Verification of Parametric Concurrent Systems with Prioritized FIFO Resource Management

Ahmed Bouajjani, Peter Habermehl, Tomas Vojnar

Verification of Parametric Concurrent Systems with Prioritized FIFO Resource Management*

Ahmed Bouajjani, Peter Habermehl, and Tomáš Vojnar

LIAFA, Paris University 7
Case 7014, 2, place Jussieu, 75251 Paris Cedex 05, France
e-mail: {abou,haberm,vojnar}@liafa.jussieu.fr

Abstract. We consider the problem of parametric verification over a class of systems of processes competing for access to shared resources. We suppose the access to the resources to be controlled according to a FIFO-based policy with a possibility of distinguishing low-priority and high-priority resource requests. We propose a model of the concerned systems based on extended automata with queues. Over this model, we address verification of properties expressed in LTL$\setminus$X enriched with global process quantification and interpreted on finite as well as fair behaviours of the given systems. In addition, we examine parametric verification of process deadlockability too. By reducing the parametric verification problems to finite-state model checking, we establish several decidability results for different classes of the considered properties and systems (including the special case of systems with the pure FIFO resource management). Moreover, we show that parametric verification against formulae with local process quantification is undecidable in the given context.

1 Introduction

Managing concurrent access to shared resources is a fundamental problem that appears in many contexts, e.g., operating systems, multithreaded programs, control software, etc. The critical properties to ensure are typically (1) mutual exclusion when exclusive access is required, (2) absence of starvation (a process that requires a resource will eventually get it), and (3) absence of deadlocks. Many different instances of this problem can be defined depending on the assumptions on the allowed actions for access to resources and the policies for managing the access to these resources.

In this work, we consider systems with a finite number of resources shared by a set of identical processes. These processes can require a set of resources, get access and use the requested resources, and release the used resources. The requests can be of a low-priority or a high-priority level. The access to the resources is managed by a locker according to a FIFO-based policy taking into account the priorities of the requests, i.e. a waiting high-priority request can overtake waiting low-priority ones. As a special case allowing for an optimized treatment, we then examine the situation when no high-priority requests are used, and the locker behaves according to the pure FIFO discipline.

As mentioned later in related work, the above framework is, in particular, inspired by a need to verify the use of shared resources in some of Ericsson's ATM switches. However, the operations for access to shared resources and the resource management policies used are quite natural in general in concurrent applications dealing with shared resources.

* This work was supported in part by the European Commission (FET project ADVANCE, contract No. IST-1999-29082).

Verification of the described systems can, of course, be carried out using finite-state model-checking if we fix the number of processes. However, a precise number of processes present in such a system in practice is usually not known in advance, and it is thus crucial to verify that the system behaves correctly for *any* number of them. This yields a parametric verification problem that is quite nontrivial as we have to deal with an infinite number of system instances.

The aim of this paper is to study decidability of the described problem for a significant class of properties including the three most important ones given above.

For an abstract description of the concerned systems, we define a model based on extended automata with queues recording the identities of the waiting processes for each resource. Then, we address the verification problem for families of such systems with an arbitrary number of processes (called *RTR families*—RTR stands for request-take-release) against formulae of the temporal logic $LTL\backslash X$ with *global process quantification*. We consider two interpretation domains for the logic: the set of finite behaviours (which is natural for safety properties), and the set of fair behaviours (in order to cover liveness properties). In addition, we consider the parametric verification problem of process deadlockability too.

We adopt the approach of finding *cut-off bounds* to show that many interesting parametric verification problems in the given context can be reduced to finite-state model checking. This means that given a class of formulae, we prove that deciding whether all systems of a family satisfy a formula is equivalent to deciding whether some finite number of systems in the family (each of them having a fixed number of processes) satisfies this formula.

When establishing our results, we consider the question whether it is possible to find cut-off bounds that do not depend on the structure of the involved processes and the formula at hand, but only on the number of resources and the number of processes quantified in the formula. Indeed, these numbers are relatively small, especially in comparison to the size of process control automata.

We show that for RTR families where the *pure FIFO resource management* is used (i.e. no high-priority access to resources is required), parametric verification of finite as well as fair behaviour is decidable against all $\mathrm{LTL}\backslash\mathrm{X}$ formulae with global process quantification. The cut-off bound in the finite behaviour case is the number of quantified processes, whereas it is this number plus the number of resources in the fair behaviour case. These bounds lead to practical finite-state verification. Furthermore, we show that the verification of process deadlockability is decidable too (where the bound is the number of resources).

On the other hand, for the case of dealing with RTR families that *distinguish low-priority and high-priority requests*, we show that—unfortunately—general, structure-independent cut-offs do not exist neither for the interpretation of the considered logic on finite nor fair behaviours. However, we show that even for such families, parametric verification of finite behaviour is decidable, e.g., against reachability/invariance formulae, and parametric verification of fair behaviour is decidable against formulae with a single quantified process. In this way, we cover, e.g., verification of the (for the given application domain) key properties of mutual exclusion and absence of starvation. For the former case, we even obtain a structure-independent cut-off equal again to the number of quantified processes. For verification of fair behaviour against single process formu-

lae, no general structure-independent cut-off can be found, but we provide a structure-dependent one, and in addition, we determine a significant subclass of RTR families where a structure-independent cut-off for this particular kind of properties does exist. Finally, we show that process deadlockability can be solved in the case of general RTR families via the same (structure-independent) cut-off as in the case of the families not using high-priority requests.

Lastly, we show that although the queues in RTR families are not communication queues, but just waiting queues, and the above decidability results may be established, the model is still quite powerful, and decidability may easily be lost when trying to deal with a bit more complex properties to verify. We illustrate this by proving that parametric finite-behaviour verification becomes *undecidable* (even for families not using high-priority requests) for LTL$\setminus$X extended with the notion of *local process quantification* [8] allowing one to examine different processes in different encountered states.

Related Work: There exist several approaches to the parametric verification problem. We can mention, for example, the use of symbolic model checking, (automated) abstraction, or network invariants [10, 1, 3, 14, 11, 12]. The idea of cut-offs has already been used in several contexts [9, 6, 7, 5] too. However, to the best of our knowledge, there is no work covering the class of parametric systems considered here, i.e. parametric resource sharing systems with a prioritized FIFO resource management. The *two* involved obstacles (parameterization and having multiple queues over an unbounded domain of process identifiers) seem to complicate the use of any of the known methods. Using cut-offs appears to be the easiest approach here.

