A Genetic Approach for Conjunction Scheduling in Symbolic Equivalence Checking

Lun Li, Mitchell A. Thornton, Stephen A. Szygenda
Department of Computer Science and Engineering
Southern Methodist University, Dallas, TX 75275, USA
lli,mitch,szygenda@engr.smu.edu

Abstract

A key issue in symbolic equivalence checking algorithms is image computation. Conjunction scheduling is a strategy to keep the size of BDDs small for the intermediate results of image computation. Conjunction scheduling consists of ordering bit transition relations, clustering subsets of them and ordering the clusters. We present a genetic algorithm (GA) approach for conjunction scheduling based on the dependency matrix of transition relations. Our GA approach offers improvement over existing algorithms by minimizing the active lifetime and total lifetime of variables at the same time. Our experimental results show the effectiveness of the algorithm.

1. Introduction

The correct design of complex hardware continues to challenge engineers. Bugs in a design that are not uncovered in an early design stage can be extremely expensive. Simulation is a predominantly used tool to validate a design in industry. It can only validate all possible behaviors of a design in a brute-force manner. However, rapidly evolving markets demand short design cycles while the increasing complexity of a design makes simulation coverage less and less complete. Formal verification overcomes the weakness of exhaustive simulation by applying mathematical methodologies to validate a design. Formal approaches have the potential to scale to the complexity of VLSI designs because they exploit powerful tools in mathematics.

Binary Decision Diagrams (BDD) have led to a breakthrough in formal verification methods such as model checking and equivalence checking. Symbolic equivalence checking algorithms rely on Finite State Machine (FSM) state-space traversal. Key issues of BDD-based implementations of these algorithms include modeling a machine's transition relation as a cluster of BDDs and image computation. Image computation computes the successor of a set of states and it can be a bottleneck that consumes a lot of time. The transition relation of a machine is a relation denoted as $TR(S, X, S')$, where S and S' are present and next states respectively and X represents the inputs. A particular (S_i, X, S_i') triplet is in the TR if the machine will transition from state S_i to state S_i' under the input vector X. For a deterministic circuit, each binary memory element of the circuit gives rise to one term of the transition relation called a bit relation. When the circuit is synchronous, the partitioning is conjunctive, and can be written as a product of bit relations. Construction of a monolithic transition relation BDD is typically impractical. Instead, a partitioned transition relation BDD is represented by a set of conjunctions of disjoint subsets of the bit relation BDDs.

The efficient computation of images for partitioned transition relations depends heavily on the solutions to the following to two problems:

Clustering: Clustering bit relations so that the number of clustered relations is reduced without negatively affecting the size of intermediate BDDs.

Ordering: Ordering bit relations to cluster more bit relations and ordering clustered relations to minimize intermediate BDDs for image computation.

These two problems are not independent. A bad clustering usually results in a bad ordering. The benefit of clustering and ordering is early quantification. Early quantification quantifies out variables that will not appear in future computations. It is usually advantageous to quantify many variables soon, because the ensuing reduction in the support of the relations helps keep the size of the BDDs under control. Therefore, ordering and clustering relations is often viewed as the problem of finding a good quantification or conjunction schedule.

This problem has attracted significant attention over the last few years. Since the problem is known to be NP-hard [9], the algorithms that are commonly in use for clustering and ordering are heuristic in nature.

The rest of the paper is organized as follows. In section 2, we review background material and related work. Section 3 describes our GA approach and section 4 describes experimental results. Conclusions are provided in Section 5.

2. Preliminaries and related work

Notation: A synchronous sequential circuit or machine can be represented as a FSM. An FSM is a quintuple, $M = \{S, X, Y, \lambda, \delta\}$, where X denotes input wires, Y denotes output wires, S is a set of states, δ is the next state function, and λ is the output function. The next state function is a completely-specified function with domain ($X \times S$) and range S .

