# Disjunctive Partitioning and Partial Iterative Squaring:
## an effective approach for symbolic traversal of large circuits

Gianpiero Cabodi [†] Paolo Camurati [#] Luciano Lavagno [‡] Stefano Quer [†]

[†] Politecnico di Torino
Dip. di Automatica e Informatica
Turin, ITALY

[#] Università di Udine
Dip. di Matematica e Informatica
Udine, ITALY

[‡] Politecnico di Torino
Dip. di Elettronica
Turin, ITALY

**Abstract**

*Extending the applicability of reachability analysis to large and real circuits is a key issue. In fact they are still limited for the following reasons: peak BDD size during image computation, BDD explosion for representing state sets and very high sequential depth.*

*Following the promising trend of partitioning and problem decomposition, we present a new approach based on a disjunctive partitioned transition relation and on an improved iterative squaring. In this approach a Finite State Machine is decomposed and traversed one "functioning–mode" at a time by means of the "disjunctive" partitioned approach.*

*The overall algorithm aims at lowering the intermediate peak BDD size pushing further reachability analysis. Experiments on a few industrial circuits containing counters and on some large benchmarks show the feasibility of the approach.*

## 1 Introduction

State-of-the-art approaches for state space exploration of Finite State Machines (FSMs) exploit symbolic techniques based on Binary Decision Diagrams (BDDs). But even symbolic techniques reach their limits on large practical examples. As a consequence extending the applicability of the reachability analysis to new fields is a key issue.

Ravi *et. al.* [1] show that, in general, intermediate steps of traversal and intermediate representation are more expensive that final ones. This is due to the fact that even breadth–first traversal reaches "relatively" few states at each iteration; moreover these states are "sparse", in the sense that their code differ for too many bits, and have also "sparse" BDD representation. This generally implies the following limitations on symbolic traversals:

- Large peak BDD size during inner traversal steps.
- BDD explosion for representing state sets.
- High sequential depth, i.e., the state transition graph requires too many traversal iterations.


DAC 97, Anaheim, California


To solve the above problems we propose a method based on the following contributions:

- An approach to extend the application of *disjunctive partitioned transition relations* from asynchronous circuits to synchronous ones modeled as FSMs.
- A *partial* and *incremental iterative squaring* to deal with subsets of the state transition graph characterized by long (counter–like) chains of states.
- The application of the partitioning strategy presented in [2] to solve complex operations required by iterative squaring.

The disjunctive partitioned transition relation [3], allows early smoothing and a mixed used of breadth–first and depth–first traversal. This constitutes a good way to deal with different ordering requirements, by visiting a state transition graph in an "incremental" way, giving preference to those subgraphs whose states are "easy" to represent and traverse. We found experimentally that "modes of operation" of several circuits are "easy" to traverse one at a time. In the current implementation the state graph is partitioned into "modes of operation" *manually* using information coming from the synthesis tool or the designer's knowledge. The goal is to isolate high-level components such as counters, registers, comparators, and so on. Such subgraphs usually have a compact BDD representation and include dense subgraphs or long sequences of states.

Another potential problem, that can be made even worse by partitioning, is the existence of very long state sequences. We use, whenever necessary for each partition, an improved version of iterative squaring [3] to deal with it. The improvement is in computing it incrementally (see Section 4) and partially (i.e., inside one partition). It allows to substitute a set of states with a kind of "macro–state" representing them in a compact way. Like in [1] the distance of a state from the reset states can be overestimated. For example, let us consider an $n$–bit up–counter with a proper order of the variables to create BDDs (as the one with the MSB at the top of the BDD and the LSB at the bottom). If we analyze the BDD representing the set of reached states during the counting phase the newly reached states are incrementally represented in neighboring branches of the BDD (as they are the states represented

as 00...00, 00...01, 00...10, etc.). The BDD then fills–in "progressively" and its maximum dimension is linear in the number of bits of the counter. On the other hand, if we change the variable order or we visit states in a different order, the BDD dimension can grow much more also if the final dimension will be the same. Finding the ideal variable ordering can become impossible, if we consider complex or different requirements (such as representing at the same time different sub-circuits, or sets of states and the transition relation) and anyway it cannot solve the problem completely. In fact, the overall strategy decreases complexity beyond the limit achievable just using different ordering.

