Enhancing FSM Traversal by Temporary Re-Encoding

Gianpiero Cabodi † Paolo Camurati # Luciano Lavagno ‡ Enrico Macii †
Massimo Poncino † Stefano Quer † Ellen Sentovich $

† Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY

‡ Politecnico di Torino
Dip. di Elettronica
Torino, ITALY

Università di Udine
Dip. di Matematica e Informatica
Udine, ITALY

$ Ecole des Mines de Paris
Centre de Mathématiques Appliquées
Sophia–Antipolis, FRANCE

Abstract

Synthesis and optimization of large finite–state machines has improved dramatically over the last few years with the introduction and rapid improvement of symbolic–state manipulation techniques. The algorithms efficiently visit each reachable state in the machine while computing and storing information about these states. We propose a new technique for improving the efficacy of traversal algorithms: re-encoding the states of the machine to more efficiently represent state sets or state transitions, or to more efficiently compute the next set of states. Our technique can be embedded in existing traversal algorithms. Experiments reveal that re-encoding can indeed reduce the time and/or space required for traversal.

1 Introduction

Breadth-first state search methods, combined with symbolic manipulation of finite sets based on BDDs, have been investigated extensively in the past few years to obtain high–performance finite state machine (FSM) traversal algorithms. These algorithms have become invaluable for machine verification, optimization, and test generation. State-of-the-art traversal algorithms can be characterized as follows:

- They are either *exact* (i.e., they visit precisely the reachable states) or *approximate* (i.e., they visit a superset [1] or a subset [2] of the states).
- They traverse states in a *forward*, *backward*, or *mixed forward/backward* [3] fashion.
- They carry out the computations using either the *transition function* or the *transition relation* (possibly in *partitioned/clustered* form [4, 5]) of the FSM.
- They use sophisticated symbolic algorithms based on BDDs for efficient state–set and transition relation representation and manipulation (e.g., [6]).
- They use a variety of techniques for reducing the size of the BDDs, e.g., *Dynamic variable reordering* [7] and *truth–table permutations* [8].

Even with the improvement these algorithms have made, traversing very large state machines is still limited by the size of the reachable state set and the size of intermediate data representations at each computation step.

Reducing the size of BDDs and improving the performance of exact traversal algorithms is possible by re-encoding. There are, in essence, two options. First, one may re-encode reached states during traversal, trying to reduce the size of the BDD representing this set of states. Second, one may re-encode the transition relation to simplify its representation, and therefore make image computation less memory and time consuming. In principle, the two transformations may be applied simultaneously.

Our experience has shown that re-encoding state sets very often has a negative impact on the performance of traversal. On the other hand, re-encoding the transition relation is more complex than re-encoding state sets, because this is done by considering a set of transitions rather than a set of states, but also more promising. Each transition is a present–state/next–state pair, so the desired "dense" set (leading to a compact BDD) requires the re-encoding of the present–state and the next–state variables. These cannot be considered independently: Once a re-encoding for a present–state is determined, the re-encoding for the next–states cannot be chosen arbitrarily. Then, the re-encoding functions that make a transition relation small are usually difficult to find, extremely large, and hard to apply.

The solution we propose here is to perform only a temporary and partial re-encoding of the transition relation, with the objective of simplifying each individual step of image computation. It is partial because it considers only a subset of the state variables. It is temporary because, after each image computation, the original encoding of the transition relation is restored by application of the inverse transformation. Clearly, this technique trades space for time, since application of the direct and the inverse transformations may be time consuming. However, it has allowed us to get improvements over existing traversal algorithms in terms of the size of the FSMs it is able to handle.

The remainder of the paper is organized as follows. Section 2 summarizes notation and background definitions. In Section 3 we propose our new techniques for applying partial and temporary re-encoding to the transition relation during FSM traversal. In Section 4 we detail possible strategies to choose a good re-encoding relation. Section 5 reports preliminary experimental results. Finally, Section 6 is devoted to conclusions.