Given an FSM, its' transition relation is represented as the Boolean function $TR(S, X, S')$. Variable sets $S = s_1,...s_n$, $S' = s'_1,...s'_n$ and $X = x_1,...x_n$ are current state, next state, and input variables respectively. $TR(S_i, X, S'_i) = 1 \Leftrightarrow S'_i = \delta(X, S_i)$. For a deterministic circuit, each binary memory element of the circuit gives rise to one term of the transition relation, called the bit relation. When the circuit is synchronous, the partitioning is conjunctive, and can be written as the product of bit relations. In this paper, we assume that the transition relation is given as a product of bit relations TR_i.

$$TR(S, X, S') = \prod_{i=1}^{n} TR_i(S_i, X, S'_i) = \prod_{i=1}^{n} (S'_i \equiv \delta(X, S_i))$$

Partitioned transition relation: For a design that has over 20 bit relations, it is impractical to build a monolithic transition relation BDD for the entire machine. Therefore, ordering and clustering allow the transition relation to be written as:

$$TR(S, X, S') = \prod_{i=1}^{l} \hat{TR}_i(S_i, X, S'_i) \qquad (1)$$

where $l \le n$, and each cluster $\hat{TR}_i$ is the conjunction of some set of TR_i's. The conjunction scheduling problem consists of deciding the value of l and what bit relations are clustered together.

Given an FSM with a transition relation TR and a set of present states S, the image of S is the set of its successors and is computed by:

$$\text{Img(S)} = \exists S . \exists X . \left(S \wedge \prod_{i=1}^{l} \hat{TR}_i(S_i, X, S'_i) \right) \qquad (2)$$

Early quantification: Usually, the size of the BDD reduces by quantifying away variables in its' support set. Let Q denote the variables to be quantified, which is $Q = X \cup S$ and Q_i denotes the set of variables which do not appear in $\hat{TR}_1,...,\hat{TR}_{i-1}$. The image computation can be performed as follows:

$$\text{Img(S)} = \exists Q_1 \cdot (\hat{TR}_1 \wedge \exists Q_2 \cdot (\hat{TR}_2 \cdots \exists Q_l \cdot (\hat{TR}_l \wedge S))) \qquad (3)$$

The size of intermediate BDDs and the effectiveness of early quantification depends heavily on the order in which BDDs are conjoined in Eq. 3.

Conjunction scheduling: The ordering algorithm discussed here is based on the dependency matrix of the transition relation as defined in [6].

Definition 1 (Moon et al): The dependence matrix of an ordered set of functions ($f_1,...,f_m$) depending on variables $x_1,...,x_n$ is a matrix D with m rows and n columns such that $d_{i,j} = 1$ if function f_i depends on the variable x_j , and $d_{i,j} = 0$ otherwise.

When ordering transition relations for image computation, the rows of the dependence matrix correspond to a permutation of the transition relations; while the columns correspond to the variables that should be quantified out (i.e. present state variables $s_1,...s_n$ and input variables $x_1,...x_m$). We will assume that the conjunction is taken in the order of a bottom-up manner (i.e., $f_m, f_{m-1},..., f_2, f_1$). If $l_j(h_j)$ is the smallest (largest) index i in column j such that $d_{i,j} = 1$ respectively, then variable x_j can be quantified as soon as the subrelation corresponding to d_i is conjoined. This observation motivates the following definitions (Moon et al).

The normalized average total lifetime of the variables in matrix D is given by

$$\lambda = \frac{\sum_{i=1}^{n} (m - l_j + 1)}{n \cdot m}$$

The normalized average active lifetime of the variables in matrix D is given by

$$\alpha = \frac{\sum_{i=1}^{n} (h_j - l_j + 1)}{n \cdot m}$$

The two quantities λ and α are related to the quality of a conjunction schedule.

Related work: The importance of the quantification schedule was first recognized by Burch et al. [1] and Touati et al. [2]. Geist et al. [3] proposed a simple circuit independent heuristic algorithm, in which they ordered conjuncts by minimizing the maximal number of state variables of the intermediate BDDs in the processing of

performing image computation. Ranjan et al. [4] proposed a successful heuristic procedure (known as IWLS95). The algorithm begins by first ordering the bit relations and then clustering them and finally ordering the clusters again using the same heuristics. The order of relations is chosen using four normalized factors; the number of variables that will be quantified, the number of present state and primary input variables, the number of next state variables that would be introduced, and the maximum BDD index of a variable that can be quantified. After the ordering phase, the clusters are derived by repeatedly conjoining the bit relations until the size of the clustered BDD exceeds a given threshold, at which point a new cluster is started.

