Partitioning Transition Relations
Efficiently and Automatically

Z. Zhou† X. Song‡ F. Corella‡ E. Cerny† M. Langevin§

† D'IRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Canada H3C 3J7
‡ IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
§ GMD-SET, Schloss Birlingshoven, 53757 St. Augustin, Germany

Abstract
Multiway Decision Graphs (MDGs) have been recently proposed as an efficient representation of Extended Finite State Machines (EFSMs), suitable for automatic hardware verification of Register Transfer Level (RTL) designs [7, 14]. We report here on the results of our research into automatic partitioning of state transition relations described using MDGs. The objective is to achieve the maximum possible performance during an abstract implicit state enumeration procedure that is at the basis of our automatic verification method.

1 Introduction
Bryant's Reduced Ordered Binary Decision Diagrams (ROBDDs) [2] have proved to be a powerful tool for automated hardware verification [3, 4, 6, 8, 13]. ROBDDs have a drawback, however, when verifying hardware designs at higher levels of abstraction, such as the Register-Transfer Level: They require a binary representation of the entire circuit including the datapath operations, and the size of the corresponding ROBDDs can grow, sometimes exponentially, with the number of variables. Yet, the design of the data operation units can be verified in part independently of their instantiation context or they are correct by construction through the use of module generators. Therefore, when verifying a processor design, for instance, one can concentrate on the correct implementation of the sequencing of operations on the datapath against the instruction definition, and assume that the operation units are correct. To support efficient and automatic verification methods of this kind, we have recently proposed [7] a new class of decision graphs called Multiway Decision Graphs (MDGs) that comprise but are much broader than ROBDDs. In an MDG, data are represented by single variables of abstract sort rather than by vectors of Boolean variables, and data operations become uninterpreted function symbols. MDGs are thus much more compact than ROBDDs for circuits containing datapaths, which considerably increases the range of circuits that can be verified. 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 have developed a verification technique using MDGs for synchronous RTL designs [7, 14] that proves the equivalence of two Extended Finite State Machines by performing reachability analysis of the product machine1. The efficiency of the method thus strongly depends on the efficiency of the computation of the set of reachable states. We employ implicit abstract state enumeration in which both sets of states and state transition relations are represented using MDGs. It resembles the implicit enumeration used with ROBDDs, except that the states have both concrete and abstract components.

To compute the effect of a transition on a set of states we developed a relational product algorithm using MDGs. We use partitioned transition relations with a heuristic ordering of individual transition relations [4, 9]. Besides lifting this technique from the Boolean domain to the abstract domain by the use of MDGs, we made the following improvements:

1. We extended the relational product algorithm [4] to an n-ary operation in the more general MDG setting, i.e. it applies to a list of arguments containing multiple MDGs. Our experiments indicate that this algorithm is faster than a repetitive application of a binary operation.

2. We used the n-ary relational product operation to compute more efficiently the MDG of each individual transition relation.

3. We modified the ordering heuristics of [9] to take into account (i) the characteristics of MDGs, and (ii) the fact that the transition relation is that of a product machine.

1To form the product machine, we drive the two designs by common inputs and derive the transition relation(s) for the combined system.
4. We do not compose in advance the MDGs representing the individual transition relations in a block to form a single MDG as [4] did for ROBDDs, but rather we leave them as a group of MDGs composed on the fly using our n-ary operation whenever needed.

In this paper we evaluate experimentally these improvements.

The paper is organized as follows: In Section 2, we briefly introduce MDGs and the n-ary relational product operation. In Section 3, we explain how we use MDGs to represent EFSMs. In Section 4 we describe our approach to partitioning transition relations, and we present experimental results in Section 5. A summary and conclusions are in Section 6.

2 Multiway Decision Graphs

The formal logic underlying MDGs is many-sorted first-order logic, augmented with a distinction between abstract sorts and concrete sorts. Concrete sorts have enumerations, while abstract sorts do not. An enumeration is a set of individual constants. Beside individual constants, the vocabulary consists of generic constants, variables, and function symbols (also called operators). Generic constants and variables each have a sort. An n-ary function symbol \( (n > 0) \) has a type \( \alpha_1 \times \cdots \times \alpha_n \rightarrow \alpha_{n+1} \), where \( \alpha_1 \ldots \alpha_{n+1} \) are sorts. The (well-typed) terms and their sorts are defined as usual.

