Abstracting WS1S Systems to Verify Parameterized Networks*

Kai Baukus[1]**, Saddek Bensalem[2], Yassine Lakhnech[2]**, and Karsten Stahl[1]

[1] Institute of Computer Science and Applied Mathematics
University of Kiel
Preusserstr. 1–9, D-24105 Kiel, Germany.
{kba, kst}@informatik.uni-kiel.de
[2] VERIMAG, Centre Equation
2 Av. de Vignate, 38610 Gières, France.
{bensalem, lakhnech}@imag.fr

Abstract. We present a method that allows to verify parameterized networks of finite state processes. Our method is based on three main ideas. The first one consists in modeling an infinite family of networks by a single WS1S transition system, that is, a transition system whose variables are set (2nd-order) variables and whose transitions are described in WS1S. Then, we present methods that allow to abstract a WS1S system into a finite state system that can be model-checked. Finally, in order to verify liveness properties, we present an algorithm that allows to enrich the abstract system with strong fairness conditions while preserving safety of the abstraction. We implemented our method in a tool, called PAX, and applied it to several examples.

1 Introduction

Recently there has been much interest in the automatic and semi-automatic verification of parameterized networks, i.e., verification of a family of systems $\{\mathcal{P}_i \mid i \in \omega\}$, where each $\mathcal{P}_i$ is a network consisting of i finite-state processes. Apt and Kozen show in [AK86] that the verification of parameterized networks is undecidable. Nevertheless, automated and semi-automated methods for the verification of restricted classes of parameterized networks have been developed. The methods presented in [GS92,EN95,EN96] show that for classes of ring networks of arbitrary size and client-server systems, there exists k such that the verification of the parameterized network can be reduced to the verification of networks of size up to k. Alternative methods presented in [KM89,WL89,BCG89,SG89,HLR92,LHR97] are based on induction on the number of processes. These methods require finding a network invariant that abstracts any arbitrary number of processes with respect to a pre-order that

* This work has been partially supported by the Esprit-LTR project Vires.
** Contact Author.

preserves the property to be verified. While this method has been originally presented for linear networks, it has been generalized in [CGJ95] to networks generated by context-free grammars. In [CGJ95], abstract transition systems were used to specify the invariant. An abstract transition system consists of abstract states specified by regular expressions and transitions between abstract states. The idea of representing sets of states of parameterized networks by regular languages is applied in [KMM$^+$97], where additionally finite-state transducers are used to compute predecessors. These ideas are applied to linear networks as well as to processes arranged in a tree architecture and semi-automatic symbolic backward analysis methods for solving the reachability problem are given. The work presented in [ABJN99] extends the ideas in [KMM$^+$97] by considering the effect of applying infinitely often a transition that satisfies certain restrictions. These restrictions allow to characterize the effect of the repeated application of the transition by finite-state transducers. Moreover, the method presented in [ABJN99] allows to consider networks of processes guarded by both local and global conditions that restrict the context in which a transition can be taken. Global conditions are typically used in many mutual exclusion algorithms such as Szymanski's algorithm, the bakery and ticket algorithms by Lamport and Dijkstra's algorithm.

In this paper we present a method for the verification of parameterized networks that can deal with a larger class of networks than the methods presented in [KMM$^+$97,ABJN99] and that is applicable to verify *communal liveness* (also referred to as *weak liveness*) properties [MP94].

To verify parameterized protocols we first transform a given infinite family of networks of finite processes into a bisimilar single transition system whose variables are set variables and whose transitions are described in WS1S, the weak monadic second order logic of one successor. We call such systems WS1S transition systems. Then, we abstract the obtained WS1S transition system into a finite abstract system that can be analyzed using model-checking techniques. To obtain such a finite abstraction, one needs to come up with an appropriate abstraction relation and to construct a *correct* abstract system, i.e., a system which exhibits for every behavior of the WS1S system a corresponding abstract behavior. We present a method to construct an abstraction relation from the given WS1S transition system and three techniques that allow to *automatically* construct either a correct abstract system, the reachability state graph of a correct abstract system without constructing the system itself or, finally, an abstraction of such a graph. Our experience shows that our method for constructing an abstraction relation is useful for many examples and that it is useful to have the three techniques for constructing an abstract system.

It is well known that an obstacle to the verification of liveness properties using abstraction, e.g. [CGL94,LGS$^+$95,DGG94], is that often the abstract system contains cycles that do not correspond to paths in the concrete system. This is not surprising since the main goal of an abstraction relation is to identify and merge states together, which consequently introduces new cycles. A way to overcome this difficulty is to enrich the abstract system with fairness conditions or more generally ranking functions over well-founded sets that eliminate undesirable cycles, that is, cycles that do not correspond to concrete computations. The main problem is, however, to find such fairness conditions. To tackle this problem, we present an algorithm that given a reachability state graph of an abstraction of a WS1S system enriches the graph with strong fairness conditions while preserving the property that to each concrete computation corresponds an abstract *fair* one. Hence, we can use the enriched graph to prove liveness properties of the WS1S systems, and consequently, of the parameterized network.

We implemented our method in a tool, we call PAX, that uses the decision procedures of MONA [KM98,HJJ+96] to check the satisfiability of WS1S formulas. We then applied our method to several examples including Szymanski's mutual exclusion algorithm, a server-ring as well as a token passing protocol. The first results obtained using our method and PAX are very encouraging.

2 Preliminaries

In this, section, we briefly recall the definition of weak second order theory of one successor (WS1S for short) [Büc60,Tho90] and introduce the logic we use to describe the class of parameterized systems we are interested in.

Terms of WS1S are built up from the constant 0 and 1st-order variables by applying the successor function $succ(t)$ ("$t+1$"). *Atomic formulae* are of the form b, $t = t'$, $t < t'$, $t \in X$, where b is a boolean variable, t and t' are terms, and X is a set variable (2nd-order variable). WS1S-formulae are built up from atomic formulae by applying the boolean connectives as well as quantification over both 1st-order and 2nd-order variables. *First-order monadic formulae* are WS1S-formulae in which no 2nd-order variables occur.

