On partitioning and model checking

Subramanian Iyer[1], Debashis Sahoo[2], Jawahar Jain[3], and E. Allen Emerson[1]

[1] University of Texas at Austin, Austin, TX 78712, USA
[2] Stanford University, Stanford CA 94305, USA
[3] Fujitsu Laboratoies of America, Sunnyvale CA 94085, USA

Abstract. State space partitioning-based approaches have been proposed in the literature to address the *state-space explosion* problem in model checking. These approaches, whether sequential or distributed, perform a large amount of work in the form of inter-partition (*cross-over*) image computations, which can be expensive. We present a model checking algorithm that aggregates such cross-over images by localising computation to individual partitions. Experimentally, this algorithm consistently outperforms extant approaches in terms of time taken, as well as *cross-over* image computation cost.

1 Introduction

Model checking is performed by means of successive backward image computations. Image computation becomes difficult as the data structures representing the state sets grow larger. Large state sets are a direct consequence of the *state-space explosion* problem. Model checking is unable to handle data structures when their size exceeds (roughly by an order of magnitude) what can be reasonably handled in main memory. This frequently happens when handling large designs.

From a practical standpoint, a monolithic approach to model checking fails due to this excessive memory requirement for state set representation. Partitioned symbolic data structures have been proposed in the literature to handle this *memory explosion problem.* Partitioning of the state space is found to balance the trade-off between compactness and canonicity of symbolic BDD representations. In such a framework, each partition of the state space may obey a different variable variable order.

In a partitioned approach, the state space S is partitioned into subspaces $S_1, S_2, \dots S_n$. This induces a disjunctive partitioning on the transition relation T into the parts T_{ij} which represents the set of transitions from states in a source partition i to states in the destination partition j. The size of each such transition relation can be further reduced by an implicitly conjoined implementation.

Each partition can be thought of as being the *owner* of a set of states. Transitions from each partition naturally comprise of two components - ones that are wholly local to individual partitions, and ones that span multiple partitions. Correspondingly, the computed image X comprises of a local component X_l and a *cross-over* component X_c. The states corresponding to X_l may be computed locally in each partition. On the other hand, the states in X_c arise out of transitions that originate at a state in one partition and terminate at a state in another, thus, "crossing over" into the destination partition.

Computing cross-over component of the image is often significantly more expensive than the local component for various reasons. Firstly, the cross-over component involves transitions into a potentially larger subspace. Secondly, this incurs the overhead of transporting these states to the partition that "owns" them. Thirdly, the source and destination partitions likely obey different variable orders, and therefore the state set needs to be reordered, which is a known difficult problem as representation sizes become large.

The naive way of combining partitioning with the classical model checking algorithm of [1] performs repeated exact images. Each such image computation requires a quadratic number of cross-over computations.

Notice that the set obtained by performing operation EX_l is a subset of the actual image, and in this sense, can be thought of as an under-approximation to EX. This allows for an efficient analysis of reachability [4] and a subset of CTL [2] by replacing a sequence of EX operations by a sequence of the less expensive EX_l operations, interspersed with an occasional EX_c to maintain completeness.

The problem is trickier with greatest fix-points, e.g. the EG operator. The EG operator and its dual AF are important in falsifying and verifying liveness properties. In this case, the final result is the conjunction of successively smaller supersets of the result. If operation EX_c is ignored in pre-image computations, then the result is a subset of the actual pre-image EX, and this means that some states get pruned early in the greatest fixpoint computation for computing the set EG. Since the convergence is on a sequence which is monotonically decreasing, these states pruned early may be lost for ever. Consequently, EX cannot be replaced by EX_l as it compromises on soundness. An important question arises as to how to compute greatest fix-points in the partitioned framework without having to perform repeated cross-over image computations.

In this paper, we propose an alternative *piece-wise* algorithm for model checking CTL formulae in a partitioned setting that addresses these concerns. Our approach exploits the separability of the local and cross-over components of image computation. It performs a number of image computations locally within each partition, and synchronises occasionally by doing cross-over image computations only when a fixpoint is reached locally in each partition.

In section 2, we recall the notions of state space partitioning and the definition of model checking. We present the partitioned version of the classical model checking algorithm in section 3. Section 4 describes our algorithm designed to localise computation by postponing cross-over image computations. Details of implementation and experimental results follow in Section 6.