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 \( \alpha_1 \times \cdots \times \alpha_n \rightarrow \alpha_{n+1} \). If \( \alpha_{n+1} \) 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 \( \alpha_{n+1} \) is concrete while at least one of \( \alpha_1 \ldots \alpha_n \) is abstract, then we refer to \( f \) as a cross-operator; cross-operators are useful for modeling feedback from the data path to the control circuitry.

A term that has no concrete subterms other than individual constants is said to be concretely reduced. A term of the form \( f(A_1, \ldots, A_n) \) where \( f \) is a cross-operator and \( A_1 \ldots A_n \) are concretely reduced terms, is a cross-term. An equation is an expression \( A_1 = A_2 \), where \( A_1 \) and \( A_2 \) are terms of same type \( \alpha \). Atomic formulas are the equations, plus \( \top \) (truth) and \( \bot \) (falsity). Formulas are built from the atomic formulas in the usual way using logical connectives and quantifiers.2

In the semantics of the logic, given in [7], the abstract sorts, the abstract function symbols, and the cross-operators are assigned arbitrary interpretations. We say informally that they are uninterpreted.

A multiway decision graph (MDG) is a finite, rooted directed acyclic graph (DAG) 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 \( N \). Just as Bryant's ROBDDs [2] must be reduced and ordered, MDGs must obey a set of well-formedness conditions [7] to be a canonical representation. Among other things, these conditions specify the kinds of nodes that may appear in an MDG. An internal node may be labeled by a variable of concrete sort, with edges issuing from the node labeled by individual constants in the enumeration of the sort; or by a variable of abstract sort, with edges labeled by concretely reduced terms of that sort; or by a cross-term, of concrete sort \( \alpha \), with edges labeled by individual constants in the enumeration of \( \alpha \). All leaf nodes are labeled \( \top \), except when the graph has a single node, which may be labeled \( \top \) or \( \bot \). From now on MDG will mean well-formed MDG unless otherwise stated. If an MDG \( G \) represents a formula \( P \), we shall use \( P \) to refer both to the formula and to the graph when there is no risk of confusion.

For example, consider the synchronous machine in Figure 1(a) in which the data register \( r \) loads a new (abstract) input value \( x \) if it is less than the current value in \( r \), else \( r \) keeps the old value. Figure 1(b), (c), (d) show the MDGs for \( r \), the inverter and the comparator \( g eq \), respectively.3 The data input variable \( x \), the state variable \( y \) and the next state variable \( y' \) are of abstract sort, while \( z_0 \) and \( z_1 \) are concrete (more specifically, Boolean). The comparator, which provides feedback from the data path to the control of the machine, is modeled as a black box whose functionality is represented by \( g eq \), which is an uninterpreted cross-operator.

![Figure 1: Representing circuits using MDGs.](image)

ROBDD operations can be implemented using a single generic algorithm \textit{Apply} [2]. This is because the two edges that issue from an ROBDD node labeled \( x \) span the range of values \{0, 1\} that \( x \) can take, and this makes it possible to reason by cases. Since MDGs do not enjoy this property, a separate algorithm must be provided for each operation. We have developed algorithms [7] for disjunction, relational product4, and

---

2These are the formulas of the underlying logic, not those representable by well-formed MDGs.

3Though the graphs are shown as tree structures, they are actually a forest of MDGs with sharing of subgraphs.

4The conjunction and existential quantification operations can be obtained as special cases of the relational product algo-
an algorithm called pruning-by-subsumption that provides a termination test for state space exploration in the presence of abstract types. These algorithms are implemented in a basic MDG package in Prolog. Our n-ary relational product algorithm deserves special mention here, as it is used extensively in the verification method and is one of the reasons for its efficiency: It accepts a set of MDGs \( \{ P_i \}_{1 \leq i \leq n} \) which do not share common abstract node labels, a set \( V \) of (concrete and abstract) variables which only occur as node labels in the \( P_i \), a renaming substitution \( \eta \), and produces an MDG \( R \) such that

\[
\models R \Leftrightarrow (\exists V)( \bigwedge_{1 \leq i \leq n} P_i) \cdot \eta. 
\]  

The result \( R \) is obtained by taking the conjunction of the \( P_i \), eliminating (abstracting) the variables in \( V \) by existential quantification, and renaming the variables in the domain of \( \eta \) among the remaining node labels. These operations are performed in one simultaneous traversal of all the \( P_i \), without building intermediate graphs. The substitution \( \eta \) is used to rename next-state variables to present-state variables.

