s_0 to s are exactly the solutions of the equation $\boldsymbol{a}.\boldsymbol{x} = \gamma$. The only accepting state s_F of $\mathcal{A}_{\boldsymbol{a},b}$ is $(b, 0)$.

The next step is to define the transitions of $\mathcal{A}_{\boldsymbol{a},b}$. Consider first moving from a state s of the form $(\gamma, 0)$. The next n digits $d_1, \ldots, d_n$ that will be read lengthen the encoding of each component of the vector by 1 and hence, if $\boldsymbol{x}$ is the value of the vector read before inputing the digits $d_1, \ldots, d_n$, then its value $\boldsymbol{x'}$ after reading these digits is $\boldsymbol{x'} = r\boldsymbol{x} + (d_1, \ldots, d_n)$. If only the first $i < n$ of the digits $d_1, \ldots, d_n$ have been read, the value of the vector will be taken to be $\boldsymbol{x'} = r\boldsymbol{x} + (d_1, \ldots, d_i, 0, \ldots, 0)$. Therefore, for a state s of the form (γ, i), an outgoing transition labeled d must lead to a state $s' = (\gamma', (i+1) \bmod n)$ such that $\gamma' = r\gamma + a_1 d$ if $i = 0$ and $\gamma' = \gamma + a_{i+1} d$ if $i > 0$.

For transitions from the single initial state s_0, one has to take into account the fact the the first n digits read are the sign bits of the vector components. Thus a digit $r-1$ should be interpreted as -1, no other digit except 0 being allowed. The states other than s_0 reached during the reading of the sign digits will be characterized, as all other states, by a value and a position in the serial reading of digits, but this position will be represented as a negative number $-(n-1) \leq i \leq -1$ in order to distinguish these states from the states reached after the sign digits have been read. Thus a transition labeled $d \in \{0, r-1\}$ from s_0 leads to the state $s' = (-a_1, -1)$ if $d = r-1$, and $s' = (0, -1)$ if $d = 0$. Similarly, a transition labeled $d \in \{0, r-1\}$, from a state $s = (\gamma, -i)$ leads to the state $s' = (\gamma - a_{i+1}, -((i+1) \bmod n))$ if $d = r-1$, and $s' = (\gamma, -((i+1) \bmod n))$ if $d = 0$.

Notice that the transition relation we have defined is deterministic and that only a finite number of states are needed. Indeed, from a state $s = (\gamma, 0)$ such that $|\gamma| > (r-1)\sum_{i=1}^{n} |a_i|$, one can only reach states s' such that $|\gamma'| > |\gamma|$, hence all states $s = (\gamma, 0)$ such that $|\gamma| > (r-1)\sum_{i=1}^{n} |a_i|$ and $|\gamma| > |b|$, as well as their successors, can be pruned. If all unnecessary states (those from which the accepting state cannot be reached) are pruned, the automaton obtained is, within a small exception, a minimal deterministic automaton. Indeed, if it was possible to merge two states $s = (\gamma, i)$ and $s' = (\gamma', i)$ with $\gamma' \neq \gamma$, then different right-hand sides for the equation $\boldsymbol{a}.\boldsymbol{x} = \beta$ would yield identical solutions, a contradiction. However, absolute minimality is not guaranteed since it could still be possible to merge a state (γ, i) with the state $(\gamma, -i)$ without changing the accepted language.

In practice, to avoid the pruning of unnecessary states, it is convenient to compute the automaton $\mathcal{A}_{\boldsymbol{a},b}$ backwards, starting from the accepting state $s_F = (b, 0)$. Computing the backwards transitions is quite straightforward. Indeed, an incoming d-labeled transitions to a state $s = (\gamma, i)$ with $1 < i \leq n-1$ has to originate from the state $s' = (\gamma - a_i d, i-1)$. If $i = 0$, the origin state is $s' = (\gamma - a_n d, n-1)$, and, if $i = 1$ it is $s' = ((\gamma - a_1 d)/r, 0)$. If $(\gamma - a_1 d)/r$ is not an integer or is not divisible by $\gcd(a_1, \ldots, a_n)$, the state s' should not be created, and the states only reachable from it should be pruned. Finally, from a state s of the form $(\gamma, 0)$, one should also consider the possibility that the digits read to reach it are the sign digits. This is only possible if $\exists \sigma \subseteq \{a_1, \ldots, a_n\}\ \gamma = -\sum_{a_i \in \sigma} a_i$, in which case one should also move backwards from the state $(\gamma, 0)$ to $(\gamma + a_n, -(n-1))$, or $(\gamma, -(n-1))$, and so on for all possibilities that finally go back to the unique initial state s_0.

