Multiway Decision Graphs Reduction Approach based on the HOL Theorem Prover

Sa’ed Abed, Otmane Ait Mohamed and Ghiath Al Sammane
Department of Electrical and Computer Engineering
Concordia University, Montreal, Canada
{s_abed, ait, sammane}@ece.concordia.ca

Abstract

Multiway Decision Graphs (MDGs) subsume Binary Decision Diagrams (BDDs) by representing formulae which are suitable for first-order model checking able to handle large datapath circuits. In this paper, we propose a reduction approach to improve MDGs model checking. We use a reduction platform based on combining MDGs with the rewriting engine of the HOL theorem prover. The idea is to prune the transition relation of the design 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.

Keywords: Multiway Decision Graphs, the HOL Theorem Prover, Reduction

1. INTRODUCTION

Model checking [18] 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, ROBDDs-based model checkers suffer from the state space explosion specially for circuits with large datapath. Model reduction approaches are then used in order to reduce the module size prior to verification. Model reduction approaches such as the ones based on abstract interpretation allows to reduce 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) [9] 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 [24]. In MDG-based verification, abstract description of states machines (ASMs) are used for modeling systems. An automatic tool is used to generate the circuit that represents the additional ASM for the property [15]. 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 [26].

In this paper, we propose a reduction approach based on the HOL theorem prover to improve the MDGs model checking. We use a reduction platform based on combining MDG with the rewriting engine of the HOL theorem prover (MDG-HOL) [4]. Both of the transition relation of the design and the properties represented in terms of Directed Formulae (DFs) are fed to our reduction platform. The reduction idea is to assume the correctness of some elements of the specification and to use them to prune the transition relation of the circuit in order to prove the rest of the specifications. For example, by applying the antecedent variables from the properties we obtain a reduced MDG. The obtained MDG will be fed to the MDG model checker with the properties. Then, the actual proof of temporal MDG formulae will be achieved by the MDG model checker.

The paper is organized as follows: Section 2 gives some preliminaries on MDGs and HOL systems, respectively. The main contribution of the paper describing the reduction approach is presented in Section 3. Section 4 reviews the related work in this area. Finally, Section 5 concludes the paper and gives some future research directions.
2. PRELIMINARIES

2.1. Multiway Decision Graphs

MDGs subsume the class of Bryant's (ROBDD) [7] 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)[9], 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.

Figure 1 shows two MDGs example G0 and G1. In G0, X is a variable of the concrete sort [0, 2, 3], while in G1, X is a variable of abstract sort; a, b and f(c) are abstract terms.

Figure 1: Example of Multiway Decision Graphs Structure

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 $F$ be a set of function symbols and $V$ a set of variables. We denote the set of terms freely generated from $F$ and $V$ by $T(F, V)$. The syntax of a Directed Formula is given by the grammar below [26]. The underline is used to differentiate between the concrete and abstract variables.

\[
\begin{align*}
\text{Sort } S & ::= S_0 | S \\
\text{Abstract Sort } S & ::= \alpha | \beta | \gamma | \cdots \\
\text{Concrete Sort } S & ::= \alpha | \beta | \gamma | \cdots \\
\text{Generic Constant } C & ::= a | b | c | \cdots \\
\text{Concrete Constant } C & ::= a | b | c | \cdots \\
\text{Variable } X & ::= V | V \\
\text{Abstract Variable } V & ::= x | y | z | \cdots \\
\text{Concrete Variable } V & ::= x | y | z | \cdots \\
\text{Directed Formulae } DF & ::= \text{Disj} | \top | \bot \\
\text{Disj} & ::= \text{Conj} \lor \text{Disj} | \text{Conj} \\
\text{Conj} & ::= A \land \text{Conj} | A \\
\text{Eq} & ::= A = C (A \in T(F, V)) \\
& | V = C \\
& | V = A (A \in T(F, X))
\end{align*}
\]
Conj is defined to be an equation only term. the equality of a concrete variable and an individual constant, or the equality of an abstract variable and an abstract 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 Formulae 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 [9].

Intuitively, in a DF of type $U \rightarrow V$, the $U$ variables play the role of independent variables (secondary variables), the $V$ variables play the role of dependent variables (primary variables), and the disjuncts enumerate possible cases. DFs are used for two purposes: to represent sets (viz. sets of states as well as sets of input vectors and output vectors) and to represent relations (viz. the transition and output relations).

