Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 300 (2003) 161–179

Theoretical
Computer Science

www.elsevier.com/locate/tcs

On the non-termination of M_{DG}-based abstract state enumeration

Otmane Aït Mohamed[a, 1], Xiaoyu Song[b, *], Eduard Cerny[c]

[a] *Concordia University, ECE Department, 1455 de Maisonneuve Blvd. W., Montreal, Canada H3G 1M8*
[b] *Portland State University, P.O. Box 751, Portland, OR 97207-0751, USA*
[c] *Université of Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Canada H3C 3J7*

Received 20 December 1999; received in revised form 8 March 2001; accepted 23 August 2001
Communicated by R. Gorrieri

Abstract

Multiway decision graphs are a new class of decision graphs for representing abstract states machines. This yields a new verification technique that can deal with the data-width problem by using abstract sorts and uninterpreted functions to represent data value and data operations, respectively. However, in many cases, it may suffer from the non-termination of the state enumeration procedure. This paper presents a novel approach to solving the non-termination problem when the generated set of states, even infinite, represents a structured domain where terms (states) share certain repetitive patterns. The approach is based on the *schematization* method developed by Chen and Hsiang, namely ρ-terms. Schematization provides a suitable formalism for finitely manipulating infinite sets of terms. We illustrate the effectiveness of our method by several examples.

Keywords: Multiway decision graphs; Reachability analysis; Recurrent domains; ρ-terms

1. Introduction

Two main approaches of formal verification have been studied: interactive verification using a theorem prover, and model checking. Each method possesses its own strengths and weaknesses. Mechanical theorem proving is more general, but requires intensive human guidance and is thus time consuming. Model checking is automatic, but it is applicable to finite state machines and suffers from the state-explosion problem.

* Corresponding author. Tel.: +1-503-725-5398; fax: +1-503-725-3807.
E-mail addresses: ait@ece.concordia.ca (O.A. Mohamed), song@ee.pdx.edu (X. Song), cerny@iro.umontreal.ca (E. Cerny).

[1] This work was performed while the first author was at University of Montreal.

0304-3975/03/$ - see front matter
PII: S0304-3975(01)00345-0

This limits its use to relatively small circuits. The problem was widely addressed in the literature. The works described in [1, 3, 4, 7, 8, 11, 16] exploit Bryant's reduced ordered binary decision diagrams (ROBDDS) [2] to encode sets of states and to perform implicit enumeration of the state space. However, these methods are not adequate in general for verifying circuits with large and complex data paths, because of the Boolean representation of circuits. More specifically, every individual bit of every data signal is represented by a Boolean variable, while the size of a ROBDD grows, sometimes exponentially, with the number of variables. This means that ROBDD-based verification methods often take too much time, or run out of memory, when applied to circuits having a complex data path.

Recently, a new verification approach was presented to overcome the above drawbacks. This approach is based on *abstract descriptions of state machines* (ASM) which are encoded by a new class of decision graphs, called *multiway decision graphs* (MDGS) [10], of which ROBDDS are a special case. With MDGS, one can integrate two verification techniques that have been very successful: *implicit state enumeration* and *the use of abstract sorts and uninterpreted function symbols*. MDGS are decision graphs that can represent relations as well as sets of states, and they incorporate variables of abstract sorts to denote data values, and uninterpreted function symbols to denote data operations. A set of basic operations on MDG graphs was implemented to perform various kinds of verification. It includes the algorithms for disjunction, relational product (image computation), pruning by subsumption, and rewriting [10].

Unfortunately, the method suffers in many cases from an important problem, namely non-termination when computing the set of reachable states. This can be a severe limitation on the use of MDGS as a verification tool. For example, consider an abstract description of a conventional (non-pipelined) microprocessor where a state variable *pc* of abstract sort represents the program counter, a generic constant *zero* of the same abstract sort denotes the initial value of *pc*, and an abstract function symbol *inc* describes how the program counter is incremented by a non-branch instruction. The MDG representing the set of reachable states of the microprocessor would contain states of the form

$$(pc, \underbrace{inc(\ldots inc(}_{k} zero)\ldots))$$

for every $k \geqslant 0$. Consequently, there is no finite MDG representation of the set of reachable states, and hence the reachability algorithm will not terminate. This illustrates a typical form of non-termination, due to the fact that the terms can be arbitrarily large and hence arbitrarily many.

In this paper, we present a method based on schematization to deal with this kind of non-termination problem. Schematization is a method for finitely representing infinite sets of objects [12, 5, 15, 9, 14]: terms, rewrites rules, substitutions, etc. If some infinite sets can be represented finitely and if algebraic operations can be performed on these finite representations, then the problem of non-termination can be avoided. The abstract machines we consider are those that present a cyclic behavior starting

from any state of the machine. Let us return to the above example: as we can see, the labels of the *pc* node in the MDGs generated by the reachability analysis procedure form a structured domain whose terms share a repetitive pattern. For instance, the pattern of the domain $\{inc^k(zero) : k \in \mathbb{N}\}$ is $inc(\diamond)$ such that, for every term $inc^k(zero)$ in the domain, $inc(inc^k(zero))$ is also in the domain. Such a set is usually represented by a single expression $\Phi(inc(\diamond), N, zero)$, where the first argument of Φ represents the pattern, the second argument serves as a counter, called the *degree variable*, and the third argument represents the base term. This kind of an expression is called a ρ-term [5]. By allowing ρ-terms to be part of the language, we are able to represent an infinite MDG finitely, by labeling some edges by a ρ-term, i.e., we can represent a logical formula having an infinite number of disjuncts of the form: $pc = zero \vee pc = inc(zero) \vee pc = inc(inc(zero)) \vee \cdots \vee inc^k(zero) \vee \cdots$ as $pc = \Phi(inc(\diamond), N, zero)$.

The paper is organized as follows: In Section 2, we briefly present MDGs. In Section 3, we provide a background on ρ-terms, and we define the MDG extension which incorporates the ρ-terms. In Section 4, we illustrate our method on several examples. Finally, we conclude with some remarks and discuss the direction of future work.

