Approximate Reachability Don't Cares for CTL Model Checking *

In-Ho Moon[†] **Jae-Young Jang**[†] **Gary D. Hachtel**[†] **Fabio Somenzi**[†] **Jun Yuan**[††] **Carl Pixley**[††]

[†]Dept. of ECE
University of Colorado, Boulder, CO
{mooni,jjang,hachtel,fabio}@vlsi.colorado.edu

[††]Design Verification
Motorola Inc., Austin, TX
{jun_yuan,carl_pixley}@email.mot.com

Abstract

RDCs (Reachability Don't Cares) can have a dramatic impact on the cost of CTL model checking [18]. Unfortunately, RDCs, being a global property, are often much more difficult to compute than the satisfying set of typical CTL formulas. We address this problem through the use of Approximate Reachability Don't Cares (ARDCs), computed with the algorithms developed for the VERITAS sequential synthesis package [4, 5]. Approximate Reachable states represent an upper bound on the set of true reachable states, and thus a lower bound on the set of unreachable (Don't Care) states. ARDCs can be 10X to 100X (or much more for very large circuits) cheaper to compute than RDCs, and in some cases have the same dramatic effect on CTL model checking as the real RDCs. We also discuss the application of ARDCs to the problem of exact computation of the RDCs themselves. Experiments on industrial benchmarks show that order of magnitude speedups are possible, and occur frequently. The experimental results presented strongly support our claim that ARDCs play a safe and important way out of a serious dilemma: RDCs are necessary for tractable model checking of many large circuits, but the computation of the RDCs themselves is often intractable. We include, and theoretically justify, significant extensions of the VERITAS algorithms, and show that they can be up to an order of magnitude faster, while computing a virtually identical upper bound.

1 Introduction

Although the effects are well known and intuitive, scant attention has been paid to RDCs (Reachability Don't Cares) in the prominent literature of traversal techniques for CTL model checking [7, 17, 3, 2, 8]. However, recent quantitative studies have shown that RDCs can have a dramatic impact on the cost of CTL model checking. For example, the Ethernet benchmark was shown [18] to model check more than 10X faster with the help of RDCs than without. Unfortunately, RDCs, being a global property, are often much more difficult to compute than the satisfying set of typical CTL formulas. In fact, command help in the VIS verification package [1] advises users not to use RDCs for large circuits. Unfortunately, larger circuits are often the ones for which RDCs are most beneficial.

We address this problem through the use of Approximate Reachability Don't Cares (ARDCs) in the VIS model checking package [15]. Approximate reachable states represent an upper bound on the set of true reachable states, and thus a lower bound on the set of unreachable (Don't Care) states. ARDCs can be 10X to 100X (or much more, simply because exact reachability analysis is intractable on many circuits) cheaper to compute than RDCs, and in some cases have the same dramatic effect on CTL model checking as the real RDCs. We further propose to investigate applying the cheaply generated ARDCs as a means of significantly reducing the cost of computing the (exact) RDCs. Results on several industrial benchmarks, of moderate size (hundreds of latches) have demonstrated that speedups of 8X-10X are attainable in some circuits, whereas ARDCs make the difference between intractability and tractability in other circuits.

The experimental results presented strongly support our claim that ARDCs play a safe, robust, and important way out of a serious dilemma: RDCs are necessary for tractable model checking of large circuits, but the computation of the RDCs themselves is often intractable.

We include, and theoretically justify, significant extensions of the VERITAS algorithms.We demonstrate that our extension of the VERITAS MBM algorithm, which we call FastMBM, can be up to an order of magnitude faster, while computing a virtually identical upper bound. We present a comprehensive theory that was able to satisfactorily explain several seemingly anomalous characteristics of our approximations.

In Section 2, we briefly recapitulate the VERITAS ARDC generation algorithms, and our extensions thereof. In Section 3, we discuss the general problem of exploiting RDCs in CTL model checking and reachability analysis (we generally follow the approach of [15], but show that some departures are necessary to obtain significant speedup). In Section 4, we discuss our experimental results, and then we conclude with some recommendations for future work.

2 Preliminaries

The approximate reachability analysis of a large FSM (circuit) M has two steps: (1) State Space Decomposition of M into m sub-FSMs M^j, and (2) Approximate Traversal of M, based on exact traversal of the individual M^j, while approximating their interaction. The set of reachable states $\widetilde{R}^j(s^j)$ is computed for each submachine, and the upper bound on the set of reachable states for M is taken as the cartesian product of the $\widetilde{R}^j(s^j)$. Large machines may be treated by increasing the severity of the overapproximation. In the extreme, the submachines don't interact at all. In this case, the upper bound is easily computed even for large m.

We first discuss the state space decomposition [5]. Suppose the overall FSM M is described by transition relation

$$T(s,x,t) = \prod_{i=1}^{n_S} T_i(s,x,t_i) = \prod_{i=1}^{n_S} (t_i \equiv \delta_i(s,x)).$$

where s and t are the overall vectors of n_S binary state variables (latches), δ_i is the state transition function for the i^{th}

*This work was supported in part by NSF grant MIP-94-22268 and SRC contract 96-DJ-560.

latch, and x is the vector of n_X primary inputs[1]. M may be decomposed into the following product of m sub-FSMs by simply grouping the state variables (latches), as follows.

$$T(s,x,t) = \prod_{j=1}^{m} T^j(s,x,t^j) = \prod_{j=1}^{m}(\prod_{i=1}^{n_S^j}(t_i^j \equiv \delta_i^j(s,x))).$$

Here t^j is a vector of n_S^j state variables, where the n_S^j satisfies

$$n_S = \sum_{j=1}^{m} n_S^j, \qquad \{0,1\}^{n_S} = \prod_{j=1}^{m}(\{0,1\}^{n_S^j}).$$

Thus the overall state space $\{0,1\}^{n_S}$ is decomposed into the Cartesian product of m state spaces. We perform state space decomposition with the algorithms of [5]. These algorithms are based on the functional dependencies of the transition functions $\delta_i(s,x)$, and on the functional correlations between these functions. So, from a given latch (seed), a latch with bigger dependency and correlation weight is aggregated.

