Auxiliary Variables for Extending Symbolic
Traversal Techniques to Data Paths

Gianpiero Cabodi  Paolo Camurati  Stefano Quer
Politecnico di Torino
Dipartimento di Automatica e Informatica
Turin, Italy

Abstract—Symbolic state space traversal techniques are best on control-dominated circuits, not on data paths. This paper extends their applicability to data paths. We use auxiliary variables to decompose Boolean functions and to manipulate them in decomposed form. Experimental results demonstrate the gain both in terms of BDD size and CPU time.

I. INTRODUCTION

Symbolic state space traversal techniques are one of the most notable recent achievements in the fields of formal verification, automated synthesis, testing, and diagnosis.

Their model, the Finite State Machine (FSM) [1], makes them suitable for control dominated circuits, not for data paths. In the latter case, the state explosion problem is evident: each flip-flop could make the state space double and in the data path long chains of memory elements, such as registers, are very common. The contribution of this paper is to extend the applicability of symbolic traversal techniques to data paths.

Auxiliary variables decompose Boolean functions in simpler ones, then we manipulate such decomposed forms. Examples of these functions are the transition relation [2], [3], [4], [5] of the FSM or the characteristic functions of the state sets produced by image or pre-image computations. Early results were presented in [6] and [7].

Auxiliary variables are temporary because they exist for simplifying functions, then they are existentially quantified and disappear. Another major result of this paper shows that quantification can be delayed by one clock cycle, without affecting the validity of the proofs or the traversal of the state space. This corresponds to the insertion in the circuit of “dummy” memory elements. They result in additional search space pruning, because relevant information is added to the transition relation.

Auxiliary variables are an alternative to variable ordering strategies, especially when different sub-blocks [8], [9], [10] or state transition function and set representations [11] require contradictory orderings. They can improve performances even when the ordering is poor, yielding results that are comparable with the ones obtained with a good ordering.

Section 2 describes the role of auxiliary variables in simplifying functions. Section 3 shows where they can be placed, manually or automatically. Experimental results support the claim that, with the introduction of auxiliary variables, symbolic traversal techniques can be applied to data paths.

II. THE ROLE OF AUXILIARY VARIABLES

A completely specified FSM $M$ is 6-tuple $M = (I, O, S, \delta, \lambda, S_0)$ where $I$ ($O$) is the input (output) space, $S$ is the state space, $\delta$ ($\lambda$) is the state transition (output) function, and $S_0$ is the set of initial states [1]. All spaces are Boolean.

In this paper we represent the state transition function $\delta$ as a transition relation [2], [3], [4], [5]. With abuse of notation, we give the function’s name to its characteristic function.

Let $X = (x_1, \ldots, x_n)$ be the input vector variable, let $S = (s_1, \ldots, s_k)$ and $Y = (y_1, \ldots, y_k)$ be the current and the next state vector variables, respectively. The transition relation $T_M$ of FSM $M$ contains all the couples (current state $S$ - next state $Y$) such that there is at least an input value $X$ that brings the FSM from state $S$ to state $Y$: $T_M(Y, S) = \exists X \prod_{i=1}^{k} (y_i = \delta_i(X, S))$. 

This work has been partially supported by the ESPRIT Working Group 6018 "CHARME-2".
FSM symbolic forward traversal consists in a least fixed-point computation and requires the ability to calculate images of a given state set according to δ.

A. Function Decomposition with Auxiliary Variables

Let f be a function of the vector variable X = (x₁, ..., xₙ). Given a Boolean functional vector G(X) = (g₁, ..., gₘ), f can be expressed as a composition f(X) = h(X, G(X)), the functions of G acting as inputs to h.

Composition is a heavy task, but, according to a result proven in [6], h(X, G(X)) has an alternative, more efficient representation. Let \( \tilde{X} = (\tilde{x}_1, ..., \tilde{x}_m) \) be a vector of auxiliary variables, let \( f \) be a function derived from \( f \) such that its set of support includes both \( X \) and \( \tilde{X} \), and let \( \tilde{x}_1 \equiv g_1 \), \( \tilde{x}_2 \equiv g_2 \), ..., \( \tilde{x}_m \equiv g_m \). Then f can be expressed as:

\[
f(X) = \exists_{\tilde{X}} (f(\tilde{X}, \tilde{X})) \prod_{i=1}^{m} (\tilde{g}_i \equiv g_i)
\]  

