Chapter 32

Symbolic methods and automata

Bernard Boigelot

Contents

1. Introduction

In addition to being useful mathematical objects, automata can be employed as actual data structures. In this chapter, we use automata for symbolically representing sets of values over a given (generally infinite) domain D. The aim is thus to describe subsets of D, without ambiguity, using finite-state machines. Note that a data structure that only admits a countable number of possible instances will never be able to represent all the subsets of an infinite domain. In most cases, the symbolic representation will need to be expressive enough to cover a class of sets that is deemed to be useful for the considered application. Additional requirements include the possibility of building representations of elementary sets from their description, and performing operations of interest on the represented sets.

A finite automaton can be seen as a data structure that provides a symbolic representation of its accepted language. In order to use automata as symbolic representations of sets over a domain D, a natural idea is to define a mapping from the elements of D to the words read by the automata [8]. This mapping needs to be unambiguous, in the sense that every word can only correspond to at most one value in D. In this way, a set $X \subseteq D$ is mapped onto a language that precisely describes its elements, and a finite automaton that accepts this language can be seen as a symbolic representation of X.

Definition 1.1. Let D be a domain and A be a finite alphabet. An *encoding relation* from D to A is a relation E such that

- either $E \subseteq D \times A^*$ (finite encodings) or $E \subseteq D \times A^\omega$ (infinite encodings),
- the relation E is total over D, that is, for all $d \in D$, there exists $w \in A^* \cup A^\omega$ such that $(d, w) \in E$, and
- the relation E is injective, that is, for all $(d, w), (d', w') \in E$, the condition $w = w'$ implies $d = d'$.

The *encoding* of a set $X \subseteq D$ by E is the language $E(X) = \bigcup_{d \in X} E(d)$.

Definition 1.2. Let E be an encoding relation from a domain D to an alphabet A, and let $X \subseteq D$ be a set of values. An automaton over A is a *finite-state representation* of X or, equivalently, is said to *recognise* X, if it accepts the language $E(X)$.

In this definition, we have not fixed the form of the automata used for representing sets, which can be specific to the application of interest.

One advantage of using automata-based representations of sets is that computing Boolean combinations of sets reduces to performing the same operations on the languages accepted by their finite-state representations. The same property holds for set comparison operations such as emptiness, inclusion or equality checking.

Theorem 1.1. *Let E be an encoding relation from a domain D to an alphabet A, and $X, X' \subseteq D$ be sets of values. We have*

$$\begin{aligned} E(X \cup X') &= E(X) \cup E(X'), \\ E(X \cap X') &= E(X) \cap E(X'), \\ E(X \setminus X') &= E(X) \setminus E(X'). \end{aligned}$$

Furthermore,

$$\begin{aligned} X = \emptyset &\iff E(X) = \emptyset, \\ X \subset X' &\iff E(X) \subset E(X'), \\ X = X' &\iff E(X) = E(X'). \end{aligned}$$

Since regular and ω-regular languages are closed under Boolean operators, it follows from Theorem 1.1 that finite-state representations of sets naturally enjoy the same closure property. Combining automata by means of Boolean operators over their accepted languages is algorithmically simple with most forms of automata [29] (a noteworthy exception being the operations that rely on complementing Büchi automata [32]).

From an algorithmic point of view, another advantage of automata-based representations is that deterministic automata over finite words, as well as some types of infinite-word automata, can easily be minimised into a canonical form [28] and [35]. This makes it possible to obtain a symbolic representation of a set that is independent of the history of its construction. In particular, this simplifies set comparison operations, since equality testing then reduces to checking isomorphism between transition graphs of automata.

Note that Theorem 1.1 would not hold any longer if encodings of sets were allowed to contain only some encodings of their elements, as opposed to all of them. Also notice that all words over the considered alphabet A are not required to be encodings of values in D. We have the following definition.

Definition 1.3. Let $E \subseteq D \times A$ be an encoding relation over a domain D. The set V of *valid encodings* of E over D is defined as the language $V = E(D)$.

In most applications, a natural requirement is to impose that V is a regular or ω-regular language. In such a case, the complement $D \setminus X$ of a set $X \subseteq D$ is encoded by the language $V \setminus E(X)$, as a consequence of Theorem 1.1, and is thus representable by an automaton provided that X is representable as well.

In the next sections, we present finite-state representation systems suited for several data domains that are relevant to actual applications, and discuss their properties.

2. Integer domain

In this section, we consider the domain $D = \mathbb{Z}^n$, with $n > 0$, that is, we study the representations of sets of integer vectors with a fixed dimension n.

2.1. Encoding relation. In order to obtain a finite-state representation system, we need to define an encoding relation over the elements of the domain. We first discuss the encoding of natural numbers. A simple solution is to use the positional k-ary notation. One selects a *numeration base* $k > 1$, which provides an alphabet of *digits* $A_k = \{0, 1, \ldots, k-1\}$. A finite word $a_{p-1}a_{p-2}\cdots a_0 \in A_k^*$, with $p \geqslant 0$, then encodes the number $\sum_{i=0}^{p-1} a_i k^i$. The length p of an encoding does not need to be fixed, provided that it is large enough. Every natural number thus admits infinitely many encodings that only differ by the number of repetitions of a leading digit equal to 0.

This encoding relation generalises to signed numbers thanks to the *base-complement* method. Under this scheme, a number $z \in \mathbb{Z}$ is encoded by the last p digits $a_{p-1}a_{p-2}\cdots a_1a_0 \in A_k^*$ of the (unsigned) encodings of $k^p + z$, for all $p > 0$ such that $z \in [-k^{p-1}, k^{p-1} - 1]$. Notice that if z is negative, then we have $k^p + z \in [(k-1)k^{p-1}, k^p - 1]$, which implies $a_{p-1} = k - 1$. Otherwise, if z is nonnegative, then we have $k^p + z \in [k^p, k^p + k^{p-1} - 1]$, and hence $a_{p-1} = 0$. It follows that the leading digit of an encoding can be understood as a *sign digit*, and belongs to the restricted alphabet $\{0, k-1\}$. Every integer admits an infinite number of encodings, which only differ by the number of repetitions of their sign digit. The valid encodings of integers in base k form the language $\{0, k-1\}A_k^*$.

Let us now discuss encodings of vectors $\vec{v} = (v_1, v_2, \ldots v_n) \in \mathbb{Z}^n$. The idea is to first encode each component v_i separately into a word w_i, choosing the same number p of digits for each of them, that is, $|w_1| = |w_2| = \cdots = |w_n| = p$. This is always possible, for the sign digit of an encoding can be repeated at will. Then, one reads sequentially and repeatedly one symbol in each w_i, obtaining a word w of length p over the alphabet $(A_k)^n$ that encodes $\vec{v}$. In other words, if we have $w_i = a_{i,p-1}a_{i,p-2}\cdots a_{i,0}$ for each $i \in [1, n]$, then the encoding is defined as the word $w = (a_{1,p-1}, \ldots, a_{n,p-1})(a_{1,p-2}, \ldots, a_{n,p-2})\cdots(a_{1,0}, \ldots, a_{n,0})$.

With this encoding technique, the leading symbol of a vector encoding characterises the sign of its components, and can thus be seen as a *sign symbol*. This symbol belongs to the restricted alphabet $\{0, k-1\}^n$. The encodings of a given vector only differ by the number of repetitions of their sign symbol. The language containing the valid base-k encodings of vectors in $\mathbb{Z}^n$ is given by $\{0, k-1\}^n((A_k)^n)^*$.

Example 2.1. Let $\vec{v} = (0, -7, 3)$ and $k = 2$. The encodings of the components of $\vec{v}$ form the languages $E(0) = 0^+$, $E(-7) = 1^+001$ and $E(3) = 0^+11$. The encodings of $\vec{v}$ must therefore be at least of length 4, which yields $E(\vec{v}) = (0,1,0)^+(0,0,0)(0,0,1)(0,1,1)$.

In most applications, relying on an alphabet that is exponential in the dimension of the domain is problematic. A simple workaround is to *serialise* the encodings, which consists in reading the tuple components of a symbol $(a_1, a_2, \ldots, a_n)$ sequentially, that is, as the word $a_1 a_2 \cdots a_n$, rather than simultaneously as a single symbol. Serialised encodings in base k are thus expressed over the alphabet A_k. The signs of the components of a vector correspond to the first n digits of its serialised encodings, which are known as their *sign header*. The language of valid serialised encodings is given by

$$\{uv \mid u \in \{0, k-1\}^*, v \in (A_k)^*, |u| = n, |v| \equiv 0 \,(\mathrm{mod}\, n)\}.$$

Example 2.2. Let $\vec{v} = (0, -7, 3)$. The serialised encodings of $\vec{v}$ in base $k = 2$ form the language $E(\vec{v}) = (010)^+000001011$.

From a theoretical point of view, working with serialised or unserialised encodings is mostly equivalent. Indeed, an automaton recognising unserialised encodings can easily be turned into one operating on serialised ones by renaming its edge labels. The reciprocal transformation can be achieved by, for instance, using a finite-state transducer that rewrites sequences of n consecutive digits into a single symbol. In this chapter, for clarity's sake, we will assume that encodings of vectors are not serialised, unless stated otherwise. In practical implementations however, working with a small alphabet is essential to achieving efficiency, and serialised encodings are preferred.

Definition 2.1. Let $n > 0$ be a dimension, and $k > 1$ be a numeration base. Let E denote the (either serialised or unserialised) encoding relation suited for $\mathbb{Z}^n$ in base k. Let $X \subseteq \mathbb{Z}^n$. A finite-word automaton is a *number decision diagram* (NDD) representing X if it accepts the language $E(X)$.

Example 2.3. Let $n = 2$ and $k = 2$. The automaton illustrated in Figure 1 is an NDD representing the set $\{(y, z) \in \mathbb{N}^2 \mid y < z\}$.

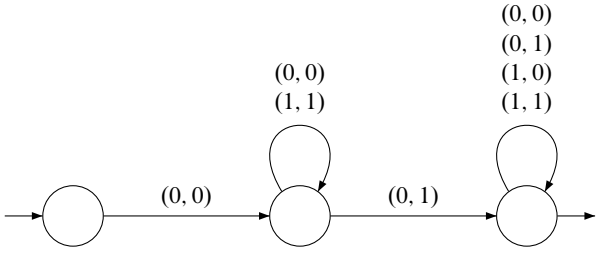

Figure 1. Example of NDD

2.2. Expressive power. The expressive power of automata recognising sets of integers and vectors has been well studied. We recall some results that are discussed in detail in Chapter 26. The following theorem characterises the sets that can be recognised by NDD in a given base, see [23] and [22].