2 Background

Notation

- Lower–case letters denote Boolean Functional Vectors (BFVs), e.g., $f = (f_1,\ f_2,\ \ldots, f_n)$, or Boolean Variable Vectors (BVVs), e.g., $x = (x_1,\ x_2,\ \ldots, x_n)$;
- Subscripts denote single components, e.g., f_i, x_i;
- Subscripts like $\leq i$ denote the components with index $j \leq i$, e.g., $x_{\leq i} = (x_1, x_2, \ldots, x_i)$, and subscripts like $\geq i$ denote the components with index $j \geq i$, e.g., $x_{\geq i} = (x_i, x_{i+1}, \ldots, x_n)$.

Boolean Operators

Given a BFV, $f(x)$, of BVV x, the *positive* and *negative cofactors* of f with respect to x_i are defined as:

$$\begin{aligned} f_{x_i} &= f(x_{\leq(i-1)}, 1, x_{\geq(i+1)}) \\ f_{x_i'} &= f(x_{\leq(i-1)}, 0, x_{\geq(i+1)}) \end{aligned}$$

Given a BFV, $f(x)$, of BVV x, the *existential quantification* of f with respect to x_i is defined as:

$$\exists_{x_i} f = f_{x_i} + f_{x_i'}$$

Sets and Characteristic Functions

Given a Boolean space B^n, a set S of minterms in B^n can be represented by its *characteristic function*, $\chi_S(x)$, which satisfies the following property:

$$s \in S \iff \chi_S(s) = 1$$

for all $s \in B^n$. $\chi_{B^n} = 1$ and $\chi_\phi = 0$.

Generalized Cofactor and Constrain

Given two Boolean functions f and g, the *generalized cofactor* of f with respect to g, denoted by f_g, is any function which satisfies:

$$f \cdot g \leq f_g \leq f + g'$$

f and f_g agree in the onset of g, whereas f_g may take any value in the offset of g.

Generalized cofactors satisfy the Boole expansion theorem

$$f = g \cdot f_g + g' \cdot f_{g'}$$

Since generalized cofactors are incompletely specified by definition, we use the *constrain* operator [9], denoted as $\downarrow$, to find a generalized cofactor with a small BDD representation.

$$(f \downarrow g)(r) = \begin{cases} f(r) & \text{if} \quad g(r) = 1 \\ f(s) & \text{if} \quad g(r) = 0 \end{cases}$$

where s is the minterm such that $g(s) = 1$ and the distance between minterms r and s is minimum.

Image and Pre-Image

Let $\Xi \subseteq A \times A$ be a relation between the elements of set A, and let x and y represent single elements of A. Then $\Xi(x, y) = 1$ if and only if x is in relation Ξ with y. Let Q be a subset of A. The *image* of Q under Ξ is the set of elements of A which are in relation Ξ with the elements of A belonging to Q. Formally:

$$Img(\Xi, Q) = \exists_x (\Xi(x, y) \cdot Q(x))$$

The *pre-image* of Q under Ξ is similarly defined:

$$Pre(\Xi, Q) = \exists_y (\Xi(x, y) \cdot Q(y))$$

Finite State Machines

A *finite state machine* (FSM), M, is defined as the 5-tuple $M = (X, Z, S, S^0, R)$, where X is the input alphabet, Z is the output alphabet, S is the finite set of states, S^0 is the set of initial states, and $R \subseteq X \times S \times S \times Z \rightarrow \{0, 1\}$ is the *global relation*. We have that $R(x, s, y, z) = 1$ if and only if, under input $x \in X$, the FSM makes a transition from present–state $s \in S$ to next–state $y \in S$ outputting $z \in Z$. A FSM can be represented by a *state transition graph* (STG), whose nodes are elements of $s \in S$ and edges are labeled with pairs $(x, z) \in X \times Z$.

In the sequel, whenever it is unnecessary to distinguish state and input variables in our computations, we use a single variable u that denotes the union of the two. Let n be the number of state elements.