The MDG operations and verification procedures are packaged as a tool and implemented in Prolog [8]. The MDG-tool [27] 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) [9], 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 $L_{MDG}$ [26]. 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

$$D = (X, Y, Z, F_I, F_T, F_O),$$

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 $L_{MDG}$. $L_{MDG}$ 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 [26]:

- Each atomic formula is a Next Let formulas;
- If $p, q$ are Next Let formulas, then so are: $\neg p$ (not $p$), $p \& q$ (and $p$ and $q$), $p \lor q$ (p or $q$), $p \rightarrow q$ (p implies $q$), $Xp$ (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 $L_{MDG}$ 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:

$$\text{Property} ::= AG(\text{Next Let formula}) \Rightarrow F(\text{Next Let formula})$$

$$AG((\text{Next Let formula}) \Rightarrow ((\text{Next Let formula}) \cup \text{Next Let formula}))$$

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 [26].
2.2. The HOL Theorem Prover

The HOL system is an LCF [13] (Logic of Computable Functions) style proof system. Originally intended for hardware verification, HOL uses higher-order logic to model and verify 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 and algorithms verification, modeling, and automation theory [12].

HOL provides a wide range of proof commands, including rewriting tools and decision procedures. The system is user-programmable which allows proof tools to be developed for specific applications; without compromising reliability [14].

The HOL syntax contains syntactic categories of types and terms whose elements are intended to denote respectively certain sets and elements of sets. The types of HOL logic are expressions that denote sets while the terms are expressions that denote elements of the sets denoted by types.

The basic interface to the system is a Standard Meta Language (SML) interpreter. SML [21] is both the implementation language of the system and the Meta Language in which proofs are written. 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 enable 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 simplLib. These libraries integrate rewriting, conversion and decision procedures to free the user from performing low-level proof.

3. THE REDUCTION IN THE HOL THEOREM PROVER

3.1. The Reduction Algorithm

In our algorithm, the design transition relation \( T_r \) should satisfy \( n \) properties \( \{ \phi_i \}_{1 \leq i \leq n} \). The algorithm requires that \( T_r \) and \( \{ \phi_i \}_{1 \leq i \leq n, j \neq i} \) are embedded in HOL. For this purpose, we use an embedding of the MDG structure as Directed Formulae (DF) in HOL [4]. We need one DF for \( T_r \) of the original design under verification (\( DF_D \)) and a set of DFs for each property (\( DF_P_i \)), where \( DF_P_i \) represents the antecedent of the property. As shown in Algorithm 1, lines 1 and 2 store the initial DFs. The variables \( \phi \) and \( \phi_i \) denote the reduced DF of the spec and the DF of the property, respectively. Lines 3-10 repeatedly execute a loop \( n \) times, where \( n \) 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 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) and thus, the property is added to guarantee (correct) as shown in line 6. Otherwise, this property is removed from the properties clauses which means 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.

3.2. MDG-HOL Platform

The MDG-HOL platform consists firstly 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. Secondly, a HOL tactic is defined to check the well-formedness conditions of any directed formula [2]. This step is important to guarantee the canonical representation of MDG in terms of DF. Finally, based on this formalization, the MDG operations are defined and the correctness proof of each operation is provided [4].
Algorithm 1 \textsc{Reduce}_{DF}([DF_D], \{DF_{P_i}\}_{0 \leq i \leq n})

1: \(\phi_0 = DF_D;\)
2: \(\varphi_0 = True;\)
3: \(\textbf{for} \ i = 1 \ \textbf{to} \ n \ \textbf{do}\)
   4: \(\phi_i = \textsc{Simp}_\text{Conj}(\phi_{i-1}, DF_{P_i});\)
   5: \(\text{if} \ (\text{PbyS}(\phi_i, DF_D) = F) \ \text{then}\)
      6: \(\varphi_i = \textsc{Simp}_\text{Conj}(\varphi_{i-1}, DF_{P_i});\)
   7: \(\text{else}\)
      8: \(\phi_i = \phi_{i-1};\)
   9: \(\textbf{end if}\)
10: \(\textbf{end for}\)
11: Reduced_{DF} = \phi_i;

3.3. The Reduction Loop

Here we describe the simplification by MDG conjunction and pruning by subsumption (PbyS) operations. The complete embedding and proof are available in [1].

