Integrating SAT with Multiway Decision Graphs for Efficient Model Checking

Sa'ed Abed*, Otmane Ait Mohamed*, Zijiang Yang† and Ghiath Al Sammane*
*ECE Department, Concordia University, Canada
Email: {s_abed, ait, sammane}@ece.concordia.ca
† Western Michigan University, U.S.A.
Email: zijiang.yang@wmich.edu

***Abstract*— Multiway Decision Graphs (MDGs) are special decision diagrams that subsume Binary Decision Diagrams (BDDs) and extend them by a first-order formulae suitable for model checking of data path circuits. Satisfiability Checking (SAT) has emerged recently as an alternative for decision graphs. Their performance is less sensitive to the problem sizes and they do not suffer from state space explosion. In this paper, we propose a model checking methodology that allows to combine tightly MDGs and SAT. We use a rewriting based SAT solver to prune the transition relation of the circuits to produce a smaller one that is fed to the MDG model checker. We support our reduction methodology by experimental results executed on benchmark properties.**

I. Introduction

Model checking is an important formal verification technique that starts integrating digital system designs. It aims by exploring the reachable state space of a model to verify that an implementation satisfies a specification [1]. Binary Decision Diagram (BDD) [2], [3] is a canonical representation for Boolean functions that addresses this problem by providing an efficient encoding for the state space at the Boolean level. This representation allows model checkers to verify large systems. Still, most model checkers face the state space explosion problems while verifying large systems even using Symbolic Model Checking.

Multiway Decision Graph (MDG) [4] is an extension for BDD in the sense that it represents and manipulates a subset of first-order logic formulae suitable for large data path circuits. With MDGs, a data value is represented by a single variable of an abstract type and operations on data are represented in terms of an uninterpreted functions. The MDG operations and verification procedures are packaged as a set of tools and implemented in Prolog [5] providing facilities for hardware verification: invariant checking, equivalence checking and model checking.

An alternative for decision graphs is to represent the transition relation in Conjunctive Normal Form (CNF) and use Satisfiability Checking (SAT) with several properties that make them attractive compared to BDDs. Their performance is less sensitive to the problem sizes and they do not suffer from state space explosion. As a result, various researchers have developed routines for performing Bounded Model Checking (BMC) [6] and [7] using SAT. The common theme is to convert the problem of interest into a SAT problem, by devising the appropriate propositional Boolean formula, and to utilize other non-canonical representations of state sets. However, they all exploit the known ability of SAT solvers to find a single satisfying solution when it exists.

In this paper, we propose a model checking methodology that allows to tightly combine MDGs and SAT. We use a rewriting based SAT solver to prune the transition relation of the circuits to produce a smaller one that is fed to the MDG model checker. We support our reduction methodology by experimental results executed on benchmark properties.

II. Related Work

The idea of combining BDDs and SAT for verification has been the subject of several research. Given that both techniques perform an implicit search on the underlying Boolean space, it is no surprise that many different ways of combining them have been explored recently, frequently suited to the target application. Their relative benefits have been combined in many verification applications such as BMC [6], [7] and model checking [8].

In [9], the authors used BDDs to represent state sets, and a CNF formula to represent the transition relation. All valid *next state* combinations are enumerated using a backtracking search algorithm for SAT that exhaustively visits the entire space of primary input, present state and next state variables. However, rather than using SAT to enumerate each solution all the way down to a leaf, they invoked BDD-based image computation at intermediate points within the SAT decision procedure, which effectively obtains all solutions below that point in the search tree. In a sense, their approach can be regarded as SAT providing a disjunctive decomposition of the image computation into many subproblems, each of which is handled in the standard way using BDDs.

Reducing the space requirement in model checking has been suggested in several works like [10] and [11]. These studies suggest partitioning the problem in several ways. The work in [10] shows how to parallelize the model checker based on explicit state enumeration. They achieve it by partitioning the state table for reached states into several processing nodes. The work in [11] discusses techniques to parallelize the BDD-based reachability analysis. The state space on which reachability is performed is partitioned into disjoint slices, where

978-1-4244-1847-3/07/$25.00 ©2007 IEEE

each slice is owned by one process. The process performs a reachability algorithm on its own slice.

