Efficient Model Checking by Automated Ordering of Transition Relation Partitions

Daniel Geist and Ilan Beer

IBM Science and Technology, Haifa, Israel

Abstract. In symbolic model checking, the behavior of a model to be verified is captured by the transition relation of the state space implied by the model. Unfortunately, the size of the transition relation grows rapidly with the number of states even for small models, rendering them impossible to verify. A recent work [5] described a method for partitioning the transition relation, thus reducing the overall space requirement. Using this method, actions that require the transition relation can be executed by using one partition at a time. This process, however, strongly depends on the order in which the partitions are processed during the action.

This paper describes a criterion for ordering partitions which is independent of the circuit details. Based on this criterion, a heuristic algorithm for ordering partitions is described. The algorithm, which may be used in preparation for each symbolic simulation step, has been successfully implemented and has resulted in significant speed-ups of symbolic model checking. Specifically, this algorithm has made it possible to verify blocks inside an example microprocessor. The run time results are given here.

1 Introduction

The feasibility of formal verification methods is usually limited by the "state explosion problem", where the representation size of the model to be verified grows exponentially with the number of model states. Therefore, although formal verification methods have existed for many years, they have become practical only when research on reducing the representation size became successful.

The work on Binary Decision Diagrams (BDDs), and in particular Ordered Binary Decision Diagrams (OBDDs) done by Bryant [3, 2, 4], has had a big impact on formal methods [11, 7]. As a result, formal methods have become practical for moderate size models, successful systems such as SMV [11] were developed, and formal verification of real circuit designs has become feasible [10, 6].

However, even with the use of BDDs, the size of a verifiable model is still limited, and research has continued in order to find other methods of reducing the size of model representation. In systems that search the state transition graph, the model is represented as a BDD of its transition relation. The size of the BDD of the transition relation is usually the obstacle to verifying bigger circuit designs.

Recent research has shown that it is possible to significantly reduce the amount of space required by the transition relation, if it is not kept as a whole [5, 7, 12]. The space required by the transition relation can be much smaller if it is partitioned into

many small relations whose combination produces the transition relation. This partition usually increases computation time but allows the verification of bigger circuit designs.

Specifically, in a recent paper by Burch, Clarke and Long [5], it was shown that the transition relation of a model can be partitioned into smaller relations, each handling the next value of a single state variable. They showed that it is possible to do the same operations as with the transition relation without the same space requirements. Operations with the partitioned transition relation can be executed iteratively, one partition at a time. This partitioning has been successfully used in the verification of models not previously possible [5].

The above process, however, strongly depends on the order in which the partitions are introduced to the operation. In [5] it was stated that a good order can be derived by examining the model to be verified and its semantics. This process, however, requires intimate knowledge of the circuit. Furthermore, this process requires manual intervention in order to derive and input the order. Thus, in order to fully automate the use of a partitioned transition relation, it is necessary to find a method of ordering the partitions that is independent of knowledge of the semantics of the circuit.

This paper describes a criterion for ordering partitions which is independent of the model details. Based on this criterion, a heuristic algorithm for ordering partitions is described, which can be run prior to symbolic model checking. This algorithm has been successfully implemented and has resulted in significant speed ups of symbolic model checking. Specifically, this algorithm has made it possible to verify blocks inside an example microprocessor. The run time results are given here.

The rest of the paper is partitioned as follows: Section 2 gives a brief background of basic computation operations performed in symbolic model checking. Section 3 reviews the work on partitioning done by Burch et al. [5] on partitioning the transition relation. Section 4 describes the criterion for ordering and Section 5 presents the algorithm which was devised, based on this criterion. Section 6 presents experimental results of the use of this work.

2 State Set Computation in Symbolic Model Checking

Throughout the paper it is assumed that the model to be verified is a design of a synchronous circuit, and the work in the paper is only relevant to this kind of a model.