(1)

An even better form of \( f(X) \) is obtained exploiting the "exist" cofactor of [5] that allows existential quantification and logical and to distribute:

\[
\exists_{X}(a(X,Y) \cdot b(X,Y)) = \exists_{X} a(X,Y) \cdot \exists_{X} ((\exists_{X} a(X,Y))
\]

For sake of simplicity, let us consider just one \( \tilde{x} \equiv g_1 \); combining the previous result on cofactoring with the use of auxiliary variables, \( f \) becomes:

\[
f(X) = \exists_{\tilde{x}} (f(\tilde{x}, \tilde{x}) \cdot \exists_{\tilde{x}} (\tilde{g} \equiv g(x)))
\]

The generalization to \( m \) auxiliary variables is trivial. As cofactoring in general simplifies the functions, it can result in considerable savings.

B. Auxiliary Variables and the Transition Relation

There are two main reasons that explain the possibly large size of the BDDS when the transition relation is used: the complexity of the state transition functions \( \delta_t \) and the size of the state sets that result from image computations.

Auxiliary variables can help in both cases. For sake of comprehension, let \( \tilde{X} \) and \( \tilde{Y} \) be the auxiliary variables we introduce on the state transition functions and on the state sets, respectively, although their nature is identical.

1) Application to state transition functions: let us express the \( \delta_t \) functions with \( m \) auxiliary variables \( \tilde{X} = (\tilde{x}_1, ..., \tilde{x}_m) \), denoting with \( \tilde{g}_i \) the function derived from \( \delta_t \), whose set of support includes \( \tilde{X} \). One can show that the transition relation becomes:

\[
T_M(Y, S) = \exists_{\tilde{X}} (f(\tilde{X}, S, \tilde{Y}) \cdot \prod_{i=1}^{m} (\tilde{g}_i \equiv \tilde{g}_i(X, S, \tilde{X})))
\]

where logical and-ing and existential quantification on \( \tilde{X} \) distribute.

The introduction of auxiliary variables results in simpler BDDS because the \( (\tilde{g}_i \equiv \tilde{g}_i(X, S, \tilde{X})) \) terms are simpler and because terms \( (\tilde{x}_i \equiv g_i(X, S, \tilde{X})) \) are used to cofactor the \( (\tilde{g}_i \equiv \tilde{g}_i(X, S, \tilde{X})) \) ones with an additional simplification.

2) Application to state sets: auxiliary variables can also provide significant benefits when the state transition functions are individually simple, but building the transition relation is impossible, because its BDD becomes too large.

Let \( f(X, S, Y) = \prod_{i=1}^{k} (y_i \equiv \delta(X, S)) \). Considering \( m \) auxiliary variables \( \tilde{Y} = (\tilde{y}_1, ..., \tilde{y}_m) \) and the corresponding \( \tilde{h}_i \) functions, the application of (1) yields:

\[
f(X, S, Y) = \exists_{\tilde{X}} (f(\tilde{X}, S, Y, \tilde{Y})) \cdot \prod_{i=1}^{m} (\tilde{y}_i \equiv \tilde{h}_i(X, S, Y, \tilde{Y}))
\]

where \( f \) denotes the function derived from \( f \) whose set of support includes \( \tilde{Y} \). The transition relation becomes:

\[
T_M(Y, S) = \exists_{\tilde{X}} (f(\tilde{X}, S, Y, \tilde{Y})) \cdot \prod_{i=1}^{m} (\tilde{y}_i \equiv \tilde{h}_i(X, S, Y, \tilde{Y}))
\]

As explained in the previous paragraph, cofactoring reduces the complexity of the BDDS also in this case. It is easy to see that the case of \( \tilde{X} \) variables is included in this more general case.

C. Auxiliary Variables and Image Computation

Auxiliary variables \( \tilde{Y} \) are introduced to simplify Boolean functions during intermediate computations and then disappear, because they are existentially quantified. As a consequence, the transition relation remains unchanged. An alternative consists in postponing existential quantification to the next traversal step when the auxiliary variables appear as current state variables. The transition relation is transformed, images are more easily computed, and proofs are still valid on the transformed machine, provided only the original outputs are observed.

