MDGs Reduction Technique based on the HOL Theorem Prover

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

Otmane Ait Mohamed
Department of Electrical and Computer Engineering
Concordia University, Montreal, Canada
Email: ait@ece.concordia.ca

***Abstract*—Multiway Decision Graphs (MDGs) subsume Binary Decision Diagrams (BDDs) and extend them by a first-order formulae suitable for model checking of data path circuits. In this paper, we propose a reduction technique to improve MDGs model checking. We use a reduction platform based on combining MDGs together with the rewriting engine in the HOL theorem prover. The idea is to prune the transition relation of the circuits using pre-proved theorems and lemmas from the specification given at system level. Then, the actual proof of temporal MDG formulae will be achieved by the MDGs model checker. We support our reduction technique by experimental results executed on benchmark properties.**

I. Introduction

Model checking [1] is a fully automatic approach to verify a finite state machine against its temporal specifications. However, in today's multi-million-gate designs, the state-space of one single module is usually beyond the capability of a model checking tool. For example, Reduced Ordered Binary Decision Diagrams (ROBDDs) based model checkers suffer from the state space explosion especially for circuits with large datapath. Model reduction approaches are then used in order to reduce the model size prior to verification. Model reduction approaches such as the ones based on abstract interpretation support the reduction of a concrete system under verification to a more abstract and smaller one. However, both systems should be connected by an abstraction relation which is safe with respect to a given property. This means if the property holds for the abstract system, it holds for the concrete one as well.

Multiway Decision Graphs (MDGs) [2] have been proposed to accomplish abstract based model-checking. MDGs are a canonical representation of a certain class of many-sorted first-order logic formulae, where data values and operations are represented by abstract variables and uninterpreted functions, respectively [3]. In MDG-based verification, abstract descriptions of states machines (ASMs) are used for modeling systems. MDGs have been investigated from different angles and it culminated in a tool providing Prolog-style MDG-HDL as a modeling language. It implements different verification techniques based on the MDG structure including sequential and combinational equivalence checking, invariant checking and model checking [4]. An automatic tool is used to generate the circuit that represents the additional ASM for the property [5] and map it directly to a Directed Formula (DF).

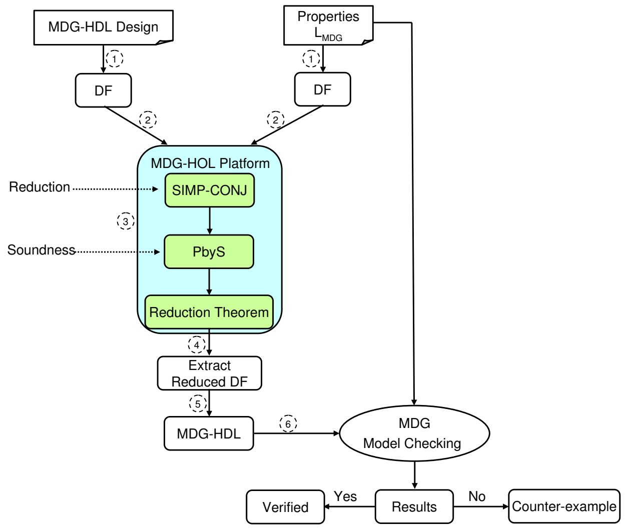

Fig. 1. Overview of the Reduction Methodology

In this paper, we propose a reduction technique based on the HOL theorem prover to improve the MDGs model checking. First, as shown in Figure 1, by starting from a behavioral design written in MDG-HDL language and a set of properties written in $\mathcal{L}_{MDG}$, we are able to generate a transition relation and a representation of the properties in terms of Directed Formulae (DFs): an alternative vision for MDGs in terms of logic and set theory [4]. Second, we feed both DFs to a reduction platform based on combining MDGs together with rewriting in the HOL theorem prover (MDG-HOL). The MDG-HOL platform theory and implementation has been described in [6]; here, we briefly review its main concept and outline its applicability. The reduction idea is to assume the correctness of some parts of the specification and to use them to prune the transition relation of the circuit in order to prove the remaining parts of the specifications. The result of the reduction platform is expressed as a theorem indicating that the behavior of the reduced DF is included in the original DF as shown in the third step. Fourth, we extract the reduced DF from the reduction theorem. Once the obtained reduced DF is translated to MDG-HDL, it will be fed to the MDG model checker along with the properties to be verified as shown in the fifth step. Finally, the actual proof of temporal