The basic operations performed in the process of symbolic model checking are computations of the next or previous set of states of a given set of states. Symbolic Model Checking [11] does not operate on individual states (or paths). Rather, it performs its processing on sets of states that satisfy a certain predicate (condition). Finding the next set of states of a given set is called a *forward simulation step*. Similarly, finding the previous set of states of a given set is called a *backward simulation step*. Thus, a forward (or backward) simulation step is done from one set of states to another set of states. Additionally, the simulation is exhaustive, i.e., all possible states of the next step are generated.

The model checker that this work relates to is SMV [9, 11]. SMV uses OBDDs to represent sets of states. Assume that the model to be verified has n binary state variables. Let $M \subseteq \{0,1\}^n$ be its state space, and let $V = \{v_1, \ldots, v_n\}$ be its set

of state variables. A state $q \in M$ of the model is an assignment of binary values to v_i. Given a set $S \subseteq \{0,1\}^n$, we associate with S the boolean function $S(V)$, where $S(q) = 1$ iff $q \in S$.

In order to perform a forward (or backward) simulation step, a transition relation of the model to be verified must first be constructed. Let $V' = \{v'_1, \ldots, v'_n\}$ also denote the model set of state variables but designating some possible "next state" of V. The transition relation $N(V, V')$ of a model is a boolean function of $2n$ variables, such that $N(q, q') = 1$ iff $q' \in M$ is a possible "next state" of $q \in M$.

The transition relation captures the model's behavior inside a boolean function. We can then efficiently store it as a OBDD. For example, Figure 1 depicts the transition relation of a 3 bit counter. b_0, b_1 and b_2 are the current state variables where b_0 is the least significant bit. Respectively, b'_0, b'_1 and b'_2 are the next state variables.

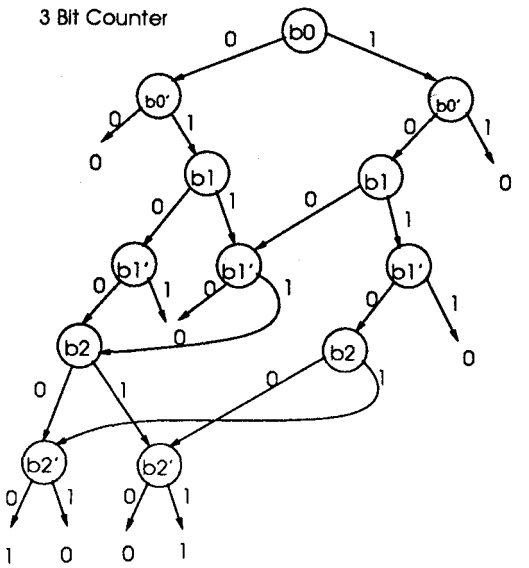

Fig. 1. The OBDD of a transition relation for a 3 bit counter

Let $S(V)$ be the characteristic function (or its OBDD) representing a set of states S. A forward simulation step to compute the next set $S'(V')$ is done as follows [11]:

$$S'(V') = \exists V \, [S(V) \wedge N(V, V')] \tag{1}$$

where $\exists V$ denotes $\exists v_1 \exists v_2 \ldots \exists v_n$. Similarly a backward simulation step is computed

$$S(V) = \exists V' \, [S'(V') \wedge N(V, V')]. \tag{2}$$

The methods to compute the operations in 1 and 2 are well known when $S(V)$ and $N(V, V')$ are represented as OBDDs (see [2]). Notice that existential quantification eliminates the dependency of the result OBDD upon the quantified variables.

3 Partitioning the Transition Relation

The main obstacle for checking bigger designs in SMV is the size of the transition relation. Burch, Clarke and Long [5] showed that it is possible to preserve the transition relation in parts whose sum of sizes is orders of magnitude smaller than the size of a full transition relation. This was based on the observation that the transition relation, is in fact, a conjunction of the set of transition relations for each state variable.

Specifically, the next value of each state variable is a boolean function of the current state.

$$v_i' = f_i(V) \quad i = 1 \ldots n.$$

A transition relation partition, $N_i(V, V')$, is defined by the following equation:

$$N_i(V, V') = v_i' \Leftrightarrow f_i(V).$$

The full transition relation is a conjunction of all partitions:

$$N(V, V') = N_1(V, V') \wedge \cdots \wedge N_n(V, V')$$

and a forward simulation step thus becomes

$$S'(V') = \exists V \, [S(V) \wedge N_1(V, V') \wedge \cdots \wedge N_n(V, V')]. \tag{3}$$

For example, Figure 2 depicts the partitions of the 3 bit counter transition relation from Figure 1.