2 Preliminaries

In this section, we briefly look at some background related to state space partitioning and image computation, leading up to a description of the classical model checking algorithm in a partitioned framework.

2.1 State Space Partitioning

The idea of partitioning was used to discuss a function representation scheme called partitioned-ROBDDs in [3] which was extensively developed in [5].
Definition. **[5]** Given a Boolean function $f : B^n \rightarrow B$, defined over n inputs $X_n = \{x_1, \ldots, x_n\}$, the partitioned-ROBDD (henceforth, POBDD) representation χ_f of f is a set of k function pairs, $\chi_f = \{(w_1, f_1), \ldots, (w_k, f_k)\}$ where, $w_i\colon B^n \rightarrow B$ and $f_i\colon B^n \rightarrow B$, are also defined over X_n and satisfy the following conditions:
1. w_i and f_i are ROBDDs respecting the variable ordering π_i, for $1 \leq i \leq k$.
2. $w_1 \vee w_2 \vee \ldots \vee w_k = 1$
3. $w_i \wedge w_j = 0$, for $i \neq j$
4. $f_i = w_i \wedge f$, for $1 \leq i \leq k$ The set $\{w_1, \ldots, w_k\}$ is denoted by W. Each w_i is called a *window function* and represents a *partition* of the Boolean space over which f is defined. Each partition is represented separately as an ROBDDs and can have a different variable order. Most ROBDD based algorithms can be adapted easily for POBDDs.

Partitioned-ROBDDs are canonical and various Boolean operations can be efficiently performed on them just like ROBDDs. In addition, they can be exponentially more compact than ROBDDs for certain classes of functions. The practical utility of this representation is also demonstrated by constructing ROBDDs for the outputs of combinational circuits [5].

2.2 Model Checking

We omit the syntax of CTL as it is widely known and readily available in the literature. We shall only note that it is possible to express any CTL formula in terms of the Boolean connectives of propositional logic and the existential temporal operators EX, EU and EG. Such a representation is called the *existential normal form.*

Model Checking is usually performed in two stages: In the first stage, the finite state machine is reduced with respect to the formula being model checked and then the reachable states are computed. The second stage involves computing the set of states falsifying the given formula. The reachable states computed earlier are used as a *care set* in this step.

Since there exist computational procedures for efficiently performing Boolean operations on symbolic BDD data structures, including POBDDs, model checking of CTL formulas primarily is concerned with the symbolic application of the temporal operators. EXq is a backward image and uses the same machinery as image computation during reachability, with the adjustment for the direction.

$EpUq$ (resp. EGp) has been traditionally represented as the least (resp. greatest) fixpoint of the operator $\tau(Z) = q \vee (p \wedge EXZ)$ (resp. $\tau(Z) = p \wedge EXZ$).

3 Classical Model Checking with Partitioning

Since backward image computation is the basic unit operation in performing model checking, we first examine POBDD based image computation.

Given a set of states, $R(s)$, that the system can reach, the set of next states, $N(s')$, is calculated using the equation $N(s') = \exists_{s,i}[T(s, s', i) \wedge R(s)]$. This calculation is also known as *image computation*. State space partitioning into n disjoint parts induces a partitioning of the transition relation T into n^2 parts T_{jk} consisting of transitions from a state in partition j to a state in partition k. We can derive T_{jk} by conjoining T with the respective window functions as $T_{jk}(s, s', i) = w_j(s)w_k(s')T(s, s', i)$. Thus we can express the transition relation $T(s, s', i) = \bigvee_j \bigvee_k T_{jk}(s, s', i)$ as an induced disjunctive partitioning. Accordingly, the image computation can be performed separately on each of these as illustrated in Figure 1.

```
**ComputeImage($R$)** {
  foreach (partition $k$)
    foreach (partition $j$)
      $Img^{jk}(s') = \exists_{s,i}[T_{jk}(s, s', i) \wedge R_j(s)]$
      reorder BDD $Img^{jk}(s')$ from partition order $j$ to order $k$
    end for
    $N_k(s') = \bigvee_j Img^{jk}(s')$
  end for
  output $N$
}
```

Fig. 1. Image Computation

Here the set R_j is the set of initial states in partition j, and the set N_k represents the set of next states in partition k which are computed by application of the transition relation $T_{jk}(s, s', i)$.

