

Verification of parametric concurrent systems with prioritised FIFO resource management

Ahmed Bouajjani · Peter Habermehl · Tomáš Vojnar

Published online: 25 January 2008
© Springer Science+Business Media, LLC 2008

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). Furthermore, we show that parametric verification against formulae with local process quantification is undecidable in the given context.

Keywords Formal verification · Parameterised verification · Infinite-state system verification · Cut off · Model checking · Parameterised networks of processes · Resource sharing

This work was supported in part by the European Commission (FET project ADVANCE, contract No. IST-1999-29082), the French government by the project ACI Sécurité Informatique, the Czech Grant Agency (project 102/07/0322), the Czech-French Barrande project 2-06-27, and the Czech Ministry of Education within the project MSM 0021630528 “Security-Oriented Research in Information Technology”.

A. Bouajjani · P. Habermehl
LIAFA, University Paris 7—Denis Diderot/CNRS, Case 7014, 2, place Jussieu, 75251 Paris Cedex 05, France

A. Bouajjani
e-mail: Ahmed.Bouajjani@liafa.jussieu.fr

P. Habermehl
e-mail: Peter.Habermehl@liafa.jussieu.fr

T. Vojnar (✉)
FIT, Brno University of Technology, Božetěchova 2, 61266 Brno, Czech Republic
e-mail: vojnar@fit.vutbr.cz

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 the above 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 request 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 optimised 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.

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 nontrivial parametric verification problem 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 $LTL \setminus X$ 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 $LTL \setminus X$ 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 formulae, 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* [9], which allows one to examine different processes in different encountered states.

Related work Verification of parameterized networks of processes is, in general, undecidable [2]. The problems we consider here turn out to be decidable due to the fact that the processes cannot communicate too much via the waiting queues.

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 [1, 4, 11–13, 15]. The idea of cut-offs has already been used in several contexts [5–8, 10] 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 prioritised FIFO resource management. The *two* involved obstacles (parameterisation 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 [6] 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 [6]). Moreover, some of the properties to be verified we consider here are different from [6] (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—both from the point of view of the systems considered as well as their key properties to be verified (i.e. mutual exclusion, non-starvation, and absence of deadlocks)—by the work presented in [3] that concerns

verification of the use of shared resources in the Ericsson's AXD 301 ATM switch. In particular, in [3], finite-state model checking was used to verify some isolated instances of the given parametric system. The parametric verification of this kind of systems was, in fact, proposed by Ericsson as a research challenge within the European project ADVANCE as it was not amenable to any existing verification technique. Although the results presented in this paper do not cover all advanced features of the real control of the AXD 301 ATM switch (which includes message passing, distributed computation, real-time features, exclusive as well as shared access to resources, etc.), these results are still significant and cover an important part of the given verification problem (whose many features are not restricted just to the mentioned kind of switches). Moreover, they provide a solid basis for future research on dealing with more advanced features of the given kind of systems as also mentioned in the concluding remarks.

Outline We first formalise 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 for properties with local process quantification. We close the paper by a brief summary of the results and a discussion of their possible future extensions.

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 utilisation), 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 $(\text{p})\text{rqt}$ transitions¹ while waiting for the requested resources to be available for them. Therefore, a single $(\text{p})\text{rqt}$ transition in a model semantically corresponds to two transitions, which we denote as $(\text{p})\text{req}$ (request a set of resources) and $(\text{p})\text{take}$ (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 $A = \{\tau\} \cup \{a(R') \mid a \in \{\text{rqt}, \text{prqt}, \text{rel}\} \wedge R' \neq \emptyset \wedge R' \subseteq R\}$. 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 whose control is given by \mathcal{A} and that are identified by elements of $P_n = \{1, \dots, n\}$. In the following, if no confusion is possible, we usually drop the reference to \mathcal{A} . We denote as *RTR\setminus P families* the special cases of RTR families whose control automata contain no high-priority request actions.

¹We use $(\text{p})\text{rqt}$ when addressing both rqt as well as prqt transitions.

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 $(p)\text{rqt}$ actions, we extend the set Q of “original” control locations to Q_{ex} containing a unique internal control location q_t for each transition $t \in T$ based on a $(p)\text{rqt}$ action. Furthermore, let T_{ex} be the set obtained from T by preserving all τ and rel transitions and splitting each transition $t = (q_1, (p)\text{rqt}(R'), q_2) \in T$ to two transitions $t_1 = (q_1, (p)\text{req}(R'), q_t)$ and $t_2 = (q_t, (p)\text{take}(R'), q_2)$.

We define the *resource queue alphabet* of S_n as $\Sigma_n = \{s(p) \mid s \in \{\text{r}, \text{pr}, \text{g}, \text{u}\} \wedge p \in P_n\}$. 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 $c : (P_n \rightarrow Q_{ex}) \cup (R \rightarrow \Sigma_n^*)$ 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.

In the following, we will need a possibility to rename processes. A *renaming of processes* of S_n is a bijection $\xi_n : P_n \rightarrow P_n$. We lift ξ_n to work over Σ_n^* by defining $\xi_n(\varepsilon) = \varepsilon$ and $\xi_n(s_1(p_1) \dots s_l(p_l)) = s_1(\xi_n(p_1)) \dots s_l(\xi_n(p_l))$ for any $l \geq 1$ and $s_1(p_1) \dots s_l(p_l) \in \Sigma_n^*$. Moreover, we also lift ξ_n to work over configurations from C_n by requiring that (1) $\xi_n(c)(p) = c(\xi_n^{-1}(p))$ for any $c \in C_n$ and $p \in P_n$ and (2) $\xi_n(c)(r) = \xi_n(c(r))$ for any $c \in C_n$ and $r \in R$.

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 $\text{RTR} \setminus \text{P}$ 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 not yet granted, high-priority request that was raised sooner than the given request. A low-priority request is granted iff the needed resources are not in use nor granted, they are not subject to any not yet granted request raised sooner than the given request, nor they are subject to any not yet granted high-priority request that was raised later than the given request, but 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 C_n equal to c up to the following for each $r \in R$:

1. If $c(r) = w_1.\text{pr}(p).w_2$ for some $p \in P_n$, $w_1, w_2 \in \Sigma_n^*$ s.t. $c(p) = q_t$ for a certain $t = (q_1, \text{prqt}(R'), q_2) \in T$ and for all $r' \in R'$, $c(r') = w'_1.\text{pr}(p).w'_2$ with $w'_1 \in \{\text{r}(p') \mid p' \in P_n\}^*$ and $w'_2 \in \Sigma_n^*$, we set $\Lambda(c)(r)$ to $\text{g}(p).w_1.w_2$.
(Intuitively, pr queue items can overtake r items if no pr item of the given request is preceded by a u , g , or pr item.)
2. If $c(r) = \text{r}(p).w$ for some $p \in P_n$, $w \in \Sigma_n^*$ s.t. $c(p) = q_t$ for a certain $t = (q_1, \text{rqt}(R'), q_2) \in T$ and for all $r' \in R'$, $c(r') = \text{r}(p).w'$ with $w' \in \Sigma_n^*$, and the premise of case 1 is not satisfied for r' , we set $\Lambda(c)(r)$ to $\text{g}(p).w$.
(All r items of a low-priority request to be granted must be the heads of the appropriate

queues and cannot be followed by any pr items of high-priority requests that can be granted.)

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

For all transitions $t = (q_1, \tau, q_2) \in T_{ex}$ and $t = (q_1, a(R'), q_2) \in T_{ex}$, $a \in \{\text{rel}, \text{req}, \text{preq}\}$, we define $en(c, t, p) \Leftrightarrow c(p) = q_1$. For each transition $t = (q_1, (\text{p})\text{take}(R'), q_2) \in T_{ex}$, we define $en(c, t, p) \Leftrightarrow c(p) = q_1 \wedge \forall r \in R' \exists w \in \Sigma_n^* : c(r) = \text{g}(p).w$. 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 $(\text{p})\text{take}$, if the appropriate request has been granted.

Firing of a transition $t = (q_1, \tau, q_2) \in T_{ex}$ 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 $(\text{p})\text{req}$ 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 $t = (q_1, (\text{p})\text{req}(R'), q_2) \in T_{ex}$, 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 $c^+ = \{(p, q_2)\} \cup \{(r, c(r).(\text{p})\text{r}(p)) \mid r \in R'\}$.

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

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

Suppose now that a process p_1 is requesting a set of resources R' in a configuration $c \in C_n$, i.e. $c(p_1) = q_2$ for some $(q_1, (\text{p})\text{req}(R'), q_2) \in T_{ex}$. We say that p_1 and its current request are *blocked* by a process p_2 on a resource $r \in R'$ in c iff $c(r) \in \Sigma_n^*.s_2(p_2). \Sigma_n^*.s_1(p_1). \Sigma_n^*$ and $s_1 = \text{pr} \Rightarrow s_2 \neq \text{r}$. In other words, p_1 is blocked by p_2 on r iff p_2 has a suitable item in the queue of r that precedes some queue item of p_1 in this queue. Low-priority requests may be blocked by queue items of any type, in the case of high-priority requests, r items are not powerful enough.

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 $c_1(p_1, t_1)c_2 \dots (p_l, t_l)c_{l+1}$ such that for each $i \in \{1, \dots, l\}$, $en(c_i, t_i, p_i)$ 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. More formally, we say a complete behaviour β_n is weakly fair iff for each $p \in P_n$, if β_n is infinite, and $\exists i : \forall j \geq i : \exists t \in T_{ex} : en(c_j, t, p)$ holds, then $\forall i : \exists j \geq i : \exists t \in T_{ex} : c_{j+1} = to(c_j, t, p)$ holds too.

We call a complete behaviour *strongly (process) fair* iff each process that is always eventually enabled to fire some transitions, always eventually fires some transitions. Formally, a complete behaviour β_n is strongly fair iff for each $p \in P_n$, if β_n is infinite, and

$\forall i : \exists j \geq i : \exists t \in T_{ex} : en(c_j, t, p)$ holds, then $\forall i : \exists j \geq i : \exists t \in T_{ex} : c_{j+1} = to(c_j, t, p)$ holds too. However, we do not deal with strong fairness in the following because in our model, the notions of strong and weak fairness coincide.

Lemma 1 *A complete behaviour of a system S_n of an RTR family \mathcal{F} is strongly fair iff it is weakly fair.*

Proof It is enough to show that if the premise of strong fairness is satisfied by a behaviour, then either the premise of weak fairness is also satisfied, or this is not the case, but strong fairness is anyway not broken. In systems of RTR families, a transition based on a τ , $(p)req$, or rel action is enabled in a process when reached by this process. A transition based on a $(p)take$ action is enabled when reached and when the concerned resources are granted to the involved process. Once issued grants of some resources cannot be cancelled. Therefore, a transition enabled in some process in a system of some RTR family can be disabled only when some transition is fired by the given process. Thus, if a process is always eventually enabled to fire some transitions, it is either eventually always enabled to fire some transitions, or this is not the case, but the process always eventually fires some transitions. \square

For a behaviour $\beta_n = c_1(p_1, t_1)c_2(p_2, t_2) \dots$ of a system S_n of an RTR family \mathcal{F} starting at $c_1 \in C_n$, we call the configuration sequence $\pi_n = c_1c_2 \dots$ a *path* of S_n corresponding to β_n and starting at c_1 . Next, we call the transition firing sequence $\rho_n = (p_1, t_1)(p_2, t_2) \dots$ a *run* of S_n corresponding to β_n and starting from c_1 . If the behaviour is not important, we do not mention it. Moreover, if the run or path starts from $c_0 \in C_n$, we may drop the reference to it too. We denote $\Pi_n^{fin}, \Pi_n^{fin} \subseteq C_n^+$, the set of all finite paths of S_n and $\Pi_n^{wf}, \Pi_n^{wf} \subseteq C_n^+ \cup C_n^\omega$, the set of all paths of S_n corresponding to complete, weakly fair behaviours.

Before proceeding further on, let us stress that the behaviour of systems from RTR families does not depend on the identifiers of the involved processes as stated in the below simple lemma.

Lemma 2 *For any system $S_n \in \mathcal{F}$ from an RTR family \mathcal{F} , any behaviour $\beta_n = c_1(p_1, t_1)c_2(p_2, t_2) \dots$ of S_n , and any renaming ξ_n of processes of P_n , the sequence $\xi_n(\beta_n) = \xi_n(c_1)(\xi_n(p_1), t_1)\xi_n(c_2)(\xi_n(p_2), t_2) \dots$ is a behaviour of S_n too.*

Proof Immediate as the names of processes do not influence transition enabledness nor firing in systems of RTR families. \square

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 $LTL \setminus X$, 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 will get back to this point in the conclusions).

We extend $LTL \setminus X$ by *global process quantification*² in a way inspired by $ICTL^*$ (see, e.g., [9]) 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 $\top, \text{ff}, \varphi(V) \wedge \varphi(V), \Box\varphi(V)$, 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:

$$\Phi_a ::= \forall V : A \varphi(V) \quad \Phi_e ::= \exists V : E \varphi(V)$$

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 $v_n : PV \rightarrow P_n$ 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 $\pi_n = c_1 c_2 \dots c_{|\pi_n|}$ ($\pi_n = c_1 c_2 \dots$), let π_n^l denote the suffix $c_l c_{l+1} \dots c_{|\pi_n|}$ ($c_l c_{l+1} \dots$) 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 v_n of PV , we inductively define the semantics of MPTL path subformulae $\varphi(V)$ as follows:

$$- \pi_n, v_n \models at(p, q) \text{ iff } \pi_n = c.\pi_n' \text{ for some path } \pi_n' \text{ and } c(v_n(p)) = q.$$

²Later, we will extend MPTL by local process quantification too and show that this leads to undecidability. We do not introduce local process quantification immediately in order not to complicate the presentation of the positive decidability results.

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

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 v_n as processes *visible* in π_n via v_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 v_n of PV and all $\pi_n \in \Pi_n, \pi_n, v_n \models \varphi(V)$.
- $\Pi_n \models \exists V: E \varphi(V)$ iff $\pi_n, v_n \models \varphi(V)$ for some PV valuation v_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.³ 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, we define $\mathcal{F}, l \models_{fin}^a \Phi_a$ 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 $\mathcal{F}, l \models_{fin}^e \Phi_e$ to hold iff $S_n \models_{fin} \Phi_e$ holds for *some* system $S_n \in \mathcal{F}$ with at least l processes. We suppose the same notions of MPTL formulae satisfaction over families to 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 $\mathcal{F}, l \models_{fin}^a \Phi_a$ 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 $\mathcal{F}, l \models_{fin}^e \Phi_e$ 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\setminus P families

We first examine the parametric finite-behaviour verification problem for the case of RTR\setminus P families. Let $\Phi_a \equiv \forall p_1, \dots, p_k : A \varphi(p_1, \dots, p_k)$ be a universal MPTL formula with k globally quantified process variables. We show that for any RTR\setminus P family \mathcal{F} , the problem of

³Fixing 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.

checking $\mathcal{F}, l \models_{fin}^a \Phi_a$ 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, \dots, p_k may be fixed to $1, \dots, k$. We denote the resulting verification problem as checking whether $S_k \models_{fin} A \varphi(1, \dots, k)$ holds. Consequently, we can say that, e.g., to verify mutual exclusion in an $RTR \setminus P$ family \mathcal{F} , it suffices to verify it for processes 1 and 2 in the system of \mathcal{F} with only these two processes.

A key to the proof of the above result is the fact that if we take a finite (or, in fact, even infinite) run ρ_n of a system S_n from some $RTR \setminus P$ family \mathcal{F} and we remove from it all actions of processes $k + 1, \dots, n$ for any $1 \leq k < n$, the remaining sequence of actions of processes $1, \dots, k$ is a run of the system $S_k \in \mathcal{F}$. Formally, let $\rho_n = (p_1, t_1)(p_2, t_2) \dots$ be a (finite or infinite) run of a system S_n of an $RTR \setminus P$ family \mathcal{F} . For any $1 \leq k < n$, we denote by $\rho_n^{\downarrow k}$ the sequence obtained from ρ_n by removing every action $(p_i, t_i) \in \{k + 1, \dots, n\} \times T_{ex}$. Then, the following holds.

Lemma 3 *For any (finite or infinite) run $\rho_n = (p_1, t_1)(p_2, t_2) \dots$ of a system S_n of an $RTR \setminus P$ family \mathcal{F} and any $1 \leq k < n$, $\rho_n^{\downarrow k}$ is a run of S_k .*

Proof A run (whose initial configuration is not given explicitly) is a transition sequence firable from the initial configuration of S_n , i.e. from $c_0 \in C_n$. In systems of $RTR \setminus P$ families, the only transitions that need not be firable immediately when their source control location is reached are the ones based on τ_{ake} . Their enabledness requires a grant by the locker. In $RTR \setminus P$ based on the pure FIFO resource management policy, a grant is issued only when all the queue items recording the concerned intention to take some resources are the heads of the appropriate queues. They may only be the heads when they are the first items ever inserted into the queues or when all the previously inserted items have been released. Indeed, in $RTR \setminus P$, any other action than τ_{rel} does not (up to relabelling) modify the contents of the queues, or it leads to an addition of some new items to the tails of the queues.