MDG formulae will be achieved by the MDG model checker. We support our reduction technique by experimental results executed on benchmark properties.

The paper is organized as follows: Section II reviews the related work in this area. Section III gives some preliminaries on MDGs and HOL systems, respectively. The main contribution of the paper describing the reduction technique is presented in Section IV. Section V discusses applications and experimental results of applying our reduction methodology. Finally, Section VI concludes the paper and gives some future research directions.

II. Related Work

Most model checking and theorem proving combined approaches try to reduce the size of the problem before solving it. Some of them exploit the modular structure of the system under consideration to decompose the initial problem into smaller sub-problems. Other ones build on abstraction techniques, and some take advantage of both compositionality [7] and abstraction. Approaches based on theorem proving are more closely related work.

From theorem proving world and from the point of view of temporal specifications, there are two types of induction that can be applied. One is induction on time, and the other is induction on the data structures. Nowadays there are a few tools that help compute inductive invariants automatically [8]. While proving properties about complex or infinite data structures, one may need to use natural or structural induction within the current state of the system. In [9] work, large systems (sometimes infinite-state) are reduced to small finite-state systems by rewriting in the ACL2 (A Computational Logic for Applicative Common Lisp) theorem prover. The reduced systems are then amenable to model checking.

From model reduction techniques side, there has been extensive research on state space reduction either for hardware systems or for software systems. For example, we cite reduction compositional reasoning [7], cone of influence [10], the symbolic representation of states and states transitions [11], state abstraction [12], partial order reduction [13], symmetry reduction [14] or combinations of these methods.

Another category of techniques are property-based reduction techniques. Such techniques target the property being checked by using it to simplify the design under verification [15]. SAT techniques are lower-level techniques that seek to improve the execution of the underlying BDD engine or SAT solver by exploiting the structure of the model and/or the property [16].

In [15] work, an MDG reduction technique was proposed. A reduced abstract transition system is derived from the original ASM using only the transition relation of the so-called property dependent state variables of the property to be verified. This reduction technique is equivalent only to a cone of influence reduction. The authors in [17] proposed a reduction technique based on SAT solver. They used a rewriting based SAT solver to produce a smaller model that is fed to the MDG model checker.

The work presented in this paper provides a novel reduction technique based on MDG operations and the rewriting engine of the HOL theorem prover to produce a sound reduced model that is fed to the MDG model checker. The work here is a continuation to the work presented in [18]. In fact, all related work do not provide any guarantee that the reduction technique is applied correctly and that the reduced model of a certain circuit is compliant to the non-reduced one. In our case, we check the compliance of the original and reduced model inside the theorem prover. According to our knowledge, this is the first time that the theorem prover is used for this objective.

III. Preliminaries

A. Multiway Decision Graphs

MDGs are graph representation of a class of quantifier-free and negation-free first-order many sorted formulae. It subsumes the class of Bryant's (ROBDDs) [19] while accommodating abstract data and uninterpreted function symbols. It can be seen as a Directed Acyclic Graph (DAG) with one root, whose leaves are labeled by formulae of the logic True (T)[2], 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 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. Concrete sort is equipped with finite enumerations, lists of individual constants. Concrete sorts are used to represent control signals. Abstract sort has no enumeration available. A signal of an abstract sort represents a data signal.