To compute the image, the n^2 computations $T_{jk}(R_j)$ need to be performed, followed by n disjunctions as shown. Recall that when using a partitioned-BDD to represent the set of states, each partition is maintained separately in memory, under differing variable orders. It is therefore natural that the image of states in partition j under the trasitions leading to each partition k, i.e. the computation $T_{jk} \wedge (R_j)$, is performed in partition j. Each partition j thus computes states that potentially belong to every other partition. Subsequently the disjunction to obtain the pre-image lying with partition k, i.e. the computation of $\bigvee_j Img^{jk}$, is performed by partition k. As a consequence, the set Img^{jk} needs to be transferred from partition j to partition k, when j and k differ.

We call these $n^2 - n$ computations as *cross-over image* computations, in the sense that the source and destination partitions are different. It must be emphasised that *Cross-over image computation is expensive* for various reasons: First, a quadratic number of image computations need to be performed as above and the BDDs need to be accessed from every partition. In the case of large designs, where the BDDs of even a single partition can run into millions of nodes, this usually means accessing stored partitions from secondary memory. Then, the BDD variable order of the computed imageset must be changed from the order of the source partition to that of each of its target partitions, before the new states can be added to the reached set in the target. Reordering large BDDs can be very expensive. There may also be other overhead, for eg., in the case of a parallel implementation there is the overhead of physically transmitting a large number of these BDDs over the network.

```
computeEU($p, q$) {
  $S := q$ and $S.old := \phi$
  repeat
    $S.old := S$
    $temp :=$ computeEX($S$)
    forall (partitions $j$)
      $S_j := q_j \vee (p_j \wedge temp_j)$
    end for
  until($S = S.old$)
  output $S$
}
```

a) Least fixpoint, $E(pUq)$

```
computeEG($p$) {
  $S := p$
  repeat
    $S.old := S$
    $temp :=$ computeEX($S$)
    forall (partitions $j$)
      $S_j := p_j \wedge temp_j$
    end for
  until($S = S.old$)
  output $S$
}
```

b) Greatest fixpoint, EGp

Fig. 2. Classical Model Checking of Fixpoints in presence of Partitioning

The classical fixpoint algorithms for $E(pUq)$ and EGp as modified to use the POBDD data structure and image computation are illustrated in Fig. 2. Notice that these rely on the partitioned image computation and therefore perform one set of cross-over images in each iteration.

In the next section, we present a model checking algorithm that localises computation to individual partitions by postponing these cross-over image computations.

4 Partitioned Model Checking

In this section we present a new partitioned model checking algorithm which works by postponing cross-over image computations. When the design is defective and is falsified, this algorithm discovers bugs faster, by virtue of computations being localised to individual partitions. Even when the design is correct and is verified, this algorithm converges after fewer cross-over image computations.

We show that in the worst case, this algorithm has at most as many cross-over image computations as the partitioned version of the classical algorithm, presented in the previous section.

Model checking of boolean connectives is well-known for the partitioned approach, so we will only describe the image and fixpoint computations. It must however be mentioned that all boolean operations - conjunction, disjunction as well as negation - are local to individual partitions and involve no interaction between them. Also, it suffices to consider the existential operators EX, EG and EU.

4.1 Image Computation

The main computation in the partitioned form of the classical model checking algorithm is image computation. As noted in the previous section, the computation of EXp from p comprises of n^2 image computations, reorderings and state set transfers between partitions and this can get expensive. Even though our focus is on trying to avoid computing the entire image *at every step*, it is still essential to perform teh full image computation in two cases – firstly, for the occasional cross-over images, and secondly, when the property is expressed in terms of the EX or AX operators. In this section, we look at some of the issues in computing the image.

