Incremental FSM Re-encoding for Simplifying Verification by Symbolic Traversal

S. Quer*† UCB Dept. of EECS Berkeley, CA (USA)

G. Cabodi Politecnico di Torino Dip. Automatica e Informatica Turin (Italy)

P. Camurati Università di Udine Dip. Matematica e Informatica Udine (Italy)

L. Lavagno Politecnico di Torino Dip. Elettronica Turin (Italy)

E.M. Sentovich Cadence Berkeley Labs Berkeley, CA (USA)

R.K. Brayton UCB Dept. of EECS Berkeley, CA (USA)

Abstract

State space exploration of finite state machines is used to prove properties about sequential behavior, such as the equivalence of two machines. The three paradigms for exploring reachable states, forward traversal, backward traversal and a combination of the two, reach their limits on large practical examples. Approximate techniques and combinational verification are far less expensive but these imply sufficient, not strictly necessary conditions. Extending the applicability of the purely combinational check to pairs of machines with different encodings can be achieved through state minimization, partitioning, and re-encoding the FSMs to factor out their differences. This paper focuses on re-encoding*: we present an incremental approach to re-encoding for sequential verification. We transform the product machine traversal to a combinational verification in the best case, and to a computationally simpler product machine traversal in the general case. Experimental results demonstrating the effectiveness of these techniques o medium-large circuits where other techniques may fail.*

1 Introduction

Exploring the state space of a *finite state machine* (FSM) is used to prove properties like *equivalence* [15], *resetability* [14], and *synchronization* [16]. An FSM is identified by its Boolean input and output domains, its initial state set, and its next state and output relations. Many properties of interest can be identified by an *initial state set* and a *bad state set*. Proving that a property holds requires showing that no state in the bad set can be reached by any state in the initial state set. Thus, exploring all states reachable from an initial state set is important for property verification.

Many properties are related to the composite behavior of two FSMs. This behavior is captured by the *product machine*, which is an FSM composed of the two. The inputs are shared and a single output is created that records the differences between the output signals of the two.

A common verification problem is that of comparing two machines that are behaviorally equivalent but structurally different. This is the case when one FSM has been obtained from the other by means of sequential optimization (such as partial or total re-encoding, retiming and resynthesis, sequential redundancy removal, etc. [8]). The equivalence check is used to verify the validity of manual synthesis and optimization steps against a specification, or to find bugs in the synthesis algorithms.

State-space traversal is typically performed via symbolic exploration of the *state transition graph* (STG) using BDDs to represent the next-state functions and reachable state sets. The standard paradigms for symbolic exploration are *forward traversal*, *backward traversal*, and *forward/backward traversal*. Relevant papers on proving equivalence and temporal properties with forward traversal include [2], [5], [19], [1]. The application of backward traversal to equivalence proofs is the subject of [15], [9], [11]. Test pattern generation with forward/backward traversal is the subject of [3]. The inability of exact traversals to deal with very large FSMs fostered the development of *approximate* techniques [6], [7] but the latter support only sufficient and not necessary equivalence checks.

Traversing a product machine's state space is more costly than doing so for a single FSM, since the state space is increased exponentially. In principle this implies exponential growth in the representation and computation of state sets. Fortunately, this is not the case in practice since the component machines are strongly

---

[1]S. Quer is also with the Politecnico di Torino, Dipartimento di Automatica ed Informatica, Turin (Italy).

[2]Contact Address: Quer Stefano, Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso Duca degli Abruzzi 24, I-10129 Turin (Italy), e-mail: quer@polito.it.

coupled by primary input sharing, and they evolve in lockstep during traversal. Furthermore, if the component machines are "similar", they have similar reached state sets which implies a smaller reached state set in the product machine. For example, for two machines with identical encodings the reached state set of the product machine has the same cardinality as the reached state set of one of the constituent machines. It also has a simple structure since the encodings are identical. With similar machines, some correspondence between their state elements may be known, in which case computations can be made more efficient by interleaving corresponding state variables in the BDD ordering. Experimental evidence reported in [19] and [5] indicates that traversing the product machine can cost only two to three times as much as traversing a single one. Thus, in many cases (especially for similar machines), traversing the product machine should be as computationally tractable as traversing a single machine.

In this paper, we address the FSM verification problem, and in particular consider *similar* machines. We deliberately do not define the term "similar" precisely. It indicates (1) identical or nearly identical input/output behavior, (2) a correspondence between the machines that may or may not be explicitly known. The ability to traverse the product machine is closely related to the ability to detect and utilize similar sub-blocks of the component FSMs. A given pair of equivalent machines falls into one of the following categories:

1. State encodings are identical and state variable correspondence is evident. In test pattern generation, for example, each state variable of the fault-free device maps onto an equivalent state variable in the faulty device.

2. State encodings are identical, but state variable correspondence is not known. In this case, variable ordering techniques may fail to detect the optimum variable interleaving and the BDDs may be much larger than necessary. Signature-like algorithms or partner patterns [4] can help in identifying state variable correspondence.

