The Compositional Far Side of Image Computation [*]

Chao Wang Gary D. Hachtel Fabio Somenzi

Department of Electrical and Computer Engineering
University of Colorado at Boulder, CO 80309-0425

{wangc,hachtel,fabio}@Colorado.EDU

Abstract

Symbolic image computation is the most fundamental computation in BDD-based sequential system optimization and formal verification. In this paper, we explore the use of over-approximation and BDD minimization with don't cares during image computation. Our new method, based on the partitioned representation of the transition relation, consists of three phases: First, the model is treated as a set of loosely coupled components, and over-approximate images are computed to minimize the transition relation of each component. A refined overall image is then computed using the simplified transition relation. Finally, the exact image is obtained by a *clipping* operation that recovers all previous over-approximations. Since BDD minimization employs constraints on the next-state variables of the transition relation, instead of the customary constraints on the present-state variables, we call the resulting method *far side* image computation.

The new method can be implemented on top of any image computation algorithm that is based on the partitioned transition relation. (For example, IWLS95 [21], MLP [17], and Fine-Grain [16].) We demonstrate the effectiveness of our approach by experiments on models ranging from easy to hard: The new method wins significantly over the best known algorithms so far in both CPU time and memory usage, especially on the hard models.

Categories and Subject Descriptors
B.6.3 [**Logic design**]: Design aids—*Verification*
General Terms: Verification, Algorithms
Keywords: Model checking, image computation, Binary Decision Diagrams, Symbolic

1. Introduction

Image computation, as the most fundamental computation during symbolic state space exploration (also called reachability analysis), is crucial in sequential system optimization and formal verification based on the use of Binary Decision Diagrams (BDDs [2]). The *Preimage* operation is equally important in verification problems such as CTL model checking, but even in those algorithms, reachability analysis is usually applied to compute don't care sets to constrain the search and minimize the BDDs.

Image computation consists of finding all the successors of a given set of states D according to a set of transitions T. There are two basic classes of image computation methods, among which one is based on the transition function [8], and the other on the transition relation [11, 21, 17, 5, 4, 16]. There are also hybrid algorithms that combine these two techniques together [20]. Our new method is based on the transition relation.

1.1 Related Work

Except for small systems, the transition relation (TR) can not be represented by a monolithic BDD. Instead, it is usually represented by a collection of BDDs (called clusters) whose conjunction is the entire transition relation. This representation is called the *partitioned transition relation*. Image computation is based on conjoining the given set of states with all the transition relation clusters while existentially quantifying the present-state variables and inputs. The complexity of this computation depends heavily on the size of the BDDs that represent the given set of states, the transition relation, and the intermediate products during the *conjoin-quantify* operations.

In the *conjoin-quantify* operations, the way in which transition bit-relations (i.e., the transition relations for individual binary memory elements) are grouped into clusters has a large impact on the overall performance; equally important is the order in which clusters are conjoined and quantified. The problem of clustering and ordering so as to minimize peak BDD size of the intermediate products is called the *quantification scheduling* problem.

The various existing image computation methods differ mainly in their *quantification schedules*. Early studies on the effect of early quantification can be traced back to [24, 3, 11], and in [13], the quantification scheduling problem was proved to be NP-complete. A successful heuristics, known as IWLS95, was proposed in [21]; it first orders the transition bit-relations based on a heuristic score, and then linearly clusters these bit-relations until the BDD size exceeds certain threshold; finally it orders the clusters based on the same heuristic score. The IWLS95 algorithm is based on a linear quantification schedule; recent progress along this line are the algorithm based on Minimum Lifetime Permutation (known as MLP) [17, 5] and the Fine-Grain method [16]. In [13, 4], however, image computation is viewed as a problem of constructing an optimal parse tree for the image set, which may result in more general quantification schedules. The new method we introduce in this paper is not another heuristic for the quantification scheduling; rather, it provides a higher-level framework and thus can be implemented on top of any of these heuristics.

Don't care information extracted from exact or approximate reachability analysis is commonly used in the preimage computation in symbolic model checking; one can add or remove transitions *from* unreachable states in order to reduce the BDD size without changing the results of fix-point computations restricted to reachable states. However, using approximate reachable don't cares in the image computation is not straightforward: Adding or removing transitions *from* unreachable states (note that the minimization is

[*]This work was supported in part by SRC contract 2002-TJ-920 and NSF grant CCR-99-71195.

.

done on the "near side," i.e., the present-state variables) itself does not change the intermediate products and image result, thus does not reduce the peak BDD size during the computation. (In [18], the exact reachability analysis actually benefits from simplification with approximate reachable states, but the benefit is due to second-order effects like re-clustering and fewer reorderings of the BDDs [22].)

Our new method treats each transition relation cluster as the transition relation of an abstract machine; in this sense it is directly related to previous work on state space decomposition [7, 6, 18, 12, 19]. The main difference is that these previous works compute the entire reachable states for each sub machine, whereas here we are applying the approximation/refinement inside the image computation, not at the fix-point level. However, these two levels of approximation could be combined.

