MDG-SAT: an automated methodology for efficient safety checking

Khaza Anuarul Hoque and
Otmane Ait Mohamed*

Department of Electrical and Computer Engineering,
Concordia University,
1455 de Maisooneve Blvd. W, Montreal, Quebec H3G 1M8, Canada
E-mail: k_hoque@ece.concordia.ca
E-mail: ait@ece.concordia.ca
*Corresponding author

Sa'ed Abed

Department of Computer Engineering,
Hashemite University,
P.O. Box 150459, Zarqa 13115, Jordan
E-mail: sabed@hu.edu.jo

Mounir Boukadoum

Department of Computer Science,
University of Quebec at Montreal,
CP. 8888, Succ. Centre-Ville, Montreal, QC H3P3P8, Canada
E-mail: boukadoum.mounir@uqam.ca

Abstract: Multiway decision graph (MDG) is a canonical representation of a subset of many-sorted first-order logic. It generalises 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. In this paper, we propose a new design verification tool integrating MDG and SAT, to check the safety of a design by invariant checking. Using MDG to encode the set of states provides a powerful mean of abstraction. We use a SAT solver to search for paths of reachable states violating the property under certain encoding constraints. In addition, 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. The formal verification of this conversion is conducted within the HOL theorem prover. Finally, we present experimental results and a case study to show the correctness and the efficiency of our proposed methodology.

Keywords: satisfiability; SAT; invariant checking; model checking; multiway decision graph; MDG; CNF conversion.

Reference to this paper should be made as follows: Hoque, K.A., Ait Mohamed, O., Abed, S. and Boukadoum, M. (2012) 'MDG-SAT: an automated methodology for efficient safety checking', *Int. J. Critical Computer-Based Systems*, Vol. 3, Nos. 1/2, pp.4–25.

Biographical notes: Khaza Anuarul Hoque received his MASc in Electrical and Computer Engineering from Concordia University, Montreal, QC, Canada, in 2011 and his BSc in Computer Science and Engineering from Ahsanullah University of Science and Technology, Dhaka, Bangladesh, in 2007. He is an Associate Member of Institute of Engineers in Bangladesh (IEB). He is a Research Engineer with the Hardware Verification Group, Concordia University, Montreal, QC, Canada. His research interest includes formal methods, model checking, SAT/SMT solvers, fault tolerant systems, reconfigurable and embedded computing.

Otmane Ait Mohamed is an Associate Professor at the ECE Department, Concordia University, Montreal, Canada. He has been working on formal verification for hardware and communication protocol since 1992. He contributed to the development of the MDG tool, a formal verification tool developed at the University of Montreal from 1996–1998. His main research areas include software model checking, assertion-based verification, automatic test generations, and FPGA-based design and verification. He also served as a reviewer for several related conferences and journals. He was the Programme Chair for the prestigious TPHOLs conference in 2008. He is a Professional Engineer at the Province of Quebec (OIQ). He is also a member of the Institute of Electrical and Electronics Engineering (IEEE), the Association for Computing Machinery (ACM), and the IEEE Computer Society.

Sa'ed Abed received his BSc in Electrical Engineering in 1994 and his MSc in Computer Engineering in 1996 both from Jordan University of Science and Technology (JUST), Jordan. In 2008, he received his PhD in Computer Engineering from Concordia University, Canada. He has previously worked as a Lecturer at the Department of Computer Science, King Faisal University in Saudi Arabia until 2003. In 2008, he joined the Department of Computer Engineering of Hashemite University, Jordan, as an Assistant Professor. His research interests include hardware specification and verification, formal methods, synthesis, computer architecture, VLSI design and automation.

Mounir Boukadoum studied Physics at the University of Algiers, Algeria, before receiving his Master's of Electrical Engineering degree from Stevens Institute of Technology, USA, in 1978 and his PhD in Electrical Engineering from the University of Houston, USA, in 1983. Since 1984, he has been with the University of Quebec at Montreal (UQAM), Canada, where he is now a Full Professor of Microelectronics Engineering. He was the Director of Microelectronics Programs at UQAM for a three-year mandate in 1994, 1998 and 2001, and Director of the PhD Programing Cognitive Informatics from 2006 to 2009. He currently chairs the Microelectronics-Prototyping Research Laboratory at UQAM, Executive Member of the Microsystems Strategic Allianceo of Quebec (ReSMiQ), and President of the Montreal chapter of the IEEE Computational Intelligence Society. He is also the co-founder of the IEEE NEWCAS conference. His research interests focus primarily on instrument design and nature-inspired computing.

This paper is a revised and expanded version of a paper entitled 'Multiway decision graphs reduction approach based on HOL theorem prover' presented at 2nd International Workshop on Verification and Evaluation of Computer and Communication Systems, Leeds, UK, 2–3 July 2008.

1 Introduction

Faulty systems (bugs in digital systems) can be very dangerous and expensive; especially for safety critical systems such as magnetic resonance imaging (MRI) machines, space shuttles, microprocessors, etc. There is a great advantage to being able to verify the correctness of such systems, whether they are hardware, software, or a combination of both. In the case of safety-critical systems, this is most obvious, but it also applies to commercially-critical systems, such as mass-produced chips, mission critical systems, etc. Formal verification methods have recently become usable by industry and there is a growing demand for professionals able to apply them, since the detection of bugs in a design by alternative techniques usually involves extra effort, time and cost. The overhead is even worse if the bug is detected late in the design cycle, thus increasing the overall cost of a project. In VLSI design, the traditional debugging technique is simulation. However, due to the increasing size and complexity of VLSI circuits, it is impossible to simulate large designs properly. To overcome these limitations, formal verification comes into play as a complement to simulation to detect errors in the design as early as possible.

Multiway decision graphs (MDGs) (Corella et al., 1997), 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 tool is a decision diagram-based verification tool, primarily designed for hardware verification that supports both equivalence and model checking. With MDG, a data value is represented by a single variable of an abstract type and data operations are represented as uninterpreted functions.

Satisfiability checking (SAT-based tools to perform several forms of model checking have achieved a lot of attention these days (Déharbe and Moreira, 1997, Abed et al., 2007), as they are less sensitive to problem size and the state explosion problem of classical BDD-based model checkers (Bryant, 1986). Expressing transition relations using the conjunctive normal form (CNF) along with SAT is an alternative to decision graphs and BDD-based approaches. Such an approach, performance-wise, is less sensitive to problem size. Moreover, it does not suffer from state space explosion. As a result, various researchers have developed methods for performing bounded model checking (BMC) (Ganai and Gupta, 2008, Strichman, 2001) using SAT. The common theme in these works is to convert the problem of interest into a SAT problem, by figuring out an appropriate propositional Boolean formula, and to utilise other non-canonical representations of state sets. These methods exploit the ability of SAT solvers to find a single satisfying solution, when it exists. In recent years, the SAT solver technology has improved significantly and a number of sophisticated packages are now available. Some of the well known state-of-the-art SAT solvers include CHAFF (Moskewicz and Madigan, 2001), GRASP (Silva and Sakallah, 1996) and SATO (Zhang, 1997). Most model checking techniques involve state set manipulations for their implementations. The state set manipulation problem can be transformed into a SAT problem. SAT solvers, thus, have the potential of enormously boosting the speed and applicability of model checking techniques.

