Book Title
Book Editors
IOS Press, 2003

Number-Set Representations for Infinite-State Verification

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

Abstract. In order to compute the reachability set of infinite-state models, one needs a technique for exploring infinite sequences of transitions in finite time, as well as a symbolic representation for the finite and infinite sets of configurations that are to be handled. The representation problem can be solved by *automata-based* methods, which consist in representing a set by a finite-state machine recognizing its elements, suitably encoded as words over a finite alphabet. Automata-based set representations have many advantages: They are expressive, easy to manipulate, and admit a canonical form.

In this survey, we describe two automata-based structures that have been developed for representing sets of numbers (or, more generally, of vectors): The *Number Decision Diagram (NDD)* for integer values, and the *Real Vector Automaton (RVA)* for real numbers. We discuss the expressiveness of these structures, present some construction algorithms, and give a brief introduction to some related acceleration techniques.

Keywords. Infinite-state systems, symbolic representations, number-set representations.

1. Introduction

Although all realizable computer systems are inherently finite, being able to verify infinite-state systems is interesting for several reasons. First, infinite-state models are good abstractions of large finite-state systems. Indeed, approximating a large finite domain by an unbounded one is often more precise than imposing unrealistically small bounds on data values, which is often done in order to keep the number of reachable configurations manageable. Second, infinite-state systems are natural models of *parameterized* systems, when the range of parameter values is unbounded. Finally, the solutions developed for analyzing infinite-state systems are usually also applicable to very large finite ones, which are out of reach of enumerative state-space exploration techniques.

In order to compute the reachability set of an infinite-state model, one needs a way of generating an infinite number of reachable configurations in finite time.

This problem is solved by *acceleration* methods, which are able to compute with a finite amount of resources the effect of following unbounded, and possibly infinite, sequences of transitions. As an example of acceleration technique, *meta-transitions* are generalized transitions that capture the repeated effect of cycles present in the control graph of the model [17,5]. Intuitively, following a meta-transition once leads to the set of all configurations that can be reached by following the corresponding cycle any number of times. After adding meta-transitions to the transition relation of the model, it may become possible to compute all its reachable configurations in a finite number of exploration steps. Note that, since the reachability problem is undecidable for most classes of systems that we consider, finding a suitable set of meta-transitions will not always be feasible in all situations.

Besides acceleration, one also needs a symbolic representation for the sets of configurations that are handled. If the models being analyzed have finite control, it is actually sufficient to represent symbolically sets of data values. A good representation system is one that satisfies the following requirements. First, in order to be able to carry out state-space exploration, sets of initial values need to be representable. Then, the representation should be effectively closed under all data operations performed by the models. One should also be able to compute unions of represented sets (so as to collect new reachable configurations at each exploration step), as well as to check inclusion between sets (in order to detect termination). The representation system should also be concise and efficient. Finally, it should be possible to apply acceleration methods to the represented sets.

A first solution to the representation problem is to use a formula-based data structure. For instance, in the case of programs based on unbounded integer variables, one can represent sets of data values by formulas expressed in Presburger arithmetic, i.e., the first-order theory $\langle \mathbf{Z}, +, \leq \rangle$ [43,44]. This representation system has the advantage of being closed under all Boolean operators, Cartesian product, and set projection. Moreover, reachable sets with a polyhedral and/or periodic structure, such as those often observed in models of communication protocols and distributed algorithms, are representable. However, formula-based representations have disadvantages. First, the representation of a given set is not unique, and its structure usually mimics the sequence of operations from which the set has been constructed. This is problematic for state-space exploration applications, in which reachable sets often have a simple structure, but are obtained after long sequences of manipulations. Second, comparing two formulas for equality or entailment is usually difficult and/or costly.

Another approach is to use automata-based representations [49,50,8,5,19,30, 42]. The idea consists in *encoding* each data value as one or many words over a given alphabet. This encoding scheme thus maps a data set into a language which, if it is regular, can be accepted by a finite-state automaton. An automata-based representation of a data set is thus a finite-state machine that accepts all encodings of all the values belonging to the set. This representation strategy has many advantages. First, it is sufficiently expressive for many applications. For instance, in the case of integer vector values encoded in the positional number system, with respect to a given integer base $r > 1$, it is known that the class of finite-state representable sets contains all Presburger-definable sets [22,21].

Second, automata are easy to manipulate algorithmically, and regular languages are closed under all usual set-theory operators: Boolean connectors, Cartesian product, projection, ... [29]. Finally, deterministic finite-state automata admit a minimized form that is canonical and easy to compute [28], which makes it possible to represent a set independently from the history of its construction.

Two main automata-based data structures have currently been defined for handling number sets. The first is the *Number Decision Diagram (NDD)* [49,5], in which the vectors components are encoded in a fixed base $r > 1$, either least or most significant digit first, with a number of digits that is not bounded. The encodings of all vector components are read synchronously, and thus the number of digits chosen for each of them must be identical. Negative numbers are represented by their r's complement. Algorithms have been developed for synthesizing NDDs from equations, inequations, or general Presburger formulas [49,20,51,12,18], as well as for creating and computing the effect of meta-transitions [5,6].

