MDG Tools for the Verification of RTL Designs*

Z. Zhou[†] X. Song[†] F. Corella[‡] M. Langevin[§] E. Cerny[†] S. Tahar[†]

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

http://www.iro.umontreal.ca/labs/lasso/research /mdgverif/mdgverif_eng.html

Abstract

Multiway Decision Graphs (MDGs) [3] have been proposed as a new kind of decision graphs to represent Register-Transfer (RT) level hardware designs. MDG-based verification techniques have been developed for combinational and sequential designs. In our presentation, we shall describe the structure of the MDG tools we implemented and demonstrate their applications on a number of benchmarks including the Fairisle 4X4 ATM switch fabric. The MDG Tools comprise an MDG package that contains basic MDG operators and an application layer that implements the verification algorithms.

1 Introduction

Although ROBDDs [1, 2, 4] have proven 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 data path, and this greatly increases the range of applications.

The contribution of MDGs as a representation and computation tool 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.

In [3], we described a basic set of MDG operators. Using those operators, we developed a reachability analysis algorithm for abstract machines that is called *abstract implicit enumeration* [3]. By reachability analysis we mean checking that an invariant holds in all the reachable states of a state machine. Its direct application is the verification of safety properties, including the observational equivalence of two state machines. We also have developed a procedure for combinational circuit verification based on the canonicity of MDGs [3].

*This research is partially supported by NSERC Canada Grant No. OGP0138506 and No. STR0167079, and FCAR No. 94-NC-1087.

The MDG tools are a collection of the above operators and applications. As a prototype, they are all implemented in Quintus Prolog Version 3.2. In this presentation, we show the structure and the functions of the MDG tools.

2 The structure of MDG Tools

The current MDG Tools are logically organized into five modules in three layers as shown in Figure 1.

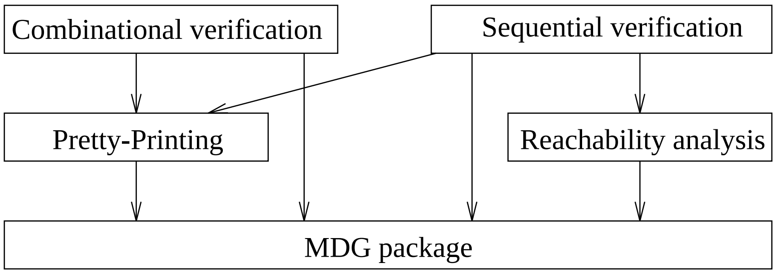

Figure 1: The structure of MDG Tools

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

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 based on abstract implicit enumeration. It also contains a counterexample facility which can provide explanations (a sequence of input-state pairs leading to the faulty behavior) when the invariant being verified is violated.

Currently, there are two application modules. The combinational verification module implements the equivalence checking of input-output relations of two combinational circuits using the canonicity property of MDGs [3]. The sequential verification module implements (i) safety property checking and (ii) equivalence checking of two state machines.

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] and the MDG manuals [8, 9].

- Disjunction (Disj): the disjunction operator takes as input a set of MDGs $\{P_i\}_{1\leq i\leq n}$ and produces an MDG which is logically equivalent to the disjunction of $\{P_i\}_{1\leq i\leq n}$.

- Relational product (RelP): relational product is a combination of conjunction, abstraction by existential quantification and renaming operations. Each of them can be obtained as a special case of the relational product operator. The operator takes as input a set of MDGs $\{P_i\}_{1\leq i\leq n}$, a set V of variables, and a renaming substitution η; it produces an MDG which is equivalent to the conjunction of $\{P_i\}_{1\leq i\leq n}$, with variables in V existentially abstracted and with renaming of the other variables according to η. Relational product is used for image computation in state space exploration.

- Pruning-by-subsumption (PbyS): it 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 can be used for subsumption checking and frontier set simplification.

- Rewriting (Rew): the rewriting operator takes as input an MDG P and a conditional term rewriting system H, and produces an MDG obtained by rewriting the first-order terms appearing in P as edge and node labels, and reordering as needed.

Among various other utilities, graph assembly and term assembly are the two major ones. They exploit structure sharing by using reduction tables (implemented as hash tables), for graphs and terms. Whenever a graph (term) is to be constructed, we first look it up in the reduction table. If the graph (term) already exists, then a pointer to it is returned, thus avoiding the creation of a redundant isomorphic structure.

