Verification of Parametric Concurrent Systems with Prioritized FIFO Resource Management*

Ahmed Bouajjani[1], Peter Habermehl[1], and Tomáš Vojnar[2]

[1] LIAFA, University Paris 7, Case 7014, 2, place Jussieu, F-75251 Paris Cedex 05, France
e-mail:{Ahmed.Bouajjani,Peter.Habermehl}@liafa.jussieu.fr
[2] FIT, Brno University of Technology, Božetěchova 2, CZ-61266, Brno, Czech Republic
e-mail: vojnar@fit.vutbr.cz

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\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.

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 require a set of resources, get access and use the requested resources, and release the used resources. The

* An extended version of the paper published at CONCUR'03 (LNCS 2761, Springer-Verlag). This work was supported in part by the European Commission (FET project ADVANCE, contract No. IST-1999-29082) and by the Czech Grant Agency (GA CR projects GA102/01/1485 and GA102/03/D211).

requests can be of a low-priority or a high-priority level. The access to the resources is managed by a locker according to a FIFO-based policy taking into account the priorities of the requests—i.e. a waiting high-priority request can overtake waiting low-priority ones. As a special case allowing for an optimized treatment, we then examine the situation when no high-priority requests are used, and the locker behaves according to the pure FIFO discipline.

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

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

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

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

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

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

We show that for RTR families where the *pure FIFO resource management* is used (i.e. no high-priority access to resources is required), parametric verification of finite as well as fair behaviour is decidable against all LTL\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$\backslash$X extended with the notion of *local process quantification* [8], which allows one to examine different processes in different encountered states.

Related Work: There exist several approaches to the parametric verification problem. We can mention, for example, the use of symbolic model checking, (automated) abstraction, or network invariants [10, 1, 3, 14, 11, 12]. The idea of cut-offs has already been used in several contexts [9, 6, 7, 4, 5] too.

However, to the best of our knowledge, there is no work covering the class of parametric systems considered here, i.e. parametric resource sharing systems with a prioritized FIFO resource management. The *two* involved obstacles (parameterization and having multiple queues over an unbounded domain of process identifiers) seem to complicate the use of any of the known methods. Using cut-offs appears to be the easiest approach here.

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

Finally, let us add that our work was originally motivated by [2] that concerns verification of the use of shared resources in Ericsson's AXD 301 ATM switch.

In [2] finite-state model checking is used to verify some isolated instances of the given parametric system. Then, in the concluding remarks, a need for a more complete, parametric verification is mentioned, which is what our work is aiming at on the level of a general abstract model covering the given application domain.

Outline: We first formalize the notion of RTR families and define the specification logic we use. Then, we present our cut-off results for finite and fair behaviour and process deadlockability as well as the undecidability result. In the body of the paper, we mostly provide proof ideas of the presented results only—the complete proofs can be found in the appendix.

2 RTR Families

2.1 The Model of RTR Families

Processes in systems of RTR families are controlled by *RTR automata*. An RTR automaton over a finite set of resources is a finite automaton with the following kinds of actions joint with transitions: skip (denoted by τ—an abstract step not changing resource utilization), request and, when it is the turn, take a set of resources at the low- or high-priority level (`rqt`/`prqt`), and, finally, release a set of resources (`rel`).

Let us, however, stress that we allow processes to block inside (`p`)`rqt` transitions[3] while waiting for the requested resources to be available for them. Therefore, a single (`p`)`rqt` transition in a model semantically corresponds to two transitions, which we denote as (`p`)`req` (request a set of resources) and (`p`)`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* $A = \{\tau\} \cup \{a(R') \mid a \in \{\texttt{rqt}, \texttt{prqt}, \texttt{rel}\} \wedge R' \neq \emptyset \wedge R' \subseteq R\}$*. The sets* R*,* Q*,* T*, and* A *are nonempty, finite, pairwise disjoint, and disjoint with* $\mathbb{N}$.

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

2.2 Configurations

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

To make the semantics of RTR families reflect the fact that processes may block in (`p`)`rqt` actions, we extend the set Q of "explicit" control locations to Q_t containing a unique internal control location q_t for each transition $t \in T$ based on

[3] We use (`p`)`rqt` when addressing both `rqt` as well as `prqt` transitions.

a (p)$\mathtt{rqt}$ action. Furthermore, let T_t be the set obtained from T by preserving all τ and $\mathtt{rel}$ transitions and splitting each transition $t = (q_1, (\mathtt{p})\mathtt{rqt}(R'), q_2) \in T$ to two transitions $t_1 = (q_1, (\mathtt{p})\mathtt{req}(R'), q_t)$ and $t_2 = (q_t, (\mathtt{p})\mathtt{take}(R'), q_2)$.

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

2.3 Resource Granting and Transition Firing

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

A high-priority request is granted iff none of the needed resources is in use by or granted to any process, nor it is subject to any sooner raised, but not yet granted, high-priority request. A low-priority request is granted iff the needed resources are not in use nor granted and they are not subject to any sooner raised request nor any later raised high-priority request that can be granted at the given moment. (High-priority requests that currently cannot be granted do not block sooner raised low-priority requests.) Formally, for $c \in C_n$, we define $\Lambda(c)$ to be a configuration of C_n equal to c up to the following for each $r \in R$:

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

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

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

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

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

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

Finally, a $\mathtt{rel}$ transition removes the head $\mathtt{u}$ 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 $t = (q_1, \mathtt{rel}(R'), q_2) \in T_t$, we fix $to(c, t, p) = \Lambda((c \setminus c^-) \cup c^+)$ with $c^- = \{(p, q_1)\} \cup \{(r, c(r)) \mid r \in R' \wedge \exists w \in \Sigma_n^* : c(r) = \mathtt{u}(p).w\}$ and $c^+ = \{(p, q_2)\} \cup \{(r, w) \mid r \in R' \wedge w \in \Sigma_n^* \wedge c(r) = \mathtt{u}(p).w\}$.

2.4 Behaviour of Systems of RTR Families

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

We say a complete behaviour is *weakly (process) fair* iff each process that is eventually always enabled to fire some transitions, always eventually fires some transitions. 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 $\exists i : \forall j \geq i : \exists t \in T_t : en(c_j, t, p)$ holds, then $\forall i : \exists j \geq i : \exists t \in T_t : c_{j+1} = to(c_j, t, p)$ holds too.

We may call a complete behaviour *strongly (process) fair* iff each process that is always eventually enabled to fire some transitions, always eventually fires some transitions. Formally, a complete behaviour β_n is strongly fair iff for each $p \in P_n$, if β_n is infinite, and $\forall i : \exists j \geq i : \exists t \in T_t : en(c_j, t, p)$ holds, then $\forall i : \exists j \geq i : \exists t \in T_t : c_{j+1} = to(c_j, t, p)$ 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. (Idea) Due to the separation of requesting resources and starting to use them and the impossibility of cancelling once issued grants of resources, a process cannot temporarily have no enabled transitions without firing anything. ☐

For a behaviour $\beta_n = c_1(p_1, t_1)c_2(p_2, t_2)...$ of a system S_n of an RTR family $\mathcal{F}$, we call the configuration sequence $\pi_n = c_1c_2...$ a *path* of S_n corresponding to β_n

and the transition firing sequence $\rho_n = (p_1, t_1)(p_2, t_2)...$ a *run* of S_n corresponding to β_n. If the behaviour is not important, we do not mention it. We denote Π_n^{fin}, $\Pi_n^{fin} \subseteq C_n^+$, the set of all finite paths of S_n and Π_n^{wf}, $\Pi_n^{wf} \subseteq C_n^+ \cup C_n^\omega$, the set of all paths of S_n corresponding to complete, weakly fair behaviours.

3 The Specification Logic

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

We extend LTL\X by *global process quantification*[4] in a way inspired by ICTL* (see, e.g., [8]) and allowing us to easily reason over systems composed of a parametric number of identical processes. We also allow for an explicit distinction whether a property should hold for all paths or for at least one path out of a given set. Therefore, we introduce a single top-level *path quantifier* to our formulae. We restrict quantification in the following way: (1) We implicitly require all variables to always refer to distinct processes. (2) We allow only uniformly universal (or uniformly existential) process and path quantification.

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

3.1 The Syntax of MPTL

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

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

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

Subsequently, we define the syntax of *universal* and *existential MPTL formulae*, which extend MPTL path subformulae by process and path quantification used in a uniformly universal or existential way. For $V \subseteq PV$, we have:

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

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 $\forall p : A\ \Box\ (at(p, req) \Rightarrow \Diamond\ at(p, use))$.

[4] Later 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.

3.2 The Formal Semantics of MPTL

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

Suppose further that we have a system S_n of an RTR family $\mathcal{F}$. Let $\pi_n \in C_n^* \cup C_n^\omega$ denote a (finite or infinite) path of S_n. For a finite (or infinite) path $\pi_n = c_1 c_2 ... c_{|\pi_n|}$ ($\pi_n = c_1 c_2 ...$), let π_n^l denote the suffix $c_l c_{l+1} ... c_{|\pi_n|}$ ($c_l c_{l+1} ...$) of π_n, respectively. (For a finite π_n with $|\pi_n| < l$, $\pi_n^l = \epsilon$.)

Given a path π_n of S_n and a valuation ν_n of PV, we inductively define the semantics of MPTL path subformulae $\varphi(V)$ as follows:

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

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

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

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

3.3 Evaluating MPTL over Systems and Families

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

Next, we introduce a notion of MPTL formulae satisfaction over RTR families, in which we allow for specifying the minimum size of the systems to be considered.[5] 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.

[5] Fixing the maximum size would lead to finite-state verification. Although our results could still be used to simplify such verification, we do not discuss this case here.

4 Verification of Finite Behaviour

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

4.1 A Cut-Off Result for RTR\P Families

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

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

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

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

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

By using Lemma 2 and properties of MPTL, we now easily obtain the above promised result.

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

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 $S_k \models_{fin} A\ \varphi(1, ..., k)$ holds, Lemma 2 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 $l \leq n$, $S_k \models_{fin} A\ \varphi(1, ..., k)$ follows either directly for $l \leq k$ or from Lemma 2. $\square$

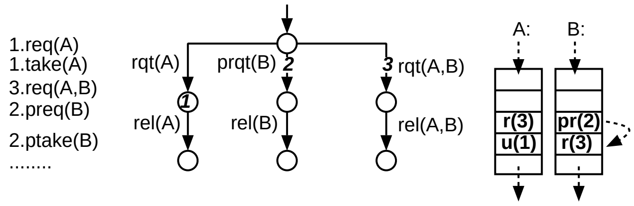

Fig. 1. A scenario problematic for the application of cut-offs (the run from the left is visualized on the RTR automaton and the appropriate resource queues)

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

Unfortunately, as we prove below, for families with prioritized resource management, the same reduction as above cannot be achieved even when we allow the bound to also depend on the number of available resources and fix the minimum considered number of processes to one.