3. State encodings are not identical. In this case, the product machine becomes much more difficult to traverse because the notion of variable correspondence is lost. This case is also problematic for the accuracy of approximate traversal, which is based on functional and/or structural decomposition. It is difficult to group corresponding sub-blocks of the two machines.

We assume similar machines are given (which is often the case in logic synthesis) but their correspondence is not explicit. We propose techniques for *re-encoding* one machine to make it structurally more similar to the other, and hence exploit the underlying correspondence. This makes traversal-based verification more efficient, and in the limit permits using purely combinational verification. The techniques are applicable when the machines are very different, but the results will be best for similar machines. We search for a re-encoding function $\rho_{M^1M^2}$ such that M^1 re-encoded (in one step) by $\rho_{M^1M^2}$ is combinationally equivalent to M^2. Such a function may not exist, and even if it does it may be impossible to compute. The representation of the recoding characteristic function is directly related to the reachable state set of the product machine $M^1 \times M^2$. Therefore, we search for a sequence of simpler re-encoding functions ρ whose successive application to M^1 generates a sequence of FSMs that converge to M^2. Each FSM in the sequence is equivalent to M^1. The reason such a converging sequence of machines should exists is that M^2 could have been obtained from M^1 by a sequence of sequential optimizations whose corresponding re-encoding functions are simple.

If M^1 and M^2 are not state-minimal, we could state minimize both. This issue is beyond the scope of this paper, that deals only with 1:1 re-encodings.

The main contributions of this paper are as follows:

- A conservative verification strategy with efficient state-space traversal that dynamically re-encodes the machines to make them more similar

- A procedure for finding an equivalence-preserving re-encoding once some knowledge of the product machine state space is available; development of the theoretical aspects of finding approximate re-encodings on subsets of state variables

- Experimental evidence of the effectiveness of these techniques on medium-large circuits where state-of-the-art techniques fail.

The remainder of the paper is organized as follows. We assume a basic knowledge of FSMs, FSM equivalence and graph isomorphism, BDDs, and image computation of sets of states. In Section 2, we analyze the requirements of re-encoding and its complexity. Section 3 describes the verification strategy based on re-encoding and combinational verification. Since the equivalence-preserving re-encodings we consider are $1:1$ functions, we show how to obtain them by reducing $m:n$ relations or by extending partial $1:1$ mappings. We give some experimental results in Section 4 to justify the feasibility of this approach. Conclusions and directions for further development are given in Section 5.

2 Verification by Re-encoding

Several algorithms are involved in this work when re-encoding FSMs for more efficient state-space traversal:

- *Heuristics for FSM decomposition and product machine correspondence recognition.* These are important for reducing the traversal cost and increasing approximate traversal accuracy.

- *Exact and/or approximate product machine traversal*, carried out as long as the representation of the reachable states remains simple. The reached states of the product machine are used as a starting point in generating the re-encoding function.

- *Extending partial re-encodings and/or reducing n:m relations to 1:1 functions.* This step is the computational framework for handling incomplete or over-estimating re-encoding informations. In selecting a re-encoding function, trade-offs between low complexity and good re-encoding are explored.
- *Constraining the product machine state space during traversal.* This is mandatory to reduce the complexity of approximate and exact traversals, and to reduce the complexity of the re-encoding function.

State minimization of the initial machines reduces the state space of each machine, and in some cases results in machines with isomorphic STGs. This is an orthogonal technique that has been explored in the literature and will not be discussed here.

The overall algorithm for verification works as follows. We begin with two FSMs M^1 and M^2. If a sufficient equivalence condition holds between M^1 and M^2, the process terminates. Otherwise a re-encoding function ρ is computed and used to transform M^1 into an equivalent FSM M^*. The process can be iterated replacing M^1 with M^*. This iteration yields a *sequence* of FSMs that hopefully converge from M^1 to M^2.

Disregarding the trivial case in which the initial FSMs M^1 and M^2 are already combinationally equivalent, the two may be related in one of the following ways:

1. M^1 and M^2 are equivalent and their STGs are isomorphic: there exists some 1:1 re-encoding function working in one step ($\rho_{M^1M^2}$) or a sequence of re-encoding functions ρ for M^1 resulting in next-state and output functions identical to M^2 when both are restricted to the reachable state set or to an over-estimation of it. We compute the re-encoding function through a converging sequence, because it is computationally easier. A combinational check suffices to demonstrate equivalence.
2. M^1 and M^2 are equivalent but their STGs are not isomorphic. State-minimized versions of them should be extracted to treat the problem as in the previous case. Exact state-minimization is not an easy task especially for large FSMs. Therefore we cannot deal specifically with this problem because the final equivalence check cannot be done combinationally (if the machines are not state-minimized) when only $1:1$ re-encoding functions are considered. However, there may be equivalent state pairs with identical encodings or equivalent state variables. Identification of these reduces the complexity of exact and approximate traversals.
3. M^1 and M^2 are not equivalent. A re-encoding function transforming each reachable state of M^1 into an equivalent reachable state of M^2 does not exist. To determine that the machines are not equivalent, a traversal-based verification is necessary but it is performed on re-encoded, more similar machines. This traversal-based verification may also happen in the previous two cases, when the $\rho_{M^1M^2}$ re-encoding exists but is too complex or difficult to obtain and we resort to a converging sequence of re-encodings ρ.

