Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Domain-specific regular acceleration*

Bernard Boigelot

Institut Montefiore, B28
Université de Liège
B-4000 Liège Sart-Tilman
Belgium
e-mail: boigelot@montefiore.ulg.ac.be

The date of receipt and acceptance will be inserted by the editor

Abstract. The regular model-checking approach is a set of techniques aimed at exploring symbolically infinite state spaces. These techniques proceed by representing sets of configurations of the system under analysis by regular languages, and the transition relation between these configurations by a transformation over such languages. The set of reachable configurations can then be computed by repeatedly applying the transition relation, starting from a representation of the initial set of configurations, until a fixed point is reached.

In order for this computation to terminate, it is generally needed to introduce so-called acceleration operators, the purpose of which is to explore in one computation step infinitely many paths in the transition graph of the system. A simple form of acceleration operator is one that is associated to a cycle in the transition graph, computing the set of states that can be obtained by following this cycle arbitrarily many times.

The computation of acceleration operators is strongly dependent on the type of the data values that are manipulated by the system, and on the symbolic representation chosen for handling sets of such values. In this survey, we describe acceleration operators suited for the regular state-space exploration of systems relying on FIFO communication channels, as well as those based on integer and real variables.

Key words: Regular model-checking, infinite-state systems, state-space exploration, acceleration techniques

1 Introduction

This survey addresses verification of safety properties of infinite-state systems. We consider systems modeled by extended automata: A model is composed of a control part described by a finite graph, augmented by a data part represented by variables defined over a given, generally infinite, domain. The nodes and edges of the control graph respectively correspond to control locations and to the possible transitions between them. A configuration of the model is then obtained by combining a control location with a data value for the variables. One can move from one configuration to another by following an edge in the control graph, in which case the variable values are updated according to the data transformation that labels this edge. A control edge can also be labeled by a constraint over the variable values, that must be satisfied in order to follow the edge; this provides a way for the data part of the system to influence its flow of control. Finally, we also make it possible for a system configuration to evolve without changing the current control location, in order to cover formalisms such as hybrid automata [ACH+95, AHH93, Hen96]. This is achieved by associating each control location with an evolution law that describes the transformations undergone by the data values when the system spends some time at that location.

It is worth mentioning that the extended automaton modeling a given system does not necessarily need to be represented explicitly. If the system takes the form, for instance, of a combination of concurrent processes, then its model may be expressed as the product of separate control graphs, and be generated on-the-fly during the state-space exploration procedure. In such a setting, the variables extending the control part of the system may be employed as communication objects between the concurrent processes [BZ83].

It might be objected that infinite-state models do not correspond to physical reality, since all realizable computer systems are inherently restricted to processing only a finite amount of information, and are thus fundamentally finite-state. Infinite-state models are indeed an abstraction of real systems, but being able to analyze them is nonetheless useful for several reasons. First, we expect the techniques that will be developed for dealing with infinite-state models to be also ap-

* Work partially supported by the *Interuniversity Attraction Poles* program *MoVES* of the Belgian Federal Science Policy Office, and by the grants 2.4530.02 and 2.4607.08 of the Belgian Fund for Scientific Research (F.R.S.-FNRS).

plicable to those with a finite but very large state space. Second, some systems rely on variables that are bounded not by a constant, but by a parameter whose precise value is unknown. In order to analyze such a system regardless of this parameter, a natural strategy is to study an infinite-state model in which the value of the parameter is first selected non-deterministically, before running the original system with the chosen value. Third, some classes of systems admit variables that are bounded, but that can nevertheless take an infinite number of possible values. This is the case, for instance, for timed systems [Dil89, ACD90], in which some variables are used as timers for measuring delays between actions that are performed, or events that are observed. If a dense representation is preferred to a discretization of time, such delays can take uncountably many values in any interval of non-zero width, which leads to an infinite state space.

Our goal is to check whether a system modeled by an extended automaton satisfies a safety property. Such a property can be specified by associating to each control location a predicate that must be satisfied by the variable values at that location, at each step of every possible execution of the model. In other words, the problem is to check whether each reachable configuration of the model satisfies the stated property. We will also consider a stronger form of this problem, the reachability computation problem, whose aim is to compute the set of reachable configurations of a given model. Once this set has been computed, it can then be compared against the set of configurations that satisfy a given property. The advantage of this approach is that the result of the reachability computation contains enough information for deciding any safety property, without requiring a new computation when a new property needs to be investigated.

The main approach used for addressing the reachability computation problem is state-space exploration. The procedure starts from the initial configurations and propagates reachability information both by following control transitions and letting time elapse at control locations, which produces new reachable configurations. This operation is repeated until no new reachable configuration is discovered, i.e., a fixed point has been reached. This approach has successfully been applied to the analysis of finite-state systems, where it benefits from a theoretical guarantee of termination[1]. It is implemented in tools such as Spin [Hol91].

In the case of models that have infinitely many reachable configurations, state-space exploration algorithms need to be adapted. In order to be able to obtain infinite sets of configurations in a finite number of computation steps, one uses a symbolic procedure: Instead of manipulating individual configurations, state-space exploration proceeds by handling sets of configurations represented with the help of a suitable symbolic formalism [McM93]. This approach is not restricted to infinite-state systems; symbolic representations such as Binary Decision Diagrams (BDDs) [Bry92], suited for finite-state models, make it possible to carry out symbolic exploration of large state spaces [BCM+92].

Using a symbolic representation for the configurations visited during state-space exploration would not be useful if each computation step yielded only a single new reachable configuration. Indeed, if the exploration of an infinite state-space starts from finitely many initial configurations and has to terminate in finite time, then clearly some computation steps will have to produce infinitely many new reachable configurations. It is thus essential to develop operators that are able to synthesize the largest possible sets of reachable configurations from an initial one. There are essentially two approaches to solving this problem. The first one is to apply *widening* operators, which guarantee termination of state-space exploration at the expense of producing only an over-approximation of the reachable set of configurations [CC77, BT11]. Alternatively, *acceleration* operators make it possible to perform an exact computation of the reachable configurations, but without a guarantee of termination with most classes of systems [BW94]. These acceleration operators can either be generic, in the sense that they operate independently from any data domain [BLW03, Leg11], or specific to a particular domain.

This article gives a survey of acceleration methods that are specific to a few data domains among those that are relevant to actual applications. The main idea behind those methods is to iterate cycles in the control graph of the model under analysis: From an original set of reachable configurations, represented symbolically, one computes the set of configurations that can be reached by following a given control cycle an arbitrary number of times. This operation will generally not be feasible for every cycle, since it may produce a set of configurations that cannot be expressed in the symbolic representation that is employed. Thus, the algorithms developed for accelerating cycles will strongly depend on the expressive power of this symbolic representation.

In the regular model-checking framework, sets of configurations are represented by regular languages or, equivalently, by finite-state automata [AJNd03, AJNS04]. Since we consider models that have a finite control part associated with unbounded data, it is actually sufficient to represent symbolically the sets of data values that need to be handled. In this survey, we introduce automata-based symbolic data structures suited for the data domains we consider, and present corresponding acceleration algorithms.

2 Modeling Formalism

We introduce a formalism that combines features from extended finite-state machines as well as from hybrid automata.

2.1 Syntax

Definition 1. An *Extended Hybrid Automaton (EHA)* is a tuple $(D, C, E, c_0, V_0, I, T, G, A)$, where:

[1] In practice however, finite state-space exploration may fail to be achievable with a reasonable amount of time and memory.

- D is a finite or infinite *data domain*. This domain can either be monolithic, or structured as the Cartesian product $D = D_1 \times D_2 \times \cdots \times D_n$ of simpler domains D_i, with $n > 0$. In the former case, the data part of the EHA is composed of a single variable, usually denoted x, taking its values in D. In the latter case, the EHA has n variables $x_1, x_2, \ldots, x_n$ over the respective domains $D_1, D_2, \ldots, D_n$. These variables can be grouped into a single vector $\mathbf{x}$ taking its values in D.
- (C, E), with $E \subseteq C \times C$, is a finite directed *control graph*, the nodes of which are the *control locations*.
- $c_0 \in C$ is an *initial control location*.
- $V_0 \subseteq D$ is a set of *initial data values*.
- $I : C \to 2^D$ associates to each control location an *invariant* that specifies the data values that are valid at that location.
- $T : C \to 2^{D \times D}$ associates to each control location an *evolution law*, which is a relation that describes the possible evolutions of the current data value when staying at that location for some amount of time.
- $G : E \to 2^D$ associates to each edge of the control graph a *guard*, which is a predicate that must be satisfied by the current data value in order to be able to follow that control edge.
- $A : E \to 2^{D \times D}$ associates to each edge of the control graph a *transformation* that is applied to the current data value when that control edge is followed.

In order to be well formed, an EHA must satisfy some integrity constraints. First, the initial data values must satisfy the invariant at the initial control location: The function I must be such that $V_0 \subseteq I(c_0)$. Second, we require evolution laws to be reflexive, in order to not force data values to systematically evolve at control locations. Formally, for each $c \in C$ and $v \in D$, the function T must be such that $(v, v) \in T(c)$.