Given the global relation R of a finite state machine, M, the *transition relation*, Δ, and the *output relation*, Λ, of M are defined as:

$$\begin{aligned} \Delta(u, y) &= \exists_z R(u, y, z) \\ \Lambda(u, z) &= \exists_y R(u, y, z) \end{aligned}$$

Re-Encoding Relations

Given a relation $T(u, y)$, expressing a mapping between an input space U and an output space Y (with $u \in U$, and $y \in Y$), the re-encoded relation $\widehat{T}$, obtained by re-encoding the U and Y spaces into the re-encoded spaces $\widehat{U}$ and $\widehat{Y}$, is defined as:

$$\widehat{T}(\widehat{u}, \widehat{y}) = \exists_{u,y} (\rho_U(\widehat{u}, u) \cdot T(u, y) \cdot \rho_Y(y, \widehat{y}))$$

where ρ_U and ρ_Y are the relations that represent the input and output transformations, respectively.

Symbolic FSM Traversal

The FSM traversal problem consists of computing the set of reachable states of a given FSM starting from a set of initial states. The approach based on breadth–first search, and called *symbolic FSM traversal*, involves a fixed point computation of the reachable set of states.

A simplified description of the breadth–first traversal procedure is presented in Figure 1.

```
procedure BFS_Traversal (Δ, S⁰) {
    from = reached = S⁰;
    while (1) {
        to = Img (Δ, from);
        new = to · ¬reached;
        if (new = ∅) then
            return (reached);
        reached = reached + new;
        from = new;
    }
}
```

Figure 1: Symbolic FSM Traversal Algorithm.

Partitioned Transition Relation

The next value, y_i, of the generic present–state variable, s_i, is a Boolean function of u, i.e., inputs and present–state:

$$y_i \equiv f_i(u), \quad \forall_i$$

The transition relation can be expressed as:

$$\Delta(u, y) \quad = \quad \prod_{i=1}^{n} \Delta_i(u, y_i)$$

where $\Delta_i(u, y_i) = (y_i \equiv f_i(u))$.
Given a set of states $from(u)$, its image $to(y)$ under $\Delta(u, y)$ is computed by existential quantification of $\prod_i \Delta_i$ conjoined or cofactored with *from* (s):

$$\begin{aligned} to(y) &= \exists_u (from(u) \cdot \textstyle\prod_{i=1}^{n} \Delta_i(u, y_i)) \\ &= \exists_u (\textstyle\prod_{i=1}^{n} \Delta_i(u, y_i) \downarrow from(u)) \end{aligned}$$

Analyzing the sets of support of the functions involved in the conjunction, Burch *et al.* [5] determine whether existential quantification can migrate inside the conjunction (*early quantification).* This results in a simplification as the number of variables in the conjunction term is reduced. Clearly, the order in which each Δ_i is conjoined to the already formed product is key in permitting extensive early quantification. Several heuristics have been presented to sort the partitions. Further improvements are obtained through clustering, that may be used to decrease the number of partitions by performing some products once forall before image computations.

3 Re-Encoding

Several variants to the truth table permutation method of [8] have been tried to reduce the size of entire sets of states during traversal. Unfortunately, a noticeable reduction in size of some sets usually corresponds to an even more remarkable increase in the size of the BDDs representing other state sets involved in the computation. Alternative strategies, inspired by the *subsetting* technique of [2], may also be used. In this case, a set of states is reduced by:

- Identifying a dense subset, D, of the original set, i.e., a set containing very many states from the original set but with a compact BDD representation;
- Re-mapping the minterms (representing states) outside D onto minterms that can be easily added to the BDD of D without a dramatic size increase.

Clearly, there is a lot of freedom in the choice of the re-mapping function, and it may be possible to control the sizes of the other state sets involved in the traversal.
As a good state set re-encoding is not necessarily a good re-encoding for the transition relation, nor does it guarantee to make image computations easier re-encoding only state sets may actually prove to be a disadvantage in terms of performance of the traversal algorithm. For this reason, we focus primarily on transition–relation re-encoding.
Since global re-encodings of the transition relation are difficult to find and apply, we propose a temporary and partial re-encoding strategy which is applied to state sets at each image computation. Re-encoding is partial because only a subset of the state variables is affected, and temporary because, during each image computation, we first modify the transition relation and the partial products; the image state set is computed with this new encoding, but it is soon restored to the original one before proceeding to the next traversal step.
During one step of image computation, the conjunction of new Δ_i terms to a partial product and the existential quantification of some u variables are performed in an interleaved fashion. After k steps within one image computation we have the following situation:

$$to(y) = \exists_{u_b}((\exists_{u_a}(from(u) \cdot \prod_{i=1}^{k} \Delta_i(u, y_i))) \cdot \prod_{i=k+1}^{n} \Delta_i(u_b, y_i))$$

where u_a is the set of early quantified variables after k steps, and $u_b = u - u_a$, is the set of variables that still need to be quantified.
We denote the product $\exists_{u_a}(from(u) \cdot \prod_{i=1}^{k} \Delta_i(u, y_i))$ as $P^k(u_b, y_{\leq k})$. Then, we have:

$$to(y) \quad = \quad \exists_{u_b}(P^k(u_b, y_{\leq k}) \cdot \textstyle\prod_{i=k+1}^{n} \Delta_i(u_b, y_i)) \qquad (1)$$

Re-encoding can partially reduce the size of P^k. Clearly, P^k grows as new Δ_i's are conjoined to it, and it may blow up before image computation terminates.
Note that the present–state and next–state variables can be re-encoded separately because the re-encodings are only temporary. This means that u and y can be decoupled, admitting more opportunities for reducing the size of P^k.
There are two choices for re-encoding:

- Re-encoding P^k in the u domain. This would require re-encoding both P^k and the $\Delta_{\geq k+1}$ terms since they are conjoined. This is difficult to do efficiently because a better encoding for P^k may result in larger BDDs for the $\Delta_{\geq k+1}$'s. However, no inverse re-encoding is required because the u variables are quantified out at each image computation step.
- Re-encoding P^k in the y domain. This requires the application of an inverse re-encoding to obtain the *to* set in the original space for the next image computation. However, the $\Delta_{\geq k+1}$ terms remain unchanged (P^k and $\Delta_{\geq k+1}$ depend on disjoint sets of y variables).

We have chosen to re-encode P^k in the y domain using the following strategy. Given a re-encoding $\rho(y_{\leq k}, \widehat{y})$ (a mapping from a set of encoded states represented by $y_{\leq k}$ to a new set of encoded states represented by $\widehat{y}$) we apply it to P^k to obtain the re-encoded partial product $\widehat{P^k}$:

$$\widehat{P^k}(u_b, \widehat{y}) \quad = \quad \exists_{y_{\leq k}}(P^k(u_b, y_{\leq k}) \cdot \rho(y_{\leq k}, \widehat{y}))$$

Note ρ is in general a relation and that the number of $\widehat{y}$ variables can be different than k.
After application of ρ to P^k, the image computation resumes, resulting in the re-encoded $\widehat{to}$ set:

$$\widehat{to}(\widehat{y}, y_{\geq k+1}) \quad = \quad \exists_{u_b}(\widehat{P^k}(u_b, \widehat{y}) \cdot \prod_{i=k+1}^{n} \Delta_i(u_b, y_i))$$

$\widehat{to}$ now must be decoded by applying the inverse transformation $\rho^{-1}(\widehat{y}, y_{\leq k})$:

$$to(y) \quad = \quad \exists_{\widehat{y}}(\widehat{to}(\widehat{y}, y_{\geq (k+1)}) \cdot \rho^{-1}(\widehat{y}, y_{\leq k}))$$

Clearly, the overall process is worthwhile if the global cost is lower than the original image computation, i.e we choose ρ so that $|\widehat{P^k}| \ll |P^k|$, $|\rho| \leq |P^k|$.
The above procedure can be used in two ways:

- Static: ρ and $\widehat{P^k}$ are computed once and for all.
- Dynamic: ρ and $\widehat{P^k}$ are computed at each image computation step.

The first solution is less expensive but does not take into account the $from(u)$ contribution. The second solution is more expensive, as we have to compute a re-encoding function at each step, but is more efficient in term of BDD reduction. Intermediate solutions are also possible, and re-encoding may be applied more than one time within a single image computation.

