Journal of Universal Computer Science, vol. 7, no. 2 (2001), 141-158
submitted: 1/9/00, accepted: 13/10/00, appeared: 28/2/01 *© Springer Pub. Co.*

Verification of Parameterized Protocols

Kai Baukus

(Institute of Computer Science and Applied Mathematics
CAU Kiel, Preusserstr. 1–9, D-24105 Kiel, Germany.
kba@informatik.uni-kiel.de)

Yassine Lakhnech

(Verimag, Centre Equation, 2 Av. de Vignate,
38610 Gières, France.
lakhnech@imag.fr)

Karsten Stahl

(Institute of Computer Science and Applied Mathematics
CAU Kiel, Preusserstr. 1–9, D-24105 Kiel, Germany.
kst@informatik.uni-kiel.de)

Abstract: 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 processes.

In this paper, we present a method for the verification of so-called *universal* properties of fair parameterized networks of similar processes, that is, properties of the form $\forall p_1 \ldots p_n : \psi$, where ψ is a quantifier-free LTL formula and the p_i refer to processes. To prove an universal property of a parameterized network, we first model the infinite family of networks by a single fair WS1S transition system, that is, a transition system whose variables are set (2nd-order) variables and whose transitions are described in WS1S. Then, we abstract the WS1S system into a finite state system that can be model-checked. We present a generic abstraction relation for verifying universal properties as well as an algorithm for computing an abstract system.

However, the abstract system may contain infinite computations that have no corresponding fair computations at the concrete level, and hence, in case the property of interest is a progress property, verification may fail because of this. Therefore, we present methods that allow to synthesize fairness conditions from the parameterized network and discuss under which conditions and how to lift fairness conditions of this network to fairness conditions on the abstract system. We implemented our methods in a tool, called PAX, and applied it to several examples.

Key Words: Parameterized systems, verification, abstraction, model checking, WS1S
Category: F.3.1

1 Introduction

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 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.

Formally, in this paper, we present a method for tackling the following problem:

Given a parameterized network $P_1 \parallel \cdots \parallel P_n$, fairness conditions, and a quantifier-free linear-time temporal property $\psi(p_1 \ldots, p_k)$, we want to prove $P_1 \parallel \cdots \parallel P_n \models \forall p_1 \ldots, p_k \leq n : \psi(p_1 \ldots, p_k)$, for every $n \in \omega$, i.e., every fair computation of $P_1 \parallel \cdots \parallel P_n$ satisfies $\psi(p_1 \ldots, p_k)$.

Our method uses the verification by abstraction approach [CGL94, DGG94] and consists of the following steps:

1. Representing the infinite family of fair networks $P_1 \parallel \cdots \parallel P_n$ as a single fair transition system $\mathcal{S}$ whose variables range over finite sub-sets of ω and whose transitions are expressed in WS1S, the weak second-order logic of one-successor [Büc60]. We call such systems fair WS1S transition systems.

2. Constructing an abstraction relation that maps the states of the WS1S transition system $\mathcal{S}$ to abstract states which are valuations of boolean variables. We present a generic abstraction relation for verifying universal properties.

3. Automatically constructing a finite abstract system $\mathcal{S}_A$ that is an abstraction of $\mathcal{S}$ which implies that every computation of $\mathcal{S}$ can be mapped to a computation of $\mathcal{S}_A$. Moreover, we construct an abstract formula ψ_A such that if $\mathcal{S}_A$ satisfies ψ_A then, using the preservation results of [CGL94, DGG94], we can deduce $P_1 \parallel \cdots \parallel P_n \models \forall p_1 \ldots, p_k \leq n : \psi(p_1 \ldots, p_k)$, for every $n \in \omega$.

4. Since the abstract system is finite, we can use model-checking to verify that it satisfies ψ_A. However, verifying progress properties using abstractions often fails because of infinite computations in the abstract system that do not correspond to fair infinite ones in the concrete one. To mitigate this problem, we augment the abstract system with *safe* fairness conditions, that is, conditions that only remove infinite computations that do not correspond to concrete ones. We present two techniques for synthesizing fairness conditions from the concrete system that can be safely added to the abstract one:

   (a) An algorithm that given a WS1S formula characterizing a ranking function computes pairs of sets of transitions expressing strong fairness conditions that are *guaranteed* to hold for the parameterized network, and hence, abstractions of them can be safely added at the abstract level.

(b) A method that allows the generation of fairness conditions at the abstract level from the fairness conditions of the parameterized network. In particular, we discuss which kind of weak/strong fairness can be lifted from the concrete to the abstract level.

We implemented our method in a tool, we call PAX [1], that uses the decision procedures of MONA [HJJ$^+$96] to check the satisfiability of WS1S formulae. We then applied our tool and method to several examples including Dijkstra's and Szymanski's mutual exclusion algorithms as well as a time-triggered group membership protocol.

2 Preliminaries

In this section we briefly recall the definition of weak second order theory of one successor (WS1S for short) [Büc60, Tho90].

Terms of WS1S are built up from the constant 0 and 1st-order variables by applying the successor function suc(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.

WS1S-formulae are interpreted in models that assign finite sub-sets 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 $\mathit{free}(f)$.

In addition to the usual abbreviations, we use $\forall_n\, i : f$ as an abbreviation for $\forall i : i < n \rightarrow f$ and $\exists_n\, i : f$ for $\exists i : i < n \wedge f$. Moreover, 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]).

3 WS1S Transition Systems

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 parameterized networks. In order to simulate the behavior of parameterized networks with fairness conditions we also need the notion of fairness for WS1S transition systems.

Definition 1 (Fair WS1S Transition Systems).
A *fair WS1S transition system* $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C})$ is given by the following components:

[1] http://www.informatik.uni-kiel.de/~kba/pax

- $\mathcal{V} = \{X_1, \ldots, X_k\} \cup \{m_1, \ldots, m_l\}$: A finite set of second order variables X_i ranging over finite sets of natural numbers and first order variables m_i ranging over the natural numbers itself.
- Θ: A WS1S formula with $\mathit{free}(\Theta) \subseteq \mathcal{V}$ 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., $\mathit{free}(\rho_\tau) \subseteq \mathcal{V} \cup \mathcal{V}'$, where primed variables refer to the post state.
- $\mathcal{J}$: A set of pairs of second order variables expressing a weak fairness condition. Each pair $(X_m, X_{m'})$ requires, for each $i \in \omega$, the weak fairness condition that i cannot be continuously in X_m without being eventually in $X_{m'}$, that is, $\forall i \in \omega : (\Diamond\Box(i \in X_m) \rightarrow \Box\Diamond(i \in X_{m'}))$.
- $\mathcal{C}$: A set of pairs $(X_m, X_{m'})$ of second order variables expressing the strong fairness condition $\forall i \in \omega : (\Box\Diamond(i \in X_m) \rightarrow \Box\Diamond(i \in X_{m'}))$. $\Box$