The \textsc{Simp}_\text{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 is performed over MDG structures; examples of MDG conjunction is shown in Figure 2. In F1, the two top variables of P and Q are the same concrete variables or cross-terms. In F2, the top variable of P is a concrete variable or cross-term A, and order(A) < order(top variable of Q). Finally in F3, P and Q have different primary abstract variables, and order(A) < order(top variable of Q). However, it is not a well formed MDG (canonical), and \(T_i\) must be substituted for A having secondary occurrences in Q.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{The conjunction operation}
\end{figure}

In terms of DF’s, 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 \(R = \text{Conj}(\{P_i\}_{1 \leq i \leq 2})\) of type \((\bigcup_{1 \leq i \leq 2} U_i) \setminus (\bigcup_{1 \leq i \leq 2} V_i) \rightarrow (\bigcup_{1 \leq i \leq 2} V_i)\) such that:

\[
\models R \Leftrightarrow (\bigwedge_{1 \leq i \leq 2} P_i)
\]

The pruning by subsumption operation is used in checking set inclusion (fixed point detection and in invariant checking). In Figure 3 an example is shown: \(Q = \text{PbyS}(NS, R)\). In NS, the path of the concrete variable \(c\), labeled by
0, is subsumed by $R$. Thus, it will be removed from $Q$. The other path, labeled by 1, cannot be removed and appears in $Q$. Informally, it removes all the paths of $N$ from $Q$.

![Diagram showing the PbyS operation](image)

**FIGURE 3**: The PbyS operation

In terms of DF’s, 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 $R = \text{PbyS}(P, Q)$ of type $U \rightarrow V_1$ derivable from $P$ by pruning (i.e., by removing some of disjoints) such that:

$$\models R \lor (\exists E)Q \Leftrightarrow P \lor (\exists E)Q \quad (2)$$

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

We can then view $R$ as approximating the logical difference of $P$ and $(\exists E)Q$, this approximation may lead to non-termination problem (see [26] for more details). In general, there is no DF logically equivalent to $P \land \neg (\exists E)Q$.

### 3.4. 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 $M$: $M : M \models P$.

**Theorem 1**: Soundness of the MDG-HOL Reduction

**ASSUME**:  
1. $M$ and $M'$ be a two ASM models such that: $M' \preceq M$.  
2. $DF_D$ and $Reduced_DF$ be the respective transition relation in terms of DF of $M$ and $M'$.

Then the reduction approach is sound if:

$$\text{PbyS}(Reduced_DF, DF_D) = F$$

**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 2).

The only interesting case is when $Reduced_DF$ is not $T$ or $F$. 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 $\models Reduced_DF \land \neg (\exists E)DF_D \Rightarrow R$. Thus we have:

$$\models (Reduced_DF \land \neg (\exists E)DF_D \Rightarrow R) \land (R \Rightarrow Reduced_DF)$$

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.  

---

*International Workshop on Verification and Evaluation of Computer and Communication Systems*  
*VECoS 2008*
Illustrative Example

Just as an example, consider a bridge for cars traveling in only one direction with two operations enter and exit. Each end of the bridge has a sensor to detect a car. SensorA for entrance and SensorB for exit. Each sensor has a separate controller and only one sensor is allowed to work at the same time. The number of cars allowed on the bridge is max\textsubscript{car}. If SensorA detects a car and the number of cars is less than max\textsubscript{car}, then the entrance door will open and the counter will increment otherwise the car will stay. If SensorB detects a car then the exit door will open and the counter will decrement. Initially, there is no cars on the bridge, the entrance and exit doors are closed and the counter is set to zero. The behavior level of the bridge is defined as:

**Inputs:** \{reset, SensorA, SensorB, n\}  
**Outputs:** \{door\_enter, door\_exit\}

**Enter Process**

if reset=1 then

\((\text{count}'=0)\land (\text{door\_enter}=0)\land (\text{door\_exit}=0)\)

else if \(\text{SensorA}=1\) and \(\text{less\_fun}(n)=1\) then

\((\text{door\_enter}=1)\land (\text{count}'=\text{inc\_fun(count)})\)

else if \(\text{SensorA}=1\) and \(\text{less\_fun}(n)=0\) then

\((\text{door\_enter}=0)\land (\text{count}'=\text{count})\)