This idea is related to the work of Ravi *et. al.* [1]. They introduced the concept of *dense* BDD representing a large number of states with a small number of nodes. They traversed a circuit with a mixed breadth–first and depth–first traversal aimed at increasing density extracting dense subset of the obtained BDDs. Our purpose is to obtain dense BDD by partitioning the graph and "directly" working on its representation with the transition relation representation. The subject is also related with the work of Cabodi *et. al.* [2] where the authors partitioned BDDs representing set of states. The goal there was to achieve a split BDD representation as balanced as possible and with a minimum overhead in term of nodes. The heuristic was based on choosing single splitting variables and using a cofactoring strategy. In this case, we partition the state transition using a strategy based on the topology of the state transition graph of the FSM, and not on the BDD representation of the reachable state set.

Experiments on complex industrials circuits containing counters and on large ISCAS'89 and ISCAS'89–addendum show the feasibility of the approach.

The remainder of the paper is organized as follows. Section 2 summarizes some useful concepts. Section 3 describes the application of the disjunctive partitioned approach. Section 4 describes our *ad-hoc* enhanced iterative squaring. Section 5 shows experimental results. Section 6 closes the paper with a brief summary and reports on future work.

# 2 Preliminaries

## 2.1 The Model

A *Finite State Machine* is an abstract model describing the behavior of a sequential circuit. A completely specified FSM $M$ is a 6–tuple $M = (I, O, S, \delta, \lambda, S_0)$ where $I$ is the input alphabet, $O$ is the output alphabet, $S$ is the state space, $\delta : S \times I \rightarrow S$ is the next state function, $\lambda : S \times I \rightarrow O$ is the output function and $S_0 \subseteq S$ is the initial state set.

In the sequel we will use $s = (s_1, s_2, \ldots, s_n)$, $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_n)$ to indicate respectively the vector of present state, primary input and next state variables.

BDDs will be used to represent functions, relations and sets.

## 2.2 The Transition Relation and Symbolic Traversals

The *transition relation* $\mathsf{TR}$ associated to a FSM $M$ is:

$$\mathsf{TR}(s,x,y) = \prod_{i=1}^{n} (y_i \equiv \delta_i(s,x)) = \prod_{i=1}^{n} tr_i(s,x,y_i)$$

Very often the existence of an input leading from a state to the next is important, rather than its value. Then primary inputs may be abstracted from $\mathsf{TR}$, defining a new relation $\mathsf{T}$

$$\mathsf{T}(s,y) \quad = \quad \exists_x \mathsf{TR}(s,x,y)$$

usually called *non–deterministic transition relation.*

The image of a set of states described by its characteristic function $\mathsf{C}(s)$ according to the transition relation is defined as:

$$\mathbf{Img}(\delta(s,x), \mathsf{C}(s)) \quad = \quad \exists_{s,x} \ (\mathsf{TR}(s,x,y) \cdot \mathsf{C}(s)) \qquad (1)$$

The set of forward reachable state set $\mathsf{R}$ can be computed with the following fixed point computation:

$$\begin{aligned} \mathsf{R}_i(y) &= \mathsf{R}_{i-1}(y) + \mathbf{Img}(\delta, \mathsf{R}_{i-1})(y) \\ &= \mathsf{R}_{i-1}(y) + \exists_{s,x} \ (\mathsf{TR}(s,x,y) \cdot \mathsf{R}_{i-1}(s)) \end{aligned}$$

$\mathsf{R}_0$ is set to the initial set of states and $\mathsf{R}_i$ is the reachable state set at step $i$. In the sequel we will call $\mathsf{R}^*$ the values of $\mathsf{R}_i$ at the fixed point.

