Parameterized Systems with Resource Sharing

Ahmed Bouajjani, Peter Habermehl, Tomáš Vojnar

LIAFA, Paris University 7
1. Introduction

The resource management problem:

an arbitrary number of client processes compete for an access to an arbitrary number of resources under the supervision of a single locker process
1. Introduction 2/4

- Due to the *broad importance* of the problem, it is interesting to be able to deal with
  - different classes of clients and/or lockers and
  - different classes of properties.
1. Introduction

- Due to the *broad importance* of the problem, it is interesting to be able to deal with
  - different *classes of clients* and/or *lockers* and
  - different *classes of properties*.

- In general, the problem is very complex – it involves coping with
  - up to two *parameters* and
  - (possibly several different) *infinite data structures* in lockers.
1. Introduction

Due to the *broad importance* of the problem, it is interesting to be able to deal with

- different classes of clients and/or lockers and
- different classes of properties.

In general, the problem is very complex – it involves coping with

- up to two parameters and
- (possibly several different) infinite data structures in lockers.

In this work, we restrict ourselves to dealing with:

- queue-based locker strategies
- a fixed number of resources
- a parametric number of identical clients
1. Introduction 3/4

- We split the problem into two parts:
  1. verifying systems of clients provided they are controlled by a locker with a certain locker strategy
  2. checking that the locker implements the appropriate strategy
1. Introduction

- We split the problem into two parts:
  1. verifying systems of clients provided they are controlled by a locker with a certain locker strategy
  2. checking that the locker implements the appropriate strategy

- We concentrate on the first issue for two important locker strategies: FIFO and FIFO with priorities.
1. Introduction

- Different approaches to verifying parameterized/infinite-state systems have been proposed: symbolic methods, network invariants, cut-offs, ...

- We have chosen the use of cut-offs:

  We are looking for “cut-off” numbers of clients such that verifying systems with up to this number of clients is enough to verify systems with an arbitrary number of clients.
1. Introduction

- Different approaches to verifying parameterized/infinite-state systems have been proposed: symbolic methods, network invariants, cut-offs, ...

- We have chosen the use of cut-offs:

  We are looking for “cut-off” numbers of clients such that verifying systems with up to this number of clients is enough to verify systems with an arbitrary number of clients.

- We obtain three kinds of results:
  - structure independent cut-offs
  - structure dependent cut-offs
  - undecidability
An Overview of the Rest of the Talk

❖ RTR families

❖ Specifying properties to be checked

❖ Verification of
  • finite behaviour
  • fair behaviour
  • process deadlockability

❖ Undecidability
2. RTR Families of Systems  (1/3)

- An RTR family $\mathcal{F}$ of systems of identical processes is given by:
  1. a finite set of resources $R$

2. RTR Families of Systems

- An RTR family $\mathcal{F}$ of systems of identical processes is given by:
  1. a finite set of resources $R$
  2. a finite control of the processes defined by:
     - a finite set of control states $Q$
     - the initial control state $q_0 \in Q$
     - a transition relation $T \subseteq Q \times A \times Q$ with $A$ including:
       - $\tau$
       - $\text{req}(R'), \text{take}(R'), \text{and } \text{rel}(R')$
       - $\text{rqt}(R')$
       - $\text{preq}(R'), \text{ptake}(R'), \text{and } \text{prqt}(R')$
2. RTR Families of Systems (1/3)

- An **RTR family** $\mathcal{F}$ of systems of identical processes is given by:
  1. a finite set of **resources** $R$
  2. a finite **control** of the processes defined by:
      - a finite set of control states $Q$
      - the initial control state $q_0 \in Q$
      - a transition relation $T \subseteq Q \times A \times Q$ with $A$ including:
         - $\tau$
         - $\text{req}(R'), \text{take}(R'), \text{and rel}(R')$
         - $\text{rqt}(R')$
         - $\text{preq}(R'), \text{ptake}(R'), \text{and prqt}(R')$
  3. a **locker policy** $L$ (FIFO or PRIO)
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:

```plaintext
rqt(A) → rel(A) → rel(A,B)
rqt(B) → rel(B) → rel(A,B)
req(A,B) → take(A) → rel(A,B)
take(B) → rel(A,B)
```

A: 

B: 

---

Parameterized Systems with Resource Sharing – p.8/36
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:

![Diagram of the FIFO locker policy]

- A:
- B:

Parameterized Systems with Resource Sharing – p.8/36
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:

```
req(A,B)
take(A)
take(B)
rel(A,B)
rqt(A)
rqt(B)
rel(A)
rel(B)
```

```
1  2  3
```
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:

```
1. rq(A) -> rel(A)
2. rq(B) -> rel(B)
3. req(A,B) -> take(A) -> take(B) -> rel(A,B)
```