A state s of $\mathcal{S}$ maps the 2nd-order variables in $\mathcal{V}$ into finite sub-sets of ω and the 1st-order variables to natural numbers. A *computation* of $\mathcal{S}$ is a sequence $(s_i)_{i\in\omega}$ of states such that $\Theta[s_0(\mathcal{V})/\mathcal{V}]$ and $\bigvee_{\tau\in\mathcal{T}} \tau[s_i(\mathcal{V}), s_{i+1}(\mathcal{V})/\mathcal{V}, \mathcal{V}']$ are valid formulae. A computation $(s_i)_{i\in\omega}$ satisfies a weak fairness condition $(X_m, X_{m'}) \in \mathcal{J}$ iff the following condition holds for every $x \in \omega$:

if $\exists i \in \omega.\forall j \geq i : x \in s_j(X_m)$*, then there exist infinitely many* i*'s such that* $x \in s_i(X_{m'})$.

The computation satisfies the strong fairness condition $(X_m, X_{m'}) \in \mathcal{C}$ iff the following condition holds for every $x \in \omega$:

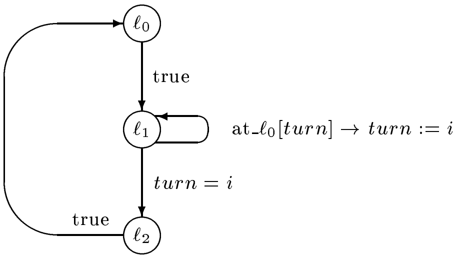
if there exist infinitely many i*'s such that* $x \in s_i(X_m)$*, then there exist infinitely many* i*'s such that* $x \in s_i(X_{m'})$.

Then, a *fair computation* of $\mathcal{S}$ is a computation that satisfies all fairness conditions in $\mathcal{J}$ and $\mathcal{C}$. Henceforth, we denote the set of fair computations of $\mathcal{S}$ by $[\![\mathcal{S}]\!]$.

4 Representing Parameterized Protocols as WS1S Systems

Before presenting the methods to abstract and analyze WS1S systems we show how to model parameterized protocols as WS1S systems. As examples we use a simple mutual exclusion algorithm operating in an asynchronous manner, i.e., we have an interleaving semantics, and a fault detection protocol for processes organized in a ring operating in a time-triggered, synchronous manner.

Example 1 (Mutual Exclusion). The parameterized network consists of processes where each process is described as follows:



The variable $turn$ is meant to indicate which process has the right to enter ℓ_2. If the process having the $turn$ is at ℓ_0, $turn$ is free and another process at ℓ_1 may take it.

The transition from ℓ_0 to ℓ_1 is weak fair whereas the loop from ℓ_1 to ℓ_1 is strong fair. Initially, all processes are at ℓ_0. Location ℓ_2 represents the critical section.

It is easy to see how each process P_i can be described using a boolean variable $\text{at\_}\ell_m[i]$ for each control point $\ell_m[i]$.

We will verify that the algorithm satisfies the mutual exclusion property as well as the universal property that each process p reaches its critical section infinitely often, i.e., $\forall_n\, p : \Box\Diamond \text{at\_}\ell_2[p]$.

To represent this network as a WS1S system we introduce three set variables $\text{At\_}\ell_0, \text{At\_}\ell_1, \text{At\_}\ell_2$ corresponding to the control locations, the first-order variable $turn$, and a set variable P representing the set of processes being part of the network. Moreover, we need two additional set variables E_τ and T_τ for each transition to express the fairness conditions, a process index will be member of these sets whenever the corresponding transition is enabled (resp. just taken) for this process. Let $\mathcal{V}$ denote this set of variables. If we denote by τ_0 the self-loop at ℓ_1, then $\mathcal{C} = \{(E_{\tau_0}, T_{\tau_0})\}$. The liveness property we will check later can then be expressed by $\forall_P\, p : \Box\Diamond(p \in \text{At\_}\ell_2)$. For the sake of illustration, we show the representation of τ_0:

$$\exists_P\, i : \rho_{\tau_0}(\mathcal{V}', \mathcal{V}'', i) \wedge \bigwedge_{\tau \in \mathcal{T}} E'_\tau = \{i \in P \mid \exists \mathcal{V}'' : \rho_\tau(\mathcal{V}', \mathcal{V}'', i)\} \ ,$$

where $\rho_{\tau_0}(\mathcal{V}', \mathcal{V}'', i)$ characterizes those $i \in P$ which can take a τ_0-step, the existential quantification corresponds to an interleaving semantics such that only one process proceeds in one step, and the last conjunct gives the enabled transitions in the post state. $\rho_{\tau_0}(\mathcal{V}', \mathcal{V}'', i)$ (and similarly ρ_τ for the other transitions) is defined as:

$$\begin{aligned}\rho_{\tau_0}(\mathcal{V}', \mathcal{V}'', i) \equiv\ & i \in \text{At\_}\ell_1 \wedge turn \in \text{At\_}\ell_0 \wedge i \in \text{At\_}\ell'_1 \wedge i = turn' \\ & \wedge (\forall_P\, j : j \neq i \Rightarrow \bigwedge_{k=0,1,2} (j \in \text{At\_}\ell_k \Leftrightarrow j \in \text{At\_}\ell'_k)) \\ & \wedge P = P' \wedge \bigcup_{B \in \widetilde{V}} B' \subseteq P' \wedge \bigcup_{\tau \neq \tau_0} T'_\tau = \emptyset \wedge T'_{\tau_0} = \{i\} \ .\end{aligned}$$

$\Box$

Obviously, synchronous systems can be modeled using universal quantification in the definition of transitions. We illustrate that with a small example and give a more sophisticated application in Section 7.

Example 2 (Simple Fault Detection). In this example we have a parameterized number of processes organized in a ring structure operating in a synchronous manner. In each round it is the turn of one process to send information to all others.

The protocol requires one bit of error information piggybacked on regular broadcasts. A processor may fail to receive or send information. In case of a send error all other processes set their acknowledgment bit ack to false and the faulty processor notices that in the next round. In case of a receive error the faulty processor sets its acknowledgment bit to false and the others get this information when it is the turn of the faulty processor to send. In both cases the protocol guarantees that a send or receive error by one process is detected within one round, i.e., all processes have their acknowledgment bit set to false.