Related Works. The non-termination problem was studied in [17]. The authors presented a method based on the generalization of the state variable that causes divergence, like the variable *pc* in the example. Rather than starting the reachability analysis with a generic constant *zero* as the value of *pc*, a fresh[2] variable is assigned to *pc* at the beginning of the analysis. As a consequence, the set of states represented by *pc* is enlarged, so that any incrementation of *pc* leads the ASM to a state, where the new value of *pc* is an instance of its arbitrary initial value. This technique is applicable only to circuits which, like a conventional (non-pipelined) processor, has a cyclic behavior: starting from a ready state, doing some work, and returning to a ready state. This class of circuits is called *processor-like loop* circuit. Unfortunately, if the entrance of the loop does not start in the initial state, then this generalization technique may not work. The solution proposed by the authors consists of finding manually the entry state of the loop, where the generalization must be done. In general, it is difficult to identify processor-like loops in a machine. The heuristics used by the authors require human interferences at different stages of the verification process. Furthermore, the main drawback of this generalization method is the loss of information provided by axioms which partially interpret abstract function symbols. This means that we become deprived of powerful automated deduction technique, such as rewriting, which is useful when carrying out verification with MDGs. In this work, we use a schematization which possesses the advantages of generalization while avoiding its weaknesses to deal with non-termination. Rewriting rules that characterize uninterpreted functions can still be used. It suffices to instantiate the degree variables to specific values and apply suitable rules.

[2] A *fresh* variable is disjoint from all other variables.

2. Multiway decision graphs

An MDG is a finite, directed acyclic graph (DAG) where the leaves are labeled by True ($\top$), the internal nodes are labeled by terms, and the edges issuing from an internal node v are labeled by terms of the same sort as the label of v. Such graph is a canonical representation, modulo a set of well-formedness conditions (see [10] for details), of a certain quantifier-free formula, called a *directed formula* (DF). A DF formula is 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. Concrete sorts have enumerations which are sets of individual constants, while abstract sorts do not.

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

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

The vocabulary consists of generic constants, concrete constants, abstract variables, concrete variables and function symbols. 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 α_{n+1} is an abstract sort then f is an *abstract function symbol*. If all the $\alpha_1 \ldots \alpha_{n+1}$ are concrete, f is a *concrete* function symbol. If α_{n+1} is concrete while at least one of $\alpha_1 \ldots \alpha_n$ is abstract, then we refer to f as a *cross-operator*; cross-operators are useful for modeling feedback from the data path to the control circuitry. Atomic formulae are the equations, generated by the clause *Eq*,

plus $\top$ (truth) and $\bot$ (false). Directed formulae are a disjunction of conjunction of equations.

An equation is *well-typed* if the sort of the term on the left-hand side of the equation is the same as the sort of the term on the right-hand side.

A directed formula is well-typed and is of type $\mathbf{U} \to \mathbf{V}$ if and only if: (1) each equation is well-typed, and (2) each term A in the equation of the form $\underline{A} = \underline{C}$ or $\boldsymbol{X} = \boldsymbol{A}$ is in $\mathbf{T}(\mathbf{F}, \mathbf{U})$, and (3) for every abstract variable $v \in \mathbf{V}$ appears as the LHS of an equation $v = A$ in each of the disjuncts.

Just as ROBDDS [2] must be *reduced* and *ordered*, MDGS must obey a set of well-formedness conditions given in [10]. 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 a sort $\underline{\alpha}$, with edges labeled by the individual constants in the enumeration of $\underline{\alpha}$. All leaf nodes are labeled $\top$, except when the graph has a single node, which may be labeled $\top$ or $\bot$.

Semantics. An *interpretation* is a mapping ψ that assigns a denotation to each sort, constant and function symbol, and satisfies the following conditions:

- The denotation $\psi(S)$ of an abstract sort S is a non-empty set.
- If $\underline{S} = \{\underline{a_1}, \ldots, \underline{a_m}\}$ then $\psi(\underline{S}) = \{\psi(\underline{a_1}), \ldots, \psi(\underline{a_m})\}$ and $\psi(\underline{a_i}) \neq \psi(\underline{a_j})$ for all i, j such that $a_i \neq a_j,\ i \neq j$.
- A *variable assignment* with domain X compatible with an interpretation ψ, is a function γ that maps every variable x of X of sort $\mathbf{S}$ to an element $\gamma(x)$ of $\psi(\mathbf{S})$. We write I_X^{ψ} for the set of ψ-compatible assignments to the variables in X.
- If $f(t_1, \ldots, t_n)$ is a term of sort $\mathbf{S}_{n+1}$ and $t_1, \ldots, t_n$ are terms of sorts $\mathbf{S}_1, \ldots, \mathbf{S}_n$, respectively, then the denotation $\psi(f(t_1, \ldots, t_n))$ is defined as $\psi(f)(\psi(t_1), \ldots, \psi(t_n))$. In particular, if the arity of f is equal to 0, (i.e., f is a generic constant of sort $\mathbf{S}$), $\psi(f) \in \psi(\mathbf{S})$.
- We write $\psi, \gamma \models P$ if a formula P denotes truth under an interpretation ψ and ψ-compatible variable assignment γ to the free variables of P, $\psi \models P$ if $\psi, \gamma \models P$ for all such assignments γ, and $\models P$ if $\psi \models P$ for all interpretations ψ. Two formulae P and Q are *logically equivalent* iff $\models P \Leftrightarrow Q$.