Diagram:

```
A:      B:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>r(3)</td>
<td>u(1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>r(3)</td>
<td></td>
</tr>
</tbody>
</table>
```

Parameterized Systems with Resource Sharing – p.8/36
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:

```
1. \text{rq}t(A) \rightarrow \text{rel}(A) \\
2. \text{rq}t(B) \rightarrow \text{rel}(B) \\
3. \text{req}(A,B) \rightarrow \text{t}(3) \rightarrow \text{take}(A) \rightarrow \text{take}(B) \rightarrow \text{rel}(A,B)
```

- A:
  ```
  t(3)
  u(1)
  r(3)
  ```

- B:
  ```
  t(2)
  ```
2. RTR Families of Systems (2/3)

- The **FIFO locker policy**:

```
2. rqt(A)  3 take(A)
   1 rel(A)  3 rqt(B)  1 take(B)
      req(A,B)  2 rqt(B)  2 rel(B)  2 rel(A)
```

A:  
```
  t(2)
g(3)
```

B:  
```
r(3)
```
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:

```
\begin{align*}
\text{req}(A,B) & \xrightarrow{1} \text{rel}(A) \\
\text{req}(A,B) & \xrightarrow{2} \text{rel}(B) \\
\text{req}(A,B) & \xrightarrow{3} \text{take}(A) \\
\text{req}(A,B) & \xrightarrow{} \text{take}(B) \\
\text{req}(A,B) & \xrightarrow{} \text{rel}(A,B) \\
\end{align*}
```

A: \hspace{2cm} B:

- A: 
  
- B:
  
  - t(2)
  
  - r(3)

Parameterized Systems with Resource Sharing – p.8/36
2. RTR Families of Systems  

❖ The **PRIIO** locker policy:

```
\begin{align*}
&\text{A:} \\
&\quad \downarrow \\
&\quad 2 \text{ prqt}(B) \quad 3 \text{ rqt}(A,B) \\
&\text{B:} \\
&\quad \downarrow \\
&\quad \text{t(3)} \quad \text{pt(2)} \\
&\text{A:} \\
&\quad \downarrow \\
&\quad 2 \text{ prqt}(B) \quad 3 \text{ rqt}(A,B) \\
&\text{B:} \\
&\quad \downarrow \\
&\quad \text{t(3)} \quad \text{pt(2)} \\
&\quad \text{u(1)} \quad \text{t(3)} \\
&\text{A:} \\
&\quad \downarrow \\
&\quad 2 \text{ prqt}(B) \quad 3 \text{ rqt}(A,B) \\
&\text{B:} \\
&\quad \downarrow \\
&\quad \text{t(3)} \quad \text{pt(2)} \\
&\quad \text{u(1)} \quad \text{t(3)} \\
&\quad \text{u(1)}
\end{align*}
```
3. Properties to Check

- We can build on the notion of ICTL*.
3. Properties to Check

- We can build on the notion of ICTL*.

- **Global process quantification** – valid along paths:
  - **mutual exclusion**: $\forall p_1, p_2 \ AG (\neg (.p_1 = q_{cs} \land .p_2 = q_{cs}))$
  - **absence of starvation**: $\forall p \ AG (.p = q_{req} \Rightarrow AF .p = q_{grant})$
3. Properties to Check

- We can build on the notion of ICTL*.

Global process quantification – valid along paths:
- mutual exclusion: $\forall p_1, p_2 \ AG \neg (.p_1 = q_{cs} \land .p_2 = q_{cs})$
- absence of starvation: $\forall p \ AG (.p = q_{req} \Rightarrow AF .p = q_{grant})$

Local process quantification – valid within states:
- global response: $AG ((\exists p .p = q_{req}) \Rightarrow AF (\exists p .p = q_{resp}))$
3. Properties to Check

- We can build on the notion of ICTL*.

- **Global process quantification** – valid along paths:
  - **mutual exclusion**: \( \forall p_1, p_2 \; AG \neg (.p_1 = q_{cs} \land .p_2 = q_{cs}) \)
  - **absence of starvation**: \( \forall p \; AG (.p = q_{req} \Rightarrow AF .p = q_{grant}) \)

- **Local process quantification** – valid within states:
  - **global response**: \( AG ((\exists p \; .p = q_{req}) \Rightarrow AF (\exists p \; .p = q_{resp})) \)

- We will consider the **parametric verification problem** in the form:

\[
\forall S \in \mathcal{F} : S \models \Phi \quad \text{or} \quad \exists S \in \mathcal{F} : S \models \Phi
\]
4. Verification of Finite Behaviour