2.1 Transforming FSMs

An FSM M is given by $M = (I, O, S, \delta, \lambda, S_0)$, where I is the set of input variables, O the set of output variables, S the set of states, δ the set of next-state functions, λ the set of output functions, and S_0 the set of initial states. The input, output, and state variables sets are denoted x, z, and s. The reachable state set is denoted R, where $R \subseteq S$; an over-estimation of this set is denoted R^+, where $R \subseteq R^+$; The product machine of M^1 and M^2 is denoted M^p. Machine equivalence is denoted $M^1 \doteq M^2$; equivalence means identical input/output behavior from the initial state set. The one-step re-encoding function is denoted by $\rho_{M^1M^2}$, the generic re-encoding is denoted by ρ, by $\rho(s^1, s^2)$ to indicate the characteristic function of the $1:1$ mapping, and by the pair $\rho v(s^1)$ and $\rho v^{-1}(s^2)$ to indicate the Boolean functional vectors that re-express s^2 in terms of s^1 and vice-versa.

Re-encoding transforms M^1 into M^*. The goal of the transformation is to make M^* similar to M^2, where similarity is a measure of the number of equivalent states that have identical encodings. M^* is used as the new M^1 for the next re-encoding iteration.

Constraints on ρ Re-encoding should preserve the input/output behavior of M^1:

$$(M^* = \rho(M^1)) \;\Leftrightarrow\; (M^1 \doteq M^*)$$

Among many equivalence-preserving re-encodings, we select ρ to be a $1:1$ mapping from S^1 to $S^* = S^2$, possibly restricted to either R^1 or R^{1+}.

Application of ρ Once the characteristic function of the $1:1$ mapping $\rho(s^1, s^2)$ is known, we apply it to the state elements on the feedback loops. Two Boolean function vectors $\rho v(s^1)$ and $\rho v^{-1}(s^2)$ are generated. The next-state and output functions of M^* are computed as follows:

$$\begin{aligned} \delta^*(s^*, x) &= \rho v(\delta^1(\rho v^{-1}(s^*)), x)) \\ \lambda^*(s^*, x) &= \lambda^1(\rho v^{-1}(s^*), x) \end{aligned}$$

Consequently we obtain $M^* = (I, O, S^*, \delta^*, \lambda^*, S_0^*)$. The transformation only affects the state elements on the feedback loops, so the equivalence of M^1 and M^* is guaranteed by the existence of the inverse ρv^{-1} of ρv. Since ρv is a 1:1 function, ρv^{-1} exists.

Complexity of re-encoding The one step re-encoding function $\rho_{M^1M^2}$ might be too complex to represent and/or to apply. In the worst case, $\rho_{M^1M^2}$ is a canonic function whose complexity is exponential in the number of variables. The size of the BDD of its characteristic function representation is directly related to the reachable state set of the product machine. i.e., to how different the encodings are, to the variable orderings, and to the mutual dependency between subsets of the state variables.

Then we try to find a sequence of simpler re-encodings. Once we have an expressible re-encoding function ρ, the cost of applying it to M^1 is mainly due to the double composition with ρv and ρv^{-1}. As we poorly exploit the don't care set related to the unreachable states, the δ^* and λ^* functions in our current implementation are more complex than necessary.

A technique we adopt to decrease the complexity of ρ, is to re-encode a subset of the state variables at a time. This strategy is particularly suited for networks of interacting FSMs, where independent encodings are applied to disjoint subsets of state elements. Let us partition the s^1 and s^2 variables into two subsets s^{1a}, s^{1b} and s^{2a}, s^{2b}. This is easily generalized to an arbitrary number of partitions. Let us express $\rho(s^1, s^2)$ as:

$$\rho(s^1, s^2) = \rho^a(s^{1a}, s^{2a}) \times \rho^b(s^{1b}, s^{2b})$$

If ρ^a and ρ^b a re $1:1$, so is ρ.

The complexity is further reduced by observing that a partitioned re-encoding can be applied in two steps,

$$\rho_1(s^1, s^2) = \rho^a(s^{1a}, s^{2a}) \times \prod_{i=1}^{|s^b|}(s_i^{1b} = s_i^{2b})$$

$$\rho_2(s^1, s^2) = \rho^b(s^{1b}, s^{2b}) \times \prod_{i=1}^{|s^a|}(s_i^{1a} = s_i^{2b})$$

which ensure a smaller size of the intermediate BDDs, because part of the re-encoding is simply the identity function, i.e., a wire.

3 The Verification Procedure

Verification is done by using partial state-space exploration to iteratively compute a sequence of re-encoding functions. The goal is to use low-cost traversals and yet obtain enough information to compute a useful re-encoding function. Two traversal methods are used:

- i steps of exact forward traversal of M^p, with i much less than the index of the fixed-point. This computation returns R_i^p.
- an approximate forward traversal of M^p, possibly stopping before a fixed-point is reached. This computation returns R^{p+}.

It is also possible to compute a mix of exact but partial information (R_i^p) and complete but approximate information (R^{p+}).