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

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

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

4.3 Cut-Offs for Subclasses of MPTL

The first subclass of MPTL formulae we consider is the class of *invariance* and *reachability* formulae of the form

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

in which $\psi(V)$ is a boolean combination of atomic formulae $at(p, q)$. Mutual exclusion is an example of a property that falls into this class.

Let $\Psi_a \equiv \forall p_1, ..., p_k : A \; \Box\psi(p_1, ..., p_k)$ be an arbitrary invariance MPTL formula with k quantified process variables. We show that for any RTR family $\mathcal{F}$, the parametric problem of checking $\mathcal{F}, l \models^a_{fin} \Psi_a$ can be reduced to the finite-state problem of verifying $S_k \models_{fin} A \; \Box\psi(1, ..., k)$ with the number of processes fixed to k and the processes to be monitored via $p_1, ..., p_k$ fixed to $1, ..., k$. As above, we first state a basic cut-off lemma, which we subsequently generalize.

Lemma 3. *For any RTR family $\mathcal{F}$ and any nontemporal MPTL path formula $\psi(p_1, ..., p_k)$, the following holds for systems of $\mathcal{F}$:*

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

Proof. (Sketch) The case of ($\Rightarrow$) can be treated like in Lemma 2. In the ($\Leftarrow$) case, to reduce a counterexample behaviour of S_n to one of S_k, we first leave out all invisible processes and the transitions fired by them and permute the processes to make $1, ..., k$ visible (again like in Lemma 2). This is, however, not sufficient. The obtained transition sequence may not be firable because some overtaking among visible processes (a high-priority request is granted before a sooner issued low-priority one) may be possible only with the help of some invisible process, which blocks the low-priority request (cf. Fig. 1).

We solve the above problem in such a way that we replace overtaking among processes by postponed firing of their requests. More precisely, we postpone firing of `(p)req` transitions to be just before firing of the corresponding `(p)take` transitions (or at the very end of the run if the appropriate `(p)take` transition is not fired). Then, since the preserved processes release resources as they used to and they do not block them by requests before all originally overtaking requests are served, it can be shown that the firability of the reduced transition sequence is guaranteed. Moreover, the behaviour is modified in a way invisible for a reachability formula (negation of $\Box\psi$), which ensures that we obtain the desired counterexample in S_k. $\Box$

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

Proof. Similar to the proof of Theorem 1 with the use of Lemma 2 replaced by Lemma 3. $\Box$

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 for which whenever there is a transition $t = (q_1, (\mathtt{p})\mathtt{rqt}(R'), q_2) \in T$, we never use $at(p, q_1)$ or $at(p, q_t)$, but at most $(at(p, q_1) \vee at(p, q_t))$. Let us denote such *location-restricted* MPTL formulae by Υ_a/Υ_e and their path subformulae by υ. 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).

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

Proof. (Sketch) Similar to Lemma 3 and Theorem 3 when we take into account that the postponed firing of requests manifests itself just by some processes staying longer at locations q_1 before going to q_t for some $t = (q_1, (\mathtt{p})\mathtt{rqt}(R'), q_2) \in T$, which cannot be distinguished by $(at(p, q_1) \vee at(p, q_t))$. $\square$

5 Verification of Fair Behaviour

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

We show first that under the pure FIFO resource management, considering up to $m + k$ processes—with m being the number of resources and k the number of visible processes—suffices for parametric verification of weakly fair behaviour against any MPTL formulae. By contrast, for prioritized resource management, we prove that (as for finite behaviour verification) there does not exist any general, structure-independent cut-off that would allow us to reduce parametric verification of weakly fair behaviour to finite-state verification. Moreover, we show that, unfortunately, the inexistence of a structure-independent cut-off concerns, among others, also verification of the very important property of absence of starvation. Thus, for the needs of parametric verification of fair behaviour, we subsequently examine in more detail the possibility only sketched in the previous section, i.e. trying to find a cut-off reflecting the structure of the appropriate RTR automaton and/or the structure of the formula.

5.1 A Cut-Off Result for RTR\P Families

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

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

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

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

Proof. (Sketch) The case of ($\Leftarrow$) is similar to Lemma 2. The processes that keep running forever in a counterexample behaviour β_n in S_n will run even when we remove some processes. To keep blocked the visible processes that eventually block in β_n, we need at most one auxiliary invisible process per resource.

($\Rightarrow$) We show that we can extend a counterexample behaviour β_{m+k} to one of S_n by allowing the additional processes to fire some steps without influencing the visible behaviour. We distinguish three cases:

Case 1. If all processes block in a counterexample behaviour β_{m+k}, then in S_n, we let them block in the same way with the additional processes idling at q_0. Subsequently, the additional processes can also block by replaying the steps of any of the original processes.

Case 2. If all processes keep running forever in β_{m+k}, and eventually none of them releases any resources any more, at least one process eventually runs without using any resources any more—m resources can be used by at most m processes. The behaviour of such a process can be easily mimicked by the additional processes. (They first go to the location from which they do not use any resources any more, and their behaviour can then be arbitrarily interleaved with the behaviour of the original processes.) Otherwise, at least one resource is always eventually released. As $m-1$ resources can be used by $m-1$ processes only, a configuration in which $k+1$ processes do not use any resource is being regularly passed. At least one of these processes (say p) is invisible. In RTR\P, if p regularly passes a configuration in which it does not use anything, it must regularly pass a configuration in which it does not use nor request anything. The additional processes can then mimic the behaviour of p such that p plays its original role up to completing one loop on the configuration where it does not use nor request any resource, then the same is done by one of the new processes

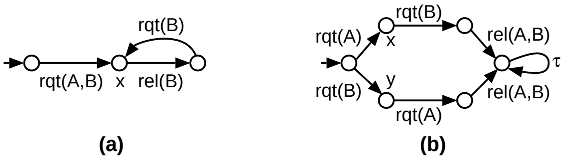

Fig. 2. Two simple RTR\P automata

(which get into the appropriate control location before all other processes start), then another one, and so on up to the last one when the scenario starts repeating. (As the concerned processes do not interfere with any resource in the given state, they do not block each other nor the other processes.)

Case 3. The remaining scenario, in which some processes keep running forever and some block in β_{m+k}, is the most subtle one, but it may be split to several subcases that can be shown to be solvable similarly to Case 1 or Case 2. $\square$

Now, the theorem generalizing the lemma can be easily obtained by exploiting properties of MPTL.

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

Proof. (Sketch) With respect to Lemma 4, we can use a similar argument as in the proof of Theorem 1, but as the system size of k (when we can already choose the given number of distinct processes) and the cut-off size of $m + k$ allowing us to generalize the results do not coincide, we have to separately examine each $S_n \in \mathcal{F}$ where $k \leq n \leq m + k$ and $l \leq n$. $\square$

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 $\forall p : A\ \Diamond\Box\ \neg at(p, \mathtt{x})$ does not hold in S_k (i.e. S_1), but it holds in all bigger systems. Fig. 2 (b) shows the automaton of a simple family for which $\forall p : A\ \Diamond\ (at(p, \mathtt{x}) \vee at(p, \mathtt{y}))$ 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 optimization 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 practical applicability of the result.

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

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

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

Proof. (Sketch) To witness starvation in the RTR automaton in Fig. 3, we need at least $x + 3$ processes with x depending on the number of transitions only. The starving process is blocked in `rqt(A)`. The blocking processes run in the right control branch. When such a process releases `A`, another has to block it instead. A process running in the upper exclusive part of the right branch needs several times help from the other processes. However, none of the latter processes can help the former several times in a single loop because the former process prevents the latter ones from going from one lower `A`-blocking section to another. □

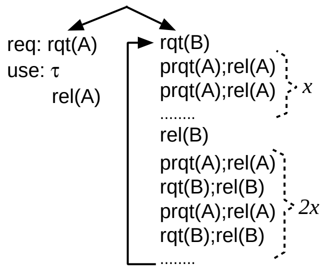

Fig. 3. An RTR automaton problematic for verification of absence of starvation

5.3 A Cut-Off for Single-Process MPTL Formulae

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

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

The key cut-off lemma below shows that if a formula is true in systems having between $m+1$ and $B_{\mathcal{F}}$ processes, it is also true in systems with more than $B_{\mathcal{F}}$ processes. This and Lemma 7 stating the opposite allows us to reduce the parametric verification problem to verification of systems with up to $B_{\mathcal{F}}$ processes.

Lemma 5. *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. (Sketch) We have to show that given a weakly fair counterexample behaviour β_n in S_n, we can obtain a weakly fair counterexample behaviour $\beta_{n'}$ in $S_{n'}$ with $m+1 \leq n' \leq B_{\mathcal{F}}$. There are several cases to consider. The most difficult one is the case where in β_n, the visible process is blocked forever, and invisible processes are running. To make sure that the visible process stays blocked, and the behaviour remains weakly fair, we have to preserve some invisible processes in $\beta_{n'}$. As shown in Section 5.2, this number cannot be bounded independently of the structure of the system. We show that less than $B_{\mathcal{F}}$ processes suffice. This is done basically in two steps.

First, we show that we can always reorder transitions fired in the counterexample behaviour β_n in such a way that the contents of queues is restricted such that it can be deduced from the control location of the involved processes (not considering forever blocked processes).

Second, we show that only at most $B_{\mathcal{F}}$ processes are needed to obtain a behaviour where the visible process stays blocked. To prove this we note that in a behaviour of a system reordered as mentioned above, the number of invisible processes that can be at control locations from Q_o is bounded by one, whereas the number of invisible processes that can be at control locations from Q_n is not bounded. At the same time, the exact identity of invisible processes is not important. Thus, the relevant information of a configuration is which control locations of Q_o are occupied ($\mathsf{q} \in 2^{Q_o}$) and how many processes are at each control location of Q_n ($\boldsymbol{x} \in \mathbb{N}^F$). The number of possible values of q is bounded by C, and so it remains to show that the values of $\boldsymbol{x}$ can be bounded too.

