Verifying Systems with Infinite but Regular State Spaces

Pierre Wolper, Bernard Boigelot*

Université de Liège
Institut Montefiore, B28
B-4000 Liège Sart-Tilman, Belgium
{pw,boigelot}@montefiore.ulg.ac.be

Abstract. Thanks to the development of a number of efficiency enhancing techniques, state-space exploration based verification, and in particular model checking, has been quite successful for finite-state systems. This has prompted efforts to apply a similar approach to systems with infinite state spaces. Doing so amounts to developing algorithms for computing a symbolic representation of the infinite state space, as opposed to requiring the user to characterize the state space by assertions. Of course, in most cases, this can only be done at the cost of forgoing any general guarantee of success. The goal of this paper is to survey a number of results in this area and to show that a surprisingly common characteristic of the systems that can be analyzed with this approach is that their state space can be represented as a regular language.

1 Introduction

If a system is finite-state, its set of reachable states can, at least in theory, always be computed. The sometimes "theoretical" nature of this possibility comes from the fact that, even for simple systems, finite state spaces can be much too large to be computed with any realistic amount of resources. This is not surprising since, for instance, state reachability for a concurrent system is a PSPACE-complete problem. In spite of such rather discouraging complexity results, much effort has been devoted to making state-space exploration practically feasible. These efforts have been quite successful and techniques such as symbolic verification [BCM+92] or partial-order methods [Val92, WG93] are quite effective and tools based upon them are in regular use.

For infinite-state systems, even the theoretical possibility of exploring the state space disappears. Indeed, except for severely restricted classes of systems, most problems about reachable states become undecidable. This has long been taken as evidence that infinite-state systems had to be handled by "assertional" methods in which the user is requested to characterize the system behavior by logical assertions, the validity of which is then established by a formal proof. However, undecidability only excludes perfectly general algorithmic solutions,

* "Chargé de Recherches" (Post-Doctoral Researcher) for the National Fund for Scientific Research (Belgium).

not solutions that work on restricted cases or for which termination is not guaranteed. Note that this situation is to some extent similar to the one for finite-state systems. Indeed, in the latter case high complexity excludes always efficient algorithmic solutions, but years of experimental work have confirmed the existence of solutions that work perfectly well on many practically relevant instances.

Work on the algorithmic verification of infinite-state systems has thus proceeded in two directions. The first is the study of classes of infinite-state systems that are decidable, e.g. [BS95, FWW97]. In general such classes are somewhat artificial and practical examples of systems that fall within them are hard to find. There are fortunately exceptions to this rule, for instance timed automata [AD94], which have been used as the basis of exploited verification tools. The second direction is to consider a larger class of systems, but to be satisfied with a semi-algorithmic solution, i.e., an algorithmic solution that is allowed to give up or run forever on some instances.

In this paper, we will consider examples of both categories in the context of closed systems whose infinite state space originates from the possibility of executing arbitrarily long computations with unbounded data. The focus on closed systems is typical of many verification approaches and allows us to use a simple semantical model. Furthermore, by eliminating the possibility of reading arbitrarily large values, it restricts the source of the infinite number of states to arbitrarily long computations. We will also limit our focus to the problem of computing a representation of the set of reachable states of the system. Indeed, this allows the verification of many properties of the state space such as reachability of a given state or truth of an invariant. Furthermore, once the reachable states can be computed, other verification problems can often also be solved. For instance, model checking for linear-time safety properties reduces to reachability, and model checking of general linear-time temporal properties reduces to repeated reachability [VW86].

The first class of systems we will consider are finite-state systems with one pushdown stack. This is a decidable class for which the state space can always be algorithmically computed. Concretely, we will show that a finite-automaton representation of reachable states (control state and stack content) can simply and easily be computed [FWW97]. Next, we will turn to finite-state systems communicating through unbounded message queues. This is an undecidable class and thus only semi-algorithmic solutions are possible. The approach we will consider represents queue contents by finite automata and focuses on cycles in the control graph in order to finitely generate infinite state spaces [BW94, BG96, BGWW97].

