Efficient State Classification of Finite Markov Chains

Aiguo Xie and Peter A. Beerel

CENG 97-20

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4481

November 1997

Efficient State Classification of Finite Markov Chains

Aiguo Xie and Peter A. Beerel

EE-Systems Department
University of Southern California
Los Angeles, CA 90089

Abstract

This paper presents an efficient method for state classification of finite Markov chains using BDD-based symbolic techniques. The method exploits the fundamental properties of a Markov chain and classifies the state space by iteratively applying reachability analysis. We compare our method with the current state-of-the-art technique [1] which requires the computation of the transitive closure of the transition relation of a Markov chain. Experiments in over a dozen of synchronous and asynchronous systems demonstrate that the present method dramatically reduces (with up to several orders of magnitude) the CPU time needed, and solves much larger problems because of reduced memory requirements.

1 Introduction

Markov chains are an important class of stochastic processes that can effectively model randomly evolving systems. They are conceptually simple since they have *short memory*, i.e., their future evolution depends only on the current outcome and not on further history. Although they are simple, Markov chains have found vast applications in engineering including the analysis of system reliability [2, 3], control [4, 5, 6], power consumption [7, 8, 9] and performance [10, 11, 12, 13, 14]. The first two applications are usually concerned with the short-term (*transient*) behavior or *time to certain events* of the Markov chain whereas the latter two applications often need knowledge of the long-term (*stationary*) behavior. An important step in carrying out these analyses is the structural analysis of the Markov chain which classifies its state space into different sets, namely, the set of *transient states* (those that are expected not to be visited in the long run) and the set of *recurrent states* (those that may be visited infinitely often). The set of recurrent states may be further classified into

different classes which once entered trap the Markov chain inside the class indefinitely. The purpose of this paper is to present an efficient method that performs this state classification step.

The structure of the state space of a Markov chain may be represented by a directed graph where vertices correspond to the states and edges correspond to the transitions among the states. One method to perform structural analysis of the state space is to *explicitly* identify all the (maximal) *strongly connected components* (SCCs) of the graph [15]. Components (those called terminal SCCs) that do not have edges leading to other components correspond to the recurrent state classes and all other components correspond to the set of transient states. Since Markov chains derived from real systems often possess huge state space [10, 1, 12], this method can be intolerably slow because it analyzes states *sequentially*. Another method for structural analysis of large Markov chains uses *implicit* (symbolic) techniques [1] based on *binary decision diagrams* (BDDs) [16] so that all the SCCs of the corresponding graph are *simultaneously* computed and the terminal SCCs are subsequently identified. A key step of this method uses the recursive paradigm in [17] to compute all the states that can be reached from individual states, or from a graph point of view, the *transitive closure* of the corresponding graph. Unfortunately, this step tends to be *memory-intensive* and *time-consuming*.

This paper also proposes a BDD-based symbolic technique but exploits the fundamental property of transience and recurrence of a Markov chain. Specifically, we determine whether a randomly taken state s (called a *trial state*) is transient or recurrent by simply comparing two particular sets of states associated with it, namely, the set of states it can reach (called its *forward set*) and the set of states that reach it (called its *backward set*). These two sets are computed via forward and backward symbolic reachability analysis [18, 19]. We prove that if its backward set contains its forward set, the state s is recurrent and its forward set forms a recurrent class. Moreover, we show that all states in its backward set can always be classified as transient or recurrent using simple BDD operations. We randomly take a state to be the next trial state from the remaining states that have not been classified and repeat the above described analysis until all the states have been determined for their transience/recurrence property.

Our method *sequentially* computes a subset of SCCs each containing a trial state via symbolic reachability analysis whereas the previous symbolic method [1] computes all the SCCs of the graph *concurrently* via computation of transitive closure of the graph. Since

most systems contain only a few recurrent classes and a large portion of their state spaces are recurrent, the number of states to be taken as trial states in our method often turns out to be very small, which limits the number of times reachability analysis is performed. Finally, the fact that reachability analysis is typically much faster and less memory-intensive than the computation of transitive closure makes our method dramatically outperform the previous method [1] on almost all the examples from synchronous and asynchronous systems we have tested. In particular, we observe run time reduction of up to several orders of magnitude, and our method solves all the examples on which the previous method fails due to insufficient memory.