The work [5] targets verification of systems with shared resources (and even employs cut-offs), but the system model and the employed proof techniques differ. The involved processes need not be identical, the number of resources is not bounded, but, on the other hand, only two fixed processes may compete for a given resource, and their requests are served in random order (there are no FIFO queues in [5]). Moreover, some of the properties to be verified we consider here are different from [5] (e.g., we deal with the more realistic notion of weak/strong fairness compared to the unconditional one used there, etc.).

Finally, let us add that our work was originally motivated by [2] that concerns verification of the use of shared resources in Ericsson's AXD 301 ATM switch. In [2] finite-state model checking is used to verify some isolated instances of the given parametric system. Then, in the concluding remarks, a need for a more complete, parametric verification is mentioned, which is what our work is aiming at on the level of a general abstract model covering the given application domain.

Outline: We first formalize the notion of RTR families and define the specification logic we use. Then, we present our cut-off results for finite and fair behaviour and process deadlockability as well as the undecidability result. Due to space limitations, we provide proof ideas for some of the results only—the complete proofs can be found in [4].

2 RTR Families

2.1 The Model of RTR Families

Processes in systems of RTR families are controlled by *RTR automata*. An RTR automaton over a finite set of resources is a finite automaton with the following kinds

of actions joint with transitions: skip (denoted by τ—an abstract step not changing resource utilization), request and, when it is the turn, take a set of resources at the low- or high-priority level (`rqt`/`prqt`), and, finally, release a set of resources (`rel`).

Let us, however, stress that we allow processes to block inside $(\mathtt{p})\mathtt{rqt}$ transitions[1] while waiting for the requested resources to be available for them. Therefore, a single $(\mathtt{p})\mathtt{rqt}$ transition in a model semantically corresponds to two transitions, which we denote as $(\mathtt{p})\mathtt{req}$ (request a set of resources) and $(\mathtt{p})\mathtt{take}$ (start using the requested resources when enabled to do so by the locking policy).

Definition 1. *An* RTR automaton *is a 4-tuple $\mathcal{A} = (R, Q, q_0, T)$ where R is a set of resources, Q is a set of control locations, $q_0 \in Q$ is an initial control location, and $T \subseteq Q \times A \times Q$ is a transition relation over the set of actions $A = \{\tau\} \cup \{a(R') \mid a \in \{\mathtt{rqt}, \mathtt{prqt}, \mathtt{rel}\} \wedge R' \neq \emptyset \wedge R' \subseteq R\}$. The sets R, Q, T, and A are nonempty, finite, pairwise disjoint, and disjoint with $\mathbb{N}$.*

An *RTR family* $\mathcal{F}(\mathcal{A})$ over an RTR automaton $\mathcal{A}$ is a set of *systems* S_n consisting of $n \geq 1$ identical processes controlled by $\mathcal{A}$ and identified by elements of $P_n = \{1, \ldots, n\}$. (In the following, if no confusion is possible, we usually drop the reference to $\mathcal{A}$.) We denote as *RTR\P families* the special cases of RTR families whose control automata contain no high-priority request actions.

2.2 Configurations

For the rest of the section, let us suppose working with an arbitrary fixed RTR family $\mathcal{F}$ over an automaton $\mathcal{A} = (R, Q, q_0, T)$ and with a system $S_n \in \mathcal{F}$.

To make the semantics of RTR families reflect the fact that processes may block in $(\mathtt{p})\mathtt{rqt}$ actions, we extend the set Q of "explicit" control locations to Q_t containing a unique internal control location q_t for each transition $t \in T$ based on a $(\mathtt{p})\mathtt{rqt}$ action. Furthermore, let T_t be the set obtained from T by preserving all τ and `rel` transitions and splitting each transition $t = (q_1, (\mathtt{p})\mathtt{rqt}(R'), q_2) \in T$ to two transitions $t_1 = (q_1, (\mathtt{p})\mathtt{req}(R'), q_t)$ and $t_2 = (q_t, (\mathtt{p})\mathtt{take}(R'), q_2)$.

We define the *resource queue alphabet* of S_n as $\Sigma_n = \{s(p) \mid s \in \{\mathtt{r}, \mathtt{pr}, \mathtt{g}, \mathtt{u}\} \wedge p \in P_n\}$. The meaning is that a process has requested a resource in the low- or high-priority way, it has been granted the resource, or it is already using the resource. A *configuration* c of S_n is then a function $c : (P_n \to Q_t) \cup (R \to \Sigma_n^*)$ that assigns the current control locations to processes and the current content of queues of requests to resources. Let C_n be the set of all such configurations.

2.3 Resource Granting and Transition Firing

We now introduce the *locker function* Λ implementing the considered FIFO resource management policy with low- and high-priority requests. This function is to be applied over configurations changed by adding/removing some requests to/from some queues in order to grant all the requests that can be granted wrt. the given strategy in the given situation. Note that in the case of RTR\P families, the resource management policy can be considered the pure FIFO policy.

A high-priority request is granted iff none of the needed resources is in use by or granted to any process, nor it is subject to any sooner raised, but not yet granted,

[1] We use $(\mathtt{p})\mathtt{rqt}$ when addressing both `rqt` as well as `prqt` transitions.

high-priority request. A low-priority request is granted iff the needed resources are not in use nor granted and they are not subject to any sooner raised request nor any later raised high-priority request that can be granted at the given moment. (High-priority requests that currently cannot be granted do not block sooner raised low-priority requests.) Formally, for $c \in C_n$, we define $\Lambda(c)$ to be a configuration of G_n equal to c up to the following for each $r \in R$:

1. If $c(r) = w_1.\mathtt{pr}(p).w_2$ for some $p \in P_n$, $w_1, w_2 \in \Sigma_n^*$ s.t. $c(p) = q_t$ for a certain $t = (q_1, \mathtt{prqt}(R'), q_2) \in T$ and for all $r' \in R'$, $c(r') = w_1'.\mathtt{pr}(p).w_2'$ with $w_1' \in \{\mathtt{r}(p') \mid p' \in P_n\}^*$ and $w_2' \in \Sigma_n^*$, we set $\Lambda(c)(r)$ to $\mathrm{g}(p).w_1.w_2$.
2. If $c(r) = \mathtt{r}(p).w$ for some $p \in P_n$, $w \in \Sigma_n^*$ such that $c(p) = q_t$ for a certain $t = (q_1, \mathtt{rqt}(R'), q_2) \in T$ and for all $r' \in R'$, $c(r') = \mathtt{r}(p).w'$ with $w' \in \Sigma_n^*$, and the premise of case 1 is not satisfied for r', we set $\Lambda(c)(r)$ to $\mathrm{g}(p).w$.

