Symbolic Verification with Periodic Sets*

Bernard Boigelot** and Pierre Wolper

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

Abstract. Symbolic approaches attack the state explosion problem by introducing implicit representations that allow the simultaneous manipulation of large sets of states. The most commonly used representation in this context is the Binary Decision Diagram (BDD). This paper takes the point of view that other structures than BDD's can be useful for representing sets of values, and that combining implicit and explicit representations can be fruitful. It introduces a representation of complex periodic sets of integer values, shows how this representation can be manipulated, and describes its application to the state-space exploration of protocols. Preliminary experimental results indicate that the method can dramatically reduce the resources required for state-space exploration.

1 Introduction

Verification by state-space exploration is an old technique [Wes78] whose many advantages have long been outweighed by its main drawback: the state explosion problem. However, in recent years, this central problem has been attacked from several directions with sufficient success to give, if not the hope of total victory, at least the possibility of containment in some contexts. Two main tactics have been used against the state explosion problem. The first is to limit the search to a reduced state-space that is still sufficient for verifying the property of interest. Among these, one can cite on-the-fly approaches [Hol85, VW86, JJ89, FM91, CVWY92], partial-order methods [Val90, God90, HGP92, McM92, GW93, Pel93, WG93], and abstraction techniques [GL93]. The second tactic avoids handling each state individually by using symbolic representations. This makes it possible to manipulate very large sets of states simultaneously [BCM$^+$90, CMB90].

The main representation of sets of states that has been used in symbolic verification is the Binary Decision Diagram (BDD) [Bry92]. While this representation is simple and general, and can be extremely effective [BCM$^+$90], it is not a panacea. Indeed, for fundamental reasons, not all sets of states can be

* This work was supported by the Esprit BRA action REACT and by the Belgian Incentive Program "Information Technology" - Computer Science of the future, initiated by the Belgian State - Prime Minister's Office - Science Policy Office. The scientific responsibility is assumed by its authors.

** "Aspirant" (Research Assistant) for the National Fund for Scientific Research (Belgium).

represented by small BDD's. Whether BDD's will be effective or not depends on the nature and structure of the sets of states that have to be represented. Symptomatically of this, there are more success stories about the use of BDD's for hardware verification than for other applications such as protocol verification.

Another family of verification approaches that can broadly be classified as symbolic are those that have been developed for the verification of real-time properties [ACD93]. Indeed, in these approaches, sets of time values are represented with the help of polyhedra rather than explicitly. However, as opposed to what is done with BDD's the rest of the state information is represented explicitly rather than symbolically.

The work we present here is based on a similar intuition: it can be fruitful to represent states partly explicitly and partly symbolically. The question then is: which symbolic representation can be used effectively in such a combined approach?

This paper attempts to give an answer to this question in a specific context: protocols using integer variables. Indeed, when analyzing protocols, it often turns out that the state space explodes due to the presence of integer variables used for instance as counters. Even though this cause of state explosion is often not inherent to the protocol design, it is resistant to existing techniques. However, a close look at the sets of values taken by these variables reveals that they often are periodic or projected from a periodic set. Based on this motivation, we introduce a representation of periodic subsets of $\mathbf{Z}^n$ (which we name *periodic vector sets*) as a tool to be used in protocol verification.

The representation we propose is derived fairly naturally from the main source of periodicity in sets of reachable integer values: the iteration of linear transformations. It allows the representation of finite and infinite sets and is based on a few simple concepts: linear combinations, linear constraints, and projection. We show that it is closed under the application of iterated linear transformations. Moreover, we establish that it can also represent the composition of transformations, as well as their iterations, though not always their nested iterations.

Next, we turn to the use of our new representation in the context of the exploration of the state-space of protocols. There are several approaches to using a representation of periodic vector sets in this context. We present one that is based on the selective precomputation of the fixpoint of some program transitions. These fixpoints are then added as generalized transitions to the program, and a traditional search of this modified program is then performed with the help of our representation of periodic sets. This can dramatically reduce the resources needed for the state-space search compared to a simple enumerative exploration. Moreover, state reachability questions can still be fully answered, which makes it possible to use the method for the verification of a large class of properties [CVWY92].

Finally, we present some experimental results that were obtained with a preliminary implementation, compare our method to existing work, and discuss its benefits and limits.

