Convergence Behaviour of Structural FSM Traversal

Dominik Stoffel Wolfgang Kunz
Dept. of Computer Science, Electronic Design Automation Group
Johann Wolfgang Goethe-Universität, Frankfurt, Germany

Abstract

We present a theoretical analysis of *structural FSM traversal*, which is the basis for the sequential equivalence checking algorithm *Record & Play* presented earlier [16]. We compare the convergence behaviour of exact and approximative structural FSM traversal with that of standard BDD-based FSM traversal. We show that for most circuits encountered in practice exact structural FSM traversal reaches the fixed point as fast as symbolic FSM traversal, while approximation can significantly reduce in the number of iterations needed. Our experiments confirm these results.

1 Introduction

A central problem in verification of sequential circuits is *reachability analysis*. The properties to be checked by an automatic verification tool are required to hold in those states that the system can assume after starting in a designated start state. Reachability analysis is the task of finding this set. Since sequential circuits are often modelled as *finite state machines* (FSMs), reachability analysis corresponds to a traversal of the *state transition graph* (STG) of an FSM and is therefore often called *FSM traversal*.

Standard FSM traversal algorithms [3, 12] are based on implicit representations of state sets using *binary decision diagrams (BDDs)* [2]. Large sets of states can be represented by constructing the BDD of the characteristic function of a state set. Because a single BDD represents a set of states, BDD-based reachability analysis is also called *symbolic FSM traversal*. An important property of BDDs is that they are a canonical representation of a Boolean function. This property makes them especially attractive for use in formal verification. For example, in FSM traversal this property is used to check that two sets of states are identical by proving the isomorphism of their BDDs. Despite its usefulness, the canonicity property can lead to exponential growth of data structures for the state sets even though the BDD representation is an implicit one. This holds for sequential as well as for combinational circuit verification.

For this reason, research efforts have been made to develop formal verification methods that can operate without the use of canonical representations of Boolean functions. In the domain of combinational equivalence checking there has been some notable success with methods operating directly on the structural gate netlist of the circuit [9, 1]. Since they are capable of making efficient use of structural design properties they are often referred to as *structural* techniques. These techniques and their further developments (e.g., [14, 7, 11, 8]) have made combinational equivalence checking feasible for circuits with up to one million gates. For sequential equivalence checking there have been only a few approaches making use of structural techniques ([16, 6]).

In [16], a technique called *Record & Play* is presented which uses a "structural fixed point iteration" for verifying the equivalence of two sequential circuits. The method is based on an expansion of the product machine of the two circuits into time frames. Circuit transformations are made to reduce the complexity of the verification task by merging of logic in each time frame. These transformations are stored as "instructions" in an *instruction queue* and reused in subsequent time frames if possible. In each time frame, merged logic is cut off. The algorithm terminates when a sequence of transformations and cuts is found that can be repeated over and over ("structural fixed point"). The fixed point iteration of *Record & Play* can be interpreted as approximative *structural FSM traversal*.

In this paper we study the basic properties of structural FSM traversal. In Section 2 we analyze how a time frame expansion can be used to explore the reachable state space of a finite state machine in a fixed point iteration and present results on the convergence behaviour of this iteration. In Section 3 we review how existential quantification can be performed on a non-canonical structural representation of the reachable state set. We formulate the exact algorithm of structural FSM traversal and give some comments on the approximative algorithm used in *Record & Play*. Section 4 presents experimental results.

2 FSM Traversal by Time Frame Expansion

The algorithm *Record & Play* is based on a time frame expansion of the product machine of the two designs to be compared. In this section we will study how such a time frame expansion can be used to traverse the state transition graph of a finite state machine. A finite state machine M is a 6-tuple $M = (I, S, \delta, S_0, O, \lambda)$ where I is the input alphabet, S is the set of states, $\delta : S \times I \longrightarrow S$ is the next-state function, S_0 is a set of initial states, O is the output alphabet and $\lambda : S \times I \longrightarrow O$ is the output function. For simplicity we restrict our discussion to a single initial state $S_0 = \{s_0\}$. However, a set S_0 containing several initial states can be treated in a similar way. A sequential circuit (Fig. 1) implementing such a finite state machine has a set of *(primary) inputs* $\underline{x}$, a set of *(present) state variables* $\underline{s}$ which are fed by registers, a set of *next-state variables* $\underline{z}$ providing the input of the registers and a set of *(primary) output variables* $\underline{y}$.