We can suppose that the counterexample behaviour β_n consists of a prefix followed by a loop. From this loop, we can extract the relevant information on configurations. In this way, we obtain a *quotient loop behaviour* which is a sequence of couples $(\mathsf{q}, \boldsymbol{x})$. Lemma 6 given below then provides us with a smaller quotient loop behaviour using less than a bounded number of processes and satisfying some additional conditions. This quotient loop can be used to obtain a weakly fair behaviour using less than $B_{\mathcal{F}}$ (and at least $m+1$) processes. $\square$

Lemma 6. *If there is a quotient loop $\overline{\gamma}$ of length bigger than $S = CM_C(M_C+1)(2FC(M_C+1))^F + 2C(M_C+1)$ starting from a configuration $(\mathsf{q}, \boldsymbol{x})$, then there is a quotient loop $\overline{\gamma'}$ of length smaller than S starting from a configuration $(\mathsf{q}, \boldsymbol{x}')$. Furthermore, (1) For all configurations $(\mathsf{p}, \boldsymbol{y})$ in $\overline{\gamma'}$, we have $|\mathsf{p}| + \sum_{i=1}^{F} \boldsymbol{y}_i \leq S$, (2) A transition appearing in $\overline{\gamma}$ appears at least once in $\overline{\gamma'}$, (3) $\forall i$ with $1 \leq i \leq F$, there exists a configuration $(\mathsf{p}, \boldsymbol{y})$ in $\overline{\gamma'}$ with $\boldsymbol{y}_i = 0$.*

Proof. (Sketch) $\overline{\gamma}$ can be decomposed into repeating short *structural loops* (i.e. quotient behaviours starting from some configuration $(\mathsf{p}, \boldsymbol{y})$ and going to $(\mathsf{p}, \boldsymbol{y'})$ where the occupied locations in Q_o are the same, but the number of invisible processes in locations Q_n may change) and some short inner quotient behaviours not containing such loops. Then, we construct a linear equation of the form $\mathbf{A}\boldsymbol{x} = \mathbf{B}$ where, intuitively, $\mathbf{A}$ encodes the effect of all short structural loops on vectors $\boldsymbol{y} \in \mathbb{N}^F$ and $\mathbf{B}$ encodes the effects of short inner quotient behaviors. The entries of both $\mathbf{A}$ and $\mathbf{B}$ are bounded. We know that there is a solution $\boldsymbol{x}$ for this equation given by $\overline{\gamma}$ which indicates how many times the short loops have to be repeated. Then, we use a lemma from the theory of Linear Integer Programming [13] showing that if $\mathbf{A}\boldsymbol{x} = \mathbf{B}$ has a solution and entries of $\mathbf{A}$ and $\mathbf{B}$ are bounded, then it has a bounded solution from which we construct a loop behaviour of the required size satisfying the three conditions of the lemma. $\square$

We now formalize the counterpart to Lemma 5. We show that if the weakly fair behaviours of a system with more than $m+1$ processes satisfies 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 behaviour of systems of RTR families.

Lemma 7. *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 ($\Rightarrow$) in Lemma 4. $\square$

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 $\mathcal{F}, l \models^a_{wf} \Phi_a$ is equal to checking $S_n \models_{wf} A\ \varphi(1)$ for all $S_n \in \mathcal{F}$ with $l \leq n \leq m+1$ or $n = B_{\mathcal{F}}$.*

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 7 implies $S_n \models_{wf} \forall p : A\ \varphi(p)$ holds for $m+1 \leq n \leq B_{\mathcal{F}}$. Using Lemma 5, 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 7 (note that $m+1 \leq B_{\mathcal{F}}$). $\square$

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 be simultaneously in control locations Q_o.

Lemma 8. *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. (Sketch) We only show here the case where the visible process is blocked and some invisible processes are running. We proceed as in the proof of Lemma 5 to get a counterexample behaviour β_n with bounded queues. Since $|Q_n| = 1$, the relevant information of a configuration is a couple (p, x) with $\mathsf{p} \in 2^{Q_o}$ and $x \in \mathbb{N}$. From β_n (without forever blocked processes from which we have to preserve at most m invisible ones) we can obtain a quotient behaviour where for all pairs (p, x), we have $|\mathsf{p}| + x \leq m+1$. Therefore, we need to preserve at most $m+1$ running processes. Together with the visible and the m blocked invisible ones, this gives at most $2m+2$. □

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 $\mathcal{F}, l \models^a_{wf} \Phi_a$ 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 8 instead of Lemma 5. □

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 generalized to families whose Q_n corresponds to such a component. Moreover, the same idea can be used to optimize the general $B_{\mathcal{F}}$ bound.

6 Process Deadlockability

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

Theorem 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. (Sketch) A full proof of this fact is quite subtle. We can encounter scenarios where a group of processes is mutually deadlocked due to some circular dependencies in queues of requests, but also situations where a process is deadlocked due to being always inevitably overtaken by processes that keep running and do not even own any resource forever. However, when we (partially) replace overtaking by postponed firing of requests (cf. Lemma 3), when we push blocked high-priority requests before the low-priority ones (the former block the latter, but not vice versa), and when we preserve only the running processes that never release all resources at the same time, we can show we suffice with one (primary) blocked and/or blocking process per resource. $\square$

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\P families with no overtaking.

7 RTR Families and Undecidability

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

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

Proof. (Idea) The result can be proved by a reduction from the problem of checking nonemptiness of languages of push-down automata (PDAs) with two push-down stacks. Simulation of PDAs by RTR\P families is quite complex due to the following facts: The queues in RTR are used as waiting queues, and not communication queues. They contain just the identities of waiting processes, all the processes have identical control, no process can manipulate queue items of other processes, and a process cannot even know whether or not it will have to wait when requesting some resource (nor whether it had to wait for a resource it is using now).

We only present the main ideas of the construction here. We simulate stack symbols as well as control states of a given two-stack PDA M by processes running in different branches of a suitably designed RTR\P automaton. (Input symbols need not be taken into account.) For each occurrence of a state or a stack symbol in a run of M, a fresh process is used. The processes wait for their use in the queue of a certain resource, and parameterization assures that there are enough of them. The stacks of M are simulated by queues of requests. The simulation is driven by processes corresponding to encountered control states of M. Such processes always control most of the resources used in the construction, and thus keep other processes (representing the current content of the stacks or just waiting for their future use) in the appropriate queues.

By blocking some resources and unblocking others, a current-state process can allow a certain process to leave the queue simulating some PDA stack (which corresponds to the pop operation), some processes to enter such a queue (simulating push), and some new state process to take over its role (simulating passage to a new control state of M). The simulation is such that if a process not representing the right symbol or not being at the top leaves a stack, the whole simulation deadlocks without reaching an acceptance state. (For example, to check which symbol a process represents, we have a special resource for each symbol, only one of these resources is unblocked, and the appropriate process has to pass through the resource corresponding to the symbol it simulates.) Similar constructions can be used to implement the other needed operations.

The above construction guarantees that when the involved processes do not cause a premature global deadlock nor ignore any opportunity to exhibit some progress, their activities correspond to a correct simulation of M. A violation of the latter condition cannot be detected and handled in an RTR\P automaton, but we can use a suitable MPTL formula with local process quantification to focus on runs where the processes really exhibit the activities they are supposed to exhibit. (For example, we check whether if some process is supposed to leave a stack, some process really leaves it, etc.) Finally, correct initialization and occurrence of an acceptance state can be checked via MPTL with local process quantification too. □

8 Conclusions

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

The structure-independent cut-offs we have presented in the paper are small and—for verification of finite behaviour and process deadlockability—optimal. They provide us with practical decision procedures for the concerned parametric verification problems and, moreover, they can also be used to simplify finite-state verification for systems with a given large number of processes.

The structure-dependent cut-off for single-process formulae 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 optimize this bound. Although we know that no general structure-independent cut-off exists, the bound we have provided is not optimal, and significantly improved cut-offs could be found especially for particular classes of systems as we have already shown for simple RTR families.

Another interesting 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.

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

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

References

1. P. Abdulla, A. Bouajjani, B. Jonsson, M. Nilsson. Handling Global Conditions in Parameterized System Verification. In *Proc. of CAV'99*, vol. 1633 of *LNCS*. Springer, 1999.
2. T. Arts, C. Earle, J. Derrick. Verifying Erlang Code: A Resource Locker Case-Study. In *Proc. of FME'02*, vol. 2391 of *LNCS*. Springer, 2002.
3. K. Baukus, S. Bensalem, Y. Lakhnech, K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In *Proc. of TACAS 2000*, vol. 1785 of *LNCS*. Springer, 2000.

4. E. Emerson, V. Kahlon. Reducing Model Checking of the Many to the Few. In *Proc. of CADE 2000*, vol. 1831 of *LNCS*. Springer, 2000.
5. E. Emerson, V. Kahlon. Model Checking Large-Scale and Parameterized Resource Allocation Systems. In *Proc. of TACAS'02*, vol. 2280 of *LNCS*. Springer, 2002.
6. E. Emerson, K. Namjoshi. Reasoning about Rings. In *Proc. of POPL'95*, 1995.
7. E. Emerson, K. Namjoshi. Automatic Verification of Parameterized Synchronous Systems. In *Proc. of CAV'96*, vol. 1102 of *LNCS*. Springer, 1996.
8. E. Emerson and A. Sistla. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach. *ACM Transactions on Programming Languages and Systems*, 19(4):617–638, 1997.
9. S. German, A. Sistla. Reasoning about Systems with Many Processes. *Journal of the ACM*, 39(3):675–735, 1992.
10. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *Proc. of CAV'97*, vol. 1254 of *LNCS*, 1997.
11. R. Kurshan, K. McMillan. A Structural Induction Theorem for Processes. *Inform. and Computation*, 117(1), 1995.
12. A. Pnueli, S. Ruah, L. Zuck. Automatic Deductive Verification with Invisible Invariants. In *Proc. of TACAS 2001*, vol. 2031 of *LNCS*. Springer, 2001.
13. J. von Zur Gathen, M. Sieveking. A Bound on Solutions of Linear Integer Equalities and Inequalities. *Proceedings of the American Mathematical Society*, 1978.
14. P. Wolper, V. Lovinfosse. Verifying Properties of Large Sets of Processes with Network Invariants. In *Proc. of Automatic Verification Methods for Finite State Systems*, vol. 407 of *LNCS*, 1989. Springer.

