A New Approach for the Construction of Multiway Decision Graphs

Y. Mokhtari[1], Sa'ed Abed[1], O. Ait Mohamed[1], S. Tahar[1] and X. Song[2]

[1]Dept. of ECE, Concordia University, Canada
{mokhtari,s_abed,ait,tahar}@ece.concordia.ca

[2]Dept. of ECE, Portland State University, USA
song@ee.pdx.edu

Abstract. Multiway Decision Graphs (MDGs) are a canonical representation of a subset of many-sorted first-order logic. It generalizes classical BDDs with abstract data and uninterpreted functions. In this paper, we describe a new MDG construction based on the Generalized-If-Then-Else (GITE) operator. Consequently, we review the main algorithms used for verification techniques i.e. relational product and pruning by subsumption. Unlike an earlier version of the MDG package, basic MDG algorithms are defined uniformly through this single GITE operator which will lead to a more efficient implementation. The new tool, called NuMDG, accepts an extended SMV language, supporting abstract data sorts.

1 Introduction

Reduced and Ordered Binary Decision Diagrams (ROBDDs) [1] have been widely studied due to their successful use in automated hardware verification. The key of the success is a canonical representation and easy manipulation of Boolean functions. Most BDD packages provide an efficient implementation based on recursive operations using a three operand function commonly known as ITE. Moreover, they provide many operations that are extensively used in automated verification methods. However, these methods suffer from the drawback that they require a binary representation of the circuit. Every individual bit of every data signal must be encoded by a separate Boolean variable, while the size of ROBDD grows, sometimes exponentially, with the number of variables. This leads to a state explosion problem when ROBDD-based methods are applied to circuits with complex datapath.

Multiway Decision Graphs (MDGs) [2] have been proposed to overcome this limitation. 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. Therefore, especially for circuits having a complex datapath, MDGs are much more compact than ROBDDs and enhance the capability to verify a broader range of circuits [3]. In MDG-based verification, abstract description of states machines (ASM) are

used for modeling systems. In contrast to ordinary Finite State Machines (FSM), the ASM supports non-finite state machines as models in addition to their intended interpretations. The intent is to rise the abstraction level of automated verification methods to approach those of interactive theorem proving methods without sacrificing automation. MDGs have been investigated from different angles and it culminated in a MDG tool providing Prolog-style MDG-HDL for modeling and different verification techniques including sequential and combinational equivalence checking, invariant checking and a subset of first-order LTL model checking [4, 5].

The work presented here mainly reviews the previous work [2] in one respect. The set of basic operations on MDGs was implemented separately, while ROBDD operations are implemented using a single generic algorithm ITE. This is because the two edges that issue from an ROBDD node labeled x span the ranges of values $\{\mathsf{F}, \mathsf{T}\}$ that x can take, and this makes it possible to reason by case analysis. Consequently, MDGs do not enjoy this property due to abstract variables. The GITE operation can be considered to be a functionally complete three-input logic gate that implements the expression $GITE = (P \wedge Q) \vee (\neg P \wedge H)$. If P is an abstract variable, then there is no MDG representing the formula $\neg P$. In this paper, we claim that it is possible to use the GITE operation to produce an MDG R that is logically equivalent to $(P \wedge Q) \vee (\neg P \wedge H)$ except for some cases that will be discussed later. This leads to improve the efficiency of the existing basic MDG algorithms.

Finally, we provide an architecture for our new tool. The goal here is to build a robust model checking tool that accepts an extended SMV input language and supports an abstraction mechanism through abstract sorts and uninterpreted functions. Indeed, more work should be spent in implementing and developing the tool in order to enhance the performance.

The paper is organized as follows: Section 2 reviews the closest related work. Section 3 introduces a subset of many-sorted first-order logic that gives MDGs their meaning. Section 4 describes basic MDG algorithms, their optimization and their correctness proof. Section 5 introduces the architecture of NuMDG. Finally, Section 6 concludes our paper and presents the future work.

2 Related Work

Approaches that capture non-finite aspects of the system, by using uninterpreted functions or similar notion like first-order formulae with quantification, are more closely related work.

Burch and Dill [6] have proposed a verification method that uses uninterpreted functions to denote data operations and a decision procedure as a theorem-proving search method. Compared to MDG, their method does not support representation of a set of states, fixpoint calculation and the transition relation can be applied only a given number of times. Since then, uninterpreted functions have generated a considerable interest in two respects: integration into

a symbolic model checkers [7, 8] or developing BDD-based decision procedures [9, 10].

More recently, Bryant *et al.* [11] translate a formula with uninterpreted functions to propositional formula within the theory of equality while preserving validity. Therefore, the resulting formula can be checked efficiently either by a BDD or SAT solver. This reduction is based on Ackermann's approach [12] that consists of replacing each occurrence of a function with a new (domain) variable and adding functional consistency constraints in the formula. A similar approach is also proposed by Pnueli *et al.* [13] where the key differences are emphasized in [11].

These approaches are applicable when data operations can be viewed as black-boxes, i.e., the correctness of the system being modeled does not depend on the meaning of these operations. This is usually the form of RTL designs generated by high-level synthesis algorithms that schedule and allocate data operations without being concerned with the specific nature of the operations. However, ignoring properties of data operations leads sometimes to false negatives. For example, a multiplier can be abstracted away when one of its inputs is 0 or 1. In MDG, a simple rewriting system is used to deal with such cases. In [14], Velev combines rewriting rules and Burch and Dill's method [6] to verify out-of-order processors that have a Reorder Buffer.

