Heuristic Symmetry Reduction for Invariant Verification

William Hung   Adnan Aziz
Electrical and Computer Engineering
The University of Texas
Austin TX

Ken McMillan
Cadence Berkeley Labs
Cadence
Berkeley CA

Abstract

We describe techniques that use symmetry to perform efficient invariant checking. We start by developing the theory needed to exploit symmetry for designs specified at the gate level. This is followed by a proof of the inadequacy of BDD based methods for highly symmetric designs; this motivates the use of explicit state enumeration. Exact symmetry reduction has been conjectured to be computationally intractable; we propose fast heuristic reduction procedures. Experiments with these routines demonstrate their effectiveness in practice; we also compare running times with a BDD based tool.

1 Introduction

A common problem in formal verification of hardware designs is to determine if every state reachable from the reset state lies in a set of "good states"; this is sometimes referred to as *invariant checking*. This can be achieved either by explicit state enumeration or by symbolic methods.

Invariant checking can be performed by reachability analysis. Starting from the set of reset states, we traverse the state transition graph and check whether all reachable states belong to the invariant. This approach leads to the state explosion problem — for a design with n latches, there may be as many as 2^n reachable states. In practice, many designs are well structured, and this can be exploited to devise heuristic procedures which perform well on specific classes of designs.

A large set of designs incorporate symmetry. For certain classes of properties, verification of the invariant at a particular state ensures its correctness at all symmetrically equivalent states. Various researchers have exploited this fact to reduce the complexity of verification [1, 2, 3].

In this paper, we prove the existence of an exponential lower bound on the size of BDDs needed to represent the reached state sets of completely symmetric systems; this provides theoretical justification for using explicit state enumeration. We also developed heuristic symmetry reduction procedures (the need for such procedures was stressed by Clarke et al. [1]). Another (minor) contribution is the interpretation of the theory of symmetries for designs specified as netlists. We have implemented the procedures described in this paper, and experimented with examples incorporating various degrees of symmetry. We also compared explicit and symbolic (BDD based) verification.

2 Formal Models for Hardware

We use two formalisms for expressing designs, namely finite state machines and netlists. Hopcroft and Ullman described the theoretical aspects of FSMs in [4]. A netlist is a representation of a design at the *structural level*. It is closer to the actual implementation of the design than FSMs, which can be viewed as behavioral level descriptions of the design [5]. Precise descriptions of finite state machines and netlists are given in [6].

3 Lower Bounding the Complexity of BDD Based Invariant Checking

We show that there is no variable ordering under which a polynomial sized Reduced Ordered Binary Decision Diagram can be built for the characteristic function of the set of permutations; this implies that a straightforward implementation of a BDD based invariant check will fail for designs with highly symmetric reached state sets.

The Characteristic Function for Permutations

The characteristic function of the set of permutations on $\{0, 1, \ldots, N-1\}$ is the Boolean function $f_N : 2^{N \cdot \log N} \rightarrow 2$ defined as follows: (here $n = \log N$ for convenience)

$$f_N(\alpha_{00}, \alpha_{01}, \ldots, \alpha_{0(n-1)}, \alpha_{10}, \alpha_{11}, \ldots, \alpha_{(N-1)(n-1)}) = 1$$

if and only if $(\mathbf{a_0}, \mathbf{a_1}, \ldots, \mathbf{a_{N-1}})$ (where a_i is the integer derived by viewing $\alpha_{i0}\alpha_{i1} \cdots \alpha_{i(n-1)}$) as the binary representation of an integer) is a permutation of $\{0, 1, \ldots, N-1\}$, that is for each $p \in \{0, 1, \ldots, N-1\}$, there is a k so that $p = \mathbf{a_k}$.

Lower Bounding the BDD Size

Theorem 3.1 *The BDD for f_N has at least $2^{N/2}$ nodes, under any variable ordering.*

The proof is available in [6].

This suggests that BDDs are not a good data structure for reachability analysis for highly symmetric systems. An example for BDD explosion is a multiprocessor network, where each processor has multiple memory units, shown in Figure 1. For many randomized routing protocols, the set of states this network can get into will be an arbitrary permutation of the values in memory.

4 Exploiting Symmetry

Explicit state enumeration suffers from the state explosion problem. However, much of the state space search can be "pruned" for symmetric systems.

