Term Ordering Problem on MDG

Yi Feng, Eduard Cerny[1]
Département IRO
Université de Montréal
Montréal, QC, H3C 3J7 Canada
1-514-739 2638
{feng, cerny}@iro.umontreal.ca

ABSTRACT

As an efficient representation of Extended Finite State Machines, Multiway Decision Graphs (MDG) are suitable for automatic hardware verification of Register Transfer Level (RTL) designs. However, in some cases, MDG-based verification suffers from the state explosion problem. Some of cases are caused by the standard order used by MDG to order cross-terms that have the same top-level function symbol. These terms usually label decision nodes and must be ordered. We call this kind of state explosion the *standard term ordering problem*. A solution based on function renaming and cross-term rewriting is proposed in this paper. Experimental results show that this solution can solve the problem completely and thus increase the range of circuits that can be verified by MDG.

Categories and Subject Descriptors

B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids – *verification.*

General Terms

Verification.

Keywords

Multiway Decision Graphs, Standard term ordering, Variable Ordering, First-order terms, Function symbols, Function renaming, Term rewriting.

1. INTRODUCTION

Several approaches to formal verification have been proposed over the years. One of the most important approaches that are beginning to be used in industry is model checking. Model checking works on a finite state model of the system to be verified, and the logical specification of the desired behavior of the system model [4]. Finite state models of concurrent systems sometimes grow exponentially as the number of components of the system increases. This is known as the *state explosion problem* in automatic verification. The main challenge in model checking is to deal with this problem.

The most promising approach to the state explosion problem has been the application of ROBDD (Reduced Ordered Binary Decision Diagrams) to the representation of state graphs [1]. ROBDDs can encode sets of states as well as transition and output relations, and perform an implicit enumeration of the state space thus making it possible to verify finite state machines with a large number of states. However, ROBDD-based verification methods suffer from the drawback that they require a Boolean representation of the circuit. Therefore, ROBDD-based verification cannot be directly applied to circuits with complex and large datapaths.

To overcome this limitation, the verification group at the University of Montreal has proposed a new class of decision graphs called Multiway Decision Graphs (MDGs) [2]. MDGs efficiently represent a class of formulas of a many-sorted first-order logic with a distinction of abstract and concrete sorts. In an MDG, a data signal is represented by a single variable of abstract sort rather than by a vector of Boolean variables, and a data operation is represented by an uninterpreted function symbol. MDGs compactly encode sets of (abstract) states and transition/output relations for abstract description of state machines. The implicit enumeration technique is lifted from the Boolean level to the abstract level and referred to as implicit abstract enumeration. MDGs are thus much more compact than ROBDDs for circuits having complex and large datapaths. This increases the range of circuits that can be verified.

MDG-based verification still suffers from the problem of state explosion when handling realistic circuits. To reduce the effects of this problem, one of the most important approaches is to select a good variable order like ROBDD. We explored automatic static and dynamic variable ordering algorithms on MDG [3]. In this paper, we focus on another ordering problem, the so-called standard term order.

The paper is organized as follows: Section 2 describes the basic concepts of MDG. Variable ordering algorithms on MDG are summarized in Section 3. Section 4 explains the standard term ordering problem and a solution is presented in Section 5. Experimental results are shown in Section 6.

[1] The work has been carrid out at the Universite de Montreal. Dr. Cerny is now with Synopsys in Marlborough, MA.

2. MULTIWAY DECISION GRAPHS

The formal logic underlying MDGs is a many-sorted first-order logic, augmented with the distinction between abstract sorts and concrete sorts [2]. This distinction is motivated by the natural division of datapath and control circuitry in RTL designs.

Concrete sorts have enumerations that are sets of individual constants, while abstract sorts do not. Variables of concrete sorts are used for representing control signals, and variables of abstract sorts are used for representing datapath signals. Data operations are represented by uninterpreted function symbols. An n-ary function symbol has a type $\alpha_1 \times \ldots \times \alpha_n \rightarrow \alpha_{n+1}$, where $\alpha_1 \ldots \alpha_{n+1}$ are sorts.