3 Multiway Decision Graphs Overview

3.1 Sorted Signature

A sorted signature $\Sigma(\mathcal{V}, \mathcal{L}, \mathcal{S})$ consists of an infinite set of variables $\mathcal{V}$, partitioned into a set $\mathcal{V}_{abs}$ of abstract variables and a set $\mathcal{V}_{con}$ of concrete variables, a set of symbols $\mathcal{L}$, partitioned into a set $\mathcal{L}_{CO}$ of cross-operators and a set $\mathcal{L}_F$ of function symbols and a set of sort symbols $\mathcal{S}$, partitioned into a set $\mathcal{S}_{con}$ of concrete sorts and a set $\mathcal{S}_{abs}$ of abstract sorts. All these sets are disjoint. Furthermore there is:

- An arity function that associates to each symbol in $\mathcal{L}$ a natural number. Constant symbols are 0-ary function symbol.
- A function $\eta : \mathcal{V} \rightarrow \mathcal{S}$ which gives a sort to each variable symbol.
- A set of sort declarations for terms. A sort declaration for a term is a tuple $t : S$, where t is a non-variable term and $S \in \mathcal{S}_{abs}$ is a sort symbol. We sometimes abbreviate sort declaration $f(x_1, \ldots, x_n) : S$ as $f : S_1 \times \ldots \times S_n \rightarrow S$ where S_i is the sort of the variable x_i.
- A set of sort declaration for cross-operators. A sort declaration for a cross-operator is of the form $p : S_1 \times \ldots \times S_n \rightarrow S$ where the S_i are sorts and $S \in \mathcal{S}_{con}$

3.2 Well Sorted Terms

The set of well sorted terms $\mathcal{T}(\Sigma, S)$ of sort S in signature Σ is the smallest set such that:

- $x \in \mathcal{T}(\Sigma, S)$ if $x \in \mathcal{V}$ and $\eta(x) \in S$
- $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, S)$ if $t_i \in \mathcal{T}(\Sigma, S_i)$ for $i = 1, \ldots, n$ and $f : S_1 \times \ldots \times S_n \rightarrow S$ is a term sort declaration in Σ

The set $\mathcal{T}(\Sigma)$ of all well sorted terms is defined as the union $\bigcup\{\mathcal{T}(\Sigma, S) : S \in \mathcal{S}\}$. If $\mathcal{V} = \emptyset$, then $\mathcal{T}_G(\Sigma, S)$ denotes a set of ground terms i.e. terms that are not containing variables. A substitution σ is represented as a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where $\mathsf{Dom}(\sigma) = \{x_1, \ldots, x_n\}$ and is defined on terms as usual. Its extension by another substitution σ', written $\sigma \oplus \sigma'$, is another substitution such that:

- $\mathsf{Dom}(\sigma) \cap \mathsf{Dom}(\sigma') = \emptyset$ and
- for every variable $x \in \mathsf{Dom}(\sigma \oplus \sigma')$:

$$(\sigma \oplus \sigma')(x) \;=\; \begin{cases} \sigma(x) & \text{if } x \in \mathsf{Dom}(\sigma) \\ \sigma'(x) & \text{if } x \in \mathsf{Dom}(\sigma') \end{cases}$$

3.3 Well Formed Directed Formulae (DFs)

The set of well formed formulae $\mathcal{F}(\Sigma, S)$ of sort S in signature Σ is the smallest set such that:

- $x = t$ if $x \in \mathcal{T}(\Sigma, S)$, $t \in \mathcal{T}_G(\Sigma, S)$ and $S \in \mathcal{S}_{con}$.
- $x = t$ if $x, t \in \mathcal{T}(\Sigma, S)$ and $S \in \mathcal{S}_{abs}$.
- $p(t_1, \ldots, t_n) = t$ if $p : S_1 \times \ldots \times S_n \rightarrow S$ is a cross-operator declaration in Σ, either $t_i \in \mathcal{T}(\Sigma, S_i)$ and $S_i \in \mathcal{S}_{abs}$ or $t_i \in \mathcal{T}_G(\Sigma, S_i)$ and $S_i \in \mathcal{S}_{con}$ for $i = 1, \ldots, n$ and $t \in \mathcal{T}_G(\Sigma, S)$.
- $\neg P$ is a formula if $\mathit{Vars}(P) \cap \mathcal{V}_{abs} = \emptyset$.
- $P \wedge Q$ is a formula if $\mathit{Vars}(P) \cap \mathit{Vars}(Q) = \emptyset$.
- $P \vee Q$ is a formula if $\mathit{Vars}(P) \cap \mathcal{V}_{abs} = \mathit{Vars}(Q) \cap \mathcal{V}_{abs}$ and for each variable $x \in \mathit{Vars}(P)$ either it occurs as a primary or secondary occurrence but not both.
- $(\exists x : S)P$ is a formula where x can be both primary and secondary occurrence in P.

where further connectives like T F $\Rightarrow$, $\Leftrightarrow$ and $\forall$ are defined as the standard abbreviations. $\mathit{Vars}(P)$ denotes the variables occurring in P. The occurrence of the variable x in a LHS of the formula $x = t$ is called a *primary occurrence*, otherwise it is a *secondary occurrence*. Note that by our syntax definition, only abstract variables have secondary occurrences. We say a DF formula P is of type $U \rightarrow V$ iff (i) the set of abstract primary variables of P is equal to V_{abs}, (ii) the set of secondary abstract variables is a subset of U_{abs} and (iii) the concrete variables have occurrences in a set $U_{con} \cup V_{con}$. Intuitively, the set U represents the *independent variables* while V represents the *dependent variables*[1]. In the absence of abstract variables, the sets of variables U and V play symmetrical roles.

