A Partial-Order Approach to the Verification of Concurrent Systems: Checking Liveness Properties

Dominique Bolignano

Bull, 78340 Les Clayes-sous-Bois, France
D.Bolignano@frcl.bull.fr

Abstract. We present the foundations of an approach for exploiting the partial ordering of events in the verification of concurrent systems. The main objective of the approach is to avoid the state explosion that is due to the use of the standard interleaving semantics of concurrency. The approach has been applied successfully to the verification of complex hardware and software systems such as a shared memory with multi-cache for a multi-processor architecture. The technique is described for finite state systems and applied to the checking of liveness properties using a model-checking approach. Most existing approaches use the partial ordering of events as a means of reducing the number of traces to check: checking is in particular done on normal totally ordered traces and the reduction (i.e. the selection of representatives) is dependent on the property at hand. We strongly differ from these approaches by directly performing the checking on the partial order graphs themselves, not on particular linearizations. These partial order graphs are not dependent on the property to check: only the checking is. For this we introduce models based on tuples to represent partial orders, and a special kind of automaton that we call partial order automaton which generates the set of all possible partial ordering that can result from the execution of a system.

1 Introduction

Various approaches have been developed based on the use of partial orders. They have first concentrated on the checking of specific properties: the verification method of Katz and Peled [7], the method of Mac Millan [9] and the model checking approaches of Valmari [11], and Godefroid [5] were limited to dealing with safety properties, termination, local and stable properties. Later the techniques have been generalized by Godefroid and Wolper [6], Valmari [12], and Peled [10].

Although the various approaches are based on different techniques (the approaches in [10], [12] are based on stubborn set whereas [5] and [6] are based on trace automata) in all these approaches, equivalence classes are identified, and at least one representative per class is checked. In [6] for example, the equivalence classes are Mazurkievicz's traces [8]. The first characteristic of these approaches is that representatives are particular linearization (total ordering) that have

to be chosen and checked. But more importantly, class partitioning is strongly dependant on the property at hand: the property should be true for the representative iff it is true for the whole equivalence class. In our technique, verification is done on models directly representing the partial orders that result from the various possible execution of a system. No specific linearization is used, and the construction of the model is not done with respect to a particular property.

The approach has been applied successfully to the verification of complex hardware and software systems such as a shared memory with multi-cache for a multi-processor architecture. Although relying on asynchronous parallelism the approach can be used in the synchronous case when communication delays are not constant (serial lines...) or more generally when the correctness of the design should not depend on particular timings so as to allow for example for future technology evolutions. We present here the approach foundations.

The paper is organized as follows. In the next section we recall basic concepts and notations of Mazurkiewicz's traces and then introduce the notion of *feasible tuple* that we use for representing the partial ordering of events that result from the parallel composition of n processes. The third section is devoted to the exploitation of partial order. We introduce partial order automata and use them for the checking of order-based properties. In section 4 we show how to generate such partial order automata, starting from a automaton description of each process. We conclude the paper with an illustration of the technique on an example and then with a discussion on the benefits, limitations and perspective of the proposed approach.

2 Representation of partial order

As in [6] we consider a system as being the composition of n concurrent processes P_i. Each process is described using a language L_i of ω-words (i.e. functions of $N \rightarrow \Sigma_i$) defined over an alphabet Σ_i. Each language is defined using a generalized Büchi automaton A_i, i.e. A_i is a tuple $(\Sigma, Q, \Delta, q_0, \mathcal{F})$ where:

- Σ is an alphabet,
- Q is a set of states,
- $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation,
- $q_0 \in Q$ is the starting state, and
- $\mathcal{F} = \{F_1, ..., F_k\} \subseteq 2^Q$ is a family of sets of accepting states.

A word is accepted by a generalized Büchi automata if and if the automaton has an infinite execution sequence that intersects infinitely often each set F_i of $\mathcal{F}$. Formally we define the concept of a run over a ω-word $a_1 a_2 ...$ as being an infinite sequence $q_0 \stackrel{a_1}{\rightarrow} q_1 \stackrel{a2}{\rightarrow} q_2 ...$ (i.e. a function of $N \rightarrow (Q \times \Sigma)$) such that $(q_i, a_{i+1}, q_{i+1}) \in \Delta$ for all $i \geq 0$. A run is said to be accepting if and only if for each F_j in $\mathcal{F}$ there exists infinitely many q_i in the run such that q_i is in F_j.

Now we recall some basic results presented in [8]. We associate to each finite word $w = a_1 a_2 ... a_l$ the set $Set(w)$ of pairs of the form (a_i, n_i) where $n_i =$

$card\{j|j \leq i, a_i = a_j\}$ and $1 \leq i \leq l$. This set is ordered by the relation $Ord(w)$ where

$$\{((a_i, n_i), (a_j, n_j))|1 \leq i \leq j \leq l\}$$

This relation is the reflexive transitive closure of the relation $Succ(w) = \{((a_i, n_i), (a_{i+1}, n_{i+1}))|1 \leq i < l\}$. Given any word w defined on an alphabet Σ the projection of w onto Σ' is a word noted w/Σ' on alphabet $\Sigma \cap \Sigma'$ defined as follows:

- $\epsilon/\Sigma' = \epsilon$
- $(wa)/\Sigma' = (w/\Sigma')a$, if $a \in \Sigma'$
- $(wa)/\Sigma' = (w/\Sigma')$, if $a \notin \Sigma'$

Proposition 1. *For any word w:* $w/(\Sigma \cap \Sigma') = (w/\Sigma)/\Sigma'$

Proof. Straightforward by structural induction on words.

Now $Set(w)$, $Ord(w)$, $Succ(w)$ can be extended in a straightforward manner to apply to any ω-word w. According to [4], for any ω-word, $\lim_{n\to\infty} w^{[n]}$ where $w^{[n]}$ is the prefix of size n exists and is equal to w. The domain of Σ^N of ω-words on alphabet Σ with the distance between two different ω-words being defined as the inverse of the size of the biggest common (finite) prefix is shown to be a metric space. For any ω-word w, of Σ^N, w/Σ' can be defined as the projection w'/Σ' on the finite word w' when w can be obtained as the concatenation of a finite word w' and a ω-word $w'' \in (\Sigma - \Sigma')^N$. It is defined as $\lim_{n\to\infty} (w^{[n]})/\Sigma'$ otherwise.