Thus, the only transitions whose removal from ρ_n could disable some of the originally enabled transitions are the transitions based on τ_{rel} . However, as we consider runs that start from the initial configurations of RTR where all queues are empty, and a process cannot influence queue items related to other processes, τ_{rel} in processes different from $1, \dots, k$ does not have any effect when transitions based on the other actions of these processes are removed. So, by removing all transitions done by processes not among $1, \dots, k$ from ρ_n , we obtain a run of S_k . □

Using Lemma 3, we now prove a basic cut-off lemma for finite behaviours of systems from $RTR \setminus P$ families allowing us to reduce verification of a system with $n \geq k$ processes to the system with k processes.

Lemma 4 *For an $RTR \setminus P$ family \mathcal{F} and an MPTL path formula $\varphi(p_1, \dots, p_k)$, the following holds for systems of \mathcal{F} :*

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

Proof It suffices to show that when the assertions of the implications do not hold, the premises do not hold either.

The case of (\Rightarrow) is easy. Suppose there is a behaviour β_k of S_k that satisfies $\neg\varphi(p_1, \dots, p_k)$ for the valuation $v_k(p_i) = i, i \in \{1, \dots, k\}$. Given any $n \geq k, n - k$ processes can be added to β_k by simply letting them idle at q_0 . This way the additional processes do

not block any resources and do not restrict any other processes, and we clearly obtain a counterexample behaviour of S_n for v_n such that $v_n(p_i) = i, i \in \{1, \dots, k\}$.

Let us concentrate on the (\Leftarrow) case. Suppose there is a finite behaviour β_n of S_n for some $n \geq k$ such that $\pi_n, v_n \models \neg\varphi(p_1, \dots, p_k)$ for the path π_n corresponding to β_n and some valuation $v_n : PV \rightarrow P_n$. Let $\xi_n : P_n \rightarrow P_n$ be any renaming of processes of S_n such that $\xi_n(v_n(p_i)) = i$ for $1 \leq i \leq k$. Let $\beta'_n = \xi_n(\beta_n)$. Due to Lemma 2, β'_n is a behaviour of S_n . Let π'_n be the path associated with β'_n , and v'_n be such that $v'_n(p_i) = i$ for $1 \leq i \leq k$. Since φ does not depend on the concrete names of processes, $\pi'_n, v'_n \models \neg\varphi(p_1, \dots, p_k)$.

Moreover, Lemma 3 says that the sequence $\rho_k = (\rho'_n)^{\downarrow k}$ for ρ'_n being the run corresponding to β'_n is a run of S_k . Hence, ρ_k forms a basis of some behaviour β_k of S_k . It remains to show that $\pi_k, v_k \models \neg\varphi(p_1, \dots, p_k)$ for π_k corresponding to β_k and v_k such that $v_k(p_i) = i$ for $i \in \{1, \dots, k\}$. However, this is clearly the case because the control locations of the processes $1, \dots, k$ are being changed in the same way in β_k as in β'_n (and from the same initial q_0), the control locations of processes not among $1, \dots, k$ are invisible for φ , and MPTL is stuttering insensitive. \square

By using Lemma 4 and properties of MPTL, we easily obtain the above promised result.

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

Proof First, for all $S_n \in \mathcal{F}$ with $l \leq n < k$, $S_n \models_{fin} \Phi_a$ trivially holds as there are not k distinct processes here. If $S_k \models_{fin} A \varphi(1, \dots, k)$ holds, Lemma 4 implies $S_n \models_{fin} \Phi_a$ holds for all $k \leq n$, which is what is needed or even more (for $k < l$). If $S_n \models_{fin} \Phi_a$ holds for $l \leq n$, $S_k \models_{fin} A \varphi(1, \dots, k)$ follows either directly for $l \leq k$ or from Lemma 4. \square

4.2 Inexistence of structure-independent cut-offs for RTR families

Unfortunately, as we prove below, for families with prioritised resource management, the same reduction as above cannot be achieved even when we allow the bound to also 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 $\mathcal{F}, l \models_{fin}^a \Phi_a$ holds cannot, in general, be solved by examining just the behaviour of the systems $S_1, \dots, S_n \in \mathcal{F}$ with n being a function of k and/or m only.*

Proof Let us consider the RTR family \mathcal{F} over the automaton from Fig. 1. We may concentrate on the dual problem to $\mathcal{F}, l \models_{fin}^a \Phi_a$ and ask whether in some system of \mathcal{F} , a process requesting only B can overtake a process requesting both A and B as follows: $\exists S_n \in \mathcal{F} : S_n \models_{fin} \exists p_1, p_2 : E ((\neg at(p_2, B.1)) \mathcal{U} at(p_1, AB.1)) \wedge ((\neg at(p_1, AB.2)) \mathcal{U} at(p_2, B.2))$. The query refers to the control locations defined in Fig. 1 and checks whether some process p_1 can request A and B (location AB.1, which is internal for the concerned transition) before some other process p_2 requests B (location B.1), but still the wish of p_2 is granted (B.2) before the wish of p_1 (AB.2). Such a situation can really happen in $S_n \in \mathcal{F}$ with $n \geq 3$ (cf. the run shown on the left in Fig. 1 and the overtaking between processes 2 and 3). Note, however, that without an auxiliary process, the overtaking is not possible (when process 1 is left out from the presented witness behaviour, A and B are immediately granted to process 3).

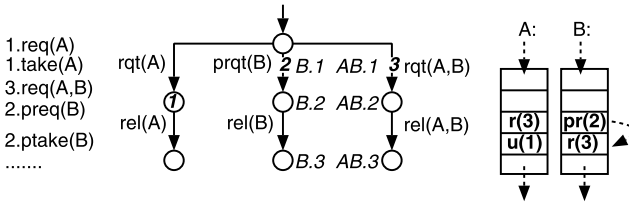


Fig. 1 An RTR scenario problematic for the application of cut-offs

Let us now extend the B branch by another pair of subsequent actions $prqt(B)$ and $rel(B)$ and the AB branch by another pair $rqt(A, B)$ and $rel(A, B)$. Moreover, in order to exclude the possibility of having more than one active process in each of these branches, let us add pairs $rqt(C)/rel(C)$ ($rqt(D)/rel(D)$) at their beginnings and ends, respectively. Over the new family, one may ask whether after the first overtaking, the processes can meet in B.3 and AB.3 and then exhibit another overtaking. As the left-most branch is not extended, and no auxiliary processes can run in the middle or right branch, at least 4 processes are needed to show satisfaction of the new property in the new setting. Obviously, the system and the formula may be arbitrarily further extended in the described way (of course, without adding further operations on C and D), which leads to a necessity of considering bigger and bigger systems to witness satisfaction of the considered formulae although k as well as m remain the same. \square

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\setminus 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 that we consider is the class of *invariance* and *reachability* formulae of the form

$$\Psi_a ::= \forall V : A \Box \psi(V) \quad \Psi_e ::= \exists V : E \Diamond \psi(V)$$

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 $\Psi_a \equiv \forall p_1, \dots, p_k : A \Box \psi(p_1, \dots, p_k)$ 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 $\mathcal{F}, l \models_{fin}^a \Psi_a$ can be reduced to the finite-state problem of verifying $S_k \models_{fin} A \Box \psi(1, \dots, k)$ with the number of processes fixed to k and the processes to be monitored via p_1, \dots, p_k fixed to $1, \dots, k$.

As in Sect. 4.1, a key to the proof of the above result is a lemma that we state below and that allows us to drop actions of certain processes from a run without changing its

enabledness. Due to the presence of prioritised transitions, the situation is, however, more complicated, and we have to not only drop some actions, but also re-order the remaining ones. More precisely, let $\rho_n = (p_1, t_1)(p_2, t_2) \dots (p_l, t_l)$ where $l \geq 1$ be a finite run of a system S_n of an RTR family \mathcal{F} . For any $1 \leq k < n$, we denote by $\rho_n^{\downarrow i(k)}$ the sequence obtained from $\rho_n^{\downarrow k}$ by moving all occurrences of (p)req transitions to the right such that they get just before firing of the appropriate (p)take transitions if they are fired in ρ_n , or to the very end of $\rho_n^{\downarrow i(k)}$ (with the original mutual ordering among such transition occurrences being preserved). We can prove the following result.

Lemma 5 *For any finite run $\rho_n = (p_1, t_1)(p_2, t_2) \dots (p_l, t_l)$ of a system S_n of an RTR family \mathcal{F} where $l \geq 1$ and for any $1 \leq k < n$, $\rho_n^{\downarrow i(k)}$ is a run of S_k .*

Proof As in the proof of Lemma 3, we have to show that all the involved (p)take transitions are fireable in the appropriate processes at all the points in $\rho_n^{\downarrow i(k)}$ where they appear. All other transitions are automatically enabled in RTR when a process reaches their source control location. Let us therefore examine what happens with the enabledness of a transition t based on (p)take(R') and fired by a process $p \in P_k$ at a certain position in ρ_n after the transformation of ρ_n to $\rho_n^{\downarrow i(k)}$.

First, the enabledness cannot be hampered by any $r \in R'$ being blocked due to some (p)req(R'') transition ($r \in R''$) fired by a process not among P_k because no such transition is preserved in $\rho_n^{\downarrow k}$. The removal of rel transitions fired by processes not in P_k is also not important because we always start with empty queues, the (p)req transitions of processes not in P_k are removed, and the processes not in P_k cannot release resources occupied by P_k processes.

Next, the mutual order of rel and (p)take transitions of P_k processes is preserved. Moreover, it is clear that between the last rel on any $r \in R'$ in some P_k process (or c_0 if r has not been used by a P_k process yet) and the given firing of t in ρ_n (that must be preceded by granting r to p) there is no (p)take transition on r in a P_k process (otherwise there would have to be another rel). The construction of $\rho_n^{\downarrow i(k)}$ then guarantees that in $\rho_n^{\downarrow i(k)}$ there is no (p)req on r between the concerned rel and firing the (p)req transition corresponding to t at all, and r is thus ready to be granted to p immediately upon request. As this construction deals with any t of the (p)take type at any position in ρ_n and with any r related to t , the proof of $\rho_n^{\downarrow i(k)}$ being fireable in S_k is complete. \square

We are now ready to state and prove a basic cut-off lemma for prioritised RTR families and invariance MPTL formulae allowing us to reduce verification of a system with $n \geq k$ processes to the system with k processes.

Lemma 6 *For any RTR family \mathcal{F} and any MPTL path formula $\psi(p_1, \dots, p_k)$ that is just a boolean combination of the at(p, q) atomic formulae, the following holds for systems of \mathcal{F} :*

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

Proof It suffices to show that when the assertions of the implications do not hold, the premises do not hold either. The case of (\Rightarrow) is easy and very similar to that of Lemma 4, and so we skip it here. Let us concentrate on the (\Leftarrow) case. Suppose there is a finite behaviour β_n of S_n (for any $n \geq k$) such that $\pi_n, v_n \models \Diamond \neg \psi(p_1, \dots, p_k)$ holds for the path π_n corresponding to β_n and some valuation $v_n : PV \rightarrow P_n$. We show that we can reduce this counterexample to one in S_k with processes $1, \dots, k$ being visible via p_1, \dots, p_k .

Let $\xi_n : P_n \rightarrow P_n$ be any renaming of processes of S_n such that $\xi_n(v_n(p_i)) = i$ for $1 \leq i \leq k$. Let $\beta'_n = \xi_n(\beta_n)$. Due to Lemma 2, β'_n is a behaviour of S_n . Let π'_n be the path associated with β'_n , and v'_n be such that $v'_n(p_i) = i$ for $1 \leq i \leq k$. Since φ does not depend on the concrete names of processes, $\pi'_n, v'_n \models \neg\varphi(p_1, \dots, p_k)$.

Moreover, Lemma 5 says that the sequence $\rho_k = (\rho'_n)^{\downarrow 1(k)}$ for ρ'_n being the run corresponding to β'_n is a run of S_k . Hence, ρ_k forms a basis of some behaviour β_k of S_k .

The transitions fired by processes $P_k = \{1, \dots, k\}$ in β_k are the same as the ones fired by these processes in β'_n , which means that at the end of β_k , the P_k processes will be at exactly the same control locations as at the end of β'_n . As ψ refers only to control locations of the visible processes, it is clear that $\pi_k, v_k \models \Diamond\neg\psi_k(p_1, \dots, p_k)$ holds for π_k corresponding to β_k and v_k such that $v_k(p_i) = i$ for $i \in \{1, \dots, k\}$. Thus, as we needed, we have shown existence of a counterexample in S_k with $1, \dots, k$ being visible via p_1, \dots, p_k . \square

It remains to use Lemma 6 to prove the final result for verification of prioritised RTR families against invariance MPTL formulae introduced at the beginning of the section.

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

Proof Similar to the proof of Theorem 1 with the use of Lemma 4 replaced by Lemma 6. \square

We now discuss yet another subclass of MPTL that can be handled within parametric finite-behaviour verification of RTR in the same way as above. This time, 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. In other words, we allow only the MPTL formulae in which the atomic formulae $at(p, q_1)$ and $at(p, q_i)$ for any transition $t = (q_1, (\mathfrak{p})\text{rcqt}(R'), q_2) \in T$ can be used only as subformulae of the formula $(at(p, q_1) \vee at(p, q_i))$. Let us denote such *location-restricted* MPTL formulae by Υ_a/Υ_e and their path subformulae by ν . 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).

Lemma 7 *Given an RTR family \mathcal{F} and a location-restricted MPTL path formula $\nu(p_1, \dots, p_k)$, the following holds for systems of \mathcal{F} :*

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

Proof The case of (\Rightarrow) is the same as in the proof of Lemma 6. Moreover, for the (\Leftarrow) case, we can use the same argument as in the proof of Lemma 6 to show that from a counterexample behaviour of S_n , we can obtain a behaviour of S_k with the processes $1, \dots, k$ to be made visible via p_1, \dots, p_k . The fact that we get a counterexample behaviour follows from ν not being able to detect the removal of transitions fired in invisible processes (ν refers to control locations of visible processes only and is stuttering-insensitive), the change of names of processes, nor the postponed firing of some transitions. The last claim holds because the postponed firing manifests itself just by some processes staying longer at locations q_1 before going to q_i for some transitions $t = (q_1, (\mathfrak{p})\text{rcqt}(R'), q_2) \in T$, which cannot be distinguished via $(at(p, q_1) \vee at(p, q_i))$. \square

Theorem 4 *Let \mathcal{F} be an RTR family and let $\Upsilon_a \equiv \forall p_1, \dots, p_k : A \ v(p_1, \dots, p_k)$ be a location-restricted MPTL formula. Then, checking whether $\mathcal{F}, l \models_{fin}^a \Upsilon_a$ holds is equal to checking whether $S_k \models_{fin} A \ v(1, \dots, k)$ holds.*

Proof We can use the same proof construction as in the case of Theorem 3 with references to Lemma 6 replaced by Lemma 7. □

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 $\mathcal{F}, l \models_{wf}^a \Phi_a$ 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 the prioritised resource management, we prove that 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. This is similar to the case of verification of finite behaviour of RTR, but for parametric verification of weakly fair behaviour, we further 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 $\Phi_a \equiv \forall p_1, \dots, p_k : A \ \varphi(p_1, \dots, p_k)$ 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, \dots, p_k may again be fixed to $1, \dots, k$. We denote the thus arising finite-state verification tasks as checking whether $S_n \models_{wf} A \ \varphi(1, \dots, k)$ holds.

To obtain the above result, we first prove that if a property holds in the system with $m + k$ processes, then it holds in any system with more than $m + k$ processes too. Alternatively, this also shows that if the property does not hold in a system with more than $m + k$ processes, then it does not hold in the system with $m + k$ processes either.⁴

Lemma 8 *For systems of an RTR\P family \mathcal{F} with m resources and an MPTL path formula $\varphi(p_1, \dots, p_k)$, the following holds:*

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

⁴Compared to Lemma 4, this result corresponds to the left implication. We now separate the left and right implications as their proofs are getting more complex.

Proof We show that when the assertion of the implication does not hold, the premise does not hold either. For any $n \geq m + k$, suppose there is a weakly fair behaviour β_n of S_n such that $\pi_n, v_n \models \neg\varphi(p_1, \dots, p_k)$ holds for the path π_n corresponding to β_n and some valuation $v_n : PV \rightarrow P_n$. We show that β_n can be reduced to a (weakly fair) counterexample in S_{m+k} . The reduction is partially similar to the one used in the proof of Lemma 4, but some auxiliary processes have to be additionally preserved to ensure that all the preserved processes that are eventually forever blocked in β_n are eventually forever blocked in the reduced counterexample too.

Let P' be the maximum set of processes containing elements p which are either visible in π_n (i.e. $v_n(p_i) = p$ for some $i \in \{1, \dots, k\}$), or there is a resource $r \in R$ such that from a certain point in π_n on, some queue item of p is permanently in the head of the queue of r . There are k visible processes, each queue can have at most one head item that is eventually always in this queue, and so $|P'| \leq m + k$. Let P be any set such that $P' \subseteq P$ and $|P| = m + k$.

