An Automated SAT Encoding-Verification Approach for Efficient Model Checking

Khaza Anuarul Hoque, O. Ait Mohamed
ECE Dept, Concordia Univ., Canada
{k_hoque,ait}@ece.concordia.ca

Sa'ed Abed
CE Dept., Hashemite Univ., Jordan
sabed@hu.edu.jo

Mounir Boukadoum
CS Dept., UQAM, Canada
boukadoum.mounir@uqam.ca

***Abstract*— In this paper, we introduce an automated conversion-verification methodology to convert a Directed Formula (DF) into a Conjunctive Normal Form (CNF) formula that can be fed to a SAT solver. In addition, the formal verification of this conversion is conducted within the HOL theorem prover. Finally, we conduct experimental results with different-sized formulas to show the effectiveness of our methodology.**

I. Introduction

Many efforts are spent today on developing Satisfiability Checking tools (SAT) to perform 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. As a result, researchers have developed several routines for performing Bounded Model Checking (BMC) [1][2] 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.

The Multiway Decision Graph (MDG) [3] is an extension of the Binary Decision Diagram (BDD) in the sense that it represents and manipulates a subset of first-order logic formulae suitable for large datapath circuits. With MDGs, a data value is represented by a single variable of an abstract type and data operations are represented in terms of an uninterpreted functions.

In a previous work [4], a methodology integrating SAT and the 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 [3]. As a continuation of that work, this paper presents a completely automated conversion-verification approach for the conversion of MDG Direct Formulas (DF) to CNF (which will be fed to the SAT solver). Also, we provide the correctness of this conversion. The Tseitin approach is used [5] for the conversion part while only introducing "fresh variables" for AND gates. The obtained CNF formula is formally compared to the original DF using the HOL theorem prover. This will enhance confidence in the whole verification process.

II. Related Works

Their exist two possible ways to eliminate Equality with Uninterpreted Functions(EUFs), while enforcing their property of functional consistency: Ackermann constraints [6] and nested If-Then-Else operators (ITE) [7]. In [6], the UF was replaced with a new term variables and the next application of UF with respect to the previous one was enforced by extending the resulting formula with constraints. Such constraints were added for each pair of applications of that UF. Bryant presented an approach to eliminate the applications of UF with nested ITEs in [7]. In the nested ITE scheme, the first application of the UF is still replaced by a new term variable. However, the subsequent applications are eliminated by introducing nested ITEs with new term variables while preserving functional consistency. We prefer the nested ITE scheme which directly captures the functional consistency and readily exploits the maximal diversity property while Ackermann's can not[7].

Lack of a fast and efficient CNF generation algorithm has always been a bottleneck for CNF based SAT solvers and, hence, researchers paid much attention to this point [8]. Moreover, most of the CNF generation algorithms used in practice were minor variations of Tseitin's linear time algorithm [5]. Another CNF conversion algorithm came from Velve [9], showing an efficient CNF generation technique with identifying gates with fan-out count of 1 and merging them with their fan-out gate to generate a single set of equivalent CNF clauses. Nested ITE chains, where each ITE is used only as else argument of the next ITE, are similarly merged and represented with a single set of clauses without introducing intermediate variables. Such approach is good for pipelined machine verification problems, identifying certain patterns arising in formulas.

Very recently, an algorithm was proposed [10] for converting Negation, ITE, Conjunction and Equivalence (NICE dags) to CNF. A new data structure called NICE dag subsumes AND-Inverter Graphs (AIG). In all, the approaches described above use an intermediate representation or data structure for the boolean formula (either RBC, AIGER or NICE dag). The MDG Directed Formula is itself a DAG, so intermediate DAG representation is not required before conversion. The most interesting thing we observed is that, in most of the papers, a "paper and pencil" sketch was given as proof of their conversion approach. This also motivated us to build an automated tool for the verification of conversion.