2 Defining Periodic Vector Sets

Consider the simple program represented in Figure 1, where `x` is an integer variable. Although its number of *control* states is limited to four, it is easily seen that

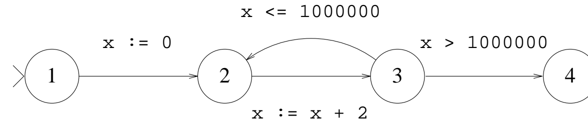

Fig. 1. A simple program.

its state-space contains more than one million reachable states. However, this does not make it impractical to deal with this set of states: it is straightforwardly represented as follows

$$S = \{(①, \bot), (②, 2k), (③, 2k+2), (④, 1000002) \,|\, k \in \mathbf{N} \wedge 0 \leq k \leq 500000\}$$

Moreover, this "symbolic" description of the set of reachable states can be used to validate simple properties of the state-space, for instance the reachability of $(③, 456)$.

Intuitively, the reason for which the set of reachable states of the program of Figure 1 can be easily represented is that it has a *periodic* structure. Such periodic sets of reachable states are due to the repeated execution of instructions forming a *cycle* in the control graph. For instance, the set $\{2k \,|\, k \in \mathbf{N} \wedge 0 \leq k \leq 500000\}$ of reachable values at control state ② may be seen as the result of multiple executions of the cycle ②–③–②, starting with the initial reachable value 0.

Our goal is to provide a general symbolic representation system for sets of values such as those appearing in the example above. This system has to be able to represent single values as well as periodic sets resulting from cyclic executions, and it should allow elementary operations to be easily applied to the representations. The requirements on the representation and on the operations it supports are linked to the operations that are allowed in the programs to be analyzed. Let us thus briefly define the programming formalism within which we will work.

We consider extended finite-state machines with unbounded integer variables on which the only allowed operations are constant assignment (`x := k`), adding a constant to a variable (`x := x + k`), and testing linear equalities (`x <= 2y + z - 1`). In other words, each transition between control states can be labeled with a normalized instruction of the form $T\mathbf{x} \leq \mathbf{u} \rightarrow \mathbf{x} := A\mathbf{x} + \mathbf{b}$, where each component of the vector $\mathbf{x}$ is a variable, $T\mathbf{x} \leq \mathbf{u}$ is the set of linear inequalities serving as precondition to the transition and $\mathbf{x} := A\mathbf{x} + \mathbf{b}$ is the linear transformation associated with the transition. A is a diagonal matrix whose non-zero elements are equal to 1 and $\mathbf{b}$ is an integer vector.

Such normalized instructions have nice properties. First, it is always possible to express the *composition*[3] of two instructions as a normalized instruction. Hence, the body of any cycle in the control graph is always equivalent to a single instruction whose representation may easily be computed by successive compositions. Second, given that the matrix A is idempotent ($A^2 = A$), executing such an instruction repeatedly always leads to a periodic set[4]. Indeed, for a vector $\mathbf{x}_0$ of initial values, the sequence of values obtained by the repeated execution of a normalized instruction is

$$A\mathbf{x}_0 + \mathbf{b},\ A\mathbf{x}_0 + A\mathbf{b} + \mathbf{b},\ A\mathbf{x}_0 + 2A\mathbf{b} + \mathbf{b},\ \ldots, A\mathbf{x}_0 + kA\mathbf{b} + \mathbf{b},\ \ldots, \qquad (1)$$

which is periodic with period $A\mathbf{b}$ (the *periodicity vector* of the set). The constant k appearing in an element $A\mathbf{x}_0 + kA\mathbf{b} + \mathbf{b}$ of the set is called the *repetition count* of that element. Notice that only a subset of (1) is generally reached. Indeed, the iteration process only proceeds up to repetition counts for which $A\mathbf{x}_0 + kA\mathbf{b} + \mathbf{b}$ satisfies the precondition $T\mathbf{x} \leq \mathbf{u}$. Notice that this last condition can be written as a set of linear inequalities to be satisfied by k: $TA\mathbf{x}_0 + kTA\mathbf{b} + T\mathbf{b} \leq \mathbf{u}$. Generalizing this to multiple periodicity vectors and repetition counts, we obtain our definition of a periodic vector set.