Theorem 2.1 (Büchi–Bruyère). *Let $n > 0$ be a dimension and $k > 1$ be a numeration base. Let $V_k\colon \mathbb{N} \to \mathbb{N}$ denote the function such that $V_k(0) = 1$ and $V_k(x)$ is the highest integer power of k that divides x for all $x > 0$. A set $X \subseteq \mathbb{Z}^n$ is recognisable by an NDD in base k if and only if it can be defined in the first-order theory $\langle \mathbb{Z}, +, <, V_k \rangle$.*

Actually, the result expressed by Theorem 2.1 was originally established for the domain $\mathbb{N}^n$. It generalises to $\mathbb{Z}^n$ thanks to the following mechanism. A formula in the theory $\langle \mathbb{Z}, +, <, V_k \rangle$ can be rewritten as a formula in $\langle \mathbb{N}, +, <, V_k \rangle$ by replacing each variable x over $\mathbb{Z}$ by the difference $y - z$ of two variables over $\mathbb{N}$. The reciprocal transformation is immediate. A similar transformation is applicable to NDD: an automaton recognising vectors $(x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n$ can be turned into one operating on vectors $(y_1, z_1, y_2, z_2, \ldots, y_n, z_n) \in \mathbb{N}^{2n}$ such that $x_i = y_i - z_i$ for all i, and *vice versa*. Algorithms for performing these transformations on NDD will be presented in § 2.4.

The following result is a corollary of Cobham and Semenov's theorems (see [24] and [42]), and characterises the sets of integer vectors that are recognisable by NDD regardless of the chosen numeration base. A detailed discussion of this theorem, its consequences, and its generalisations can be found in Chapter 26.

Theorem 2.2. *Let $n > 0$ be a dimension. A set $X \subseteq \mathbb{Z}^n$ is recognisable by an NDD in every base $k > 1$ if and only if it can be defined in the first-order theory $\langle \mathbb{Z}, +, < \rangle$.*

The theory $\langle \mathbb{Z}, +, < \rangle$ can be seen as an extension to integers of the first-order additive theory of natural numbers $\langle \mathbb{N}, + \rangle$, also known as *Presburger arithmetic,* see [40] and [41]. In this chapter, we will refer to Presburger arithmetic as the theory $\langle \mathbb{Z}, +, < \rangle$ when reasoning about signed integers, and say that a set that is definable in that theory is *Presburger definable*.

Presburger arithmetic is quite an expressive formalism. In particular, every *modular linear constraint* of the form $\{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} \mathbin{\#} b\}$, with $\vec{a} \in \mathbb{Z}^n$, $b \in \mathbb{Z}$ and $\# \in \{<, \leqslant, =, \geqslant, >, \equiv_2, \equiv_3, \equiv_4, \ldots\}$ is Presburger definable, where $y \equiv_i z$ is shorthand for $y \equiv z \pmod{i}$ for all $i > 1$. In addition, Presburger arithmetic is, by definition, closed under Boolean operators, and operations such as Cartesian product and projection of sets.

It follows from the previous results that NDD provide a symbolic representation of Presburger-definable sets of integer vectors. In § 2.3 and § 2.4, we will introduce algorithms for constructing NDD that represent some elementary sets, and for performing operations on represented sets.

2.3. Construction of elementary sets

2.3.1. Linear equalities. Consider a dimension $n > 0$ and a linear constraint

$$X = \{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} = b\},$$

with $\vec{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$. From Theorem 2.1, the set X is recognisable by an NDD in any base $k > 1$. We now develop an algorithm for constructing such a NDD.

In an NDD recognising X, the paths originating from an initial state and ending in an accepting state q precisely read the encodings of the vectors $\vec{x}$ that satisfy the constraint $\vec{a}.\vec{x} = b$. This can be seen as a property of the state q, which can be labelled by the constraint $\vec{a}.\vec{x} = b$, or more simply, by its constant term b. This prompts us to define a labelling β of the states of the automaton by integer values, such that $\beta(q) = b$ for the state q that we have considered.

Now let q' be a state from which q can be reached by following a single edge e, and let $\vec{d} \in (A_k)^n$ be the tuple of digits labelling this edge. Any path π' from an initial state to q' can be completed to a path π that reaches q by appending the edge e. If π' is not empty, then the vectors $\vec{x}$ and $\vec{x}'$ respectively recognised by π and π' satisfy the relation $\vec{x} = k\vec{x}' + \vec{d}$. If π' is empty, then e reads a sign symbol; this case will be discussed later.

We therefore have $k\vec{a}.\vec{x}' + \vec{a}.\vec{d} = \vec{a}.\vec{x} = \beta(q)$, which gives $\vec{a}.\vec{x}' = \beta(q')$, with

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

In other words, every path from an initial state to q' recognises a vector $\vec{x}'$ that satisfies $\vec{a}.\vec{x}' = \beta(q')$. Recursively applying the same reasoning on the predecessor states of q', one eventually obtains a labelling of all the states of the NDD from which an accepting state can be reached (with the possible exception of initial ones, since we have rejected empty paths).

Those observations lead to a procedure for constructing a deterministic NDD representing $X = \{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} = b\}$. The first step is to create a single accepting state q labelled by b, that is, such that $\beta(q) = b$. Then, for every symbol $\vec{d} \in (A_k)^n$, one checks whether q admits an incoming edge labelled by $\vec{d}$, originating from some state q'. This is done by applying the backward propagation rule (1), and checking that the resulting value of $\beta(q')$ matches the constant term of a constraint $\vec{a}.\vec{x} = \beta(q')$, for some $\vec{x} \in \mathbb{Z}^n$. This is the case if and only if $\beta(q')$ is an integer divisible by $\gcd(a_1, \ldots, a_n)$. In this case, a new edge ending in q and labelled by $\vec{d}$ is created. The value of $\beta(q')$ then either identifies a previously created state q', or prompts the creation of a new one. In the latter case, the procedure is recursively applied to the new state.

The last step is to create the initial state q_0 of the NDD. According to our encoding relation, a single sign symbol $\vec{d} = (d_1, d_2, \ldots, d_n) \in \{0, k-1\}^n$ encodes a vector $(x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n$ such that for all $i \in [1, n]$, $x_i = 0$ if $d_i = 0$, and $x_i = -1$ if $d_i = k-1$. In other words, we have $\vec{x} = 1/(1-k)\vec{d}$. One can therefore explore all the sign symbols $\vec{d} \in \{0, k-1\}^n$ and for each of them, compute the corresponding value of $\vec{x}$ and check whether a state q such that $\vec{a}.\vec{x} = \beta(q)$ has been built. In the positive case, an edge from q_0 to q labelled by $\vec{d}$ needs to be added.

The algorithm for constructing a deterministic NDD recognising the set

$$\{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} = b\}$$

is summarised in Algorithm 1.

Algorithm 1 Algorithm for building a base-k NDD representing $\vec{a}.\vec{x} = b$

1. Create a table Q of states and a list L of labels of "active" states, both initialised to $\{b\}$.
2. Set $I = E = \emptyset$, and $T = \{b\}$.
3. While $L \neq \emptyset$, remove a value v from L, and for every $\vec{d} \in \{0, 1, \ldots, k-1\}^n$,
   - if $v' = \frac{v - \vec{a}.\vec{d}}{k}$ is an integer multiple of $\gcd(a_1, \ldots, a_n)$, then
     1. if $v' \notin Q$, then add v' to Q and L;
     2. add an edge $(v', \vec{d}, v)$ to E.
4. Add a new initial state q_0 to Q and I, and, for every $\vec{d} \in \{0, k-1\}^n$,
   - if $v = \frac{\vec{a}.\vec{d}}{1-k}$ is such that $v \in Q$, then add an edge $(q_0, \vec{d}, v)$ to E.
5. Return (Q, I, E, T).

It is shown in [19] that this procedure always terminates. Consider an accepting state q, and one of its immediate predecessors q' such that $q' \neq q_0$. From (1), we get

$$\beta(q') \in \frac{1}{k}[b - (k-1)a_+, b - (k-1)a_-],$$

where $a_+ = \sum_{a_i > 0} a_i$ and $a_- = \sum_{a_i < 0} a_i$.

A state from which q is reachable after following a path of length $p > 1$ thus has a label that belongs to the interval

$$\left[\frac{b}{k^p} - \sum_{i=1}^{p} \frac{k-1}{k^i} a_+, \frac{b}{k^p} - \sum_{i=1}^{p} \frac{k-1}{k^i} a_-\right] \subseteq \left[\frac{b}{k^p} - (k-1)a_+, \frac{b}{k^p} - (k-1)a_-\right].$$

If $r^p > |b|$, since the state labels are restricted to integer values, this interval reduces to $[-(k-1)a_+, -(k-1)a_-]$. It follows that the only labels different from b that have to be considered during the construction are those belonging to the union of intervals

$$\bigcup_{i=1}^{\ell} \left[\frac{b}{k^i} - (k-1)a_+, \frac{b}{k^i} - (k-1)a_-\right],$$

where $\ell = \lfloor \log_k |b| \rfloor + 1$ if $b \neq 0$, and $\ell = 0$ if $b = 0$. The construction thus terminates after having created at most $\ell((k-1)(a_+ - a_-) + 1)$ states, in addition to the initial and accepting ones.

Theorem 2.3. *Let $n > 0$ be a dimension, $k > 1$ be a base and let $X = \{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} = b\}$ be a linear constraint, with $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$. There exists an NDD recognising X in base k with $O\big(k(\log |b|) \sum_{i=1}^{n} |a_i|\big)$ states.*

Example 2.4. An NDD representing the set $\{(x_1, x_2) \in \mathbb{Z} \mid 2x_1 - x_2 = -4\}$ in base 2 is illustrated in Figure 2. The state labels correspond to those used by Algorithm 1.

Finally, it is worth mentioning that the procedure that constructs an NDD from a linear constraint can easily be adapted for producing automata that operate on serialised encodings of vectors.

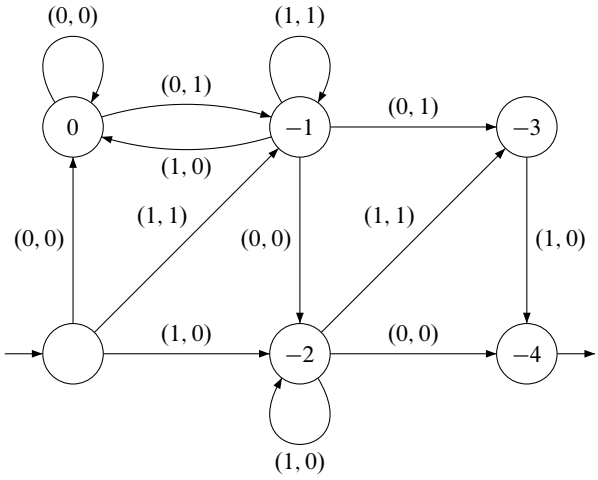

Figure 2. NDD recognising $2x_1 - x_2 = -4$

2.3.2. Linear inequations. The algorithm introduced in § 2.3.1 can straightforwardly be turned into one constructing the base-k representation of a linear inequation $X = \{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} \leqslant b\}$, with $k > 1$, $\vec{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$. By the same reasoning, each state q of an NDD recognising X can be labelled by a value $\beta(q)$, such that the paths from an initial state to q recognise the vectors $\vec{x}$ that satisfy $\vec{a}.\vec{x} \leqslant b$.