MDGs are canonical representations, which means that the MDGs structure has: a fixed node order, no duplicate edges, no redundant nodes, no isomorphic subgraphs, terms concretely reduced that have no concrete subterms other than individual constants, disjoint primary (nodes label) and secondary variables (edges label).

MDGs represent and manipulate a certain subset of first order formulae, which we call Directed Formulae (DFs). DFs can represent the transition and output relations of a state machine, as well as the set of possible initial states and the sets of states that arise during reachability analysis.

Let $\mathcal{F}$ be a set of function symbols and $\mathcal{V}$ a set of variables. We denote the set of terms freely generated from $\mathcal{F}$ and $\mathcal{V}$ by $\mathcal{T}(\mathcal{F},\mathcal{V})$. The syntax of a Directed Formula is given by the grammar below [4]. The underline is used to differentiate between the concrete and abstract variables.

$$\begin{array}{lll}
\text{Sort } \mathcal{S} & ::= & \mathcal{S} \mid \underline{\mathcal{S}} \\
\text{Abstract Sort } S & ::= & \alpha \mid \beta \mid \gamma \mid \cdots \\
\text{Concrete Sort } \underline{S} & ::= & \underline{\alpha} \mid \underline{\beta} \mid \underline{\gamma} \mid \cdots \\
\text{Generic Constant } C & ::= & a \mid b \mid c \mid \cdots \\
\text{Concrete Constant } \underline{C} & ::= & \underline{a} \mid \underline{b} \mid \underline{c} \mid \cdots \\
\text{Variable } \mathcal{X} & ::= & V \mid \underline{V} \\
\text{Abstract Variable } V & ::= & x \mid y \mid z \mid \cdots \\
\text{Concrete Variable } \underline{V} & ::= & \underline{x} \mid \underline{y} \mid \underline{z} \mid \cdots \\
\text{Directed Formulae } DF & ::= & Disj \mid \top \mid \bot \\
Disj & ::= & Conj \vee Disj \mid Conj \\
Conj & ::= & Eq \wedge Conj \mid Eq \\
Eq & ::= & \underline{A} = \underline{C} \quad (A \in \mathcal{T}(\mathcal{F}, V)) \\
 & & \mid \underline{V} = \underline{C} \\
 & & \mid V = A \quad (A \in \mathcal{T}(\mathcal{F}, \mathcal{X}))
\end{array}$$

The vocabulary consists of generic constants, concrete constants (individual), abstract variables, concrete variables and function symbols. DF are always disjunctions of conjunctions of equations or $\top$ (true) or $\bot$ (false). The conjunction *Conj* is defined to be an equation only *Eq* or a conjunction of at least two equations. Atomic formulae are the equations, generated by the clause *Eq*. The equation can be the equality of concrete terms and an individual constant, the equality of a concrete variable and an individual constant, or the equality of an abstract variable and an abstract term.

Given two disjoint sets of variables U and V, a *Directed Formula* of type $U \rightarrow V$ is a formula in Disjunctive Normal Form (DNF). Just as ROBDDs must be *reduced* and *ordered*, DFs must obey a set of well-formedness conditions given in [2].

The MDG operations and verification procedures are packaged as a tool and implemented in Prolog. The MDG-tool [20] provides facilities for invariant checking, verification of combinational circuits, sequential verification, equivalence checking of two state machines and model checking. The input language of the MDGs tool is a Prolog-style hardware description language called (*MDG-HDL*) [2], which supports structural specification, behavioral specification or a mixture of both. A structural specification is usually a netlist of components connected by signals, and a behavioral specification is given by a tabular representation of transition/output relations or a truth table.

The MDG model checking is based on an abstract implicit state enumeration. The circuit to be verified is expressed as an Abstract State Machine (ASM) and the properties to be verified are expressed by formulae in $\mathcal{L}_{MDG}$. The ASM describes digital systems under verification at a higher level of abstraction.