The distinction between abstract and concrete sorts leads to a distinction between three kinds of function symbols. Let f be a function symbol of type $\alpha_1 \times \ldots \times \alpha_n \rightarrow \alpha_{n+1}$. If α_{n+1} is an abstract sort then f is an abstract function symbol. If all the $\alpha_1 \ldots \alpha_{n+1}$ are concrete, f is a concrete function symbol. If α_{n+1} is concrete while at least one of $\alpha_1 \ldots \alpha_n$ is abstract, then we refer to f as a cross-operator. While abstract function symbols are used to denote data operations, cross-operators are useful for modeling feedback signals from the datapath to the control circuitry. Both abstract function symbols and cross-operators are uninterpreted, i.e., their intended interpretation is not specified.

A multiway decision graph (MDG) is a finite directed acyclic graph G where the leaf nodes are labeled by formulas, the internal nodes are labeled by terms, and the edges issuing from an internal node N are labeled by terms of the same sort as the label of N. Such a graph represents a formula defined inductively as follows: (i) if G consists of a single leaf node labeled by a formula P, then G represents P; (ii) if G has a root node labeled A with edges labeled $B_1 \ldots B_n$ leading to subgraphs $G_1' \ldots G_n'$, and if each G_i' represents a formula P_i, then G represents the formula $\bigvee_{1 \leq i \leq n}((A = B_i) \wedge P_i)$.

We refer to an occurrence of a variable in a term that labels an edge or in a cross-term that labels a node as a secondary occurrence, while an occurrence of a variable as the label of a node is a primary occurrence. The primary variables (resp. secondary variables) of a graph G are those that have primary (resp. secondary) occurrences in G.

MDG must obey a set of conditions to keep it well-formed [2]. Just as Bryant's ROBDD must be reduced and ordered, MDG must also be reduced and ordered. The concept of ordering in MDG concerns two orders: the standard term order and the custom symbol order. The standard term order is a lexicographical order of all the terms of the logic. The custom symbol order is a total order of a set of symbols that includes the concrete variables, abstract variables, and some but not necessarily all of the operators. It is selected specifically for each model, like in ROBDD. The custom symbol order is independent of the standard term order. Along each path of an MDG, the variables and the cross-operators of the cross-terms that label the nodes must appear in a custom symbol order, and cross-terms with the same cross-operator must appear in the standard term order; there must be no repeated labels. The labels of the edges issuing from a given node must appear in the standard term order, without repetitions. In the next section, we summarize custom variable ordering algorithms on MDG.

3. VARIABLE ORDERING ON MDG

Variable ordering on MDG is more complicated than ROBDDs because of the presence of the first order terms in MDGs. Three constraints on ordering are introduced by MDG's well-formedness conditions:

1. If an abstract variable *a* appears as a secondary variable in an edge label of node *b*, then $a < b$.
2. If a variable *a* appears as a secondary variable in a cross-term having cross-operator *f*, then $a < f$.
3. The present and next state variables must be in a corresponding order. If the present state variables are in the order $a < b < c$, then the corresponding next state variables should be in the order $a' < b' < c'$.

The static variable ordering algorithm generates a variable order before an MDG is built and the order is chosen using information about the circuit topology. Our algorithm is based on several heuristic rules we explored for MDG. Experiments show that the algorithm can give a better order than manually generated ones in most cases, especially in circuits with a large number of variables when manual ordering becomes difficult due to the multiplicity of constraints that must be satisfied [3].

The dynamic reordering algorithm minimizes the size of the MDGs during the verification process and allows a verification task to finish when the task may run out of memory with a fixed order. We implemented in MDG the sifting algorithm that has been successful in ROBDD [6]. Since sifting is too time consuming, we developed a reordering algorithm that considers symmetry and state group sifting to improve the performance of the original algorithm. Our algorithm can greatly reduce the size of MDGs and it is particular useful for verification of circuits where lack of memory is a problem [3].