The application of the propagation rule (1) needs to be adapted slightly. Consider a state q for which the value of $\beta(q)$ is known, and a state q' linked to q by a single edge $(q', \vec{d}, q)$. If the value of $\beta(q')$ given by (1) is not an integer multiple of $\gcd(a_1, \ldots, a_n)$, then the inequality $\vec{a}.\vec{x} \leqslant \beta(q')$ may nevertheless admit integer solutions $\vec{x}$. The set of these solutions is actually identical to that of the constraint $\vec{a}.\vec{x} \leqslant g\lfloor \beta(q')/g \rfloor$, where $g = \gcd(a_1, \ldots, a_n)$. In this case, the value of $\beta(q')$ does not have to be discarded as in the case of linear equalities, but must be suitably rounded into an integer value.

A similar adaptation is needed for generating the edges linking the initial state q_0 to labelled states. Recall that a sign symbol $\vec{d} \in \{0, k-1\}^n$ encodes the vector $\vec{x} = 1/(1-k)\vec{d}$. For each such $\vec{d}$, one computes the corresponding value of $\vec{x}$, and creates edges labelled by $\vec{d}$ from s_0 to all the states q such that $\vec{a}.\vec{x} \leqslant \beta(q)$. The resulting algorithm is given in Algorithm 2.

Algorithm 2 Algorithm for building a base-k NDD representing $\vec{a}.\vec{x} \leqslant b$

1. Create a table Q of states and a list L of labels of "active" states, both initialised to $\{b\}$.
2. Set $I = E = \emptyset$.
3. While $L \neq \emptyset$, remove a value v from L, and for every $\vec{d} \in \{0, 1, \ldots, k-1\}^n$,
   a. compute $v' = \frac{v - \vec{a}.\vec{d}}{k}$ and $v'' = g\lfloor \frac{v'}{g} \rfloor$, where $g = \gcd(a_1, \ldots, a_n)$;
   b. if $v'' \notin Q$, then add v'' to Q and L;
   c. add an edge $(v'', \vec{d}, v)$ to E.
4. Add a new initial state q_0 to Q and I, and, for every $\vec{d} \in \{0, k-1\}^n$ and $v \in Q$,
   - if $\frac{\vec{a}.\vec{d}}{1-k} \leqslant v$, then add an edge $(q_0, \vec{d}, v)$ to E.
5. Set $T = \{b' \in Q \mid b' \leqslant b\}$.
6. Return (Q, I, E, T).

An important difference between Algorithm 1 and Algorithm 2 is that the latter generally produces nondeterministic NDD. This may be problematic in some applications, in particular if automata need to be minimised in order to obtain canonical set representations.

One can of course always compute a deterministic form of an NDD using the traditional subset construction method [29]. It is however known that the automata generated by Algorithm 2 have a special structure that makes it possible to determinise them much more efficiently. The following result is established in [44].

Theorem 2.4. *Let $\mathcal{A}$ be a nondeterministic NDD produced by Algorithm 2. This NDD can be determinised in linear time with respect to its number of states.*

A construction that directly generates a deterministic and minimal NDD recognising the set of solutions of a linear inequation is given in [30]. Finally, notice that the results of this section can also be applied to the construction of base-k NDD recognising sets of the form $\{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} \# b\}$, with $k > 1$, $\vec{a} \in \mathbb{Z}^n$, $b \in \mathbb{Z}$, and $\# \in \{<, >, \geqslant\}$. Indeed, in the integer domain, we have $\vec{a}.\vec{x} < b \iff \vec{a}.\vec{x} \leqslant b - 1$, $\vec{a}.\vec{x} > b \iff -\vec{a}.\vec{x} \leqslant -b - 1$, and $\vec{a}.\vec{x} \geqslant b \iff -\vec{a}.\vec{x} \leqslant -b$.

2.3.3. Modular constraints. We now study the construction of NDD recognising sets of the form $X = \{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} \equiv_m b\}$ in a base $k > 1$, with $m > 1$, $\vec{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$, and $b \in [0, m-1]$. Recall that $y \equiv_m z$ is shorthand for $y \equiv z \pmod{m}$.

The construction algorithm is based on similar principles as the one developed for linear equalities, but the propagation rule (1) needs to be adapted to modular arithmetic. A state q of the constructed automaton is now labelled by a value $\beta(q)$ such that every vector $\vec{x}$ recognised by a path leading from the initial state to q satisfies $\vec{a}.\vec{x} \equiv_m \beta(q)$. As a consequence, it is sufficient to consider the labels that belong to $\{0, 1, \ldots, m-1\}$ and that are divisible by $g = \gcd(m, a_1, a_2, \ldots, a_n)$.

In order to construct a deterministic NDD recognising X in base k, it is simpler to apply a forward rather than backward propagation rule. First, one creates $\frac{m}{g}$ labelled

states, as well as a separate initial state q_0. The state q_T whose label satisfies $\beta(q_T) \equiv_m b$ is made accepting.

For each labelled state q and symbol $\vec{d} \in (A_k)^n$, the destination q' of the edge $(q, \vec{d}, q')$ must be such that

$$\beta(q') \equiv_m k\beta(q) + \vec{a}.\vec{d},$$

which provides a simple rule for creating the edges between labelled states.

It remains to connect the initial state q_0 to the other states of the automaton. For a sign digit $\vec{d} \in \{0, k-1\}^n$, the edge $(q_0, \vec{d}, q)$ recognises the vector $\vec{x} = (1/(1-k))\vec{d}$, and its destination is thus the state q that satisfies $\beta(q) \equiv_m \vec{a}.\vec{x}$.

The algorithm for constructing a deterministic NDD recognising the set

$$\{\vec{x} \in \mathbb{Z}^n \mid \vec{a}.\vec{x} \equiv_m b\}$$

is summarised in Algorithm 3.

Algorithm 3 Algorithm for building a base-k NDD representing $\vec{a}.\vec{x} \equiv_m b$

1. Compute $g = \gcd(m, a_1, \ldots, a_n)$, and create a table Q of states, initialised to $\{ig \mid i \in \mathbb{N}, ig < m\}$.
2. Set $I = E = \emptyset$, and $T = \{b'\}$, where $b' \in [0, m-1]$ and $b' \equiv_m b$.
3. For each $v \in Q$ and $\vec{d} \in \{0, 1, \ldots, k-1\}^n$:
   a. compute $v' \in [0, m-1]$ such that $v' \equiv_m kv + \vec{a}.\vec{d}$;
   b. add an edge $(v, \vec{d}, v')$ to E.
4. Add a new initial state q_0 to Q and I, and, for every $\vec{d} \in \{0, k-1\}^n$,
   a. compute $v' \in [0, m-1]$ such that $v' \equiv_m \frac{\vec{a}.\vec{d}}{1-k}$;
   b. add an edge $(q_0, \vec{d}, v')$ to E.
5. Return (Q, I, E, T).

2.4. Operations on sets. In this section, we study algorithms for performing various operations on sets represented by NDD. A first problem is to construct an NDD representing a set $X \subseteq \mathbb{Z}^n$ defined by a Presburger formula. Without loss of generality, we may assume that the formula defining X is of the form

$$\phi = Q_1 y_1 Q_2 y_2 \cdots Q_m y_m \theta(x_1, \ldots, x_n, y_1, \ldots, y_m),$$

where $x_1, \ldots, x_n$ are the free variables, each Q_i is a quantifier equal to either $\exists$ or $\forall$, and θ is a quantifier-free Presburger formula. We may furthermore assume that the formula θ does not contain occurrences of the negation operator, since negations can always be pushed inwards and then applied to atomic subformulas. In other words, θ is expressed as a Boolean combination of linear equalities and linear inequations.

2.4.1. Quantifier-free Presburger formulas. The algorithms presented in § 2.3.1 and § 2.3.2 can be employed for constructing deterministic NDD $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p$ that recognise the sets of vectors satisfying the atomic subformulas of θ, in a given base $k > 1$. From Theorem 1.1, an NDD recognising the characteristic set of θ, that is, the set $\{(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathbb{Z}^{n+m} \mid \theta(x_1, \ldots, x_n, y_1, \ldots, y_m)\}$ can be obtained by combining the automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p$ by means of a product operation [29]. In other words, one builds an automaton that simulates the concurrent operation of $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p$ on a common input word, with an accepting condition derived from the Boolean structure of θ. This construction yields a deterministic NDD.

2.4.2. Quantified Presburger formulas. It remains to show how to handle quantifiers. We have the following definition.

Definition 2.2. Let $Z \subseteq \mathbb{Z}^p$ be a set, with $p > 1$, and let $i \in [1, p]$ be a vector component. The *projection* of Z over the components different from i, denoted $Z|_{\neq i}$, is the set

$$\{(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_p) \mid \exists z_i \in \mathbb{Z}\colon (z_1, z_2, \ldots, z_p) \in Z\}.$$

It follows from Definition 2.2 that applying an existential quantifier amounts to projecting a set: for a Presburger formula $\varphi(x_1, \ldots, x_p)$ admitting the characteristic set $Y \subseteq \mathbb{Z}^p$, the characteristic set of $\exists x_i\, \varphi(x_1, \ldots, x_p)$ is equal to $Y|_{\neq i}$, for any $i \in [1, p]$.

Universal quantifiers can be handled by reducing them to existential ones: the formula $\forall x_i\, \varphi(x_1, \ldots, x_p)$ is equivalent to $\neg\exists x_i\, \neg\varphi(x_1, \ldots, x_p)$, hence its characteristic set can be computed by first complementing the characteristic set of φ, projecting it over the components different from i, and then again complementing the result.

2.4.3. Projection operation. In order to compute the finite-state representation of sets defined by quantified Presburger formulas, we thus need an algorithm for applying a projection operator to a set of integer vectors recognised by an NDD. We now focus on this operation.

Let $X \subseteq \mathbb{Z}^n$ be a set of vectors recognised by an NDD $\mathcal{A}$ in a base $k > 1$, and let $i \in [1, n]$ be a vector component. In order to obtain $X|_{\neq i}$, one needs to remove the i-th component from all the vectors belonging to X. A simple way of obtaining an NDD recognising $X|_{\neq i}$ is thus to remove the i-th component from each symbol labelling an edge of $\mathcal{A}$. In other word, the operation turns the automaton $\mathcal{A} = (Q, I, E, T)$ into (Q, I, E', T), where $E' = \{(q, \vec{d}|_{\neq i}, q') \mid (q, \vec{d}, q') \in E\}$. Note that this procedure generally produces a nondeterministic automaton.