The remaining of this paper is organized as follows. Section 2 reviews the basics of a Markov chain necessary for the further discussion of this paper followed by the previous work on its structural analysis. Section 3 is the main section of this paper which describes the principles, proofs, and the algorithm of our method. Experimental results are given in Section 4 and our conclusion is given in Section 5.

2 Preliminaries and Previous Work

2.1 Finite-state Markov chains

Consider a discrete state space $S = \{1, 2, \cdots\}$. A discrete-time stochastic process X on S is a sequence of random variables, $\{X_n \in S : n \geq 0\}$. We call the value assumed by random variable X_n the state of X at time instant n. X is a *Markov chain* (or *Markovian*) if its future evolution depends only on its current state. More formally, if X is Markovian and it is in state $i \in S$ at time n, the probability that state $j \in S$ is visited at time $n+1$ is independent of its history before time n, i.e, $Pr(X_{n+1} = j \mid X_n = i, (X_{n-1}, \cdots, X_0) \in S) = Pr(X_{n+1} = j \mid X_n = i)$. X is *time-homogeneous* if the above conditional probability is independent of time instant n. In this case, the evolution of X is governed by a (1-step) *transition probability matrix* $P = (p_{ij}), i, j \in S$ where $p_{ij} = Pr(X_{n+1} = j \mid X_n = i) = Pr(X_1 = j \mid X_0 = i)$.

Real systems can often be modeled by time-homogeneous Markov chains with finite state space so that one can write $S = \{1, 2, \cdots, N\}$ where N is the size of S. In this paper, we assume the state space is finite. Systems with massive concurrency (e.g., those modeled by Petri-nets) [10] or having correlations among system events [7, 8, 20] can all be modeled by Markov chains with properly defined state spaces.

Let us assume $M = (S, P)$ is a time-homogeneous Markov chain in discrete time with finite state space S and a probability transition matrix P.

To consider the way in which states of S are related to each other, one needs the notion of *communication* between states. A state $i \in S$ communicates with state $j \in S$, denoted by $i \to j$, if there is a positive probability that state j will eventually be visited once state i is visited. Or equivalently, $Pr(M_n = j \mid M_0 = i) = p_{ij}(n) > 0$ for some $n > 0$. Otherwise, state i does not communicate with state j, denoted by $i \not\to j$. States i and j *intercommunicate* if $i \to j$ and $j \to i$, written $i \leftrightarrow j$.

Definition 2.1 *State $i \in S$ is recurrent if $\forall j \in S$ such that $i \to j$, $j \to i$ as well. Otherwise, state i is transient, i.e, $\exists j \in S$ such that $i \to j$ but $j \not\to i$.*

It can be easily checked that a recurrent state is visited infinitely often once it is visited, while the expected number of the times M visits a transient state is bounded from above.

Many of the nice properties of Markov chain M can be derived from the famous *Chapman-Kolmogorov* theorem [21] which expresses the probability of M to go from one state to another in $n \geq 0$ time steps in terms of the 1-step transition probability matrix P. One such property concerning the communication relation between two states is as follows.

Theorem 2.1 *If $i \to j$ and $j \to k$, then $i \to k$. Further, if $i \leftrightarrow j$, then $j \leftrightarrow i$.*

By Definition 2.1 and Theorem 2.1, it is easily verified that if states i, and j intercommunicate, either both of them are recurrent or both of them are transient.

Theorem 2.1 ensures that the intercommunication relation, $\leftrightarrow$, is an equivalent relation (i.e., it is transitive and reflective) so that the state space S can be partitioned into equivalent classes according to $\leftrightarrow$. That is to say, $S = T \cup R_1 \cup R_2 \cdots \cup R_K$ where T is set of all transient states of S, R_1 through R_K are K recurrent classes of S each containing recurrent states of S that do not communicate with states in other recurrent classes.