4 Applications to hardware verification

Hardware descriptions

We call *implementation* an RTL structural description which is a network of components connected by signals, and *specification* a behavioral description using, equivalently, ITE (If-Then-Else) or CASE formulas or a tabular representation.

We defined a Prolog-style HDL, called MDG–HDL, as an input language for our applications. Eventually we will compile a subset of VHDL into MDG–HDL. The MDG–HDL has a predefined set of most commonly used logic gates, multiplexers and registers. The multiplexers and registers can be modeled at the abstract level, i.e., they can take abstract terms as inputs. The MDG–HDL also contains a general *tabular* constructor which is able to define arbitrary logic relations. It is similar to a truth table but it allows first-order terms in rows.

Like ROBDDs, the MDGs require a fixed node ordering. At the current stage, the node ordering has to be given by the user explicitly. Unlike ROBDDs where all variables are Boolean, every variable used in MDGs need to be assigned an appropriate sort and a type definition must be provided for all functions. If needed, rewrite rules have to be provided to partially interpret the otherwise uninterpreted function symbols.

Hardware verification

The sequential verification module provides a safety property checking procedure which is also applicable to machine equivalence checking. Property checking is used to verify that the design satisfies certain requirements. As a case study [7], we verified several useful safety properties of the Fairisle ATM switch fabric used in the Cambridge ATM network [5].

For the equivalence checking of two machines at the same time scale, we make a product machine for two circuits by putting them together and feeding same inputs, then we perform reachability analysis for the product machine and check an invariant stating the equivalence of the corresponding outputs. For machines at different time scales, it is possible to synchronize them first if they have cyclic behavior [8]. Once they are synchronized, we can perform reachability analysis on the product machine as usual. Using the above techniques, we verified one implementation against another implementation, and one implementation against the instruction set architecture, for the Tamarack-3 and the CompAs processors [8].

When an invariant fails at some stage of the analysis, our 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. Using the counterexample facility we were able to find and correct several errors in a published design [6]. We were also able to find artificially introduced design errors in the Tamarack-3 processor and the ATM switch fabric.

For combinational verification, we derive an MDG representing its input-output relation for each output. Then using the canonicity property of MDGs, we check if the two corresponding MDGs are the same by comparing their unique identifiers.

Scope of application

In optimized designs, for example, property checking may fail because of the non-interpreted nature of the function symbols in our basic model. For instance, a multiplier can be bypassed when one of the operands is 1. To verify such designs, we need to use the fact that, e.g., 1 is the unit element of multiplication. One solution is to use rewrite rules. For our example, the algebraic equations $1 \times x = x$ and $x \times 1 = x$ could be used as rewrite rules. Our MDG system incorporates a term rewriting procedure which can be activated when required.

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

5 Conclusions and future work

We presented MDG tools which can reason at an abstract level. We also discussed several applications of combinational and sequential verification of RTL designs. 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 OBDD techniques that have been successful at the Boolean level.

The MDG Tools will be made public in the near future. Currently, we are preparing the necessary documentation.

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] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems.* Springer Verlag, 1989. Lecture Notes in Computer Science, vol. 407.

[5] P. Curzon. The Formal Verification of the Fairisle ATM Switching Element. Technical Reports No. 328 & No. 329, University of Cambridge, Computer Laboratory, March 1994.

[6] K. Fisler and S. Johnson. Integrating design and Verification Environments Through A Logic Supporting Hardware Diagrams. In *Proc. of IFIP Conference on Hardware Description Languages and their Applications (CHDL'95).* August 1995, Chiba, Japan.

[7] 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.* March, 1996, Iowa, USA.

[8] Z. Zhou, X. Song, F. Corella, E. Cerny, and M. Langevin. Description and verification of RTL designs using Multiway Decision Graphs. In *Proc. of IFIP Conference on Hardware Description Languages and their Applications (CHDL'95).* August 1995, Chiba, Japan.

[9] Z. Zhou and N. Bouleric. MDG Tools (V1.0) User's Manual. University of Montreal, 1995.

[10] Z. Zhou. MDG Tools (V1.0) Developer's Manual. University of Montreal, 1995.