Given the decomposition, we then use modifications of the basic procedures of [4] for computing the ARDCs. One of these modified procedures, called FastMBM (Fast Machine By Machine), is shown in Figure 1. We refer the reader to [4] for detailed descriptions of MBM and the other associated algorithms. Here, we limit ourselves to briefly characterizing FastMBM, and showing how MBM and a related variant, called TightMBM, are different from Procedure FastMBM.

Procedure FastMBM($\{T^j, j = 1, \ldots m\}$,$I(s)$) {
1 **for** ($j = 1, \ldots, m$) {
$\widetilde{I}^j(s^j)$ = ProjectInitial($I(s)$, j)
$\widetilde{R}^j(s^j)$= BDD_ONE
} /*Note $\widetilde{R}^j(s^j)$ depends only on (s^j).*/
2 **do** {
$Changed$ =FALSE
3 **for** ($j = 1, \ldots, m$) {
$Previous^j(s^j) = \widetilde{R}^j(s^j)$
4 $\widetilde{T}^j(s,x,t^j) = T^j(s,x,t^j)$
5 **for** ($i = 1, \ldots, m, \quad i \neq j$)
6 **if**($i \in FanIn(j)$) $\widetilde{T}^j = \widetilde{T}^j \dagger \widetilde{R}^i$
for ($i = 1, \ldots, m, \quad i \neq j$)
7 **if**($i \in FanIn(j)$) $\widetilde{T}^j = \exists_{s^i} \widetilde{T}^j$
8 $\widetilde{R}^j(s^j)$ =FsmTraversal($\widetilde{T}^j, \widetilde{I}^j(s^j)$)
if $(Previous^j(s^j) \neq \widetilde{R}^j(s^j))$ $Changed$ =TRUE
}
9 } **while** ($Changed$)
10 **return**({ $\widetilde{R}^j(s^j)$ })
}

Figure 1: FastMBM procedure for computing the ARDCs.

Procedure FastMBM begins with a **for** loop which calls subprocedure ProjectInitial, for each of the submachines M^j. This projects the initial state set $I(s)$, with full support, to the overapproximation $\widetilde{I}^j(s^j)$, which has only local support. This is done by existential abstraction of the non-local present state variables s^i, $i \neq j$. Then the reachable states sets for each submachine is initialized to tautology(Line 1).

[1]Note ([6]) that even nondeterministic machines can be handled in this way by adding extra primary inputs.

Then a **do-while** loop is entered, in which current approximations are refined. This loop is exited only when two successive passes produce the same overapproximation. Inside this loop (Line 3), a **for** loop is entered, which performs FSM traversal of the submachines in turn. This is done in a heuristic order determined by a "minimum feedback edge set" heuristic, as part of the state space decomposition step. This order tries to closely approximate a serial decomposition for the original overall FSM.

This main **for** loop begins by recording the previous version of the reachable state set $\widetilde{R}^j(s^j)$ of M^j. The boolean variable $Changed$ is processed so that the **do-while** loop is not exited if $\widetilde{R}^j(s^j)$ is changed for any one of the m submachines. In Line 4, the local transition relations $\widetilde{T}^j(s,x,t^j)$ are initialized to their original state, and then modified in the **for** loop of Line 5, which considers every other fanin submachine beside the one being traversed.

In Line 6, $\widetilde{T}^j(s,x,t^j)$ is restricted to the states reached previously by fanin submachines. The dagger stands for an arbitrary generalized cofactor operation. In this paper it will either refer to the down arrow, signifying the CONSTRAIN algorithm, or the double down arrow, signifying the RESTRICT algorithm [7]). Submachine M^i is considered to be in the fanin of submachine M^j if any of the variables of s^i is in the support of any of the δ^j of submachine M^j.

In Line 7, all state variables of other submachines behave like pseudo primary inputs and are existentially abstracted. This introduces a further overapproximation when $\widetilde{T}^j$ is partitioned transition relation.

The MBM algorithm may be obtained from FastMBM by deleting the existential abstraction of Line 7, and assigning the dagger to the down arrow and not allowing dynamic BDD variable ordering in Line 6 when $\widetilde{T}^j$ is partitioned transition relation, whereas FastMBM allows. TightMBM can be made by using the initial states(I(s)) of total machine instead of ProjectInitial($\widetilde{I}^j(s^j)$) from MBM, in order to get tighter upper bound. Since $\widetilde{I}^j(s^j)$ of j^{th} submachine is not always subset of the reachable states($\widetilde{R}^i$) of i^{th} submachine($i \neq j$), we may have extra reachable states. Therefore, by using I(s) we can get tighter upper bound. All of these algorithms are greatest fixed point procedures, with the embedded, essentially standard least fixed point of FsmTraversal.

We now present a theory which shows that all two nested fixed point procedures converge to valid overapproximations of the actual reached set of the given FSM.