else

\((\text{door\_enter}=0)\)

end if

**Exit Process**

if \(\text{SensorB}=1\) and \(\text{SensorA}=0\) then

\((\text{door\_exit}=1)\land (\text{count}'=\text{dec\_Fun(count)})\)

else

\((\text{door\_exit}=0)\)

end if

The transition relations for the entrance operation in terms of DF \(\text{Enter\_Tr\_DF}\) is written as follows:

\[
\begin{align*}
&[\text{reset} = 1 \land \text{count}' = 0 \land \text{door\_enter} = 0 \land \text{door\_exit} = 0] \\
&[\text{reset} = 0 \land \text{SensorA} = 1 \land \text{less\_fun(count, max\_car)} = 1)\land
\text{door\_enter} = 1 \land \text{count}' = \text{inc\_fun(count)})] \\
&[\text{reset} = 0 \land \text{SensorA} = 1 \land \text{less\_fun(count, max\_car)} = 0)\land
\text{door\_enter} = 0 \land \text{count}' = \text{count})] \\
&[\text{reset} = 0 \land \text{SensorA} = 0 \land \text{door\_enter} = 0]
\end{align*}
\]

and for the exit operation \(\text{Exit\_Tr\_DF}\):

\[
\begin{align*}
&[\text{SensorB} = 1 \land \text{SensorA} = 0 \land \text{door\_exit} = 1 \land \text{count}' = \text{dec\_Fun(count)})] \\
&[\text{SensorB} = 0 \land \text{SensorA} = 0 \land \text{door\_exit} = 0]
\end{align*}
\]

The transition relation for the design is the conjunction of both transition relations of the enter and the exit operations, in DF that will be written as follows:

\[
\text{Bridge\_Tr\_DF} = \text{Enter\_Tr\_DF} \land \text{Exit\_Tr\_DF}
\]

If we want to verify a property about the entrance operation then the property will include \((\text{SensorA} = 1)\). Then by applying Algorithm 1, the transition relation of the exit operation will be eliminated from the design DF. Also, the
reset input is eliminated by expressing this as a property: activating it once and then remains deactivated since the interesting properties does not include reset.

Let's have the following property on the entrance operation with the condition that the number of cars on the bridge is equal to max_car, then the property can be expressed as: \( AG((SensorA = 1 \& less\_fun(count, max\_car) = 0) \Rightarrow X(door\_enter = 0)) \) and its antecedent will be \( P_{enter\_ant} = (SensorA = 1 \& less\_fun(count, max\_car) = 0) \). In this case, the new reduced DF will be:

\[
Reduced\_Bridge\_TrDF = ((SensorA = 1) \& (less\_fun(count, max\_car) = 0) \& (door\_enter = 0) \& (count' = count))
\]

which is expressed as theorem such that:

\[
\vdash Reduced\_Bridge\_TrDF = SIMP\_CONJ(Bridge\_TrDF, P_{enter\_ant})
\]

Finally, it is easy to extract the reduced DF from the left hand side of the theorem and map it with the property to the MDGs model checker. Thus, an important reduction is gained on the resulting transition relation which improves the performance of the model checker in terms of memory and CPU time.

4. RELATED WORK

Model reduction techniques attempt to reduce the size of the model to be checked. There has been extensive research on state space reduction either for hardware systems and software systems. For example, we cite reduction compositional reasoning [19], cone of influence [6, 10], the symbolic representation of states and states transitions [23], state abstraction [20], partial order reduction [25], symmetry reduction [11] 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 [16]. 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 [17].

The reduction techniques mentioned above come mostly from the model checking world. 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 [5]. 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 [16] 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.

In [22], the authors proposed a technique to construct a reduced MDG model for circuits described at system level in VHDL. The simplified model is obtained using a high level symbolic simulator called TheoSim, and by running an appropriate symbolic simulation patterns. Later, the authors in [3] proposed another 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 approach 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. In fact, all related work do not provide any guarantee that the reduction approach 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.

5. CONCLUSION AND FUTURE WORK

We have proposed a reduction approach 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 approach comes from applying MDG operations and rewriting engine of the HOL theorem prover to prune the transition relation of the design. Then, the reduced model is proved sound inside the HOL theorem prover.

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 automatism, and more importantly, automatic soundness checking. Indeed, case studies and experiments are required to check our approach and compare the results with similar work. 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