MDG-based abstract state enumeration. A circuit is described at the register-transfer level as a collection of components interconnected by nets that carry signals. Each signal is represented by a variable. Variables denoting control signals have concrete sorts, while variables denoting data values have abstract sorts. An Uninterpreted function symbols are used to model control operations, which must have a concrete sort, while data operations are viewed as black boxes and are modeled by an uninterpreted function symbol which must have an abstract sort. A set of basic operation on the MDGS graph is implemented to perform various kind of verification for a given circuit. This set include algorithms for disjunction, relational product (image computation), pruning by subsumption, and rewriting. A good discussion on that is described in [10]. Most of the verification techniques with MDGS are based on an implicit reachability analysis

algorithm which forms the kernel of this tool. The algorithm is based on abstract state enumeration [10], where sets of states, as well as transition and output relations, are represented using MDGs. Because of abstract variables and the uninterpreted nature of function symbols, the reachability analysis algorithm may not terminate, and the least fixed point may not be reached in state enumeration. The procedure, called ReAn for *Reachability Analysis*, is described by the following pseudo-code:

```
1.  Proc ReAn(D)
2.      R := F_I; Q := F_I; K := 0;
3.      loop
4.          K := K + 1;
5.          I := Fresh(X, K);
6.          N := RelP({I, Q, F_T}, X ∪ Y, η);
7.          Q := PbyS(N, R);
8.          if Q = ⊥ then return success;
9.          R := PbyS(R, Q);
10.         R := Disj(R, Q);
11.     end loop;
12. end ReAn;
```

where $D=(X,Y,Z,F_{\mathrm{I}},F_{\mathrm{T}},F_{\mathrm{O}})$ 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 a set of initial states, the transition, and the output relations, respectively.

We describe the most important steps of the algorithm, for more details see [17, 10].

In this pseudo-code, I, N, Q and R are program variables that take as values MDGs representing sets of states. We will identify the program variables and their values in the following explanations when there is no risk of confusion.

Before each loop iteration, R represents the set of reachable states found so far, while Q represents the frontier set, i.e., a subset of $Set_Y^{\psi}(R)$ containing at least all those states that entered $Set_Y^{\psi}(R)$ for the first time in the previous iteration.

In line 5, $\mathsf{Fresh}(X,K)$ constructs a one-path MDG representing a conjunction of equations $x=u$, one for each abstract input variable $x\in X$, where u is a fresh variable from the set of auxiliary abstract variables U. The value of the loop counter K is used to generate the fresh variables. This one-path MDG is assigned to I, which represents the set of input vectors.

In line 6, the relational product operation computes the MDG N representing the set of states reachable in one step from the frontier set Q of states that have not been visited.

Note that the MDG Q representing the frontier set is of type $U\to Y$, the MDG I representing the set of input vectors is of type $U\to X$, and the MDG F_{T} representing the transition relation is of type $(X\cup Y)\to Y'$. The result of taking the conjunction of these three MDGs would be of type $U\to(X\cup Y\cup Y')$, the result of subsequently removing the variables in $X\cup Y$ by existential quantification would be of type $U\to Y'$,

and the result of subsequently applying the renaming substitution η would be of type $U \to Y$. The RelP operation performs these three operations in one pass, and assigns the resulting MDG of type $U \to Y$ to N.

Lines 7 and 8 check if the states reachable in one step are included in the sets of states found in the previous iterations. This is done by using the prune-by-subsumption operation PbyS which removes from N those paths which are instances of some paths in R. This operation uses syntactic matching between terms that label two corresponding nodes to find such paths. If the result is the empty set, then the procedure terminates and reports success. Otherwise, the MDG Q represents the new frontier set.

Line 9 simplifies R by removing from it any paths that are subsumed by Q, using PbyS. There may be such paths because Q was not computed earlier as an exact difference. Then line 10 computes the new value of R by taking the disjunction of R and Q, which represents the set of states $Set_Y^{\psi}(R) \cup Set_Y^{\psi}(Q)$, and assigning it to R.

In general, this procedure may not terminate. When the MDGs generated in line 7 have a regular structure, we can schematize them by labeling some of its edges by a special term that represents this family of infinite objects. By using this finite representation, we can manipulate them by appropriate algebraic operations, such as unification. The non-termination problem can thus be avoided. In the next sections, we define ρ-terms and we show how to use them to solve the non-termination problem.

3. A solution to the non-termination problem

Schematization is a formalism for finitely describing infinite families of objects. Different schematizations were studied during the last years, using *term schemes* [12], *recurrence terms* [6, 15], *rules with membership constraints* [9], *meta-rules* [14], and *primal grammars* [13]. We chose recurrence terms (ρ-terms) [5] for schematizing the infinite states generated during reachability analysis because their algebraic operations are decidable and ρ-terms have been used as an extension to the Prolog language [5] in which our MDG verification system was implemented too.

3.1. Preliminaries: ρ-terms [5]

Let **F** be a set of function symbols, **V** a set of variables and **D** a set of degree variables. Let Φ be a special function symbol of arity 3. We use sequences of positive numbers, also called positions, to refer to specific subterms in a term. The empty sequence ε is a position in any term t, while a sequence $i \cdot u$ is a position in a term $t = f(t_1, \ldots, t_n)$ only if $1 \leqslant i \leqslant n$ and u is a position in t_i, where $\cdot$ is concatenation. If u is a position in a term t, then the subterm $t|_u$ of t at position u is t, if $u = \varepsilon$, and $t_i|_v$, if $t = f(t_1, \ldots, t_n)$ and $u = i \cdot v$, for some i with $1 \leqslant i \leqslant n$. We denote by $t[s]_u$, the result of replacing in t the subterm at position u by s. We define $t[s]_u$ to be the term s, if $u = \varepsilon$, and the term $f(t_1, \ldots, t_{i-1}, t_i[s]_v, t_{i+1}, \ldots, t_n)$, if $t = f(t_1, \ldots, t_n)$ and $u = i \cdot v$.

Definition 1 (*ρ-term*). A ρ-term is either a variable in $\mathbf{V}$ or an expression $f(t_1,\ldots,t_n)$, where f is a function symbol of arity n and $t_1,\ldots,t_n$ are ρ-terms, or $\Phi(h[\diamond]_u, N, l)$, where $h[l]_u$ is a ground term, u is a non-root position of h and N is a degree variable and $\diamond$ is a special symbol serving as a place holder. $\Phi(h[\diamond]_u, N, l)$ is called a generator. A ρ-term is ground if it does not contain any variable, and is called a *proper ρ-term* if its degree variables are uninstantiated.

