SAT Based Model Checking for MDG Models

Khaza Anuarul Hoque, O. Ait Mohamed
Dept. of ECE,
Concordia University,
Montreal, Canada
Email: {k_hoque,ait}@ece.concordia.ca

Sa'ed Abed
Dept. of Computer Engineering,
Hashemite University,
Zarqa, Jordan
Email: sabed@hu.edu.jo

Mounir Boukadoum
Dept. of Computer Science,
UQAM,
Montreal, Canada
Email: boukadoum.mounir@uqam.ca

***Abstract*— Multiway Decision Graph (MDG) is a canonical representation of a subset of many-sorted first-order logic. It generalizes the logic of equality with abstract types and uninterpreted function symbols. The area of Satisfiability (SAT) has been the subject of intensive research in recent years, with significant theoretical and practical contributions. From a practical perspective, a large number of very effective SAT solvers have recently been proposed, most of which based on improvements made to the original Davis-Putnam algorithm. Local search algorithms have allowed solving extremely large satisfiable instances of SAT. The combination between various verification methodologies will enhance the capabilities of each and overcome their limitations. In this paper, we introduce a model checking methodology for MDG based models using MDG tool and SAT solver. We use SAT solver searching for feasible paths of reachable states satisfying the property under certain encoding constraints. Finally, we provide a case study showing the correctness and the efficiency of our approach.**

I. Introduction

Most of the efforts today are spent on developing Satisfiability Checking (SAT) based tools to perform several forms of model checking as they are less sensitive to the problem sizes and the state explosion problem of classical Binary Decision Diagram (BDD) based model checkers.

Multiway Decision Graphs (MDGs) [1] are a special kind of decision diagrams that subsumes Binary Decision Diagrams (BDDs) and extends them by canonically and compactly representing a subset of first-order functions. The MDG system is a decision diagram based verification tool, designed for hardware verification. It supports both equivalence checking and model checking. With MDGs, a data value is represented by a single variable of an abstract type, and operations on data are represented in terms of uninterpreted functions.

An alternative for decision graphs is to represent the transition relation in Conjunctive Normal Form (CNF), and use SAT with several properties that make them attractive in comparison to BDDs. There 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) [2],[3] 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 ability of SAT solvers to find a single satisfying solution, when it exists.

A desirable approach is to develop synergies between various verification methodologies, and between design and verification, in order to overcome the limitations and to to enhance the capabilities of each and hence; our work is motivated by this goal. In this paper, we present a new model checking methodology using SAT on MDG based models. The main idea is to use the SAT solver as a verification engine for proving a correctness formula as a tautology. We present our step-by-step procedure for building the correctness formula with our proposed methodology.

The paper is organized as follows: Section II gives a brief description of related works in this area. Section III gives some preliminaries on MDG and SAT. Section IV presents the main contribution of the paper, the detailed verification flow for MDG models. A case study and experimental results are presented in section V. Finally, Section VI concludes the paper and points to future work.

II. Related Works

BDD and SAT based verification have been a major field of interest for researchers for a long time. Given that both techniques perform an implicit search in the underlaying Boolean space, it is no surprise that different approaches have been explored recently to combine both of them for target applications. Their benefits have been combined in many applications such as BMC[4],[5] and model checking [6].

In [7], 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.

Model checking techniques for security protocol analysis based on reduction to Boolean logic has been explained in [8]. The main idea was, given a protocol description in multi-set formalism and an integer k, to build propositional formulas whose models correspond to attacks on the protocol.

978-1-4244-6805-8/10/$26.00 ©2010 IEEE

Propositional formulas are checked for satisfiability to indicate an attacks on the protocol. In [9], an overview of a very interesting methodology integrating SAT and MDG model checker tool was presented by the authors with preliminary experimental results. The basic idea was to use the SAT solver as a reduction engine within the MDG model checker tool [1]. As a continuation of that work, in this paper, we present a SAT encoding technique (CNF conversion) for MDG Direct formula (DF). For the conversion part, Tseitin approach has been used[10] while introducing "fresh variables" only for AND gates. Also, we present the application of our SAT encoded Directed Formula for a complete SAT based model checking methodology of MDG models using SAT solver as a verification engine. Implementation of SAT for model checking with Multiway Decision Graph(MDG) distinguishes our approach from others.

III. Preliminaries

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 $\perp$ (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 [1]. 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).

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 Conjunctive Normal Form (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. In practice, most of the current SAT solvers are based on the Davis-Putnam algorithm [11].

IV. SAT Based Verification approach

Figure 1 shows our methodology. The basic idea is as simple as basic model checking, but the implementation of SAT on MDG models makes the approach unique, unlike traditional BDD or other SAT based verification approaches. We construct a propositional formula Ω which is $satisfiable$ iff f is valid along a path and prove the formula as a tautology.