- We consider properties of the form

\[ \Phi_{fin}^k \equiv [\exists \forall]_{p_1, \ldots, p_k} [E|A]_{fin} \varphi(p_1, \ldots, p_k) \]

where:

1. \( p_1, \ldots, p_k \) are required to refer to different processes
2. \( \varphi(p_1, \ldots, p_k) \) is an LTL\( \setminus X \) formula over atoms of the kind \( p_i = q \)
4. Verification of Finite Behaviour

>We consider properties of the form

\[ \Phi^k_{fin} \equiv [\exists | \forall]_{p_1, \ldots, p_k} [E | A]_{fin} \varphi(p_1, \ldots, p_k) \]

where:

1. \( p_1, \ldots, p_k \) are required to refer to different processes
2. \( \varphi(p_1, \ldots, p_k) \) is an LTL\( \setminus \)X formula over atoms of the kind \( .p_i = q \)

>**Mutual exclusion** is an example of such a property:

\[ \forall_{p_1, p_2} A_{fin} G (\neg (.p_1 = q_{cs} \land .p_2 = q_{cs}) \]
5. Finite Behaviour of $\text{RTR}_{FIFO}$ (1/3)

- In order to verify $\forall S \in \mathcal{F} : S \models \Phi_{fin}^k$ within $\text{RTR}_{FIFO}$, it is enough to consider systems with $k$ processes.

- In other words, the following holds:

$\forall S \in \mathcal{F} : S \models \Phi_{fin}^k \iff S_k \models \Phi_{fin}^k$
5. Finite Behaviour of RTR\textsc{FIFO} (2/3)

\begin{itemize}
  \item proof sketch:
  \begin{itemize}
    \item \( \forall l \geq k : S_k \models \exists_{p_1, \ldots, p_k} E_{fin} \varphi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{fin} \varphi \)
  \end{itemize}
\end{itemize}
5. Finite Behaviour of RTR\textsubscript{FIFO} (2/3)

- proof sketch:
  \begin{itemize}
  \item \(\forall l \geq k : S_k \models \exists p_1,\ldots,p_k \ E_{fin} \varphi \iff S_l \models \exists p_1,\ldots,p_k \ E_{fin} \varphi\)
  \end{itemize}

  \((\Rightarrow)\) Obvious—we let the additional processes of \(S_l\) idle.
5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

- **proof sketch:**
  - $\forall l \geq k : S_k \models \exists p_1, \ldots, p_k \ E_{fin} \varphi \iff S_l \models \exists p_1, \ldots, p_k \ E_{fin} \varphi$

  (⇒) **Obvious**—we let the additional processes of $S_l$ idle.

  (⇐) We take a witness from $S_l$, remove actions of invisible processes, and obtain a witness in $S_k$: 
5. Finite Behaviour of RTR\textsuperscript{FIFO} (2/3)

- **proof sketch:**
  - \( \forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k} E_{fin} \varphi \iff S_l \models \exists_{p_1,\ldots,p_k} E_{fin} \varphi \)

  (\( \Rightarrow \)) Obvious—we let the additional processes of \( S_l \) idle.

  (\( \Leftarrow \)) We take a witness from \( S_l \), remove actions of invisible processes, and obtain a witness in \( S_k \):
  - A removal of \texttt{req, take} makes the situation for other processes “easier”.
  - As we always start with empty queues, \texttt{rel} just neutralizes \texttt{req, take}.
5. Finite Behaviour of \( \text{RTR}^{\text{FIFO}} \) (2/3)

- **proof sketch:**
  - \( \forall l \geq k : S_k \models \exists_{p_1, \ldots, p_k} E_{\text{fin}} \varphi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{\text{fin}} \varphi \)

  \((\Rightarrow)\) Obvious—we let the additional processes of \( S_l \) idle.
  
  \((\Leftarrow)\) We take a witness from \( S_l \), remove actions of invisible processes, and obtain a witness in \( S_k \):
  - A removal of \( \text{req, take} \) makes the situation for other processes “easier”.
  - As we always start with empty queues, \( \text{rel} \) just neutralizes \( \text{req, take} \).
  - We remove actions of invisible processes and LTL\( \setminus X \) is stuttering insensitive.
5. Finite Behaviour of RTR$_{FIFO}$ (3/3)

- If the control of processes of a given family does not contain a loop with 
  $\#\text{req}(r) > \#\text{take}(r)$, $r \in R$, we suffice with finite-state techniques.

- The above restriction is relatively practical because if it does not hold, 
  there is either a possibility of a process deadlock, or the content of the 
  queues may grow over every bound.