We define *enabling* and *firing of transitions* in processes of S_n via a predicate $en \subseteq C_n \times T_t \times P_n$ and a function $to : C_n \times T_t \times P_n \to C_n$.

For all transitions $t = (q_1, \tau, q_2) \in T_t$ and $t = (q_1, a(R'), q_2) \in T_t$, $a \in \{\mathtt{rel}, \mathtt{req}, \mathtt{preq}\}$, we define $en(c,t,p) \Leftrightarrow c(p) = q_1$. For each transition $t = (q_1, (\mathrm{p})\mathtt{take}(R'), q_2) \in T_t$, we define $en(c,t,p) \Leftrightarrow c(p) = q_1 \wedge \forall r \in R'\ \exists w \in \Sigma_n^* : c(r) = \mathrm{g}(p).w$. Intuitively, a transition is enabled in some process if the process is at the source control location of the transition and, in the case of $(\mathrm{p})\mathtt{take}$, if the appropriate request has been granted.

Firing of a transition $t = (q_1, \tau, q_2) \in T_t$ simply changes the control location mapping of p from q_1 to q_2, i.e. $to(c,t,p) = (c \setminus \{(p,q_1)\}) \cup \{(p,q_2)\}$.

Firing of a $(\mathrm{p})\mathtt{req}$ transition t corresponds to registering the request in the queues of all the involved resources and going to the internal waiting location of t. The locker is applied to (if possible) immediately grant the request. For $t = (q_1, (\mathrm{p})\mathtt{req}(R'), q_2) \in T_t$, we define $to(c,t,p) = \Lambda((c \setminus c^-) \cup c^+)$ where $c^- = \{(p,q_1)\} \cup \{(r, c(r)) \mid r \in R'\}$ and $c^+ = \{(p,q_2)\} \cup \{(r, c(r).(\mathrm{p})\mathtt{r}(p)) \mid r \in R'\}$.

For a transition $t = (q_1, (\mathrm{p})\mathtt{take}(R'), q_2) \in T_t$, we simply change all the appropriate g queue items to $\mathtt{u}$ items and finish the concerned $(\mathrm{p})\mathtt{rqt}$ transition, i.e. $to(c,t,p) = (c \setminus c^-) \cup c^+$ with c^- as in the case of $(\mathrm{p})\mathtt{req}$ and $c^+ = \{(p,q_2)\} \cup \{(r, \mathtt{u}(p).w) \mid r \in R' \wedge c(r) = \mathrm{g}(p).w\}$.

Finally, a $\mathtt{rel}$ transition removes the head $\mathtt{u}$ items from the queues of the given resources provided they are owned by the given process. The locker is applied to grant all the requests that may become unblocked. Formally, for $t = (q_1, \mathtt{rel}(R'), q_2) \in T_t$, we fix $to(c,t,p) = \Lambda((c \setminus c^-) \cup c^+)$ with $c^- = \{(p,q_1)\} \cup \{(r, c(r)) \mid r \in R' \wedge \exists w \in \Sigma_n^* : c(r) = \mathtt{u}(p).w\}$ and $c^+ = \{(p,q_2)\} \cup \{(r,w) \mid r \in R' \wedge w \in \Sigma_n^* \wedge c(r) = \mathtt{u}(p).w\}$.

2.4 Behaviour of Systems of RTR Families

Let S_n be a system of an RTR family $\mathcal{F}$. We define the *initial configuration* c_0 of S_n to be such that $\forall p \in P_n : c_0(p) = q_0$ and $\forall r \in R : c_0(r) = \varepsilon$. By a *finite behaviour* of S_n starting from $c_1 \in C_n$, we understand a sequence $c_1(p_1,t_1)c_2...(p_l,t_l)c_{l+1}$ such that for each $i \in \{1,...,l\}$, $en(c_i,t_i,p_i)$ holds, and $c_{i+1} = to(c_i,t_i,p_i)$. If c_1 is the initial configuration c_0, we may drop a reference to it and speak simply about a finite behaviour of S_n. The notion of *infinite behaviours* of S_n can be defined in an analogous way. A *complete behaviour* is then either infinite or such that it cannot be extended any more.

We say a complete behaviour is *weakly (process) fair* iff each process that is eventually always enabled to fire some transitions, always eventually fires some transitions. We may call a complete behaviour *strongly (process) fair* iff each process that is always eventually enabled to fire some transitions, always eventually fires some transitions. However, we do not deal with strong fairness in the following as in our model, the notions of strong and weak fairness coincide: Due to the separation of requesting resources and starting to use them and the impossibility of cancelling issued grants of resources, a process cannot temporarily have no enabled transitions without firing anything.

For a behaviour $\beta_n = c_1(p_1,t_1)c_2(p_2,t_2)...$ of a system S_n of an RTR family $\mathcal{F}$, we call the configuration sequence $\pi_n = c_1c_2...$ a *path* of S_n corresponding to β_n and the transition firing sequence $\rho_n = (p_1,t_1)(p_2,t_2)...$ a *run* of S_n corresponding to β_n. If the behaviour is not important, we do not mention it. We denote Π_n^{fin}, $\Pi_n^{fin} \subseteq C_n^+$, the set of all finite paths of S_n and Π_n^{wf}, $\Pi_n^{wf} \subseteq C_n^+ \cup C_n^\omega$, the set of all paths of S_n corresponding to complete, weakly fair behaviours.

3 The Specification Logic

In this work, we concentrate (with the exception of process deadlockability) on verification of process-oriented, linear-time properties of systems of RTR families. For specifying the properties, we use the below described extension of LTL\X, which we denote as *MPTL* (i.e. temporal logic of many processes). We exclude the next-time operator from our framework because it sometimes allows a certain kind of counting of processes, which is undesirable when trying to limit/reduce the number of processes to be considered in verification.

We extend LTL\X by *global process quantification* in a way inspired by ICTL* (see, e.g., [8]) and allowing us to easily reason over systems composed of a parametric number of identical processes. We also allow for an explicit distinction whether a property should hold for all paths or for at least one path out of a given set. Therefore, we introduce a single top-level *path quantifier* to our formulae. We restrict quantification in the following way: (1) We implicitly require all variables to always refer to distinct processes. (2) We allow only uniformly universal (or uniformly existential) process and path quantification.

Finally, we limit *atomic formulae* to testing the current control locations of processes. We allow for referring to the internal control locations of request transitions too, which corresponds to asking whether a process has requested some resources, but has not become their user yet.