In [12], the authors proposed a technique to construct a reduced MDG model for circuits described at system level in VHDL. The simplified model can be obtained using a high level symbolic simulator called *TheoSim*, and by running an appropriate symbolic simulation patterns. The work here provides another technique based on SAT solver. A comparison and case study are part of an ongoing large project that aims to develop a complete system level verification flow.

III. Background

A. Boolean Satisfiability

The Boolean Satisfiability (SAT) problem is a well-known constraint satisfaction problem with many applications in computer-aided design, such as test generation, logic verification and timing analysis. Given a Boolean formula, the objective is to either find an assignment of 0-1 values to the variables so that the formula evaluates to true, or establish that such an assignment does not exist. The Boolean formula is typically expressed in CNF, also called product-of-sums form. Each sum term (clause) in the CNF is a sum of single literals, where a literal is a variable or its negation. An n-clause is a clause with n literals. For example, $(v_i + v'_j + v_k)$ is a 3-clause. In order for the entire formula to evaluate to 1, each clause must be satisfied, i.e. evaluate to 1.

In practice, most of the current SAT solvers are based on the Davis-Putnam algorithm [13]. The basic algorithm begins from an empty assignment, and proceeds by assigning a 0 or 1 value to one free variable at a time. After each assignment, the algorithm determines the direct and transitive implications of that assignment on other variables, typically called Boolean Constraint Propagation (BCP). If no contradiction is detected during the implication procedure, the algorithm picks the next free variable, and repeats the procedure. A conflict occurs when implications for setting the same variable to both 1 and 0 are produced. Otherwise, the algorithm attempts a new partial assignment by complementing the most recently assigned variable for which only one value has been tried so far. This step is called backtracking. The algorithm terminates either when all clauses have been satisfied and a solution has been found, or when all possible assignments have been exhausted. The algorithm is complete in that it will find a solution if it exists.

B. Model Checking using Multiway Decision Graph

MDG is a finite Directed Acyclic Graph (DAG) with one root, whose leaves are labeled by formulae of the logic True (T). The internal nodes are labeled by terms, and the edges issuing from an internal node v are labeled by terms of the same sort as the label of v. Such graph is a canonical representation of a certain quantifier-free formulae, called a *Directed Formulae* (DF). Each term in a DF belongs to either a concrete or abstract sort. Concrete sorts have enumerations, while abstract sorts do not.

A directed formula DF of type $U \rightarrow V$ is a formula in Disjunctive Normal Form (DNF) plus $\top$ (truth) and $\bot$ (false), where U and V are two disjoint sets of variables. Just as ROBDD must be *reduced* and *ordered*, MDGs must obey a set of well-formedness conditions given in [4]. DFs are used for two distinct purposes: to represent relations (transition and output relations) and to represent sets (sets of states as well as sets of input vectors and output vectors).

The input language of the MDG-tool is a Prolog-style hardware description language (MDG-HDL), which supports structural specification, behavioral specification or a mixture of both. A structural specification is usually a netlist of components connected by signals, and a behavioral specification is given by a tabular representation of transition relations or a truth table.

In MDG model checking, the properties to be verified are expressed by formulas in $\mathcal{L}_{MDG}$. $\mathcal{L}_{MDG}$ atomic formulae are Boolean constants True and False, or equations of the form $t_1 = t_2$, where t_1 is a variable (input, output or state variable) and t_2 is either a variable, an individual constant, an ordinary variable or a function of ordinary variables. Ordinary variables are defined to remember the values of the variables in the current state. The basic formulas (called $Next\_let\_formulas$) in which only the temporal operator **X** (next time) is allowed as follows [14]:

- Each atomic formula is a $Next\_let\_formulas$;
- If p, q are $Next\_let\_formulas$, then so are: !p (not p), p&q (p and q), p|q (p or q), p $\rightarrow$ q (p implies q), **X**p (next-time p) and LET (v=t) IN p, where t is a system variable and v an ordinary variable.