The second data structure is the *Real Vector Automaton (RVA)* [7,16], and is suited for sets of integer and/or real values, for instance the sets of reachable data values of models combining discrete variables and a dense representation of time. The idea is that a real number can be encoded in a base $r > 1$ as a word composed of a finite prefix, corresponding to the integer part of the number, followed by a separator, and then by an infinite suffix for the fractional part. A set of real vectors is thus mapped onto a language of infinite words over an alphabet composed of r digits and a separator. A RVA is simply a Büchi automaton that accepts such a language. Although infinite-word automata are often difficult to manipulate in practice [46,32,35,34], it has been shown that the sets defined in the first-order theory $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$ (the extension of Presburger arithmetic to real and integer variables) can always be represented by *weak* deterministic automata [10,11]. The advantage is that this restricted form of infinite-word automaton is practically as efficient to handle as finite-word ones. Just as for NDDs, a full set of theoretical tools is available for constructing the representation of sets defined by linear constraints or arithmetical formulas, and for applying various operators to the represented sets [16]. Finally, acceleration techniques have been developed for data transformations with both discrete and continuous features, when the sets are represented with RVA [9].

2. Number Decision Diagrams

2.1. Principles

Let $r \in \mathbf{N}$, with $r > 1$, be a *base*. The *positional number system* in base r consists in *encoding* numbers $z \in \mathbf{N}$ as words $d_{p-1}d_{p-2}\ldots d_1 d_0$ over the *digit alphabet* $\{0, 1, \ldots, r-1\}$, such that $z = \sum_{i=0}^{p-1} d_i r^i$.

This encoding scheme can easily be generalized to signed numbers using the *r's complement* method: An encoding of a number $z \in \mathbf{Z}$ such that $-r^{p-1} \leq z \leq 0$, with $p > 0$, is given by the last p digits of any unsigned encoding of $r^p + z$. With this method, the leading digit of an encoding corresponds to the *sign* of the number: It is equal to "0" for positive or zero values, and to "$r-1$" for negative

numbers. The leading digit of an encoding will hence be referred to as the *sign digit*. The number of digits p chosen for encoding a number $z \in \mathbf{Z}$ does not need to be fixed, but must be large enough to satisfy the constraint $-r^{p-1} \leq z < r^{p-1}$.

Note that this scheme associates every number with an infinite number of encodings, which can be obtained from the shortest one by repeating an arbitrary number of times the sign digit. Besides, remark that a word over $\{0, 1, \ldots, r-1\}$ is a valid encoding of a number only if its leading digit is either "0" or "$r-1$".

Example 1 *Let $Enc_r(z)$ denote the language of all base-r encodings of z. We have*

$$Enc_2(12) = (0)^+1100$$
$$Enc_2(-6) = (1)^+010.$$

□

In order to encode a vector $\vec{z} \in \mathbf{Z}^n$, with $n > 0$, one first encodes separately its components $z_1, z_2, \ldots, z_n$ into words $w_1, w_2, \ldots, w_n$, choosing for each of them the same number $p > 0$ of digits. The *synchronous encoding scheme* then consists in grouping successively, for each $i = 1, 2, \ldots, p$, the i-th digit of $w_1, w_2, \ldots, w_n$ into a n-tuple. The synchronous encodings of $\vec{v}$ thus take the form of words defined over the alphabet of tuples $\{0, 1, \ldots, r-1\}^n$. Note that the signs of the components of a vector are represented by the first symbol of its synchronous encodings, which we then call the *sign header*. The sign header can be repeated at will without modifying the value of the encoded vector.

Example 2 *Let $SynEnc_r(\vec{z})$ denote the language of all base-r synchronous encodings of $\vec{z}$. We have*

$$SynEnc_2((12, -6, 1)) = (0, 1, 0)^+(1, 1, 0)(1, 0, 0)(0, 1, 0)(0, 0, 1).$$

□

The disadvantage of the synchronous scheme is that the size of the word alphabet becomes exponential in the dimension of the vectors. The *serial encoding scheme* is a simple variant that does not have this drawback. The idea is to translate each tuple symbol $(w_{1i}, w_{2i}, \ldots, w_{ni}) \in \{0, 1, \ldots, r-1\}^n$ appearing in a synchronous encoding into the sub-word $w_{1i}w_{2i}\cdots w_{ni}$, which is defined over the much simpler alphabet $\{0, 1, \ldots, r-1\}$. In other words, a synchronous encoding

$$(w_{11}, w_{21}, \ldots, w_{n1})(w_{12}, w_{22}, \ldots, w_{n2})\cdots(w_{1p}, w_{2p}, \ldots, w_{np})$$

is translated into the serial one

$$w_{11}w_{21}\cdots w_{n1}\ w_{12}w_{22}\cdots w_{n2}\ \cdots\ w_{1p}w_{2p}\cdots w_{np}.$$

Note that all serial encodings of a vector $\vec{z} \in \mathbf{Z}^n$ have a length equal to an integer multiple of n. The sign header now corresponds to the n leading symbols of such encodings.

Example 3 *Let $SerEnc_r(\vec{z})$ denote the language of all base-r serial encodings of $\vec{z}$. We have*

$$SerEnc_2((12,-6,1)) = (010)^+110100010001.$$

□

We are now ready to define our finite-state representation of sets of vectors.