Given n languages L_1 ,...,L_n of ω-words, describing n processes with respective alphabets Σ_1,...,Σ_n, the parallel composition of the n processes is described using the alphabet $\Sigma_1 \cup ... \cup \Sigma_n$ and the language

$$\{w \in (\Sigma_1 \cup ... \cup \Sigma_n)^N | w/\Sigma_1 \in L_1 \wedge ... \wedge w/\Sigma_n \in L_n\}$$

The parallel composition of these n processes will be noted $||_{(\Sigma_1,...,\Sigma_n)}(L_1, ..., L_n)$. A precise modelling of the parallel composition of processes would require the use of finite prefixes in addition to that of ω-words so as to be able to model deadlocks resulting from the parallel composition of processes. Such a modelling is done in [2]. For the sake of conciseness we only use ω-words here. The parallel composition can also be defined for languages on finite words: $||_{(\Sigma_1,...,\Sigma_n)}(L_1, ..., L_n)$=$\{w \in (\Sigma_1 \cup ... \cup \Sigma_n)^* | w/\Sigma_1 \in L_1 \wedge ... \wedge w/\Sigma_n \in L_n\}$.

A tuple of finite words (a finite tuple for short) or a tuple of ω-words (a ω-tuple for short) can be associated to a tuple of alphabets meaning that each component of the tuple is defined on the corresponding alphabet. This tuple of alphabets will be called a signature (e.g. $(\Sigma_1, ..., \Sigma_n)$ is the signature of $(L_1, ..., L_n)$). We will only consider tuples with $n \geq 2$.

In [8] the partial order is defined by identifying equivalence classes between words. Here we proceed differently. We only consider systems consisting of the parallel composition of n processes and derive the partial ordering directly from the process parallel composition structure. For this we use particular tuples of words:

Definition 2. A tuple (a finite or ω-tuple) $(w_1, ..., w_n)$ with signature $(\Sigma_1,...,\Sigma_n)$ will be said to be *feasible* for (i.e. with respect to) signature $(\Sigma_1,...,\Sigma_n)$ if and only if the composition $||_{(\Sigma_1,...,\Sigma_n)}(\{w_1\}, ..., \{w_n\})$ is not empty. The feasibility will be expressed more formally by using the predicate $feasible_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$.

Lemma 3. *Given a tuple $(w_1, ..., w_n)$ of n words with signature $(\Sigma_1,...,\Sigma_n)$, than $feasible_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$ iff*

$$(\forall i, j. 1 \leq i \leq j \leq n \Rightarrow w_i/(\Sigma_i \cap \Sigma_j) = w_j/(\Sigma_i \cap \Sigma_j))$$

Proof. Directly comes from the definition of the $||$ operator.

Definition 4. If $(w_1, ..., w_n)$, is a feasible tuple for $(\Sigma_1, ..., \Sigma_n)$ we define $Ord_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$ to be the reflexive and transitive closure of the relation $\bigcup_{1\leq i\leq n} Ord(w_i)$, $Set_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$ to be $\bigcup_{1\leq i\leq n} Set(w_i)$ and $Succ_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$ to be the relation $\bigcup_{1\leq i\leq n} Succ(w_i)$.

Lemma 5. *The relation $Ord_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$ is a partial ordering on the set $Set_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$, for any feasible tuple $(w_1, ..., w_n)$.*

Furthermore, the reflexive and transitive closure of the relation $Succ_{(\Sigma_1,...,\Sigma_n)}(\{w_1\}, ..., \{w_n\})$ is equal to $Ord_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n)$.

Finally $Ord_{(\Sigma_1,...,\Sigma_n)}(w_1, ..., w_n) = \bigcap_{w\in||_{(\Sigma_1,...,\Sigma_n)}(\{w_1\},...,\{w_n\})} Ord(w)$.

Proof. The transitivity follows immediately from the transitive closure property of the relation. The asymmetry is proved using the fact that

$$1 \leq i < j \leq n \Rightarrow (Set(w_i) \cap Set(w_j)) = Set(w_i/\Sigma_i \cap \Sigma_j) = Set(w_j/\Sigma_i \cap \Sigma_j)$$
$$1 \leq i < j \leq n \Rightarrow (Ord(w_i) \cap Ord(w_j)) = Ord(w_i/\Sigma_i \cap \Sigma_j) = Ord(w_j/\Sigma_i \cap \Sigma_j)$$

If w is a word (resp. a tuple) $\preceq_w$ will be the operator corresponding to $Ord(w)$ (resp. $Ord_{(\Sigma_1,...,\Sigma_n)}(w)$) and the operator $\prec_w$ will be defined accordingly: $x \prec_w y$ iff $x \preceq_w y \wedge x \neq y$. The prefix inclusion will be noted $\subset$ and so will the tuple prefix inclusion (i.e. $(t_1, ..., t_n) \subset (t'_1, ..., t'_n)$ iff $t_1 \subseteq t'_1 \wedge ... \wedge t_n \subseteq t'_n$ and $t_i \subset t'_i$ for at least one $i \in 1..n$). Given any kind of automaton A, $L(A)$ will stand for the language recognized by this automaton.

Definition 6. If $(w_1, ..., w_n)$ is a tuple with signature $(\Sigma_1, ..., \Sigma_n)$ then we define $wordsof_{(\Sigma_1,...,\Sigma_n)}((w_1, ..., w_n))$ to be $||_{(\Sigma_1,...,\Sigma_n)}(\{w_1\}, ..., \{w_n\})$.

By definition a tuple $(w_1, ..., w_n)$ is thus feasible for $(\Sigma_1, ..., \Sigma_n)$ if and only if $wordsof_{(\Sigma_1,...,\Sigma_n)}((w_1, ..., w_n))$ is not empty.

In the sequel we will use tuples of feasible words as models of partial orders. In [8] two equivalent notions (i.e. equivalence classes among words and dependant graphs) are used for the same purpose. The equivalence of the three notions is quite immediate in the case where a system is obtained as the parallel composition of n systems. We will neither prove, neither use this fact here.