1.2 Our Contribution

We propose a new approach for image computation, which identifies the largest possible don't care set within each image step, and uses it to reduce the peak BDD size during the computation. It can be implemented on top of any image computation method that is based on the partitioned transition relation.

Our method consists of three phases: Approximation, minimization and refinement. First, over-approximations of the image are computed for the abstract machines (each transition cluster induces an abstract model). Then, the BDDs of the transition relation are minimized by using these over-approximations as care sets; A refined image, which is still an over-approximation, is computed with the minimized transition relation. Finally, the exact image is obtained by conjoining all these over-approximations, which clips out any error states that might be introduced by the minimization.

Two problems arise when one tries to modify the "far side" of the transition relation: (1) If a transition is added from a reachable state into a non-reachable state, the results of the image computation are changed. (2) Minimizing the BDD representation of the transition relation on the "far side" with the entire approximate reachable states is not effective because of how the algorithms used for this task [9, 10] work. Both of these problems are solved by our method: For the former, we show how the error states introduced by the spurious transitions can be eliminated from the result of each image computation. For the latter, we use *local* approximations of the reachable states, which are practically effective at simplifying the transition relation representations.

We report experimental results for the implementation of our method on top of both the MLP method [17] and the Fine-Grain method [16], on a series of test cases ranging from easy to hard. We demonstrate the effectiveness of our approach in reducing the BDD size of the transition relation, and show how the smaller-size transition relation translates into less run time and memory usage overall. On the hard problems, our new method has an average speed-up of 35% over the standard MLP method, and finishes 2 hard problems that can not be finished by MLP. Furthermore, on those harder problems where both methods can not finish, our new method can complete more image steps.

2. Preliminaries

In symbolic computations, the objects manipulated are sets of states, rather than individual states in explicit algorithms. A set of states is represented in terms of its Boolean characteristic function, which in turn is represented by a BDD.

Let the sequential system be represented in terms of: (1) A set of *present-state* variables $x = \{x_1,...,x_n\}$; (2) a set of inputs $w = \{w_1,...,w_n\}$; and (3) a set of *next-state* variables $y = \{y_1,...,y_n\}$. The sequential system can be represented symbolically by the set $\langle T(x,w,y), I(x)\rangle$, where $T(x,w,y)$, known as the transition relation, is the Boolean characteristic function that represents the transitions in the state transition graph, and $I(x)$ is the characteristic function that represents the set of initial states.

2.1 Partitioned Transition Relation

The partitioned representation of $T(x,w,y)$ is the conjunction of transition relation clusters

$$T(x,w,y) = \bigwedge_{1\le i\le k} T^i(x,w,y^i) \ ,$$

where $T^i(x,w,y^i)$ is the i-th transition relation cluster, and $y^i \subseteq y$ is a subset of next-state variables.

Each one of the k transition relation clusters induces an (over-approximate) abstract model; the entire model (or sequential system) is a synchronous composition of all these abstract models. Let $\delta_j(x,w,y_j)$ be the next-state function of the j^{th} binary memory element, and $T_j = (y_j \leftrightarrow \delta_j)$ be its *transition bit relation*. The transition relation cluster T^i, which contains k_i memory elements, can be represented as

$$T^i = \bigwedge_{1\le j\le k_i} T_j = \bigwedge_{1\le j\le k_i} (y_j \leftrightarrow \delta_j) \ .$$

Note that each transition bit-relation T_j is included in exactly one transition relation cluster T^i. Because of this, T^i depends on only a relatively small subset of the next-state variables, but may depend on any or all of the present-state variables and inputs.

2.2 Image Computation

Given a transition relation T and a set of states $D(x)$, the image of $D(x)$ is

$$\textsc{Img}(T,D(x)) = \exists x,w.\ \bigwedge_{1\le i\le k} T^i(x,w,y^i) \wedge D(x) \ . \qquad (1)$$

The technique of *early quantification* tries to exploit the fact that each T^i usually depends on a subset of the present-state variables and inputs, and some of these variables can by quantified out before all the clusters are conjoined. Let $Q_1,...,Q_k$ be a partition of the set $\{x,w\}$, the image can be computed by interleaving *conjoin* and *quantify*,

$$\begin{aligned}\textsc{Img}(T,D(x)) = \quad & \exists Q_k\,.\{T^k(x,w,y^k) \wedge \{\\ & \exists Q_{k-1}\,.\{T^{k-1}(x,w,y^{k-1}) \wedge \{\\ & \dots\\ & \exists Q_1\,.\{T^1(x,w,y^1) \wedge D(x)\}\}\}\} \ . \qquad (2)\end{aligned}$$

The order in which the present-state and input variables are quantified is called the *quantification schedule*. Intermediate results, like $\exists Q_1\,.\{T^1(x,w,y^1) \wedge D(x)\}$, are called the *intermediate products*. The BDD size of the intermediate products encountered during a single image computation is often the controlling factor in determining whether a given fixed point operation can be completed on a given computer.