Using the temporal operators **AG** (*always*), **AF** (*eventually*) and **AU** (*until*), the properties allowed in $\mathcal{L}_{MDG}$ can have the following forms:

$$
\begin{aligned}
\text{Property} ::= \; & A(Next\_let\_formula) \\
| \; & AG(Next\_let\_formula) \\
| \; & AF(Next\_let\_formula) \\
| \; & A(Next\_let\_formula)\, U\, (Next\_let\_formula) \\
| \; & AG(Next\_let\_formula) \Rightarrow F(Next\_let\_formula) \\
| \; & AG((Next\_let\_formula) \Rightarrow \\
& \quad ((Next\_let\_formula)\, U\, Next\_let\_formula)))
\end{aligned}
$$

Model checking in the MDG system is carried out by building automatically additional circuit that represents the $Next\_let\_formulas$ appearing in the property to be verified, compose it with the original circuit, and then check a simpler property on the composite machine [15].

IV. Combining SAT and MDG

We start with a system level design and a set of properties written in $\mathcal{L}_{MDG}$. As shown in Figure 1, we extract form the behavioral design a transition relation in terms of DF. Then we apply an abstraction technique to create a CNF formula and a set of associated truth assignments constraints: B_{DF}. During this step we introduce Boolean variables for every clause in the transition relation with suitable arguments (primary variables (LHS) and uninterpreted function arguments). Also

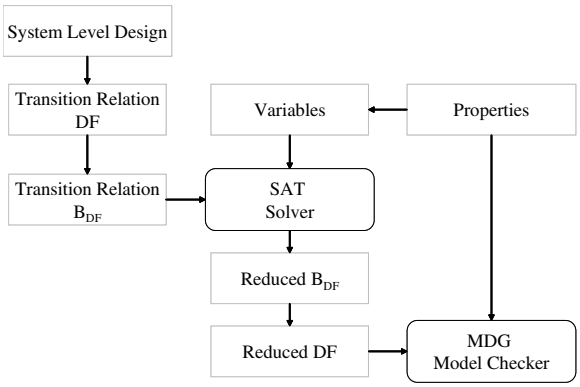

Fig. 1. Overview of the Methodology

we specify additional constraints between clauses with similar arguments to be mutual. From the properties, we extract the set of reduction variables and feed them with the B_{DF} to the rewriting based SAT solver which will decide the truth assignment and the implication of this assignment and produce a reduced transition relation: *Reduced* B_{DF}. Then, we transform again the *Reduced* B_{DF} to the reduced transition relation in term of DF. The obtained DF with the $\mathcal{L}_{MDG}$ properties will be fed to the MDG Model Checker. The formal verification is performed then on this obtained reduced MDG using the existing MDG package.

A. Abstracting CNF from DF

Algorithm 1 CREATECNFFORMULA(SYSTEM)

1: Formula = CreateLogicFormula(System);
2: BoolFormula = replace each term in Formula with a predicate;
3: Infer constraints between predicates;
4: Transform predicate to Boolean variable;
5: CNFFormula = ConvertToCNF(BoolFormula);
6: Return CNFFormula;

Algorithm 1 shows a sketch on how to obtain a transition relation in CNF. It first creates the transition relation in a general format at line 1. Assume the formula is

$$((x = 3) \wedge (y = 2)) \vee ((x = 5) \wedge (y = 4))$$

Line 2 will then introduce *n* predicates for every clause with LHS argument, so in the above formula we need four predicates and the formula becomes $(b_1(x) \wedge b_2(y)) \vee (b_3(x) \wedge b_4(y))$. Line 3 introduces additional constraints such that clauses with a similar LHS argument must be mutual. In this example we know that $b_1(x)$ and $b_3(x)$ cannot be true at the same time. Meanwhile, one of them has to be true, otherwise the formula cannot be satisfied $(b_1(x) \oplus b_3(x))$. Similar constraints can be applied to $b_2(y)$ and $b_4(y)$. Therefore, the Boolean formula $B(Tr_{DF})$ and the truth assignment constraints are shown below:

$$\begin{bmatrix} B(Tr_{DF}): & (b_1(x) \wedge b_2(y)) \vee (b_3(x) \wedge b_4(y)) \\ Constraints: & (b_1(x) \oplus b_3(x)) \\ & (b_2(y) \oplus b_4(y)) \end{bmatrix}$$

