MDG Tools for the Verification of RTL Designs

K.D. Anon[1] N. Boulerice[1] E. Cerny[1] F. Corella[2]
M. Langevin[3] X. Song[1] S. Tahar[1] Y. Xu[1] Z. Zhou[1]

[1] D'IRO, Université de Montréal, Canada, {cerny,song,zhouz}@iro.umontreal.ca
[2] Hewlett-Packard Company, USA, fcorella@hprpcd.rose.hp.com
[3] GMD-SET, Germany, langevin@gmd.de

1 Introduction

Although ROBDDs [1, 2] have proved to be a powerful tool for automated hardware verification, they require a Boolean representation of the circuit. Since the size of an ROBDD grows, sometimes exponentially, with the number of Boolean variables, ROBDD-based verification cannot be directly applied to circuits with complex datapaths.

We have recently proposed a new class of decision graphs, called *Multiway Decision Graphs* (MDGs) [3], that comprises, but is much broader than, the class of ROBDDs. The underlying logic of MDGs is a subset of many-sorted first-order logic with a distinction between *concrete* and *abstract* sorts. A concrete sort has an enumeration while an abstract sort does not. Hence a data value can be represented by a single variable of abstract sort, rather than by a vector of Boolean variables, and a data operation can be viewed as a black box and represented by an *uninterpreted* function symbol. MDGs are thus much more compact than ROBDDs for designs containing a datapath, and this greatly increases the range of applications.

We have developed a collection of MDG tools that include implementations of the basic MDG operators and verification procedures for RTL designs.

2 The Structure of MDG Tools

The current MDG tools (V1.0) are logically organized into five modules in three layers as shown in Figure 1.

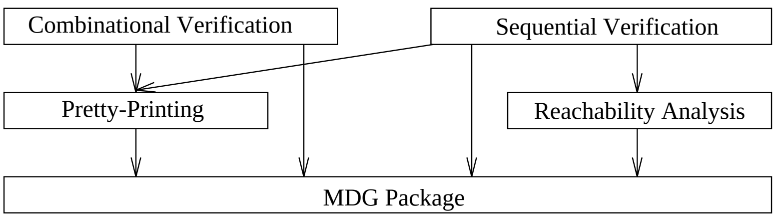

Fig. 1. The structure of MDG tools

The lowest layer is the *MDG package* module. It provides a set of MDG operators and a number of utilities, which are the basic building blocks for MDG-based applications.

In the middle layer, there are two modules: (i) the *printing module* provides various pretty-printing procedures; (ii) the *reachability analysis module* implements the reachability analysis algorithm for abstract state machines (ASMs) [4]. It also contains a counterexample facility which generates explanations when the invariant being verified is violated.

Currently, there are two modules in the application layer. The *combinational verification module* provides the equivalence checking of combinational circuits. The *sequential verification module* have two applications: (i) invariant checking and (ii) equivalence checking of two state machines.

As a prototype, the MDG tools have been implemented in Prolog and currently run under Quintus Prolog Version 3.2.

3 The MDG Package

Like an ROBDD package, the MDG package contains the basic procedures for developing MDG-based applications. Here we briefly review the MDG operators and the utilities. For more details see [3, 6].

- *Disjunction* (Disj): performs disjunction for a set of MDGs.
- *Relational product* (RelP): performs conjunction, abstraction by existential quantification and renaming operations in one traversal of graphs for a set of MDGs. It is used for image computation.
- *Pruning-by-subsumption* (PbyS): takes as input two MDGs P and Q, and produces an MDG that approximates $P \wedge \neg Q$ by removing zero or more paths in MDG P which are subsumed by MDG Q. When P and Q represents sets of states, if all paths in P are removed, then $P \subseteq Q$. PbyS is used for subsumption checking and frontier-set simplification.
- *Rewriting* (Rew): rewrites the first-order terms appearing in an MDG as edges and node labels according to a conditional term rewriting system.

Computed-tables, implemented as hash tables, are used to reduce the complexity of the algorithms by exploiting the structure sharing of MDGs.

The major utilities provided with the package are term assembly and graph assembly, which construct terms and graphs using reduction tables, also implemented as hash tables.

4 Applications to Hardware Verification

Abstract State Machines. The MDG tools are intended for Abstract State Machines (ASM) verification [4, 3] rather than Finite State Machine (FSM) verification. They can be used for FSMs as well, but they are less efficient than ROBDDs for this purpose, due in part to the space requirements of our current Prolog implementation.