Bwolen Yang improved the IWLS95 heuristic in his thesis [5] by introducing a pre-merging phase where bit relations are initially merged pair-wise based on the sharing of support variables and the maximum BDD size constraint. Moon et al. [6] presented an ordering algorithm (known as FMCAD00) based on computing the Bordered Block Triangular form of the dependence matrix. Their ordering algorithm minimizes the active lifetime of variables, α. Instead of clustering ordered bit relations in a sequential order, the bit relations are clustered according to the affinity between them. Affinity measures the sharing of the support variables.

Chauhan et al. [7] extended FMCAD2000 and used combinatorial algorithms to improve the performance (i.e. simulated annealing). They also argue in favor of using α. All these techniques are static techniques. Subsequently, the same clusters and ordering are used for all the image computations during symbolic analysis.

Chauhan et al. [8] also proposed a non-linear dynamic quantification scheduling method by viewing the image computation as a problem of constructing an optimal parse tree for the image set. Their "Basic" algorithm is as follows: a heuristic score is computed for each variable in a set of variables Q to be quantified. The variable with the lowest score, say q, is chosen and two smallest BDDs in whose support set that q appears were conjoined. The overall approach is a two-phase approach combining static and dynamic schemes. Before image computation, only as many as the primary input variables are quantified out using the Basic algorithm. Then, for each image computation step, the remaining input and all present state variables are quantified out using the Basic algorithm.

3. Genetic Algorithm

A genetic algorithm emulates the metaphor of natural biological evolution to solve optimization problems. Genetic algorithms operate as shown in figure 1 through a simple flow.

1. *Initializing population*: find a collection of potential solutions to the problem, also called the current population.

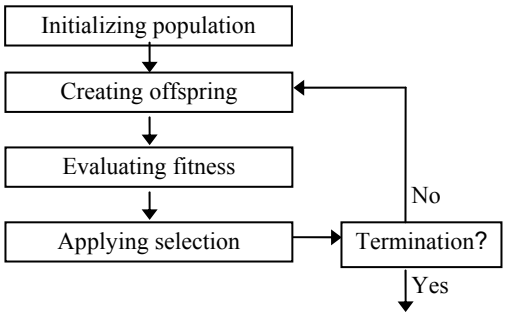

Figure 1. EA operation flow

2. *Creating offspring*: produce a new population through the application of genetic operations on selected members of the current generation.
3. *Evaluate fitness*: evaluate the quality of the solution in the new generation.
4. *Applying selection*: select solutions that will survive to become parents of the next generation based on their quality of solution to the problem. In this way, it is more likely that desirable characteristics are inherited by the offspring solutions.
5. This cycle repeats until some threshold or stopping criterion is met.

Typical genetic operations in a genetic algorithm include crossover, mutation and inversion. Crossover combines characteristics from two or more parents and places them in the resulting offspring. Mutation introduces new characteristics in the population by randomly modifying some of their building blocks, helping the search algorithm escape from local minima traps. Inversion rearranges the order of some characteristics.

3.1 Fitness function and selection

A fitness function measures the quality of each element. In [6] and [7], they both argue in favor of active lifetime α. However, the total lifetime λ and active lifetime α are not independent. A better λ could also result in a better α. For example, consider a 3-bit counter with present state variables s_1, s_2, s_3 and next state variables s_1', s_2', s_3', where s_3 and s_3' are the most significant bit. The transition relation of the counter can be expressed as the conjunction of the bit relations: $TR_1 = (s_1' \equiv \overline{s_1})$, $TR_2 = (s_2' \equiv s_1 \oplus s_2)$ and $TR_3 = (s_3' \equiv (s_1 \wedge s_2) \oplus x_3)$. Consider the following two

dependency matrices with order of $\pi_1(TR_1, TR_2, TR_3)$ and $\pi_2(TR_3, TR_2, TR_1)$ as shown in Figure 2.

