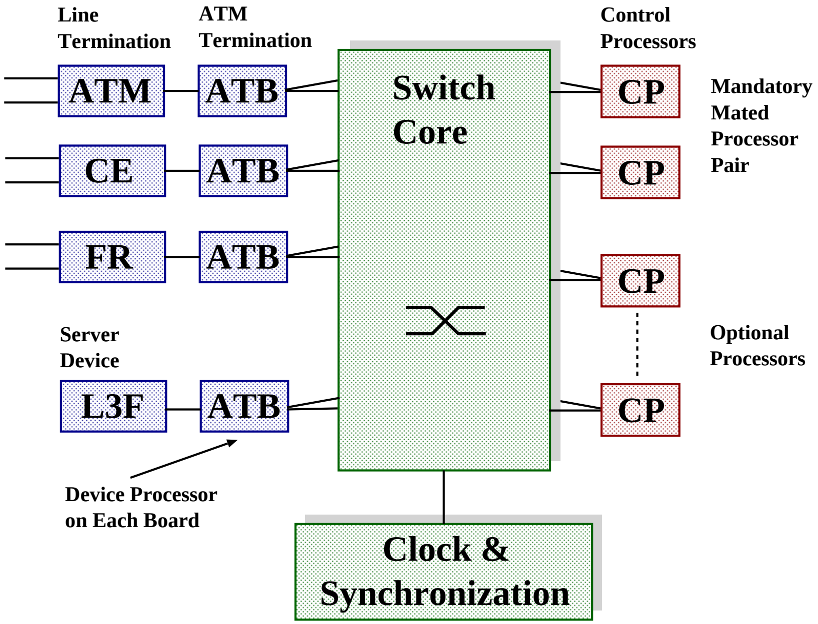
Parameterized Systems with Resource Sharing

Ahmed Bouajjani, Peter Habermehl, Tomáš Vojnar

LIAFA, Paris University 7

1. Introduction 1/5

❖ **Our original motivation: verifying the use of shared resources in Ericsson's AXD 301 switch.**

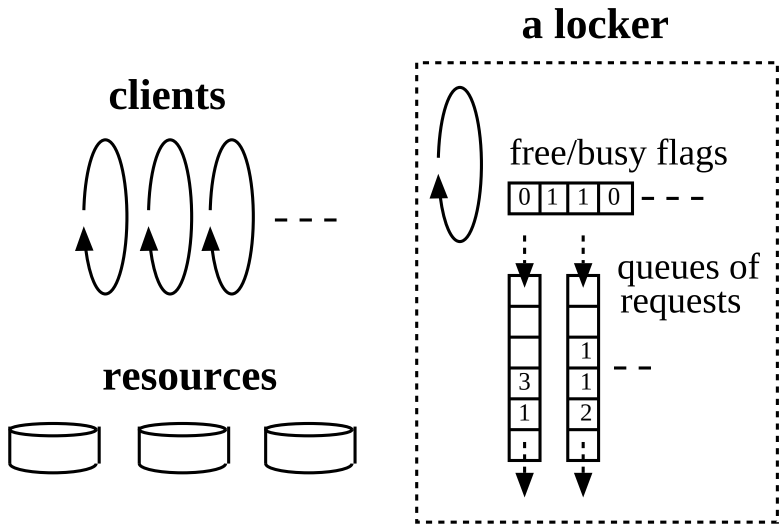


❖ **The problem is, however, common in operating systems, control software, multithreaded programs, etc.**

1. Introduction 2/5

❖ **A general view of 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 3/5

❖ **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 3/5

- **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 3/5

❖ 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.

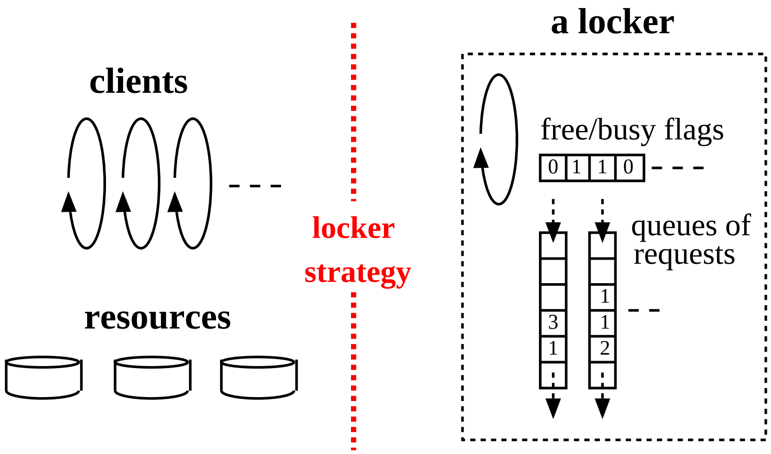
❖ In this work, we restrict ourselves to dealing with:

- queue-based locker strategies
- a fixed number of resources
- a parametric number of identical clients

1. Introduction 4/5

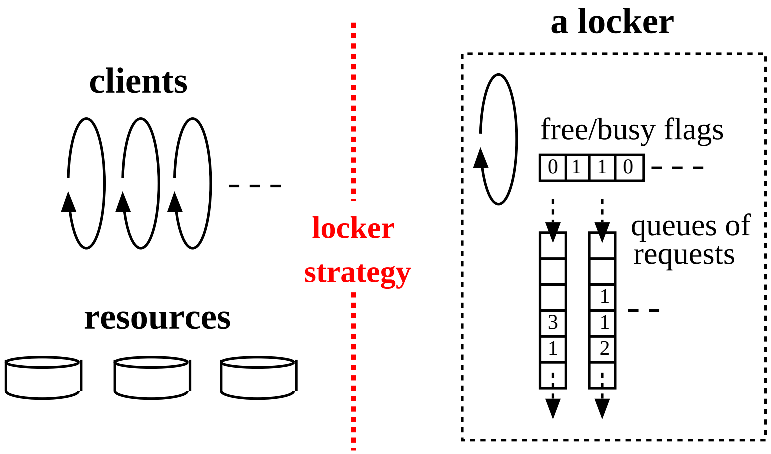
❖ We split the problem into two parts:

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 4/5

- **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 5/5

❖ 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 5/5

❖ **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 (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:
       - τ
       - $\texttt{req}(R')$, $\texttt{take}(R')$, and $\texttt{rel}(R')$
       - $\texttt{rqt}(R')$
       - $\texttt{preq}(R')$, $\texttt{ptake}(R')$, and $\texttt{prqt}(R')$