The last class of systems we will consider is that of finite-state systems augmented with a number of integer variables. The traditional way to represent sets of integer values is to use arithmetic constraints. Here, we will turn to an alternative representation with potential computational advantages: finite automata operating on the binary (or in general base-r) encodings of integer vectors. This representation is as expressive as Presburger arithmetic, but is much more computationally oriented, just like BDDs [Bry92] are a computationally-oriented representation of Boolean functions. With this representation, the effect of repeating a cycle of linear operations can often be computed and, furthermore, a natural characterization of cycles for which this computation is possible has been given [Boi98].

2 Modeling Infinite-State Systems

We consider systems that can be modeled as *extended automata*, i.e., state machines with a finite control and possibly infinite data. In most cases, an extended automaton will be obtained from a higher level representation of the system, for instance a concurrent program. The class of systems that can be modeled by extended automata is thus quite large.

An extended automaton A is a tuple $(C, c_0, M, m_0, Op, \Delta)$, where

- C is a finite set of *control locations*;
- M is a (possibly infinite) *memory domain*;
- $Op \subseteq 2^{M \to M}$ is a set of *memory operations*;
- $\Delta \subseteq C \times Op \times C$ is a finite set of *transitions*;
- c_0 is an *initial control location*, and m_0 is an *initial memory content*.

A *state* of A is an element of $C \times M$, i.e., a pair (c, m) composed of a control location c and a memory content m. The initial state is (c_0, m_0). A state (c', m') is *directly reachable* from a state (c, m) if there exists a transition $(c_1, \theta, c_2) \in \Delta$ such that $c_1 = c$, $c_2 = c'$ and $m' = \theta(m)$. For this, we use the notation $(c, m) \Rightarrow (c', m')$. Furthermore, we denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. A state (c', m') is *reachable from* a state (c, m) if $(c, m) \Rightarrow^* (c', m')$. The *reachable states* of A are those that are reachable from the initial state.

The main problem we will address in subsequent sections is that of computing a representation of the set of reachable states of various classes of extended automata.

3 Pushdown Systems

A pushdown system is a system composed of a finite control associated with an unbounded stack over a finite alphabet Σ. Such a system can be modeled by a *pushdown automaton*, i.e., an extended automaton $A = (C, c_0, M, m_0, Op, \Delta)$ whose memory domain $M = \Sigma^*$ is the set of all the potential *stack contents*, and whose set of memory operations Op contains the two *stack operations* a_+ and a_- for every $a \in \Sigma$. These operations are defined by $a_+(w) = wa$ and $a_-(wa) = w$ for every $w \in \Sigma^*$ (the value of $a_-(w)$ is not defined if w does not end with the symbol a).

It is known (see for instance [Cau92]) that the set of reachable states of a pushdown automaton is regular, or more precisely, that for each control location $c \in C$, the memory contents $m \in M$ for which (c, m) is a reachable state form a regular set. In [FWW97], a very simple construction of a finite automaton accepting the possible stack contents for each control location is given and is shown to be implementable in $O(n^3)$, n being the size of the pushdown automaton. This construction is the following. Given a stack alphabet Σ and a pushdown automaton $A = (C, c_0, M, m_0, Op, \Delta)$ over Σ, one constructs the *reachability automaton* of A, which is the finite-state automaton $A_r = (Q_r, \Sigma_r, \Delta_r, q_r^0, F_r)$ such that

- The set of states Q_r is identical to C;
- The input alphabet Σ_r is identical to Σ;
- The transition relation $\Delta_r \subseteq Q_r \times (\Sigma_r \cup \{\varepsilon\}) \times Q_r$ is the smallest relation that satisfies the following conditions, where Δ_r^* denotes the reflexive and transitive closure of Δ_r, and ε denotes the empty word:
  - If $(q, a_+, q') \in \Delta$, then $(q, a, q') \in \Delta_r$, and
  - If $(q, a_-, q') \in \Delta$ and $(q'', a, q) \in \Delta_r^*$, then $(q'', \varepsilon, q') \in \Delta_r$;
- The initial state q_r^0 is c_0;
- All the states are accepting, i.e., we have $F_r = Q_r$.

