URL: http://www.elsevier.nl/locate/entcs/volume23.html 18 pages

Improving Termination of MDG-Based Abstract State Enumeration via Term Schematization

S. Renault and E. Cerny [1]

D'IRO, Université de Montréal
C.P. 6128, Succ. Centre-Ville,
Montréal, H3C 3J7, Canada

Abstract

The MDG approach for hardware verification was recently proposed to overcome some limitations of traditional ROBDD-based methods of automated verification. One of its weakness, however, is that it may suffer from the non termination of the state enumeration procedure. In this paper, we exploit an idea recently proposed of using term schematization to address this problem. We propose a strategy to direct the introduction of schematized terms in the state enumeration procedure. We show that for certain cases of non termination, it significantly improves over the previously known approaches.

1 Introduction

The MDG approach for hardware verification [12] has been introduced to enhance and overcome some limitations of traditional ROBDD-based methods of automated verification [7]. It is based on *abstract descriptions of state machines* (ASM) which are represented by a class of decision graphs called *Multiway Decision Graphs* (MDGs). MDGs can represent relations as well as sets of states and unlike ROBDDs, can support the use of abstract data representations. More precisely, MDGs allow the use of abstract sorts and uninterpreted functions to represent data values and data operations, respectively. To limit the range of possible interpretations of the function symbols that denote data operations, it is possible to define axioms which (partially) specify their intended semantics. These axioms are given by means of (conditional) rewriting rules.

[1] Acknowledgments: the work was partially supported by NSERC Canada - Nortel Cooperative R&D grant No. CRD 191958. The work was carried out using a workstation on loan from the Canadian Microelectronics Corp.

The main strength of the MDG approach is to integrate the combination of two powerful techniques, namely the implicit state enumeration and the use of abstract sorts and uninterpreted function symbols. MDGs have been implemented and many experiments have been carried out [6,13,18,21,22], showing that they provide a powerful tool for fully automatic combinational verification, state space exploration, verification of behavioral equivalence of sequential circuits. Model checking for a first-order temporal logic using MDGs has also been proposed to express both safety and liveness properties [19].

Its main weakness, however, is that in many cases it may suffer from the non termination of the state enumeration procedure. Different solutions have been proposed to solve certain cases of non terminating problems. In [12], a solution is given to a class of problems known as *processor-like circuits*. It relies upon the generalization of the set of initial states so as to obtain a larger set of reachable states that is representable by a finite MDG. Technically, the initial state generalization method is done by replacing abstract constants with abstract variables as initial values of some registers. An extension of this method has also been proposed [2] that applies to circuits in which the *processor-like* loop does not start necessarily at the initial state, but instead at any other control state. It can therefore deal with a larger class of circuits, but as a counterpart, it requires to detect when the generalization is to be performed, i.e., to find the entry states of the processor-like loops. State generalization has been used successfully to achieve termination of the abstract state enumeration on several examples. Sometimes this achievement is at the cost of a false negative during invariant checking, i.e., the enlarged set of reachable sets may include states that do not correspond to the intended interpretation of the data operations and for which the invariant to verify does not hold.

Another approach for solving non termination is presented in [1]. It is based on retiming and semantics preserving transformations of the circuit to verify. It uses generalization of state variables as in the previous approaches, but to avoid the loss of information in the generalization step, the value of the state variable to be generalized is kept in a new state variable so that it can be recovered later. This method requires the user to have a good understanding of the circuit under verification and the skills to perform on it the adequate transformations.

In [3] a new approach to state generalization based on term schematizations has been introduced. The authors show how schematization can be incorporated into MDG to solve some cases of non termination due to a state variable which produces an infinite sequence of structurally similar terms. One of the main objectives of this approach is to reduce as much as possible the amount of information lost during generalization. Schematization exploits the recurring term structure in infinite sequences of terms, in our case the successive values taken by a state variable, and provides the least common generalization subsuming those terms. This allows not only to maintain the possibility

of using some information provided by a partial interpretation of the symbols occurring in the schematization, and therefore to increase the chance of achieving termination, but also to reduce considerably the risk of producing false negatives. The generalization method proposed in [3] requires the user to supply both the state variable(s) to generalize and the appropriate schematization. Moreover, the generalization is always done on the initial state which might be an obstacle to some simplification via rewriting, as it is the case in the initial state generalization method. In this paper, we propose a strategy which improves over the approach of [3] and which can deal with more complicated cases of non termination. In particular, our strategy does not require human guidance: the introduction and the handling of schematized terms during reachability analysis is performed fully automatically. Moreover, generalization is not performed at the initial step so that one can still fully benefit from a partial interpretation of the abstract functions on the initial values. Another advantage of postponing the generalization is to allow a state variable to be generalized by different term schematizations in alternative branches of the state exploration. Hence, our strategy allows to cope with situations where a state variable gives rise to different patterns of divergence in alternative branches of the state exploration.
This paper is organized as follows: In Section 2, we briefly recall the concept of Multiway Decision Graphs and its application to the state exploration of Finite State Machines. In Section 3, we recall some notions and operations on recurrence-terms. In Section 4, we propose an extension of MDGs and of the state exploration procedure based on recurrence-terms. Section 5 is devoted to some examples.

2 MDGs and state exploration

In this section we briefly present the framework of MDGs. For more details, we refer the interested readers to [8,12].

2.1 Syntax and semantics

MDGs are Directed Acyclic Graphs (DAGs) which represent formulas in a variant of first-order logic with equality and sorts, with a distinction between *concrete* sorts and *abstract* sorts. This distinction is a syntactic counterpart of the hardware difference between data-path and control. Unlike abstract sorts, concrete sorts have *enumerations* which are sets of *individual constants.* MDGs are limited to a class of quantifier free formulas, called the *Directed Formulas* (DFs), which satisfy several well-formedness conditions, in particular *canonicity* (see [12] for details).