Definition 1: An abstract state machine M is described by a tuple:

$$D = (X, Y, Z, F_I, F_T, F_O), \text{ where}$$

X, Y and Z are sets of variables of input, state and output respectively, and F_I, F_T and F_O are the abstract descriptions of the set of initial states, the transition relation and the output relation respectively.

In MDG model checking, the properties to be verified are expressed by formulas in $\mathcal{L}_{MDG}$. $\mathcal{L}_{MDG}$ atomic formulae are Boolean constants (True and False), or equations of the form $(t_1 = t_2)$, where t_1 is an ASM variable (input, output or state variable) and t_2 is either an ASM system variable, an individual constant, an ordinary variable or a function of ordinary variables. Ordinary variables are defined to memorize the values of the system variables in the current state. The basic formulas (called $Next\_let\_formulas$) in which only the temporal operator **X** (next time) is defined as follows [4]:

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

Using the temporal operators **AG** (*always*), **AF** (*eventually*) and **AU** (*until*), the supported $\mathcal{L}_{MDG}$ properties used in this paper are in the form of $[Ante \rightarrow Cons]$, where both $Ante$ and $Cons$ are directed formulae called $antecedent$ and $consequent$, and defined by the following BNF grammar:

$$\begin{array}{lll}
\text{Property} & ::= & AG(Next\_let\_formula) \\
 & & \quad \Rightarrow F(Next\_let\_formula) \\
 & \mid & AG((Next\_let\_formula) \Rightarrow \\
 & & \quad ((Next\_let\_formula)\, U \\
 & & \quad Next\_let\_formula)))
\end{array}$$

Model checking in the MDG system is carried out by building automatically additional circuit that represents the $Next\_let\_formulas$ appearing in the property to be verified, compose it with the original circuit, and then check a simpler property on the composite machine [4].

B. The HOL Theorem Prover

The HOL system is an LCF [21] (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; without compromising reliability [22].

The basic interface to the system is a Standard Meta Language (SML) interpreter. SML is both the implementation language of the system and the Meta Language in which proofs are written. Proofs are input to the system as calls to SML functions. The HOL system supports two main different proof methods: forward and backward proofs in a natural-deduction style calculus.

Theorems in HOL are represented by values of the ML abstract type **thm**. There is no way to construct a theorem except by carrying out a proof based on the primitive inference rules and axioms. HOL has many built-in inference rules and ultimately all theorems are proved in terms of the axioms and basic inferences of the calculus. By applying a set of primitive inference rules, a theorem can be created. Once a theorem is proved, it can be used in further proofs without recomputation of its own proof. HOL also has a rudimentary library facility which enables theories to be shared. This provides a file structure and documentation format for self contained HOL developments. Many basic reasoners are given as libraries such as *mesonLib*, *bossLib*, and *simpLib*. These libraries integrate rewriting, conversion and decision procedures to free the user from performing low-level proof.

IV. The Reduction in the HOL Theorem Prover

A. The Reduction Algorithm

In our algorithm, the design transition relation Tr should satisfy n properties $\{\varphi_i\}_{1\leq i\leq n}$. The algorithm requires that Tr and $\{\varphi_i\}_{1\leq i\leq n}$ are embedded in HOL. For this purpose, we use an embedding of the MDG structure as Directed Formula (DF) in HOL [6]. We need one DF for Tr of the original design under verification (DF_D) and a set of DFs for each property (DF_{P_i}). As shown in Algorithm 1, lines 1 and 2 store the initial DFs. The variables ϕ and φ denote the reduced DF of the spec and the DF of the property, respectively. Lines 3-10 repeatedly execute a loop n times, where n still represents the number of properties. The loop consists of two main steps: simplification step and soundness checking step. Line 4 computes the simplification step by assuming one property is satisfied over the DF of the design (Section 4.3) by evaluating the conjunction operation and then applying the propagation of the property as a rewriting rule. The soundness of the reduction step is tested in line 5 by using the prune by subsumption operation (PbyS). If $(PbyS(\phi(i), DF_D) = F)$ then the behavior of the reduced model is included in the original model (Section 4.4) as shown in line 6. Otherwise, this property is removed from the properties clauses which mean that the property does not reduce the system without influencing the behavior (over reduction). This property will not be used in the reduction process. The algorithm terminates and returns the reduced DF as a theorem as shown in line 11. The reduced DF is then extracted from the theorem (left hand side) and translated to MDG files. Finally, the reduced MDG is fed with the property to the MDG model checker.