An *abstract description* of a state machine, called *abstract state machine* (ASM) [4], is obtained by letting some data input, state or output variables be of an abstract sort, and the datapath operations be uninterpreted function symbols. Just as ROBDDs for encoding FSMs, MDGs are used to compactly represent sets of (abstract) states and transition/output relations for ASMs.

The MDG tools accept as hardware description a Prolog-style HDL, MDG–HDL, which allows the use of abstract variables for representing data signals. The MDG–HDL description is then compiled into the ASM model in internal MDG data structures.

MDG–HDL supports structural descriptions, behavioral ASM descriptions, or a mixture of structural and behavioral descriptions. A structural description is usually a netlist of components (predefined in MDG–HDL) connected by signals. A behavioral description is given by a *tabular* representation of the transition/output relation or truth table. Currently, we are implementing a translator for a subset of VHDL to MDG–HDL.

Like ROBDDs, MDGs require a fixed order of node labels along all paths. This order is supplied by the user. Unlike ROBDDs where all variables are Boolean, in MDGs every variable/signal must be assigned an appropriate sort, and a type definition must be provided for all functions. If needed, rewrite rules may be used to partially interpret the otherwise uninterpreted function symbols.

Reachability Analysis. The reachability analysis for an ASM is based on a technique called *abstract implicit enumeration* [3] which is analogous to the implicit enumeration [2] used for FSMs. It verifies whether an invariant holds in all reachable states of the ASM.

The image computation is based on the RelP operator. It uses transition relation partitioning and early quantification heuristics. The special operator PbyS is used for multiple purposes: frontier-set simplification, detection of termination and invariant checking.

When the invariant is violated at some stage of the reachability analysis, a counterexample facility gives a sequence of input-state pairs leading from the initial state to the faulty behavior. This is very helpful for identifying design errors.

Applications. Invariant checking is the direct application of reachability analysis. A variation is the equivalence checking of two ASMs. We make a product machine for two ASMs by putting them together and feeding them the same inputs, then perform reachability analysis for the product machine and check an invariant stating the equivalence of the corresponding outputs.

In addition, we have a procedure for checking the combinational equivalence of circuits having the same input, output and state variables (if any). For each circuit, we derive MDGs relating each output and state variable to the inputs and state variables. These MDGs collectively represent the output and transition relations. Then, using the canonicity property of MDGs, we simply check that corresponding MDGs for the two circuits have the same MDG identifier.

Due to the use of many-sorted logic and uninterpreted function symbols, the specification and the implementation to be compared must be couched in terms of the same set of uninterpreted function symbols and sort assignments. This restriction, however, does not rule out using rewrite rules to exploit partial meaning of the uninterpreted function symbols.

The reachability analysis procedure may not terminate in general. However, for a class of interesting problems, the non-termination problem can be avoided by state generalization [3] and/or by using the term rewriting facility.

5 Conclusions and Future Work

We presented MDG tools which can reason at the abstract level and are thus suitable for RTL design verification. The contribution of MDGs as a representation and computation tool, beyond the use of abstract types, is that they open the way to the development of new techniques for the verification of circuit and system designs at higher levels of abstraction, making it possible to lift some of the ROBDD techniques that have been successful at the Boolean level.

We are currently in the course of developing a model checking algorithm for a restricted first-order temporal logic. This additional feature would allow us to perform the verification for temporal properties.

The MDG home page [6] contains a complete list of MDG references, including algorithms, case studies and the most recent work on the verification of an ATM switch fabric [5].

References

1. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.
2. J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.
3. F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway decision graphs for automated hardware verification. To appear in the journal *Formal Methods in System Design*. Available as IBM technical report RC19676.
4. F. Corella, M. Langevin, E. Cerny, Z. Zhou and X. Song. State enumeration with abstract descriptions of state machines. In *Proc. of IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (Charme'95).* October, 1995, Frankfurt, Germany.
5. S. Tahar, Z. Zhou, X. Song, E. Cerny and M. Langevin. Formal Verification of an ATM Switch Fabric using Multiway Decision Graphs. In *IEEE Proc. of Sixth Great Lakes Symposium on VLSI.* Iowa, USA, March, 1996.
6. The home page of the multi-site MDG verification group: http://www.iro.umontreal.ca/labs/lasso/research/mdgverif/mdgverif_eng.html.