Definition 2.1 (Directed Formula) *Let $\mathcal{F}$ be a set of function symbols and $\mathcal{V}$ a set of variables. We denote the set of terms freely generated from $\mathcal{F}$ and $\mathcal{V}$ by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The syntax of a directed formula is then given by the grammar* *below:*

$$\begin{array}{lll}
\textit{Sort} & \mathcal{S} & ::= S \mid \underline{S} \\
\textit{Abstract sort} & S & ::= \alpha \mid \beta \mid \gamma \mid \ldots \\
\textit{Concrete sort} & \underline{S} & ::= \underline{\alpha} \mid \underline{\beta} \mid \underline{\gamma} \mid \ldots \\
\textit{Generic constant} & C & ::= a \mid b \mid c \mid \ldots \\
\textit{Concrete constant} & \underline{C} & ::= \underline{a} \mid \underline{b} \mid \underline{c} \mid \ldots \mid \underline{0} \mid \underline{1} \mid \ldots \\
\textit{Variable} & \mathcal{X} & ::= V \mid \underline{V} \\
\textit{Abstract variable} & V & ::= x \mid y \mid z \mid \ldots \\
\textit{Concrete variable} & \underline{V} & ::= \underline{x} \mid \underline{y} \mid \underline{z} \mid \ldots \\
\textit{Directed Formula} & \textit{Disj} & ::= \textit{Conj} \vee \textit{Disj} \\
 & \textit{Conj} & ::= \textit{Eq} \wedge \textit{Conj} \mid \textit{Eq} \\
 & \textit{Eq} & ::= \underline{A} = \underline{C} \quad (A \in \mathcal{T}(\mathcal{F}, \mathcal{X})) \\
 & & \quad \mid \underline{V} = \underline{C} \\
 & & \quad \mid V = A \quad (A \in \mathcal{T}(\mathcal{F}, \mathcal{X})) \\
 & & \quad \mid \top \\
 & & \quad \mid \bot
\end{array}$$

The vocabulary consists of generic constants, concrete constants, abstract variables, concrete variables and function symbols. A DF is a formula in disjunctive normal form, each disjunct is a conjunction of equations that corresponds to an alternative *path.* Each equation corresponds to a node-edge pair in the corresponding path, the left-hand side (resp. the right-hand side) of the equation being the label of the node (resp. the edge).

The variables which occur as the label of a node are the *primary variables*, while the other variables are the *secondary variables.*

A DF D is of type $\mathcal{U} \rightarrow \mathcal{V}$ iff (i) the set of abstract primary variables of D is equal to $\mathcal{V}_{abs}$, (ii) the set of abstract secondary variables of D is a subset of $\mathcal{U}_{abs}$, and (iii) the set of concrete variables having occurrences in D is a subset of $\mathcal{U}_{conc} \cup \mathcal{V}_{conc}$.

The basic MDG algorithms which serve in the procedures for combinational verification and reachability analysis include the operations of *Disjunction, Relational product* and *Pruning by subsumption.* A detailed description of these algorithms, and others like *Conditional term rewriting* can be found in [13,21].

Since abstract sorts admit infinite interpretations, an abstract description will represent infinite as well as finite state machines (FSMs). An abstract description will determine a machine for every interpretation ψ. MDGs are well-suited to represent both sets of states and transition/output relations of state machines. Let P be an MDG of type $\mathcal{U} \rightarrow \mathcal{V}$, where $\mathcal{U}$ contains only abstract variables. Then, for a given interpretation ψ, P can be used to represent the set

(1) $Set^{\psi}(P) = \{\phi \in \Psi^{\psi}_{\mathcal{V}} \mid \psi, \phi \models \exists\, \mathcal{U}.\ P\}$

where $\Psi^{\psi}_{\mathcal{V}}$ denotes the set of ψ-compatible assignments of the variables in $\mathcal{V}$, that is, the set of functions ϕ that map every variables $x \in \mathcal{V}$ of sort α to an element $\psi(x)$ of $\phi(\alpha)$.
Note that although the formulas represented by MDGs are quantifier free, it is useful to think of the secondary variables as being existentially quantified, which amounts to viewing the formula as having an existential prefix.

2.2 MDG-based state exploration

Abstract state exploration is the kernel of the MDG tools. In particular, it is used by the reachability analysis procedure which verifies that after each transition step the outputs satisfy a given property. This procedure is described by the pseudo-code below, where $D = (X, Y, Z, F_I, F_T, F_O)$ is an abstract state machine description in which X, Y, Z are disjoint sets of variables, viz. the input, state, and output variables, respectively, and F_I, F_T, F_O are DFs representing the set of initial states, the transition relation and the output relation respectively.

```
Proc ReAn(D)
    $R := F_I$; $Q := F_I$; $K := 0$;
    loop
        $K := K + 1$;
        $I :=$ Fresh$(X, K)$;
        $N :=$ RelP$(\{I, Q, F_T\}, X \cup Y, \eta)$;
        $Q :=$ PbyS$(N, R)$;
        if $Q = \bot$ then return success;
        $R :=$ PbyS$(R, Q)$;
        $R :=$ Disj$(R, Q)$;
    end loop;
end ReAn;
```

R represents the set of visited states and Q the current frontier set. State exploration starts from a set of initial states which forms the initial frontier set. Then the Relational Product (RelP) operation computes the MDG N representing the set of states reachable after one transition from the frontier set Q according to the transition relation. The *Fresh* operation ensures that all input variables are renamed into new fresh variables at each iteration of the procedure. Then the Pruning-By-Subsumption (PbyS) algorithm removes, by subsumption, from the newly reached states N, those that are also in R, i.e. that have already been reached in a previous state. The states that are not removed form the new frontier from which state exploration continues. State exploration terminates when all the states reachable from the frontier states have been visited. Informally, PbyS performs paths subsumption as follows. Let π_1 and π_2 be two paths from two MDGs of type $\mathcal{U} \to \mathcal{V}$ corresponding to the states s_1 and s_2 respectively. By definition, π_1 is subsumed by π_2 if there exists a substitution σ with domain $\mathcal{U}$ such that if π_2' is obtained by applying σ to the nodes and edges of π_2, then every node-edge pair in π_1 is a node-edge pair in π_2'. A complete description of the PbyS algorithm and other operations on MDGs can be found in [13,21].