In the next section, we define the standard term ordering problem that cannot be solved by the usual ordering strategies.

4. STANDARD TERM ORDERING PROBLEM

Although selecting a good static variable order and changing it dynamically as the application proceeds can minimize the size of the MDG, there are cases where under any order the size of the MDG is exponential with the number of function instances. These cases may be caused by the standard order adopted in MDG to order cross-terms with the same cross-operator. We call this kind of state explosion the *standard term ordering problem.* Before we identify this problem, we first explain the internal representation of Terms, which plays an important role in shaping the problem.

4.1 Internal Representation of Terms and TermID Assignment

A term in an MDG is defined as follows [8]:

- A (individual or generic) constant is a term.
- A (concrete or abstract) variable is a term.
- If $a_1, \ldots, a_n$ are terms and *f* is a function symbol, then $f(a_1, \ldots, a_n)$ is a term, which is also referred to as a compound term.

In an MDG, there may be many very large compound terms. Thus, it is important to have compound terms share some common components. In order to achieve term sharing, MDG system assigns a unique identifier (TermID) to each term. The TermID for a constant or a variable is the constant or the variable itself. The TermID for a compound term is a number obtained by hashing the function symbol using a Quintus Prolog built-in hash function. It is assigned to a compound term when the term is first generated in the verification process.

For example, in Figure 1, root node n_1 and its labeling term $eq(x_1, x_2)$ is generated first. The TermID of $eq(x_1, x_2)$ is computed by hashing the function symbol *eq*. Suppose hash(*eq*) = *k*, the TermID of $eq(x_1, x_2)$ would be *k*. Since the MDG is built in a depth-first order, the left-hand branch is constructed first. Thus, nodes n_2 and its labeling term $eq(y_1, y_2)$ are generated next. To compute the TermID for $eq(y_1, y_2)$, we hash again the function symbol *eq*. In Quintus Prolog, hashing collision is resolved by linear probing, i.e., search the hash table sequentially, starting from the original hash location [9]. Suppose $k + 1$, $k + 2$ … are all available, hashing *eq* again would get $k + 1$ and thus the TermID of $eq(y_1, y_2)$ would be $k + 1$. Nodes n_3 and n_4 are generated next in the depth-first order. Similarly, the TermIDs of their labeling compound terms $eq(y_3, y_4)$ and $eq(z_1, z_2)$ are $k + 2$ and $k + 3$ if there are no other new cross-terms in the branches of nodes n_2 and n_3. From this example, it is easy to see that the TermIDs of cross-terms with the same operator are assigned sequentially according to the order of the terms constructed in the MDG. The first generated one has the smallest TermID (original hash location). The rest has TermIDs sequentially starting from the original location.

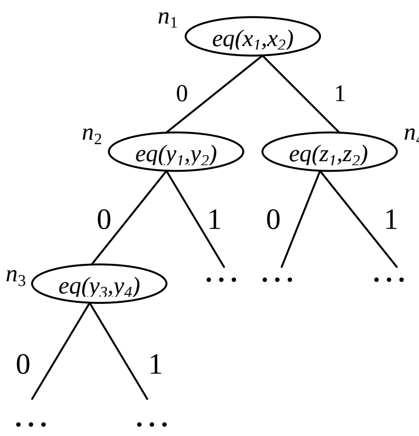


Figure 1. TermID assignment for cross-terms with the same function symbol

The internal representation of a compound term is *term*(*Function_symbol, TermID, Subterms*). For instance, $f(a_i, f(a_j))$ is represented as *term*(*f*, *Id*1, [*term*(*f*, *Id*2, [])]). Standard order of the cross-terms with the same cross-operators is the lexicographical order of their internal representation. Since those cross-terms have the same function symbol (no matter if they have Subterms), their order is thus decided by the value of their TermIDs. In other words, the standard order of cross-terms with the same function symbol is the same order as when the terms are first generated during the construction of the MDG.