The relation between A and A_r is given by the following theorem.

Theorem 1. *A state (q, w) is reachable in a pushdown automaton A if and only if the state q is reachable in the reachability automaton A_r through the word w.*

In other words, the stack contents with which a control location c is reachable are exactly the words accepted by A_r when c is taken as the unique accepting state. It follows that A_r represents exactly and effectively the set of reachable states of A. The automaton A_r can then be used to check properties of the system than are reducible to reachability properties. Furthermore, in [FWW97] it is shown how repeated reachability and hence temporal logic model checking can be handled with related constructions.

4 Queue Systems

A queue system is a system composed of a finite control together with one or several unbounded FIFO channels (also called queues) containing elements of finite alphabets. It is a very common model of distributed systems communicating through unbounded queues. Such a system can be modeled by a *queue automaton*, which is an extended automaton $A = (C, c_0, M, m_0, Op, \Delta)$ satisfying the following.

- The memory domain M is of the form $\Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, where $n > 0$ represents the *number of queues* of A, and each Σ_i is the finite *queue alphabet* of the i-th queue of A (this queue is usually denoted q_i). For simplicity, we assume that the different queue alphabets are distinct. Each element $(w_1, \ldots, w_n)$ of M associates a content w_i to each queue q_i of the system and is called a *queue-set content*.
- The set of memory operations Op contains the two *queue operations* $q_i!a$ and $q_i?a$ for each queue q_i and symbol $a \in \Sigma_i$. The *send* operation $q_i!a$ is defined by $(q_i!a)(w_1, \ldots, w_n) = (w_1, \ldots, w_{i-1}, w_i a, w_{i+1}, \ldots, w_n)$. The *receive* operation $q_i?a$ is defined by $(q_i?a)(w_1, \ldots, w_{i-1}, a w_i, w_{i+1}, \ldots, w_n) = (w_1, \ldots, w_n)$ (this operation is not defined if the content of q_i does not start with the symbol a).

Unsurprisingly, computing the set of reachable states of a queue automaton, or more precisely, a finite and effective representation of this set, is in general impossible. It is indeed well known that queue automata for which there is more than one symbol in at least one queue alphabet can simulate arbitrary Turing machines.

This does not, however, exclude partial algorithmic approaches to computing the set of reachable states of queue systems. One such approach relies upon the concept of *meta-transition* introduced in [BW94] and applied to queue systems in [BG96]. A meta-transition is a derived transition that in one step generates a potentially infinite set of states. Precisely, a meta-transition is a triple (c, f, c'), where $c, c' \in C$ are the *origin* and the *destination* locations and $f : M \rightarrow 2^M$ is the *memory function* of the meta-transition. The memory function must be such that, for every set $U \subseteq C \times M$ of reachable states of A, its *image* $U' = \{(c', m') \mid (\exists m \in M)((c, m) \in U \wedge m' \in f(m))\}$ by the meta-transition only contains reachable states. A particular class of meta-transitions are the *cycle* meta-transitions. These correspond to the arbitrarily repeated execution of a given cycle in the control graph (C, Δ) of A. The origin and destination locations of a cycle meta-transition are thus the origin of the cycle. Its memory function maps any memory content m to the set of values that can be obtained by applying any number of times the sequence of operations σ labeling the cycle. In other words, the memory function of a cycle transition computes the image of memory contents by the closure σ^* of the sequence of operations of the cycle.

To compute the reachable states of a queue automaton, one can thus proceed by augmenting the automaton with a finite set of meta-transitions and then exploring the state space of the augmented automaton. By the definition of meta-transitions, this state space is guaranteed to be identical to the one of the original automaton. While exploring the augmented automaton, one follows both transitions and meta-transitions, each time expanding the set of known reachable states. The search terminates when a stable set is obtained. Of course, there is no guarantee that this will eventually happen, but the fact that a meta-transition can produce in one step an infinite number of states makes termination possible, even when the number of reachable states is infinite.