B. MDG-HOL Platform

The MDG-HOL platform consists from defining the DF in the HOL theorem prover where the many sorted first-order logic is characterized as a HOL built-in data type. Then, a HOL tactic is defined to check the well-formedness conditions of any directed formula. Finally, based on this formalization, the MDG operations are defined and the correctness proof of each operation is provided [6].

C. The Reduction Loop

Here we describe the simplification by MDG conjunction and pruning by subsumption (PbyS) operations. Then, we provide proof of the MDG correctness operation. The complete embedding and proof are available in [6].

Algorithm 1 Reduce_DF($\{DF_D\}$, $\{DF_{P_i}\}_{0<i\leq n}$)

```
1: $\phi_0 = DF_D$;
2: $\varphi_0 = True$;
3: for $i = 1$ to $n$ do
4:     $\phi_i = SIMP\_CONJ(\phi_{i-1}, DF_{P_i})$;
5:     if $(PbyS(\phi_i, DF_D) = F)$ then
6:         $\varphi_i = SIMP\_CONJ(\varphi_{i-1}, DF_{P_i})$;
7:     else
8:         $\phi_i = \phi_{i-1}$;
9:     end if
10: end for
11: $Reduced\_DF = \phi_i$;
```

The SIMP_CONJ function used in the above algorithm is obtained by applying the conjunction operation and the rewriting rules of the HOL theorem prover. The conjunction operation takes as inputs two DFs P_i, $1 \leq i \leq 2$, of types $U_i \rightarrow V_i$, and produces a DF $R =$ **Conj** $(\{P_i\}_{1\leq i\leq 2})$ such that:

$$\models R \Leftrightarrow (\bigwedge_{1\leq i\leq 2} P_i) \tag{1}$$

The pruning by subsumption (PbyS) operation is used in checking set inclusion (fixed point detection and in invariant checking); *Frontier* set simplification. The PbyS takes as inputs two DFs P and Q of types $U \rightarrow V_1$ and $U \rightarrow V_2$ respectively, where U contains only abstract variables that do not participate in the symbol ordering, and produces a DF $R =$ **PbyS** (P, Q) of type $U \rightarrow V_1$ derivable from P by *pruning* (i.e. by removing some of disjoints) such that:

$$\models R \vee (\exists E)Q \Leftrightarrow P \vee (\exists E)Q \tag{2}$$

The disjuncts that are removed from P are *subsumed* by Q, hence the name of the algorithm.

Theorem 1: Operation Correctness
ASSUME:

- df1 and df2 are well-formed DF.
- L is an order list equal to the union of df1 and df2 order lists.

Where the order list L represents the node-label order. Then, the MDG operation of df1 and df2, and HOL logical operation of df1 and df2, are equivalent.

```
⊢  ∀df1 df2. ∃L. Is_Well_Formed_DF df1 ∧
   Is_Well_Formed_DF df2 ∧ (ORD_LIST df1 df2=L)⟹
   (Logic_HOL_Opr df1 df2=MDG_Opr_HOL df1 df2 L)
```

Proof: By structural induction on df1 and df2 and rewriting rules [6]. ■

D. The Reduction Soundness

The most important of the reduction approach that it should be sound. In this context, the following definition describes the reduction soundness:

Definition 2: Reduction Soundness
Let M and M' be a two ASM models. We say that M' is soundly reduced model: $M' \preceq M$ if and only if:

- for any property P such that: $M' \models P$ then P holds in the original model M: $M \models P$.