We find that performing the cross-over images one partition at a time is memory intensive and often the intermediate BDDs get very large for many examples. Therefore, we advocate performing these cross-over image computations from each partition into many partitions at a time.[4]

In order to perform cross-over images efficiently, we maintain a *transfer manager* M. Given the set p, in order to compute EXp, each partition i computes the image $T_{ii}(p_i)$ which it keeps locally and the set of unowned states $U_i = T_{i\bar{i}}(p_i)$ which is communicated to the manager M. M uses the window functions w_j to calculate the sets $S_j = \bigvee_{i \neq j} U_i * w_j$ and then transmits the states S_j to partition j. Thus EXp is computed by doing $2n$ image computations and $2n$ transfers between partitions, although the number of reorderings remains n^2.

It should be mentioned here that in a multiprocessor environment, such a manager can become a bottleneck, and should perhaps be dispensed with. But the point is that only a constant number of image computations be performed in each partition, rather than a number linear in the number of partitions.

We call the fraction $T_{ii}(p_i)$ which is computed locally as the local image EX_l and the rest as the cross-over image EX_c.

4.2 Fixpoint computations

The main idea for model checking fixpoints is that the computations can be significantly localised to individual partitions by postponing the cross-over image

[4] Here, it must be noted that we address the case of verification using uniprocessor systems. The partitioned approach easily extends to distributed and parallel computing environments and our improvements are expected to scale accordingly.

computations, which are then aggregated and performed infrequently. Accordingly, we define the fixpoint operators in terms of two operations – local image computations and cross-over image computations, rather than the classical definition in terms of just the image computation operation, EX.

The algorithms for computing $E(pUq)$ and EGp are shown in Figure 3. The key idea is to create an under-approximation (resp. over-approximation) to EXp, which can be wholly calculated locally within individual partitions, so that the least (resp. greatest) fixpoint computation can be localised.

```
computeEU($p, q$) {
  $S := q$
  $S.old := \phi$
  repeat
    $S.old := S$
    forall (partitions $j$)
      repeat
        $S_j.old := S_j$
        $S_j := S_j \vee (p_j \wedge EX_l(S_j, j))$
      until($S_j = S_j.old$)
    end for
    $S := S \vee (p \wedge EX_c(S))$
  until($S = S.old$)
  output $S$
}
        a) Least fixpoint, $E(pUq)$
```

```
computeEG($p$) {
  $S := p$
  $Border := p \wedge EX_c(S)$
  repeat
    $S.old := S$
    forall (partitions $j$)
      repeat
        $S_j.old := S_j$
        $S_j := p_j \wedge (EX_l(S_j, j) \vee Border_j)$
      until($S_j == S_j.old$)
    end for
    $Border := p \wedge EX_c(S)$
  until($S == S.old$)
  output $S$
}
        b) Greatest fixpoint, $EGp$
```

Fig. 3. Fixpoint Computations localised by postponement of cross-over image computation

We call each iteration of the outermost repeat-until loop in algorithm 2 and algorithm 3 as a *phase*. We will show that algorithm 3 terminates with the correct result and that the number of its phases is at most the number of phases in algorithm 2. Since each such phase has precisely one cross-over image computation, we have that the number of cross-over images computed is no more than that for the algorithm of the previous section.

In the rest of this section, we prove the correctness of the model checking algorithm of Fig. 3.

Theorem 1. *a)[2] The procedure computeEU of Fig 3a, given the set of states corresponding to formulas p and q as inputs, terminates with the output S being precisely the set of states that model the formula $E(pUq)$.*
b) The number of its phases does not exceed the number of phases for algorithm 2a.

Proof: Let the set of states S at the end of the i^{th} phase be called S^i. The termination is guaranteed because the sequence of sets S^i is strictly monotonic increasing.

We first show the soundness of algorithm 3a, i.e., at all times $S \models E(pUq)$. We show this by induction on the sets S^k. This clearly holds for any state in S^0, since every state in S^0 satisfies q and therefore $E(pUq)$. Assume that $S^i \models E(pUq)$. Consider a state $s \in S^{i+1} - S^i$. Then, by construction of S^{i+1} from S^i, we have $s \models p$. Either s is added in the local image computation EX_l for some partition j or in the cross-over image computations EX_c. In either case, $s \models p$. It remains to show that s is the predecessor of a state that models $E(pUq)$. In the first case, such a state is int he same partition as s and in the second case, such a state exists in partition k such that s was added in the cross-over image computation from k to j. Thus in either case, s models $EX(E(pUq))$. Consequently, Alg 3a is sound.