Applying this method requires the ability to represent possibly infinite sets of states, and to perform operations on represented sets. Since queue automata have a finite control, a simple idea consists of associating to each control location a set of corresponding queue-set contents represented with the help of a specific representation system. The *Queue Decision Diagram, or QDD* [BG96], is such a symbolic representation system. It relies on an *encoding scheme* which maps every queue-set content $(w_1, \ldots, w_n)$ onto the concatenation $w_1 \cdot w_2 \cdots w_n$ of the individual queue contents. Given a set $U \subseteq M$ of queue-set contents, a QDD A representing U is simply a finite-state automaton accepting all the encodings of the elements of U.

Of course, QDDs cannot represent all the subsets of M. The following theorem, which appears in [BGWW97], characterizes exactly the sets of queue-set contents that can be represented by a QDD.

Theorem 2. *A set $U \subseteq M$ is representable by a QDD if and only if it can be expressed as a finite union of Cartesian products of regular languages over the queue alphabets.*

A positive point of QDDs is that they can easily be manipulated algorithmically. First, computing the union, intersection, complement and difference of sets represented as QDDs simply amounts to performing the corresponding operation over finite-state automata. This is a consequence of the fact that the encoding scheme that has been chosen maps every queue-set content onto a unique and unambiguous word over $\Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^*$. Second, it has been shown in [BG96] that one can always compute the effect of a transition of a queue automaton on a set represented as a QDD:

Theorem 3. *Let $U \subseteq M$ be a set represented by a QDD. Given a queue q_i and a symbol $a \in \Sigma_i$, one can compute QDDs representing the sets $(q_i!a)(U)$ and $(q_i?a)(U)$.*

In order to compute the set of reachable states of a queue automaton with the help of QDDs, one must also be able to add meta-transitions to the automaton. Selecting cycles that are suitable for meta-transitions can be done thanks to the following result, which appears in [BGWW97] and is proved in [Boi98].

Theorem 4. *Given a queue automaton A and a cycle in its control graph (C, Δ) labeled by the sequence of operations $\sigma \in Op^*$, it is decidable whether the closure σ^* preserves the representable nature of sets of queue-set contents, i.e., whether $\sigma^*(U)$ is always representable by a QDD whenever U is representable by a QDD.*

Note that in particular, the necessary and sufficient condition presented in [BGWW97] implies that for every sequence σ in which the queue operation only involves a single queue, σ^* always preserves the representability of sets of queue-set contents.

The fact that a meta-transition preserves representability is not sufficient. One also needs to be able to effectively compute its effect. The required result is given by the following theorem.

Theorem 5. *If $\sigma \in Op^*$ is a sequence of queue operations such that $\sigma^*(U)$ is representable for every representable set $U \subseteq M$, then one can compute a QDD representing $\sigma^*(U)$ given a QDD representing U.*

An algorithm implementing this computation is presented in [Boi98].

5 Linear Integer Systems

A linear integer system is a system composed of a finite control together with one or several unbounded integer variables on which linear operations are performed. Such a system can be modeled by a *linear integer automaton*, which is an extended automaton $A = (C, c_0, M, m_0, Op, \Delta)$ satisfying the following.

– Its memory domain M is $\mathbf{Z}^n$, where $n > 0$ represents the *number of variables* of A (these variables are usually denoted $x_1, x_2, \ldots, x_n$). Each element $(v_1, \ldots, v_n)$ of M is called a *variable-set content* and associates one value v_i to each variable x_i of the system.
– Its set of memory operations Op contains all functions $M \to M$ of the form $P\mathbf{x} \leq \mathbf{q} \to \mathbf{x} := T\mathbf{x} + \mathbf{b}$, where $P \in \mathbf{Z}^{m\times n}, \mathbf{q} \in \mathbf{Z}^m, m \in \mathbf{N}, T \in \mathbf{Z}^{n\times n}$ and $\mathbf{b} \in \mathbf{Z}^n$. The linear system $P\mathbf{x} \leq \mathbf{q}$ is the *guard* of the operation and expresses a condition that must be satisfied by the variable vector $\mathbf{x} = (x_1, \ldots, x_n)$ for the operation to be defined. The linear transformation $\mathbf{x} := T\mathbf{x} + \mathbf{b}$ is the *assignment* of the operation and expresses the transformation undergone by the variable values when the operation is performed.