Theorem 2: Soundness of the MDG-HOL Reduction
`ASSUME:`

- M and M' be a two ASM models such that: $M' \preceq M$.
- DF_D and $Reduced\_DF$ be the respective transition relation in terms of DF.

Then the reduction approach is sound if:

$$PbyS(Reduced\_DF, DF\_D) = F$$

Proof: Since $Reduced\_DF$ represents the transition relation of the model M' which should be included in M, the $Reduced\_DF$ formula can not be a T or F (see definition 1).

By applying the definition of PbyS as shown in (2), the result R is derivable from $Reduced\_DF$ by pruning. Hence $\models R \Rightarrow Reduced\_DF$. And, from (2), it follows tautologically that $\models$Reduced_DF$\wedge\neg(\exists E)DF_D \Rightarrow R$. Thus we have

$$\models (Reduced\_DF \wedge \neg(\exists E)DF_D \Rightarrow R) \wedge (R \Rightarrow Reduced\_DF)$$

which holds if and only if R is F, then it follows tautologically from (2) that $\models Reduced\_DF \Rightarrow (\exists E)DF_D$. We have thus proved the soundness the reduction. ∎

V. Application and Results

The MDG tool has been demonstrated on the example of the Island Tunnel Controller (*ITC*) in [23], which was originally introduced by Fisler and Johnson [24]. The *ITC* controls the traffic lights at both ends of a tunnel based on the information collected by sensors installed at both ends of the tunnel: there is one lane tunnel connecting the mainland to an island. At each end of the tunnel, there is a traffic light as depicted in Figure 2. There are four sensors for detecting the presence of cars: one at tunnel entrance on the island side (*ie*), one at tunnel exit on the island side (*ix*), one at tunnel entrance on the mainland side (*me*), and one at tunnel exit on the mainland side (*mx*). The specification of *ITC* is composed of three communication controllers and two counters: The Island Light Controller (*ILC*), the Tunnel Controller (*TC*), the Mainland Light Controller (*MLC*), the Island Counter and the Tunnel Counter.

We would like to establish that the *ITC* has at least the following properties:

- **P1**: The cars at the island entrance will enterally pass the tunnel:
  `AG( (is=red & ix=1) ==> (igl=0) );`
- **P2**: The green light of *ILC* must be off if there is a car exiting the tunnel:
  `AG( (ie=1 & ix=1) ==> (F(igl=1)) );`
- **P3**: The island will eventually release the control right of the tunnel controller requests:
  `AG( (is=green) ==> (F(is=red)) );`
  under the fairness constraint
  `AG( !(is=green) ==> (iy=0) );`

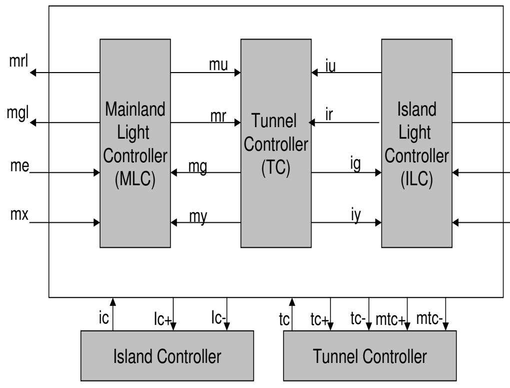

Fig. 2. The Island Tunnel Controller Structure

- **P4**: The tunnel counter keeps the old value if ordered to increment and decrement at the same time:
  ```
  AG( (tc_min=1 & tc_plus=1) ==>
      LET (v=tc) IN X(tc=v));
  ```
- **P5**: The green light of MLC must be on if there is no request to yield control of the tunnel and the number of cars on the island are less than n:
  `AG((my=0 & lessn_ic=1) -> (mgl = 1));`

We take the same case study and we consider the ITC with its properties as a benchmark in order to measure the performance of our approach. Table I compares the CPU time measured in seconds, the number of nodes, and the memory measured in MB that are used in building the reduced machine and checking the property, run on a Sun Enterprise server with Solaris 5.7 OS and 6.0 GB memory.