A process p takes one of the following transitions synchronously with the other processes.

$$\neg\text{arrived}(p) \rightarrow \text{ack}'(p) := \mathit{ff} \tag{1}$$
$$\text{arrived}(p) \rightarrow \text{ack}'(p) := \text{ack}(p) \wedge \text{ack}(\mathit{turn}) \tag{2}$$
$$p = \mathit{turn} \rightarrow \text{ack}'(p) := \text{ack}(p) \tag{3}$$

It is process turn which has to send at the moment. Each transition increases the turn variable by 1 modulo n. The turn process maintains its ack bit. arrived(p) stands for: $\mathit{turn} \neq p$, p is not receive-faulty, and process turn is not send-faulty at this step.

To model this system as a WS1S system we need 2 second-order variables P, Ack where P contains all participating processes $\{0, \ldots, n-1\}$ and Ack contains those having their ack bit set to true.

Transition 2 where no new errors occur and the ack bits are just propagated can be characterized as:

$$\begin{aligned}
&(\forall_P\, p : p \neq \mathit{turn} \Rightarrow \\
&\quad (p \in \text{Ack}' \Leftrightarrow p \in \text{Ack} \wedge \mathit{turn} \in \mathit{Ack})) \\
&\wedge (\mathit{turn} \in \text{Ack}' \Leftrightarrow \mathit{turn} \in \mathit{Ack}) \\
&\wedge \mathit{turn}' = (\mathit{turn} + 1 \bmod n) \wedge P = P'
\end{aligned}$$

Then, we would like to prove; whenever one process has detected an error, i.e., $\exists_P\, p : p \notin \text{Ack}$, then eventually all processes will notice that, i.e., $\text{Ack} = \emptyset$. That can be formulated as a universal property:

$$\forall_P\, p : \Box(p \notin \text{Ack} \Rightarrow \Diamond \text{Ack} = \emptyset)$$

□

Note that the class of systems we can model as WS1S systems is restricted such that each process has to be finite state and the transitions can be characterized in WS1S. The definition of a class that can be modeled as WS1S system and the translation can be found in [BLS00].

5 Abstracting WS1S Systems

In Section 3, we have shown how we model parameterized networks as fair WS1S systems. An infinite family of systems is represented by a single, though infinite-state, transition system.

In the following, we present a method to construct a finite abstraction of a given WS1S systems. Then, we show in Section 6 how the obtained abstract system can be enriched with fairness conditions such that interesting progress properties of the WS1S system can be verified. Let us first define what we mean by universal temporal properties.

Let Π be a countable set of process indices p that range over natural numbers and let Ξ be a countable set of variables X that range over finite sets of natural numbers. The set LTL of linear-time temporal properties over Π and Ξ is defined as follows:

$$\varphi ::= i \in X \mid p \in X \mid t_1 = t_2 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\, \mathcal{U}\, \varphi,$$

where i is a constant in ω and t_1, t_2 are first resp. second order terms in WS1S. As usual, we use the temporal modalities $\Box$ (always) and $\Diamond$ (eventually) which can be introduced as abbreviations.

Formulae in LTL are interpreted over infinite sequences of structures of the form $(\mathcal{I}, \mathcal{I}')$, where $\mathcal{I}$ maps each variable $X \in \Xi$ to a finite sub-set of ω and $\mathcal{I}'$ maps each variable in Π to an element of ω. The definition of the interpretation of LTL is not given here as it is standard.

Let φ be an LTL formula with $\{X_1, \ldots, X_k\}$ as free 2nd-order variables and $\{p_1, \ldots, p_n\}$ as free 1st-order variables. Moreover, let $\mathcal{S}$ be a WS1S transition system with $\{X_1, \ldots, X_k\}$ as variables. A computation $(s_i)_{i\in\omega}$ satisfies φ iff for every injective mapping $\mathcal{I}'$ from $\{p_1, \ldots, p_n\}$ into ω, the sequence $(s_i, \mathcal{I}')_{i\in\omega}$ satisfies φ. In other words, the computation $(s_i)_{i\in\omega}$ satisfies φ, if it satisfies all the temporal formulae obtained by instantiating the variables $p_1, \ldots, p_n$. We say that $\mathcal{S}$ satisfies φ, denoted by $\mathcal{S} \models \varphi$, if every fair computation of $\mathcal{S}$ satisfies φ.

A temporal property is called *universal*, if it can be described by a formula in LTL. For instance, mutual-exclusion in Example 1 can be described by the formula $\Box\neg(p_1 \in \mathrm{At\_}\ell_2 \wedge p_2 \in \mathrm{At\_}\ell_2)$, which is an universal temporal property. However, the communal liveness property stating whenever some process is in $\mathrm{At\_}\ell_1$, eventually some process (not necessarily the same) reaches $\mathrm{At\_}\ell_2$ is not an universal temporal property. On the other hand, the stronger liveness property stating that every process in $\mathrm{At\_}\ell_1$ eventually reaches $\mathrm{At\_}\ell_2$ is an universal property as it can be described by the property $\Box(p \in \mathrm{At\_}\ell_1 \Rightarrow \Diamond p \in \mathrm{At\_}\ell_2)$.

The problem we are interested in is given a WS1S system $\mathcal{S}$ and given an universal temporal formula φ to show $\mathcal{S} \models \varphi$.

5.1 Abstractions and fair abstractions

Given a deadlock-free[2] 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.

[2] Throughout this paper we only consider deadlock free transition systems which can be achieved by adding an idle transition.

α, denoted by $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}_A$, if the following conditions are satisfied: (1) $s_0 \models \Theta$ implies $\alpha(s_0) \models \Theta_A$ and (2) $\tau \circ \alpha^{-1} \subseteq \alpha^{-1} \circ \tau_A$.

In case Σ_A is finite, we call α finite abstraction relation. Let φ, φ_A be LTL formulae and let $[\![\varphi]\!]$ (resp. $[\![\varphi_A]\!]$) denote the set of models of φ (resp. φ_A). Then, from $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}_A$, $\alpha^{-1}([\![\varphi_A]\!]) \subseteq [\![\varphi]\!]$, and $\mathcal{S}_A \models \varphi_A$ we can conclude $\mathcal{S} \models \varphi$ (here we identify α^{-1} and its point-wise lifting to sequences). 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].

In case $\mathcal{S}$ is a fair transition system with $\mathcal{F}$ as fairness formula and if $\mathcal{F}_A$ is the fairness formula of $\mathcal{S}_A$, then by requiring $\alpha^{-1}([\![\neg\mathcal{F}_A]\!]) \subseteq [\![\neg\mathcal{F}]\!]$, we have the same preservation result as above. We indicate this type of abstraction by $\mathcal{S} \sqsubseteq_\alpha^F \mathcal{S}_A$.