[1] The definition of dependent/independent notion is related to the case statement not with respect to classical function dependency

3.4 Semantics

A Σ-structure $\mathcal{M}$ consists of:

- $\mathcal{D}$ is a carrier set defined as the union of the denotations for each Sort S i.e. $\mathcal{D} = \bigcup\{\mathcal{D}_S : S \in \mathcal{S}\}$ such that if $S \in \mathcal{S}_{abs}$ then $\mathcal{D}_S$ is non-empty set and if $S \in \mathcal{S}_{con}$ then $\mathcal{D}_S = \{a_1, \ldots, a_n\}$ where $a_i \neq a_j$ for $1 \leq i < j \leq n$.
- a n-ary function $\mathcal{M}(f) : \mathcal{D}^n \to \mathcal{D}$ for every n-ary function symbol f.
- a n-ary cross-operator $\mathcal{M}(p) : \mathcal{D}^n \to \mathcal{D}$ for every n-ary cross-operator symbol p.

We say a partial mapping $\phi : \mathcal{V} \to \mathcal{D}$ is a partial Σ-assignment iff $\phi(x) \in \mathcal{D}_{\eta(x)}$ for every variable $x \in \mathsf{Dom}(\phi)$. We assume that the structure $\mathcal{M}$ is fixed and the formal definition of the semantics relative to the mapping ϕ is:

$$
\begin{aligned}
[\![x]\!]^\phi &= \phi(x) \quad \text{for } x \in \mathcal{V} \\
[\![f(t_1, \ldots, t_n)]\!]^\phi &= \mathcal{M}(f)([\![t_1]\!]^\phi, \ldots, [\![t_n]\!]^\phi) \\
[\![x = t]\!]^\phi &= tt \text{ iff } [\![x]\!]^\phi = [\![t]\!]^\phi \\
[\![p(t_1, \ldots, t_n)]\!]^\phi &= tt \text{ iff } \mathcal{M}(p)([\![t_1]\!]^\phi, \ldots, [\![t_n]\!]^\phi) = tt \\
[\![\neg P]\!]^\phi &= tt \text{ iff } [\![P]\!]^\phi = ff \\
[\![P \wedge Q]\!]^\phi &= tt \text{ iff } [\![P]\!]^\phi = tt \text{ and } [\![Q]\!]^\phi = tt \\
[\![(\exists x : S)P]\!]^\phi &= tt \text{ iff } [\![P]\!]^{\phi[c/x]} = tt \\
&\qquad \text{for some } c \in \mathcal{D}_S \\
&\qquad \text{such that } \phi[c/x] \text{ is like } \phi \\
&\qquad \text{but maps } x \text{ to } c
\end{aligned}
$$

The remaining logical connectives are interpreted as usual.

3.5 MDG Structure

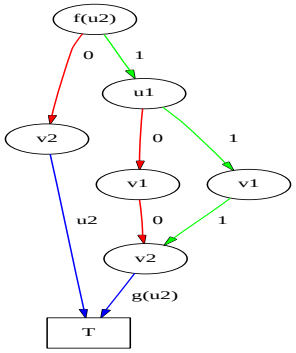
An *MDG* of type $U \to V$ is a directed acyclic graph (DAG) G with one root and ordered edges, such that:

1. Every leaf node is labeled by the formula T, except if G has a single node, which may be labeled T or F.
2. For every internal node N, either
   (a) N is labeled by $\mathcal{T}(U \cup V_{con}, \mathcal{L}_{CO}, \mathcal{S})$ and the edges that issue from N are labeled by $\mathcal{T}_G(\mathcal{S}_{con})$, or
   (b) N is labeled by a variable in V_{abs} and the edges that issue from N are labeled by $\mathcal{T}(U_{abs}, \mathcal{L}_F, \mathcal{S})$

MDG is a canonical representation of DFs and therefore must be *reduced* and *ordered* like ROBDD [1]. Consequently, DFs must obey a set of well-formedness conditions given in [2]. Some of them are already stated above. Intuitively, these conditions represent pre-conditions for some basic MDG algorithms which are mainly disjunction, relational product and pruning by subsumption. We will investigate these algorithms in next Section. In order to illustrate the above definitions, we consider the following example DF of type $\{u_1, u_2\} \to \{v_1, v_2\}$,

where u_1 and v_1 are variables of a concrete sort *bool* with enumeration $\{0, 1\}$ while u_2 and v_2 are variables of an abstract sort α, g is an abstract function symbol of type $\alpha \rightarrow \alpha$ and f is a cross-operator of type $\alpha \rightarrow bool$. The MDG of this formula is as follows:

$$((f(u_2) = 0) \wedge (v_2 = u_2)) \vee$$
$$((f(u_2) = 1) \wedge (u_1 = 0) \wedge (v_1 = 0) \wedge (v_2 = g(u_2))) \vee$$
$$((f(u_2) = 1) \wedge (u_1 = 1) \wedge (v_1 = 1) \wedge (v_2 = g(u_2)))$$