The best gain in performance is obtained with property P2 where the time is reduced by 2175 times the original one and the memory is reduced by a factor of 220 times. The worst case is the property P3 where the reduction in time and memory is 5 times the original one. In the case of property P2 the assumptions include two global signals that causes a huge reduction on the complete transition relation which really was much more efficient. For P3, it was only one local signal in the assumption of the property which results a small impact on the global transition relation.

These differences show the sensitivity of the reduction technique to the property verified. Despite these fluctuations, the average of the gain in performance is a factor of 7.4 which is considered a good result in the case of model checking approaches.

We compare these results with [25], [17] reduction techniques. Our proposed technique has an average gain in performance of factor 7.4, while the gain obtained in [25], [17] is 4 and 2, respectively. The reason for better gain regarding to the fact that we are using simplification by conjunction operation and the rewriting engine of the HOL theorem prover which is higher level of abstraction. In [25], [17], the reduction is based on the assumptions and the functionality of the property tested which might needs several runs (case splitting). However, the limitation we have reside in the limited integration of the

TABLE I
COMPARING MDG MODEL CHECKING RESULTS WITHOUT & WITH REDUCTION

Prop	*Without Reduction*			*With Reduction*		
	Time	**Mem**	**Nodes**	**Time**	**Mem**	**Nodes**
P1	121.12	100.91	226070	17.05	10.92	26787
P2	65.26	48.6	123080	0.03	0.22	44
P3	81.73	58.2	171060	15.84	10.24	22278
P4	67.07	48.8	123080	9.28	6.88	20125
P5	66.08	49.4	123080	12.09	8.01	20278
Avg	80.25	61.18	153274	10.86	7.254	17902

HOL theorem prover in industrial flow comparing to symbolic simulation and SAT in [25], [17].

VI. CONCLUSION AND FUTURE WORK

We have proposed a reduction technique for MDG model checking that uses an MDG-HOL integrated platform. Consequently, we have formalized the main operations on which the MDG verification techniques are based. We have used the specification of the design described at high level language along with properties to extract a reduced model. The originality of our reduction technique comes from applying MDG operations and rewriting engine based HOL theorem prover to prune the transition relation of the design. Then, the reduced model is proved sound inside the theorem prover.

We support our reduction technique by experimental results executed on benchmark properties. The obtained performance is promising as it has been shown in the experimental results.

The reduction strategy in our case was limited to property re-use and to the propagation of antecedents of the properties. We believe that our approach can be used to express more reduction techniques without any loss of generality, without much loss of automation, and more importantly, automatic soundness checking. Also, this approach can be easily generalized to any other verification tools such as commercial model checker or SAT solvers. In this case, we need to build a parser to translate the reduced model to the language of the verification tool. Thus, we can replace the MDG model-checker by any of these commercial tools.

REFERENCES

[1] T. Kropf, *Introduction to Formal Hardware Verification*. Springer Verlag, 1999.

[2] 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.

[3] S. Tahar, X. Song, E. Cerny, , Z. Zhou, M. Langevin, and O. Ait Mohamed, "Modelling and automatic verification of the fairisle ATM switch fabric using mdgs." *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 18, no. 17, pp. 955 – 972, July 1999.

[4] Y. Xu, X. Song, E. Cerny, and O. A. Mohamed, "Model checking for a first-order temporal logic using multiway decision graphs (MDGs)," *The Computer Journal*, vol. 47, no. 1, pp. 71–84, 2004.

[5] K. Hussain, "Abstract property verifier based on multiway decision graphs. Master thesis, Concordia University. Montreal, Canada," Master thesis, Concordia University. Montreal, Quebec, Canada, 2007.