Next, we explain the main steps of our approach for verifying universal temporal properties before presenting each step in more detail.

5.2 Approach

Let $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C})$ be a fair WS1S system and let ψ be an universal temporal formula with $\{X_1, \ldots, X_k\} \cup \{p_1, \ldots, p_m\}$ as free variables. To simplify the presentation assume that $m = 1$ and write p instead of p_1. Moreover, we denote by $\psi(i)$ the formula obtained from ψ by replacing p by the constant $i \in \omega$.

For each $i \in \omega$, we construct in Section 5.3 a finite abstraction relation α_i which maps states of $\mathcal{S}$ to abstract states. The abstract state space defined by α_i is such that it contains for each sub-formula $i \in X$ of $\psi(i)$ a boolean variable b^i_X and for each X_j an abstract variable b_j. Then, α_i relates a concrete state s to an abstract state s^A such that $s^A(b^i_X) \Leftrightarrow i \in s(X)$ and $s^A(b_j) \Leftrightarrow s(X_j) \neq \emptyset$. Henceforth, let $\widehat{\alpha_i}$ be a predicate defining α_i, i.e., $\widehat{\alpha_i}$ is a conjunction of those equivalences mentioned above. Clearly, the abstract state spaces defined by α_i and α_j are the same modulo renaming of the variables b^i_X.

Then, for each $i \in \omega$, one can *effectively* construct a finite abstract system S^i_A and an LTL-formula[3] ψ^i_A such that $S^i_A \models \psi^i_A$ implies $\mathcal{S} \models \psi(i)$. One can even effectively construct the set $\{S^i_A \mid i \in \omega\}$ of abstract systems. However, although this set is finite, it is computationally costly to construct. Therefore, we present in Section 5.4 an algorithm for constructing a *single* finite abstract system S_A which is itself an abstraction of each S^i_A and, as we show, is an abstraction of $\mathcal{S}$. Moreover, we show how to construct an LTL-formula ψ_A such that $S_A \models \psi_A$ implies $\mathcal{S} \models \psi$.

5.3 Abstraction relation α_i

The set $\mathcal{V}^i_A$ of abstract variables consists of boolean variables. For each set X in the WS1S system $\mathcal{S}$ we have an abstract boolean variable $b^i_X \in \mathcal{V}^i_A$ corresponding to $i \in X$. Thus, in particular we have the variable $b^i_{E_\tau}$ and $b^i_{T_\tau}$, for each $\tau \in$

[3] Here, we extend the definition of LTL formulae with boolean variables.

$\mathcal{T}$. Additionally, for each strong fairness condition $(E_\tau, T_\tau) \in \mathcal{C}$ we introduce boolean variables e_τ, t_τ such that α_i implies:

$$\begin{aligned} e_\tau &\equiv \exists_P\, j : j \in E_\tau \\ t_\tau &\equiv \exists_P\, j : j \in T_\tau \ . \end{aligned}$$

That means that $\widehat{\alpha}_i$ contains these equivalences as conjuncts. For all other global state properties φ that may influence the progress of a certain process p another variable is added for which the abstraction is given by φ. This includes an adequate abstraction of the used natural numbers to express their influence on the behavior of the system.

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

5.4 Construction of $\mathcal{S}_A$

As mentioned it is costly to compute $\{S_A^i \mid i \in \omega\}$ explicitly. Therefore, we show how one can construct a system that abstracts each of the elements of this set, and hence, by transitivity of $\sqsubseteq$ abstracts $\mathcal{S}$. The set $\mathcal{V}_A$ of abstract variables of $\mathcal{S}_A$ contains for each abstract variable $b^i \in \mathcal{V}_A^i$ a variable b.

We define the transitions of $\mathcal{S}_A$ by the following WS1S formula:

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

Thus, we make sure that the choice of p in the concretizations of the source and target states of an abstract transition is the same. We can then show the following:

Proposition 2. *$\mathcal{S}_A$ is an abstraction of $\mathcal{S}$, i.e., $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}_A$ with $\alpha = \bigcup_{i \in \omega} \alpha_i$.*

Notice that the formulae above are WS1S formulae, and hence, by Büchi and Elgot's result, the sets of numbers satisfying these formulae can be characterized by finite automata. We use Mona [HJJ+96] to construct these automata.

6 Fair Abstractions

It is well known that an obstacle to the verification of liveness properties using abstraction, is that often the abstract system contains cycles that do not correspond to fair computations of the concrete system. 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 computations. We present a marking 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. The enriched graph is used to prove liveness properties of the WS1S systems, and consequently, of the parameterized network. Moreover, we discuss under which requirements the fairness conditions of the parameterized system can be lifted to the finite abstract one. In particular, we show that by requiring some conditions on the abstraction relation, it is sound to lift strong fairness. Weak fairness can only be lifted for a distinguished process.

Throughout this section, we fix a WS1S system $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C})$ and an abstraction relation α constructed as explained in Section 5. Then, let $\mathcal{S}_A = (\mathcal{V}_A, \Theta_A, \mathcal{T}_A)$ be the finite abstract system (without fairness) obtained by the method introduced in Section 5. We show how to add fairness conditions to $\mathcal{S}_A$ leading to a fair abstract system $\mathcal{S}_A^F = (\mathcal{V}_A, \Theta_A, \mathcal{T}_A, \mathcal{J}_A, \mathcal{C}_A)$ such that $\mathcal{S} \sqsubseteq_\alpha^F \mathcal{S}_A^F$.

6.1 Marking algorithm

We use WS1S formulae to express ranking functions. Let $\chi(i, X_1, \cdots, X_k)$ be a predicate with i as free 1st-order variable and $X_1, \cdots, X_k \in \mathcal{V}$ as free 2nd-order variables. Given a state s of $\mathcal{S}$, i.e., a valuation of the variables in $\mathcal{V}$, the ranking value $\zeta(s)$ associated to s by ζ is the cardinality of $\{i \in \omega \mid \chi(i, s(X_1), \ldots, s(X_k))\}$.

The marking algorithm we present labels each abstract transition of the abstract system with one of the symbols $\{+_\chi, -_\chi, =_\chi\}$. Intuitively, an abstract transition τ_A is labeled by $-_\chi$, if it is guaranteed that the concrete transition τ associated with τ_A decreases the ranking value, i.e., $(s, s') \in \tau$ implies $\zeta(s) > \zeta(s')$. If that cannot be shown one checks whether at least the ranking value is never increased. Then, τ_A is labeled by $=_\chi$. Otherwise, the abstract transition is labeled by $+_\chi$. Since these properties of the concrete transitions can be formulated in WS1S, they are decidable.