Let $\xi_n : P_n \rightarrow P_n$ be any renaming of processes of S_n such that $\forall p \in P : 1 \leq \xi_n(p) \leq m + k$ and, moreover, $\xi_n(v_n(p_i)) = i$ for $1 \leq i \leq k$ (i.e. all the processes from P are numbered between 1 and $m + k$, and a processes visible via a process variable p_i is numbered i). Due to Lemma 2, $\beta'_n = \xi_n(\beta_n)$ is a behaviour of S_n , which is clearly complete and weakly fair as well. Moreover, for the run ρ'_n corresponding to β'_n , Lemma 3 implies that $\rho_{m+k} = (\rho'_n)^{\downarrow m+k}$ is a run of S_{m+k} . Let β_{m+k} be the behaviour corresponding to ρ_{m+k} . We show that β_{m+k} is weakly fair.

Since a transition enabled in a process remains enabled in a system of an RTR\P family till the process fires some transition, in a weakly fair behaviour of such a system, each process must either keep firing some transitions, or eventually never have an enabled transition. The processes from P that keep firing some transitions in ρ'_n do so in ρ_{m+k} as well—due to the construction of ρ_{m+k} , they keep firing exactly the same transitions. In a behaviour of RTR\P, a process may never have an enabled transition from a certain point on only when it reaches a terminal control location or when it is inside some `rq` transition whose `take` part never becomes enabled. The processes of P that reach a terminal control location in β'_n reach the same location in β_{m+k} too because they fire exactly the same transitions. Thus, as β'_n is weakly fair, it remains to show that the processes from P that are eventually forever blocked in some `rq` transitions in β'_n are eventually forever blocked before the appropriate `take` transitions in β_{m+k} too.

Let us consider a process $p \in P$ that is eventually forever blocked in some `rq`(R') in β'_n . As the appropriate `take` transition would be fired in β'_n if this was possible, and as no overtaking among queue items is allowed in RTR\P, at least one $r \in R'$ must have a head item produced by a request preceding the considered one and never removed from the queue of r in β'_n . However, according to the construction of P , the process p' that fired the latter request is in P , the transitions fired by p and p' in β_{m+k} (and their ordering) are equal to those of p and p' in β'_n , no transitions are added, and so the blocking is preserved.

Wrt. the same argument as in the proof of Lemma 4, it is now clear that $\pi_{m+k}, v_{m+k} \models \neg\varphi(p_1, \dots, p_k)$ for π_{m+k} corresponding to β_{m+k} and v_{m+k} such that $v_{m+k}(p_i) = i$ for $i \in \{1, \dots, k\}$. □

We next aim at proving a counterpart of Lemma 8 showing that if a property holds in a system with more than $m + k$ processes, then it holds in the system with $m + k$ processes too. That is, if a property does not hold in the system with $m + k$ processes, then it does not hold in any system with more than $m + k$ processes either. To be able to prove this fact, we need to show that any counterexample behaviour in the system with $m + k$ processes

may be expanded to a counterexample behaviour in an arbitrarily larger system. However, lifting a counterexample behaviour from a small system to a big one is now much more involved than in the case of safety properties. To ensure weak process fairness, newly added processes must be allowed to fire some transitions, but at the same time, this should not influence the behaviour of the visible processes. Below, we introduce three ways of expanding counterexample behaviours applicable under different circumstances, and prove that they preserve weak process fairness of the behaviours and do not influence the behaviour of visible processes. Then, we use this to obtain the desired counterpart of Lemma 8.

Assume that we are given a weakly fair behaviour β_n of a system S_n of an RTR\setminus P family \mathcal{F} such that $\pi_n, v_n \models \varphi(p_1, \dots, p_k)$ holds for the path π_n corresponding to β_n , an LTL\setminus X formula $\varphi(p_1, \dots, p_k)$, and a valuation v_n of the variables p_1, \dots, p_k . The first case we consider is that *there exists a process $p \in P_n$ that eventually never uses nor requests any resource in β_n* . Then, we expand the run ρ_n corresponding to β_n to a sequence of transitions $\rho_n^{\uparrow \varepsilon(p,l)}$ fired by $n+l$ processes for any $l \geq 1$ as follows. The additional processes initially one-by-one follow the transitions of p fired in ρ_n up to some point from which p never uses any resource any more. Afterwards, the original processes may start replaying the run ρ_n and, from the point where p is not using any resources any more, interleave it with further transitions fired by the newly added processes such that each time p fires a transition, all the new processes (ordered according to their numbers) fire the same transition.⁵ It is easy to prove the following fact.

Lemma 9 *Let us have a weakly fair behaviour β_n of a system S_n of an RTR\setminus P family \mathcal{F} such that $\pi_n, v_n \models \varphi(p_1, \dots, p_k)$ holds for the path π_n corresponding to β_n , an LTL\setminus X formula $\varphi(p_1, \dots, p_k)$, and a valuation v_n of the variables p_1, \dots, p_k . Let ρ_n be the run corresponding to β_n and let $p \in P_n$ be a process that eventually never uses nor requests any resource in β_n . Then, for any $l \geq 1$, $\rho_n^{\uparrow \varepsilon(p,l)}$ yields a weakly fair behaviour β_{n+l} of the system S_{n+l} such that $\pi_{n+l}, v_{n+l} \models \varphi(p_1, \dots, p_k)$ holds for the path π_{n+l} corresponding to β_{n+l} and $v_{n+l}(p_i) = v_n(p_i)$ for $i \in \{1, \dots, k\}$.*

Proof Immediate: As the additional processes do not access any resources after firing their initial transition sequence, they do not influence each other nor the original processes. Thus, $\rho_n^{\uparrow \varepsilon(p,l)}$ is a transition sequence fireable from $c_0 \in S_{n+l}$. The original processes fire exactly the same transitions as before, and none of their originally forever blocked transitions (if there are some like this) may get enabled by the new processes (that do neither use nor request any resources any more when the original processes start). Moreover, the new processes fire the same transitions as p and either terminate or go on firing some transitions forever. Hence, β_{n+l} is clearly a weakly fair behaviour whose visible part is the same as that of β_n , and the lemma holds. \square

Next, assume that in β_n *there appears a process $p \in P_n$ that eventually never fires any further transition while requesting some resources, but not using any resources, i.e. p blocks in β_n as its last request is never granted*. Then, we expand the run ρ_n corresponding to β_n to a sequence of transitions $\rho_n^{\uparrow \varepsilon(p,l)}$ fired by $n+l$ processes for any $l \geq 1$ as follows. The new processes first one-by-one fire the transitions fired by p in β_n besides the last one. Then, the original processes follow the run ρ_n up to the last request of p . Next, p fires its last request

⁵Note that a special case is that p terminates while not using nor requesting any resources. Then, all the new processes will terminate in the same way too.

followed by the new processes doing the same. Finally, the rest of ρ_n is followed by the original processes.

Lemma 10 *Let us have a weakly fair behaviour β_n of a system S_n of an RTR\(\mathcal{P}\) family \mathcal{F} such that $\pi_n, v_n \models \varphi(p_1, \dots, p_k)$ holds for the path π_n corresponding to β_n , an LTL\(\mathcal{X}\) formula $\varphi(p_1, \dots, p_k)$, and a valuation v_n of the variables p_1, \dots, p_k . Let ρ_n be the run corresponding to β_n and let $p \in P_n$ be a process that eventually never fires any further transitions while requesting some resources, but not using any resources. Then, for any $l \geq 1$, $\rho_n^{\uparrow b(p,l)}$ yields a weakly fair behaviour β_{n+l} of the system S_{n+l} such that $\pi_{n+l}, v_{n+l} \models \varphi(p_1, \dots, p_k)$ holds for the path π_{n+l} corresponding to β_{n+l} and $v_{n+l}(p_i) = v_n(p_i)$ for $i \in \{1, \dots, k\}$.*

Proof Firing $\rho_n^{\uparrow b(p,l)}$ up to the point when p fires its last request is possible because in β_n , p gets blocked without being the user of any resources, and so just before its last request, it is not involved with any resources—and the same holds for the newly added processes too. Next, as the blocked request of p does not hamper firing the rest of ρ_n within β_n and as blocking exactly the same resources several times does not change anything, the rest of $\rho_n^{\uparrow b(p,l)}$ is firable too after p and the new processes get blocked. Thus, $\rho_n^{\uparrow b(p,l)}$ yields a behaviour β_{n+l} of S_{n+l} . Moreover, p and the new processes are eventually never enabled, the remaining processes fire the same transitions as before, and none of their originally forever blocked transitions (if there are some like this) can get enabled by the newly added processes (which just block resources that were blocked originally too). Hence, β_{n+l} is clearly weakly fair. At the same time, the behaviour of the visible processes is unchanged. \square

Finally, assume that β_n loops through some configuration $c \in C_n$ where some invisible process $p \in P_n$ does not own nor request any resources (i.e. β_n consists of a stem and a loop in which p releases all resources it acquired before asking for some resources again). Then, we expand the run ρ_n corresponding to β_n to a sequence of transitions $\rho_n^{\uparrow x(p,l)}$ fired by $n+l$ processes for any $l \geq 1$ as follows. The additional processes first one-by-one follow the transitions of p up to the first occurrence of c in the loop of β_n . Then, the original processes replay the run corresponding to the part of β_n leading up to the point when β_n starts looping through c . Subsequently, a loop is formed by repeating $l+1$ times the loop of β_n , first with p firing its transitions, and the l new process idling, then with the first new process playing the role of p , and so on. It is then easy to see that the following holds.

Lemma 11 *Let us have a weakly fair behaviour β_n of a system S_n of an RTR\(\mathcal{P}\) family \mathcal{F} that ends by a loop that repeatedly visits a configuration $c \in C_n$ where some process $p \in P_n$ does not own nor request any resources. Suppose that $\pi_n, v_n \models \varphi(p_1, \dots, p_k)$ holds for the path π_n corresponding to β_n , an LTL\(\mathcal{X}\) formula $\varphi(p_1, \dots, p_k)$, and a valuation v_n of the variables p_1, \dots, p_k such that $v_n(p_i) \neq p$ for any $i \in \{1, \dots, k\}$. Let ρ_n be the run corresponding to β_n . Then, for any $l \geq 1$, $\rho_n^{\uparrow x(p,l)}$ yields a weakly fair behaviour β_{n+l} of the system S_{n+l} such that $\pi_{n+l}, v_{n+l} \models \varphi(p_1, \dots, p_k)$ holds for the path π_{n+l} corresponding to β_{n+l} and $v_{n+l}(p_i) = v_n(p_i)$ for $i \in \{1, \dots, k\}$.*

Proof At the end of their initial transition sequence, the newly added processes are not using nor requesting any resource, and so they do not prevent each other from firing this sequence nor later the original processes from replaying the run corresponding to the initial part of β_n up to c . Moreover, the newly formed loop is firable because p and the additional processes do not block any resource when idling. Thus, we obtain a transition firing sequence in which

all the newly added processes are running, p is running, the other original processes fire the transitions they used to, and none of their originally forever blocked transitions (if there are some like this) can get enabled by the new processes (whose behaviour is from the point of view of the original processes other than p indistinguishable from the one of p). Hence, $\rho_n^{\uparrow x(p,l)}$ yields a weakly fair behaviour β_{n+l} of S_{n+l} . At the same time, the behaviour of the visible processes is unchanged. \square

We are now ready to state the counterpart of Lemma 8 showing that if a property holds in a system with more than $m + k$ processes, then it holds in the system with $m + k$ processes too (or, alternatively, that if a property does not hold in the system with $m + k$ processes, then it does not hold in any system with more than $m + k$ processes either).

Lemma 12 *For systems of an RTR\mathcal{P} family \mathcal{F} with m resources and an MPTL path formula $\varphi(p_1, \dots, p_k)$, the following holds:*

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

Proof We show that when the assertion of the implication does not hold, the premise does not hold either. Assume that we have a weakly fair behaviour β_{m+k} of S_{m+k} such that $\pi_{m+k}, v_{m+k} \models \neg\varphi(p_1, \dots, p_k)$ for the path π_{m+k} corresponding to β_{m+k} and the valuation v_{m+k} such that $v_{m+k}(p_i) = i$ for $i \in \{1, \dots, k\}$. We show that for any $n \geq m + k$, the run ρ_{m+k} corresponding to β_{m+k} can be extended by transitions fired by the processes of S_n not present in S_{m+k} in such a way that (1) we obtain a weakly fair behaviour of S_n , and (2) the sequence of transitions fired by the visible processes is not changed. This then means that a counterexample behaviour exists in S_n too.

We distinguish three cases: (1) No process is eventually running in β_{m+k} . (2) All processes keep firing forever some transitions in β_{m+k} . (3) At least one process keeps firing some transitions in β_{m+k} .

Case 1. This is the simplest case. There is either a process that terminates while not using nor requesting any resources, and then Lemma 9 can be applied. Otherwise, as there are $m + k$ processes but only m resources, there is at least one process that blocks while not owning any resource, which allows Lemma 10 to be applied.

Case 2. As S_{m+k} is a finite-state system and $\neg\varphi(p_1, \dots, p_k)$ for a fixed valuation of p_1, \dots, p_k is a common LTL\mathcal{X} formula, with no loss of generality, we may suppose that β_{m+k} consists of an initial part and a repeating part (loop). Clearly, if a given counterexample behaviour is not of such a form, we can find another that is. We consider two sub-cases: No resource is ever released in the loop, or at least one resource is always eventually released.

If no resource is ever released in the loop of β_{m+k} , at most m processes can use some resources in the loop. Since there are $m + k$ processes, at least k processes never use (and since all of them are running, nor request) any resource in the loop. Then, Lemma 10 can be applied.

If at least one resource is always eventually released in the loop of β_{m+k} , then there is a configuration c in the loop where at least one resource is not in use. This means that at most $m - 1$ processes may use some resources in c . Therefore, at least $k + 1$ processes are not using any resource in c . Hence, there is an invisible process p not using anything in c . In RTR\mathcal{P}, to get to a state where it is not using anything and it asking for something, p must first release all the resources it is using and only then request something. Consequently, in the loop, c must be preceded by c' in which p is not using nor requesting any resource. This allows Lemma 11 to be applied.

Case 3. As in the previous case, we may suppose that β_{m+k} consists of an initial part and a loop. Let b be the number of processes blocked in β_{m+k} . If at least one process p out of the b processes is not using any resource (it is at most requesting some) after it fires its last transition, Lemma 9 or 10 can be applied.

It remains to examine the case when all of the b processes are using some resources when they definitively block in β_{m+k} . Then, we clearly have $b \leq m$, and at least b resources cannot be used nor requested by the processes that keep forever looping in β_{m+k} —a multiple use of resources is excluded and these processes never block forever. This means that at most $m - b = m'$ resources are available for these processes. The number of such processes is $m + k - b = m' + k$. We can then use the same argument over m' and $m' + k$ as over m and $m + k$ in Case 2 to show that the new processes can mimic the behaviour of some original process without influencing the visible processes, which concludes the proof. (If $m' = 0$, which is excluded for m , the running processes eventually never use nor request any resources, and so the first subcase of Case 2 applies here.) \square

Now, the final theorem for verification of weakly fair behaviours of RTR\P families easily follows from the lemmas we proved and the properties of MPTL.

Theorem 5 *Let \mathcal{F} be an RTR\P family with m resources and let $\Phi_a \equiv \forall p_1, \dots, p_k : A \varphi(p_1, \dots, p_k)$ be an MPTL formula. Then, checking whether $\mathcal{F}, l \models_{wf}^a \Phi_a$ holds is equal to checking whether $S_n \models_{wf} A \varphi(1, \dots, k)$ holds for all $S_n \in \mathcal{F}$ such that $\min(\max(l, k), m + k) \leq n \leq m + k$.*

Proof First, for all $S_{n'}$ such that $l \leq n' < k$, $S_{n'} \models_{wf} \Phi_a$ is trivially satisfied—there is no choice of k distinct processes in such systems.

Next, for the (\Rightarrow) case, if $S_{n'} \models_{wf} \forall p_1, \dots, p_k : A \varphi(p_1, \dots, p_k)$ holds for all $n' \geq l$, and $l \leq m + k$, the satisfaction of $S_n \models_{wf} A \varphi(1, \dots, k)$ for $\max(l, k) \leq n \leq m + k$ is covered already by the premise. If $l > m + k$, Lemma 12 implies that $S_{m+k} \models_{wf} A \varphi(1, \dots, k)$ holds.