3.1 The Syntax of MPTL

Let PV, $PV \cap \mathbb{N} = \emptyset$, be a set of process variables. We first define the syntax of *MPTL path subformulae*, which we build from atomic formulae $at(p,q)$ using boolean connectives and the until operator. For $V \subseteq PV$ and $p \in V$, we have:

$$\varphi(V) ::= at(p,q) \mid \neg\varphi(V) \mid \varphi(V) \vee \varphi(V) \mid \varphi(V)\ \mathcal{U}\ \varphi(V)$$

As syntactical sugar, we can then introduce in the usual way formulae like `tt`, `ff`, $\varphi(V) \wedge \varphi(V)$, $\Box\varphi(V)$, or $\Diamond\varphi(V)$.

Subsequently, we define the syntax of *universal* and *existential MPTL formulae*, which extend MPTL path subformulae by process and path quantification used in a

uniformly universal or existential way. For $V \subseteq PV$, we have $\Phi_a ::= \forall V : A\ \varphi(V)$ and $\Phi_e ::= \exists V : E\ \varphi(V)$.

In the rest of the paper, we commonly specify sets of quantified variables by listing their elements in some chosen order. Using MPTL formulae, we can then express, for example, *mutual exclusion* as $\forall p_1, p_2 : A\ \Box\ \neg(at(p_1, cs) \wedge at(p_2, cs))$ or *absence of starvation* as $\forall p : A\ \Box\ (at(p, req) \Rightarrow \Diamond\ at(p, use))$.

3.2 The Formal Semantics of MPTL

Suppose working with a set of process variables PV. As we require process quantifiers to always speak about distinct processes, we call a function $\nu_n : PV \to P_n$ a *valuation* of PV iff it is an injection.

Suppose further that we have a system S_n of an RTR family $\mathcal{F}$. Let $\pi_n \in C_n^* \cup C_n^\omega$ denote a (finite or infinite) path of S_n. For a finite (or infinite) path $\pi_n = c_1 c_2 ... c_{|\pi_n|}$ ($\pi_n = c_1 c_2 ...$), let π_n^l denote the suffix $q c_{l+1} ... c_{|\pi_n|}$ ($c_l c_{l+1} ...$) of π_n, respectively. (For a finite π_n with $|\pi_n| < l$, $\pi_n^l = \varepsilon$.) Given a path π_n of S_n and a valuation ν_n of PV, we inductively define the semantics of MPTL path subformulae $\varphi(V)$ as follows:

- $\pi_n, \nu_n \models at(p, q)$ iff $\pi_n = c.\pi_n'$ and $c(\nu_n(p)) = q$.
- $\pi_n, \nu_n \models \neg\varphi(V)$ iff $\pi_n, \nu_n \not\models \varphi(V)$.
- $\pi_n, \nu_n \models \varphi_1(V) \vee \varphi_2(V)$ iff $\pi_n, \nu_n \models \varphi_1(V)$ or $\pi_n, \nu_n \models \varphi_2(V)$.
- $\pi_n, \nu_n \models \varphi_1(V)\ \mathcal{U}\ \varphi_2(V)$ iff there is $l \geq 1$ such that $\pi_n^l, \nu_n \models \varphi_2(V)$ and for each k, $1 \leq k < l$, $\pi_n^k, \nu_n \models \varphi_1(V)$.

As for any given behaviour β_n of S_n, there is a unique path π_n corresponding to it, we will also sometimes say in the following that β_n satisfies or unsatisfies a formula φ meaning that π_n satisfies or unsatisfies φ. We will call the processes assigned to some process variables by ν_n as processes *visible* in π_n via ν_n.

Next, let $\Pi_n \subseteq C_n^* \cup C_n^\omega$ denote any set of paths of S_n. (Later we concentrate on sets of paths corresponding to all finite or fair behaviours.) We define the semantics of MPTL universal and existential formulae as follows:

- $\Pi_n \models \forall V : A\varphi(V)$ iff for all valuations ν_n of PV and all $\pi_n \in \Pi_n$, $\pi_n, \nu_n \models \varphi(V)$.
- $\Pi_n \models \exists V : E\varphi(V)$ iff $\pi_n, \nu_n \models \varphi(V)$ for some PV valuation ν_n and some $\pi_n \in \Pi_n$.

3.3 Evaluating MPTL over Systems and Families

Let S_n be a system of an RTR family $\mathcal{F}$. Given a universal or existential MPTL formula Φ, we say the finite behaviour of S_n satisfies Φ, which we denote by $S_n \models_{fin} \Phi$, iff $\Pi_n^{fin} \models \Phi$ holds. We say the weakly fair behaviour of S_n satisfies Φ, which we denote by $S_n \models_{wf} \Phi$, iff $\Pi_n^{wf} \models \Phi$ holds.

Next, we introduce a notion of MPTL formulae satisfaction over RTR families, in which we allow for specifying the minimum size of the systems to be considered.[2] We go on with the chosen uniformity of quantification and for a universal MPTL formula Φ_a, an RTR family $\mathcal{F}$, and a lower bound l on the number of processes to be considered,

[2] Specifying the minimum size allows one to exclude possibly special behaviours of small systems. Fixing the maximum size would lead to finite-state verification. Although our results could still be used to simplify such verification, we do not discuss this case here.

we define $\mathcal{F}, l \models^{a}_{fin} \Phi_a$ to hold iff $S_n \models_{fin} \Phi_a$ holds for *all* systems $S_n \in \mathcal{F}$ with $l \leq n$. Dually, for an existential MPTL formula Φ_e, we define $\mathcal{F}, l \models^{e}_{fin} \Phi_e$ to hold iff $S_n \models_{fin} \Phi_e$ holds for *some* system $S_n \in \mathcal{F}$ with at least l processes. The same notions of MPTL formulae satisfaction over families can be introduced for weakly fair behaviour too.

4 Verification of Finite Behaviour

As we have already indicated, one of the problems we examine in this paper is verification of finite behaviour of systems of RTR families against correctness requirements expressed in MPTL. In particular, we concentrate on the *parametric finite-behaviour verification problem* of checking whether $\mathcal{F}, l \models^{a}_{fin} \Phi_a$ holds for a certain RTR family $\mathcal{F}$, a universal MPTL formula Φ_a, and a lower bound l on the number of processes to be considered. The problem of checking whether $\mathcal{F}, l \models^{e}_{fin} \Phi_e$ holds for a certain existential MPTL formula Φ_e is dual, and we will not cover it explicitly in the following.