In Abed et al. (2007), a methodology that integrates SAT and MDG model checker was presented with preliminary experimental results. A SAT solver has been used as a reduction engine to prune the transition relation of the circuits to produce a smaller one that is fed to the MDG model checker. In this work, we propose a new methodology to

build a verification tool for invariant checking where SAT is used as a verification engine. MDG as a data structure for representing transition systems or set of state, provides a powerful mean of abstraction for large models intended for model checking. A SAT solver is used to search along the decision diagram to look for bad states violating the properties specified by the user. As an alternative of using MDG as a stand alone tool for invariant checking, we explore the benefits of combining SAT and MDG in our proposed methodology. In addition, we also propose a methodology to convert MDG directed formula (DF) to CNF with automated verification of the conversion using HOL theorem prover (Hoque et al., 2010).

The paper is organised as follows: Section 2 gives a survey of some related works in this area; Section 3 gives some preliminaries on MDG, SAT and HOL. Section 4 concentrates on our main contribution integrating SAT and MDG. Experimental results and a case study using the proposed methodology are presented in Section 5. Section 6 concludes the paper providing some future research directions.

2 Related works

Related research in the area of SAT-based verification can be divided in three different categories. The first category focuses on different techniques to translate equality with uninterpreted functions (EUF) to propositional logic, the second category describes several algorithms for the conversion of propositional formula to CNF, and the last category discusses some related SAT-based verification techniques.

2.1 EUF elimination

Their exist two possible ways to eliminate EUFs – while enforcing their property of functional consistency, Ackermann constraints (Ackermann, 1954) and nested if-then-else operators (ITE) (Bryant and Velev, 2001; Lahiri et al., 2004). In Ackermann's approach, the UF is replaced with a new domain variable and the next application of UF with respect to the previous one is enforced by extending the resulting formula with constraints. Such constraints are added with the formula expressing functional consistency. Bryant and Velev presented an approach to eliminate the applications of UF with nested ITEs (Bryant and Velev, 2001). In the nested ITE scheme, the first application of the UF is still replaced by a new domain variable. However the subsequent applications are eliminated by introducing nested ITEs with new domain variables while preserving functional consistency. For our methodology, we prefer the nested ITE scheme which directly captures the functional consistency and readily exploit the maximal diversity property while Ackermann's cannot (Bryant and Velev, 2001). However, we add small modifications to match the MDG DF syntax (Hoque et al., 2010).

2.2 CNF conversion

The lack of a fast and efficient CNF generation algorithm has always been a bottleneck for CNF-based SAT solvers. Hence researchers have paid much attention to this point. Until recently (Bryant et al., 2009), most of the CNF generation algorithms used in practice were minor variations of Tseitin linear time algorithm (Tseitin, 1968). Another

CNF conversion algorithm came from Velve (2004) 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 chain, 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 an approach is good for pipelined machine verification problems, identifying certain patterns arising in formulas. Another algorithm for CNF generation is based on technological mapping (Eén et al., 2007) and is implemented in ABC tool. Their algorithm computes the mapping sequence, partial functions from and-inverter-graph (AIG) nodes in order to cut-off the graph for minimisation of the heuristics cost function. The CNF is then generated for the cuts of the nodes with respect to the final mapping by using their sum of products representation. Very recently an algorithm was presented (Chambers et al., 2009) for converting negation, ITE, Conjunction and equivalence (NICE DAGS) to CNF. The new data structure called NICE DAG subsume AIGs.

All the algorithms described above, use an intermediate representation or data structure to represent the Boolean formula (either AIG or NICE DAG). The MDG DF is itself a DAG; so an intermediate DAG representation is not required to facilitate the conversion. More interestingly none of the works mentioned above attempted an automated proof of their proposed conversion algorithm. This motivated us to build an automated tool for the verification of conversion as well.

2.3 SAT and BDD-based verification

Most of the efforts today are spent on developing SAT-based tools to perform several forms of model checking as they are less sensitive to problem size and the state explosion problem of classical BDD-based model checkers. As a result, various researchers have developed routines for performing BMC (Ganai and Gupta, 2008, Strichman, 2001) using SAT.

BDD and SAT-based verification have been of great interest to 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 (Ganai and Aziz, 2002, Abdulla et al., 2000) and model checking (Gupta et al., 2003). In Gupta et al. (2000), 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 handled in the standard way using BDDs. For checking invariant properties of the form AGp (P is globally true for along paths) of transition systems using induction (Déharbe and Moreira, 1997), Déharbe and Moreira modified a standard model checking algorithm. The set of states and image computation are expressed using BDD. Velev presented an indirect method to automatically prove the safety and liveness of for a pipelined microprocessor. The term-level simulator TLSim (Velev and Bryant, 2005), used for the

symbolic simulation of the implementation and specification and a EUFM correctness formula is produced. The decision procedure EVC (Velev and Bryant, 2005) exploits the positive equality, performs some other optimisations and converts the EUFM formula to an equivalent Boolean formula. An efficient SAT solver proves the formula to be a tautology in order for the implementation to be correct. In Sheeran et al. (2000), a safety property checking technique of finite state machines using SAT solver was presented. Their approach demonstrates the practicality of combining a SAT-solver-based safety property checking in a real design flow using induction. All the works described above relies on BDD-based state encoding. In our case, we use MDG to compactly encode the set of sates.

A model checking methodology integrating SAT and MDG, was proposed in Abed et al. (2007), while using SAT solver as a reduction engine. They used 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. Our work concentrates on SAT-based invariant checking methodology for MDG models, using SAT solver as a verification engine. Moreover, we implement SAT encoding technique (CNF conversion) for MDG DF and propose another automated methodology to formally verify the correctness of the conversion. For the conversion part, we use Tseitin (1968) approach while introducing 'fresh variables' only for AND gates and for the verification part we use the HOL theorem prover (Gordon and Melham, 1993). Implementation of SAT for model checking with MDG distinguishes our approach from others.

3 Background

In this section, we give a brief introduction to the MDG system, SAT solver and the HOL theorem prover. The intent is to familiarise the reader with the main ideas and notations used in the rest of the paper.