4 Choosing the Re-Encoding $\rho(y_{\leq k}, \widehat{y})$

In principle, we would like to obtain a re-encoded partial product, $\widehat{P^k}$, such that $\widehat{y}_i = u_i$ for as many variables as possible. This heuristic choice has the beneficial effect of providing a regular present– and next–state relation.
Let us select the partial re-encoding as:

$$\rho(y_{\leq k}, \widehat{y}) \quad = \quad P^k(u_b, y_{\leq k})\big|_{u_b \longrightarrow \widehat{y}}$$

If we select P^k as the transformation ρ (with proper variable relabeling $u_b \longrightarrow \widehat{y}$), the size of the resulting $\widehat{P^k}$ is possibly minimum. In fact, with this choice next states and present states have similar codes. Any given next state is re-encoded to the set of present states connected to it in P^k. This tends to produce similar present and next state codes in $\widehat{P^k}$, but it is not an equivalence preserving transformation in the case of $m : n$ P^k relations. To ensure correct transformations, we must guarantee that non equivalent states described by y variables are re-encoded to disjoint sets of codes in $\widehat{y}$. This is obtained by properly pruning $\widehat{P^k}$ from multiply mapped $\widehat{y}$ codes.
Note that the inverse re-encoding is

$$\rho^{-1}(\widehat{y}, y_{\leq k}) = \rho(y_{\leq k}, \widehat{y}) = P^k(\widehat{y}, y_{\leq k})$$

The application of ρ to P^k, taking into account the pruning effect described above, yields $\widehat{P^k}^*$:

$$\widehat{P^k}^*(u_b, \widehat{y}) = prune(\exists_{y_{\leq k}}(P^k(u_b, y_{\leq k}) \cdot P^k(\widehat{y}, y_{\leq k})))$$

Let us consider ρ and P^k as mappings among cubes in the involved state spaces. We restrict each $\widehat{y}$ cube in $\widehat{P^k}$ to the cubes with minimum distance from the corresponding u cube in P^k. We allow equal re-encodings only for equivalent present states in P^k. The only constraint we must satisfy is that for every transition (u, y) in P^k there must be one transition $(u, \widehat{y})$ in $\widehat{P^k}^*$.
Intuitively, we associate to each present–state code the code of all the present–states that share with it at least one next–state; then we prune, according to the minimum distance criterion.
>From an algorithmic point of view, procedure `Re-Encode` computes $\widehat{P^k}^*$ in one shot, avoiding the costly steps of computing the conjunction of P^k and ρ, then the existential quantification, and eventually the pruning. The pseudo-code is shown in Figure 2.

procedure `Re-Encode` $(P^k(u_b, y_{\leq k}))$ {
  $\rho^{-1}(\widehat{y}, y_{\leq k}) = P^k(\widehat{y}, y_{\leq k})$; $\widehat{P^k} = P^k$;
  for_each $(y_i \in \texttt{Support}(P^k))$ {
    $f_0 = \widehat{P^k}^*_{y_i'} \downarrow \exists y_i \widehat{P^k}^*$;
    $f_1 = \widehat{P^k}^*_{y_i} \downarrow \exists y_i \widehat{P^k}^*$;
    $a = \texttt{Insert\_Ident\_Var}\ ((f_0 \cdot f_1'), u_b, \widehat{y})$;
    $b = \texttt{Insert\_Ident\_Var}\ ((f_0' \cdot f_1), u_b, \widehat{y})$;
    $c = f_0 \cdot f_1$;
    $\widehat{P^k}^* = \exists_{y_i} \widehat{P^k}^* \cdot (a + b + c)$;
  }
  return$(\widehat{P^k}^*(u_b, \widehat{y}), \rho^{-1}(\widehat{y}, y_{\leq k}))$;
}

Figure 2: The `Re-Encode` Algorithm.

Intuitively, the re-encoding algorithm eliminates by quantification each y_i variable from P^k, re-encoding it to $\widehat{y}$s. There are two relevant points for efficiency: first, the $\widehat{y}$ variables are inserted with fixed polarity, to produce minimal BDD perturbations. Second, while eliminating the y_i's, we keep their "trace" in P^k, i.e., the y_i portion of P^k is partially substituted by a set of $\widehat{y}_i$ variables added in such a way to represent (and hence allow to re-build) all the information carried by the y_i's that we just removed.
To keep a trace of a y_i, we first find its on–set, f_1, and off–set, f_0, in the original P^k; then we derive the subspaces where its polarity is fixed to 0, a, to 1, b, or may vary, assuming either value 0 or 1, c. In subspaces a and b, where polarity is determined uniquely, y_i is re-encoded

with $\widehat{y}_i$. In subspace c no $\widehat{y}$ is introduced. Notice that the y^* variables are introduced to replicate the non re-encoded y's. Function `Insert_Ident_Var` (f, u_b, $\widehat{y}$) inserts the $\widehat{y}$ variables with fixed polarity along all the arcs exiting the u_b nodes in the BDD of function f: $\widehat{y}_i = 1$ on left arcs, and $\widehat{y}_i = 0$ on right arcs. This process can be repeated again in successive steps of image computation, when new y variables appear in P^k.