In general, because of abstract variables and the uninterpreted nature of function symbols, the state exploration algorithm may not terminate. It may be caused by a state variable whose successive values grow infinitely according to a certain recursive pattern. Consider for instance an abstract description of a conventional microprocessor where a state variable c of abstract sort *wordn* (for n-bit words) with an initial value equal to a generic constant *zero* of the same sort is incremented via an abstract function symbol s representing the incrementation by one. Then, the state exploration would explore the infinite sequence $c = zero$, $c = s(zero)$, $c = s(s(zero))$, $c = s(s(s(zero)))$, ...

In the remainder of the paper we propose and illustrate a solution to this kind of non termination based on term schematization which improves over the previous proposal by Ait et *al* [3].

3 Recurrence-terms

Non terminating computations due to infinite sequences of structurally similar terms is a problem which arises in various fields using first-order terms. Term schematization has been proposed to solve this problem by using a meta-language based on *recurrence-terms* which are capable of finitely representing infinite sets of terms. As an example, the recurrence-term $A = \Phi(s(\diamond), N, zero)$ represents the infinite set $S = \{zero, s(zero), s(s(zero)), \ldots\}$. N is a variable which ranges over natural numbers and which serves as a counter specifying the iteration level. The instance of A via the substitution $5/N$ is $s(s(s(s(zero)))))$.

Various schematization formalisms, based on the same principle of the iteration of first-order contexts, have been proposed [10,11,14,17] which mainly differ in the expressive power. Dropping restrictions may increase the expressive power but affect the complexity and/or decidability of algebraic operations, in particular unification.

In this paper, we restrict ourselves to simple forms of recurrence-terms, the *ρ-terms* as defined in [10]. We now recall the main definitions and concepts of this formalism.

Syntax

Let $\mathcal{F}$ be a finite set of function symbols, $\mathcal{V}$ be a countable set of first-order variables, Φ be a special symbol of arity 3, $\diamond$ be a special constant symbol serving as a place holder, and let $\mathcal{V}_\Phi$ be a countable set of variables over natural numbers, called *degree variables*. We assume that $\mathcal{V} \cap \mathcal{V}_\Phi = 0$. A *position* u in a term t is either the root position denoted by the empty string λ, or a non root position denoted by $i.u_i$, if t is of the form $f(t_1, \ldots, t_i, \ldots, t_n)$ and u_i is a position in t_i. We denote by t/u the term which occurs in t at position u and by $t[s]_u$ the term obtained by replacing t/u by s.

Definition 3.1 (ρ-term) *A ρ-term is defined inductively as follows:*

- *Every variable in $\mathcal{V}$ is a ρ-term.*
- *If $f \in \mathcal{F}$ has arity n, and $t_1, \ldots, t_n$ are ρ-terms, then $f(t_1, \ldots, t_n)$ is a ρ-term.*
- *If u is a non root position of h, $h[l]_u$ is a first-order term, and N is a degree variable, then $\Phi(h[\diamond]_u, N, l)$ is a ρ-term.*

$\Phi(h[\diamond]_u, N, l)$ *is called a* generator*,* $h[\diamond]_u$ *represents the recurring pattern and is referred to as the* iterated context*, and l is the non-recurring part of the term, usually called the* base term.

Semantics

A ρ-term represents either a first-order term, when its degree variables are all constants, or an infinite set of first-order terms when it contains a non instantiated degree variable. We denote by $H[n_1/N_1, \ldots, n_m/N_m]$ the ρ-term obtained from the ρ-term H by instantiating the degree variables $N_1, \ldots, N_m$ to the natural numbers $n_1, \ldots, n_m$ respectively, and then unfolding the instantiated ρ-term.

Definition 3.2 (Unfolding) *Unfolding is defined as follows:*

$$\Phi(h[\diamond]_u, 0, l) \overset{def}{=} l$$
$$\Phi(h[\diamond]_u, N+1, l) \overset{def}{=} h[\Phi(h[\diamond]_u, N, l)]_u$$

As an example, $\Phi(f(\diamond), N, g(a))[3/N] = f(f(f(g(a))))$.

Definition 3.3 (Recurrence-domain) *The* recurrence-domain *of a ρ-term H is the set $\Omega(H)$, obtained by unfolding H for all possible values of its degree variables. The elements of $\Omega(H)$ are called the* constituents *of H.*

Note that different ρ-terms may have the same recurrence-domain. For instance, the two ρ-terms $f(\Phi(f(\diamond), N, a))$ and $\Phi(f(\diamond), N, f(a))$ have the same recurrence-domain, namely $\{f(a), f(f(a)), f(f(f(a))), \ldots\}$.

Unification and matching

The unification of ρ-terms, also called ρ-unification, differs from unification of terms in that unifications are performed on sets of terms instead of terms. A ρ-substitution is a pair of substitutions whose domains are the first-order variables and the degree variables respectively. Two ρ-terms have a most general ρ-unifier, but it may not be unique. In [10], an algorithm which computes a *finite* and *complete* set of ρ-unifiers is given. Informally, $\{\Theta_1, \ldots, \Theta_m\}$ is a complete set of unifiers of two ρ-terms L and R if $\Theta_1, \ldots, \Theta_m$ cover all the most general unifiers of l and r, for all pairs $(l, r) \in \Omega(L) \times \Omega(R)$. It is shown in [10] that computing the intersection of two recurrence-domains is the same as finding a complete set of unifiers between two ρ-terms whose degree variables are disjoint.
In [15], an algorithm is given to check that a ρ-term T subsumes a ρ-term S. The subsumption problem is reduced to the unification problem by replacing all first-order variables in S by new constants and then applying a unification algorithm. The subsumption of ρ-terms is also addressed in [9] and an algorithm is given for the particular case where one of the two ρ-terms is a (first-order) term. As it can be seen above, we have that if a ρ-term T subsumes a ρ-term S, each term in $\Omega(S)$ is an instance of a term in $\Omega(T)$. However, the converse does not hold. Let us consider the ρ-terms $T = \Phi(f(\diamond), N, f(a))$ and $S = f(\Phi(f(\diamond), M, a))$, we have that $\Omega(S) = \Omega(T)$, but there is no ρ-substitution σ such that $T = S\sigma$ or $S = T\sigma$. This is in contrast with the first-order case, where a term t subsumes a term s iff s is an instance of t. The inclusion of ρ-terms is also a decidable problem [4]. A simple algorithm is given in [5] in the case where S and T do not contain any first-order variables.