Finally, for the (\Leftarrow) case, if there are some $S_{n'} \in \mathcal{F}$ with $\max(l, k) \leq n' \leq m + k$, they are covered directly by exploiting just the interchangeability of processes. Moreover, if $S_{m+k} \models_{wf} A \varphi(1, \dots, k)$ holds, Lemma 8 implies $S_{n'} \models_{wf} \forall p_1, \dots, p_k : A \varphi(p_1, \dots, p_k)$ holds for all $n' \geq m + k$, which covers all the remaining (i.e. directly not covered) cases or even more (for $l > m + k$). \square

Let us note that examining the systems S_k (if $l \leq k$) and S_{m+k} is necessary for the general result presented in Theorem 5: Fig. 2(a) shows the RTR\P automaton of a simple family for which $\forall p : A \diamond \square \neg at(p, x)$ does not hold in S_k (i.e. S_1), but it holds in all bigger systems. Figure 2(b) shows the automaton of a simple family for which $\forall p : A \diamond (at(p, x) \vee at(p, y))$ does not hold in S_{m+k} (i.e. S_3 where two processes may deadlock and another will not even pass the first request), but it holds in any smaller system. The question of a potential further optimisation of the presented 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 a 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,

Fig. 2 Two simple RTR\P automata

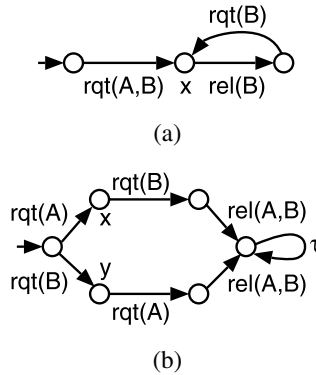
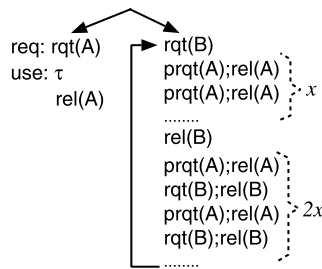


Fig. 3 An RTR automaton problematic for verification of absence of starvation



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.

Theorem 6 *For RTR families \mathcal{F} with m resources and the property of absence of starvation expressed as $\Phi_a \equiv \forall p : A \square (at(p, req) \Rightarrow \diamond at(p, use))$, the problem of checking whether $\mathcal{F}, 1 \models_{wf}^a \Phi_a$ holds cannot, in general, be solved by examining just the systems $S_1, \dots, S_n \in \mathcal{F}$ with n being a function of m only.*

Proof Let us consider the RTR family whose RTR automaton is depicted in Fig. 3 and that has $3x + 2$ transitions in its right branch. As we show below, at least $x + 3$ processes (i.e. a number depending on the structure of the control automaton, but not on m) is needed to witness starvation in the given family.

The visible process goes to the left branch, requests A, and only if there are (at least) $x + 2$ invisible processes running in the right branch, the visible process can stay blocked forever: Inside the upper part of the right branch (between $rqt(B)$ and $rel(B)$) there can only be one invisible process, but each time this process releases A, another process in the bottom part of the right branch has to already be inside some $prqt(A)$ action to keep the

visible process blocked. At the same time, due to the $\text{rqt}(B); \text{rel}(B)$ actions, no process can go through two $\text{prqt}(A)$ actions in the bottom part of the right branch while a single process is passing the upper part. Thus, one process in the bottom part is needed to block A while another process—let us say p —is entering the upper part, then $x - 1$ processes to keep A blocked while p is inside the upper part, and, finally, one further process in the bottom part to block A while p is leaving the upper part. \square

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_{ex} 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 always owns the same resources at a given control location⁶) 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 $B_{\mathcal{F}} = CM_C(M_C + 1)(2FC(M_C + 1))^F + 2C(M_C + 1) + 2m + 1$.

In order to formally prove that the parametric verification problem for weakly fair behaviours of RTR families against single-process MPTL formulae can be reduced to verification of systems with up to $B_{\mathcal{F}}$ processes, we first state several auxiliary results on loop behaviours. Then, we give several lemmas needed to prove the key cut-off Lemma 22 showing 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. Then, we present Lemma 23 stating the opposite. These two lemmas together give us the desired result captured by Theorem 7.

Basic results on loop behaviours A loop behaviour is a behaviour

$$\beta_n = c_1(p_1, t_1)c_2(p_2, t_2) \dots (p_{L-1}, t_{L-1})c_L$$

starting from $c_1 \in C_n$ with $c_1 = c_L$.

Before stating some further needed results on loop behaviours, we first introduce an equivalence relation \sim on configurations C_n of RTR families relating configurations being equal up to renaming. Formally, we define $c_1 \sim c_2$ to hold for $c_1, c_2 \in C_n$ iff there exists a renaming ξ_n such that $\xi_n(c_1) = c_2$. The equivalence class of a configuration c is denoted by $[c]$. Let \bar{c} be a unique representative of the equivalence class $[c]$. We will later define the unique representative explicitly. Let \bar{C}_n be the set of all representatives of equivalence classes of C , i.e. the quotient of C_n .

Now, a quotient behaviour $\bar{\gamma}_n$ starting at \bar{c}_1 is a sequence $\bar{c}_1 t_1 \bar{c}_2 t_2 \dots$ where $\bar{c}_i \in \bar{C}_n$, $t_i \in T_{ex}$ and $\exists c_i \in C_n, p_i \in \{1, \dots, n\}$ with $c_i \in [c_i]$ such that $c_1(p_1, t_1)c_2(p_2, t_2) \dots$ is a behaviour of S_n starting from $c_1 \in C_n$. In the following, if the number of processes n in a quotient behaviour $\bar{\gamma}_n$ is clear from the context, we will drop it and just speak about a

⁶If this is not the case, then one can construct from \mathcal{A} another RTR automaton \mathcal{A}' with control states Q'_{ex} where the condition holds. One can use a finite memory to remember for each original control state the set of resources owned. Notice that $|Q'_{ex}|$ is bounded by $2^m |Q_{ex}|$.

quotient behaviour $\bar{\gamma}$. The length of a quotient behaviour $\bar{\gamma}$ is the number of transitions it contains. A *quotient loop* is a finite quotient behaviour $\bar{c}_1 t_1 \bar{c}_2 t_2 \dots \bar{c}_L$ starting at \bar{c}_1 with $\bar{c}_1 = \bar{c}_L$.

Lemma 13 *Given a quotient loop $\bar{\gamma}_n$, there is a loop behaviour γ_n .*

Proof A quotient loop $\bar{\gamma}_n = \bar{c}_1 t_1 \bar{c}_2 t_2 \dots t_{L-1} \bar{c}_L$ with $\bar{c}_1 = \bar{c}_L$ corresponds to a finite behaviour $\beta_n = c_1(p_1, t_1)c_2(p_2, t_2) \dots (p_{L-1}, t_{L-1})c_L$ starting at $c_1 \in C_n$ with $[c_1] = [c_L]$. This means that there exists a permutation σ with $\sigma(c_1) = c_L$ but not necessarily $c_1 = c_L$. We can obtain a finite behaviour which is a loop by repeating again and again the same (up to permutation of process names) behaviour β_n . In this way, we get a sequence of configurations $c_L, c_{2L}, c_{3L}, \dots$ with $c_{j*L} \in [c_L]$. Because $[c_L]$ is finite, there are k, m with $c_{kL} = c_{mL}$. The behaviour between these two configurations is a loop. \square

We will further be especially interested in the so-called simple behaviours. A *simple behaviour* is a behaviour where in all configurations in each queue there is at most one pr or r item. In these behaviours there can be at most one process at a given location $q \in Q_o$, and an arbitrary number of processes can be at a given location $q \in Q_n$. Furthermore, from the control locations of processes, one can uniquely determine the entries in the queues of resources. Therefore, we can define as a unique representative of an equivalence class $[c]$ a couple (q, \mathbf{x}) , where $q \in 2^{Q_o}$ is the set of all locations in Q_o occupied by some process in c , and $\mathbf{x} \in \mathbb{N}^F$ with $F = |Q_n|$ gives for each location in Q_n the *number* of processes which occupy this location. From now on, we consider quotient behaviours $\bar{\gamma}$ as sequences of the form $(q_1, \mathbf{x}_1)t_1(q_2, \mathbf{x}_2)t_2 \dots (q_L, \mathbf{x}_L)$. The *effect* of $\bar{\gamma}$ is defined as $\text{eff}(\bar{\gamma}) = \mathbf{x}_L - \mathbf{x}_1$. Obviously, if $\bar{\gamma}$ is a quotient loop, $\text{eff}(\bar{\gamma}) = \mathbf{0}$. The absolute value of each component of $\text{eff}(\bar{\gamma})$ is bounded by $L - 1$. The composition of two quotient behaviours $\bar{\gamma} = (q_1, \mathbf{x}_1)t_1(q_2, \mathbf{x}_2)t_2 \dots (q_L, \mathbf{x}_L)$ and $\bar{\gamma}' = (p_1, \mathbf{y}_1)t'_1(p_2, \mathbf{y}_2)t'_2 \dots (p_M, \mathbf{x}_M)$ is possible if $(q_L, \mathbf{x}_L) = (p_1, \mathbf{y}_1)$ and is defined by $\bar{\gamma}\bar{\gamma}' = (q_1, \mathbf{x}_1)t_1(q_2, \mathbf{x}_2)t_2 \dots (q_L, \mathbf{x}_L)t'_1(p_2, \mathbf{y}_2)t'_2 \dots (p_M, \mathbf{x}_M)$. We have $\text{eff}(\bar{\gamma}\bar{\gamma}') = \text{eff}(\bar{\gamma}) + \text{eff}(\bar{\gamma}')$. Given a quotient behaviour $\bar{\gamma}$, the corresponding *projected quotient behaviour* $\hat{\gamma}$ is the sequence $q_1 t_1 q_2 t_2 \dots q_L$. Furthermore, we call $\bar{\gamma}$ a *structural loop* if $q_1 = q_L$. If $\bar{\gamma}$ is a quotient behaviour $(q_1, \mathbf{x}_1)t_1(q_2, \mathbf{x}_2)t_2 \dots (q_L, \mathbf{x}_L)$, then *plus* $(\bar{\gamma}, \mathbf{d})$ where $\mathbf{d} \in \mathbb{N}^F$ is defined to be the quotient behaviour $(q_1, \mathbf{x}_1 + \mathbf{d})t_1(q_2, \mathbf{x}_2 + \mathbf{d})t_2 \dots (q_L, \mathbf{x}_L + \mathbf{d})$.

The following lemma shows that a quotient loop which contains two identical structural loops can be rearranged in such a way that the two identical structural loops follow each other.

Lemma 14 *Let π be a quotient loop of the form $\bar{\gamma}_1 \bar{\gamma}_2 \bar{\gamma}_3 \bar{\gamma}_4 \bar{\gamma}_5$ where $\hat{\gamma}_2$ is a structural loop and $\hat{\gamma}_4 = \hat{\gamma}_2$. Let the first element of $\bar{\gamma}_2$ be (q_2, \mathbf{d}) . Then,*

$$\bar{\gamma}'_n = \text{plus}(\bar{\gamma}_1, \mathbf{d}) \text{plus}(\bar{\gamma}_2, \mathbf{d}) \text{plus}(\bar{\gamma}_2, \mathbf{d} + \text{eff}(\bar{\gamma}_2)) \text{plus}(\bar{\gamma}_3, \mathbf{d} + \text{eff}(\bar{\gamma}_2)) \text{plus}(\bar{\gamma}_5, \mathbf{d})$$

is also a quotient loop.

Proof Obvious by observing that the different parts of the behaviour can be composed. We have to add \mathbf{d} to the vectors to make sure that they stay non-negative ($\text{eff}(\bar{\gamma}_2)$ is not necessarily non-negative). \square

Recall that $C = 2^{|Q_o|}$. $M_C = C^C$ is an upper bound on the number of different structural loops of length at most C . The following lemma shows that a quotient loop $\bar{\gamma}$ can always be rearranged such that the same short structural loops in $\bar{\gamma}$ follow each other.

Lemma 15 *Let $\bar{\gamma}$ be a quotient loop. Then, there is a D with $1 \leq D \leq M_C$ such that for $1 \leq i \leq D$, there are values $L_i > 0$ such that we can obtain a quotient loop of the same length as $\bar{\gamma}$ of the form*

$$\bar{\gamma}_1 \bar{\delta}_1^1 \bar{\delta}_1^2 \dots \bar{\delta}_1^{L_1} \bar{\gamma}_2 \bar{\delta}_2^1 \bar{\delta}_2^2 \dots \bar{\delta}_2^{L_2} \dots \bar{\gamma}_D \bar{\delta}_D^1 \bar{\delta}_D^2 \dots \bar{\delta}_D^{L_D} \bar{\gamma}_{D+1}$$

where:

- $\forall i \in \{1, \dots, D\} \forall j \in \{1, \dots, L_i\} : \widehat{\delta}_i^j$ is a structural loop of length at most C and $\widehat{\delta}_i^1 = \widehat{\delta}_i^j$.
- $\forall i, j : i \neq j \Rightarrow \forall k, l : \widehat{\delta}_i^k \neq \widehat{\delta}_j^l$.
- Each $\bar{\gamma}_i$ has length smaller than C .

Proof Repeated application of Lemma 14 and the fact that a quotient behaviour of length at least C contains a structural loop. □

Reducing the size of weakly fair counterexamples We now prove one more auxiliary lemma and then state a key result allowing us to reduce the size of loops of weakly fair counterexample behaviours.

Lemma 16 *Let \mathbf{A} be an $M \times N$ (with $M \leq N$) integer matrix, \mathbf{B} an $M \times 1$ integer matrix, G a bound on the absolute values of entries of \mathbf{A} and \mathbf{B} . If there exists a nonnegative integer solution \mathbf{x} to the equation $\mathbf{A}\mathbf{x} = \mathbf{B}$, then there exists a nonnegative integer solution where the value of all its entries is bounded by $(N + 1)(MG)^M$.*

Proof The lemma follows directly from [14]. □

Lemma 17 *If there is a quotient loop $\bar{\gamma}$ of length bigger than $S = CM_C(M_C + 1)(2FC(M_C + 1))^F + 2C(M_C + 1)$ starting from a configuration (\mathbf{q}, \mathbf{x}) , then there is a quotient loop $\bar{\gamma}'$ of length smaller than or equal to S starting from a configuration $(\mathbf{q}, \mathbf{x}')$. Furthermore, (1) for all configurations (\mathbf{p}, \mathbf{y}) in $\bar{\gamma}'$, we have $|\mathbf{p}| + \sum_{i=1}^F \mathbf{y}_i \leq S$, (2) a transition appearing in $\bar{\gamma}$ appears at least once in $\bar{\gamma}'$, and (3) $\forall i$ with $1 \leq i \leq F$, there exists a configuration (\mathbf{p}, \mathbf{y}) in $\bar{\gamma}'$ with $\mathbf{y}_i = 0$.*

Proof We transform $\bar{\gamma}$ using Lemma 15 into

$$\bar{\gamma}_1 \bar{\delta}_1^1 \bar{\delta}_1^2 \dots \bar{\delta}_1^{L_1} \bar{\gamma}_2 \bar{\delta}_2^1 \bar{\delta}_2^2 \dots \bar{\delta}_2^{L_2} \dots \bar{\gamma}_D \bar{\delta}_D^1 \bar{\delta}_D^2 \dots \bar{\delta}_D^{L_D} \bar{\gamma}_{D+1}$$

Obviously, we have $\forall i \in \{1, \dots, D\} \forall j \in \{1, \dots, L_i\} : \text{eff}(\bar{\delta}_i^j) = \text{eff}(\bar{\delta}_i^1)$. We define the matrices $\mathbf{A} = (\text{eff}(\bar{\delta}_1^1), \dots, \text{eff}(\bar{\delta}_D^1))$ and $\mathbf{B} = -(\sum_{i=1}^D \text{eff}(\bar{\gamma}_i \bar{\delta}_i^1)) - \text{eff}(\bar{\gamma}_{D+1})$. Let $\mathbf{x} = (L_1 - 1, L_2 - 1, \dots, L_D - 1)^T$. The absolute value of each entry of \mathbf{B} is bounded by $2(M_C + 1)C$ and that of \mathbf{A} by C —the length of all the $\bar{\gamma}_i$ s and $\bar{\delta}_i^j$ s is at most C , and each step decreases by 1 the number of processes at one control location and increases it by 1 at another location. Then, since $\bar{\gamma}$ is a loop, $\text{eff}(\bar{\gamma}) = \mathbf{0}$, i.e. we have $\mathbf{A}\mathbf{x} = \mathbf{B}$. Applying Lemma 16 gives us a small solution \mathbf{x} of the equation $\mathbf{A}\mathbf{x} = \mathbf{B}$. From this solution, which indicates how many times the structural loops have to be repeated, we can easily construct the quotient loop $\bar{\gamma}'$ of the required length: Every $\widehat{\delta}_i^1$ is fired at most $(M_C + 1)(2FC(M_C + 1))^F$ times within \mathbf{x} , we have at most M_C different $\widehat{\delta}_i^1$ s, the length of each $\widehat{\delta}_i^1$ is at most C , and we add the effect \mathbf{B} . Each $\widehat{\delta}_i^1$ will be performed at least once, therefore each transition appearing in $\bar{\gamma}$ appears at least once in $\bar{\gamma}'$ (Condition 2). $\bar{\gamma}'$ has a length smaller than or equal to S . This

means that at most S processes perform a transition in the loop. Those who do not perform any step can be removed (Condition 1)—they do not block any resources and thus do not influence any other processes (if they did, as they are in a loop, the other processes could not run at all either). Furthermore, if Condition 3 is not satisfied for some i , then a process from component i is removed and we still have a quotient loop. \square

The following lemmas are needed in the proof of the cut-off Lemma 22. They allow in certain cases to reduce the number of processes needed to witness a weakly fair counterexample to an $LTL \setminus X$ formula.