Definition 7. Given a tuple $(L_1, ..., L_n)$ of n languages with signature $(\Sigma_1, ...\Sigma_n)$ we define $feasibleset_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)$ to be the following set of tuples:

$$\{(w_1, ..., w_n) | (w_1, ..., w_n) \in L_1 \times ... \times L_n \wedge feasible_{(\Sigma_1,...\Sigma_n)}(w_1, ..., w_n)\}$$

Corollary 8. *The parallel composition* $||_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)$ *of n languages is*

$$\bigcup_{t \in feasibleset_{(\Sigma_1,...\Sigma_n)}(L_1,...,L_n)} wordsof_{(\Sigma_1,...\Sigma_n)}(t)$$

Proof. We use the definitions 2, 7 and the fact that $||_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)$ is equal to $\bigcup_{(w_1,...,w_n) \in L_1 \times ... \times L_n} ||_{(\Sigma_1,...,\Sigma_n)}(\{w_1\}, ..., \{w_n\})$.

We now define an operator $\odot$ that allows for the concatenation of two tuples:

Definition 9. Given a finite tuple $(t_1, ..., t_n)$ and a tuple $(s_1, ..., s_n)$ we define $(t_1, ..., t_n) \odot (s_1, ..., s_n)$ to be the tuple $(t_1 s_1, ..., t_n s_n)$.

Lemma 10. *The concatenation* $(t_1, ..., t_n) \odot (s_1, ..., s_n)$ *of two tuples* $(t_1, ..., t_n)$ *and* $(s_1, ..., s_n)$ *feasible for a given signature* $(\Sigma_1, ...\Sigma_n)$, *is feasible with respect to the same signature.*

Proof. This directly follows from the definitions of the $\odot$ and $||$ operators, by noting that for any words s and t and alphabet Σ, $(st)/\Sigma = (s/\Sigma)(t/\Sigma)$

3 Partial Order Automata

We now introduce a special kind of automata that we call *partial order automata* as a representation of all finite partial ordering of events that can result from all possible execution of n concurrent processes.

Definition 11 Partial order automata. A partial order automata is a pair $\langle A_{PO}, h \rangle$ where A_{PO} is a generalized Büchi automata $A_{PO} = (\Sigma_{PO}, S, \Delta, s_0, \mathcal{F})$ and $h{:}\Sigma_{PO} \rightarrow (\Sigma_1)^* \times ... \times (\Sigma_n)^*$ a function whose range only contains feasible tuples. The language $L(\langle A_{PO}, h \rangle)$ recognized by such an automaton is defined as being the image of $L(A_{PO})$ by $h^{\omega}_{\odot}$ where $h_{\odot}{:}\Sigma^N_{PO} \rightarrow (\Sigma_1)^* \times ... \times (\Sigma_n)^*$ and $h^{\omega}_{\odot}(t)$ is defined for any ω-word t as $\lim_{n \rightarrow \infty} h_{\odot}(t^{[n]})$. (with $h_{\odot}{:}\Sigma^*_{PO} \rightarrow (\Sigma_1)^* \times ... \times (\Sigma_n)^*$ defined recursively: $h_{\odot}(\epsilon) = (\epsilon, ..., \epsilon)$ and $h_{\odot}(wx) = h_{\odot}(w) \odot h(x))$ for any word w of Σ^*_{PO} and element x of Σ^*_{PO}.

Now if a partial order automaton $\langle A_{PO}, h \rangle$ is such that $L(\langle A_{PO}, h \rangle) = feasibleset_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)$ then it can be used according to corollary 8 as a description of a system. We will of course want to do the verification on the partial order automaton, without having to consider the various possible linearizations.

Instead of using a standard quintuple $(\Sigma, S, \Delta, s_0, \mathcal{F})$ to represent a generalized Büchi automaton we will in the case of partial automata, and to allow for

more compact representation, use quintuples $(\Sigma, S, \Delta, s_0, \mathcal{G})$ where $\mathcal{G}$ is a family of set of symbols of the alphabet instead of being a family $\mathcal{F}$ of set of states. Now a run will be accepted if and only if it intersects infinitely often each set of symbols. It is easy to verify that any quintuple $(\Sigma, S, \Delta, s_0, \mathcal{G})$ represents a generalized Büchi automata: just associate $(\Sigma, S \times \Sigma, \{((q_1, a), b, (q_2, b))|(q_1, b, q_2) \in \Delta, a \in \Sigma\}, (s_0, a_0), \bigcup_{G_i \in \mathcal{G}}\{\{(q, a)|q \in S \wedge a \in G_i\}\})$ where a_0 is any element of Σ.

4 Checking liveness properties

A number of properties satisfied by the set of words associated to a feasible tuple can be checked on the tuple itself in a very efficient manner and in particular without having to generate and navigate through the set of words.

The benefit of using partial order depends on the property at hand: if the property expresses all potential linearizations for all events of a given concurrent system, the use of partial order based techniques will be of little help. We illustrate the use of our models in a area of high potential benefits when properties express sequencing constraint on a subset of events.

If L is a language of ω-words on an alphabet Σ resulting from the parallel composition of n processes we concentrate here on properties that can be expressed using a Büchi automaton A_P on a subset Σ_P of Σ, in the form $L/\Sigma_P \subseteq L(A_P)$ (or that $L/\Sigma_P = L(A_P)$). Since liveness (as well as safety) properties can be represented using Büchi automata [1], the particular form of property we concentrate on is very general in theory, since we can take Σ_P to be Σ. But the technique described in the sequel finds its main interest and efficiency in the case where Σ_P is smaller than Σ. It is meant to be an illustration of the use of partial order automata for verification.