Definition 1. A *periodic vector set* is a set of vectors $\mathbf{x} \in \mathbf{Z}^n$ such that

$$\exists\, \mathbf{k} \in \mathbf{Z}^m \ :\ \mathbf{x} = C\mathbf{k} + \mathbf{d} \ \wedge\ P\mathbf{k} \leq \mathbf{q}.$$

In this definition, m periodicity vectors are grouped in the matrix C, and $\mathbf{k}$ gathers the corresponding repetition counts. Similarly, the linear equalities bounding the repetition are gathered into a single linear system $P\mathbf{k} \leq \mathbf{q}$. To represent a periodic vector set, one needs to represent the values of m, C and $\mathbf{d}$ as well as the linear system $P\mathbf{k} \leq \mathbf{q}$. For the latter, it is often useful for efficiency reasons to use representations other than the direct syntactic one.

The set of solutions of a linear system of the form $P\mathbf{k} \leq \mathbf{q}$ is a *closed convex polyhedron* (or polyhedron, for short) which can be represented as follows [CH78, Hal93]. It is the set of points $\mathbf{v}$ satisfying

$$\mathbf{v} = \sum_{i=1}^{\sigma}(\lambda_i \mathbf{s}_i) + \sum_{j=1}^{\rho}(\mu_j \mathbf{r}_j) + \sum_{k=1}^{\delta}(\nu_k \mathbf{d}_k)$$

with the constraints

$$\begin{cases} 0 \leq \lambda_i & i = 1, 2, \ldots, \sigma \\ \sum_{i=1}^{\sigma} \lambda_i = 1 & \\ 0 \leq \mu_j & j = 1, 2, \ldots, \rho \end{cases}$$

[3] The *composition* of two instructions is the instruction equivalent to their sequential execution.

[4] Actually, it is possible to extend the set of operations allowed in instructions as long as this leads to an idempotent matrix A. Going further in this direction, one can allow any linear assignment in instructions, and then apply the method we describe only to transitions for which A is idempotent.

where $V = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_\sigma\}$ is the set of *vertices*, $R = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_\rho\}$ is the set of *rays*, and $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_\delta\}$ is the set of *lines* of the polyhedron. Thus a polyhedron is entirely characterized by the sets V, R, and D. In practice, we maintain both the direct representation of the system of linear inequalities as well as the sets V, R, and D characterizing the corresponding polyhedron. This allows us to choose the most convenient representation for each operation that has to be performed.

3 Operations on Periodic Vector Sets

The use of periodic vector sets in verification involves applying a number of operations to these sets. In this section, we describe the required operations as well as the corresponding algorithms.

3.1 Execution of a Single Instruction

Given the representation of a periodic vector set S and a normalized instruction I, our goal is to compute the representation of the set of values that is obtained by applying I to each element $\mathbf{x}$ of S, i.e. the set $S' = \{I(\mathbf{x}) : \mathbf{x} \in S\}$. Recall that S is the set of all the points $\mathbf{x}$ satisfying

$$\exists \mathbf{k} \in \mathbf{Z}^m : \mathbf{x} = C\mathbf{k} + \mathbf{d} \wedge P\mathbf{k} \leq \mathbf{q}.$$

Consequently, the image of S obtained by applying the instruction $T\mathbf{x} \leq \mathbf{u} \rightarrow \mathbf{x} := A\mathbf{x} + \mathbf{b}$ is the set of all $\mathbf{x}'$ such that

$$\exists \mathbf{k} \in \mathbf{Z}^m : \mathbf{x}' = AC\mathbf{k} + (A\mathbf{d} + \mathbf{b}) \wedge T(C\mathbf{k} + \mathbf{d}) \leq \mathbf{u} \wedge P\mathbf{k} \leq \mathbf{q}.$$

Each element $\mathbf{x}'$ of S' thus satisfies

$$\exists \mathbf{k} \in \mathbf{Z}^m : \mathbf{x}' = C'\mathbf{k} + \mathbf{d}' \wedge P'\mathbf{k} \leq \mathbf{q}',$$

with $C' = AC$, $\mathbf{d}' = A\mathbf{d} + \mathbf{b}$, and where the linear system $P'\mathbf{k} \leq \mathbf{q}'$ is the conjunction of the systems $TC\mathbf{k} \leq \mathbf{u} - T\mathbf{d}$ and $P\mathbf{k} \leq \mathbf{q}$. There are known effective algorithms for computing the intersection of the corresponding polyhedra described by their vertices, rays and lines [CH78, Hal93, LeV92].