A simplified version of the code is shown in Fig. 1. A new re-encoded machine is generated in each iteration of the outer loop. The loop terminates when equivalence is determined (a complete re-encoding function was found) or the number of allowed iterations is exceeded. In the latter case, the final re-encoded machine is used in the subsequent product machine traversal for determining equivalence.

Each re-encoded machine is computed as follows. First, a heuristic analysis of the product machine M^p is performed with the procedure PM_HEURISTIC. This corresponds to an initial decomposition and re-ordering of M^p and will be analyzed in the next section. The re-encoding is generated and updated in the inner loop. At each iteration, a forward traversal of M^p is performed using either exact or approximate techniques. This generates the set `reached`, which is equal to R_i^p if the exact techniques is used in all the cycles and is equal to R_i^{p+} in the general case. If the graphs are isomorphic, R_i^p represents a $1:1$ correspondence. In the general case both R_i^p and R_i^{p+} are an $m:n$ state correspondence.

Let ρ^- be a $1:1$ under-estimation of ρ (initially, $\rho^- = \emptyset$), and let ρ^+ be a $m:n$ over-estimation of ρ ($\rho^+ =$ `reached`). Information is successively added to ρ^- from ρ^+, while keeping the result $1:1$ (function EXTEND_RE-ENCODING in Fig. 1). This is done at the fixed point or periodically during traversal (when ++level MOD step $= 0$ is true in the pseudo-code), allowing the inner steps to have more information in computing the function. ρ^- is then extended to encompass all state pairs (s^1, s^2) of R^p (function COMPLETE_RE-ENCODING in Fig. 1). This last procedure, for sake of simplicity, extends the encoding to the whole state space, but in practice this is limited to an over-estimation of the reachable states. At the end of each cycle, the function RE-ENCODE computes the re-encoded machine M^* using the previous version and the ρ mapping.

3.1 Approximate forward traversal for the product machine

We resort to exact forward traversal in the algorithm of Fig. 1 only in some cases and for few steps. Otherwise, we are forced to adopt an approximate algorithm. Referring to the pseudo-code of Fig. 1, it is enough to use an approximate image computation procedure.

We refer to [6] and [7] for approximate traversal computation: overestimating reachable states stems from FSM partitioning and loosening mutual constraints and dependencies.

We have already pointed out that traversing a product machine can be significantly more complex than traversing a single one. One of the main issues is to try and keep corresponding next-state functions and variables interleaved.

To identify permutations, we follow an heuristic approach inspired by Chen *et al.* [4] that computes signatures like syndromes and partner patterns associated to input variables and that is implemented in the PM_HEURISTIC function. Suppose the correspondence between the i-th and j-th output functions λ_i^1 and λ_j^2 if the two FSMs is known. Disregarding the primary inputs x, we use syndromes and partner patterns to establish a possibly partial correspondence between state variables s_i^1 and s_j^2. For the next-state functions, whose matching is still uncertain, we repeat a similar procedure, working just on the primary inputs. The correspondence is a simple form of re-encoding, restricted to the permutation of state elements. The shuffling step contributes to a more accurate product machine traversal; however in the general case, we must look for a re-encoding.

Sometimes the re-encodings ρ are too complex, although the above product machine oriented variable ordering strategy is used. It becomes necessary to prune the state space of the product machine during traversal, eliminating those state pairs that are badly coupled. In principle, we should keep just equivalent state pairs, but determining them is a computationally hard task. We therefore consider 0-equivalent state pairs, i.e., pairs for which the outputs don't differ ($\lambda^1 = \lambda^2$). We are working to find better product machine oriented decompositions and/or other constraints to be imposed on the search space. For example, we could work just on a subset of the state variables, restricting investigation to a subset of the state space. If the re-encoding can be expressed as a cartesian product of two, we could consider just one of the two partitions during traversal, implicitly assuming that the other state variables are re-encoded in themselves (i.e., a wire acts as the re-encoding function). This reduces complexity, because for the variables in the second partition it as if we considered a single, rather than a product machine.

3.2 Extending a $1:1$ partial re-encoding

Suppose we are given $A^1 \subseteq S^1$ and $A^2 \subseteq S^2$ of equal cardinality, and a $1:1$ function ρ^- that is defined on a subset of A^1. If a complete re-encoding is required, ρ^- must be extended to the whole set A^1. To do this, we first extract the subsets of A^1 and A^2 included in the ρ^- function, $A^1_{\rho^-}$ and $A^2_{\rho^-}$. Then we consider the bipartite graph $\rho^+ - (A^1_{\rho^-} \times A^2) - (A^1 \times A^2_{\rho^-})$. The vertices in one partition of the graph are states in S^1 that have been reached during traversal and which do not appear in the re-encoding function ρ^-, the states in the second part are reached states in S^2 that have not been re-encoded. An edge in the graph connects a state of S^1 and a state of S^2 that are simultaneously reached in the product machine. We apply a $1:1$ reduction procedure to the graph. This procedure *implicitly* considers each node which has not been matched yet and finds a suitable match. The result is OR-ed with ρ^- to produce a $1:1$ function from A^1 to A^2. (This procedure is guaranteed to find a complete encoding given a complete graph.) The extension procedure is biased by a minimum distance heuristic that is described in the following section.