4.2 Identification of the Standard Term Ordering Problem

Consider the circuit shown in Figure 2. Function *f* is a cross-function with two arguments c_i and a_i, where c_i is a Boolean variable and a_i is an abstract variable.

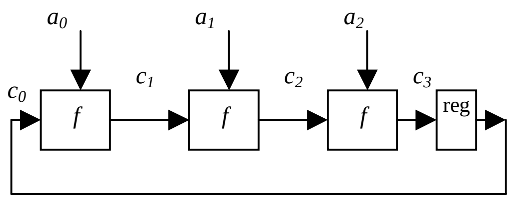


Figure 2. A circuit exhibiting the standard term ordering problem

The MDG representation of the new states in the first transition step of reachability analysis is shown in Figure 3. $f(b, a_i)$, $0 \le i \le 2$, is a cross-term where *b* is either 0 or 1. $f(b, a_i)$ is a compound term. The leftmost branch is built first and the order of the cross-terms is thus as follows: $f(0, a_0) < f(0, a_1) < f(0, a_2) < f(1, a_2)$. When the system starts to build the next left-hand branch (in grey), $f(0, a_2)$ and $f(1, a_2)$ have been built before and thus have smaller TermID than $f(1, a_1)$, which is newly generated. The left-hand branch is thus constructed under the order $f(0, a_0) < f(0, a_2) < f(1, a_2) < f(1, a_1)$. Note that no node sharing is possible for this branch under this order.

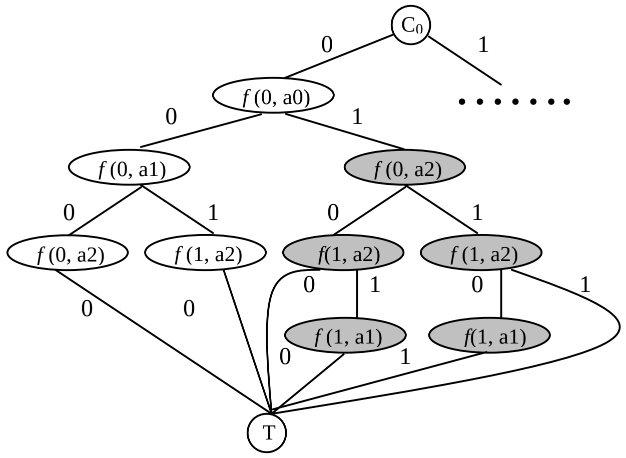


Figure 3. The MDG of new states in the first transition step of the reachability analysis

The number of paths that the MDG has is thus exponential with the number of the function instances as it decodes all possible combinations of the values of the first argument of *f*. The exponential number of paths in the MDG increases the execution time of reachability analysis. In the reachability analysis procedure, the Pruning-by- Subsumption (PbyS) operation is used to simplify the set of reachable states found so far by removing from it any paths that are subsumed by the frontier set [1]. The large number of paths in the MDG representing the set of reachable states raises the time and space needed by the PbyS operation. In this example, with the increasing number of function instances, PbyS operations could not terminate because of insufficient memory when it reached 4 instances.

4.3 A Chain Circuit Structure with the Standard Term Ordering Problem

In Figure 2, the output of each function instance is the input to the next function instance, but the standard term ordering decides that the cross-terms cannot be in the order leading to a linear size MDG. We further found out that the standard term ordering problem only happens to a circuit containing a chain of the same cross-functions as shown in Figure 4. The output of a function instance is the direct/indirect input of the next function instance. The circuit can have pipeline registers (or multiplexers) between function instances. Experiments show that the verification of the circuits with this kind of a chain structure would require rapidly increasing amounts of memory with an increasing number of function instances.