WS1S-formulae are interpreted in models that assign finite subsets of ω to 2nd-order variables and elements of ω to 1st-order variables. The interpretation is defined in the usual way.

Given a WS1S formula f, we denote by $[\![f]\!]$ the set of models of f. The set of free variables in f is denoted by $\mathit{free}(f)$. We say that f is (1st-order) *closed*, if it does not contain (1st-order) free variables. In addition to the usual abbreviations, given a 2nd-order variable P, we write $\forall_P\, i : f$ instead of $\forall i : i \in P \rightarrow f$ and $\exists_P\, i : f$ instead of $\exists i : i \in P \wedge f$.

Finally, we recall that by Büchi [Büc60] and Elgot [Elg61] the satisfiability problem for WS1S is decidable. Indeed, the set of all models of a WS1S-formula is representable by a finite automaton (see, e.g., [Tho90]).

Let n be a variable. To define parameterized systems, we introduce the set $\mathrm{AF}(n)$ of formulae f defined by:

$$f \ ::= \ b[x] \mid \neg f \mid f \wedge f \mid \forall_n\, x : f \mid \exists_n\, x : f \ ,$$

where x is a *position variable*, b is a *boolean array variable*. Let $m \in \omega$. We denote by Σ_m the set of evaluations s such that $s(n) = m$, $s(x) \in \{0, \cdots, m-1\}$ and $s(b) : \{0, \cdots, m-1\} \rightarrow \{\text{true}, \text{false}\}$. Then, formulae in $\mathrm{AF}(n)$ are interpreted over evaluations $s \in \bigcup_{m \in \omega} \Sigma_m$ in the usual way. In the sequel, we also assume the usual notion of free variables, closed formulae, etc., as known.

3 Parameterized Systems

We now introduce a class of parameterized systems for which we develop an abstraction-based verification technique.

Definition 1 (Boolean Transition Systems). *A* boolean transition system *$S(i,n)$, parameterized by n and i, where n and i are variables ranging over natural numbers, is described by the triple $(V, \Theta, \mathcal{T})$, where*

- *$V = \{b_1, \ldots, b_k\}$ and each b_j, $1 \leq j \leq k$, is a boolean array of length n.*
- *Θ is a formula in $\mathrm{AF}(n)$ with $free(\Theta) \subseteq V \cup \{i\}$ and which describes the set of initial states.*
- *$\mathcal{T}$ is a finite set of transitions where each $\tau \in \mathcal{T}$ is given by a formula $\rho_\tau \in \mathrm{AF}(n)$ such that $free(\rho_\tau) \subseteq V \cup V' \cup \{i\}$ and V' contains a primed copy for each variable in V.* □

We denote by $\mathcal{P}_m$ the parallel composition $\|_{l=0}^{m-1} S(l,m)$, where $S(l,m)$ is obtained from $S(i,n)$ by substituting m for n and l for i and where $\|$ denotes the interleaving-based parallel composition (asynchronous product). Notice, that if we identify the boolean array variables b_j by the m boolean variables $b_j[1], \cdots, b_j[m]$, the formulae describing the initial states as well as the transitions of $\|_{l=0}^{m-1} S(l,m)$ are first-order WS1S formulae whose free variables are in $\mathcal{V}$, respectively, $\mathcal{V} \cup \mathcal{V}'$. Thus, $\mathcal{P}_m$ is a transition system in the usual sense, i.e., it does not contain the parameters n and i. Hence, we assume the definition of a computation of $\mathcal{P}_m$ as known and we denote by $[\![\mathcal{P}_m]\!]$ the set of its computations. Then, a *monadic parameterized system* $\mathcal{P}$ (MPS for short) built from $S(i,n)$ is the set $\{\mathcal{P}_m \mid m \geq 1\}$.

To illustrate the above definitions we consider Szymanski's mutual exclusion algorithm [Szy88] as a boolean transition system.

Example 1 (Szymanski's mutual exclusion algorithm). Consider the following version of Szymanski's mutual exclusion algorithm (cf. [ABJN99]), where each process $S(i,n)$ is described as follows:

ℓ_1: **await** $\forall_n j : \mathrm{at\_}\ell_1[j] \vee \mathrm{at\_}\ell_2[j] \vee \mathrm{at\_}\ell_4[j]$
ℓ_2: **skip**
ℓ_3: **if** $\exists_n j : \mathrm{at\_}\ell_2[j] \vee \mathrm{at\_}\ell_5[j] \vee \mathrm{at\_}\ell_6[j] \vee \mathrm{at\_}\ell_7[j]$
  then goto ℓ_4
  else goto ℓ_5
ℓ_4: **await** $\exists_n j : \mathrm{at\_}\ell_5[j] \vee \mathrm{at\_}\ell_6[j] \vee \mathrm{at\_}\ell_7[j]$
ℓ_5: **await** $\forall_n j : \mathrm{at\_}\ell_1[j] \vee \mathrm{at\_}\ell_2[j] \vee \mathrm{at\_}\ell_5[j] \vee \mathrm{at\_}\ell_6[j] \vee \mathrm{at\_}\ell_7[j]$
ℓ_6: **await** $\forall_n j : j < i \rightarrow (\mathrm{at\_}\ell_1[j] \vee \mathrm{at\_}\ell_2[j] \vee \mathrm{at\_}\ell_4[j])$
ℓ_7: $\langle$ *critical section* $\rangle$; goto ℓ_1

For the sake of presentation, the example is given in the style of [MP95]. The **await** statements express the guards for the transitions leading from one control location to the next one, i.e., the processes have to wait until the guard becomes true. In our formal model the control locations are modeled by a boolean array variable at_ℓ_k for each location ℓ_k, $1 \leq k \leq 7$. According to Definition 3 the transition from ℓ_1 to ℓ_2 is given by the AF(n) formula:

$$(\forall_n j : \text{at\_}\ell_1[j] \vee \text{at\_}\ell_2[j] \vee \text{at\_}\ell_4[j]) \wedge \forall_n j : j \neq i \rightarrow \bigwedge_{l=1}^{7} \text{at\_}\ell_l[j] \leftrightarrow \text{at\_}\ell'_l[j]$$
$$\wedge \text{at\_}\ell_1[i] \wedge \neg\text{at\_}\ell'_1[i] \wedge \text{at\_}\ell'_2[i] \wedge \bigwedge_{l=3}^{7} \text{at\_}\ell_l[i] \leftrightarrow \text{at\_}\ell'_l[i] \enspace .$$

The initial condition states that each process starts in ℓ_1.