3.2 Repeated Execution of an Instruction

The purpose of this operation is to compute the set S' of all values resulting from executing one or more times a normalized instruction I on a given periodic vector set S. In other words, we have $S' = I^+(S)$, where $I^+(X)$ denotes the infinite union $I(X) \cup I^2(X) \cup I^3(X) \cup \cdots$. The set S' can be viewed as the least fixpoint containing $I(S)$ of $f(X) = X \cup I(X)$.

If $A\mathbf{x}+\mathbf{b}$ is the linear transformation of I and A is idempotent, the elements of S' are of the form $\mathbf{x}' = A\mathbf{x} + kA\mathbf{b} + \mathbf{b}$ for $\mathbf{x} \in S$ (cf Equation 1 in Section 2). Now, for a given $\mathbf{x} \in S$, such an element is only in S' for values of k (the

repetition count) such that the enabling condition $T\mathbf{x} \le \mathbf{u}$ of the instruction is satisfied before each execution of the instruction, in other words if

$$T\mathbf{x} \le \mathbf{u} \wedge \forall i \in \{0, 1, \ldots, k-1\} : T(A\mathbf{x} + iA\mathbf{b} + \mathbf{b}) \le \mathbf{u}.$$

It can easily be seen that, given that we are dealing with convex sets of constraints, this expression can be rewritten as

$$\begin{aligned} &(k = 0 \wedge T\mathbf{x} \le \mathbf{u}) \\ \vee\ &(k \ge 0 \wedge T\mathbf{x} \le \mathbf{u} \wedge T(A\mathbf{x} + \mathbf{b}) \le \mathbf{u} \wedge T(A\mathbf{x} + (k-1)A\mathbf{b} + \mathbf{b}) \le \mathbf{u}) \end{aligned}$$

Since this expression is disjunctive, it cannot in general be represented by a closed convex polyhedron. To avoid this problem, we notice that the term $(k = 0 \wedge T\mathbf{x} \le \mathbf{u})$ is needed only to ensure that all values resulting from a single execution of the instruction I are represented. So, dropping this term amounts at most to dropping part of $I(S)$, i.e. of computing a set S' such that $II^+(S) \subset S' \subset I^+(S)$. This is not a real problem since the whole of $I^+(S)$ can always be obtained by computing separately $I(S)$ and our approximation S'.

Recalling that the periodic vector set S is the set of all $\mathbf{x}$ satisfying $\exists\, \mathbf{k} \in \mathbf{Z}^m : \mathbf{x} = C\mathbf{k} + \mathbf{d} \wedge P\mathbf{k} \le \mathbf{q}$, the set S' we are computing can be expressed as the set of all $\mathbf{x}'$ such that

$$\begin{aligned} \exists\, \mathbf{k} \in \mathbf{Z}^m,\ k \in \mathbf{Z} : \mathbf{x}' = AC\mathbf{k} + kA\mathbf{b} + A\mathbf{d} + \mathbf{b} \wedge P\mathbf{k} \le \mathbf{q} \wedge k \ge 0 \\ \wedge\, T(C\mathbf{k} + \mathbf{d}) \le \mathbf{u} \\ \wedge\, T(AC\mathbf{k} + A\mathbf{d} + \mathbf{b}) \le \mathbf{u} \\ \wedge\, T(AC\mathbf{k} + (k-1)A\mathbf{b} + A\mathbf{d} + \mathbf{b}) \le \mathbf{u}. \end{aligned} \qquad (2)$$

Thus, each element $\mathbf{x}'$ of S' satisfies $\exists\, \mathbf{k}' \in \mathbf{Z}^{m+1} : \mathbf{x}' = C'\mathbf{k}' + \mathbf{d}' \wedge P'\mathbf{k}' \le \mathbf{q}'$, where we have $\mathbf{k}' = \begin{bmatrix} \mathbf{k} \\ k \end{bmatrix}$, $C' = [AC; A\mathbf{b}]$, $\mathbf{d}' = A\mathbf{d} + \mathbf{b}$, and where the system $P'\mathbf{k}' \le \mathbf{q}'$ represents the conjunction of the constraints present in (2). Hence, we have a direct method of computing the representation of S' given S and I. Notice that one of the effects of applying an iterative transformation to a periodic vector set is the creation of a new column in its C matrix. In practice, it is often possible to limit the size of C by first checking whether the new column is identical to an existing one, and simplifying the linear system accordingly.