Lemma 12. *Given a finite tuple t that is feasible for $(\Sigma_1, ...\Sigma_n)$ and given F a subset of $Set_{(\Sigma_1,...\Sigma_n)}(t)$, that we call subset of concern, it is possible to associate to each element x of $Set_{(\Sigma_1,...\Sigma_n)}(t)$, a subset E_x of $Set_{(\Sigma_1,...\Sigma_n)}(t)$, such that*

$$y \in F \wedge (y \preceq_t x) \; iff \; y \in E_x \; (1)$$

Furthermore the annotation function $E : Set_{(\Sigma_1,...\Sigma_n)}(t) \rightarrow \mathcal{P}(Set_{(\Sigma_1,...\Sigma_n)}(t))$ that associates each element x with an annotation E_x is computable and can defined recursively as follows:

$$E_x = if \; x \in F \; then \; (\bigcup_{y \in \{y|(y,x) \in Succ_{(\Sigma_1,...\Sigma_n)}(t)\}} E_y) \cup \{x\}$$
$$else \; (\bigcup_{y \in \{y|(y,x) \in Succ_{(\Sigma_1,...\Sigma_n)}(t)\}} E_y)$$

where we suppose that by convention $\bigcup_{x \in \emptyset} X_x = \emptyset$

Proof. The relation $Succ_{(\Sigma_1,...\Sigma_n)}(t)$ is compatible with a partial order relation since the transitive closure of $Succ_{(\Sigma_1,...\Sigma_n)}(t)$ precisely is $Ord_{(\Sigma_1,...\Sigma_n)}(t)$ according to lemma 5. All domains (and in particular $Succ_{(\Sigma_1,...\Sigma_n)}(t)$) are finite.

It is thus straightforward to show that the function E is well defined for any element of its domain of definition. Then we prove that for any x of $Set_{(\Sigma_1,...\Sigma_n)}(t)$, E_x satisfies property (1). The property can be shown by natural induction on the size of the greatest strictly increasing chain leading to x. The initial case (i.e. for the elements for which the greatest chain is of size 1) is straightforward: the only element y such that $y \preceq_t x$ is x itself and $E_x = \{x\}$ or $E_x = \emptyset$ depending on whether x is in F or not. If the property is true for i and if x is such that the greatest strictly increasing chain leading to x is of size $i+1$. Then the set $\{y|(y,x) \in Succ_{(\Sigma_1,...\Sigma_n)}(t)\}$ of immediate predecessors is not empty and the size of the corresponding chain for immediate predecessors are lesser or equal to i. Furthermore any element y such that $y \preceq_t x$ is either x itself, or either lesser or equal to at least one of the immediate predecessors. These facts allow to conclude easily.

Note that annotating the whole set $Succ_{(\Sigma_1,...\Sigma_n)}(t)$ can be done in l steps where l is the cardinal of $Set_{(\Sigma_1,...\Sigma_n)}(t)$ and where each step consists in computing the annotation of a particular element as the distributed union of the annotations of its immediate successors. The annotation process just has to proceed in a order that is compatible with the partial order $Succ_{(\Sigma_1,...\Sigma_n)}(t)$.

Lemma 13. *Given a finite and feasible tuple of words, $(t_1,...,t_n)$, with a signature $(\Sigma_1,...\Sigma_n)$, and two sets F and F' such that $F' \subseteq F \subseteq Set_{(\Sigma_1,...\Sigma_n)}(t)$, then the annotations E_x and E'_x that can be computed according to the previous lemma using respectively F and F' satisfy for each element x of $Set_{(\Sigma_1,...\Sigma_n)}(t)$ the following equality: $E'_x = E_x \cap F'$.*

Proof. This follows directly from property (1) of previous lemma.

Since we are concerned here with properties that can be expressed on abstractions we will use the following property of feasible tuples.

Lemma 14. *If $t = (w_1,...,w_n)$ is a feasible tuple for $(\Sigma_1,...\Sigma_n)$ and if $Ord_{(\Sigma_1,...\Sigma_n)}(t)$ defines a total order, let say $Ord_{(\Sigma_1,...\Sigma_n)}(t)/\Sigma_P$, on the set $\{(a,n) \in Set_{(\Sigma_1,...\Sigma_n)}(t)|a \in \Sigma_P\}$ then the projection of the set $||_{(\Sigma_1,...\Sigma_n)}(\{w_1\},...,\{w_n\})$ onto Σ_P is a singleton (i.e. $\exists t'.\{w/\Sigma_P|w \in ||_{(\Sigma_1,...\Sigma_n)}(\{w_1\},...,\{w_n\})\} = \{t'\}$). Furthermore, if we note t/Σ_P the unique element of this singleton, we have the following property:*

$$Ord_{(\Sigma_1,...\Sigma_n)}(t)/\Sigma_P = Ord(t/\Sigma_P)$$

Proof. Any projection preserves ordering of elements that are not abstracted away as it is easy to check, and the indices of remaining elements are kept unchanged since projection removes all or no instances of a same element of the alphabet.

Proposition 15. *Given a language tuple $(L_1,...,L_n)$ of signature $(\Sigma_1,...\Sigma_n)$, Σ_P a subset of $\Sigma_1 \cup ... \cup \Sigma_n$ and a partial order automaton $\langle A_{PO}, h\rangle$*

such that $L(\langle A_{PO}, h\rangle) = feasibleset_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)$ *then for any tuple* t *of the* $L(\langle A_{PO}, h\rangle)$, *if* $Ord_{(\Sigma_1,...\Sigma_n)}(t)/\Sigma_P$ *is a total order then* $||_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)/\Sigma_P = L(\langle A_{PO}, h\rangle)/\Sigma_P$.

Proof. Directly follows from the lemmas ??, ?? and from corollary 8.

This provides us with a sufficient condition for checking a property expressed on a projection. In fact it gives more than that. First if one of the feasible tuple t of the language $L(\langle A_{PO}, h\rangle)$ is such that $Ord_{(\Sigma_1,...\Sigma_n)}(t)/\Sigma_P$ is not a total order this means that the ordering between events that we are willing to check is not imposed by process sequencing and synchronization but by interleaving: $||_{(\{x,u,v\},\{y,u,v\})}(\{xu, vx\}, \{uy, yv\})/\{x, y\} = \{xy, yx\}$ and $||_{(\{x\},\{y\})}(\{x\}, \{y\})/\{x, y\} = \{xy, yx\}$ but in the first system x and y are ordered in all feasible tuples, while they are not in the second. Our modelling of a system based on the language generated by a partial automaton keeps information relative to the partial ordering of events imposed by sequencing and synchronization constraints and would allow for the checking of more discriminating properties based on partial orders and in particular properties involving true concurrency aspects. For the sake of conciseness we will not investigate this possibility any further in the sequel and consider our technique only as means of simplifying verification and not as a means of expressing or verifying finer properties. Now with this objective, it is easy to always consider systems for which the projection of all feasible tuples always is a total order: just use $||_{(\Sigma_1,...\Sigma_n,\Sigma_P)}(L_1, ..., L_n, \Sigma_P^*)$ instead of $||_{(\Sigma_1,...\Sigma_n)}(L_1, ..., L_n)$. The two sets are equal, as it is easy to see by returning to the definition of the $||$ operator, and the first one always satisfies the total ordering condition: this one is imposed by the last component.