Example Figure 1 shows a Markov chain containing 7 states. State 1 is clearly transient because $1 \to 2$ but $2 \not\to 1$. States 3, 4 and 5 are also transient for a similar reason. The chain has two recurrent classes, namely, $\{2\}$ and $\{6, 7\}$. □

From the above discussion, we see that the transience and recurrence property of states in a Markov chain can be derived from the their communication relation. In particular,

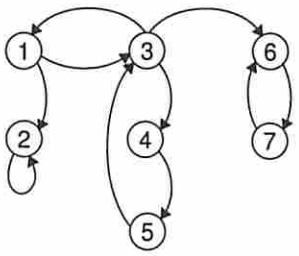


Figure 1: A small Markov chain example.

under our assumption that the state space S is finite, the structure of the state space can be equivalently represented by a directed graph $G = (V, E)$, where the vertex set V corresponds to all the states in S and there is an edge e_{uv} from vertex u to v if and only if $p_{ij} > 0$ where u and v are the corresponding vertices of states i and j in G. Then, $i \rightarrow j, i, j \in S$ is equivalent to saying there is a path from vertex u to v in G. Similarly, $i \leftrightarrow j$ is equivalent to saying vertices u and v are in the same strongly connected component. For convenience, we call graph G the boolean transition graph of M, and its associated matrix $Q = (q_{uv})$ the boolean transition matrix of M where $q_{uv} = \begin{cases} 1 & \textit{if} \quad e_{uv} \in E, \\ 0 & \textit{otherwise}. \end{cases}$ Further, since vertices of V are in one-to-one correspondence with states in S, we replace the set V with S whenever convenient. In the sequel, we use the boolean transition matrix Q instead of the transition probability matrix P when performing structural analysis of M.

To obtain a particular characteristic of a real system, many different types of analysis may be carried out on the Markov chain M that models the system. These analyses include the computation of the probability distribution of M over the state space S at a fixed time instant n (*transient analysis*), the expected time and probability to enter a set of states (*first-passage-time and absorption probability analysis*) [22], and the asymptotic probability distribution of M over the state space S after an infinite amount of time (*stationary analysis*). Identification of transient states and all the recurrent classes plays an important role in these analyses. For instance, first-passage-time analysis is often performed against a particular recurrent class. By treating each recurrent class as a single state, absorption analysis may also be accelerated. Moreover, the stationary analysis is not possible without the identification of the recurrent classes (unless it is guaranteed that M contains only one recurrent class) because the system will be under-determined if the chain has multiple recurrent classes. In fact, even when the number of the recurrent classes is expected to be one, it is often neccessary to verify that this is the case because modeling errors or design bugs may introduce extra recurrent classes.

In the next subsection, we briefly go over the previous work on state classification of a Markov chain.

2.2 Previous work

We focus on the recent work in [1] which uses symbolic techniques based on binary decision diagrams (BDDs) [16]. The method relies on the fact that recurrent classes of a Markov chain $M = (P, S)$ correspond to the *terminal* SCCs of the transition graph G associated with M. A terminal SCC is one that do not have an edge emitting out of it. By the discussion in the previous section, the correctness of the method is easily determined.

The transition graph G is represented by a BDD. To compute all the SCCs in G, the method first computes the transitive closure TC of G which describes the set of states that each state of S communicates with. One may think of TC as a boolean matrix of size $S \times S$, i.e, $TC_{ij} = \begin{cases} 1 & \textit{if} \quad i \rightarrow j, \\ 0 & \textit{otherwise.} \end{cases}$ Therefore, checking if $i \rightarrow j$ is equivalent to checking if the element $TC_{ij} = 1$. Suppose it is determined that $i \rightarrow j$ by the fact that $T_{ij} = 1$. To see if $j \rightarrow i$ so that i, j are strongly connected, one merely needs to check if $T_{ji} = 1$. Thus, the second step in the method is to *term-wise* multiply TC with its transpose, TC^t, yielding a symmetric matrix (let it be denoted by $TC * TC^t$). Clearly, i and j are in the same SCC *iff* $(TC * TC^t)_{ij} = (TC^t * TC)_{ji} = 1$. All above operations are performed symbolically.

Next, each SCC is represented by a particular state (called the representative state) of the SCC using the so called *datum priority function*[23] of the corresponding BDD. A new directed graph consisting of these representative states is then constructed so that it has an edge from one (representative) state i to another (representative) state j *iff* there is an edge from some state in the SCC represented by i to some state in the SCC represented by j. This new graph is *acyclic* such that the terminal SCCs are represented by the corresponding leaf vertices.

The bottleneck of the method is the computation of the transitive closure TC of G. To this end, a recursive algorithm presented in [17] which also uses BDD-based symbolic operations is adopted. Nevertheless, it can still take intensive amount of CPU time and memory to compute TC when G is large.