Let $\rho_n = (p_1, t_1)(p_2, t_2) \dots$ be an infinite run of a system S_n of an RTR family \mathcal{F} . Let k with $1 \leq k < n$ such that all processes 1 to k are running forever in ρ_n . We denote by $\rho_n^{\downarrow a(k)}$ the sequence obtained from $\rho_n^{\downarrow k}$ by moving all occurrences of $(p) \text{r}\in\text{q}$ transitions to the right such that they get just before firing of the appropriate $(p) \text{t}\text{a}\text{k}\text{e}$ transitions. Since all processes in $\rho_n^{\downarrow k}$ are running forever, this is always possible. We can prove the following result.

Lemma 18 *For any infinite run $\rho_n = (p_1, t_1)(p_2, t_2) \dots$ of a system S_n of an RTR family \mathcal{F} and k with $1 \leq k < n$ such that all processes 1 to k are running forever in ρ_n , $\rho_n^{\downarrow a(k)}$ is a run of S_k .*

Proof Exactly like the proof of Lemma 5. \square

Lemma 19 *Let us have a weakly fair behaviour β_n of a system S_n of an RTR family \mathcal{F} such that $\pi_n, v_n \models \varphi(p)$ holds for the path π_n corresponding to β_n , an $LTL \setminus X$ formula $\varphi(p)$, and a valuation v_n of the variable p . Let ρ_n be the run corresponding to β_n . If $v_n(p)$ is running forever in ρ_n and more than $m + 1$ processes are running forever in ρ_n , then there is a weakly fair behaviour β_{m+1} in the system S_{m+1} such that for the corresponding path π_{m+1} and some valuation $v_{m+1} : \{p\} \rightarrow P_{m+1}$, we have $\pi_{m+1}, v_{m+1} \models \varphi(p)$.*

Proof We preserve $m + 1$ forever running processes including the visible one. The run ρ_n of β_n is transformed into ρ_{m+1} by keeping transitions of the $m + 1$ running processes only. Formally, let $P = \{v_n(p), p_2, \dots, p_{m+1}\}$ be a set with $m + 1$ processes running in β_n . Let $\xi_n : P_n \rightarrow P_n$ be any renaming of processes of S_n such that $\xi_n(p) = 1$ and $\xi_n(p_i) = i$ for $2 \leq i \leq m + 1$. Let $\beta'_n = \xi_n(\beta_n)$, π'_n be the path associated with β'_n , and v'_n be such that $v'_n(p) = 1$. Due to Lemma 2, we have that $\beta'_n = \xi_n(\beta_n)$ is a behaviour of S_n , which is clearly weakly fair as well. Moreover, for the run ρ'_n corresponding to β'_n , Lemma 18 implies that the sequence $\rho_{m+1} = (\rho'_n)^{\downarrow a(m+1)}$ is a run of S_{m+1} . Hence, ρ_{m+1} forms a basis of some behaviour β_{m+1} of S_{m+1} . β_{m+1} is weakly fair because all processes in β_{m+1} are running forever. Furthermore, for the corresponding π_{m+1} , we have $\pi_{m+1}, v_{m+1} \models \varphi(p)$ since the visible process is doing exactly the same actions in β_{m+1} as in β_n . \square

Lemma 20 *Let us have a weakly fair behaviour β_n of a system S_n of an RTR family \mathcal{F} such that $\pi_n, v_n \models \varphi(p)$ holds for the path π_n corresponding to β_n , an $LTL \setminus X$ formula $\varphi(p)$, and a valuation v_n of the variable p . Let ρ_n be the run corresponding to β_n . If $v_n(p)$ is running forever in ρ_n and strictly less than $m + 1$ processes are running forever in ρ_n , then there is a weakly fair behaviour $\beta_{n'}$ in a system $S_{n'}$ with $m + 1 \leq n' \leq B_{\mathcal{F}}$ such that for the corresponding $\pi_{n'}$ and some valuation $v_{n'} : \{p\} \rightarrow P_{n'}$, we have $\pi_{n'}, v_{n'} \models \varphi(p)$.*

Proof To obtain the weakly fair behaviour $\beta_{n'}$, some blocked processes have to be preserved. We transform the run ρ_n of β_n to a run $\rho_{n'}$ where a set P of n' processes is preserved. P contains the following processes: First, the forever running processes (including the visible one) and the processes that get blocked while being users of some resources (i.e. owning u items in the queues of some resources). Second, for each resource, the blocked process (if it exists) that has the first pr item in the resource's queue, and if it does not exist, the blocked process (if it exists) that has the first r item in the resource's queue. There are at most $2m < B_{\mathcal{F}}$ processes of this kind. If the total number of these processes is less than $m + 1$, then we add arbitrarily some more blocked processes. Let $P = \{v_n(p), p_2, \dots, p_{n'}\}$. Let $\xi_n : P_n \rightarrow P_n$ be any renaming of processes of S_n such that $\xi_n(p) = 1$ and $\xi_n(p_i) = i$ for $2 \leq i \leq n'$. Let $\beta'_n = \xi_n(\beta_n)$, π'_n be the path associated with β'_n , and v'_n be such that $v'_n(p) = 1$. Due to Lemma 2, $\beta'_n = \xi_n(\beta_n)$ is a behaviour of S_n , which is clearly weakly fair as well.

Now, we show how to obtain $\rho_{n'}$ starting from β'_n . Since S_n is a finite-state system, without loss of generality, we can suppose that the behaviour β'_n is given by a prefix α'_n followed by a behaviour γ'_n repeated infinitely often. It is clear that the blocked processes in β'_n are either at a location where they cannot do any actions any more or at an internal control location of some $(p)req$ action having executed as their last transition a $(p)req$ transition. Obviously, they execute this transition in the prefix α'_n . We will transform the run ρ'_n of β'_n to $\rho_{n'}$ by first considering the sequence of transitions $(\rho'_n)^{\downarrow n'}$ of the processes in P . This sequence is not necessarily a run. We define the sequence $(\rho'_n)^{\downarrow si(n')}$ as follows. Transitions from $(\rho'_n)^{\downarrow n'}$ are reordered in the following way: First, occurrences of all $(p)req$ transitions are postponed such that they get just before the corresponding $(p)take$ transitions (if they exist). Then, all $preq$ transitions whose $ptake$ were enabled by a $rel(R)$ in β_n are moved before that $rel(R)$. The $(p)req$ transitions of blocked processes are postponed to the end of α_n preserving their original order with the $preq$ transitions coming before the req transitions.

Then, as in the proof of Lemma 5, it is easy to see that $\rho_{n'} := (\rho'_n)^{\downarrow si(n')}$ is a run from which we obtain a behaviour $\beta_{n'}$. Moreover, $\beta_{n'}$ is weakly fair because out of the preserved processes, the originally running ones keep running (they do the same actions), and the blocked ones remain blocked: If they were blocked by some blocked process, they remain blocked because for each resource r , one (preferably high-priority) blocked process blocking r is preserved—if such a process exists. The prioritised requests are preferred (and fired first) because they can block both types of requests, and thus it is not necessary to preserve up to two blocking and blocked requests per resource, which could otherwise happen with a queue content $\Sigma_n^*r(1)\Sigma_n^*r(2)\Sigma_n^*pr(3)\Sigma_n^*pr(4)\Sigma_n^*$ where 2, 4 are to be preserved, and 2 is blocked by 1 and 4 by 3. Blocking by running processes is also preserved because the blocking $preq$ is always fired before the appropriate resources are released (blocking by overtaking req is impossible because they cannot overtake). Finally, for $\pi_{n'}$ corresponding to $\beta_{n'}$, we have $\pi_{n'}, v_{n'} \models \varphi(p)$ since the visible process is doing exactly the same actions in $\beta_{n'}$ as in β_n . □

Lemma 21 *Let us have a weakly fair behaviour β_n of a system S_n of an RTR family \mathcal{F} such that $\pi_n, v_n \models \varphi(p)$ holds for the path π_n corresponding to β_n , an LTL\X formula $\varphi(p)$, and a valuation v_n of the variable p . Let ρ_n be the run corresponding to β_n . If $v_n(p)$ is blocked in ρ_n and some invisible processes are running forever in ρ_n , then there is a weakly fair behaviour $\beta_{n'}$ in a system $S_{n'}$ with $m + 1 \leq n' \leq B_{\mathcal{F}}$ such that for the corresponding $\pi_{n'}$ and some valuation $v_{n'} : \{p\} \rightarrow P_{n'}$, we have $\pi_{n'}, v_{n'} \models \varphi(p)$.*

Proof If the visible process is blocked and some invisible processes are running, then the visible process p is either blocked because some resource is used forever, or due to high priority requests of other processes, p is never enabled (if this was not the case, the behaviour would be unfair). Therefore, we cannot just take a behaviour with only the visible process. As in Lemma 20, we can suppose that the behaviour β_n is given by a prefix α_n followed by a behaviour γ_n repeated infinitely often.

The proof is done in two parts. First, we show that the transitions of β_n can be reordered in such a way that we get a *simple* behaviour (not taking into account blocked processes). Then, we show that for this behaviour there is one using at most $B_{\mathcal{F}}$ processes and more than $m + 1$ ones.

First, some invisible blocked processes and all invisible running processes are preserved and occurrences of transitions are reordered. The same blocked processes as in Lemma 20 are preserved and β_n is reordered as in Lemma 20. We obtain an infinite behaviour β_k where some blocked processes have been removed. β_k is composed of a prefix α_k and a loop behaviour γ_k (starting from a configuration c). Notice that the preserved blocked processes do not influence the behaviour of the running processes. Thus, they can be dropped from the behaviour γ_k to get a *simple* loop behaviour $\gamma_{k'}$. Thus, $\overline{\gamma_{k'}}$ is a quotient loop. Using Lemma 17, we obtain a quotient loop $\overline{\gamma'}$ using at most $B_{\mathcal{F}} - 2m - 1$ processes. Using Lemma 13, we obtain a loop behaviour γ' starting from a configuration c' with at most $B_{\mathcal{F}} - 2m - 1$ processes (due to Condition 1 of Lemma 17). Furthermore, the same control locations of Q_o are occupied in c and c' . All processes which own or request resources in c' are running because all transitions appearing in $\overline{\gamma_{k'}}$ appear also in $\overline{\gamma'}$ (due to Condition 2). Furthermore, all processes not owning anything in c' are running (due to Condition 3).

The blocked processes dropped earlier and the visible process are added to the loop γ' . They stay blocked in the loop—due to the arguments used in Lemma 20 and due to Lemma 17 changing just the number and ordering of some short loops within the main loop of β_n . We obtain a loop behaviour $\gamma_{n'}$ with $n' \leq B_{\mathcal{F}} - m$ starting from a configuration c'' . This configuration can be reached using a modification of α_k where all processes not present in c'' are dropped and some renaming of process names is done. Should this cause a lack of processes at locations from Q_n , the appropriate number of processes can be added to the locations of Q_n as follows. Before any other processes start, the added processes just go to the appropriate location of Q_n (following the steps originally fired by any of the processes passing such a location) and then wait till they are necessary (they do not block anything because they are not involved with any resources). We obtain a weakly fair behaviour $\beta_{n'}$ such that for the corresponding $\pi_{n'}$ and a suitable valuation $v_{n'} : \{p\} \rightarrow P_{n'}$, we have $\pi_{n'}, v_{n'} \models \varphi(p)$.

It remains to check whether $n' \geq m + 1$. If $n' < m + 1$, some more blocked processes are preserved to obtain $m + 1$. If this is not possible, then in the original β_n at least m processes are running. If no process in the configuration c is at a control location Q_n , then all of them are in Q_o which means that they are all in c' and thus running in $\beta_{n'}$, and therefore $n' \geq m + 1$. If a process p_1 in the configuration c is at a control location $q \in Q_n$, then the necessary number of processes can be added to the behaviour $\beta_{n'}$: As all transitions that used to be fired by some processes in β_n are fired by at least one process (Condition 3 of Lemma 17) in $\beta_{n'}$, there is some p_2 in $\beta_{n'}$ which goes through $q \in Q_n$. The additional processes may first follow one-by-one the transitions of p_2 at the beginning of $\beta_{n'}$ and go to q . (This is always possible, as at q , they do not use nor request any resource and thus do not block anything.) Then, they mimic one after another the behaviour of p_2 —they cyclically swap their roles at q as in the proof of Lemma 11. In such a way, we obtain a weakly fair behaviour $\beta_{n''}$ with $n'' \geq m + 1$. □

Finally, we can state and prove the key cut-off Lemma 22 showing 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.

Lemma 22 *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 We prove this lemma by contradiction. Suppose we are given a weakly fair counterexample behaviour β_n of S_n such that $\pi_n, v_n \models \neg\varphi(p)$ holds for the path π_n corresponding to β_n and some valuation $v_n : \{p\} \rightarrow P_n$. We show that there is a weakly fair counterexample behaviour $\beta_{n'}$ in a system $S_{n'}$ with $m + 1 \leq n' \leq B_{\mathcal{F}}$ such that for the corresponding $\pi_{n'}$ and some valuation $v_{n'} : \{p\} \rightarrow P_{n'}$, we have $\pi_{n'}, v_{n'} \models \neg\varphi(p)$. There are three cases: (1) All processes in β_n get blocked. (2) The visible process is running forever. (3) The visible process is eventually blocked forever.

Case 1. This situation corresponds to a static deadlock scenario, and we can use a slightly modified version of Lemma 26 (presented later) where m processes and the visible process are preserved. The applied reordering does not matter because we see actions of just one process and their mutual ordering is preserved.

Case 2. There are two subcases: (2.1) There are more than $m + 1$ processes running forever. (2.2) There are strictly less than $m + 1$ processes running forever.

Case 2.1. Here, n' is chosen to be $m + 1$. Then we apply Lemma 19.

Case 2.2. This case is done in Lemma 20.

Case 3. This case is done in Lemma 21. □

The cut-off for single-process MPTL formulae We now formalise the counterpart to Lemma 22. We show that if the weakly fair behaviours of a system with more than $m + 1$ processes satisfy some single-process MPTL formula (it suffices when this holds for process 1 being visible), then the formula is satisfied for the smaller systems (with at least $m + 1$ processes) too. Subsequently, we use this fact together with the above lemmas to give a complete cut-off result for single-process MPTL formulae and weakly fair behaviours of systems of RTR families.

Lemma 23 *Let \mathcal{F} be an RTR family and $\varphi(p)$ an MPTL path formula. Then, for systems of \mathcal{F} , we have: $\forall n' \geq m + 1, n \geq n' : S_n \models_{wf} A \varphi(1) \Rightarrow S_{n'} \models_{wf} \forall p : A \varphi(p)$*

Proof A straightforward analogy of the proof of Lemma 12. □

Theorem 7 *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 $\mathcal{F}, l \models_{wf}^a \Phi_a$ 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 $n = B_{\mathcal{F}}$.*

Proof If there are $S_n \in \mathcal{F}$ with $l \leq n \leq m + 1$, they are covered directly exploiting just the interchangeability of processes. Then, for the (\Leftarrow) case, if $S_{B_{\mathcal{F}}} \models_{wf} A \varphi(1)$ holds, Lemma 23 implies $S_n \models_{wf} \forall p : A \varphi(p)$ holds for $m + 1 \leq n \leq B_{\mathcal{F}}$. Using Lemma 22, this can be extended to hold for $S_n \in \mathcal{F}$ with $m + 1 \leq n$, which is what is needed or even more (for $m + 1 < l$). For the (\Rightarrow) case, If $S_n \models_{wf} \forall p : A \varphi(p)$ holds for $l \leq n$, $S_{B_{\mathcal{F}}} \models_{wf} A \varphi(1)$ follows either directly for $l \leq B_{\mathcal{F}}$ or from Lemma 23 (note that $m + 1 \leq B_{\mathcal{F}}$). □

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 is simpler and 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 resources. 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 simple RTR families, 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 simultaneously be in control locations Q_o .

Lemma 24 *Let \mathcal{F} be a simple RTR family with m resources and $\varphi(p)$ an MPTL path formula. Then, the following holds for systems of \mathcal{F} :*

$$\forall n \geq 2m + 2 : (\forall n', m + 1 \leq n' \leq 2m + 2 : S_{n'} \models_{wf} \forall p : A \varphi(p)) \Rightarrow S_n \models_{wf} \forall p : A \varphi(p)$$

Proof We prove this lemma by contradiction. Given a weakly fair counterexample behaviour β_n of S_n such that $\pi_n, v_n \models \neg\varphi(p)$ holds for the path π_n corresponding to β_n and some valuation $v_n : \{p\} \rightarrow P_n$, we show that there is a weakly fair counterexample behaviour $\beta_{n'}$ in a system $S_{n'}$ with $m + 1 \leq n' \leq 2m + 2$ such that for the corresponding $\pi_{n'}$ and some valuation $v_{n'} : \{p\} \rightarrow P_{n'}$, we have $\pi_{n'}, v_{n'} \models \neg\varphi(p)$. There are three cases: (1) All processes in β_n get eventually blocked forever. (2) The visible process is running forever. (3) The visible process gets eventually blocked. Cases 1 and 2 are the same as in Lemma 22.