If one wishes to construct k automata $\mathcal{A}_{\boldsymbol{a},b_1}, \mathcal{A}_{\boldsymbol{a},b_2}, \ldots, \mathcal{A}_{\boldsymbol{a},b_k}$ with $b_1, \ldots, b_k \in \mathbb{Z}$ (for instance, as an application of the method presented in Section 4.1, in which the b_i are all the integers satisfying φ), then a technique more efficient than repeating the construction k times consists of starting from the set $\{(b_1, 0), \ldots, (b_k, 0)\}$, rather than from a set containing a single state. The states and transitions computed during the construction will then be shared between the different $\mathcal{A}_{\boldsymbol{a},b_i}$, and each $(b_i, 0)$ will be the only accepting state of the corresponding $\mathcal{A}_{\boldsymbol{a},b_i}$.

As is shown in [BRW98], the number of states of the automaton constructed for an equation $\boldsymbol{a}.\boldsymbol{x} = b$ is logarithmic in the value of b and linear in the absolute value of the elements of $\boldsymbol{a}$. Finally, note that for any equation and base there is a unique minimal deterministic automaton accepting the solutions of the equation. It can always be obtained by applying standard minimization techniques [Hop71] to the automaton obtained from the construction described above or, as a matter of fact, to any automaton accepting the same language.

Recognizing Fractional Solutions We now address the computation of a Büchi automaton $\mathcal{A}'_{\boldsymbol{a},b}$ that accepts all the infinite words $w \in \Sigma^\omega$ such that $0^n \star w$ encodes a solution $\boldsymbol{x} \in [0,1]^n$ of the equation $\boldsymbol{a}.\boldsymbol{x} = b$.

The construction is similar to the one of the previous section, except that we are now dealing with the expansion of fractional numbers. The states s of $\mathcal{A}'_{\boldsymbol{a},b}$ are in a one-to-one correspondence with a pair (γ, i), where γ is an integer and $0 \leq i \leq n-1$ denotes a position in the serial reading of the vector digits. A state s of the form $(\gamma, 0)$, corresponding to the situation in which the number of digits read is the same for each vector component, has the property that the vectors $\boldsymbol{x} \in [0,1]^n$ accepted by the infinite paths starting from s are exactly the solutions of the equation $\boldsymbol{a}.\boldsymbol{x} = \gamma$. The set of initial states contains only the pair $(b, 0)$. All the states are accepting.

The transitions of $\mathcal{A}'_{\boldsymbol{a},b}$ are defined as follows. Consider first moving from a state s of the form $(\gamma, 0)$. The next n digits $d_1, \ldots, d_n$ that will be read lengthen the encoding of each component of the vector by 1. This amounts to prefixing the digits $d_1, \ldots, d_n$ to the word that will be read from next state s' of the form $(\gamma', 0)$ that will be reached. The value $\boldsymbol{x}$ of the word read from s is thus related to the value $\boldsymbol{x}'$ of the word read from s' by $\boldsymbol{x} = (1/r)(\boldsymbol{x}' + (d_1, \ldots, d_n))$, which can be rewritten as $\boldsymbol{x}' = r\boldsymbol{x} - (d_1, \ldots, d_n)$. If only the first $i < n$ of the digits $d_1, \ldots, d_n$ have been read, the value of the vector will be taken to be $\boldsymbol{x}' = r\boldsymbol{x} + (d_1, \ldots, d_i, 0, \ldots, 0)$. Therefore, for a state s of the form (γ, i), an outgoing transition labeled d must lead to a state $s' = (\gamma', (i + 1) \bmod n)$ such that $\gamma' = r\gamma - a_1 d$ if $i = 0$ and $\gamma' = \gamma - a_{i+1} d$ if $i > 0$.

Note that for states of the form $(\gamma, 0)$, γ must belong to the interval $[\alpha, \alpha']$, where α and α' are as defined in Section 4.1, otherwise there would be no solution in $[0, 1]^n$ to $\boldsymbol{a}.\boldsymbol{x} = \gamma$. Only a finite number of states are thus necessary.

The automaton $\mathcal{A}'_{\boldsymbol{a},b}$ can be constructed by starting from the state $s = (b, 0)$, and then repeatedly computing the outgoing transitions from the current states until stabilization occurs. Like in Section 4.1, the construction of k automata $\mathcal{A}'_{\boldsymbol{a},b_1}, \mathcal{A}'_{\boldsymbol{a},b_2}, \ldots, \mathcal{A}'_{\boldsymbol{a},b_k}$, with $b_1, \ldots, b_k \in \mathbb{Z}$ (for instance, as an application of the method presented in Section 4.1) can simply be done by starting from the set $\{(b_1, 0), \ldots, (b_k, 0)\}$, rather than from a set containing a single state. The computation terminates, since for every state of the form $s = (\gamma, 0)$, the integer γ belongs to the bounded interval $[\alpha, \alpha']$. Once dead-end states are pruned, the automaton obtained is deterministic. Notice also that it is weak since all its states are accepting and can be minimized by the procedure sketched in Section 5.4. Interestingly, because we are dealing with automata on infinite words, states (γ, i) and (γ, j) with $i \neq j$ can occasionally be merged, which precludes the construction above from always producing a minimal automaton.