4.1 Symmetries

We develop the basic terminology for symmetries. Armstrong [7] is a good general reference to symmetries.

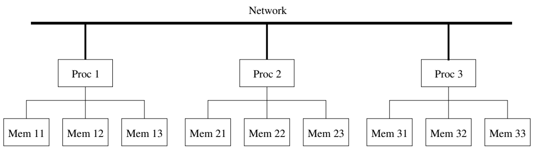

Figure 1: Multiprocessor network with groups of symmetry.

Let J_p be the set of integers $\{1, 2, \ldots, p\}$. A bijective mapping $\sigma : J_p \rightarrow J_p$ is called a *permutation* of the integers from 1 to p. The set of all permutations on J_p is denoted as S_p. For any $\sigma_1, \sigma_2 \in S_p$, the composition $\sigma_1 \circ \sigma_2$ is a mapping $J_p \rightarrow J_p$ such that $i \mapsto \sigma_1(\sigma_2(i))$. A subset $\Pi \subseteq S_p$ is said to be *closed under inversion* if $(\forall \sigma \in \Pi)\ \sigma^{-1} \in \Pi$, *closed under composition* if $(\forall \sigma_1, \sigma_2 \in \Pi)\ \sigma_1 \circ \sigma_2 \in \Pi$. The set Π is referred as a *subgroup* of S_p if it is closed under both inversion and composition.

Given a subset $T = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of S_p, define the subgroup *generated* by T to be the smallest subgroup of S_p containing T. We will denote the closure of T by $[\![T]\!]$. The elements of T will be referred to as the *generators* of $[\![T]\!]$.

Given an n-dimensional vector $\vec{v} = (v_1, v_2, \ldots, v_p)$, and a permutation $\sigma \in S_p$ the vector $\sigma(\vec{v}) = (v_{\sigma(1)}, v_{\sigma(2)}, \ldots, v_{\sigma(p)})$ will sometimes be referred to as the *action* of σ on $\vec{v}$. Given a set of vectors $V = \{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_l\}$, and a subset $T = \{\sigma_1, \sigma_2, \ldots, \sigma_k\} \subset S_p$, we have $T \cdot V = \{\vec{v} | \vec{v} = \sigma_i(\vec{v'}) \text{ for } \sigma_i \in T \text{ and } \vec{v'} \in V\}$. We will refer to $T \cdot V$ as the *action* of T on V.

A subgroup Π of S_p gives rise to a natural equivalence relation $\mathcal{E}_\Pi$ on bit-strings of length p, i.e., on elements of $\{0,1\}^p$. The equivalence is $(\alpha, \beta) \in \mathcal{E}_\Pi$ exactly when $(\exists \sigma \in \Pi)(\beta = \sigma(\alpha))$.

The equivalence classes of $\mathcal{E}_\Pi$ are referred to as its *orbits*. The set $\{0,1\}^p$ can be totally ordered by the *lexicographic* order relation $\prec_{lex}$. The *canonical representative* $\hat{\alpha}_\Pi$ of an orbit is the largest element of the orbit containing $\alpha \in \{0,1\}^p$ under the relation $\prec_{lex}$. The subset $A \subset \{0,1\}^p$ is *invariant* under Π when $\Pi \cdot A = A$.

The following lemma is easily proved:

Lemma 4.1 *When Π is generated from the set $\Pi = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, the subset A is invariant under $[\![\Pi]\!]$ iff A is invariant under $\{\sigma_i\}$ for every $\sigma_i \in [\![\Pi]\!]$.*

The power of this lemma is that we can check if a given set of states is invariant under $[\![\Pi]\!]$ simply by computing the action of each $\sigma \in \Pi$ on A; the latter can be done using BDDs by reordering the present state variables.

4.2 State Space Reduction

Let η be a netlist with n latches denoted by the vector $\vec{x} = x_1, x_2, \ldots, x_n$ and k inputs denoted by the vector $\vec{u} = u_1, u_2, \ldots, u_k$. Let the next state functions of the latches be $\vec{F}_\eta = (f_1, f_2, \ldots, f_n)$. Given a state $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in 2^n$ and an input $\vec{\iota} = (\iota_1, \iota_2, \ldots, \iota_k) \in 2^k$, the next state of the design will be $\vec{F}_\eta(\vec{\alpha}, \vec{\iota})$.