3 State classification by iterative application of reachability analysis

The method we present in this section does not compute the transitive closure of the graph G. Rather, it relies on the definition of a transient/recurrent state. Once we determine a state to be transience/recurrent, the transience and recurrence properties of other states that communicate with this state are deduced and they are removed from further consideration.

3.1 The principles

We start by defining the *forward* and *backward* sets of a state.

Definition 3.1 *The forward set of state $i \in S$, denoted by $\mathcal{F}(i)$, is the set of states that i communicates with. That is, $\mathcal{F}(i) = \{j \in S \mid i \to j\}$. Similarly, the backward set of state i, denoted by $\mathcal{B}(i)$, is the set of states that communicate with i. That is, $\mathcal{B}(i) = \{j \in S \mid j \to i\}$.*

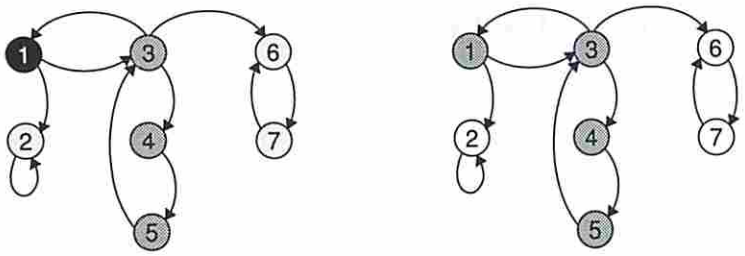
The following lemma shows a simple property of forward and backward sets.

Lemma 3.1 *Let $i, j \in S$. If $j \in \mathcal{F}(i)$, then $\mathcal{F}(j) \subseteq \mathcal{F}(i)$. Similarly, if $j \in \mathcal{B}(i)$, then $\mathcal{B}(j) \subseteq \mathcal{B}(i)$.*

Proof The proof is a direct consequence of the transitivity property of relation $\to$ given in Theorem 2.1. Suppose $j \in \mathcal{F}(i)$, i.e., $i \to j$, then, $\forall k \in \mathcal{F}(j)$, i.e., $j \to k$, we have $i \to k$, meaning $k \in \mathcal{F}(i)$ so that $\mathcal{F}(j) \subseteq \mathcal{F}(i)$. Similarly, if $j \in \mathcal{B}(i)$, i.e., $j \to i$, then, $\forall k \in \mathcal{B}(j)$, i.e., $k \to j$, we have $k \to i$, which implies $\mathcal{B}(j) \subseteq \mathcal{B}(i)$. □

Theorem 3.1 *A state $i \in S$ is recurrent if and only if $\mathcal{F}(i) \subseteq \mathcal{B}(i)$. In other words, i is transient if and only if $\mathcal{F}(i) \not\subseteq \mathcal{B}(i)$.*

Proof Suppose $i \in S$ is recurrent, then $\forall j \in S$ such that $i \to j$, i.e., $j \in \mathcal{F}(i)$, we have $j \to i$ by Definition 2.1 of a recurrent state. That is to say, $j \in \mathcal{B}(i)$ so that $\mathcal{F}(i) \subseteq \mathcal{B}(i)$. Conversely, if $\mathcal{F}(i) \subseteq \mathcal{B}(i)$, which means $\forall j \in S$ such that $i \to j$, we have $j \in \mathcal{F}(i) \subseteq \mathcal{B}(i)$ so that $j \to i$. Then i is recurrent by Definition 2.1. The fact that a state is either recurrent or transient concludes the proof. □



(a) computing $\mathcal{F}(1)$ and $\mathcal{B}(1)$. (b) states in $\{1\} \cup \mathcal{B}(1)$ all transient.

Figure 2: Determining if a state is recurrent.