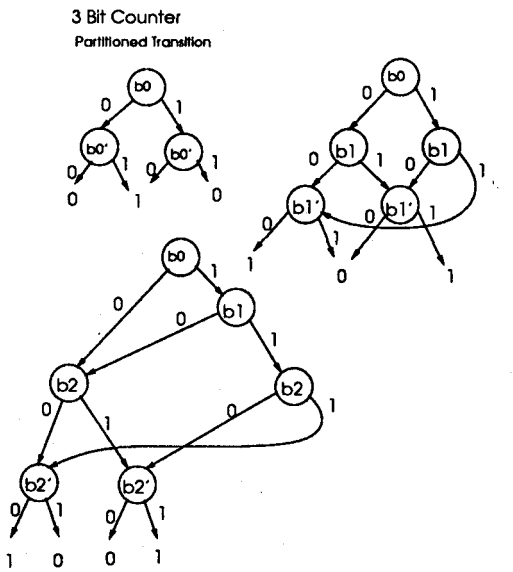

Fig. 2. OBDD representation of the partitioned transition relation for a 3 bit counter

Using 3 we can compute a simulation step. However, care should be taken how the operation is performed. We can, for example, begin by conjuncting all the partitions, leaving $S(V)$ as last, but then we will eventually create the full transition relation which we are trying to avoid. In order to truly exploit partitioning, the simulation step must be performed iteratively without creating a full transition relation in the process.

The technique suggested by Burch et al. [5] is to iteratively conjunct in the partitions, and to quantify out variables as soon as further steps do not depend on them: More explicitly, the user must choose an ordering ρ of $\{1, \ldots, n\}$ which determines the order in which the $N_i(V, V')$ are conjuncted.

Let D_i be the set of variables $N_i(V, V')$ depend on. Also let

$$E_i = D_{\rho(i)} - \bigcup_{k=i+1}^{n} D_{\rho(k)}.$$

A simulation step is done iteratively as follows:

$$\begin{aligned} S_1(V, V') &= \exists E_1 \left[S(V) \wedge N_{\rho(1)}(V, V') \right] \\ S_2(V, V') &= \exists E_2 \left[S_1(V, V') \wedge N_{\rho(2)}(V, V') \right] \\ &\vdots \\ S'(V') &= \exists E_n \left[S_{n-1}(V, V') \wedge N_{\rho(n)}(V, V') \right] \end{aligned} \tag{4}$$

where the expression $\exists E_1$ means that we quantify out all variables in E_1. Therefore, S_1 depends only on $V - E_1$, S_2 depends only on $V - E_1 - E_2$, etc. For example, consider the consider the counter in Figure 2. Assume the partitions are ordered $(3, 2, 1)$ (from b_2' to b_0'). In this case $D_1 = \{b_0, b_1, b_2\}$, $D_2 = \{b_0, b_1\}$, $D_3 = \{b_0\}$, and $E_1 = \{b_2\}$, $E_2 = \{b_1\}$, $E_3 = \{b_0\}$. Thus in the first iteration we quantify out b_2, then b_1, and finally b_0. This happens to be a good order. The method in which to evaluate it is given in the next section.

4 The Conjunction Ordering Criterion

The success of using conjunctive partitioning depends on finding a good order (ρ). Without it, the process of 4 will usually be much slower and consume more space than using the full transition relation as in 1. The difficulties in choosing an order manually were already pointed out in Section 1.

As mentioned before, in symbolic model checking sets of states are stored as OBDDs, and processing is conducted via OBDD operations. The space and time used in the computation depend primarily on the size of the OBDDs generated. Our goal is to minimize the maximal OBDD generated in the process of performing 4. However, predicting the size of the maximal OBDD is difficult, especially since there are going to be many $S(V)$ sets generated during the verification and it will mean considering each one of them. Instead, the following criterion is much simpler to compute.