Unfortunately, the resulting automaton is generally not a valid NDD. Indeed, even though it clearly only accepts encodings of the vectors in $X|_{\neq i}$, it may not accept all such encodings. Consider, for instance, the set $X = \{(0, -7, 3)\}$ introduced in Example 2.1. In the base $k = 2$, this set is encoded by the language $(0, 1, 0)^+(0, 0, 0)(0, 0, 1)\ (0, 1, 1)$. Let us now attempt to compute the encoding of $X|_{\neq 2}$. By removing the second component from each symbol, one obtains the language $(0, 0)^+(0, 0)(0, 1)(0, 1)$. This language is not equal to the language $(0, 0)^+(0, 1)(0, 1)$ that encodes $X|_{\neq 2}$, since the former does not contain the word $(0, 0)(0, 1)(0, 1)$, which is a valid encoding of $(0, 3)$.

In order to obtain a valid NDD recognising $X|_{\neq i}$, it is thus not enough to project the language accepted by an NDD recognising X. This operation has to be followed by a transformation aimed at ensuring that the language contains all the encodings of the vectors it represents. Recall that two encodings of the same vector only differ in the number of repetitions of their sign symbol. For each word belonging to the projected language, one can thus remove the copies of its sign symbol $\vec{d}$, and replace them by $\vec{d}^+$.

On a given automaton, this completion operation can be carried out as follows. One enumerates the possible sign symbols $\vec{d} \in \{0, k-1\}^{n-1}$. For each of them, the states that are reachable from an initial one by reading words from the language $\vec{d}^+$ are identified. Then, each of those states is made reachable after reading any word from $\vec{d}^+$, which amounts to creating a new initial state q_0 with a self-edge $(q_0, \vec{d}, q_0)$, and edges $(q_0, \vec{d}, q)$ linking this state to every identified state q.

The procedure for applying the projection operator to a set represented by an NDD, including the completion step, is summarised in Algorithm 4.

Algorithm 4 Algorithm for projecting a set represented by an NDD

1. Let (Q, I, E, T) be an NDD representing $X \subseteq \mathbb{Z}^n$ in a base $k > 1$, and let $i \in [1, n]$ be the vector component that is projected away.
2. Compute $E' = \{(q, \vec{d}|_{\neq i}, q') \mid (q, \vec{d}, q') \in E\}$, and sets $Q' := Q$ and $I' := I$.
3. For every $\vec{d} \in \{0, k-1\}^{n-1}$,
   a. create a new state q_I and add it to Q' and I';
   b. add an edge $(q_I, \vec{d}, q_I)$ to E';
   c. for every state q of (Q, I, E', T) reachable by reading a word in $\vec{d}^+$, add an edge $(q_I, \vec{d}, q)$ to E'.
4. Return the NDD (Q', I', E', T).

This algorithm has two drawbacks. First, it outputs a nondeterministic automaton. The production of nondeterminism is inherent to projection since this operation has a surjective behaviour, that is, it may map two distinct vectors into a single output value. As a consequence, the resulting automaton may admit multiple paths reading the same word. If this automaton needs to be complemented, for instance in order to apply a universal quantifier, or to be minimised into a canonical form, it will have to be determinised, which might potentially be a costly operation.

The second drawback is the exponential cost of the transformation ensuring that the resulting NDD accepts all the encodings of the vectors it recognises. It is shown in [16] that this cost can be mitigated if one works with serialised encodings of numbers. The idea is that the NDD undergoing the projection generally does not distinguish all the individual values of the sign header of encodings. More precisely, two sign headers u_1 and u_2 are considered to be equivalent if any state of the automaton that can be reached after reading u_1^i, for any $i > 0$, can also be reached by reading u_2^i. In such a case, those two sign headers do not need to be considered separately by the algorithm.

A detailed projection algorithm based on this principle and applicable to serialised NDD is presented in [16].

2.4.4. Deciding Presburger arithmetic. Number decision diagrams provide an elegant method for deciding Presburger formulas. Using the construction algorithms presented in §§ 2.3.1–2.3.3, as well as the Boolean operators provided by Theorem 1.1, one can build an NDD representing the set of solutions of any quantifier-free Presburger formula. Then, using the mechanisms discussed in §§ 2.4.1–2.4.3, one can apply quantifiers by computing projections and complement operations. Finally, the formula is decided by checking whether the resulting automaton accepts an empty language.

Since the complement operation is only applicable to deterministic automata, each alternation of quantifiers requires one determinisation step, which may incur an exponential blowup. Since a Presburger formula may include an arbitrarily large number of quantifier alternations, the size of the automaton constructed from a formula may potentially become nonelementary in the size of the formula. It has been shown in [30] that the automata that are constructed from Presburger formulas acquire a special structure that curbs this nonelementary blowup. The following result provides an upper bound on the size of NDD recognising Presburger-definable sets [30].

Theorem 2.5. *Let ϕ be a Presburger formula. The size of the minimal deterministic NDD recognising the set of solutions of ϕ is at most $2^{2^{2^{O(n)}}}$.*

The bound expressed by Theorem 2.5 is known to be tight [30]. Note that this bound only applies to the size of the automata, and not to the cost of their construction. During the construction of an NDD representing the set of solutions of a Presburger formula, each determinisation step may possibly incur an exponential blowup, hence the worst-case cost of deciding a Presburger formula by using NDD constructs is bounded by $O(2^{2^{2^{2^{O(n)}}}})$. It is known, however, that Presburger formulas can be decided in $O(2^{2^{2^{O(n)}}})$ time (see [39] and [26]), which shows that the NDD-based decision procedure for Presburger arithmetic is, from a theoretical point of view, less than optimal. In typical practical applications however, the triple exponential blowup in the size of the formulas is seldom experienced, and the cost of performing a determinisation and minimisation step after each projection operation is often acceptable. An experimental account of this property is given in [44].

2.4.5. Presburger-definable transformations. We now address the problem of applying a transformation $f : \mathbb{Z}^n \to \mathbb{Z}^n$ to a set $X \subseteq \mathbb{Z}^n$ recognised by an NDD in some base $k > 1$. This operation is essential to applications such as symbolic state-space exploration of programs, in which sets of integer vectors are used for representing sets of configurations of the programs under analysis, and the effect of executing a program instruction is described by a transformation.

If the transformation f can be expressed in Presburger arithmetic, which means that the set $\{(x_1, \ldots, x_n, y_1, \ldots, y_n\} \in \mathbb{Z}^{2n} \mid (y_1, \ldots, y_n) = f((x_1, \ldots, x_n))\}$ is Presburger-definable, then an NDD recognising $f(X)$ can easily be computed from one

recognising X. The first step is to build an NDD recognising the characteristic formula $\phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ of f. Each edge of this automaton is labelled by a vector of digits $(d_1, \ldots, d_n, d'_1, \ldots, d'_n) \in \{0, 1, \ldots, k-1\}^{2n}$, which can be split into two vectors $(d_1, \ldots, d_n)$ and $(d'_1, \ldots, d'_n)$ that respectively correspond to digits in the encodings of $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$. The automaton can then be seen as a transducer that turns encodings of $\vec{x}$ into encodings of $\vec{y}$. By combining the NDD recognising X with this transducer, one obtains an automaton that recognises encodings of the vectors in $f(X)$.

As in the case of the projection operation discussed in § 2.4.3, the resulting automaton might not accept all the encodings of the vectors it recognises, and thus has to be completed by the same procedure. Furthermore, this automaton is generally nondeterministic. This was expected, since projection is a particular case of Presburger-definable operation.

2.4.6. Other operations. Finally, it is worth mentioning that techniques have been developed for deciding whether an NDD represents a Presburger-definable set of vectors (see [34]) or a periodic one (see [27], [36], and [18]), as well as for extracting from an NDD a Presburger formula that defines the set it recognises (see [33] and [34]). Another operation of interest is the computation of the image of an NDD-represented set by the iterative closure of an affine transformation, discussed in [9] and [5].

3. Real domain

In this section, we apply finite-state representation systems to the domain $\mathbb{R}^n$, that is, we study representations suited for sets of real vectors with a fixed dimension $n > 0$.

3.1. Encoding relation. The positional encoding of integer vectors introduced in § 2.1 extends naturally to the real domain. The first step is to decompose a real number $x \in \mathbb{R}$ into an *integer part* $x_I \in \mathbb{Z}$ and a *fractional part* $x_F \in [0, 1]$, such that $x = x_I + x_F$. Note that, for reasons that shall become apparent soon, integers can be decomposed in two ways, for instance, $x = 3$ leads to both $x_I = 3, x_F = 0$ and $x_I = 2, x_F = 1$.

After choosing a numeration base $k > 1$, an integer part x_I is encoded using the k-ary positional encoding of integers described in § 2.1, negative values being handled by the base-complement method. One obtains a finite word $w_I \in \{0, k-1\}A_k^*$, where $A_k = \{0, 1, \ldots, k-1\}$, such that $x_I \in [-k^{p-1}, k^{p-1} - 1]$, with $p = |w_I|$.

In order to encode a fractional part x_F, one moves to infinite words, considering that the word $a_{-1}a_{-2}a_{-3}\cdots \in A_k^\omega$ encodes the value $x_F = \sum_{i<0} a_i k^i$. The encodings of a number $x \in \mathbb{R}$ then take the form of words $x_I \star x_F$, in which x_I and x_F are a matching pair of integer and fractional parts of x, and the special symbol $\star$ is used as a separator.

By this method, real numbers are encoded over the alphabet $A_k \cup \{\star\}$, and their valid encodings form the language $\{0, k-1\}A_k^* \star A_k^\omega$. Note that the fractional part of encodings, that is, the suffix following the separator $\star$, is able to encode all values in

$[0, 1]$, which is our motivation for allowing $x_F = 1$ in the definition of fractional parts of numbers.

As in the case of integer encodings, the first symbol of an encoding is its *sign digit*, and takes the value 0 for nonnegative integer parts and $k - 1$ for negative ones. The sign digit of an encoding can be repeated at will without affecting the encoded value. Every real number thus admits an infinite number of possible encodings in a given base $k > 1$. However, unlike integer encodings, some real numbers admit distinct encodings that do not only differ by the number of repetitions of their sign digit: numbers that can be expressed as a fraction y/k^m, where $y, m \in \mathbb{Z}$, can be encoded both by words ending in 0^ω, and by other ones ending in $(k-1)^\omega$, which we call *dual* encodings. In particular, all integers admit dual encodings.

Example 3.1. Let $x = -\frac{11}{4}$ and $k = 2$. The encodings of x form the language $1^+01 \star 010^\omega \cup 1^+01 \star 001^\omega$.