Input: WS1S system $\mathcal{S} = (\mathcal{V}, \Theta, \mathcal{T})$, abstraction $\mathcal{S}_A = (\mathcal{V}_A, \Theta_A, \mathcal{T}_A)$, set of predicates $\chi(i, X_1, \cdots, X_k)$

Output: Labeling of $\mathcal{T}_A$

Description: For each $\chi(i, X_1, \cdots, X_k)$, for each edge $\tau_A \in \mathcal{T}_A$, let τ be the concrete transition in $\mathcal{T}$ corresponding to τ_A. Moreover, let $\Delta(\chi, \tau, \prec)$, with $\prec \in \{\subset, \subseteq\}$, denote the WS1S formula:

$$\widehat{\alpha}(\mathcal{V}, \mathcal{V}_A) \wedge \widehat{\alpha}(\mathcal{V}', \mathcal{V}'_A) \wedge \rho_\tau(\mathcal{V}, \mathcal{V}') \Rightarrow \{i \mid \chi'(i)\} \prec \{i \mid \chi(i)\} \ .$$

Then, mark τ_A with $-_\chi$, if $\Delta(\chi, \tau, \subset)$ is valid. If not, check validity of $\Delta(\chi, \tau, \subseteq)$ and mark τ_A with $=_\chi$ if it is valid; otherwise mark τ_A with $+_\chi$.

Now, for a ranking function χ we denote with $\mathcal{T}_\chi^+$ the set of edges labeled with $+_\chi$. Then, we add for each such χ and each transition τ_A labeled with $-_\chi$ the fairness condition $(\tau_A, \mathcal{T}_\chi^+)$ which states that τ_A can only be taken infinitely often when one of the transitions in $\mathcal{T}_\chi^+$ are taken infinitely often.

Obviously, these generated fairness conditions are strong fairness requirements. However, for the abstract systems we have not defined formally how to express weak and strong fairness. Since we want to prove properties of the abstract system with model checking techniques we express the fairness conditions generated with the algorithm and those mentioned in the next section as LTL formulae. Then, we are able to ask the model checker if a desired property holds under the assumption that all traces are fair.

6.2 Lifting fairness

Recall that by definition of α (see Section 5), we introduce the abstract variables $e_\tau \equiv \exists_P\, i : i \in E_\tau$ and $t_\tau \equiv \exists_P\, i : i \in T_\tau$. We now argue that it is safe to augment $\mathcal{S}_A$ with the strong fairness $\mathcal{C}_A = \{(e_\tau, t_\tau) \mid (E_\tau, T_\tau) \in \mathcal{C}\}$, i.e., if e_τ is infinitely often true, then also t_τ is infinitely often true. Consider a computation where e_τ is infinitely often true, that is, $\exists_P\, i : i \in E_\tau$ is infinitely often true. Now, each instance of the parameterized system only contains a bounded number of processes, hence, by König's lemma, there must exists some i such that $i \in E_\tau$ infinitely often in this computation. Therefore, by the strong fairness condition of the concrete system, we must have $i \in T_\tau$ infinitely often, and hence, the computation satisfies $\Box\Diamond(\exists_P\, i : i \in T_\tau)$. Consequently:

Lemma 3. *Under the assumptions above we have $\mathcal{S} \sqsubseteq_\alpha^F \mathcal{S}_A$.* $\Box$

The reasoning above does not hold for weak fairness. Indeed, $\Diamond\Box e_\tau$ may hold for a computation without the existence of an i with $\Diamond\Box(i \in E_\tau)$.

Recall also that as explained in Section 5, we introduce for each transition of the distinguished process p abstract variables b_{E_τ} and b_{T_τ} expressing whether the transition is enabled, respectively, taken. We can show that it is safe to augment the abstract system with strong and weak fairness conditions on the transitions of p.

Lemma 4. *For the concrete WS1S system $\mathcal{S}$ and the abstract system $\mathcal{S}_A$ we have:*

$$\mathcal{S} \sqsubseteq_\alpha^F \mathcal{S}_A^F \ ,$$

with abstraction relation $\alpha = \bigcup_{i\in\omega} \alpha_i$ for a generic abstraction function α_i and fair abstract system $\mathcal{S}_A^F = (\mathcal{V}_A, \Theta_A, \mathcal{T}_A, \mathcal{J}_A, \mathcal{C}_A)$ with strong fairness requirements $\mathcal{C}_A = \{(b_{E_\tau}, b_{T_\tau}) \mid (E_\tau, T_\tau) \in \mathcal{C}\}$ and weak fairness requirements $\mathcal{J}_A = \{(b_{E_\tau}, b_{T_\tau}) \mid (E_\tau, T_\tau) \in \mathcal{J}\}$. $\Box$

Example 3. Recall that we want to verify that our algorithm of Example 1 satisfies the mutual exclusion property as well as the universal property that each process p reaches its critical section infinitely often, i.e., $\forall_n\, p : \Box\Diamond \mathrm{at\_}\ell_2[p]$.

According to the method presented in Section 5 we construct the abstract system $\mathcal{S}_A$ from the WS1S system. For the mutual exclusion property we take as abstract variable $inv \equiv \mathrm{At\_}\ell_2 \subseteq Turn \wedge \forall_P\, i, j : (i \in Turn \wedge j \in Turn) \Rightarrow i = j$. Our tool PAX constructs the abstract system and provides translations to several input languages for model-checkers, e.g., Spin and SMV. Also, the abstract state space can be explored to prove that inv is indeed an invariant of the abstract system and, hence, mutual exclusion holds for the original system.

Next, using the marking algorithm, we augment $\mathcal{S}_A$ with the strong fairness requirements $\{(t_{01}, t_{20}), (t_{12}, t_{01}), (t_{20}, t_{12})\}$ to obtain a fair abstract system $\mathcal{S}_A^F$. Moreover, with Lemma 3 we can lift the strong fairness (e_{11}, t_{11}). Lemma 4 allows us to augment $\mathcal{S}_A^F$ with another strong fairness condition $(b_{E_{11}}, b_{T_{11}})$ for the distinguished process as well as with the weak fairness $(b_{E_{01}}, b_{T_{01}})$.