In line 4, we have resolved all dependencies and the predicates will be transformed to Boolean variables (i.e. $b_1(x)$ becomes b_{1x}). Note the Boolean formula is not in CNF yet. There exists linear algorithm to convert any Boolean formula to CNF [16], with additional variables introduced. As mentioned in line 5, the CNF representation for the above formula is:

$$\begin{bmatrix} B(Tr_{DF}): & (b_{1x} \vee b_{3x}) \wedge (b_{2y} \vee b_{3x}) \wedge \\ & (b_{1x} \vee b_{4y}) \wedge (b_{2y} \vee b_{4y}) \\ Constraints: & (b'_{3x} \vee b'_{1x}) \wedge (b_{1x} \vee b_{3x}) \\ & (b'_{4y} \vee b'_{2y}) \wedge (b_{2y} \vee b_{4y}) \end{bmatrix}$$

B. Extracting Variables from Properties

Our approach to select a variable and assign it a value is based on (assumption) extracted from the dependent variables on the property and hence the resulting transition relation will be much smaller. In fact, in large systems where the design can be expressed as a conjunction of the individual transition relations of the state variables, it consumes large memory and time to verify a property. Our approach gives the possibility to assign a concrete variables to the inputs of the system. Thus, an important reduction is gained on the resulting transition relation which improves the performance of the MDG model checker in terms of memory and CPU time.

Just as an example, if we assume that P1 is dependent on b_{1x}, then if the SAT solver decides b_{1x} to be true, then the implication we can get is:

$$\begin{bmatrix} B(Tr_{DF}): & b_{2y} \\ Constraints: & (b'_{4y} \vee b'_{2y}) \wedge (b_{2y} \vee b_{4y}) \end{bmatrix}$$

which represents a very small transition relation consisting of only 1 clause compared to the original one of 4 clauses, and hence improve the performance.

V. APPLICATION AND RESULTS

The MDG tool has been demonstrated on the example of the Island Tunnel Controller (ITC) in [17], which was originally introduced by Fisler and Johnson [18]. The ITC controls the traffic lights at both ends of a tunnel based on the information collected by sensors installed at both ends of the tunnel: there is one lane tunnel connecting the mainland to an island. At each end of the tunnel, there is a traffic light as depicted in Figure 2.

The ITC is composed of five modules: The Island Light Controller (ILC), the Tunnel Controller (TC), the Mainland Light Controller (MLC), the Island Counter and the Tunnel Counter (refer to [18] for the state transition diagrams of each component).

We use the same case study and we consider the ITC with its properties as a benchmark in order to measure the influence of our method on the MDG model checking performance. Table 1 compares the verification results with and without reduction for five properties, run on a Sun enterprize server with Solaris 5.7 OS and 6.0 GB memory. We give the CPU time measured in seconds and the memory measured in MB that are used in building the reduced machine and checking the property.

TABLE I

COMPARING MODEL CHECKING RESULTS WITH & WITHOUT REDUCTION

Benchmark Properties	*Without Reduction*			*With Reduction*		
	Time	**Memory**	**Nodes**	**Time**	**Memory**	**Nodes**
P1	65.35	50.1	123080	53.65	47.6	121060
P2	0.12	0.57	263	0.10	0.4	211
P3	65.45	48.6	123085	9.73	5.24	12292
P4	65.61	46.4	123082	35.05	26.11	63419
P5	65.89	48.3	123080	48.42	34.95	69966
Average	52.48	38.79	98518	29.39	22.86	53389

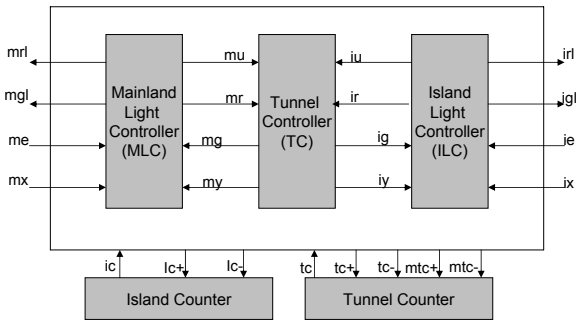

Fig. 2. Island Tunnel Controller Structure

We note that the reduction gain depends on the properties. The best gain in performance is obtained with property P3 where the time is reduced by 6.7 times the original one and the memory is reduced by a factor of 9.3 times. The worst case is the property P1 where the time is reduced by 1.2 times the original one and the memory reduction is not profitable.

