Biasing Symbolic Search by means of Dynamic Activity Profiles

Gianpiero Cabodi Paolo Camurati Stefano Quer

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

Abstract

We address BDD based reachability analysis, which is the core technique of symbolic sequential verification and Model Checking.

Within this framework, non purely breadth-first and guided traversals have shown their value to improve efficiency by reducing memory consumption for BDD representation.

We propose a guided search strategy exploiting performance statistics. These activity figures are gathered through a continuous and dynamic learning process on a variable-by-variable basis. This technique is completely integrated with the reachability analysis routine, as it is fully compatible with dynamic reordering and allows multiple partial traversal phases. We thus move away from the static and manual schemes, which are one of the main limitations of previous approaches.

Experiments are given to demonstrate the efficiency and robustness of the approach.

1 Introduction

State-of-the-art approaches for reachability analysis and formal verification of circuits modeled as Finite State Machines (FSMs) exploit symbolic techniques based on Binary Decision Diagrams (BDDs).

Given a FSM, the core computation of all symbolic reachability analysis methods is a breadth-first search of the state space. For medium-small circuits symbolic breadth-first search is usually quite efficient, but it reaches its limits on large examples because of the BDD explosion problem.

Recently, many researchers have followed the trend of guided search, interleaving breadth and depth–first analysis of the state transition graph. Since traversals often produce the largest BDDs during intermediate steps, the key idea is that a sequence of simpler traversals is a good way to obtain space and time improvements. Several techniques have been proposed to tailor partitioned/partial traversals. Among them, let us remember: BDD sub-setting and decomposition based on high density [1], over–approximate forward traversals to prune exact backward verification [2], partitioning based on Shannon decomposition [3, 4], manual insertion of *guards* in the hardware description [5].

In [6, 7] we propose a technique to estimate the activity of recursive BDD operators in terms of memory and time. We apply it to symbolic traversals of FSMs. Statistics are collected for each BDD node of the transition relation T. They are intended to record the involvement of that node in memory and/or time consuming image computations. The activity profile is used to bias a two-phase guided search T sub-setting. The pruned transition relation is used for a first *partial* traversal, exploring a subset of the state transition graph. A further *full* traversal completes the process by using the original (non pruned) T. As in all guided search schemes, performance gains come from the non purely breadth–first strategy, and the main novelty of this method lies in taking into account performance measures to drive the overall process. The achieved performance is very good, but it is obtained with a static learning method, and the two-phase traversal, i.e., still a very trivial scheme.

In [8] the authors propose another guided search technique, based on “distance driven” partial traversals. The basic intuition is that states with similar codes, i.e., having a small Hamming distance, are supposed to combine in compact BDDs. T and state sets are thus partitioned in order to reach only state “with a small distance” from the set of already reached states. Pruning is performed dynamically, dynamic variable reordering is allowed during traversal, and the decomposition method is supposed to maintain the efficiency of the Computed Table.

The approach presented in this paper follows the *activity profiles* idea of [6, 7]. To guide traversal we still exploit information directly coming from the process itself (by means of learning), but we use a new metric to characterize the activity of recursive BDD operators with the aim of producing a fully automated and dynamic technique.

Instead of relating activity measures to BDD nodes, we collect them by variables and variable cofactors. We thus have a less detailed and coarser characterization of the observed process, but we can better interact with dynamic reordering and multiple variable orders, which do not support BDD node related statistics (see [6, 7]). Thanks to those features, activity profiles are collected through a continuous and dynamic learning process, i.e., during each image computation. We thus move away from the static scheme, one of the main limitations of [6, 7].

Given a set of activity measures collected by variables, we use them to select proper variable cofactors in order to restrict and guide the traversal. This is activated taking into account the cost of image computations. Whenever the memory and/or time cost of an image computation is too high, we perform a sub-setting of the transition relation, which is equivalent to choosing a working subspace, in order to reduce the expected cost of coming images. The pruned transition relation is kept until images attain a low cost, and traversal resumes with the original T. The process can be nested and/or repeated several times, globally producing a mixed breadth–depth first traversal guided by performance related measures.

Notice that the learning process has a negligible cost with respect to reachability analysis, and the memory overhead is limited to a few counters for each variable of the transition relation.