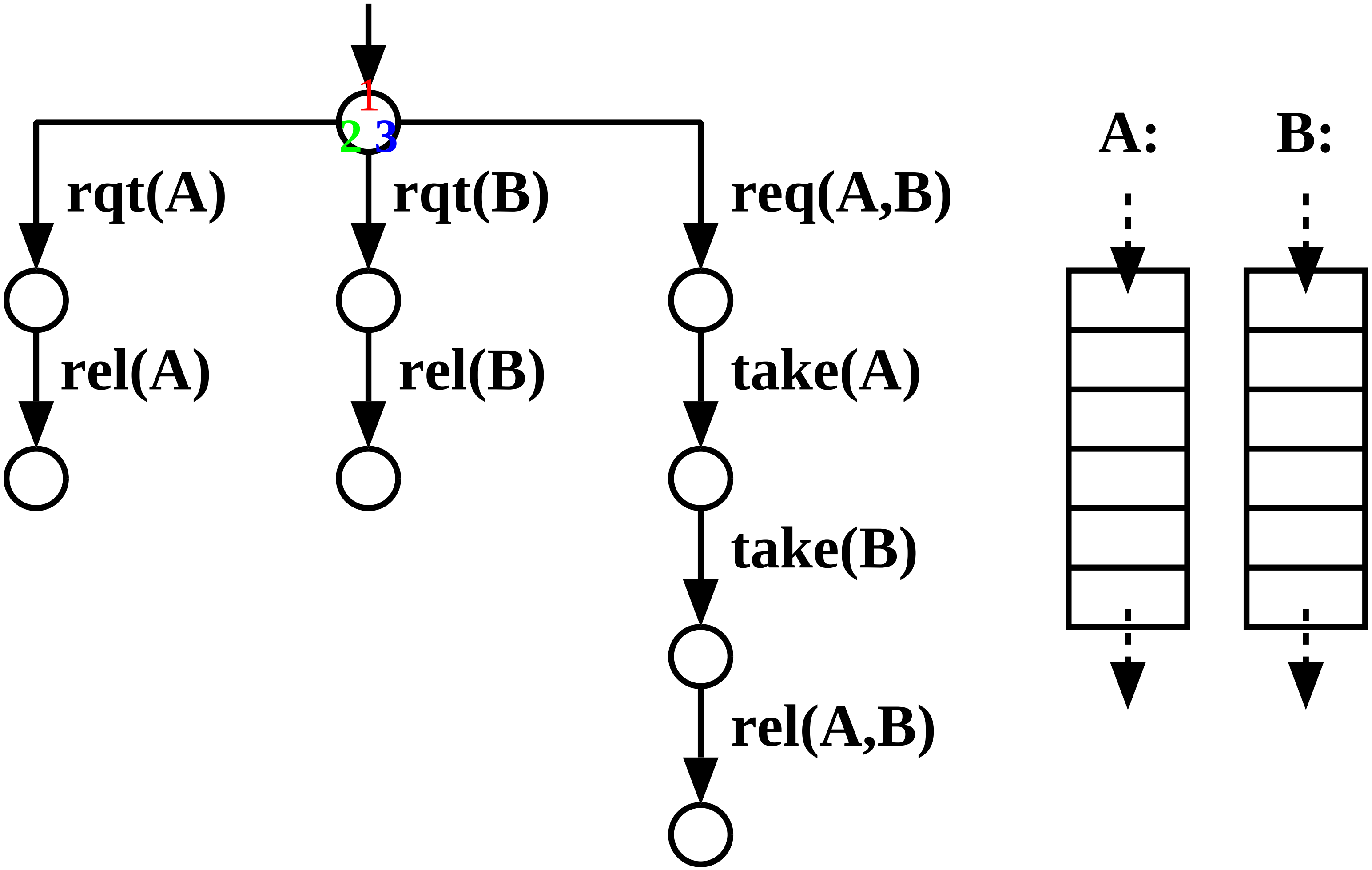
2. RTR Families of Systems (1/3)

❖ **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:**
     - τ
     - $\mathtt{req}(R')$, $\mathtt{take}(R')$, **and** $\mathtt{rel}(R')$
     - $\mathtt{rqt}(R')$
     - $\mathtt{preq}(R')$, $\mathtt{ptake}(R')$, **and** $\mathtt{prqt}(R')$
3. **a *locker* policy L (FIFO or PRIO)**

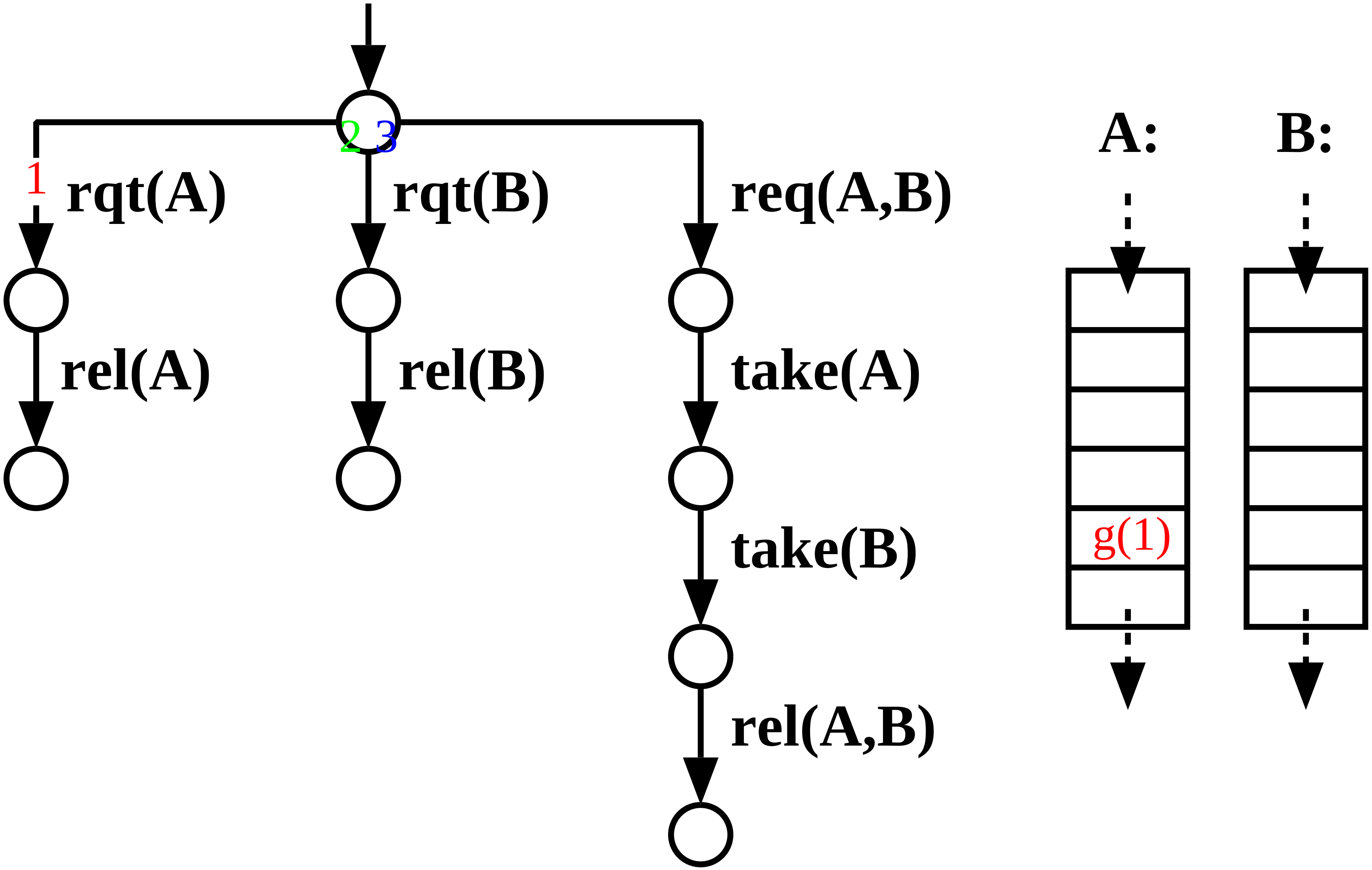
2. RTR Families of Systems (2/3)

❖ The **FIFO** locker policy:



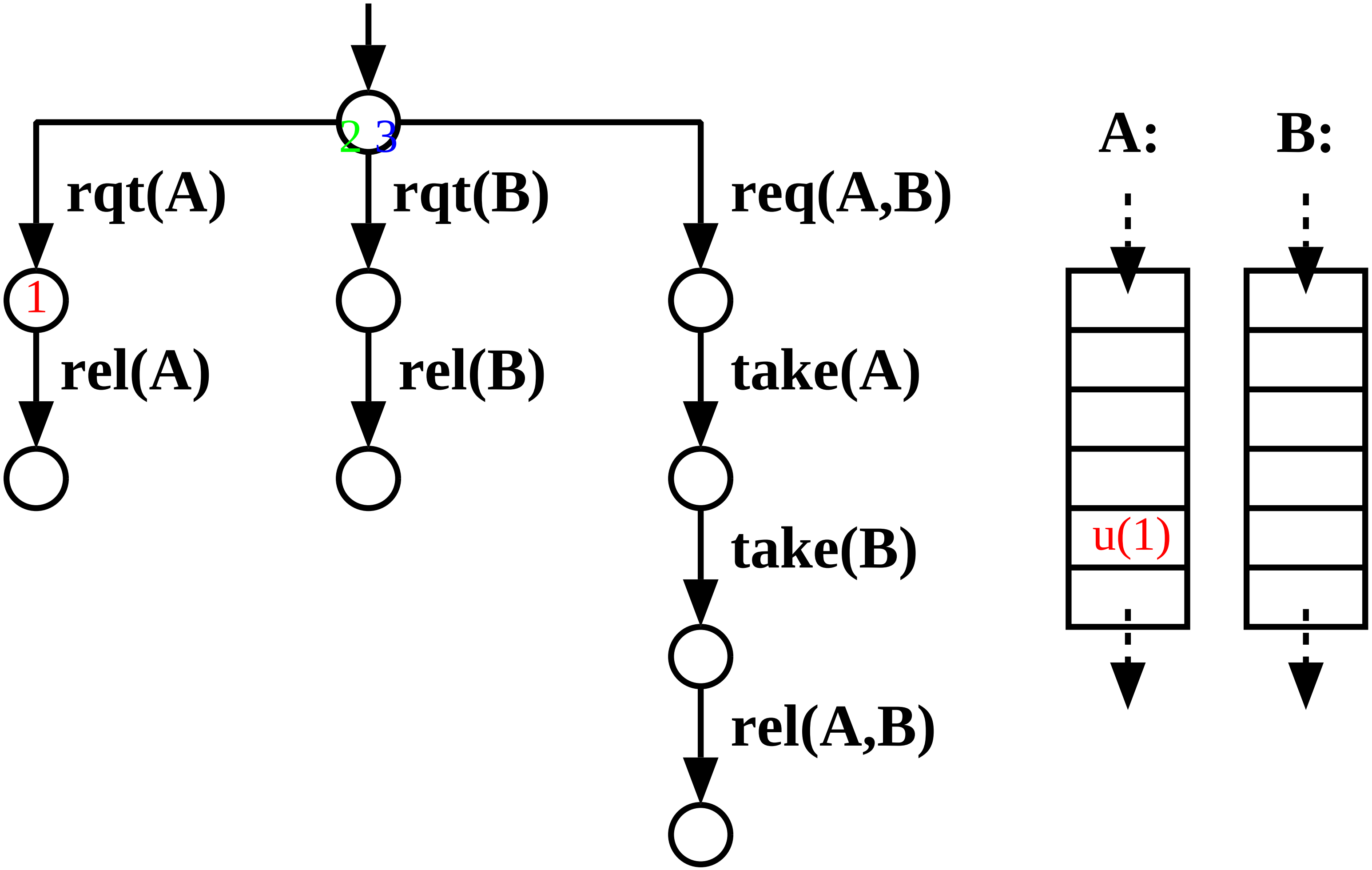
2. RTR Families of Systems (2/3)

❖ The **FIFO** locker policy:



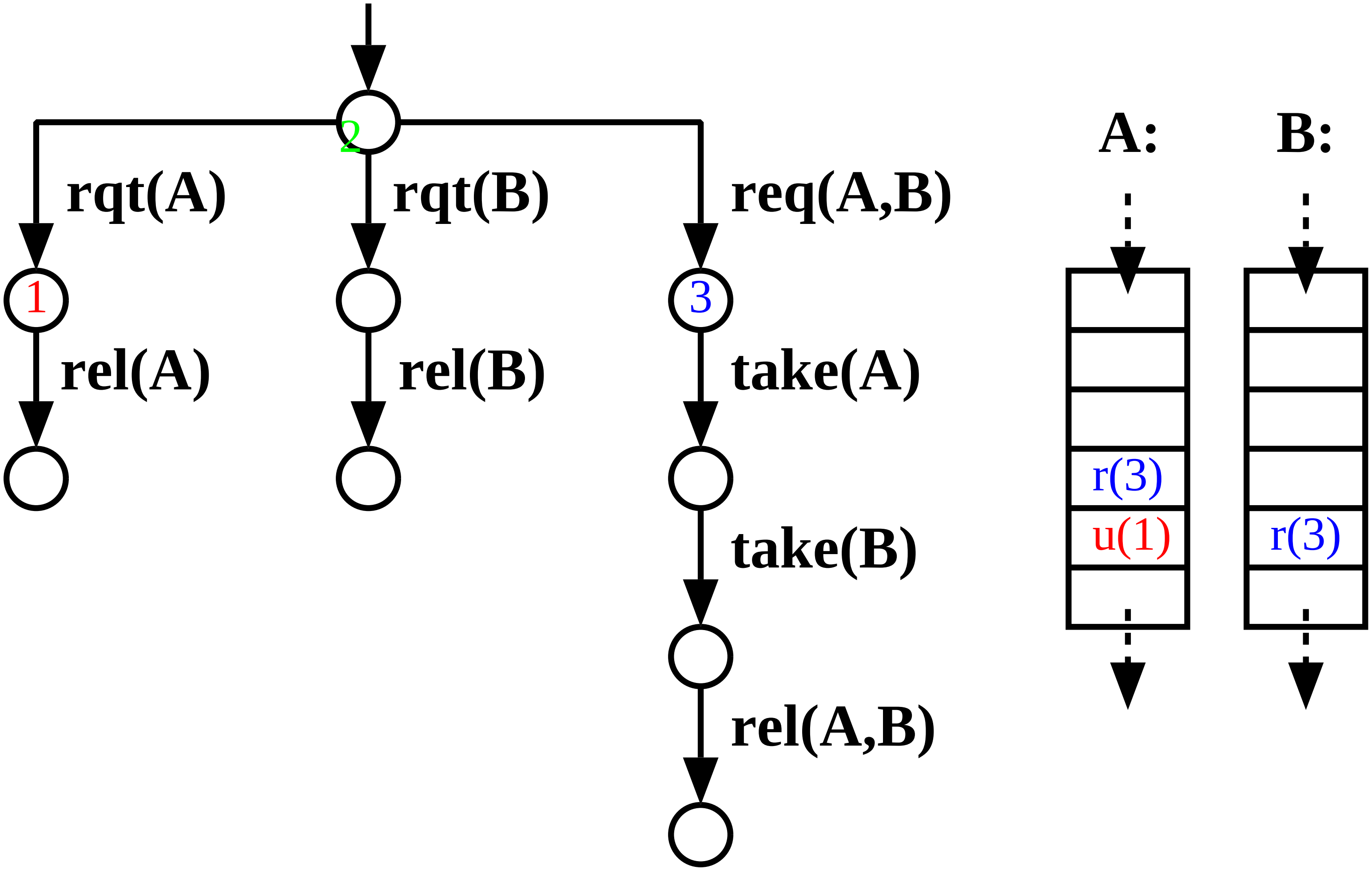
2. RTR Families of Systems (2/3)