4 MDG Construction

Let P be an MDG of the form:

$$\text{MDG}(x, \{a_1, \ldots, a_m\}, \{l_1, \ldots, l_n\}, \{m_1, \ldots, m_n\})$$

then $\mathsf{top}(P)$ denotes the root node x, $\mathsf{arg}(P)$ denotes the set $\{a_1, \ldots, a_m\}$ (eventually empty) of the cross-operator arguments, $\mathsf{edges}(P)$ denotes a non-empty set $\{l_1, \ldots, l_n\}$ of labels (edges), and $\mathsf{childs}(P)$ denotes a non-empty set $\{m_1, \ldots, m_n\}$ of sub-MDGs.

In a ROBDD, Boolean variables are used to encode the enumerated types. This can be done by simply using a recursive function that divides the values into two subsets of roughly equal size, creates a variable to distinguish between them, and then recurses on the two subsets. It results in an Algebraic Decision Diagram (ADD) [16] that extends BDD's by allowing values from arbitrary finite domain to be associated with the terminal nodes. Then this ADD is translated to ROBDD. Due to the presence of abstract sorts, this approach cannot be used for MDG. Therefore, an equation (atomic formula with equality) is used to represent directly the MDG without encoding the concrete domains. We will use the notation $Eq(x, \{a_1, \ldots, a_n\}, l)$ to denote an MDG such that (i) the root node is labeled with x and the (eventually empty) set $\{a_1, \ldots, a_n\}$ (ii) the edge is labeled with l and (iii) the terminal node is labeled with T.

4.1 Generalized-If-Then-Else (GITE)

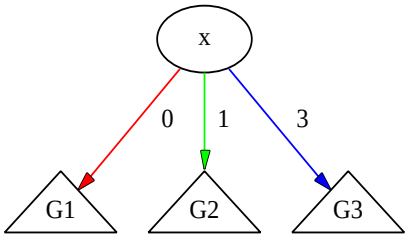
Given a ROBDD b, a boolean function f represented by b is recursively defined by:

$$f = (\neg x \wedge f_{x=0}) \vee (x \wedge f_{x=1})$$

where x is the variable in b's root node and the cofactor function $f_{x=0}$ is defined by the reachable subgraph of b's 0-branch child. Similarly, $f_{x=1}$ is recursively defined by the reachable subgraph of b's 1-branch child. Therefore a ROBDD node can be naturally represented by an If-Then-Else statement, i.e. $\mathsf{ITE}(x, f_{x=1}, f_{x=0})$.

Given a variable ordering and three ROBDDs f, g and h, the ROBDD result of f, g and h is easily constructed by Shannon's expansion in the depth-first manner. This expansion process repeats recursively following the given variable order for the Boolean variables in f, g, and h. The base case (also called the terminal case) is when f, g or h are representing a terminal node (i.e. Tor F node). For example, $\mathsf{ITE}(\mathsf{T}, g, h)$ can be trivially evaluated to g. The recursive process will terminate because restricting all the variables of functions produces constant functions T or F. At the end of the expansion phase, the uniqueness of ROBDD representation is ensured by reducing expressions like $\mathsf{ITE}(x, f, f)$ to f. This bottom-up reduction phase is performed in the reverse order of the expansion phase. Finally, since all the boolean connectives can be expressed as If-Then-Else statement, this construction provides a uniform way to build arbitrary Boolean functions.

Similarly, our goal is to provide the same construction for MDGs. The definition of the cofactor function is made upon the following observation. Assuming that x ranges over $\{0, 1, 3\}$ and that there could be, say, only three edges issuing from the root, as in the following graph:



and G_1, G_2 and G_3 represent the formulae P_1, P_2 and P_3 respectively, then this MDG could represent the formula

$$(x = 0 \wedge P_1) \vee (x = 1 \wedge P_2) \vee (x = 3 \wedge P_3)$$

When x denotes 2, this formula is simply a false sentence. Therefore, the cofactor $P_{x=l,\mathsf{arg}(x)}$ with respect to a (concrete or abstract) variable x restricted to label l and a set of the cross-operator arguments $\mathsf{arg}(x)$ (possibly empty) is defined as follows:

$$P_{x=l,\mathsf{arg}(x)} = \begin{cases} P & \text{if } x < \mathsf{top}(P) \\ m_i & \text{if } \exists i (l = l_i) \wedge (\mathsf{arg}(P) = \mathsf{arg}(x)) \\ \mathsf{F} & \text{otherwise} \end{cases}$$

While concrete sorts have enumerations, abstract sorts do not. To overcome this problem, we can collect all the labels of the abstract variable x from the MDGs

involved in the construction. This task is achieved by the function enum which is defined as:

$$\mathsf{enum}(x, P) \;=\; \begin{cases} S_{con} & \text{if } x \in S_{con} \text{ and } \mathsf{top}(P) = x \\ \mathsf{edges}(P) & \text{if } x \in S_{abs} \text{ and } \mathsf{top}(P) = x \end{cases}$$

This function exploits the variable ordering, hence there is no need to traverse all the children of P to collect the edges. The generalization of this function to a set of MDGs is defined as usual. Moreover, we assume that the set of edges are ordered.