Our experiments indicate that the n-ary product is faster than a repetitive application of a binary relational product used in [4].

3 Representation of EFSMs

Sequential RTL designs can be described using Extended Finite-State Machines (EFSMs). An extended deterministic Mealy machine is usually defined by a 7-tuple \((X, Y, Y', Z, \delta, \lambda, S_0)\), where \(X\) is a set \(\{x_1, \ldots, x_k\}\) of input variables, \(Y\) is a set \(\{y_1, \ldots, y_m\}\) of present state variables, \(Y'\) is a set \(\{y'_1, \ldots, y'_n\}\) of next state variables, and \(Z\) is a set \(\{z_1, \ldots, z_m\}\) of output variables. \(Y\) and \(Y'\) have one-to-one correspondence, i.e., each variable \(y_i \in Y \) for \(1 \leq i \leq n\) has a corresponding variable \(y'_i \in Y'\), and \(y_i\) and \(y'_i\) are of the same sort. The EFSM behavior is given by a transition relation \(\delta\) over variables in \(X \cup Y \cup Y'\) and an output relation \(\lambda\) over variables in \(X \cup Y \cup Z\). \(S_0\) is the initial set of states.

The application of EFSMs in formal verification usually requires the traversal of its state space. We lift the ROBDD-based implicit enumeration technique to the abstract level by using MDGs to represent sets of states, and the transition and output relations. The relational product algorithm described earlier is used for image computation.

In contrast to an ROBDD representation of transition (output) relations in which the variables are only concrete and the constraints are implicit in the structure of the ROBDD, an MDG describing the transition (output) relation has abstract variables appearing in edges or cross-terms and includes constraints imposed over the abstract values by cross-terms. As an example, Figure 2(a) shows an MDG for the transition relation of the circuit in Figure 1(a), where the cross-term

\[
\text{geo}(x, y) \text{ is used to impose the constraint } \text{geo}(x, y) = 0 \text{ on the left path and } \text{geo}(x, y) = 1 \text{ on the right path. Figure 2(b) shows an MDG for the initial state of the circuit in Figure 1(a). An abstract variable } y_0 \text{ represents all the possible values that register } r \text{ can take. Figure 2(c) shows an MDG that supplies a variable } x#1 \text{ as the abstract input at clock cycle 1. The relational product of the MDGs in Figures 2(a), (b) and (c) (with elimination of } x \text{ and } y, \text{ and renaming of } y' \text{ to } y \text{) represents the set of states reached in one step from the initial state and is shown in Figure 2(d).}
\]

![Figure 2: Representing a transition relation and sets of states using MDGs.](image)

Conceptually, it is possible to represent the transition relation \(\delta\) by a single MDG, called monolithic transition relation (MTR) [4]. But in practice, a monolithic transition relation is either too big to fit in memory or takes too much time to build. Consequently, we use an alternate representation, known as conjunctive partitioned transition relation (CPTR) [4], for synchronous circuits.

For each state variable \(y_i\), \(1 \leq i \leq n\), we define an individual transition relation \(\delta_i\) which gives the value of \(y_i\) as a function of the values of the variables in \(X \cup Y\). The MDG representing \(\delta\) is the conjunction of the MDGs representing the \(\delta_i\):

\[
\delta = \delta_1 \land \cdots \land \delta_i \land \cdots \land \delta_n. 
\]

In a CPTR, each partition block may contain one or several individual transition relations. During state exploration, we perform relational product operation iteratively over those blocks. In contrast with [4] where there is one ROBDD per block, we take advantage of our n-ary relational product to represent a block in the CPTR as a group of one or more MDGs representing individual transition relations without taking their conjunction:

\[
\{\delta_1 \land \cdots \land \delta_i\} \land \cdots \land \{\delta_j \land \cdots \land \delta_n\}
\]

where \(\{\cdots\}\) represents a partition block. More details are given in Section 4.3.