2.2 Semantics

A *configuration* of an EHA $(D, C, E, c_0, V_0, I, T, G, A)$ is a pair (c, v) with $c \in C$ and $v \in D$. The transition relation between configurations is defined as follows. A configuration (c', v') is said to be *edge-reachable* from a configuration (c, v) if there exists an edge $e \in E$ such that $e = (c, c')$, $v \in G(e)$, $(v, v') \in A(e)$, and $v' \in I(c')$. This is denoted $(c, v) \xrightarrow{e} (c', v')$. Additionally, a configuration (c', v') is said to be *location-reachable* from a configuration (c, v) if $c' = c$, $(v, v') \in T(c)$, and $v' \in I(c)$. This is denoted $(c, v) \xrightarrow{c} (c', v')$. A configuration (c', v') is *reachable in one step* from a configuration (c, v), which is denoted $(c, v) \to (c', v')$, if one has either $(c, v) \xrightarrow{e} (c', v')$ for some $e \in E$, or $(c, v) \xrightarrow{c} (c', v')$. A configuration (c', v') is *reachable* from a configuration (c, v) if one has $(c, v) \to^* (c', v')$, where $\to^*$ denotes the reflexive and transitive closure of the one-step reachability relation $\to$. Finally, a configuration (c, v) is *reachable* if one has $(c_0, v_0) \to^* (c, v)$ for some $v_0 \in V_0$. The *reachability set* of the EHA is the set of all its reachable configurations.

EXPLORE$((D, C, E, c_0, V_0, I, T, G, A) :$ EHA$)$:

$U := \{c_0\} \times V_0$;
$S := U$;
while $S \neq \emptyset$ **do**:
  $S := \{(c', v') \mid \exists (c, v) \in S : c = c' \wedge (v, v') \in T(c) \wedge v' \in I(c)\}$;
  $S' := \{(c', v') \mid \exists (c, v) \in S, e \in E : e = (c, c') \wedge v \in G(e) \wedge (v, v') \in A(e) \wedge v' \in I(c')\}$;
  $S := (S \cup S') \setminus U$;
  $U := U \cup S$;
return U.

Fig. 1. State-space exploration algorithm.

3 Domain-specific regular model checking

3.1 Symbolic state-space exploration

The *reachability computation problem* consists in computing the set of all reachable configurations of an EHA $(D, C, E, c_0, V_0, I, T, G, A)$. Since this set is generally infinite, the result of this computation will have to be represented symbolically, by means of a suitable data structure. The set of control locations C is finite, hence it is actually sufficient to consider a symbolic representation system for subsets of the data domain D, and to represent a set $S \subseteq C \times D$ of configurations by associating one such subset $S_c \subseteq D$ to every control location $c \in C$; in other words, one has $S = \{(c, v) \mid c \in C \wedge v \in S_c\}$.

A procedure for carrying out state-space exploration is given in Figure 1. It proceeds by building successively larger sets of reachable configurations, starting from the initial ones, and adding iteratively those that are reachable in one step from the current set, until a fixed point is reached. This algorithm maintains two main variables U and S, for respectively storing the set of reachable configurations obtained so far, and the subset from which the exploration can continue. Each computation step consists of first applying the evolution laws at the locations reached by the states in S, and then attempting to follow a transition from the resulting configurations.

The algorithm in Figure 1 is not guaranteed to terminate for most classes of EHA, since the reachability computation problem is undecidable for all Turing-capable models. It can be implemented symbolically, provided that the data structure chosen for representing data values is closed under the needed operations, in particular set union, set difference, test of emptiness, intersection with guards and invariants, and application of transformations and evolution laws.

3.2 Acceleration

In order to speed up state-space exploration of a model, or to help it terminate, one can augment its transition relation with *acceleration* operators, also known under the term *meta-transitions* [BW94, Boi98]. Given an EHA $(D, C, E, c_0, V_0,$

EXPLORE-META($(D, C, E, c_0, V_0, I, T, G, A)$: EHA,
M : finite set of meta-transitions) :
$U := \{c_0\} \times V_0$;
$S := U$;
while $S \neq \emptyset$ **do**:
$S' := S$;
for each $m \in M$ **do**:
$S' := S' \cup \{q' \mid \exists q \in S : (q, q') \in m\}$;
$S' := \{(c', v') \mid \exists (c, v) \in S' : c = c' \wedge (v, v') \in T(c) \wedge v' \in I(c)\}$;
$S'' := \{(c', v') \mid \exists (c, v) \in S', e \in E : e = (c, c') \wedge v \in G(e) \wedge (v, v') \in A(e) \wedge v' \in I(c')\}$;
$S := (S' \cup S'') \setminus U$;
$U := U \cup S$;
return U.

Fig. 2. Accelerated state-space exploration algorithm.

$I, T, G, A)$, a meta-transition is a relation $m \subseteq (C \times D)^2$ that is a subset of the relation $\rightarrow^*$. It thus has the property of preserving reachability; in other words, the relation m is such that for every $q, q' \in C \times D$, if q is reachable and $(q, q') \in m$, which can be denoted $q \xrightarrow{m} q'$, then the configuration q' is reachable as well.

An algorithm that performs state-space exploration of an EHA by exploiting a finite set M of meta-transitions is described in Figure 2. In order to implement this algorithm, one needs a symbolic representation system that allows to compute the image of a symbolically represented set of configurations by a meta-transition.

This algorithm is only applicable if a method for synthesizing meta-transitions from an EHA is available. A simple approach is based on the iteration of control cycles. A *control cycle* is a sequence $\sigma = e_1 e_2 \dots e_m \in E^+$ of edges such that for all $i \in [1, m]$, $e_i = (c_i, c_{i+1})$, with $c_1, c_2, \dots, c_{m+1} \in C$ and $c_{m+1} = c_1$. Following the cycle σ from the configuration $q \in C \times D$ leads to the configurations q' such that there exist $q_1, q_1', q_2, q_2', \dots, q_{m+1} \in C \times D$ such that $q = q_1$, $q_1 \xrightarrow{c_1} q_1' \xrightarrow{e_1} q_2 \xrightarrow{c_2} q_2' \xrightarrow{e_2} \cdots \xrightarrow{e_m} q_{m+1}$ and $q_{m+1} = q'$. This is denoted $q \xrightarrow{\sigma} q'$. The *cycle meta-transition* associated to σ is then defined as the reflexive and transitive closure of the relation $\xrightarrow{\sigma}$, and is denoted $\xrightarrow{\sigma^*}$. In other words, following once that cycle meta-transition produces in one step all the configurations that could have been obtained by following the sequence σ any number of times.

Note that the configurations q and q' share the same control location c_1, hence following the cycle meta-transition associated to σ amounts to applying the data transformation $\{(v, v') \in D^2 \mid (c_1, v) \xrightarrow{\sigma^*} (c_1, v')\}$. Abusing the notation, we will also denote this relation by σ^*. Synthesizing cycle meta-transitions thus requires algorithms for computing the data transformations associated to given control sequences, as well as for computing the image of sets of data values by the transitive and reflexive closure of such transformations. For the latter operation, it is essential to take into account the limitations of the symbolic representation system used for handling sets of data values: Given a representable set $V \subseteq D$ and a cycle σ, the set $\sigma^*(V)$ needs to be representable in order to be able to turn σ into a cycle meta-transition. One must also be able to compute a representation of $\sigma^*(V)$ from a representation of V and a specification of σ. A natural solution is to provide an algorithm that checks, for a given sequence σ, whether it satisfies this property for all representable sets V. This decision procedure does not have to be perfect; a sufficient criterion is acceptable.

There are several possible ways of selecting the control cycles from which cycle meta-transitions will be created. A first approach is to require the user to provide those cycles together with the model specifications [Boi98]. Another solution is to search exhaustively or heuristically for control cycles during state-space exploration, and to turn into meta-transitions those that can be accelerated according to the criteria that have been discussed [Boi98, BFLP03].

In the next sections, we will consider several classes of EHA motivated by actual applications. For each of them, we will describe a finite-state representation system suited for its data domain, as well as algorithms for synthesizing meta-transitions and carrying out accelerated symbolic state-space exploration.

4 FIFO systems

We now describe a modeling formalism that is useful for reasoning about systems composed of concurrent processes communicating asynchronously.

4.1 Definition

A *FIFO channel*, or *queue*, is an object q whose possible values are the finite words defined over a finite set Σ_q of symbols called its *channel alphabet*. Two operations can be applied to FIFO channels: Given a channel q and a symbol $a \in \Sigma_q$, the *send* operation $q!a$ consists in appending that symbol to the content of q; formally, $q!a(w) = wa$ for every $w \in \Sigma_q^*$. The *receive* operation $q?a$ is only defined if a is the leading symbol of the content of q, and consists in removing this symbol. Formally, $q?a(aw) = w$, and $q?a(bw)$ is undefined for every $w \in \Sigma_q^*$ and $b \in \Sigma_q \setminus \{a\}$. The value of $q?a(\varepsilon)$ is undefined as well for every $a \in \Sigma_q$, where ε denotes the empty channel content.