For sake of simplicity, let us consider only \( \tilde{Y} \) variables and let us analyze how the modified transition relation

\[
T_M(Y, S, \tilde{Y}) = \exists_{\tilde{X}} (f(\tilde{X}, S, Y, \tilde{Y})) \cdot \prod_{i=1}^{m} (\tilde{y}_i \equiv \tilde{h}_i(X, S, Y, \tilde{Y}))
\]

affects image computation, pre-images being dealt with in a similar way.

Starting from a current state set \( \text{Curr}(S) \), after logically and-ing it with \( T_M(Y, S, \tilde{Y}) \) and existentially quantifying \( S \), one gets a next state set with variables \( Y \) and \( \tilde{Y} \) as support. This state set will become the current state set at the next step and thus an eventual relabeling of the form \( Y/S \) and \( \tilde{Y}/S \) is needed:

\[
\text{Next}(S, \tilde{S}) = (\exists_{\tilde{Y}} (T_M(Y, S, \tilde{Y}) \cdot \text{Curr}(S)))_{Y/S, \tilde{Y}/S}
\]

The image of the original transition relation could be easily computed as:

\[
\text{Next}(S) = (\exists_{\tilde{Y}} (T_M(Y, S, \tilde{Y}) \cdot \text{Curr}(S)))_{Y/S}
\]

but an explicit existential quantification on \( \tilde{Y} \) would waste the advantages of introducing the auxiliary variables. Instead, noting that, moving one step ahead in time, Next*
becomes $\text{Curr}^*$ and that $\text{Curr}(S) = \exists_S^\delta(\text{Curr}^*(S, \delta))$, we can write:

$$\text{Next}^*(S, \delta) = (\exists_S^\delta(T^\delta_y(Y, S, \delta) \cdot \text{Curr}^*(S, \delta)))_{Y'/S}$$  (2)

Formula (2) shows that the existential quantification of $\delta$ is delayed by a clock tick, i.e., it is not explicitly performed until $\delta$ becomes $\delta$.

This approach has an interpretation on the structure of a circuit, as it corresponds to the introduction of “dummy” state elements on relevant points.

Adding this kind of auxiliary variables may introduce in the transition relation an information that wouldn’t otherwise appear. This information can reduce the complexity of the state space BDDs, although the state space itself could become more complex because of the additional state elements. The state elements’ outputs do not feed, through the combinational logic, other state elements. They thus contribute just additional primary outputs. Any proof performed on this modified circuit is still valid on the original one, provided the outputs of the auxiliary state elements are not observed.

D. Auxiliary Variables and Ordering

Variable ordering plays a key role in limiting BDD size, but, for many circuits, good orderings do not exist. In fact, the requirements of different sub-blocks might conflict [8], [10] or state transition functions and state sets representation might impose contradictory orderings [11]. In other cases, good orderings are expensive to find. Auxiliary variables can help because they can improve performances when the ordering is poor, yielding results that are comparable with the ones obtained with a good but expensive ordering and because they can overcome contradictory requirements on ordering, because their effect is to “duplicate” the variables, letting them locally appear with the best ordering.

Auxiliary variables are most useful when they represent relevant points of the circuit or when they decompose complex functions in simpler ones. In general, their characteristic is to “discriminate” very much the functions, thus they should be placed on top of the other variables.

Experiments were performed on a 70 MIPS DEC-ALPHA with 64 Mbytes of memory. Results are for some scalable home-made circuits:

- $ca_n$ is a portion of an ALU loading an accumulator with the sum of two $n$ bit registers
- $cb_{n,m}$ is derived from the previous circuit with the addition of a $m$-bits shift-register loaded serially with carry-out bit of the adder
- $mm^+_n$ is a modified version of the MCNC min_max (mm) benchmark that has $4n$ bit registers instead of $3$ (min, max and last); one register (avg) stores the average $\frac{(\text{min} + \text{max})}{2}$.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Best Algorithmic Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ca_6$</td>
<td>Traditional</td>
</tr>
<tr>
<td>$ca_6$</td>
<td>With Aux. Var.</td>
</tr>
<tr>
<td>$cb_{6,12}$</td>
<td>Traditional</td>
</tr>
<tr>
<td>$cb_{6,12}$</td>
<td>With Aux. Var.</td>
</tr>
<tr>
<td>$mm^+_6$</td>
<td>Traditional</td>
</tr>
<tr>
<td>$mm^+_6$</td>
<td>With Aux. Var.</td>
</tr>
</tbody>
</table>