3.3 Repeated Execution of Nested Instructions

Programs often contain nested loops. If we want to apply the construction described in the previous section in this context, we need to be able to compute a normalized instruction that is equivalent to the body of a cycle, itself containing a cycle. In this section, we show that this can be done provisionally. One condition that needs to be satisfied is that the precondition and the exit condition of a cycle are mutually exclusive (not at all uncommon in practice). For instance, let us consider the program depicted in Figure 2 where the cycle ④–③–④ has the precondition $y < x$ and the exit condition $y \ge x$. Any repeated execution of the cycle, starting in ④ and followed by the transition ending in ①, is equivalent

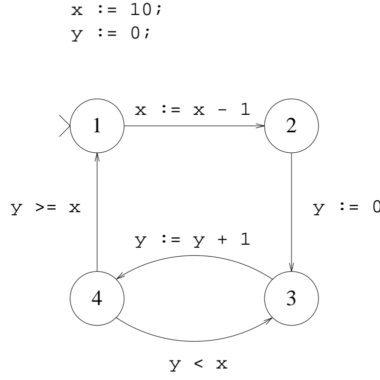

Fig. 2. A program with nested loops.

to the simpler instruction `y := x`. We present a systematic way of computing such an equivalent instruction.

Formally, given a cycle instruction I and an exit condition φ, we are looking for a normalized instruction I' such that for each periodic vector set S we have $I'(S) = \varphi(I^+(S))$, where $\varphi(X)$ denotes the subset of X containing the values satisfying φ. Using the results of the previous section[5] $I'(S)$ is the set of all $\mathbf{x}'$ such that

$$\exists\, k \in \mathbf{Z},\, \mathbf{x} \in S \,:\, \mathbf{x}' = A\mathbf{x} + kA\mathbf{b} + \mathbf{b} \,\wedge\, T' \begin{bmatrix} \mathbf{x} \\ k \end{bmatrix} \le \mathbf{u}',$$

where T' and $\mathbf{u}'$ are a linear system equivalent to the conjunction of $k \ge 0$, $T\mathbf{x} \le \mathbf{u}$, $T(A\mathbf{x}+\mathbf{b}) \le \mathbf{u}$, $T(A\mathbf{x}+(k-1)A\mathbf{b}+\mathbf{b}) \le \mathbf{u}$ and $\varphi(A\mathbf{x}+kA\mathbf{b}+\mathbf{b})$.

Assume now that the linear system defined by T' and $\mathbf{u}'$ contains at least one equation $\boldsymbol{\alpha}_1 \cdot \mathbf{x} + \alpha_k k = \beta$ for which $\alpha_k \neq 0$. This means that the number of iterations is determined by the initial values, i.e. that the loop is deterministic. In this situation, k can be expressed as a linear function $k(\mathbf{x}) = \frac{1}{\alpha_k}(\beta - \boldsymbol{\alpha}_1 \cdot \mathbf{x})$, and if all the coefficients in $k(\mathbf{x})$ are integers, the values $\mathbf{x}' \in I'(S)$ can be expressed as

$$\exists \mathbf{x} \in S \,:\, \mathbf{x}' = A\mathbf{x} + (k(\mathbf{x})A\mathbf{b} + \mathbf{b}) \,\wedge\, T' \begin{bmatrix} \mathbf{x} \\ k(\mathbf{x}) \end{bmatrix} \le \mathbf{u}',$$

which can be turned into the canonical form of a normalized instruction by expanding $k(\mathbf{x})$. In practice, the equation used to compute $k(\mathbf{x})$ is chosen at random among the suitable ones. These are obtained straightforwardly from the representation of the system [CH78, Hal93, LeV92].

[5] The same approximation applies, namely that we compute a set S' such that $II^+(S) \subset S' \subset I^+(S)$.

3.4 Inclusion Between Periodic Vector Sets

Testing inclusion between periodic vector sets is essential for detecting termination of a state-space exploration. Unfortunately, it is a very difficult problem. We take the pragmatic approach of using an approximate test which never gives false positives (claims that there is inclusion when there is not) but which can give false negatives (not detecting inclusion when it actually occurs). This is not too bothersome given the way we use periodic vector sets (see next section): false negatives when testing inclusion can lengthen the exploration of the state-space, but will never lead to incorrect results.