[6] S. Abed, O. A. Mohamed, and G. A. Sammane, "An abstract reachability approach by combining hol induction and multiway decision graphs," *Journal of Computer Science and Technology*, vol. 24, no. 1, pp. 76–95, 2009.

[7] K. McMillan, "Verification of infinite state systems by compositional model checking," in *CHARME '99: Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. London, UK: Springer-Verlag, 1999, pp. 219–234.

[8] S. Bensalem, Y. Lakhnech, and S. Owre, "InVeSt: A tool for the verification of invariants," in *Computer-Aided Verification, CAV '98*, A. J. Hu and M. Y. Vardi, Eds., vol. 1427. Vancouver, Canada: Springer-Verlag, 1998, pp. 505–510.

[9] P. Manolios, "Mechanical verification of reactive systems," Ph.D. dissertation, The University of Texas at Austin, 2001.

[10] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction of approximation of fixpoints," in *4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, USA, 1977, pp. 238–252.

[11] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," in *7th. Symposium of Logics in Computer Science*. Santa-Cruz, California: IEEE Computer Scienty Press, 1992, pp. 394–406.

[12] K. Namjoshi and R. Kurshan, "Syntactic program transformations for automatic abstraction," in *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 2000, pp. 435–449.

[13] A. Valmari, "A stubborn attack on state explosion," in *CAV '90: Proceedings of the 2nd International Workshop on Computer Aided Verification*. London, UK: Springer-Verlag, 1991, pp. 156–165.

[14] F. Emerson and A. Sistla, "Symmetry and model checking," *Formal Methods in System Design*, vol. 9, no. 1/2, pp. 105–131, August 1996.

[15] J. Hou and E. Cerny, "Model reductions in MDG-based model checking," in *13th Annual IEEE International ASIC/SOC Conference*. IEEE, 2000, pp. 347–351.

[16] K. McMillan, "Interpolation and SAT-based model checking," in *Proceedings of the International Conference On Computer Aided Verification*, Warren A., Hunt Jr. and Fabio Somenzi, Ed., vol. 2725. Springer Verlag, 2003, pp. 1–13.

[17] S. Abed, O. A. Mohamed, and G. A. Sammane, "Integrating SAT with multiway decision graphs for efficient model checking," in *Proc. of IEEE ICM'07*. Egypt: IEEE Press, 2007.

[18] S. Abed and O. Ait Mohamed and G. Al Sammane, "Multiway decision graphs reduction approach based on the HOL theorem prover," in *Second International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2008)*. Leeds, UK: The British Computer Society, July 2008.

[19] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, August 1986.

[20] Z. Zhou and N. Boulerice, *MDGs Tools (V1.0) User's Manual*. D'IRO, University of Montreal, June 1996.

[21] M. Gordon, R. Milner, and C. Wadsworth, *Edinburgh LCF*, ser. Lecture Notes in Computer Science. Springer, 1979, vol. 78.

[22] M. J. C. Gordon and T. F. Melham, Eds., *Introduction to HOL: a theorem proving environment for higher order logic*. New York, NY, USA: Cambridge University Press, 1993.

[23] Z. Zhou, X. Song, S. Tahar, E. Cerny, F. Corella, and M. Langevin, "Formal verification of the island tunnel controller using multiway decision graphs," in *FMCAD '96: Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*. London, UK: Springer-Verlag, 1996, pp. 233–247.

[24] K. Fisler and K. Johnson, "Integrating design and verification environments through a logic supporting hardware daigrams," in *Proc. IFIP Conference on Hardware Description Languages and their Applications (CHDL'95)*, Chiba,Japan, August 1995.

[25] G. A. Sammane, S. Abed, and O. A. Mohamed, "High level reduction technique for multiway decision graphs based model checking," in *First International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2007)*. Algiers, Algeria: The British Computer Society, May 2007.