Definition 1 *Given a dimension $n > 0$, a base $r > 1$, and a (either synchronous or serial) encoding scheme, a* Number Decision Diagram (NDD) *representing a set $S \subseteq \mathbf{Z}^n$ is a finite-state automaton that accepts the language of all the encodings of the elements of S.*

A NDD can be deterministic or non-deterministic, depending on the properties of the underlying automaton. Converting a synchronous NDD into a serial one, or the other way around, can be done by simple operations. In practice, theoretical developments are easier to express using synchronous representations, while actual implementations preferably rely on the serial scheme to avoid the alphabet size overhead. It is also worth mentioning that we have arbitrarily defined NDDs as machines reading numbers most significant digit first. Since the properties of regular languages are mostly insensitive to the direction in which words are parsed, least-significant-digit-first NDDs can indifferently be used.

2.2. Expressiveness

The properties of automata recognizing sets of numbers have been exploited for a long time [22] for establishing the decidability of arithmetical theories. The following result [21] characterizes the expressiveness of NDDs in a given base $r > 1$.

Theorem 1 *A set $S \subseteq \mathbf{Z}^n$ is representable by an NDD in a base $r > 1$ iff it can be defined in the first-order theory $\langle \mathbf{Z}, +, \leq, V_r \rangle$, where V_r is defined as*

$$V_r : \mathbf{Z} \to \mathbf{N} : z \mapsto \begin{cases} \text{the greatest power of } r \text{ dividing } z & \text{if } z \neq 0 \\ 1 & \text{if } z = 0. \end{cases}$$

The following theorem characterizes the sets that can be represented in all integer bases [23,47,21].

Theorem 2 *A set $S \subseteq \mathbf{Z}^n$ is representable by an NDD in every base $r > 1$ iff it can be defined in the first-order theory $\langle \mathbf{Z}, +, \leq \rangle$, i.e., in Presburger arithmetic [43].*

Informally, Theorem 2 states that a set can be represented by an NDD if it can be expressed in linear additive arithmetic, i.e., by a formula in which variables are not multiplied together. The expressiveness of NDDs thus covers the sets defined as combinations of linear constraints and modular periodicities. These correspond to the reachability sets often observed during the analysis of programs manipulating integer variables [17], such as communication protocols or distributed algorithms.

2.3. Construction and Manipulation

There exists a simple algorithm [20] for constructing an NDD $\mathcal{A}$ representing the set of solutions of a linear equation $\vec{a}.\vec{x} = b$, with $a \in \mathbf{Z}^n$ and $b \in \mathbf{Z}$. This algorithm is based on the property that each non-initial node q of such an automaton accepts a language $L(q)$ that encodes the set of solutions of $\vec{a}.\vec{x} = \beta(q)$, i.e., the original equation in which the right-hand side has been replaced by a value $\beta(q)$ associated to q. The construction proceeds by computing the value of β for each state of the automaton, starting from a single accepting state q_F for which we know that $\beta(q_F) = b$ holds.

Then, one propagates the value of β to all the states of the automaton, moving backwards along its transitions. Consider a state q that leads to a state q' by a transition labeled by a tuple of digits $\vec{d} \in \{0, 1, \ldots, r-1\}^n$. A path from the initial state that reaches q after having read the encoding w of a vector $\vec{v}$ can be extended into a path that reaches q' by appending $\vec{d}$ to w. The resulting path thus reads an encoding of $\vec{v}' = r\vec{v} + \vec{d}$. Introducing $\vec{v}$ and $\vec{v}'$ into the equation, we get

$$\beta(\vec{v}) = \frac{\beta(\vec{v}') - \vec{a}.\vec{d}}{r},$$

which provides a way of computing $\beta(\vec{v})$ from $\beta(\vec{v}')$. Note that non-integer values of β do not correspond to valid states and can be discarded.

This procedure always terminates after a finite number of propagation steps [20] (thanks to the division by r performed by the propagation rule). In order to transform the resulting automaton into an NDD, one simply needs to add an initial state q_I, the outgoing transitions from which are labeled by the possible sign headers. A transition from q_I labeled by a tuple of digits $\vec{d} \in \{0, r-1\}^n$ reads an encoding of $-\vec{d}/(r-1)$, and thus leads to a state q such that

$$\beta(q) = -\frac{\vec{a}.\vec{d}}{r-1}.$$

Example 4 *A NDD representing the set of solutions of* $4x_1 - 2x_2 = -8$*, using a synchronous encoding in base* 2*, is given in Figure 1.* □

The construction algorithm that has just been outlined produces NDDs that are deterministic and minimal. This algorithm can easily be adapted to the set of solutions of linear inequalities [20,16,51].

A major advantage of automata-based set representations is that they can easily be combined by Boolean operators. Indeed, computing the intersection, union, difference, symmetric difference, or Cartesian product of two sets represented by NDDs simply reduces to carrying out the same operation on the languages accepted by the finite-state machines, for which there exist simple algorithms based on product constructions [29]. In the same way, testing NDD-represented sets for equality, inclusion, or emptiness also reduces to performing the same operations on the accepted languages. Furthermore, if the automata are systematically determinized and minimized into their canonical form [28], testing set equality amounts to a simple isomorphism check between the finite-state machines.