The number of iterations of the recurrence equation gives the *sequential depth* of the machine, i.e., the longest of the shortest paths that connect each state to any of the initial states.

## 2.3 Iterative Squaring

Although standard symbolic traversals are usually quite efficient, they become impractical for very high sequential depth. In these cases a systematic method, called *iterative squaring* transformation [3], can result in an exponential reduction in the number of iterations necessary to reach fixed points.

$\mathsf{TR}(s,x,y)$ describes triples of the form $(\hat{s}, \hat{x}, \hat{y})$, where state $\hat{y}$ is reached from state $\hat{s}$ by applying input $\hat{x}$, whereas $\mathsf{T}$ describes pairs of adjacent states, i.e., states that are connected by at least one edge in the transition graph of $M$. The transitive closure $\mathsf{T}^*$ of $\mathsf{T}$ describes the pairs of states that are connected by at least one path in the state graph of $M$. $\mathsf{T}^*$ is found by computing the least fixed point of the following recurrence equations[†]:

$$\mathsf{T}^{2^{i+1}}(s,y) \ = \ \mathsf{T}(s,y) + \exists_z \ (\mathsf{T}^{2^i}(s,z) \cdot \mathsf{T}^{2^i}(z,y)) \qquad (2)$$

setting $\mathsf{T}^1(s,y)$ to $\mathsf{T}(s,y)$. The number of iterations required to compute $\mathsf{T}^*$ is logarithmic in the maximum distance between two states in the state transition graph (i.e., the diameter of the graph).

Finding the set of states reachable from $S_0$ then amounts to computing:

$$\mathsf{R}^*(y) \quad = \quad S_0(y) + \exists_s \ (T^*(s,y) \cdot S_0(s))$$

[†] The notation is not standard, but we introduced it to allow the composition of terms not power of two.

### 2.4 Partitioning

It has been shown experimentally that partitioning BDDs representing set of states can be of great benefits in reachability analysis. In [2] the authors show that calling $v$ the split variable and partitioning the state set in the following way:

$$\begin{aligned}\mathbf{Img}(\delta, C(s)) &= \mathbf{Img}(\delta, (v \cdot C_v(s) + \overline{v} \cdot C_{\overline{v}}(s))) \\ &= \mathbf{Img}(\delta_v, v \cdot C_v(s)) + \mathbf{Img}(\delta_{\overline{v}}, \overline{v} \cdot C_{\overline{v}}(s))\end{aligned}$$

reduces the complexity of the original image computation and results in the efficient use of the secondary memory to store unused BDDs. The pseudo–code is represented in figure 1 and it is derived from the standard traversal.
$\mathsf{R}_p$, $\mathsf{From}_p$ and $\mathsf{Next}_p$ represent sets in monolithic or partitioned form. They are initially set to $S_0$. At each step, for each subset $from$ of $\mathsf{From}_p$ the image computation procedure is called. Images are collected in $\mathsf{To}_p$. This allows the image computation procedure to work on just a subset at a time, decreasing peak BDD size.

```
Partitioned_Traversal (δ, S0, th)
    {
    Rp = Fromp = Nextp = S0;
    while (Nextp ≠ ∅)
        {
        Top = ∅;
        foreach from ∈ Fromp
            Top = (Top, Img (δ, from));
        Nextp = Fromp = Set_Diff (Top, Rp);
        Rp = Set_Union (Nextp, Rp);
        Fromp = Re_Partition (Fromp, th);
        Rp = Re_Partition (Rp, th);
        }
    return (Rp);
    }
```

Figure 1: Partitioned Forward Traversal.