To see the advantage of this method, we must analyze how complex it is to compute:

$$\begin{aligned} to(y) &= \exists_{\widehat{y}}(\exists_{u_b}(\widehat{P^k}(u_b,\widehat{y}) \cdot \textstyle\prod_{i=k+1}^{n} \Delta_i(u_b, y_{\geq(k+1)}))\cdot \\ &\quad \rho^{-1}(\widehat{y}, y_{\leq k})) \end{aligned}$$

where $\rho^{-1}(\widehat{y}, y_{\leq k}) = P^k(\widehat{y}, y_{\leq k})$ instead of Equation 1. Since $|\widehat{P^k}| \ll |P^k|$, there is a clear advantage in performing $\exists_{u_b}(\widehat{P^k} \cdot \prod_{i=k+1}^{n} \Delta_i)$ instead of $\exists_{u_b}(P^k \cdot \prod_{i=k+1}^{n} \Delta_i)$. The difference in the two approaches resides in the application of the inverse transformation. Inverse re-encoding is a sort of delayed conjunction, performed when all possible simplifications have already been done. The $\widehat{y}$ variables represent a mask that avoids quantifying a portion of the present–state space. The advantage with respect to normal early quantification is that the latter is horizontal, i.e., an u_i variable is kept (along all BDD paths of partial products) if it appears somewhere in the terms not already conjoined. Our technique seems more selective because before performing inverse re-encoding, it performs path/variable masking on quantification.

Example

We illustrate the transformation that P^k undergoes during re-encoding with a simple example that follows the algorithm in Figure 2. Given an initial P^k:

$$P^k(u_0, u_1, y_0, y_1) \quad = \quad \{0010, 0011, 0111, 1\text{-}01\}$$

In the first iteration, with $\widehat{P^k}^*$ initially set to P^k, the state–space is split according to variable y_0, using the cofactors

$$\begin{aligned} f_0(u_0, u_1, y_1) &= \widehat{P^k}^*_{y_0'} \downarrow \exists y_i \widehat{P^k}^* = \{1\text{--}\} \\ f_1(u_0, u_1, y_1) &= \widehat{P^k}^*_{y_0} \downarrow \exists y_i \widehat{P^k}^* = \{0\text{--}\} \end{aligned}$$

to compute the subspaces:

$$f_0 \cdot f_1' = \{1\text{--}\} \quad f_0' \cdot f_1 = \{0\text{--}\} \quad f_0 \cdot f_1 = \{\}$$

After variable insertion, we have

$$\begin{aligned} a(u_0, \widehat{y}_0, u_1, \widehat{y}_1, y_1) &= \{11\text{---}\} \\ b(u_0, \widehat{y}_0, u_1, \widehat{y}_1, y_1) &= \{00\text{---}\} \\ c(u_0, \widehat{y}_0, u_1, \widehat{y}_1, y_1) &= \{\} \end{aligned}$$

where $\widehat{y}_0$ is inserted into a and $\widehat{y}_0'$ is inserted into b. Finally, combining these subspaces yields

$$\widehat{P^k}^*(u_0, \widehat{y}_0, u_1, \widehat{y}_1, y_1) \quad = \quad \{000\text{-}0, 000\text{-}1, 001\text{-}1, 11\text{--}1\}$$