Theorem 3.2 *If state $i \in S$ is transient, then states in $\mathcal{B}(i)$ are all transient. If state i is recurrent, on the other hand, states in $\mathcal{F}(i)$ are all recurrent. In the latter case, set $\mathcal{F}(i)$ is a recurrent class, and set $\mathcal{B}(i) - \mathcal{F}(i)$ (if not empty) contains only transient states.*

Proof Suppose state i is transient. By Theorem 3.1, $\mathcal{F}(i) \not\subseteq \mathcal{B}(i)$, i.e., $\exists k \in \mathcal{F}(i)$ such that $k \notin \mathcal{B}(i)$. Now, suppose state $j \in \mathcal{B}(i)$, then $k \in \mathcal{F}(j)$. This is because $i \in \mathcal{F}(j)$ so that $\mathcal{F}(i) \subseteq \mathcal{F}(j)$ by Lemma 3.1. On the other hand, $\mathcal{B}(j) \subseteq \mathcal{B}(i)$ since $j \in \mathcal{B}(i)$. Therefore, we have state $k \in \mathcal{F}(j)$ but $k \notin \mathcal{B}(j)$ since $k \notin \mathcal{B}(i)$, which implies $\mathcal{F}(j) \not\subseteq \mathcal{B}(j)$ so that j is transient by Theorem 3.1.

Now, if state i is recurrent, i.e., $\mathcal{F}(i) \subseteq \mathcal{B}(i)$ by Theorem 3.1, then, $\forall j \in \mathcal{F}(i) \Longrightarrow i \leftrightarrow j$ by definition so that we have $\mathcal{F}(j) \subseteq \mathcal{F}(i)$ and $\mathcal{B}(i) \subseteq \mathcal{B}(j)$ by Lemma 3.1. Thus, $\mathcal{F}(j) \subseteq \mathcal{F}(i) \subseteq \mathcal{B}(i) \subseteq \mathcal{B}(j)$, which implies j is recurrent by Theorem 3.1. Finally, if i is recurrent and $\mathcal{B}(i) - \mathcal{F}(i)$ is not empty, let state $k \in \mathcal{B}(i) - \mathcal{F}(i)$, we merely need to show that $\mathcal{F}(k) \not\subseteq \mathcal{B}(k)$ so that k is transient. In fact, $k \in \mathcal{B}(i) \Longleftrightarrow i \in \mathcal{F}(k)$, and $k \notin \mathcal{F}(i) \Longleftrightarrow i \notin \mathcal{B}(k)$, which implies $\mathcal{F}(k) \not\subseteq \mathcal{B}(k)$. □.

Example Figure 2(a) shows the small example Markov chain in Section 2. To determine if state 1 is recurrent, we first compute $\mathcal{F}(1) = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathcal{B}(1) = \{1, 3, 4, 5\}$. Since $\mathcal{F}(1) \not\supseteq \mathcal{B}(1)$, not only is state 1 transient, all states in $\mathcal{B}(1)$ are transient as well. Thus, only states 2, 6 and 7 need to be further classified. □

Theorem 3.1 states that we can check if a state is recurrent by simply checking if its forward set is contained in its backward set. If it is, then a recurrent class has been found which equals to the forward set so that the states of this forward set can be removed from consideration. Moreover, according to Theorem 3.2 if the backward set properly contains the forward set, those states in the backward set not belonging to the forward set are all

found to be transient. In the case the forward set is not contained in the backward set, we have found a set of transient states equal to $\{i\} \cup \mathcal{B}(i)$. They can be removed from further consideration.

The next subsection gives our symbolic algorithm for state classification based on the above theory.

3.2 The algorithm

As seen in Section 1, knowing the Boolean transition matrix Q of a Markov chain $M = (S, P)$ is sufficient for us to perform the state classification of state space S. One can also think of Q as a Boolean transition relation defined from $S \times S$ into $B = \{0, 1\}$ such that a transition from state $s \in S$ to $s' \in S$ is possible *iff* $Q(s, s') = 1$. We say s is the current state while s' is the next state.

To represent the transition relation Q with BDDs, let X and Y be two sets of boolean variables encoding current states and next states, respectively. Details of BDDs and their operations can be found in [16].

3.2.1 Computing the forward and backward sets

Computation of the forward set can be performed exactly as the symbolic iterative reachability analysis for finite state machines using *fixed-point calculations* [18, 19].

The idea of fixed point calculation may be explained by the concept of the frontier set, FS. Initially, the reachable state set, RS, is set to $\emptyset$, and the frontier set is set to be the initial states. That is, $RS^{(0)} = \emptyset$, and $FS^{(0)} = I$, where I is the set of initial states. Next, the set of states, Z, that can be reached from current frontier set $FS^{(0)}$ in one time step (or one state transition) is computed. Those states that are newly reached during this iteration are taken as the frontier for the next iteration. The process repeats until the frontier set is empty.