Definition 2. A *FIFO EHA* is an EHA $(D, C, E, c_0, V_0, I, T, G, A)$ such that:

- Its data domain is of the form $D = \Sigma_{q_1}^* \times \Sigma_{q_2}^* \times \cdots \times \Sigma_{q_n}^*$, where $n > 0$ and q_1, q_2, …, q_n are FIFO channels. For simplicity sake, we assume w.l.o.g. that the channel alphabets $\Sigma_{q_1}, \Sigma_{q_2}, \dots, \Sigma_{q_n}$ are pairwise disjoint.
- Its control edges are labeled by send and receive operations, i.e., for each $e \in E$, $A(e) \in \bigcup_{i \in [1,n],\, a \in \Sigma_{q_i}} \{q_i!a, q_i?a\}$. An operation of the form $q_i!a$ or $q_i?a$ only affects the FIFO channel q_i; the contents of the channels q_j such that $j \neq i$ stay unchanged.

– Its invariant, guard, and evolution law relations are trivial, i.e., for every $c \in C$ and $e \in E$, $I(c) = D$, $G(e) = D$, and $T(c) = \{(v, v) \mid v \in D\}$.

4.2 Symbolic representation of data sets

In order to be able to carry out symbolic state-space exploration, we need to introduce a symbolic data structure suited for representing subsets of the data domain $D = \Sigma_{q_1}^* \times \Sigma_{q_2}^* \times \cdots \times \Sigma_{q_n}^*$. Since we have assumed the channel alphabets to be pairwise disjoint, a value $v = (w_1, w_2, \ldots, w_n) \in D$ is uniquely determined by the word $w_1 w_2 \cdots w_n$ over the alphabet $\Sigma = \Sigma_{q_1} \cup \Sigma_{q_2} \cup \ldots \cup \Sigma_{q_n}$, where the order in which the channel contents are concatenated is fixed arbitrarily. We then say that the word w is an *encoding* of the data value v.

This encoding scheme maps every set $V \subseteq D$ of data values onto a language $L(V) \subseteq \Sigma^*$. If a set V is such that the language $L(V)$ is regular, then we say that V is *representable*, and that every finite-state automaton accepting $L(V)$ is a *finite-state representation* of V. In the case of FIFO EHA, finite-state representations are known as *Queue-content Decision Diagrams (QDDs)* [BGWW97, Boi98].

Representing sets of FIFO channel contents by finite-state automata presents several advantages. First, computing the intersection, union, or difference of represented sets, as well as checking whether such a set is empty, reduce to performing the same operations on the languages accepted by finite automata, which is algorithmically simple. Second, finite automata can be determinized and then minimized into a canonical form. This makes it possible to obtain a symbolic representation of a set that is independent from the history of its construction. This feature is useful in state-space exploration applications, which often produce sets with a simple structure, but only after long sequences of operations. The drawback is that determinizing automata may incur an exponential blowup, but this worst-case cost is seldom experienced in some specific application fields [WB00]. In situations where determinization is deemed to be a prohibitive operation, one can nevertheless apply transformations aimed at reducing the size of automata without altering their accepted language, such as quotienting their set of states by a simulation or bi-simulation relation.

4.3 Algorithms

In order to be able to carry out symbolic state-space exploration of FIFO EHA, one needs algorithms for computing the image by send and receive operations of sets of channels contents represented by QDDs The following result has been established in [BGWW97, Boi98].

Theorem 1. *Let q_1, q_2, …, q_n, with $n > 0$, be FIFO channels with respective channel alphabets Σ_{q_1}, Σ_{q_2}, …, Σ_{q_n}, supposed to be finite and pairwise disjoint. Let $V \subseteq \Sigma_{q_1}^* \times \Sigma_{q_2}^* \times \cdots \times \Sigma_{q_n}^*$ be a QDD-representable set of channels contents. Let $i \in [1, n]$ and $a \in \Sigma_{q_i}$. The sets $q_i!a(V) = \{q_i!a(w) \mid w \in V\}$ and $q_i?a(V) = \{q_i?a(w) \mid w \in V\}$ are representable as well. There exist algorithms for computing QDDs representing $q_i!a(V)$ and $q_i?a(V)$ from a representation of the set V.*

Intuitively, a QDD representing V reads the contents of the FIFO channels q_1, q_2, …, q_n sequentially, in that order. Performing an operation $q_i!a$ thus amounts to inserting additional transitions reading the symbol a between those reading symbols from the alphabets Σ_{q_i} and Σ_{q_j} with $j > i$ (or between a transition reading a symbol from Σ_{q_i} and an accepting state if $i = n$). Similarly, applying an operation $q_i?a$ amounts to modifying the destinations of some transitions labeled by symbols from Σ_{q_j} with $j < i$ (or the initial states of the automaton if $i = 1$), so as to skip exactly one transition labeled by a before recognizing the content of q_i. These operations can be performed with simple automata constructions. Alternatively, they can also be expressed as the application to QDDs of some carefully crafted *transducers*, i.e., finite automata with an input and an output tape.

We now study the possibility of extracting cycle meta-transitions from FIFO EHA. Since invariant, guards, and evolution laws of these EHA are trivial, following a control cycle corresponds to applying to the channel contents a sequence $\sigma = \sigma_1 \sigma_2 \ldots \sigma_m$, where each σ_i is either a send or a receive operation. The following result states that, in the particular case of a FIFO EHA with only one channel, every such sequence can be turned into a cycle meta-transition [BGWW97, Boi98]. In this case, QDDs degenerate into finite automata directly accepting the channel contents.

Theorem 2. *Let q be a FIFO channel over the finite alphabet Σ, and let σ be a finite sequence of send and receive operations on this channel. For every regular set $V \subseteq \Sigma^*$ of channel contents, the set $\sigma^*(V)$ is regular as well. There exists an algorithm for computing an automaton accepting $\sigma^*(V)$ from one accepting V.*

The proof of Theorem 2 is technical, but based on a simple principle. When a sequence σ of send and receive operations is repeatedly applied to a one-channel QDD, the automata $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, …, that are obtained after each iteration acquire periodicity properties: There exists a constant $p \in \mathbb{N}_{>0}$ such that for every $i \in \mathbb{N}_{>0}$, the automata $\mathcal{A}_{pi}$ and $\mathcal{A}_{p(i+1)}$ only differ by a constant "increment", which is an additional set of states and transitions that is present in $\mathcal{A}_{p(i+1)}$ but not in $\mathcal{A}_{pi}$. By finite-state constructions, one can then build an automaton simulating the effect of an arbitrary number of repetitions of such an increment, from which an automaton accepting the union of the languages accepted by all the $\mathcal{A}_i$ can easily be obtained. The complete construction is described in [Boi98]. It is similar in principles to the mechanism proposed in [BLW03].

If a FIFO EHA contains more than one FIFO channel, extracting cycle meta-transitions becomes more difficult. Indeed, in this case, iterating a sequence of send and receive operations does not always preserve the representability of

sets of channel contents. For instance, in the case of an EHA relying on two channels q_1 and q_2 with the respective channel alphabets $\Sigma_{q_1} = \{a\}$ and $\Sigma_{q_2} = \{b\}$, iterating the sequence $q_1!a;\ q_2!b$ from the initial set of contents $\{(\varepsilon, \varepsilon)\}$ produces the channel contents $\{(a^i, b^i) \mid i \in \mathbb{N}\}$, encoded by the non-regular language $\{a^i b^i \mid i \in \mathbb{N}\}$. The following definition and theorem fully characterize the sequences of send and receive operations that can be accelerated [BGWW97, Boi98].

Definition 3. Let σ be a finite sequence of send and receive operations over the FIFO channels $q_1, q_2, \ldots, q_n$, with $n > 0$, of respective channel alphabets $\Sigma_{q_1}, \Sigma_{q_2}, \ldots, \Sigma_{q_n}$. For each $i \in [1, n]$, let $|\sigma_{!i}|$ and $|\sigma_{?i}|$ denote respectively the number of instances of send and receive operations in σ involving the channel q_i. The sequence σ is *counting* for the channel i if one of the following conditions is fulfilled:

- $|\Sigma_{q_i}| > 1$ and $|\sigma_{!i}| > 1$, or
- $|\Sigma_{q_i}| = 1$ and $|\sigma_{!i}| > |\sigma_{?i}|$.