If A^1 and A^2 have different cardinalities, we look for a $1:1$ mapping of the smaller set onto a subset of the larger one and then extend it according to the above procedure.

The pseudo-code is shown in Fig. 2. EXTEND_RE-ENCODING extends ρ^- using ρ^+. It restricts ρ^+ to states not in ρ^-, uses the reduction procedure of Fig. 3 to reduce ρ^+ to a $1:1$ function, and adds ρ^+ to ρ^-. ρ^+ contains, in general, an over-estimation of the state correspondence which looses accuracy as the traversal proceeds; we heuristically avoid re-considering states that have already been matched. COMPLETE_RE-ENCODING uses EXTEND_RE-ENCODING to extend ρ^- to the whole space of M^p.

3.3 Reducing an $m:n$ relation to a $1:1$ function

The last algorithm to be described is that of reducing an $m:n$ relation $\rho^+ \subseteq A^1 \times A^2$ to a $1:1$ function $\rho^+_{1:1}$.

The relation is given in implicit form as a characteristic function and can be represented by a bipartite graph. Transforming it into a $1:1$ function is equivalent to solving a perfect bipartite matching problem.

If a perfect matching exists, it is also a maximum one. As a consequence, we could use well-known algorithms for maximum matching based on the min-cut max-flow theorem (or use their symbolic versions [10]). Unfortunately the matching problem is in the inner loop of our algorithm, so exact algorithms, though polynomial in the number of states, would still require an excessive amount time.

We propose a quick heuristic algorithm, based on Lin's *et al. cproject* operator [12] and Tamisier's *cross-section* [18], without guarantee of maximality.

Depth-first recursive processing is currently biased by a minimum distance criterion from a special re-encoding minterm σ of the product machine. The re-encodings computed at the previous steps can be seen as a set of minterms. We set individual variables of σ based on the behavior of the majority of corresponding variables in the minterms set. This can be considered as a mechanism to favor re-encodings of the complete state space which are similar to those which have already been determined more precisely.

The pseudo-code is shown in Fig. 3. Let ρ^+ be the BDD representing the characteristic function of the $n:m$ relation. Let σ be a set of literals in the product machine state space (initially the special minterm for minimum distance selection).

After testing terminal recursion cases the procedure SELECT_LITERAL selects a splitting literal (v) from σ and uses it for recursive traversal of (in theory, all) paths from the root to a 1 leaf. Each path represents an encoding for s^1 and a corresponding state s^2. Recursion proceeds on the two cofactors of ρ^+ with respect to the splitting literal, producing the then (T) and else (E) components of the result. The reduction is guaranteed to be $1:1$ by an existential quantification on $\rho^+_{\overline{v}}$, before computing the E component that prevents the re-use of mapped states. If $v \in A^1$, existentially quantifying all s^1 results in P, i.e., all the states in A^2 in the subspace that has already been considered. The state in P should therefore not be further considered. The procedure of Fig. 3 returns the BDD of a $1:1$ function, without any guarantee of completeness. Random choices are made in selecting a $1:1$ function contained in ρ^+. The pseudo-code does not show the use of cache tables to store previously computed results, which greatly improves efficiency.

4 Experimental Results

The verification by recoding technique has been implemented in a prototype software written in C. We report the results of the experiments performed on a *DEC ALPHA 7000* with *256MByte* of memory. The circuits used are taken from the well known ISCAS'89 (widh '93 addendum) and MCNC set. Different encodings have been obtained either by random re-encoding or using the *state_assign* command of SIS ([17]). Due to the lack of tools able to provide different encodings for large FSMs, we also generated large circuits by separately encoding and then grouping together some smaller ones.

Table 1 demonstrates the importance of good product machine coupling (see section 3.1) for exact and approximate traversals. We present product-machine traversal for two instances of the same small-medium circuit to allow comparison between exact and approximate techniques. A bad initial coupling was obtained by manually shuffling one machine's variables from the ideal interleaved ordering. The left part of Table 1 shows that this has a negative impact especially in approximate traversals (which are based on circuit decomposition). #FF indicates the number of state variables, #P indicates the number of partitions used in approximate traversal (#P = 1 indicates exact traversal). Level is the number of traversal steps (image computations) and $\#R^+$ shows the number reachable states. Time is the CPU time in seconds. The right part shows the effectiveness of the PM coupling heuristic (routine PM_HEURISTICS), called before traveral. The gain in terms of CPU time and accuracy (for approximate traversals) is evident.

The next tables concern verification by recoding. We do not include data about small FSMs, because exact traversal techniques are always convenient with them, due to the relatively small number of state variables and of reachable states.