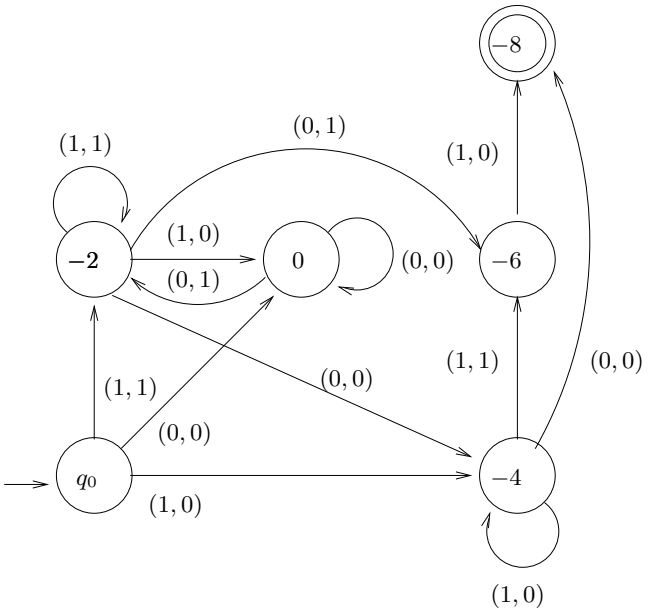

Figure 1. NDD representing the set of solutions of $4x_1 - 2x_2 = -8$.

Another operation of interest is the projection, which computes from a set $S \subseteq \mathbf{Z}^n$ and a variable index $i \in \{1, \ldots, n\}$ the set

$$\exists_i S = \{(v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n) \mid \exists v_i \in \mathbf{Z} : (v_1, \ldots, v_n) \in S\}.$$

A first idea for computing a representation $\mathcal{A}'$ of $\exists_i S$ from a NDD $\mathcal{A}$ representing S is to remove the i-th tuple component from all the transition labels of $\mathcal{A}$. Unfortunately, this method produces an automaton $\mathcal{A}|_{\neq i}$ that, even though it accepts valid encodings of all the elements of $\exists_i S$, is generally not an NDD. Indeed, for some vectors, the automaton may recognize their encodings only if they are of sufficient length (consider for instance $\exists_1\{(4, 1)\}$). In order to build $\mathcal{A}'$ from $\mathcal{A}|_{\neq i}$, one thus has to transform the automaton so as to make it also accept the shorter encodings of the vectors that it recognizes. An efficient solution to this problem is described in [12,13].

The constructions that have been outlined provide a simple algorithm for deciding Presburger arithmetic. Indeed, given a Presburger formula, one can build a NDD representing its set of solutions by starting from the atoms of the formula, which can be handled by the construction algorithms for linear equalities and inequalities. Logical connectors correspond to Boolean operations. Quantifiers are dealt with using projections and, for universal ones, complementation (which can be implemented as a set difference operation). The original formula is then satisfiable iff its set of solutions is non-empty. This decision procedure has been implemented in the tools LASH [36] and MONA [26].

Finally, an operation that has some applications [45] and that can be performed efficiently on NDD-represented sets is to count their number of elements (when it is finite). This operation can be performed by counting the number of

accepting paths in the underlying automaton, which can be done in linear time. A minor complication is that, since any given vector has an infinite number of valid encodings, one has to filter out the paths that do not encode distinct vectors. Algorithms for performing this operation are described in [12,13].

3. Real Vector Automata

In this section, we show that the automata-based set representations described in Section 2 can be generalized so as to handle sets of real and integer values (or vectors). The main motivation behind such a representation is the development of symbolic methods for analyzing *hybrid systems* [27], which combine discrete and continuous features, as well as *timed systems* relying on a dense representation of time [2].

3.1. Principles

Let $v \in \mathbf{R}$ be a real number and $r > 1$ be an integer. We encode v in base r, most significant digit first, using r's complement for negative numbers. The result is a word of the form $w = w_I \star w_F$, where w_I encodes the integer part v_I of v as a finite word over the alphabet $\{0, \ldots, r-1\}$, the symbol "$\star$" is a separator, and w_F encodes the fractional part v_F of v as an infinite word over the alphabet $\{0, \ldots, r-1\}$. We do not fix the length p of w_I, but only require it to be non-zero and large enough for $-r^{p-1} \leq v_I < r^{p-1}$ to hold. Hence, the most significant digit of a number will be "0" if this number is positive or equal to zero, and "$r-1$" otherwise. The length $|w_I|$ of w_I will be called the *integer-part length* of the encoding of v by w. For simplicity, we require that the length of w_F always be infinite (this is not a real restriction, since an infinite number of "0" symbols can always be appended harmlessly to w_F).

It is important to note that some numbers $v \in \mathbf{R}$ have two distinct encodings with the same integer-part length. For example, in base 10, the number $v = 11/2$ has the following two encodings with integer-part length 3 : $005 \star 5(0)^\omega$ and $005 \star 4(9)^\omega$ ($^\omega$ denotes infinite repetition). Such encodings are said to be *dual*.