2.3 The Generalized Cofactor

In [9, 10] BDD operators called *restrict* and *constrain* were introduced as techniques for minimizing BDD size without changing the result of the computation which used the BDD. In [24] it was shown that *restrict* and *constrain* were specific instances of a more general operation called the generalized cofactor. Other generalized cofactors were proposed in [23, 14].

A generalized cofactor of T with respect to R, $\widehat{T} = T \Downarrow R$, can be any characteristic function in the interval

$$(T \wedge R) \leq \widehat{T} \leq (T \vee \neg R) \ .$$

An important property of the generalized cofactor is

$$\widehat{T} \wedge R = T \wedge R \ .$$

restrict and *constrain* are particular operators that make the choice so that the BDD of $\widehat{T}$ is minimized in some sense. Because of this, we call the operation indicated by $(T \Downarrow R)$ BDD minimization.

BDD minimization must be done very carefully to be effective. Usually minimization using either *restrict* or *constrain* is poor when R has a large BDD or when R contains many variables that do not appear in T. This is precisely the case when one tries to minimize a sub-relation $T^i(x,w,y^i)$, which contains a small subset of next-state variables y^i, with respect to an over-approximation of the reachable states that contains most of the next-state variables. This is why simplification of the transition relation on the "far side" has not been in common use.

3. The Far Side Image

3.1 The Algorithm

The FARSIDEIMG algorithm is given in Figure 1. The inputs to the procedure are the partitioned transition relation $\{T^i\}$ and the set of states $D(x)$. We use IMG to represent the generic image computation, as described by Equation (2); given different quantification scheduling, the generic image computation can represent the methods in [21, 17, 4, 16].

```
FARSIDEIMG($\{T^i\}, D$) {
1   **for each** $i \in \{1,...,k\}$ **do**
2       $x_T \leftarrow$ present-state variables in the support of $T^i$
3       $x_D \leftarrow$ present-state variables in the support of $D$
4       $Q_A \leftarrow \{x_D\} \setminus \{x_T\}$
5       $Q_B \leftarrow \{x_T\} \setminus \{x_D\}$
6       $Q_C \leftarrow \{x_T\} \cap \{x_D\}$
7       $R_i^+ = \exists Q_C \,.\, (\exists w, Q_B \,.\, T^i) \wedge (\exists Q_A \,.\, D)$
8       $\widehat{T}^i = T^i \Downarrow R_i^+$
9   **od**
10  $\widehat{R} = $ IMG $(\{\widehat{T}^i\}, D)$
11  $R = \widehat{R} \wedge \bigwedge R_i^+$                    // clipping
12  **return** $R$
}
```

Figure 1: The far side image algorithm.

Each T^i is an (over-approximate) abstraction of the overall transition relation $T = \bigwedge_i T^i$. FARSIDEIMG first computes the over-approximate images of D on all these abstractions,

$$R_i^+(y^i) = \exists x, w \,.\, T^i(x,w,y^i) \wedge D(x) \ .$$

Because D does not contain any input variables, w can be quantified out from T^i before the conjunction. A similar argument applies to those present-state variables that appear only in D (represented by Q_A) and those that appear only in T^i (represented by Q_B). These early quantifications make R_i^+ much easier to compute.

In Line 8, the BDD of each T^i is minimized with respect to its R_i^+. It is important to observe that $\bigwedge_i R_i^+$ could be used instead of R_i^+ to minimize T^i. In theory, a smaller care set (i.e., a larger don't care set) provides more degrees of freedom for minimization. However, the subsets of next-state variables in different $R_i^+(y^i)$ are disjoint; any other $R_j^+(y^j)$ such that $j \neq i$ contains only the next-state variables that do not appear in $T^i(x,w,y^i)$. This makes the minimization with respect to $\bigwedge R_i^+$ ineffective. On the the hand, the "local" approximation $R_i^+(y^i)$ contains only those next-state variables in $T^i(x,w,y^i)$; both T^i and R_i^+ typically depend only on a small subset of the next-state variables y^i. Heuristic algorithms like *constrain* and *restrict* perform much better in practice if minimization is with respect to $R_i^+(y^i)$. *Restrict* is more robust because it prevents unwanted BDD variables from appearing in the result, thus it is used in our implementation.

The minimized transition relation $\widehat{T}^i$ is a characteristic function within the interval

$$(T^i \wedge R_i^+) \leq \widehat{T}^i \leq (T^i \vee \neg R_i^+) \ .$$

This can be viewed as adding or removing transitions pointing to $\neg R_i^+$. Figure 2 illustrates this minimization process: The minimization might add transitions from inside or outside D to $\neg R_i^+$ represented by the dotted lines, which are not originally present in T^i. Similarly any transitions to $\neg R_i^+$ originally in T^i might be removed during the minimization, like the solid line marked by a cross.