Table 1 – Symbolic traversals: relation with algorithmic variable ordering

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Manual Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ca_6$</td>
<td>Traditional</td>
</tr>
<tr>
<td>$ca_6$</td>
<td>With Aux. Var.</td>
</tr>
<tr>
<td>$cb_{6,12}$</td>
<td>Traditional</td>
</tr>
<tr>
<td>$cb_{6,12}$</td>
<td>With Aux. Var.</td>
</tr>
<tr>
<td>$mm^+_6$</td>
<td>Traditional</td>
</tr>
<tr>
<td>$mm^+_6$</td>
<td>With Aux. Var.</td>
</tr>
</tbody>
</table>

Table 2 – Symbolic traversals: relation with manual variable ordering

The experiments concerned reachable state space generation with an exact forward traversal and aimed at comparing the traditional approach and the one based on auxiliary variables, considering the effect of variable ordering. In both Tab. 1 and 2, column labeled $# \ N$ lists the global number of BDD nodes used during all the phases of symbolic traversal. Among the algorithmic ordering known from the literature, we selected the best one and applied it with and without auxiliary variables (Tab. 1). As the results were not satisfactory, we repeated the comparison using a manual ordering (Tab. 2). In both cases, the gain due to auxiliary variables is considerable. Moreover, they make traversals more ordering-independent.

III. Placing Auxiliary Variables

Among the criteria to select the nodes for auxiliary variable insertion, the main ones are functional, i.e., based on the knowledge of what a certain node does, or structural, being the function of the node unknown. The former exploits the knowledge a designer usually has about the circuit, the latter is more suited to the benchmarks whose function is unknown. Structural insertion of $X$ auxiliary variables is described in [6]. This paper focuses on functional insertion of $Y$ variables, possibly postponing existential quantification.

Extending the scope of symbolic traversals beyond control-dominated circuits is an open field for research, as currently data paths can’t be dealt with such techniques. We consider systems consisting of a control unit and a data path and regular structures.
A. Systems with Data Path and Control Unit

Complex systems are often modeled as a control unit and a data path, that exchange primary inputs and primary outputs with the external environment and control and status signals between themselves.

Auxiliary variables are inserted at those nodes that allow abstracting from data. Natural candidates are the control and status signals, e.g., the terminal count of a counter, the carry-out of an adder, the outputs of a comparator, the load or reset lines of a register, etc. The ability to keep the functions of the data path and of the control unit apart, modeling nevertheless their interaction, allows us to limit the BDD explosion.

B. Regular Structures

Iterative circuits play a key role in data path design, because data have an inherently regular structure, whereas control is random in nature. Regular structures consist of iterated basic cells, each modeled as a FSM. Signals entering the i-th cell belong to the following classes:

- global signals \( g \cdot X \)
- local signals \( X^i \)
- signals \( X^i_{-d_i} \) that are functions of the outputs of the first \( d_i \) cells to the right of cell \( i \)
- signals \( X^i_{-d_2} \) that are functions of the outputs of the first \( d_2 \) cells to the left of cell \( i \).

Focusing only on linear interconnections, if the signals depend both on cells to the right and to the left, the resulting structure is bi-directional. Otherwise, it is uni-directional.

The effect of auxiliary variables \( \hat{X} \) and \( \hat{Y} \) on regular structures is to break long chains, simplifying the Boolean functions and decoupling portions of the circuit.

1) Uni-directional regular structures: A linear uni-directional regular structure with dependencies on local signals and on the previous cell is shown in Fig. 1. Examples encountered in data path designs are counters, registers and uni-directional shift registers, LFSRs, etc.

The transition relation of the i-th cell is:

\[
T_{bd}(Y^i, S^i) = \exists X^i_{-1}, \exists X^i, \exists X^i \left( \prod_{j=1}^{k} \left( y^i_j \equiv \delta_j^i(X^i_{-1}, g \cdot X^i, X^i, S^i) \right) \right)
\]

For sake of simplicity, let us limit the chain to two cells, each having one state transition function \( \delta \) (\( k = 1 \)), one input from the cell to the right \( (X^i_{-1} = X^i_{-1}) \), and one local input \( (X^i = x^i) \). In this case \( X^i_{-1} = (X^i_{-1}, x^i_{-1}) \), \( X = (x^i, x^2) \), \( Y = (y^1, y^2) \), and \( S = (s^1, s^2) \). Line \( x^i_{-1} \) can be seen as an output of cell \#1 and is a natural candidate for placing an auxiliary variable: \( \bar{z} \equiv x^i_{-1} = \lambda_1(x^i_{-1}, x^1, s^1) \).