Criterion: Minimize the maximal number of state variables that participate in any OBDD generated in the process of performing the computation of 4.

The criterion has the following advantages:

1. This criterion does not depend on understanding the semantics of the verified model and the role each variable has in the original circuit. It can be derived from examining the OBDDs of the transition partitions and deriving the set of variables (D_i), each partition N_i depends on.
2. This criterion is independent of any other optimization criteria used in symbolic model checking. Specifically, it is independent of variable ordering which is done to reduce OBDD sizes.
3. This criterion is independent of the set of states $S(V)$, that we wish to perform a simulation step on. The reasons are explained in the next section. Thus we only have to perform ordering once for all forward simulation steps that will be computed during verification. The same applies to backward simulation.

5 The Heuristic Algorithm

An optimal ordering for the criterion discussed in the previous section may be difficult to compute. The ordering problem may very well be intractable. However, it is possible to generate a heuristic ordering that will give good results in practice.

We assume that the set $S(V)$ depends on all n state variables. Our experience has shown that when $S(V)$ depends on only a few state variables then the simulation step will be fast even without ordering, and that space problems only arise when sets on which a simulation step is performed, depend on the full set of variables.

The heuristic algorithm is based on the following observations:

1. Each simulation step begins with an OBDD ($S(V)$ or $S'(V')$) that depends on n state variables and generates an OBDD that depends on n state variables (V' or V). In the process, an OBDD that depends at most on $2n$ variables may be generated (if the ordering is poor).
2. Each partition depends on exactly one variable of V'. In particular N_i depends on v_i'.
3. In a backward simulation step, a varying number of state variables is added in each iteration and exactly one variable of V' is quantified out, since each v_i' appears in exactly one partition.
4. In a forward simulation step a varying number of variables is quantified out in each iteration, and exactly one more variable (the v_i' variable) is added.

The above observations, assist in finding an order for the N_i partitions such that the maximal number of variables over all steps is minimized. From Observations 3 and 4 it follows that the way the number of state variables change in a forward simulation step is different than the way it changes in a backward simulation step. Let us first concentrate on a forward simulation step.

A forward simulation step begins with an OBDD that depends on all the current state variables and iteratively the next state variables (v_i') are added in, while current state variables (v_j) are removed. Thus the order should be such that many v_j as possible will be eliminated in the earlier iterations, before many v_i' are added in.

For example, consider the partition of the counter in Figure 2. The partition of the first bit depends only on b_0 and b_0'. While the partition of the last bit depends on b_2' and

the whole set of b_0, b_1 and b_2. If the partitions are ordered from b'_0 to b'_2 it is not possible to quantify out any state variables until the last iteration and an OBDD that depends on $2n$ variables is generated. However, the partitions are ordered b'_2 to b'_0, it is possible to eliminate one state variable at each iteration and the maximal number of state variables will be $n + 1$. This example shows how crucial the ordering can be since the OBDD size is in the worst case an exponent of the number of variables. The latter ordering is in fact optimal according to the criterion.

The algorithm we implemented was a greedy algorithm that searched at each step for the partition which would result in the maximum number of variables quantified out. This partition chosen had the most number of *unique* variables (not appearing in any other partition that was not previously chosen). The minimization is therefore locally optimal at each iteration but is not necessarily globally optimal. If more than one partition is a possible candidate then a second criterion is applied: From the candidate partitions the algorithm looks for the one that shares most variables with other partitions that have not been conjuncted yet. The rational for this criterion is that it will set the stage for quickly quantifying out many variables in the next steps since all of the partitions that depend on them have been conjuncted.