4.2 Linear Inequations

The method presented for equations can be easily adapted to linear inequations. The problem consists of computing an RVA representing the set of all the solutions $\boldsymbol{x} \in \mathbb{R}^n$ of an inequation of the form $\boldsymbol{a}.\boldsymbol{x} \leq b$, given $n \geq 0$, $\boldsymbol{a} \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$.

The decomposition of the problem into the computation of representations of the sets of integer solutions and of solutions in $[0, 1]^n$ of linear inequations is identical to the one proposed for equations in Section 4.1.

Given an inequation of the form $\boldsymbol{a}.\boldsymbol{x} \leq b$, where $\boldsymbol{a} \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$, the definition of an automaton $\mathcal{A}_{\boldsymbol{a},b}$ that accepts all the finite words $w \in \Sigma^*$ such that $w \star 0^\omega$ encodes an integer solution of $\boldsymbol{a}.\boldsymbol{x} \leq b$ is very similar to the one given for equations in Section 4.1. Indeed, it is sufficient to now consider all states $s = (\gamma, 0)$ with $\gamma \leq b$ as accepting. Furthermore, noticing that all states $s = (\gamma, 0)$ such that $\gamma < 0$, $|\gamma| > (r - 1)\sum_{i=1}^{n} |a_i|$ and $|\gamma| > |b|$ can only lead to accepting states, and that all states $s = (\gamma, 0)$ such that $\gamma > 0$, $|\gamma| > (r - 1)\sum_{i=1}^{n} |a_i|$ and

$|\gamma| > |b|$ can only lead to nonaccepting states, these can be respectively collapsed into single accepting and nonaccepting states. The automaton thus only has a bounded number of states.

The backward construction can also be adapted in the following way: the states s for which the computed γ is not an integer or is not divisible by $\gcd(a_1, \ldots, a_n)$ are not discarded, but their value is rounded to the nearest lower integer γ' that is divisible by $\gcd(a_1, \ldots, a_n)$. This operation is correct since the sets of integer solutions of $\boldsymbol{a}.\boldsymbol{x} \leq \gamma$ and of $\boldsymbol{a}.\boldsymbol{x} \leq \gamma'$ are in this case identical. The resulting automaton is however no longer deterministic, but as was shown in [WB00], it can be determinized efficiently.

The construction of an automaton $\mathcal{A}_{\boldsymbol{a},b}$ that accepts all the infinite words $w \in \Sigma^\omega$ such that $0 \star w$ encodes a solution of $\boldsymbol{a}.\boldsymbol{x} \leq b$ that belongs to $[0,1]^n$ is again very similar to the one developed for equations in Section 4.1. The difference with the case of equations, in that we do not discard here the states $s = (\gamma, 0)$ for which the computed γ is greater than α'. Instead, we simply replace the value of γ by α', since the sets of solutions in $[0,1]^n$ of $\boldsymbol{a}.\boldsymbol{x} \leq \gamma$ and of $\boldsymbol{a}.\boldsymbol{x} \leq \alpha'$ are in this case identical. On the other hand, we still discard the states $s = (\gamma, 0)$ for which the computed γ is lower than α, since this implies that the inequation $\boldsymbol{a}.\boldsymbol{x} \leq \gamma$ has no solution in $[0,1]^n$. Notice again that this will produce a weak automaton that can be determinized and minimized.

5 Manipulating Sets of Integer and Real Vectors

Most applications of linear constraints require the possibility of transforming, combining, and checking represented sets according to various operators. For instance, deciding whether a formula ϕ of $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$ is satisfiable can be done by first building the representations of the atoms of ϕ, which are equations and inequations (see Section 4), then applying the connectors and the quantifiers that appear in φ to these representations, and finally checking whether the resulting automaton (which recognizes exactly all solutions of ϕ) represents a non-empty set. The last step simply amounts to check that the language accepted by the finite-state representation is not empty. In this section, we give algorithms for applying Boolean connectors, quantifiers, and other specific operators to the finite-state representations of arithmetic sets.

5.1 Computing Boolean Combinations of Sets

Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ be automata representing in a base $r > 1$ (respectively) the sets $S_1, S_2, \ldots S_k \subseteq \mathbb{R}^n$. For each $i \in \{1, \ldots, k\}$, let $\mathcal{A}_i = (Q_i, \Sigma, \delta, Q_{0,i}, F_i)$, with $\Sigma = \{0, 1, \ldots, r-1, \star\}$, and let L_i denote the language accepted by $\mathcal{A}_i$. We assume that each $\mathcal{A}_i$ is weak, deterministic, and complete (meaning that for every state $q \in Q_i$ and symbol $a \in \Sigma$, there exists an outgoing transition from q labeled by a).

Let $S \subseteq \mathbb{R}^n$ be a Boolean combination $\mathcal{B}(S_1, S_2, \ldots, S_k)$ of the sets S_i, i.e., a set obtained by applying the operators $\cup$ (union), $\cap$ (intersection) and $\neg$ (complement) to the S_i.