- **The FIFO locker policy:**



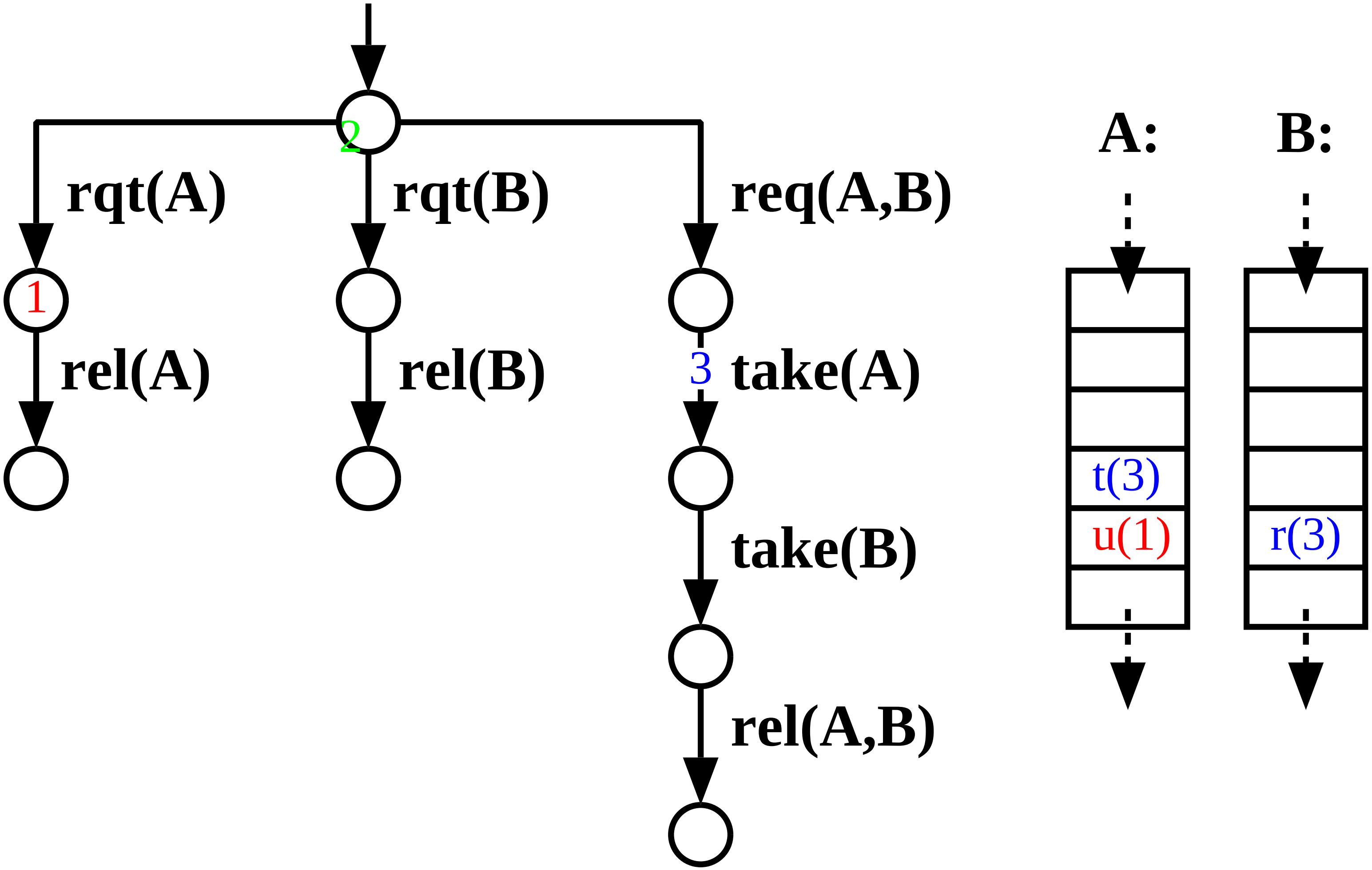
2. RTR Families of Systems (2/3)

❖ The **FIFO** locker policy:



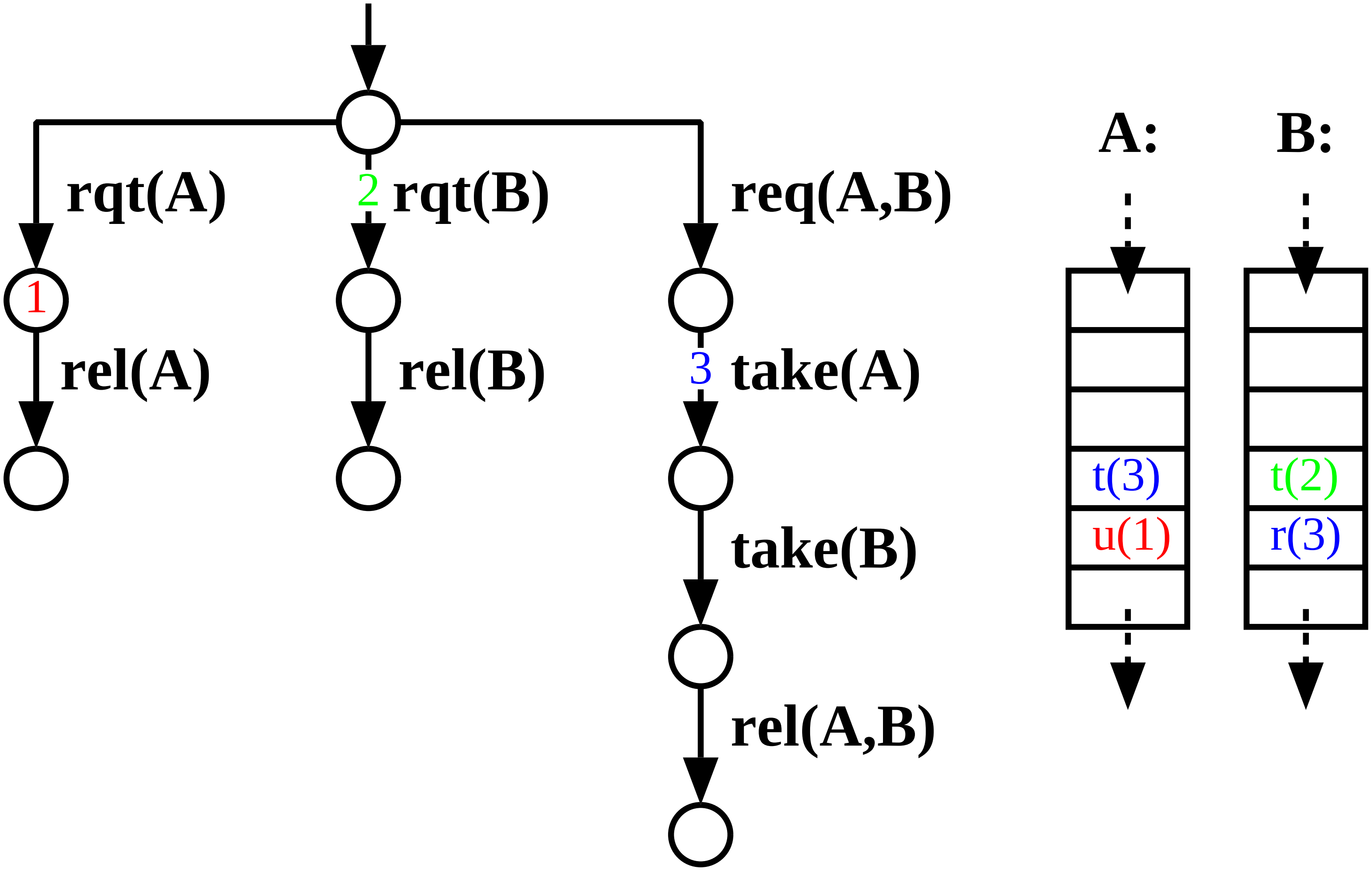
2. RTR Families of Systems (2/3)

❖ The **FIFO** locker policy:



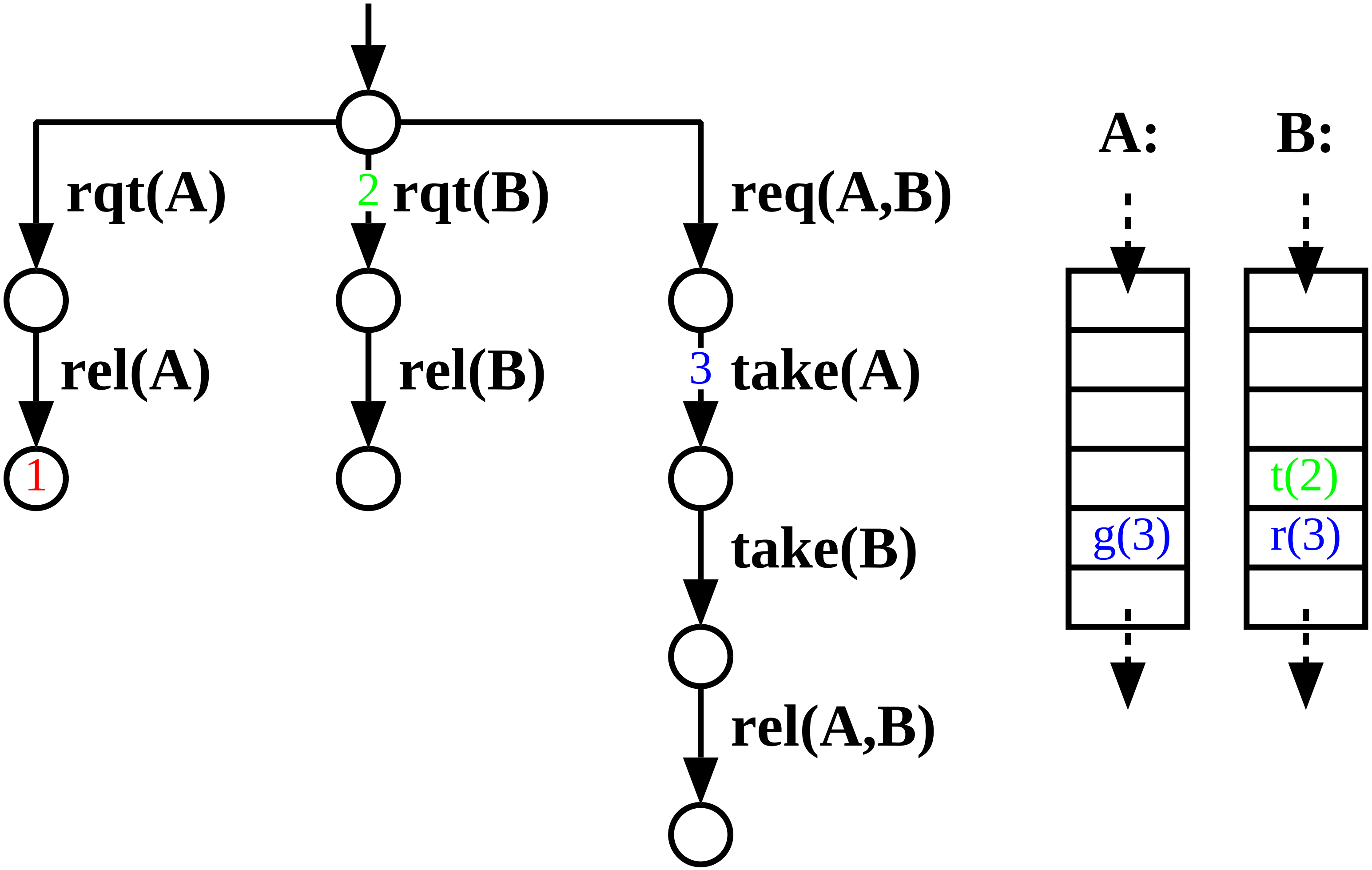
2. RTR Families of Systems (2/3)

❖ The **FIFO** locker policy:



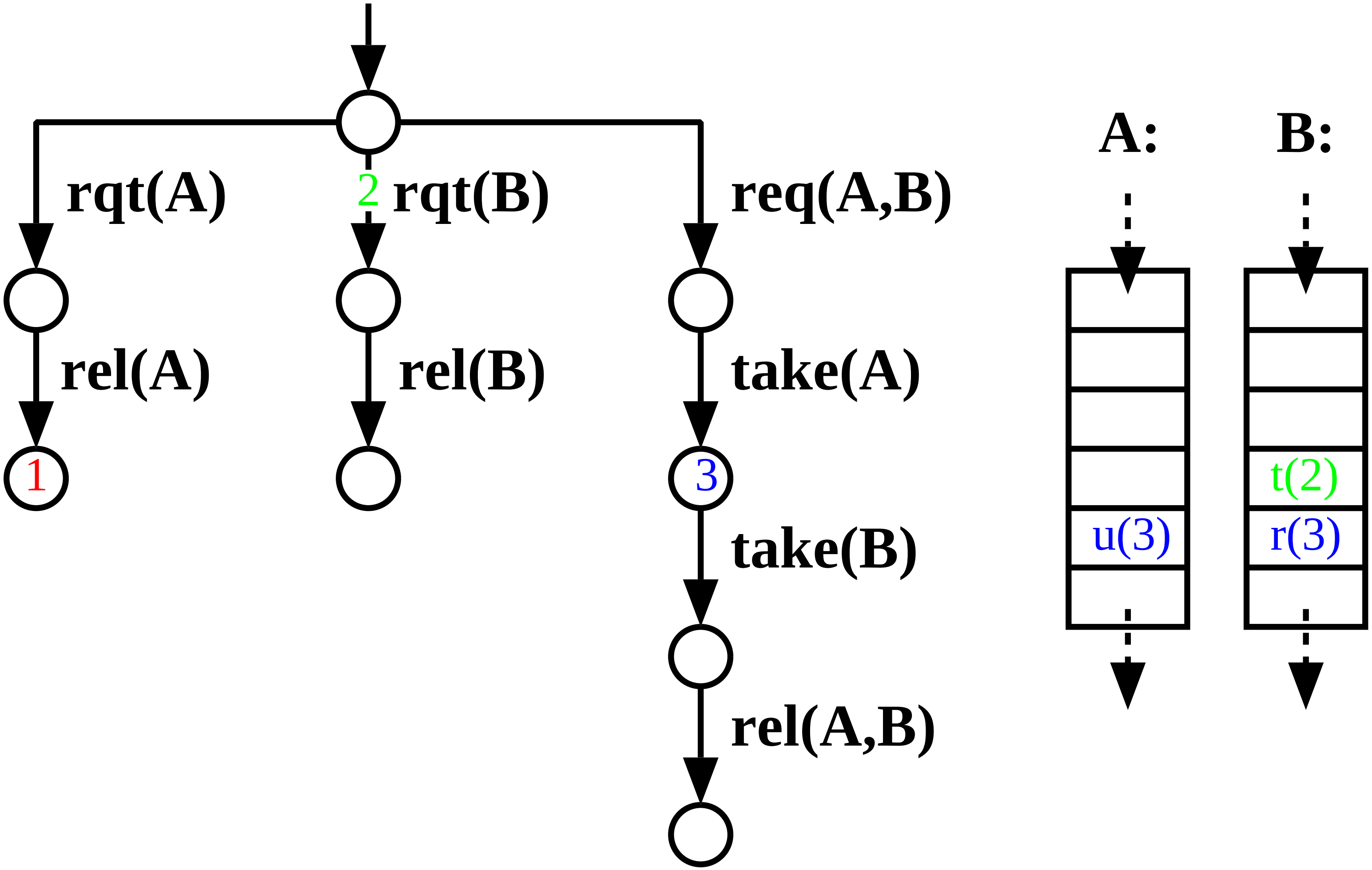
2. RTR Families of Systems (2/3)

❖ The **FIFO** locker policy:



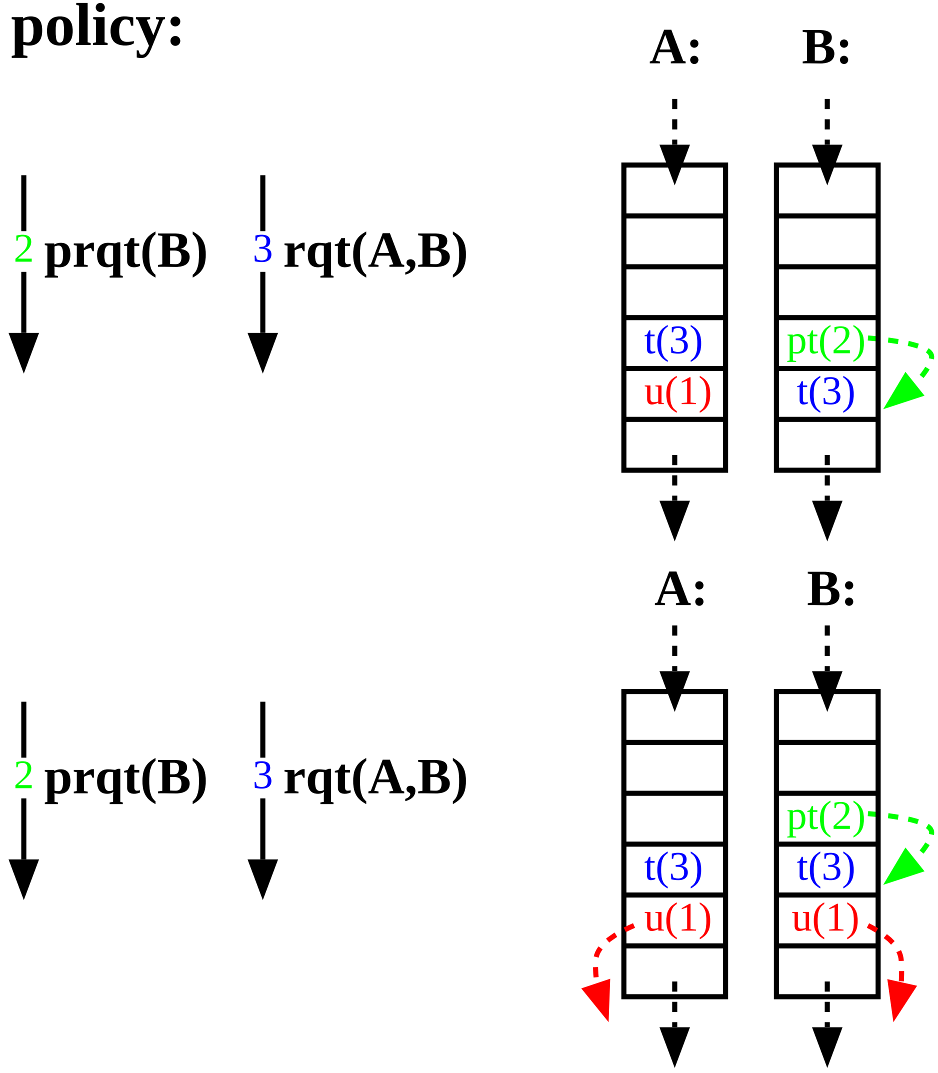
2. RTR Families of Systems (2/3)

- The **FIFO** locker policy:



2. RTR Families of Systems (3/3)

❖ The **PRIO** locker policy:



3. Properties to Checked

- We can build on the notion of ICTL*.

3. Properties to Checked

- **We can build on the notion of ICTL*.**

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

3. Properties to Checked

❖ **We can build on the notion of ICTL*.**

❖ **Global process quantification – valid along paths:**

- **mutual exclusion:** $\forall_{p_1 \neq p_2}\ AG\ \neg(.p_1 = q_{cs} \wedge .p_2 = q_{cs})$
- **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 Checked

- **We can build on the notion of ICTL*.**

- **Global process quantification – valid along paths:**
  - **mutual exclusion:** $\forall_{p_1 \neq p_2}\ AG\ \neg(.p_1 = q_{cs} \wedge .p_2 = q_{cs})$
  - **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:**

$$\forall S \in \mathcal{F} : S \models \Phi \qquad \textbf{or} \qquad \exists S \in \mathcal{F} : S \models \Phi$$

4. Verification of Finite Behaviour

- **We consider properties of the form**

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

where:

1. ι **is a conjunction of** $p_i \neq p_j$ **(for** $i \neq j$**)**
2. $\varphi(p_1,\ldots,p_k)$ **is an LTL$\backslash$X formula over atoms of the kind** $.p_i = q$ **and/or** $.p_i = .p_j$

4. Verification of Finite Behaviour

- **We consider properties of the form**

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

where:

1. ι **is a conjunction of** $p_i \neq p_j$ **(for** $i \neq j$**)**
2. $\varphi(p_1,\ldots,p_k)$ **is an LTL\X formula over atoms of the kind** $.p_i = q$ **and/or** $.p_i = .p_j$

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

$$\forall_{p_1 \neq p_2}\ A_{fin}\ G\ \neg(.p_1 = q_{cs} \wedge .p_2 = q_{cs})$$

5. Finite Behaviour of RTR$_{FIFO}$ (1/3)

❖ **In order to verify $\forall S \in \mathcal{F} : S \models \Phi^k_{fin}$ within RTR$_{FIFO}$, it is enough to consider systems with up to k processes.**

❖ **In other words, the following holds:**

$$\forall S \in \mathcal{F} : S \models \Phi^k_{fin} \quad \Leftrightarrow \quad \bigwedge_{S \in \{S_i \in \mathcal{F} | 1 \leq i \leq k\}} S \models \Phi^k_{fin}$$

5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

❖ **proof sketch:**

- $\forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi$

5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

- **proof sketch:**
  - $\forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi$

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

5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

- **proof sketch:**
  - $\forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi$

($\Rightarrow$) **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:**

5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

- **proof sketch:**
  - $\forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi$

  ($\Rightarrow$) **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 `req`, `take` makes the situation for other processes "easier".**
    - **As we always start with empty queues, `rel` just neutralizes `req`, `take`.**

5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

- **proof sketch:**
  - $\forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi$

    ($\Rightarrow$) **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 `req`, `take` makes the situation for other processes "easier".**
    - **As we always start with empty queues, `rel` just neutralizes `req`, `take`.**
    - **We remove actions of invisible processes and LTL$\setminus$X is stuttering insensitive.**

5. Finite Behaviour of RTR$_{FIFO}$ (2/3)

❖ **proof sketch:**

- $\forall l \geq k : S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\, \varphi$

  ($\Rightarrow$) **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 `req`, `take` makes the situation for other processes "easier".**
  - **As we always start with empty queues, `rel` just neutralizes `req`, `take`.**
  - **We remove actions of invisible processes and LTL$\setminus$X is stuttering insensitive.**

- **Identity of processes and transformations of the formulae.**

5. Finite Behaviour of RTR$_{FIFO}$ (3/3)

- **If the control of processes of a given family does not contain a loop with $\#req(r) > \#take(r)$, $r \in R$, 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 `rqt` is used is covered.**

5. Finite Behaviour of RTR$_{FIFO}$ (3/3)

❖ **If the control of processes of a given family does not contain a loop with $\#req(r) > \#take(r)$, $r \in R$, 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.**