Our GITE algorithm takes as input three MDGs P, Q and H of type $U_i \rightarrow V_i$ for $i = 1..3$ respectively and produces an MDG $R = GITE(P, Q, H)$ of type $\bigcup_{1 \leq i \leq 3} U_i \rightarrow \bigcup_{1 \leq i \leq 3} V_i$ such that $\models R \Leftrightarrow (P \wedge Q) \vee (\neg P \wedge H)$. Such MDG R does not always exist due to abstract variables. For example, let x be an abstract variable and a be an abstract generic constant. Let P be $x = a$ (i.e., an MDG with a root node labeled x and a single edge labeled a leading to T), then there is no MDG representing the formula $\neg(x = a)$. Thus there can be no algorithm for *general negation*. On the other hand, it is easy to compute a formula logically equivalent to $\neg P$ that has no nodes labeled by abstract variables. Similarly, there does not always exist an MDG R such that $\models R \Leftrightarrow (P \vee Q)$. For example, let x and y be distinct abstract variables, and a and b distinct abstract generic constants, then there exists no well-formed MDG representing the formula $x = a \vee y = b$. Finally, it may be impossible to compute the conjunction of two MDGs whose root nodes have the same label, if that label is an abstract variable (i.e., $x = a \wedge x = b$). Note all these formulae are not DFs since they do not respect the syntax constraints defined in Section 3. Moreover, we claim that the logical equivalence between R and $(P \wedge Q) \vee (\neg P \wedge H)$ can be shown independent of the negation of P, particularly when the top symbol of P is an abstract variable. For example, it is easy to show that $\models (x = a \vee x = b) \Leftrightarrow (x = a \wedge \mathsf{T}) \vee (\neg(x = a) \wedge x = b)$ in classical logic. The detailed algorithm is given below:

```
GITE(P, Q, H)
1.  if (terminal case) then
2.      return (R = trivial result);
3.  else
4.      if (computed table has entry {(P, Q, H), R}) then
5.          return R from computed table ;
6.      else
7.          x = top variables of P, Q and H;
8.          S = enum(x, P, Q, H);
9.          a = arg(x);
10.         l, m = ∅;
11.         for (each s s.t. s ∈ S) do
12.             R = GITE(P_{x=s,a}, Q_{x=s,a}, H_{x=s,a});
13.             if (R ≠ F) then
14.                 append(l,s); append(m,R);
15.             endif
```

16. endfor
17. if($l = \emptyset$) then ($R = \mathsf{F}$);
18. else R = find_or_add_unique(x, a, l, m);
19. endif
20. insert (P, Q, H, R) in the computed table
21. return R;
22. endif
23. endif

The result MDG is constructed by recursively performing Shannon's expansion. This recursive expansion ends when a terminal node is reached (lines 1 and 2) or when it is found in the computed table (line 4 and 5). A computed table stores previously computed results to avoid repeating work that was done previously. Line 7 determines the top variable of P, Q and H. Line 8 extracts a set of labels (edges) S according to the top variable sort. When this sort is concrete, then S is equal to the enumeration of this sort. Otherwise, we collect the labels from the MDGs involved in the construction. Line 9 and 10 extract eventually the arguments if the top variable is a cross-operator and initialize the new set of labels and MDGs to be constructed. Lines 11 to 16 recursively perform Shannon's expansion on the cofactor in respect to S and computes the new edges and MDGs by discarding the elements of S that result in a terminal MDG F. At the end of the expansion (line 17), either the resulting MDG is F or the reduction step and uniqueness of the resulting MDG are performed (line 18). The reduction step is applied only on the concrete sorts. Therefore a node is redundant if all the edges are in the enumeration of the concrete sort and the corresponding MDGs are the same.

Theorem 1. *The GITE algorithm is correct and terminates*[2].

4.2 Relational Product (RelP)

The relational product combines conjunction and existential quantification in one step. We provide an algorithm that extends the ROBDD relational product. It takes the conjunction of two MDGs having disjoint sets of abstract primary variables and existentially quantifies with respect to some abstract or concrete variables that have primary occurrence in at least one of the MDGs. The primary occurrence of an abstract variable in one MDG can be a secondary occurrence in the other MDG. For this reason, we have introduced a substitution that includes those variables during the construction (i.e., the secondary variables are implicitly quantified). The substitution is applied in the reverse order of the expansion phase on the edges labeled with secondary occurrence variables and cross-operators arguments. However, while the ordering variable cannot be preserved in case of cross-operators, there may exist redundant or contradictory

[2] The correctness proof of all the algorithms is included in a technical report[19]

MDG result during intermediate steps. For example, let $x < m < M$ be an ordering variables and let P be $leq(x, m) = 1 \wedge leq(x, M) = 0$ where x, m and M are secondary abstract variables that having primary occurrences in another MDG, say, Q, and $\sigma = \{x \mapsto x\#3, m \mapsto x\#2, M \mapsto x\#1\}$, then the resulting MDG $leq(x\#3, x\#2) = 1 \wedge leq(x\#3, x\#1)) = 0$ does not preserve the order[3]. Therefore, we will distinguish the case of the cross-operator and provide a special construction for it.