Lemma 2.1 *Procedure FastMBM converges to the same result it would achieve if the generalized cofactor of Line 6 were replaced by its original form as a conjunction inside the image computations of Line 8.*

Proof. The first image computation of FsmTraversal is the following.

$$(\widetilde{R}^j)^1 = Img(T^j, \widetilde{I}^j \cdot \widetilde{R}),\ where\ \widetilde{R} = \prod_{i \neq j} \widetilde{R}^i.$$

From [13] Theorem 6, $f_{A \cdot B} = (f_A)_{B_A}$,

$$(\widetilde{R}^j)^1 = Img((T^j_{\widetilde{R}})_{\widetilde{I}^j_{\widetilde{R}}}, 1).$$

Now if A and B are disjoint, $f_{A \cdot B} = (f_A)_B$. So,

$$(\widetilde{R}^j)^1 = Img((T^j_{\widetilde{R}})_{\widetilde{I}^j}, 1) = Img(T^j_{\widetilde{R}}, \widetilde{I}^j)$$

Therefore, by induction,

$$\widetilde{R}^j = FsmTraversal(T^j, \widetilde{I}^j \cdot \widetilde{R}) = FsmTraversal(T^j_{\widetilde{R}}, \widetilde{I}^j)$$

□

Lemma 2.2 *If C(s) ⊆ R(s), then* $Img(T(s,x,t), C(s)) = Img(\widetilde{T}(s,x,t), C(s))$, *where* $\widetilde{T}(s,x,t) = \prod(T_i(s,x,t) \dagger R(s))$.

Proof. Let T(s,x,t) be original partitioned transition relation, and $\widetilde{T}$(s,x,t) be minimized transition relation with reachable states as follows. Again the dagger stands for generalized cofactor.

$$\begin{aligned} T(s,x,t) &= \prod T_i(s,x,t) \\ \widetilde{T}(s,x,t) &= \prod(T_i(s,x,t) \dagger R(s)), \end{aligned}$$

Since C(s) ⊆ R(s),

$$\begin{aligned} &Img(T(s,x,t), C(s)) = \exists_{s,x}[T(s,x,t) \cdot C(s)] \\ &= \exists_{s,x}[T(s,x,t) \dagger C(s)] \\ &= \exists_{s,x}[[\prod(T_i(s,x,t) \dagger C(s))] \cdot C(s)] \quad ..... \ (a) \\ &= \exists_{s,x}[[\prod(T_i(s,x,t) \dagger R(s))] \cdot C(s)] \quad ..... \ (b) \\ &= \exists_{s,x}[\widetilde{T}(s,x,t) \cdot C(s)] \\ &= Img(\widetilde{T}(s,x,t), C(s)) \end{aligned}$$

The reason for (b) from (a) is that if h ≥ g, then

$$g \cdot (f \dagger g) = g \cdot (f \dagger h).$$

□

Lemma 2.3 *Let* $M = Img(T, g \cdot h)$, *and* $F = Img(T \dagger g, h)$. *Then* $M \subseteq F$. *If* $h \subseteq g$, $M = F$.

Proof. $Img(T, g \cdot h) = Img(T \dagger g, g \cdot h) \leq Img(T \dagger g, h)$, since $g \cdot h \leq h$, and Img is monotonic. Now if $h \subseteq g$,$g \cdot h = h$, so in this case $F = M$. □

We now show how this lemma can be developed into a proof that FastMBM converges to an upper bound of the result of MBM, which is in turn an upper bound on true set of reachable states.

First however, note that this means that any generalized cofactor operation, in particular RESTRICT, and not just CONSTRAIN, can be used to minimize the transition relation while (conservatively) preserving the image. If $h \subseteq g$, the image is exactly preserved. This may not have been previously known. Further, it proves that this holds in the presence of dynamic BDD reordering, which again was not previously known.

Theorem 2.1 *Procedure FastMBM converges to an upper bound*

$$\widetilde{R} = \prod_j \widetilde{R}^j \geq R,$$

where $\widetilde{R}^j \geq R, \forall j$.

Proof. We first consider the case where the abstraction of Line 7 is excluded. By Lemma 2.3, and Lemma 2.1, it follows that each $\widetilde{R}^j$ computed in Line 8 is a valid upper bound of R, regardless of whether $\dagger = \downarrow$ or $\dagger = \Downarrow$. For the same reason, it follows that the bound produced is an upper bound if that is produced by FastMBM.

Now, consider the case where the abstraction of Line 7 is included. In this case a further upper bound approximation is imposed on the transition relation, which in turns leads to a further upper bound on the final result $\widetilde{R} = \prod_j \widetilde{R}^j$.

Finally, we have to prove convergence, considering the aforementioned two upper bound effects, due to use of RESTRICT (without satisfying $h \subseteq g$ in the application of Lemma 2.3), and due to the abstraction of Line 7. Since these were not part of MBM, the convergence proof for MBM does not carry over directly.

In fact, we have observed experimentally that the functional of the greatest fixed point is not monotonic in general, although it is in a strong majority of cases. Thus to prove convergence, we need the following additional lemma. □

Lemma 2.4 *FastMBM is monotonic if in the greatest fixed point computation we correct any non-contractions by a technique similar to that of [14]. Thus we use IteConstant to check whether each successive iterate is contained in the previous one. If this condition is violated, containment is restored by intersecting the new iterate with previous one.*

Proof. This "RestoreContainment" operation is guaranteed to produce a contraction by the elementary properties of conjunction. □

Note that this is somewhat informal in the sense that a contraction would be produced independent of all other previous considerations.

As a final point, note that the pseudo code of Figure 1 oversimplifies what was actually implemented in two respects. First, instead of updating every submachine on each pass through the refinement loop, the actual code implements an "event driven" procedure in which only the fanout machines of submachines whose reached sets have changed are scheduled for updating. This is the technique used in VERITAS, and significantly reduces the number of calls to FsmTraversal (Line 8).

Another oversimplification is the fact that the T^j have been treated as if they were monolithic transition relation blocks. In fact, the VIS "IWLS95" heuristic was used, in which each submachine is scheduled for early quantification in the partitioned transition relation approach [15]. So in actuality, the RESTRICT operations are done to the individual transition subrelations of the submachines.