As a final remark, notice that the method is valuable as far as the activity recorded in the past is a good prediction for the future. We do not have any theoretical support for this assumption, but our experimental results confirm its validity, as they show the robustness and effectiveness of the approach.

2 Preliminaries

A completely specified FSM M is a 6–tuple $M = (I, O, S, \delta, \lambda, \mathsf{S_0})$ where I is the input alphabet, O is the output alphabet, S is the state space, $\delta : S \times I \rightarrow S$ is the next state function, $\lambda : S \times I \rightarrow O$ is the output function and $\mathsf{S_0} \subseteq S$ is the initial state set. In the sequel, we will use $s = (s_1, s_2, \ldots, s_n)$, $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_n)$ to indicate respectively the vector of present state, primary input and next state variables.

The *transition relation* T associated to a FSM M is:

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

The image To of a set of states described by its characteristic function From according to the transition relation is defined as:

$$\begin{array}{rcl} \mathsf{To}(y) & = & \textsc{Img}(\mathsf{T}(s, x, y), \mathsf{From}(s)) \\ & = & \exists_{s,x} \; (\mathsf{T}(s, x, y) \cdot \mathsf{From}(s)) \end{array}$$

As the monolithic representation for T can be difficult or even impossible to obtain, T is usually represented in a partitioned (conjunctive or disjunctive) form. In the "conjunctive" approach T is expressed as a conjunction of terms. The process proceeds by *sorting*, i.e., reordering terms, by *clustering*, i.e., partially computing products P_i, and by *early quantification*, i.e., moving the existential quantification inside the conjunction:

$$\begin{array}{rcl} \mathsf{To}(y) & = & \exists_{s,x} (\prod_{i=1}^{n} t_i(s, x, y_i) \cdot \mathsf{From}(s)) \\ & = & \exists_{s,x} (\prod_{i=1}^{m} \mathtt{P_i}(s, x, y) \cdot \mathsf{From}(s)) \\ & = & \exists_{s,x} (\mathtt{P_m} \cdot \mathtt{P_{m-1}} \cdot \ldots \cdot \mathtt{P_1} \cdot \mathsf{From}) \\ & = & \exists_{s^m,x^m} (\mathtt{P_m} \cdot \ldots \cdot \exists_{s^1,x^1} (\mathtt{P_1} \cdot \mathsf{From}) \ldots)) \end{array} \quad (1)$$

where the innermost products may be computed, with early quantification of variables not in the support of the outer terms. The process results in a linear chain of relational products (unified conjunction and existential quantification), usually producing peak BDD sizes within intermediate steps [1].

The set of forward reachable state set R can be computed with the well-known pseudo–code represented in Figure 1.

```
TRAVERSAL (T, S_0)
    New ← S_0
    From ← S_0
    R ← S_0
    while (New ≠ ∅)
        To ← IMG (T, From)
        New ← To · $\overline{R}$
        R ← R + New
        From ← BESTBDD (New, R)
    return (R)
```

Figure 1: Symbolic Forward Traversal.

3 Dynamic Activity Profile: A New Approach

Following [6, 7] we work on relational products, which are the core operations of transition relation based image computations.

Let us then consider a relational product $\exists_{\mathtt{v}}(\mathtt{t} \cdot \mathtt{f})$. In case of a monolithic transition relation, `t` is the transition relation itself T, and `f` is the set From. In case of a partitioned/clustered transition relation, `t` is the generic partition/cluster $\mathtt{t_i}$, and `f` is an intermediate product (previously obtained). `v` is the set of variables to be existentially quantified.

The activity profile is meant to record time and memory related measures through recursions of the relational product operation. We do this on a variable-by-variable basis (instead of a node–by–node basis as in [6, 7]). For each variable `var` of `t`, we define five counters recording the activity of relational product recursions on BDD nodes of variable `var`:

- `var.cache`: Cache hits on `var`, i.e., the number of times `var` appears as an operand of a recursive call whose result is found in the operation cache table.
- `var.t.rec`: Number of recursions on the `then` child of `var`, i.e., the number of times the then edge is traversed.
- `var.e.rec`: Number of recursions on the `else` child of `var`.
- `var.t.size`: Node increment (size cost) within the BDD manager due to recursions on the `then` child of `var`, i.e., the difference between the total number of nodes after and before any recursion on the variable `var`.
- `var.e.size`: Node increment (size cost) within the BDD manager due to recursions on the `else` child of `var`.