- The case of the (RT)R$_{FIFO}$ families where only $rq_t$ is used is covered.
5. Finite Behaviour of $\text{RTR}_{FIFO}$ (3/3)

- If the control of processes of a given family does not contain a loop with $\#\text{req}(r) > \#\text{take}(r)$, $r \in R$, we suffice with finite-state techniques.

- The above restriction is relatively practical because if it does not hold, there is either a possibility of a process deadlock, or the content of the queues may grow over every bound.

- The case of the $(\text{RT})\text{R}_{FIFO}$ families where only $r_{qt}$ is used is covered.

- The described result cannot be used within $\text{RTR}_{Prio}$ nor $(\text{RT})\text{R}_{Prio}$. 
6. Finite Behaviour of $(RT)R_{PRIO}$ (1/3)

- In $(RT)R_{PRIO}$, when we remove actions of some processes from a behaviour, we need not obtain a behaviour:

```
1. rqt(A)-start
2. prqt(B)-start
3. rqt(A,B)-start
```

```
B.1

AB.1

B.2

AB.2

```

```
2. rel(A)-start
3. rel(A,B)-start
```

```
1. rqt(A)-end
2. prqt(B)-end
```

```
A: B:
```

Parameterized Systems with Resource Sharing – p.15/36
6. Finite Behaviour of (RT)R^{PRIO} (1/3)

❖ In (RT)R^{PRIO}, when we remove actions of some processes from a behaviour, we need not obtain a behaviour:

We cannot go down to \( k \) processes here:

\[
\exists S \in \mathcal{F} : S \models \exists_{p_1, p_2} E_{fin} \\
((p_2 \neq B_1) \cup (p_1 = AB_1)) \land ((p_1 \neq AB_2) \cup (p_2 = B_2))
\]
6. Finite Behaviour of \((RT)R_{PRIO}\) (2/3)

- For reachability/invariance properties based on \([EF | AG]\) \(\pi(p_1, \ldots, p_k)\), we suffice with \(k\) processes in \((RT)R_{PRIO}\).
6. Finite Behaviour of $(RT)R_{PRIO}$ (2/3)

- For reachability/invariance properties based on $[EF|AG] \pi(p_1, \ldots, p_k)$, we suffice with $k$ processes in $(RT)R_{PRIO}$.

- **proof sketch:**
  
  \begin{itemize}
  \item $\forall l \geq k :$
  \begin{align*}
  S_k &\models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \iff S_l &\models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \\
  1. rqt(A)-start &\quad 3. rqt(A,B)-start \\
  2. rel(B) &\quad 1. rqt(A)-end \\
  2. prqt(B)-end &\quad 2. prqt(B)-start
  \end{align*}
  \end{itemize}
6. Finite Behaviour of \((RT)R_{PRIO}\) (2/3)

- For reachability/invariance properties based on \([EF|AG] \pi(p_1, \ldots, p_k)\), we suffice with \(k\) processes in \((RT)R_{PRIO}\).

- **proof sketch:**
  
  - \(\forall l \geq k: S_k \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi\)
  
  - (\(\Rightarrow\)) Obvious—we let the additional processes of \(S_l\) idle.
6. Finite Behaviour of \((\text{RT})R_{PRIO}\) (2/3)

- For reachability/invariance properties based on \([EF|AG]\pi(p_1, \ldots, p_k)\), we suffice with \(k\) processes in \((\text{RT})R_{PRIO}\).

- proof sketch:
  
  - \(\forall l \geq k:\)
    
    \[S_k \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi\]

    \((\Rightarrow)\) Obvious—we let the additional processes of \(S_l\) idle.

    \((\Leftarrow)\)  
    - We take a witness from \(S_l\).
    - We remove actions of invisible processes.
6. Finite Behaviour of (RT)R_{PRIO} (2/3)

 Exists reachability/invariance properties based on \([EF|AG] \pi(p_1, \ldots, p_k)\), we suffice with \(k\) processes in (RT)R_{PRIO}.

 proof sketch:

\( \forall l \geq k : \)
\[ S_k \models \exists p_1, \ldots, p_k \ E_{fin} F \pi \iff S_l \models \exists p_1, \ldots, p_k \ E_{fin} F \pi \]

\((\Rightarrow)\) Obvious—we let the additional processes of \(S_l\) idle.

\((\Leftarrow)\)
- We take a witness from \(S_l\).
- We remove actions of invisible processes.

Parameterized Systems with Resource Sharing – p.16/36
6. Finite Behaviour of $(RT)R_{PRIO}$ (2/3)

- For reachability/invariance properties based on $[EF | AG] \, \pi(p_1, \ldots, p_k)$, we suffice with $k$ processes in $(RT)R_{PRIO}$.