4.1 A Cut-Off Result for RTR\P Families

We first examine the parametric finite-behaviour verification problem for the case of RTR\P families. Let $\Phi_a \equiv \forall p_1, ..., p_k : A\ \varphi(p_1, ..., p_k)$ be a universal MPTL formula with k globally quantified process variables. We show that for any RTR\P family $\mathcal{F}$, the problem of checking $\mathcal{F}, l \models^{a}_{fin} \Phi_a$ can be reduced to simple finite-state examination of the system $S_k \in \mathcal{F}$ with k processes. At the same time, the processes to be monitored via $p_1, ..., p_k$ may be fixed to $1, ..., k$. We denote the resulting verification problem as checking whether $S_k \models_{fin} A\ \varphi(1, ..., k)$ holds. Consequently, we can say that, e.g., to verify mutual exclusion in an RTR\P family $\mathcal{F}$, it suffices to verify it for processes 1 and 2 in the system of $\mathcal{F}$ with only these two processes.

Below, we first give a basic cut-off lemma and then we generalize it to the above.

Lemma 1. *For an RTR\P family $\mathcal{F}$ and an MPTL path formula $\varphi(p_1, ..., p_k)$, the following holds for systems of $\mathcal{F}$:*

$$\forall n \geq k : S_n \models_{fin} \forall p_1, ..., p_k : A\ \varphi(p_1, ..., p_k) \Leftrightarrow S_k \models_{fin} A\ \varphi(1, ..., k)$$

Proof. (Sketch) ($\Rightarrow$) We convert a counterexample behaviour of S_k to one of S_n by adding some processes and letting them idle at q_0. ($\Leftarrow$) To reduce a counterexample behaviour of S_n to one of S_k, we remove the invisible processes and the transitions fired by them (these processes only restrict the behaviour of others by blocking some resources) and we permute the processes to make $1, ..., k$ visible (all processes are initially equal and their names are not significant). □

Lemma 1 and properties of MPTL now easily yield the above promised result.

Theorem 1. *Let $\mathcal{F}$ be an RTR\P family and let $\Phi_a \equiv \forall p_1, ..., p_k : A\ \varphi(p_1, ..., p_k)$ be an MPTL formula. Then, checking whether $\mathcal{F}, l \models^{a}_{fin} \Phi_a$ holds is equal to checking whether $S_k \models_{fin} A\ \varphi(1, ..., k)$ holds.*

4.2 Inexistence of Structure-Independent Cut-Offs for RTR Families

Unfortunately, as we prove below, for families with prioritized resource management, the same reduction as above cannot be achieved even when we allow the bound to also

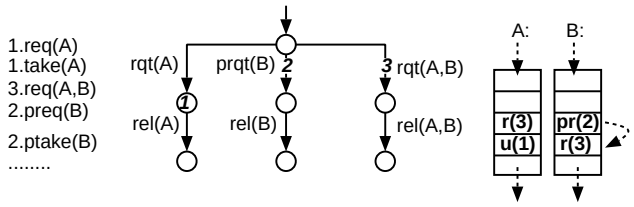

Fig. 1. A scenario problematic for the application of cut-offs (the run from the left is visualized on the RTR automaton and the appropriate resource queues)

depend on the number of available resources and fix the minimum considered number of processes to one.

Theorem 2. *For MPTL formulae Φ_a with k process variables and RTR families $\mathcal{F}$ with m resources, the parametric finite-behaviour verification problem of checking whether $\mathcal{F}, 1 \models^a_{fin} \Phi_a$ holds is not, in general, decidable by examining just the systems $S_1, ..., S_n \in \mathcal{F}$ with n being a function of k and/or m only.*

Proof. (Idea) In the given framework, we can check whether in some system of the RTR family $\mathcal{F}$ based on the automaton from Fig. 1, some process p_1 can request A,B before some process p_2 requests B, but the wish of p_2 is granted before that of p_1. As shown in Fig. 1, the above happens in $S_n \in \mathcal{F}$ with $n \geq 3$, but not in $S_2 \in \mathcal{F}$ (the overtaking between visible processes 2 and 3 is impossible without invisible process 1). Moreover, when we start extending the B and AB branches by more and more pairs of the appropriate (p)rqt/rel actions without extending the A branch, we exclude more than one process to run in these branches via adding rqt(C)/rel(C) (rqt(D)/rel(D)) at their beginnings and ends, and we ask whether p_1 and p_2 can exhibit more and more overtaking, we will need more and more auxiliary processes in the A branch although *k* and *m* will not change. □

Despite the above result, there is still some hope that the parametric finite-behaviour verification problem for RTR and MPTL can be reduced to finite-state model checking. Then, however, the bound on the number of processes would have to also reflect the structure of the RTR automaton of the given family and/or the structure of the formula being examined. We leave the problem in its general form open for future research. Instead, we show below that for certain important subclasses of MPTL, the number of processes to be considered in parametric finite-behaviour verification can be fixed to the number of process variables in the formula at hand as in the RTR\P case (although the underlying proof construction is more complex). In this way, we cover, among others, mutual exclusion as one of the key properties of the considered class of systems.

4.3 Cut-Offs for Subclasses of MPTL

The first subclass of MPTL formulae we consider is the class of *invariance* and *reachability* formulae of the form $\Psi_a ::= \forall V : A\ \Box\psi(V)$ and $\Psi_e ::= \exists V : E\ \Diamond\psi(V)$ in which $\psi(V)$ is a boolean combination of atomic formulae $at(p,q)$. Mutual exclusion is an example of a property that falls into this class.

Let $\Psi_a \equiv \forall p_1, ..., p_k : A\ \Box\psi(p_1, ..., p_k)$ be an arbitrary invariance MPTL formula with *k* quantified process variables. We show that for any RTR family $\mathcal{F}$, the parametric

problem of checking $\mathcal{F}, l \models^{a}_{fin} \Psi_a$ can be reduced to the finite-state problem of verifying $S_k \models_{fin} A\ \Box\psi(1,...,k)$ with the number of processes fixed to k and the processes to be monitored via $p_1,...,p_k$ fixed to $1,...,k$. As above, we first state a basic cut-off lemma, which we subsequently generalize.