Definition 16. Given a partial automaton $\langle A, h\rangle$ for a signature $(\Sigma_1, ...\Sigma_n)$, (i.e. $A = (\Sigma, Q, \Delta, q_0, \mathcal{G})$) and Σ_P a subset of $\Sigma_1 \cup ... \cup \Sigma_n$ we define the following algorithm that constructs a partial order automaton $\langle A', h'\rangle$ (i.e. $A' = (\Sigma', Q', \Delta', q_0', \mathcal{G}')$) with unary tuples by adding new nodes or transitions at each step until failing or until reaching a fixed point in which no failing condition applies:

1. start with the automaton containing only one node (i.e. $q_0' = \{(q_0, (v, ..., v))\}$), and no transition where v is a constant not in $\Sigma_1 \cup ... \cup \Sigma_n$;
2. repeat the following operation until reaching a fixed point or failing: if $(nd, (w_1, ..., w_n))$ is a node of the current automaton and if nd' is such that $nd \xrightarrow{e} nd'$ is a transition of A and $h(e) = (w_1', ..., w_n')$,
   (a) compute annotations for the graph $Succ_{(\Sigma_1,...\Sigma_n)}((w_1, ..., w_n) \odot (w_1', ..., w_n'))$ using the algorithm described in lemma 12 and using $F = \{(a_j, n_j) \in Set_{(\Sigma_1,...\Sigma_n)}((w_1, ..., w_n) \odot (w_1', ..., w_n'))|a_j \in \Sigma_P \cup \{v\}\}$ as the subset of concern;
   (b) check that the annotations correspond to a total order[1] for the element

[1] This just means checking that corresponding annotations can be ordered as a chain of strictly increasing sets.

F; if not then **fail**; if yes then the ordered list of element of F has the form $(a'_1, n'_1), ..., (a'_l, n'_l)$ with $(a'_1, n'_1) = (v, 1)$;

(c) add $(nd, (w_1, ..., w_n)) \overset{(a'_2 ... a'_l, e)}{\rightarrow} (nd', (w''_1, ..., w''_n))$ where w''_i is v if $w_i w'_i \neq \epsilon$ and (a'_l, n'_l) is in the annotation of last element of the ordered set $Set(w_i w'_i)$ and ϵ otherwise;

(d) if there exists two nodes $(nd, (w_1, ..., w_n))$ and $(nd', (w'_1, ..., w'_n))$ such that $n = n'$ and $(w_1, ..., w_n) \subset (w'_1, ..., w'_n)$ then collapse the two nodes by replacing the second node by the first one in the automata.

3. the set $\mathcal{G}' = \{G'_1, ..., G'_k\}$ is computed from the set $\mathcal{G} = \{G_1, ..., G_k\}$ by letting each $G'_i = \{(w, e) \in Q' | e \in G_i\}$
4. the function h' is defined to be such that $h((w, e) = w$.

Theorem 17. *The algorithm of definition 16 terminates for any partial order automaton $\langle A, h\rangle$, of signature $(\Sigma_1, ...\Sigma_n)$ and for any subset Σ_P of $\Sigma_1 \cup ... \cup \Sigma_n$. It terminates by a failure if and only if there exists a word in the language generated by the partial automaton which does not impose a complete ordering of events in Σ_P. It terminates successfully by returning an automaton $\langle A', h'\rangle$ whose language is equal[2] to $(\bigcup_{t \in L(\langle A, \Sigma, h\rangle)} wordsof(t))/\Sigma_P$. Furthermore in that latter case, if the partial automaton describes the parallel composition of n processes (i.e. if $L(\langle A, h\rangle) = feasibleset_{(\Sigma_1, ...\Sigma_n)}(L_1, ...L_n)$ where $(L_1, ...L_n)$ is a language tuple of signature $(\Sigma_1, ...\Sigma_n)$) then L_P is the projection of the composition onto Σ_P:*

$$L(\langle A', h'\rangle) = (||_{(\Sigma_1, ...\Sigma_n)}(L_1, ...L_n))/\Sigma_P$$

Proof. The proof of the termination of the algorithm is straightforward since the nodes of the tree built by the algorithm range in the finite set $Q \times \{\epsilon, v\}^n$.

In order to show that $L(\langle A', h'\rangle) \subseteq (||_{(\Sigma_1, ...\Sigma_n)}(L_1, ...L_n))/\Sigma_P$ we associate to each run $r' = (q_0, (v, ..., v)) \overset{(w^1, e_1)}{\rightarrow} ... \overset{(w^{l-1}, e_{l-1})}{\rightarrow} (nd_l, (w^l_1, ..., w^l_n)) \overset{(w^l, e_l)}{\rightarrow} ...$ of A' the run $r = q_0 \overset{e_1}{\rightarrow} ... \overset{e_{l-1}}{\rightarrow} nd_l \overset{e_l}{\rightarrow} ...$ of A. We then show that according to lemma 14, t, the projection onto Σ_P of each element of $wordsof_{(\Sigma_1, ...\Sigma_n)}(h^\omega_\odot(e_1 ... e_{l-1} e_l ...))$, where $wordsof_{(\Sigma_1, ...\Sigma_n)}(h^\omega_\odot(e_1 ... e_{l-1} e_l ...))$ is the set of elements of $||_{(\Sigma_1, ...\Sigma_n)}(L_1, ...L_n)$ itself corresponding to r, exist, is the same for all elements and is equal to the element $t' = w^1 ... w^{l-1} w^l ...$ of $L(\langle A', h'\rangle)$ corresponding to r'. For this we first show by induction that the property is true for all finite prefixes $r'^{[i]}$, and $r'^{[i]}$ of size i, and then derive t and t' as the limit of the same series. The proof by induction relies in particular on the following fact: the tuple t being of the form $(w''_1, ..., w''_n)$, and the word $w_1 ... w_l$ being of the form $x_1 ... x_m$ where x_i are element of the alphabet, then for each $i \in 1..n$, the annotation of the last element of $Ord(w''_i)$, if we were to apply the algorithm of lemma 12 to tuple t taking $F = \{(x_m, k)\}$ as subset of concern, where (x_m, k) is the last instance of x_m in $Ord_{(\Sigma_1, ..., \Sigma_n)}(t)$, would be

[2] To be more precise it is only isomorphic since it is a set of unary tuples: $\{t | (t) \in L(\langle A', h'\rangle)\}$ would be equal.