Case 3. We proceed as in the proof of Lemma 21 to get a counterexample behaviour β_n . Then, we can obtain a behaviour β_k where some blocked processes have been removed. β_k is composed of a prefix α_k and γ_k (starting from a configuration c) repeated infinitely often. The preserved blocked processes do not influence the behaviour of the running processes. Thus, we can drop them from the behaviour γ_k to get a *simple* behaviour $\gamma_{k'}$ starting from c' . $\gamma_{k'}$ is a loop behaviour. Obviously, $\bar{\gamma}_{k'}$ is a quotient loop of the form $(q_1, x_1)t_1(q_2, x_2)t_2 \dots (q_L, x_L)$. It satisfies the property $\forall i : 1 \leq i \leq L : |q_i| \leq m + 1$. This is true due to the reordering of transitions in β_n : It is clear that at most m processes are at a location where they own one or several resources. Furthermore, at the internal control locations of some $(p)\text{rqt}$ transitions there may be k processes that have requested some resources which are released by another process just after the requests. If there are k such processes, then only $m - k + 1$ processes are at locations owning something because at least one process must own k resources (those that are subsequently released).

Then, we set $\text{min} = \text{minimum}\{x_1, \dots, x_L\}$. Clearly, $(q_1, x_1 - \text{min})t_1(q_2, x_2 - \text{min})t_2 \dots (q_L, x_L - \text{min})$ is still a quotient loop with $\exists i : x_i = 0$. Together with the fact that we have $\forall i : 1 \leq i \leq L : |q_i| \leq m + 1$, this gives $\forall i : 1 \leq i \leq L : |q_i| + x_i \leq m + 1$. This means that we have a quotient loop with at most $m + 1$ running processes. As in the proof of Lemma 21, we add back the preserved blocked processes and the visible one and obtain a counterexample behaviour $\beta_{n'}$ with $n' \leq 2m + 2$ and we can show that $n' \geq m + 1$ like in the proof of Lemma 21. □

Theorem 8 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 whether $\mathcal{F}, l \models_{wf}^a \Phi_a$ holds is equal to checking whether $S_n \models_{wf} A \varphi(1)$ holds for all $S_n \in \mathcal{F}$ such that $l \leq n \leq m + 1$ or $n = 2m + 2$.

Proof As the proof of Theorem 7 using Lemma 24 instead of Lemma 22. \square

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 8 can be generalised to families whose Q_n corresponds to such a component. Moreover, the same idea can be used to optimise 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 in which p could fire some transition. 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 specialised (structure-independent) cut-off result for dealing with it.

In order to prove this result, we first introduce notions of the so-called static and dynamic process deadlocks, which are two basic deadlock situations that we may witness in the systems we are interested in. For both of them, we prove that they can be detected with m processes. Then, we join these two results to prove the overall cut-off for deadlockability. Intuitively, a static deadlock is a situation where no process in a set of processes can advance due to conflicting requests. A dynamic deadlock is a situation where a process is blocked due to the fact that in the future, it is always inevitably overtaken by other processes and therefore can never access the requested resources although they are always eventually released by their users.

Static process deadlocks We say that a set of processes P is a set of processes that are *statically deadlocked* in a configuration c iff each $p_1 \in P$ is asking for some resources R' in c and it is blocked by some $p_2 \in P$ on some resource $r \in R'$.

Note that each process from a set P of processes that are statically deadlocked in c is indeed deadlocked in c . If this was not the case, in some future of c , some $p_1 \in P$ could eventually fire the first future transition fired by some process of P . This transition could only be a $(p)\text{take}$ transition for whose enabledness wrt. the semantics of RTR, all the queue items blocking it in c would have to disappear from the appropriate queues. At least one of these items belongs to some $p_2 \in P$. In RTR, such an item can only be removed by p_2 itself. For this, p_2 would have to first finish the $(p)\text{rqz}$ transition inside which it is in c . Then, the $(p)\text{take}$ transition fired by p_2 would precede the one fired by p_1 . However, we consider the transition fired by p_1 to be the first fired by a process from P , which is a contradiction.

In order to show that for witnessing a static deadlock, m processes are always enough, we first define a notion of the set $ub(c)$ of processes that are *users of some resources or that are blocked in a way that is critical* for a configuration $c \in C_n$ of a system S_n of an RTR family \mathcal{F} to contain a non-empty set P of statically deadlocked processes. In particular, we define $ub(c)$ as the maximum subset of P such that in c , one of the following conditions holds for

each $p \in ub(c)$: (1) p is the current user of some resource r (i.e. $c(r) \in u(p) \cdot \Sigma_n^*$). (2) p is inside some $\text{prqt}(R')$ operation and on some $r \in R'$, it is not blocked by any $p' \in P$. (3) p is inside some $\text{rqt}(R')$, it is not blocked by any $p' \in P$ on some $r \in R'$, and no process in P is inside some $\text{prqt}(R'')$ with $r \in R''$.

Further, for a run ρ_n of S_n that leads to a configuration $c \in C_n$ containing a static deadlock, let $\rho_n^{\downarrow \text{sd}}$ denote the sequence of transitions that we obtain from ρ_n by removing transitions not fired by processes of $ub(c)$ and by reordering the preserved occurrences of $(\text{p})\text{req}$ transitions as follows: (1) All preserved occurrences of $(\text{p})\text{req}$ transitions whose corresponding $(\text{p})\text{take}$ transitions are fired in ρ_n are put off such that they get just before firing of these $(\text{p})\text{take}$ transitions. (2) The remaining $(\text{p})\text{req}$ transitions are put off to the very end of $\rho_n^{\downarrow \text{sd}}$, and, moreover, we fire first all the preq transitions that are among these transitions (with their mutual ordering preserved) and only then the remaining req transitions (with their mutual ordering again preserved). We can now prove the following.

Lemma 25 *For any run ρ_n of a system S_n of an RTR family \mathcal{F} that leads to a configuration $c \in C_n$ that contains a non-empty set $P \subseteq P_n$ of statically deadlocked process, $\rho_n^{\downarrow \text{sd}}$ is a run of S_n . Moreover, it leads to a configuration $c' \in C_n$ where the processes from $ub(c)$ are statically deadlocked.*

Proof The first part of the proof is a straightforward modification of the proof of Lemma 6. Let us concentrate on the second part, i.e. on showing that the processes from $P' = ub(c)$ remain statically deadlocked in c' .

The $(\text{p})\text{req}$ transitions that were fired by processes P' in ρ_n and that were not followed by an occurrence of the corresponding $(\text{p})\text{take}$ transitions are preserved in $\rho_n^{\downarrow \text{sd}}$, no additional $(\text{p})\text{take}$ transitions are fired, and so all processes P' are inside some $(\text{p})\text{rqt}$ operations in c' . Thus, the only condition that can be broken is that some of them is not blocked by any process of P' in c' . We prove by contradiction that this is not the case. Suppose the condition is broken. As in c' there is no other possibility of blocking than by requests of processes P' , some originally blocked request is now granted. We can then distinguish the following four cases that are the only possible ones. (Recall that there are no g items related to processes P in c —otherwise they would not be blocked.)

Case 1. The newly granted request for a set of resources R' was originally blocked (among all) on some $r \in R'$ being in use by a $p' \in P$. As $p' \in P$ and it is the current user of r in c , $p' \in P'$. Moreover, r is still in use by p' in c' because all $(\text{p})\text{take}$ transitions that are fired by processes P' in ρ_n are fired by them in $\rho_n^{\downarrow \text{sd}}$ too, and no occurrences of any rel transitions are added. A resource that is in use cannot be granted (not even repeatedly to the same process), and we have a contradiction.

Case 2. The newly granted request for a set of resources R' was not blocked on any resource in use by a process from P , but it was blocked by a $\text{pr}(p')$ item of some $p' \in P$ in the queue of some $r \in R'$. Then, it was also blocked via the first pr item in the queue of r that belonged to some process from P , let us say p'' . Clearly, p'' is inside some prqt action in c and it is not blocked by any process of P on r (p'' has the first pr item belonging to a process of P in the queue of r , r cannot be granted to any process of P , and wrt. the precondition of Case 2, r is not in use by any process from P). Thus, $p'' \in P'$, and the transitions fired by it in ρ_n are preserved in $\rho_n^{\downarrow \text{sd}}$. As no overtaking among not granted requests is possible in RTR, the request that produced the concerned $\text{pr}(p'')$ item to the queue of r must have originally preceded the request that is now granted. Moreover, as the former request is a high-priority request, the construction of $\rho_n^{\downarrow \text{sd}}$ guarantees that it still precedes the latter one. Then, if the former request is granted in $\rho_n^{\downarrow \text{sd}}$, it is not released (no

additional $\text{r}\in\text{l}$ is fired), multiple granting is impossible, and we have a contradiction. If it is not granted, overtaking of pr items is not possible, and we have a contradiction too.

Case 3. The newly granted request for a set of resources R' was not blocked in any of the above ways, it was blocked by processes from P via r items only, but in c in the queue of some $r \in R'$ there is a $\text{p}\text{r}(p')$ item belonging to some $p' \in P$. The now granted request is clearly some low-priority request $\text{r}\text{q}\text{t}(R')$. Suppose that the first pr item belonging to a process of P in the queue of r in c belongs to some $p'' \in P$. Then, due to the same reasons as above, $p'' \in P'$, and the transitions fired by it in ρ_n are preserved in $\rho_n^{\downarrow\text{sd}}$. The construction of $\rho_n^{\downarrow\text{sd}}$ guarantees that the high-priority request that generated the concerned $\text{p}\text{r}(p'')$ item to the queue of r precedes the now granted low-priority request in $\rho_n^{\downarrow\text{sd}}$. The same reasoning as above can then be used to show a contradiction.

Case 4. The now granted request was originally blocked as in Case 3, it is thus some low-priority request $\text{r}\text{q}\text{t}(R')$, but in c there is no pr item belonging to a process from P in the queue of any resource from R' . In such a case, we can start with the fact that in c , the now granted request is blocked on some $r \in R'$ by an $\text{r}(p')$ item belonging to some $p' \in P$. Then, it is also blocked by the first r item in the queue of r belonging to some process from P (say p'') in c . In $\rho_n^{\downarrow\text{sd}}$, transitions fired by p'' in ρ_n are preserved because $p'' \in P'$. This is due to in c , p'' is inside an rqt action asking among all the resource r , it has the first r item in the queue of r , r cannot be granted to any process of P (otherwise such a process could proceed), and wrt. the precondition of Case 4, r is not in use by any process in P nor there is a pr item belonging to a process from P in the queue of r (neither before nor after the item of p''). Now, it is possible to obtain a contradiction as in Case 2 taking into account that both of the involved requests are low-priority ones, and their original mutual ordering is thus guaranteed to be preserved by the construction of $\rho_n^{\downarrow\text{sd}}$. \square

Using Lemma 25, we can now easily prove the cut-off lemma for static deadlocks.

Lemma 26 *Let \mathcal{F} be an RTR family over an automaton $\mathcal{A} = (R, Q, q_0, T)$ with $|R| = m$. If some system $S_n \in \mathcal{F}$ with $m \leq n$ has a reachable configuration c in which a nonempty set P of processes is statically deadlocked, the same holds for some system $S_{n'} \in \mathcal{F}$ with $n' \leq m$ too.*

Proof Suppose there are S_n , P , and c that fulfil the assumption of the implication of the lemma for some RTR family \mathcal{F} .

Let $P' = \text{ub}(c)$. The definition of $\text{ub}(c)$ justifies the addition of some $p \in P$ to P' by arguing about the contents of the queue of some resource $r \in R$. At the same time, no two of these conditions can hold concurrently over the same resource $r \in R$, and if some of them holds, it holds for a single process only. (If a process from P is added to P' due to its relation with $r \in R$, it is the user of r , or if r is not used by any process from P , it is the process from P with the first pr item in the queue of r , or if there is no such process either, it is the process from P with the first r item in the queue of r .) Consequently, we see that $|P'| \leq m$.

Let ρ_n be the run corresponding to a behaviour that leads to c . Lemma 25 tells us that $\rho_n^{\downarrow\text{sd}}$ is a run which leads to a configuration c' where the processes from P' are still statically deadlocked (and all other processes idle at q_0). Moreover, Lemma 2 implies that a configuration c'' is reachable in S_n in which all processes $1, \dots, n'$, $n' = |P'| \leq m$, are statically deadlocked, with all other processes idling at q_0 . The idling processes do not influence firability of transition sequences. Moreover, the notion of a set of statically deadlocked processes does not rely upon processes outside of this set, and thus the processes idling at q_0 are not important for the fact that c'' contains a set of statically deadlocked processes.

Hence, a configuration with a non-empty set of statically deadlocked processes is clearly reachable in $S_{n'}$ too. \square

Dynamic process deadlocks For a family \mathcal{F} , a system $S_n \in \mathcal{F}$, and a configuration $c \in C_n$, let $reach(S_n, c)$ be the set of the configurations reachable from c in S_n . Given a family \mathcal{F} with a set of resources R and a system $S_n \in \mathcal{F}$, we say that a process $p \in P_n$ is *dynamically deadlocked* in a configuration $c_1 \in C_n$ iff in each $c_2 \in reach(S_n, c_1)$, p is blocked by some $p' \in P_n$ on some resource $r \in R$, but each process that blocks p on any $r' \in R$ in any configuration $c_3 \in reach(S_n, c_1)$ does not block p on r' in some $c_4 \in reach(S_n, c_3)$. Clearly, a process that is dynamically deadlocked is deadlocked.

In order to prove that for witnessing a dynamic deadlock, m processes always suffice, we first introduce a special notion of a reduced behaviour. Assume that we are given a run ρ_n of a system S_n of an RTR family \mathcal{F} that leads to a configuration $c \in C_n$ in which a process $p \in P_n$ is dynamically deadlocked. Let P be the set of all the processes deadlocked in c apart from p . Let $\rho_n^{\downarrow \text{dad}(p)}$ denote a sequence of transitions that we obtain from ρ_n by leaving out transitions fired by the processes in P and by a subsequent reordering as in $\rho_n^{\downarrow \text{sd}}$ with the following modification: At the end when firing the $(p)\text{req}$ transitions not followed by the corresponding $(p)\text{take}$ ones in ρ_n , we first fire all the requests that become granted when firing ρ_n . (We preserve the original firing order inside the set of such transitions as well as inside the set of the remaining ones.) We can prove the following important property.

Lemma 27 *Let us have a run ρ_n of a system S_n of an RTR family \mathcal{F} that leads to a configuration $c \in C_n$ in which a process $p \in P_n$ is dynamically deadlocked. Let P be the set of all the processes deadlocked in c apart from p . Then, $\rho_n^{\downarrow \text{dad}(p)}$ is a run of S_n . Moreover, it leads to a configuration $c' \in C_n$ that is equal to c up to the processes from P that remain at q_0 and that do not have any queue items in any queues in c' .*

Proof The first part of the proof is a straightforward modification of the proof of Lemma 6. Let us concentrate on the second part. Let $\rho'_n = \rho_n^{\downarrow \text{dad}(p)}$ and $P' = P_n \setminus P$.

First, processes P' fire the same transitions in ρ'_n as in ρ_n and so they reach the same control locations. The resources they are using in c' are then, of course, the same too. Similarly, their current requests are the same. The fact that all the requests granted to processes P' in c are granted to them in c' as well is clear from the following: The resources in use are at most the same, all other resources are free when we start firing the requests of P' granted in the behaviour β_n corresponding to ρ_n but not followed by firing of the corresponding $(p)\text{take}$ transitions, and, finally, as it was originally possible to grant the concerned requests, they do not collide with each other nor with the resources in use.

The not granted requests fired by processes P' in ρ_n cannot be blocked by the deadlocked processes in c —the not deadlocked processes would have to be deadlocked; the dynamically deadlocked process could not be dynamically deadlocked. Thus, in c , the not granted requests of P' can be blocked by processes P' themselves only. If such a request is then blocked by g/u queue items of P' in c , it remains blocked by them in c' as well. (As noted above, the g/u queue items of P' generated in β_n are generated in β'_n based on ρ'_n too and they appear before the transitions to be blocked are fired.) If this is not the case and if the given not granted request is blocked in c by pr queue items produced by some high-priority request of P' , it remains blocked in c' too. (As there is no reordering among blocked requests in RTR, the latter request must be fired first in ρ_n . The mutual firing order of not granted requests is preserved in ρ'_n . If the latter request is granted in β'_n , it is not released because no rel is added, and multiple granting is excluded. If it is not granted, pr items

cannot be overtaken.) Finally, the only remaining possibility is that the blocked request is a low-priority one and in c , it is blocked by r queue items of another low-priority request fired by some process from P' . As overtaking among r items is not possible, a similar argument to the above implies that blocking of the given request is preserved in c' in this last case too.