To encode a vector of real numbers, we encode each of its components with 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. It follows that any vector has an infinite number of possible encodings. Using the same idea as the one discussed in Section 2 for integers, an encoding of a vector of reals $\vec{v} = (v_1, \ldots, v_n)$ can indifferently be viewed as a word over either the alphabet $\{0, \ldots, r-1\}^n \cup \{\star\}$ (synchronous encoding), or $\{0, \ldots, r-1, \star\}$ (serial encoding). Note that it is sufficient to make the separator occur only once in vector encodings, since it is always read simultaneously in all components.

Definition 2 *Given a dimension $n > 0$, a base $r > 1$, and a (either synchronous or serial) encoding scheme, a* Real Vector Automaton (RVA) *[7] representing a set $S \subseteq \mathbf{R}^n$ is a Büchi automaton [22] that accepts all encodings of the elements of S.*

3.2. Expressiveness

The sets of real vectors that can be represented by RVA in a given base $r > 1$ have been characterized in [16].

Theorem 3 *A set $S \subseteq \mathbf{R}^n$ is representable by an RVA in a base $r > 1$ iff it can be defined in the first-order theory $\langle \mathbf{R}, \mathbf{Z}, +, \leq, X_r \rangle$, where X_r is the predicate over $\mathbf{R}^3$ such that $X_r(x, u, k) = \mathbf{T}$ iff u is a (positive or negative) integer power of r, and there exists an encoding of x such that the digit at the position specified by u is equal to k.*

Thus, as in the case of NDDs, the expressiveness of RVA corresponds to first-order additive arithmetic, augmented by a base-dependent predicate. In most intended applications of RVA, the expressive power of this additional predicate is not needed, i.e., the represented sets can be restricted to the sub-theory $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$, which can be seen as a generalization of Presburger arithmetic to mixed real and integer variables. In this setting, the following result [10,11] states that Büchi automata can be replaced by a much simpler form of finite-word automaton.

Theorem 4 *Let $S \subseteq \mathbf{R}^n$ be definable in $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$. The set S can be represented by a* weak deterministic [41]. *RVA, i.e., a deterministic automaton whose state set Q can be partitioned into disjoint subsets $Q_1, \ldots, Q_m$ such that*

- *each Q_i contains only either accepting or non-accepting states, 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 there exists a transition from q to q', we have $Q_j \leq Q_i$.*

The advantage of weak deterministic automata is that they can be manipulated algorithmically in mostly the same way as finite-word automata, avoiding the intricacies of some manipulation procedures for infinite-word automata [46,32,35,34]. In particular, they can easily be complemented, as well as minimized into a canonical form [39]. Non-deterministic weak automata can also be determinized into co-Büchi automata [40,33] by a variant of the subset construction used with finite-word automata.

3.3. Construction and Manipulation

The constructions presented in Section 2.3 can be adapted to RVA. The idea behind the construction of a RVA representing the set of solutions in $\mathbf{R}^n$ of an equation $\vec{a}.\vec{x} = b$, with $\vec{a} \in \mathbf{Z}^n$ and $b \in Z$, is the following [16]. First, one decomposes the unknown $\vec{x}$ into the sum of two variables $\vec{x}_I$ and $\vec{x}_F$, respectively defined over $\mathbf{Z}^n$, and the real interval $[0, 1]^n$. Thus, $\vec{x}_I$ represents the integer part of the solution, and $\vec{x}_F$ its fractional part. The original equation then splits into $\vec{a}.\vec{x_I} = b'$ and $\vec{a}.\vec{x_F} = b - b'$.

From the former equation, b' must be equal to an integer multiple of $\gcd(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are the components of $\vec{a}$. From the latter, we get that $b - b'$ belongs to the interval $[a_-, a_+]$, where a_- (resp. a_+) is the sum

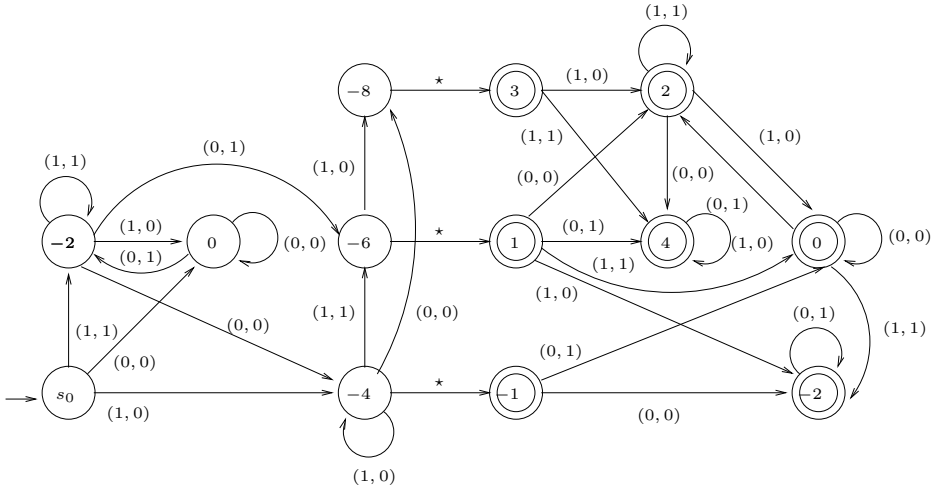

Figure 2. RVA representing the set of solutions of $4x_1 - 2x_2 = -8$.

of the negative (resp. positive) components of $\vec{a}$. Therefore, the range of possible values of b' forms a finite set that can easily be computed.