In [4], a preliminary approach to integrate SAT with the MDG model checker tool was presented with an overview of the algorithm for CNF conversion. But the complete algorithm for DF to CNF encoding and implementation details were

978-1-4244-5816-5/09/$26.00 ©2009 IEEE

missing in that work. In this paper, we bring two main contributions: We have automated the complete algorithm to convert MDG DF to CNF using the Tseitin algorithm [5] while introducing "fresh variables" only for AND gates. The native structure of MDG direct formula made our life easy for the conversion. Also unlike other researchers, for our methodology, we present an automated process to perform the formal verification of the implementation of CNF conversion algorithm using the HOL theorem prover.

III. Preliminaries

Multiway Decision Graph: MDG is a graph representation of a class of quantifier-free and negation-free first-order many sorted formulae. It subsumes the class of Bryant's (ROBDDs) [11] while accommodating abstract data and Uninterpreted Function symbols. MDG can be seen as a Directed Acyclic Graph (DAG) with one root, whose leaves are labeled by formulae of the logic True (T) [3], such that: (1) Every leaf node is labeled by the formula T, except if the graph G has a single node, which may be labeled T or F. (2) The internal nodes are labeled by terms, and the edges issuing from an internal node v are labeled by first-order terms of the same sort as the label of v.

Terms are made out of sorts, constants, variables, and function symbols. Two kinds of sorts are distinguished: concrete and abstract. A concrete sort is equipped with finite enumerations, lists of individual constants. Concrete sorts are used to represent control signals. An abstract sort has no enumeration available. A signal of an abstract sort represents a data signal. MDGs represent and manipulate a certain subset of first order formulae, which we call Directed Formulae (DFs). DFs are used for two purposes: to represent sets (viz. sets of states as well as sets of input vectors and output vectors) and to represent relations (viz. the transition and output relations).

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 [12]. 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. 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.

HOL Theorem Prover: The HOL system is an LCF (Logic of Computable Functions) style proof system. Originally intended for hardware verification, HOL uses higher-order logic to model and verify a variety of applications in different areas; serving as a general purpose proof system. We cite for example: reasoning about security, verification of fault-tolerant computers, compiler verification, program refinement calculus, software verification, modeling, and automation theory.

HOL provides a wide range of proof commands, rewriting tools and decision procedures. The system is user programmable which allows proof tools to be developed for specific applications [13].The basic interface to the system is a Standard Meta Language (SML) interpreter. The HOL system supports two main different proof methods: forward and backward proofs in a natural-deduction style calculus.

Normal Forms:

Definition 1: A formula is in Disjunctive Normal Form (DNF) if it is a disjunction of minterms (conjunctions of literals). In other words, a DNF formula is a $sum\ of\ products$ and looks like:

$$(x_{11} \wedge x_{12} \wedge ..... \wedge x_{1n_1}) \vee (x_{21} \wedge ... \wedge x_{2n_2}) \vee ..... \vee (x_{m1} \wedge .... \wedge x_{mn_m})$$

where each x_{ij} is a literal. Literal is a variable or it's negation. In short:

$$\bigvee_i \bigwedge_j x_{ij}$$

Definition 2: A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of maxterms (disjunctions of literals). The maxterms are often referred to as clauses in this context. So, a formula in CNF looks like:

$$\bigwedge_i \bigvee_j x_{ij}$$

IV. Conversion-Verification Methodology

Figure 1 shows our conversion-verification methodology. The conversion part contains a preprocessor and a CNF converter. The verification engine contains a goal generator and a theorem prover. We start from the DF. After preprocessing, the formula goes through the CNF converter. The output CNF formula is then fed to the verification engine for automated verification of the conversion.