Indeed, expressing \( \delta^2 \) as a function of just the primary inputs would require a heavy composition operation with the \( \lambda \)’s:

\[
T(Y, S) = \exists x^i_{-1}, x^i \cdot \left( \left( \left( \left( \left( \left( y^1 \equiv \delta^1(x^i_{-1}, x^1, s^1) \right) \cdot \left( y^2 \equiv \delta^2(x^i_{-1}, x^1, s^1, x^2, s^1) \right) \right) \right) \right) \right) \right)
\]

Auxiliary variables can provide relevant benefits: introducing only \( \bar{z} \) and applying formula 1, the transition relation becomes:

\[
T(Y, S) = \exists x^i_{-1}, x^i \cdot \left( \left( \left( \left( \left( \left( y^1 \equiv \delta^1(x^i_{-1}, x^1, s^1) \right) \cdot \left( y^2 \equiv \delta^2(x^i_{-1}, x^1, s^1, x^2, s^1) \right) \right) \right) \right) \right) \right)
\]

As a consequence of the introduction of the auxiliary variable the two cells have been decoupled and, because of the disjoint support sets, existential quantifiers on the local signals \( X^i \) have been pulled inside the product sign.

The same holds for \( \bar{Y} \) whose existential quantification is possibly delayed by one clock tick. In general, it is not necessary to introduce an auxiliary variable at each line connecting two cells: depending on the size of the BDDs, cell can be grouped in sub-chains, decoupled by auxiliary variables.

2) Bi-directional regular structures: A linear bi-directional regular structure with dependencies limited to the previous and next cells is shown in Fig. 2. Examples encountered in data path designs are bi-directional shifters, systolic arrays, etc.

Auxiliary variables are conveniently inserted both on \( X^i_{-d_1} \) and \( X^i_{-d_2} \) lines, resulting in the same decoupling and simplification effects as for uni-directional structures.

For sake of readability, we omit a detailed analysis of this case.

C. Experimental Results

Experimental results are presented in Tab. 3 and 4. Manual ordering is used. They concern the same circuits used in the previous section, with the addition of:
\(cc_{n,m}\) \(m\)-bit shift register (with shift left, right and hold) controlled by the output of an \(n\) bit comparator; \(MCNC\ \text{min.max}\) 9 and \(sbc\) [12].

Tab. 3 reports some reachability analysis statistics. \# \(D\) denotes the number of traversal steps, \# \(R\) represents the cardinality of the reachable state set. Tab. 4 refers to exact forward symbolic state space traversal. \# \(\tilde{x}\) represents the number of auxiliary variables we inserted. Auxiliary variable selection is manual except for \(sbc\). For this circuit the heuristic described in [6] is used for automatic generation of auxiliary variables. Some of the data of Tab. 4 are identical to Tab. 2.

Our home-made circuits are presented with growing size, and the results shows that the gain (an order of magnitude in some cases), of the auxiliary variables approach over the traditional one, both in terms of BDD nodes used and CPU time, increases with the increase in circuit parallelism. Automatic insertion of auxiliary variables does not improve performance in the \(sbc\), like in our benchmarks set, because of the random nature of this circuit.

### IV. Conclusions

The main theoretical contribution of this paper is the application of auxiliary variables to symbolic traversals. Auxiliary variables are used to decompose Boolean functions in simpler ones and to manipulate them in decomposed form. They also serve the purpose of overcoming the limits of expensive or contradictory variable orderings.

From a practical point of view, symbolic techniques are now applicable to data paths made of regular structures, because auxiliary variables help in circuit decoupling and in abstracting from data, which are the main causes for state explosion.

Symbolic traversal techniques are a suitable answer to the need for verifying synthesized synchronous circuits and may be of considerable help also in ATPG. Future work will consider more complex designs consisting of data processing and control parts and the use of auxiliary variables in backward traversal.

### REFERENCES