Our aim is to prove that this algorithm satisfies the mutual exclusion property, which can be expressed by $\Box\neg\, \exists_n\, i, j : i \neq j \wedge \text{at\_}\ell_7[i] \wedge \text{at\_}\ell_7[j]$. $\diamond$

4 WS1S Transition Systems

In this section, we introduce WS1S transition systems which are transition systems with variables ranging over finite sub-sets of ω and show how they can be used to represent infinite families of boolean transition systems.

Definition 2 (WS1S Transition Systems). *A* WS1S transition system $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$ *is given by the following components:*

- $\mathcal{V} = \{X_1, \ldots, X_k\}$*: A finite set of second order variables where each variable is interpreted as a finite set of natural numbers.*
- Θ*: A WS1S formula with free*$(\Theta) \subseteq \mathcal{V}$ *describing the initial condition of the system.*
- $\mathcal{T}$*: A finite set of transitions where each* $\tau \in \mathcal{T}$ *is represented as a WS1S formula* $\rho_\tau(\mathcal{V}, \mathcal{V}')$*, i.e., free*$(\rho_\tau) \subseteq \mathcal{V} \cup \mathcal{V}'$*.* $\Box$

The computations of $\mathcal{S}$ are defined as usual. Moreover, let $[\![\mathcal{S}]\!]$ denote the set of computations of $\mathcal{S}$.

Relating parameterized and WS1S transition systems. We define a translation that maps an MPS to a bisimilar WS1S system.

We fix a 2nd-order variable P that is used to model the set of indices of the processes up to n. The translation from MPS to WS1S systems uses a function *tr* from AF(n) into WS1S. This function replaces in an AF(n)-formula all occurrences of atomic sub-formulae of the form $b[i]$ by $i \in B$, all n by $\max(P) + 1$[1], and λ_n by λ_P where λ is one of the quantifiers $\forall$ or $\exists$.

Definition 3 (Translation of Boolean to WS1S Systems). *Consider an MPS system* $\mathcal{P}$ *built from* $S(i,n)$ *where* $S(i,n) = (V, \Theta, \mathcal{T})$*. Define a WS1S system* $(\widetilde{V}, \widetilde{\Theta}, \widetilde{\mathcal{T}})$ *by constructing the variable set, the initial condition, and the transitions as follows:*

- *For each boolean array* b_k *in* V*,* $\widetilde{V}$ *contains the variable* B_k*. Additionally,* $\widetilde{V}$ *contains the set variable* P*.*
- *Let* $\widetilde{\Theta}$ *be* $\exists n : P = \{0, \ldots, n-1\} \wedge \bigcup_{l=1}^{k} B_l \subseteq P \wedge (\forall_P\, i : tr(\Theta))$*.*
- *Let* $\widetilde{\mathcal{T}}$ *be the set* $\{\exists_P\, i : tr(\rho_\tau) \wedge P = P' \wedge \bigcup_{l=1}^{k} B'_l \subseteq P' \mid \tau \in \mathcal{T}\}$*.*

[1] It is not difficult to check that $\max(P)$ can be expressed in WS1S.

We denote the above transformation of an MPS system $\mathcal{P}$ by $Tr(\mathcal{P})$. □

Example 2 (Szymanski cont'd). The translation of Example 1 into a WS1S system introduces a set variable P and set variables $\mathrm{At\_}\ell_1, \ldots, \mathrm{At\_}\ell_7$. According to the above definition of the translation we have as initial condition $\widetilde{\Theta}$

$$\exists n : P = \{0, \ldots, n-1\} \wedge \bigcup_{l=1}^{k} B_l \subseteq P \wedge \forall_P\, i : (i \in \mathrm{At\_}\ell_1 \wedge \bigwedge_{l=2}^{7} i \notin \mathrm{At\_}\ell_l) \ .$$

The translation of the $\mathrm{AF}(n)$ formula characterizing the transition from ℓ_1 to ℓ_2 presented in Example 1 yields

$$\begin{array}{l}(\forall_P\, j : j \in \mathrm{At\_}\ell_1 \cup \mathrm{At\_}\ell_2 \cup \mathrm{At\_}\ell_4) \wedge \forall_P\, j : j \neq i \rightarrow \bigwedge_{l=1}^{7} j \in \mathrm{At\_}\ell_l \leftrightarrow j \in \mathrm{At\_}\ell'_l \\ \quad \wedge\, i \in \mathrm{At\_}\ell_1 \wedge i \notin \mathrm{At\_}\ell'_1 \wedge i \in \mathrm{At\_}\ell'_2 \wedge \bigwedge_{l=3}^{7} i \in \mathrm{At\_}\ell_l \leftrightarrow i \in \mathrm{At\_}\ell'_l\end{array}$$

which, using the invariant $\bigcup_{l=1}^{k} B_l \subseteq P$, can be simplified to

$$\begin{array}{l}(\forall_P\, j : j \in \mathrm{At\_}\ell_1 \cup \mathrm{At\_}\ell_2 \cup \mathrm{At\_}\ell_4) \wedge \bigwedge_{l=3}^{7} \mathrm{At\_}\ell_l = \mathrm{At\_}\ell'_l \\ \quad \wedge\, \mathrm{At\_}\ell'_1 = \mathrm{At\_}\ell_1 \setminus \{i\} \wedge \mathrm{At\_}\ell'_2 = \mathrm{At\_}\ell_2 \cup \{i\} \ .\end{array}$$

◇

In order to state the relationship between an MPS $\mathcal{P}$ and its translation, we introduce a function h relating the states of both systems. Let $\mathcal{P}$ be an MPS built from $S(i,n) = (V, \Theta, \mathcal{T})$ with $V = \{b_1, \ldots, b_k\}$. Let m be a natural number. Let $\widetilde{\Sigma}$ denote the set of interpretations $\tilde{s}$ of $\widetilde{V}$ such that $\tilde{s}(X) \subseteq \tilde{s}(P)$, for each $X \in \widetilde{V}$. Then, define $h_m : \Sigma_m \rightarrow \widetilde{\Sigma}, s \mapsto \tilde{s}$ by $\tilde{s}(P) = \{0, \ldots, m-1\}$ and $\tilde{s}(B_j) = \{l < m \mid s(b_j[l]) = \mathrm{true}\}$, for every $1 \leq j \leq k$. Then, we define $h = \bigcup_{m \in \omega} h_m$. Notice that h is a bijection.