All the fairness conditions can be expressed as LTL formulae. We used Spin to prove that $\Box\Diamond b_{\mathrm{At\_}\ell_2}$ holds in $\mathcal{S}_A^F$ which means that, in the original system, each process reaches its critical section infinitely often. $\Box$

Example 4. In Example 2 we want to prove the universal property

$$\forall_P\, p : \Box(p \notin \mathrm{Ack} \Rightarrow \Diamond \mathrm{Ack} = \emptyset).$$

Hence, we choose as abstract variables $err_p \equiv p \notin \mathrm{Ack}$, $detected \equiv \mathrm{Ack} = \emptyset$, $turn_p \equiv p = turn$. Since our second example is based on synchronous semantics we have no fairness conditions at the concrete level. But, to prove the property on the abstract level we need to know that eventually it is the turn of process p to send. Of course, at the concrete level this is guaranteed since $turn$ is increased by 1 modulo n in each step. For the abstract system the needed fairness condition $\Box\Diamond turn_p$ can be generated with the marking algorithm choosing $\chi(i) \equiv (turn < i < p) \vee (p < turn < i) \vee (i < p < turn)$ which characterizes all processes i which will get $turn$ before p. Using this fairness condition the property can be proven for the abstract system and hence holds also for the concrete one. □

7 Examples

In this section we present some more complicated examples to prove applicability of our approach to a broader class of algorithms. First we prove some properties for a version of Szymanski's mutual exclusion algorithm proposed by Amir Pnueli. To our knowledge this is the first time that also individual liveness is automatically proven for this algorithm. The second example, a mutual exclusion algorithm inspired by Dijkstra's algorithm, strongly relies on a lot of fairness conditions that have to be added to the abstract system to be able to prove individual accessibility there. Whereas the mutual exclusion algorithms are naturally assumed to run in an asynchronous manner, the third example is a time-triggered group membership algorithm.

7.1 Szymanski's Mutual Exclusion Algorithm

We first give a pseudo code characterization of the algorithm. The translation to a WS1S system is straightforward and can be seen in detail in [BBLS00].

The code for one process of Szymanski's mutual exclusion algorithm is given as:

ℓ_0: $\langle$ noncritical section $\rangle$
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: $\langle$ entry $\rangle$
ℓ_3: **if** $\exists_n\, j : \mathrm{at\_}\ell_1[j] \vee \mathrm{at\_}\ell_2[j]$
  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** $\forall_n\, j : \neg(\mathrm{at\_}\ell_3[j] \vee \mathrm{at\_}\ell_4[j])$
ℓ_6: **await** $\forall_n\, j : j < i : \mathrm{at\_}\ell_0[j] \vee \mathrm{at\_}\ell_1[j] \vee \mathrm{at\_}\ell_2[j]$
ℓ_7: $\langle$ critical section $\rangle$; **goto** ℓ_0

The control flow is modeled by a boolean variable $\mathrm{at\_}\ell_k[p]$ for each location ℓ_k and each process p. The initial condition states that each process starts in ℓ_0.

We would like to prove that for each process p, $\Box(\mathrm{at\_}\ell_7[p] \Rightarrow \Diamond \mathrm{at\_}\ell_7[p])$ holds, i.e., each process that wants to enter the critical section eventually does so.

The skip transition from ℓ_1 to ℓ_2 results from the translation of the algorithm into our formalism. The transition should not be omitted, since otherwise the mutual exclusion is trivially fulfilled. It represents the entry to the waiting room where all processes gather to enter their critical section.

The abstraction focuses on two arbitrary processes $a < b$ with abstract variables $\ell_1^a, \ell_1^b, \ell_2^a, \ell_2^b, \ldots$ expressing the actual location of the processes. This kind of abstraction allows to analyze individual properties for these arbitrary processes.

The translation of the algorithm to a WS1S system introduces a set variable $\mathrm{At\_}\ell_k$ for each location ℓ_k. Using those sets as ranking functions $\chi(i, \mathrm{At\_}\ell_k) \equiv i \in \mathrm{At\_}\ell_k$ we can use the marking algorithm to generate fairness conditions for the abstract system. With all the *taken* variables $tk_{ll'}$ the fairness due to the marking algorithm can be expressed as an LTL formula *fair*:

$$\begin{array}{l}(\Box\Diamond tk_{01} \Rightarrow \Box\Diamond tk_{70}) \land (\Box\Diamond tk_{12} \Rightarrow \Box\Diamond tk_{01}) \land \\ (\Box\Diamond tk_{23} \Rightarrow \Box\Diamond tk_{12}) \land (\Box\Diamond tk_{34} \Rightarrow \Box\Diamond tk_{23}) \land \\ (\Box\Diamond tk_{35} \Rightarrow \Box\Diamond tk_{23}) \land (\Box\Diamond tk_{45} \Rightarrow \Box\Diamond tk_{34}) \land \\ (\Box\Diamond tk_{56} \Rightarrow \Box\Diamond (tk_{45} \lor tk_{35})) \land \\ (\Box\Diamond tk_{67} \Rightarrow \Box\Diamond tk_{56}) \land (\Box\Diamond tk_{70} \Rightarrow \Box\Diamond tk_{67})\end{array}$$

With this fairness condition some nice properties can be proven to hold for the abstract system:

Safety: $\Box(\neg(\ell_7^a \land \ell_7^b))$

Individual accessibility: $\mathit{fair} \Rightarrow \Box(\ell_1^b \Rightarrow \Diamond \ell_7^b)$

Competition: If 2 processes compete the one with the smaller ID wins:

$$\mathit{fair} \Rightarrow \Box(\ell_1^b \land \ell_1^a \Rightarrow ((\neg \ell_7^b)\,\mathcal{U}\,\ell_7^a))$$

holds, but

$$\mathit{fair} \Rightarrow \Box(\ell_1^b \land (\ell_0^a \lor \ell_1^a) \Rightarrow ((\neg \ell_7^a)\,\mathcal{U}\,\ell_7^b))$$

does not hold for the abstract system.

In general, we have bounded overtaking:

$$\mathit{fair} \Rightarrow \Box(\ell_1^b \land (\ell_0^a \lor \ell_1^a) \Rightarrow ((((\neg \ell_7^a)\,\mathcal{U}\,\ell_7^a)\,\mathcal{U}\,(\neg \ell_7^a))\,\mathcal{U}\,\ell_7^b))$$

Therefore, as shown in Sections 5 and 6, the corresponding properties hold in the original protocol.

7.2 Dijkstra's Mutual Exclusion Algorithm

Consider the following version of a mutual exclusion algorithm inspired by Dijkstra's algorithm, where each process $S(i, n)$ is given in Figure 1. In this description, we have one single global variable *turn* that ranges over natural numbers. When the algorithm is translated into WS1S this variable is translated identically.