A Appendix

In the following, we provide complete proofs of the Lemmas and Theorems supported in the body of the paper by proof sketches only.

A.1 Behaviour of Systems of RTR Families

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. ☐

A.2 Finite Behaviour Verification

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

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

Proof. 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 $\neg\varphi(p_1, ..., p_k)$ for the valuation $\nu_k(p_i) = i$, $i \in \{1, ..., k\}$. Given any $n \geq k$, we add $n - k$ processes, let them idle at q_0 (where they do not block any resources and do not restrict any other processes), and we obtain a counterexample behaviour of S_n for ν_n such that $\nu_n(p_i) = i$, $i \in \{1, ..., k\}$.

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 $\pi_n, \nu_n \models \neg\varphi(p_1, ..., p_k)$ for the path π_n corresponding to β_n and some valuation $\nu_n : PV \to P_n$. We take the run ρ_n corresponding to β_n and transform it to ρ'_n by removing all transitions fired by processes different from $\nu_n(p_1), ..., \nu_n(p_k)$ and by a subsequent one-to-one replacement of all appearances of processes $\nu_n(p_1), ..., \nu_n(p_k)$ in the preserved part of ρ_n by $1, ..., k$. We show that ρ'_n is a run corresponding to some behaviour β_k of S_k (and, subsequently, that β_k is still a counterexample behaviour).

A run is a firable transition sequence. In systems of RTR\P families, the only transitions that need not be firable immediately when their source control location is reached are the ones based on `take`. Their enabledness requires a grant by the locker. In RTR\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\P, any other action than `rel` does not (up to relabeling) modify the contents of the queues or 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 `rel`. However, as we always start with empty queues, and a process cannot influence queue items related to other processes, `rel` in processes different from $\nu_n(p_1), ..., \nu_n(p_k)$ will not have any effect after transitions based on the other actions of these processes have been removed. So, by removing all transitions done by processes not among $\nu_n(p_1), ..., \nu_n(p_k)$ from ρ_n, we obtain a run of S_n. As $p_1, ..., p_k$ refer to k different processes, after their renaming to $1, ..., k$, we still have actions of k different processes. As the names of processes do not influence transition enabledness nor firing in RTR\P, ρ'_n is firable in S_n. Moreover, the idling processes do not have any influence on the firability of ρ'_n, and so ρ'_n is clearly firable in S_k as well and forms the run corresponding to some behaviour β_k of S_k.

It remains to show that $\pi_k, \nu_k \models \neg\varphi(p_1, ..., p_k)$ for π_k corresponding to β_k and ν_k such that $\nu_k(p_i) = i$ for $i \in \{1, ..., k\}$. However, this is clearly the case because the control locations of $1, ..., k$ are being changed in the same way in β_k as the ones of $\nu_n(p_1), ..., \nu_n(p_k)$ in β_n (and from the same initial q_0), the control locations of processes not among $\nu_n(p_1), ..., \nu_n(p_k)$ are invisible for φ, φ does not depend on the concrete names of processes, and MPTL is stuttering insensitive. □

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

Proof. Let us consider the RTR family $\mathcal{F}$ over the automaton from Fig. 4. We may concentrate on the dual problem to $\mathcal{F}, 1 \models^a_{fin} \Phi_a$ 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: $\exists S_n \in \mathcal{F} : S_n \models_{fin} \exists p_1, p_2 : E\ ((\neg at(p_2, \mathtt{B.1}))\ \mathcal{U}\ at(p_1, \mathtt{AB.1})) \wedge ((\neg at(p_1, \mathtt{AB.2}))\ \mathcal{U}\ at(p_2, \mathtt{B.2}))$. In the query, we refer to the control locations defined in Fig. 4 and we ask 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. 4 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`).

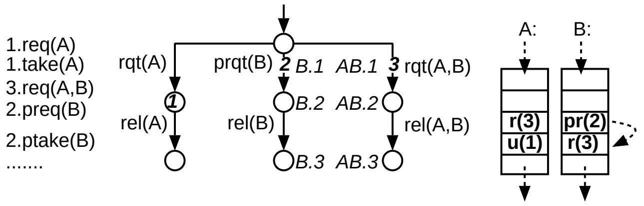

Fig. 4. An RTR scenario problematic for the application of cut-offs

We can 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, we can exclude the possibility of having more than one active process in each of these branches by adding `rqt(C)/rel(C)` (`rqt(D)/rel(D)`) at their beginnings and ends, respectively. Over the new family, we ask whether after the first overtaking, the processes can meet in `B.3` and `AB.3` and then exhibit another overtaking. As we do not extend the left-most branch, and no auxiliary processes can run in the middle or right branch, we need at least 4 processes to show satisfaction of the new property in the new setting. Obviously, we can keep extending the system and the formula in the described way (of course, without adding further operations on `C` and `D`), which will lead to a necessity of considering bigger and bigger systems although k as well as m remain the same. □

Lemma 3. *For any RTR family $\mathcal{F}$ and any nontemporal MPTL path formula $\psi(p_1, ..., p_k)$, the following holds for systems of $\mathcal{F}$:*

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

Proof. 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 $\Diamond\neg\psi(p_1, ..., p_k)$ for the valuation $\nu_k(p_i) = i$, $i \in \{1, ..., k\}$. For any $n \geq k$, we add $n - k$ processes, let them idle at q_0 (where they do not block any resources and do not restrict any other processes), and we obtain a counterexample behaviour of S_n for ν_n such that $\nu_n(p_i) = i$, $i \in \{1, ..., k\}$.

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 $\pi_n, \nu_n \models \Diamond\neg\psi(p_1, ..., p_k)$ holds for the path π_n corresponding to β_n and some valuation $\nu_n : PV \to P_n$. We show that we can reduce this counterexample to one in S_k with processes $1, ..., k$ being visible via $p_1, ..., p_k$.

We transform the run ρ_n corresponding to β_n to ρ'_n by removing all transitions fired by invisible processes (i.e. the ones different from $\nu_n(p_1), ..., \nu_n(p_k)$) and by postponing occurrences of all `(p)req` transitions such that they get just before firing of the appropriate `(p)take` transitions if they are fired. We put off firing of the preserved `(p)req` transitions that do not have their corresponding `(p)take` transitions fired in ρ_n to the very end of the new transition sequence and we keep the original order among them when firing them. We prove that ρ'_n

is a run of S_n. For this, we have to show that all the involved (p)take transitions are firable in the appropriate processes at all the points in ρ'_n where they appear. All other transitions are automatically enabled in RTR when a process reaches their source control location.

Let us now examine what happens with the enabledness of a transition t based on $(\mathtt{p})\mathtt{take}(R')$ and fired by a visible process p at a certain position in ρ_n after the transformation of ρ_n to ρ'_n. First, the enabledness cannot be hampered by any $r \in R'$ being blocked due to some $(\mathtt{p})\mathtt{req}(R'')$ transition ($r \in R''$) fired by some invisible process because no such transition is preserved. The removal of rel transitions fired by invisible processes is also not important because we always start with empty queues, the (p)rqt transitions of invisible processes are removed, and the invisible processes cannot release resources occupied by visible processes. Next, the mutual order of rel and (p)take transitions of visible processes is preserved. Moreover, it is clear that between the last rel on any $r \in R'$ in some visible process (or c_0 if r has not been used by a visible 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 visible process (otherwise there would have to be another rel). The construction of ρ'_n then guarantees that in ρ'_n 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 ρ'_n being firable in S_n is complete.

Next, we transform ρ'_n to ρ''_n by a one-to-one replacement of all appearances of $\nu_n(p_1), ..., \nu_n(p_k)$ by $1, ..., k$. As $p_1, ..., p_k$ refer to k different processes, after their renaming to $1, ..., k$, we still have actions of k different processes. As the names of processes do not influence transition enabledness nor firing in RTR, and all the processes start from the same initial location and without having any queue items, ρ''_n is firable in S_n. Moreover, the idling processes do not have any influence on the firability of ρ''_n, and so ρ''_n is clearly firable in S_k as well and forms the run corresponding to some behaviour β_k of S_k.

The transitions fired by processes $1, ..., k$ in β_k are the same as the ones fired by $\nu_n(p_1), ..., \nu_n(p_k)$ in β_n, which means that at the end of β_k, the processes $1, ..., k$ will be at exactly the same control locations as the processes $\nu_n(p_1), ..., \nu_n(p_k)$ at the end of β_n. As ψ does not depend on the concrete names of processes and it refers only to control locations of the involved processes, it is clear that $\pi_k, \nu_k \models \Diamond\neg\psi_k(p_1, ..., p_k)$ holds for π_k corresponding to β_k and ν_k such that $\nu_k(p_i) = i$ for $i \in \{1, ..., k\}$. Thus, as we needed, we have shown existence of a counterexample in S_k with $1, ..., k$ being visible via $p_1, ..., p_k$. $\square$

Lemma A.1. *Given an RTR family $\mathcal{F}$ and a location-restricted MPTL path formula $\upsilon(p_1, ..., p_k)$, the following holds for systems of $\mathcal{F}$:*

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

Proof. The case of ($\Rightarrow$) is the same as in the proof of Lemma 3. Moreover, for the ($\Leftarrow$) case, we can use the same argument as in the proof of Lemma 3 to show that from a counterexample behaviour of S_n, we can obtain a behaviour of S_k with the processes $1, ..., k$ to be made visible via $p_1, ..., p_k$. The fact that we get a counterexample behaviour follows from υ not being able to detect the removal of transitions fired in invisible processes (υ 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_t for some transitions $t = (q_1, (\mathtt{p})\mathtt{rqt}(R'), q_2) \in T$, which cannot be distinguished via $(at(p, q_1) \vee at(p, q_t))$. □

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

Proof. We can use the same proof construction as in the case of Theorem 3 with references to Lemma 3 replaced by Lemma A.1. □

A.3 Fair Behaviour Verification

Lemma 4. *For systems of an RTR$\backslash$P family $\mathcal{F}$ with m resources and an MPTL path formula $\varphi(p_1, ..., p_k)$, the following holds:*

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

Proof. We show that when the assertions of the implications do not hold, the premises do not hold either. We start with the easier case of ($\Leftarrow$) and then go on with ($\Rightarrow$).