The following lemma shows that h is consistent with the translation *tr* from $\mathrm{AF}(n)$ into WS1S.

Lemma 1. *Let f be a formula in $\mathrm{AF}(n)$ with $free(f) \subseteq V \cup V' \cup \{i\}$ and let h be the function introduced above. Then, for all $m \in \omega$, for all $s, s' \in \Sigma_m$, we have:*

$$(s, s') \models f \quad \textit{iff} \quad (h(s), h(s')) \models tr(f) \ .$$

□

Using this lemma we can prove the following theorem that justifies our verification method given in Section 5. The theorem states that $\mathcal{P}_m$ is bisimilar to $Tr(\mathcal{P})$ when we initialize P to $\{0, \ldots, m-1\}$.

Theorem 1 (Relating Boolean and WS1S Systems). *Let $\mathcal{P}$ be an MPS built from $S(i,n)$ and $m \in \omega$. Then, h is a bisimulation between $\mathcal{P}_m$ and $Tr^*(\mathcal{P})$, where $Tr^*(\mathcal{P})$ is obtained from $Tr(\mathcal{P})$ by taking as initial condition $\widetilde{\Theta} \equiv P = \{0, \ldots, m-1\} \wedge \bigcup_{l=1}^{k} B_l \subseteq P \wedge \forall_P\, i : tr(\Theta)$.*

Proof: Consider s_0 to be an initial state of $\mathcal{P}_m$, i.e., $s_0 \models \Theta$. With Lemma 1 it follows that

$$h(s_0) \models P = \{0, \ldots, m-1\} \wedge \bigcup_{l=1}^{k} B_l \subseteq P \wedge (\forall_P\, i : tr(\Theta)) \ .$$

Vice versa, for any initial state $\tilde{s}_0$ of some computation in $[\![Tr^*(\mathcal{P})]\!]$ the state $h^{-1}(\tilde{s}_0)$ is an initial state of $\mathcal{P}$.

With the same argumentation we can show for $s, s' \in \Sigma_m$ and $\tilde{s}, \tilde{s}' \in \widetilde{\Sigma}$ with $\tilde{s} = h(s)$ that for any $\tau \in \mathcal{T}$,

- if s' is a τ-successor of s then $h(s')$ is a $tr(\rho_\tau)$-successor of $\tilde{s}$, and
- if $\tilde{s}'$ is an $tr(\rho_\tau)$-successor of $\tilde{s}$ then $h^{-1}(\tilde{s}')$ is a τ-successor of s.

Hence, both systems are bisimilar. □

Using Theorem 1, we can prove the following:

Corollary 1. *Let $\mathcal{P}$ be an MPS built from $S(i, n)$. Then, lifting h to computations, we have a bijection between $\bigcup_{m\in\omega} [\![\mathcal{P}_m]\!]$ and $[\![Tr(\mathcal{P})]\!]$.* □

5 Abstracting WS1S Systems

In Section 4, we have shown how parameterized boolean transition systems can be translated into WS1S systems. This translation allows us to consider a single, though infinite-state, transition system instead of an infinite family of systems. In the following, we present a method to construct finite abstractions of WS1S systems.

To do so, we first present a heuristic that allows to construct for a given WS1S system an abstraction function such that the corresponding abstract system has a finite state space. Then, we present a method that, given such a function, constructs a finite transition system that is an abstraction of the given WS1S system. Model-checking techniques can then be used to construct a state graph of this abstract system.

The nodes of the constructed graph represent sets of abstract states and the edges correspond to abstract transitions. This graph contains all reachable abstract states. The nodes of the finest graph that can be constructed represent singleton sets, i.e., single abstract states. The coarsest graph has one node corresponding to the set of reachable states. In fact, the granularity of the computed graph depends on the techniques used during exploration.

While the previous method first constructs an abstract system from which a graph is computed, we also present two methods inspired by [GS97] for computing an abstract state graph, resp. an abstraction of it, without computing the abstract transition system at all. These methods are useful in case the abstract system is not computable for size reasons.

Finally, we describe our PAX tool which implements these techniques using MONA [KM98,HJJ+96].

We first recall some definitions and the idea of proving properties of systems by abstraction.

Given a transition system $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$ and a total abstraction relation $\alpha \subseteq \Sigma \times \Sigma_A$, we say that $\mathcal{S}_A = (\mathcal{V}_A, \Theta_A, \mathcal{T}_A)$ is an *abstraction* of $\mathcal{S}$ w.r.t. α, denoted by $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}_A$, if the following conditions are satisfied:

- $s_0 \models \Theta$ implies $\alpha(s_0) \models \Theta_A$
- $\tau \circ \alpha^{-1} \subseteq \alpha^{-1} \circ \tau_A$.

Let $\Box\varphi, \Box\varphi_A$ be invariance formulae, i.e., φ, φ_A are state formulae. Then, from $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}_A$, $\alpha^{-1}([\![\varphi_A]\!]) \subseteq [\![\varphi]\!]$, and $\mathcal{S}_A \models \Box\varphi_A$ we can conclude $\mathcal{S} \models \Box\varphi$. This statement, which is called preservation result, shows the interest of verification by abstraction: since if $\mathcal{S}_A$ is finite, it can automatically be checked whether $\mathcal{S}_A \models \Box\varphi_A$. In fact a similar preservation result holds for any temporal logic without existential quantification over paths, e.g., $\forall CTL^\star$, LTL, or $\mu_\Box$ [CGL94,DGG94,LGS+95].