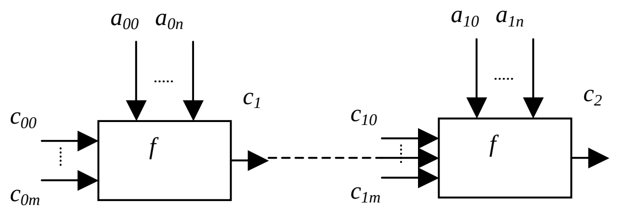


Figure 4. A common circuit structure resulting in the standard term ordering problem

Since the standard term ordering problem is caused by one of MDG well-formedness conditions, we are not able to solve this problem using the usual ordering strategies. In the custom order, we only consider the order of cross-functions. The cross-terms are created dynamically in the verification process and no static ordering or reordering procedure is possible at this stage. In the next section, we present a solution to this problem by integrating function renaming and cross-term rewriting.

5. A SOLUTION FOR THE STANDARD TERM ORDERING PROBLEM

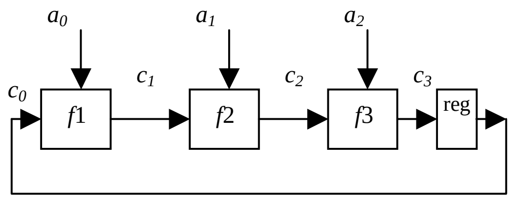
A solution to the standard term ordering problem is to reorder the cross-terms with the same cross-operators to make the corresponding MDG linear in size. The order of those cross-terms is decided by the lexicographical order of their MDG internal representation, hence we can only change the order by renaming the function instances, thereby allowing them to be freely placed within the custom order. Rewriting rules have to be used to map the renamed functions back to their original functions during the PbyS operation. We describe next our solution.

5.1 Function Renaming

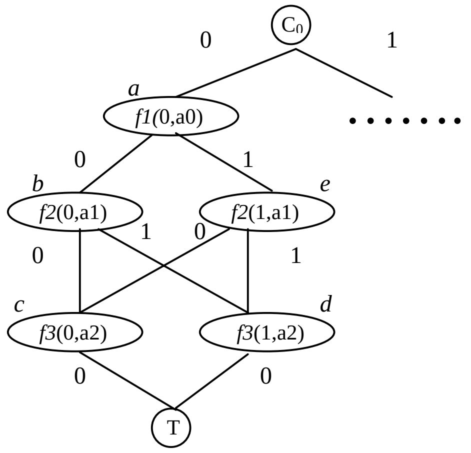
After identifying the chain structure (Figure 4) in a circuit, we can rename all the cross-functions instances in the chain to different names. For instance, for the circuit shown in Figure 2, we rename the three instances of *f* to *f*1, *f*2, *f*3 and impose the order *f*1 < *f*2 < *f*3. The circuit and the MDG of the new states in the first transition step are shown in Figure 5. The MDG becomes linear in size because of the sharing of subgraphs from the second level.

As explained in [3], the order of an output should come after all its inputs. Thus, after renaming all the functions in the chain structure, a static order for the renamed functions is chosen according to the positions of the functions appearing in the chain structure, i.e., from inputs to outputs. In this example (Figure 5), the order *f*1 < *f*2 < *f*3 is imposed. The resulting MDG is 26% smaller than using the reverse order.

Function renaming solves the standard term ordering problem. However, it brings out a new problem: the actual function of the circuit has changed and the circuit is thus more general than needed. To solve this new problem, we introduce cross-term rewriting. With cross-term rewriting, the MDG system can rewrite the renamed functions back to the original function when an actual comparison is being done and thus the verification result is not changed because of function renaming.