3.1 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 ROBDDs (Bryant, 1986) while accommodating abstract data and uninterpreted function symbols. MDG can be seen as a DAG with one root, whose leaves are labelled by formulae of the logic true (T) (Corella et al., 1997), such that:

1 every leaf node is labelled by a formula T, except if the graph *G* has a single node, which may be labelled T or F

2 the internal nodes are labelled by terms, and the edges issuing from an internal node *v* are labelled by terms of the same sort as the label of *v*.

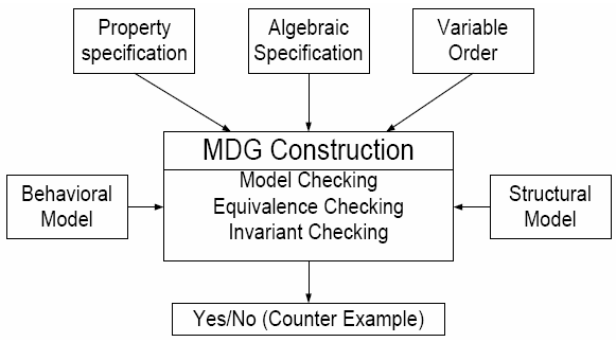
As in ordinary many-sorted first order logic (FOL), terms are made out of sorts, constants, variables, and function symbols. Two kinds of sorts are distinguished: concrete and abstract. A concrete sort is equipped with finite enumerations, lists of individual constants. Concrete sorts are used to represent 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 DFs

(Aït-Mohamed et al., 2003). DFs are used for two purposes: to represent sets (sets of states as well as sets of input vectors and output vectors) and to represent relations (the transition and output relations).

The MDG-tool (Corella et al., 1997) supports invariant checking, sequential equivalence checking, and model checking. The MDGs tool uses a Prologue-style hardware description language called the *MDG-HDL* (Corella et al., 1997). MDG-HDL supports structural, behavioural and mixed styles of coding. A structural specification is usually a netlist of components connected by signals. A behavioural description consists of a tabular representation of the transition and output relations in the form of a truth table.

The first step in the verification is to describe the design specifications and implementation are using MDG-HDL, as shown in Figure 1. An MDG-HDL algebraic specification consist of sorts, function types, and generic constants. Rewrite rules needed for interpreting function symbols are also provided. Symbol ordering (like for ROBDD) can either be specified by the user, or can be dynamically generated by the MGD tool. Symbol ordering can critically affect the size of the generated MDGs and the performance of the verification.

Figure 1 The structure of the MDGs-tool



3.2 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 T, 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.

In practice, most of the current SAT solvers are based on the Davis-Putnam algorithm (Davis and Putnam, 1960). 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.

3.3 HOL theorem prover

The HOL system is an logic of computable functions (LCF) 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, modelling, 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 (Gordon and Melham, 1993). 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.

4 Integrating SAT with MDG

We propose and implement a methodology for SAT-based invariant checking. As a solution for state explosion problem, SAT has already been integrated with MDG tool as a *reduction engine* in Abed et al. (2007). Our work is motivated by the goal – to integrate SAT with MDG for verification purpose. The following subsections provide a step-by-step description of the complete methodology using SAT as *verification engine* with MDG.

4.1 Formalisation of the problem

Given a state machine M with initial states I and transition relation Tr, we would like to check whether a property P holds for all the reachable states. The reachable states can be defined as states that can be reached by Tr transitions starting from an initial state. A system is *safe*, where all the reachable states satisfy P. We write $Tr(x, y)$ to indicate that x is related to y by a transition relation Tr. We define the state sequence to be a path through Tr.

$$path\left(s_{[0\ldots n]}\right) \equiv \bigwedge_{0\le i\le n} Tr\left(s_i, s_{i+1}\right) \tag{1}$$

In equation (1), '≡' sign means 'is defined to be' and $s_{(0\ldots n)}$ denotes a sequence of states (set of state), e.g., s_0, s_1, s_2 … s_n. A path can have a length n, if it makes n transitions. In this work, we are interested in showing that, starting from an initial state and repeated application of transition relation always leads to a state that satisfies P. We want to show that,

$$\forall i.\forall s_0 \ldots s_i.\left(I\left(s_0\right) \wedge path\left(s_{[0\ldots i]}\right) \rightarrow P\left(s_i\right)\right) \tag{2}$$

In equation (2), $i \geq 0$ and $s_i \in S$. Similarly, proving backward from bad states involves showing that, starting from a state that violates *P* and going backwards through *Tr* always leads to a non-initial state, which is

$$\forall i.\forall s_0 \ldots s_i.\neg\left(I\left(s_0\right) \leftarrow path\left(s_{[0\ldots i]}\right) \wedge \neg P\left(s_i\right)\right)$$

More symmetric view at the problem can be achieved saying that, there are no paths that start in an initial state and end in a state violating P, that is,

$$\forall i.\forall s_0 \ldots s_i.\neg\left(I\left(s_0\right) \wedge path\left(s_{[0\ldots i]}\right) \wedge \neg P\left(s_i\right)\right)$$

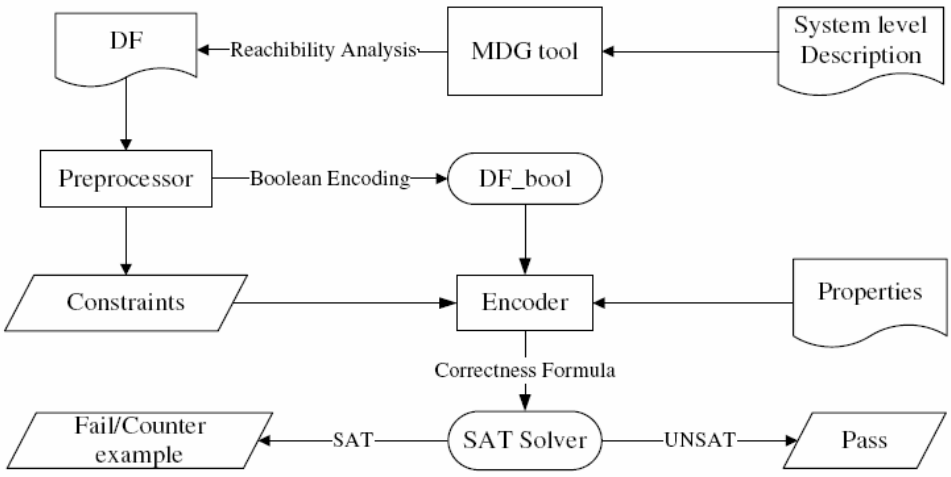
4.2 Proposed methodology

We propose a methodology (Figure 2) to formulate and verify a formula (we call this formula *correctness formula*), to check the safety of a system or design. Hence, we are interested in checking if the formula,

$$\forall i.\forall s_0 \ldots s_i.\neg\left(I\left(s_0\right) \wedge path\left(s_{[0\ldots i]}\right) \wedge \neg P\left(s_i\right)\right)$$

holds for $i = 0$, $i = 1$, $i = 2$ and so on. This is similar to checking that $I\left(s_0\right) \wedge path\left(s_{[0\ldots i]}\right) \wedge \neg P\left(s_i\right)$ is a *contradiction* for each i, for a path s_0 to s_i; i.e., $\neg\left(I\left(s_0\right) \wedge path\left(s_{[0\ldots i]}\right) \wedge \neg P\left(s_i\right)\right)$ is a *tautology*. If the property P is violated in a reachable state, then, $\exists i.I\left(s_0\right) \wedge path\left(s_{[0\ldots i]}\right) \wedge \neg P\left(s_i\right)$ is satisfiable. A *satisfiable* solution refers that there exists a path of length *i* starting from initial states that violates *P* and it can be used for tracing errors.