3 Exploitation of ARDCs in CTL Model Checking and Reachability Analysis

In this section we discuss the deployment of ARDCs in both exact reachability analysis and CTL model checking. Since reachability analysis, often called FSM traversal, is actually "past tense" CTL model checking we will present just one procedure, even though VIS actually has separate procedures for these two activities.

VIS uses syntactic identities to parse arbitrary CTL formulas into parse trees containing calls to either **EX**p, **EG**p or **EU**(p, q). The CTL formula **EF**p is interpreted as **EU**(TRUE,p). However, for our purposes it suffices, and is most expedient, to discuss only **EF**p. In VIS, the actual procedures implemented use the ARDCs in exactly the way we show here.

The basic procedure for model checking with ARDCs is shown in Figure 2. The procedure we used for exact reacha-

bility analysis is just the very same procedure, with the PreImg subprocedure of Line 6 replaced by an Img subprocedure.

Procedure EF$p(\{T_i, i = 1, \ldots n\}, p, \{R_i^+, i = 1, \ldots m\}, I)$ {
1 **for** $(i = 1, \ldots, n)$ {
$\tilde{T}_i = T_i$
for $(j = 1, \ldots, m)$ {
$\tilde{T}_i = \tilde{T}_i \Downarrow R_j^+, \;\; j = 1, \ldots, m$
}
}
2 $L = U = p; \;\; Current = \emptyset$
3 **do** {
$Previous = Current$
4 $To(t)$ =BddBetween(L, U)
5 $From(s) =$ PreImg($\{\tilde{T}_i, i = 1, \ldots, n\}, To(t), \{R_i^+\}$)
6 $New(s) = From(s) - Previous(s)$
7 $Current(t) = Previous(t) + New(t)$
8 $U = Current$; L = New
9 }**while** $(Previous \neq Current)$
10 **if** $(I(s) \subseteq Current(s))$**return**(TRUE)
else return(FALSE)
}

Figure 2: Procedure for Deciding $\mathbf{EF}p$.

In Line 1, the partitioned transition relation clusters are restricted with respect to the reached states of each submachine formed by state space decomposition. This BDD minimization is responsible for a large part of the savings compared to model checking without reachability don't cares. The **do-while** loop of Lines 3-9 compute the least fixed point of the $\mathbf{EF}p$ predicate transformer, using the method of [15].

Notice that the Approximate rechable states are passed as a care set to the PreImg computation of Line5. Thus unreachable states can be used as Don't Cares in the PreImg computation. The call to BddBetween tries to return the smallest Bdd among all sets which contain New and are contained in $Current$.

4 Experimental Results

4.1 ARDC-Accelerated Model Checking

Table 1 and Table 2 show the result of model checking with ARDCs. In these two tables, the column headers may be understood as follows. #Latches is the number of latches in each design, #CTLs is the number of CTL formulas, and #ARDCs is the number of approximate unreachable states, #reached is the number of exact or approximate reachable states depending on dcLevel option in VIS, #bdd is the size of the bdd of exact or approximate reachable states, #peak is the peak bdd size during model checking, Trch is the time for exact or approximate reachability analysis, Tmc is the time for model checking.

For each circuit we decomposed the original machine into submachines with 8 latches each. This is arbitrary, and indicates that we have not tried to tune the state space decomposition to the particular problems at hand.

In the data tables that follow, the circuit names are qualified by a suffix indicating the VIS runtime options used to obtain the data. The "n", "r", and "a" suffixes indicate the VIS `model_check` command don't care options. The circuit name followed by the letter "n" signifies VIS computations without DCs(Don't Cares), whereas the "r" suffix denotes with exact reachability analysis and the "a" suffix denotes with approximate reachability analysis. In both "r" and "a" cases, the unreachable states are used as don't cares in the fixed point computations of model checking. Here we use the FastMBM method as the reference method for ARDC computations.

The data was compiled on the following machines. An Ultra Sparc 1 (167Mhz, with 192MB RAM) was used for circuits cps, ethernet, exam1, model Table 1. A Pentium-pro (200Mhz, with 256MB RAM) was used for cps in Table 3. A DEC Alpha (300Mhz, with 256MB RAM, using 32-bit pointers) was used for fabric and hw_top in Table 1, and for s3271 and s3330 in Table 3. Finally, in Table 2, a second Ultra Sparc 1 (167Mhz, with 128MB RAM) was used for design1, design2, design3, and design4.

Most model checking examples are industrial examples, and they are divided into two groups. Those of the first group (Table 1) were obtained and run at our site (but most are not redistributable), whereas those of the second group (Table 2) were proprietary and run on-site by our industrial co-authors. This required us to write a BLIF-MV translator from the internals of the industrial verifier. Thus the Verilog description was translated into an intermediate form by the industrial verifier, and then the intermediate form was translated into BLIF-MV for input into VIS.

We can say something about some of the Group 1 examples. Circuit cps is a model of the control circuitry of the landing gear of an aircraft. The ethernet circuit is a model of the Ethernet protocol to communication between a set of processors. The definition of this system includes several parameters that can be used to scale up the size of the design. The specification consists of 6 CTL formulas. Production cell is a control circuit for automated manufacturing with 61 memory elements [12]. The specification contains 38 formulas.

The circuits of Table 1 with only 1 CTL formula to be checked were circuits that got to us without CTL formulae, like cps. For these circuits, we used the "dead-lock free" property **AG(EF**$reset$). However, all the circuits of Table 2 had "Industrial" CTL formulas.