The case of ($\Leftarrow$). For any $n \geq m+k$, suppose there is a weakly fair behaviour β_n of S_n such that $\pi_n, \nu_n \models \neg\varphi(p_1, ..., p_k)$ holds for the path π_n corresponding to β_n and some valuation $\nu_n : PV \to P_n$. We show that we can reduce β_n to a (weakly fair) counterexample in S_{m+k}. The reduction is partially similar to the one used in the proof of Lemma 2, but we have to preserve some auxiliary processes 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. $\nu_n(p_i) = p$ for some $i \in \{1, ..., k\}$), 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. We have 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 $|P| = m + k$.

We take the run ρ_n corresponding to β_n and transform it to $\rho'_n = \rho_{m+k}$ by removing all transitions done by processes that are not in P and by a subsequent one-to-one replacement of all appearances of $\nu_n(p_1), ..., \nu_n(p_k)$ by $1, ..., k$ and all the other processes by $k + 1, ..., m + k$. Wrt. the proof of Lemma 2, it is clear that ρ_{m+k} corresponds to some behaviour β_{m+k} of S_{m+k}. We show now that β_{m+k} is complete and 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 complete and weakly fair behaviour of such a system, each process must either keep firing some transitions, or eventually never have an enabled transition. Counterparts of all 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 `rqt` transition whose `take` part never becomes enabled. Counterparts of 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 complete and weakly fair, it remains to show that counterparts of the processes from P that are eventually forever blocked in some `rqt` 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 $\texttt{rqt}(R')$ 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 the counterparts of 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 2, it is now clear that $\pi_{m+k}, \nu_{m+k} \models \neg\varphi(p_1, ..., p_k)$ for π_{m+k} corresponding to β_{m+k} and ν_{m+k} such that $\nu_{m+k}(p_i) = i$ for $i \in \{1, ..., k\}$.

The case of ($\Rightarrow$). We have a weakly fair behaviour β_{m+k} of S_{m+k} such that $\pi_{m+k}, \nu_{m+k} \models \neg\varphi(p_1, ..., p_k)$ for the path π_{m+k} corresponding to β_{m+k} and the valuation ν_{m+k} such that $\nu_{m+k}(p_i) = i$ for $i \in \{1, ..., k\}$. We show that for any $n \geq m + k$, we can extend the run ρ_{m+k} corresponding to β_{m+k} by transitions fired by the processes of S_n not present in S_{m+k} in such a way that (1) we obtain a complete, 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 subcases: (1) No process is eventually running in β_{m+k}, i.e. β_{m+k} leads to a global deadlock. (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}.

Subcase 1 of ($\Rightarrow$). This is the simplest case. We let the original $m+k$ processes replay the run ρ_{m+k} corresponding to β_{m+k} with the additional processes idling. As noted in the proof of Theorem 9, the idling processes cannot change the fact that all the original processes eventually deadlock. Then the additional processes may one-by-one try to fire the same sequence of transitions as any of the original processes. Clearly, they all will deadlock too—at latest when they reach the same terminal control location as the original process or when they fire the request in which the original process is deadlocked. Thus, we have a complete, weakly fair behaviour, and the behaviour of the visible processes is unchanged.

Subcase 2 of (⇒). As S_{m+k} is a finite state system and $\neg\varphi(p_1, ..., p_k)$ for a fixed valuation of $p_1, ..., p_k$ is a common LTL\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 again consider two cases: No resource is ever released in the loop, or at least one resource is always eventually released. We show that in both cases, we can add processes that keep firing forever some transitions, but do not influence the visible behaviour.

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. The additional processes can then initially one-by-one follow the transitions of any of these processes up to some point from which it never uses any resource any more. Afterwards, the original processes may start replaying the run ρ_{m+k} corresponding to β_{m+k} and arbitrarily interleave it with further transitions fired by the newly added processes that go on in following the transition sequence of the process they mimic. As the additional processes do not access any resources from the point when we start firing transitions of the original processes, we clearly obtain a firable transition sequence that yields a complete, weakly fair behaviour whose visible part is the same as in β_{m+k}.

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\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.

In a behaviour β_n of S_n, the behaviour of p can now be mimicked by the additional processes—let us suppose there are $a = n - (m + k)$ of them—in the following way: The additional processes first one-by-one follow the transitions of p in β_{m+k} from c_0 to c' without looping yet. Then, we replay the run corresponding to the part of β_{m+k} leading from c_0 to c' still with no occurrence of the loop. 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_{m+k} from c_0 to c'. Subsequently, a loop is formed in β_n by repeating $a + 1$ times the loop of β_{m+k}, first with p firing its transitions, and the a new process idling, then with the first new process playing the role of p, and so on. Clearly, such a loop is firable because p and the additional processes do not block any resource when idling. Thus, we obtain a behaviour in which all the original as well as new processes are running, and the visible part of the behaviour is unchanged.

Subcase 3 of (⇒). 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, the behaviour of p can be easily mimicked by the additional processes.

If p reaches a terminal control location in β_{m+k}, all the new processes first one-by-one fire the transitions fired by p in β_{m+k}, and then we replay the run corresponding to β_{m+k}. As the new processes followed the behaviour of p, they are not blocking any resource in their terminal control location. Therefore, they do not block each other nor the original processes, and we obtain a run corresponding to some β_n.

If p blocks in some request in β_{m+k}, the new processes first one-by-one fire the transitions fired by p in β_{m+k} besides the last one. Then, we replay the run ρ_{m+k} corresponding to β_{m+k} up to the last request of p. This is possible because in β_{m+k}, p gets blocked without being the user of any resources, and so just before its last request, it is not involved with any resources. Thus, the additional processes do not block each other nor the original processes. As a continuation of the new transition sequence, p may fire its last request followed by the new processes doing the same. As the blocked request of p does not hamper firing the rest of ρ_{m+k} within β_{m+k} and as blocking exactly the same resources several times does not change anything, we may now again fire the rest of ρ_{m+k} and obtain a run corresponding to some β_n.

In both the above cases, we obtain a behaviour, this behaviour is complete and weakly fair (the new processes are eventually blocked forever), and the behaviour of the visible processes is unchanged.

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 forever looping processes—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 Subcase 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 do not use nor request any resources, and so the first case of Subcase 2 applies here.) □

Theorem 5. *Let $\mathcal{F}$ be an RTR\P family with m resources and let $\Phi_a \equiv \forall p_1, ..., p_k$: $A\ \varphi(p_1, ..., p_k)$ be an MPTL formula. Then, checking whether $\mathcal{F}, l \models^a_{wf} \Phi_a$ holds is equal to checking whether $S_n \models_{wf} A\ \varphi(1, ..., k)$ holds for all $S_n \in \mathcal{F}$ 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 $S_{n'} \models_{wf} \forall p_1, ..., p_k : A\ \varphi(p_1, ..., p_k)$ holds for all $n' \geq l$, and $l \leq m + k$, the satisfaction of $S_n \models_{wf} A\ \varphi(1, ..., k)$ for $max(l, k) \leq n \leq m + k$ is covered already by the premise. If $l > m + k$, Lemma 4 implies that $S_{m+k} \models_{wf} A\ \varphi(1, ..., k)$ 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 exploiting just the interchangeability of processes. Moreover, if $S_{m+k} \models_{wf} A\ \varphi(1, ..., k)$ holds, Lemma 4 implies $S_{n'} \models_{wf} \forall p_1, ..., p_k : A\ \varphi(p_1, ..., p_k)$ holds for all $n' \geq m + k$, which covers all the remaining (i.e. directly not covered) cases or even more (for $l > m + k$). □

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

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, we need at least $x + 3$ processes (i.e. a number depending on the structure of the control automaton, but not on m) 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 `rqt(B);rel(B)` actions, no process can go through two `prqt(A)` actions in the bottom part of the right branch while a single process is passing the upper part. Thus, we need one process in the bottom part 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. □

Basic results on loop behaviours. In the following paragraphs, we give some definitions and results on loop behaviours used to prove Lemma 5. A loop behaviour is a behaviour $\beta_n = c_1(p_1, t_1)c_2(p_2, t_2) \dots (p_{L-1}, t_{L-1})c_L$ with $c_1 = c_L$.

First, we define an equivalence relation $\sim$ on configurations C_n of RTR families as $c_1 \sim c_2$, iff there exists a permutation $\sigma : \{1, \dots, n\} \mapsto \{1, \dots, n\}$ such that $\sigma(c_1) = c_2$ (If c_1 is the function $c_1 : (P_n \to Q_t) \cup (R \to \Sigma_n^*)$, then $\sigma(c_1)$ is the function which assigns to $i \in P_n$ the configuration $c_1(\sigma(i))$ and to $r \in R$ the word $w \in \Sigma_n^*$ such that for all process i, each occurrence of i in $c_1(r))$ is replaced by $\sigma(i)$). The equivalence class of a configuration c is denoted by $[c]$. Let $\overline{c}$ be a unique representative of the equivalence class $[c]$. We will later define the unique representative explicitly. Let $\overline{C}_n$ be the set of all representatives of equivalence classes of C, i.e. the quotient of C_n. Then a *quotient behaviour* $\overline{\gamma}_n$ is a sequence $\overline{c}_1 t_1 \overline{c}_2 t_2 \dots$ with $\overline{c}_i \in \overline{C}_n$, $t_i \in T_t$ such that $\exists c_i \in C_n, p_i \in \{1, \dots, n\}$

with $c_i \in [c_i]$ and $c_1(p_1, t_1)c_2(p_2, t_2)\cdots$ is a behaviour of S_n. In the following, then the number of processes n in a quotient behaviour $\overline{\gamma}_n$ is clear from the context, we will drop it and just speak about a quotient behaviour $\overline{\gamma}$. The length of a quotient behaviour $\overline{\gamma}$ is the number of transitions it contains. A *quotient loop* is a finite quotient behaviour $\overline{c}_1 t_1 \overline{c}_2 t_2 \dots \overline{c}_L$ with $\overline{c}_1 = \overline{c}_L$.