5.1 Constructing the abstraction function

Our heuristic to construct an abstraction function for WS1S systems assumes the transitions to have the following form:

$$\exists_P\, i : G \wedge L(i) \wedge C(i) \wedge \mathcal{V}' = exp(\mathcal{V}, \mathcal{V}') \ ,$$

where $G, L(i), C(i)$ are WS1S formulae whose free variables are in $\mathcal{V}$ and such that:

- G is a 1st-order closed WS1S formula. Intuitively, G describes a global condition. E.g., in the Szymanski example (see Example 2), the presented transition contains the global condition $\forall_P\, j : j \in \mathrm{At\_}\ell_1 \cup \mathrm{At\_}\ell_2 \cup \mathrm{At\_}\ell_4$.
- $L(i)$ is a quantifier-free formula with i as the unique free 1st-order variable. Intuitively, if i models a process index then $L(i)$ is a condition on the local state of this process.
- $C(i)$ is a condition that as in the case of $L(i)$ has i as the unique free 1st-order variable but which contains 1st-order quantifiers. Intuitively, it imposes conditions on the context of process i.

Though, the above requirements restrict the set of considered WS1S systems, it still includes all translations of *MPS* systems.

Let $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$ be a WS1S system whose transitions satisfy the restriction above.

We are now prepared to present our heuristic for constructing abstraction functions. The set $\mathcal{V}_A$ of abstract variables contains a boolean variable b_X for each variable $X \in \mathcal{V}$. Moreover, for each global guard G, resp. local guard L occurring in a transition, it contains a boolean variable b_G, resp. b_L. Since the context guards $C(i)$ describe a dependence between process i and the remaining processes, it turns out to be useful to combine them with the local guards. Indeed, this allows to check, for instance, whether some dependence is propagated over some transitions. Therefore, we introduce boolean variables b_{L_k,C_l} for some boolean combinations of local guards and context guards. Additionally, $\mathcal{V}_A$ contains a boolean variable for each state formula appearing in the property to be verified.

It remains now to present how we relate the concrete and abstract states, i.e., to describe an abstraction relation α. The abstraction relation α can be expressed on the syntactic level by a predicate $\widehat{\alpha}$ over $\mathcal{V}, \mathcal{V}_A$ which is defined as the conjunction of the following equivalences:

$$\begin{aligned} b_X &\equiv X \neq \emptyset \\ b_G &\equiv G \\ b_L &\equiv \exists_P\, i : L(i) \\ b_{L_k,C_l} &\equiv \exists_P\, i : L_k(i) \wedge C_l(i) \\ b_\xi &\equiv \xi \end{aligned}$$

Henceforth, we use $\widehat{\alpha}(\mathcal{V}', \mathcal{V}'_A)$ to denote the predicate obtained from $\widehat{\alpha}$ by substituting the unprimed variables with their primed versions.

Example 3 (Szymanski cont'd). Applying the above heuristic on Szymanski's algorithm (see Example 2) we get seven boolean variables:

$$\psi_i \equiv \mathrm{At\_}\ell_i \neq \emptyset, \text{ for each } 1 \leq i \leq 7 \ .$$

The global guards not referring to i can be derived by the above variables and do not lead to a finer partitioning of the state space. All the local guards $i \in \mathrm{At\_}\ell_l$ would introduce a boolean variable with meaning $\exists i : i \in \mathrm{At\_}\ell_l$ which is equivalent to stating that $\mathrm{At\_}\ell_l$ is not empty. In the transition leading from ℓ_6 to ℓ_7 we have a context $\forall j < i : j \in \mathrm{At\_}\ell_1 \cup \mathrm{At\_}\ell_2 \cup \mathrm{At\_}\ell_4$ which we have to combine with the local guards $i \in \mathrm{At\_}\ell_l$. For this example it turns out to be enough to take only one combination, namely

$$\varphi \equiv \exists i : i \in \mathrm{At\_}\ell_7 \wedge \forall j < i : j \in \mathrm{At\_}\ell_1 \cup \mathrm{At\_}\ell_2 \cup \mathrm{At\_}\ell_4 \ .$$

Moreover, for the property of interest we introduce

$$\xi \equiv \neg \exists l, j : l \neq j \wedge l \in \mathrm{At\_}\ell_7 \wedge j \in \mathrm{At\_}\ell_7 \ .$$

$\diamond$

5.2 Constructing the abstract system

In Section 5.1, we presented a method that allows to construct an abstraction function from a given WS1S system. In this section, we show how to use this abstraction function to automatically construct a finite transition system that can be model-checked.

Let $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$ be a given WS1S system that satisfies the restriction given in Section 5.1 and let α be the abstraction function constructed by the method given in the same section. Notice that since the abstract variables are booleans, the abstract system we construct is finite, and hence, can be subject to model-checking techniques. Moreover, we make use of the fact that both $\widehat{\alpha}(\mathcal{V}, \mathcal{V}_A)$ and the transitions in $\mathcal{T}$ are expressed in WS1S to give an effective construction of the abstract system.

Henceforth, given a set γ of abstract states, we denote by $\widehat{\gamma}(\mathcal{V}_A)$ a WS1S formula that characterizes this set.

The abstract system we construct contains for each concrete transition τ an abstract transition τ_A, which is characterized by the formula

$$\exists \mathcal{V}, \mathcal{V}' : \widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \wedge \rho_\tau(\mathcal{V}, \mathcal{V}') \wedge \widehat{\alpha}(\mathcal{V}', \mathcal{V}'_A)$$

with free variables $\mathcal{V}_A$ and $\mathcal{V}'_A$.

The initial states of the abstract system we construct can be described by the formula

$$\exists \mathcal{V} : \widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \ .$$

To compute them, one has to find all states fulfilling this formula, which is possible since this is a WS1S formula.

5.3 Constructing abstract state graphs

We first define state graphs of transition systems. Note that there is a whole set of state graphs for a given transition system.