Figure 3(a) gives the procedure that computes the forward set of state s where $\exists$ and $\wedge$ denote BDD operations of existential quantification and AND, respectively. Applying the same idea above, the backward set $\mathcal{B}(i)$ of state i can be computed by the procedure shown in Figure 3(b).

```
forward_set(s, Q) {
    RS^(0) ← ∅, FS^(0) ← {s};
    k ← 0;
    while(FS^(k) ≠ ∅) {
        FS^(k+1)(x) = ∃_y FS^(k)(y)Q(y, x) ∧ ¬RS^(k)(x);
        RS^(k+1)(x) = RS^(k)(x) ∨ FS^(k+1)(x);
        k ← k + 1;
    }
    return(F(s) ← RS^(k));
}
```

(a)

```
backward_set(s, Q) {
    RS^(0) ← ∅, FS^(0) ← {k};
    k ← 0;
    while(FS^(k) ≠ ∅) {
        FS^(k+1)(x) = ∃_y FS^(k)(y)Q(x, y) ∧ ¬RS^(k)(x);
        RS^(k+1)(x) = RS^(k)(x) ∨ FS^(k+1)(x);
        k ← k + 1;
    }
    return(B(s) ← RS^(k));
}
```

(b)

Figure 3: Computing the forward set and backward set.

```
State_classification(S, Q) {
    S' ← S;
    while(S' ≠ ∅) {
        s ← random_take(S');
        F(s) ← forward_set(s, Q);
        B(s) ← backward_set(s, Q);
        if(F(s) ∧ ¬B(s) = ∅)
            report  F(s)  a recurrent class;
            report  B(s) ∧ ¬F(s)  all transient;
        else
            report  s ∨ B(s)  all transient;
        S' ← S' ∧ ¬(s ∨ B(s));
    }
}
```

Figure 4: Computing the forward set and backward set.

3.2.2 Classifying states

Figure 4 shows our state classification algorithm. It *iteratively* finds all the transient states and the recurrent classes. During each iteration, a state from the remaining state space S' is taken as a *trial* state whose forward set and and backward set are computed using forward and backward reachability analysis given in Figure 3. Next, the trial state is determined to be either recurrent or transient by checking the containment of its forward and backward sets as discussed in the Section 3.1. States that communicate with this trial state are also determined for their recurrence/transience property. The iteration terminates when there is no state to be classified.

The function *random_take*(S') works as follows. It randomly takes a state, say s_1 from S' as a start point, and tries to find a state s_2 which has not been considered to be a trial state but can be reached from s_1 through a user definable number of steps. The intuition is that, state s_2 is more likely to be recurrent because $s_2 \in \mathcal{F}(s_1)$. Since this search can be done iteratively and after each iteration only one state is taken for the next iteration, the time spent in this function is negligible.

In general, our algorithm has much better complexity than the method in [1] because the reachability analysis is performed only for part of the state space, i.e, for those trial states returned by function *random_take*() whereas the method in [1] computes the transitive closure of the state transition graph which is equivalent to computing the reachable state sets for all the states in the state space S.

The worst case of our algorithm may occur if each iteration determines the transience or recurrence property of only the corresponding trial state when its backward set contains at most itself. Then, exactly $|S|$ iterations would be required. This could happen for instance if (1) every state in S forms a single-state recurrent class or (2) all states except one are transient and each transient state reaches only the sole recurrent state. Since most real systems contains relatively few recurrent classes and a significant portion of the state space is recurrent, very few (typically only one) iterations are performed so that the worst case complexity is rarely observed. Another case where our algorithm may also be slow is when all states (nearly) form a loop. This case does not require a large number of iterations, but the time spent in the reachability analysis during each iteration could be significant. However, the time complexity of our algorithm still remains at the same order of that of the reachability analysis. The *counter* example in the next section is a case of this kind. Advanced reachability analysis techniques such as iterative squaring [18] may help in this case.

4 Experimental Results

We implement both our method and the method in [1] based on CUDD [24] and SMV [25, 26] packages. Efficiency of the two methods are compared on over a dozen real examples, including both synchronous and asynchronous systems. All of them are specified in SMV format.