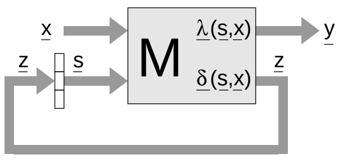


Figure 1: Sequential circuit implementing a finite state machine

The *time frame expansion model* of a finite state machine M is obtained by replicating the combinational logic for each clock cycle being considered. Each time frame is a copy of the combinational logic implementing the transition function $\delta(\underline{s}, \underline{x})$ and output function $\lambda(\underline{s}, \underline{x})$ of the FSM. The circuit structure obtained by this time frame expansion is a purely combinational structure called *iterative circuit array*. There are no storage elements. Instead, the values of the state variables of the product machine at different points in time are associated with signal values in different time frames of the iterative circuit array.

As an example, Figure 2 shows the expansion of an FSM into three time frames. The initial state, s_0, is injected at the present state variables, $\underline{s}_0$, of the first time frame. This circuit array is a combinational network which calculates for a given input sequence $(\underline{x}_0, \underline{x}_1, \underline{x}_2)$ the output sequence and next-state response, $\underline{s}_3$, of the FSM. By applying all input sequences of length 3 to this circuit we obtain at the state variables $\underline{s}_3$ all possible states the machine can assume at $t = 3$. Obviously, an expansion of a finite state machine into t time frames is a an implicit representation of the set of states, $R(t)$, the FSM can assume after exactly t clock ticks.

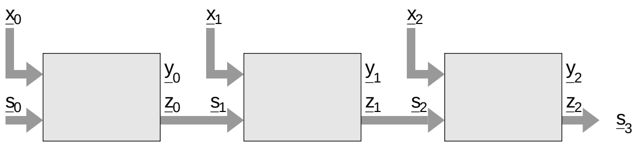


Figure 2: Time frame expansion of a finite state machine

Definition 1 (Reachable state set) *The set of all states, $R(t)$, being possible in an FSM at a specific time t after initialization is called* reachable state set at time t.

Note that this definition of the reachable state set is different from the understanding of the reachable state set R_t in conventional FSM traversal. For comparison, Table 1 shows a standard forward FSM traversal algorithm using breadth-first search. The *img*(δ, S) operation calculates for a set of states, S, the set of their immediate successors in the state transition graph. In each iteration of the loop, a set union is formed of the newly reached states with the states collected so far. Therefore, the set R_t refers to all states that can the FSM can assume at *some time* t', with $t' \leq t$.

The sets of states that can be produced by the iterative circuit array of Figure 2 at the state variables $\underline{s}_0, \underline{s}_1, \underline{s}_2, \ldots$ are the sets of reachable states $R(0), R(1), R(2), \ldots$, respectively, as given in Definition 1. In order to emphasize this difference, different notations are used for the two notions of reachable state set.

```
reachable_state_set($\delta$, $S_0$)
{
    $t$ := 0;
    $R_t$ := $S_0$;    /* initial state set */
    repeat
        $t$ := $t$ + 1;
        $R_t$ := $R_{t-1} \cup img(\delta, R_{t-1})$;
    until ($R_t = R_{t-1}$);    /* fixed point reached */
    return $R_t$;
}
```

Table 1: Standard forward FSM traversal using breadth-first search

Because the state set of an FSM is finite, obviously, there will be a time t_s for every state s of the FSM, such that $s \in R(t_s)$. In other words, the time series $R(t)$ does somehow "traverse" the state transition graph of the FSM and "visits" every state. The questions we are interested in, however, are: how can a point in time, t_{fix}, be determined for which it is guaranteed that all states reachable from the initial state have been visited? How is the convergence behaviour of the time series $R(t)$ and how is it related to the convergence behaviour of conventional FSM traversal as shown in Table 1?