Theorem 3. *Let σ be a finite sequence of send and receive operations over the FIFO channels $q_1, q_2, \ldots, q_n$, with $n > 0$. The set $\sigma^*(V)$ of channel contents is representable for every representable set V of channel contents iff there does not exist $i, j \in [1, n]$ such that $i \neq j$ and σ is counting for both the channels i and j.*

Intuitively, a sequence σ_i operating on a specific channel q_i is counting for that channel if there exists an initial regular set V_i of contents of q_i such that, for every $k > 0$, the value of k can be inferred from the contents of $\sigma_i^k(V_i)$. As a consequence, if a sequence σ is counting for at least two distinct channels, one can produce a representable set V of channel values such that $\sigma^*(V)$ cannot be represented by a finite-state automaton. Reciprocally, if a sequence σ is counting for at most one channel, then a QDD representing $\sigma^*(V)$ can be constructed from one representing V, using Theorem 2 and applying simple automata constructions. This leads to the following result [BGWW97, Boi98].

Theorem 4. *Let σ be a finite sequence of send and receive operations over the FIFO channels $q_1, q_2, \ldots, q_n$, with $n > 0$, that is counting for at most one channel. There exists an algorithm for computing, for every representable set V of channels contents, a QDD representing $\sigma^*(V)$ from a QDD representing V.*

4.4 Extension to unreliable FIFO systems

The FIFO EHA that we have considered so far are based on perfect communication channels: A symbol placed in a FIFO channel by a send operation can always be reliably retrieved by a subsequent receive operation. In order to model more accurately some systems, it is also useful to consider *lossy* FIFO channels, in which symbols introduced by send operations can either be successfully delivered or be dropped by the channel [AJ96, ACABJ04]. It is known that safety properties of systems modeled by finite-state machines extended with lossy FIFO channels are decidable [AJ96], but the set of reachable configurations of such systems is generally uncomputable [CFI96].

The results outlined in Sections 4.1 to 4.3 can easily be adapted to lossy FIFO channels [ABJ98]. A simple way of modeling the loss of channel symbols consists in introducing evolution laws that allow to drop non-deterministically at each control location the symbols stored into the channels. We obtain the following definition.

Definition 4. A *Lossy FIFO EHA* is an EHA $(D, C, E, c_0, V_0, I, T, G, A)$ such that:

- Its data domain D and control graph (C, E) are identical to those of a FIFO EHA.
- Its invariant and guard relations are trivial, i.e., for every $c \in C$ and $e \in E$, $I(c) = D$ and $G(e) = D$.
- At each location $c \in C$, the evolution law $T(c)$ is the relation that maps each channels content $v \in D$ onto the contents v' that can be obtained by dropping any number of symbols from v. Let $w' \preceq w$ denote the fact that the word w' is a subword of w, in the sense that w' is composed of non-necessarily consecutive symbols extracted from w, in the same order. Formally, we have $w' \preceq w$ iff there exist words $u'_1, u'_2, \ldots, u'_p$ and $u_1, u_2, \ldots, u_{p+1}$ such that $w' = u'_1 u'_2 \ldots u'_p$ and $w = u_1 u'_1 u_2 u'_2 \ldots u_p u'_p u_{p+1}$. The evolution law $T(c)$ is then defined as $T(c) = \{((w_1, \ldots, w_n), (w'_1, \ldots, w'_n)) \mid \forall i \in [1, n] : w'_i \preceq w_i\}$.

The state-space exploration and acceleration algorithms discussed in Section 4.3 are also applicable to Lossy FIFO EHA. For state-space exploration, one additionally needs an algorithm for applying an evolution law to a set of channel contents represented to a QDD. This operation can be carried out thanks to the following theorem.

Theorem 5. *Let $q_1, q_2, \ldots, q_n$, with $n > 0$, be lossy FIFO channels over the respective alphabets $\Sigma_{q_1}, \Sigma_{q_2}, \ldots, \Sigma_{q_n}$. If a set $V \subseteq \Sigma_{q_1}^* \times \Sigma_{q_2}^* \times \cdots \times \Sigma_{q_n}^*$ is QDD-representable, then the set $V' = \{(w'_1, \ldots, w'_n) \mid \exists (w_1, \ldots, w_n) \in V : \forall i \in [1, n] : w'_i \preceq w_i\}$ is representable as well. There exists an algorithm for constructing a QDD representing V' from one representing V.*

Proof. Let $\mathcal{A}$ be a QDD representing a set V of channel contents. In order to turn $\mathcal{A}$ into a QDD representing the set V' obtained by non-deterministically removing symbols from the channels contents in V, it suffices to add for each transition (q, a, q') of $\mathcal{A}$, with $a \in \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n$, another transition (q, ε, q') labeled by the empty word. This models the fact that the symbol a transmitted by the channel can be either delivered or dropped. □

In addition to lossy FIFO EHA, we can also consider systems in which the source of unreliability is the insertion of arbitrary symbols in the contents of the communication queues [CFI96]. Such systems are defined as follows.

Definition 5. A *FIFO EHA with insertion errors* is an EHA $(D, C, E, c_0, V_0, I, T, G, A)$ such that:

– Its data domain D and control graph (C, E) are identical to those of a FIFO EHA.
– Its invariant and guard relations are trivial, i.e., for every $c \in C$ and $e \in E$, $I(c) = D$ and $G(e) = D$.
– At each location $c \in C$, the evolution law $T(c)$ is the relation that maps each channels content $v \in D$ onto the contents v' that can be obtained by inserting any number of symbols at arbitrary positions in v. Formally, we have $T(c) = \{((w_1, \ldots, w_n), (w'_1, \ldots, w'_n)) \mid \forall i \in [1, n] : w_i \preceq w'_i\}$.

Symbolic state-space exploration of FIFO EHA with insertion errors only differs from that of lossy FIFO EHA in the application of evolution laws. This operation is feasible thanks to the following theorem.

Theorem 6. *Let q_1, q_2, …, q_n, with $n > 0$, be FIFO channels with insertion errors, over the respective alphabets Σ_{q_1}, Σ_{q_2}, …, Σ_{q_n}. If a set $V \subseteq \Sigma_{q_1}^* \times \Sigma_{q_2}^* \times \cdots \times \Sigma_{q_n}^*$ is QDD-representable, then the set $V' = \{(w'_1, \ldots, w'_n) \mid \exists (w_1, \ldots, w_n) \in V : \forall i \in [1, n] : w_i \preceq w'_i\}$ is representable as well. There exists an algorithm for constructing a QDD representing V' from one representing V.*

Proof. Let $\mathcal{A}$ be a QDD representing a set V of channel contents. This QDD can be turned into one representing the set V' obtained by non-deterministically inserting symbols at arbitrary positions in the channels contents by the following procedure. First, since $\mathcal{A}$ reads the contents of the channels in a fixed order, which we can assume w.l.o.g. to be $q_1; q_2; \ldots; q_n$, it can be decomposed into a sequence $\mathcal{A}_1; \mathcal{A}_2; \ldots; \mathcal{A}_n$ of finite-state machines such that each $\mathcal{A}_i$ operates on the channel alphabet Σ_{q_i}. An automaton accepting the same language as $\mathcal{A}$ can then be reconstructed by linking, for each $i \in [1, n-1]$, some final states of $\mathcal{A}_i$ to some initial states of of $\mathcal{A}_{i+1}$ by means of transitions labeled by the empty word. It then suffices to add, for each $i \in [1, n]$, state q of $\mathcal{A}_i$, and symbol $a \in \Sigma_i$, a transition (q, a, q) to $\mathcal{A}_i$, in order to model the fact that arbitrary symbols can be inserted in the contents of q_i. □

5 Integer counter systems

In this section, we study systems modeled by finite-state machines extended with a fixed number of unbounded integer variables, on which linear operations are performed.

5.1 Definition

Definition 6. An *Integer EHA* is an EHA $(D, C, E, c_0, V_0, I, T, G, A)$ such that:

– Its data domain is of the form $D = \mathbb{Z}^n$, with $n > 0$. In other words, a data value is a vector with n components, and can be seen as the assignment of values to n unbounded integer variables x_1, x_2, …, x_n.
– Its invariant and guards are conjunctions of linear constraints expressed over the vector of variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Formally, for each $c \in C$ and $e \in E$, the invariant $I(c)$ and the guard $G(e)$ are systems of constraints of the form $P\mathbf{x} < \mathbf{q}$, with $P \in \mathbb{Z}^{m \times n}$, $\mathbf{q} \in \mathbb{Z}^m$, and $m > 0$.
– Each control edge $e \in E$ is labeled by a *linear assignment* of the form $A(e) = \{(\mathbf{v}, P\mathbf{v} + \mathbf{q} \mid \mathbf{v} \in \mathbb{Z}^n\}$, with $P \in \mathbb{Z}^{n \times n}$ and $\mathbf{q} \in \mathbb{Z}$. Such an assignment can alternatively be denoted $\mathbf{x} := P\mathbf{x} + \mathbf{q}$.
– Its evolution laws are trivial, i.e., for every $c \in C$, $T(c) = \{(\mathbf{v}, \mathbf{v}) \mid \mathbf{v} \in \mathbb{Z}^n\}$.

5.2 Symbolic representation of data sets

In order to obtain a finite-state representation system suited for the data domain $\mathbb{Z}^n$, one needs to define an encoding relation mapping integer vectors onto words over a finite alphabet.