Algorithm - Obtaining the order ρ for a forward simulation step

1. **Initialize remaining partition set to the whole partition set.**
2. **Loop**
   - **(a) From the remaining partitions find the partition(s) that has the maximal number of unique variables. If there is only one, chose it and go to step (d).**
   - **(b) From the selected partitions in step (a), find the partition(s) that has the maximal number of variables occurring in other remaining partitions. If there is only one, chose it and go to step (d).**
   - **(c) If there is more than one partition satisfying (b), then choose from those one arbitrarily.**
   - **(d) Make the chosen partition the next in the ρ order. Remove it from the remaining partition set.**

   Until the remaining partition set is empty

A backward step begins with an OBDD that depends on all the next state variables, and iteratively the current state variables (v_i) are added in, while the next state variables (v'_j are removed. Thus, the order should be such that the number of current state variables remains small until many of the v'_i are eliminated.

For example, consider the partition of the counter in Figure 2. The partition of the first bit depends only on b_0 and b'_0. While the partition of the last bit depends on b'_2 and the whole set of b_0, b_1 and b_2. Ordering the partitions from b'_2 to b'_0 will generate, in the first iteration, an OBDD that depends on $2n$ variables. On the other hand, ordering from b'_0 to b'_2 will generate OBDDs that depend on at most $n + 1$ variables.

The ordering algorithm for a backward simulation step is similar to the forward one. It too, minimizes locally. A partition is preferred if it does not increase the number of variables participating in the conjunction, or if it contains the minimum number of variables not introduced before. It also uses a secondary criterion which is the same as

for the forward simulation step ordering but for a different reason: From the candidate partitions, the algorithm looks for the one that shares most variables with other partitions that have not been conjuncted yet. The rational here, is that it will set the stage for choosing a partition in the next step, that will introduce only a small number of new variables.

Algorithm - Obtaining the order ρ for a backward simulation step

1. **Initialize remaining partition set to the whole partition set.**
2. **Loop**
   - **(a) From remaining partitions find the partition(s) that has the smallest number of variables not introduced before. If there is only one, chose it and go to step (d).**
   - **(b) From the selected partitions in step (a), find the partition(s) that has the maximal number of variables occurring in other remaining partitions. If there is only one, chose it and go to step (d).**
   - **(c) If there is more than one partition satisfying (b), then choose from those one arbitrarily.**
   - **(d) Make the chosen partition the next in the ρ order. Remove it from the remaining partition set.**

   Until the remaining partition set is empty

Notice that the ordering does not depend on $S(V)$ if we assume that $S(V)$ (or $S'(V')$) depends on the full set of n state variables.

A final remark is that Burch et al. [5] state that it is usually not beneficial to completely partition the transition relation. Rather, it is more useful to work with conjunctions of a small number of partitions. In this case, the algorithms need to be slightly modified to accommodate the fact that the number of v_i' in each partition is not exactly one. The details are left to the reader.

6 Results

The algorithms described in the previous section were implemented as an addition to the SMV system running on an IBM RISC System 6000. They are completely automatic and no extra human intervention is necessary for their operation. They were tested in the verification of various example circuits and have proven to be very beneficial. The extra cost of performing ordering is insignificant both in time and space as compared to other operations that are performed while running SMV.

Our experience has shown that partitioning becomes significantly beneficial only when the verified model becomes relatively big (above 20 state variables). For small models partitioning may unnecessarily result in slower execution times.

In general, the results show that performing a simulation step with partitions is slower. However, computing the transition relation partitioned is much faster and of course requires much less space. Therefore, excluding the case where a full transition relation cannot fit in memory, the payoff of using partitions should be measured by the time it requires to construct the transition relation versus the time to perform a simulation

step, and also by the number of times a simulation step is performed. As the number of simulation steps increases, the advantage of using partitions becomes smaller.

The transition relation can be downsized considerably if the reachable state space is computed and the transition relation is computed assuming only reachable states (this is an SMV option). The process of computing the reachable space is done incrementally with computation of a partial transition relation at each increment. This transition relation is computed to be confined to the "thus far" discovered reachable set. It is used exactly once in a forward simulation step to compute a next set of states for the "thus far" discovered reachable set, and then it is discarded. Using partitions is by far a superior method for performing this incremental process, since only one simulation step is performed for each transition relation construction. The advantage in the transition relation construction is many times greater than the loss in one simulation step.