Figure 2 Verification methodology using MDG tool and SAT solver



Automation of this approach with MDG involves four main tasks:

- Compute the reachable states, starting from the initial state. This gives us the path by which each possible reachable states can be reached, by each transition, until all the reachable states has been visited.
- Remove *uninterpreted functions* and introduce Boolean encoding to convert the formula suitable for Boolean SAT solvers.
- Perform the *CNF* conversion of the Boolean formula using a linear algorithm to avoid exponential blow up (direct conversion from *DNF* to *CNF* has exponential blow up).
- Feed the formula to a SAT solver to check the satisfiability.

Using SAT solver with MDG tool is a new and efficient approach for invariant checking. The steps in the methodology are as follows:

1 We use MDG tool to compute the sets of reachable states for the given MDG model (behavioural/RTL) written in MDG-HDL language. Any other C/C++ implementation of MDG reachability analysis algorithm can be used instead of MDG tool. We conjunct all the sets of states which gives us the set $S_{reachable}$ consisting of all the reachable states for the system in DF format.

2 Boolean encoding is imposed by the preprocessor to reduce the DF in propositional logic. After removal of *uninterpreted functions* the encoder generates a pure Boolean formula DF_{bool} with certain encoding constraints.

3 We get formula B_{DF} after conjunction of DF_{bool} with encoding constrains and *negated* invariant property.

4 The B_{DF} is converted into CNF using Tseitin algorithm. The output is SAT encoded CNF formula in DIMACS format. At this stage we call this formula *correctness formula*.

5 The SAT encoded correctness formula is fed to a SAT solver to prove $\neg(S_{reachable} \wedge \neg P \wedge constraints)$ is a *tautology* or $(S_{reachable} \wedge \neg P \wedge constraints)$ is a *contradiction*.

Detail description of these steps is explained in the following subsections.

4.3 Using MDG for reachability analysis

The presence of uninterpreted symbols in the logic means that we must distinguish between a state machine M and its abstract description D in the logic. This is called abstract state machine, a state machine given an abstract description in terms of DFs, or equivalently MDGs, as defined in Corella et al. (1997) and Abed (2008).

The MDG tool applies the reachability algorithm (Abed et al., 2009) and gives all the possible sets of reachable states in terms of DF. For our work, we conjunct *initial state* with *frontier sets* (Abed et al., 2009) and *output relations* computed by MDG tool for each transitions to construct the complete DF, representing all the sets of reachable states,

e.g., $DF_{complete} = DF_0 \wedge DF_1 \wedge DF_2 \wedge DF_3 \ldots\ldots \wedge DF_n$. Here n is the number of transitions the reachability analysis algorithm needs to terminate. DF_0 indicates initial state and rest of each of the *DFs* is the conjunction of *frontier sets* and *outputs relations*.

4.4 Preprocessing to impose Boolean encoding

The naive structure of DF contains UF and predicates. We convert the DF formula to a Boolean formula. The preprocessor eliminates the EUF applications and introduces Boolean encoding with adequate constraints.

4.4.1 Boolean encoding for clauses with constraints

Consider a DF $(r = 0) \wedge (f = 1) \vee (r = 1) \wedge (f = 0)$. We introduce Boolean variables r_0, f_1, r_1 and f_0 respectively for abstracting the clause $(r = 0)$, $(f = 1)$, $(r = 1)$ and $(f = 0)$. Constraints are 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 $(f = 1)$. So, after reduction to propositional logic the DF looks like:

$$(r_0) \wedge (f_1) \vee (r_1) \wedge (f_0)$$

The constraints for the this example are: $r_0 \oplus r_1$ and $f_0 \oplus f_1$.

4.4.2 EUF elimination

The logic of EUF was first presented by Burch et al. (1994). The syntax of EUF logic in DF is given in Aït-Mohamed et al. (2003). Our EUF elimination approach is inspired by using nested ITEs (Bryant and Velev, 2001). We introduce domain variables replacing each function application term with a nested ITE structure that directly holds the functional consistency. 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:

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

where $encTr(T_1)$ and $encTr(T_2)$ represent Boolean encoded terms and $encDF(G)$ represents an encoded propositional 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:

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

4.5 CNF conversion for DF

The *encoder* takes the Boolean encoded DF_{bool} as input and conjunct the encoding constrains and the negated property $\neg P$ with it. In this step, the formula to be converted to CNF can be expressed as:

$$B_{DF} = DF_{bool} \wedge Constraints \wedge \neg P$$

Algorithm 1 CreateCNFFormula(DF)

1: Formula = MDG Direct Formula;
2: DF_{bool} = Replace UF's by introducing term variables;
3: Infer constraints between predicates;
4: Transform predicates to Boolean variables;
5: **for** each DNF_i in DF_{bool} **do**
6: CNF_{DNFi} = Convert to CNF CNF_{DNFi}
7: **end for**
8: $CNF_{complete}$ = Conjunct all CNF_{DNFi}
9: $ReturnCNF_{complete}$;

After CNF conversion, we call this formula a correctness formula:

$$Correctness\ formula = CNF\left(B_{DF}\right)$$

Algorithm 1 shows the complete steps for the encoding and conversion. A B_{DF} can be a single DNF formula (representing the set of states) or conjunction of several individual DF, where each of these DF is in DNF format (representing transition relations):

$$DF_{complete} \equiv \bigwedge_i DF \tag{3}$$

where i is the number of transitions 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.