A natural solution is to use the *positional notation* that is usually employed for denoting integer numbers. The first step is to choose a *numeration base* $r \in \mathbb{N}_{>1}$, which provides an alphabet of *digits* $\Sigma_r = \{0, 1, \ldots, r-1\}$. A natural number $m \in \mathbb{N}$ is then encoded by the words $a_{p-1}a_{p-2}\ldots a_0 \in \Sigma_r^*$, with $p \geq 0$, such that $m = \sum_{i \in [0,p-1]} a_i r^i$.

This encoding technique generalizes to signed numbers thanks to the *base complement* method, which consists in encoding a signed number $z \in \mathbb{Z}$ such that $-r^{p-1} \leq z < r^{p-1}$ by the last p digits of the encodings of $r^p + z$, for any $p > 0$. With this scheme, the leading digit of an encoding is always equal to $r-1$ for negative numbers, and to 0 for non-negative ones. This digit is thus referred to as the *sign digit* of encodings. The sign digit of an encoding can be repeated at will without influencing the value of the encoded number. Every number $z \in \mathbb{Z}$ admits an infinite number of distinct encodings, which only differ in the number of repetitions of their sign digit [WB95].

The positional encoding of signed numbers can be generalized to vectors with a fixed dimension. A vector $(v_1, v_2, \ldots, v_n)$ is encoded by first obtaining encodings w_i of its individual components v_i, in such a way that they share the same length p. This can always be achieved by repeating the sign digit of encodings the appropriate number of times. Then, the words w_i are read synchronously from left to right, one symbol at a time, which yields a single word of length p over the alphabet Σ_r^n, containing n-tuples of digits. The first symbol of a vector encoding necessarily belongs to the restricted alphabet $\{0, r-1\}^n$, and characterizes the sign of the vector components. This symbol is thus referred to as the *sign symbol* of encodings. Every vector in $\mathbb{Z}^n$ thus admits infinitely many distinct encodings, that only differ in the number of times their sign symbol is repeated.

This encoding relation is well suited for theoretical developments. However, in practical applications, the exponential size of the alphabet with respect to the data space dimension n becomes problematic. This problem can be mitigated

by *serializing* the encodings of vectors, which consists in replacing a tuple $(w_1, w_2, \ldots, w_n)$ of same-length encodings $w_i \in \Sigma_r^p$ of vector components by a single word w over the simpler alphabet Σ_r, obtained by reading one symbol from each w_i repeatedly and in turn [Boi98, BL04]. Formally, we have $w = w_1[1]w_2[1]\ldots w_n[1]\ w_1[2]w_2[2]\ldots w_n[2]\ \ldots\ w_1[p]w_2[p]\ldots w_n[p]$ where, for each $i \in [1, n]$, $w_i[1]$, $w_i[2]$, $\ldots$, $w_i[p]$ denote the successive symbols composing the word w_i.

After choosing a numeration base and an either serialized or unserialized scheme, one obtains an encoding relation that maps a set $V \subseteq \mathbb{Z}^n$ onto a language $L(V)$ that describes the contents of V without ambiguity. If $L(V)$ is regular, then any finite-state automaton accepting this language is called a *Number Decision Diagram (NDD)* representing V. The set V is then said to be representable [WB95, Boi98]. Note that, in order to be valid, a NDD representing V must accept all the encodings of the vectors that belong to V. This condition is introduced in order to be able to compute unions, intersections, and differences of sets represented by NDDs by simply applying the same operations to the languages accepted by the automata.

Whether a set of integer vectors is or is not representable does not depend on the choice of a serialized or unserialized encoding; an automaton recognizing the unserialized encodings of vectors can easily be converted into one reading serialized encodings, and vice-versa [BL04]. On the other hand, the representability of sets is influenced by the numeration base. We have the following results [Büc62, BHMV94].

Theorem 7. *Let $n \in \mathbb{N}_{>0}$ be a dimension and $r \in \mathbb{N}_{>1}$ be a numeration base. A subset of $\mathbb{Z}^n$ is representable by an NDD in base r iff it is definable in the first-order theory $\langle \mathbb{Z}, +, <, V_r \rangle$, where V_r is a function that maps every non-zero integer to the largest integer power of r that divides it, for instance, $V_2(12) = 4$.*

The proof of this result is based on the following ideas. In order to establish that a set S definable in $\langle \mathbb{Z}, +, <, V_r \rangle$ is representable in base r, it suffices to provide base-r NDDs for the elementary relations $\{(z_1, z_2, z_1 + z_2) \mid z_1, z_2 \in \mathbb{Z}\}$, $\{(z_1, z_2) \in \mathbb{Z}^2 \mid z_1 < z_2\}$ and $\{(z_1, z_2) \in \mathbb{Z}^2 \mid z_2 = V_r(z_1)\}$ of this theory, and then show how to combine them into one representing S. For the other direction of the proof, one starts from an arbitrary base-r NDD, and builds a formula φ of $\langle \mathbb{Z}, +, <, V_r \rangle$ that simulates the behavior of this NDD when it reads a word encoding the value of the free variables of φ.

The following result then characterizes the sets of integer vectors that can be represented by NDDs regardless of the numeration base [Cob69, Sem77, BHMV94].

Theorem 8. *Let $n \in \mathbb{N}_{>0}$ be a dimension. A subset of $\mathbb{Z}^n$ is representable by an NDD in every base $r \in \mathbb{N}_{>1}$ iff it is definable in the first-order theory $\langle \mathbb{Z}, +, < \rangle$.*

The first-order additive theory of integer numbers $\langle \mathbb{Z}, +, < \rangle$ is also known as *Presburger arithmetic* [Pre29]. Its expressive power is well suited for state-space exploration of Integer EHA: A Presburger set is essentially a combination of linear constraints, such as those found in the guards and invariants, and discrete periodicities, which appear as the result of iterating transformations such as $\mathbf{x} := \mathbf{x} + \mathbf{b}$, with $\mathbf{b} \in \mathbb{Z}^n$.

5.3 Algorithms

As discussed in the previous section, applying to NDD-represented sets of vectors set-theory operators such as union, intersection and difference simply reduces to performing the same operations on the languages accepted by the automata, just like in the case of QDDs. This does not generalize to all operations over sets. Consider, in particular, the *projection* operator, used for reducing the spatial dimension of a set of vectors. Formally, given a set $V \subseteq \mathbb{Z}^n$, with $n > 1$, and a dimension $i \in [1, n]$, its projection $\pi_{\neq i}(V)$ over the vector components that differ from i is defined as the set $\pi_{\neq i}(V) = \{(v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n) \mid \exists v_i \in \mathbb{Z} : (v_1, \ldots, v_n) \in V\}$. In other words, $\pi_{\neq i}(V)$ is obtained by removing the i-th vector component from each vector in V.

One could be tempted to compute an NDD representing $\pi_{\neq i}(V)$ from one representing V by simply removing the i-th component from all the tuples labeling its transitions. While this procedure indeed yields an automaton that accept encodings of the vectors in $\pi_{\neq i}(V)$, this automaton is, however, not a valid NDD because it generally does not accept all such encodings. For instance, all binary encodings of the vector $(0, 8)$ are at least of length 5, since the shortest signed encoding of 8 is 01000. Applying the procedure that has been discussed to the computation of $\pi_{\neq 2}(\{(0, 8)\})$ thus produces an automaton accepting the language 000000^*, which misses the shorter encodings of 0. In order to solve this problem, one needs an algorithm for adding to a regular language all the encodings of the vectors it represents. Such an algorithm is developed in [BL04].

Performing symbolic state-space exploration of Integer EHA using NDDs requires algorithms for computing the image of a represented set of integer vectors by the model transitions. Formally, the problem consists in computing from an NDD representing a set $V \subseteq \mathbb{Z}^n$, with $n > 0$, an NDD representing the set $\{A\mathbf{v} + \mathbf{b} \mid \mathbf{v} \in V \wedge P\mathbf{v} < \mathbf{q}\}$, where $A \in \mathbb{Z}^{n \times n}$, $\mathbf{b} \in \mathbb{Z}^n$, $P \in \mathbb{Z}^{m \times n}$, $\mathbf{q} \in \mathbb{Z}^m$ and $m > 0$. This set can be seen as the image $\theta(V)$ of V by a *guarded linear transformation* θ composed of a guard $P\mathbf{x} < \mathbf{q}$ followed by a linear assignment $\mathbf{x} := A\mathbf{x} + \mathbf{b}$. A representation of $\theta(V)$ can be obtained thanks to the following theorem [WB95, BC96, Boi98, Kla08].