Lemma 2. *For any RTR family $\mathcal{F}$ and any nontemporal MPTL path formula $\psi(p_1,...,p_k)$, the following holds for systems of $\mathcal{F}$:*

$$\forall n \geq k : S_n \models_{fin} \forall p_1,...,p_k : A\ \Box\psi(p_1,...,p_k) \Leftrightarrow S_k \models_{fin} A\ \Box\psi(1,...,k)$$

Proof. (Sketch) We modify the proof of Lemma 1: In the ($\Leftarrow$) case, to resolve the problem with possibly disallowed overtaking among visible processes that could be enabled only due to some invisible processes blocking some low-priority visible ones (cf. Fig. 1), we postpone firing of $(\mathtt{p})\mathtt{req}$ transitions to be just before firing of the corresponding $(\mathtt{p})\mathtt{take}$ transitions (or at the very end). Then, since the preserved visible processes release resources as before and do not block them by requests till all originally overtaking requests are served, it can be shown the firability of the reduced transition sequence is guaranteed. Moreover, the behaviour is modified in a way invisible for a reachability formula (negation of $\Box\psi$). □

Theorem 3. *Let $\mathcal{F}$ be an RTR family and let $\Psi_a \equiv \forall p_1,...,p_k : A\ \Box\psi(p_1,...,p_k)$ be an invariance MPTL formula. Then, checking whether $\mathcal{F}, l \models^{a}_{fin} \Psi_a$ holds is equal to checking whether $S_k \models_{fin} A\ \Box\psi(1,...,k)$ holds.*

Another subclass of MPTL that can be handled within parametric finite-behaviour verification of RTR in the same way as above is the class of formulae in which we allow any of the MPTL operators to be used, but we exclude distinguishing whether a process is at a location from which it can request some resources or whether it has already requested them. Using such formulae, we can, for example, check whether some overtaking among the involved processes is possible or excluded (though not on the level of particular requests). Due to space limitations, we skip a precise formulation of this result here and refer an interested reader to the full version of the paper [4].

5 Verification of Fair Behaviour

We next discuss verification of fair behaviour of systems of RTR families against correctness requirements expressed in MPTL. The results presented in this section can be applied for verification of liveness properties, such as absence of starvation, of systems of RTR families. As for finite-behaviour verification, we consider the *problem of parametric verification of weakly fair behaviour*, i.e. checking whether $\mathcal{F}, l \models^{a}_{wf} \Phi_a$ holds for an RTR family $\mathcal{F}$, a universal MPTL formula Φ_a, and a lower bound l on the number of processes.

We show first that under the pure FIFO resource management, considering up to $m+k$ processes—with m being the number of resources and k the number of visible processes—suffices for parametric verification of weakly fair behaviour against any MPTL formulae. By contrast, for prioritized resource management, we prove that (as for finite behaviour verification) there does not exist any general, structure-independent cut-off that would allow us to reduce parametric verification of weakly fair behaviour

to finite-state verification. Moreover, we show that, unfortunately, the inexistence of a structure-independent cut-off concerns, among others, also verification of the very important property of absence of starvation. Thus, for the needs of parametric verification of fair behaviour, we subsequently examine in more detail the possibility only sketched in the previous section, i.e. trying to find a cut-off reflecting the structure of the appropriate RTR automaton and/or the structure of the formula.

5.1 A Cut-Off Result for RTR\P Families

Let $\mathcal{F}$ be an RTR\P family with m resources and $\Phi_a \equiv \forall p_1,...,p_k : A\ \varphi(p_1,...,p_k)$ a universal MPTL formula with k process variables. We show that the parametric verification problem of weakly fair behaviour for $\mathcal{F}$ and Φ_a can be reduced to a series of finite-state verification tasks in which we do not have to examine any systems of $\mathcal{F}$ with more than $m+k$ processes. The processes to be monitored via $p_1,...,p_k$ may again be fixed to $1,...,k$. We denote the thus arising finite-state verification tasks as checking whether $S_n \models_{wf} A\ \varphi(1,...,k)$ holds.

As in Section 4, we now first state a basic cut-off lemma and then we generalize it. However, the way we establish the cut-off turns out to be significantly more complex, because lifting a counterexample behaviour from a small system to a big one is now much more involved than previously. To ensure weak process fairness, newly added processes must be allowed to fire some transitions, but at the same time, this cannot influence the behaviour of the visible processes.

Lemma 3. *For systems of an RTR\P family $\mathcal{F}$ with m resources and an MPTL path formula $\varphi(p_1,...,p_k)$, the following holds:*

$$\forall n \geq m+k : S_n \models_{wf} \forall p_1,...,p_k : A\ \varphi(p_1,...,p_k) \Leftrightarrow S_{m+k} \models_{wf} A\ \varphi(1,...,k)$$

Proof. (Idea) ($\Leftarrow$) Similar to Lemma 1. To eventually forever block in S_{m+k} the visible processes that forever block in S_n, we need at most one invisible process per resource. ($\Rightarrow$) To extend a counterexample behaviour β_{m+k} of S_{m+k} to one of S_n, we distinguish three cases: (1) If all original processes deadlock in β_{m+k}, the newly added ones can deadlock too. (2) If all processes run in β_{m+k}, at least one process may be shown to eventually not use any resource or always eventually release all of them. The new processes can mimic its behaviour. As they regularly do not block any resources, we may interleave them in the non-blocking phases with each other and with the original processes (without influencing the visible ones). (3) The case when some processes get blocked and some run forever in β_{m+k} may be split to subcases solvable like (1) or (2). □

Now, the theorem generalizing the lemma can be easily obtained by exploiting properties of MPTL. Note, however, that unlike in Theorems 1 and 3, it leads to a necessity of examining several systems, which is due to the difference between the cut-off bound $m+k$ and the number k of visible processes.

Theorem 4. *Let $\mathcal{F}$ be an RTR\P family with m resources and let $\Phi_a \equiv \forall p_1,...,p_k : A\ \varphi(p_1,...,p_k)$ be an MPTL formula. Then, checking whether $\mathcal{F}, l \models^a_{wf} \Phi_a$ holds is equal to checking whether $S_n \models_{wf} A\ \varphi(1,...,k)$ holds for all systems $S_n \in \mathcal{F}$ such that $min(max(l,k), m+k) \leq n \leq m+k$.*

We show in [4] that examining the systems S_k (if $l \leq k$) and S_{m+k} is necessary for the above result. The question of a potential optimization of the result by not having to examine all the systems between $max(l,k)$ and $m+k$ remains open for the future, but this does not seem to be a real obstacle to practical applicability of the result.

5.2 Absence of Structure-Independent Cut-Offs for RTR families

In verification of weakly fair behaviour of RTR families against MPTL formulae, we examine complete, usually infinite behaviours of systems of the considered families. However, to be able to examine such behaviours, we need to examine their finite prefixes as well. Then, Theorem 2 immediately shows that there does not exist any structure independent cut-off allowing us to reduce the given general problem to finite-state verification. Moreover, for the case of verifying fair behaviour of RTR families against MPTL formulae, no structure-independent cut-offs exist even for more restricted scenarios than in finite behaviour verification. Namely, the query used in the proof of Theorem 2 speaks about two processes. However, below, we give a theorem showing that for the case of parametric verification of weakly fair behaviour, no structure-independent cut-off exists even for *single-process MPTL formulae*, i.e. formulae having a single process variable and thus speaking about a single visible process. In particular, such a cut-off does not exist for a single-process formula encoding absence of starvation. The theorem is proven in [4] by giving an example family.