$$DF_{CNF} \equiv \bigwedge_i CNF_{DF} \tag{4}$$

Linear algorithm for computing $CNF(DF)$ is well known as Tseitin (1968) algorithm. 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

$$(a \wedge b) \vee (c \wedge d) \tag{5}$$

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

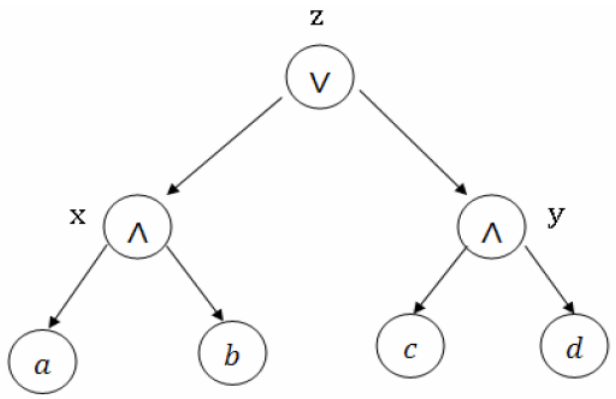
$c \wedge d$). We also introduce a new variable z to represent the top most operator. For DF, the top most operator which is always an 'OR' gate connected with several 'AND' gates. Figure 3 illustrates the parse tree of our formula. We need to satisfy the two equivalences:

$$\begin{aligned} x &\Leftrightarrow a \wedge b \\ y &\Leftrightarrow c \wedge d \end{aligned} \tag{6}$$

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

$$\begin{aligned} &(\neg x \vee a) \wedge (\neg x \vee b) \wedge (\neg a \vee \neg b \vee x) \\ &\wedge (\neg y \vee x) \wedge (\neg y \vee d) \wedge (\neg c \vee \neg d \vee y) \end{aligned}$$

Figure 3 Tesitin encoding to convert a propositional formula to CNF linearly



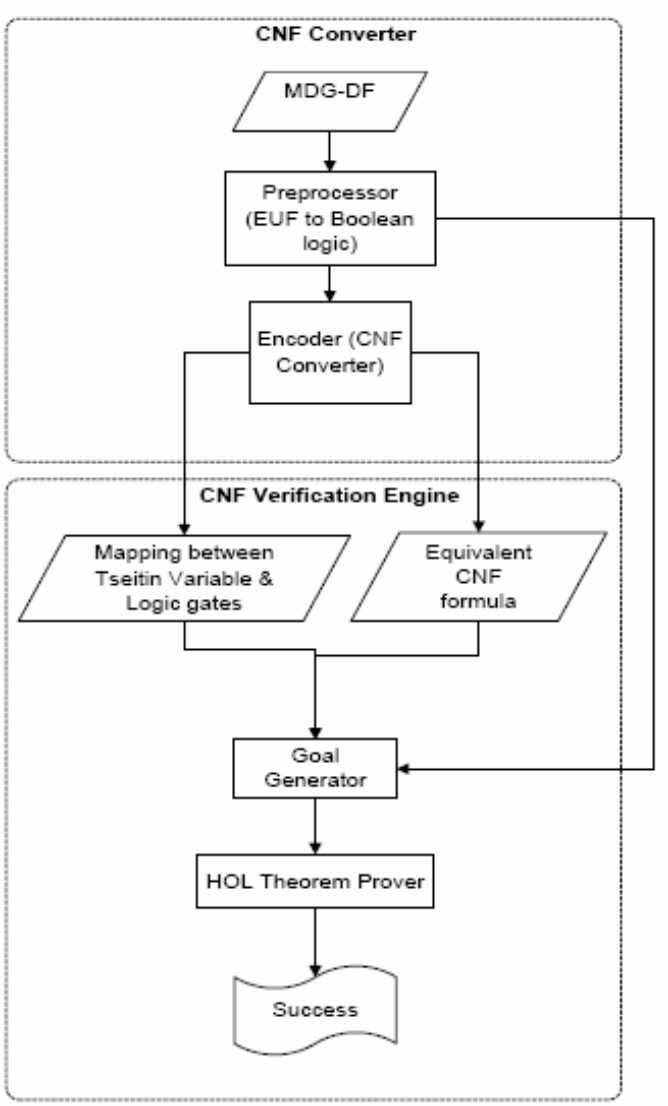
The unit clause (z) which represents the top most operator. Instead of (z) we use ($x \vee y$), which represents the same. The converter keeps track of the mapping of each Tseitin variable for each logic gate. In the example, equation (6) represents this mapping. Such mapping will be fed to the goal generator in the next step for verification of the conversion.

4.6 Verification of the conversion

Application and improvement of different linear algorithms for CNF conversion has been a major research interest for researchers (Eén et al., 2007, Chambers et al., 2009). However, we could not find any automated methodology to formally verify the conversion algorithm. This motivated us to integrate a small tool that formally verifies our CNF conversion.

Our automated conversion-verification methodology is shown in Figure 4 using HOL theorem prover demonstrate the correctness of the CNF conversion automatically. The obtained CNF formula is compared formally to the original DF using the HOL theorem prover. This enhances confidence in the whole verification process. 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.

Figure 4 Overview of the DF to CNF conversion-verification methodology



4.6.1 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 assumptions are computed by the goal generator. The assumptions for the previous conversion example (Figure 3) can be written as:

$$\begin{aligned} x &= a \wedge b \\ y &= c \wedge d \end{aligned} \tag{7}$$

Finally, the goal generator generates a goal to be proved in HOL:

$$Assumptions \Rightarrow (EncodedDF \Leftrightarrow CNFFormula)$$

4.6.2 Call to the HOL theorem prover

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.

4.7 Specification of invariant property and correctness formula

In our proposed methodology, we check the safety of a design by checking invariant property. The specification is in a commonly encountered generic form of safety properties, $M \vDash P_{init} \Rightarrow AGP_s$, where P_{init} and P_s are *instantaneous formulas* not containing temporal operators. A safety property of this form is called invariant, has the intuitive interpretation that every computation of M, which starts in a state satisfying P_{init} also satisfies P_s at all reachable states. For example, heating should be turned off when the door of a microwave-oven is open. This invariant property can be expressed in *CTL* logic as follows:

$$AG(!(door = open)\,\&\,(heating = on))$$

In order to build a correctness formula we consider $EF(\neg P)$; negation of the property. The encoder in Figure 2 conjuncts the negated property with the encoding constraints and Boolean encoded DF. The CNF representation of this formula is called correctness formula:

$$\begin{aligned} Correctness\ formula = CNF(&DF\ representing\ all\ reachable\ states \\ &\wedge\ Encoding\ constraints \wedge Negated\ invariant\ property). \end{aligned}$$

We use SAT solver to prove the correctness formula *UNSAT*, i.e., *contradictory*. For the microwave-oven example, we use the SAT solver to prove that there is a state where (*door* = *open*) and (*heating* = *on*). If no such path exist, where such state occurs, SAT solver will give an *UNSAT* decision.

4.8 Using SAT as a verification engine

SAT solver integration with MDG as a *reduction engine* is proposed in Abed et al. (2007). Our methodology uses SAT solver as a *verification engine* for MDG model. Given a correctness formula, 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. We use *MiniSAT 2.0* (Eén and Sörensson, 2003) as an efficient SAT solver. As our approach is to prove the correctness formula as a tautology, so, 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 and a theorem that the counter example implies the negated formula can be derived.