For analyzing the evolution of the reachable state set $R(t)$ over time, it is helpful to imagine $R(t)$ as a set of states in the state transition graph with a special mark that is passed on to successor states in the next time step. Consider a set of marked states, $R(t)$, at time t. One time step later, at $t + 1$, these states pass their marks on to all of their immediate successor states. Their own mark is erased, unless they receive a new one from an immediate predecessor state. Now, all marked states form the reachable state set $R(t + 1)$. As we proceed in time, the states in the STG become repeatedly marked and unmarked as described. Intuitively, since the FSM has a finite set of states, and its next-state behaviour is deterministic, the time series $R(t)$ must at some point in time enter a stationary behaviour. Either there will be a single final set R_∞ or $R(t)$ will periodically cycle through a set of state sets. If $R(t)$ converges to a final set R_∞, this set corresponds to a pattern of marks which beginning at a certain time t_{fix} does not change any more, i.e., it becomes a static pattern. If $R(t)$ exhibits a periodic long-term behaviour, this corresponds to periodically repeating patterns of marks in the STG, beginning at a certain time t_{fix}. As we will see, whether we have an aperiodic or a periodic behaviour is determined by certain structural properties of the STG.

For the following analysis we need the following recursive definition:

Definition 2 (Recurrent State) *A state in the state transition graph G is called* recurrent state *if it lies on a cycle in G, or if it has a predecessor which is a recurrent state.*

Recurrent states are contained in the reachable state set $R(t)$ infinitely often, and they appear periodically. This is easy to see for recurrent states which are lying on a cycle (called *cycle states* in the sequel), by observing the $R(t)$–marks they are sending and receiving. The length of the cycle determines how many time steps it takes until a mark that has been sent out by a state returns back to it and is sent again. This time is called the state's *period of recurrence* and it is equal to the length of the cycle. However, there are also states which are not involved in a cycle but are nevertheless recurrent. Such states are reachable from cycle states, and therefore they receive, with some delay, all $R(t)$–marks which the cycle states send out. We say, a state *inherits* the recurrence periods of the states from which it can be reached. Note that in a particular run of the machine, a recurrent state that is not a cycle state can occur only once. However, for different runs of the machine it may occur at different times. The reachable state set $R(t)$ contains information about all possible runs of the machine. Therefore, a recurrent but non-cycle state is an element of $R(t)$ periodically just like a "true" cycle state.

Definition 3 (Transient State) *A state which is not recurrent is a* transient state.

Transient states are not reachable by recurrent states. They occur only once in the reachable state set $R(t)$. Transient states are only possible if the initial state of the FSM is a transient state. They are usually part of the initialization process for the machine and do not belong to the normal mode of operation.

Note that our definitions of recurrent and transient states differ from literature concerned with a probabilistic analysis of finite state machines such as [4]. The objective there is to determine the long-run probability for an FSM to be in a certain state. In that context, for example, a state is defined transient if there is a non-zero probability that the FSM will not return to it. Our definition requires that it is *impossible* to return to a transient state. In case there is a *possibility* that the FSM returns to a state, we call it a recurrent state, because it will be an element of the reachable state set, even if the *probability* for this to happen may be zero.

Since transient states occur only once, they cannot be part of the reachable state set in the fixed point we are seeking. So we can focus the analysis of our FSM traversal solely on the recurrent states.

As discussed above, a recurrent state may inherit recurrence periods from its predecessor states. It also has periods of recurrence associated with the cycles on which it is located. The following lemma tells us how these different recurrences interact.

Lemma 1 *Consider an arbitrary state s lying on a cycle of the state transition graph of length q. Furthermore, let s have a recurrence period p. Then, after a finite number of time steps, state s also has a recurrence period p_g which is the greatest common divisor of p and q.*

For a proof of this lemma and all following lemmas and theorems, please refer to [15].

The state transition graph of an FSM can be decomposed into its "cyclic" parts, the *strongly connected components (SCCs)* [5]. All states in an SCC are reachable from each other, i.e., they are lying on cycles. The FSM can be in these states arbitrarily often, therefore the SCCs determine the "long-run" or "steady-state" behaviour of the machine. By collapsing the SCCs into single vertices, we obtain a direct acyclic graph called the *SCC graph*.