If each partition block in a CPTR contains exactly one individual transition relation then it is a full CPTR:

\[
\{\delta_1\} \land \cdots \land \{\delta_i\} \land \cdots \land \{\delta_n\}. 
\]

In the next section, we present an automatic facility for deriving an efficient conjunctive partitioned transition relation. The same technique is also applied to output relations.
4 Deriving partitioned transition relation

To construct an efficient CPTR, we first derive individual transition relations and order them using our heuristics. Then, we subdivide the ordered individual transition relations into groups each containing several MDGs such that the total number of nodes in a group is within certain limits.

4.1 Deriving individual relations

The MDGs of the individual relations are determined by computing the MDGs of all internal lines in a topological order over the structure of the circuit, starting from the primary inputs and the outputs of registers until all the inputs of registers are reached. The MDG of an output of a combinational component is computed by composing the MDGs of the inputs of the component (they have been computed earlier in the topological order) with the MDG of the component itself and abstracting the input variables in one n-ary operation. Experiment results (Table 3) show that this is more efficient than a repetitive use of a binary operation.

4.2 Ordering individual relations

Given an MDG $S_k$ representing the set of present states, an MDG $I_k$ supplying the abstract input values at clock cycle $k$, and a monolithic transition relation MDG $\delta$, the image of the set of present states is computed as the relational product:

$$\exists (X \cup Y)[S_k \land I_k \land \delta] \cdot \eta.$$  \hspace{1cm} (5)

If we use the full partitioned transition relation, then (5) can be rewritten as [4]:

$$\exists E_n[\ldots \exists E_1[S_k \land I_k \land \{\delta_1\}] \land \ldots \land \{\delta_n\}] \cdot \eta.$$  \hspace{1cm} (6)

Equation (6) allows us to compute the image iteratively over a list of smaller MDGs starting with $S_k$ and $I_k$ without computing $\delta$ first. At the ith ($1 \leq i \leq n$) iteration, we quantify over the input and state variables that do not appear in $\delta_{i-1}$ and $\delta_{j}$, for all $j > i$. These variables are denoted by the set $E_i$. This early quantification technique is the reason for the efficiency of CPTR [4, 5, 9]. For each $i$, $1 \leq i \leq n$, $E_i$ can be computed as follows [4]:

$$E_i = D_i - \bigcup_{k=i+1}^{n} D_k$$  \hspace{1cm} (7)

where $D_j$ ($1 \leq j \leq n$) is the set of input and state variables that occur in $\delta_j$. Such variables are called dependent variables. In ROBDDs, they are simply node variables, but in MDGs they also include the abstract variables appearing in both the cross-terms and edge labels.

In practice, we must be careful when using Equation (6). Suppose $D_n = V$, then we have $E_1 = \ldots = E_{n-1} = \emptyset$ and $E_n = V$, which means the state and input variables can be quantified away only in the last product operation. This may be no easier than using Equation (5) where $\delta$ is computed once and for all. It is thus important to order the $\delta_i$ in such a way that the variables can be quantified away as early as possible.

Geist [9] proposed ordering heuristics that work well with ROBDD-based symbolic model checking. Due to the different structure of MDGs and the structure of the product machine, we augment the heuristics given in [9] ($H1$ & $H2$) by two additional rules ($H3$ & $H4$):

$H1$. It is preferable to compose as early as possible the $\delta_i$ that have more dependent variables that do not appear in other individual transition relations. These variables are referred to as unique variables [9]. Those variables can be quantified away immediately after composing with $\delta_i$.

$H2$. It is preferable to compose earlier those $\delta_i$ that have more dependent variables. The justification is that once $\delta_i$ is composed, it is most likely to enable the following graphs to have more unique variables.

$H3$. MDG structure: It is preferable to perform as early as possible the product of MDGs that contain the same cross-terms. This is because the interaction between the constraints over the same cross-terms generally reduces the size of the resulting MDG. In practice, we use a less strict heuristic: compose as early as possible the MDGs having more cross-terms.