After image computation, functions **Set_Diff**, **Set_Union**, and **Re_Partition** are called. These functions are relatively simple and perform the computation of sets $\mathsf{Next}_p$, $\mathsf{From}_p$, and $\mathsf{R}_p$ for the next iteration. In particular, function **Re_Partition** carries out set union and set splitting, according to the size of their BDD representation and to parameter $th$. The $th$ parameter controls the complexity of the image computation procedure and the size and number of the state set partitions. If necessary sets can be decomposed also inside the function **Img**, also if the pseudo–code does not exemplify this feature.

## 3 Partitioning the Transition Relation

As the monolithic representation for TR can difficult or even impossible to obtain, TR is usually represented as a product of terms. In this case the image computation is generally expensive because existential quantification and logical conjunction cannot distribute.
Analyzing the sets of support of the functions involved in the conjunction, Burch *et. al.* [4] determine whether existential quantification can be moved inside the conjunction (*early quantification*). This results in a simplification as the number of variables in the conjuncted terms is reduced. Several heuristics have been presented to sort the functions. Further improvements are obtained through *clustering* and *tree–like* processing [5], that may be used to decrease the complexity of the problem by performing some products just once before image computations.
This implies to partition the transition relation in a set of terms that are conjuncted together.
Burch *et. al.* [3] also propose a different approach in which the transition relation is partitioned in a set of terms that are left disjoint. They called this approach *disjunctive partitioned transition relation*. They propose the conjunctive transition relation to model synchronous circuits, while they adopt the disjunctive transition relation to deal with asynchronous circuits, because their behavior can be described as a conjunction of sums which is transformed (through the distributive law) in sum of conjunctions. Calling $\mathsf{TR}_i$ the disjunctive partitions of TR, and following Equation (1), we can write:

$$\mathsf{To}(y) = \exists_{s,x}((\textstyle\sum_{i=1}^{m} \mathsf{TR}_i(s,x,y)) \cdot \mathsf{From}(s))$$

In this case the existential quantifier can be pulled inside disjunction and applied to the different terms.

$$\mathsf{To}(y) = \textstyle\sum_{i=1}^{m}(\exists_{s,x}(\mathsf{TR}_i(s,x,y) \cdot \mathsf{From}(s)))$$

Starting from this last idea, we extended its application to synchronous circuits, modeled as FSMs. Conjunctive partitioned transition relations are (either partially or fully) transformed to disjunctive partitioned ones by means of proper decompositions, based on automatic splitting variable selection [2] or, more generally, on a constraining function possibly related to knowledge of the circuit's behavior:

$$\begin{aligned}\mathsf{TR}(s,x,y) = \;& \mathsf{TR}(s,x,y) \cdot p(s,x,y) + \\ & \mathsf{TR}(s,x,y) \cdot \overline{p}(s,x,y)\end{aligned}$$

Splitting can be recursive, leading to the following form:

$$\begin{aligned}\mathsf{TR}(s,x,y) &= \textstyle\sum_{k=1}^{m} \mathsf{TR}(s,x,y) \cdot p_k(s,x,y) \\ &= \textstyle\sum_{k=1}^{m} (\prod_{i=1}^{n} tr_i(s,x,y) \cdot p_k(s,x,y)) \\ &= \textstyle\sum_{k=1}^{m} \mathsf{TR}_{p_k}(s,x,y)\end{aligned}$$

with $\sum_{k=1}^{m} p_k = 1$ and where we call $\mathsf{TR}_{p_k}$ the logical product $\mathsf{TR} \cdot p_k$.
The inner conjunctions can be kept in the partitioned form, or (better) can be performed just once, leading to a disjunctive partitioned transition relation. Primary input variables can be existentially quantified in this phase, resulting in the following expression:

$$\begin{aligned}\mathsf{T}(s,y) &= \exists_x \mathsf{TR}(s,x,y) \\ &= \exists_x (\textstyle\sum_{k=1}^{m} \mathsf{TR}_{p_k}(s,x,y)) \\ &= \textstyle\sum_{k=1}^{m} \mathsf{T}_{p_k}(s,y)\end{aligned}$$