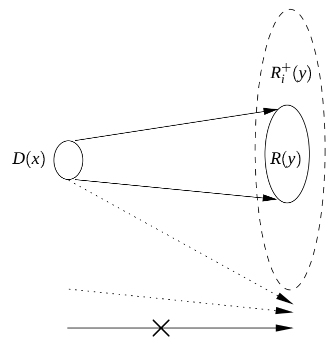


Figure 2: Minimizing the transition relation.

In Line 10, we compute another over-approximation of the overall image, $\widehat{R}$, with the generic image computation on the minimized transition relation. $\widehat{R}$ contains all the states in the exact image (represented by R), and possibly some states in $\neg R_i^+$ because of the added transitions. The clipping operation of Line 11 gets rid of those error states by conjoining $\widehat{R}$ with all the other over-approximations.

3.2 Proof of Correctness

We now prove that FARSIDEIMG computes the correct image, i.e., the same result as that of IMG.

Theorem 1 FARSIDEIMG$(\{T^i\},D) = $ IMG$(\{T^i\},D)$.

PROOF. Let $R(y)$ be the result of IMG$(\{T^i\},D)$, as described by Equations (1,2). The over-approximate images (Lines 2-7 in Figure 1) are based on individual transition relation clusters. Because R_i^+ is an upper bound of R, we have

$$R_i^+(y^i) \wedge R(y) \quad = \quad R(y) \ .$$

Line 8 compute the minimized transition relation cluster $\widehat{T}^i = T^i \Downarrow R_i^+$. According to the definition of the generalized cofactor,

$$\widehat{T}^i \wedge R_i^+ \quad = \quad T^i \wedge R_i^+ \ .$$

By Lines 10-12 in Figure 1

$$\begin{aligned}
&\textsc{FarSideImg}(\{T^i(x,w,y^i)\}, D(x)) \\
&= \widehat{R}(y) \wedge \bigwedge R_i^+(y^i) \\
&= \textsc{Img}(\bigwedge \widehat{T}_i(x,w,y^i), D(x)) \wedge \bigwedge R_i^+(y^i) \\
&= \{\exists x, w\,.\bigwedge \widehat{T}^i(x,w,y^i) \wedge D(x)\} \wedge \bigwedge R_i^+(y^i)
\end{aligned}$$

because $R_i^+(y^i)$ does not depend on x and w

$$= \exists x, w\,.\{\bigwedge \widehat{T}^i(x,w,y^i) \wedge R_i^+(y^i) \wedge D(x)\}$$

by the property of the generalized cofactor

$$\begin{aligned}
&= \exists x, w\,.\{\bigwedge T^i(x,w,y^i) \wedge R_i^+(y^i) \wedge D(x)\} \\
&= \{\exists x, w\,.\bigwedge T^i(x,w,y^i) \wedge D(x)\} \wedge \bigwedge R_i^+(y^i) \\
&= \textsc{Img}(\bigwedge T^i(x,w,y^i), D(x)) \wedge \bigwedge R_i^+(y^i) \\
&= R(y) \wedge \bigwedge R_i^+(y^i) \\
&= R(y) \\
&= \textsc{Img}(\bigwedge T^i(x,w,y^i), D(x))
\end{aligned}$$

□

4. Experimental Results

We have implemented the FARSIDEIMG procedure in the symbolic model checker VIS 2.0 [1, 25], on top of both the MLP [17] and the Fine-Grain [16] algorithms. We compare our new algorithm with the standard MLP and Fine-Grain algorithm in the reachability analysis on 35 circuits from the public domain as well as industry. The "S" circuits come from the ISCAS'89 benchmark [15], the "D" circuits come from the industry, and the others come from the VIS verification benchmark [25]. All the experiments are run on an IBM IntelliStation with a 1.7 GHz Pentium IV CPU and 2GB of RAM; The data size limit for each process is set to 750MB.

Table 1 compares the run time and memory usage of our Far Side algorithm and MLP, with dynamic variable reordering method "sift". The image cluster threshold is set to the default value, 5000. Columns 1-3 show the name, number of binary state variables and number of inputs of each circuit. Columns 4-6 compare the CPU time; Columns 7-9 compare the peak number of live BDD nodes during the image computations. Note that all methods can not complete the last 5 circuits; the run time and peak live BDD nodes are for the reachability up to certain steps. For example, both the run time and the peak BDD nodes compared for *D14* are up to 12 steps; this is indicated in Column 1 by "(12)". Within the 8 hours time limit, FARSIDE was able to finish 1 more steps; this is indicated in Column 5 by "[13]".