The salient features of this table are illustrated in Figure 3. Here the two data series represent Tmc_{VIS}/Tmc_{ARDC} with

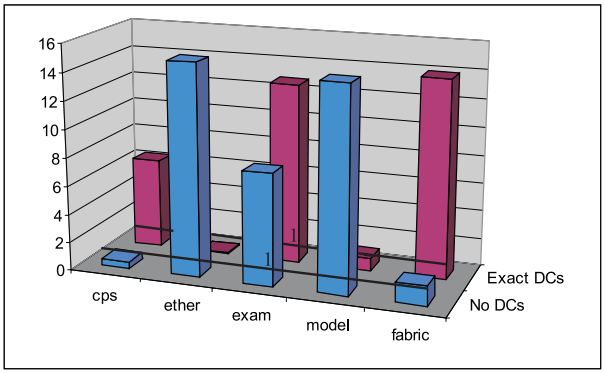

Figure 3: Accelerating Model Checking for Group 1 Industrial Designs.

total time that is the sum of Trch and Tmc, the last two columns in tables. The series are for VIS model checking with and without DCs. Note that 2 of the 10 cases results in space-out (256MB), 4 of 10 had a speedup of more than 10X, and 6 of 10 had a speed up of more than 6X.

In only 2 of the 10 cases (cps and hw_top without DCs), VIS without DCs was faster in total time. In circuit cps, model

design	#Latches	#CTLs	#ARDCs
cps	231	1	3.45087e+69
ethernet	78	6	3.02231e+23
exam1	115	34	4.15384e+34
production cell	61	38	3.08140e+16
fabric	190	1	1.10239e+57
hw_top	356	1	1.46784e+107

	Model Checking with ARDCs–Group 1				
name	#reached	#bdd	#peak	Trch	Tmc
cps_n			6.00e+05		637.8
cps_r	1.11e+10	36685	1.90e+06	8311.0	264.5
cps_a	1.27e+50	203	1.40e+06	1297.9	38.7
eth_n			2.50e+06		57293.3
eth_r	8.62e+02	2974	6.64e+04	15.0	23.1
eth_a	5.22e+16	183	6.15e+05	4.1	835.3
exm_n			3.98e+05		1224.2
exm_r	1.44e+17	113388	4.14e+05	1450.0	536.7
exm_a	1.19e+27	104	3.61e+05	16.0	139.2
mod_n			2.10e+06		8515.0
mod_r	3.41e+04	7804	6.64e+04	3.2	536.9
mod_a	3.70e+08	543	6.77e+05	7.6	582.8
fab_n			8.48e+05		139.5
fab_r				Memory out 256MB	
fab_a	5.45e+56	46	4.13e+05	22.7	77.1
top_n			3.14e+05		317.3
top_r				Memory out 256MB	
top_a	5.35e+81	182	8.85e+05	216.6	274.4

Table 1: Model Checking with ARDCs–Group 1 Industrial Designs

checking with ARDCs was much (6X) faster than with RDCs. However, in this case, model checking without ARDC or RDC was 2X faster than with ARDCs. This is explained by the fact that of the $2^{231} = 3e + 69$ total states, very few, only $1e + 10$ were reachable. This circuit is known to have significant latch redundancies. However, if we focus only on the time for model checking (Tmc), model checking with ARDCs is the best. Also, this Tmc is only for one CTL property. So, if we assume that we have many CTL properties, then the total times may change a lot.

Circuit ethernet was similar in this respect but the results were very different. Of the $2^{78} = 3e + 23$ total states, extremely few, only 862, were reachable. This means that virtually every state was unreachable, and therefore don't care. Thus it is not surprising that model checking with RDCs (suffix "r") is three orders of magnitude faster than without (suffix "n"). Even though model checking with RDCs is 20X faster than with ARDCs, model checking with ARDCs is still 70X faster than without DCs.

In terms of circuit exam1, model checking with ARDCs is the best, by almost an order of magnitude. Note that though RDC reached only on the order of 10^{17} states (out of 4e+34), ARDC obtained a superset that was about 10 orders of magnitude larger.

In case of "production cell", model checking with ARDCs is very comparable to model checking with RDCs. Both with RDCs and with ARDCs is 15X faster than without Don't Cares.

Circuits fabric and hw_top highlight the operational dilemma faced by verification engineers. If they have seen circuits like ethernet or production cell, they might be seriously inclined to keep the RDC option on. But then they get hammered on circuits like fabric and hw_top for which reachability analysis is essentially intractable. This gives verification a bad name. In fact the VIS help facility advises designers *not* to use reachability don't cares for large circuits. Clearly ARDCs provide a way out of this dilemma.

design	#Latches	#CTLs	#ARDCs
design1	217	1	2.106250e+65
design2	158	1	3.100960e+47
design3	172	1	5.939540e+51
design4	115	56	4.153840e+34

	Model Checking with ARDCs–Group 2				
name	#reached	#bdd	#peak	Trch	Tmc
des1_n			7.95e+05		1788.5
des1_r				Memory out 256MB	
des1_a	1.49e+57	137	9.03e+05	31.4	1007.2
des2_n			1.60e+06		3142.4
des2_r				Time out 48000sec	
des2_a	1.61e+47	54	1.20e+06	6.8	1612.5
des3_n				Memory out 128MB	
des3_r				Memory out 128MB	
des3_a	4.68e+49	2472	4.63e+05	88.8	500.4
des4_n			1.30e+06		20089.8
des4_r	1.44e+17	78360	8.50e+05	613.0	440.5
des4_a	1.10e+27	462	8.82e+05	13.8	2225.7

Table 2: Model Checking with ARDCs–Group 2 Industrial Designs

Table 2 has similar results. For designs 1-3, reachability analysis is intractable. For designs 1 and 2 model checking is tractable without RDCs, but the results with ARDCs are faster. For design 3, model checking is intractable without RDCs (128MB memory out), and, exact reachability analysis is also intractable for the same reason. But model checking with ARDCs completes in under 10 minutes on an UltraSparc 1. This result is to be emphasized, because the circuits of Group 2 are not the full scale circuits but "reduced" versions, for which exact reachability analysis is much more intractable.