4 Recurrence-terms in MDGs

4.1 Intuition

By introducing ρ-terms in MDGs, we intend to generalize a state by an infinite set of states whose state variables have values that share structural similarities. For instance, let us consider again the microprocessor of Section 2.2. After the k-th instruction has been executed, the machine is in a state represented by the equation $c = s(s(\ldots s(zero)\ldots))$, where s is iterated k times. We may generalize the set of states visited along the sequence (path) to c by replacing the value of c by the infinite set of terms$\{zero, s(zero), s(s(zero)), s(s(s(zero))), \ldots\}$ represented finitely by the ρ-term $\Phi(s(\diamond), N, zero)$.

4.2 Syntax and semantics

We extend the definition of DF to that of ρ-DF by allowing occurrences of ρ-terms in the right-hand side of the equations. In addition to the primary and secondary variables, we consider a third type of variables, namely the degree variables that occur in the ρ-terms of a ρ-DF, and which range over natural numbers. Let P be a ρ-DF of type $\mathcal{U} \rightarrow \mathcal{V}$, such that $\Omega(P) = \{P_1, P_2, P_3, \ldots\}$. Then for a given interpretation ψ, P represents the set

(2) $Set^{\psi}(P) = \{\phi \in \Psi^{\psi}_{\mathcal{V}} \mid \psi, \phi \models \exists\ i.\ \exists\ \mathcal{U}.\ P_i\}$

Note that (2) differs from (1) in Section 2.1, by the additional quantification over the degree variables occurring in P. Intuitively, it seems natural that the degree variables are existentially quantified since they can be seen as a particular case of secondary variables ranging over natural numbers.
A generator $\Phi(h[\diamond]_u, N, l)$ can be viewed as an ordinary MDG term, where $\diamond$ is a constant symbol of concrete sort, and Φ an abstract function symbol of arity 3 of abstract sort, the sort of the elements in its recurrence-domain. We assume that those elements are all of the same sort [2]. Since a ρ-term can occur only in the right-hand side of an equation, neither N nor Φ participates in the custom symbol order. One can verify that the well-formedness conditions of MDGs [12] can be extended to ρ-terms in a straightforward way. By considering the ρ-terms occurring in a ρ-MDG as ordinary MDG terms, we can directly apply to ρ-MDGs the basic operations defined for ordinary MDGs. However, in the case of the Pruning-By-Subsumption operation, we have to extend the ordinary subsumption, so as to be able to recognize that the states represented by the formula $x = s(\Phi(s(\diamond), N, zero))$ are subsumed by those represented by the formula $x = \Phi(s(\diamond), N, zero)$. This is discussed in the rest of this section.

4.3 State exploration for ρ-MDGs

We have seen in Section 2.2, that the set of new reachable states in MDG is based on the PbyS algorithm which applies paths subsumption to remove redundant states. Technically, to check whether a path π_i subsumes a path π_j, it looks for a (sub)path π'_j, obtained from π_j by removing some node-edge pairs, such that π_i subsumes π'_j. Let us consider again the example given at the beginning of this section where, after generalization, we get the ρ-MDG $s_k :\ c = \Phi(s(\diamond), N, zero)$. Now, the states reachable from s_k after one transition step are represented by the ρ-MDG $s_{k+1} :\ c = s(\Phi(s(\diamond), N, zero))$. By replacing the ordinary subsumption in PbyS by the ρ-subsumption, we have that the path s_{k+1} is subsumed by s_k via the substitution $N+1/N$ and therefore the state exploration terminates [3].
Note that, since set inclusion and term subsumption are not equivalent notions in the algebra of ρ-terms, the PbyS algorithm based on the ρ-subsumption might not be able to detect all states of the frontier that belong to the set of visited states. However, since we can only detect some particular cases of

[2] Note that we could easily deal with the case where the base term and the recurrence-terms are of different sorts, by considering the degree variables in ρ-terms to range over non null natural numbers. This variation is actually used in [4,11] to avoid the association between unrelated terms.
[3] Note that the degree variables have to be considered as existential variables and therefore can be renamed in two different states.

non termination, we have to evaluate the practical gain of dealing with set inclusion rather than subsumption. This question is out of the scope of the present paper. For the examples of Section 5, the ρ-subsumption is sufficient.

Generalization rule and strategy to solve non termination

We propose now a simple generalization technique for introducing ρ-terms during the state exploration. The idea is to generalize the value of a state variable as soon as we detect in it a context that is iterated at least twice. As an example, suppose that we have for a state variable c with the initial value n, the following infinite sequence of states produced by applying at each transition the functions f and g alternatively: $c = n$, $c = f(n)$, $c = g(f(n))$, $c = f(g(f(n)))$, $c = g(f(g(f(n))))$, $c = f(g(f(g(f(n)))))$, $c = g(f(g(f(g(f(n))))))$, $\ldots$. After the fourth transition, the value of c consists of the context $g(f(\diamond))$ that is iterated twice. Thus, we generalize the state $c = g(f(g(f(n))))$ by $s:\ c = \Phi(g(f(\diamond)), N, n)$. After two more transition steps, we reach the state $t:\ g(f(\Phi(g(f(\diamond)), N, n)))$, and t is subsumed by s via the substitution $N+1/N$. We define formally our generalization rule as follows:

Definition 4.1 (Generalization rule) *Let t be a term, X be a variable which does not occur in t, and s be a term occurring in t at the position p. If s has a proper subterm s' which occurs in s at the position q such that $s[X]_q$ subsumes s' via the substitution v/X, then by generalization, t is replaced by the ρ-term $t[\Phi(s[\diamond]_q, N, v)]_p$.*