We now extend this encoding scheme to vectors $\vec{x} \in \mathbb{R}^n$, with $n > 0$, using the same principles as in § 2.1. One separately encodes each component of $\vec{x}$ and, in order for the separator symbol to be read simultaneously in all components, one chooses encodings that share the same integer-part length. This is always possible, for the sign digit of an encoding can be repeated at will. Then, one reads simultaneously and repeatedly one symbol in each component encoding, which yields a word w over the alphabet $(A_k)^n \cup \{\star\}$ that encodes $\vec{x}$. Note that, since the separator symbol is read at the same time in all components, it can be denoted by a unique symbol.

As in the case of integer encodings, the first symbol of an encoding is a *sign symbol*, and can be repeated without affecting the encoded vector. The language containing the valid encodings of vectors in base k is $\{0, k-1\}^n((A_k)^n)^* \star ((A_k)^n)^\omega$.

Example 3.2. Let $\vec{x} = \left(-\frac{11}{4}, \frac{2}{3}\right)$ and $k = 2$. The encodings of the components of $\vec{x}$ form the languages $E\left(-\frac{11}{4}\right) = 1^+01 \star 010^\omega \cup 1^+01 \star 001^\omega$ and $E\left(\frac{2}{3}\right) = 0^+ \star (10)^\omega$. The encodings of $\vec{x}$ are thus given by $E(\vec{x}) = (1,0)^+(0,0)(1,0) \star (0,1)(1,0)[(0,1)(0,0)]^\omega \cup (1,0)^+(0,0)(1,0) \star (0,1)(0,0)[(1,1)(1,0)]^\omega$.

If the exponential size of the alphabet with respect to the domain dimension is problematic, then encodings of real vectors can be serialised in the same way as integer vectors. A serialised encoding of a vector is simply obtained by replacing each tuple of digits $(a_1, a_2, \ldots, a_n)$ by the word $a_1 a_2 \cdots a_n$. The separator symbol $\star$ is left unchanged. The first n symbols of a serialised vector encoding correspond to its *sign header*, and can be freely repeated. The language of valid serialised encodings is given by $\{uv \star w \mid u \in \{0, k-1\}^*, v \in (A_k)^*, w \in (A_k)^\omega, |u| = n, |v| \equiv 0 \pmod n\}$.

Example 3.3. Let $\vec{x} = \left(-\frac{11}{4}, \frac{2}{3}\right)$ and $k = 2$. The serialised encodings of $\vec{x}$ form the language $E(\vec{x}) = (10)^+0010 \star 0110(0100)^\omega \cup (10)^+0010 \star 0100(1110)^\omega$.

We are now ready to define finite-state representations of sets of real vectors.

Definition 3.1. Let $n > 0$ be a dimension, and $k > 1$ be a numeration base. Let E denote the (either serialised or unserialised) encoding relation suited for $\mathbb{R}^n$ in base k. Let $X \subseteq \mathbb{R}^n$. An infinite-word automaton is a *real vector automaton* (RVA) representing X if its accepts the language $E(X)$.

Example 3.4. Let $n = 2$ and $k = 2$. The automaton illustrated in Figure 3 is an RVA with a Büchi acceptance condition, representing the set

$$\{(y, z) \in \mathbb{R}^2, y \geqslant 0, z \geqslant 0, y < z\}.$$

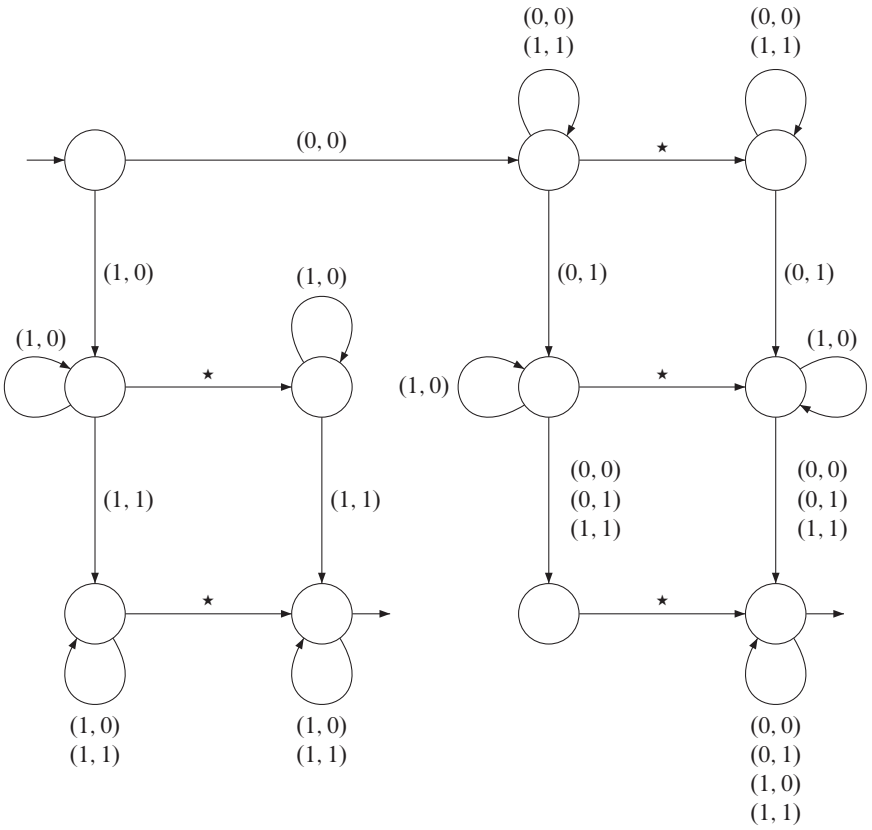

Figure 3. Example of RVA

Notice that Definition 3.1 does not impose a specific form of infinite-word automaton. In [10], RVA are defined as Büchi or, equivalently, Muller automata. This choice may be problematic for actual applications, since some operations such as complementation can become costly on such automata [32]. In the next section, we show that a restricted form of infinite-word automaton that is more easily handled algorithmically suffices for a large class of applications.

3.2. Expressive power. The expressive power of unrestricted RVA, that is, of RVA based on Büchi or Muller automata, in a base $k > 1$ has been established in [19]. This result is expressed in terms of a predicate $X_k : \mathbb{R} \times \mathbb{N} \times \{0, 1, \ldots, k-1\} \to \{\text{true}, \text{false}\}$ defined as follows.

Definition 3.2. Let $x \in \mathbb{R}$, $y \in \mathbb{N}$ and $d \in \{0, 1, \ldots, k-1\}$. We have $X_k(x, y, d)$ if and only if y is a power of k, and x admits an encoding $w = a_{p-1}a_{p-2}\cdots a_0 \star a_{-1}a_{-2}\cdots$ in which the digit a_i such that $y = k^i$ is equal to d.

Theorem 3.1. *Let $n > 0$ be a dimension and $k > 1$ be a numeration base. A set $X \subseteq \mathbb{R}^n$ is recognisable by an RVA in base k if and only if it can be defined in the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, <, X_k \rangle$.*

The following result characterises the expressiveness of RVA regardless of the chosen numeration base [12].

Theorem 3.2. *Let $n > 0$ be a dimension. A set $X \subseteq \mathbb{R}^n$ is recognisable by an RVA in every base $k > 1$ if and only if it can be defined in the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$.*

Theorem 3.2 generalises Cobham and Semenov's theorem to the real domain. The theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$ can be seen as an extension of Presburger arithmetic to mixed integer and real variables. This theory is quite expressive, since it covers linear constraints over both integer and real variables, as well as modular constraints over integers, and is closed under Boolean combinations, projection, Cartesian product of sets, ...

It is known that the full expressive power of infinite-word automata is not needed for representing sets that are definable in $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$, see [15]. We have the following definition and result.

Definition 3.3. A *weak automaton* is a Büchi automaton such that every strongly connected component of its transition graph contains either only accepting or only nonaccepting states.

Theorem 3.3. *Let $n > 0$ be a dimension and $k > 1$ be a numeration base. Every set $X \subseteq \mathbb{R}^n$ that is definable in the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$ can be recognised by a weak deterministic RVA in base k.*

The advantage of working with weak deterministic automata is that these automata are much more easily manipulated algorithmically than Büchi or Muller ones. In particular, complementing a weak deterministic automaton simply amounts to inverting the accepting status of each strongly connected component in its transition graph, provided that this graph is complete. These automata can also be efficiently minimised into a canonical form [35].

3.3. Construction of elementary sets

3.3.1. Linear equalities. Let $n > 0$ be a dimension, and $X = \{\vec{x} \in \mathbb{R}^n \mid \vec{a}.\vec{x} = b\}$ be a linear constraint over real vectors, with $\vec{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$. We now adapt the algorithm introduced in § 2.3.1 for constructing a weak deterministic RVA that recognises X in a base $k > 1$.

Recall that an encoding of a vector $\vec{x} \in \mathbb{R}^n$ takes the form $w_I \star w_F$, where $w_I \in ((A_k)^n)^*$ and $w_F \in ((A_k)^n)^\omega$ respectively encode an integer part $\vec{x}_I \in \mathbb{Z}^n$ and a fractional part $\vec{x}_F \in [0,1)^n$ of $\vec{x}$, such that $\vec{x} = \vec{x}_I + \vec{x}_F$. Thus, an RVA that recognises $\vec{x}$ first reads its integer part, then a separator, and then its fractional part.

Consider a deterministic RVA that recognises the set X. Each $\vec{x} \in X$ satisfies $\vec{a}.\vec{x} = b$, hence after decomposing $\vec{x}$ into an integer part $\vec{x}_I$ and a fractional part $\vec{x}_F$, we obtain $\vec{a}.\vec{x}_I + \vec{a}.\vec{x}_F = b$. Notice that we have $\vec{a}.\vec{x}_I \in \mathbb{Z}$, which implies $\vec{a}.\vec{x}_F \in \mathbb{Z}$. Furthermore, if $g = \gcd(a_1, a_2, \ldots, a_n)$ is such that $g > 1$, then we have $\vec{a}.\vec{x}_I \equiv 0 \pmod g$, hence $\vec{a}.\vec{x}_F \equiv b \pmod g$. Since $\vec{x}_F \in [0,1)^n$, the possible values of $\vec{a}.\vec{x}_F$ are, moreover, restricted to belong to the interval $[a_-, a_+]$, where $a_- = \sum_{a_i<0} a_i$ and $a_+ = \sum_{a_i>0} a_i$.