Definition 2. We inductively define the function *Dvar* which computes the degree variable of a ρ-term as follows:

$$
\begin{aligned}
&Dvar(x) = \emptyset && \text{if } x \in \mathbf{V}\\
&Dvar(f(t_1,\ldots,f_{ar(f)})) && \text{if } f \in \mathbf{F} \text{ and } t_1,\ldots,f_{ar(f)} \text{ are } \rho\text{-terms}\\
&\quad = Dvar(t_1) \cup \cdots \cup Dvar(t_{ar(f)})\\
&Dvar(\Phi(h[\diamond]_u, N, l)) = \{N\}
\end{aligned}
$$

A proper ρ-term represents an infinite set of terms of $\mathbf{T}(\mathbf{F},\mathbf{V})$.

We use $H[n_1/N_1,\ldots,n_m/N_m]$ to denote the ρ-term obtained from a proper ρ-term H by instantiating the degree variables $N_1,\ldots,N_m$ by natural numbers $n_1,\ldots,n_m$, respectively, and then evaluating the instantiated generators $\Phi(h[\diamond]_u, n, l)$ to $\underbrace{h[h[h[\ldots h}_{n}[l\underbrace{]_u\ldots]_u]_u]_u}_{n}$, denoted by $h^n[l]_{u^n}$.

It means that, for a particular natural number n, a ρ-term denotes one term from $\mathbf{T}(\mathbf{F},\mathbf{V})$. Formally, the unfolding is defined as follows:

$$
\begin{aligned}
\Phi(h[\diamond]_u, 0, l) &\stackrel{\text{def}}{=} l\\
\Phi(h[\diamond]_u, n+1, l) &\stackrel{\text{def}}{=} h[\Phi(h[\diamond]_u, n, l)]_u
\end{aligned}
$$

Example 3. $g(\Phi(inc(\diamond), N, a), \Phi(inc(\diamond), M, b))[0/N, 2/M] = g(a, inc(inc(b)))$.

Definition 4 (*Recurrence domain*). A recurrence domain, denoted $\Omega(H)$, is a set obtained by unfolding a ρ-term H for all the possible values of each of its degree variables:

$$
\begin{aligned}
\Omega(x) &\stackrel{\text{def}}{=} x\\
\Omega(\Phi(h[\diamond]_u, N, l)) &\stackrel{\text{def}}{=} \{\Phi(h[\diamond]_u, n, l) \mid n \in \mathbb{N}\}\\
\Omega(f(t_1,\ldots,t_n)) &\stackrel{\text{def}}{=} \{f(s_1,\ldots,s_n) \mid s_i \in \Omega(t_i); 1 \leqslant i \leqslant n\}
\end{aligned}
$$

Definition 5 (*Equivalence of ρ-terms*). Two ρ-terms L and R which share the same degree variables $N_1,\ldots,N_m$ are equivalent, denoted as $L \equiv R$, if and only if $\Omega(L[n_1/$

$N_1,\ldots,n_m/N_m]) = \Omega(R[n_1/N_1,\ldots,n_m/N_m])$ for all $n_1,\ldots,n_m \in \mathbb{N}$. If L and R do not share any degree variables, then $L \equiv R$ if and only if $\Omega(L) = \Omega(R)$.

Definition 6 (*Inclusion of ρ-terms*). A ρ-term L is included in a ρ-term R, denoted as $L \subseteq R$, where L is a ground term and the set of degree variables of R is disjoint from those of L if and only if $\Omega(L) \subseteq \Omega(R)$. If R is also ground, then $L \subseteq R$ if and only if for all s in $\Omega(L)$, there exists a term t in $\Omega(R)$ such that s is identical to t.

We call an *unfolding substitution* a finite set $us = \{q_1 * Q_1 + k_1/N_1,\ldots,q_m * Q_m + k_m/N_m\}$ such that $N_i \neq N_j$ for all $i \neq j$ and $N_i \neq Q_j$ for all i and j, where q and k are integers, N_i and Q_i are degree variables, $1 \leqslant i \leqslant m$, $*$ is arithmetic multiplication and $+$ is arithmetic addition. An empty unfolding substitution is denoted by id.

Applying $q * Q + k/N$ to a ρ-term H is defined as:

$$x[q * Q + k/N] = x$$

$$f(s_1,\ldots,s_m)[q * Q + k/N] = f(s_1[q * Q + k/N],\ldots,s_m[q * Q + k/N])$$

$$\Phi(h[\diamond]_u, M, l)[q * Q + k/N] = \Phi(h[\diamond]_u, M, l) \text{ if } M \neq N$$

$$\Phi(h[\diamond]_u, N, l)[q * Q + k/N] = h^k[\Phi(h^q[\diamond]_{u^q}, Q, l)]_{u^k}$$

3.2. Application of ρ-terms in MDGs

The extension of the syntax of directed formulae to incorporate ρ-terms is straightforward. We allow the term A, defined in the syntax of Section 2, to be a proper ρ-term. ρ-DF is a directed formula where some of its terms are ρ-terms. For a ρ-DF P with m degree variables $N_1,\ldots,N_m$, we extend the function Ω by morphism as:

$$\Omega(P) = \vee_{n_1,n_2,\ldots,n_m \in \mathbb{N}} P[n_1/N_1,\ldots,n_m/N_m]$$

where $P[n_1/N_1,\ldots,n_m/N_m]$ denotes the ρ-DF obtained by instantiating each degree variables $N_1,\ldots,N_m$ by natural numbers $n_1,n_2,\ldots,n_m$.

Thus, $\Omega(P)$ is an infinite disjunction obtained by unfolding each ρ-term in P for all possible values of each of its degree variables.

If t is a ρ-term $\Phi(h[\diamond]_u, N, l)$, then the denotation $\psi(\Phi(h[\diamond]_u, N, l))$ under an interpretation ψ, is defined as $\Phi(\psi(h)[\diamond]_u, N, \psi(l))$, where $\psi(h)$ and $\psi(l)$ are the interpretations of terms h and l, respectively.

Let s denote a finite simultaneous substitution $[n_1/N_1,\ldots,n_m/N_m]$. We write $\psi,\gamma,s \models P$ if a formula $P[s]$ denotes the truth under an interpretation ψ, a ψ-compatible variable assignment γ to the free variables of $P[s]$; $\psi,\gamma \models P$ if $\psi,\gamma,s \models P$ for all substitutions s; $\psi \models P$ if $\psi,\gamma \models P$ for all ψ-compatible assignments γ; and $\models P$ if $\psi \models P$ for all interpretations ψ. Two formulae P and Q are *logically equivalent* if and only if $\models P \Leftrightarrow Q$.