$w_i^l[x_m/v]$ (the set obtained by substituting v by x_m in the set w_i^l), if x_m exists and ϵ otherwise.

Finally we prove using the same correspondence between runs of A and A' that the algorithm fails if and only if there exists a word in the language generated by the partial automaton which does not impose a complete ordering of events in Σ_P, and that any run of automaton A will have a corresponding element of A' if the algorithm does not fail.

5 Generating the partial order automata

Now we come to the construction of the partial order automata. We define here a basic algorithm, that can be further improved as done in [2]. The objective here is to show the existence of such an algorithm as well as the main idea behind it. We assume in the following that all automata have disjoint nodes so as to allow to use transition relation without having to make reference to the corresponding automata.

In order to introduce the algorithm we introduce a few additional notations: for any sequence $x_1...x_k$, the function $elems(x_1...x_k)$ returns the set $\{x_1, ..., x_k\}$ of elements of the sequence; $nd \overset{a_1...a_k}{\underset{nd_1...nd_{k-1}}{\Rightarrow}} nd'$ if and only if $nd \overset{a_1}{\rightarrow} .nd_1..nd_{k-1} \overset{a_k}{\rightarrow} nd'$ where k is potentially null; $nd \overset{a_1...a_k}{\Rightarrow} nd'$ if and only if $nd \overset{a_1...a_k}{\underset{nd_1...nd_{k-1}}{\Rightarrow}} nd'$ for some sequence of nodes $nd_1...nd_{k-1}$; $nd \overset{w}{\Rightarrow}_{St} nd'$ where St is a set of states if and only if $nd \overset{w}{\underset{r}{\Rightarrow}} nd'$ for some r with $elems(r) \cap St = \emptyset$ and $\{nd, nd'\} \subseteq St$.

Definition 18. Given a tuple $(A_1, ..., A_n)$ of n generalized Büchi automata (i.e. $A_i = (\Sigma_i, Q_i, \Delta_i, q_i^0, \mathcal{F}_i)$) and a tuple $(St_1, ..., St_n)$ where each St_i is a subset of Q_i, we define an algorithm that builds a partial order automaton $\langle A, h \rangle$ (i.e. $A = (\Sigma, Q, \Delta, q^0, \mathcal{G}_i)$) in a stepwise manner starting with an automaton with an empty transition relation and with only one node: the initial node $(q_1^0, ..., q_n^0)$. The family $\mathcal{G}$ associates one set $G_{(i,j)}$ for each set F_j of each $\mathcal{F}_i$, $i \in 1..n$. All sets are initially empty. The algorithm proceeds by applying the following operations repeatedly until the algorithm fails or a fixed point is reached and no failing condition applies:

1. if for some node $(nd_1, ..., nd_n)$ of the current automaton, there are some $i, j \in 1..n$ such that $nd_i \overset{w_i}{\Rightarrow}_{St_i} nd_i'$ and $nd_j \overset{w_j}{\Rightarrow}_{St_j} nd_j'$ and $\epsilon \subset w_i/(\Sigma_i \cap \Sigma_j) \subset w_j/(\Sigma_i \cap \Sigma_j)$ then **fail**
2. if for some node nd_i and nd_i' both in St_i (i.e. $i \in 1..n$), $nd_i \overset{w}{\underset{r}{\Rightarrow}} nd_i'$ for some r, w with $elems(r) \cap St_i = \emptyset \wedge r[k] = r[k']$ (where $r[k]$ and $r[k]$ are the k^{th} and k'^{th} elements of r) for $k \neq k'$ then **fail**
3. if $(nd_1, ..., nd_n)$ is a node of the current automaton, if $(w_1, ..., w_n)$ is feasible for $(\Sigma_1, ..., \Sigma_n)$ where there exist r_i and w_i such that $nd_i \overset{w_i}{\underset{r_i}{\Rightarrow}} nd_i'$ and $elems(r_i) \cap St_i = \emptyset$ for each $i \in 1..n$ and if $(w_1, ..., w_n)$ is minimal (i.e. there

is no other candidate $(w'_1, ..., w'_n)$ such that $\epsilon \subset (w'_1, ..., w'_n) \subset (w_1, ..., w_n))$, then add $(nd_1, ..., nd_n) \overset{((w_1,...,w_n),G)}{\rightarrow} (nd'_1, ..., nd'_n)$ where $G = \{(i,q) | w_i \neq \epsilon \text{ and } \exists F_j \in \mathcal{F}_i \,.\, q \in ((elems(r_i) \cup \{nd'_i\}) \cap F_j)\}$, and add $((w_1, ..., w_n), G)$ to each $G_{(i,j)}$ such that $(i,q) \in G$ and $F_j \in \mathcal{F}_i$ and $q \in F_j$.

The function h is defined to be such that $h(((w_1, ..., w_n), G)) = (w_1, ..., w_n)$.

Theorem 19. *The algorithm of definition 18 always terminates. It terminates either by failing or either by producing a partial order automaton $\langle A, h \rangle$ such that:*

$$L(\langle A, h \rangle) = feasibleset_{(\Sigma_1,...,\Sigma_n)}(L(A_1), ..., L(A_n))$$

Proof. In a first step we prove that any tuple element of $L(\langle A, h \rangle)$ is in $feasibleset_{(\Sigma_1,...,\Sigma_n)}(L(A_1), ..., L(A_n))$. To any such element t we can associate by definition $w \in L(A)$ such that $h^{\omega}_{\odot}(w) = t$. For each $i \in 1..n$, we can then easily associate an infinite run of A_i. The fact that the run is an acceptable one directly comes from the fact that w is also an acceptable run (i.e. intersects infinitely often each set of $\mathcal{F}_i$). The infinite tuple is in $feasibleset_{(\Sigma_1,...,\Sigma_n)}(L(A_1), ..., L(A_n))$ since we wcan prove that for each $i, j \in 1..n$, $w_i/(\Sigma_i \cap \Sigma_j) = w_j/(\Sigma_i \cap \Sigma_j)$ by returning to the definition of the projection for ω-words.