Let

$$Z = \begin{cases} \{z \in \mathbb{Z} \mid a_- \leqslant z \leqslant a_+\} & \text{if } g = 1, \\ \{z \in \mathbb{Z} \mid a_- \leqslant z \leqslant a_+, z \equiv b \pmod g\} & \text{if } g > 1. \end{cases}$$

For each $\vec{x} \in X$ and decomposition $\vec{x}_I + \vec{x}_F$ of $\vec{x}$ into an integer and a fractional part, we have $\vec{a}.\vec{x}_F = z$ for some $z \in Z$, and $\vec{a}.\vec{x}_I = b - z$.

This property makes it possible to construct a weak deterministic RVA recognising X by separately constructing the parts of the automaton that deal with the integer and the fractional part of encodings. This can be done by enumerating the elements of the set Z. For each $z \in Z$, one constructs an NDD recognising the set $\{\vec{x}_I \in Z \mid \vec{a}.\vec{x}_I = b - z\}$. From the accepting state(s) of this NDD, one creates edges labelled by $\star$ and leading to a new state. From this state, one builds an automaton accepting the fractional parts w_F of encodings, such that the corresponding vector $\vec{x}_F$ satisfies $\vec{a}.\vec{x}_F = z$.

In order to construct the part of the automaton that deals with the integer part of encodings, the first idea is to apply the algorithm proposed in § 2.3.1 for each value of z. This procedure can be improved by constructing a single automaton in which the states that have identical labels in different runs of the algorithm are merged. In other words, this amounts to starting the construction described in Algorithm 1 from a set of states labelled by the elements of Z, instead of from a single state. The application of the backward propagation rule and the construction of the initial states are unchanged. This procedure produces a single deterministic automaton, with a unique output state q_z associated to each value $z \in Z$, such that the paths leading from the initial state to q_z recognise the vectors $\vec{x}_I \in \mathbb{Z}$ such that $\vec{a}.\vec{x}_I = b - z$.

We now address the construction of the part of the automaton that deals with the fractional part of encodings. Let $z \in Z$. The goal is to construct a weak deterministic automaton accepting the words $w_F \in ((A_k)^n)^\omega$ such that $0 \star w_F$ encodes a vector $\vec{x}_F \in [0,1)^n$ satisfying $\vec{a}.\vec{x}_F = z$. As in § 2.3.1, we proceed by labelling the states of the constructed automaton. Labelling the state q with $\gamma(q)$ means that the paths originating from q recognise the solutions $\vec{x}_F$ of $\vec{a}.\vec{x}_F = \gamma(q)$. Hence, the construction starts by labelling the initial state q_0 of the automaton with $\gamma(q_0) = z$.

Given a state q with a known label $\gamma(q)$, the labels of its successors can be computed as follows. Consider a symbol $\vec{d} \in (A_k)^n$ and an edge $(q, \vec{d}, q')$ to some

state q'. By definition of the labelling, every path leaving q' recognises a solution $\vec{x}'_F$ of $\vec{a}.\vec{x}'_F = \gamma(q')$. Every such path can be turned into a path leaving q by prefixing it with the edge $(q, \vec{d}, q')$, which results in a path reading $\vec{x}_F = \left(\frac{1}{k}\right)(\vec{x}'_F + \vec{d})$. This yields $\vec{a}.\vec{x}'_F + \vec{a}.\vec{d} = k\vec{a}.\vec{x}_F$, and hence

$$\gamma(q') = k\gamma(q) - \vec{a}.\vec{d}. \tag{2}$$

The forward propagation rule (2) can be used for recursively generating the state labels starting from the initial state, but an additional constraint has to be imposed in order to force termination. In the fractional part of the automaton, each path recognises a vector that belongs to $[0, 1]^n$; hence each state q must have a label that satisfies $\gamma(q) \in [a_-, a_+]$. If the application of (2) yields a label that violates this constraint, then the corresponding state can be safely discarded.

This procedure produces a deterministic automaton in which each infinite path is accepting. All its states can thus be marked as being accepting, which results in a weak automaton. Finally, note that, similarly to the case of integer parts, the automata corresponding to the different values of z do not have to be built separately, and can share states with identical labels.

The algorithm for constructing a weak deterministic RVA recognising the set

$$\{\vec{x} \in \mathbb{R}^n \mid \vec{a}.\vec{x} = b\}$$

is summarised in Algorithm 5.

The size of the RVA constructed by this algorithm can be estimated by the same technique as in § 2.3.1. For the integer part of the automaton, we obtain that the only labels that are considered belong to the union of intervals

$$\begin{aligned}[b - a_+, b - a_-] \cup \bigcup_{i=1}^{\ell} \left[\frac{b - a_+}{k^i} - (k-1)a_+, \frac{b - a_-}{k^i} - (k-1)a_-\right] \\ \subseteq \bigcup_{i=0}^{\ell} \left[\frac{b}{k^i} - ka_+, \frac{b}{k^i} - ka_-\right],\end{aligned}$$

where $\ell = \lfloor \log_k \max(|b - a_+|, |b - a_-|, 1) \rfloor + 1$.

In the fractional part of the automaton, the labels of the states necessarily belong to the interval $[a_-, a_+]$.

In summary, the RVA produced by Algorithm 5 contains at most

$$(\ell + 1)((k(a_+ - a_-) + 1))$$

states in its integer part and at most $a_+ - a_- + 1$ states in its fractional part, in addition to a unique initial state. We thus have the following result.

Theorem 3.4. *Let $n > 0$ be a dimension, $k > 1$ be a numeration base, and $X = \{\vec{x} \in \mathbb{R}^n \mid \vec{a}.\vec{x} = b\}$ be a linear constraint, with $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$. There exists a weak deterministic RVA recognising X in base k with $O\big(k(\log |b|) \sum_{i=1}^{n} |a_i|\big)$ states.*

Algorithm 5 Algorithm for building a base-k RVA representing $\vec{a}.\vec{x} = b$

1. Compute the set $Z = \{z \in \mathbb{Z} \mid a_- \leqslant z \leqslant a_+\}$, where $a_- = \sum_{a_i<0} a_i$ and $a_+ = \sum_{a_i>0} a_i$.
2. Set $g = \gcd(a_1, a_2, \ldots, a_n)$. If $g > 1$, replace Z by $Z \cap \{z \in \mathbb{Z} \mid z \equiv b \pmod g\}$.
3. Create a table Q_I of states and a list L of labels of "active" states, both initialised to $\{b - z \mid z \in Z\}$.
4. Set $I = E = \emptyset$.
5. While $L \neq \emptyset$, remove a value v from L, and for every $\vec{d} \in \{0, 1, \ldots, k-1\}^n$,
   - if $v' = \frac{v - \vec{a}.\vec{d}}{k}$ is an integer multiple of g, then
     a. if $v' \notin Q_I$, then add v' to Q_I and L;
     b. add an edge $(v', \vec{d}, v)$ to E.
6. Add a new initial state q_0 to Q_I and I, and, for every $\vec{d} \in \{0, k-1\}^n$,
   - if $v = \frac{\vec{a}.\vec{d}}{1-k}$ is such that $v \in Q_I$, then add an edge $(q_0, \vec{d}, v)$ to E.
7. Create a table Q_F of states, considered distinct from those in Q_I, and initialised to Z.
8. For each $z \in Z$, add an edge $(q_I, \star, q_F)$, where $q_I \in Q_I$, $q_F \in Q_F$, and $q_I + q_F = b$.
9. Set $L = T = Z$.
10. While $L \neq \emptyset$, remove a value v from L, and, for every $\vec{d} \in \{0, 1, \ldots, k-1\}^n$,
    - if $v' = kv - \vec{a}.\vec{d}$ is such that $a_- \leqslant v \leqslant a_+$, then
      a. if $v' \notin Q_F$, then add v' to Q_F, L, and T;
      b. add an edge $(v, \vec{d}, v')$ to E.
11. Compute the union Q of Q_I and Q_F, considering that states that have the same label in both sets are distinct.
12. Return (Q, I, E, T).

Example 3.5. An RVA representing the set $\{(x_1, x_2) \in \mathbb{R} \mid 2x_1 - x_2 = -4\}$ in base 2 is illustrated in Figure 4. The state labels used in the integer and fractional parts of this automaton correspond to those used by Algorithm 5.

As in § 2.3.1, Algorithm 5 can be adapted for generating automata reading serialised encodings of vectors.

3.3.2. Linear inequations. The algorithm developed in § 3.3.1 can easily be adapted for constructing an RVA that recognises the set of solutions $X = \{\vec{x} \in \mathbb{R}^n \mid \vec{a}.\vec{x} \leqslant b\}$ of a linear inequality, with $\vec{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$.

The decomposition of a vector $\vec{x} \in \mathbb{R}^n$ into an integer part $\vec{x}_I \in \mathbb{Z}^n$ and a fractional part $\vec{x}_F \in [0, 1]^n$, such that $\vec{x} = \vec{x}_I + \vec{x}_F$ is unchanged. During the construction of the integer part of the automaton, one simply uses the technique proposed in § 2.3.2: when the propagation rule (1) yields a label $\beta(q')$ that is not an integer multiple of $g = \gcd(a_1, \ldots, a_n)$ for some state q', this value is rounded down to $g\lfloor \beta(q')/g \rfloor$ instead of being discarded.

A similar reasoning can be applied to the fractional part of the automaton. Consider a state q for which the propagation rule (2) gives a value $\gamma(q')$ for some successor q' of q such that $\gamma(q') \notin [a_-, a_+]$, with $a_- = \sum_{a_i<0} a_i$ and $a_+ = \sum_{a_i>0} a_i$. If $\gamma(q') > a_+$, then the constraint $\vec{a}.\vec{x}_F \leqslant \gamma(q')$ is satisfied by all $\vec{x}_F \in [0, 1]^n$, and hence the value

of $\gamma(q')$ can be replaced by a_+. On the other hand, if $\gamma(q') < a_-$, then the constraint $\vec{a}.\vec{x}_F \leqslant \gamma(q')$ is unsatisfiable in $[0,1]^n$, and the state q' does not have to be created.

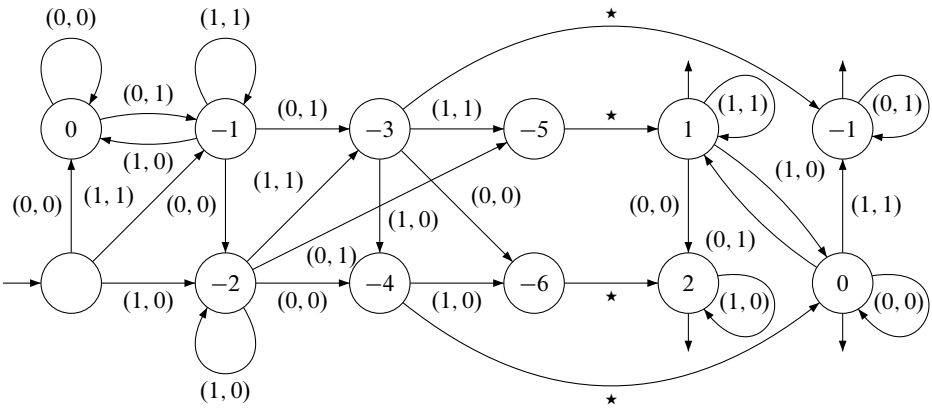

Figure 4. RVA recognising $2x_1 - x_2 = -4$

An algorithm formalising this construction is given in Algorithm 6. The size of the generated RVA is similar to that produced by Algorithm 5; however this automaton is generally nondeterministic. Since the nondeterminism is only located in the integer part of the automaton, Theorem 2.4 can be applied, and a deterministic RVA can be obtained in linear time. The constructions proposed in [30] can also be used for directly generating a deterministic automaton dealing with the integer part of encodings.