The recurrence period p in the lemma may be inherited from a predecessor state. Or it may be due to another cycle of length p that state s is lying on. We can apply Lemma 1 successively to all pairs of periods p and q that a state has due to period inheritance and the cycles in which it is involved. This leads us to an interesting lemma for the states of an SCC:

Lemma 2 *After a finite transition time, the smallest recurrence period, p_{SCC}, is the same for all states in an SCC. This period p_{SCC} is given by the greatest common divisor of all cycle lengths in the SCC and of all recurrence periods for states in the SCC that have been inherited from predecessor states outside the SCC.*

If we view again the set of reachable states, $R(t)$, as a set of states in the STG carrying a mark, then this lemma says that after a sufficient amount of time each state in an SCC will be marked periodically. If the period is, for example, $p_{SCC} = 5$, then, a state will be marked in every fifth time step and will be unmarked during the remaining four time steps. Since at every time step at least one state of the SCC is marked, we can partition the states in the SCC into equivalence classes of simultaneously marked states:

Lemma 3 *The set of states of an SCC with a recurrence period p_{SCC} can be partitioned into p_{SCC} disjoint subsets C_m with $m \in \{0, 1, ..., (p_{SCC}-1)\}$, such that $C_m = R(t_{SCC}+k\cdot p_{SCC}+m)$, for all integers $k \geq 0$.*

This is an interesting first result. Let us consider the special case of a finite state machine that has all its states including the initial state within one strongly connected component. Such systems are sometimes called *non-decomposable* sequential systems, because the SCC graph of their state transition graph consists of only a single vertex. In this case, the set of reachable states, $R(t)$, converges to a series of final sets C_m that oscillate with a period p_{SCC} related to the structure of the STG.

It is possible, however, that p_{SCC} is equal to 1, in yielding a single final set $R_\infty = C_0$ to which $R(t)$ converges. This happens if there is a state in the SCC which has a self-edge or if there are cycles with lengths whose greatest common divisor is 1. In fact, such an aperiodic behaviour is very typical for FSMs implemented by practical systems. This is also confirmed by our experiments (Section 4).

One way of obtaining a non-decomposable system is by modelling the process of initialization within the FSM description itself. The FSM can then be put into its initial state by applying a special input sequence called *initializing* or *synchronizing sequence*.

Definition 4 *A* synchronizing sequence *of a finite state machine M is an input sequence that brings M to a known state s_0 regardless of the initial state or the output sequence. The state s_0 is called* synchronization state *of M.*

If a finite state machine has a synchronizing sequence, then the synchronization state s_0 and all states reachable from it must be located within one terminal strongly connected component (TSCC). The reason is that from all states reachable from s_0 the machine can be put back into s_0 by applying the synchronizing sequence. In other words, s_0 is reachable from all states which are reachable from s_0. Hence, machines with synchronizing sequences are non-decomposable.

In the special case of a non-decomposable system, the basic definitions of transient and recurrent states of an FSM and those of a Markov chain are equivalent. Therefore, the following lemma which was originally derived in [4] for homogeneous discrete-parameter Markov chains with a finite state space can also be formulated in this context.

Lemma 4 *If a finite state machine has a synchronizing sequence, then the fixed point recurrence period of all its states is 1.*

Lemma 5 *If a finite state machine has a synchronizing sequence of length l and if d is the sequential depth of the machine, then it takes at most $l + d$ time steps until all states of the machine are in $R(t)$ and have a recurrence period of 1.*

Note that the *sequential depth* of a finite state machine is given by the longest path among all shortest paths from the initial state to all nodes in the state transition graph of the FSM. In conventional symbolic FSM traversal, the sequential depth is equal to the number of iterations needed to reach the fixed point.

Although most finite state machines encountered in practice actually fall into the category of non-decomposable systems [13], it is generally possible that the SCC graph contains more than one SCC. Also, the initial state does not have to be a recurrent state. We therefore need to discuss the general case of an arbitrarily structured SCC graph.