The active lifetime and total lifetime for these two orders are $\alpha_{\pi_1} = 2/3$, $\alpha_{\pi_2} = 2/3, \lambda_{\pi_1} = 2/3$, $\lambda_{\pi_2} = 1$ respectively.

	s_1	s_2	s_3
TR_1	1		
TR_2	1	1	
TR_3	1	1	1

	s_1	s_2	s_3
TR_3	1	1	1
TR_2	1	1	
TR_1	1		

Figure 2. Dependency Matrices

From the example, we can see that both orders have the same active lifetime, but order π_1 is better than order π_2 because it has a small total lifetime. For order π_1, present state variables could be quantified out in the order of s_3, s_2, s_1, while for order π_2, no variables can be smoothed out in the intermediate computation. An advantage of the GA algorithm is that we can minimize total lifetime α and active lifetime λ at the same time. The fitness function we use includes these two parameters, shown as follows:

$$C(\pi_i) = a_0\lambda + a_1\alpha$$

where π_i is a permutation of transition relations. Weights a_0, a_1 are attached to two time parameters.

The selection is performed by linear ranking selection (i.e the probability that one element is chosen is proportional to its fitness). The size of the population is constant after each generation. Additionally, some of the best elements of the old population are inherited in the new generation. This strategy guarantees that the best element never gets lost and a fast convergence is obtained. Genetic algorithm practice has shown that this method is usually advantageous [10].

3.2 Genetic Operators

Two genetic operators are used in the algorithm: *Partially Matched Crossover* (PMX) [11] and a random *Mutation* (MUT).

PMX generates two children from two parents. The parents are selected by the method described above. The operator chooses two cut positions at random. Notice that a simple exchange of the parts between the cut positions would often produce invalid solutions. A validation procedure has to be executed after exchange. The detailed procedure for PMX follows.

Construct the children by choosing the part between the cut positions from one parent and preserve the position and order of as many variables as possible from the second parent. For example, $p_1 = \pi(1,2,3,4)$ and $p_2 = \pi(2,4,3,1)$ are the parents while $i_1 = 1$ and $i_2 = 3$ are the two cut positions. The resulting children before the application of the validation procedure are $c_1' = \pi(1,4,3,4)$ and $c_2' = \pi(2,2,3,1)$. The validation procedure goes through the elements between the cut positions and restores the ordering. This results in the two valid children $c_1 = \pi(1,4,3,2)$ and $c_2 = \pi(4,2,3,1)$.

MUT selects a parent by the method described above and randomly choose two positions. Two values at these two positions are exchanged.

3.3 Algorithm

Our genetic algorithm is outlined as follows:

1. The initial population is generated using the original order as first individual and by applying MUT to create more elements.
2. Genetic operators are selected randomly according to a given probability. The selected operator is applied to the selected parent (MUT) or parents (PMX). The better half of the population is inherited in each iteration without modification.
3. The new generation is updated according to their fitness.
4. The algorithm stops if no improvement is obtained for 50 iterations.

```
Genetic algorithm(){
  Generate_initial_population;
  Update_population;
  do{
       for( each child i ){
      j =linear_ranking_selection();
      randomly_select_method;
      case MUT: child( i ) = MUT(parent j );
      case PMX:  k = linear_ranking_selection();
      child( i , i +1 ) = MUT(parent j , k );
       }
    }
}
```

4. Experimental results

In order to evaluate the GA approach, we ran FSM traversal experiments on benchmarks from the ISCAS'89 and LGSYNTH'91 suites. The algorithm is implemented using the CUDD BDD package. All experiments are carried out on a 733MHz HP PC running cygwin under Windows XP with 192MB of main memory.

We compare the GA approach with FMCAD2000.

FMCAD00 is implemented in VIS 2.0. All parameters are set as default. Dynamic BDD variable reordering is enabled in both approaches and a time limit of 7200 seconds is used. The two parameters we measured are run time and peak number of live BDD nodes. Table 1 shows the result. Compared with FMCAD, our GA approach has a better result in most cases in terms of memory requirements. The GA approach spends significantly more time in the initial phase than FMCAD. When the number of image computations is large, the time cost of initial phase could be amortized.