The ρ-directed formulae are used to represent either sets or relations. For a ρ-DF P of type $\mathbf{U} \to \mathbf{V}$, where $\mathbf{U}$ contains only abstract variables, and for an interpretation ψ, P represents the following set of vectors:

$$Set^{\psi}(P) = \{\gamma \in \Gamma^{\psi}_{\mathbf{V}} \mid \psi, \gamma \models \exists \vec{N} \cdot (\exists U)P\}$$

where $\vec{N}$ represents a vector of degree variables that appear in the ρ-terms contained in P.

Lemma 7. *$\Phi(inc(\diamond), N, zero)$ is concretely reduced term.*

The proof could be done by induction on the degree variable N.

- *Basic case*: $N = 0$. $\Phi(inc(\diamond), 0, zero) = inc(zero) \cdot inc(zero)$ is a concretely reduced term by definition.
- *Inductive case*: Using the unfolding rule the term $\Phi(inc(\diamond), N+1, zero)$ is equal to $inc(\Phi(inc(\diamond), N, zero))$. This term is concretely reduced term, since $inc(\Phi(inc(\diamond), N, zero))$ is concretely reduced using the inductive hypothesis, thus the term $\Phi(inc(\diamond), N+1, zero)$ is concretely reduced term.

Theorem 8. *The extension of* MDG *by ρ-term is conservative.*

Since a ρ-term is concretely reduced. Thus the definition of *well-formedness* of MDG still valid. This fact ensures that all the proven results for MDG [10] remain true for ρ-MDG.

3.3. Abstract state enumeration with ρ-terms

Having incorporated ρ-terms in the syntax of DF, we need some extensions of the existing algorithms for MDGS. In this section, we present an extension of the reachability analysis algorithm that includes the appropriate handling of ρ-terms.

```
1.  Proc Generalize(Q, K, v, t)
2.      if t is a generator then t := t[Dvar(t)#K/Dvar(t)];
3.      for each equation in Q like v = rhs where rsh is not a ρ-term
4.          replace rhs by t;
5.      end for
6.  end Generalize
```

Given a ρ-term t and an abstract variable v, this procedure generalizes the variable v to this ρ-term with its degree variable replaced by a fresh one, obtained by concatenating the degree variable of t with a value of the counter K. This counter counts the number of passes through the reachability analysis loop (i.e., the number of transitions by which the state machine advanced from the initial state). The second condition in line 3 is necessary because we need to generalize again during reachability analysis (see

example 11). The procedure is called by a modified version of the ReAn′ described below:

```
1.  Proc ReAn′(D, v, t)
2.      K := 0;  F_I := Generalize(F_I, K, v, t);  R := F_I;  Q := F_I;
3.      loop
4.          K := K + 1;
5.          Q := Generalize(Q, K, v, t)
6.          I := Fresh(X, K);
7.          N := RelP′({I, Q, F_T}, X ∪ Y, η);
8.          N := Unfold(N)
9.          Q := PbyS′(N, R);
10.         if Q = ⊥ then return success;
11.         R := PbyS′(R, Q);
12.         R := Disj′(R, Q);
13.     end loop;
14. end ReAn′;
```

where D is a description of an abstract machine as described in Section 2, v is a state variable to be generalized and t is a ρ-term.[3]

We use a modified version of the relational product, the disjunction algorithm and the prune-by-subsumption algorithms. These new versions are extended by suitable rules to handle ρ-terms.

In *line* 2 we generalize the state variable v to the ρ-term t. This operation is done on the DF describing the initial state. The new operations[4] to perform at each iteration in the loop are:

Line 5: Generalizing the variable v in the frontier set Q. This operation can be just a renaming of the degree variable of a ρ-term.

Line 7: Computing the states reachable in one transition from the states in Q.

Line 8: unfolding each ρ-term in N by using the unfolding rules given in Section 3. For example, the unfolding of the ρ-term $eqz(\Phi(inc(\diamond), M, zero)) = 1$ gives $eqz(zero) = 1$ and $eqz(inc(\Phi(inc(\diamond), M', zero))) = 1$.

It is not difficult to generalize ReAn′ when several state variables cause divergence. The variable to be generalized and the ρ-term are supplied by the user after observation of the trace of the original ReAn algorithm. Unlike the generalization by variable where we lose any partial interpretations of uninterpreted functions, our method allows applying those rules during reachability analysis, by using the unfolding rules. This permits a useful simplification of the sets of states, thus reducing the possibility of false negative answers to invariant checking that can result from the simple variable generalization.

[3] For the moment, this term is supplied by the user, we plan in the near future to infer automatically this term from structural pattern of divergence.

[4] The other operations remain unchanged.

4. Examples

In this section, we illustrate our method on three different examples. Each example is chosen to demonstrate one particular aspect of the method. The first one shows that in some cases the generalization of the initial state is sufficient. Also, we give an outline of the unification algorithm of ρ-terms [5]. The second one illustrates the use of rewriting rules to obtain a regular structure. For this example, it also suffices to generalize the initial state using a ρ-term. The third and final example is more complicated and requires more than one generalization during reachability analysis.

Example 9. Consider a synchronous machine which includes a data register *count*, a multiplexer *mux*, and a functional block represented by the uninterpreted function symbol *inc* which takes *count* as its input and produces an abstract value $inc(count)$. The transition relation *Tr* of this machine is as follows:

$$Tr \equiv ((\underline{y} = 0) \wedge count' = count) \vee ((\underline{y} = 1) \wedge count' = inc(count))$$

where $count'$ is the next state variable of the register.