The recursion indicators count the number of recursions on the Then (Else) cofactor of each variable. They are related to the time spent working on each specific variable, i.e., the higher this number, the more the related `var` variable is active. Differentiating the cofactors enables showing which one is more active. The cache hits counter, `cache`, is an indicator of activity stopping at level `var` due to a cache hit. The size indicators are related to the amount of BDD nodes generated by recursions on `var`.

In the specific case of conjunctively partitioned transition relations, we may decide to collect a different activity profile for each cluster, or we may prefer a unique profile for the whole image computation. With the first option, one tries to focus on single relational products involving transition relation clusters, to concentrate on the one(s) involved in expensive operations. The second case is more general and it might be applied to other image computation strategies. In the sequel we will follow the former case, with different activity profile tables for each cluster. The extension to the other case is trivial.

Example 1 *Figure 2(a) shows a forest of BDDs for a generic transition relation* T *made up of* m *clusters* P_i*. Figure 2(b) shows the information on activity associated with one single cluster; for the considered cluster it reports one entry for each BDD variable. Figure 2(c) shows the specific kind of information associated with the variable: cache hits, recursion for each cofactor, and size cost for each cofactor.*

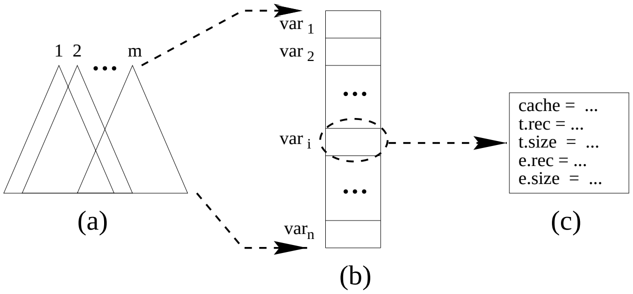

Figure 2: BDD with Learning Activity for Each Variable.

```
IMG$_{Profile}$(T, From, stats)
    Init stats array of $m$ components
    $P_0 \leftarrow$ From
    for $i \leftarrow 1$ to $m$
        $v \leftarrow$ Early Quantifiable Variables of T for $t_i$
        $t_i \leftarrow$ i-th Cluster of T
        $P_i \leftarrow$ RP$_{Profile}$($v$, $t_i$, $P_{i-1}$, stats$_i$)
    return ($P_m$)
```

Figure 3: Image Computation with Learning.

Following Equation 1, Figure 3 shows the pseudo-code for an image computation routine which includes the collection of the learning activity. The stats array is used to collect activity measures. It has a number of components equal to the number of clusters of T, and for each component a slot for each variable in the support of the cluster. In our application, this overhead is insignificant compared to the number of BDD nodes usually involved in the computation. Nevertheless, it should be traded-off with the performance improvements obtained.

The intermediate product P_0 is initially set to the set From. Then, the algorithm proceeds following Equation 1, where function RP$_{Profile}$, described in Figure 4, is called to compute the next intermediate product P_i. This function receives as parameters the quantifiable set of variables v, the current cluster t_i, the current intermediate product P_{i-1}, and the components of the statistic array associated to the current cluster stats$_i$.

Figure 4 shows our modified relational product operator handling the previously defined counters. RP$_{Profile}$ describes the recursive procedure for the relational product (the RP function) $\exists_v(t \cdot f)$, with activity learning enabled.

Our relational product is quite similar to the original one. The main difference appears whenever the array stats is taken into consideration to update the counters cache, recursions, and sizeCosts for the current variable var.

Given the statistics just introduced, in the following section we show how to bias a guided traversal, through transition relation sub-setting.

4 Transition Relation Sub-setting

Given an activity profile collected throughout previous relational product calls, we use it to generate a subset of T. The goal is generating a BDD with expected lower activity in future computations.