Similarly, in the second iteration we have:

$$\begin{aligned} f_0(u_0, \widehat{y}_0, u_1, \widehat{y}_1) &= \widehat{P^k}^*_{y_1'} \downarrow \exists y_i \widehat{P^k}^* = \{000\text{-}\} \\ f_1(u_0, \widehat{y}_0, u_1, \widehat{y}_1) &= \widehat{P^k}^*_{y_1} \downarrow \exists y_i \widehat{P^k}^* = \{\text{----}\} \end{aligned}$$

$$f_0 \cdot f_1' = \{\} \quad f_0' \cdot f_1 = \{001\text{-}, 1\text{---}\} \quad f_0 \cdot f_1 = \{000\text{-}\}$$

$$\begin{aligned} a(u_0, \widehat{y}_0, u_1, \widehat{y}_1) &= \{\} \\ b(u_0, \widehat{y}_0, u_1, \widehat{y}_1) &= \{0011, 11\text{--}\} \\ c(u_0, \widehat{y}_0, u_1, \widehat{y}_1) &= \{0000\} \\ \widehat{P}^k(u_0, \widehat{y}_0, u_1, \widehat{y}_1) &= \{0000, 0011, 11\text{--}\} \end{aligned}$$

5 Experimental Results

We have implemented the re-encoding–based traversal algorithm using CUDD [10] as underlying BDD package.

We have used for the experiments some large examples taken from the ISCAS'89 ADDENDUM and the MCNC'91 benchmark suites. In addition, some optimized circuits were considered. Their characteristics are summarized in Table 1. For each circuit, columns PI, PO, FF, Gates and Levels report the number of primary inputs, primary outputs, latches, gates, and maximum number of combinational logic levels, respectively.

Circuit	PI	PO	FF	Gates	Levels
sbc	40	56	28	1011	22
dsip_sim	229	198	224	4109	21
s1269	19	10	37	625	35
s5378_Opt	35	49	121	2565	25
s3271	27	14	116	1715	28
s6669_Opt	84	55	204	3403	93

Table 1: Circuit Characteristics.

Circuit s5378_Opt is an optimized version of the original s5378 ISCAS'89 benchmark, which has been obtained by iterative application of the redundancy removal procedure of [11], based on the approximate traversal of the state machine. The resulting circuit is sensibly smaller and contains 58 less latches than the original one. Circuit s6669_Opt is an optimized version of the original s6669 ISCAS'89 ADDENDUM benchmark, which has been obtained by removing state elements that are not input to any other state element, i.e., their influence is restricted to the circuit primary outputs. The resulting circuit contains 35 less latches than the original one.

In Table 2 we compare the results obtained using different traversal strategies. Columns headed Traditional report data obtained with our FSM traversal program without any re-encoding mechanism enabled, while columns Static Re-Encoding and Dynamic Re-Encoding show results obtained using the two re-encoding–based traversal methods described in this paper.

For each circuit, we first report its sequential depth (column Depth), the total number of reachable states (column R States), and the size (in terms of BDD nodes) of set R (column R Size). Then, for each traversal method, we show the number of nodes of the largest BDD created during traversal (column Nodes) and the total traversal time, in seconds (column Time).

Experiments were run on a DEC–ALPHA with 256 MB of memory, and the limit on the number of nodes in the BDDs was set to 4 millions.

Circuit	Depth	R		Traversal Method					
		States	Size	Traditional		Static Re-Encoding		Dynamic Re-Encoding	
				Nodes	Time	Nodes	Time	Nodes	Time
sbc	10	154592	4277	150432	60	92187	50	77960	54
dsip_sim	16	$2.69{\cdot}10^{67}$	6923	52468	430	41887	320	22459	395
s1269	10	$1.13{\cdot}10^{9}$	612	1032354	1322	741990	1134	647896	1245
s5378_Opt	43	$2.58{\cdot}10^{17}$	24721	49781	954	37045	694	28865	756
Global Savings						29%	21%	40%	11%

Table 2: Traversal Results.

Circuit	Traditional			Static Re-Encoding			Dynamic Re-Encoding		
	Depth	States	Time	Depth	States	Time	Depth	States	Time
s3271	6	$2.04{\cdot}10^{19}$	11213	6	$2.04{\cdot}10^{19}$	253	6	$2.04{\cdot}10^{19}$	185
	-	-	-	9	$4.96{\cdot}10^{26}$	56200	9	$4.96{\cdot}10^{26}$	51400
s6669_Opt	2	$3.74{\cdot}10^{39}$	53	2	$3.74{\cdot}10^{39}$	34	2	$3.74{\cdot}10^{39}$	26
	-	-	-	3	$1.90{\cdot}10^{45}$	645	3	$1.90{\cdot}10^{45}$	530

Table 3: Partial Traversal Results. - means overflow on BDD nodes.