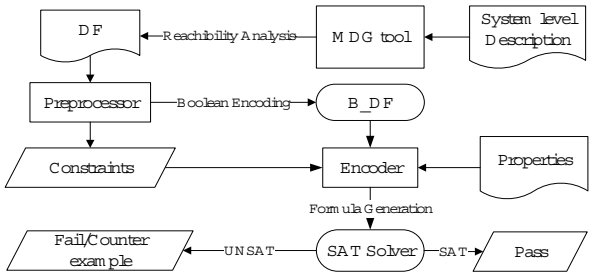

Fig. 1. SAT based verification for MDG models

We start from a Behavioral/RTL description of the system. The MDG tool explores the sets of reachable states and outputs the DF (containing all the set of states). We pass the DF through the preprocessor. The preprocessor generates the Boolean encoded B_DF with constraints after the elimination of Equality with Uninterpreted Functions (EUF) applications. Then the encoded B_DF is fed to the Encoder along with the property and the constraints to provide a linear DF to CNF conversion. Finally, we generate a correctness formulas and use the SAT solver to prove that each of these formulas is a tautology. The procedure's details are described in the following subsections.

A. DF Extraction

The input language of the MDG-tool is a Prolog-style hardware description language (MDG-HDL), that 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. The MDG tool applies the reachibility algorithm[12] and gives all the possible sets of reachable states in terms of DF. We conjunct all these DFs to construct the complete DF, representing all the sets of reachable states e.g. $DF_{complete} = DF_1 \wedge DF_2 \wedge DF_3 ..... \wedge DF_n$. Here n is the number of transitions the reachibility analysis algorithm needs to terminate.

B. Preprocessing Step

Before applying the CNF conversion algorithm on the DF, the preprocessor eliminates the EUF applications and introduces boolean encoding with adequate constraints :

Boolean Encoding for Clauses and Introducing Constrains: Given the formula $(r = 0) \bigwedge (f = 1) \bigwedge (v_2 = u_2)) \bigvee (r = 1) \bigwedge (f = 0) \bigwedge (v_2 = g(u_2)))$, we introduce Boolean encoding for abstracting the clause $(r = 0), (r = 1), (v_2 = u_2) and (f = 0)$. Constraints are also introduced at the same time. For this example, we know that $(r = 0)$ and $(r = 1)$ can not be true at the same time. Meanwhile one of them must be true, forcing them to be mutually exclusive, otherwise the equation will not be satisfiable. A similar constraint is also applicable to $(f = 0)$ and $(f = 1)$.

EUF Elimination: Our EUF elimination approach is inspired by using nested ITEs [13]. For example, if $g(x_1, y_1)$, $g(x_2, y_2)$

and $g(x_3, y_3)$ are three applications of UF $g()$, then the first application will be eliminated by a new term variable c_1. The second one will be replaced by $ITE((x_2 = x_1) \wedge (y_2 = y_1), c_1, c_2)$, where c_2 is a new term variable. The third one will be replaced by $ITE((x_3 = x_1) \wedge (y_3 = y_1), c_1, ITE((x_3 = x_2) \wedge (y_3 = y_2), c_2, c_3))$, where c_3 is a new term variable. For ITE terms, we define $encITE$ as:

$$encTr(ITE(G, T_1, T_2)) = encDF(G) \wedge encTr(T_1) \vee \neg encDF(G) \wedge encTr(T_2)$$

where $encTr(T_1)$ and $encTr(T_2)$ represent Boolean encoded terms and $encDF(G)$ represents an encoded propositional of formula G. For some cases, we modified Bryant's encoding slightly for the MDG DF case. For example, if the formula inside ITE contains a comparison between two different constants (such cases sometime occurs in MDG DF), then it is always false. So, we define the encoding for such cases as:

$$encTr(ITE(G_{const_1 = const_2}, T_1, T_2)) = encTr(T_2)$$

C. Encoding

The encoder takes the DF and converts them to CNF. Algorithm 1 shows the complete algorithm for the encoding and conversion. A DF is the conjunction of several individual DF, where each of these DF is in DNF format:

$$DF_{complete} = \bigwedge DF_i; \quad (1)$$

where i is the number of transitions the reachibility analysis algorithm needs to terminate and DF_i is a DNF. So, it is enough to get the equivalent CNF for each DF_i and conjunct them because conjunction of CNF is also a CNF.