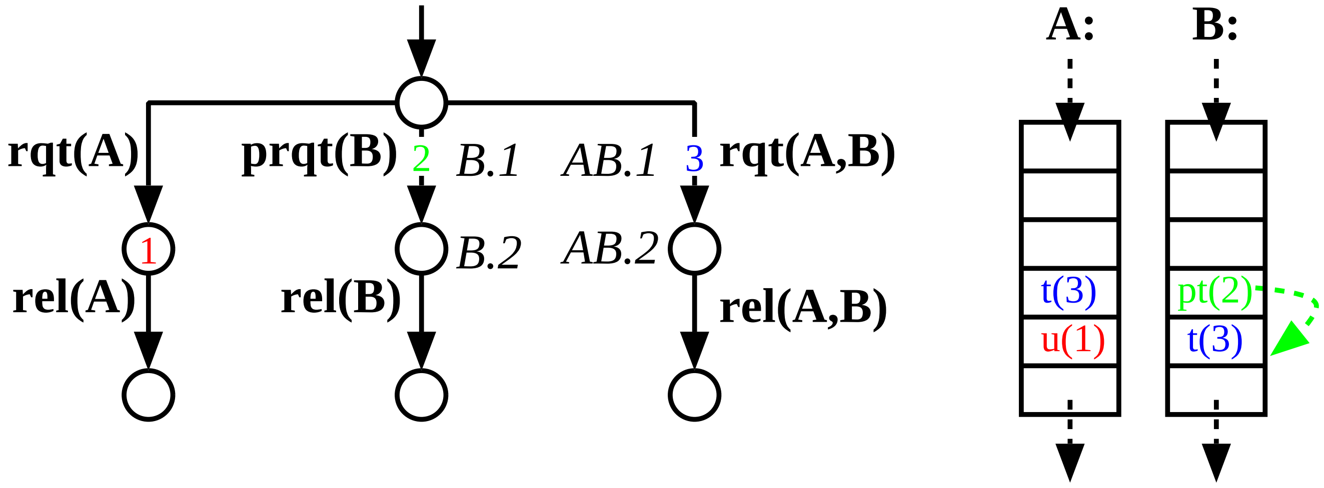
❖ **The case of the (RT)R$_{FIFO}$ families where only `rqt` is used is covered.**

❖ **The described result *cannot be used* within RTR$_{PRIO}$ nor (RT)R$_{PRIO}$.**

6. Finite Behaviour of (RT)R$_{PRIO}$ (1/3)

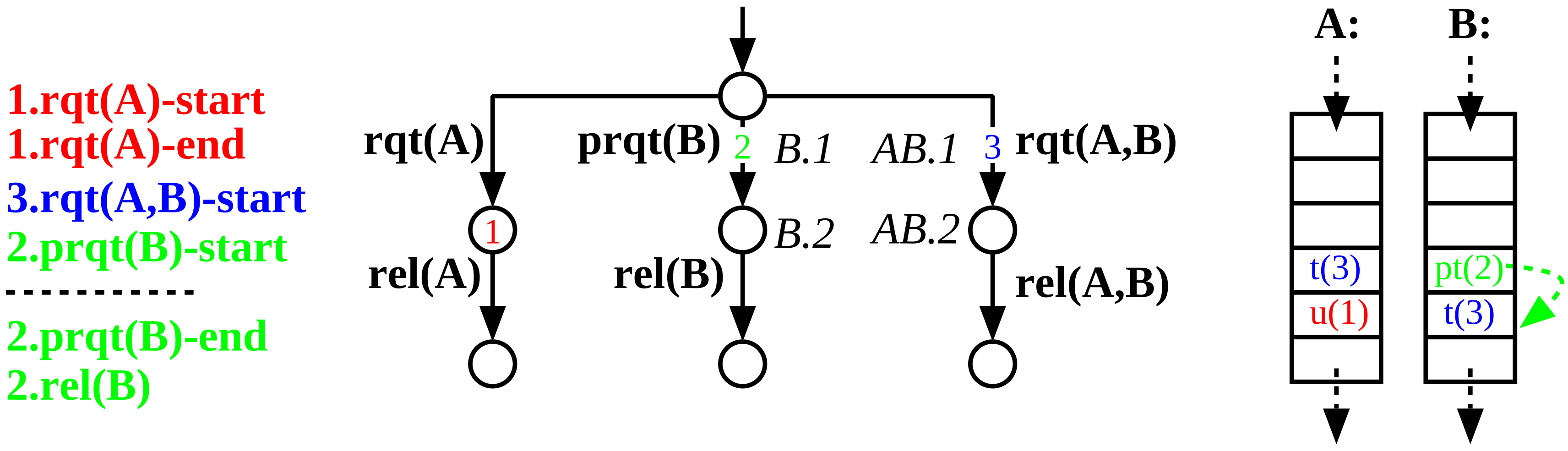
❖ **In (RT)R$_{PRIO}$, when we remove actions of some processes from a behaviour, we need not obtain a behaviour:**

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



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:

$$\exists S \in \mathcal{F} : S \models \exists_{p_1 \neq p_2} E_{fin}$$
$$((.p_2 \neq B_1)\ U\ (.p_1 = AB_1))\ \wedge\ ((.p_1 \neq AB_2)\ U\ (.p_2 = B_2))$$

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

❖ **proof sketch:**

- $\forall l \geq k :$
  $S_k \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi$

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

- **proof sketch:**
  - $\forall l \geq k :$
    $S_k \models \exists_{p_1,...,p_k|\neq} E_{fin}\, F\, \pi \Leftrightarrow S_l \models \exists_{p_1,...,p_k|\neq} E_{fin}\, F\, \pi$

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

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

❖ **proof sketch:**

- $\forall l \geq k :$
  $S_k \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi$

  ($\Rightarrow$) 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)

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

❖ **proof sketch:**

- $\forall l \geq k :$
  $S_k \models \exists_{p_1,\dots,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,\dots,p_k|\neq} E_{fin}\ F\ \pi$

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

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

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

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

- **proof sketch:**
  - $\forall l \geq k :$
    $S_k \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi$

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

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

~~1.rqt(A)-start~~
~~1.rqt(A)-end~~
3.rqt(A,B)-start
2.prqt(B)-start
- - - - - - - - - - -
2.prqt(B)-end
2.rel(B)

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

- **proof sketch:**
  - $\forall l \geq k$:
    $S_k \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi$

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

(⇐)
- **We take a witness from** S_l**.**
- **We remove actions of invisible processes.**
- **We postpone** $\texttt{rqt}(R') - start$ **to be just after all the "overtaking"** $\texttt{prqt}(R'') - start$**.**

~~1.rqt(A)-start~~
~~1.rqt(A)-end~~
3.rqt(A,B)-start
2.prqt(B)-start
- - - - - - - - - - -
2.prqt(B)-end
2.rel(B)

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

- **proof sketch:**
  - $\forall l \geq k$:
    $S_k \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,...,p_k|\neq} E_{fin}\ F\ \pi$

($\Rightarrow$) **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 $\texttt{rqt}(R') - start$ to be just after all the "overtaking" $\texttt{prqt}(R'') - start$.**

~~1.rqt(A)-start~~
~~1.rqt(A)-end~~
3.rqt(A,B)-start
2.prqt(B)-start
- - - - - - - - - - -
2.prqt(B)-end
2.rel(B)

6. Finite Behaviour of (RT)R$_{PRIO}$ (2/3)

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

❖ **proof sketch:**

- $\forall l \geq k :$
  $S_k \models \exists_{p_1,\ldots,p_k|\neq}\ E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq}\ E_{fin}\ F\ \pi$

($\Rightarrow$) **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 $\texttt{rqt}(R') - start$ to be just after all the "overtaking" $\texttt{prqt}(R'') - start$.**
- **We obtain a behaviour in S_k.**
- **The visible final state is not changed.**

~~1.rqt(A)-start~~
~~1.rqt(A)-end~~
3.rqt(A,B)-start
2.prqt(B)-start
- - - - - - - - - -
2.prqt(B)-end
2.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:**