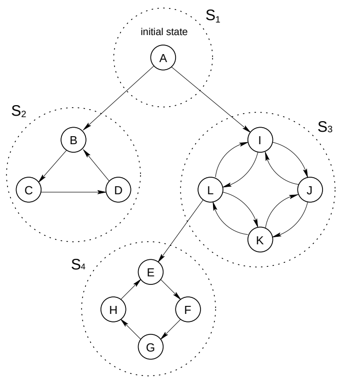
If the SCC graph of a state transition graph has more than one SCC vertex, the periods are inherited along the directed edges between the SCCs. According to Lemma 2, the recurrence period of an SCC is a proper divisor of all its predecessor SCCs. Therefore, the largest periods of recurrence are found in the "earliest" SCCs after initialization of the FSM. We call them *entry SCCs (ESCCs)*.

Definition 5 (Entry SCC) *An SCC in the state transition graph which is entered by initialization or reached exclusively via transient states after initialization is called* entry SCC (ESCC).

Note that an entry SCC need not necessarily be a source of the SCC graph. Only if the initial state is a recurrent state there is a unique entry SCC which is also the source of the acyclic SCC graph. If, however, the initial state is a transient state, then there can be several entry SCCs.

Figure 3 shows an example of such a state transition graph.

This STG is composed of four SCCs: $S_1 = \{A\}$, $S_2 = \{B, C, D\}$, $S_3 = \{E, F, G, H\}$ and $S_4 = \{I, J, K, L\}$. S_1 is an SCC without cycles. It contains only the initial state A which is a transient state. S_2 and S_3 are entry SCCs according to Definition 5. S_4 is a terminal SCC.



t	$R(t)$	C_j
0	$\{A\}$	—
1	$\{B, I\}$	—
2	$\{C, J, L\}$	—
3	$\{D, E, I, K\}$	—
4	$\{B, F, J, L\}$	—
5	$\{C, E, G, I, K\}$	C_0
6	$\{D, F, H, J, L\}$	C_1
7	$\{B, E, G, I, K\}$	C_2
8	$\{C, F, H, J, L\}$	C_3
9	$\{D, E, G, I, K\}$	C_4
10	$\{B, F, H, J, L\}$	C_5
11	$\{C, E, G, I, K\}$	C_0
12	$\{D, F, H, J, L\}$	C_1
13	$\{B, E, G, I, K\}$	C_2
...	...	...

Figure 3: A state transition graph and the first values of its reachable state set $R(t)$

Also shown in Figure 3 are the first values of the time series of the reachable state set, $R(t)$. The initial state is contained in $R(t)$ only once for $t = 0$. The remaining states are recurrent, and we can easily verify that Lemma 1 and Lemma 2 are correct. In SCC S_2 there is a unique cycle of length 3. Therefore, the states B, C and D are contained in $R(t)$ alternately every three time steps. In SCC S_3 there are several cycles whose lengths all are multiples of 2. Hence, the states in S_3 recur in $R(t)$ in two alternating sets, $\{I, K\}$ and $\{J, L\}$. The SCC S_4 contains only one cycle of length 4. However, the recurrence period of its states is 2. The reason for this is that whenever state L (of SCC S_3) is in $R(t)$, state E will be in $R(t+1)$, one time step later. State E inherits state L's recurrence period. The greatest common divisor of the inherited period of 2 and the cycle length of 4 is 2.

Obviously, after a sufficient amount of time ($t = 5$), all recurrence "interferences" have taken place and a stationary oscillation has evolved. In this example, there are six different values for the set of reachable states, $R(t)$, which are repeated in the same order with a period of six time steps. These six sets together form a cover of the set of all recurrent states.

Theorem 6 *Let P be the set of all recurrent states of a finite state machine. There always exists a cover $\{C_0, C_1, ..., C_{T-1}\}$ of P with $C_k \in 2^P$, and there is a time $0 \leq t_{fix} \leq \infty$, such that*

$$R(t_{fix} + n \cdot T + k) = C_k, \quad 0 \leq n, \quad 0 \leq k < T$$

T is equal to the least common multiple of the recurrence periods of the entry SCCs *of the FSM. T is called* fixed point oscillation period.

In our example of Figure 3 there are two entry SCCs, S_2 and S_3, with periods $p_2 = 3$ and $p_3 = 2$. The least common multiple of these two numbers is $T = 6$. This is the fixed point oscillation period of the reachable state set that we have found also empirically for this state transition graph.