Let $G \subset S_\eta = 2^n$ be an invariant to be checked. Let Q be a subset of $S_n \times S_k$, i.e., a set of ordered pairs where the first component is a permutation from S_n and the second from S_k. Let T be the projection of Q to the first component, i.e., the set of all permutations σ in S_n so that there exists some τ in S_k such that $(\sigma, \tau) \in Q$; similarly, let R be the projection of Q to the second component.

Suppose G is invariant under $[\![T]\!]$. Furthermore, suppose every (σ, τ) in Q satisfies the following:

$$\sigma(\vec{F}(\vec{x}, \vec{u})) \quad = \quad \vec{F}(\sigma(\vec{x}), \tau(\vec{u})) \qquad (1)$$

Then the following lemma holds:

Lemma 4.2 Let s be a state and u an input; take $t = \vec{F}_\eta(s, u)$. Then for every s' in $[\![T]\!] \cdot \{s\}$ there is an input u' so that $t' = \vec{F}_\eta(s', u')$ is in $[\![T]\!] \cdot \{t\}$.

The proof is available in [6].

Since G is invariant under $[\![T]\!]$ it has the property that if it contains a state s, it contains every state in the orbit of s under $[\![T]\!]$. Coupling this fact with Lemma 4.2, we can immediately infer the following:

Corollary 4.3 Suppose G is invariant under $[\![T]\!]$; then a state s can reach a state outside G if and only if its canonical representative $\hat{s}$ can reach a state outside G.

4.3 Symmetry Reduction

The result of Corollary 4.3 suggests the following strategy for reducing the complexity of invariant checking: (1) Have the designer suggest permutations for $Q \subset S_n \times S_k$. (2) Check that the permutations in Π satisfy the condition of Equation 1. (3) Traverse the STG of the design, while "canonicalizing" states, i.e., mapping states to the canonical representatives elements of their orbits.

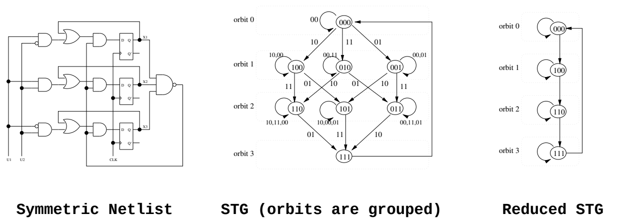

Figure 2: A symmetric netlist, its STG, and reduced STG. The orbits are the circled sets of states.

The advantage of this approach is that it is not necessary to store all states — only canonical representatives are stored. Additionally, many states may be avoided all together, since they may not be reached by passing directly through canonical representatives. An example of this approach is shown in Figure 2.

The check that the permutations satisfy Equation 1 is easily implemented using BDDs: it amounts to re-ordering [8], and the resultant BDDs are exactly the same size as the original BDDs (when the symmetries do satisfy Equation 1).

The problem with restricting traversal to canonical elements is that there is no known efficient method for taking a set of permutations T and a state s, and computing the canonical representative of s under $[\![T]\!]$. Indeed, Clarke et al. [1] have shown that it is extremely unlikely that such a method exists:

Theorem 4.4 *Given states s and t and a set of permutations T, determining if s and t lie in the same orbit is as hard as the graph isomorphism problem.*

The graph isomorphism problem has been intensely studied by researchers in the field of computational complexity theory [9]. It is conjectured that there is no polytime algorithm for it; a consequence would be that there is no polytime procedure for canonicalization.

5 Heuristics for Canonicalization

It is not necessary to find the lexicographically largest equivalent state. In fact, any function $C : S_\eta \to S_\eta$ which has the property that $C(s) \in \Pi \cdot \{s\}$ can be used; the algorithm will continue to be correct, though it will traverse more states. We will refer to such functions as *reduction functions*.

5.1 Exact Reduction Functions

There are several important sets of generators for which canonicalization can be performed in polynomial time. For example, for a set $\Pi \subset S_N$ of permutations consisting solely of *transpositions* (i.e., permutations which interchange a pair of numbers and leave the remainder unchanged) and a vector $\vec{v} \in \{0,1\}^N$, canonicalization of $\vec{v}$ can be performed by "bubble sorting" $\vec{v}$ with respect to the transpositions in Π [3]. Similarly, if Π consists of a single permutation (which includes the special case of *rotational symmetry*), $[\![\Pi]\!]$ will have at most N elements; hence $[\![\Pi]\!] \cdot \{\vec{v}\}$ can be exhaustively searched.