Example 4.2 *Let t be the term $g(x, h(f(a, b, f(a, b, g(b, a))), y))$. We have $s = f(a, b, f(a, b, g(b, a)))$ at position $p = 2.1$ in t. $s' = f(a, b, g(b, a))$ is a proper subterm of s that is subsumed by $s[U]_3$ via the substitution $g(b, a)/U$. Then, t can be generalized by $g(x, h(\Phi(f(a, b, \diamond), N, g(b, a)), y))$.*

The generalization strategy we propose for the state exploration procedure in MDGs can be described as follows. After each transition step, the generalization rule of Definition 4.1 is applied to each state of the frontier set whose abstract state variables have values which satisfy the conditions on its application. This can be checked automatically. To handle the ρ-terms introduced by generalization, we extend the existing PbyS algorithm by using the ρ-subsumption instead of the ordinary subsumption. This strategy will be illustrated on a number of examples in Section 5. The ρ-subsumption used in these examples is particularly simple, and merely amounts to a one level folding followed by ordinary subsumption. This means that few changes are necessary to the existing implementation of the PbyS algorithm in MDG to handle those examples, namely the addition of a folding step. We do not rule out the fact, however, that a more complicated ρ-subsumption could be needed in some more subtle cases of non termination of ASM state exploration.

Note that the conditions on the application of the generalization rule as defined in Definition 4.1 suggest to generalize a term if it contains a context iterated at least twice. This property might be too weak in some cases and might yield to inappropriate schematizations. Let us consider for instance the infinite sequence n, $f(n)$, $f(f(n))$, $g(f(f(n)))$, $g(g(f(f(n))))$, $f(g(g(f(f(n)))))$, $f(f(g(g(f(f(n))))))$, $g(f(f(g(g(f(f(n)))))))$, …, produced by a transition relation which applies in alternance the two functions f and g twice each time. Then, after the second transition, the conditions on the application of the generalization rule are satisfied, thereby generalizing the term $f(f(n))$ into the ρ-term $\Phi(f(\diamond), N, n)$. However, this ρ-term does subsume none of its successor in the sequence.

A straightforward extension of our generalization rule would be to promote the number of the iterations of the iterated context into a variable parameter k. With k set to the value 3 in the above sequence, the first term on which the extended generalization rule would apply is
$g(g(f(f(g(g(f(f(g(g(f(f(n))))))))))))$, which would be generalized by $T = \Phi(g(g(f(f(\diamond)))), N, n)$. Then, after four transition steps from T, we would get $g(g(f(f(\Phi(g(g(f(f(\diamond)))), N, n)))))$ which is subsumed by T via the subsitution $N+1/N$.

5 Examples

5.1 First example

Let us consider a state machine with three state variables $c0$, $c1$, and $c2$, of abstract sort *wordn*. $c0$ represents a counter and is incremented via a functional block represented by the uninterpreted function symbol s which takes x as its input and produces an abstract value $s(x)$ of abstract sort *wordn*. $c1$ (resp. $c2$) is updated via the abstract function f (resp. g) which takes as its abstract input the value of $c1$ (resp. $c2$) and produces the abstract output $f(c1)$ (resp. $g(c2)$). The cross-operator *eq* is of type $wordn \times wordn \rightarrow bool$ and denotes the equality relation. We use the rewriting rule r1: $eq(x, x) \rightarrow 1$. Consider now the following transition relation:

$$Tr \equiv ((\neg eq(c1, c2) \ \wedge \ c0' = zero \ \wedge \ c1' = c1 \ \wedge \ c2' = c1) \vee$$
$$(eq(c1, c2) \ \wedge \ c0' = s(c0) \ \wedge \ c1' = f(c1) \ \wedge \ c2' = g(c2))$$

where $c0'$ (resp., $c1'$, $c2'$) is the next state variable of $c0$ (resp., $c1$, $c2$). Suppose that the initial state of this machine is $s_0 : c0 = zero \wedge c1 = n \wedge c2 = n$, where n is an abstract variable of sort *wordn* and *zero* is a generic constant, and let us perform the state exploration using our generalization strategy. After one transition we reach the state s_1 and after two transitions, we may reach the states s_{21} and s_{22} as follow:
$s_1 : c0 = s(zero) \wedge c1 = f(n) \wedge c2 = g(n)$
$s_{21} : c0 = zero \wedge c1 = f(n) \wedge c2 = f(n) \wedge \neg eq(f(n), g(n))$
$s_{22} : c0 = s(s(zero)) \ \wedge \ c1 = f(f(n)) \ \wedge \ c2 = g(g(n)) \ \wedge \ eq(f(n), g(n))$.
Now, state s_{21} is subsumed by state s_0 via the substitution $f(n)/n$, and s_{22} is generalized by abstracting the values of the state variables $c0$, $c1$, and $c2$,

thereby giving the state:
$s'_{22} : c0 = \Phi(s(\diamond), N_0, zero) \ \wedge \ c1 = \Phi(f(\diamond), N_1, n) \ \wedge \ c2 = \Phi(g(\diamond), N_2, n) \ \wedge$ $eq(f(n), g(n))$.
From s'_{22}, we may reach the following states:

$$s_{31} : c0 = zero \ \wedge c1 = \Phi(f(\diamond), N_1, n) \ \wedge \ c2 = \Phi(f(\diamond), N_1, n) \wedge$$
$$eq(f(n), g(n)) \ \wedge \ \neg eq(\Phi(f(\diamond), N_1, n), \Phi(g(\diamond), N_2, n)$$
$$s_{32} : c0 = s(\Phi(s(\diamond), N_0, zero)) \ \wedge \ c1 = f(\Phi(f(\diamond), N_1, n)) \wedge$$
$$c2 = g(\Phi(g(\diamond), N_2, n)) \ \wedge \ eq(f(n), g(n)) \wedge$$
$$eq(\Phi(f(\diamond), N_1, n), \Phi(g(\diamond), N_2, n))$$