Lemma A.2. *Given a quotient loop $\overline{\gamma}_n$, there is a loop behaviour γ_n.*

Proof. A quotient loop $\overline{\gamma}_n = \overline{c}_1 t_1 \overline{c}_2 t_2 \dots t_{L-1} \overline{c}_L$ with $\overline{c}_1 = \overline{c}_L$ corresponds to a finite behaviour $\beta_n = c_1(p_1, t_1)c_2(p_2, t_2)\dots(p_{L-1}, t_{L-1})c_L$ 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. Like this, 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. $\square$

In the proof of Lemma 5 the behaviours we consider are *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 in a given location $q \in Q_o$ and an arbitrary number of processes can be in 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 $(\mathsf{q}, \boldsymbol{x})$, where $\mathsf{q} \in 2^{Q_o}$ is the set of all locations in Q_o occupied by some process in c and $\boldsymbol{x} \in \mathbb{N}^F$ with $F = |Q_n|$ gives for each locations in Q_n the *number* of processes which occupy this location. From now on we consider quotient behaviours $\overline{\gamma}$ as sequences of the form $(\mathsf{q}_1, \boldsymbol{x_1})t_1(\mathsf{q}_2, \boldsymbol{x_2})t_2 \dots (\mathsf{q}_\mathsf{L}, \boldsymbol{x_L})$. The effect of $\overline{\gamma}$ is defined as $\mathit{eff}(\overline{\gamma}) = \boldsymbol{x_L} - \boldsymbol{x_1}$. Obviously, $\mathit{eff}(\overline{\gamma}) = \mathbf{0}$ if $\overline{\gamma}$ is a quotient loop. The absolute value of each component of $\mathit{eff}(\overline{\gamma})$ is bounded by $L-1$. The composition of two quotient behaviours $\overline{\gamma} = (\mathsf{q}_1, \boldsymbol{x_1})t_1(\mathsf{q}_2, \boldsymbol{x_2})t_2 \dots (\mathsf{q}_\mathsf{L}, \boldsymbol{x_L})$ and $\overline{\gamma'} = (\mathsf{p}_1, \boldsymbol{y_1})t'_1(\mathsf{p}_2, \boldsymbol{y_2})t'_2 \dots (\mathsf{p}_\mathsf{M}, \boldsymbol{x_M})$ is possible if $(\mathsf{q}_1, \boldsymbol{x_1}) = (p_1, \boldsymbol{y_1})$ and is defined by $\overline{\gamma}\overline{\gamma'} = (\mathsf{q}_1, \boldsymbol{x_1})t_1(\mathsf{q}_2, \boldsymbol{x_2})t_2 \dots (\mathsf{q}_\mathsf{L}, \boldsymbol{x_L})t'_1(p_2, \boldsymbol{y_2})t'_2 \dots (\mathsf{p}_M, \boldsymbol{x_M})$. We have $\mathit{eff}(\overline{\gamma}\overline{\gamma'}) = \mathit{eff}(\overline{\gamma}) + \mathit{eff}(\overline{\gamma'})$. Given a quotient behaviour $\overline{\gamma}$ the corresponding *projected quotient behaviour* $\widehat{\gamma}$ is the sequence $\mathsf{q}_1 t_1 \mathsf{q}_2 t_2 \dots \mathsf{q}_\mathsf{L}$. Furthermore, we call $\overline{\gamma}$ a structural loop if $\mathsf{q}_1 = \mathsf{q}_\mathsf{L}$. If $\overline{\gamma}$ is a quotient behaviour $(\mathsf{q}_1, \boldsymbol{x_1})t_1(\mathsf{q}_2, \boldsymbol{x_2})t_2 \dots (\mathsf{q}_\mathsf{L}, \boldsymbol{x_L})$ then $plus(\overline{\gamma}, \boldsymbol{d})$ where $\boldsymbol{d} \in \mathbb{N}^F$ is defined to be the quotient behaviour $(\mathsf{q}_1, \boldsymbol{x_1} + \boldsymbol{d})t_1(\mathsf{q}_2, \boldsymbol{x_2} + \boldsymbol{d})t_2 \dots (\mathsf{q}_\mathsf{L}, \boldsymbol{x_L} + \boldsymbol{d})$.

The following key 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 A.3. *Let π be a quotient loop of the form $\overline{\gamma}_1\overline{\gamma}_2\overline{\gamma}_3\overline{\gamma}_4\overline{\gamma}_5$ where $\widehat{\gamma}_2$ is a structural loop and with $\widehat{\gamma}_4 = \widehat{\gamma}_2$. Let the first element of $\overline{\gamma}_2$ be $(\mathsf{q}_2, \boldsymbol{d})$. Then,*

$$\overline{\gamma'}_n = plus(\overline{\gamma}_1, \boldsymbol{d})plus(\overline{\gamma}_2, \boldsymbol{d})plus(\overline{\gamma}_2, \boldsymbol{d} + \mathit{eff}(\overline{\gamma}_2))plus(\overline{\gamma}_3, \boldsymbol{d} + \mathit{eff}(\overline{\gamma}_2))plus(\overline{\gamma}_5, \boldsymbol{d})$$

is also a quotient loop.

Proof. Obvious by observing that the different parts of the behaviour can be composed. We have to add $\boldsymbol{d}$ to the vectors to make sure that they stay positive ($\mathit{eff}(\overline{\gamma}_2)$ is not necessarily positive). $\square$

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

Lemma A.4. *Let $\overline{\gamma}$ 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 $\overline{\gamma}$ of the form:*

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

where

- $\forall i \in \{1,\ldots,D\}\forall j \in \{1,\ldots,L_i\} : \widehat{\delta_i^j}$ *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* $\overline{\gamma}_i$ *has length smaller than* C.

Proof. Repeated application of Lemma A.3 and the fact that a quotient behaviour of length at least C contains a structural loop. $\square$

The following lemma follows directly from [13].

Lemma A.5. *Let $\mathbf{A}$ be a $M \times N$ (with $M \leq N$) integer matrix, $\mathbf{B}$ a $M \times 1$ integer matrix, G a bound on the absolute values of elements of $\mathbf{A}$ and $\mathbf{B}$. If there exists a nonnegative integer solution $\boldsymbol{x}$ to the equation $\mathbf{A}\boldsymbol{x} = \mathbf{B}$, then there exists a nonnegative integer solution where the value of all elements is bounded by $(N + 1)(MG)^M$.*

We now first prove Lemma 6 and then Lemma 5 based on it—in the body of the paper, these lemmas were swapped for better readability.

Lemma 6. *If there is a quotient loop $\overline{\gamma}$ of length bigger than $S = CM_C(M_C + 1)(2FC(M_C+1))^F + 2C(M_C+1)$ starting from a configuration $(\mathsf{q}, \boldsymbol{x})$, then there is a quotient loop $\overline{\gamma'}$ of length smaller than S starting from a configuration $(\mathsf{q}, \boldsymbol{x'})$. Furthermore, (1) For all configurations $(\mathsf{p}, \boldsymbol{y})$ in $\overline{\gamma'}$, we have $|\mathsf{p}| + \sum_{i=1}^{F} \boldsymbol{y}_i \leq S$, (2) A transition appearing in $\overline{\gamma}$ appears at least once in $\overline{\gamma'}$, (3) $\forall i$ with $1 \leq i \leq F$, there exists a configuration $(\mathsf{p}, \boldsymbol{y})$ in $\overline{\gamma'}$ with $\boldsymbol{y}_i = 0$.*

Proof. We transform $\overline{\gamma}$ using Lemma A.4 into

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

Obviously, we have $\forall i \in \{1,\ldots,D\}\forall j \in \{1,\ldots,L_i\} : \mathit{eff}(\overline{\delta}_i^j) = \mathit{eff}(\overline{\delta}_i^1)$. We define the matrices $\mathbf{A} = (\mathit{eff}(\overline{\delta}_1^1),\ldots,\mathit{eff}(\overline{\delta}_D^1))$ and $\mathbf{B} = -(\sum_{i=1}^{D} \mathit{eff}(\overline{\gamma}_i\overline{\delta}_i^1)) -$

$\mathit{eff}(\overline{\gamma}_{D+1})$. Let $\boldsymbol{x} = (L_1-1, L_2-1, \ldots, L_D-1)^T$. The absolute value of each entry of $\mathbf{B}$ is bounded by $2(M_C+1)C$ and that of $\mathbf{A}$ by C. Then, because $\overline{\gamma}$ is a loop, $\mathit{eff}(\overline{\gamma}) = \mathbf{0}$, i.e. we have $\mathbf{A}\boldsymbol{x} = \mathbf{B}$. Applying Lemma A.5 gives us a small solution $\boldsymbol{x}$ of the equation $\mathbf{A}\boldsymbol{x} = \mathbf{B}$. From this solution which indicates how often the structural loops have to be repeated, we can easily construct the quotient loop $\overline{\gamma'}$ of the required length. Each $\widehat{\delta_i^1}$ will be performed at least once, therefore each transition appearing in $\overline{\gamma}$ appears at least once in $\overline{\gamma'}$ (condition 2). $\overline{\gamma'}$ has a length smaller than S. This means that at most S processes perform a transition in the loop, those who don't can be removed (condition 1). Furthermore, if condition 3 is not satisfied for some i, than we remove a process from component i and we still have a quotient loop. □

Lemma 5. *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. Given an infinite counterexample behaviour β_n of S_n such that $\pi_n, \nu_n \models \neg\varphi(p)$ holds for the path π_n corresponding to β_n and some valuation $\nu_n : PV \to P_n$. we show that there is an infinite 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 $\nu_{n'} : PV \to P_{n'}$ we have $\pi_{n'}, \nu_{n'} \models \neg\varphi(p)$. There are three cases. (1) All processes in β_n are blocked. (2) The visible process is running. (3) The visible process is blocked.

Case 1. This corresponds to a static deadlock scenario and we can use a slightly modified version of Lemma A.6 where we preserve m processes and the visible process.

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

Case 2.1. Here n' is chosen to be $m+1$. We preserve n' running processes including the visible one. We transform the run ρ_n of β_n to $\rho_{n'}$. We preserve transitions of the n' running processes only. We postpone firing of `(p)req` transitions to be just before the corresponding `(p)take` transitions like in the proof of Lemma 3 and rename all processes to numbers from 1 to n'. It is clear that the behaviour $\beta_{n'}$ corresponding to $\rho_{n'}$ is also a counterexample weakly fair behaviour because the visible process is doing exactly the same actions in $\beta_{n'}$ than in β_n.