From a behavioral point of view, each term in a conjunctive partitioned transition relation corresponds to a particular sub-circuit, while terms of a disjunctive partitioned transition relation represent subsets of the state transition graph. In the former case, dealing with only a sub-circuit at a time causes approximated (overestimated) analysis of the state space [6]. In the latter case, each subgraph

can be used to represent the exact behavior of the whole FSM under particular constraining conditions. Of course a good choice for constraining functions is a key issue, to discriminate among different behaviors of the FSM (and of its state graph). A heuristic technique, in the case of circuits including counters, is discriminating between counter idle and active, or counter active only on a subset of its bits. In this way partitioning the graph possibly isolates subgraphs with very high sequential depth. Standard breadth–first traversal techniques are usually inefficient with such graphs, and we can resort in this case to partial iterative squaring as a complementary technique to compress a portion of the state graph.

As an example, let us suppose to have a circuit including a counter and other devices, such that for each state of the counter the other devices have a very complex state graph. The overall circuit can be difficult to traverse with standard techniques, but we can partition it in the following way: one mode in which the counter is active and every other device is idle and a second mode in which the counter is idle and the other devices are active. The first mode can be easily traversed with iterative squaring and knowing the corresponding subset of the reachable state space can be of great help in computing the complete reachable states. In fact, this knowledge allows us to analyze simultaneously all the sub-sets of states that can be reached in parallel with the counter states. This again mixes breadth–first and depth–first search, according to the structure of the state transition graph. Moreover, the behavior of the circuit somewhat resembles that of asynchronous circuits (for which the disjunction method was developed), in that the various parts are relatively independent.

## 4 Improving Iterative Squaring

A few drawbacks drastically limit the application of "standard" iterative squaring:

- Computing $\mathsf{TR}$ and $\mathsf{T}$ is very often quite expensive or even impossible.
- The computation of $\mathsf{T}^*$ possibly blows up in few iterations due to the growing size of the BDDs.
- $\mathsf{TR}$, $\mathsf{T}$ and $\mathsf{T}^*$ also consider paths originating at states that cannot be reached from the initial states. Hence squaring may be inefficient if a large fraction of the state graph is unreachable or if the sequential depth of the FSM is low.

For the above reasons iterative squaring is effective on "pure" counters, while this is not the case with other circuits. On the other hand, counters represent a difficult application field for standard traversal techniques, due to their high sequential depth. Moreover our partition strategy applied to the state transition graph identifies subgraphs with very high depth. Taking into account these considerations, we resort to iterative squaring to solve reachability analysis problems whenever necessary and we modify it in the following ways.

We apply iterative squaring inside some partitions substituting $\mathsf{T}_{p_k}$ to $\mathsf{T}$ and $\mathsf{T}^*_{p_k}$ to $\mathsf{T}^*$ in Equation (2).

This kind of computation, that we call *incremental squaring*, decreases complexity as it proceed incrementally. Generally the squaring can be computed incrementally and sequentially on different partitions.

As each step of the squaring can be really complex we apply to each complex step the partitioning strategy analyzed in [2]. This approach is well suited in the squaring computation as at each step two $T^{2^i}$ relations are composed (see Equation (2)); this is done by conjunction–quantification operations, that can fruitfully exploit partitioning, as described in [2]. Moreover as the two terms to compose are equal the same partitioning strategy can be used. We call this kind of computation *partitioned squaring*.

Finally, experimental results show that (like usual breadth–first traversal) also iterative squaring is prone to higher complexity during intermediate steps. Then we often avoid reaching the transitive closure $\mathsf{T}^*_{p_k}$ and we stop squaring at an intermediate iteration, returning a proper subset $\mathsf{T}^k_{p_k}$ of $\mathsf{T}^*_{p_k}$. This has the advantage of reducing the complexity of squaring while lowering traversal iterations to an acceptable amount. We call *partial computation* this approach.