Therefore, the most effective way to perform model checking, as the results indicate, is to find the reachable state space using a partitioned transition relation, construct a full transition relation, and then do the model checking and counter examples non-partitioned. An exception is the cases where the full transition relation is too big and there is no alternative, but to use partitions.

In summary, partitioning is used simply because it is faster, and for some cases also because without partitioning the execution would reach core memory sizes (about 130MB) where it eventually exits without completing verification. We used partitioning in backward steps only when we could not fit a full transition relation in memory. This happened only in our bigger examples (around 90 state variables). There were also some examples where even with ordering the space required became too large.

Table 1 contains results of some examples we ran. Six runs were executed for each example: Two runs without using partitioning, two runs using partitioning with some arbitrary order and two runs where the partitions were ordered. The runs were done twice for the following reason: Once without proving any SPECs (temporal propositions [9]) - measuring only the time to create the transition relation and another run with some SPECs evaluated. The difference in the run times is the time it took to evaluate SPECs. The transition relation was built incrementally in each run.

The second column in the table is the number of model state variables. The next there numbers are the CPU times for the runs without SPEC evaluation. The next three numbers are the runs with SPEC evaluation.

The next two numbers are the maximum number of variables that participate in any conjunction of a forward step. One number for partitioning without ordering and another for partitioning with ordering. Since we used the incremental option of building the transition relation, we performed partition construction and ordering many times. Thus the two numbers are actually an average of all transition relation constructions. The numbers are an increment to the minimal number of n state variables. For example if there were 86 variables in the model and the average maximum number is 4 then it means that the BDDs that participated in the conjunction depended on at most 90 state variables (and not $2 \times 86 = 172$). Presenting the increment gives a clearer picture of how the algorithm performs since an increment of 1 is always the smallest possible value we can hope to reach.

The last two columns describe the maximum number of variables that participate in

a backward step. The transition relation for backward steps was constructed only once. As with forward chaining the number given is the increment over the number of state variables of the model. Here, 0 is the smallest possible we can hope to reach.

Table 1. Results

Unit Name	Num. of States	Transition CPU (sec)			Transition+SPEC CPU (sec)			Max Frwrd (avg.)		Max Bckwrd	
		Full	Unord.	Ord.	Full	Unord.	Ord.	Unord.	Ord.	Unord.	Ord.
Bus Interface	86	∞	∞	1684	∞	∞	1902	31	4	- N/A -	
Request Que.	41	85	72	48	108	465	169	22	10	28	18
Bus Slave	35	44	20	18	45	27	20	25	6	15	2
Bus Master	53	1062	503	345	1091	1148	436	40	5	28	0

The first example in table 1 is a bus interface unit of a microprocessor. This unit accepts one request for a read or write from the cpu and executes it by initiating and controlling bus cycles. In addition the unit has many variants. It can pipeline two requests. It can also execute a burst cache line read or write, etc. The data and address information are reduced [1] before the process of verification takes place. This example has 86 state variables and it was not possible to verify it without partitioning or without ordering. The maximum number of variables for backward steps is not given since we could not produce it for the unordered mode.

The second example is a unit that interfaces the BIU with the cache. It queues cache requests from the cache and sends them one at a time to the BIU. It can send the requests out of order according to certain control priorities. Again the data and addresses are reduced. This is a typical example of a circuit that can be verified both ways. Notice that although we can generate the transition relation twice as fast when using partitions, the SPECs can be checked much faster if the full transition relation is used.

The third example is a bus slave. The unit verified transfers data to and from an internal FIFO. This is a small example. It runs quickly in all modes but ordered partitions are the fastest. Again we can see that checking SPECs with partitioning is slower. Notice that the maximum number in backward steps was 0. This may happen if one of the state variables remains constant throughout the reachable state space. The partition for this variable will be ordered first in backward steps since it reduces the number of variables that the conjunction depends on without adding new variables. Such a partition is ordered last in forward steps (in fact, an optimization to the algorithm which we do is to treat such constant partitions separately).

The fourth example is a bus master. It is a medium sized example. The transition relation is built significantly faster if the partitions are ordered. Notice how dramatic the difference is in computation of SPECs. If we use partitions without ordering we lose all the advantage we gained by building the transition relation partitioned. Notice that, again the fastest way to compute SPECs is with the full transition relation.