In the case of property P1 the assumptions and the functionality tested needs several runs (when using our SAT reduction as case splitting). The sum of these runs for this particular case is a little bit lower to a single run without reduction. For P3, case splitting was really much more efficient. These differences show the sensitivity of the reduction technique to the property verified. Despite these fluctuations, the gain average in performance is a factor of 2 which is considered as a good result in the case of model checking approaches.

VI. CONCLUSIONS

We have proposed a reduction technique to integrate SAT solvers within the MDG model checking tool. We have used the specification of the design provided as properties to extract a reduced model that is verified by the MDG-tool. In fact, the SAT solver reduces the control aspects of the design while MDG abstracts the data aspect in order to targets the formal verification of increasingly larger designs. The obtained performance is promising as it has been shown in the experimental results. However, the approach still in its early stages. Future work includes further applications using challenging industrial benchmark in order to compare results with commercial model checking tools.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. E. Long, *Model Checking*. In Nato ASI, vol. 152 of F, Springer-Verlag, 1996.

[2] R. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Transactions in Computer Systems, 35(8): 677-691, August 1986.

[3] R. Bryant, *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams*. ACM Computing Systems, 24(8): 293-318, 1992.

[4] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny, *Multiway Decision Graphs for Automated Hardware Verification*. Formal Methods in System Design, 10(1): 7-46, 1997.

[5] W. Clocksin and C. Mellish. *Programming in Prolog*. Springer-Verlag, 3rd edition, 1987.

[6] M. Ganai and A. Aziz, *Improved SAT-based Bounded Reachability Analysis*. In Proceedings of VLSI Design Conference, 2002.

[7] P. A. Abdulla, P. Bjesse, and N. Een, *Symbolic Reachability Analysis based on SAT-Solvers*. In Proc. of Workshop on Tools and Algorithms for the Analysis and Construction of Systems (TACAS), 2000.

[8] A. Gupta, M. Ganai, Chao Wang, Zijiang Yang, and P. Ashar, *Learning from BDDs in SAT-based bounded model checking*. Design Automation Conference, 2003. Proc., Vol., Iss., 2-6 June 2003, pp. 824- 829.

[9] A. Gupta, Z. Yang, P. Ashar, and A. Gupta, *SAT Based State Reachability Analysis and Model Checking*. 3rd International Conference on Formal Methods in Computer-Aided Design (FMCAD), 2000.

[10] U. Stern, and D.L. Dill, *Parallelizing the Murphi Verifier*. In *Presented at Computer-Aided Verification*, 1997.

[11] T. Heyman, D. Geist, O. Grumberg and A. Schuster, *Achieving scalability in parallel reachability analysis of very large circuits*. Presented at Computer-Aided Verification, 2000.

[12] G. Al Sammane, S. Abed, and O. Ait Mohamed, *High level reduction technique for multiway decision graphs based model checking*. In First International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS'07), British Computer Society 2007.

[13] M. Davis and H. Putnam. *A computing procedure for quantification theory*. Journal of the ACM, 7:201205, 1960.

[14] O. Ait Mohamed, X. Song, and E. Cerny. *On the non-termination of MDG-Based Abstract State Enumeration*. Theoretical Computer Science Journal, 1-3(300): 161-179 ,2003.

[15] Y. Xu, E. Cerny, X. Song, F. Corella and O. Ait Mohamed. *Model Checking for a First-order Temporal Logic Using Multiway Decision Graphs*. In Proc. Conf. on Computer-Aided Verification (CAV'98), pp. 219-231, volume 1427 of Lecture Notes in Computer Science, Springer-Verlag, Vancouver, Canada, July 1998.

[16] Miroslav N. Velev. *Efficient translation of boolean formulas to CNF in formal verification of microprocessors*. In Proceedings of the 2004 conference on Asia South Pacific design automation (ASP-DAC '04), pp. 310-315, IEEE Press, Yokohama, Japan, 2004.

[17] Z. Zhou, X. Song, S. Tahar, E. Cerny, F. Corella, and M. Langevin. *Formal verification of the island tunnel controller using multiway decision graphs*. In Formal Methods in Computer Aided Design (FMCAD), 1996.

[18] K. Fisler and S. Johnson. *Integrating design and Verification Environments Through A Logic Supporting Hardware Diagrams*. In Proc. of IFIP Conference on Hardware Description Languages and their Applications (CHDL'95), Chiba, Japan, August 1995.