We then prove that any tuple element t of $feasibleset_{(\Sigma_1,...,\Sigma_n)}(L_1, ..., L_n)$ is in $L(\langle A, h \rangle)$. For this we consider each component t_i of this feasible tuple and associate an accepting run for the corresponding automata A_i. These n runs are than used to build a strictly increasing chain whose limit is t and that correspond to a strictly increasing chain of w an element of $L(A)$. Building a strictly increasing chain of prefixes of t is quite easy and mimics the construction of the partial order automata. But even with the constraint imposed by the n runs the construction is not necessarily deterministic and the only difficulty is to build a chain that is converging to t. For this we select each time there is more than one possibilities, the one (or one of those if more than one) that allows to increase the smallest of the n prefixes of the n runs (one of the smallest if more than one). It is then quite easy to show by contradiction that in the case of our particular algorithm this guaranties proper convergence: the corresponding series is lower and upper bounded by two series converging to the same element.

Corollary 20. *Given any tuple $(L_1, ..., L_n)$ of languages with signature $(\Sigma_1, ..., \Sigma_n)$, there exists a partial automaton $\langle A, h \rangle$ such that $L(\langle A, h \rangle) = feasibleset_{(\Sigma_1,...,\Sigma_n)}(L_1, ..., L_n)$.*

Proof. Take any tuple $(A_1, ..., A_n)$ of automaton such that $L(A_i) = L_i$. Apply the algorithm of definition 18 using for each St_i of $(St_1, ..., St_n)$ the set of states of automaton A_i. It is then straightforward to check that the algorithm always succeed, and that the construction is isomorphic to a standard construction for the parallel composition.

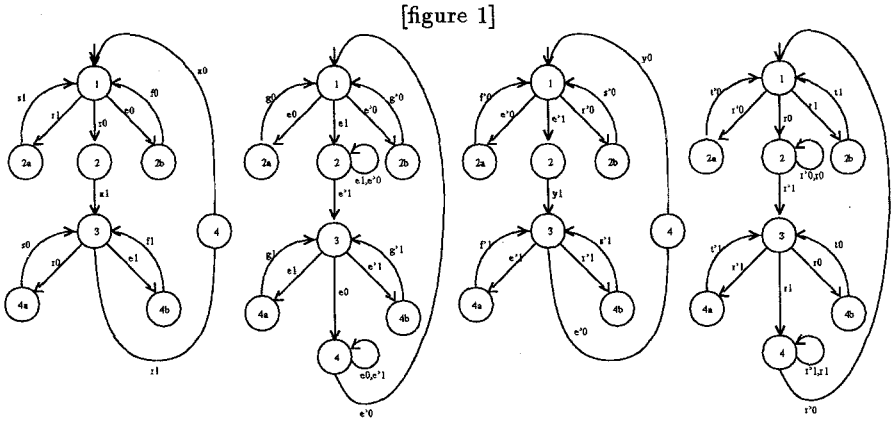
The two theorems thus provide a method for checking properties. On one hand the property is expressed as a Büchi automaton. On the other hand we

build the partial order automaton, corresponding to the system at hand and then compute an automaton from it using the algorithm in definition 16. Finally we compare this latter automaton with the automaton describing the property.

6 Illustration

The resulting reduction is in many cases very high as illustrated in [2], but as it is the case for other approaches, it is possible to identify situations, mainly artificial ones, in which no significant reduction can be obtained using the approach. The most significant benefits can be obtained on large and complex systems. Here we illustrate the technique and its benefits on a very simple example inspired from the alternated bit protocol, and than try to explain how the benefits would become bigger on more complex versions of the same example. Consider the four following Büchi automata of figure 1 where the families of acceptance sets for the first and third automaton are both equal to $\{\{2, 4, 2_a, 2_b\}, \{2, 4, 4_a, 4_b\}\}$, where the families for the second and fourth are both equal to $\{\{1, 3, 2_a, 2_b\}, \{1, 3, 4_a, 4_b\}\}$.

[figure 1]

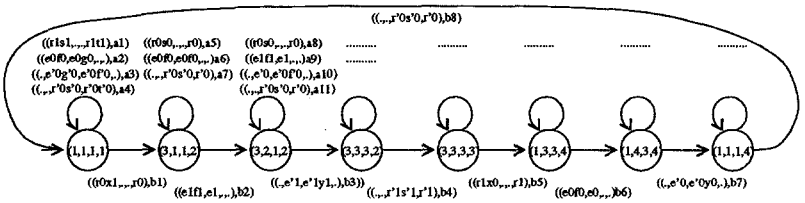


Let us now consider the parallel composition of the four processes modelled by these automata, where the respective alphabets are taken to be the symbols appearing in each automaton. We than build the partial automaton by providing a tuple of states $(St_1, ..., St_n)$ to the algorithm of definition 18. The choice of the tuple $(St_1, ..., St_n)$ is an important issue. The idea is to use sets that are as small as possible, so as to reduce the size of the partial automaton. The tuple chosen for the previous proof was clearly not a good choice to this respect: it would not bring any reduction in the size of the automaton to visit; it would just allow for the possibility of checking partial order based properties, which is an aspect that we do not develop here. Although there is no formal criteria for choosing an appropriate tuple, selecting one state in each loop[3] turns out to be a simple but still

[3] Loops of an associated regular expression for example.

a good strategy. Here using this simple strategy we would provide the following tuple $(\{1,3\},\{1,2,3,4\},\{1,3\},\{1,2,3,4\})$. The algorithm succeeds and builds the partial order automaton of figure 2 where a dot stands for the empty string ϵ, $a_1 = \{(1,2_a),(4,2_b),(4,1)\}, a_2 = \{(1,2_b),(2,2_a),(2,1)\}, ..., b_1 = \{(1,2)\}, b_2 = \{(1,4_b)\}, ...$ and where the family of $\mathcal{G}$ acceptance sets would have eight elements $G_{1,1}, G_{1,2}, G_{2,1}, ..., G_{4,2}$, and with for example $((r_1 s_1, \epsilon, \epsilon, r_1 t_1), a_1)$ being in $G_{1,1}$, $G_{4,1}$ and $G_{4,2}$, $((e_0 f_0, e_0 g_0, \epsilon, \epsilon), a_2)$ being in $G_{1,1}$, $G_{2,1}$ and $G_{2,2}$, $((r_0 x_1, \epsilon, \epsilon, r_0), b_1)$ being in $G_{1,1}$ and $G_{1,2}$, $((e_1 f_1, e_1, \epsilon, \epsilon), b_2)$ being in $G_{1,2}$ only, etc ...