Comparing designs 3 and 4, note the aforementioned dilemma re-emerging. Design 4 cries out for reachability don't cares, while design 3 suggests the impossibility of obtaining them exactly. Again, ARDCs offer a robust way out of the dilemma.

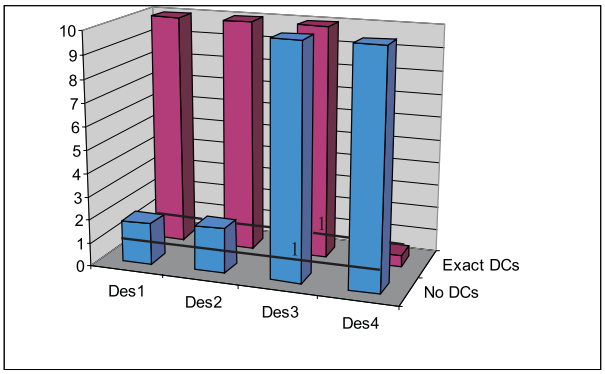

Figure 4: Accelerating Model Checking for Group 2 Industrial Designs.

The salient features of this table are illustrated in Figure 4 and Figure 5. These are ratio tables, with the 1 (breakeven) lines indicated in bold for each series. In the acceleration table, 5 out of the 8 cases show spaceout or timeout, and hence could be regarded as an infinite speedup. However the figure

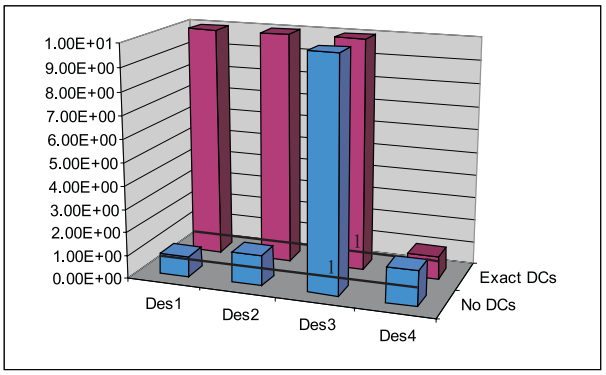

Figure 5: Compressing Model Checking for Group 2 Industrial Designs.

design	Depth	#Latches	#Reached	#ARDCs
cps	10	231	1.10834e+10	3.45087e+69
s1269	10	37	1.13134e+09	1.76161e+10
s3271	17	116	1.31768e+31	4.15384e+34
s3330	9	132	7.27780e+17	5.44396e+39

	Exact Reachability Analysis with ARDCs				
name	#BddAr	#BddEx	#BddPk	Tarch	Trch
cps_Ex		41640	2.4e+06		11396.4
cps_Ar	203	41740	2.2e+06	518.8	6370.5
s1269_Ex		1652	1.4e+08		58818.7
s1269_Ar	25	1376	8.6e+07	6.6	18501.8
s3271_Ex				Memory out 256MB	
s3271_Ar	17	189182	2.3e+06	12.8	5481.4
s3330_Ex				Memory out 256MB	
s3330_Ar	62	54842	3.0e+06	5.4	8287.5

Table 3: Exact Reachability Analysis with ARDCs

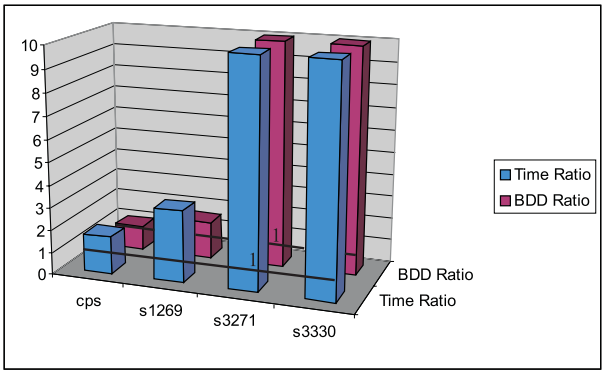

Figure 6: Accelerating Exact Reachability Analysis.

just truncates these to show a 10X speedup. In only 1 case, Des4, does VIS (with exact RDCs) beat VIS with ARDCs, in this case by about a factor of 2 in total time. The minimum speedup in the remaining 7 out of 8 cases was 1.8. In the compression table we compare the ratio of peak overall memory requirements The results were roughly similar to those for acceleration. ARDC won in every case, except for 2 narrow losses (.88 and .95).

4.2 ARDC-Accelerated Reachability Analysis

In this section we present some results of using ARDCs to accelerate exact reachability analysis. The results are similar to those obtained for ARDC accelerated model checking. While this may seem surprising, similar results should actually be expected because the dominant mechanism for improvement derives from the BDD minimization in inner **for**-loop of Line 1 in Procedure **EF**p. The only difference between reachability analysis (past tense **EF**p) and model checking eventualities (future tense **EF**p) is the substitution of Img computations for PreImg computations.

We might expect some degradation because we usually think of reachability don't cares in terms of PreImg computations, which routinely try to bring large numbers of unreachable states into the fixed point computations. This effect doesn't really occur in Img computations (all image states are by definition reachable), and could affect and limit the degree of speedup in cases when exact reachability analysis is tractable. However, the results show that ARDCs can definitely make the difference between tractability and intractability.

Table 3 shows the results of exact reachability analysis with ARDCs. In this table, Depth is FSM depth of design, #Latches is the number of latches in each design, #Reached is the number of reachable states, #ARDCs is the number of approximate unreachable states, #BddAr is the bdd size of approximate reachable states, #BddEx is the bdd size of exact reachable states, #BddPk is the peak bdd size during reachability analysis, Tarch is the time for computing approximate reachable states, Trch is the time for computing exact reachable states. Here cps is as discussed above, and s1269, s3271 and s3330 are ISCAS89 FSM benchmarks.