- **proof sketch:**
  - $\forall l \geq k : S_k \models \exists_{p_1, \ldots, p_k} E \, F \, \pi \iff S_l \models \exists_{p_1, \ldots, p_k} E \, F \, \pi$
    
    ($\Rightarrow$) Obvious—we let the additional processes of $S_l$ idle.
    
    ($\Leftarrow$) • We take a witness from $S_l$.
    • We remove actions of invisible processes.
6. Finite Behaviour of (RT)R_{PRIO} (2/3)

- For reachability/invariance properties based on $[EF|AG] \pi(p_1, \ldots, p_k)$, we suffice with $k$ processes in (RT)R_{PRIO}.

- proof sketch:
  - $\forall l \geq k :$
    $S_k \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi$
  
  ($\Rightarrow$) Obvious—we let the additional processes of $S_l$ idle.
  
  ($\Leftarrow$)
  - We take a witness from $S_l$.
  - We remove actions of invisible processes.
  - We postpone $rqt(R') - start$ to be just after all the “overtaking” $prqt(R'') - start$.  

\[1.\text{rqt(A)-start} \quad 1.\text{rqt(A)-end} \quad 3.\text{rqt(A,B)-start} \quad 2.\text{prqt(B)-start} \quad 2.\text{prqt(B)-end} \quad 2.\text{rel(B)}\]
6. Finite Behaviour of \( (RT)R_{PRIO} \) (2/3)

- For reachability/invariance properties based on \([EF|AG] \pi(p_1, \ldots, p_k)\), we suffice with \( k \) processes in \( (RT)R_{PRIO} \).

- **proof sketch:**
  - \( \forall l \geq k : \)
    \[
    S_k \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi
    \]
  - \( \Rightarrow \) Obvious—we let the additional processes of \( S_l \) idle.
  - \( \Leftarrow \) We take a witness from \( S_l \).
    - We remove actions of invisible processes.
    - We postpone \( rqt(R') - start \) to be just after all the “overtaking” \( prqt(R'') - start \).
6. Finite Behaviour of (RT)R<sub>PRI</sub>O (2/3)

- For reachability/invariance properties based on $[EF|AG] \pi(p_1, \ldots, p_k)$, we suffice with $k$ processes in (RT)R<sub>PRI</sub>O.

- proof sketch:
  - $\forall l \geq k :$
    - $S_k \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi \iff S_l \models \exists_{p_1, \ldots, p_k} E_{fin} F \pi$

  $(\Rightarrow)$ Obvious—we let the additional processes of $S_l$ idle.

  $(\Leftarrow)$
  - We take a witness from $S_l$.
  - We remove actions of invisible processes.
  - We postpone $rqt(R') - start$ to be just after all the “overtaking” $prqt(R'') - start$.
  - We obtain a behaviour in $S_k$.
  - The visible final state is not changed.
The result holds also for general LTL\(\setminus X\) formulae not distinguishing the control pre- and post-conditions of \(rqt(\rho') - \text{start}\).
7. Verification of Fair Behaviour

- We consider properties of the form

\[ \Phi_{wf}^k \equiv [\exists \forall]_{p_1, \ldots, p_k} [E|A]_{wf} \varphi(p_1, \ldots, p_k) \]

where:

1. \( \varphi \) is as in \( \Phi_{fin}^k \)
2. \( wf \) represents weak (process) fairness
7. Verification of Fair Behaviour

- We consider properties of the form

\[ \Phi_{wf}^k \equiv [\exists \forall]_{p_1, \ldots, p_k} [E \backslash A]_{wf} \varphi(p_1, \ldots, p_k) \]

where:

1. \( \varphi \) is as in \( \Phi_{fin}^k \)
2. \( wf \) represents weak (process) fairness

- Weak fairness coincides with strong fairness in our case.
In order to verify $\forall S \in \mathcal{F} : S \models \Phi^k_{wf}$ within $(RT)R_{FIFO}$ families with $|R| = m$, it is enough to consider systems with up to $m + k$ processes.
8. Fair Behaviour of $(RT)R_{FIFO}$ (1/3)

- In order to verify $\forall S \in \mathcal{F} : S \models \Phi^k_{wf}$ within $(RT)R_{FIFO}$ families with $|R| = m$, it is enough to consider systems with up to $m + k$ processes.

- **proof sketch:**
  $\forall l \geq m + k : S_{m+k} \models \exists_{p_1,\ldots,p_k} E_{wf} \varphi \iff S_l \models \exists_{p_1,\ldots,p_k} E_{wf} \varphi$

  ($\iff$) $m$ invisible processes can block all resources if need be.
8. Fair Behaviour of (RT)R_{FIFO} (1/3)

- In order to verify $\forall S \in \mathcal{F} : S \models \Phi^k_{wf}$ within (RT)R_{FIFO} families with $|R| = m$, it is enough to consider systems with up to $m + k$ processes.