Definition 4 (State Graphs). *Let $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$ be a transition system and Σ be the set of all reachable states of $\mathcal{S}$. A* state graph *$\mathcal{G}$ of $\mathcal{S}$ is a tuple $(\mathcal{N}, \mathcal{E}, \mathcal{N}_0, \mu)$, where $\mathcal{N}$ is a set of nodes, $\mathcal{N}_0 \subseteq \mathcal{N}$ is the set of initial nodes, $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{T} \times \mathcal{N}$ is a set of labeled edges, and $\mu : \mathcal{N} \to 2^\Sigma$ is a labeling function, such that the following conditions are satisfied:*

1. $[\![\Theta]\!] \subseteq \bigcup_{q \in \mathcal{N}_0} \mu(q)$.
2. $\forall q \in \mathcal{N}, \tau \in \mathcal{T} : post_\tau(\mu(q)) \subseteq \bigcup_{(q,\tau,q') \in \mathcal{E}} \mu(q')$, *where $post_\tau(\mu(q))$ is the set of successors of $\mu(q)$ w.r.t. τ.* □

There are several strategies to calculate a state graph for an abstract system. First of all, if one has previously computed the abstract system as explained in Section 5.2, one can use model-checking techniques, for example a forward state exploration, to construct a state graph. The actual graph obtained depends on the computation techniques used.

Another way is to compute such a graph without calculating the abstract system at all. The nodes of the graph are sets of abstract states. One starts with an initial node representing the set of abstract initial states, which are computed as shown in Section 5.2. Assume that a concrete transition τ and node q are given. The formula

$$\exists \mathcal{V}_A, \mathcal{V}, \mathcal{V}' : \widehat{\mu(q)}(\mathcal{V}_A) \wedge \widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \wedge \rho_\tau(\mathcal{V}, \mathcal{V}') \wedge \widehat{\alpha}(\mathcal{V}', \mathcal{V}'_A)$$

describes the set M of abstract post states w.r.t. τ_A. Then, a node q' with $\mu(q') = M$ and an edge (q, τ_A, q') are added to the graph. This process is repeated until no new nodes or edges can be added. The so obtained graph is a state graph of $\mathcal{S}_A$, we call this calculation *precise computation*.

Since we restrict ourselves to abstraction functions of the form

$$\widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \equiv \bigwedge_{1 \leq i \leq k} a_i \leftrightarrow \varphi_i(\mathcal{V}) \ ,$$

where $\mathcal{V}_A = \{a_1, \ldots, a_k\}$, we can also eliminate the abstract variables $\mathcal{V}_A$ by replacing each abstract variable a_i by $\varphi_i(\mathcal{V})$.

The decision procedure for WS1S is based on constructing a finite automaton over finite words that recognizes the set of models of the considered formula. In practice, however, it can happen that the automaton cannot be constructed because of its size. In this case, we propose to go on as follows to obtain an abstract state graph of $\mathcal{S}_A$.

We start with the same initial node as before. Given a node q and concrete transition τ, we construct new nodes by computing for each abstract variable a_i the set $M_i \subseteq \{\text{true}, \text{false}\}$ of fulfilling values for a'_i of

$$\exists \mathcal{V}_A, \mathcal{V}, \mathcal{V}' : \widehat{\mu(q)}(\mathcal{V}_A) \wedge \widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \wedge \rho_\tau(\mathcal{V}, \mathcal{V}') \wedge (a'_i \leftrightarrow \varphi_i(\mathcal{V}')) \ .$$

Again, the abstract variables $\mathcal{V}_A$ can be eliminated. In case M_i is empty for some i, then there does not exist any post state, and hence, the computation for the other variables can be omitted. Otherwise, instead of taking a node representing the set of abstract post states, one takes a new node q' with $\mu(q') = \{s \mid \forall i : s(a_i) \in M_i\}$ and a new edge (q, τ_A, q'). The set $\mu(q')$ contains at least all possible abstract post states w.r.t. τ_A. It is not difficult to see that this method computes an abstraction of $\mathcal{S}_A$, and hence, is also an abstraction of $\mathcal{S}$.

5.4 The pax tool

We use MONA [KM98,HJJ+96] to decide the predicates mentioned above. In fact, MONA is able to construct all models of a WS1S predicate.

Our system PAX constructs state graphs. It uses MONA to compute the abstract initial states, the abstract transitions, or the set of abstract post states represented by a state graph node. To do so, it creates from its input files input for MONA and interprets the MONA results. Once an abstract system (resp. abstract state graph) is constructed, a translation to SMV, alternatively to SPIN, can be used to model-check the obtained abstract system (resp. abstract state graph).

PAX allows the combination of the method of Section 5.2, which consists in computing abstractions of concrete transitions independently of any source abstract state, with the methods of Section 5.3. This is helpful in case MONA does not succeed in computing an abstract transition because of memory limitations.

In Table 1 we give run time results for the calculation of the abstract systems of some examples, one of it is the Szymanski mutual exclusion algorithm given in Example 3. The construction of the state graph from the abstract system does not require mentionable time. We used a Sun Ultra Sparc 5/10 with 768 MB of memory and 333 MHz processor.

Example	Abstract Transitions System
Szymanski	7 min 28 sec
Server-ring	40 sec
Token passing	5 sec

Table 1. PAX construction of abstract transition systems

The run time results for the construction of a state graph (resp. an abstraction) without previous computation of the abstract system of these examples is summarized in Table 2.