Consider a hierarchical design at two levels, where the lower level has a number of replicated components. When there are symmetries on both the individual components, and at the higher level, between the components themselves, a global state can be canonicalized by first individually canonicalizing the local state of the components, and then canonicalizing the global state keeping the relative order of local states unchanged. More technically, suppose that there are n components, each with m local states bits. Then the following lemma holds:

Lemma 5.1 *Let $A = B \cup C_1 \cup C_2 \cup \cdots \cup C_n$ be a set of permutations from $S_{n \cdot m}$ so that every permutation σ in B preserves the relative ordering of the state bits local to each component, every permutation C_i acts only on the state bits local to component i, and for every i, C_i is a shifted version of C_1.*

Then the canonical representative of a vector s from $2^{n \cdot m}$ under $[\![A]\!]$ is the same as the state $\breve{s}$ derived iteratively by setting: $s_0 = s$, $s_{i+1} = \widehat{s_i}_{[\![C_i]\!]} (\forall i \in J_n)$, and $\breve{s} = \widehat{s_n}_{[\![B]\!]}$.

The complete proof is in [6].

5.2 Greedy Algorithms for Reduction

In this section we describe greedy reduction algorithms. We first consider a simple procedure for finding the orbit of a state which iteratively generates all the states derivable from the specified states by successive applications of permutations in Π, i.e., in a breadth-first manner. The drawback of this approach is that the size of the orbit can be very large.

Consider a modified procedure shown in Figure 3(a). Here at each step, we continue exploring only from the lexicographically largest state seen at the current iteration. Note that it is not necessary to store all the visited states, since only the lexicographically maximal state visited (shaded in black) is desired. The termination of the algorithm follows from the fact that the **while** loop continues only as long as $s_{\mathtt{max}}$, the best state seen so far, increases. Since the state space is finite, this can not go on forever.

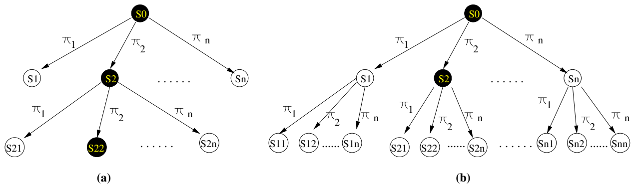

Figure 3: Greedy orbit traversal with (a) lookAhead 1, and (b) lookAhead 2 step 1.

The main problem with the algorithm in Figure 3(a) is that being greedy it can get trapped in local minima. One way of overcoming this is to add more "lookahead". For example, we could apply pairs of permutations to the current state, and in this way determine the best state obtainable by iteratively applying two permutations from Π. A greedy technique that has performed well in practice on a number of combinatorial optimization problems uses a lookahead of 2, but a "step" of only 1 [5]. This algorithm is shown in Figure 3(b).

Both the greedy procedures outlined above are susceptible to getting trapped in local minima [6]. In practice, it appears this happens mostly when presented with a single permutation which represents a rotational symmetry; it can be overcome by pre-processing the set Π by adding all permutations π^k for $\pi \in \Pi$.

6 Experimental Results

We have implemented an enumerative state space traversal routine with greedy symmetry reduction in VIS [10]. We experimented on examples which incorporate symmetry. The `4Tree` and `8Tree` designs are implementations of a tree-structured mutual exclusion protocol; they are hierarchical and afford substantial reduction. `3Cube` and `4Cube` are hypercube topologies incorporating a simple randomized routing protocol; `Dist` and `Star` are hybrid interconnection of processors, also incorporating a simple randomized routing protocol [11]. `DetGame` is included with VIS. All these examples are essentially asynchronous, and have very few inputs (corresponding to nondeterministic scheduling and routing).

Table 1: Results on Reachability Analysis (max. states, memory in bytes, time in seconds).