Let E be the set of quantified variables, our algorithm takes two MDGs P, Q of type $U_i \rightarrow V_i$ for $i = 1..2$ and a substitution σ with $\mathsf{Dom}(\sigma) = E$ and returns an MDG $R = \mathsf{RelP}(P, Q, E, \sigma)$ of type $(\bigcup_{1 \leq i \leq 2} U_i \setminus \bigcup_{1 \leq i \leq 2} V_i) \rightarrow (\bigcup_{1 \leq i \leq 2} V_i \setminus \bigcup_{1 \leq i \leq 2} U_i)$ such that $\models R \Leftrightarrow \exists E(P \wedge Q)$.

$\mathsf{RelP}(P, Q, E, \sigma)$
1. if (terminal case) then
2. return (R = trivial result);
3. else
4. if (computed table has entry $\{(P, Q, E, \sigma), R\}$) then
5. return R from computed table ;
6. else
7. x = top variables of P, Q
8. $S = \mathsf{enum}(x, P, Q)$;
9. $a = \mathsf{arg}(x)$;
10. $l, m = \emptyset$;
11. for (each s s.t. $s \in S$) do
12. $R = \mathsf{RelP}(P_{x=s,a}, Q_{x=s,a}, E, Extend(\sigma, x, s, E))$;
13. if ($R \neq \mathsf{F}$) then
14. append(l,s); append(m,R);
15. endif
16. endfor
17. if($l = \emptyset$) then ($R = \mathsf{F}$);
18. else
19. if($x \in E$) then
20. $R = Or(m)$
21. else
22. if($a = \emptyset$) then
23. R = find_or_add_unique$(x, a, \sigma(l), m)$;
24. else
25. $R = \mathsf{F}$
26. for (each $l_i \in l$ and $m_i \in m$)
27. $R = Or(R, And(Eq(x, \sigma(a), l_i), m_i))$
28. endfor
29. endif
30. endif

[3] the variable $x\#i$ serves as a symbolic value of x at the i^{th} step and $i < j \Rightarrow x\#i < x\#j$

31. endif
32. insert (P, Q, E, σ, R) in the computed table
33. return R
34. endif
35. endif

Like ROBDD relational product algorithm, RelP uses a result cache. If the entry (P, Q, E, σ) is in the cache, then it means that a previous call to $\mathsf{RelP}(P, Q, E, \sigma)$ returned R as result. Lines 7 and 16 apply recursively the relational product with respect to a top symbol x where $Extend(\sigma, x, s, E)$ returns $\sigma \oplus \{s/x\}$ if $x \in E$ otherwise it returns σ. Lines 19 to 31 apply either quantification or conjunction depending whether the variable x occurs in E or not. As explained above, we distinguish the cross-operators case (lines 25 to 28), where we construct a new MDG that respects the ordering variable, thus avoiding any contradictions.

Theorem 2. *The* RelP *algorithm is correct and terminates*

4.3 Pruning by Subsumption (PbyS)

The pruning by subsumption algorithm approximates the difference of sets represented by MDGs (i.e. DFs). We propose a new algorithm which uses restricted operators and builds an MDG in a similar manner as GITE does. The proposed algorithm improves the original one in many ways. First, the expansion is done only on the first argument i.e., P rather than on P and Q. Indeed, we can view each disjunct of DF as a state description. Without loss of generality, we can assume that P and Q contain only one disjunct. Then, we can say that P is subsumed by Q if and only if there exists a substitution σ such that the state description of $Q\sigma$ is a subset of the state description of P. Therefore the size of P should be at least equal to the size of Q. Next, when the top variable of Q is less than the top variable of P, it is obvious that the state description of Q is not a subset of P. Hence, the cofactor of Q should be F, which improves drastically the original algorithm. Finally, when P and Q have the same top symbol cross-operator but there is a mismatch either on the edges or on the arguments, the cofactor of Q is Q itself and we discard the substitution if any resulting from the unification of their arguments. These observations lead to a new restricted operator defined as follows.

Given an MDG Q, the restriction of Q with respect to a variable x, an edge l, a set of cross-operator arguments $\mathsf{arg}(x)$ and a substitution σ, written $Q|_{x=l,\mathsf{arg}(x),\sigma}$, returns a pair of MDG-substitution $\langle m, \sigma' \rangle$ as:

$$\begin{cases} \langle Q, \sigma \rangle & \text{if } x < \mathsf{top}(Q) \\ \langle \mathsf{F}\ , \sigma \rangle & \text{if } \mathsf{top}(Q) < x \\ \langle m_i, \sigma' \rangle & \text{if } (\exists i)(l = l_i\sigma') \wedge \mathsf{arg}(Q) = \mathsf{arg}(x) = \emptyset \\ \langle Q, \sigma \rangle & \text{if } (\nexists i)(l = l_i\sigma') \wedge \mathsf{arg}(Q) = \mathsf{arg}(x) = \emptyset \\ \langle m_i, \sigma'' \rangle & \text{if } \exists i(l = l_i\sigma'') \wedge (\mathsf{arg}(Q)\sigma'' = \mathsf{arg}(x)) \\ \langle Q, \sigma \rangle & \text{if } \nexists i(l = l_i\sigma'') \vee (\mathsf{arg}(Q)\sigma'' \neq \mathsf{arg}(x)) \\ \langle \mathsf{F}\ , \sigma \rangle & \text{otherwise} \end{cases}$$

where $\sigma' = \sigma \oplus \{l_i \mapsto l\}$ and $\sigma'' = \sigma \oplus \{\mathsf{arg}(Q) \mapsto \mathsf{arg}(x)\}$.