Our synchronous examples are the synchronous bus arbiter (*syncarb*) [26], the mutual exclusion element (*mutex*), the counter circuit (*counter*), and a 3-stage synchronized pipeline (*periodic*) which are all from the SMV package. Our asynchronous examples include the distributed mutual exclusion (*dme*) circuit [27, 28, 26], the pausible clock interface (*pci*) [29], the asynchronous differential equation solver [29] with its estimated version (*diffeq_est*) and its back-annotated version (*diffeq_bka*) both of which were analyzed symbolically for their average performance in [12] and finally the asynchronous FIFO circuit *fifo* also discussed in [12].

From the results listed in Table 1, we see all the above examples contain a unique recurrent class. Moreover, all the synchronous examples have no transient states. Among the asynchronous examples, the *diffeq_est* and *diffeq_bka* contain a significant amount of transient states while the *fifo* designs have a small portion of transient states.

Comparing the memory requirements, our method finishes all the examples while the previous method fails on half of them. Comparing the run time, except on the counter circuit, our method dramatically outperforms the previous method. In particular, we observe reductions of over three orders of magnitude for *periodic*, and *fifo* examples that have a large number of stages, over two orders of magnitude for *pci* and *diffeq*, and over one order of magnitude for *dme*. With more experiments, we notice that for the *fifo* example, the CPU time required by our method increases polynomially while the state space grows exponentially. This is because the BDD size of its transition relation Q grows polynomially with each added stage. Similar performance is achieved in the *syncarb*, *pci* and *dme* examples.

5 Conclusions

We have presented a BDD-based symbolic method for state classification of discrete-time finite-state Markov chains along with the proofs of its correctness. The method iteratively identifies all the transient states and the recurrent classes by the standard reachability analysis. Typically, the number of times the reachability analysis required is very small. Compared with the previously proposed method which requires the computation of the transitive closure to concurrently determine all the maximal SCCs of the state transition graph, the present method has been demonstrated to achieve dramatic improvements on the run time complexity as well as on the memory requirement on almost all examples tested.

Examples	State space					Run time (CPU sec)	
	Invariant states	Reachable states	Transient states	Recurrents tates	Recur. classes	Previous method	Our method
dme3cells	1.80×10^{16}	6,579	0	6,579	1	417	7.1
dme6cells	3.25×10^{32}	8.217×10^{6}	0	8.217×10^{6}	1	8,140 mem out	123.7
syncarb5	32,768	5,120	0	5,120	1	0.63	0.1
syncarb10	1.07×10^{9}	1.049×10^{7}	0	1.049×10^{7}	1	2.97	0.28
mutex8	524,288	2,048	0	2,048	1	0.48	0.18
periodic	6.87×10^{10}	3,952	0	3,952	1	1,577 mem out	1.3
counter8	256	256	0	256	1	0.23	0.18
counter16	65,536	65,536	0	65,536	1	0.43	100.9
pci-2-1	4.29×10^{9}	14,556	0	14,556	1	3,514 mem out	14.7
pci-4-1	2.88×10^{17}	1.76×10^{7}	0	1.76×10^{7}	1	34,620 mem out	116.5
pci-8-1	1.30×10^{33}	5.24×10^{13}	0	5.24×10^{13}	1	11h incomplete	426.3
diffeq-est	4.29×10^{9}	7,632	2,762	4,870	1	3,180 mem out	4.0
diffeq-bka	2.75×10^{11}	11,036	6,176	4,860	1	1,059 mem out	6.5
fifo2	6,5536	98	2	96	1	8.0	0.23
fifo4	1.68×10^{7}	1,150	18	1,132	1	72.2	0.58
fifo8	1.10×10^{12}	171,518	730	170,788	1	1,535 mem out	2.9
fifo16	4.72×10^{21}	4.258×10^{9}	2.48×10^{6}	4.256×10^{9}	1	11h incomplete	24.2
fifo32	5.44×10^{39}	7.736×10^{17}	2.88×10^{13}	7.736×10^{17}	1	11h imcomplete	277.0

Table 1: Experimental results.

References

[1] G. D. Hachtel, E. Macii, A. pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Transactions on Computer-Aided Design*, pages 1479–11493, December 1996.