5 Experimental results and case study

In this section, we discuss the implementation details and the experimental results of our proposed methodology, to integrate SAT as a *verification engine* with MDG. Unlike other researchers, we implement not just the conversion algorithm to convert MDG DF to CNF (which is be fed to the SAT solver), but also implement an automated verification technique to formally verify the conversion. Finally, we present and analyse the experimental results obtained for both the methodologies.

5.1 Conversion-verification of DF

We implement our methodology in C++ and run it on several different sized DFs, each of which contains different number of clauses and variables. For the experiment, we consider DF with minimum 100 clauses to a maximum of 1000 clauses. Each clause containing from 38 upto 168 different variable. The experiments are performed under Fedora Core 9 on an Intel Xeon 3.4 GHz processor with 3 GB of RAM.

Table 1 CNF conversion time

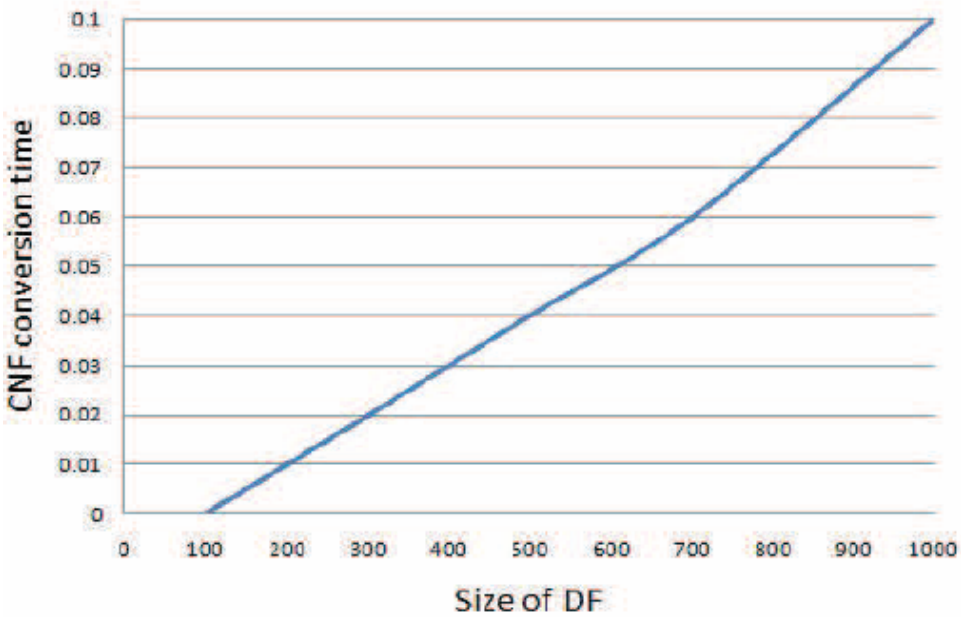
DF size	*No. of variables*	*Conversion time (sec)*	*Verification time (sec)*
100	38	Less than 0.00	4.010
200	58	0.01	8.231
300	78	0.02	14.908
400	98	0.03	19.042
500	118	0.04	28.021
700	148	0.06	53.098
1,000	168	0.10	93.118

Table 1 shows the experimental results. For the DF with 100 clauses, the conversion time was less than *zero* second. Hence, we increase both the number of clauses and the number of variables with some bigger sized DFs to check the performance. We observe, a very fast response time of 0.02 second with larger DF with 300 clauses of 78 variables. Conversion time increases to 0.04 second for the DF with 500 clause and 118 variables. The largest DF we tested with our methodology is 1000 clauses with 168 variables. Our program took only 0.1 second to compute the CNF of that DF. Figure 5 shows a nearly linear behaviour for our implementation. The slight deviation from linearity is ignored.

On the other hand, the verification time in HOL increases with the size of DF. HOL took a few seconds for the verification of smaller sized DFs. For larger sized DFs, HOL takes a much longer time to prove. As we mentioned earlier, the way we construct the goal requires only one Tactic (`DECIDE_TAC`) for proving the goal, which facilitates to completely automate the methodology. Although our conversion took less than żerośecond for a DF with 100 clauses, HOL took about 4.010 seconds to verify the same conversion. HOL took about 14.901 and 28.021 seconds to prove the conversion of DFs

with sizes 300 and 500, respectively, which is more than the expected time. The verification time increases sharply for the DF with 1,000 clauses of 168 variables. But for all cases, HOL successfully verified the conversion.

Figure 5 DF size vs. CNF conversion time (see online version for colours)



5.2 Case study: ITC

The main goal of our work is to integrate SAT with MDG for a new invariant checking methodology. We present a case study to show the performance of our approach.

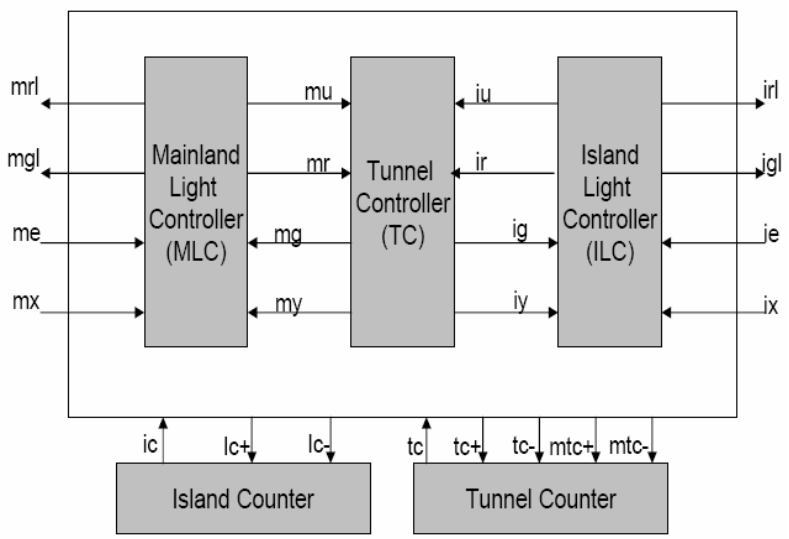
5.2.1 System description

The SAT-MDG reduction technique is demonstrated on the example of the island tunnel controller (ITC) (Abed et al., 2007), which was originally introduced by Fisler and Johnson (1995). We illustrate our SAT-MDG verification methodology on the same example. Based on the information collected by sensors installed at both ends of the tunnel, the ITC controls the traffic lights at both ends of a tunnel: there is one lane tunnel connecting the mainland to an island.