It is interesting to note that for the fixed point oscillation period only the ESCCs of the state transition graph are relevant. All other SCCs including the TSCCs (unless they are, at the same time, ESCCs) do not influence the oscillation period T. (It should be noted, however, that the cycle lengths of non-entry SCCs determine how long it takes until the fixed point is reached.) This observation again points out the difference between a possibilistic state space analysis such as the characterization of the time series $R(t)$, and a probabilistic state space analysis [4], where the terminal SCCs play the important role in the analysis.

Theorem 6 justifies the formulation of a structural FSM traversal based on a time frame expansion of a finite state machine. By considering the states that can be produced by the state vectors $\underline{s}_t$ of the iterative circuit array we are able to traverse the state transition graph of the machine, visiting all states reachable from the set of initial states. For most practical systems (e.g., systems with synchronous resets or initializing sequences), the set of reachable states grows monotonically from time frame to time frame and the fixed point of the iteration consists of a single set, R_∞. For these systems, the number of iterations needed to reach the fixed point is only slightly larger (by the length of the initializing sequence) than in conventional FSM traversal.

3 Existential Quantification

In order to formulate an FSM traversal algorithm based on a time frame expansion of a finite state machine, it is necessary to have a means of recognizing the fixed point of the expansion. The sets of reachable states, $R(t)$, are represented by the iterative circuit array in a non-canonical form. Consider, for example, a fixed point oscillation period of $T = 1$. After reaching the fixed point, i.e., for $t \geq t_{fix}$, attaching a new time frame to the end of the circuit array yields the same state set, $R(t+1) = R(t) = R_\infty$, but in a different structural representation.

Note that in BDD-based FSM traversal it is easy to recognize the fixed point, because sets of states and their images under the transition function are stored *canonically* as BDDs. The BDDs representing R_t and R_{t-1} in Table 1 simply have to be checked for isomorphism to detect the fixed point. The iterative circuit array, however, represents state sets *non-canonically* as the *range* of a multi-output Boolean function implemented as a combinational circuit. In order to detect the fixed point, it is necessary to check that the ranges of the Boolean functions of two time frame expansions of lengths t and $t+T$, respectively, are equal.

To accomplish this, a structural form of existential quantification was introduced in [16], which is based on a functional decomposition of the iterative circuit array and a network cut as shown in Figure 4. The upper part of Figure 4 shows the iterative circuit array representing the set of reachable states, $R(t)$, at the state variables $\underline{s}_t$ at time t. This combinational circuit comprises the complete functional information relating input sequences of length t to the resulting states of the FSM (and can be quite large for large values of t). The information we are interested in, however, is only *what* states are possible, not *how* they can be produced in the machine. We are only interested in the *range* of the multi-output function given by the state variables $\underline{s}_t$. Therefore, the circuit array is decomposed into two parts: a so-called stub circuit and the remaining circuitry (denoted "R" in Figure 4). The stub circuit has the same range as the original circuit array. By cutting the remaining circuitry off along the indicated cut line, the functional dependency of

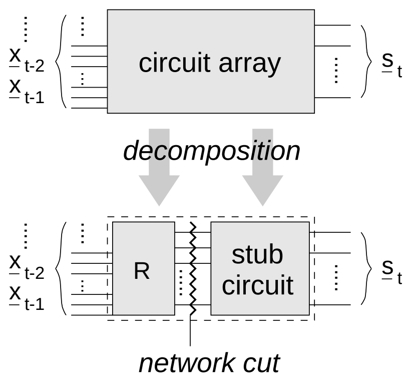


Figure 4: Structural form of existential quantification

the state variables, $\underline{s}_t$, on the input variables, $\underline{x}_t, \underline{x}_{t-1}, \ldots$, is removed. We have existentially quantified the structural set representation given by the circuit array with respect to these input variables. This greatly simplifies the set representation, and, most importantly, it allows us to detect the fixed point. If $R(t)$ and $R(t+T)$ are equal, it must be possible to decompose the circuit arrays of length t and of length $(t+T)$ such that identical stub circuits are produced in both cases. This means that the fixed point check amounts to checking the *identity of the decomposition steps* used for the existential quantifications. In *Record & Play*, an *instruction queue* is used for this purpose. The decomposition is performed by synthesizing the stub circuit using implicant-based network transformations [10]. For reasons of space, the details of the stub circuit synthesis cannot be reviewed here. The interested reader may refer to [15] for an in-depth treatment of these concepts including examples.