- proof sketch:
  $\forall l \geq m + k : S_{m+k} \models \exists p_1,\ldots,p_k \ E_{wf} \varphi \iff S_l \models \exists p_1,\ldots,p_k \ E_{wf} \varphi$

$(\iff)$ $m$ invisible processes can block all resources if need be.

$(\implies)$ Additional processes can be added:
- No process is running – trivial.
- All processes are running – cf. the next slide.
- Otherwise – a combination of the above.
8. Fair Behaviour of (RT)R\textsubscript{\textit{FIFO}} (2/3)

- Adding new processes when all original processes run forever:

  - At least one resource is always eventually released:
    - There is a state \(s\) where at least 1 resource is unused.
    - At most \(m\) processes may use some resources in \(s\).
    - At least \(k+1\) processes do not use any resource in \(s\).
    - There is an invisible process \(p\) not using anything in \(s\).
    - The behaviour of \(p\) can be mimicked.

  - No resource is ever released in the loop:
    - At most \(m\) processes use some resources in the loop.
    - At least \(k\) processes use no resources in the loop.
    - Any of the latter can be mimicked.
8. Fair Behaviour of \( (RT)R_{FIFO} \) (2/3)

- Adding new processes when all original processes run forever:

  At least one resource is always eventually released:
  - There is a state \( s \) where at least 1 resource is unused.
  - At most \( m - 1 \) processes may use some resources in \( s \).
  - At least \( k + 1 \) processes do not use any resource in \( s \).
  - There is an invisible process \( p \) not using anything in \( s \).
  - The behaviour of \( p \) can be mimicked.
8. Fair Behaviour of \((RT)R_{FIFO}\) (2/3)

- Adding new processes when all original processes run forever:

At least one resource is always eventually released:

- There is a state \(s\) where at least 1 resource is unused.
- At most \(m - 1\) processes may use some resources in \(s\).
- At least \(k + 1\) processes do not use any resource in \(s\).
- There is an invisible process \(p\) not using anything in \(s\).
- The behaviour of \(p\) can be mimicked.

No resource is ever released in the loop:

- At most \(m\) processes use some resources in the loop.
- At least \(k\) processes use no resources in the loop.
- Any of the latter can be mimicked.
8. Fair Behaviour of (RT)R_{FIFO} (3/3)

- Adding new processes when $b < m + k$ processes block forever:

  1. At least 1 process $p$ out of the $b$ processes does not use any resource (it is just asking for some).
     - $p$ can be easily mimicked.
8. Fair Behaviour of (RT)R_{FIFO} (3/3)

- Adding new processes when \( b < m + k \) processes block forever:

  1. At least 1 process \( p \) out of the \( b \) processes does not use any resource (it is just asking for some).
     - \( p \) can be easily mimicked.
  2. All of the \( b \) processes use some resources.
     - \( b \leq m \)
     - At least \( b \) resources cannot be used by the looping processes.
     - At most \( m - b = m' \) resources can be used by these processes.
     - There are \( m + k - b = m' + k \) looping processes.
     - With \( m' \) and \( m' + k \), we can use similar arguments as on the previous slide.
9. Fair Behaviour of $(RT)R_{PRIO}$ (1/5)

- The same result as for $(RT)R_{FIFO}$ cannot be obtained for $(RT)R_{PRIO}$. 
- Even for 1-process queries, there is no cut-off based just on $m$ and $k$ here.
9. Fair Behaviour of \((RT)R_{PRIO}\) (1/5)

- The same result as for \((RT)R_{FIFO}\) cannot be obtained for \((RT)R_{PRIO}\).
- Even for 1-process queries, there is no cut-off based just on \(m\) and \(k\) here.
- For example, we need \(l + 2\) invisible process to show starvation in:
9. Fair Behaviour of (RT)R_{PRIO} (2/5)

- A *structure-dependent* cut-off bound $F(|R|, |Q| + |T|)$ exists for (RT)R_{PRIO} and 1-process queries.
9. Fair Behaviour of (RT)R_{PRIO} (2/5)

- A *structure-dependent* cut-off bound $F(|R|, |Q| + |T|)$ exists for (RT)R_{PRIO} and 1-process queries.

- proof idea – showing that we can bound the number of invisible processes that keep running and block the visible process forever:
9. Fair Behaviour of (RT)R_{PRIO} (2/5)

❖ A *structure-dependent* cut-off bound $F(|R|, |Q| + |T|)$ exists for (RT)R_{PRIO} and 1-process queries.