It is well known that linear integer systems that have at least two variables can simulate two-counter machines and are therefore as expressive as Turing machines. As a consequence, one cannot in general compute the set of reachable states of a linear integer automaton.

One can however follow the same semi-algorithmic approach as in Section 4, adding meta-transitions to the system and then exploring the resulting augmented linear integer automaton. This requires the ability to represent possibly infinite subsets of $\mathbf{Z}^n$, to apply linear operations to represented sets (for computing the effect of a transition), and to apply the repetition of linear operations to represented sets (for computing the effect of meta-transitions).

There are many ways of representing sets of integer vectors, but we will adopt an automaton-based representation that is far from new since it can be found in [Büc60], but has only recently been investigated as a potentially usable representation [WB95, BC96, BBR97, BRW98]. It consists of representing the elements of the vector in binary (or some other base) and then viewing the result as a word. Sets of vectors thus become languages and can be recognized by finite automata.

Precisely, given an integer vector $\mathbf{v} = (v_1, \ldots, v_n)$ and an integer *base* $r > 1$, one encodes each positive component v_i as a finite word $a_{p-1}a_{p-2}\cdots a_0$ over the alphabet $\{0, \ldots, r-1\}$, such that $v_i = \sum_{i=0}^{p-1} a_i r^i$. If $v_i < 0$, then the encoding of v_i consists of the last p digits of the encoding of $r^p + v_i$ (the number $r^p + v_i$ is called the *r's complement* of v_i). The number of digits p is not fixed, but chosen identical for each v_i and such that $-r^{p-1} \leq v_i < r^{p-1}$. An encoding of $\mathbf{v}$ is then obtained by grouping together the digits that share the same position in the encodings of the v_i. The encoding of $\mathbf{v}$ can be viewed either as a tuple of words of identical length over the alphabet $\{0, \ldots, r-1\}$, or as a single word over the alphabet $\{0, \ldots, r-1\}^n$. The latter corresponds to simultaneously reading all digits at a given position and is the one we adopt. Every vector in $\mathbf{Z}^n$ has an infinite number of possible encodings, the length of the shortest being determined by the component with the highest magnitude.

Example 1. A representation of the vector $(3, -1)$ in base 2 is $(0011, 1111)$ or $(0,1)(0,1)(1,1)(1,1)$.

Given a set $U \subseteq \mathbf{Z}^n$ of vectors, a *Number Decision Diagram, or NDD* [WB95, Boi98] representing U is simply a finite-state automaton accepting all the encodings of all the elements of U. The following results, due to Büchi [Büc60], Cobham [Cob69] and Semenov [Sem77], characterize precisely the sets of integer vectors that can be represented by NDDs:

Theorem 6. *A set $U \subseteq \mathbf{Z}^n$ is representable by an NDD in a base $r > 1$ if and only if it can be defined in the first-order theory $\langle \mathbf{Z}, +, \leq, V_r \rangle$, where V_r is a function that maps every nonzero integer onto the highest power of r dividing it.*

Theorem 7. *A set $U \subseteq \mathbf{Z}^n$ is representable by an NDD in any base if and only if it can be defined in Presburger arithmetic, i.e., in the first-order theory $\langle \mathbf{Z}, +, \leq \rangle$.*

An important corollary of Theorem 7 is that any vector transformation that can be expressed in Presburger arithmetic can be applied to sets of integer vectors represented as NDDs. In particular, one can always compute the image of such a set by any linear operation that belongs to Op. It is thus possible to compute the effect of a transition of a linear integer automaton on a set of variable-set values represented as an NDD.