As depicted in Figure 6, the ITC specification is composed of three communication controllers and two counters. The communication controllers are: the island light controller (*ILC*), the tunnel controller (*TC*), the *mainland light controller* (MLC). The two counters are: the island counter and the tunnel counter [refer to Fisler and Johnson (1995) for the state transition diagrams of each component]. The *ILC* has four states: *green*, *entering*, *red* and *exiting*. The green and red lights on the island side are controlled by the outputs *igl* and *irl* respectively; *iu* denotes that the cars from the island side are currently occupying the tunnel, and *ir* denotes that *ILC* is requesting the tunnel. As shown in Figure 6, the input *iy* requests the *ILC* to release the control of the tunnel, and *ig* grants

control of the tunnel from the island side. For the *MLC*, a similar set of signals is defined. The requests for access issued by *ILC* and *MLC* is processed by *TC*. The number of cars currently on the island and in the tunnel is monitored by the *island counter* and the *tunnel counter*, respectively. In the case of tunnel controller, the counter *tc* is increased by 1 depending on *tc*+ or decremented by 1 depending on *tc*– unless it is already 0. The island counter functions in a similar way, except that increment and decrement depend on *ic*+ and *ic*–, respectively: one for the island lights, one for the mainland lights, and one tunnel controller to process the access requests issued by the other two controllers.

Figure 6 The island controller



5.2.2 Verification using SAT-MDG approach

Property checking is handy to verify that a specification meets the certain requirements. In Xu et al. (2004) and Abed et al. (2007), verification of ITC thorough model checking is performed. Three different invariant properties are verified for this circuit in Aït-Mohamed et al. (2004). We list below those three properties with their corresponding CTL formulas:

- Property 1: cars never travel in both directions in the tunnel at the same time.

  $$AG(!((igl = 1)\,\&\,(mgl = 1)))$$

- Property 2: the tunnel counter is never incremented by the ILC and the MLC simultaneously.

  $$AG(!((itc+ = 1)\,\&\,(mtc+ = 1)))$$

- Property 3: the island counter is never incremented and decremented at the same time. controller requests.

  $$AG(!((ic- = 1)\,\&\,(ic+ = 1)))$$

To check the correctness and efficiency of our proposed methodology, we modified the property in a way so that it fails in both: invariant checking by MDG – as a stand alone tool and SAT-MDG-based approach:

- Property 1: $AG(!((igl = 1)\&(mgl = 0)))$
- Property 2: $AG(!((itc+ = 1)\&(mtc+ = 0)))$
- Property 3: $AG(!((ic- = 1)\&(ic+ = 0)))$

The MDG tool computes the reachable states for the given MDG-HDL model of ITC tunnel controller. To ensure the correctness of the design, all the reachable states should be considered and conjuncted to build the complete $S_{reachable}$. We consider the first five reachable states and the initial state. We were able to identify the violation of property within first five reachable states (the granularity can be adjusted to identify the first violation of the property). Also as we use MDG tool for reachability analysis, so we do not include the reachable state computation time in the MDG-SAT experimental results.

To evaluate three different properties, we generate three different *correctness formulas*. An UNSAT decision from SAT solver validates the property whereas a SAT decision indicates the violation of the property. Our methodology is implemented in C++. For the experiments, Solaris 5.10 workstation was used containing a quad-core processor running at 2.5 GHz and having 6 GB of physical memory.

5.2.3 Experimental results

Table 2 summarises the results of our MDG-SAT approach. Preprocessor imposes the Boolean encoding on it with adequate constraints. For *Property 1*, the preprocessor took only 0.5 seconds and similar time was taken for the other two properties. *Correctness formula* is generated by the encoder. Encoder conjuncts the constraints and the negated property with the DF representing the reachable states. Later on, the encoder generates a correctness formula, i.e., an equivalent CNF representation. In our experiment, correctness formula generation for all the properties took same time, 0.06 seconds, because of the similar size of the property. We check the satisfiability of the correctness formula using MiniSAT 2.0 (Eén and Sörensson, 2003). MiniSAT took 0.00361 seconds to fail *Property 1*. *Property 2* and *Property 3* took 0.00538 seconds and 0.00539 to fail the property.

Table 2 Total time for SAT-MDG approach

Benchmark properties	*Preprocessing time (sec)*	*Encoding time (sec)*	*Decision time (sec)*	*Total time (sec)*
P1	0.05	0.06	0.00361	0.11361
P2	0.04	0.06	0.00538	0.10538
P3	0.04	0.06	0.00539	0.10539

We verify these properties with MDG tool and summarise the results in Table 3. As we use the reachability analysis feature of MDG tool so we do not compare the results directly. However, the table clearly shows the efficiency of our MDGSAT approach. Property-1 failed in 0.95 seconds using MDG-tool, where as, MDGSAT approach took only 0.11361 seconds. For Property-2 and Property-3, MDGtool took 0.92 and 0.91 seconds to fail them. On the other hand, MDG-SAT approach took only 0.10538 and

0.10539 second to fail Property-2 and Property-3. Implementation of MDG reachability analysis algorithm will give us a completely new tool combining both SAT and MDG. Although the performance of the tool depends on the implementation of the reachability analysis algorithm, the time taken by MDG-SAT approach small enough to support our claim for a new efficient tool. Use of MDG to represent the circuit behaviour, allows design description using a higher level of abstraction. The use of SAT solver and its fast search algorithm facilitate efficient property violation checking.

Table 3 Invariant checking time: SAT-MDG and MDG tool

Benchmark properties	*MDG time (sec)*	*MDG-SAT time (sec)*
P1	0.81	0.11361
P2	0.920	0.10538
P3	0.910	0.10539

6 Conclusions and future work

Integrating SAT with MDG is a new concept to enhance the performance of safety checking. The experimental results of ITC case study showed the efficiency of our proposed SAT-MDG tool, in terms of performance. In our work, we proposed the integration of a SAT solver with MDG as a verification engine. In addition, conversion-verification methodology for CNF conversion of MDG DF with verification of this conversion enhances the confidence in whole verification approach. The automated verification technique for the CNF conversion is a new contribution to this field of research. Researchers working with CNF conversions inspired by Tseitin algorithm, or slight modification/enhancement of Tseitin algorithm can easily apply this automated technique to formally verify their conversion.

Using SAT solver as a verification engine with MDG has a wide range of research area. The experimental results showed that with increasing the size of DF, HOL suffers to prove the goal with larger runtime. This gives us area to improvise the performance. Also, using different algorithms on MDG DF for CNF conversion and comparing the SAT solvers performance for invariant checking, can also be interesting. As a continuation of this research, future works will concentrate on application of the SAT-MDG tool on industrial circuits and comparing the results with other industrial model checkers. For the completeness of the SAT-MDG tool, we also aim to implement the MDG reachability analysis algorithm.

References

ABC *Berkeley Logic Synthesis and Verification Group*, available at http://www.eecs.berkeley.edu/alanmi/abc.

Abdulla, P.A., Bjesse, P. and Eén, N. (2000) 'Symbolic reachability analysis based on SAT solvers', *Proc. TACAS*, Vol. 1785 of LNCS, pp.411–425.