Next, we show completeness, i.e., that every state of $E(pUq)$ is indeed in set S. For every state $s \models E(pUq)$, there exists a sequence of states $s_0, s_1, \ldots, s_k$ that has the smallest length $k \geq 0$ such that $s_0 = s$, $s_k \models q$, $\forall i < k : s_i \models p$ and $\forall i < k : s_i \in EX(s_{i+1})$. This sequence of states is called a *witness* for the inclusion of s in $E(pUq)$, and k is its *length.* Let T^k be the set of states whose inclusion in $E(pUq)$ is witnessed by a path of length at most k. We prove by induction on k that $T^k \subseteq S$. In the base case, this trivially holds because $T^0 = q = S^0 \subseteq S$. Now, assume that $T^i \subseteq S$. For any state $s \in T^{i+1}$ consider the sequence of states $s_0 = s, s_1, \ldots, s_{i+1}$ that witnesses its inclusion in $E(pUq)$. The sequence $s_1, \ldots, s_{i+1}$ is a witness for s_1, therefore $s_1 \in T^i \subseteq S$. In particular, there exists a smallest j so that $s_1 \in S^j$. We know that $s \models p$ and $s \in EX(s_1) \subseteq EX(S^j)$. From the definition of S^j and Algorithm 3a, we have that $s \in S^{j+1}$, whereby $T^{i+1} \subseteq S^{j+1} \subseteq S$. By induction, this gives us $E(pUq) \subseteq S$.

This proves that algorithm 3a terminates with the set $S = E(pUq)$. Notice that the set of states at the end of the k^{th} phase of algorithm 2a is precisely T^i. As above, $\forall i, T^{i+1} \subseteq S^{j+1} \subseteq S^{i+1}$. Hence algorithm 3a has at most as many phases as algorithm 2a. ∎

Before proving an analogous result for the greatest fixpoint operator EGp, we briefly motivate its construction. As EX_lp is a subset of EXp, the result of localising the computation by performing repeated EX_l operations yields an underapproximation at every step. Since the greatest fixpoint operator converges by a sequence of monotonically decreasing sets, underapproximation leads to some states being pruned too early and being lost for ever. States that may be incorrectly pruned early in the computation of EG comprises of states, each of which lies in a different partition from its predeccesor, and can therefore be discovered only by performing the operation EX_c, which is the expensive component of image computation.

Algorithm 3b compensates for this by maintaining a set *Border*, which is the set of all states which have a successor in a different partition than themselves. This is, clearly an overapproximation to EX_c in each partition. This superset of EX_c is used to calculate a superset of EX at every image. This *Perimeter* is updated only once in each phase, when each partition has reached a fixpoint with respect to local images EX_l. These overapproximations are monotonically decreasing, and so the computed set eventually converges to the desired set EG.

We now prove the following theorem.

Theorem 2. *a) The procedure computeEG of Fig 3b, given the set of states corresponding to formula p as input , terminates with the output S being precisely the set of states that model the formula EGp.*
b) The number of its phases does not exceed the number of phases for algorithm 2b.

Proof: Again, let the set of states S at the end of the i^{th} phase be called S^i. The termination is guaranteed because the sequence of sets S^i is strictly monotonic decreasing.

We first show the soundness of algorithm 3b, i.e., the algorithm only deletes states which do not satisfy EGp. Note that a state can be deleted only in the two circumstances. The first is if it does not satisfy p and is deleted in the very beginning. We can therefore assume that all states under consideration satisfy p. The second way a state may be deleted is during some phase, when it is not a predecessor to any state in its own partition, and it is not on the Border, i.e., it has been determined previously that this state is not a predecessor to any state in another partition. Thus all successors to such a state satisfy $\neg p$, and therefore any deleted state is not in EGp.