Note that the control flow is again modeled with boolean variables $\mathrm{at\_}\ell_i$. In the translation to WS1S systems this introduces five set variables $\mathrm{At\_}\ell_0, \ldots,$

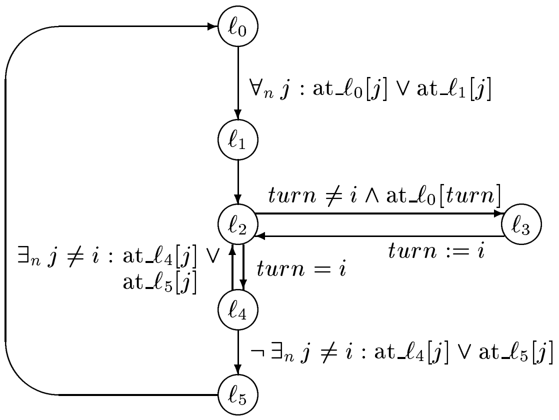


Figure 1: Dijkstra's mutual exclusion algorithm

At_ℓ_5. As one and only non-boolean variable the $turn$ variable ranging over the domain of involved process indices may be read and written by all processes.

Initially, all processes start at ℓ_0, and the $turn$ variable has an arbitrary value.

In fact, the algorithm is not the original algorithm from Dijkstra since Dijkstra's algorithm does not satisfy the individual accessibility property, which we like to prove on our algorithm. To achieve individual accessibility we block processes in ℓ_0 if the 'waiting room' ℓ_2, ℓ_3, ℓ_4 is non-empty. This causes an interesting behavior in the waiting room. And this is what we are aiming for in this example; to show how to deal with different kinds of fairness requirements.

For the generic abstraction α_p we also monitor the behavior of the current $turn$ process, i.e., the process having the turn:

$$\begin{array}{ll}
\phi_i \equiv turn \in \mathrm{At_}\ell_i \quad \text{for } 0 \leq i \leq 5 & \xi_2 \equiv \exists_P\, k : \mathrm{At_}\ell_5 \subseteq \{k\} \\
\psi_i \equiv p \in \mathrm{At_}\ell_i \quad \text{for } 0 \leq i \leq 5 & \gamma \equiv p = turn \\
\xi_1 \equiv \mathrm{At_}\ell_5 \neq \emptyset & \delta \equiv \mathrm{At_}\ell_4 \setminus \{turn\} \neq \emptyset
\end{array}$$

We leave out here the abstract variables monitoring whether p or $turn$ takes a transition, but they are added to the variables given.

Using NuSMV we can prove the following LTL formula to be correct:

$$(\Diamond\Box\xi_1 \Rightarrow \Box\Diamond tk_{50}) \ \wedge \tag{4}$$

$$(\Diamond\Box\delta \Rightarrow \Box\Diamond(tk^o_{42} \vee tk^o_{45})) \ \wedge \tag{5}$$

$$(\Diamond\Box\psi_1 \Rightarrow \Box\Diamond\psi_2) \ \wedge \tag{6}$$

$$(\Box\Diamond tk_{12} \Rightarrow \Box\Diamond tk_{50}) \ \wedge \ (\Box\Diamond tk_{45} \Rightarrow \Box\Diamond tk_{12}) \wedge \tag{7}$$

$$(\Box\Diamond tk_{01} \Rightarrow \Box\Diamond tk_{50}) \ \wedge \ (\Box\Diamond tk_{32} \Rightarrow \Box\Diamond tk_{23}) \wedge \tag{8}$$

$$(\Box\Diamond tk_{42} \Rightarrow \Box\Diamond tk_{24}) \ \wedge \ (\Box\Diamond tk^o_{42} \Rightarrow \Box\Diamond tk_{32}) \wedge \tag{9}$$

$$(\Box\Diamond tk^o_{23} \Rightarrow \Box\Diamond(tk_{32} \ \vee \ tk_{42} \vee tk_{12})) \tag{10}$$

$$\Rightarrow \Box(\psi_1 \Rightarrow \Diamond\psi_5) \tag{11}$$

Fairness condition (4) is due to the requirement that the critical section has to be left eventually. The weak fairness condition on the concrete level is equivalent to a strong fairness condition, since the transition can only be disabled by taking it.

Condition (5) expresses the strong fairness for processes unequal $turn$ for taking a transition leaving ℓ_4. It is the combination of two strong fairness conditions (for transition ℓ_4 to ℓ_2 resp. ℓ_5).

(6) is a weak fairness condition for process p lifted from the concrete level. Conditions (7) to (10) are generated by the marking algorithm. For example, (7) is the result of the algorithm for the set $\mathrm{At\_}\ell_2 \cup \mathrm{At\_}\ell_3 \cup \mathrm{At\_}\ell_4$, and (9) is the result for the set $(\mathrm{At\_}\ell_4 \setminus \{turn\}) \cup (\mathrm{At\_}\ell_2 \cap \{turn\})$.

Checking the mutual exclusion property, i.e., the formula $\Box\xi_2$, takes 3.6 sec, while checking the liveness property takes 1100 sec on a Sun Ultra 5/10 UPA/PCI (UltraSPARC-IIi 440MHz) with 1GB of memory.

7.3 A Time-Triggered Group Membership Protocol

The protocol was presented by S. Katz, P. Lincoln, and J. Rushby in [KLR97]. The protocol requires one bit of membership information piggybacked on regular broadcasts. With assumptions on the fault model, the protocol guarantees that a faulty processor will be diagnosed and removed from the agreed group of non-faulty processors. Therefore, each processor p keeps a set of processor IDs $\mathrm{mem}(p)$ that he believes to be non-faulty.