One can thus first build the part of the RVA recognizing the integer part of solutions by carrying out the procedure described in Section 2.3, starting from a set of states corresponding to the possible values β of b'. For each of those states, one then creates an outgoing transition, labeled by the separator "$\star$", leading to a state that will accept the fractional part of the solutions of $\vec{a}.\vec{x} = \beta'$, with $\beta + \beta' = b$.

In order to build the part of the RVA that accepts the fractional part of solutions, one can apply a forwards propagation rule. Consider a state q' that accepts the fractional part of the solutions of $\vec{a}.\vec{x} = \beta'(q')$. If q' can be reached by a state q by a transition labeled by $\vec{d} \in \{0, 1, \ldots, r-1\}^n$, and q accepts the fractional part of solutions of $\vec{a}.\vec{x} = \beta'(q)$, then we have

$$\beta'(q') = r\beta(q) - \vec{a}.\vec{d}.$$

This expression allows to compute the value of $\beta'(q')$ given $\beta'(q)$ and $\vec{d}$, i.e., to determine the outgoing transitions from the state q. Note that $\beta'(q')$ must belong to the interval $[a_-, a_+]$, otherwise there is no solution in $[0, 1]^n$ to $\vec{a}.\vec{x}' = \beta'(s')$. If this requirement is not satisfied, then there is no outgoing transition from q labeled by $\vec{d}$. The part of the RVA that recognizes the fractional part of solutions can then be constructed by applying repeatedly the propagation rule. Since the possible values of β' belong to a finite set, this operation always terminates. Finally one marks as being accepting all the states belonging to the fractional part of the RVA.

Example 5 *A RVA representing the set of solutions of* $4x_1 - 2x_2 = -8$*, using a synchronous encoding in base* 2*, is given in Figure 2.* □

A generalization of this construction to inequations is described in [16].

Boolean operators, Cartesian product, as well as emptiness and inclusion tests can be applied to RVA in much the same way as to NDDs, by using product constructions. Projecting sets represented by RVA is more difficult, because it generally produces non-deterministic Büchi automata. However, if one restricts the sets that are represented to the theory $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$, it has been shown in [10,11] that all RVA can straightforwardly be transformed into equivalent weak deterministic automata. Since all projections of a set in $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$ can themselves be expressed in that theory, one can always determinize the result of a projection, and then turn it into a weak deterministic RVA. This yields an algorithm for deciding $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$ that avoids the usual intricacies of infinite-word automata manipulation procedures. Such an algorithm is implemented in the tool LASH [36].

4. Acceleration Methods

We now give a brief overview of some techniques that have been developed for exploring infinite sequences of transitions in finite time, when the sets of reachable configurations are represented by NDDs or RVA.

4.1. Integer Transformations

We consider models for which the data domain is $\mathbf{Z}^n$, with $n > 0$, and the data transformations are *linear*, i.e., of the form

$$P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b},$$

with $P \in \mathbf{Z}^{m \times n}$, $\vec{q} \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, with $m \geq 0$. The left-hand part $P\vec{x} \leq \vec{q}$ of such a transformation is its *guard*, that must be satisfied by a data value in order to be able to apply the transformation. The right-hand part $\vec{x} := A\vec{x} + \vec{b}$ is an *assignment* that defines the transformation undergone by the values.

Linear transformations are closed under sequential composition, i.e., for every sequence $\theta_1; \theta_2; \ldots; \theta_p$ of such transformations, one can compute a single equivalent linear transformation θ. The idea behind *meta-transition*-based acceleration is to study sequences of transformations that can be *iterated* during state-space exploration. Consider for instance systems modeled by a finite control graph extended with n unbounded integer variables, the edges of this control graph being labeled by linear transformations. If there is a control cycle looping from some location to itself, the effect of following this cycle once can be described by a linear transformation θ. The *meta-transition* corresponding to that cycle is defined as the transformation $\theta^* = Id \cup \theta \cup \theta^2 \cup \cdots$, i.e., applying the meta-transition once amounts to following the cycle any possible number of times.

Meta-transitions preserve reachability, i.e., when applied to reachable sets of configurations, they only lead to configurations that are reachable as well. They can thus be added to the transition relation of the model in order to speed up state-space exploration, and even in some cases force its termination.

In order to construct meta-transitions, one needs to be able to check that the closure θ^* of a given transformation θ preserves the representability of sets,

i.e., to guarantee that $\theta^*(S)$ is representable for all representable sets S. This problem has been solved in [5,6] for linear transformations without guards, and sets of values represented by NDDs. The solution is formalized by the following theorems.