We have thus shown that the queues contain the same queue items of P' at the end of β'_n as at the end of β_n . At the same time, g/u queue items are always the heads of queues. Queue items related to blocked requests are produced in the original order and they are not subject to any reordering in RTR. Consequently, the ordering of the queue items belonging to P' is preserved in c' , and the proof of the described relation of c and c' is now complete. \square

We are now ready to state and prove the cut-off lemma for dynamic deadlocks.

Lemma 28 *Let \mathcal{F} be an RTR family over an automaton $\mathcal{A} = (R, Q, q_0, T)$ with $|R| = m$. If some system $S_n \in \mathcal{F}$ with $m \leq n$ has a reachable configuration c in which some process $p \in P_n$ is dynamically deadlocked, then the same holds for some system $S_{n'} \in \mathcal{F}$ with $n' \leq m$.*

Proof Suppose there are S_n , p , and c that fulfil the assumption of the implication of the lemma for some RTR family \mathcal{F} . The state space of S_n is finite, and thus from each configuration, a configuration that is in some terminal strongly-connected component (TSCC) is reachable. So, there is $c_1 \in \text{reach}(S_n, c)$ that is in such a component. As a process that is dynamically deadlocked remains dynamically deadlocked forever, p is dynamically deadlocked in c_1 too. Let P be the set of all the processes deadlocked in c_1 apart from p , and $P' = P_n \setminus P$. Moreover, let ρ_n be the run corresponding to a behaviour β_n leading to c_1 .

From Lemma 27, we know that $\rho'_n = \rho_n^{\downarrow \text{dd}(p)}$ is a run of S_n leading to a configuration $c_2 \in C_n$ that is equal to c_1 up to the processes from P that remain at q_0 and that do not have any queue items in any queues in c_1 . Let β'_n be the run corresponding to ρ'_n . We further modify β'_n by renaming the preserved processes to $1, 2, \dots, n'$ and by entirely leaving out all the idling processes. By Lemma 2 and by the fact that processes idling at q_0 do not influence the firability of transition sequences in RTR, we obtain a behaviour $\beta_{n'}$ of $S_{n'}$ that leads to a configuration c_3 that is equal to c_2 up to renaming of the preserved processes and up to the removal of the idling processes. In c_3 , the process p' corresponding to p is deadlocked, which can be shown by contradiction.

Suppose p' can be unblocked by a sequence of transitions fired from c_3 by the processes of $S_{n'}$ different from p' , which correspond up to renaming to the not deadlocked processes in c_1 . As c_1 is in a TSCC, the processes not deadlocked in c_1 cannot deadlock regardless of the path in the given RTR automaton they follow in the future. Thus, if there was a path in the automaton from a location that some of the unblocked processes was at in c_1 to a $(p)rqt$ transition t that (when fired) would be blocked by some of the processes in P , the concerned process could eventually fire the $(p)r \in q$ transition corresponding to t . Then, as all processes P are deadlocked, it would also deadlock, which would be a contradiction. Therefore, from no control location that some of the processes not blocked in c_1 is currently at there is a path in the given automaton to a $(p)rqt$ transition that would be blocked by some of (the queue items of) the deadlocked processes in P . Consequently, firing of any transition reachable by the unblocked processes from c_1 cannot be influenced by any queue items belonging to processes P . Therefore, as c_1 and c_3 differ only in the names of the processes from P' and in the presence of the processes from P and their queue items, the sequence of transitions that unblocks p' from c_3 in $S_{n'}$ can be fired by the unblocked processes from c_1 in S_n too, and these processes do not block p afterwards. As p is not blocked by the deadlocked processes, and this cannot change, p gets unblocked, which is a contradiction.

A similar reasoning as above can then be used to show that the processes different from p' in $S_{n'}$ do not deadlock in any future of c_3 and that p' is dynamically deadlocked.

Next, suppose that in $S_{n'}$, we can reach c_4 from c_3 , and, in c_4 there is some process p_4 that does not have any queue item in any queue. Using a similar argument as above, it is easy to show that in $S_{n'-1}$, we can reach a configuration c_5 that is equal to c_4 up to renaming and up to not containing p_4 . Moreover, it is clear that the process corresponding to p in c_5 remains dynamically deadlocked, and all other processes are not deadlocked and cannot deadlock: If it was possible to break this condition by firing some transition sequence, it would be possible to fire a corresponding sequence with the same effect in $S_{n'}$ as well (p_4 that is not present in any queue cannot influence the behaviour of the other processes), which would lead to a contradiction.

We can keep repeating the above step till there is some process that may be eliminated. We obtain a system $S_{n''}$ in which such a configuration c_6 is reachable that the process corresponding to p is dynamically deadlocked in c_6 , all other processes are not deadlocked and cannot deadlock, and in each $c_7 \in reach(S_{n''}, c_6)$, each process has some queue items in some queues.

We further show that such a configuration c_8 is reachable from c_6 that each of the processes that cannot deadlock in any future of c_6 is a user of some resource in each configuration reachable from c_8 : First, each of the concerned processes is a user of some resources in c_6 or it is at least requesting some. As none of the latter processes can deadlock, there must be a single behaviour leading from c_6 in which they all eventually take the resources they are requesting in c_6 . Then, it suffices to show that once such a process becomes a user of some resources, it is always a user of some resources in all possible futures. However, this is immediate because in RTR, in order to always have some queue items in some queues, a process has to request and take some new resources before releasing all of the old ones.

In a system with m resources, we can have at most m processes that are concurrently using some resources. At the same time, in $S_{n''}$, the process corresponding to p is dynamically deadlocked in c_6 (and thus also c_8), and so at least one resource must always eventually be released in at least one future of c_8 . In RTR, a released resource does not have a user at least just after the release. Consequently, as in all futures of c_8 , each of the not blocked processes always uses some resources, there can be at most $m - 1$ such processes (in addition to the deadlocked one), which concludes the proof. \square

The cut-off on process deadlockability Before proving the final result, we add one more lemma that is a counterpart to the above lemmas and that shows that if a deadlock is possible in a small system, it is possible in a bigger system of a given RTR family too.

Lemma 29 *If a deadlock is possible in a system S_n of an RTR family \mathcal{F} , it is possible in any system $S_{n'} \in \mathcal{F}$ with $n' \geq n$.*

Proof The additional processes may stay idle while the run originally leading to a process deadlock is being replayed and then they cannot resolve the deadlock situation any more: If a static deadlock occurs in S_n , once the run originally leading to the deadlock situation is replayed, no additional process can remove any of the blocking queue items, and they are sufficient for the deadlock. If the originally deadlocked process terminated, it is terminated forever. If some blocking process could not deadlock, but the control automaton did not allow it to release some resource any more, no additional process can change this either.

Finally, in the case of a dynamic deadlock, as the additional processes cannot remove queue items of the original processes, when they just reduce the number of possible behaviours of the original processes (after replaying the behaviour leading to the original deadlock

scenario), the deadlocked process will stay deadlocked—although not necessarily dynamically. Indeed, all the originally possible behaviours of the original processes are safe from the point of view of blocking the deadlocked process. The only possibility how the additional processes can enable some new behaviours of the original processes is to enable some new overtaking on the level of their $(p)rqt$ actions. However, the deadlocked process will not be unblocked in such behaviours either because the overtaking can be replaced by a postponed firing of some $(p)req$ transitions, which yields some of the original behaviours, and all of them are known to preserve blocking of the deadlocked process. \square

We now join the above results and cover the (mostly trivial) cases not yet covered under static and dynamic deadlocks. This way, we obtain the overall (structure-independent) result for deadlockability.

Theorem 9 *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 We first show that if some process may deadlock in some $S_n \in \mathcal{F}$ with $n > \max(m, 2)$, some process may deadlock in a certain $S_{n'}$ with $n' \leq \max(m, 2)$ too. Then, we note that if a process deadlock is possible in some S_n , it is also possible in any $S_{n'}$ with $n \leq n'$.

Suppose some process p may deadlock in some reachable configuration c of $S_n \in \mathcal{F}$ with $n > \max(m, 2)$. The state space of S_n is finite, and thus from each configuration, a configuration that is in some TSCC can be reached. Therefore, there is $c' \in \text{reach}(S_n, c)$ that is in such a component. As a process that is deadlocked remains deadlocked forever, p is deadlocked in c' too. Let P be the set of all the processes deadlocked in c' .

If P is a set of processes that are statically deadlocked in c' , we can apply Lemma 26. Otherwise, there is $p' \in P$ that is deadlocked, but not blocked by any deadlocked process. In such a case, p' may have reached a terminal control location (i.e. $q \in Q$ for which there is no $t \in T$ such that $t = (q, a, q')$). Then, using Lemma 5, it is clear that process 1 can deadlock in S_1 by reaching q .

Let us exclude the above trivial case. In RTR, the only other possible cause of the process deadlock situation is that p' is before some $(p)\text{take}(R')$ transition t , and t can never become enabled for p' from c' . Indeed, all other kinds of transitions than $(p)\text{take}$ are automatically enabled in RTR when their source control location is reached, and, on the other hand, $(p)\text{take}$ transitions do not share their source control locations with other transitions.

Another trivial process deadlock scenario is the following: In c' , p' is requesting a resource r owned by another process that is not deadlocked (and cannot deadlock any more), but that is at a control location from which there is no way to reach a release on r . Then, to show process deadlockability, we suffice with S_2 —one process first goes to the control location in which it blocks r and from which it cannot release r any more, and then the second process tries to repeat its behaviour. We exclude such a scenario in the following.

Now, we know that p' is deadlocked, but it is not blocked by any deadlocked process. We have excluded the possibility that p' is not blocked by any process. Furthermore, we have also excluded the possibility that some process that is blocking p' on some resource r does not have a path in the given RTR automaton from its current control location to a release on r , and thus also the possibility that such a process cannot release r any more despite it cannot deadlock itself. (When a process cannot deadlock, it can fire any path in the concerned RTR automaton leading from its current control location.) As c' is in a TSCC, the excluded possibilities are excluded forever. Therefore, p' is dynamically deadlocked in c' , and we can apply Lemma 28.

Finally, by Lemma 29, we know that if a deadlock is possible in a system S_n of an RTR family \mathcal{F} , it is possible in any system $S_{n'} \in \mathcal{F}$ with $n' \geq n$. This closes the proof. \square

Let us note that the possibility of inevitable overtaking examined in the proof of Theorem 9 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\setminus P families with no overtaking.

7 RTR families and undecidability

Finally, we discuss an extension of MPTL by *local process quantification* [9] 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 $\pi_n, v_n \models \forall V' : \varphi(V \cup V')$ iff $\pi_n, v'_n \models \varphi(V \cup V')$ holds for all valuations v'_n of PV such that $\forall p \in PV \setminus V' : v'_n(p) = v_n(p)$. Such a quantification can be used to express, e.g., the global response property $A \square((\exists p_1 : at(p_1, req)) \Rightarrow \diamond(\exists p_2 : at(p_2, resp)))$, 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\setminus P families.

Theorem 10 *The parametric finite-behaviour verification problem of checking $\mathcal{F}, 1 \models_{fin}^a \Phi_a$ for an RTR\setminus P family \mathcal{F} and an MPTL formula Φ_a with local process quantification is undecidable even when the only temporal operators used are \square and \diamond and no temporal operator is in the scope of any local process quantifier.*

Proof We prove undecidability of checking $\mathcal{F}, 1 \models_{fin}^a \Phi_a$ via undecidability of the dual problem of checking $\mathcal{F}, 1 \models_{fin}^e \Phi_e$.

We show that for any PDA M with one, two, or more push-down stacks, we can construct an RTR\setminus P family \mathcal{F} and an existential MPTL formula Φ_e with a path subformula φ using local process quantification such that $\exists S_n \in \mathcal{F} : S_n \models_{fin} E \varphi$ holds iff $L(M)$ is nonempty. We concentrate on dealing with PDAs with a single push-down stack and we indicate how our construction can be easily generalised to dealing with two (or more stacks). We build the formula φ that we use to check nonemptiness of $L(M)$ for a given M from four subformulae, $\varphi \equiv \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$, which we introduce one-by-one later in the proof. We take care so that φ satisfies the requirement that there is no temporal operator in the scope of any local process quantifier and that the only temporal operators used are \square and \diamond .

Given a PDA M with a set of states Q_M and a stack alphabet Γ_M ($Q_M \cap \Gamma_M = \emptyset$), we simulate it in a non-deterministic way such that in the simulating family \mathcal{F} there is $S_n \in \mathcal{F}$ with a path satisfying φ iff there is an accepting run in M . We leave out dealing with input symbols—they are not important when only checking emptiness of $L(M)$. We represent states as well as stack symbols of M by processes. All these processes must behave according to the single RTR\setminus P control automaton of the simulating RTR\setminus P family, but their roles may be distinguished wrt. the control branches they are running in. For each $X \in Q_M \cup \Gamma_M$, we introduce a separate branch in the control automaton. All these branches lead from the same control location (at the beginning the processes pass the same initial phase) and

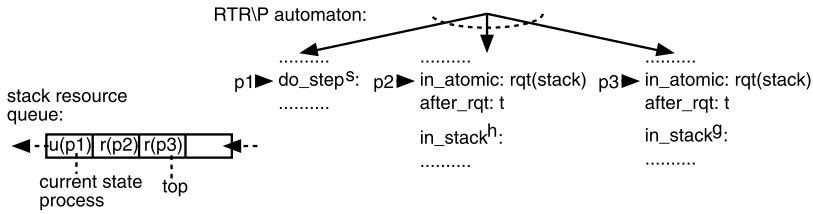


Fig. 4 Encoding of the (s, hg) configuration of a PDA being simulated

they do not intersect before the role played by the processes is over. Thus, once the processes pass the initial part of their behaviour, we can distinguish them to (s) -state processes (for $s \in Q_M$) and (g) -stack-symbol processes (for $g \in \Gamma_M$). We let the state processes simulate transitions from one state of M to another as well. A multiple use of the same state or stack symbol in a simulation run corresponds to using multiple processes of the appropriate type.

In the family \mathcal{F} simulating M , we use the following resources (and their queues—let us note that, for brevity, we may sometimes use the same name for a resource and its queue): (1) *stack* corresponding to the push-down stack of M , (2) *gate1* and *gate2* for controlling the possible progress of the involved processes, (3) *gate_X* for each $X \in Q_M \cup \Gamma_M$ allowing the roles played by the different processes to be checked, (4) *input* storing processes to be used to represent future states or stack symbols, and (5) *init* and other used within the initialisation of the simulation.

Representation of configurations. We consider a configuration $c \in C_n$ of a simulating system $S_n \in \mathcal{F}$ to represent a configuration $(s, \gamma) \in Q_M \times \Gamma_M^*$ of M iff the following is satisfied: (1) All the resources (besides *init* and *other*) are owned by an s -state process that is at the control location *do_step^s* (cf. Fig. 7(a)). (2) *init* and *other* are owned by an initialisation process that has already terminated its activity at the location *init_end* (see Fig. 7(b)). (3) Stack-symbol processes blocked by *rqt(stack)* before the appropriate *in_stack^X* locations (cf. Fig. 7(a)) are such that when we map the contents of the *stack* queue to the symbols X associated with the control branches of the processes waiting here, we obtain $s.\gamma$ (with the top of the stack—supposed to be on the right—represented by the tail of the *stack* queue as shown in Fig. 4). (4) Processes that will play the role of the future states and stack symbols are blocked by *rqt(input)* before the *in_input* location (see Fig. 7(a), (b)). (5) All other processes are before (or inside) *rqt(init)* or *rqt(other)* (cf. Fig. 7(b)) and will not be able to significantly influence the simulation any more.

Clearly, in any configuration corresponding to any $(s, \gamma) \in Q_M \times \Gamma_M^*$, the only significant process that is unblocked is an s -state process, which is at the location *do_step^s*. We build the RTR\NP automaton such that the s -state process may do a τ step to *accept* and terminate the simulation (cf. Fig. 7(c)) iff $s \in F_M$. Apart from this, we add a separate control branch for each transition of M leading from s . We do not introduce any other possible transitions leading from *do_step^s*.

Simulation of transition firing. Suppose now that the simulation is in a configuration corresponding to some $(s_1, \gamma_1) \in Q_M \times \Gamma_M^*$. If $s_1 \notin F_M$, the s_1 -state process that is at *do_step^{s1}* may try to simulate any transition of M leading from s_1 , and at the same time, it does not have any other option. Suppose it will attempt to simulate a transition $t_1 \equiv s_1 \xrightarrow{a, g/\gamma} s_2$ where $a \in \Sigma_M$, $g \in \Gamma_M$, $\gamma \in \Gamma_M^*$. The corresponding control structure is detailed in Fig. 7. Dealing with a is left out because it is not needed as we have already mentioned.