(a) Circuit structure



(b) MDG of the new states in the first transition step

Figure 5. MDG after function renaming

5.2 An Unconditional Cross-term Rewriting System

A rewriting system was first used in MDG to partially interpret an uninterpreted function symbol [8]. A data operation is represented by an uninterpreted function symbol. However, function symbols need to be partially interpreted in many designs where optimization is used. For example, a multiplier can be bypassed when one of the operands is 1. To verify the designs including such bypassing logic, we need to use the fact that 1 is the unit element of multiplication. One effective way to reason about the partially interpreted function symbols is term rewriting. For this multiplication example, the algebraic equations $1 * x = x$ and $x * 1 = x$ could be used as rewrite rules. Rewrite rules for cross-terms can be used to shrink the MDG size on the fly. For example, if there is a path in an MDG which has a cross-term $eq(a, a) = 0$ where *eq* stands for equality, we could use the rewrite rule $eq(a, a) \rightarrow 1$ to eliminate this path. The scope of MDG-based verification is thus extended.

In the original rewriting system, a cross-term cannot be rewritten into another cross-term having a different cross-operator [8]. This is to avoid the necessity of node reordering after rewriting. We thus introduce an unconditional cross-term rewriting system, as follows:

Definition 6.1: An unconditional cross-term rewriting system (CTRS) H is a finite set of formulas, called rewrite rules, having the following form:

$$\text{LHS} \rightarrow \text{RHS}$$

where LHS and RHS are cross-terms. (LHS/RHS stands for the left/right-hand side.)

When a cross-term is matched with the LHS of a rule, LHS will be substituted by RHS unconditionally. For instance, in Figure 5, a set of rewrite rules can be defined as follows:

$$f1(x, y) \rightarrow f(x, y)$$
$$f2(x, y) \rightarrow f(x, y)$$
$$f3(x, y) \rightarrow f(x, y)$$

So the cross-terms in Figure 5(b) are substituted under the following rules:

$$f1(0, a_0) \rightarrow f(0, a_0) \qquad f1(1, a_0) \rightarrow f(1, a_0)$$
$$f2(0, a_1) \rightarrow f(0, a_1) \qquad f2(1, a_1) \rightarrow f(1, a_1)$$
$$f3(0, a_2) \rightarrow f(0, a_2) \qquad f3(1, a_2) \rightarrow f(1, a_2)$$

To avoid the necessity to reorder, the rewriting system rewrites the cross-terms in an MDG only when a comparison is needed. By inspecting all the model checking algorithms used in the MDG system, comparisons are needed only in the PbyS (Prune by Subsumption) operation [7].

We use function renaming to build an MDG with a good order generated by our variable ordering algorithm, then use rewriting to restore the original function of circuits without changing the size of the MDG. We further found out that our renaming and rewriting method can be extended to all the circuits containing cross-operators which have more than three instances placed irregularly. Although the standard term ordering problem may not happen in these circuits, the instances of the same cross-operator have different positions in the circuit and thus renaming of the instances allows them to have different positions in the custom order. Our method provides more flexibility in variable ordering. As each instance has a different function symbol, it can now be freely placed in the custom order. The MDG can thus achieve better sub-graphs sharing and reduce its size and execution time.

6. EXPERIMETAL RESULTS

6.1 A Case Study

We introduce an ATM congestion controller [5] in this subsection. It exhibits the standard term ordering problem in MDG-based verification. In an ATM switch, during the periods of heavy traffic within the network, an outgoing link may become temporarily overloaded and data packets (cells) begin to build up in the outgoing queue. This is known as congestion.

A congestion controller is used to solve congestion by comparing the priorities of two packets. The packets with a higher priority will be passed, the other ones will be discarded. This does not mean that the discarded messages would be lost. The packets can be resent by the senders when they do not receive acknowledgement from the receivers, using higher-level protocols.

The controller is shown in Figure 6. a_0, b_0 are abstract variables representing information packets to be sent. The cross-function *compare* is used to compare the priorities of two packets. Multiplexers are used to transfer the packets with high priority. When a circuit contains more than four cross-functions, the standard ordering problem occurs.