Theorem 5. *For RTR families $\mathcal{F}$ with m resources and the property of absence of starvation expressed as $\Phi_a \equiv \forall p : A\ \Box\ (at(p,req) \Rightarrow \Diamond\ at(p,use))$, the problem of checking whether $\mathcal{F}, 1 \models^a_{wf} \Phi_a$ holds is not, in general, decidable by examining just the systems $S_1, ..., S_n \in \mathcal{F}$ with n being a function of m only.*

5.3 A Cut-Off for Single-Process MPTL Formulae

There is no simple cut-off for verification of weakly fair behaviours of RTR families against single-process MPTL formulae since a lot of invisible processes requesting resources with high priority may be needed to block a visible process. Their number depends on the structure of the control automaton. However, this number can be bounded as shown in this section.

To give the bound we need some definitions. Let $\mathcal{F}(\mathcal{A})$ be an RTR family with m resources. The set of control locations Q_t of $\mathcal{A}$ is split into two disjoint parts: Q_o (all internal control locations and those where processes own at least one resource, without loss of generality a process owns always the same resources at a given control location) and Q_n (the others). Let $F = |Q_n|$ ($F \geq 1$ as Q_n contains the initial location q_0) , $C = |2^{Q_o}| = 2^{|Q_o|}$ and $M_C = C^C$. Then, we can define the needed bound as $B_{\mathcal{F}} = CM_C(M_C+1)(2FC(M_C+1))^F + 2C(M_C+1) + 2m + 1$.

The key cut-off lemma below shows that if a formula is true in systems having between $m+1$ and $B_{\mathcal{F}}$ processes, it is also true in systems with more than $B_{\mathcal{F}}$ processes. This and Lemma 5 stating the opposite allows us to reduce the parametric verification problem to verification of systems with up to $B_{\mathcal{F}}$ processes.

Lemma 4. *Let $\mathcal{F}$ be an RTR family with m resources and $\varphi(p)$ an MPTL path formula. Then the following holds for systems of $\mathcal{F}$:*

$$\forall n \geq B_{\mathcal{F}} : (\forall n', m+1 \leq n' \leq B_{\mathcal{F}} : S_{n'} \models_{wf} \forall p : A\ \varphi(p)) \Rightarrow S_n \models_{wf} \forall p : A\ \varphi(p)$$

Proof. (Idea) The full proof is very involved. It is based on the fact that the exact identity of invisible processes is not important, only their number is. The problem of finding the bound can then be seen as finding a bounded solution of a linear equation system encoding properties of admissible counterexamples, which is possible with a lemma from Linear Integer Programming [13]. □

We now give the counterpart to Lemma 4 whose proof is similar to the proof of the appropriate direction of Lemma 3.

Lemma 5. *Let $\mathcal{F}$ be an RTR family and $\varphi(p)$ an MPTL path formula. Then, for systems of $\mathcal{F}$, we have: $\forall n' \geq m+1, n \geq n' : S_n \models_{wf} A\ \varphi(1) \Rightarrow S_{n'} \models_{wf} \forall p : A\ \varphi(p)$.*

We use Lemmas 4 and 5 to give the complete cut-off result for single-process MPTL formulae and weakly fair behaviour of systems of RTR families.

Theorem 6. *Let $\mathcal{F}$ be an RTR family with m resources and let $\Phi_a \equiv \forall p : A\ \varphi(p)$ be a single-process MPTL formula. Then, checking $\mathcal{F}, l \models^a_{wf} \Phi_a$ is equal to checking $S_n \models_{wf} A\ \varphi(1)$ for all $S_n \in \mathcal{F}$ with $l \leq n \leq m+1$ or $n = B_{\mathcal{F}}$.*

5.4 Simple RTR Families

Above, we have shown that parametric verification of weakly fair behaviour of RTR families against single-process MPTL formulae is decidable, but no really simple reduction to finite-state verification is possible in general. We now give a restricted subclass of RTR families for which the problem can be solved using a structure-independent cut-off bound.

An RTR family $\mathcal{F}$ is *simple* if the set of control locations Q_n contains only the initial location q_0: Processes start from it by requesting some resources (possibly in different ways) and then they may request further resources as well as release some. However, as soon as they release all of their resources, they go back to q_0. This class is not unrealistic; it corresponds to systems with a single resource-independent computational part surrounded by actions using resources. For this class we show an improved cut-off bound using $2m+2$ processes, which is better than $B_{\mathcal{F}}$ for $F = 1$. This is basically due to the fact that only m invisible processes can be simultaneously in control locations Q_o.

Theorem 7. *Let $\mathcal{F}$ be a simple RTR family with m resources and let $\Phi_a \equiv \forall p : A\ \varphi(p)$ be a single-process MPTL formula. Then, checking $\mathcal{F}, l \models^a_{wf} \Phi_a$ is equal to checking $S_n \models_{wf} A\ \varphi(1)$ for all $S_n \in \mathcal{F}$ with $l \leq n \leq m+1$ or $n = 2m+2$.*

Notice that an invisible process can freely move among all locations in a subcomponent Q' of $\mathcal{A}$ which is strongly connected by τ-transitions. Therefore, Theorem 7 can be generalized to families whose Q_n corresponds to such a component. Moreover, the same idea can be used to optimize the general $B_{\mathcal{F}}$ bound.

6 Process Deadlockability

Given an RTR family $\mathcal{F}$ and a system $S_n \in \mathcal{F}$, we say that a *process p is deadlocked* in a configuration $c \in C_n$ if there is no configuration reachable from c from which we could fire some transition in p. As is common for linear-time frameworks, process deadlockability cannot be expressed in MPTL, and so since it is an important property to check for the class of systems we consider, we now provide a specialized (structure-independent) cut-off result for dealing with it.

Theorem 8. *Let $\mathcal{F}$ be an RTR family with m resources. For any l, the systems $S_n \in \mathcal{F}$ with $l \leq n$ are free of process deadlock iff $S_{max(m,2)} \in \mathcal{F}$ is.*

Proof. (Sketch) We can encounter scenarios where a group of processes is mutually deadlocked due to some circular dependencies in queues of requests, but also situations where a process is deadlocked due to being always inevitably overtaken by processes that keep running and do not even own any resource forever. However, when we (partial-

ly) replace overtaking by postponed firing of requests (cf. Lemma 2), push blocked high-priority requests before the low-priority ones (the former block the latter, but not conversely), and preserve only the running processes that never release all resources simultaneously, we can show that we suffice with one (primary) blocked and/or blocking process per resource. □

Let us note that the possibility of inevitable overtaking examined in the proof of Theorem 8 as a possible source of process deadlocks in systems of RTR families is stronger than starvation. Starvation arises already when there is a single behaviour in which some process is eventually always being overtaken. Interestingly, as we have shown, inevitable overtaking is much easier to handle than starvation, and we obtain a cut-off bound that cannot be improved even when we restrict ourselves to RTR\P families with no overtaking.