Being memory explosion the main hurdle for BDD based reachability analysis, we primarily operate on size costs. Similar considerations could be applied in the case of time related costs, by taking into account recursion counters.

```
RP$_{Profile}$(v, t, f, stats)
    if (terminal case)
        return (result of terminal case evaluation)
    let var be the top variable of t and f
    if (result of RP$_{Profile}$(v, t, f) is cached)
        stats[var].cache += 1
        return (result)
    $r_0 \leftarrow$ RP$_{Profile}$(v, $t_{\overline{var}}$, $f_{\overline{var}}$)
    stats[var].e.rec += 1
    stats[var].e.size += (total node increment)
    if (var $\in$ v)
        if ($r_0$ = 1)
            $r \leftarrow 1$
            $r_1 \leftarrow 0$
        else
            $r_1 \leftarrow$ RP$_{Profile}$(v, $t_{var}$, $f_{var}$)
            stats[var].t.rec += 1
            stats[var].t.size += (total node increment)
            $r \leftarrow r_0 + r_1$
    else
        $r_1 \leftarrow$ RP$_{Profile}$(v, $t_{var}$, $f_{var}$)
        stats[var].t.rec += 1
        stats[var].t.size += (total node increment)
        r $\leftarrow$ reduced, unique BDD node for (var, $r_0$, $r_1$)
    cache the result of this operation
    return (r)
```

Figure 4: Relational Product with Activity Counters Handling.

We prune nodes with an excessively high size activity, in order to remove those nodes that are (expected to be) too expensive in terms of memory. Pruning is controlled by a size threshold th_{size}. All variables statistics are first normalized. Then all variables with costs larger than th_{size} are selected, and one of their cofactors is pruned. More specifically, a variable var such that

$$\mathtt{var.t.size} + \mathtt{var.e.size} > th_{size}$$

is selected. We then detect the heavier and the lighter cofactor, by comparing var.t.size and var.e.size, and, following [6], we keep either the heavier or the lighter cofactor, based on a user selectable setting. For example, in the former case, let $\mathtt{var}_{heavy}$ be the variable cofactor with heavier size cost. The pruned transition relation is obtained by conjoining the original T with the selected variable cofactor:

$$T_{pruned} = \mathtt{var}_{heavy} \cdot T$$

To avoid excessive or poorly effective prunings, we filter out from this process variables with high activity unbalancing between the two cofactors.

The process results in a subset T_{pruned} of the original T, by using whom we expect a lower activity for coming computations.

5 Guided Traversal biased by Activity Profiles

Going to our overall traversal procedure, it follows a guided search scheme, controlled by a heuristic selection of the transition relation to be used. Depending on the memory cost of the last image operation, we may decide to:

- Prune (or further prune) T because the last image had a high cost
- Keep T (pruned or non pruned) as it is because the cost of the last image was acceptable
- Add previously pruned cofactors to T because of low cost image.

Figures 5 reports our modified traversal procedure with dynamic learning and partitioning. It is directly derived from the original pseudo-code of Figure 1, including the modification previously indicated. While T_{Active} is the subset of T currently in use, Removed is the pruned part. The parameter th_{size} defines the threshold beyond which nodes are pruned from T_{Active} during each partitioning operation (see Section 4).

Partitions can be kept in a stack (functions PUSH and POP), or in a priority queue. Statistics can be initialize before each image computation or just after any T pruning. This second possibility is shown in the pseudo-code. The cost of each image computation is evaluated by function IMGCOST (described in Figure 6). Depending on this evaluation the "case" statement performs the three possible basic operations: T_{Active} is (further) pruned, it is left unchanged, or one of its previously removed partition is added to the current relation.

```
INCREMENTALTRAVERSAL (T, $S_0$, $th_{size}$)
    New $\leftarrow$ $S_0$
    From $\leftarrow$ $S_0$
    R $\leftarrow$ $S_0$
    $T_{Active}$ $\leftarrow$ T
    Removed $\leftarrow$ $\emptyset$
    INITACTIVITYPROFILES (stats)
    continue $\leftarrow$ TRUE
    while (continue)
        To $\leftarrow$ IMG$_{Profile}$($T_{Active}$, From, stats)
        New $\leftarrow$ To $\cdot$ $\overline{R}$
        R $\leftarrow$ R + New
        From $\leftarrow$ BESTBDD (New, R)
        if (New = $\emptyset$ AND Removed = $\emptyset$)
            continue $\leftarrow$ FALSE
        else
            case IMGCOST()
                IMG_COMPLEX:
                    /* Prune $T_{Active}$ */
                    $T_{New}$ $\leftarrow$ PRUNE ($T_{Active}, th_{size}$)
                    $T_{Removed}$ $\leftarrow$ $T_{Active} \cdot \overline{T_{New}}$
                    PUSH (Removed, $T_{Removed}$ )
                    $T_{Active}$ $\leftarrow$ $T_{New}$
                    RESETACTIVITYPROFILES (stats)
                IMG_SIMPLE:
                    /* Add removed component to $T_{Active}$ */
                    $T_{Active}$ $\leftarrow$ $T_{Active}$ $+$ POP (Removed)
                    From $\leftarrow$ R
                IMG_OK:
                    /* Keep $T_{Active}$ as it is */
    return (R)
```