Consider a periodic vector set S, i.e. the set of all $\mathbf{x}$ such that $\exists\,\mathbf{k} \in \mathbf{Z}^m : \mathbf{x} = C\mathbf{k} + \mathbf{d} \,\wedge\, P\mathbf{k} \leq \mathbf{q}$. We want to test whether all elements of this set are included in a periodic vector set $S' = \{\mathbf{x} \mid \exists\,\mathbf{k} \in \mathbf{Z}^m : \mathbf{x} = C'\mathbf{k} + \mathbf{d'} \,\wedge\, P'\mathbf{k} \leq \mathbf{q'}\}$. If we manage to find an integer matrix M and an integer vector $\mathbf{n}$ such that $\mathbf{d} - \mathbf{d'} = C'\mathbf{n}$ and $C = C'M$, we have for all $\mathbf{x}$ in $S : \exists\,\mathbf{k} \in \mathbf{Z}^m : \mathbf{x} = C'(M\mathbf{k} + \mathbf{n}) + \mathbf{d'} \,\wedge\, P\mathbf{k} \leq \mathbf{q}$. Hence, if all the solutions of $P\mathbf{k} \leq \mathbf{q}$ are solutions of $P'(M\mathbf{k} + \mathbf{n}) \leq \mathbf{q'}$, we have $\exists\,\mathbf{k'} \in \mathbf{Z}^{m'} : \mathbf{x} = C'\mathbf{k'} + \mathbf{d'} \,\wedge\, P'\mathbf{k'} \leq \mathbf{q'}$ and $\mathbf{x}$ belongs to S'.

The existence of suitable M and $\mathbf{n}$ is related to the possibility of expressing each periodicity vector of S, as well as the difference of initial values $\mathbf{d} - \mathbf{d'}$, as linear combinations of the periodicity vectors of S'. This allows us to associate a vector of repetition counts of S' to each point of S. The inclusion test can be carried out by first finding M and $\mathbf{n}$ and thereafter performing an inclusion test between linear systems of inequalities. The latter problem is equivalent to an inclusion test between polyhedra and can be easily solved in the dual representation of polyhedra [CH78, Hal93]. The former is equivalent to solving a general linear diophantine equation where each component of M and $\mathbf{n}$ is considered as a variable. This problem can be simplified by looking only for solutions where $\mathbf{n}$ and each column of M have no more than one non-zero component. In that case, each column of C is constrained to be an integer multiple of a column of C' and the computation of M is reduced to determining the coefficients of the mapping. This heuristic has shown itself to be powerful enough in most cases.

3.5 Testing for Emptiness

A periodic vector set is called *empty* if it does not contain any point. Testing for emptiness is useful, for instance, in order to determine if there exists a reachable state belonging to a reachable group of states. A periodic vector set is empty if and only if its linear system of inequalities $P\mathbf{k} \leq \mathbf{q}$ does not admit any integer solution (in other words, the polyhedron determined by this linear system does not contain any integer point). The test of emptiness can thus be reduced to an *integer programming problem* [Mur76, Gre71].

4 Verification with Periodic Vector Sets

The simplest way to explore the state-space of a program is to use a search to enumerate all reachable states one-by-one. Of course, this approach fails when

the state-space is too large. In what follows we describe how periodic vector sets can be used to significantly extend its applicability.

All enumerative state-space searches are based on a common principle, they spread the reachability information along the transitions of the system to be analyzed. The exploration process starts with the initial state of the system, and tries at every step to enlarge its current set of reachable states by propagating these states through transitions. The procedure terminates when a stable set is reached.

The idea behind the use of periodic vector sets is to process sets of states rather than individual states which makes it possible to reach a stable set faster. The problem is that, starting with a single state, and applying only deterministic transitions to individual states, one never generates sets of states that can be manipulated simultaneously. A solution is the use of *meta-transitions* able to generate large sets of reachable states from a single reachable state. This is exactly what we have done while computing the effect of the repeated application of a transition. So, we have the tools for using the following approach.

One first analyses the cycles and nested cycles of the control graph of the program. For each cycle of interest, a corresponding meta-transition is added to the program. Now, the state-space search algorithm is rewritten in a such a way that it works with sets of states (represented as a finite union of periodic vector sets) rather than with individual states. One still starts with a single initial state, but each time a meta-transition is encountered a periodic vector set is produced. It is thus a good heuristic to give priority to meta-transitions. The exploration terminates when the representation of the set of reachable states stabilizes. This happens when every new deductible periodic vector set is included in the current list of reachable sets. Notice that all approximations we have made in the implementation of operations on periodic vector sets are conservative. They will reduce the benefit of using periodic vector sets, but will not lead to states being missed or to states being incorrectly considered as reachable.