A. Conversion Methodology

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_{DNF_i} = ConvertoCNF(DNF_i);
7: **end for**
8: $CNF_{complete}$ = Conjunct all CNF_{DNF_i};
9: $ReturnCNF_{complete}$;

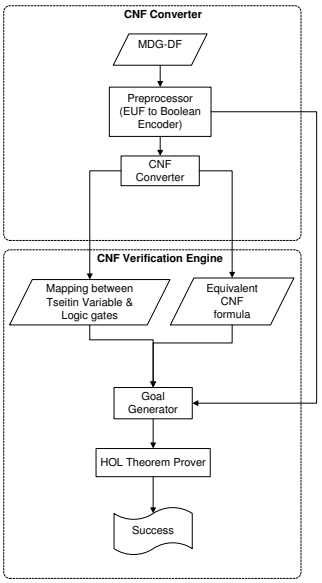

Fig. 1. Overview of the DF to CNF conversion-verification methodology

Before applying the conversion algorithm, the preprocessor inside the conversion engine eliminates the EUF applications and introduces boolean encoding. The CNF converter does the conversion for the DF and ends up with generating the essential mapping for the goal generator in verification engine.

EUF Elimination: The preprocessing part of our methodology eliminates the UF by using nested ITEs [7]. 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 $encTr(T_1)$, $encTr(T_2)$ represent boolean encoded terms and $encDF(G)$ represents propositional formula, an encoded representation of a 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's always false. So we define the encoding for such cases as :

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

At the end of the preprocessing stage the Boolean formula is fed to the CNF converter.

Linear conversion to CNF: Algorithm 1 shows the complete algorithm for the encoding and conversion. An MDG Direct Formula (DF) is the conjunction of several individual DF, where each of these DF is in DNF format:

$$DF_{complete} = \bigwedge DF_i; \tag{1}$$

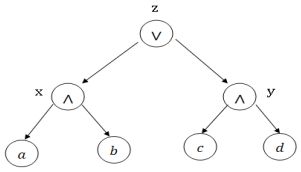

Fig. 2. Tesitin encoding to convert a propositional formula to CNF linearly

In this equation, i is the number of state variables and DF_i is a DNF. So, for our case, it is enough to get the equivalent CNF for each DF_i and conjunct them because conjunction of CNF is also a CNF.

$$DF_{CNF} = \bigwedge CNF_{DF_i}; \tag{2}$$

Linear algorithm for computing $CNF(DF)$ is well known as Tseitin [5]. In Tseitin, a new variables for every logical gate is introduced. Thus variables impose a constraint that preserve the function of that gate. Given a DNF formula

$$(a \wedge b) \vee (c \wedge d) \tag{3}$$

With Tseitin encoding, a new variable for each subexpression is introduced. In this example, 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 DF, the top most operator which is always an 'OR' gate connected with several 'AND' gates. Figure 2 illustrates the parse tree of our formula. We need to satisfy the two equivalences:

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

The overall CNF formula is the conjunction of the two equivalences written in CNF as:

$$(\neg x \vee a) \wedge (\neg x \vee b) \wedge (\neg a \vee \neg b \vee x) \bigwedge (\neg y \vee c) \wedge (\neg y \vee d) \wedge (\neg c \vee \neg d \vee y)$$

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 will be fed to the goal generator in the next step for verification.

B. Verification of Conversion

The verification part of the methodology contains a goal generator and HOL Theorem prover. The goal generator generates the goal to be proved by the HOL theorem prover. At the end, HOL provides a decision based on the inputs.

Goal Generator: The goal generator takes the CNF formula, Tseitin variable for each logic gate mapping generated by the converter and the Boolean encoded DF as input. Given the Tseitin variable for each logic gate mapping, the goal generator

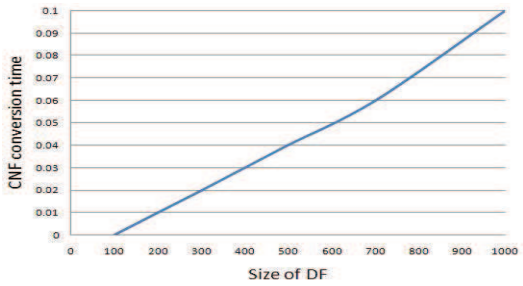

Fig. 3. DF size vs. CNF conversion time

generates the assumptions. The assumptions for the previous conversion example can be written as:

$$\begin{aligned} x &= a \wedge b \\ y &= c \wedge d \end{aligned} \tag{5}$$

At the end, the goal generator ends up with generating a complete goal to prove in HOL:

$$Assumptions \rightarrow EncodedDF \Longleftrightarrow CNFFormula$$

Call to the HOL theorem prover: As we mentioned earlier, we used HOL theorem prover to prove the goal. After generating the goal, the goal generator places a call to the HOL theorem prover. Given the input goal, the proof is conducted by applying rewriting rules. Note that the goal is generated in such a way that only one Tactic is enough to decide the goal.