Traversing a partition of a FSM is thus equivalent to computing the least fixed point of the following recurrence equation:

$$\mathsf{R}_i(y) \quad = \quad \mathsf{R}_{i-k}(y) + \exists_{s,x} \ (\mathsf{T}^k_{p_k}(s,x,y) \cdot \mathsf{R}_{i-k}(y))$$

where $R_i$ is defined (as in Section 2.2) as the states reachable from the initial state set in $i$ steps on the original state graph. When transitive closure is used ($\mathsf{T}^k_{p_k} = \mathsf{T}^*_{p_k}$) a single traversal step is enough to compute all reachable states.

## 5 Experimental Results

Our main novelty is to propose a technique based on a disjunctive partitioned transition relation and on iterative squaring. This can deal with some large circuits on which other tools and techniques fail.

We implemented a traversal program built on top of the Colorado University Decision Diagram (CUDD) package. Our experiments ran on a 200 MHz DEC Alpha with a 256 Mbyte main memory, by imposing a working memory limit of 228 Mbyte. We experimented on a set of industrial circuits and a couple of large ISCAS'89 and ISCAS'89–addendum benchmarks. These benchmarks will be analyzed in the next subsection.

### 5.1 Benchmarks

The industrial counters we have used for the experiments reported below originate from the output compare functions of the timing unit of the Motorola 68HC11 microcontroller [7]. A full description of these timer can be found in [8]. Briefly, we have chosen to model and traverse the following functions:

1. oc1_self a self-triggered fixed period counter.
2. oc1_fix a fixed period counter.
3. oc1_prog_abs a programmable absolute counter.
4. oc1_prog_rel a programmable relative counter.

| Circuit | $\|T\|$ | $\|T^*\|$ | # Reached States | # P | $\|T^i\|_{peak}$ | # Iterations | Memory | Time |
|---|---|---|---|---|---|---|---|---|
| oc1_self | 5240 | 12962 | $1.3750 \cdot 10^{10}$ | 1 | 33355 | 21 | 8.3 | 13474 |
| | | | | 1 | 31336 | 21+2 | 7.8 | 11266 |
| oc1_fix | 3817 | 17552 | $2.190 \cdot 10^{11}$ | 1 | *ovf* | *ovf* | *ovf* | *ovf* |
| | | | | 8 | 71695 | 21 | 13.5 | $27.5^h$ |
| | | | | 2 | 35085 | 21+8+4 | 7.8 | 16291 |
| oc1_prog_abs | 1999 | 1691 | $2.190 \cdot 10^{11}$ | 1 | 10587 | 22 | 5.4 | 337 |
| | | | | 2 | 7940 | 22+4 | 4.5 | 68 |
| oc1_prog_rel | 2030 | 1611 | $2.190 \cdot 10^{11}$ | 1 | 9871 | 22 | 4.8 | 258 |
| | | | | 1 | 8370 | 22+4 | 4.6 | 62 |

Table 1: Improved Iterative Squaring results on industrial circuits. *ovf* means overflow on time ($> 36^h$).

These counters are representative of the functions available also in other similar devices, such as the Intel 8254.
Finally we will experiment on circuit s1423 of the ISCAS'89 suite and on s1512 on the ISCAS'89–addendum. The first one despite its relatively small size, 74 memory elements, is very difficult to handle during reachability analysis, whereas the second one has 57 memory elements and a sequential depth equal to 1024 [2].