Case 2.2. In this case, some blocked processes have to be preserved. We transform the run ρ_n of β_n to $\rho_{n'}$ where n' processes are preserved. These processes are chosen as follows: First, we preserve the running processes. Second, for each resource, we preserve the blocked process (if it exists) which has the first `pr` item in the resource's queue and if it does not exist the blocked process (if it exists) which has the first `r` item in the resource's queue. At most we preserve like this $2m < B_{\mathcal{F}}$ processes. If the total number of processes preserved like this is less than $m+1$, we preserve arbitrarily some more blocked ones. Now, we show how to obtain $\rho_{n'}$. Because 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. From now on we consider the *finite* behaviour $\beta_n = \alpha_n\gamma_n$ which can always be extended to an infinite one. It is clear that blocked process in β_n are either in a location there they cannot do any actions any more or are in an internal control location having executed as 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 postponing occurrences of all (p)`req` transitions such that they get just before firing of the corresponding (p)`take` transitions (if it exists). Then, in front of each `rel`(R) we move the `ptake` transitions which were enabled by it in β_n. This ensures that blocked processes remain blocked. The (p)`req` transitions of blocked processes are postponed to the end of α_n and are fired in their original order with the `preq` transitions coming in front of `req` transitions. We rename all preserved processes to numbers from 1 to n'. As in the proof of Lemma 3 we can show that we obtain a finite behaviour $\beta_{n'}$ which can be extended to a weakly fair infinite one. It is a counterexample because the order of actions of the visible process does not change from β_n to $\beta_{n'}$.

Case 3. The visible process is blocked and some invisible processes are running. This means that either all blocked processes are blocked because a resource is used forever or due to high priority requests of other processes, it is nether enabled (if not, the behaviour would be unfair). Therefore, we can not just take a behaviour with only the visible process. As in Case 2.2 we can suppose that the behaviour β_n is given by a prefix α_n followed by a behaviour γ_n repeated infinitely often. From now on we consider the *finite* behaviour $\beta_n = \alpha_n\gamma_n$ which can always be extended to an infinite one.

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, we preserve some invisible blocked processes and all invisible running processes and we reorder transitions. We preserve the same blocked processes as in case 2.2. We reorder β_n as in case 2.2. We obtain a finite 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, we can drop them from the behaviour γ_k to get a *simple* loop behaviour $\gamma_{k'}$. Thus, $\overline{\gamma}_{k'}$ is a quotient loop. Using Lemma 6 we obtain a quotient loop $\overline{\gamma'}$ using at most $B_{\mathcal{F}} - 2m - 1$ processes. Using Lemma A.2 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 6). Furthermore, the same control locations of Q_n 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 process not owning anything in c' are running (due to condition 3). We add to the loop γ' the blocked processes we have dropped earlier and the visible process. They stay blocked in the loop. 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. We obtain a weakly fair behaviour $\beta_{n'}$ which is a counterexample.

It remains to show that $n' \geq m + 1$. If $n' < m + 1$, we preserve some more blocked processes 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 configuration c is in 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 configuration c is in a control location $q \in Q_n$, then we can add the necessary number of processes to the behaviour $\beta_{n'}$: The additional processes first follow one-by-one the transitions of p_1 at the beginning of $\beta_{n'}$ and go to q (This is always possible, as in q they do not use or request a resource). Then, they mimic one after the other the behaviour of p_1. In such a way we obtain a weakly fair behaviour $\beta_{n'}$ with $n' \geq m + 1$. □

Lemma 8. *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 an infinite counterexample behaviour β_n of S_n such that $\pi_n, \nu_n \models \neg\varphi(p)$ holds for the path π_n corresponding to β_n and some valuation $\nu_n : PV \rightarrow P_n$, we show that there is an infinite 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 $\nu_{n'} : PV \rightarrow P_{n'}$ we have $\pi_{n'}, \nu_{n'} \models \neg\varphi(p)$. There are three cases. (1) All processes in β_n are blocked. (2) The visible process is running. (3) The visible process is blocked. Cases 1 and 2 are the same as in Lemma 5.

Case 3. We proceed as in the proof of Lemma 5 to get a counterexample behaviour β_n. Then, we can obtain a finite 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, $\overline{\gamma}_{k'}$ is a quotient loop of the form $(\mathsf{q}_1, x_1)t_1(\mathsf{q}_2, x_2)t_2 \ldots (\mathsf{q}_\mathsf{L}, x_L)$. It satisfies the property $\forall i : 1 \leq i \leq L : |\mathsf{q}_\mathsf{i}| \leq m + 1$. This is true due to the reordering of transitions in β_n: It is clear that at most m processes are in a location where they own one or several resources. Furthermore, there may be k more processes in an internal control location having requested some resources which are released directly after these requests. If there are k such processes, then only $m - k + 1$ processes are in locations owning something, because at least one process must own k resources (those who are released).

Then, we set $min = minimum\{x_1, \ldots, x_L\}$. Clearly, $(\mathsf{q}_1, x_1 - min)t_1(\mathsf{q}_2, x_2 - min)t_2 \ldots (\mathsf{q}_\mathsf{L}, x_L - min)$ is still a quotient loop with $\exists i : x_i = 0$. Together with the fact that we have $\forall i : 1 \leq i \leq L : |\mathsf{q_i}| \leq m+1$ this gives $\forall i : 1 \leq i \leq L : |\mathsf{q_i}| + x_i \leq m+1$. This means that we have a quotient loop with at most $m+1$ running processes. As in the proof of Lemma 5 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 5. $\square$

A.4 Process Deadlockability

Let us have an RTR family $\mathcal{F}$ over an automaton $\mathcal{A} = (R, Q, q_0, T)$ and a system $S_n \in \mathcal{F}$. Suppose a process p_1 is requesting a set of resources R' in a configuration $c \in C_n$, i.e. $c(p) = q_2$ for some $(q_1, (\mathtt{p})\mathtt{req}(R'), q_2) \in T_t$. 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 $s_1 = \mathtt{pr} \Rightarrow s_2 \neq \mathtt{r}$. 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, $\mathtt{r}$ items are not powerful enough. We say 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 is blocked by some $p_2 \in P$ on some resource $r \in R'$.

Note that all processes in a set P of processes statically deadlocked in c are 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 $(\mathtt{p})\mathtt{take}$ 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 $(\mathtt{p})\mathtt{rqt}$ transition inside which it is in c. Then, the $(\mathtt{p})\mathtt{take}$ 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.

Lemma A.6. *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 fulfill the assumption of the implication of the lemma for some RTR family $\mathcal{F}$.

Let P' be the maximum subset of P such that in c, one of the following conditions holds for each $p \in P'$: (1) p is the current user of some resource r (i.e. $c(r) \in \mathtt{u}(p).\Sigma_n^*$). (2) p is inside some $\mathtt{prqt}(R')$ operation and on some $r \in R'$, it is not blocked by any $p' \in P$. (3) p is inside some $\mathtt{rqt}(R')$, it is not blocked by any $p' \in P$ on some $r \in R'$, and no process in P is inside some $\mathtt{prqt}(R'')$ with $r \in R''$.

The above conditions justify 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 the behaviour that leads to c. Let ρ'_n be its modification obtained by removing transitions not fired by processes of P' and by a subsequent reordering of the remaining transitions as described in the proof of Lemma 3 with the following change: At the end of ρ'_n, we first fire all the preserved `preq` transitions from ρ_n that are not followed by the corresponding `ptake` transitions in ρ_n and we keep their mutual ordering. Only then we fire all the preserved `req` transitions of ρ_n that are not followed by the corresponding `take` transitions in ρ_n and we again keep their mutual ordering.

Let c' be the configuration we reach via ρ'_n. We show that in c', P' is a set of statically deadlocked processes. The (`p`)`req` transitions that were fired by processes P' in ρ_n and that were not followed by an occurrence of the corresponding (`p`)`take` transitions are preserved, no additional (`p`)`take` transitions are fired, and so all processes P' are inside some (`p`)`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.)

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 (`p`)`take` transitions that are fired by processes P' in ρ_n are fired by them in ρ'_n 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 $\mathtt{pr}(p')$ 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 ρ'_n. As no overtaking among not granted requests is possible in RTR, the request that produced the concerned $\mathtt{pr}(p'')$ 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 ρ'_n guarantees that it still precedes the latter one. Then, if the former request is granted in ρ'_n, it is not released (no additional `rel` 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 $\mathtt{pr}(p')$ item belonging to some $p' \in P$. The now granted request is clearly some low-priority request $\mathtt{rqt}(R')$. 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 ρ'_n. The construction of ρ'_n guarantees that the high-priority request that generated the concerned $\mathtt{pr}(p'')$ item to the queue of r precedes the now granted low-priority request in ρ'_n. 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 $\mathtt{rqt}(R')$, 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 $\mathtt{r}(p')$ 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 ρ'_n, 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, 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 ρ'_n.

Finally, in RTR, neither the concrete names of processes nor a presence of some processes idling at q_0 can influence the firability of a transition sequence. Therefore, in $S_{n'}$ with $n' = |P'| \leq m$, it is possible to reach a configuration c'' equal to c' up to renaming of the processes and up to ignoring the processes outside P'. Moreover, the notion of a set of statically deadlocked processes does not rely upon processes outside of this set, and thus removing the processes idling at q_0 cannot influence the fact that c'' contains a set of statically deadlocked processes. □

Lemma A.7. *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 fulfill 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, we can reach a configuration that is in some terminal strongly-connected component

(TSCC). So, there is $c_1 \in reach(S_n, c)$ 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.

We modify ρ_n to ρ'_n by leaving out transitions fired by processes P and by a subsequent reordering as in the proof of Lemma 3 with the following modification: At the end when firing the (`p`)`req` transitions not followed by the corresponding (`p`)`take` ones in ρ_n, we first fire all the requests that become granted in β_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 use the proof argument of Lemma 3 to show that ρ'_n is firable. The modification at the end of ρ'_n does not break the proof because all the concerned transitions are (`p`)`req` transitions, which become enabled in the appropriate processes simply when the processes are at the source control locations of such transitions. We show that the configuration c_2 we reach through ρ'_n 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_2.

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_2 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_1 are granted to them in c_2 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 β_n but not followed by firing of the corresponding (`p`)`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_1—the not deadlocked processes would have to be deadlocked; the dynamically deadlocked process could not be dynamically deadlocked. Thus, in c_1, 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_1, it remains blocked by them in c_2 as well. (The same `g`/`u` queue items of P' are generated in β'_n based on ρ'_n as in β_n and they appear before the transitions to be blocked are fired.) Moreover, if the given not granted request is blocked in c_1 by `pr` queue items produced by some high-priority request of P', it remains blocked in c_2 too. (According to the definition of RTR, to be able to play the described blocking role in c_1, 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_1, 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_2 in this last case too.

We have 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. Thus, the ordering of the queue items belonging to P' is preserved in c_2, and the proof of the described relation of c_1 and c_2 is now complete.