[2] M. Malhotra and K. S. Trivedi. Power-hierarchy of dependability-model types. *IEEE Transactions on Reliability*, 43(3), Sept. 1994.

[3] O. Daniel Z. Simeu-Abazi and B. Descotes-Genon. Analytical method to evaluate the dependability of manufacturing systems. *Reliability Engineering & Systems Safety*, 55(2), Feb 1997.

[4] A.M. Stankovic, G.C. Verghese, and D. J. Perreault. Randomized modulation of power converters via markov chains. *IEEE Transactions on Control Systems Technology*, 5(1), Jan 1997.

[5] G. Santharam and P.S. Sastry. A reinforcement learning neural network for adaptive control of markov chains. *IEEE Transactions on Systems, Man & Cybernetics (Part A)*, 27(5), Sept. 1997.

[6] R. Bolla and F. Davoli. Control of multirate synchronous streams in hybrid TDM access networks. *IEEE/ACM Transactions on Networking*, 5(2), April 1997.

[7] R. Marculescu, D. Marculescu, and M. Pedram. Logic level power estimation considering spatiotemporal correlations. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1994.

[8] R. Marculescu, D. Marculescu, and M. Pedram. Efficient power estimation for highly correlated input streams. In *Proc. ACM/IEEE Design Automation Conference*, 1995.

[9] P. A. Beerel and H. S. Wadekar. Energy estimation of speed-independent control circuits. *IEEE Transactions on Computer-Aided Design*, pages 672–680, June 1996.

[10] M. Holiday and M. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Transactions on Software Engineering*, 13:1297–1310, December 1987.

[11] P. Kudva, G. Gopalakrishnan, E. Brunvand, and V. Akella. Performance analysis and optimization of asynchronous circuits. In *Proc. International Conf. Computer Design (ICCD)*, pages 221–225, October 1994.

[12] A. Xie and P. A. Beerel. Symbolic techniques for performance analysis of asynchronous systems based on average time separations of events. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 64–75. IEEE Computer Society Press, 1997.

[13] R. M. Izquierdo and D. S. Reeves. MPEG VBR slice layer model using linear predictive coding and generalized periodic markov chains. In *IEEE International Performance, Computing and Communications Conference*, 1997.

[14] T. Takine, T. Suda, and T. Hasegawa. Cell-loss and output process analysis of a finite-buff discrete-time atm queueing system with correlated arrival. *IEEE Transactions on Communications*, 43(2–4), Feb.-March 1995.

[15] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[16] R. E. Bryant. Graph-based algorithm for boolean function munipulation. *IEEE Transactions on Computers*, 35, August 1986.

[17] Y. Matsunaga, P. C. McGeer, and R. K. Brayton. On computing the transitive closure of a state transition relation. In *Proc. ACM/IEEE Design Automation Conference*, 1993.

[18] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4), April 1994.

[19] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verification'90*. American Mathematical Society, June 1990.

[20] L. Deng, K. Hassanein, and M. Elmasry. Analysis of the correlation structure for a neural predictive model with application to speech recognition. *Neural Networks*, 7(2), Feb. 1994.

[21] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Springer, 1976.

[22] R. A. Howard. *Dynamic Probabilistic Systems, Volume I: Markov Models*. John Wiley & Sons, Inc, 1971.

[23] G. D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0-1 networks. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1993.

[24] R. I. Bahar, E. A. Frohm, C. M. Gaona, and G. D. Hachtel. Algebraic decision diagrams and their applications. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1993.

[25] J. R. Burch, E. M. Clarke, and K. L. McMillan. Symolic model checking: 10^{20} states and beyond. In *Fifth Annual IEEE Symposium on Logic in Computer Science (Philadelphia, June)*, 1990.

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

[27] A. J. Martin. The design of a self-timed circuit for distributed mutual exclusion. In Henry Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 245–60. Computer Science Press, Inc., 1985.

[28] D. L. Dill and E. M. Clarke. *Trace theory for automatic hierarchical verification of speed-independent circuits.* Techical Report, Carnegie Mellon University, Computer Science Department, 1988.

[29] K. Y. Yun and R. P. Donohue. Pausible clocking: A first step toward heterogeneous systems. In *Proc. International Conf. Computer Design (ICCD)*, pages 118–123. IEEE Computer Society Press, October 1996.