Suppose that *count* initially contains a generic constant *zero*. If we explore the state space using ReAn, the procedure never terminates and generates an unbounded sequences of values for count: $zero, inc(zero), inc(inc(zero)), \ldots, inc^k(zero)$. The variable to be generalized is *count* and the ρ-term is $H = \Phi(inc(\diamond), N, zero)$. Hence, we use ReAn$'$ to do reachability analysis as follows:

$$\mathsf{ReAn}'((\{y\}, \{count\}, \emptyset, count = zero, Tr, \emptyset), count, H).$$

In line 5, the call to $\mathsf{Gen}(count = zero, 0, count, H)$ returns a ρ-DF representing the initial state of the machine

$$P_0 = [count = \Phi(inc(\diamond), N_0, zero)]$$

The next states computed in line 7 are described by the formula

$$P_1 = (count = \Phi(inc(\diamond), N_1, zero)) \vee (count = inc(\Phi(inc(\diamond), N_1, zero))) \quad (1)$$

The ρ-DF formula in Eq. (1) represents the set

$$Set^{\psi}(P_1) = \{\gamma \in \Gamma^{\psi}_{\{count\}} \mid \psi_{\xi}\gamma \models \exists N_1.P_1\}$$

and the set represented by initial ρ-DF P_0 is

$$Set^{\psi}(P_0) = \{\gamma \in \Gamma^{\psi}_{\{count\}} \mid \psi_{\xi}\gamma \models \exists N_0.P_0\}$$

The problem now is to show that

$$Set^{\psi}(P_1) \subseteq Set^{\psi}(P_0)$$

This inclusion is checked by $\mathsf{PbyS}'(P_1, P_0)$. Informally, this operation can be viewed as $\mathsf{PbyS}(\Omega(P_1), \Omega(P_0))$. There are two edges issuing from the node *count* in P_1, labeled by $\Phi(inc(\diamond), N_1, zero)$ and $inc(\Phi(inc(\diamond), N_1, zero))$, respectively. There is one edge issuing from the same node *count* in P_0, labeled by $\Phi(inc(\diamond), N_0, zero)$. It remains to show that the ρ-terms in P_1 are included in the ρ-term in P_0.

For the first subproblem, $\forall N_1.\exists N_0.\Phi(inc(\diamond), N_1, zero) \subseteq \Phi(inc(\diamond), N_0, zero)$ we replace N_0 by N_1. Hence, we get $\Phi(inc(\diamond), N_1, zero) \subseteq \Phi(inc(\diamond), N_1, zero)$. For the second, $\forall N_1.\exists N_0.inc(\Phi(inc(\diamond), N_1, zero)) \subseteq \Phi(inc(\diamond), N_0, zero)$, we replace the variable N_0 by $N_1 + 1$. By applying the unfolding rules, we get:

$$inc(\Phi(inc(\diamond), N_1, zero)) \subseteq inc(\Phi(inc(\diamond), N_1, zero)$$

Therefore, after one pass through the loop, the newly reached states are covered by the initial states and the procedure terminates.

Example 10. Consider a more complex synchronous circuit which consists of a data register *count*, two multiplexers mux_1 and mux_2, and three functional blocks represented by uninterpreted function symbols *inc*, *dec*, and *eqz*. The functions *inc* and *dec* take as their input *count* and produce an abstract output $inc(count)$ and $dec(count)$, respectively. The cross-term *eqz* takes as its abstract input *count* and produces a concrete output of sort *bool*. The transition relation R of this machine is as follows:

$$\begin{aligned} R \equiv [&((\underline{y} = \underline{0}) \wedge count' = inc(count)) \vee \\ &((\underline{y} = \underline{1}) \wedge eqz(count) = \underline{0} \wedge count' = dec(count)) \vee \\ &((\underline{y} = \underline{1}) \wedge eqz(count) = \underline{1} \wedge count' = count)] \end{aligned}$$

where $count'$ represents the next state variable of *count*.

We define the following rewriting rules to give a partial meaning to the function symbols:

$$\begin{aligned} &(1) \quad eqz(zero) \rightarrow \underline{1} \\ &(2) \quad eqz(inc(x)) \rightarrow \underline{0} \\ &(3) \quad dec(inc(x)) \rightarrow x \end{aligned}$$

Suppose that register *count* initially contains a generic constant *zero*. Reachability analysis of this machine produces an infinite number of states for the register *count*, containing the values $zero, inc(zero), inc(inc(zero)), \ldots$. This regular structure is obtained by removing the *dec* operator by rewites. This divergence suggests to generalize the register *count* to the ρ-term $H = \Phi(inc(\diamond), N, zero)$. The initial state is thus described by the ρ-DF: $s_0 = [count = H]$.

After one transition, the reached states are:

$$s_1 = [eqz(H[N_1/N]) = \underline{1} \wedge count = H[N_1/N]],$$

$$s_2 = [eqz(H[N_1/N]) = \underline{0} \wedge count = dec(H[N_1/N])], \text{ and}$$

$$s_3 = [count = inc(H[N_1/N])].$$

It is clear that s_3 is covered by s_0, since $inc(H[N_1/N]) \subseteq H$ (see the first example). The state s_1 is also covered by the initial state, because

- If $N_1 = 0$, then s_1 can be rewritten to:

$$\begin{aligned} s_1 &= [eqz(\Phi(inc(\diamond), 0, zero)) = \underline{1} \\ &\qquad \wedge\ count = \Phi(inc(\diamond), 0, zero)] \\ &\qquad\qquad (\text{ by applying the unfolding substitution}) \\ &= [eq(zero) = \underline{1} \wedge count = zero] \\ &\qquad\qquad (\text{ by applying the unfolding rules}) \\ &= [count = zero] \ (\text{ simplification by rewriting with rule (1)}) \\ &= s_1' \end{aligned}$$

We can see immediately that s_1' is an instance of the initial state s_0, by replacing the degree variable N of H by 0.

- If $N_1 = N' + 1$, s_1 can be rewritten to:

$$\begin{aligned} s_1 &= [eqz(\Phi(inc(\diamond), N' + 1, zero)) = \underline{1} \\ &\qquad \wedge\ count = \Phi(inc(\diamond), N' + 1, zero)] \\ &\qquad\qquad (\text{ by applying the unfolding substitution}) \\ &= [eqz(inc(\Phi(inc(\diamond), N', zero))) = \underline{1} \\ &\qquad \wedge\ count = inc(\Phi(inc(\diamond), N', zero))] \\ &\qquad\qquad (\text{ by applying the unfolding rules}) \\ &= [\underline{0} = \underline{1} \wedge count = inc(\Phi(inc(\diamond), N', zero))] \\ &\qquad\qquad (\text{ simplification by rewriting with rule (2)}) \\ &= \perp \text{ (It means that in this case } s_1 \text{ is unreachable.)} \end{aligned}$$

It remains to show that the sets of states described by s_2 is covered by s_0.