$H4$. Product machine: It is preferable to compose first the MDGs belonging to one machine. This is because the transition relations of the two different machines are relatively independent, sharing only the primary input variables. The state variables of one machine can thus be abstracted at early stages of the composition.

Based on our experience, we found $H1$ and $H3$ to be the most effective, and $H2$ having the least effect. Our ordering algorithm combines the above rules accordingly, and automatically produces a total order over the MDGs of the individual transition relations.

4.3 Grouping individual relations

In general, the cost of manipulating fewer but larger MDGs must be compared with the cost of manipulating many smaller MDGs. On one side of the spectrum we have Equation (6) where $n + 2$ small(er) MDGs must be composed, and on the other side is Equation (5) with only three MDGs to compose. Burch et al [4], using their intuition, recombine some (manually selected) individual transition relations into a single ROBDD, thus forming a CPTR containing fewer ROBDDs than the full CPTR.

Our n-ary relational product operation allows us to group individual transition relations without computing the conjunction of the MDGs in each group. The experimental results show that taking the n-ary product of each intermediate result with all the MDGs in the next group takes practically only a little longer than taking the product of the intermediate result with the conjunction of those MDGs. Yet, by grouping rather than recombing we save the time of computing the conjunction of the MDGs in each block, which
Table 1: Partitioned vs. monolithic transition relations (CPU time in seconds)

<table>
<thead>
<tr>
<th></th>
<th>TR</th>
<th>Re</th>
<th>Misc</th>
<th>GC</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tamarack-3 (Imp, Spec)</td>
<td>m</td>
<td>14</td>
<td>3</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>Tamarack-3 (Imp, Imp)</td>
<td>m</td>
<td>93</td>
<td>7</td>
<td>21</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>15</td>
<td>12</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>Tamarack-3 (Imp, Spec)</td>
<td>m</td>
<td>103</td>
<td>40</td>
<td>7</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>36</td>
<td>82</td>
<td>10</td>
<td>184</td>
</tr>
<tr>
<td>Tamarack-3 (Imp, Imp)</td>
<td>m</td>
<td>473</td>
<td>-</td>
<td>-</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>40</td>
<td>98</td>
<td>19</td>
<td>217</td>
</tr>
</tbody>
</table>

Table 2: Partitioned vs. monolithic transition relations (Memory use in Mbytes)

<table>
<thead>
<tr>
<th></th>
<th>TR</th>
<th>ReAn</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompAS (Imp, Spec)</td>
<td>m</td>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>2</td>
<td>0.9</td>
</tr>
<tr>
<td>CompAS (Imp, Imp)</td>
<td>m</td>
<td>8.4</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>2.3</td>
<td>2.7</td>
</tr>
<tr>
<td>CompAS (Imp, Spec)</td>
<td>m</td>
<td>11</td>
<td>6.7</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>8.5</td>
<td>10.5</td>
</tr>
<tr>
<td>CompAS (Imp, Imp)</td>
<td>m</td>
<td>88</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>84</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3: Deriving individual transition relations.

It follows from the above tables that the partitioned transition relation is more efficient than the monolithic transition relation, both in run time and memory consumption. It also indicates that it takes much more time to build the monolithic transition relation by conjunction of all the individual transition relations.

In Table 3, we present the experimental results for deriving individual transition relations. Besides using the n-ary relational product operation, we also simulated the binary operation. The improvements shown in the table speak for themselves.

Tables 4 and 5 present the results of computing all the reachable states of the product state machines. Table 4 shows the effect of various ordering strategies with the full partitioned transition relations, i.e., one individual transition relation per group (Equation (6)). Row B.H. used the order generated according to the heuristics H1 and H2 in [9] (BDD Heuristics). Row M.H. used the order generated by our heuristic ordering algorithm (MDG Heuristics, H1–H4). Table 5 shows the results using various grouping strategies with the individual transition relation order obtained from our heuristic algorithm. Row full uses the full partitioned transition relations. Row single corresponds to a single group containing all the individual transition relations. Row auto corresponds

<table>
<thead>
<tr>
<th></th>
<th>B.H.</th>
<th>M.H.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tamarack-3 (Imp, Spec)</td>
<td>25</td>
<td>21</td>
</tr>
<tr>
<td>Tamarack-3 (Imp, Imp)</td>
<td>79</td>
<td>67</td>
</tr>
<tr>
<td>CompAS (Imp, Spec)</td>
<td>483</td>
<td>331</td>
</tr>
<tr>
<td></td>
<td>1175</td>
<td>916</td>
</tr>
</tbody>
</table>

Table 4: Results using various ordering strategies.