The total run time shown for the 35 examples was 171,876 seconds for the original MLP, and 114,710 seconds for FARSIDEIMG. Overall this is a 33% win for FARSIDEIMG. However, note that the two entries, *am2901* and *palu*, for MLP are time-outs (8 hours). This means that the 33% win is a lower bound. It is instructive to put into one group all the "easy" circuits whose reachability analysis can be finished within 15 minutes (the first 15 circuits), to put into the second group all the "hard" circuits whose reachability can be finished by at least one method within 8 hours, and to leave the rest into another group, comprising the "harder" problems. By doing so, we get the following tally in which the average run time and the geometric mean of the peak live BDD nodes are compared separately for the three groups,

	Group "easy"			Group "hard"			Group "harder"		
	MLP	FARSIDE	%	MLP	FARSIDE	%	MLP	FARSIDE	%
CPU(s)	281	280	0	6871	4311	+37	14233	9972	+30
BDD(k)	157	161	-2	1181	845	+25	3348	2729	+18

These data show that the run time and the peak BDD size for Groups "hard" and "harder" average an order of magnitude larger than those for Group "easy". On the "easy" problems, our method does not win because of its additional overhead in approximation/refinement; On the "hard" and "harder" problems, however, controlling BDD size by using approximation/refinement inside the image computation pays off.

Table 1: Reachability analysis: FARSIDEIMG vs. MLP. With dynamic variable ordering.

			CPU (s)			Peak BDD nodes (k)		
Design	L#	I#	MLP	FarSide	%	MLP	FarSide	%
D12	48	16	6	7	-16	204	197	+ 3
abs_fabr	87	21	26	20	+25	43	46	- 5
D23	85	22	11	11	0	24	24	0
nosel	128	65	30	32	- 5	57	53	+ 7
bpb	36	9	33	53	-60	94	108	-14
shampoo	140	21	55	79	-44	91	98	- 7
soap	140	11	73	85	-17	101	97	+ 3
3_proc	62	18	103	106	- 2	213	183	+13
soapLtl3	142	11	360	329	+ 8	341	296	+13
s1512	57	29	364	435	-19	91	89	+ 2
Feistel	293	68	541	567	- 4	159	229	-43
D5	319	24	545	522	+ 4	250	301	-20
D1	101	76	556	452	+18	665	610	+ 8
s4863o	88	35	582	510	+12	402	402	0
cps1364o	134	97	598	672	-12	421	447	- 6
D21	92	6	626	610	+ 2	466	474	- 1
D2	94	6	1024	862	+15	765	745	+ 2
cps1364	231	97	1198	971	+18	363	372	- 2
s4863	104	49	1274	1037	+18	749	602	+19
D4	230	22	1370	1259	+ 8	540	493	+ 8
icctl	62	27	1462	1934	-32	1503	1515	0
s5378opt	121	35	1476	552	+62	508	335	+34
FIFOs	142	7	2129	1907	+10	1098	1021	+ 6
prolog	136	36	2443	7907	-223	1935	2287	-18
s3271	116	26	2627	1788	+31	1085	820	+24
s1269	37	18	3513	2958	+15	3588	3529	+ 1
D22	140	20	9351	12821	-37	3356	2834	+15
s3330	132	40	10733	2382	+77	3110	2922	+49
am2901	68	27	>28800	4827	>+83	-	2849	-
palu	37	10	>28800	19153	>+33	-	8985	-
D14 (12)	96	21	19978	6099 [13]	+69	4833	3066	+63
D15 (31)	106	31	12021	9795 [35]	+19	5855	4435	+24
D16 (16)	531	16	11264	6845 [17]	+40	3142	3142	0
D18 (23)	507	200	11994	11458	+ 4	1621	1385	+14
D20 (7)	562	31	15910	15666	+ 1	2926	2561	+12

We note there is one anomalous "hard" circuit, called *prolog*, in which MLP outperforms FARSIDEIMG by more than a factor of 3. We believe that the anomaly is due to the noise introduced by the BDD dynamic variable reordering during reachability analysis. To look at this effect we extracted the final BDD orderings (that is at the end of reachability analysis) for those circuits that can be finished with dynamic reordering, and with these fixed variable orderings we re-run all the experiments; for those "harder" circuits where reachability analysis can not be finished, we use the default fixed variable orderings generated by VIS's *static_order* command. The results are given in Table 2.

With fixed orderings, some circuits run much faster (such as *prolog*), some run much slower, for example *s5378opt*, which is about 4 times slower, and there are others, like *s3271*, run out of memory. Note in particular, for the anomalous circuit *prolog*, FARSIDEIMG has a marginal win (94 seconds vs. 104; 7200k BDD nodes vs.

Table 2: Reachability analysis: FARSIDEIMG vs. MLP. With fixed variable ordering. “M/O” means out of memory.