Next we show completeness, i.e., the algorithm deletes all states that do not satisfy EGp. Consider a state $s \not\models EGp$. Then there exists a sequence of states $s_0, s_1, \ldots, s_k$, which is cycle-free that has the greatest length $k \geq 0$ such that $s_0 = s$, $s_k \models \neg p$, $\forall i < k : s_i \models p$ and $\forall i < k : s_i \in EX(s_{i+1})$. This sequence of states is called a *witness* for the exclusion of s from EGp, and k is its *length*. Now, let T^k be the set of states whose exclusion from EGp is witnessed by a longest cycle-free path of length at most k. We prove by induction on k that $T^k \cap S^k = \phi$. In the base case, this trivially holds because $T^0 = \neg q$ and $S^0 = q$. Now, assume that $T^i \cap S^i = \phi$. For any state $s \in T^{i+1}$ consider the sequence of states $s_0 = s, s_1, \ldots, s_{i+1}$ that witnesses its exclusion from EGp. The sequence $s_1, \ldots, s_{i+1}$ is a witness for s_1, therefore $s_1 \in T^i$, and therefore $s_1 \notin S^i$. In particular, there exists a smallest j so that s_1 was deleted in the j^{th} stage of the algorithm. Two cases arise, either both s_0 and s_1 are in the same partition or they are in different partitions. If they are in the same partition, then s_0 is deleted in the j^{th} stage also when a fixpoint is computed locally in that partition. If they are in different partitions, then s_0 is in the border set for its partition, and is deleted from this border set at the end of the j^{th} stage because its last successor s_1 is deleted and no other successors can exist because this is the longest witness. Therefore s is deleted in the $j + 1^{th}$ stage, as required to be proved.

This proves that algorithm 3b terminates with the set EGp. Notice that the set of states T^i is precisely the set of states deleted in phase i of algorithm 2b. As above, states in T_i have all been deleted by the end of i phases of the algorithm. Hence algorithm 3b has at most as many phases as algorithm 2b. ∎

It is noteworthy that, in the worst case, the algorithms of Fig 3 require at most as many phases as that of Fig 2. However, in practice, this algorithm outperforms Alg 2, because the fixpoints localised to individual partitions often discover or prune, as the case may be, many more states than when performing just one image computation in each phase, and thus the postponement of cross-over images is found to afford a significant benefit in overall faster convergence of the algorithm, often reducing the number of phases.

In the next section, we describe our experimental setup and the results of the same.

5 Experimental Results

Our sequential implementation of this algorithm shows us the following results:

1. Crossover image computation is a bottleneck: On a sequential machine, there are examples where performing crossover images one partition at a time runs out of memory, whereas doing the image all at a time, and then restricting to individual window functions, finishes. However, performing all cross-overs at a time is expensive because it does not make effective use of partitioning. Also, doing them all at a time is unfavorable to effective parallelization.
2. The time taken by cross-over images as a percentage of total time is reduced.
3. Experimentally, the proposed algorithm converges faster, both in terms of total time, as well as in terms of number of expensive cross-over image computations that are performed.
4. The proposed algorithm is more easily parallelizable than the existing model cehcking algorithms.

6 Conclusion

We have presented a model checking algorithm in the presence of state space partitioning, that aggregates and postpones cross-over image computations, allowing for significant localisation of image computations. This is also found in practice to reduce the number of iterations in fixpoint computations.

Our experiments have been conducted on uniprocessor machines, but this algorithm can be easily parallelized and we believe its benefits would scale to an implementation in a multi-processor environment.

In the worst case, this method would be identical to the naive one, with strict alternation between localised (EX_l) and cross-over (EX_c) image operations in every fixpoint calculation. However, this is extremely unlikely because it corresponds to a case where every "path" corresponding to a formula comprises of states **each** of which lies in a different partition than its predecessor. This does not happen in practice when partitioning is done properly.

We believe this algorithm can be generalised to more expressive logics liek the alternation free μ-calculus, as well as the full μ-calculus with slight modifications.

References

1. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.
2. S. Iyer, D. Sahoo, C. Stangier, A. Narayan, and J. Jain. Improved symbolic Verification Using Partitioning Techniques. In *Proc. of CHARME 2003*, volume 2860 of *Lecture Notes in Computer Science*, 2003.
3. J. Jain. On analysis of boolean functions. *Ph.D Dissertation, Dept. of Electrical and Computer Engineering, The University of Texas at Austin*, 1993.
4. A. Narayan, A. Isles, J. Jain, R. Brayton, and A. Sangiovanni-Vincentelli. Reachability Analysis Using Partitioned-ROBDDs. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 388–393, 1997.
5. A. Narayan, J. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli. Partitioned-ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 547–554, 1996.