Using this structural form of existential quantification, structural FSM traversal can be described by the pseudo-code of Table 2. The image computation operation *img*() consists of attaching a new time frame to the current circuitry which represents the state set $R(t-1)$ and then performing existential quantification as described. The *until* condition corresponds to checking whether a previously recorded instruction queue was played successfully, i.e., it could be applied for creating the stub circuit during quantification without any modification.

```
structural_FSM_traversal($\delta$, $S_0$)
{
   $t$ := 0;
   $R(t)$ := $S_0$;    /* initial state set */
   repeat
      $t$ := $t$ + 1;
      $R(t)$ := img($\delta$, $R(t-1)$) ;
   until (exists $T$ such that $R(t)$ = $R(t-T)$);
}
```

Table 2: Structural FSM traversal

As pointed out in [16], the algorithm *Record & Play* is based on an approximative structural FSM traversal. The approximation occurs during the synthesis of the stub circuit. The network transformations which synthesize the approximative stub circuit are equivalence transformations, i.e., before cutting, the original circuit array and the decomposed network are functionally equivalent and hence have identical ranges. After cutting, the range of an exact stub circuit is still identical with that of the original network, while the range of an approximated stub is a superset of the original range. Additional state vectors may be introduced by the cut which can be modelled by a set, $A(t)$, being added to the set of reachable states, $R(t)$, in each iteration of the traversal of Table 2.

As a consequence, not only the states reachable from the initial state will be visited during traversal but also all states "injected" by the approximation and all states reachable from these. For the application of approximative structural FSM traversal, it is necessary that the properties to be checked hold in all these states. Otherwise, false negatives may occur.

On the other hand, the over-approximation of the reachable state set has a very beneficial effect: if the additional states, $A(t)$, contain states of the reachable state set, then the fixed point iteration is accelerated, because these states and their successors are visited earlier than in the exact traversal. Refer to [15] for an example illustrating this effect.

4 Experimental Results for Sequential Equivalence Checking

The algorithm *Record & Play* for sequential equivalence checking is based on approximative structural FSM traversal. Time frame expansion of the product machine of two designs to be compared allows very effectively the use of structural information when calculating the (approximative) stub circuit. Table 3 presents results for checking the equivalence of circuits of the ISCAS 89 benchmark set against their optimized and retimed versions. The experiments were conducted on a SUN Ultra I workstation. The circuits were verified with *Record & Play* and, for comparison, also with an equivalence checker based on a standard BDD-based FSM traversal (VIS-1.3).

The column labelled T gives the fixed point oscillation period encountered. As can be seen, there were no oscillations encountered in the experiments. The third column shows the number of iterations needed for the traversal. As can be observed, the over-approximation significantly accelerates the traversal. The last column shows the sequential depth of the state transition graph of the original circuit, i.e., the number of iterations of a standard (BDD-based) FSM traversal. In most cases where standard traversal did not fail, *Record & Play* needed fewer iterations. False negatives could be avoided in all experiments.

circuit name	*record_and_play()* (HANNIBAL)			*seq_verify* (VIS-1.3)	
	CPU time h:min:sec	# iter.	T	CPU time h:min:sec	sequ. depth
s208	0:00:08	15	1	0:00:04	255
s298	0:00:09	10	1	0:00:03	18
s344	0:00:11	9	1	0:00:12	6
s349	0:00:11	9	1	0:00:12	6
s382	0:00:17	16	1	0:01:55	150
s386	0:00:48	9	1	0:00:03	7
s420	0:00:43	27	1	unable	65536
s444	0:00:18	16	1	0:01:59	150
s510	0:00:35	12	1	0:00:26	46
s526	0:00:35	21	1	0:01:35	150
s635	0:01:32	37	1	unable	—
s641	0:00:12	9	1	0:00:04	6
s713	0:00:12	9	1	0:00:03	6
s820	0:36:50	17	1	unable	10
s832	0:26:37	16	1	unable	10
s838	0:08:13	51	1	unable	—
s953	0:01:09	11	1	unable	10
s1196	0:00:40	6	1	0:00:10	2
s1238	0:00:46	6	1	0:00:11	2
s1423	0:03:31	14	1	unable	—
s1512	0:04:09	16	1	unable	—
s3271	0:21:17	19	1	unable	—
s3330	0:11:33	9	1	unable	—
s3384	0:31:24	17	1	unable	—
s4863	0:36:52	8	1	unable	—
s5378	0:55:23	36	1	unable	—
s6669	0:47:15	11	1	unable	—