Now, state s_{31} is subsumed by state s_0 via the substitution $\Phi(f(\diamond), N_1, n)/n$, and s_{32} is subsumed by s'_{22} via the substitution $\{N_0 + 1/N_0, N_1 + 1/N_1, N_2 + 1/N2\}$ (remember that N_0, N_1 and N_2 are degree variables and therefore can be renamed in each state). Therefore, the state exploration terminates after the third transition. Note that the procedure also terminates after three transitions if $c1$ and $c2$ initially contain two arbitrary variables $n1$ and $n2$.
On the other hand, the strategy proposed in [3] would suggest to start the state exploration on the generalized initial state:
$s_0 : c0 = \Phi(s(\diamond), N0, zero) \ \wedge \ c1 = \Phi(f(\diamond), N1, n) \ \wedge \ c2 = \Phi(g(\diamond), N2, n)$.
After one step, we would reach the following new states:

$$s_1 : c0 = s(\Phi(s(\diamond), N0, zero)) \ \wedge \ c1 = f(\Phi(f(\diamond), N1, n)) \wedge$$
$$c2 = g(\Phi(g(\diamond), N2, n)) \ \wedge \ eq(\Phi(f(\diamond), N1, n), \Phi(g(\diamond), N2, n))$$
$$s_2 : c0 = zero \ \wedge \ c1 = \Phi(f(\diamond), N1, n) \ \wedge \ c2 = \Phi(f(\diamond), N1, n) \wedge$$
$$\neg eq(\Phi(f(\diamond), N1, n), \Phi(g(\diamond), N2, n)).$$

The state s_2 is not subsumed by s_0. From s_2 we would reach the following state:

$$s_3 : c0 = s(zero) \ \wedge \ c1 = f(\Phi(f(\diamond), N1, n)) \ \wedge \ c2 = g(\Phi(f(\diamond), N1, n)) \wedge$$
$$\neg(\Phi(f(\diamond), N1, n), \Phi(g(\diamond), N2, n)).$$

It is easy to see that the state exploration will generate for $c2$ an infinite sequence of terms of the form $g(g(g(\ldots g(\Phi(f(\diamond), N1, n)) \ldots)))$ and will never terminate.

5.2 Second example

Let us consider now a state machine with the state variable y of a concrete sort with enumeration $\{0, 1\}$ and the state variable x of abstract sort *wordn*. x is updated via an abstract function f which takes x as its input and produces $f(x)$ of abstract sort *wordn*. The cross-term *equz(x)* of type *wordn* $\rightarrow$ *bool* is used to test if x equals *zero*. r is an input variable of concrete sort with enumeration $\{0, 1\}$ representing the reset. We use the following rewriting rule r1: $equz(zero) \rightarrow 1$. The transition relation Tr of this machine is defined as follows:

$$\begin{aligned} Tr \equiv\ & (r = 1 \ \wedge\ y' = 0 \ \wedge\ x' = zero) \vee \\ & (r = 0 \ \wedge\ y = 0 \ \wedge\ equz(x) \ \wedge\ y' = 1 \ \wedge\ x' = f(x) \vee \\ & (r = 0 \ \wedge\ y = 0 \ \wedge\ \neg equz(x) \ \wedge\ y' = y \ \wedge\ x' = zero \vee \\ & (r = 0 \ \wedge\ y = 1 \ \wedge\ y' = y \ \wedge\ x' = f(x) \end{aligned}$$

where x' (resp. y') is the next state variable of x (resp. y). We start off from the initial state $s_0 : \ x = zero \wedge y = 0$. It is easy to see that the state exploration goes into an infinite loop because of the state variable x which produces the infinite sequence of increasing terms $zero$, $f(zero)$, $f(f(zero))$, $f(f(f(zero)))$ ... One can also verify that the initial state generalization technique based on variable introduction would also fail to achieve termination on this example. Actually, if the initial value of x is generalized by an abstract variable n, the state exploration produces the infinite sequence
$(x = n \wedge y = 0)$, $(x = f(n) \wedge y = 1 \wedge equz(n))$, $(x = f(f(n)) \wedge y = 1 \wedge equz(n))$, $(x = f(f(f(n))) \wedge y = 1 \wedge equz(n))$, ..., since there is no valid substitution σ such that $(x = f^p(n) \wedge y = 1 \wedge equz(n))\ \sigma = (x = f^q(n) \wedge y = 1 \wedge equz(n))$, with $p < q$.
The strategy proposed in [3] would suggest to resume the state exploration with $x = \Phi(f(\diamond), N, zero)$ as the initial state. After one transition, we would reach the following states:
$s_{11} : equz(\Phi(f(\diamond), N, zero)) \ \wedge\ x = f(\Phi(f(\diamond), N, zero)) \ \wedge\ y = 1$
$s_{12} : \neg equz(\Phi(f(\diamond), N, zero)) \ \wedge\ x = zero \ \wedge\ y = 0$.
By unfolding these two states and applying the rewriting rule r1, we get the following new states:
$s_{21} : x = f(zero) \ \wedge\ y = 1$
$s_{22} : equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = f(\Phi(f(\diamond), N, zero)) \ \wedge\ y = 1$
$s_{23} : \neg equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = zero \ \wedge\ y = 0$.
Now, from state s_{22}, the state exploration would go into the infinite loop:
$equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = f(f(\Phi(f(\diamond), N, zero))) \ \wedge\ y = 1$,
$equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = f(f(f(\Phi(f(\diamond), N, zero)))) \ \wedge\ y = 1$,
$equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = f(f(f(f(\Phi(f(\diamond), N, zero))))) \ \wedge\ y = 1$,
$\vdots$

since there is no possible substitution for the degree variable N such that
$(equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = f^n(\Phi(\diamond), N, zero) \ \wedge\ y = 1)$ is subsumed by $(equz(f(\Phi(f(\diamond), N, zero))) \ \wedge\ x = f^p(\Phi(\diamond), N, zero) \ \wedge\ y = 1)$, for all p, $p < n$.
A possible solution to achieve termination would be to introduce a second generalization step, for instance by renaming one of the two occurrences of the variable N into a new variable N'. However, this would amount to simply drop the constraint $equz(f(\Phi(f(\diamond), N, zero)))$ since it would become unrelated to x.