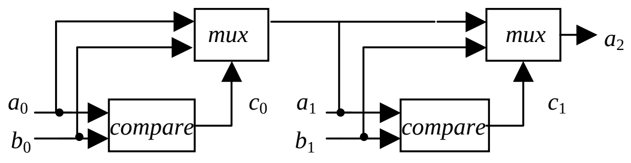


Figure 6. An ATM congestion controller

We implemented our algorithm in the reachability analysis procedure and the result of applying it to the circuit in Figure 6 is shown in Table 1. The experiments were carried out on a 333MHz Sun Ultra 10 workstation with 1GB of memory. The table shows that without the function renaming and rewriting, the controller is only limited to four blocks. With our solution, reachability analysis can be carried out with more than 320 blocks.

6.2 Experimental Results on a Scheduler

We extend this solution to the circuits where there are several function instances placed irregularly. We conducted the experiments on packet scheduler. This model contains 72 abstract state variables and 31 concrete state variables. It has 8 instances of the cross-function *not_equal_to_zero*(x) which tests if an abstract variable x is equal to zero. The experimental results are shown in Table 2. We verified two safety properties on the scheduler. Experimental results in Table 2 show that with our solution, the size of MDG can be reduced by 26.5% and the computing time can be reduced by 26.4%.

Table 1. Experimental results for an ATM congestion controller

No. of	Without the solution		With the solution	
Functions	Nodes	Time(sec)	Nodes	Time(sec)
2	129	0.40	163	0.68
3	273	1.56	329	0.79
4	541	378.02	507	0.98
5	-	-	697	1.20
6	-	-	910	1.36
7	-	-	1113	1.72
10	-	-	1402	2.12
100	-	-	25880	93.21
200	-	-	81330	883.68
300	-	-	166780	4236.64
320	-	-	187470	6226.46

Table 2. Experimental results for a scheduler

Property	Without the solution		With the solution	
	Nodes	Time(sec)	Nodes	Time(sec)
P1	427719	4847.15	247826	2808.56
P2	277101	3642.57	247847	3251.36

6.3 Conclusions

In this paper, we introduced a special problem caused by the standard term order that is used in MDGs to order cross-terms with the same function symbols. Since this problem is the result of the MDG well-formedness conditions, it cannot be solved by ordering/reordering variables. We presented a solution integrating function renaming and cross-term rewriting. Experimental results on a congestion controller and on a packet scheduler show that our solution can considerably improve the performance and increase the type and size of circuits that can be verified.

7. REFERENCES

[1] R. E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. ICCAD'95, San Jose, USA, 1995, 236-243.

[2] E. Cerny, F. Corella, M. Langevin, X. Song, S. Tahar and Z. Zhou. Automated Verification with Abstract State Machines Using Multiway Decision Graphs. Formal Hardware Verification: Methods and Systems in Comparison, edited by T. Kropf, Springer-Verlag Publishers, 1997, 79-113.

[3] Y. Feng and E. Cerny. Variable Ordering on Multiway Decision Graphs. Submitted to ISCAS 2002.

[4] A. Gupta. Formal Hardware Verification Methods: A Survey. Formal Methods in System Design, vol. 1, 1992, 151-238.

[5] H. Liu. Congestion Control for ATM in Broadband ISDN. Master Thesis, University of Saskatchewan, 1992.

[6] R. Rudell. Dynamic variable ordering for Ordered Binary Decision Diagrams, In Proc. of IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, USA, 1993, 42-47.

[7] Y. Xu, E. Cerny, X. Song, F. Corella and O. Ait Mohamed. Model Checking for a First-Order Temporal Logic using Multiway Decision Graphs. In Proceedings of Conference on Computer-Aided Verification (CAV'98), Vancouver, Canada, 1998, 219-231.

[8] Z. Zhou. Multiway Decision Graphs and Their Applications in Automatic Formal Verification of RTL Designs. PhD thesis, D'IRO, University of Montreal, 1997.

[9] Quintus Prolog Manual. Quintus Corporation, v. 3.1, 1991.