We can see from the results that running in ordered partitions mode is always faster

for creating transition relations. In small examples the difference is insignificant, but in the big examples it is not possible to verify the model in the other modes. Verifying SPECs is faster with a full transition relation (if possible). Ordering for backward steps gave a 2.5 times speedup in one case and a 6.5 times speedup in another. This depends on the number and kind of SPECs we are verifying.

We have also noticed that checking a model that has bugs tends to consume much more memory and CPU time. Once the bugs are fixed the CPU time required for verification decreases dramatically. Our intuitive explanation for this is that incorrect models diverge into areas in the state space that the designer never intended. Thus, the state space tends to grow much larger when bugs are present in the model.

7 Conclusion

Partitioning the transition relation is a method which allows us to increase the size of verified circuits when doing model checking. The success of this method depends on ordering the partitions.

This paper has shown that ordering can be automated by minimization of the number of state variables as an ordering criterion. An algorithm was devised and implemented to order partitions according to this criterion and a few circuits were tested to demonstrate its advantage.

The results have shown that ordered partitions have a clear advantage over working without ordered partitions. In some cases, the improvement resulted in twice as fast execution times, and in others it was not possible to verify without ordering. In all cases that we have encountered, it was always faster to use ordered partitions mode.

Partitioned backward simulation steps were only useful in the cases where a full transition relation could not fit in memory. For those cases ordering has resulted in significant speedups (see Table 1).

Finally, partitioning does not solve the state explosion problem, but rather postpones it. Formal verification is still far from being able to swallow, in one gulp, the designs that are out in the real world. Once the storage space of the transition relation is reduced then other OBDD bottlenecks appear. A recent paper by Hu and Dill has dealt with reducing the OBDD size of invariants [8].

8 Acknowledgements

We would like to thank, Michael Yoeli, Raanan Gewirtzman and Yaron Wolfsthal for the time they spent in reviewing the paper and the invaluable suggestions they had in improving it.

References

1. Ilan Beer, Michael Yoeli, Shoham Ben-David, and Daniel Geist. Methodology and System for Practical Formal Verification of Reactive Hardware. Accepted to CAV 94, 1994.

2. Karl. S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient Implentation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45. ACM/IEEE, 1990.
3. Randal E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35, 1986.
4. Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24:298–318, September 1992.
5. Jerry R. Burch, Edmund M. Clarke, and David E. Long. Symbolic Model Checking with Partitioned Transition Relations. In *International Conference on Very Large Scale Integration*, Edinburg, Scotland, August 1991. IFIP.
6. Edmund M. Clarke, Orna Grumberg, Hiromi Hiraishi, Somesh Jha, David L. Long, Kenneth L. McMillan, and Linda A. Ness. Verification of the Futurebus+ Cache Coherence Protocol. In *Proceedings of the 11th International Conference on Computer Hardware Description Languages*, pages 15–30, 1993.
7. Olivier Coudert, Jean C. Madre, and Christian Berthet. Verifying Temporal Properties of Sequential Machines Without Building their State Diagrams. In R. Kurshan and E. M. Clarke, editors, *Workshop on Computer Aided Verification, DIMACS*, pages 75–84. American Mathematical Society, Providence, RI, 1990.
8. Alan J. Hu and David L. Dill. Efficient Verification with BDDs using Implicitly Conjoined Invariants. In *Proceedings of the Conference on Computer Aided Verification (CAV 93)*, 1993.
9. K. L. McMillan. *The SMV System DRAFT*. Carnegie Mellon University, Pittsburgh, PA, 1992.
10. K. L. McMillan and J. Schwalbe. Formal verification of the Encore Gigamax cache consistency protocol. In *Proceedings of the 1991 International Symposium on Shared Memory Multiprocessors*, April 1991.
11. Kenneth L. McMillan. *Symbolc Model Checking*. PhD thesis, Carnegie Mellon University, May 1992.
12. H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines usin BDD's. In *IEEE International Conference on CAD*, pages 130–133, 1990.