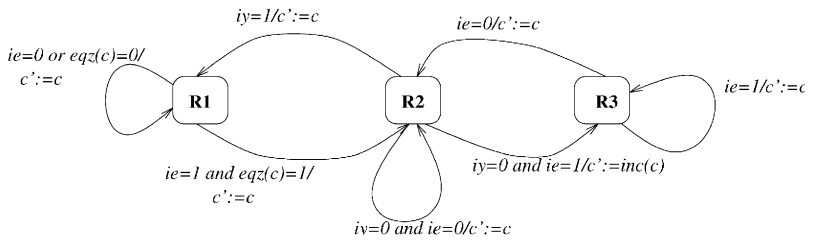

Fig. 1. A state machine.

We have:

- If $N_1 = 0$, then s_2 can be rewritten to:

$$\begin{aligned} s_2 &= [eqz(\Phi(inc(\diamond), 0, zero)) = \underline{0} \\ &\quad \wedge\ count = dec(\Phi(inc(\diamond), 0, zero))] \\ &\qquad (\text{ by applying the unfolding substitution}) \\ &= [eqz(zero) = \underline{0} \wedge count = dec(zero)] \\ &\qquad (\text{ by applying the unfolding rules}) \\ &= [\underline{1} = \underline{0} \wedge count = dec(zero)] \\ &\qquad (\text{ simplification by rewriting with rule (1)}) \\ &= \bot (\text{It means that in this case that } s_2 \text{ is unreachable.}) \end{aligned}$$

- If $N_1 = N' + 1$, then s_2 can be rewritten to:

$$\begin{aligned} s_2 &= [eqz(\Phi(inc(\diamond), N' + 1, zero)) = \underline{0} \\ &\quad \wedge\ count = dec(\Phi(inc(\diamond), N' + 1, zero))] \\ &\qquad (\text{ by applying the unfolding substitution}) \\ &= [eqz(inc(\Phi(inc(\diamond), N', zero))) = \underline{0} \\ &\quad \wedge\ count = dec(inc(\Phi(inc(\diamond), N', zero)))] \\ &\qquad (\text{ by applying the unfolding rules}) \\ &= [count = \Phi(inc(\diamond), N', zero)] \\ &\qquad (\text{ simplification by rewriting with rules (2) and (3)}) \\ &= s_2' \end{aligned}$$

s_2' is instance of s_0, by letting N equal to N'.

Example 11. Our third example concerns the synchronous machine shown in Fig. 1. It consists of three states **R1**, **R2** and **R3**. We use one state variable R of concrete sort $\{R_1, R_2, R_3\}$ to describe the behavior of this machine.

The transition relation is as follows:

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

Define the following rewriting rules:

$$
\begin{aligned}
eqz(zero) &\rightarrow \underline{1} \\
eqz(inc(x)) &\rightarrow \underline{0}
\end{aligned}
$$

The initial state of this machine is $s_0 = [\underline{R} = R_1 \wedge c = zero]$.

The states reached after one transition are $s_1 = [\underline{R} = R_1 \wedge c = zero]$, and $s_2 = [\underline{R} = R_2 \wedge c = zero]$.

State s_1 is covered by s_0, hence we continue the analysis with s_2. After the next transition, there are three possible states

$$
\begin{aligned}
s_3 &= [\underline{R} = R_3 \wedge c = inc(zero)], \\
s_4 &= [\underline{R} = R_2 \wedge c = zero], \qquad \text{and} \\
s_5 &= [\underline{R} = R_1 \wedge c = zero].
\end{aligned}
$$

s_4 is covered by s_2 and s_5 is covered by s_0. If we continue the exploration, we would find that the procedure never terminates. The divergence occurs because the value of the register c can be unbounded sets of values:

$$zero, inc(zero), inc(inc(zero)), \ldots$$

This divergence suggests to generalize the register c to the ρ-term $H = \Phi(inc(\diamond), N, zero)$.

The initial state becomes $s_0 = [\underline{R} = R_1 \wedge c = H]$.

After one transition, we can reach three possible states:

$$
\begin{aligned}
s_1 &= [\underline{R} = R_1 \wedge c = H[N_0/N]], \\
s_2 &= [\underline{R} = R_1 \wedge eqz(H[N_0/N]) = \underline{0} \wedge c = H[N_0/N]], \text{ and} \\
s_3 &= [\underline{R} = R_2 \wedge eqz(H[N_0/N]) = \underline{1} \wedge c = H[N_0/N]].
\end{aligned}
$$

- If $N_0 = 0$, then

$$\begin{aligned} s_2 &= [\underline{R} = R_1 \wedge eqz(zero) = \underline{0} \wedge c = zero] \\ &= \bot(\text{this state is unreachable}) \\ s_3 &= [\underline{R} = R_2 \wedge eqz(zero) = \underline{1} \wedge c = zero] \\ &= [\underline{R} = R_2 \wedge c = zero] \\ &= s_3' \end{aligned}$$

- If $N_0 = N' + 1$, then

$$\begin{aligned} s_2 &= [\underline{R} = R_1 \wedge eqz(inc(H[N'/N])) = \underline{0} \wedge c = inc(H[N'/N])] \\ &= [\underline{R} = R_1 \wedge c = inc(H[N'/N])] \\ s_3 &= [\underline{R} = R_2 \wedge eqz(inc(H[N'/N])) = \underline{1} \wedge c = inc(H[N'/N]] \\ &= \bot(\text{this state is unreachable}) \end{aligned}$$

The register c in s_3' must be generalized again, and we then continue the analysis from $s_3'' = [\underline{R} = R_2 \wedge c = H]$ (H replacing $zero$).