In order to compute the set of reachable states of a linear integer automaton with the help of NDDs, one must also be able to add meta-transitions to the automaton. Selecting cycles that are suitable for meta-transitions can be done thanks to the following result, which is proved in [Boi98].

Theorem 8. *Given a sequence $\sigma \in Op^*$ of linear integer operations without guards, it is decidable whether the closure σ^* preserves the representable nature of sets of integer vectors. Moreover, if σ^* preserves the representable nature of sets of integer vectors, then any sequence σ' obtained by adding guards to the operations composing σ is such that $(\sigma')^*$ preserves the representable nature of sets of integer vectors.*

Very roughly, the criterion under which a sequence of linear operations preserves representability is that the eigenvalues of the matrix corresponding to the transformation defined by the sequence are same-order roots of a power of the base used for the representation.

Computing the effect of a meta-transition on a represented set of variable-set values can be done thanks to the following result, which is fully developed in [Boi98].

Theorem 9. *If $\sigma' \in Op^*$ is a sequence of linear integer operations such that the corresponding guardless linear integer operation σ has a closure that preserves the representable nature of sets of integer vectors, then one can compute an NDD representing $(\sigma')^*(U)$ from σ' and an NDD representing $U \subseteq \mathbf{Z}^n$.*

6 Conclusions

For many years, there has been a dichotomy in verification approaches: algorithmic methods for finite-state systems, proof-based methods for infinite-state systems. This dichotomy has not been absolute, but when algorithmic methods have been proposed for infinite-state systems, it has usually been for restricted classes for which most problems are decidable. For instance, much research has been devoted to Petri nets, which sit interestingly close to the limit of decidability [EN94].

Most of the results presented in this paper are linked to a different starting point: consider undecidable classes, but be satisfied with partial algorithmic solutions. There are two strong reasons for doing so. The first is that for a verification approach to be usable in an industrial setting it has to be supported by tools that do most of the work. Hence, methods that at least attempt to provide results without user intervention are essential. The second reason is that there is little practical benefit from focusing on decidable classes of systems. Indeed, the high complexity of all meaningful verification problems has as consequence that, even for perfectly decidable classes, solutions are anyway only partial from a practical point of view. No verification tool is guaranteed to succeed on any but the most trivial instances and, often, the only way to know if a tool can handle a particular instance is to run the tool. Since the ideas presented in this paper lead to tools for infinite-state systems with a perfectly similar behavior, there is no reason to, a priori, doubt their acceptability for practical use. The determining factor will be how often they succeed on the program instances for which verification is indeed needed.

Another central theme of this paper is the importance of well-adapted representation systems for sets of values. In the finite-state case, BDDs have provided a substantial boost to the success of verification methods. The intuition underlying this paper is that representation methods with similar characteristics will be crucial to the success of verification techniques for infinite-state systems. With respect to this, the finite automaton, which has already proven its use for developing finite-state verification algorithms [VW86, BVW94], though probably not the ultimate solution, might again be a very fruitful starting point.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.

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

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

[BCM+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. *Information and Computation*, 98(2):142–170, June 1992.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Proceedings of Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12, New-Brunswick, NJ, USA, July 1996. Springer-Verlag.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDD's. In *Proc. of Int. Static Analysis Symposium*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186, Paris, September 1997. Springer-Verlag.

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

[BRW98] B. Boigelot, S. Rassart, and P. Wolper. On the expressiveness of real and integer arithmetic automata. to appear in Proc. ICALP'98, 1998.

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

[BS95] O. Burkart and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.

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

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, California, June 1994. Springer-Verlag. full version available from authors.

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

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

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

[EN94] J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *Bulletin of the EATCS*, 52:245–262, 1994.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). Presented at Infinity'97 (Bologna), Electronic notes in theoretical computer science, August 1997.

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

[Val92] A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1:297–322, 1992.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

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

[WG93] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Proc. CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246, Hildesheim, August 1993. Springer-Verlag.