Theorem 9. *Let $\theta \subseteq \mathbb{Z}^n \times \mathbb{Z}^n$, with $n > 0$, be a Presburger transformation, i.e., a relation definable in the first-order theory $\langle \mathbb{Z}, +, < \rangle$. There exists an algorithm for computing a representation of $\theta(V)$ for any representable set $V \subseteq \mathbb{Z}^n$.*

This result can be proved as follows. A relation $\theta \subseteq \mathbb{Z}^n \times \mathbb{Z}^n$ is uniquely described by its characteristic set $\Theta \subseteq \mathbb{Z}^{2n}$, such that $(v_1, \ldots, v_n, v'_1, \ldots, v'_n) \in \Theta$ iff $((v_1, \ldots v_n), (v'_1,$

$\ldots, v'_n)) \in \theta$. As a consequence of Theorem 8, if θ is a Presburger transformation, then the set Θ is representable by a NDD in any base $r \in \mathbb{N}_{>1}$. Such a NDD can be turned into a transducer, i.e., an automaton with an input and an output tape, that transforms any NDD representing a set $V \subseteq \mathbb{Z}^n$ into an automaton accepting encodings of the elements of $\theta(V)$. This automaton can become a valid NDD representing $\theta(V)$ by adding to its accepted language all the encodings of the vectors it represents, which can be done by the algorithm developed in [BL04].

We now discuss the creation of cycle meta-transitions. The guarded linear transformations are closed under sequential composition, hence the effect of following a control cycle σ in a Integer EHA is equivalent to applying a single guarded transformation $\theta_\sigma : \ P\mathbf{x} < \mathbf{q}\,;\ \mathbf{x} := A\mathbf{x} + \mathbf{b}$ to the vector of variables $\mathbf{x}$. A cycle meta-transition can be created from σ if the closure θ^*_σ preserves the representability of sets of vectors, and if the image of such a set by this closure can effectively be computed. The following result precisely characterizes the guardless linear transformations whose closure preserves representability [Boi98, Boi03].

Theorem 10. *Let θ be a linear transformation of the form*

$$\{(\mathbf{v}, A\mathbf{v} + \mathbf{b}) \mid \mathbf{v} \in \mathbb{Z}^n\},$$

with $n > 0$, $A \in \mathbb{Z}^{n\times n}$, $\mathbf{b} \in \mathbb{Z}^n$. Its reflexive and transitive closure θ^* is such that $\theta^*(V)$ is representable in a base $r \in \mathbb{N}_{>1}$ (resp. in every base $r \in \mathbb{N}_{>1}$) for every set $V \subseteq \mathbb{Z}^n$ that is representable in the same base r (resp. in every base) iff there exist $p \in \mathbb{N}_{>0}$ and $m \in \mathbb{N}$ such that

- *the matrix A^p is diagonalizable, and*
- *its eigenvalues belong to the set $\{0, r^m\}$ (resp. $\{0, 1\}$).*

The proof of this theorem is quite involved. Intuitively, a linear transformation θ such that θ^* preserves the representable nature of sets can be expressed as a combination of transformations τ that apply a translation ($\tau : \mathbf{x} := \mathbf{x} + \mathbf{b}$), perform a scaling ($\tau : \mathbf{x} := \lambda\mathbf{x}$), or are ultimately periodic ($\tau^{p+q} = \tau^p$ for some $p \in \mathbb{N}$ and $q \in \mathbb{N}_{>0}$). Additionally, the scaling factor λ has to be multiplicatively dependent with the chosen numeration base r, i.e., one must have $\lambda^p = r^m$ for some $p \in \mathbb{N}_{>0}$ and $m \in \mathbb{N}$. It is shown in [Boi98, Boi03] that this scaling factor is related to the eigenvalues of the transformation matrix, and that the transformations θ that correspond to combinations of translations, scalings, and ultimately periodic mapping are precisely those for which some integer power of this matrix is diagonalizable.

Theorem 11. *There exists an algorithm for deciding whether a transformation θ satisfies the criterion expressed by Theorem 10 using only integer arithmetic. Moreover, there exists an algorithm for computing a representation of $\theta^*(V)$ from a representation of any set $V \subseteq \mathbb{Z}^n$, for every transformation θ satisfying this criterion.*

For transformations that include a guard, similar results are known, but they only provide a sufficient condition for deciding whether a cycle meta-transition can be created [Boi98, Boi03].

Theorem 12. *Let θ be a linear transformation of the form*

$$\{(\mathbf{v}, A\mathbf{v} + \mathbf{b}) \mid \mathbf{v} \in \mathbb{Z}^n \wedge P\mathbf{v} < \mathbf{q}\},$$

with $n > 0$, $A \in \mathbb{Z}^{n\times n}$, $\mathbf{b} \in \mathbb{Z}^n$, $P \in \mathbb{Z}^{m\times n}$, $\mathbf{q} \in \mathbb{Z}^m$ and $m > 0$. If the guardless transformation

$$\{(\mathbf{v}, A\mathbf{v} + \mathbf{b}) \mid \mathbf{v} \in \mathbb{Z}^n\}$$

satisfies the criterion expressed by Theorem 10 (for representability either in a base $r \in \mathbb{N}_{>1}$ or in every base), then the set $\theta^*(V)$ is representable (in the same setting) for every representable set $V \subseteq \mathbb{Z}^n$. There exists an algorithm for computing a representation of $\theta^*(V)$ from a representation of V for such transformations θ.

6 Linear hybrid systems

The last modeling formalism that we consider in this survey is an extension of Integer EHA to real variables. Evolution laws are also added in order to allow the value of these variables to evolve continuously with time when the control location does not change, in order to cover formalisms such as linear hybrid systems [AHH93, ACH+95]. Like in the case of Integer EHA, the control edges are labeled by discrete transformations over the variable values.

6.1 Definition

A *Linear Hybrid EHA* is an EHA $(D, C, E, c_0, V_0, I, T, G, A)$ such that:

- Its data domain is of the form $D = \mathbb{R}^n$, with $n > 0$, i.e., the data part of the model can be seen as being composed of n real variables $x_1, x_2, \ldots, x_n$.
- Its invariant and guards are conjunctions of linear constraints over the variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Formally, for each $c \in C$ and $e \in E$, the invariant $I(c)$ and the guard $G(e)$ are systems of constraints of the form $P\mathbf{x} \,\#\, \mathbf{q}$, with $P \in \mathbb{Z}^{m\times n}$, $\mathbf{q} \in \mathbb{Z}^m$, $\# \in \{<, \leq\}^m$ and $m > 0$.
- Each control edge $e \in E$ is labeled by a linear assignment of the form $A(e) = \{(\mathbf{v}, P\mathbf{v} + \mathbf{q} \mid \mathbf{v} \in \mathbb{R}^n\}$, with $P \in \mathbb{Z}^{n\times n}$ and $\mathbf{q} \in \mathbb{Z}$. Such an assignment can alternatively be denoted $\mathbf{x} := P\mathbf{x} + \mathbf{q}$.
- Its evolution laws are characterized by lower and upper bounds on the rate of evolution of variables, with respect to the time spent at a control location. Formally, for each $c \in C$, the evolution law $T(c)$ is of the form $T(c) = \{(\mathbf{v}, \mathbf{v'}) \in \mathbb{R}^n \times \mathbb{R}^n \mid \exists t \in \mathbb{R}_{\geq 0} : t\boldsymbol{\ell} \leq \mathbf{v'} - \mathbf{v} \leq t\mathbf{u}\}$, with $\boldsymbol{\ell}, \mathbf{u} \in \mathbb{Z}^n$.

6.2 Symbolic representation of data sets

The positional notation introduced in Section 5.2 for encoding integer values can be generalized so as to obtain a finite-state symbolic representation system suited for sets of real vectors [BBR97].

Given a base $r \in \mathbb{N}_{>1}$, a number $y \in \mathbb{R}$ is encoded by infinite words $a_{p-1}a_{p-2}\ldots a_0 \star a_{-1}a_{-2}\ldots$, with $p > 0$. The symbol "$\star$" is a separator added to the alphabet of digits $\Sigma_r = \{0, 1, \ldots, r-1\}$ in order to distinguish the finite encoding $a_{p-1}a_{p-2}\ldots a_0$ of the integer part $y_I \in \mathbb{Z}$ from the infinite suffix $a_{-1}a_{-2}\ldots$ that encodes the fractional part $y_F \in [0, 1]$. The decomposition $y = y_I + y_F$ of y into an integer and a fractional part is not necessarily unique, e.g., $y = 3$ yields both $y_I = 3$, $y_F = 0$ and $y_I = 2$, $y_F = 1$. The integer part is encoded using the base complement method described in Section 5.2. As a consequence, the first symbol a_{p-1} of an encoding is a *sign digit* and belongs to the restricted alphabet $\{0, r-1\}$. The infinite word $a_{p-1}a_{p-2}\ldots a_0 \star a_{-1}a_{-2}\ldots \in \{0, r-1\}\Sigma_r^* \star \Sigma_r^\omega$ then encodes the number $\sum_{i<p} a_i r^i$ if $a_{p-1} = 0$, and $-r^p + \sum_{i<p} a_i r^i$ if $a_{p-1} = r-1$. The length p of the integer part y_I is not fixed, but must be such that $-r^p \leq y_I < r^p$.