The salient features of this table are illustrated in Figure 6. The data of Table 3 shows speedups of almost a factor of 2 for cps and a factor of 3 for s1269. Reachability analysis was intractable for s3330 and s3271 within the available memory. So in the large circuits, the data for exact reachability analysis using ARDCs is quite analogous to the the results for model checking with ARDCs. This was to be expected.

However, in a separate investigation, it was found that by using non-standard BDD techniques in VIS (enabling variable reordering only after the partitioning step, which is not consistent with standard usage), s3330 could be traversed exactly. This emphasizes again the gross dependence of the quality of the results on the vagaries of dynamic reordering. It is clear that verification engineers need robust tools that do not require the user to fiddle combinatorially with all the options, parameters and operation sequences that are possible.

4.3 Comparison between MBM and FastMBM

Table 4 shows the comparison between MBM and FastMBM. In the table, Am2901 is 4-bit ALU slice, Am2910 is a microprogram sequencer, soap is a model of a token-passing mutual exclusion algorithm for networks with arbitrary topology, and s5378, s13207, and s15850 are ISCAS89 FSM benchmarks. According to this table, FastMBM is up to 9X faster than MBM without losing much accuracy. There were a few accuracy degradations in only 2 designs(cps and s13207) out of 13. This speedup and some accuracy degradations come from Line 7 in Procedure FastMBM, using RESTRICT operation instead of CONSTRAIN, and enabling variable reordering during Line 6 in Procedure FastMBM. However, in some designs, FastMBM was slightly(less than 1.4X) slower than MBM. This might be because RESTRICT operations are not always faster than CONSTRAIN. In terms of peak BDD size, FastMBM uses mostly less peak BDD nodes than MBM, even when FastMBM loses in time. This is mainly because of

Line 7 in Procedure FastMBM that makes transition relation smaller in terms of BDD size, and since RESTRICT does not introduce a BDD node of new variable that is not in the support of transition relation(while CONSTRAIN does), RESTRICT produces smaller BDDs than CONSTRAIN, in general.

Even though FastMBM is not always faster than MBM, there are many big wins(Am2901, cps, and soap), but no big lose in time and accuracy in the table, and mostly FastMBM uses smaller BDDs. This seems that FastMBM has more possibility to deal with larger designs than MBM.

name	method	#reached	Tarch	#peak
Am2901	mbm	2.95148e+20	816.6	1.3M
	fast	2.95148e+20	93.6	0.3M
Am2910	mbm	1.16057e+26	23.8	0.2M
	fast	1.16057e+26	18.2	0.2M
cps	mbm	3.29911e+53	2245.8	1.9M
	fast	3.22907e+56	444.0	1.5M
exam1	mbm	7.03245e+33	23.9	0.1M
	fast	7.03245e+33	16.8	0.1M
fabric	mbm	1.17696e+57	23.3	0.1M
	fast	1.17696e+57	26.2	0.1M
hw_top	mbm	2.44726e+85	244.7	0.9M
	fast	2.44726e+85	283.2	1.0M
soap	mbm	1.06458e+36	74.5	0.1M
	fast	1.06458e+36	38.6	0.1M
s1269	mbm	7.51619e+09	5.3	0.1M
	fast	7.51619e+09	6.2	0.0M
s3271	mbm	6.23076e+34	8.7	0.1M
	fast	6.23076e+34	11.8	0.0M
s3330	mbm	3.16020e+23	5.6	0.1M
	fast	3.16020e+23	7.0	0.1M
s5378	mbm	4.35038e+46	69.8	0.3M
	fast	4.35038e+46	54.4	0.3M
s13207	mbm	8.94121e+116	1916.9	1.4M
	fast	6.17119e+117	1892.1	1.3M
s15850	mbm	6.04910e+100	1408.4	1.5M
	fast	6.04910e+100	1320.8	1.5M

Table 4: MBM vs. FastMBM Comparison

We also have experimented TightMBM for all examples in Table 4. We got the reachable states(and time) by TightMBM as follows: 1.88265e+53(3479.9 secs) in cps(43% decreased), 4.67157e+116(2017.1 secs) in s13207(48% decreased), 5.21505e+46(113.1 secs) in s5378(16% increased), and the rest got the same upper bound. However, as in s5378, this method may give looser upper bound because generalized cofactor operations depend on variable orders.

4.4 Forward vs. Backward Model Checking with ARDCs

We have implemented forward model checking by Iwashita [11, 10] to see the correlation between forward model checking and using don't cares in model checking, because they are in common in that they try to avoid traversing unreachable states in fixpoint computation of model checking. Forward model checking is based on forward state traversal, while backward model checking is based on backward state traversal. Table 5 compares the performance of forward model checking with the one of backward. In the methods of the table, ndc means not using don't cares, rdc means using exact don't cares, and ardc means using approximate don't cares.

The design ethernet and production cell show this correlation clearly. The exact reachable states of the two designs are very small compared to total state space of those. This means that most states of the designs are unreachable. Therefore, the forward model checking of the two designs is always very fast no matter what method was used. In case of ethernet, the speed of forward model checking is very same as the best case of backward model checking, and even much faster than the best of backward model checking in production cell.

However, forward model checking is much slower than backward in cps and exam1, and we observed this case in more designs not shown in this table. Therefore, forward model checking is not always faster than backward, however according to our experiment forward model checking is the way to go when the exact reachable states are relatively small compared to total state space, and ARDCs seem more important in backward model checking than in forward.

By using the Iwashita's conversion method from future tense CTLs to his past tense forward operators such as FwdUntil and FwdGlobal, the first operation of all properties we have used was FwdUntil(init,ture) that is to compute all reachable states from initial states. This means that forward model checking is available only when reachability analysis is tractable, while ARDCs can be used regardless of reachability analysis.