Figure 5: Incremental Partial Symbolic Forward Traversal with Dynamic Learning.

Figures 6 reports function IMGCOST. It dynamically evaluates the cost of the process.

```
IMGCOST ( )
    a $\leftarrow$ Memory In Use Before Image Computation
    b $\leftarrow$ Memory In Use After Image Computation
    c $\leftarrow$ CPU Time Before Image Computation
    d $\leftarrow$ CPU Time After Image Computation
    if ((b / a) $> (1 + w_1)$ OR (d / c) $> (1 + w_2)$)
        return (IMG_COMPLEX)
    if ((b / a) $< (1 + w_3)$ AND (d / c) $< (1 + w_4)$)
        return (IMG_SIMPLE)
    return (IMG_OK)
```

Figure 6: Cost Function.

It basically evaluates the image cost, in terms of memory occupation and CPU time. w_1, w_2, w_3, w_4 are generic weights, whose values are tuned experimentally to trade-off between the cost of each single image and the number of iterations.

6 Experimental Results

The presented technique is implemented within a traversal program built on top of the Colorado University Decision Diagram (CUDD) package [9]. Our experiments ran on a 550 MHz Pentium II Workstation with a 128 Mbyte main memory, running RedHat Linux 6.2. We present data on a few ISCAS'89 and ISCAS'89–addendum benchmarks, selected with different sizes, within the range of circuits manageable by state–of–the–art reachability techniques. We present three versions of circuit s5378, in which the subscript indicates the number of memory elements.

All initial variable orders are statically evaluated from the netlist description. For all the experiments we determine proper cluster size thresholds (in the range from 500 to 5000 BDD nodes). We enable dynamic reordering (group sifting with groups made up of corresponding present/next state variables) with the default settings of the CUDD package.

Table 1 compares standard breadth–first implementation [10], the "profile" based technique [6], the "distance driven" based technique [8], and the proposed method.

In [10] authors analyze standard breadth-first traversal focusing on a new clustering algorithm, where the clusters and their order are evaluated based on the dependence matrix of the transition relation. Results are obtained using a 400 MHz Pentium II Workstation running Linux. The memory limit in this case is 750 Mbyte. We report from this work the best data (among a series of values collected with different image approaches) with variable ordering enabled.

In [6] authors use a 266 MHz Pentium II Workstation with a 256 Mbyte main memory, with a 128 Mbyte memory limit.

In [8] authors use a Ultra-II model 2170 with 1 Gbyte main memory. Notice that in this case the initial variable orders are already optimized, whereas for all other methods the initial variable orders are statically computed.