---

5 Experimental results

We implemented the automatic facility for deriving partitioned transition relations in Section 4 and a reachability analysis algorithm based on implicit abstract enumeration [7] on top of our MDG package programmed in Prolog. We performed behavioral equivalence checking for two identical RTL implementations (referred to as (Imp, Imp)) and verification of a microprocessor RTL implementation against its instruction set architecture [7, 14] (referred to as (Imp, Spec)) using two examples. The benchmarks are a simple non-pipelined microprocessor, Tamarack-3 [10] and a more complex one CompAS [11, 12]. To illustrate the effects of our partitioning strategy, we present here comparative empirical results. The CPU times reported are in seconds on a 50MHz SUN Sparc workstation. The memory size is in mega-bytes.

Tables 1 and 2 show run times and memory consumption for verification using monolithic transition relations (rows m) vs. partitioned transition relations (rows p) based on our heuristics. In Table 1, column TR is the run time in seconds for constructing the transition relation. Column Re is the total time for computing all the reachable states. Column Misc is the time for miscellaneous tasks: consistency checking on the circuit description, output comparison, etc. Column GC is the time the Prolog system used for garbage collection. In Table 2 column TR is the memory used for constructing the transition relation and column ReAn is the memory used for reachability analysis, in megabytes. In both tables means that we were not able to complete the task.

---

6 The verification of Tamarack-3 implementation against the instruction set architecture was done only for the "full synchronous" mode of operation.

7 All the experiments in this section are based on a fixed order of nodes in the MDGs.
Table 5: Results using various grouping strategies.

<table>
<thead>
<tr>
<th></th>
<th>Tamarack-3</th>
<th>Tamarack-3</th>
<th>CompAs</th>
<th>CompAs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Imp, Spec)</td>
<td>(Imp, Imp)</td>
<td>(Imp, Spec)</td>
<td>(Imp, Imp)</td>
</tr>
<tr>
<td>full</td>
<td>21</td>
<td>67</td>
<td>331</td>
<td>916</td>
</tr>
<tr>
<td>single</td>
<td>0</td>
<td>14</td>
<td>115</td>
<td>177</td>
</tr>
<tr>
<td>auto</td>
<td>5</td>
<td>12</td>
<td>82</td>
<td>98</td>
</tr>
</tbody>
</table>

Table 6: Grouping vs. recombinating.

<table>
<thead>
<tr>
<th></th>
<th>Tamarack-3</th>
<th>Tamarack-3</th>
<th>CompAs</th>
<th>CompAs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Imp, Spec)</td>
<td>(Imp, Imp)</td>
<td>(Imp, Spec)</td>
<td>(Imp, Imp)</td>
</tr>
<tr>
<td>recom</td>
<td>c</td>
<td>r</td>
<td>c</td>
<td>r</td>
</tr>
<tr>
<td>group</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5 and Table 6 demonstrate the efficiency of our heuristics on these benchmarks.

Finally, Table 6 shows the results of a comparison between the recombinination method of [4] (row recom) and our grouping method (row group). The recombinination experiment has been performed by taking the grouping corresponding to the last row of Table 5, computing the conjunction of the MDGs in each group, and using the resulting MDGs as the partitioned transition relations. The columns labeled "c" give the time it takes to compute the conjunction, while the columns labeled "r" give the time it takes to compute the reachable states. The results confirm the discussion of Section 4.3.

6 Conclusions

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 studied the problem of partitioned transition relations as an efficient representation of EFSMs. We also developed an automatic facility to derive partitioned transition relations by ordering and grouping of individual transition relations using MDGs. These techniques are crucial both to verification efficiency and automation.

References