### 5.2 Experimental evidence

Table 1 reports experiments on industrial benchmarks. We compare in this table the standard iterative squaring with the partitioned one and the incremental–partitioned one. For each circuit various squaring experiments are reported. Circuit indicates the name of the circuit. Columns $|T|$, $|T^*|$ and $|T^i|_{peak}$ indicate the number of nodes of the BDDs representing $T$, $T^*$ and the largest BDD representing $T^i$. # Reached States is the number of states of the reachable state set. # P indicates the maximum number of partitions used to deal with the experiment, following [2]. # Iterations indicates the number of iterations performed in each of the iterative squaring phases (to compute the transitive closure $T^*$ or one of its subsets). This column indicates just a single value if the standard approach or the partitioned–standard one is applied and indicates more than one value if the incremental or incremental–partitioned approach is applied. Each number indicates the subsequent number of steps of one iterative squaring phase. Memory is the maximum working memory size used (in Mbytes). Time indicates the CPU time (in seconds, unless otherwise stated).
Tuning the package is important. For example, in the case of circuit oc1_fix the transitive closure $T^*$ is not directly computable (first row). Anyway we succeeded in squaring it, using the partitioned approach (second row), by taking 8 partitions of the squaring but this still required more than $27^h$. Using the incremental approach, with three steps of iterative squaring with depth 21, 8 and 4, together with the partitioned one with 2 partitions, reduced the peak of the BDDs and the CPU time to 16291 seconds.
The partial approach (stopping squaring before the transitive closure) reduces the CPU time even more in some experiments; for example circuit oc1_fix can be traversed in 3291 seconds if we do just 13 steps of squaring and resort to 256 steps of traversal. Analogously, the time for circuit oc1_self can be reduced to 7890 seconds.
Table 2 and 3 report experiment on a couple of large benchmark from ISCAS'89 and ISCAS'89–addendum suites.
Circuit s1512 is sequentially very deep (1024 levels) but single image computations are not particularly expensive. In [2] Cabodi *et al.* traversed the circuit for the first time using approximately 41 Mbyte of main memory in 42 hours.
Table 2 reports experiments on iterative squaring, using the monolithic and the partitioned relation, on this benchmark. N indicates that $T^N$ is computed and the number of nodes of its BDD is reported in column $|T^N|$. The partitioned approach is quite effective both in term of complexity and of CPU time.
In Table 3, on the other hand, we compare the Standard Approach and the Improved Approach, see [2], with the approach we have just proposed, the Present Approach. # Level reports the depth of the graphs traversed and Disk indicates the amount of secondary memory used.
On circuit s1512 with the present approach we could partition the state transition graph by isolating a significant part that could be traversed in 262 levels but in only 86 seconds. The reachable state set of this part has $4.1980 \cdot 10^7$ reachable states. Using this set as a new initial set for traversal, we could traverse completely the circuit (up to $1.6574 \cdot 10^{12}$ states) in less than 4 hours and with only 14.3 Mbytes of memory.
Again the first traversal allows us to compute a quite dense subset of the reachable state set, represented with only 107 BDD nodes, by obtaining an improvement of one order of magnitude on the CPU time.

| N | $\|T^N\|$ | Standard Approach Time | Partitioned Approach # P | Memory | Time |
|---|---|---|---|---|---|
| 1 | 5060 | 12 | 1 | 2.1 | 12 |
| 2 | 12993 | 40 | 1 | 4.9 | 40 |
| 4 | 34701 | *ovf* | 3 | 5.7 | 48 |
| 8 | 137060 | – | 8 | 12.1 | 7048 |
| 16 | 383607 | – | 26 | 44.6 | $32^h$ |

Table 2: Building the squaring on the ISCAS'89–addendum s1512. *ovf* means overflow on time ($> 36^h$).

The complete traversal is then carried out in two steps:

| Circuit | Statistics | Standard Approach | Improved Approach | Present Approach |
|---|---|---|---|---|
| s1512 | # Level | 1024 | | 262+766 |
| | # Reached States | $1.6574 \cdot 10^{12}$ | | |
| | # Reached Nodes | 1100 | | 592 |
| | Memory (+Disk) | 96 (+0) | 41 (+2.8) | 14.3 (+0) |
| | Time | $64^h$ | $42^h$ | $3.8^h$ |
| s1423 | # Level | 12 | 14 | $\leq$ 4096 |
| | # Reached States | $2.3035 \cdot 10^{10}$ | $1.7945 \cdot 10^{11}$ | $3.0011 \cdot 10^{14}$ |
| | # Reached Nodes | 1361263 | 13738871 | 4615383 |
| | Memory (+Disk) | 116 (+0) | 106 (+125.9) | 89 (+0) |
| | Time | 5410 | $8.4^h$ | $3.6^h$ |