$$DF_{CNF} = \bigwedge CNF_{DF_i}; \quad (2)$$

The Tseitin[10] algorithm for computing $CNF(DF)$ is well known. In Tseitin, a new variable for every logical gate is introduced. The variable imposes a constraint that preserves the function of that gate. $Example1$ Given a DNF formula

$$(a \wedge b) \vee (c \wedge d) \quad (3)$$

With Tseitin encoding, a new variable for each subexpression is introduced. Let us assign the variable x to the first 'and' gate (representing the subexpression $a \wedge b$), y for the second 'and' gate (representing the subexpression $c \wedge d$). We also introduce a new variable z to represent the top most operator. For a DF, the top most operator which is always an 'OR' gate connected with several 'AND' gates. We need to satisfy the two equivalence:

$$\begin{aligned} x &\Longleftrightarrow a \wedge b \\ y &\Longleftrightarrow c \wedge d \end{aligned} \quad (4)$$

The first equivalence can be written in CNF as:

$$(\neg x \vee a) \wedge (\neg x \vee b) \wedge (\neg a \vee \neg b \vee x) \quad (5)$$

The second equivalence can be written in CNF as:

$$(\neg y \vee c) \wedge (\neg y \vee d) \wedge (\neg c \vee \neg d \vee y) \quad (6)$$

Algorithm 1 CreateCNFFormula(DF)

```
1: Formula = MDG Direct Formula;
2: Replace UF's by term variables;
3: Infer constraints between predicates;
4: DF_bool = Transform predicates to Boolean variables;
5: for each DNF_i in DF_bool do
6:     CNF_DNFi = ConvertoCNF(DNF_i);
7: end for
8: CNF_complete = Conjunct all CNF_DNFi;
9: ReturnCNF_complete;
```

The overall CNF formula is the conjunction of (5), (6) and the unit clause (z) which represents the top-most operator. Instead of (z) we prefer to use $(x \vee y)$ which represents the same. The converter keeps track by mapping the Tseitin variable for each logic gates. In the example, $Equation$ (4) represents this mapping. Such mapping can be used for the verification of conversion.

D. Verification Using SAT Solver

Given a DF in CNF format, a SAT solver can be used to search for a path such that the property holds true at all the nodes in that path. If at least one such path exists, then the formula is satisfiable, indicating that property is true for the given model. Absence of a feasible path indicates a violation of the property. For our case, we are interested to prove that the inverse is unsatisfiable; indeed, rather than proving that F is satisfiable, it is faster and easier to prove that $\neg F$ is unsatisfiable. For example, the formula $(a = b) \Rightarrow f(a) = f(b)$ can be validated iff $(a = b) \wedge f(a) \neq f(b)$ is UNSAT. Verification using SAT solver contains several steps:

1) We Conjunct the directed formula F_{DF} with the Property P so that $F = F_{DF} \wedge P$.
2) As a second step, the conjunction is performed between encoding constraints $F_{constraint}$ with F so that, $F = F_{DF} \wedge P \wedge F_{constraint}$.
3) We check with a satisfiability solver if formula F is a $tautology$, i.e., the formula is valid. Any efficient SAT solver can be used for this purpose.

A satisfiable decision by the solver indicates violation of the property and gives a counter example, whereas an unsat decision validates the property. If satisfiable, the assignments constitutes a counter example to the original(un-negated) formula. Optionally, the satisfiable assignments can be substituted in the negation of the formula F and a theorem that the counter example implies the negated formula can be derived.

V. Application and Results

Figure 2 shows the structural description of a simple program counter. The inputs r and $f(pc)$ and its output is out_reg which gives the information about the current state of the controller $M1$. The structural description includes a data register pc and an 8 to 1 multiplexer. Two functional blocks represented by the uninterpreted function symbol inc and f. inc takes pc as input and produces an abstract value $inc(pc)$. f is a cross term that takes pc as its abstract input and produces a concrete output out_f of sort bool. The transition relation

TABLE I
VERIFICATION TIME FOR A PROGRAM COUNTER

Properties	Formula Gener. Time	Verification Time	Total Time	Memory
P1	1.22	2.142	3.36	1.34
P2	1.19	2.127	3.31	1.30
P3	1.16	2.091	3.25	1.12
P4	1.07	1.663	2.73	1.04
P5	0.98	1.652	2.63	1.01

of the circuit is defined as follows:
$TR = (out\_reg = 0 \wedge out\_f = 1 \wedge r = 1) \vee$
$(out\_reg = 1 \wedge r = 1) \rightarrow pc' = inc(pc) \vee$
$(out\_reg = 1 \wedge r = 0) \rightarrow pc' = zero \vee$
$(pc' = pc)$
The initial state is defined by the DF $(pc = zero)$. The MDG tool computes the reachable states in term of a directed formula to be fed to the preprocessor. The preprocessor takes the DF as input and generates a boolean encoded DF with constraints. The encoder collects the boolean encoded DF, constraints and the property. After CNF conversion of the DF, the encoder generates a correctness formula based on the conjunction of DF, constraints and the property. MiniSAT 2.0[14] has been used as an efficient SAT solver to verify the correctness formula.