In order to compute a finite-state representation of S, one first builds an automaton $\mathcal{A}$ that accepts the language $L = \mathcal{B}(L_1, L_2, \ldots, L_k)$. This consists of simulating the concurrent operation of the $\mathcal{A}_i$, accepting a word w whenever $\mathcal{B}(w \in L_1, w \in L_2, \ldots, w \in L_k)$ holds. Formally, we have $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$, with

- $Q = Q_1 \times Q_2 \times \cdots \times Q_k$;
- $\delta : Q \times \Sigma \to Q : ((q_1, \ldots, q_k), a) \mapsto (\delta_1(q_1, a), \ldots, \delta_k(q_k, a))$;
- $Q_0 = Q_{0,1} \times \cdots \times Q_{0,k}$;
- $F = \{(q_1, \ldots, q_k) \in Q \mid \mathcal{B}(q_1 \in F_1, \ldots, q_k \in F_k)\}$.

(It is worth mentioning that this construction is valid only because the $\mathcal{A}_i$ are deterministic and weak. Every word corresponds to exactly one run for each $\mathcal{A}_i$, which ends up visiting only a group of states with the same accepting or nonaccepting status.)

The automaton $\mathcal{A}$ accepts an encoding w of a vector $\boldsymbol{x} \in \mathbb{R}^n$ if and only if $w \in \mathcal{B}(L_1, L_2, \ldots, L_k)$, i.e., if and only if $\boldsymbol{x} \in S$. This does not imply however that $\mathcal{A}$ forms a valid representation of S, since it may accept words that are not vector encodings. A suitable representation of S can therefore be obtained by intersecting $\mathcal{A}$ with an automaton accepting the set of all valid vector encodings, in other words, an automaton representing the set $\mathbb{R}^n$. Note that this intersection operation is only needed when L is not a subset of $L_1 \cup \cdots \cup L_k$, i.e., if $b_1 \vee b_2 \vee \ldots \vee b_k$ does not imply $\mathcal{B}(b_1, \ldots, b_k)$ for all Booleans b_i. Whenever required, the additional intersection operation can conveniently be performed as a part of the product construction described above. This construction preserves the determinism and weak nature of the automata.

5.2 Constructing Cartesian Products

Let $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, F_2)$ be weak deterministic finite-state representations of (respectively) the sets $S_1 \subseteq \mathbb{R}^{n_1}$ and $S_2 \subseteq \mathbb{R}^{n_2}$, with $n_1 > 0$ and $n_2 > 0$. A representation $\mathcal{A}$ of the Cartesian product $S = S_1 \times S_2$ is a finite-state machine that simulates repeatedly the operations of $\mathcal{A}_1$ for n_1 input symbols, and then those of $\mathcal{A}_2$ for the next n_2 symbols. The transitions labeled by the separator "$\star$" are handled in a special way by ensuring that they are followed at the same time in both automata. Let $n = n_1 + n_2$. Formally, we have $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$, where

- $Q = Q_1 \times Q_2 \times \{0, 1, \ldots, n-1\}$;
- $\Sigma = \{0, 1, \ldots, r-1, \star\}$;
- $\delta : ((q_1, q_2, i), a) \mapsto \begin{cases} (\delta_1(q_1, a), q_2, i+1) & \text{if } i < n_1 \text{ and } a \neq \star, \\ (q_1, \delta_2(q_2, a), (i+1) \bmod n) & \text{if } i \geq n_1 \text{ and } a \neq \star; \\ (\delta_1(q_1, a), \delta_2(q_2, a), 0) & \text{if } a = \star. \end{cases}$
- $Q_0 = Q_{0,1} \times Q_{0,2} \times \{0\}$;
- $F = F_1 \times F_2 \times \{0, 1, \ldots, n-1\}$.

This construction preserves the determinism and the weak nature of the automata.

5.3 Applying Quantifiers

Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be a finite-state representation of a set $S \subseteq \mathbb{R}^n$, with $n > 1$, and let $i \in \{1, \ldots, n\}$ be the index of a vector component. Quantifying existentially S with respect to the i-th vector component over the real domain amounts to projecting out this component, which yields the set

$$S' = \exists_i^{\mathbb{R}} S = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \in \mathbb{R}^{n-1} \mid (\exists x_i \in \mathbb{R})((x_1, \ldots, x_n) \in S)\}.$$

In order to compute a representation of S', one needs to remove from $\mathcal{A}$ the transition labels corresponding to the i-th vector component. This is done by constructing a nondeterministic automaton $\mathcal{A}' = (Q', \Sigma, \delta', Q_0', F')$, such that

- $Q' = Q \times \{0, 1, \ldots n-2\}$;
- $\Sigma = \{0, 1, \ldots, r-1, \star\}$;
- $\delta' \ : ((q,k),a) \mapsto \begin{cases} \{(\delta(q,a), (k+1) \bmod (n-1))\} & \text{if } k+1 \neq i \text{ and } a \neq \star, \\ \bigcup_{a' \in \{0,\ldots,r-1\}} \{(\delta(\delta(q,a'),a), (k+1) \bmod (n-1))\} & \\ & \text{if } k+1 = i \text{ and } a \neq \star, \\ \{(\delta(q,a), 0)\} & \text{if } a = \star; \end{cases}$
- $Q_0' = \begin{cases} Q_0 \times \{0\} & \text{if } i \neq 1, \\ \bigcup_{a \in \{0, r-1\}, q_0 \in Q_0} \{(\delta(q_0, a), 0)\} & \text{if } i = 1; \end{cases}$
- $F' = F \times \{0\}$.