Whereas Time reports the CPU time in seconds, and Mem. the maximum memory used in Mbyte, Peak is the peak BDD size for intermediate products within image computations in terms of BDD nodes. Peak Live Nodes is the peak of node alive in the BDD package, and Peak Size is the peak BDD size for intermediate products within image computations in term of Mbytes. Overall execution times are often dominated by sifting (overall

Circuit	[10]		[6]			[8]		This Paper		
	Peak Live Nodes	Time	Peak	Mem.	Time	Peak Size	Time	Peak	Mem.	Time
	[k]	[sec]		[Mb]	[sec]	[Mb]	[sec]		[Mb]	[sec]
s1269	1413.9	1658	69615	11.5	68 (39)	0.8	52	60039	14.3	37 (21)
s1512	136.5	599	12218	9.6	215 (110)	–	–	24059	9.5	167 (80)
s3271	1094.4	2604	610185	56.1	4983 (3521)	3.5	329	565453	43.9	3049 (1570)
s3330	8778.0	5035	30904	15.5	960 (720)	1.9	320	26838	15.3	189 (86)
s4863	1058.7	1587	52335	181.5	528 (430)	0.9	49	48585	167.9	223 (101)
$s5378_{121}$	572.2	2201	82295	13.4	248 (134)	–	–	78575	13.1	169 (91)
$s5378_{164}$	–	–	–	–	–	–	–	828299	49.9	3639 (2011)
$s5378_{179}$	–	–	–	–	–	–	–	687853	47.9	9938 (4653)

Table 1: Reachability Analysis Comparison. – means data not available.

sifting time is reported between brackets when available).

It is worthwhile noticing that also if the CPUs used are different, our automatic guided search usually outperforms the standard breadth-first method, and also often improves the other techniques we compare with. In any case the main advantage to respect to [6] is that there are fewer supplied parameters, as the iterated application of the learning and pruning phases are somehow self-tuning. A few manual decision still remain (mainly the parameter th_{size}, Figure 5) but the tuning process has been drastically reduce.

7 Conclusions and Future Work

We present a new approach to characterize the involvement and the impact of transition relations within the core operations of reachability analysis. This approach is based on activity measures collected on a variable-by-variable basis for each cluster of the transition relation. They are used to dynamically partitioning the transition relation based on cofactor selection. The resulting methodology is an automatic guided traversal, where the learning activity is completely and automatically integrated with the core algorithm.

Future work will investigate some more sophisticated techniques to collect and use activity measures. In particular, we want to refine the overall approach, i.e., the technique used to gather performance statistics, the pruning methodology, and the method used to evaluate the cost of the traversal process. Moreover, comparison between cluster-based and transition relation-based statistics (see Section 3) has to be better evaluated.

References

[1] K. Ravi and F. Somenzi. High–Density Reachability Analysis. In *Proc. IEEE/ACM ICCAD'95*, pages 154–158, San Jose, California, November 1995.

[2] G. Cabodi, P. Camurati, and S. Quer. Efficient State Space Pruning in Symbolic Backward Traversal. In *Proc. IEEE ICCD'94*, pages 230–235, Cambridge, Massachussetts, October 1994.

[3] G. Cabodi, P. Camurati, and S. Quer. Improved Reachability Analysis of Large Finite State Machine. In *Proc. IEEE/ACM ICCAD'96*, pages 354–360, San Jose, California, November 1996.

[4] A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. Sangiovanni-Vincentelli. Reachability Analysis Using Partitioned–ROBDDs. In *Proc. IEEE/ACM ICCAD'97*, pages 388–393, San Jose, California, November 1997.

[5] M. Ganai and A. Aziz. Efficient Coverage Directed State Space Search. In *IWLS'98: IEEE International Workshop on Logic Synthesis*, Lake Tahoe, California, June 1998.

[6] G. Cabodi, P. Camurati, and S. Quer. Improving Symbolic Traversal by means of Activity Profiles. In *Proc. ACM/IEEE DAC'99*, pages 306–311, June 1999.

[7] G. Cabodi, P. Camurati, and S. Quer. Improving Symbolic Reachability Analisys by means of Activity Profiles. *IEEE Transactions on CAD*, 19(9):1065–1075, September 2000.

[8] A. Hett, C. Scholl, and B. Becker. Distance Driven Finite State Machine Traversal. In *Proc. ACM/IEEE DAC'00*, pages 39–42, June 2000.

[9] F. Somenzi. CUDD: CU Decision Diagram Package – Release 2.3.0. Technical report, Dept. of Electrical and Computer Engineering, University of Colorado, Boulder, Colorado, October 1998.

[10] I. Moon and F. Somenzi. Border-Block Triangular Form and Conjunction Schedule in Image Conjunction. In *Proc. FMCAD'00, To Appear*, 2000.