In Table 2 we list four experiments similar to those of Table 1. We perform self-verification after manually shuffling product machine variable ordering (to emulate the case of unknown correspondence), the goal of re-encoding being to find the exact variable correspondence. Some column labels have already been explained, the new ones are $|\rho|$) (size of the re-encoding function in terms of BDD nodes) and #Cycle (number of recoding steps preformed until equivalence is proved. All verifications in Table 2 are performed in 1 recoding cycle. This means that the re-encoding was easy to find and to express: in fact, after applying PM_HEURISTICS, all remaining bad variable couplings were well *localized* in the partitions used for approximate traversal. With the addition of the pruning constraint ($\lambda^1 = \lambda^2$) the R^{p+} computed is 1 : 1 and contains the exact re-encoding. In the TIME column CPU times in brackets refer to the cost of applying ρ to transform the initial machine in the re-encoded one: this appears to be a relevant phase in the overall process, except for circuit s713, in which the coupling heuristic was enough for combinational equivalence, and our re-encoding based verification was not necessary. With circuit s1423, the initial manual variable shuffling was *limited* (not far from ideal interleaving), because due to the size of the circuit we were not able to deal with a more general variable ordering. Anyway this experiment shows the success in proving equivalence for a large circuit where standard exact techniques fail. Lines $\mathsf{s1423}_a$ and $\mathsf{s1423}_b$ (concerning the same starting problem) differ in the accuracy of approximate traversal, and demonstrate that a looser traversal may sometimes provide an exact recoding in less time.

Table 3 presents the verification of medium-large machines composed of smaller FSMs. We used equivalent but not equal circuits (like s344 and s349) or circuits with different state assignments obtained using SIS. In the latter case we start from the STG of the machine in *kiss* format, and we synthesize it with different state assignment strategies (i.e. different options for *Nova* [20] are used to obtain equivalent machines). Using SIS implies dealing with small circuits: then we adopt *serial* (+) and *parallel* ($\|$) composition to build larger FSMs. The first two experiments were performed in 1 cycle again, due to the good combination of approximate traversals and space constraining. The other ones evolve in 3 cycles, and column $|\rho|_{avg}$ reports the average size of the applied recoding functions.

In Table 4 we list some examples of independent encodings for sub-blocks of a large circuit. Given a FSM, we randomly re-encode subsets of the state variables, and check the new circuit against the original one. In the verification phase, we exploit the advantage of iterative recoding (see section 2.1) by restricting some recoding steps to subsets of the state variables. Column Re-encoding list the sizes of the re-encoded sets of state variables: e.g. 7, 7 with 28 FFs means that two sets of 7 state variables each are re-encoded, while 14 state variables keep their original coding. Possible tunings for these experiments involve varying the accuracy of the approximate traversal and the number of times the procedure EXTEND_RE-ENCODING is used during traversal. The experimental data result from a trade-off between accuracy and cost. The level of accuracy in the approximate forward traversal was biased to guarantee satisfactory guesses for re-encoding, while keeping the traversal time low.

Again, the standard exact forward traversal is the best solution with the smaller circuits (s400 and s713), while recoding outperforms it with larger ones. In the case of sbc, we compare the single step re-encoding (sbc_a) with a three-step one (sbc_b), and the advantage of the latter is clear in term of both size of recoding and CPU time. Circuits s1269 and s1512 are selected among those that we cannot traverse exactly: the ratio of re-encoded state variables is low, but the experiments prove the ability of dealing with large circuits, provided that a proper decomposition can be found for re-encoding. At present we have no exact or heuristic method for re-encoding oriented circuit decomposition, thus we used for the above experiments manual decomposition, exploiting knowledge of the partitioning used to randomly encode circuits.

5 Conclusions

We presented an incremental approach to re-encoding one FSM composing a product machine. This technique is based on an approximate forward traversal of the product machine to obtain information for re-encoding. The equivalence-preserving re-encoding function is computed from the over-estimation by means of reductions and extensions to functions. The re-encoding functions allow us to extend the applicability of combinational checks and, in case they fail, to make traversals more efficient.

The current limits of this technique are:

- We can currently handle large circuits only if recoding is done in blocks and the partitioning information is provided to the tool. No heuristics have been developed.
- The extension of the $1:1$ function is based on a simple distance criterion.
- We still insufficiently exploit the don't care set given by the unreachable states to simplify δ and λ functions.

and these topics are the object of future work.

This approach covers verification cases where the traditional approaches fail or are quite costly. It works at its best on possibly large machines composed by networks of interacting FSMs, each simple enough to be re-encoded efficiently, but whose coupling is not loose enough to consider them as completely independent blocks.

References

[1] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, D.L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," in *IEEE Transactions on CAD*, Vol. 13, No. 4, April 1994, pp. 401–424

[2] O. Coudert, C. Berthet, J.C. Madre, "Verification of sequential machines based on symbolic execution," *Lecture Notes in Computer Science 407*, Springer Verlag, 1989, pp. 365–373

[3] G. Cabodi, P. Camurati, S. Quer, "Efficient State Space Pruning in Symbolic Backward Traversal," in *Proc. IEEE ICCD'94*, October 1994, pp. 230–235

[4] D.I. Cheng, M. Marek-Sadowska, "Verifying Equivalence of unctions with Unknown Input Correspondence," in *Proc. EDAC'93*, February 1993, pp. 81–85

[5] H. Cho, G. Hachtel, S.W. Jeong, B. Plessier, E. Schwarz, F. Somenzi, "ATPG Aspects of FSM Verification," in *Proc. IEEE ICCAD'90*, November 1990, pp. 134–137