This approach introduces two problems. First, the resulting automaton $\mathcal{A}'$ is nondeterministic, which prevents its further manipulation by the algorithm outlined in Section 5.1 (especially when the set needs to be complemented). Second, although the language accepted by $\mathcal{A}'$ contains only valid encodings of vectors in $\exists_i S$, it may not accept all of them. (For instance, removing from a binary representation of the set $\{(10, 2)\}$ the transitions related to the first vector component would produce an automaton recognizing only the encodings of 2 that contain more than four digits in their integer part.) We address these two problems separately.

In order to determinize $\mathcal{A}'$, we exploit an interesting property of weak automata, which can directly be turned into co-Büchi automata: a weak Büchi automaton $(Q, \Sigma, \delta, Q_0, F)$ accepts the same language as the co-Büchi automaton $(Q, \Sigma, \delta, Q_0, Q \setminus F)$. As a consequence, weak automata can be determinized using the simple breakpoint construction [MH84,KV97] developed for co-Büchi automata. This construction proceeds as follows.

Let $\mathcal{A}' = (Q', \Sigma, \delta', Q_0', F')$ be a weak non-deterministic co-Büchi automaton. The co-Büchi automaton $\mathcal{A}' = (Q'', \Sigma, \delta'', Q_0'', F'')$ defined as follows accepts the same language.

- $Q'' = 2^{Q'} \times 2^{Q'}$, the states of $\mathcal{A}''$ are pairs of sets of states of $\mathcal{A}'$;
- $Q_0'' = \{(Q_0', \emptyset)\}$;
- For $(S, R) \in Q''$ and $a \in \Sigma$, the transition function is defined by
  - if $R = \emptyset$, then $\delta''((S,R),a) = (T, T \setminus F')$ where $T = \{q \mid \exists p \in S \text{ and } q \in \delta'(p,a)\}$, T is obtained from S as in the classical subset construction, and the second component of the pair of sets of states is obtained from T by eliminating states in F';

- if $R \neq \emptyset$, then $\delta''((S,R),a) = (T, U \setminus F')$ where $T = \{q \mid \exists p \in S$ and $q \in \delta'(p,a)\}$, and $U = \{q \mid \exists p \in R$ and $q \in \delta'(p,a)\}$, the subset construction set is now applied to both S and R and states in F' are removed from U;

– $F' = 2^{Q'} \times \{\emptyset\}$.

When the automaton $\mathcal{A}''$ is in a state (S,R), R represents the states of $\mathcal{A}'$ that can be reached by a run that has not gone through a state in F' since that last "breakpoint", i.e. state of the form $(S,\emptyset)$. So, for a given word, $\mathcal{A}$ has a run that does not go infinitely often through a state in F' if and only if $\mathcal{A}''$ has a run that does not go infinitely often through a state in F''. Notice that the difficulty that exists for determinizing Büchi automata, which is to make sure that the *same* run repeatedly reaches an accepting state disappears since, for co-Büchi automata, we are just looking for a run that eventually avoids accepting states.

It is interesting to notice that the construction implies that all reachable states (S,R) of $\mathcal{A}''$ satisfy $R \subseteq S$. The breakpoint construction can thus be implemented as a subset construction in which the states in R are simply tagged. Experimental results (see Section 6) have shown that its practical efficiency is similar to that of the traditional subset construction for finite-word automata.

In general, determinizing a co-Büchi automaton does not produce a weak automaton. This problem can be alleviated, provided that the set of vectors undergoing the quantification operation (and thus the set obtained as a result) is definable in the theory $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$. Indeed, it has been shown using topological arguments [BJW01] that any deterministic Büchi automaton recognizing such sets can easily be turned into a weak one by a simple operation. This operation actually consists of turning all the states belonging to the same strongly connected component of the automaton into accepting or non-accepting ones, depending on whether this component contains or not at least one accepting state.

We now address the problem of turning an automaton accepting some encodings of the vectors in a set into one that recognizes all of them. More precisely, in the present case, one needs to make sure that whenever the automaton accepts an encoding $u^k \cdot w$, where $u \in \{0, r-1\}^n$ is the sign prefix, it also accepts the words $u^j \cdot w$ for all j such that $1 < j < k$. In [Boi98], this problem is solved by computing for each possible sign prefix $u \in \{0, r-1\}^n$ the set of automaton states reachable after reading any word in u^*. These states are then made reachable from the initial state after reading any number of occurrences of u, thanks to a simple construction, and the process is repeated for other u.