If $p \in \mathrm{mem}(p)$, then process p takes one of the following transitions synchronously with the other processes.

$$
\begin{aligned}
\neg\mathrm{arrived}(p) \wedge \neg\mathrm{ack}(p) &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p) \setminus \{turn, p\};\\
&\phantom{\rightarrow{}} \mathrm{ack}'(p) := \mathit{ff}\\
\neg\mathrm{arrived}(p) \wedge \mathrm{ack}(p) &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p) \setminus \{turn\};\\
&\phantom{\rightarrow{}} \mathrm{ack}'(p) := \mathit{ff}\\
\mathrm{arrived}(p) \wedge \mathrm{ack}(turn) \wedge \neg\mathrm{ack}(p) &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p) \setminus \{p\};\\
&\phantom{\rightarrow{}} \mathrm{ack}'(p) := \mathit{tt}\\
\mathrm{arrived}(p) \wedge \neg\mathrm{ack}(turn) \wedge \mathrm{ack}(p) &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p) \setminus \{turn\};\\
&\phantom{\rightarrow{}} \mathrm{ack}'(p) := \mathit{ff}\\
\mathrm{arrived}(p) \wedge \mathrm{ack}(turn) \wedge \mathrm{ack}(p) &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p); \mathrm{ack}'(p) := \mathit{tt}\\
\mathrm{arrived}(p) \wedge \neg\mathrm{ack}(turn) \wedge \neg\mathrm{ack}(p) &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p); \mathrm{ack}'(p) := \mathit{tt}\\
p = turn &\rightarrow \mathrm{mem}'(p) := \mathrm{mem}(p); \mathrm{ack}'(p) := \mathit{tt}
\end{aligned}
$$

It is process $turn$ which has to send at the moment. $\mathrm{arrived}(p)$ stands for: $turn \neq p$, p is not receive-faulty, and process $turn$ is not send-faulty at this step. The $turn$ variable is increased by 1 modulo n in each step.

If $p \notin \mathrm{mem}(p)$, process p does not send. The fault assumption is that new faults arrive at least $n + 1$ time units apart, when there are n processes. A fault may be that a processor is unable to send or to receive. Let us denote with OK the set of non-faulty processors. Then, we like to prove the following properties:

Agreement: $p \in OK \wedge q \in OK \Rightarrow \mathrm{mem}(p) = \mathrm{mem}(q)$

Validity: $p \in OK \Rightarrow \exists q : \mathrm{mem}(p) \cup \{q\} = OK$

Since p and q are arbitrary processes both properties are universal as defined in Section 5.

7.3.1 Generic Abstractions of Time-Triggered Systems

Our goal is to characterize such time-triggered systems as WS1S systems. Then, we can apply the presented automatic abstraction and model-checking techniques to analyze the system. To be able to do so each process of the system has to be finite state. This is not the case here, since each process stores the set $\mathrm{mem}(p)$. But, the properties only talk about the memory of one or two processes. Hence, the idea is to make an abstraction concentrating on these processes and give a finite-state abstraction for the others.

For the first property we keep two original processes p, q and leave the others as chaotic processes. Nevertheless, we prove that p, q agree on the set of non-faulty processes in every synchronous step. The definition of the abstract system is straightforward and not presented here.

For the property of validity we concentrate on one process p. Since, we already have proven agreement the others may use p's memory Mem_p as their own.

Our generic abstraction gives us an parameterized number of finite-state processes and a finite number of processes with unbounded state space. Since all finite-state processes are similar we introduce for each of the possible states a set variable holding those process IDs which are in the corresponding state. The remaining ones are modeled as they are.

In each computation step of the abstract time-triggered systems one process is broadcasting and all others are receiving this message. The broadcasting process is denoted by the special variable *turn* which is increased by 1 modulo n in each step.

The fault model can be encoded in these WS1S transitions using the variables *s_fault*, *r_fault*, *err_prop* which characterize whether a send or receive error has occurred and whether a new error is allowed yet or not. The variable *malfunc* holds the ID of the faulty processor.

For the abstract system and the property of validity we define our abstract system as shown in Table 1.

Using this abstraction we can automatically construct the abstract system with our tool PAX. Translating the abstract system to the SMV input language, we can use NuSMV to prove the following properties:

$$\begin{aligned}
&\Box(p_ok \wedge \neg r_fault) \Rightarrow \\
&\quad \Box(s_fault \wedge \bigcirc((\neg s_fault \wedge \neg \bigcirc s_fault)\ \mathcal{U}\ t_mem) \\
&\qquad \Rightarrow \Diamond(equal \wedge stable))
\end{aligned}$$

and

$$\begin{aligned}
&\Box(p_ok \wedge \neg s_fault \wedge (r_fault \Rightarrow \bigcirc \Box(\neg r_fault))) \Rightarrow \\
&\quad \Box(r_fault \wedge \Diamond(t_mal \wedge \Diamond \bigcirc t_mem) \Rightarrow \\
&\qquad \Diamond(equal \wedge stable))
\end{aligned}$$

Table 1: Definition of the abstract system

abstract variable	description
$p\_ok \equiv p \in OK$	process we focus on is non-faulty
$t\_ack \equiv turn \in \mathrm{Ack}$	turn process believes everything is ok
$t\_mal \equiv turn = malfunc$	turn process is faulty
$t\_mem \equiv turn \in \mathrm{Mem}_p$	p believes the *turn* process to be ok
$conf \equiv \mathrm{Ack} \cap \mathrm{Mem}_p = \emptyset$	non-faulty procs agree there is something wrong
$equal \equiv \mathrm{Mem}_p = OK$	non-faulty procs have correct membership set
$stable \equiv \mathrm{Mem}_p \subseteq \mathrm{Ack}$	everything is fine
$super \equiv (\exists q : \mathrm{Mem}_p = OK \cup \{q\} \wedge \mathrm{Mem}_p \subseteq \mathrm{Ack} \cup \{q\} \wedge q = malfunc)$	one faulty process not registered by the non-faulty ones
s_fault	send error occurred
r_fault	receive error occurred
err_prop	fault model allows no new error yet

The first one states under the assumption that the process we focus on is ok and in absence of receive errors; whenever a send error occurs all non-faulty processes eventually notice that and remove the faulty process from their membership set, provided that no new errors occur. The second one expresses a corresponding behavior for receive errors.

8 Conclusion

We presented a method for the verification of universal properties of parameterized networks. 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. To be able to prove liveness properties we presented a method to add fairness requirements to the abstract system. We have successfully applied this method, which has been implemented in our tool PAX, to a number of parameterized protocols.

References

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

[BBLS00] K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In S. Graf and M. Schwartzbach, editors, *TACAS'00*, volume 1785. Springer, 2000.

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

[BLS00] K. Baukus, Y. Lakhnech, and K. Stahl. Verifying Universal Properties of Parameterized Networks. Technical Report TR-ST-00-4, CAU Kiel, 2000.

[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.

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

[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.

[KLR97] Shmuel Katz, Pat Lincoln, and John Rushby. Low-overhead time-triggered group membership. In Marios Mavronicolas and Philippas Tsigas, editors, *11th International Workshop on Distributed Algorithms (WDAG '97)*, volume 1320 of *Lecture Notes in Computer Science*, pages 155–169, Saarbrücken Germany, September 1997. Springer-Verlag.

[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.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In O. Grumberg, editor, *Proceedings of CAV '97*, volume 1256 of *LNCS*, pages 424–435. Springer, 1997.

[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.

[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.

[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.