With our strategy the state exploration terminates after the third transition step. Actually, from the initial state s_0, we may reach after one transition the states
$s_1 : x = f(zero) \ \wedge \ y = 1$
$s_2 : x = f(f(zero)) \ \wedge \ y = 1$, which is generalized into
$s'_2 : x = \Phi(f(\diamond), N, zero) \ \wedge \ y = 1$. The next state reached from s'_2 is
$s_3 : x = f(\Phi(f(\diamond), N, zero)) \ \wedge \ y = 1$, which is subsumed by s'_2 via the substitution $N + 1/N$, and the state exploration terminates.
Note that the state exploration also terminates if we start from an initial state where x contains any generic constant n instead of *zero*. Actually, we would have the following sequence of states:
$s_1 : equz(n) \ \wedge \ y = 1 \ \wedge \ x = f(n)$
$s_2 : equz(n) \ \wedge \ y = 1 \ \wedge \ x = f(f(n))$, which after generalization becomes
$s'_2 : equz(n) \ \wedge \ y = 1 \ \wedge \ x = \Phi(f(\diamond), N, n)$, and finally
$s_3 : equz(n) \ \wedge \ y = 1 \ \wedge \ x = f(\Phi(f(\diamond), N, n))$, which is subsumed by s'_2 via the substitution $N + 1/N$.
This means, in particular, that the state exploration also terminates if we do not make use of the rewriting rule r1.

5.3 Third example

We borrow here an example from [3] of a state machine which has one state variable R of concrete sort with enumeration $\{R1, R2, R3\}$. The transition relation T_r is as follows:

$$\begin{array}{ll}
T_r \equiv & \\
(ie = 0 \ \wedge \ R = R_1 \ \wedge \ R' = R_1 \ \wedge \ c' = c) & \vee \\
(eqz(c) = 0 \ \wedge \ R = R_1 \ \wedge \ R' = R_1 \ \wedge \ c' = c) & \vee \\
(ie = 1 \ \wedge \ eqz(c) = 1 \ \wedge \ R = R_1 \ \wedge \ R' = R_2 \ \wedge \ c' = c) & \vee \\
(iy = 0 \ \wedge \ ie = 0 \ \wedge \ R = R_2 \ \wedge \ R' = R_2 \ \wedge \ c' = c) & \vee \\
(iy = 0 \ \wedge \ ie = 1 \ \wedge \ R = R_2 \ \wedge \ R' = R_3 \ \wedge \ c' = inc(c)) & \vee \\
(iy = 1 \ \wedge \ R = R_2 \ \wedge \ R' = R_1 \ \wedge \ c' = c) & \vee \\
(ie = 1 \ \wedge \ R = R_3 \ \wedge \ R' = R_3 \ \wedge \ c' = c) & \vee \\
(ie = 0 \ \wedge \ R = R_3 \ \wedge \ R' = \ R_2 \ \wedge \ c' = c) &
\end{array}$$

where R' is the next state variable of R. We have the following rewriting rules:

$$\begin{array}{lll}
eqz(zero) & \rightarrow 1 & r1 \\
eqz(inc(x)) & \rightarrow 0 & r2
\end{array}$$

The initial state of this machine is $s_0 : (R = R_1 \ \wedge \ c = zero)$. Below, we perform the state exploration according to our strategy. From s_0, we may

reach the states s_1, s_2 and s_3 as follows:
$s_1 : R = R_2 \ \wedge \ c = zero$
$s_2 : R = R_3 \ \wedge \ c = inc(zero)$
$s_3 : R = R_3 \ \wedge \ c = inc(inc(zero))$.
s_3 is generalized by abstracting the value of c by the ρ-term $\Phi(inc(\diamond), N, zero)$, thereby giving
$s_3' : R = R_3 \ \wedge \ c = \Phi(inc(\diamond), N, zero)$.
From s_3', we may reach the new state $s_4 : R = R_3 \wedge c = inc(\Phi(inc(\diamond), N, zero))$, which is subsumed by s_3' and therefore the space exploration terminates after the fourth transition.

In [3], the termination of the space exploration is achieved by starting from the generalized initial state $(R = R_1 \wedge c = \Phi(inc(\diamond), N, zero))$, provided a second generalization is performed after the first state transition. Note, however, that the strategy described in [3] does not address the question of when and how generalization steps must be performed during state exploration. Moreover, in [3], the first transition step yields the new state $\neg eqz(\Phi(inc(\diamond), N, zero)) \ \wedge \ \Phi(inc(\diamond), N, zero)$. By unfolding, the constraint $\neg eqz(\Phi(inc(\diamond), N, zero))$ disappears because the base case gives $\neg eqz(zero)$ which is rewritten to 0, and the general case gives $\neg eqz(\Phi(inc(\diamond), N, zero))$ which is rewritten to 1. This simplification is possible because we are in the special case where the cross-operator *eqz* can be fully interpreted on the ρ-term$eqz(\Phi(inc(\diamond), N, zero))$ after unfolding. However, if we modify the example by replacing the cross-term $eqz(c)$ by the cross-term $even(c)$, with the intended denotation "c is an even number", then the simplification of the constraint $\neg even(\Phi(inc(\diamond), N, zero))$ by unfolding is only possible in the base case ($even(zero)$ is rewritten to 1), but no longer in the general case since $even(inc(\Phi(inc(\diamond), N, zero)))$ cannot be simplified. One can verify that the strategy used in [3] would fail to achieve termination on the modified version of the example, while our strategy terminates in this case too. Actually, the termination of the state exploration above described can be achieved without the use of the rewriting rules r1 and r2.