Design	L#	I#	CPU (s)			Peak BDD nodes (k)		
			MLP	FarSide	%	MLP	FarSide	%
D12	48	16	1	1	0	67	74	- 9
abs_fabr	87	21	36	35	+ 2	3	2	+11
D23	85	22	1	1	0	58	63	- 7
nosel	128	65	1	2	-66	0.02	0.03	-43
bpb	36	9	66	65	+ 2	0.5	0.5	0
shampoo	140	21	14	21	-53	1	1	0
soap	140	11	16	32	-101	925	1040	-12
3_proc	62	18	10	14	-42	1092	532	+51
soapLtl3	142	11	55	99	-81	4760	4836	- 1
s1512	57	29	837	1120	-33	24085	23469	+ 2
Feistel	293	68	4	5	-29	744	684	+ 8
D5	319	24	94	114	-21	4665	4158	+10
D1	101	76	82	87	-6	8469	8469	0
s4863o	88	35	57	54	+ 5	646	646	0
cps1364o	134	97	19	21	-11	2080	1993	+ 4
D21	92	6	224	305	-35	4649	3670	+21
D2	94	6	248	326	-31	3268	3149	+ 3
cps1364	231	97	25	24	+ 2	2021	1572	+22
s4863	104	49	59	35	+39	822	867	- 5
D4	230	22	79	117	-47	1035	1096	- 5
icctl	62	27	115	134	-15	3084	2268	+26
s5378opt	121	35	6960	6593	+ 5	24160	20221	+16
FIFOs	142	7	444	424	+ 4	4252	4710	-10
prolog	136	36	104	94	+ 9	9967	7200	+27
s3271	116	26	M/O	M/O	-	M/O	M/O	-
s1269	37	18	2668	2603	+ 2	44928	44928	0
D22	140	20	2414	3158	-30	1736	1716	+ 1
s3330	132	40	931	913	+ 1	24735	24736	0
am2901	68	27	M/O	M/O	-	M/O	M/O	-
palu	37	10	M/O	M/O	-	M/O	M/O	-
D14	96	21	M/O	M/O	-	M/O	M/O	-
D15	106	31	M/O	M/O	-	M/O	M/O	-
D16	531	16	M/O	M/O	-	M/O	M/O	-
D18	507	200	M/O	M/O	-	M/O	M/O	-
D20	562	31	M/O	M/O	-	M/O	M/O	-

9967k).

With fixed ordering the average run time (among those completed) was 576 seconds for MLP and 607 seconds for FARSIDEIMG. More importantly, we note that there are now 3 circuits in Group “hard” that can no be completed by either method because they run out of memory; for many circuits, the peak number of live BDD nodes are also order-of-magnitude higher than those in Table 1.

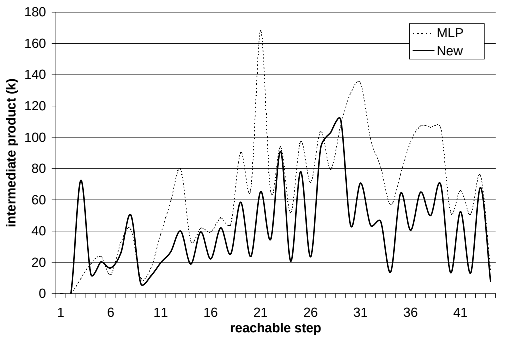
We claim that the data with dynamic reordering is more significant. In fact, computing an optimal fixed variable ordering is NP-hard, and it is generally accepted that dynamic reordering is required when dealing with industrial-strength circuits.

When based its implementation on the Fine-Grain method, FARSIDEIMG has demonstrated a similar improvement over the original Fine-Grain method.

5. Discussion of Hypothesis

One problem we faced in evaluating our hypothesis that controlling peak BDD size by using approximation/refinement inside the image computation has merit, is that the focus of the proposed method is on minimizing the BDDs used inside the image computation, not the overall BDDs used in the reachable analysis. The overall BDD data might not be very informative since a large part of the BDDs are used for representing the accumulated reachable states, which can become quite large near the end of the reachability analysis. Thus we performed some experiments which attempted to measure data more relevant to our hypothesis.

For the circuit *s5378opt*, Figure 3 compares FARSIDEIMG (solid line) and MLP (dotted line) on two different parameters: The BDD size of the intermediate products, and the peak number of live BDD nodes. Note that the latter includes the BDDs representing the accumulated reachable states. The BDD size of the intermediate products along the series of image steps is given in the upper figure (with dynamic reordering), and the peak number of live BDD nodes encountered during the image steps of reachability analysis is given in the lower figure. The horizontal axis shows the image steps, from 1 to 43, indicating the sequential depth of 43. The upper figure shows that except for a few iterations (e.g. Steps 2,3,6 and 8), the minimization is effective at reducing the maximum BDD size of the intermediate product.



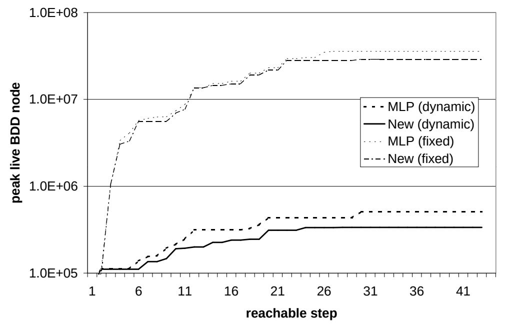


Figure 3: s5378opt: The upper part is the BDD size of the intermediate products at different steps during the reachability analysis; the lower part is the total number of live BDD nodes (includes also the BDDs representing the accumulated reachable states).