Example	Precise Computation	Abstraction
Szymanski	49 min 52 sec	56 sec
Server-ring	1 min 5 sec	13 sec
Token passing	1 min 40 sec	19 sec

Table 2. PAX results for state graph construction

Example 4 (Szymanski cont'd). For the Szymanski's algorithm and the abstraction function given in Example 3, we obtain using the last method of Section 5.3 the state graph presented in Figure 1. The labeling function μ is also given in Figure 1 in form of the predicates $\widehat{\mu(q)}$ for each node q. The initial states are marked by an edge without source node. It is not difficult to check that no state falsifying the mutual exclusion property is reachable in the obtained graph. $\diamond$

6 Proving Liveness Properties

Let $\mathcal{S}$ be a WS1S transition system and $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{N}_0, \mu)$ be a state graph of $\mathcal{S}_A$ constructed from $\mathcal{S}$ and an abstraction function α. Let φ be an LTL formula. Let $\xi_1, \cdots, \xi_n$ be the atomic subformulae of φ such that, for every node $p \in \mathcal{G}$, either all states in $\mu(p)$ satisfy ξ_i or none does, i.e., we have either $\widehat{\mu(p)} \rightarrow \xi_i$ or $\widehat{\mu(p)} \rightarrow \neg\xi_i$ is valid. For each ξ_i, we introduce a proposition P_i. Then, let $\check{\mathcal{G}} = (\mathcal{N}, \mathcal{E}, \mathcal{N}_0, \nu)$ be a Kripke structure with the same nodes, edges, and initial nodes as $\mathcal{G}$ and such that $P_i \in \nu(p)$ iff $\widehat{\mu(p)} \rightarrow \xi_i$ is valid. Let also $\check{\varphi}$ be the formula obtained from φ by substituting P_i for ξ_i. Then, $\mathcal{S} \models \varphi$, if $\check{\mathcal{G}} \models \check{\varphi}$.

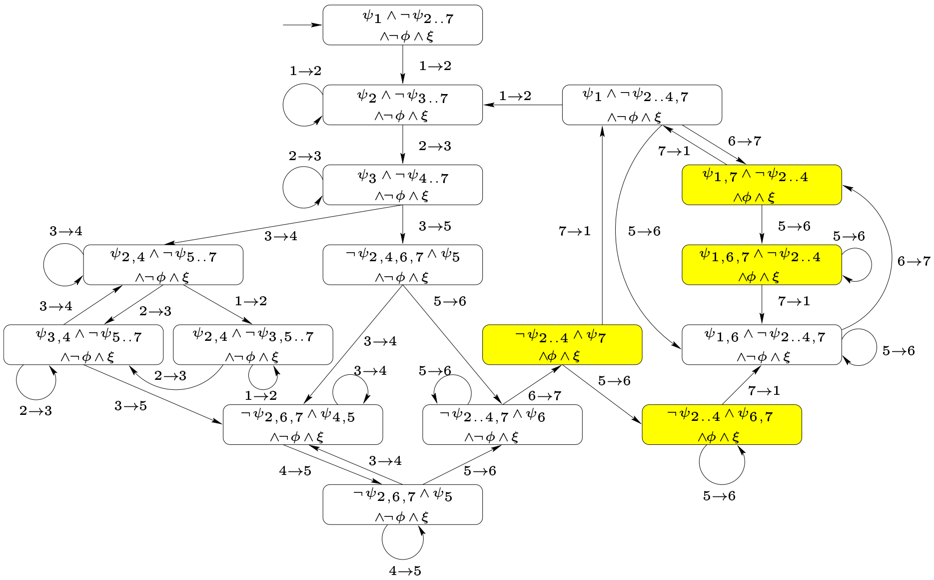

Fig. 1. Reachability graph for Szymanski's mutual exclusion algorithm

Example 5 (Szymanski cont'd). Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{N}_0, \mu)$ be the state graph for Szymanski's algorithm given in Example 4. Let us now consider the liveness property $\varphi \equiv \Box\Diamond\psi_7$ expressing that the critical section is entered infinitely often. Clearly, we can define a Kripke structure $\check{\mathcal{G}} = (\mathcal{N}, \mathcal{E}, \mathcal{N}_0, \nu)$ with the same nodes and edges as $\mathcal{G}$ and with a proposition P such that $P \in \nu(p)$ iff $\models \widehat{\mu(p)} \rightarrow \psi_7$. Thus, all nodes labeled by P are shadowed in Figure 1. Moreover, we get $\check{\varphi} \equiv \Box\Diamond P$. $\diamond$

It is easy to see that $\check{\varphi}$ does not hold in the Kripke structure $\check{\mathcal{G}}$. This is due to the cycles which generate infinite traces in $\check{\mathcal{G}}$ without ever reaching a shadowed node. However, these traces have no corresponding computations in the concrete WS1S system. E.g., the loops labeled $1 \rightarrow 2$ are the abstraction of the transition taking an element out of At_ℓ_1 and adding it to At_ℓ_2. Clearly, since WS1S is interpreted over finite sets, it is impossible to infinitely execute transition $1 \rightarrow 2$ without taking a transition that adds elements to At_ℓ_1.

In this section we present a method that allows us to add fairness conditions to the Kripke structure $\check{\mathcal{G}}$ constructing a fair Kripke structure $\check{\mathcal{G}}^F$ such that we still have $\mathcal{S} \models \varphi$, if $\check{\mathcal{G}}^F \models \check{\varphi}$ and such that the added fairness conditions rule out infinite traces in $\check{\mathcal{G}}$ that have no counter-parts in $\mathcal{S}$.

The method uses a marking algorithm that labels each edge of the considered state graph with one of the symbols $\{+_X, -_X, =_X\}$ for each set variable X of the original WS1S system. Intuitively, the labels $-_X$ resp. $=_X$ express whether the transitions at the concrete level reduce resp. maintain the cardinality of a set X, the label $+_X$ represents all other cases.

For the sake of presentation, we use $\widehat{p}(\mathcal{V}_A)$ instead of $\widehat{\mu(p)}$.

Marking Algorithm

Input: WS1S system $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$, abstraction relation $\widehat{\alpha}$, state graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{N}_0, \mu)$ of $\mathcal{S}_A$

Output: Edge labeling of $\mathcal{G}$

Description: For each $X \in \mathcal{V}$, for each edge $e = (p, \tau_A, q) \in \mathcal{E}$, let τ be the concrete transition in $\mathcal{T}$ corresponding to τ_A. Moreover, let $\Delta(X, e, \prec)$, with $\prec \in \{\subset, =\}$, denote the WS1S formula:

$$\widehat{p}(\mathcal{V}_A) \wedge \widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \wedge \widehat{q}(\mathcal{V}'_A) \wedge \widehat{\alpha}(\mathcal{V}', \mathcal{V}'_A) \wedge \tau(\mathcal{V}, \mathcal{V}') \Rightarrow X' \prec X \ .$$

Then, mark e with $-_X$, if $\Delta(X, e, \subset)$ is valid, mark e with $=_X$, if $\Delta(X, e, =)$ is valid, and mark e with $+_X$ otherwise.