In general, finding all cycles in a graph is an NP-hard problem. However, the cycle analysis needed to introduce meta-transitions does not need to be exhaustive and there are a number of techniques that can be used to make it reasonably efficient. Many programming languages include explicit instructions for loops, and thus allow cycles to be detected during the compilation of the program into a transition system. Another idea is to make the most of the exploration algorithm; for instance, if a depth-first search of the control state-space is used, one may detect duplicate control states on the search stack and compute the periodicity vector of the corresponding cycle.

5 Experimental Results

An experimental system that allows the manipulation of periodic vector sets has been implemented. This system has been used to do state-space searches for programs represented as extended finite-state machines. The method that

was used to introduce meta-transitions was to consider only the cycles found by detecting duplicate control states on the search stack during a depth-first search.

The first results are quite encouraging. As an example, we applied our method to the analysis of the lift program described in [Val89]. This program is composed of two parallel processes intended to model the interaction between the motor of a lift and one of its control panels. Each process is represented by an extended finite-state machine, and communicates with its counterpart via global integer variables. The interleaving semantics of parallelism is assumed, hence there exists an easily derivable extended finite-state machine equivalent to the parallel composition of the two processes. The program, expressed in a Promela-like language[6] [Hol91], is given in Figure 3. The global variable `c` is intended to

```
int c = 1, g = 1, a = 0, N = 10;

process motor {
  do
     :: atomic { a == 1 -> a = 0; c = c + 1 }
     :: atomic { a == 2 -> a = 0; c = c - 1 }
  od
}

process control {
  do
    :: atomic { c < g  -> a = 1 } ; a == 0 ->
    :: atomic { c > g  -> a = 2 } ; a == 0 ->
    :: atomic { c == g ->
      do
        :: g < N -> g = g + 1
        :: g > 1 -> g = g - 1
        :: break
      od }
  od
}
```

Fig. 3. The lift example program.

store at any time the *current* floor. The motor program works by waiting for an order from the control part, expressed as a non-zero value of the *aim* variable `a`, then by updating `c` accordingly. The control program repeatedly compares the values of `c` and `g`, the latter expressing the *goal* floor, then sends appropriate commands to the motor, and finally chooses a new goal floor at random when

[6] The purpose of `atomic` statements is to define sequences of states that cannot be interleaved.

the current one is reached. The valid floor numbers are the integers between 1 and the value of the constant `N`.

It can be shown that the number of reachable states of this program grows quadratically with `N`, and exceeds 20000 for $\mathtt{N} = 50$. Using our implementation, we were able to construct a representation of these reachable states in only 43 analysis steps and by consuming less than one second of CPU time. A noteworthy observation was the independence between the analysis time and the value of `N`.

6 Conclusions and Comparison with Other Work

We have suggested periodic vector sets as a representation of the sets of data-values a program can generate. This representation was developed pragmatically with the goal of improving the efficiency of state-space exploration. It can represent sets with a complex periodic structure, whether they are finite or infinite. It is far from perfect. We have been able to show how a number of operations can be applied to periodic vector sets, but we sometimes have had to make conservative approximations. Moreover, we cannot claim good general worst-case complexity bounds on the operations. The positive side is that, it works! We have been able to analyze systems for which enumerative methods are hopeless.

One major advantage of our representation, and of the way we suggest using it, is that it is very flexible and perfectly compatible with a number of other techniques. First, there is complete flexibility as to which part of the state descriptors is represented as a periodic vector set. Second, since verification is viewed as a form of extended enumerative state-space search, the techniques that have been developed for the latter approach can still be used in conjunction with periodic vector sets. This is for instance the case for partial-order methods [Val90, God90, HGP92, McM92, GW93, Pel93, WG93]. The approach we have presented thus allows the combination of symbolic and partial-order approaches which could be extremely fruitful. In summary, periodic vector sets are not *the* solution to the state-space explosion problem, but rather a useful technique that can give excellent results for particular types of large sets of states and, crucially, can be combined with other techniques.