[figure 2]

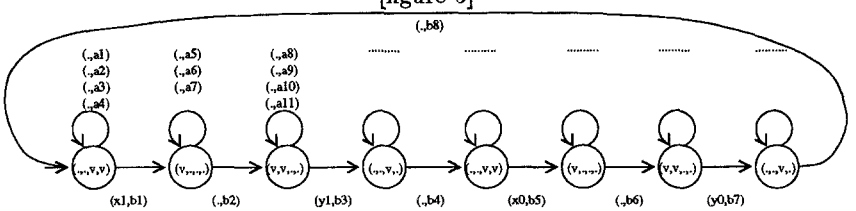


Let us now assume that we are interested in events that are instances of x_0, x_1, y_0, y_1, then the application of the algorithm of definition 16 would succeed and build the partial order automaton of figure 3, where the number of states and the size of the family of acceptance sets is the same as in the previous automata, where the state membership is isomorphic to that of the previous automata, where for more conciseness only the second component of each state is used and where a dot stands for ϵ. The number of states of the partial automata is here a few times smaller than the size of the global automaton that would correspond to standard parallel composition, and the abstraction produces an automaton describing the total ordering of abstracted events in performing a linear wandering through the nodes of each tuple. The produced automaton can easily be shown equivalent to the Büchi automaton[4] that recognizes the language with only one element: the ω-word that repeats $x_1 y_1 x_0 y_0$ for ever (i.e. $(x_1 y_1 x_0 y_0)^\omega$). It is indeed easy to check that this is the only that intersects each acceptance set of the family infinitely many times. Of course many other interesting abstractions could be investigated (e.g. abstraction focussing on $s_1, f_0, ...$) in the same way.

Let us now consider versions of the same protocol in which f_0, g_0 are considered as actions and each is replaced by a sequence of m finer actions. Then the number of states of the global automata will increase as the square root of m and up to the power 4 of m if we replace more actions in this way. At the same time the number of states of the corresponding partial automata will not *change* and the number of states that the algorithm of definition 16 has to wander through will only increase linearly with m, even if the projection includes elements of the first action or of the second.

[4] The transformation into a generalized Büchi automaton could in fact be done automatically.

[figure 3]



7 Conclusion

We have introduced and used feasible tuples to model the partial ordering of events in the case of the parallel composition of n sequential processes. We have seen how to check order-based properties directly on these structures by propagating information along the partial order graph and without having in particular to consider or generate the various corresponding linearizations. We have introduced special automata that we call partial order automata which allow to generate all partial ordering resulting from all possible executions of the parallel composition of n processes, and have shown how to exploit these automata to check order based liveness properties. We have seen that the language generated by partial order automata was more discriminating then interleaving semantics and that such automata could be used to check partial order based properties, such as properties involving true concurrency. This has not been exploited here and will be the ground for future research. We have finally shown how to build partial order automata from the description of the n processes. For the sake of conciseness we have used here models based on ω-words without considering the finite words that would allow to model the potential deadlocks resulting from composition. This extension is quite straightforward. Of course the exploitation of the partial order automata that we have introduced here is only meant to be an illustration of their potential use in verification. Various other possibilities remain to be investigated.

The presentation has focussed on the main properties of the approach. Many improvements can be made to the various algorithms, in particular by performing on-the-fly analysis: as an example the approach described was suggesting here to first build the partial order automaton, then deriving an automaton from it using the algorithm in definition 16 and finally comparing this latter automaton with the automaton describing the property at hand. None of these intermediate outputs need be generated: the various transformations can be done on-the-fly in a very straightforward manner.

As compared to other approaches that exploit partial order, the proposed technique has the disadvantage of requiring a human expertise for the identification of a tuple of states to provide to the algorithm of definition 18. But the selection of an adequate set of states is in practice very easy as illustrated in the previous section. On the other hand our technique presents significant advantages: first we directly generate and explore the partial order graph instead of using equivalence classes that would depend on (and be limited by) the property

to check; second the direct modeling of partial order allows for very promising possibilities in the expression and verification of partial order based properties.

References

1. Schneider F.B. Alpern B. Recognizing safety and liveness. *Distributed Computing*, 1987.
2. D. Bolignano. An approach to the verification of concurrent systems. Technical report, Bull Research, December 1994.
3. A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching semantics. In Springer Verlag, editor, *LNCS, 12th Colloquium on Automata, Languages, and Programming*, 1991.
4. S. Eilenberg. *Automata, Languages, and Machines (Vol. A)*. Academic Press, 1974.
5. P. Godefroid. Using partial orders to improve automatic verification methods. In *Proc. 1990 Computer-Aided Verification Workshop*, June 1990.
6. P. Godefroid and P. Wolper. A partial approach to model checking. In *Proc. 6th IEEE Symp on Logic in Computer Science*, July 1991.
7. Katz and Peled. Verification of distributed programs using representative interleaving sequences. *Distributed Computing*, 6, 1992.
8. A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationship to Other Models of Concurrency, Advances in Petri Nets 1986, PART II, Advanced Course*, pages 279–324, Bad Honnefs, September 1986. Berlin, West Germany: Springer–Verlag. LNCS–255.
9. K.l. McMillan. Using unfoldings to avoid state explosion problem in the verification of asynchronous circuits. In *Proceedings of the 4th International Conference, CAV'92*, 1992.
10. D. Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th International Conference, CAV'93*, 1993.
11. A. Valmari. Stubborn sets forreduced state space generation, 10th international conference on application and theory of petri nets. In *Vol 2*, 1989.
12. A. Valmari. On-the-fly verification with stubborn sets. In *Proceedings of the 5th International Conference, CAV'93*, 1993.