Similarly to the integer case, the sign digit of an encoding can be repeated at will, which implies that every real number admits infinitely many encodings. Some real numbers also admit distinct encodings that share the same integer-part length, which are then called *dual encodings*. This happens for numbers that can be expressed as fractions p/q, with $p \in \mathbb{Z}$ and $q \in \mathbb{Z}_{>0}$, such that the prime factors of q are also prime factors of the numeration base r. For instance, in the base $r = 2$, the encodings of the number $-3/2$ form the language $1^+0 \star 10^\omega \cup 1^+0 \star 01^\omega$.

This encoding scheme can be generalized to real vectors by the same method as in Section 5.2. In order to encode a vector $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$, with $n > 0$, one first obtains encodings w_i of the individual components y_i that share the same number of digits in their integer part. These encodings are then read synchronously one symbol at a time. Since the separator symbol is read at the same time in each vector component, it can be replaced by a single symbol. One thus obtains an encoding of $\mathbf{y}$ as an infinite word over the alphabet $\Sigma_r^n \cup \{\star\}$. In practical implementations where a simpler alphabet is needed, such encodings can also be serialized using the method described in Section 5.2.

Having selected a numeration base, a serialized or unserialized encoding scheme maps a set $V \subseteq \mathbb{R}^n$ onto an infinite-word language $L(V)$ that characterizes this set. If this language is ω-regular, then any infinite-word automaton that accepts $L(V)$ is called a *Real-Vector Automaton (RVA)* representing V [BBR97]. As in the case of NDDs, in order to be valid, a RVA representing a set must accept all encodings of the vectors belonging to that set. Other choices are possible; in particular, it is shown in [EK08] that a careful selection of the encodings accepted by the automata leads to a more compact representation, at the expense of increasing the cost of some manipulation operations.

Working with infinite-word automata is more problematic than manipulating finite-word ones. A first difficulty is that some operations, such as language complementation, are substantially more costly to perform [Saf88, Kla91, KV05]. In addition, infinite-word automata do not have an easily computable canonical form, which makes the symbolic representations of sets of data values considered during state-space exploration dependent on the history of their construction.

These problems can be solved by working with a restricted class of infinite-word automata, provided that their expressive power remains sufficient for our intended application. Recall that a Büchi automaton is an infinite-word automaton whose accepting paths are those that visit infinitely many accepting states. We use the following definition [SW74].

Definition 7. A *weak* infinite-word automaton is a Büchi automaton such that every strongly connected component of its transition graph contains either only accepting or only non-accepting states.

The advantage of working with weak automata is that they are essentially as easy to handle algorithmically as finite-word automata [Wil93]. Moreover, weak deterministic automata can easily be minimized into a canonical form [Ld01]. With respect to the positional encoding of real vectors, the expressive power of weak deterministic RVA is characterized by the following result [BJW05, BBL09].

Theorem 13. *Let $n \in \mathbb{N}_{>0}$ be a dimension. A subset of $\mathbb{R}^n$ is representable by a weak deterministic RVA in every base $r \in \mathbb{N}_{>1}$ iff it is definable in the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$.*

One direction of the proof of this theorem relies on topological characterizations of the sets that are definable in $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$, and of the languages encoding such sets in a given base. More precisely, it is shown in [BJW05] that every such language can be expressed both as a countable union of closed sets and as a countable intersection of open sets, and that all the languages that satisfy this property can be accepted by weak deterministic automata. The reverse direction of the proof is more technical, and provides a generalization of Cobham and Semenov's theorems to automata operating on real vectors [BBL09].

This result can be seen as a generalization to the mixed integer and real domain of Theorem 8. The first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$ is a strict extension of Presburger arithmetic. In this theory, one can express combinations of linear constraints over either integer or real vectors, as well as discrete periodicities [Wei99]. Every set of integer vectors that can be represented by an NDD is therefore representable as well by a weak deterministic RVA. In the rest of this section, we will only consider weak deterministic RVA, and say that a subset of $\mathbb{R}^n$ is representable if it is representable by such RVA.

6.3 Algorithms

The manipulation of sets of real vectors represented by RVA follows the same principles as for NDD-represented sets of integer vectors. Set-theory operators are applied by performing the same operations on the languages accepted by RVA, which is algorithmically simple. Care must be taken to ensure that the automata that are constructed accept all the encodings of the vectors that they recognize. In particular, the projection

operation is carried out by a method that is similar to the one described in Section 5.3.

To perform state-space exploration, one needs to compute the image of RVA-represented subsets of $\mathbb{R}^n$ by the transformations that label control edges, as well as by the evolution laws associated to control locations. The following result generalizes Theorem 9 [BJW05].

Theorem 14. *Let $\theta \subseteq \mathbb{R}^n \times \mathbb{R}^n$, with $n > 0$, be a relation definable in the first-order theory $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$. There exists an algorithm for computing a weak deterministic RVA representing $\theta(V)$ for any set $V \subseteq \mathbb{Z}^n$ represented by such an RVA.*

This theorem is proved in the same way as Theorem 9. It provides a computation method for the image of an RVA-represented set of data values by the guarded linear transformation labeling a control edge. It can also be applied for computing the effect of an evolution law at each control location, since every transformation $\{(\mathbf{v}, \mathbf{v'}) \in \mathbb{R}^n \times \mathbb{R}^n \mid \exists t \in \mathbb{R}_{\geq 0} : t\boldsymbol{\ell} \leq \mathbf{v'} - \mathbf{v} \leq t\mathbf{u}\}$, with $\boldsymbol{\ell}, \mathbf{u} \in \mathbb{Z}^n$, is definable in $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$.

Let us now address the computation of cycle meta-transitions of Linear Hybrid EHA. Following a control cycle σ amounts to applying to the current set of data values an alternation of guarded linear transformations and linear evolution laws. Formally, we have

$$\sigma(V) = \{\mathbf{v'} \mid \exists \mathbf{v}_1, \mathbf{v}'_1, \ldots, \mathbf{v}_m, \mathbf{v}'_m \in \mathbb{R}^n : \mathbf{v}_1 \in V \wedge (c_1, \mathbf{v}_1) \xrightarrow{c_1} (c_1, \mathbf{v}'_1) \xrightarrow{e_1} (c_2, \mathbf{v}_2) \xrightarrow{c_2} \cdots \xrightarrow{e_m} (c_1, \mathbf{v'})\},$$

where $c_1, c_2, \ldots c_{m-1}, c_1$ and $e_1, e_2, \ldots, e_m$ are respectively the control locations and the control edges successively visited by σ.

In the previous expression of $\sigma(V)$, the constraints over the vectors $\mathbf{v'}$, $\mathbf{v}_1$, $\mathbf{v}'_1$, $\mathbf{v}_2$, …, and $\mathbf{v}'_m$ are all equivalent to conjunctions of linear inequalities. As a consequence, this expression can be rewritten as

$$\sigma(V) = \{(v'_1, \ldots, v'_n) \in \mathbb{R}^n \mid \exists v_1, v_2, \ldots, v_{2mn} \in \mathbb{R} : (v_1, \ldots, v_n) \in V \wedge P(v_1, \ldots, v_{2mn}, v'_1, \ldots, v'_n) \leq \mathbf{q}\},$$

with $m > 0$, $P \in \mathbb{Z}^{m' \times (2m+1)n}$, $\mathbf{q} \in \mathbb{Z}^{m'}$ and $m' > 0$. By carrying out Fourier-Motzkin elimination [Sch86], one can get rid of the quantified variables in this expression, which then takes the form

$$\sigma(V) = \{(v'_1, \ldots, v'_n) \in \mathbb{R}^n \mid \exists (v_1, v_2, \ldots, v_n) \in V : P(v_1, \ldots, v_n, v'_1, \ldots, v'_n) \leq \mathbf{q}\},$$

with $P \in \mathbb{Z}^{m \times 2n}$, $\mathbf{q} \in \mathbb{Z}^m$ and $m > 0$. Such a transformation over real vectors, characterized by the pair $(P, \mathbf{q})$, is called a *Linear Hybrid Transformation (LHT)* [BHJ03].

For every control cycle σ of a Linear Hybrid EHA, one can compute a corresponding LHT $(P_\sigma, \mathbf{q}_\sigma)$. The question is then to check whether the reflexive and transitive closure of such a relation preserves the representability of sets of real vectors by weak deterministic RVA.

This problem has been studied in [BHJ03, BH06], which provide a partial solution in the form of a sufficient criterion, expressed by the following definition and theorem.

Definition 8. An LHT $(P, \mathbf{q})$ with $P \in \mathbb{Z}^{m \times 2n}$, $\mathbf{q} \in \mathbb{Z}^m$ and $n, m > 0$ is *periodic* if its underlying system of linear constraints

$$P \begin{bmatrix} \mathbf{x} \\ \mathbf{x'} \end{bmatrix} \leq \mathbf{q},$$

expressed over the vectors of variables $\mathbf{x}$ and $\mathbf{x'}$, is only composed of constraints of the form $\mathbf{p}.\mathbf{x} \leq q$, $\mathbf{p}.\mathbf{x'} \leq q$ and $\mathbf{p}.(\mathbf{x'} - \mathbf{x}) \leq q$, with $\mathbf{p} \in \mathbb{Z}^n$ and $q \in \mathbb{Z}$.