Table 3: Verification of optimized and retimed circuits

5 Conclusion

We compared structural FSM traversal with standard symbolic FSM traversal. Our theoretical analysis as well as the experimental results show that structural FSM traversal is a practical base algorithm for formal verification applications. For the circuits encountered in practice it has the same convergence behaviour as symbolic traversal. While not suffering from the memory problems encountered with canonical set representations, it allows to exploit structural circuit properties and effective approximations, by which the traversal of the state transition graph can be greatly accelerated.

References

[1] D. Brand, “Verification of Large Synthesized Designs,” in *Proc. Intl. Conf. on Computer-Aided Design (ICCAD-93)*, pp. 534–537, 1993.

[2] R. Bryant, “Graph-based Algorithms for Boolean Function Manipulation,” *IEEE Transactions on Computers*, vol. 35, pp. 677–691, August 1986.

[3] O. Coudert, C. Berthet, and J.-C. Madre, “Verification of Synchronous Sequential Machines Based on Symbolic Execution,” *Lecture Notes on Computer Science*, vol. 407, pp. 365–373, June 1989.

[4] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, “Markovian Analysis of Large Finite State Machines,” *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 1479–1493, Dec. 1996.

[5] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Boston: Kluwer Academic Publishers, 1996.

[6] S. Huang, K. Cheng, K. Chen, and U. Gläser, “An ATPG-Based Framework for Verifying Sequential Equivalence,” in *Proc. Intl. Test Conference (ITC-96)*, 1996.

[7] J. Jain, R. Mukherjee, and M. Fujita, “Advanced Verification Techniques Based on Learning,” in *Proc. 32nd ACM/IEEE Design Automation Conference (DAC-95)*, pp. 420–426, June 1995.

[8] A. Kühlmann and F. Krohm, “Equivalence Checking Using Cuts and Heaps,” in *Proc. Design Automation Conference (DAC-97)*, pp. 263–268, Nov. 1997.

[9] W. Kunz, “An Efficient Tool for Logic Verification Based on Recursive Learning,” in *Proc. Intl. Conference on Computer-Aided Design (ICCAD-93)*, pp. 538–543, Nov. 1993.

[10] W. Kunz and D. Stoffel, *Reasoning in Boolean Networks - Logic Synthesis and Verification Using Testing Techniques*. Boston: Kluwer Academic Publishers, 1997.

[11] Y. Matsunaga, “An Efficient Equivalence Checker for Combinational Circuits,” in *Proc. Design Automation Conference (DAC-96)*, pp. 629–634, June 1996.

[12] K. McMillan, *Symbolic Model Checking*. Boston: Kluwer Academic Publishers, 1993.

[13] C. Pixley, “A Theory and Implementation of Sequential Hardware Equivalence,” *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 1469–1478, Dec. 1992.

[14] S. Reddy, W. Kunz, and D. Pradhan, “A Novel Verification Framework Combining Structural and OBDD Methods in a Synthesis Environment,” in *Proc. Design Automation Conference (DAC-95)*, pp. 414–419, June 1995.

[15] D. Stoffel, *Formal Verification of Sequential Circuits Using Reasoning Techniques*. PhD thesis, Johann Wolfgang Goethe - Universität, Frankfurt am Main, Germany, http://www.em.informatik.uni-frankfurt.de, 1999.

[16] D. Stoffel and W. Kunz, “Record & Play: A Structural Fixed Point Iteration for Sequential Circuit Verification,” in *Proc. Intl. Conference on Computer-Aided Design (ICCAD-97)*, pp. 394–399, Nov 1997.

[17] C. van Eijk, “Sequential Equivalence Checking without State Space Traversal,” in *Proc. Conference on Design, Automation and Test in Europe (DATE-98)*, (Paris, France), pp. 618–623, March 1998.