Our methodology is implemented in C++. For the experiments, solaris 5.10 workstation was used containing a quad-core processor running at 2.5GHz and having 6 GB of physical memory. Five different properties was verified for this circuit (Figure 2) and obtained the results summarized in Table I. The CPU time is given in milliseconds and the memory usage is given in megabytes. For five different properties, five different formulas were generated and each of these had to be a tautology. For property-1, it took about 1.22 millisecond to generate the formula whereas only 0.98 milliseconds was enough to generate the formula for property-5. The number of variables in the property and the number of constraints in correctness formula affect the formula generation time. In our case, property-1 has the maximum number of variables and property-5 has the least. MiniSat took only 1.652 millisecond to verify property-5. The longest verification time was for property-1, about 2.142 milliseconds. The size of the correctness formula dominates the verification time using SAT. The total verification time for the whole approach was only 3.36 milliseconds for property-1. Property-2 and property-3 took similar time. Property-4 and property-5 took even less than 3 milliseconds to be verified. Memory usage never crossed more than 1.34 megabyte during the verification.

VI. CONCLUSION

We have proposed a complete model checking methodology for MDG based models using the SAT solver. The native structure of Multiway Decision Graph (MDG) facilitates the CNF conversion of directed formula. It also suited our SAT-based model checking approach and produced interesting experimental results. Future directions will concentrate on experimenting our methodology with bigger circuits and comparing the results with other model checkers. However, using different algorithms on MDG DF for CNF conversion and comparing the SAT solvers performance for model checking can also be interesting. As this paper is a continuation of an ongoing research to integrate a SAT within the MDG model checker tool, we soon expect to publish the experimental results for our complete methodology with new relevant case studies.

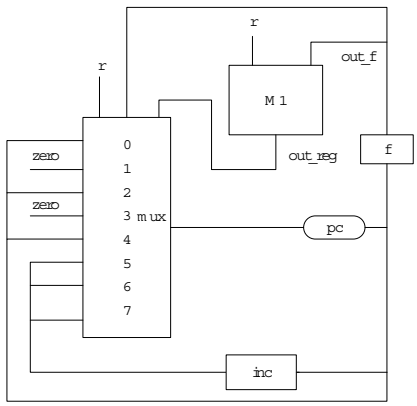

Fig. 2. Structural description of a simple program counter

REFERENCES

[1] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny, "Multiway Decision Graphs for automated hardware verification," in *Formal Methods in System Design*, vol. 10, no. 1, February 1997, pp. 7–46.

[2] M. Ganai and A. Gupta, "Completeness in SMT-based BMC for software programs," in *DATE*, 2008, pp. 831–836.

[3] O. Strichman, "Pruning techniques for the SAT-based bounded model checking problem," in *CHARME*, 2001, pp. 58–70.

[4] M. K. Ganai and A. Aziz, "Improved SAT-based bounded reachability analysis," in *VLSI Design*, 2002, pp. 729–734.

[5] P. A. Abdulla, P. Bjesse, and N. Eén, "Symbolic reachability analysis based on SAT-solvers," in *TACAS*, 2000, pp. 411–425.

[6] A. Gupta, M. K. Ganai, C. Wang, Z. Yang, and P. Ashar, "Learning from bdds in SAT-based bounded model checking," in *DAC*, 2003, pp. 824–829.

[7] A. Gupta, Z. Yang, P. Ashar, and A. Gupta, "SAT-based image computation with application in reachability analysis," in *FMCAD*, 2000, pp. 354–371.

[8] A. Armando and L. Compagna, "SAT-based model-checking for security protocols analysis," *Int. J. Inf. Sec.*, vol. 7, no. 1, pp. 3–32, 2008.

[9] S. Abed, O. A. Mohamed, Z. Yang, and G. A. Sammane, "Integrating SAT with Multiway Decision Graphs for Efficient Model Checking," in *Proc. of IEEE ICM'07*. Egypt: IEEE Press, 2007, pp. 129–132.

[10] G. Tseitin, "On the complexity ofderivation in propositional calculus," *Studies in Constrained Mathematics and Mathematical Logic*, 1968.

[11] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[12] S. Abed, O. A. Mohamed, and G. A. Sammane, "Reachability analysis using multiway decision graphs in the hol theorem prover," in *SAC*, 2008, pp. 333–338.

[13] S. G. R. Bryant and M. Velev, "Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic," *ACM Trans. Comput. Log.*, vol. 2, no. 1, pp. 93–134, 2001.

[14] N. Sorensson and N. Een, "MiniSat v1.13 a SAT solver with conflict-clause. minimizationn," in *Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, vol. 3569. St. Andrews, UK: Springer-Verlag, 2005.