Finally, notice that constraints of the form $\{\vec{x} \in \mathbb{R}^n \mid \vec{a}.\vec{x} \,\#\, b\}$, with $k > 1$, $\vec{a} \in \mathbb{Z}^n$, $b \in \mathbb{Z}$, and $\# \in \{<, >, \geqslant\}$, can be expressed as Boolean combinations of the constraints handled by Algorithm 5 and Algorithm 6. We indeed have

$$\begin{aligned}
\vec{a}.\vec{x} < b &\iff \vec{a}.\vec{x} \leqslant b \wedge \neg(\vec{a}.\vec{x} = b),\\
\vec{a}.\vec{x} > b &\iff -\vec{a}.\vec{x} \leqslant -b \wedge \neg(\vec{a}.\vec{x} = b),\\
\vec{a}.\vec{x} \geqslant b &\iff -\vec{a}.\vec{x} \leqslant -b.
\end{aligned}$$

3.4. Operations on sets. RVA lead to an elegant decision procedure for the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$. By a similar approach as in § 2.4.1 and § 2.4.2, one decides a formula by building RVA corresponding to its atomic subformulas, combining them by means of Boolean operators, and applying quantifiers by performing projection and complementation operations. RVA can be projected in the same way as NDD, by removing the vector component that is projected away from each edge label, and completing the resulting automaton so as to make it accept all the encodings of the vectors it recognises. This procedure can be carried out by a slight adaptation of Algorithm 4, in which step 2 skips the edges labelled by the separator symbol $\star$.

Algorithm 6 Algorithm for building a base-k RVA representing $\vec{a}.\vec{x} \leqslant b$

1. Compute the set $Z = \{z \in \mathbb{Z} \mid a_- \leqslant z \leqslant a_+\}$, where $a_- = \sum_{a_i<0} a_i$ and $a_+ = \sum_{a_i>0} a_i$.
2. Set $g = \gcd(a_1, a_2, \ldots, a_n)$. If $g > 1$, replace Z by $Z \cap \{z \in \mathbb{Z} \mid z \equiv b \pmod g\}$.
3. Create a table Q_I of states and a list L of labels of "active" states, both initialised to $\{b - z \mid z \in Z\}$.
4. Set $I = E = \emptyset$.
5. While $L \neq \emptyset$, remove a value v from L, and for every $\vec{d} \in \{0, 1, \ldots, k-1\}^n$,
   a. compute $v' = \frac{v - \vec{a}.\vec{d}}{k}$ and $v'' = g\lfloor \frac{v'}{g} \rfloor$;
   b. if $v'' \notin Q_I$, then add v'' to Q_I and L;
   c. add an edge $(v'', \vec{d}, v)$ to E.
6. Add a new initial state q_0 to Q_I and I, and for every $\vec{d} \in \{0, k-1\}^n$,
   - if $v = \frac{\vec{a}.\vec{d}}{1-k}$ is such that $v \in Q_I$, then add an edge $(q_0, \vec{d}, v)$ to E.
7. Create a table Q_F of states, considered distinct from those in Q_I, and initialised to Z.
8. For each $z \in Z$, add an edge $(q_I, \star, q_F)$, where $q_I \in Q_I$, $q_F \in Q_F$, and $q_I + q_F = b$.
9. Set $L = T = Z$.
10. While $L \neq \emptyset$, remove a value v from L, and for every $\vec{d} \in \{0, 1, \ldots, k-1\}^n$,
    a. compute $v' = kv - \vec{a}.\vec{d}$;
    b. if $v' > a_+$, then set $v' = a_+$;
    c. if $v' \geqslant a_-$, then
       i. if $v' \notin Q_F$, then add v' to Q_F, L, and T;
       ii. add an edge $(v, \vec{d}, v')$ to E.
11. Compute the union Q of Q_I and Q_F, considering that states that have the same label in both sets are distinct.
12. Return (Q, I, E, T).

However, compared to the case of integer vectors, working in the real domain presents an additional difficulty. If one decides to use weak deterministic automata in order to benefit from an efficient complementation algorithm, then applying a projection operator becomes problematic, because it produces a nondeterministic automaton, and the class of weak automata is not closed under determinisation.

A solution to this problem is provided in [15]. In Definition 3.3, we have introduced weak automata as a specific form of Büchi automata. Interestingly, those automata can also be seen as a particular case of co-Büchi automata, that is, infinite-word automata in which a path is accepting if and only if it does not visit infinitely many accepting states. Actually, a weak automaton can be turned into a co-Büchi one by simply inverting the accepting status of each of its states.

The advantage of working with co-Büchi automata is that those can easily be determinised, using a variant of the subset construction developed for finite-word automata, see [37] and [31]. Thus, from a nondeterministic weak automaton, one can construct a deterministic co-Büchi automaton that accepts the same language.

In general, this deterministic co-Büchi automaton cannot be turned into a weak deterministic one. It is however shown in [15] that, if the considered automaton is an

RVA that recognises a set definable in $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$, then it necessarily has a special structure that makes this conversion feasible. This special structure is formalised by the following definition.

Definition 3.4. A Büchi or co-Büchi automaton is *inherently weak* if no strongly connected component in its transition graph contains both an accepting cycle (that is, a cycle visiting at least one accepting state), and a nonaccepting one.

Clearly, an automaton that is inherently weak can easily be turned into a weak one, by making all the states accepting in strongly connected components that contain at least one accepting cycle. This operation preserves determinism. The following result is established in [15].

Theorem 3.5. *Let $n > 0$ be a dimension. Every deterministic RVA recognising a set $X \subseteq \mathbb{R}^n$ that is definable in $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$ is inherently weak.*

If a projection operator is applied to a set X definable in $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$, then the result belongs to the same theory. From Theorem 3.5, it follows that a weak deterministic automaton recognising the projected set can be constructed from an RVA recognising X.

In summary, in order to decide a formula expressed in the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$, one builds a weak deterministic RVA recognising its set of solutions, and then checks whether this RVA accepts a nonempty language. Note that weak deterministic RVA can be minimised into a canonical form [35]; performing a minimisation operation after each manipulation step actually helps to keep the size of the constructed RVA under control.

4. Conclusions and perspectives

Finite-state automata provide elegant and powerful data structures for representing sets of values symbolically. The main advantages of finite-state representations are that they are naturally closed under Boolean operators, and that they admit an easily computable canonical form, which can make the representation of a set independent of its construction. In applications such as symbolic state-space exploration of programs, in which sets of reachable configuration frequently have a simple structure but are computed as the result of lengthy sequences of operations, this property is essential.

In the particular cases of the integer and real domains, using a positional encoding leads to an expressive power that is well matched to applications relying on the manipulation of linear constraints and discrete periodicities, see [43], [7], [14], [4], [6], and [38]. Finite-state representations are not a panacea; however, since the size of NDD and RVA recognising linear constraints can grow linearly with the magnitude of their coefficients, the representations can become unnecessarily large. A possible solution to this problem, consisting of representing some internal structures of automata by algebraic means, is being investigated, see [11] and [17]. Another problem is that the presence of dual encodings is also a source of inefficiency. For instance, the minimal weak deterministic RVA recognising $\{(0, 0, \ldots, 0) \in \mathbb{R}^n\}$ has $O(2^n)$ states, since 0

admits dual encodings. A method that successfully tackles this problem is described in [25].

Finally, it is worth mentioning that finite-state representation systems have also been applied to other domains, such as the representation of sets of stack or FIFO queue contents (see [20], [13], [21], and [1]), or of configurations of parameterised programs [2]. The *regular model checking* approach to program verification relies on finite-state representations for representing the sets of configurations that need to be handled [3].

Acknowledgement. The work of this chapter has been partially supported by the *Interuniversity Attraction Poles* program *MoVES* of the Belgian Federal Science Policy Office, and by the grant 2.4530.02 of the Belgian Fund for Scientific Research (F.R.S.-FNRS).

References

[1] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson, Using forward reachability analysis for verification of lossy channel systems. *Form. Methods Syst. Des.* 25 (2004), no. 1, 39–65. Zbl 1073.68675 q.v. 1212

[2] P. A. Abdulla and B. Jonsson, Model checking of systems with many identical timed processes. *Theoret. Comput. Sci.* 290 (2003), no. 1, 241–264. MR 1935692 Zbl 1018.68046 q.v. 1212

[3] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena, A survey of regular model checking. In *CONCUR 2004–concurrency theory* (P. Gardner and N. Yoshida, eds.). Proceedings of the 15th International Conference, held in London, UK, August 31–September 3, 2004. Lecture Notes in Computer Science, 3170. Springer, Berlin, 2004, 35–48. Zbl 1099.68055 q.v. 1212

[4] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci, FAST: Fast acceleration of symbolic transition systems. In *Proceedings of the 15th International Conference on Computer Aided Verification* (W. A. H. Jr. and F. Somenzi, eds.). Lecture Notes in Computer Science, 2725. Springer, Berlin, 2003, 118–121. q.v. 1211

[5] S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen, Flat acceleration in symbolic model checking. In *Automated technology for verification and analysis* (D. A. Peled and Y. Tsay, eds.). Proceedings of the Third International Symposium, ATVA 2005, held in Taipei, Taiwan, October 4–7, 2005. Lecture Notes in Computer Science, 3707. Springer, Berlin, 2005, 474–488. Zbl 1170.68507 q.v. 1202

[6] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke, LIRA: Handling constraints of linear arithmetics over the integers and the reals. In *Proceedings of the 19th International Conference on Computer Aided Verification* (W. Damm and H. Hermanns, eds.) Lecture Notes in Computer Science, 4590. Springer, Berlin, 2007, 307–310. q.v. 1211

[7] M. Biehl, N. Klarlund, and T. Rauhe, Mona: Decidable arithmetic in practice. In *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems* (B. Jonsson and J. Parrow, eds.). Lecture Notes in Computer Science, 1135. Springer, Berlin, 1996, 459–462. q.v. 1211