❖ proof idea – showing that we can bound the number of *invisible processes that keep running and block* the visible process forever:

- By *reordering* of transition occurrences, we show that the queue content may be bounded.
9. Fair Behaviour of $(RT)R_{PRIO}$ (2/5)

❖ A structure-dependent cut-off bound $F(|R|, |Q| + |T|)$ exists for $(RT)R_{PRIO}$ and 1-process queries.

❖ proof idea – showing that we can bound the number of invisible processes that keep running and block the visible process forever:

- By reordering of transition occurrences, we show that the queue content may be bounded.
- The loop of the witness can be encoded such that:
  - We remember which control locations are occupied by processes using or requesting some resources.
  - We remember the number of other processes at each location.

(continued on the next slide)
9. Fair Behaviour of (RT)R<sub>PRIO</sub> (3/5)

❖ The proof idea continued:

- We construct a system of linear equations whose solutions describe loops over states encoded as above and guaranteeing that the visible process remains blocked.
9. Fair Behaviour of (RT)R_{PRIO} (3/5)

❖ The proof idea continued:

- We construct a system of linear equations whose solutions describe loops over states encoded as above and guaranteeing that the visible process remains blocked.
- All constants in the equations may be bounded, a solution exists (it is the given witness), and thus the theory of Linear Programming shows that there is a bounded solution.
The proof idea continued:

- We construct a system of linear equations whose solutions describe loops over states encoded as above and guaranteeing that the visible process remains blocked.
- All constants in the equations may be bounded, a solution exists (it is the given witness), and thus the theory of Linear Programming shows that there is a bounded solution.

The presented cut-off shows that the given problem is decidable, but the cut-off is not practical. We can further try to

- optimize the bound,
- which can be especially successful for subclasses of (RT)R_{PRIO}.
9. Fair Behaviour of (RT)R_{PRIO} (4/5)

- We call an (RT)R_{PRIO} family **simple** iff its control automaton contains just one “**free area**” through which processes may loop.

![Diagram of control automaton]
9. Fair Behaviour of (RT)R$_{PRIO}$ (4/5)

- We call an (RT)R$_{PRIO}$ family **simple** iff its control automaton contains just one “free area” through which processes may loop.

![Diagram](image)

- When verifying fair behaviour of simple (RT)R$_{PRIO}$ families against 1-process formulae, we suffice with considering up to $2m + 2$ processes.
proof idea:

Ensuring that the blocked visible process will remain blocked when removing some processes from the witness:

- We have $2m + 2$ processes: 1 visible blocked, up to $m$ invisible blocked, at least $m + 1$ running forever.
- When a process releases $l$ resources, at most $m - l$ processes can be using some resources.
- We have $(m + 1) - (m - l) - 1 = l$ processes ready in the free area to start blocking the released resources.
10. Process Deadlockability

❖ To check whether a process deadlock is possible in some system of an RTR_{FIFO} or (RT)R_{PRIO} family $\mathcal{F}$, it suffices to examine the system $S_{\max(m,2)} \in \mathcal{F}$ where $m = |R|$. 
10. Process Deadlockability

❖ To check whether a process deadlock is possible in some system of an RTR\textsubscript{FIFO} or (RT)R\textsubscript{PRIO} family $\mathcal{F}$, it suffices to examine the system $S_{\text{max}}(m,2) \in \mathcal{F}$ where $m = |R|$.

❖ The proof is simple for RTR\textsubscript{FIFO} where (besides some trivial cases) a process deadlock arises due to cyclic dependencies in the queues of the $m$ resources.
10. Process Deadlockability

❖ To check whether a **process deadlock** is possible in some system of an RTR\textsubscript{FIFO} or (RT)R\textsubscript{Prio} family \( \mathcal{F} \), it suffices to examine the system 
\[ S_{\text{max}}(m,2) \in \mathcal{F} \text{ where } m = |R|. \]

❖ The proof is simple for RTR\textsubscript{FIFO} where (besides some trivial cases) a process deadlock arises due to **cyclic dependencies** in the queues of the \( m \) resources.

❖ In (RT)R\textsubscript{Prio}, a process deadlock may arise due to **unavoidable overtaking** among some processes. Here, processes that always eventually do not use any resources are to be eliminated.
11. Some Undecidability Results

- In $\text{RTR}_{\text{FIFO}}$, **general reachability** referring arbitrarily both to the current control locations of processes and to the content of queues is **undecidable**.

- We can also show the following is **undecidable**:
  - for $\text{RTR}_{\text{FIFO}}$: the **EF fragment** of ICTL* with only global as well as only local process quantification
  - even for $(\text{RT})\text{R}_{\text{FIFO}}$: the **LTL\textit{\textbackslash X fragment** of ICTL* based on atomic formulae of the kind $\forall p \cdot p = q$
11. Some Undecidability Results (1/7)

- In $\text{RTR}_{FIFO}$, general reachability referring arbitrarily both to the current control locations of processes and to the content of queues is undecidable.