Our PbyS algorithm takes two MDGs P and Q of type $U \to V_1$ and $U \to V_2$ and a substitution σ initially equal to the identity and produces an MDG P' of type $U \to V_1$ such that P' is derivable from P by *pruning* some paths such that $\models P \vee (\exists U)Q \equiv P' \vee (\exists U)Q$. The paths that are removed from P are subsumed by Q, hence the name of the algorithm. If $P' = \mathsf{F}$ then, we can view P' as a logical difference of P and $(\exists U)Q$ i.e. $\models P \Rightarrow (\exists U)Q$. The detailed algorithm is given below:

```
PbyS(P, Q, σ)
1.  if (terminal case) then return (P' = trivial result);
2.  else if (pbys table has entry {(P, Q, σ), P'}) then
3.          return P' from pbys table ;
4.  else
5.      x = top(P); l, m = ∅; a = arg(P);
6.      for (each s ∈ edges(P)) do
7.          P' = P_{x=s,a};
8.          stack = Q|_{x=s,a,σ};
9.          while stack is not empty;
10.             ⟨m', σ'⟩= pop stack;
11.             P' = PbyS(P', m', σ');
12.             if (P' = F) break;
13.         endwhile;
14.         if(P' ≠ F) then
15.             append(l,s); append(m,P');
16.         endif
17.     endfor;
18.     if(l = ∅) then (P' = F);
19.     else P' = find_or_add_unique(x, a, l, m);
20.     update pbys table ({(P, Q, σ), P'}) ;
21.     return P';
22. endif
```

The result MDG is constructed by recursively performing the restricted operators introduced on P and Q until a terminal node is reached (line 1) or when it is found in the pbys table (line 2). Line 5 determines the top variable of P and the cross-operator arguments (if possible) and initializes the new edges and children to be constructed. Then from each edge issuing from the node x (line 6), we extract the cofactors of P and Q where the cofactors of Q are pairs of MDG-substitution stored in a stack. Lines 9 to 13 check whether the cofactors of P, written P', is subsumed by one of the Q paths. If so (line 12) then there is no need to try the other cofactors of Q and therefore we continue with the remaining cofactors of P and we discard P'. Otherwise, the edge and this cofactor are added to the corresponding table (lines 14-16). When we have processed all the cofactors of P (line 18) either all the paths of P are subsumed by P and

thus the result MDG is F, or the reduction step and uniqueness of the resulting MDG are performed (line 20) with all or some paths of P that not subsumed.

Theorem 3. *The* PbyS *algorithm is correct and terminates*

5 NuMDG Structure

A high level description of NuMDG is given in Figure 1. In the future, we will provide an open source tool with many functionalities independent of the model checking engine used. Like NuSMV [17], the tool will be able to process files written in an extension of the SMV language with abstract sort and uninterpreted functions. In this language, finite state machines are described by using instantiation mechanism of modules and processes, corresponding to synchronous and asynchronous composition respectively. The requirements are written in CTL, LTL or in a first-order subset of temporal logic.

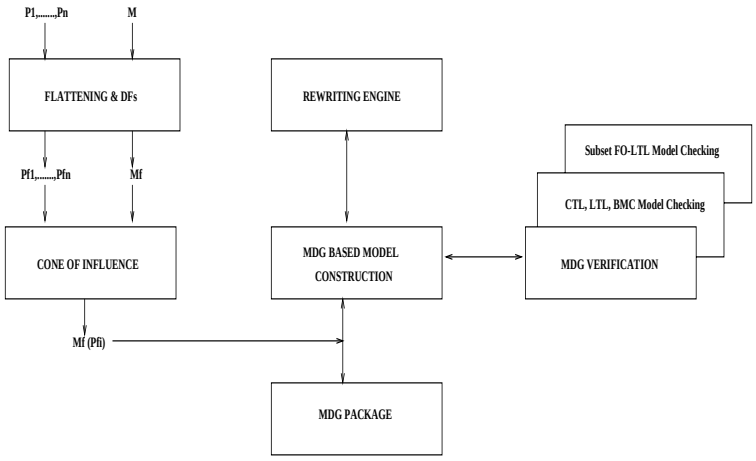


Fig. 1. Internal structure of NuMDG

An (extended) SMV file is processed in several phases. The first phase analyzes the input file with different layers in order to construct an internal representation of the model. The construction starts from modular description of a model M and of a set of properties $P_1, \ldots P_n$. The flattening step consists of eliminating modules and processes and producing a synchronous flat model, where each variable is given an absolute name. The second step, called DF, maps each expression in the flat model to a directed formula, thus obtaining the corresponding flattened directed model M_f. Compared to SMV-based tools, there is no boolean encoding. Hence, some interpreted predicates and arithmetic functions are not supported in a straightforward manner. The same reduction is applied to the properties P_i, thus obtaining the corresponding flattened directed

formula P_{if}. By cone of influence, we restrict the analysis of each property to the relevant parts of the model $M_f(P_{if})$.