6 Conclusions

We have presented a generalization method based on term schematization for solving some cases of non termination in the state exploration procedure of Abstract States Machines represented by MDGs. It is based on a generalization rule which aims at replacing a first-order term presenting a certain recurring pattern by a ρ-term, i.e., a meta-term which represents in a finite way an infinite sequence of terms sharing the same pattern. This rule allows us to achieve termination of the state enumeration in the case where non termination is caused by a certain repetitive pattern which grows infinitely along the sequence of reachable states. Since ρ-terms are capable of propagating some information about the structure of the terms they generalize, we think that they are particularly useful in reachability analysis because they reduce the risk of producing false negatives. This is in contrast with the techniques based on generalization variable introduction which may produce very poor generalizations.

Our method improves over the previous proposals, and in particular over [3] where the idea of using term schematization in MDGs has been first proposed. First and foremost, it can be applied fully automatically and does not require the user to provide the state variable to be generalized nor its generalized value. Second, generalization is not performed at the initial state, which allows to cope with situations where a state variable gives rise to different patterns of divergence in alternative branches of the state exploration. Moreover, it allows to exploit some information provided by the axioms which partially interpret abstract functions on the initial values, if any. Our technique is currently under implementation into the MDG tools and future work will include the study of its applicability to the reachability analysis in real designs. We also intend to enhance our generalization rule to detect and handle more complicated cases of recurrent term structures. We may consider in particular the method proposed in [9] for inferring recursive patterns based on the notion of *embedding relation* [16].

References

[1] O. Aït Mohamed, E. Cerny, and X. Song. Mdg-based Verification by Retiming and Combinational Transformations. *Proccedings of the IEEE 8th Great Lakes Symposium on VLSI, Louisiana, USA, 1998.*

[2] O. Aït Mohamed, X. Song., E. Cerny, S. Tahar, and Z. Zhou. Solving the Non Termination of Abstract Implicit State Enumeration Using a Circuit Transformation: A Case Study on the Island Tunnel Controller. Research report 1105, University of Montreal, 1998.

[3] O. Aït Mohamed, X. Song., and E. Cerny. On the Non-termination of MDG-based Abstract State Enumeration. *Proccedings of the IFIP W 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (Charme'97), Montréal, October 1997.* IFIP, Chapmann & Hall.

[4] A. Amaniss, M. Hermann, D. Lugiez. Set Operations for Recurrent Term Schematization. *Proccedings of the (TAPSOFT'97), Lille (France)*, volume 1214 of LNCS, pages 333-344, 1997. Springer-Verlag.

[5] A. Amaniss. Méthodes de schématisation pour la démonstration automatique. PhD thesis, Université Henri Poincaré, Nancy 1, 1996.

[6] K.D. Anon, N. Boulerice, E. Cerny, F. Corella, M. Langevin, X. Song, S. Tahar, Y. Xu, Z. Zhou. MDG tools for the Verification of RTL Designs. In *Proceedings of the Conference on Computer-Aided Verification* (CAV '96), New Jersey, USA, July 1996.

[7] R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, August 1986, 35(8):677–691.

[8] E. Cerny, F. Corella, M. Langevin, X. Song, S. Tahar and Z. Zhou. Automated Verification with Abstract State Machines Using Multiway Decision Graphs. In *Formal Hardware Verification: Methods and Systems in Comparison*, volume 1287 of LNCS, pages 79–113. Springer Verlag, 1997.

[9] H. Chen, J. Hsiang, and H.-C. Kong. On finite Representations of Infinite Sequences of Terms. *Proceedings of the 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada), 1990*, volume 516 of LNCS, pages 100–114. Springer-Verlag.

[10] H. Chen, and J. Hsiang. Recurrence domains: Their Unification and Application to Logic Programming. *Information and Computation, 1995*, 122:45–69.

[11] H. Comon. On Unification of terms with Integer Exponents. *Mathematical System Theory*, 28(1):67-88, 1995.

[12] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. *Formal Methods in System Design*, 10(1):7–46, 1997.

[13] F. Corella, Z. Zhou, X. Song, M. Langevin., and E. Cerny. MDG Algorithms (I). Technical Report. University of Montreal, 1995.

[14] M. Hermann. On the Relation Between Primitive Recursion, Schematization, and Divergence. *Proceedings of the 3rd Conference on Algebraic and Logic Programming, Volterra (Italy), 1992*, volume 632 of LNCS, pages 115–127. Springer-Verlag.

[15] M. Hermann and G. Salzer. On the Word, Subsumption, and Complement Problem for recurrent Term Schematizations. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98), Brno (Czech Republic), 1998*, volume 1450 of LNCS, pages 257-266. Springer-Verlag.

[16] newblock The Theory of Well-quasi-ordering: A Frequently Discovered Concept. *Journal of Combinatorial Theory*, November 1972, Ser.A 13(3):297–305.

[17] G. Salzer. The Unification of Infinite Sets of Terms and Its Applications. In Voronkov, A., editor, *Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning, St. Petersburg (Russia), 1992*, volume 624 of LNCS, pages 409–420. Springer-Verlag.

[18] S. Tahar, Z. Zhou, X. Song, .E Cerny, and M. Langevin. Formal Verification of an ATM Switch Fabric using Multiway Decision Graphs. In *Proccedings of the Great Lakes Symposium on VLSI (GLS-VLSI'96)*, Arres, Iowa, USA, March 1996. IEEE Computer Society Press.

[19] Y. Xu., E. Cerny, X. Song, F. Corella, and O. Aït Mohamed. Model Checking for A first-Order Temporal Logic using Multiway Decision Graphs. In Alan, J.H and Moshe, Y. Vardi, editors, *Proceedings 10th International Conference on Computer Aided Verification, CAV'98, Vancouver, BC, Canada*, volume 1427 of LNCS, pages 219–231. Springer-Verlag.

[20] Z. Zhou. MDG tools developer's manual. Technical Report. University of Montreal, 1995.

[21] Z. Zhou, X. Song, F. Corella, M. Langevin., and E. Cerny. MDG Algorithms (II). Technical Report. University of Montreal, 1995.

[22] Z. Zhou, X. Song, S. Tahar, E. Cerny, F. Corella and M. Langevin. Formal Verification of the Island Tunnel Controller using Multiway Decision Graphs. In *Proceedings of the International Conference on Formal Methods in Computer Aided Design* (FMCAD'96), USA, 1996, pages 233–247.