		Forward		Backward	
name	method	#peak	Tmc	#peak	Tmc
cps	ndc	0.6e+06	9751.9	0.6e+06	637.8
	rdc	5.5e+06	14919.2	1.9e+06	264.5
	ardc	3.3e+06	14645.7	1.4e+06	38.7
ethernet	ndc	0.1e+06	22.9	2.5e+06	57293.3
	rdc	0.1e+06	23.9	0.1e+06	23.1
	ardc	0.1e+06	35.2	0.1e+06	835.3
exam1	ndc	2.1e+06	19246.0	0.4e+06	1224.2
	rdc	2.6e+06	22085.1	0.4e+06	536.7
	ardc	1.8e+06	2986.3	0.4e+06	139.2
model	ndc	0.1e+06	29.1	2.1e+06	8515.0
	rdc	0.2e+06	27.7	0.1e+06	536.9
	ardc	0.1e+06	33.5	0.7e+06	582.8

Table 5: Forward vs. Backward Model Checking with ARDCs

5 Conclusions and Future Work

Our basic conclusion is that ARDCs pay their way for both reachability analysis and model checking. Although not a uniform win in every case, the data clearly show that for many medium circuits, ARDCs will be an outright winner, and that a package with ARDC capability will be significantly more robust than one without it. The experimental results presented strongly support our claim that ARDCs offer a safe and important way out of a serious dilemma: RDCs are necessary for tractable model checking of large circuits, but the computation of the RDCs themselves is often intractable.

Throughout our experiments there has been a persistent and complex interplay between BDD reordering and minimization, and the deployment of ARDC techniques. The results show that our approach is convergent and gives good quality abstractions.

The absence of really large circuits from our data tables indicate that there is much more to be done before circuits with thousands of latches can be handled routinely. However, our results to date, both published and too raw too publish, show that very large circuits will require RDCs, and that exact RDCs will be too expensive for large circuits. Unfortunately, to address large problems, the verification and BDD packages have

to be improved across the board, and not just with respect to model checking or reachability analysis.

Fortunately, there is still much room for improvement in ARDC technology. For example, in cases where the RDC computation is intractable, and yet the CTL formulas are strongly aided by the RDCs, we could combine ARDC accelerated RDC with model checking for possible further improvements. There is much room left for improving the accuracy of the approximations. Approximations from different algorithms can be intersected to produce tighter upper bounds, techniques like those of Warwukiewicz (Berkeley 1994 unpublished) and Govindaraju et al [9] show that overlapping subsystems in the state space decomposition can be advantageous.

Another possibility is the cooperative deployment of other upper bounding techniques such as BDD subsetting methods [16], which constitute automatic abstraction methods that also have significant impact on verification time and space requirements.

Also, despite the relative maturity of BDD dynamic reordering technology, we believe that new methods will emerge that will be necessary and productive for dealing with very large circuits. We foresee that decomposition and factorization methods will need to be incorporated into sifting and other ordering optimization strategies. We have yet to try static BDD orderings in which the BDD variables in the respective subsystems are non-interleaved. Similarly, CUDD supports group constrained sifting. This offers the possibility of significantly reducing reordering times (which are, quite consistently, the dominant component of the cpu consumption profile).

We also included the results of experiments which supported the hypothesis that ARDC effects offer the same beneficial effects as converting CTL formulae from the future tense to the past tense. Our experiments show that in a mature system, both of these effects should be exploited.

References

[1] R. K. Brayton et al. VIS: A system for verification and synthesis. Technical Report UCB/ERL M95/104, Electronics Research Lab, Univ. of California, December 1995.

[2] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.

[3] H. Cho, G. D. Hachtel, S.-W. Jeong, B. Plessier, E. Schwarz, and F. Somenzi. ATPG aspects of FSM verification. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 134–137, November 1990.

[4] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. *IEEE Transactions on Computer-Aided Design*, 15(12):1465–1478, December 1996.

[5] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi. Automatic state space decomposition for approximate FSM traversal based on circuit analysis. *IEEE Transactions on Computer-Aided Design*, 15(12):1451–1464, December 1996.

[6] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[7] 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, November 1989.

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

[9] S. Govindaraju, D. Dill, A. Hu, and M. Horowitz. Approximate reachability with bdds using overlapping projections. In *Proceedings of the Design Automation Conference*, pages 451–456, 1998.

[10] H. Iwashita and T. Nakata. Forward model checking techniques oriented to buggy designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 400–405, San Jose, CA, November 1997.

[11] H. Iwashita, T. Nakata, and F. Hirose. CTL model checking based on forward state traversal. In *Proceedings of the International Conference on Computer-Aided Design*, pages 82–87, San Jose, CA, November 1996.

[12] Thomas Lindner. *Case Study "Production Cell": A Comparative Study in Formal Software Development*, chapter 2, pages 9,21. FZI, 1994.

[13] K. L. McMillan. A conjunctively decomposed boolean representation for symbolic model checking. In R. Alur and T. A. Henzinger, editors, *8th Conference on Computer Aided Verification (CAV'96)*, pages 13–25. Springer-Verlag, Berlin, August 1996. LNCS 1102.

[14] A. Pardo and G. D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In O. Grumberg, editor, *Ninth Conference on Computer Aided Verification (CAV'97)*. Springer-Verlag, Berlin, 1997. LNCS 1254.

[15] 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.

[16] K. Ravi and F. Somenzi. High-density reachability analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 154–158, San Jose, CA, November 1995.

[17] 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, November 1990.

[18] J. Yuan, J. Shen, J. Abraham, and A. Aziz. On combining formal and informal verification. In O. Grumberg, editor, *Ninth Conference on Computer Aided Verification (CAV'97)*, pages 376–387. Springer-Verlag, Berlin, 1997. LNCS 1254.