Next, we further modify β'_n by entirely leaving out all the idling processes and by a subsequent renaming of the preserved processes to $1, 2, ..., n'$. As neither the names of processes nor a presence of some processes idling at q_0 is significant in RTR, we obtain a behaviour $\beta_{n'}$ 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 it is possible to unblock p' 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)req` 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 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 in a dynamic deadlock.

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 reordering 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 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. $\square$

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, we can reach a configuration that is in some TSCC. Therefore, there is $c' \in reach(S_n, c)$ 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 A.6.

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, wrt. the proof argument of Lemma 3, 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)take(R') transition t, and t can never become enabled for p' from c'. Indeed, all other kinds of transitions than (p)take are automatically enabled in RTR when their source control location is reached, and, on the other hand, (p)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 we are in a TSCC, the excluded possibilities are excluded forever. Therefore, p' is dynamically deadlocked in c', and we can apply Lemma A.7.

It remains to show that if a process deadlock is possible in some S_n, it is also possible in any $S_{n'}$ with $n \leq n'$. This is, however, relatively easy. 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. (In the case of overtaking among transitions already started in c', we know that c' is in a TSCC, and reordering can be applied from some of its predecessors in the TSCC.) $\square$

A.5 RTR Families and Undecidability

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

Proof. We prove undecidability of checking $\mathcal{F}, 1 \models^a_{fin} \Phi_a$ via undecidability of the dual problem of checking $\mathcal{F}, 1 \models^e_{fin} \Phi_e$.

We show that for any PDA M with one, two, or more push-down stacks, we can construct an RTR\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 generalized 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 $\Box$ 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\P control automaton of the simulating RTR\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 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

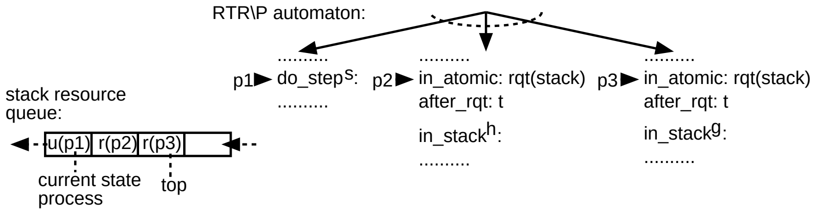

Fig. 5. Encoding of the (s, hg) configuration of a PDA being simulated

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 initialization of the simulation.

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 $\texttt{do\_step}^s$ (cf. Fig. 8 (a)). (2) `init` and `other` are owned by an initialization process that has already terminated its activity at the location `init_end` (see Fig. 8 (b)). (3) Stack-symbol processes blocked by `rqt(stack)` before the appropriate $\texttt{in\_stack}^X$ locations (cf. Fig. 8 (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. 5). (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. 8 (a), (b)). (5) All other processes are before (or inside) `rqt(init)` or `rqt(other)` (cf. Fig. 8 (b)) and will not 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 $\texttt{do\_step}^s$. We build the RTR\P automaton such that the s-state process may do a τ step to `accept` and terminate the simulation (cf. Fig. 8 (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 $\texttt{do\_step}^s$.

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 $\texttt{do\_step}^{s_1}$ 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 $t_1 \equiv s_1 - a, g/\gamma \rightarrow s_2$ where $a \in \Sigma_M$, $g \in \Gamma_M$, $\gamma \in \Gamma_M^*$. The corresponding control structure is detailed in Fig. 8. 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 trying to take `stack` back. Unless the formula $\varphi_1 \equiv \Box\ \neg((\exists p_1 :$

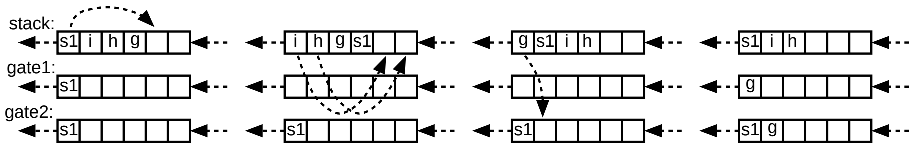

Fig. 6. Popping the top of the stack (without testing its type)

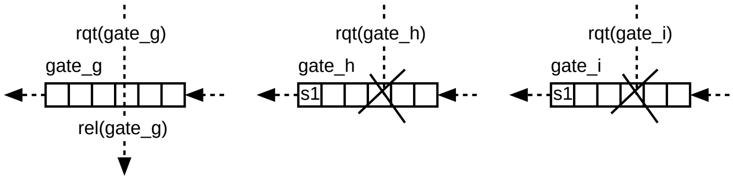

Fig. 7. Checking whether a process represents symbol g

$\texttt{in\_atomic}(p_1)) \ \wedge \ (\exists p_2 : \texttt{after\_rqt}(p_2)))$[6] 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 $\texttt{in\_stack}^X$ 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 $\texttt{rqt}(\texttt{gate\_}X)$ 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. Fig. 5, 6, and 7). 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 $\varphi_2 \equiv \Box \ \neg((\exists p_1 : \texttt{out\_check}(p_1)) \ \wedge \ \neg(\exists p_2 : \texttt{out\_done}(p_2)))$ 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) $\texttt{in\_stack}^X$ but `in_input`, and their type is "set" and not "checked" by the appropriate $\texttt{rqt}(\texttt{gate\_}X)$ 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 generalizes to PDAs with any (finite) number of push-down stacks.

---

[6] We suppose a formula $\texttt{label}(p)$ to hold iff p is at a location marked by `label:`.

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 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_`s_2 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_step`^{s} location (which is necessary for encoding configurations of M and for further simulation steps and/or acceptation). 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_step`^{s} 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`.

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\P control automaton, an s'-state process p may terminate the simulating run such that the formula $\varphi_3 \equiv \Diamond(\exists p_1 : \mathtt{accept}(p_1))$ 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. 8 (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 fulfills $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ and that leads from $c \in C_n$ corresponding to (s, γ) and having d fresh processes ready to be used in `input`.

It remains to show that the RTR\P automaton may be equipped with an initialization part such that starting from c_0, we either deadlock or invalidate some correctness requirement before any process can reach some $\texttt{do\_step}^s$ location, 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 initializing ones) ready in `input` is to be guaranteed. A solution to this problem is depicted in Fig. 8 (b).

The `rqt(init)` operation without any subsequent release of `init` ensures that there will appear at most one initialization process. The formula $\varphi_4 \equiv (\Diamond\ (\exists p_1 : \texttt{init\_ok}(p_1))) \wedge (\Box\ \neg \exists p_1, p_2 : (\texttt{init\_ok}(p_1) \wedge \neg \texttt{initial\_state}(p_2)))$ then ensures that in a valid behaviour there will appear some initialization process and that it will grab all "working" resources before any other process starts. After reaching `init_ok`, the initialization 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 initialization 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. $\square$

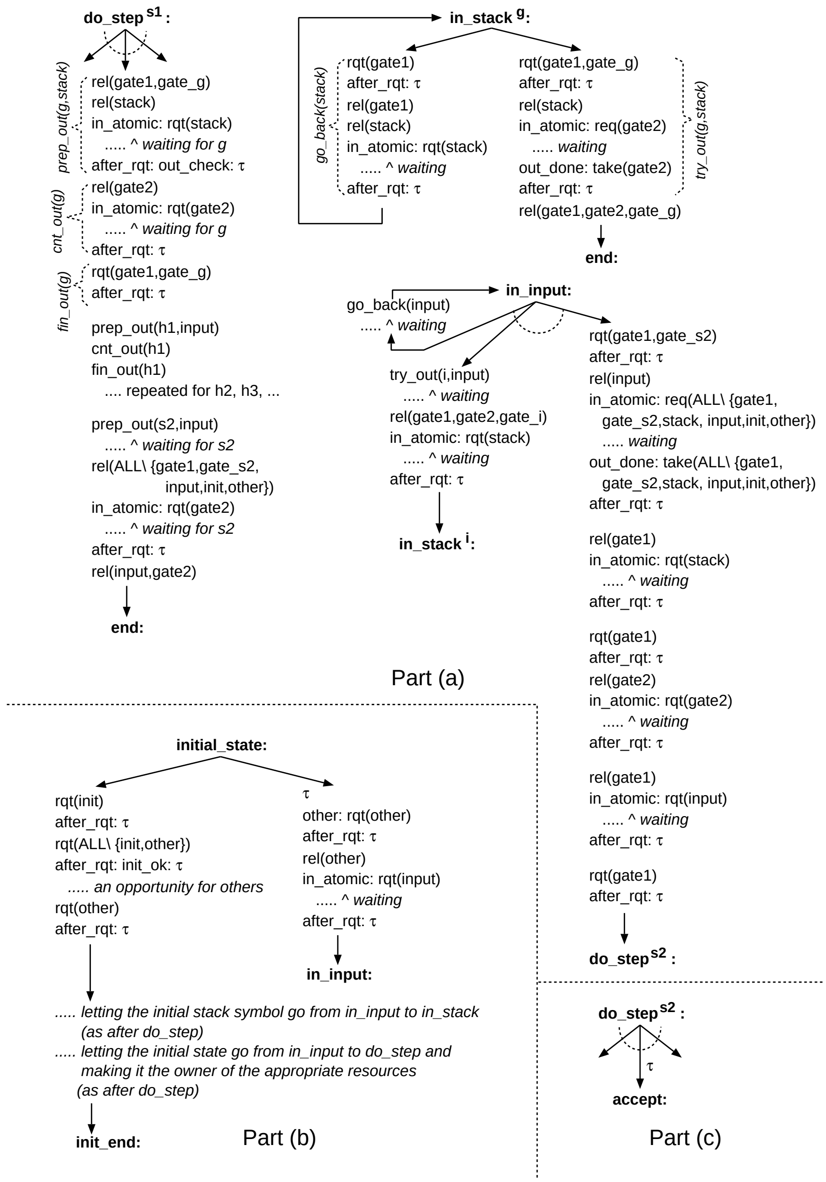

Fig. 8. (a) Simulating a PDA step $s_1 \xrightarrow{a,g/\gamma} s_2$ with $s_1, s_2 \in Q_M$, $a \in \Sigma_M$ (left out), $g \in \Gamma_M$, and $\gamma \in \Gamma_M^*$. (At the beginning of the step, an s_1-state process is the owner of all the resources besides `init` and `other`; processes representing 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) Initializing the simulation of a PDA. (c) Simulating the termination of a (possibly) accepting run of a PDA.