The drawback of this approach is its systematic cost in $O(2^n)$, which limits is applicability to problems with a very small vector dimension. An improved algorithm has been developed [BL01], in which subsets of sign headers that are not distinguished by the automaton (in the sense that their reading always lead to the same automaton states) are handled collectively rather than separately. This algorithm has been implemented in the LASH tool [LAS], and experimental results have shown it to be of moderate practical cost even for large vector dimensions. Note that this automaton transformation may produce non-determinism, and has thus to be performed prior to the determinization procedure discussed in this section.

We have only considered so far existential quantification over the reals. This generalizes naturally to universal quantification thanks to the complement operation described (as a particular instance of Boolean combination) in Section 5.1. In order to decide $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, one also needs to be able to quantify sets with respect to the integer domain, i.e., given a set $S \subseteq \mathbb{R}^n$ and a vector component index $i \in \{1, \ldots, n\}$, compute the sets

$$\exists_i^{\mathbb{Z}} S = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \in \mathbb{R}^{n-1} \mid (\exists x_i \in \mathbb{Z})((x_1, \ldots, x_n) \in S)\}, \text{ and}$$
$$\forall_i^{\mathbb{Z}} S = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \in \mathbb{R}^{n-1} \mid (\forall x_i \in \mathbb{Z})((x_1, \ldots, x_n) \in S)\}.$$

These operations are easily reduced to applying quantifiers over the reals and performing Boolean combinations, thanks to the following rules :

$$\exists_i^{\mathbb{Z}} S = \exists_i^{\mathbb{R}}(S \cap \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_i \in \mathbb{Z}\});$$
$$\forall_i^{\mathbb{Z}} S = \forall_i^{\mathbb{R}}(S \cup \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_i \notin \mathbb{Z}\}).$$

Applying these rules requires an automaton that recognizes all vectors in which the i-th component is an integer (see Section 4).

5.4 Minimizing Finite-State Representations of Sets

Although the operations described in Sections 5.1 to 5.3 produce weak and deterministic automata, these can be unnecessarily large because of redundancies in their transition graph. In a recent paper [Löd01], it has been shown that weak deterministic automata admit a minimal form, unique up to isomorphism, which can be computed with $O(n \log n)$ cost. Sketchily, in order to minimize a weak deterministic automaton, one first locates the trivial strongly connected component in its transition graph (i.e., the components that do not contain cycles). Then, one modifies the accepting of non-accepting status of the states in these components (which does not affect the language accepted by the automaton) according to some specific rules. The result is then fed to the classical Hopcroft's algorithm [Hop71] for minimizing finite-state machines on finite words.

5.5 Other Operations

In verification applications, in order to explore infinite state-spaces, one needs to be able to compute infinite sets of reachable states in finite time. The concept of *meta-transition* has been introduced for this purpose in [BW94]. Intuitively, a meta-transition is associated to a cycle in the control graph of the analyzed system, and following it once leads to all the configurations that could be reached after following repeatedly that cycle any number of times.

If the program undergoing the analysis is based on integer and real variables, RVAs can be used as symbolic representations of its sets of configurations. If the operations performed by the program are definable in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, the computation of pre or post-images of such a set of configurations with respect to a given operation follows from the algorithms given in Sections 4 to 5.

In [Boi98], is has been shown that the effect of meta-transitions based on linear transformations and guards can be expressed in the theory $\langle \mathbb{Z}, +, \leq \rangle$ (i.e., in Presburger arithmetic), under some hypotheses. We summarize this result below.

Let $A \in \mathbb{Z}^{n \times n}$, $\boldsymbol{b} \in \mathbb{Z}^n$, $P \in \mathbb{Z}^{m \times n}$ and $\boldsymbol{q} \in \mathbb{Z}^m$, and let θ be the operation $P\boldsymbol{x} \leq \boldsymbol{q} \to \boldsymbol{x} := A\boldsymbol{x} + \boldsymbol{b}$, i.e., $\theta : \mathbb{R}^n \to \mathbb{R}^n : \boldsymbol{v} \mapsto A\boldsymbol{v} + \boldsymbol{b}$ if $P\boldsymbol{v} \leq \boldsymbol{q}$.

Theorem 1. *If there exists $p \geq 1$ such that*

- *The matrix A^p is diagonalizable, and*
- *Its eigenvalues belong to $\{0, 1\}$,*

then the closure $\theta^* = id \cup \theta \cup \theta^2 \cup \cdots$ of θ is definable in Presburger arithmetic.

The hypotheses of this theorem can be checked algorithmically, using only simple integer arithmetic [Boi98]. Its proof is constructive and turns into an algorithm for computing, for any set S in $\langle \mathbb{Z}, +, \leq \rangle$ the set $\theta^*(S)$ in terms of an expression defining S. Since $\langle \mathbb{Z}, +, \leq \rangle$ is a subtheory of $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, the same construction can also be carried out with RVAs.