- We can also show the following is undecidable:
  - for $\text{RTR}_{FIFO}$: the EF fragment of ICTL* with only global as well as only local process quantification
  - even for $(\text{RT})\text{R}_{FIFO}$: the LTL\textbackslash X fragment of ICTL* based on atomic formulae of the kind $\forall p . p = q$

- Proof by reduction from testing nonemptiness of PDAs with two stacks – highly nontrivial because the queues are not communication queues, but just waiting queues.
11. Some Undecidability Results (2/7)

❖ proof idea:

- We show how to simulate PDAs in a way that can easily be generalized to using two stacks.
proof idea:

- We show how to simulate PDAs in a way that can easily be generalized to using two stacks.
11. Some Undecidability Results (2/7)

- proof idea:
  - We show how to simulate PDAs in a way that can easily be generalized to using two stacks.

\[ s_1 \xrightarrow{\Sigma \cup \{\varepsilon\}, \Gamma/\Gamma^*} s_2 \]

- The role of states, input symbols, and stack symbols is played by processes running in different control branches.
11. Some Undecidability Results  (3/7)

- proof idea (continued):
  - The **content of the queues** may be viewed by projecting PIDs to the control states of the appropriate processes (resp. the branches they are a part of).

```plaintext
\[ \begin{array}{c}
\text{output queue} \\
\vdots \\
\vdots \\
\vdots \\
3 \quad 5 \quad 9 \\
\text{a} \quad \text{c} \\
\end{array} \]
```

\[ a: 3 \quad c: 69 \]

Parameterized Systems with Resource Sharing – p.30/36
proof idea (continued):

- The simulation is **controlled** by state processes; if the current-state process \( p \) deadlocks, the whole system deadlocks.
proof idea (continued):

- The simulation is controlled by state processes; if the current-state process $p$ deadlocks, the whole system deadlocks.
- The stack is simulated by a resource that is normally owned by $p$ and stack-symbol processes wait in its queue; the top of the stack corresponds to the tail of the queue:
11. Some Undecidability Results  (5/7)

- proof idea (continued):
  - **Reading of the top symbol** can be implemented by \( p \) releasing a stack, taking it again, and ensuring that after some symbol process releases the stack and does not take it back, all further symbol processes will block before releasing the stack.
proof idea (continued):

- To **test whether a process plays the role $p$ expects it to play**, $p$ may let it take and release a resource characteristic for its control branch and release only a certain resource before such a check.
proof idea (continued):

- For the output to be valid, there must appear a word from a certain regular language in a checksum queue – this ensures that some process always did what \( p \) needed to be done.

\[
\text{... Q } \Gamma Q \Sigma Q \Gamma Q \Gamma Q \text{'Q} \Gamma Q \text{...}
\]

pop out push next state
11. Some Undecidability Results (7/7)

- proof idea (continued):
  - For the output to be valid, there must appear a word from a certain regular language in a checksum queue – this ensures that some process always did what \( p \) needed to be done.

\[
\ldots \ Q \ \Gamma Q \ \Sigma Q \ \Gamma Q \ \Gamma Q \ Q'Q \ \ldots \\
\quad \text{pop out} \quad \text{push} \quad \text{next state}
\]

- The use of the checksum queue may be replaced by checking satisfaction of suitable temporal logic formulae.
12. Conclusions (1/2)

- We have provided practical cut-off results for parametric verification of many important properties of the considered systems with resource sharing.

- We have also established some undecidability bounds and bounds of structure-independence for the application of cut-offs in the given domain.
12. Conclusions (1/2)

❖ We have provided **practical cut-off results** for parametric verification of many important properties of the considered systems with resource sharing.

❖ We have also established some **undecidability** bounds and bounds of **structure-independence** for the application of cut-offs in the given domain.

❖ In the future, we can try to
  - improve the **decidability/undecidability** bounds,
  - **optimize** the cut-off bound for verification of fair behaviour in $(RT)R_{PRIO}$,
  - establish some further practical cut-offs for **interesting subcases** of the problems found difficult in general.
12. Conclusions (2/2)

❖ For the cases where no cut-off or no small cut-off can be found, we can try to apply some other methods (e.g., symbolic verification).
12. Conclusions (2/2)

❖ For the cases where no cut-off or no small cut-off can be found, we can try to apply some other methods (e.g., symbolic verification).

❖ Finally, the following is also worth considering:
  • dealing with some other locker strategies than FIFO and PRIO
  • considering non-exclusive access to resources
  • verifying user-described lockers