Benchmark	*BDD implicit*			*explicit*			*lookAhead1*			*lookAhead2step1*		
	States	Memory	Time	States	Memory	Time	States	Memory	Time	States	Memory	Time
4Tree	8584	188760	9	8584	111592	38	1073	13949	7	1073	13949	9
8Tree	222570	153792	71	222570	3338550	1705	6381	95715	89	6381	95715	131
3Cube	40320	2307336	144	40320	967680	95	2400	57600	25	1680	40320	241
4Cube	12870	4080768	60	12870	411840	90	2268	72576	27	1638	52416	43
Dist	63349	38160	8	63349	823537	230	1255	16315	10	1255	16315	60
Star	60480	4221360	306	60480	612360	396	7560	68040	97	7560	68040	414
DetGame	181440	6255168	447	181440	1632960	35	45360	408240	14	45360	408240	27

We report results on reachability analysis for these examples (this information could be used for deadlock detection, violation of mutual exclusion, etc.) in Table 1. The results indicate that explicit state enumeration coupled with symmetry reduction can be superior to BDD based analysis for these examples — sometimes dramatically so. (These experiments were conducted on a DEC Alpha with 1 GByte of main memory.)

Somewhat to our surprise, on most examples the greedy reduction procedure with a lookahead of 1 performed as well as the lookahead of 2 with a step of 1 in terms of reduction; in terms of running time, it was always substantially faster. We conjecture this was because the netlists had a simple topology; the most complex symmetries are those on the hypercube and it was on these examples that *lookahead2Step1* was able to find more reductions than *lookAhead1*. Still, increased number of search moves makes *lookAhead2Step1* inferior; thus we propose *lookAhead1* as the method of choice for symmetry reduction.

Note that we do not pack states into bit-arrays [3]. Instead, we pack multi-valued states into byte-arrays. So there is substantial scope for reducing memory usage in the explicit column. Instead of using compiled code execution model, we use fast cycle simulation techniques based on [12] and [13]. Additionally, we report the largest memory used by BDD for state sets encountered during reachability.

7 Summary

To summarize, we have made theoretical and practical contributions towards the use of symmetry in invariant verification. We developed a theory for exploiting symmetry for designs specified at the gate level. We gave theoretical justification for the use of explicit data structures instead of BDDs. We suggested heuristic procedures for symmetry reduction, and presented experimental results on a number of examples; the results are promising. In addition, these experiments demonstrated that explicit methods coupled with symmetry can be superior to BDDs

One problem with symmetry reduction is that it is better suited to asynchronous designs; when presented with a design with a large number of inputs, simply iterating through the possible inputs becomes infeasible. We plan to study methods for dealing with this. Computational group theory [14] has a long and rich history; we intend to study the literature of this field, and see if there are any ideas we can borrow for more efficient symmetry reduction. We also plan to work on routines for enumerative state space traversal available as an add-on to VIS. Another research problem is using symmetry information to simplify BDD verification; for example exploring the possibility of variable aliasing or projections through existential quantifications of some variables.

References

[1] E. M. Clarke, T. Filkorn, and S. Jha, "Exploiting Symmetries in Temporal Logic Model Checking," in *Proc. of the Computer Aided Verification Conf.*, 1995.

[2] E. A. Emerson and A. P. Sistla, "Symmetry and Model Checking," in *Proc. of the Computer Aided Verification Conf.*, 1993.

[3] N. Ip and D. Dill, "Better Verification Through Symmetry," in *Proc. Intl. Symp. on Computer Hardware Description Languages.*, 1993.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[5] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[6] W. Hung, "Exploiting Symmetry for Formal Verification," Master's thesis, The University of Texas at Austin, May 1997.

[7] M. A. Armstrong, *Groups and Symmetry*. Springer-Verlag, 1989.

[8] R. Rudell, "Dynamic Variable Ordering for Binary Decision Diagrams," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 42–47, Nov. 1993.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability*. W. H. Freeman and Co., 1979.

[10] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa, "VIS: A system for Verification and Synthesis," in *Proc. of the Computer Aided Verification Conf.*, July 1996.

[11] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[12] P. Ashar and S. Malik, "Fast Functional Simulation Using Branching Programs," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov. 1995.

[13] P. McGeer, K. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, and P. Scaglia, "Fast Discrete Function Evaluation," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov. 1995.

[14] C. C. Sims, *Computation with Finitely Presented Groups*, vol. 48 of *Encyclopedia of mathematics and its applications*. New York: Cambridge University Press, 1994.