The last operation considered in this study is the time-elapse transformation needed for the analysis of timed systems. Let $\boldsymbol{x} \in \mathbb{R}^n$ be a vector of clocks, the value of which evolves at the rate $\dot{\boldsymbol{x}}$ under the condition $\boldsymbol{a} \leq \dot{\boldsymbol{x}} \leq \boldsymbol{b}$ (in which $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}^n$ are constant vectors). Given a initial set $S \subseteq \mathbb{R}^n$ of clock values, the values reached after letting time elapse for an arbitrarily long period form the set

$$S' = \{\boldsymbol{x}' \in \mathbb{R}^n \mid (\exists \boldsymbol{x} \in S, t \in \mathbb{R}, \boldsymbol{\delta} \in \mathbb{R}^n)(t \geq 0 \wedge t\boldsymbol{a} \leq \boldsymbol{\delta} \leq t\boldsymbol{b} \wedge \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta})\}.$$

Since this transformation is expressed in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, a RVA representing S' can easily be constructed from one representing S.

6 Examples and Experimental Results

As explained earlier, RVA have the interesting property of being able to represent both discrete and continuous features. As an example, consider the set

$$\begin{aligned}\{(x_1, x_2) \in \mathbb{R}^2 \mid (\exists x_3, x_4 \in \mathbb{R})(\exists x_5, x_6 \in \mathbb{Z})(x_1 = x_3 + 2x_5 \\ \wedge\ x_2 = x_4 + 2x_6 \wedge 0 \leq x_3 \leq x_4 \leq 1)\}.\end{aligned}$$

As illustrated in Figure 1, this set combines linear constraints and discrete periodicities. The minimal and deterministic RVA representing this set in base 2 is given in Figure 2, in its simultaneous form.

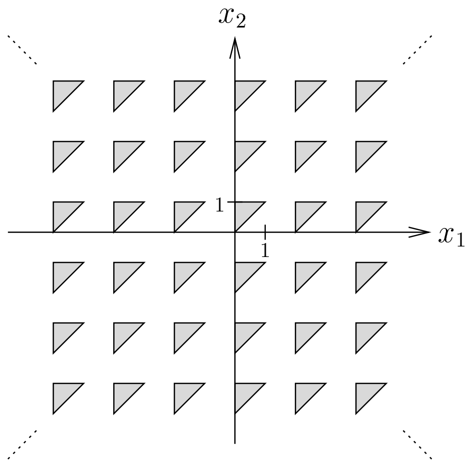

Fig. 1. A set with both continuous and discrete features.

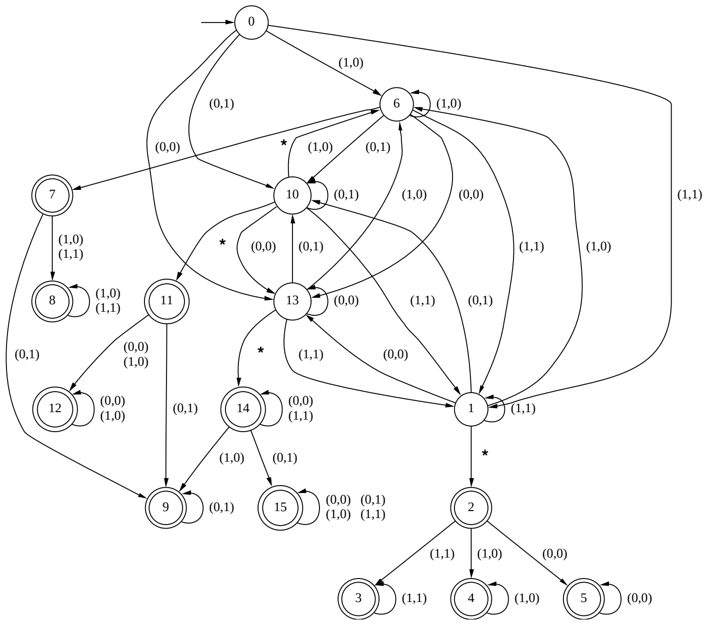

Fig. 2. RVA representing the set in Fig. 1.

It has been mentioned in Section 5 that the cost of determinizing an RVA following a projection operation appears considerably smaller in practical applications than suggested by the worst-case complexity of the "breakpoint" construction. This observation is illustrated in Figure 3, in which the size of finite-state representations of sets of values obtained by combining linear constraints with arbitrary coefficients is given before and after undergoing the projection and determinization operations. The figure also compares the behaviors with respect to these operations of sets of real vectors represented by RVAs, and sets of integer solutions represented by automata on finite words (*Number Decision Diagrams, NDDs* [WB95]). As also substantiated by other experiments, RVAs seem just as usable in practice as NDDs.