7 RTR Families and Undecidability

Finally, we discuss an extension of MPTL by *local process quantification* [8] where processes to be monitored in a behaviour are not fixed at the beginning, but may be chosen independently in each encountered state. Local process quantification can be added to MPTL by allowing $\forall V' : \varphi(V \cup V')$ to be used in a path formula $\varphi(V)$ with the semantics $\pi_n, \nu_n \models \forall V' : \varphi(V \cup V')$ iff $\pi_n, \nu'_n \models \varphi(V \cup V')$ holds for all valuations ν'_n of PV such that $\forall p \in PV \setminus V' : \nu'_n(p) = \nu_n(p)$. Such a quantification can be used to express, e.g., the global response property $A\Box((\exists p_1 : at(p_1, req)) \Rightarrow \Diamond(\exists p_2 : at(p_2, resp)))$, which cannot be encoded with global process quantifiers if the number of processes is not known. Unfortunately, it can be shown that parametric verification of linear-time finite-behaviour properties with local process quantification is undecidable even for RTR\P.

Theorem 9. *The parametric finite-behaviour verification problem of checking* $\mathcal{F}, 1 \models^{a}_{fin} \Phi_a$ *for an RTR\P family* $\mathcal{F}$ *and an MPTL formula* Φ_a *with local process quantification is undecidable even when the only temporal operators used are* $\Box$ *and* $\Diamond$ *and no temporal operator is in the scope of any local process quantifier.*

Proof. (Idea) The proof is done via simulating two-stack push-down automata and is very complex because the queues we work with are not classical communication queues, but only waiting queues. □

8 Conclusions

In this paper, we have defined an abstract model for a significant class of parametric systems of processes competing for access to shared resources under a FIFO resource management with a possibility of distinguishing low- and high-priority requests. The primitives capturing the interaction between processes and resources and the resource management policies considered are natural and inspired by real-life applications. We have established cut-off bounds showing that many practical parametric verification problems (including verification of mutual exclusion, absence of starvation, and process deadlockability) are decidable in this context. The way the obtained results were established is sometimes technically highly involved, which is due to the fact that the considered model is quite powerful and (as we have also shown) positive decidability can easily be lost if verification of a bit more complex properties is considered.

The structure-independent cut-offs we have presented in the paper are small and—for verification of finite behaviour and process deadlockability—optimal. They provide

us with practical decision procedures for the concerned parametric verification problems and, moreover, they can also be used to simplify finite-state verification for systems with a given large number of processes.

The structure-dependent cut-off for single-process formulae and verifying the fair behaviour of the general RTR families is quite big and does not yield a really practical decision procedure. One challenging problem is now to optimize this bound. Although we know that no general structure-independent cut-off exists, the bound we have provided is not optimal, and significantly improved cut-offs could be found especially for particular classes of systems as we have already shown for simple RTR families.

Another interesting issue is to improve the decidability bounds. For general RTR families and arbitrary MPTL formulae, decidability of parametric verification of finite as well as fair behaviour remains open. So far, we have only shown that these problems cannot be handled via structure-independent cut-offs. Conversely, the question of existence of practically interesting, decidable fragments of MPTL with local process quantification is worth examining too. If no (or no small) cut-off can be found, we could then try to find some adequate abstraction and/or symbolic verification techniques.

Finally, several extensions or variants of the framework can be considered. For example, the questions of nonexclusive access to resources or nonblocking requests can be examined. Moreover, several other locker policies can be considered, e.g., service in random order or a policy where any blocked process can be overtaken. We believe the results presented here and the reasoning used to establish them provide (to a certain degree) a basis for examining such questions.

Acknowledgment: We thank P.A. Abdulla and T. Arts for fruitful discussions.

References

1. P. Abdulla, A. Bouajjani, B. Jonsson, M. Nilsson. Handling Global Conditions in Parameterized System Verification. In *Proc. of CAV'99*, *LNCS* 1633. Springer, 1999.
2. T. Arts, C. Earle, J. Derrick. Verifying Erlang Code: A Resource Locker Case-Study. In *Proc. of FME'02*, *LNCS* 2391. Springer, 2002.
3. K. Baukus, S. Bensalem, Y. Lakhnech, K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In *Proc. of TACAS 2000*, *LNCS* 1785. Springer, 2000.
4. A. Bouajjani, P. Habermehl, T. Vojnar. Verification of Parametric Concurrent Systems with Prioritized FIFO Resource Management. Full version available at `http://verif.liafa.jussieu.fr/~vojnar/download/concur03.ps.gz`
5. E. Emerson, V. Kahlon. Model Checking Large-Scale and Parameterized Resource Allocation Systems. In *Proc. of TACAS'02*, *LNCS* 2280. Springer, 2002.
6. E. Emerson, K. Namjoshi. Reasoning about Rings. In *Proc. of POPL'95*, 1995.
7. E. Emerson, K. Namjoshi. Automatic Verification of Parameterized Synchronous Systems. In *Proc. of CAV'96*, *LNCS* 1102. Springer, 1996.
8. E. Emerson, A. Sistla. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach. *ACM Transactions on Programming Languages and Systems*, 19(4), 1997.
9. S. German, A. Sistla. Reasoning about Systems with Many Processes. *JACM*, 39(3), 1992.
10. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *Proc. of CAV'97*, *LNCS* 1254, 1997.
11. R. Kurshan, K. McMillan. A Structural Induction Theorem for Processes. *Information and Computation*, 117(1), 1995.
12. A. Pnueli, S. Ruah, L. Zuck. Automatic Deductive Verification with Invisible Invariants. In *Proc. of TACAS 2001*, *LNCS* 2031. Springer, 2001.
13. J. von Zur Gathen, M. Sieveking. A Bound on Solutions of Linear Integer Equalities and Inequalities. *Proceedings of the American Mathematical Society*, 1978.
14. P. Wolper, V. Lovinfosse. Verifying Properties of Large Sets of Processes with Network Invariants. In *Autom. Verification Methods for Finite State Systems*, *LNCS* 407, 1989. Springer.