Now, for a set variable X we denote with $\mathcal{E}_X^+$ the set of edges labeled with $+_X$.

Then, the fair Kripke structure is defined as $\check{\mathcal{G}}^F = (\check{\mathcal{G}}, F)$ where F is the set of strong fairness conditions containing for each edge e and each of its labels $-_X$ the set $(e, \mathcal{E}_X^+)$. Each fairness condition states that e can only be taken infinitely often if one of the edges in $\mathcal{E}_X^+$ is taken infinitely often.

Example 6 (Szymanski cont'd). Figure 2 shows part of the abstract state graph after running the labeling algorithm. All $=_X$ symbols are left out and the labels $+_{\mathrm{At\_}\ell_k}, -_{\mathrm{At\_}\ell_k}$ are abbreviated with $+_k, -_k$. The figure shows a strongly con-

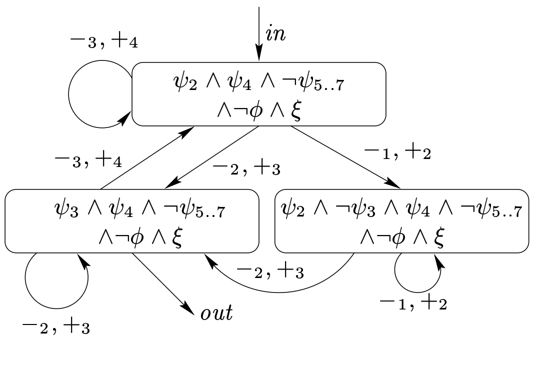

Fig. 2. Part of the labeled state graph

nected part of the graph with the only ingoing edge *in* and only outgoing edge *out*. To prove communal accessibility it is necessary to show that the system cannot cycle forever in this component.

It can be proved, e.g., using model-checking, that $\check{\mathcal{G}}^F \models \check{\varphi}$. Hence, Szymanski's algorithm satisfies $\Box\Diamond\, \exists_P\, i : i \in \mathrm{At\_}\ell_7$. $\diamond$

7 Conclusion

We have presented a method for the verification of parameterized networks of finite processes. Our method is based on the transformation of an infinite family of systems into a single WS1S transition system and applying abstraction techniques on this system. We also showed how our method can deal with liveness properties. We have applied this method, which has been implemented in our tool PAX, to a number of parameterized protocols. The obtained results are encouraging.

Closest to our work is [ABJN99]. Therefore, we give a short account of the main differences between this and our work. While the method in [ABJN99] aims at computing the exact set of reachable states, our method computes an over-approximation. On the other hand, their method may fail because of the divergence of the exploration algorithm, even when acceleration is applied. Moreover, our method can deal with a larger class of networks and with a class of liveness properties, often called communal accessibility.

We intend to extend our method to deal with a larger class of liveness properties. It is also clear that we can use WS2S instead of WS1S when we consider networks arranged in trees.

References

[ABJN99] P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling Global Conditions in Parameterized System Verification. In N. Halbwachs and D. Peled, editors, *CAV '99*, volume 1633 of *LNCS*, pages 134–145. Springer, 1999.

[AK86] K. Apt and D. Kozen. Limits for Automatic Verification of Finit-State Concurrent Systems. *Information Processing Letters*, 22(6):307–309, 1986.

[BCG89] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 1989.

[Büc60] J.R. Büchi. Weak Second-Order Arithmetic and Finite Automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[CGJ95] E. Clarke, O. Grumberg, and S. Jha. Verifying Parameterized Networks using Abstraction and Regular Languages. In I. Lee and S. Smolka, editors, *CONCUR '95: Concurrency Theory*, LNCS. Springer, 1995.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5), 1994.

[DGG94] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. In E.-R. Olderog, editor, *Proceedings of PROCOMET '94*. North-Holland, 1994.

[Elg61] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[EN95] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *22nd ACM Symposium on Principles of Programming Languages*, pages 85–94, 1995.

[EN96] E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems. In *8th Conference on Computer Aided Verification*, LNCS 1102, pages 87–98, 1996.

[Gru97] O. Grumberg, editor. *Proceedings of CAV '97*, volume 1256 of *LNCS*. Springer, 1997.

[GS92] S.M. German and A.P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

[GS97] S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In Grumberg [Gru97].

[HJJ+96] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic Second-Order Logic in Practice. In *TACAS '95*, volume 1019 of *LNCS*. Springer, 1996.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. An experience in proving regular networks of processes by modular model checking. *Acta Informatica*, 22(6/7), 1992.

[KM89] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *ACM Symp. on Principles of Distributed Computing, Canada*, pages 239–247, Edmonton, Alberta, 1989.

[KM98] N. Klarlund and A. Møller. MONA Version 1.3 User Manual. BRICS, 1998.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In Grumberg [Gru97], pages 424–435.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1), 1995.

[LHR97] D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *POPL '97*, Paris, 1997.

[MP94] Z. Manna and A. Pnueli. Verification of parameterized programs. In E. Borger, editor, *Specification and Validation Methods*, pages 167–230. Oxford University Press, 1994.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems, Safety*. Springer Verlag, 1995.

[SG89] Z. Stadler and O. Grumberg. Network grammars, communication behaviours and automatic verification. In *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science, pages 151–165, Grenoble, France, 1989. Springer Verlag.

[Szy88] B.K. Szymanski. A simple solution to Lamport's concurrent programming problem with linear wait. In *Proceedings of International Conference on Supercomputing Systems 1988*, pages 621–626, St. Malo, France, July 1988.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Methods and Semantics*, pages 134–191. Elsevier Science Publishers B. V., 1990.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants (extended abstract). In Sifakis, editor, *Workshop on Computer Aided Verification*, LNCS 407, pages 68–80, 1989.