After the preprocessing phase, the user can choose the model checking engine to be used for verification. The choice is restricted by the nature of the model being described i.e. whether it supports abstract sorts and uninterpreted functions or not. In the absence of the latter, NuMDG is acting like NuSMV and should provide the same facilities including MDG-based, SAT-based model checking and different partitioning methods. For the time being, MDG-based verification includes reachability analysis and fair CTL model checking.

The rewriting engine is used during the MDG-verification if necessary when the reachability analysis does not terminate due to the presence of abstract sort and uninterpreted functions. In this case we can interpret partially some functions or predicates in order to cope with this non termination [18]. The input language supports a rewriting layer which is extracted and feeded to the rewriting engine. Currently, we are working to complete the infrastructure shown in Figure 1.

6 Conclusion and Future work

We have described the basic MDG algorithms that incorporated many optimizations that will yield further improvements in the performance of MDG package. The efficiency is achieved through the use of the generalization of the If-Then-Else (ITE) operator defined in the BDD package. Consequently, we have redefined the main algorithms on which the MDG verification techniques are based, i.e, relational product and pruning by subsumption. These new algorithms descriptions are based mainly on the ROBDD ones and lifted to the realm of abstract sorts and uninterpreted functions.

We have also presented the internal architecture of the NuMDG tool and identified a number of open issues and future work directions. We need to complete the implementation and confirm that NuMDG can be used to check SMV specifications. However, the effect of cache and the garbage collection should be characterized according to a rigorous evaluation methodology. Also case studies and experiments are required to check the new tool and compare the results with SMV.

References

1. Bryant, R.E.: Graph-based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, 1986, 35, (8), pp. 677-691
2. Corella, F., Zhou, Z., Song, X., Langevin, M., and Cerny, E.: Multiway Decision Graphs for Automated Hardware Verification, Formal Methods in System Design, 1997, 10, (2), pp. 7-46
3. Tahar, S., Song, X., Cerny, E., Zhou, Z., Langevin, M., and Ait Mohamed, O.: Modeling and Verification of the Fairisle ATM Switch Fabric using MDGs, IEEE Transactions on CAD of Integrated Circuits and Systems, 1999, 18, (7), pp. 956-972

4. Xu, Y., Cerny, E., Song, X., Corella, F., and Ait Mohamed, O.: Model Checking for A First-Order Temporal Logic using Multiway Decision Graphs, The Computer Journal, 2004, 47, (1), pp. 71-84
5. Zhou, Z.: Mutliway Decision Graphs and Their Applications in Automatic Formal Verification of RTL Designs, PhD thesis, Montréal University, 1997
6. Burch, J.R. and Dill, D.L.: Automatic Verification of Pipelined Microprocessor Control. Proc. of Work on CAV, LNCS, vol. 818, 1994, pp. 68-80
7. Damm, W., Pnueli, A., and Ruah, S.: Herbrand Automata for Hardware Verification. Proc. of the 9th International Conference on Concurrency Theory, LNCS 1466, 1998, pp. 67-83
8. Berezin, S., Biere, A., Clarke, E.M., and Zhu, Y.: Combining Symbolic Model Checking with Uninterpreted Functions for Out-of-Order Processor Verification, Formal Methods in Computer Aided Design, 1998, 1522, pp. 187-201
9. Hojati, R., Kuehlmann, A., German, S., and Brayton, R.K.: Validity Checking in the Theory of Equality with Uninterpreted Functions using Finite Instantiations. The International Workshop on Logic Synthesis, 1997
10. Goel, A., Sajid, K., Zhou, H., Aziz, A., and Singhal, V.: BDD based Procedures for A Theory of Equality with Uninterpreted Functions. Proc. CAV, LNCS 1427, 1998, pp. 244-255
11. Bryant, R.E., German, S. and Velev, M.N.: Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic ACM Transactions on Computational Logic, 2001, 2, (1), pp. 93-134
12. Ackermann, W.: Solvable Cases of the Decision Problem (North-Holland Pub. Co., 1954)
13. Pnueli, A., Rodeh, Y., Shitrichman, O., and Siegel, M.: Deciding Equality Formulas by Small Domain Instantiations. Proc. CAV, LNCS, vol. 1633, 1999, pp. 455-469
14. Velev, M.N.: Using Rewriting Rules and Positive Equality to Formally VerifyWide-issue Out-of-Order Microprocessors with Reorder Buffer. Proc. of DAC, 2002, pp. 28-35
15. Clocksin, W., and Mellish, C.: Programming in Prolog (Springer-Verlag, 3rd edition, 1987)
16. Bahar, R., Frohm, E., Gaona, C., Hatchel, G., Macii, E., Pardo, A., and Sommenzi, F.: Algebraic Decision Diagrams and their Applications. Proc. of International Conference on Computer-Aided Design, 1993, pp. 188-191
17. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A.: NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. Proc. of International Conference on CAV, LNCS, vol. 2404, 2002
18. Ait Mohamed, O., Song, X., Cerny, E.: On the Non-termination of MDG-based Abstract State Enumeration. Theoretical Computer Science, 2003, 300, pp. 161-179
19. Mokhtari, Y., Abed, S., Ait Mohamed, O., Tahar, S., and Song, X.: A New Approach for the Construction of Multiway Decision Graphs. Technical Report 2008-3-Abed, ECE Department, Concordia University, Montreal, Canada, June 2008.