After one transition we reach

$$\begin{aligned} s_4 &= [\underline{R} = R_3 \wedge c = inc(H[N_0/N])], \\ s_5 &= [\underline{R} = R_2 \wedge c = H[N_0/N]], \qquad \text{and} \\ s_6 &= [\underline{R} = R_1 \wedge c = H[N_0/N]. \end{aligned}$$

s_5 is covered by s_3'' and s_6 is covered by s_0, hence we continue the analysis from s_4.

After one transition, we reach

$$\begin{aligned} s_7 &= [\underline{R} = R_3 \wedge c = inc(H[N_0/N][N_1/N_0])] \text{ and} \\ s_8 &= [\underline{R} = R_2 \wedge c = inc(H[N_0/N][N_1/N_0])]. \end{aligned}$$

s_7 is covered by s_4 and s_8 is covered by s_3, since $inc(H[N_0/N][N_1/N_0]) \subseteq H[N_0/N]$.

After three transitions the procedure terminates.

5. Conclusions

The non-termination problem of reachability analysis is a severe limitation of the methods based on abstract state machines. We have presented a new approach based on schematization using ρ-terms, to finitely represent the infinite sets of states generated during reachability analysis. Schematization presents a suitable formalism to deal explicitly by finite means with infinites families of objects, i.e., it describes by finite expressions infinites sets of terms. It permits to manipulate effectively the schematized sets. It is constructed so that the unification problem and the inclusion problem

for the schematized sets are decidables. This means that we can use these operations in the reachability analysis and invariant checking. We have also proposed an extension to the syntax of MDGS and to the reachability analysis algorithm to incorporate ρ-terms. Furthermore, schematization enlarges the set of states just enough to deal with non-termination, since the generalization is performed by a ρ-term representing a more restricted form of information than a fresh free variable used in the earlier approaches. Thus, the false negatives introduced by generalization with variables are considerably reduced, because the ρ-term represents all the possibles states of the machine, and contains only symbols defined in the machine.

Future work is directed to deriving sufficient conditions for detecting divergence of reachability analysis. These conditions can be described as structural patterns of the transition relation of a given abstract state machine. It would be interesting to have a method that automatically infers a ρ-term from the structural pattern of divergence.

Acknowledgements

The work was partially supported by NSERC Canada-Nortel Cooperative Research Grant *CRD* 191958.

References

[1] S. Bose, A.L. Fisher, Automatic verification of synchronous circuits using symbolic logic simulation and temporal logic, in: Proc. IMEC–IFIP Workshop on Applied Formal Methods for Correct VLSI Design, 1989.

[2] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. 35 (8) (1986) 677–691.

[3] R.E. Bryant, D.L. Beatty, C.-J.H. Seger, Formal hardware verification by symbolic ternary trajectory evaluation, In 28th ACM/IEEE Design Automation Conf., 1991.

[4] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, D.L. Dill, Symbolic model checking for sequential circuit verification, IEEE Trans. Comput. Aided Design 13 (4) (1994) 401–424.

[5] H. Chen, J. Hsiang, Recurrence domains: their unification and application to logic programming, Inform. and Comput. 122 (1995) 45–69.

[6] H. Chen, J. Hsiang, H.-C. Kong, On finite representations of infinite sequences of terms, in: S. Kaplan, M. Okada (Eds.), Proc. 2nd Internat. Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada), Lecture Notes in Computer Science, Vol. 516, Springer, Berlin, June 1990, pp. 100–114.

[7] H. Cho, G. Hachtel, S.-W. Jeong, B. Plessier, E. Schwarz, F. Somenzi, ATPG aspects of FSM verification, in: Internat. Conf. on Computer-Aided Design, 1990.

[8] L. Claesen, F. Proesmans, E. Verlind, H. De Man, SFG-tracing: a methodology for the automatic verification of MOS transistor level implementations from high level behavioral specifications, in: P.A. Subrahmanyam (Ed.), Internat. Workshop on Formal Methods in VLSI Design, Miami, FL, January 1991.

[9] H. Comon, Completion of rewrite systems with membership constraints, in: W. Kuich (Ed.), Proc. 19th ICALP Conf., Wien (Austria), Lecture Notes in Computer Science, Vol. 623, Springer, Berlin, July 1992, pp. 392–403; exists also as Research report 699, LRI, Orsay.

[10] F. Corella, Z. Zhou, X. Song, M. Langevin, E. Cerny, Multiway decision graphs for automated hardware verification, Formal Methods System Design 10 (1) (1997) 7–46.

[11] O. Coudert, J.C. Madre, A unified framework for the formal verification of sequential circuits, in: Internat. Conf. on Computer-Aided Design, 1990.

[12] B. Gramlich, Unification of term schemes — theory and applications, SEKI Report SR-88-18, Universität Kaiserslautern, Germany, 1988.
[13] M. Hermann, On the relation between primitive recursion, schematization, and divergence, in: H. Kirchner, G. Levi (Eds.), Proc. 3rd Conf. on Algebraic and Logic Programming, Volterra (Italy), Lecture Notes in Computer Science, Vol. 632, Springer, Berlin, September 1992, pp. 115–127.
[14] H. Kirchner, Schematization of infinite sets of rewrite rules generated by divergent completion process, Theoret. Comput. Sci. 67 (2–3) (1989) 303–332.
[15] G. Salzer, The unification of infinite sets of terms and its applications, in: A. Voronkov (Ed.), Proc. 3rd Internat. Conf. on Logic Programming and Automated Reasoning, St. Petersburg (Russia), Lecture Notes in Computer Science (in Artificial Intelligence), Vol. 624, Springer, Berlin, July 1992, pp. 409–420.
[16] H.J. Touati, H. Savoj, B. Lin, R.K. Brayton, A. Sangiovanni-Vincentelli, Implicit state enumeration of finite state machines using BDDs, in: Internat. Conf. on Computer-Aided Design, 1990.
[17] Z. Zhou, X. Song, S. Tahar, F. Corella, E. Cerny, M. Langevin, Formal verification of the island tunnel controller using multiway decision graphs, in: Proc. Internat. Conf. on Formal Methods in Computer Aided Design (FMCAD'96), Palo Alto, California, USA, November 1996.