Theorem 15. *Let $\theta = (P, \mathbf{q})$, with $P \in \mathbb{Z}^{m \times 2n}$, $\mathbf{q} \in \mathbb{Z}^m$ and $n, m > 0$, be a periodic LHT. The reflexive and transitive closure θ^* of θ is such that for every set $V \subseteq \mathbb{R}^n$ representable by a weak deterministic RVA, the set $\theta^*(V)$ is representable as well. There exists an algorithm that computes for any periodic LHT θ a weak deterministic RVA representing $\theta^*(V)$ from one representing V.*

Intuitively, if θ is a periodic LHT, then the linear constraints defining the transformation θ^k, for every $k \in \mathbb{N}$, can be expressed as a function of k. A representation of θ^* is then obtained by quantifying away this additional variable k over the natural numbers, which is feasible within $\langle \mathbb{R}, \mathbb{Z}, +, < \rangle$.

Finally, in Linear Hybrid EHA, it is frequent to find control cycles whose equivalent LHT is not periodic, but whose closure nevertheless preserves the representability of sets of vectors by weak deterministic RVA. In some situations, computing the effect of iterating such LHT can be reduced to computing the closure of a simpler LHT which is itself periodic. A number of reduction rules for performing this operation are introduced in [BHJ03, BH06].

7 Conclusions

In this survey, we have presented a generic approach to computing the set of reachable configurations of an infinite-state system modeled by an extended automaton. This approach is based on a finite-state representation of the sets of data values that are handled during symbolic state-space exploration, and on acceleration operators that speed up exploration by potentially generating infinitely many reachable configurations in finite time.

In Sections 4, 5 and 6, we have shown how this method can be applied to systems relying on asynchronous communication channels, as well as unbounded integer and real variables. These results have been implemented in tools such as LASH [LASH], FAST [BFLP03] and LIRA [LIRA].

We have only studied acceleration algorithms that are specific to a given data domain, and that are obtained by iterating a single control cycle. Other techniques have been developed for accelerating transformations over finite-state representations of sets independently of the data domain [AJNd03, BLW03, AJNS04, Leg11]. For some particular domains, algorithms have also been proposed for accelerating control

structures that are more complex than simple loops [FWW97, CJ98, Fri98]. Finally, the strategy that we have followed for creating cycle meta-transitions consists of first selecting control cycles and then checking whether their iteration preserves the representability of sets of data values. An alternative approach, proposed in [BFLP03], consists in restricting the data transformation that label the edges of the control graph such that every path corresponds to a transformation that can be accelerated.

One of the limitations of finite-state representations of sets is that suitable encodings are only known for simple data domains. An interesting problem is to extend this approach to heterogeneous domains, in order to be able to analyze programs relying on variables with different types. Steps in this direction have already been made, in particular for combining FIFO communication channels with Presburger constraints on the exchanged symbols [BH99].

References

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Proc. 10th International Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer-Verlag, 1998.

[ACABJ04] P. Aziz Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th Symposium on Logic in Computer Science (LICS'90)*, pages 414–425, Philadelphia, June 1990.

[ACH+95] 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*, 138:3–34, 1995.

[AHH93] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proc. 14th annual IEEE Real-Time Systems Symposium*, pages 2–11, 1993.

[AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Inf. and Computation*, 127(2):91–101, June 1996.

[AJNd03] P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proc. 15th International Conference on Computer-Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 237–248, Boulder, USA, 2003. Springer-Verlag.

[AJNS04] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.

[BBL09] B. Boigelot, J. Brusten, and J. Leroux. A generalization of Semenov's theorem to automata over real numbers. In *Proc. 22nd International Conference on Automated Deduction (CADE'09)*, volume 5663 of *Lecture Notes in Computer Science*, pages 469–484. Springer, 2009.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In *Proc. 9th International Conference on Computer-Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 167–177, Haifa, 1997. Springer.

[BC96] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In *Proc. 21st International Colloquium on Trees in Algebra and Programming (CAAP'96)*, number 1059 in Lecture Notes in Computer Science, pages 30–43. Springer-Verlag, 1996.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast acceleration of symbolic transition systems. In *Proc. 15th International Conference on Computer-Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121. Springer, 2003.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. 4th International Symposium on Static Analysis (SAS'97)*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186, Paris, September 1997. Springer-Verlag.

[BH99] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1–2):211–250, 1999.

[BH06] B. Boigelot and F. Herbreteau. The power of hybrid acceleration. In *Proc. 18th International Conference on Computer-Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 2006.

[BHJ03] B. Boigelot, F. Herbreteau, and S. Jodogne. Hybrid acceleration using real vector automata. In *Proc. 15th Int. Conf. on Computer-Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 193–205. Springer-Verlag, 2003.

[BHMV94] 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.

[BJW05] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic*, 6(3):614–633, 2005.

[BL04] B. Boigelot and L. Latour. Counting the solutions of Presburger equations without enumerating them. *Theoretical Computer Science*, 313(1):17–29, 2004.

[BLW03] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. 15th International Conference on Computer-Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, Boulder, USA, July 2003. Springer.

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

[Boi03] B. Boigelot. On iterating linear transformations over recognizable sets of integers. *Theor. Comput. Sci.*, 309(1–3):413–468, 2003.

[Bry92] R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BT11] A. Bouajjani and T. Touili. Widening techniques for regular tree model checking. Special Section on Regular Model Checking STTT, in this volume, 2011.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6th International Conference on Computer-Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67, Stanford, June 1994. Springer-Verlag.

[BZ83] D. Brand and P. Zafiropoulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Methodoloy and Philosophy of Science*, pages 1–12, Stanford, 1962. Stanford University Press.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252, Los Angeles, California, January 1977. ACM Press, New York.

[CFI96] G. Cécé, A. Finkel, and S. P. Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. and Computation*, 124(1):20–31, October 1996.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *Proc. 10th International Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer-Verlag, 1998.

[Cob69] A. Cobham. On the base-dependence of sets of numbers recognizable by finite automata. *Mathematical Systems Theory*, 3:186–192, 1969.

[Dil89] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Automatic Verification Methods for Finite-State Systems*, number 407 in LNCS, pages 197–212. Springer-Verlag, 1989.

[EK08] Jochen Eisinger and Felix Klaedtke. Don't care words with an application to the automata-based approach for real addition. *Formal Methods in System Design*, 33(1–3):85–115, 2008.

[Fri98] L. Fribourg. A closed-form evaluation for extended timed automata. Research Report LSV-98-2, LSV, March 1998.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *Electr. Notes Theor. Comput. Sci.*, 9, 1997.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[Hol91] G. Holtzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[Kla91] N. Klarlund. Progress measures for complementation of omega-automata with applications to temporal logic. In *Proc. 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 358–367, San Juan, Puerto Rico, October 1991. IEEE.

[Kla08] F. Klaedtke. Bounds on the automata size for Presburger arithmetic. *ACM Trans. Comput. Log.*, 9(2), 2008.

[KV05] O. Kupferman and M.Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *Proc. 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440 of *Lecture Notes in Computer Science*, pages 206–221, Edinburgh, April 2005. Springer.

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

[Ld01] C. Löding. Efficient minimization of deterministic weak $\omega-$automata. *Information Processing Letters*, 79(3):105–109, 2001.

[Leg11] A. Legay. Extrapolating (omega-)regular model checking. Special Section on Regular Model Checking STTT, in this volume, 2011.

[LIRA] Linear Integer/Real Arithmetic solver (LIRA). Available at : `http://lira.gforge.avacs.org/`.

[McM93] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Pre29] M. Presburger. Über die Vollständigkeit 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, 1929.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th Symposium on Foundations of Computer Science (FOCS'88)*, pages 319–327. IEEE Computer Society, October 1988.

[Sch86] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, 1986.

[Sem77] A. L. Semenov. Presburgerness of predicates regular in two number systems. *Siberian Mathematical Journal*, 18:289–299, 1977.

[SW74] L. Staiger and K. Wagner. Automatentheoretische und automatenfreie charakterisierungen topologischer klassen regulärer folgenmengen. *Elektron. Informationsverarbeitung und Kybernetik EIK*, 10:379–392, 1974.

[WB95] P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proc. 2nd International Symposium on Static Analysis (SAS'95)*, volume 983 of *LNCS*, pages 21–32. Springer, 1995.

[WB00] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, Germany, March 2000. Springer.

[Wei99] V. Weispfenning. Mixed real-integer linear quantifier elimination. In *ISSAC: Proceedings of the ACM SIGSAM International Symposium on Symbolic and Algebraic Computation*, pages 129–136, Vancouver, July 1999. ACM Press.

[Wil93] T. Wilke. Locally threshold testable languages of infinite words. In *Proc. 10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, volume 665 of *LNCS*, pages 607–616, Würzburg, 1993. Springer.