It is known that run times usually are determined by the maximum BDD size of the intermediate product encountered; the upper part in Figure 3 is more instructive in explaining the reason for the speed-up achieved by FARSIDEIMG. We note that in Figure 3 the peak occurs at iteration 21, where the MLP size is about 3 times larger than the FARSIDEIMG size. The FARSIDEIMG size peaked near iteration 29, at which the MLP size was about the same. The curve with fixed ordering is similar.

Figure 4 shows the effect of the transition relation minimization by FARSIDEIMG, i.e., its BDD size reduction in percentage at different image steps of the reachable analysis. From top down these data are for circuits *s5378opt*, *prolog*, and *s3271*. Data for the other circuits are similar. Each graph has 2 curves: One for dynamic variable reordering, and the other for fixed ordering. (The fixed ordering was taken at the end of the run with dynamic variable reordering.) For the first few iterations, the reduction in TR size is substantial. Note that 50% on the curve means that the BDD size of the minimized transition relation is half of the BDD size of the original one. As the iteration count grows, the size reductions saturate at a marginal value corresponding to a 0 to 40% reduction.

In the saturation phase (the right side of the curves) the reductions are greater when a fixed ordering is used. From the data for *s5378opt* in Tables 1 and 2 we see that even though the reductions never fell to less than 30%, reachability analysis is 5 times slower for MLP with fixed ordering than with dynamic reordering. This might appear to be anomalous since we have attributed time reductions with BDD size minimization. However, we note that these data are percentages, not absolute values. The size of the minimized TR for fixed ordering is still much larger than the size of the minimized TR for dynamic ordering, as can be seen from the minimized absolute values in the lower part of Figure 3.

The plateaus for *s5378opt* correspond to calls by the BDD manager to the reordering routine (these occurred at iterations 10, 17 and 27). In between these calls, the reductions follow a saturating pattern similar to the curves for fixed BDD ordering. Sometimes there is a final phase of increased reduction (the right side of the curves), due to the fact that image size decreases near the end of the reachability analysis. (A smaller image makes a better constraint for minimization.)

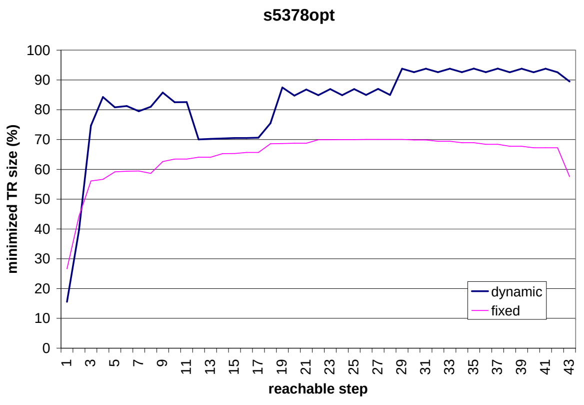
For circuit *prolog*, reachability analysis is more than an order of magnitude faster for fixed ordering. (The size of the transition relation for fixed ordering is about 2 times larger than for dynamic ordering.) This is a case where the BDD reordering itself takes a larger proportion of the time.

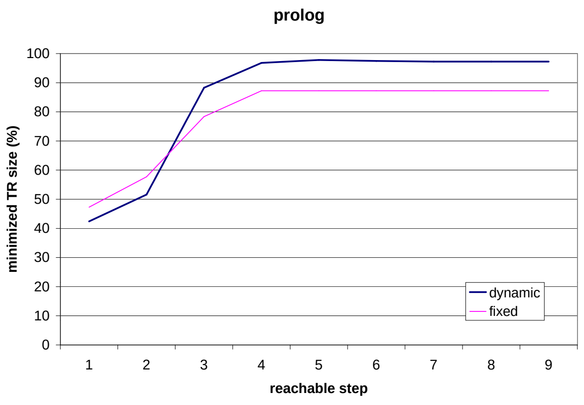
The bottom part of Figure 4 pertains to circuit *s3271*. With the fixed ordering, this circuit can complete only 4 iterations before running out of the 750MB memory, whereas with dynamic reordering, it can complete all 17 iterations in only about 30 minutes. Thus even though dynamic reordering makes it difficult to isolate the effects of algorithmic improvement, it appears to be the only viable option for some hard models.

6. Conclusions

We proposed an approximation/refinement approach for image computation, in which as much as possible don't care information is extracted and used to reduce the peak BDD size during the computation. Experiments have shown that our new method for compositional BDD minimization is effective on average, and especially effective on more difficult circuits. For circuits requiring more than 15 minutes for reachability analysis, FARSIDEIMG, implemented on top of MLP, was significantly faster than the standard MLP in 16 out of 19 cases. Due to the additional algorithmic overhead, performance was comparable and sometimes slightly worse for the easier circuits. It seems reasonable, therefore, to conclude that the new method is more robust in large industrial-strength applications.