To the best of our knowledge, no results have been published in the literature for circuit s1269, and nobody has gone beyond level 12 in traversing circuit s5378_Opt in an exact way [2]. We have been able to completely traverse these two examples. Experiments have shown that some intermediate image computations are difficult, as proved by the peak on the number of BDD nodes, though the final size of the reached set is small.

By taking a global look at Table 2, we can notice the effect of re-encoding on the performance of the traversal algorithm. In fact, both the peak on the number of BDD nodes and the execution times are considerably smaller whenever re-encoding is employed during traversal (see line Global Savings in the table for the details). The same behavior can be observed on the average BDD size.

For benchmarks s3271 and s6669_Opt the traversal did not complete. However, approximate traversal results tell us that we are not that far from termination. In Table 3 we report partial traversal data for these two circuits. Using the traditional traversal algorithm we could reach level 6 for s3271 and level 2 for s6669_Opt. By enabling the re-encoding mechanism, we have been able to go up to level 9 and level 3, respectively. Time and intermediate numbers of BDD nodes are reduced too.

6 Conclusions

Symbolic FSM state space exploration techniques represent one of the major recent results of formal verification. Its limits are data representations (state sets and transition relations) and manipulations (image computation). In this paper we propose a new technique for improving its efficacy: *re-encoding* the states of the machine to more efficiently compute the next set of states. Our technique can be embedded in existing traversal algorithms. Results show that with re-encoding it is possible to to improve existing traversals.

Acknowledgments

We wish to thank Fabio Somenzi, CU Boulder, for providing us with the CUDD package and for his comments on an earlier draft of this paper.

References

[1] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, F. Somenzi, "Algorithms for Approximate FSM Traversal," *DAC–30*, pp. 25–30, Dallas, TX, Jun. 1993.

[2] K. Ravi, F. Somenzi, "High–Density Reachability Analysis," *ICCAD–95*, pp. 154–158, San Jose, CA, Nov. 1995.

[3] G. P. Cabodi, P. Camurati, S. Quer, "Symbolic Exploration of Large Circuits with Enhanced Forward/Backward Traversals," *EURODAC–94*, pp. 22–27, Grenoble, France, Sep. 1994

[4] H. Touati, H. Savoj, B. Lin, R. K. Brayton, A. Sangiovanni–Vincentelli, "Implicit Enumeration of Finite State Machines Using BDDs," *ICCAD–90*, pp. 130–133, Santa Clara, CA, Nov. 1990.

[5] J. R. Burch, E. M. Clarke, D. E. Long: "Representing Circuits More Efficiently in Symbolic Model Checking," *DAC–28*, pp. 403–407, San Francisco, CA, Jun. 1991.

[6] G. P. Cabodi, P. Camurati, "Exploiting Cofactoring for Efficient Symbolic FSM Traversal Based on the Transition Relation," *ICCD–93*, pp. 299–303, Cambridge, MA, Oct. 1993.

[7] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *ICCAD–93*, pp. 42–47, Santa Clara, CA, Nov. 1993.

[8] M. Fujita, Y. Kukimoto, R. K. Brayton, "BDD Minimization by Truth Table Permutations," *IWLS–95*, Paper 5c, Lake Tahoe, CA, May 1995.

[9] O. Coudert, C. Berthet, J. C. Madre, "Verification of Sequential Machines Based on Symbolic Execution,", *LNCS*, Vol. 407, pp. 365–373, Springer–Verlag, 1989.

[10] F. Somenzi, *CUDD: University of Colorado Decision Diagram Package – Release 1.0.7*, Technical Report, Dept. of ECE, University of Colorado at Boulder, Nov. 1995.

[11] H. Cho, G.D. Hachtel, E. Macii, M. Poncino, K. Ravi, F. Somenzi, "Approximate Finite State Machine Traversal: Extensions and New Results," *IWLS–95*, Paper 3a, Lake Tahoe, CA, May 1995.