[6] H. Cho, G. Hachtel, S.W. Jeong, B. Plessier, E. Schwarz, F. Somenzi, "Algorithms for Approximate FSM Traversal," in *Proc. ACM/IEEE DAC'93*, June 1993, pp. 25–30

[7] H. Cho, G. Hachtel, E. Macii, M. Poncino, F. Somenzi, "A Structural to State Space Decomposition for Approximate Reachability Analysis," in *Proc. ICCD'94*, October 1994, pp. 236–239

[8] G. De Micheli, "Synthesis and optimization of digital circuits," McGraw-Hill, 1994

[9] T. Filkorn, "A Method for Symbolic Verification of Synchronous Circuits," in *Proc. IFIP CHDL'91*, April 1991, pp. 229–239

[10] G. Hachtel, F. Somenzi, "A Symbolic Algorithm for Maximum Flow in 0-1 Networks," in *Proc. IEEE ICCAD'93*, November 1993, pp. 403–406

[11] S. Krischer, "The Backward Walk Approach in FSM Verification," in *Proc. IFIP CHDL'93*, April 1993, pp. 143–150

[12] B. Lin, A. Richard Newton, "Implicit Manipulation of Equivalence Classes Using Binary Decision Diagrams," in *Proc. IEEE ICCD'91*, October 1991, pp. 81–85

[13] B. Lin, H.J. Touati, A. Richard Newton, "Don't Care Minimization of Multi-Level Sequential Logic Networks," in *Proc. IEEE ICCAD'90*, November 1990, pp. 414–417

[14] C. Pixley, G. Beihl, "Calculating Resetability and Reset Sequences," in *Proc. IEEE ICCAD'91*, November 1991, pp. 376–379

[15] C. Pixley, "A computational theory and implementation of sequential hardware equivalence," *AMS/DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 3, 1991, pp. 293–320

[16] C. Pixley, S.-W. Jeong, G.D. Hachtel, "Exact Calculation of Synchronization Sequences Based on Binary Decision Diagrams," in *Proc. ACM/IEEE DAC'92*, June 1992, pp. 614–619

[17] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Technical Report UCB/ERL M92/41, U.C. Berkeley, May 1992

[18] T. Tamisier, "Computing the Observable Equivalence Relation of a Finite State Machine," in *Proc. IEEE ICCAD'90*, November 1990, pp. 184–187

[19] H. Touati, H. Savoj, B. Lin, R.K. Brayton, A. Sangiovanni-Vincentelli, "Implicit enumeration of finite state machines using BDDs," in *Proc. IEEE ICCAD'90*, November 1990, pp. 130–133

[20] T. Villa, "NOVA. State assignment of finite state machines for optimal two-level logic implementations," in *Proc. ACM/IEEE DAC'89*, June 1989, pp. 327-332

```
VERIFY_BY_RE-ENCODING (M^1, M^2, step, max_iterations)
{
    M* = M^1;
    n_iterations = 0;
    while (1)
        {
        if (M^2 ≐ M*) return (φ, equivalent);
        M^p = BUILD_PM (M^2, M*);
        if (++n_iterations > max_iterations)
            /* M^p must be verified by traversal */
            return (M^p, aborted);

        /* initializations */
        level = 0; ρ^- = 0; from = reached = new = S_0^p;

        /* M^p traversal */
        M^p = PM_HEURISTIC (M^p);
        while (new != 0)
            {
            to = IMAGE (M^p, from);
            new = to · ¬reached;
            reached = reached + new;
            if (++level MOD step == 0)
                ρ^- = EXTEND_RE-ENCODING (M^p, ρ^-, reached);
            }
        ρ^- = EXTEND_RE-ENCODING (M^p, ρ^-, reached);

        ρ = COMPLETE_RE-ENCODING (M^p, ρ^-);
        M* = RE-ENCODE (M*, ρ);
        }
}
```

Figure 1: Re-encoding-based verification

Circuit	#FF	#P	Approximate Forward			Approximate Forward with initial Re-encoding			
			Level	$\#R^+$	Total Time	Level	$\#R^+$	Re-ordering Time	Total Time
s349	15	3	7	$27.02 \cdot 10^7$	19.95	6	18689	0.03	0.32
		1	7	2625	8.30	7	2625	0.03	1.2
s400	21	3	14	$6.78478 \cdot 10^5$	15.25	37	26525	0.08	3.72
		1	151	8865	58.3	151	8865	0.08	13.03
s420	16	4	11	576	0.15	19	432	0.13	0.25
		1	17	17	0.32	17	17	0.13	0.30
s713	19	4	7	$70.5167 \cdot 10^6$	45.80	6	8704	0.47	1.10
		1	7	1544	6.08	7	1544	0.42	2.02
s953	29	5	12	$53.2978 \cdot 10^8$	132.6	12	204568	0.22	2.03
		1	11	504	7.28	11	504	0.18	2.88
s1238	18	3	3	292040	77.98	3	37141	1.92	2.83
		1	3	2616	25.97	3	2616	1.92	27.75
sbc	28	7	6	$10.2087 \cdot 10^{13}$	30.03	5	$14.377 \cdot 10^6$	1.50	2.00
		4	6	$47.0374 \cdot 10^{12}$	2086.92	6	$10.2989 \cdot 10^6$	1.48	3.20
		1	-	-	-	9	$15.4592 \cdot 10^4$	1.54	130.50