Linear constraints have been repeatedly suggested as a useful tool in program analysis and verification [CH78, Kri93, Lub84] and, recently for the verification of real-time properties of systems [Hal93, YL93]. The work presented here is definitely in this tradition. The main innovation in our work is the introduction of periodicity in the representation which technically amounts to working with the projection of linear transformations of convex polyhedra bounded sets of integers. Representation systems for periodic sets of integers have already been proposed for instance in [Mer90], but only in the framework of single-dimensional spaces and in the particular case of infinite periodic sets. These systems are thus unable to deal with polyhedral boundaries involving more than one variable. A type of representation much closer to our work has also been considered in the context of temporal databases [KSW90]. However, the representation introduced there is more restrictive than ours and would be unsufficient for our verification

goal. The idea of analyzing cycles also appears in [Kri93], [Lub84] and [Val89]. In [Kri93] transitions do not have preconditions which substantially simplifies the problem. Moreover, no systematic representation of periodic sets is introduced. In [Lub84] and [Val89], cycles are treated with an inductive argument rather than with a powerful representation. The method presented there thus does not have the advantage of being a direct extension of a systematic state-space search.

References

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

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

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

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symposium on Principles of Programming Langages*, 1978.

[CMB90] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagram. In *Proc. 2nd Workshop on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 23–32, Rutgers, June 1990. Springer-Verlag.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[FM91] J.C. Fernandez and L. Mounier. On the fly verification of behavioural equivalences and preorders. In *Proc. 3rd Workshop on Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 181–191, Aalborg, July 1991.

[GL93] S. Graf and C. Loiseaux. A tool for symbolic program verification and abstraction. In *Computer Aided Verification, Proc. 5th Int. Workshop*, volume 697, pages 71–84, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proc. 2nd Workshop on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185, Rutgers, June 1990. Springer-Verlag.

[Gre71] H. Greenberg. *Integer Programming*. Academic Press, New York, 1971.

[GW93] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, April 1993.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Proc. 5th Workshop on Computer Aided Verification*, volume 697, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[HGP92] G. J. Holzmann, P. Godefroid, and D. Pirottin. Coverage preserving reduction strategies for reachability analysis. In *Proc. 12th International Sym-*

posium on Protocol Specification, Testing, and Verification, Lake Buena Vista, Florida, June 1992. North-Holland.

[Hol85] G. J. Holzmann. Tracing protocols. *AT&T Technical Journal*, 64(12):2413–2434, 1985.

[Hol91] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.

[JJ89] C. Jard and T. Jeron. On-line model-checking for finite linear temporal logic specifications. In *Automatic Verification Methods for Finite State Systems, Proc. Int. Workshop, Grenoble*, volume 407, pages 189–196, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[Kri93] A.S. Krishnakumar. Reachability and recurrence in extended finite state machines: Modular vector addition systems. In *Proc. 5th Workshop on Computer Aided Verification*, volume 697, pages 110–122, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[KSW90] F. Kabanza, J.-M. Stévenne, and P. Wolper. Handling infinite temporal data. In *Proc. of the 9th ACM Symposium on Principles of Database Systems*, pages 392–403, Nashville Tennessee, 1990.

[LeV92] H. LeVerge. A note on Chernikova's algorithm. Research Report 1662, INRIA, Rennes, April 1992.

[Lub84] B. Lubachevsky. An approach to automating the verification of compact parallel coordination programs. I. *Acta Informatica*, 21:125–169, 1984.

[McM92] K. McMillan. Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. 4th Workshop on Computer Aided Verification*, Montreal, June 1992.

[Mer90] N. Mercouroff. Analyse sémantique de communications entre processus de programmes parallèles. Rapport de Recherche LIX/RR/90/09, Ecole Polytechnique, Palaiseau, France, September 1990.

[Mur76] K. Murty. *Linear and Combinatorial Programming*. Wiley, New York, 1976.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *Proc. 5th Conference on Computer Aided Verification*, Elounda, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[Val89] A. Valmari. State space generation with induction. In *Proc. Scandinavian Conference on Artificial Intelligence - 89*, pages 99–115, Tampere, Finland, June 1989.

[Val90] A. Valmari. A stubborn attack on state explosion. In *Proc. 2nd Workshop on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165, Rutgers, June 1990. Springer-Verlag.

[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.

[Wes78] C.H. West. Generalized technique for communication protocol validation. *IBM J. of Res. and Devel.*, 22:393–404, 1978.

[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.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Proc. 5th Workshop on Computer Aided Verification*, volume 697, pages 210–224, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

This article was processed using the LaTeX macro package with LLNCS style