[8] B. Boigelot, *Symbolic methods for exploring infinite state spaces.* Ph.D. thesis. Université de Liège, Liège, 1998. q.v. 1189

[9] B. Boigelot, On iterating linear transformations over recognizable sets of integers. *Theoret. Comput. Sci.* 309 (2003), no. 1–3, 413–468. MR 2016536 Zbl 1070.68062 q.v. 1202

[10] B. Boigelot, L. Bronne, and S. Rassart, An improved reachability analysis method for strongly linear hybrid systems. In *Proceedings of the 9th International Conference on Computer Aided Verification* (O. Grumberg, ed.). Lecture Notes in Computer Science, 1254. Springer, Berlin, 1997, 167–177. q.v. 1204

[11] B. Boigelot, J. Brusten, and J.-F. Degbomont, Implicit real vector automata. In *Proceedings of the 12th International Workshop on Verification of Infinite-State Systems.* (Y. Chen and A. Rezine, eds.). INFINITY 2010. *Electron. Proc. Theor. Comput. Sci.* 39 (2010), 63–76. q.v. 1211

[12] B. Boigelot, J. Brusten, and J. Leroux, A generalization of Semenov's theorem to automata over real numbers. In *Automated deduction—CADE-22* (R. A. Schmidt, ed.). Proceedings of the 22nd International Conference held at McGill University, Montreal, QC, August 2–7, 2009. Lecture Notes in Computer Science, 5663. Lecture Notes in Artificial Intelligence. Springer, Berlin, 2009, 469–484. MR 2550354 Zbl 1250.03061 q.v. 1205

[13] B. Boigelot and P. Godefroid, Symbolic verification of communication protocols with infinite state spaces using QDDs. *Form. Methods Syst. Des.* 14 (1999), no. 3, 237–255. q.v. 1212

[14] B. Boigelot and F. Herbreteau, The power of hybrid acceleration. In *Computer aided verification* (T. Ball and R. B. Jones, eds.). Proceedings of the 18th International Conference, CAV 2006, held in Seattle, WA, August 17–20, 2006. Lecture Notes in Computer Science, 4144. Springer, Berlin, 2006, 438–451. q.v. 1211

[15] B. Boigelot, S. Jodogne, and P. Wolper, An effective decision procedure for linear arithmetic over the integers and reals. *ACM Trans. Comput. Log.* 6 (2005), no. 3, 614–633. MR 2147298 Zbl 1407.03052 q.v. 1205, 1210, 1211

[16] B. Boigelot and L. Latour, Counting the solutions of Presburger equations without enumerating them. *Theoret. Comput. Sci.* 313 (2004), no. 1, 17–29. MR 2055954 Zbl 1069.68063 q.v. 1200, 1201

[17] B. Boigelot and I. Mainz, Efficient symbolic representation of convex polyhedra in high-dimensional spaces. In *Automated Technology for Verification and Analysis* (Sh. K. Lahiri and C. Wang, eds.). Proceedings of the 16th International Symposium, ATVA 2018, held in Los Angeles, CA, October 7-10, 2018. Lecture Notes in Computer Science, 11138. Springer, Berlin, 2018, 284–299. q.v. 1211

[18] B. Boigelot, I. Mainz, V. Marsault, and M. Rigo, An efficient algorithm to decide periodicity of b-recognisable sets using MSDF convention. In *44th International Colloquium on Automata, Languages, and Programming* (I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, eds.). Proceedings of the colloquium (ICALP 2017) held in Warsaw, July 10–14, 2017. LIPIcs. Leibniz International Proceedings in Informatics, 80. Schloss Dagstuhl. Leibniz-Zentrum für Informatik, Wadern, 2017, Art. no. 118, 14 pp. MR 3685858 Zbl 07089069 q.v. 1202

[19] B. Boigelot, S. Rassart, and P. Wolper, On the expressiveness of real and integer arithmetic automata. In *Automata, languages and programming* (K. G. Larsen, S. Skyum, and G. Winskel, eds.). Proceedings of the 25th International Colloquium, ICALP '98, held in Aalborg, Denmark, July 13–17, 1998. Lecture Notes in Computer Science, 1443. Springer, Berlin, 1998, 152–163. Zbl 0910.68149 q.v. 1195, 1205

[20] A. Bouajjani, J. Esparza, A. Finkel, O. Maler, P. Rossmanith, B. Willems, and P. Wolper, An efficient automata approach to some problems on context-free grammars. *Inform. Process. Lett.* 74 (2000), no. 5–6, 221–227. MR 1766207 Zbl 1137.68418 q.v. 1212

[21] A. Bouajjani and P. Habermehl, Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoret. Comput. Sci.* 221 (1999), no. 1–2, 211–250. MR 1700826 Zbl 0933.68089 q.v. 1212

[22] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire, Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc. Simon Stevin* 1 (1994), no. 2, 191–238. Journées Montoises (Mons, 1992). Corrigendum, ibid. 1 (1994), no. 4, 577. MR 1318968 MR 1315840 (corrigendum) Zbl 0804.11024 Zbl 0812.11019 (corrigendum) q.v. 1193

[23] J. R. Büchi, On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science* (E. Nagel, P. Suppes, and A. Tarski, eds.). Proceedings of the 1960 International Congress. Stanford University Press, Stanford, CA, 1962, 1–11. MR 0183636 Zbl 0147.25103 q.v. 1193

[24] A. Cobham, On the base-dependence of sets of numbers recognizable by finite automata. *Math. Systems Theory* 3 (1969), 186–192. MR 0250789 Zbl 0179.02501 q.v. 1193

[25] J. Eisinger and F. Klaedtke, Don't care words with an application to the automata-based approach for real addition. *Form. Methods Syst. Des.* 33 (2008), no. 1–3, 85–115. q.v. 1212

[26] J. Ferrante and C. Rackoff, A decision procedure for the first order theory of real addition with order. *SIAM J. Comput.* 4 (1975), 69–76. MR 0389572 Zbl 0294.02022 q.v. 1201

[27] J. Honkala, A decision method for the recognizability of sets defined by number systems. *RAIRO Inform. Théor. Appl.* 20 (1986), no. 4, 395–403. MR 0880843 Zbl 0639.68074 q.v. 1202

[28] J. E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of machines and computations* (Z. Kohavi and A. Paz, eds.). Proceedings of an International Symposium on the Theory of Machines and Computations held at Technion in Haifa, Israel, on August 16–19, 1971. Academic Press, New York and London, 1971, 189–196. MR 0403320 Zbl 0293.94022 q.v. 1190

[29] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation.* Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Co., Reading, MA, 1979. MR 0645539 Zbl 0426.68001 q.v. 1190, 1197, 1199

[30] F. Klaedtke, Bounds on the automata size for Presburger arithmetic. *ACM Trans. Comput. Log.* 9 (2008), no. 2, Art. 11, 34 pp. MR 2398573 Zbl 1407.03053 q.v. 1197, 1201, 1209

[31] O. Kupferman and M. Y. Vardi, Weak alternating automata are not that weak. *ACM Trans. Comput. Log.* 2 (2001), no. 3, 408–429. MR 1859532 Zbl 1171.68551 q.v. 1210

[32] O. Kupferman and M. Y. Vardi, Complementation constructions for nondeterministic automata on infinite words. In *Tools and algorithms for the construction and analysis of systems* (K. Jensen and A. Podelski, eds.). Proceedings of the 11th international conference, TACAS 2005. Held as part of the joint European conference on theory and practice of software, ETAPS 2005, Edinburgh, UK, April 4–8, 2005. Lecture Notes in Computer Science, 3440. Springer, Berlin, 2004, 206–221. Zbl 1087.68050 q.v. 1190, 1204

[33] L. Latour, From automata to formulas: convex integer polyhedra. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.* Held in Turku, Finland, July 17, 2004. IEEE Press, Los Alamitos, CA, 2004, 120–129. IEEEXplore 1319606 q.v. 1202

[34] J. Leroux, A polynomial time Presburger criterion and synthesis for number decision diagrams. In *20th Annual IEEE Symposium on Logic in Computer Science.* LICS '05.

Held in Chicago, IL, June 26–29, 2005. IEEE Press, Los Alamitos, CA, 2005, 147–156. IEEEXplore 1509219 q.v. 1202

[35] C. Löding, Efficient minimization of deterministic weak $\omega-$automata. *Inform. Process. Lett.* 79 (2001), no. 3, 105–109. MR 1836123 Zbl 1032.68103 q.v. 1190, 1205, 1211

[36] V. Marsault and J. Sakarovitch, Ultimate periodicity of b-recognisable sets: a quasilinear procedure. In *Developments in language theory* (M. Béal and O. Carton, eds.). Proceedings of the 17th International Conference (DLT 2013) held at Université Paris-Est, Marne-la-Vallée, June 18–21, 2013. Lecture Notes in Computer Science, 7907. Springer, Berlin, 2013, 362–373. MR 3097342 Zbl 1381.68128 q.v. 1202

[37] S. Miyano and T. Hayashi, Alternating finite automata on ω-words. *Theoret. Comput. Sci.* 32 (1984), no. 3, 321–330. Zbl 0761350 MR 0544.68042 q.v. 1210

[38] H. Mousavi, Automatic theorem proving in Walnut. Preprint, 2016. arXiv:1603.06017 [cs.FL] q.v. 1211

[39] D. C. Oppen, A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. System Sci.* 16 (1978), no. 3, 323–332. MR 0478750 Zbl 0381.03021 q.v. 1201

[40] M. Presburger, Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *C. R. Congrès Math. Pays slaves* 1930, 92–101, supplement ibid., 395. JFM 56.0825.04 q.v. 1193

[41] M. Presburger and D. Jacquette, On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *Hist. Philos. Logic* 12 (1991), no. 2, 225–233. Translated from the German and with commentaries by D. Jacquette. MR 1111343 Zbl 0741.03027 q.v. 1193

[42] A. L. Semenov, Presburgerness of predicates regular in two number systems. *Siberian J. Math.* 18 (1977), 289–299. Zbl 0411.03054 q.v. 1193

[43] P. Wolper and B. Boigelot, An automata-theoretic approach to Presburger arithmetic constraints. In *Proceedings of the 2nd International Static Analysis Symposium* (A. Mycroft, ed.). Lecture Notes in Computer Science, 983. Springer, Berlin, 1995, 21–32. q.v. 1211

[44] P. Wolper and B. Boigelot, On the construction of automata from linear arithmetic constraints. In *Tools and algorithms for the construction and analysis of systems* (S. Graf and M. I. Schwartzbach, eds.). Proceedings of the 6th International Conference, TACAS 2000. Held as part of the joint European conferences on theory and practice of software, ETAPS 2000, Berlin, Germany, March 25–April 2, 2000. Lecture Notes in Computer Science, 1785. Springer, Berlin, 2000, 1–19. Zbl 0964.68082 q.v. 1197, 1201