Table 1: Experimental results on Approximate Forward Traversal. - means unknown, i.e. overflow in BDD nodes

EXTEND_RE-ENCODING $(M^p,\ \rho^-,\ \rho^+)$
{
    assert (is_11 (ρ^-));
    s^1 = first_machine_state_vars (M^p);
    s^2 = second_machine_state_vars (M^p);

    / restrict ρ^+ to states not in ρ^- */*
    $\rho^+ = \rho^+ \cdot \overline{\exists_{s^1}(\rho^-)}$;
    $\rho^+ = \rho^+ \cdot \overline{\exists_{s^2}(\rho^-)}$;

    / 1 to 1 filter */*
    $\rho^+_{1:1}$ = REDUCE $(M^p,\ \rho^+,\ s^1 + s^2)$;

    / extend ρ^- */*
    $\rho^- = \rho^- + \rho^+_{1:1}$;

    return ρ^-;
}

COMPLETE_RE-ENCODING $(M^p,\ \rho^-)$
{
    S^p = machine_state_space (M^p);

    return EXTEND_RE-ENCODING $(M^p,\ \rho^-,\ \mathsf{S}^p)$;
}

Figure 2: Pseudo-code to extend the re-encoding function

REDUCE $(M^p,\ \rho^+,\ \sigma)$
{
    / testing terminal cases */*
    if $(\rho^+ == 0)$ return 0;
    if (empty (σ))
        {
        assert $(\rho^+ == 1)$;
        return 1;
        }

    s^1 = first_machine_state_vars (M^p);
    s^2 = second_machine_state_vars (M^p);

    / recursion */*
    v = SELECT_LITERAL (σ);
    T = REDUCE $(M^p,\ \rho^+_v,\ \sigma_v)$;
    if $(v \in s^1)$
        $P = \exists_{s^1}$ T;
    else
        $P = \exists_{s^2}$ T;
    E = REDUCE $(M^p,\ \rho^+_{\overline{v}} \cdot \overline{P},\ \sigma_v)$;

    return (BUILD_BDD $(v,\ \mathrm{T},\ \mathsf{E})$);
}

Figure 3: Pseudo-code to reduce the re-encoding function

Circuit	#FF	Exact Forward			Re-encoding Strategy			
		#Level	#R	Time	$\#N_p$	$\lvert\rho\rvert$	#Cycle	Time
s400	21	151	8865	13.03	3	689	1	7.32 (3.65)
s713	19	7	1544	2.02	-	0	0	0.50 (0)
$s1423_a$	74	-	-	-	16	723	1	347 (113)
$s1423_b$	74	-	-	-	8	1014	1	520 (187)

Table 2: Experimental results on Large FSM with manual re-encoding. - means unknown, i.e. overflow in BDD nodes

$Circuit_1/Circuit_2$	#FF	Exact Forward			Re-encoding Strategy			
		#Level	#R	Time	#P	$\lvert\rho\rvert_{avg}$	#Cycle	Time
$(s344+s510_{ie})$ $(s349+s510_{ig})$	21	20	17249	19.8	3	244	1	3.7 (1.0)
$s510_{ioh}+(s641\|\|s820_{ia})$ $s510_{iov}+(s713\|\|s832_{ie})$	30	47	$18.14\cdot10^5$	145	3	987	1	85 (57)
$(s344+s510_{ie})\|\|s1488_{ia}$ $(s349+s510_{ig})\|\|s1494_{ih}$	27	25	$7.77\cdot10^5$	423	3	734	3	39 (27)
$s298_{ig}\|\|s820_{ia}\|\|(s344+s420_{ioh})$ $s298_{ih}\|\|s832_{ie}\|\|(s349+s420_{iov})$	50	-	-	-	5	1232	3	234 (115)

Table 3: Experimental results on Large composed FSMs. - means unknown, i.e. overflow in BDD nodes. $_{ig}$, $_{ih}$, $_{ioh}$, etc. indicate different options for the *state_assign* command of SIS, using Nova

$Circuit_1$	#FF	Re-encoding	Exact Forward			Verify by re-encoding			
			#Level	#R	Time	#P	$\lvert\rho\rvert_{avg}$	#Cycle	Time
s400	21	11, 10	151	8865	7.3	3	316	3	32.2 (21.2)
s713	19	10, 9	7	1544	5.1	4	139	3	23.2 (16.5)
sbc_a	28	7, 7	10	$15.4592\cdot10^4$	600.00	2	698	1	302 (197)
sbc_b	28	7, 7	10	$15.4592\cdot10^4$	600.00	2	187	3	101 (61)
s1269	37	5	-	-	-	13	231	2	72 (43)
s1512	57	5	-	-	-	12	257	4	231 (123)

Table 4: Experimental results on Large FSM with random encoding. - means unknown, i.e. overflow in BDD nodes