Table 3: Results on reachability analysis on ISCAS'89 s1423 and ISCAS'89–addendum s1512.

the first one equivalent to 262 steps of traversal and the second one equivalent to 766 steps (see column # Level). The main drawback of the actual implementation is that these choices have to be done in a manual fashion.

s1423 is very difficult to handle during reachability analysis, as shown by the extensive experiments described in [1], [2], [6]. Ravi *et. al.* [1] could reach, with a very long run ($>$ 22000 seconds) $1.67 \cdot 10^{14}$ states, by computing dense subsets of the reachable state space. On the other hand Cabodi *et al.* [2] reached exactly level 14, with a reachable set size of $1.7945 \cdot 10^{11}$ and with 13738871 nodes of the BDD representing it. In our current approach we use information presented in [6] on approximate traversal to partition the state transition graph. We could obtain a variable ordering that allows us to represent the transition relation in a monolithic way with 48000 BDD nodes. The best result has been found using different steps of traversal and iterative squaring (with a maximum depth of 4096 steps). Both manual and automatic partitioning have been used. If we compare the density of the BDDs obtained, with the improved approach one BDD node could represent $1.3 \cdot 10^4$ states whereas now it can represent $6.5 \cdot 10^7$ states.

## 6 Conclusions

Symbolic FSM state space exploration techniques represent one of the major recent results of formal verification. One of their limitations is the inability to deal with very complex and deep circuits. Iterative squaring on the other hand is well suited to deal with "pure" counters with a very high depth, and partitioning can reduce the overall complexity of the standard algorithm.

We present a new approach based on a disjunctive partitioned transition relation and on iterative squaring. The disjunctive partitioned representation for the transition relation allows us to represent and visit subsets of the state transition graph that have a compact BDD representation. These subsets, unfortunately, can have very long sequential depth, thus requiring the application of iterative squaring techniques.

The methodology is proved to be effective in traversing counters, circuits containing counters and large circuits in general and can also be used for verification and synthesis purposes [8].

Our current implementation relies on manual choices to partition the circuits. Hence we need to investigate how to use for that purpose information coming from the synthesis tools, or, when such information is not available, how to derive heuristic techniques to partition graphs.

## References


[1] K. Ravi, F. Somenzi, "High-Density Reachability Analysis," in *Proc. IEEE/ACM ICCAD'95*, pp. 154–158, November 1995

[2] G. Cabodi, P. Camurati, S. Quer, "Improved Reachability Analysis of Large Finite State Machine," in *Proc. IEEE/ACM ICCAD'96*, pp. 354–360, November 1996

[3] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, D.L. Dill: "Symbolic Model Checking for Sequential Circuit Verification," *IEEE–Transaction on CAD*, pp. 401–424, Volume 13, Number 4, April 1994

[4] J.R. Burch, E.M. Clarke, D.E. Long: "Representing Circuits More Efficiently in Symbolic Model Checking," in *Proc. ACM/IEEE DAC'91*, pp. 403–407, June 1991

[5] R. Hojati, S.C. Krishnan, R.K. Brayton, "Early Quantification and Partitioned Transition Relation," in *Proc. IEEE ICCD'96*, pp. 12–19, October 1996

[6] H. Cho, G.D. Hachtel, E. Macii, M. Poncino, K. Ravi, F. Somenzi, "Approximate Finite State Machine Traversal: Extensions and New Results," in *Proc. IWLS'95*, May 1995

[7] *M68HC11 Reference Manual*, Motorola inc., 1991

[8] G. Cabodi, P. Camurati, L. Lavagno, S. Quer, "Synthesis and verification of counters based on symbolic techniques," in *Proc. EDAA/IEEE/ACM ED&TC'97*, pp. 176–181, March 1997