The s_1 process must first read g from the stack whose top is simulated by the tail of the queue of *stack*. It unlocks *gate1* and *gate_g*, unlocks *stack*, and blocks by *try-*

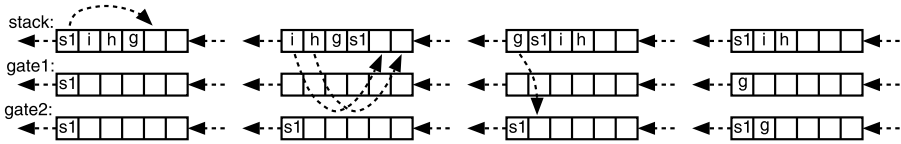


Fig. 5 Popping the top of the stack (without testing its type)

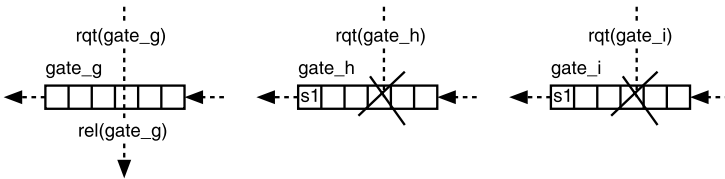


Fig. 6 Checking whether a process represents symbol g

ing to take $stack$ back. Unless the formula $\varphi_1 \equiv \square \neg((\exists p_1 : in_atomic(p_1)) \wedge (\exists p_2 : after_rqt(p_2)))^7$ is invalidated leading to an invalid simulation run, no process can leave $stack$ before s_1 tries to take it back and appears at the end of its queue. All non-tail processes from $stack$ can now loop from/to the in_stack^X states through the go_back branch (i.e. they release $stack$ and take it back). Their order in the queue is preserved, or φ_1 is invalidated. If a process decides to leave $stack$ via the try_out branch, it must be representing g , otherwise it blocks in the appropriate $rqt(gate_X)$ operation leading to a deadlock. If a non-tail g -symbol process decides to take the try_out branch, it blocks in $take(gate2)$, the process unblocked by it blocks in $rqt(gate1)$, and again the whole system deadlocks. So, only a tail g -symbol process may leave $stack$ and block in $take(gate2)$ while unblocking the s_1 process from $stack$ (cf. Figs. 4, 5, and 6). The s_1 process allows the g process to proceed and to finish its activity. The s_1 process then takes all the released resources back. If the top-stack-symbol process does not leave the stack, the formula $\varphi_2 \equiv \square \neg((\exists p_1 : out_check(p_1)) \wedge \neg(\exists p_2 : out_done(p_2)))$ is invalidated as soon as the s_1 process leaves $stack$.

Next, the s_1 -state process must push γ to the stack. This operation is either empty or corresponds to a (sequence of) similar steps as above. This time processes are not unblocked from $stack$ but input, they do not proceed from (and, possibly, do not loop back to) in_stack^X but in_input , and their type is “set” and not “checked” by the appropriate $rqt(gate_X)$ operation (the processes choose their future role, but they must choose the right one, otherwise they block). Stack-symbol processes do not terminate after leaving $gate2$ but block in $stack$.

Clearly, we could now simulate in the same way as above $pop/push$ operations on any (finite) number of other push-down stacks if M had more of them. That is why our construction of testing nonemptiness of $L(M)$ straightforwardly generalises to PDAs with any (finite) number of push-down stacks.

To go on with the simulation, the s_1 -state process has now to introduce the s_2 -state process representing the target state of the transition being fired and to help the new state process to become the owner of all the resources apart from $init$ and $other$. The new

⁷We suppose a formula $label(p)$ to hold iff p is at a location marked by $label$.

process is unblocked from `input` in a similar way as stack-symbol processes above. However, unlike these processes, it requests and takes not only `gate2` but also all the `gate_X` resources (excluding `gate1` and `gate_s2` which it already owns). When unblocked from `input`, the s_1 -state process releases `gate2`, all the appropriate `gate_X`, and also `stack` and blocks in `gate2` waiting for a further interaction with the s_2 -state process. The thus unblocked s_2 -state process now owns `gate1`, `gate2`, and all `gate_X`. The bottom-stack-symbol process is also unblocked, but it cannot go further than to trying to take `gate1`. The s_2 -state process releases `gate1` and blocks in `stack`—due to φ_1 , before the bottom-stack-symbol process may leave `gate1`. As only `gate1` is released, all the stack-symbol processes may just leave the stack and go back to it. After the rotation of `stack`, its real contents is unchanged, but it is owned by the s_2 -state process instead of the s_1 -state process. The same is then repeated with the `input` queue. Then, s_1 terminates, and s_2 takes over the full control of the simulation.

The described construction clearly ensures that if t_1 is not firable from (s_1, γ_1) in M , the s_1 -state-process-driven simulation of firing t_1 from a configuration corresponding to (s_1, γ_1) will either deadlock or will be invalidated via $\varphi_1 \wedge \varphi_2$ before any process can reach some `do_steps` location (which is necessary for encoding configurations of M and for further simulation steps and/or acceptance). If t_1 is firable in the given context in M , the simulation will again deadlock or be invalidated via $\varphi_1 \wedge \varphi_2$ before any process can reach some `do_steps` location, or the system will end up in a configuration corresponding to (s_2, γ_2) that is the target of firing t_1 from (s_1, γ_1) in M . Moreover, if t_1 is firable in (s_1, γ_1) and there are enough processes waiting in `input`, the possibility of reaching the configuration corresponding to (s_2, γ_2) without invalidating $\varphi_1 \wedge \varphi_2$ is guaranteed. For d symbols to be pushed, the guarantee is achieved with $d + 1$ processes in `input`—one for each symbol to be pushed plus one for the new state.

If $(s', \gamma') \in Q_M \times \Gamma_M^*$ is reachable in M from $(s, \gamma) \in Q_M \times \Gamma_M^*$, it is reachable in a finite number of steps of M . By induction, it is then easy to see that the following holds: If (s', γ') is not reachable from (s, γ) in M , a configuration corresponding to (s', γ') cannot be reached without invalidating $\varphi_1 \wedge \varphi_2$ from a configuration corresponding to (s, γ) in any S_n based on the above described control structure. On the other hand, if (s', γ') is reachable from (s, γ) in M , there exist d and n ($d < n$) such that in the system S_n based on the above described control structure, a configuration $c' \in C_n$ corresponding to (s', γ') can be reached in a valid way from $c \in C_n$ that corresponds to (s, γ) and that has d fresh processes ready to be used in `input`.

Simulation of acceptance. A string may be accepted from (s, γ) in M iff some (s', γ') is reachable from it and $s' \in F_M$. With respect to the described construction of the RTR\setminus P control automaton, an s' -state process p may terminate the simulating run such that the formula $\varphi_3 \equiv \diamond(\exists p_1 : \text{accept}(p_1))$ is fulfilled iff $s' \in F_M$ and p is a freshly introduced state process at the beginning of a new simulation step (cf. Fig. 7(c)). In combination with the previous results, we have the following: If no string can be accepted from (s, γ) in M , in any S_n based on the above described control structure there is no simulation run that satisfies $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ and leads from a configuration corresponding to (s, γ) . Conversely, if some string may be accepted from (s, γ) in M , there exist d and n ($d < n$) such that in the system S_n based on the above described control structure there is a simulation run that fulfils $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ and that leads from $c \in C_n$ corresponding to (s, γ) and having d fresh processes ready to be used in `input`.

Initialisation of the simulation. It remains to show that the RTR\setminus P automaton may be equipped with an initialisation part such that starting from c_0 , we either deadlock or invalidate some correctness requirement before any process can reach some `do_steps` location,

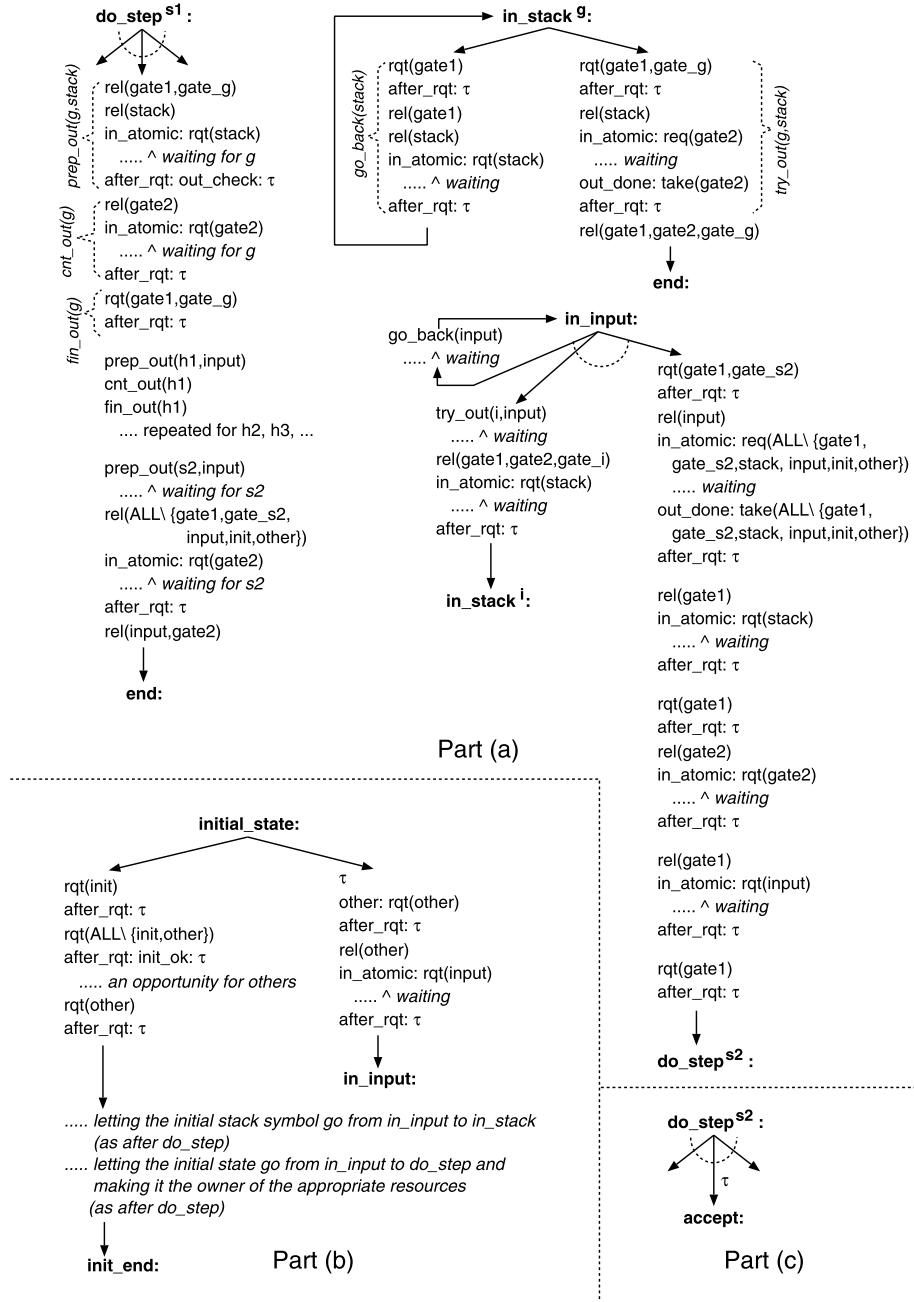


Fig. 7 (a) Simulating a PDA step $s_1 \xrightarrow{a,g/\gamma} s_2$ with $s_1, s_2 \in Q_M$, $a \in \Sigma_M$ (left out), $g \in \Gamma_M$, and $\gamma \in \Gamma_M^*$. (At the beginning of the step, an s_1 -state process is the owner of all the resources besides *init* and *other*; processes representing symbols currently stored in the stack are blocked before *in_stack* via *rqt* (*stack*); processes that should play the roles of future states and new stack symbols are blocked before *in_input* via *rqt* (*input*)). (b) Initialising the simulation of a PDA. (c) Simulating the termination of a (possibly) accepting run of a PDA

or we end up in a configuration corresponding to the initial configuration of M . Moreover, a possibility to reach such a configuration with all processes (apart from a fixed number of some initialising ones) ready in `input` is to be guaranteed. A solution to this problem is depicted in Fig. 7(b).

The `rqt(init)` operation without any subsequent release of `init` ensures that there will appear at most one initialisation process. The formula $\varphi_4 \equiv (\diamond (\exists p_1 : \text{init_ok}(p_1))) \wedge (\Box \neg \exists p_1, p_2 : (\text{init_ok}(p_1) \wedge \neg \text{initial_state}(p_2)))$ then ensures that in a valid behaviour there will appear some initialisation process and that it will grab all “working” resources before any other process starts. After reaching `init_ok`, the initialisation process may let an arbitrary number of other processes block in `input`. Then, it grabs `other`, and no processes outside `input` will be able to effectively influence the run of the system any more. Finally, the initialisation process introduces initial-stack-symbol processes, an initial-state process, and passes the ownership of all the “working” resources to the latter process as within a normal simulation step. \square

8 Conclusions and future work

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 in the case of 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 optimise 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 problem 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 is still 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. For the cases where no (or no small) cut-off can be found, we could then try to find some adequate abstraction techniques and/or symbolic verification techniques.

In the paper, we have considered a specification logic based on LTL $\setminus X$. The exclusion of the X operator is related to the fact that the proofs of our cut-off results build on adding and

removing processes (and the transitions fired by them) to/from behaviours of the examined systems. This is possible as without the X operator, the considered properties to be verified are stuttering insensitive. If X is allowed, the situation changes, and the same proof constructions do not apply. Moreover, it is not difficult to see that even the cut-offs themselves are not valid any more. The number of processes to be considered starts depending on the structure of the formulae to be verified even in the case of very simple safety properties (e.g., if we ask whether after five steps, some property holds, we may need five processes to fire these steps provided each of them can fire a single step only). Despite the most common properties to be verified on the given kind of systems do not require the use of X , its use can still be needed in some more specific situations. The treatment of such cases remains open for the future research.

Another issue is the use of more than just two levels of priorities. For the sake of clarity, we have stated our results using just two levels of priorities. However, all our results can be extended easily to more than two levels of priorities. In this case, the locker function has to be modified in order to allow overtaking of a lower priority process by a higher priority process. Let us examine this generalisation in a little bit more detail: The results in Sects. 4.3 and 5 which depend on the reordering of actions in a run can be generalised for more than two levels of priorities since the reordering does not depend on the particular level of priority. The results in Sect. 5 which depend also on the preservation of processes which block some resources can be generalised by always preserving the highest priority process blocking a resource. The results in Sect. 6 concerning static process deadlocks can be generalised by taking into account the different levels of priorities when preserving the deadlocked processes (higher priority requests are fired before lower priority ones). The results concerning dynamic process deadlocks can be generalised in a similar way.

Finally, several further 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 that the results presented here and the reasoning used to establish them can help in examining such questions.

Acknowledgement We would like to thank Parosh Aziz Abdulla and Thomas Arts for fruitful discussions.

References

1. Abdulla P, Bouajjani A, Jonsson B, Nilsson M (1999) Handling global conditions in parametrized system verification. In: Proc. of CAV'99. LNCS, vol 1633. Springer, New York
2. Apt K, Kozen D (1986) Limits for automatic verification of finite-state concurrent systems. *Inf Process Lett* 22(6):307–309
3. Arts T, Earle CB, Derrick J (2002) Verifying Erlang code: a resource locker case-study. In: Proc. of FME'02. LNCS, vol 2391. Springer, New York
4. Baukus K, Bensalem S, Lakhnech Y, Stahl K (2000) Abstracting WSIS systems to verify parameterized networks. In: Proc. of TACAS 2000. LNCS, vol 1785. Springer, New York
5. Emerson EA, Kahlon V (2000) Reducing model checking of the many to the few. In: Proc. of CADE 2000. LNCS, vol 1831. Springer, New York
6. Emerson EA, Kahlon V (2002) Model checking large-scale and parameterized resource allocation systems. In: Proc. of TACAS'02. LNCS, vol 2280. Springer, New York
7. Emerson EA, Namjoshi KS (1995) Reasoning about rings. In: Proc. of POPL'95. ACM, New York
8. Emerson EA, Namjoshi KS (1996) Automatic verification of parameterized synchronous systems. In: Proc. of CAV'96. LNCS, vol 1102. Springer, New York
9. Emerson EA, Sistla AP (1997) Utilizing symmetry when model checking under fairness assumptions: an automata-theoretic approach. *ACM Trans Program Lang Syst* 19(4):617–638

10. German SM, Sistla AP (1992) Reasoning about systems with many processes. *J ACM* 39(3):675–735
11. Kesten Y, Maler O, Marcus M, Pnueli A, Shahar E (1997) Symbolic model checking with rich assertional languages. In: *Proc. of CAV'97. LNCS, vol 1254*. Springer, New York
12. Kurshan RP, McMillan KL (1995) A structural induction theorem for processes. *Inf. Comput.*, 117(1)
13. Pnueli A, Ruah S, Zuck L (2001) Automatic deductive verification with invisible invariants. In: *Proc. of TACAS'01. LNCS, vol 2031*. Springer, New York
14. von Zur Gathen J, Sieveking M (1978) A bound on solutions of linear integer equalities and inequalities. *Proc Am Math Soc* 72:155–158
15. Wolper P, Lovinfosse V (1989) Verifying properties of large sets of processes with network invariants. In: *Proc of int workshop on automatic verification methods for finite state systems. LNCS, vol 407*. Springer, New York