In the future, we want to develop a hybrid approach, in which the abstraction/refinement is turned off if it is not providing significant BDD size reductions. Right now our new method is implemented on top of the VIS standard MLP and FINE-GRAIN methods, all the experimental data were based on a clustering algorithm designed to achieve different objectives. This should be regarded as a penalty function in favor of the standard MLP and FINE-GRAIN, since it is an open question if there is a specific clustering method that increases the performance of the new method. This also should be included in future work.





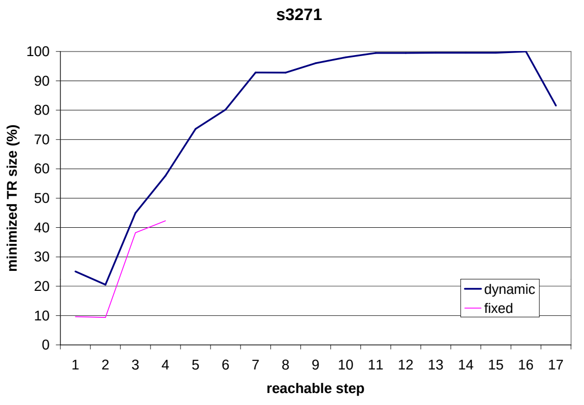


Figure 4: The BDD size reduction of the transition relation in percentage (i.e., the ratio of the BDD size of minimized transition relation to that of the original transition relation). With both dynamic variable reordering and fixed ordering. For three different circuits.

References

[1] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eighth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[2] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.

[3] J. Burch. Using BDD's to verify multipliers. In *1991 International Workshop on Formal Methods in VLSI Design*, Miami, FL, Jan. 1991.

[4] P. Chauhan, E. Clarke, S. Jha, J. Kukula, H. Veith, and D. Wang. Using combinatorial optimization methods for quantification scheduling. In T. Margaria and T. F. Melham, editors, *Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'01)*, pages 293–309. Springer-Verlag, Berlin, Sept. 2001. LNCS 2144.

[5] P. P. Chauhan, E. M. Clarke, S. Jha, J. Kukula, T. Shiple, H. Veith, and D. Wang. Non-linear quantification scheduling in image computation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 293–298, San Jose, CA, Nov. 2001.

[6] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. In *Proceedings of the Design Automation Conference*, pages 25–30, Dallas, TX, June 1993.

[7] H. Cho and F. Somenzi. Sequential logic optimization based on state space decomposition. In *Proceedings of the European Conference on Design Automation*, pages 200–204, Paris, France, Feb. 1993.

[8] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines based on symbolic execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, pages 365–373. Springer-Verlag, 1989. LNCS 407.

[9] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using Boolean functional vectors. In L. Claesen, editor, *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, Nov. 1989.

[10] O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 126–129, Nov. 1990.

[11] D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 299–310, Berlin, 1994. Springer-Verlag. LNCS 818.

[12] S. G. Govindaraju, D. L. Dill, A. J. Hu, and M. A. Horowitz. Approximate reachability with BDDs using overlappping projections. In *Proceedings of the Design Automation Conference*, pages 451–456, San Francisco, CA, June 1998.

[13] R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. In *Proceedings of the International Conference on Computer Design*, pages 12–19, Austin, TX, Oct. 1996.

[14] Y. Hong, P. A. Beerel, L. Lavagno, and E. M. Sentovich. Don't care-based BDD minimization for embedded software. In *Proceedings of the Design Automation Conference*, pages 506–509, San Francisco, CA, June 1998.

[15] Iscas benchmarks. URL: http://www.cbl.ncsu.edu/CBL_Docs/iscas89.html.

[16] H. Jin, A. Kuehlmann, and F. Somenzi. Fine-grain conjunction scheduling for symbolic reachability analysis. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'02)*, pages 312–326, Grenoble, France, Apr. 2002. LNCS 2280.

[17] I.-H. Moon, G. D. Hachtel, and F. Somenzi. Border-block triangular form and conjunction schedule in image computation. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 73–90. Springer-Verlag, Nov. 2000. LNCS 1954.

[18] I.-H. Moon, J.-Y. Jang, G. D. Hachtel, F. Somenzi, C. Pixley, and J. Yuan. Approximate reachability don't cares for CTL model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 351–358, San Jose, CA, Nov. 1998.

[19] I.-H. Moon, J. Kukula, T. Shiple, and F. Somenzi. Least fixpoint MBM: Improved technique for approximate reachability. Presented at IWLS99, Lake Tahoe, CA, June 1999.

[20] I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of the Design Automation Conference*, pages 23–28, Los Angeles, CA, June 2000.

[21] R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. Presented at IWLS95, Lake Tahoe, CA, May 1995.

[22] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the International Conference on Computer-Aided Design*, pages 42–47, Santa Clara, CA, Nov. 1993.

[23] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton. Heuristic minimization of BDDs using don't cares. In *Proceedings of the Design Automation Conference*, pages 225–231, San Diego, CA, June 1994.

[24] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDD's. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 130–133, Nov. 1990.

[25] Vis verification benchmarks. URL: http://vlsi.colorado.edu/∼vis.