5. Discussion and future work

We presented a Genetic Algorithm for the conjunction scheduling problem. It could be a practical alternative for existing methods. The result shows that GA performs well in terms of memory requirements.

Future areas of research are based on this work. The drawback of the GA approach is that, in general, good results are obtained with respect to the quality of the solution but the running times are much larger that of classical heuristics. Drechsler indicates that combining classic heuristics with a GA approach could outperform purely a GA approach in [11]. This motivates us to investigate preprocessing the dependency matrix with

Table 1. Comparison of FSM traversal

Circuits	Time (s)		Peak live nodes (K)	
	FMCAD	GA	FMCAD	GA
sbc	4.3	10.5	12.9	16.9
clma	24.7	137	142.4	32.7
clmb	30.6	137	141.6	32.7
mm9a	2.8	5.6	29.4	10.2
mm9b	55.4	5.57	702	17.1
mm30a	21.8	113	268.3	106.4
bigkey	88.8	1066	34.7	105.6
s420.1	19.7	11.0	0.673	0.642
s1512	641.6	371	121	65.9
s1269	1267	2669	1743	2079
s4863	242.5	6996	419.5	1014
s3271	Time out	2758	11256*	686.9

(* Reached to depth 12)

FMCAD2000 and use the GA approach on the result. This would guarantee that the starting points are not too bad and thus the convergence would be sped up. Our current approach is a static approach. The same clusters and ordering are used for all the image computations during symbolic analysis. A dynamic version of the GA approach is also a direction to explore. Before image computation, a GA approach could be used to quantify out as many input variables as possible. Then for each image computation, a simple GA approach could be used repeatedly to find a good order for current computation.

Acknowledgement

The authors would like to thank Dr. Nicole Drechsler for providing the PMX routine.

6. References

[1] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic Model Checking with partitioned transition relations", In Proceedings of the International Conference on Very Large Scale Integration, Edinburgh, Scotland, August 1991.

[2] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Implicit enumeration of finite state machines using BDDs", In Proceedings of the International Conference on Computer Aided Design (ICCAD), November 1990, pp. 130–133.

[3] D. Geist and I. Beer, Efficient Model Checking by automated ordering of transition relation partitions", In Proceedings of Sixth Conference on Computer Aided Verification (CAV), vol. 818 of LNCS, Stanford, USA, 1994, pp. 299–310.

[4] R. Ranjan, A. Aziz, B. Plessier, C. Pixley, and R. Brayton, "Efficient BDD algorithms for FSM synthesis and verification. In Proceedings of International Workshop on Logic Synthesis, Lake Tahoe, 1995.

[5] B. Yang, "Optimizing Model Checking Based on BDD Characterization", PhD thesis, Carnegie Mellon University, May 1999.

[6] I. Moon and F. Somenzi, "Border-block triangular form and conjunction schedule in image computation", In Proceedings of the Formal Methods in Computer Aided Design (FMCAD), vol. 1954 of LNCS, November 2000, pp. 73–90.

[7] P. Chauhan, E. Clarke, S. Jha, J. Kukula, H. Veith, and D. Wang, "Using combinatorial optimization methods for quantification scheduling", In Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME), September 2001.

[8] P. Chauhan, E. Clarke, S. Jha, J. Kukula, T. Shiple, H. Veith, and D. Wang," Nonlinear Quantification Scheduling in Image Computation", In Proceedings of International Conference on Computer Aided Design (ICCAD), 2001.

[9] R. Hojati, S. C. Krishnan, and R. K. Brayton. "Early Quantification and Partitioned Transition Relations", In Proceedings of the International Conference on Computer Design (ICCD), pp. 12-19, Austin, TX, October 1996.

[10] R. Drechsler, B. Becker, and N. Göckel, "A genetic algorithm for variable ordering of OBDDs", ", In Proceedings of the International Workshop on Logic Synthesis, Granlibakken, CA, May 1995.

[11] R. Drechsler, Evolutionary Algorithm for VLSI CAD, Kluwer Academic Publication, 1998.