Theorem 5 *Let $n > 0$ be a dimension, $r > 1$ be a base, and $\theta \equiv A\vec{x} + \vec{b}$, with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$, be a guardless linear transformation.*

For every set $S \subseteq \mathbf{Z}^n$ representable by an NDD in base r, the set $\theta^*(S)$ is representable in the same way iff there exist $m, p \in \mathbf{N}$, with $p > 0$, such that

- *A^p is diagonalizable, and*
- *the eigenvalues of A^p all belong to $\{0, r^m\}$.*

Theorem 6 *Let $n > 0$ be a dimension and $\theta \equiv A\vec{x} + \vec{b}$, with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$, be a guardless linear transformation.*

For every set $S \subseteq \mathbf{Z}^n$ representable by an NDD in every base $r > 1$, the set $\theta^*(S)$ is representable in the same way iff there exists $p \in \mathbf{N}$, with $p > 0$, such that

- *A^p is diagonalizable, and*
- *the eigenvalues of A^p all belong to $\{0, 1\}$.*

Algorithms have been developed for checking these criteria using only integer arithmetic, and without computing explicitly eigenvalues [5,6]. The proof of correctness of these results is constructive, and can be turned into an algorithm for constructing a NDD representation of $\theta^*(S)$ from the parameters of θ and a representation of S.

For linear transformations guarded by systems of linear constraints, it has been shown [5,6] that Theorems 5 and 6 provide *sufficient* criteria for the preservation of representability of sets. In the particular case of Presburger-definable sets of values, this result has been generalized to arbitrary Presburger guards in [25]. Meta-transition detection and computation algorithms, as well as an infinite-state symbolic model checker that relies on them, have been implemented in the tool LASH [36].

4.2. Hybrid Transformations

Meta-transitions can be generalized [7] to models combining discrete and continuous data transformations over real variables. It has been shown in [9] that the data transformations θ labeling arbitrary control paths of *linear hybrid automata* [3,1,27], are of the form

$$\theta : \mathbf{R}^n \to 2^{\mathbf{R}^n} : \vec{v} \mapsto \left\{ \vec{v}' \mid P \begin{bmatrix} \vec{v} \\ \vec{v}' \end{bmatrix} \leq \vec{q} \right\},$$

with $P \in \mathbf{Z}^{m\times 2n}$, $\vec{q} \in \mathbf{Z}^m$, and $m \geq 0$.

For such transformations, algorithms for creating meta-transitions and computing their effect over sets represented by RVA have been developed in [9], and successfully applied to the reachability analysis of simple systems.

4.3. Generic Methods

The acceleration techniques sketched in Sections 4.1 and 4.2 proceed by computing, for a set of initial configurations S_0 represented by a finite-state machine, and a data transformation θ, a representation of the image $\theta^*(S_0)$ of S_0 by the reflexive and transitive closure of θ.

Let $\mathcal{A}_0$ be an automaton recognizing the elements of S_0, with respect to a suitable encoding scheme. Constructing a representation of $\theta^*(S_0)$ can be seen as computing the limit of the sequence of automata $\mathcal{A}_0$, $\mathcal{A}_1$,$\mathcal{A}_2$, ... where, for each $i \geq 0$, $\mathcal{A}_i$ represents $\theta^i(S_0)$.

In order to compute the limit of $\mathcal{A}_0$, $\mathcal{A}_1$,$\mathcal{A}_2$, ..., one can check whether the transition graph of some or all of these automata share common structures. For instance, if one detects that, within some prefix of the sequence, each automaton differs from its left neighbor only by an identical "increment", one can guess that the limit of the sequence can be reached by repeating this increment any number of times. This idea, as well as techniques for checking whether the guess that has been made is correct, is developed in [14,15]. Other techniques for computing limits of sequence of automata can be found in [19,48,24,30,42].

5. Conclusions

Automata-based representations of number sets have all the good properties needed by symbolic state-space exploration: They are expressive, easy to handle, closed under a large class of operators, and have a minimized form that is canonical. Although they have been known for a long time as theoretical tools for establishing the decidability of arithmetical theories, they only have recently started to be used as actual data structures in implemented applications [36,26,4].

Automata-based representations are not perfect, though. First, they are not always able to capture concisely the simple structure of some sets [31]. For instance, the minimal and deterministic NDD representing the set of solutions of $x_1 = 2^k x_2$, with $k \in \mathbf{N}$, has a number of states that grows exponentially with k. However, this relation can clearly be checked on the synchronous binary encodings of (x_1, x_2) using only $O(k)$ memory. Another example of sets that are not represented optimally are those with a Cartesian product structure. Indeed, an automaton recognizing the product of k sets essentially simulates the concurrent operation of k individual automata, and can thus have a number of states exponential in k. Whether there exists a finite-state representation of number sets that shares the good properties of NDD and RVA, while being able to represent concisely sets with a simple structure, is an interesting open problem. Another problem is to extract efficiently information from an automata-based representation of a number set, for instance, by synthesizing an arithmetical formula equivalent to the set. This problem is investigated in [37,38].

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, February 1995.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994.

[3] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 15th Annual Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[4] S. Bardin, J. Leroux, A. Finkel, and L. Petrucci. FAST: Fast accelereation of symbolic transition systems. In *Proc. 15th Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121, Boulder, USA, July 2003. Springer-Verlag.