V. Application and Results

We implemented our methodology in C++ and ran it on several different sized DFs, each of them containing different number of clauses and variables. All the experiments were run under Fedora Core 9 on an Intel Xeon 3.4 GHz processor with 3 GB of RAM. Table I summarizes the conversion runtime and the verification. Our program produces a 'zero' delay for the DF with less than 100 clauses. We increased both the number of clauses and the number of variables with some bigger sized DFs. Table I shows a very fast response time of 0.1 second even with large DF with 1000 clauses of 168 variables. Figure 3 shows a nearly linear behavior for our implementation. The slight deviations from linearity are caused by the interruption internal processes of the operating system.

On the other hand, the verification time in HOL increases with DF size. HOL takes a few seconds for the verification of smaller sized DFs, but suffered for bigger sized DFs while taking a longer time to prove. As we mentioned earlier, the way we constructed the goal requires only one Tactic ($DECIDE\_TAC$) for proving the goal, which is a positive side for the methodology. For a DF with 100 clauses, our conversion produced a zero delay, where as HOL took about 4.010 seconds to verify the conversion. HOL took about 14.901 and 28.021 seconds to prove the conversion of DFs with sizes 300 and 500, respectively, which was more than the expected time. The verification time increased sharply for the DF with 1000 clauses of 168 variables. But for all cases, HOL successfully finished proving the verification of the conversion.

TABLE I
CNF conversion time

DF size	No. variables	Conversion time	Verific. time
100	38	0.00	4.010
200	58	0.01	8.231
300	78	0.02	14.908
400	98	0.03	19.042
500	118	0.04	28.021
700	148	0.06	53.098
1000	168	0.10	93.118

VI. Conclusions and Future Work

We have proposed and implemented a conversion-verification approach for CNF conversion of MDG DF with conversion verification. We have also presented some experimental results to show the performance of our methodology. Our automated verification technique for the CNF conversion is a new contribution to this field of research. Researchers working with CNF conversions inspired by the Tseitin algorithm, or slight modification/enhancement of it, can easily apply this automated technique to formally verify their conversion. Our future work will apply this conversion-verification technique with algorithms other than Tseitin. The experimental results showed that with the increasing size of DFs, HOL suffers to prove the goal with larger runtime. This gives us more area to improve the performance.

References

[1] M. Ganai and A. Gupta, "Completeness in SMT-based BMC for software programs," in *DATE*, 2008, pp. 831–836.
[2] O. Strichman, "Pruning techniques for the sat-based bounded model checking problem," in *CHARME*, 2001, pp. 58–70.
[3] 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.
[4] 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.
[5] G. Tseitin, "On the complexity ofderivation in propositional calculus," *Studies in Constrained Mathematics and Mathematical Logic*, 1968.
[6] G. Ackermann, "Solvable cases of the decission problem," *North-Holand, Amsterdam*, 1954.
[7] 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.
[8] R. Bryant, D. Kroening, J. Ouaknine, S. Seshia, O. Strichman, and B. Brady, "An abstraction-based decision procedure for bit-vector arithmetic," *STTT*, vol. 11, no. 2, pp. 95–104, 2009.
[9] M. Velev, "Efficient translation of boolean formulas to CNF in formal verification of microprocessors." ASP-DAC, January 2004, pp. 310–315.
[10] B. Chambers, P. Manolios, and D. Vroon, "Faster sat solving with better cnf generation." DATE, 2009.
[11] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, August 1986.
[12] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.
[13] M. Gordon and T. Melham, Eds., *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. NY, USA: Cambridge University Press, 1993.