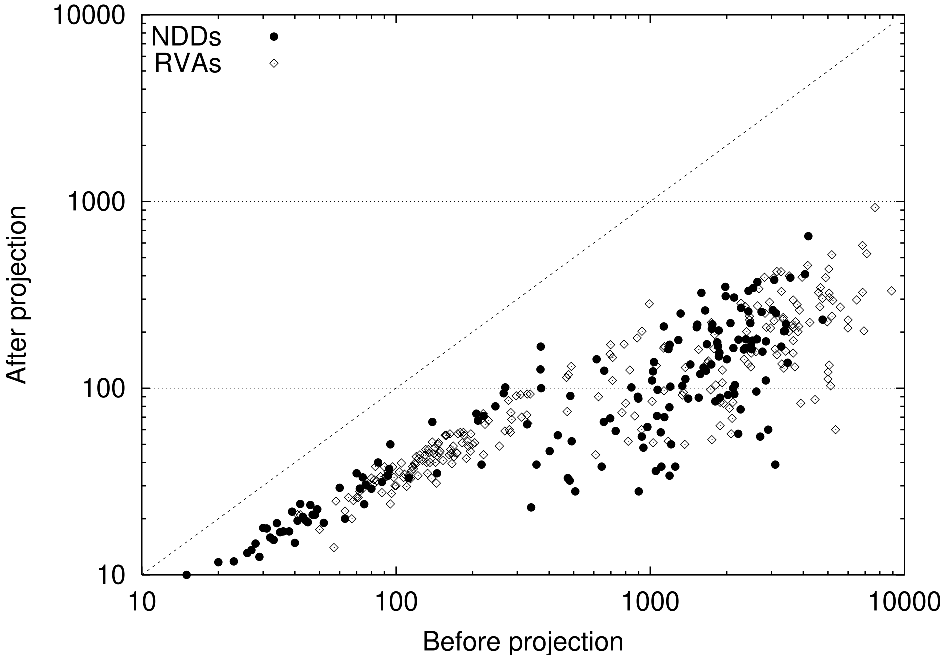

Fig. 3. Projecting and Determinizing Finite-State Representations.

7 Conclusions

This paper has overviewed automata-based techniques for handling linear constraints defined over the reals and integers. Though early experimental results are quite encouraging, there is little hope that these techniques will consistently outperform more traditional approaches when these can be applied. The reason for this is that the automaton one computed for a formula contains a lot of explicit information that might not be needed, for instance when just checking satisfiability. Computing this unneeded information comes at a cost that can be avoided by other approaches.

On the other hand, while the information made explicit by the automaton can be used repeatedly, there can be a substantial advantage to having it available: though the initial computation of the automaton will be costly it will speed up subsequent computations. This can for instance be the case in the symbolic verification of infinite-state systems [WB98].

A major advantage of the automaton-based approach is that it handles traditionally harder to deal with cases, such as nonconvex sets, just as simply as the more usual convex sets. Furthermore, it provides a normal form for represented sets in all cases, the absence of which in other approaches is a recurrent problem, for example in the case of nonconvex sets. Finally, the fact that it can handle constraints defined over the integers and reals combined is a significant new capability. An application that has already been considered is the verification of hybrid systems [BBR97], but there are probably also many others.

Acknowledgement

Sébastien Jodogne has contributed substantially to the implementation of the arithmetic part of the LASH tool and has done a careful reading of this paper.

References

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In *Proc. 9th Int. Conf.on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 167–178, Haifa, June 1997. Springer-Verlag.

[BC96] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In *Proceedings of CAAP'96*, number 1059 in Lecture Notes in Computer Science, pages 30–43. Springer-Verlag, 1996.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, June 1990.

[BJW01] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, Siena, June 2001. Springer-Verlag.

[BL01] B. Boigelot and L. Latour. Counting the solutions of Presburger equations without enumerating them. In *Proc. International Conference on Implementations and Applications of Automata*, Lecture Notes in Computer Science, Pretoria, South Africa, July 2001. Springer-Verlag. To appear.

[Boi98] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, 1998.

[BRW98] Bernard Boigelot, Stéphane Rassart, and Pierre Wolper. On the expressiveness of real and integer arithmetic automata. In *Proc. 25th Colloq. on Automata, Programming, and Languages (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*, pages 152–163. Springer-Verlag, July 1998.

[Bry86] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift Math. Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[BW94] Bernard Boigelot and Pierre Wolper. Symbolic verification with periodic sets. In *Computer Aided Verification, Proc. 6th Int. Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67, Stanford, California, June 1994. Springer-Verlag.

[Hop71] J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. *Theory of Machines and Computation*, pages 189–196, 1971.

[KV97] O. Kupferman and M. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[LAS] The Liège Automata-based Symbolic Handler (LASH). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[Löd01] C. Löding. Efficient minimization of deterministic weak $\omega-$automata, 2001. Submitted for publication.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[MS97] O. Maler and L. Staiger. On syntactic congruences for ω-languages. *Theoretical Computer Science*, 183(1):93–112, 1997.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming.* Springer-Verlag, 1986.

[Pug92] W. Pugh. A practical algorithm for exact array dependency analysis. *Comm. of the ACM*, 35(8):102, August 1992.

[Sta83] L. Staiger. Finite-state ω-languages. *Journal of Computer and System Sciences*, 27(3):434–448, 1983.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science – Volume B: Formal Models and Semantics*, chapter 4, pages 133–191. Elsevier, Amsterdam, 1990.

[WB95] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proc. Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32, Glasgow, September 1995. Springer-Verlag.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97, Vancouver, July 1998. Springer-Verlag.

[WB00] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, March 2000. Springer-Verlag.