[5] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. Collection des publications de la Faculté des Sciences Appliquées de l'Université de Liège, Liège, Belgium, 1999.

[6] B. Boigelot. On iterating linear transformations over recognizable sets of integers. *Theoretical Computer Science*, 309:413–468, 2003.

[7] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In *Proc. of the 9th International Conference on Computer-Aided Verification (CAV'97)*, number 1254 in Lecture Notes in Computer Science, pages 167–177, Haifa, Israël, June 1997. Springer-Verlag.

[8] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. *Formal Methods in System Design*, 14(3):227–255, May 1999.

[9] B. Boigelot, F. Herbreteau, and S. Jodogne. Hybrid acceleration using real vector automata. In *Proceedings 15th International Conference on Computer Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 193–205, Boulder, USA, 2003. Springer-Verlag.

[10] B. Boigelot, S. Jodogne, and P. 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'01)*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, Sienna, Italy, June 2001.

[11] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic*, 2005. To appear.

[12] B. Boigelot and L. Latour. Counting the solutions of Presburger equations without enumerating them. In *Proc. International Conference on Implementations and Applications of Automata*, volume 2494 of *Lecture Notes in Computer Science*, pages 40–51, Pretoria, July 2001. Springer-Verlag.

[13] B. Boigelot and L. Latour. Counting the solutions of Presburger equations without enumerating them. *Theoretical Computer Science*, 313:17–29, 2004.

[14] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. 15th Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, Boulder, USA, July 2003. Springer-Verlag.

[15] B. Boigelot, A. Legay, and P. Wolper. Omega-regular model checking. In *Proc. 10th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 561–575, Barcelona, Spain, 2004.

[16] B. Boigelot, S. Rassart, and P. Wolper. On the expressiveness of real and integer arithmetic automata. In *Proc. 25th Colloquium on Automata, Programming, and Languages (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*, pages

152–163. Springer-Verlag, July 1998.

[17] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In D. L. Dill, editor, *Proc. of the 6th Int. Conf. on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67, Stanford, June 1994. Springer-Verlag.

[18] B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata: An overview. In *Proc. International Conference on Logic Programming (ICLP)*, volume 2401 of *Lecture Notes in Computer Science*, pages 1–19, Copenhagen, July 2002. Springer-Verlag.

[19] A. Bouajjani, B. Jonsson, M. Nilsson, and Tayssir Touili. Regular model checking. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

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

[21] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society*, 1(2):191–238, March 1994.

[22] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. of the International Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, 1962. Stanford University Press.

[23] A. Cobham. On the base-dependence of sets of numbers recognizable by finite automata. *Mathematical Systems Theory*, 3:186–192, 1969.

[24] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In *Proceedings 13th International Conference on Computer Aided Verification (CAV)*, volume 2102 of *Lecture Notes in Computer Science*, pages 286–297, Paris, France, 2001. Springer-Verlag.

[25] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proc. 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'02)*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156, Kanpur, India, December 2002. Springer-Verlag.

[26] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, Th. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer-Verlag, 1995.

[27] T. A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

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

[29] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[30] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.

[31] Felix Klaedtke. On the automata size for Presburger arithmetic. In *Proc. of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 110–119. IEEE Computer Society Press, 2004.

[32] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, San Juan, October 1991.

[33] 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.

[34] O. Kupferman and M.Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *11th International Conference on Tools and algorithms for the construction and analysis of systems*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[35] O. Kupferman and M.Y. Vardi. From complementation to certification. In *10th International Conference on Tools and algorithms for the construction and analysis of systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 591–606. Springer-Verlag, 2004.

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

[37] L. Latour. From automata to formulas: convex integer polyhedra. In *Proc. of the 19th IEEE symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press, pages 120–129, 2004.

[38] J. Leroux. A polynomial time Presburger criterion and synthesis for number decision diagrams. In *Proc. LICS*, 2005. To appear.

[39] C. Löding. Efficient minimization of deterministic weak $\omega-$automata. *Information Processing Letters*, 79(3):105–109, 2001.

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

[41] 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*, pages 275–283, Rennes, 1986. Springer-Verlag.

[42] Marcus Nilsson. *Regular Model Checking*. PhD thesis, Uppsala University, Sweden, 2005.

[43] M. Presburger. Über die Volständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves*, pages 92–101, Warsaw, Poland, 1929.

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

[45] W. Pugh. Counting solutions to Presburger formulas: How and why. *SIGPLAN*, 94-6/94:121–134, 1994.

[46] S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 319–327, 1988.

[47] A. L. Semenov. Presburgerness of predicates regular in two number systems. *Siberian Mathematical Journal*, 18:289–299, 1977.

[48] T. Touili. Regular model checking using widening techniques. In *Proceeding of Workshop on Verification of Parametrized Systems (VEPAS'01)*, volume 50 of *Electronic Notes in Theoretical Computer Science*, 2001.

[49] P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In Mycroft, A., editor, *Proc. of the 2nd Int. Symp. on Static Analysis (SAS'95)*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 1995.

[50] P. Wolper and B. 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.

[51] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proceedings of TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, March 2000. Springer-Verlag.