Abed, S., Aït-Mohamed, O. and Sammane, G.A. (2009) 'An abstract reachability approach by combining HOL induction and multiway decision graphs', *J. Comput. Sci. Technol.*, Vol. 24, No. 1, pp.76–95.

Abed, S., Ait Mohamed, O., Yang, Z. and Sammane, G.A. (2007) 'Integrating SAT with Multiway decision graphs for efficient model checking', in *Proc. of IEEE ICM'07*, IEEE Press, Egypt, pp.129–132.

Abed, S.R.H. (2008) 'The verification of MDG algorithms in the HOL theorem prove', PhD thesis, Concordia University, Canada.

Ackermann, G. (1954) *Solvable Cases of the Decision Problem*, North-Holand, Amsterdam.

Aït-Mohamed, O., Song, X. and Cerny, E. (2003) 'On the non-termination of MDG-based abstract state enumeration', *Theoretical Computer Science*, Vol. 300, Nos. 1–3, pp.161–179.

Aït-Mohamed, O., Song, X., Cerny, E., Tahar, S. and Zhou, Z. (2004) 'MDG-based state enumeration by retiming and circuit transformation', *Journal of Circuits, Systems, and Computers*, Vol. 13, No. 5, pp.1111–1132.

Bryant, R. (1986) 'Graph-based algorithms for Boolean function manipulation', *IEEE Transactions on Computers*, August, Vol. C-35, No. 8, pp.677–691, ISSN 0018-9340, doi: 10.1109/TC.1986.1676819.

Bryant, R., Kroening, D., Ouaknine, J., Seshia, S., Strichman, O. and Brady, B. (2009) 'An abstraction-based decision procedure for bit-vector arithmetic', *STTT*, Vol. 11, No. 2, pp.95–104.

Bryant, S.G.R. and Velev, M. (2001) '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.

Burch, J., Clarke, E., Long, D., McMillan, K. and Dill, D. (1994) 'Symbolic model checking for sequential circuit verification', *IEEE Transactions on Computer-Aided Design*, April, Vol. 13, No. 4, pp.401–424.

Chambers, B., Manolios, P. and Vroon, D. (2009) 'Faster SAT solving with better CNF generation', *DATE*.

Corella, F., Zhou, Z., Song, X., Langevin, M. and Cerny, E. (1997) 'Multiway decision graphs for automated hardware verification', in *Formal Methods in System Design*, Vol. 10, No. 1, pp.7–46.

Davis, M. and Putnam, H. (1960) 'A computing procedure for quantification theory', *J. ACM*, Vol. 7, No. 3, pp.201–215, ISSN 0004-5411, doi: http://doi.acm.org/10.1145/321033.321034.

Déharbe, D. and Moreira, A.M. (1997) 'Using induction and BDDs to model check invariants', in *Proceedings of the IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods*, pp.203–213, London, UK, Chapman & Hall, Ltd., ISBN 0-412-81330-0.

Eén, N. and Sörensson, N. (2003) 'An extensible SAT-solver', *Proc. of Theory and Applications of Satisfiability Testing, 6th International Conference (SAT'2003)*, Vol. 2919 of LNCS, pp.502–518.

Eén, N., Mishchenko, A. and Sörensson, N. (2007) 'Applying logic synthesis for speeding up SAT', *Proc. of Theory and Applications of Satisfiability Testing*, *10th International Conference (SAT 2007)*, Vol. 4501 of LNCS, pp.272–286.

Fisler, K. and Johnson, S. (1995) *Integrating Design and Verification Environments through a Logic Supporting Hardware Diagrams*, August, pp.669–674.

Ganai, M. and Gupta, A. (2008) 'SMT-based BMC for software programs', *Proceedings of the Conference on Design, Automation and Test in Europe*, Munich, Germany, pp.831–836.

Ganai, M.K. and Aziz, A. (2002) 'Improved SAT-based bounded reachability analysis', in *VLSI Design*, pp.729–734.

Gordon, M. and Melham, T. (Eds) (1993) *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambridge University Press, NY, USA, ISBN 0-521-44189-7.

Gupta, A., Ganai, M.K., Wang, C., Yang, Z. and Ashar, P. (2003) 'Learning from BDDs in SAT-based bounded model checking', *Proc. 40th Design Automat. Conf.*, Anaheim, CA, pp.824–829.

Gupta, A., Ganai, M.K., Wang, C., Yang, Z. and Ashar, P. (2000) 'SAT-based image computation with application in reachability analysis', *Proc. Formal Methods in Computer-Aided Design*, LNCS 1954, Austin, TX, USA, pp.354–371.

Hoque, K.A., Ait Mohamed, O., Abed, S. and Boukadoum, M. (2010) 'An automated SAT encoding-verification approach for efficient model checking', *Proc. of the 22nd International Conference on Microelectronics*, Cairo, Egypt, 19–22 December, pp.419–422.

Lahiri, A.G.S., Bryant, R. and Talupur, M. (2004) 'Revisiting positive equality, tools and algorithms for the construction and analysis of systems', *LNCS 2988*, Springer-Verlag, pp.1–15.

Moskewicz, M.W. and Madigan, C.F. (2001) *Chaff: Engineering an Efficient SAT Solver*, pp.530–535.

Sheeran, M., Singh, S. and Stålmarck, G. (2000) 'Checking safety properties using induction and a sat-solver', in *FMCAD '00: Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pp.108–125, London, UK, Springer-Verlag, ISBN 3-540-41219-0.

Silva, J.P.M. and Sakallah, K.A. (1996) *Grasp a New Search Algorithm for Satisfiability*, pp.220–227.

Strichman, O. (2001) 'Pruning techniques for the SAT-based bounded model checking problem', in T. Margaria and T.F. Melham (Eds.): *Lecture Notes in Computer Science*, Vol. 2144, CHARME, Springer, London, UK..

Tseitin, G. (1968) 'On the complexity of derivation in propositional calculus', *Studies in Constrained Mathematics and Mathematical Logic*.

Velev, M. (2004) *Efficient Translation of Boolean Formulas to Cnf. in Formal Verification of Microprocessors*, ASP-DAC, pp.310–315.

Velev, M.N. and Bryant, R.E. (2005) 'TLSim and EVC: a term-level symbolic simulator and an efficient decision procedure for the logic of equality with uninterpreted functions and memories', *IJES*, Vol. 1, Nos. 1/2, pp.134–149.

Xu, Y., Song, X., Cerny, E. and Mohamed, O.A. (2004) 'Model checking for a first-order temporal logic using Multiway Decision Graphs (MDGs)', *Comput. J.*, Vol. 47, No. 1, pp.71–84.

Zhang, H. (1997) 'Sato: an efficient propositional prover', in *Proceedings of the International Conference on Automated Deduction*, Springer-Verlag, pp.272–275.