- $\forall l \geq k :$
  $S_k \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\ F\ \pi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{fin}\ F\ \pi$

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

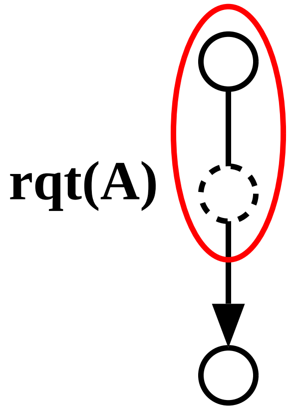
  ($\Leftarrow$)
  - **We take a witness from S_l.**
  - **We remove actions of invisible processes.**
  - **We postpone** rqt$(R') - start$ **to be just after all the "overtaking"** prqt$(R'') - start$**.**
  - **We obtain a behaviour in S_k.**
  - **The visible final state is not changed.**

- **Transformations of the formulae.**

~~1.rqt(A)-start~~
~~1.rqt(A)-end~~
3.rqt(A,B)-start
2.prqt(B)-start
- - - - - - - - - - -
2.prqt(B)-end
2.rel(B)

6. Finite Behaviour of (RT)R$_{PRIO}$ (3/3)

❖ **The result holds also for general LTL$\backslash$X formulae not distinguishing the control pre- and post-conditions of $\texttt{rqt}(R') - start$.**



7. Verification of Fair Behaviour

- **We consider properties of the form**

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

where:

1. ι **and** φ **are as in** Φ^k_{fin}
2. **wf represents weak (process) fairness**

7. Verification of Fair Behaviour

- **We consider properties of the form**

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

where:

1. ι **and** φ **are as in** Φ^k_{fin}
2. **wf represents weak (process) fairness**

- **Weak fairness coincides with strong fairness in our case.**

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.**

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:**
  $\forall l \geq m + k : S_{m+k} \models \exists_{p_1,\ldots,p_k|\neq} E_{wf}\ \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{wf}\ \varphi$
  ($\Leftarrow$) 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:**
  $\forall l \geq m + k : S_{m+k} \models \exists_{p_1,\ldots,p_k|\neq} E_{wf}\, \varphi \Leftrightarrow S_l \models \exists_{p_1,\ldots,p_k|\neq} E_{wf}\, \varphi$

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

  ($\Rightarrow$) 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$_{FIFO}$ (2/3)

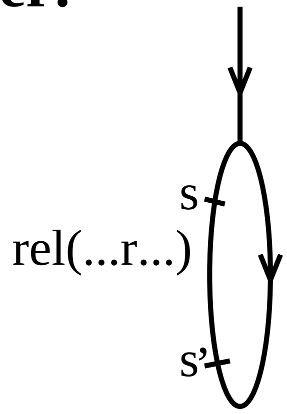
- **Adding new processes when all original processes run forever:**

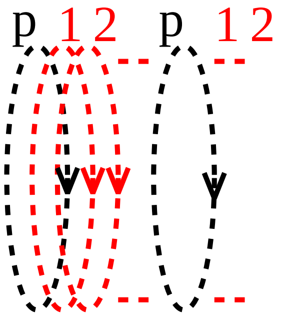
8. Fair Behaviour of (RT)R$_{FIFO}$ (2/3)

❖ **Adding new processes when all original processes run forever:**

At least one resource is always eventually released:

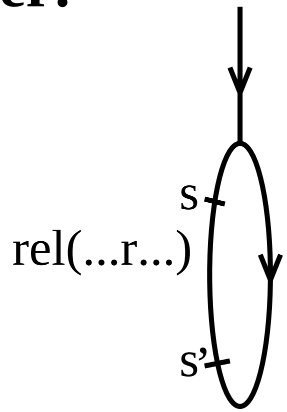
- **There is a state s where at least 1 resource is unused.**
- **At most $m - 1$ processes may use some resources in s.**
- **At least $k + 1$ processes do not use any resource in s.**
- **There is an invisible process p not using anything in s.**
- **The behaviour of p can be mimicked.**





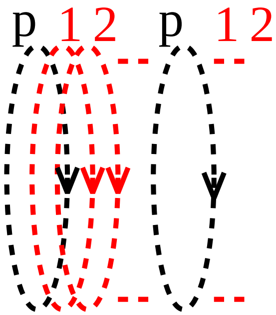
8. Fair Behaviour of (RT)R$_{FIFO}$ (2/3)

❖ **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:**

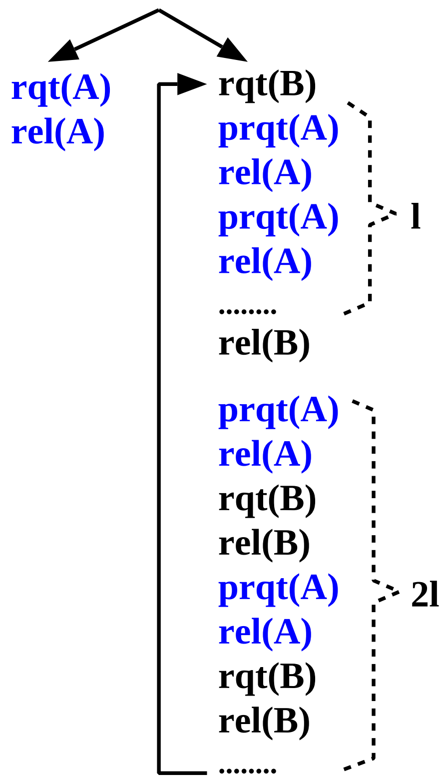
1. **At least 1 process p out of the b processes does not use any resource (it is just asking for some).**
   - p **can be easily mimicked.**
2. **All of the b processes use some resources.**
   - $b \leq m$
   - **At least b resources cannot be used by the looping processes.**
   - **At most $m - b = m'$ 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$_{PRIO}$ (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.**

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.**

9. Fair Behaviour of (RT)R$_{PRIO}$ (3/5)

❖ **The proof idea continued:**

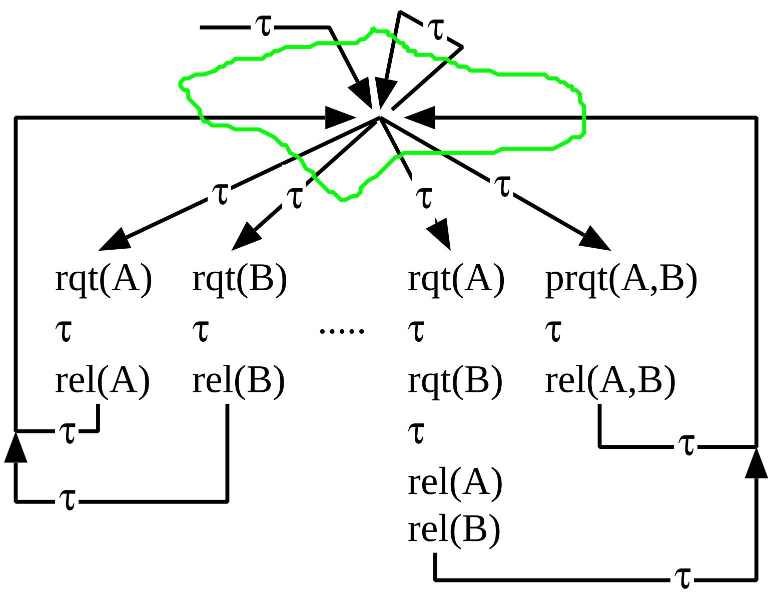
- **We construct a system of linear equations whose solutions describe loops over states encoded as above and guaranteeing that the visible process remains blocked.**
- **All constants in the equations may be bounded, a solution exists (it is the given witness), and thus the theory of Linear Programming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

❖ **The 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$_{PRIO}$.**

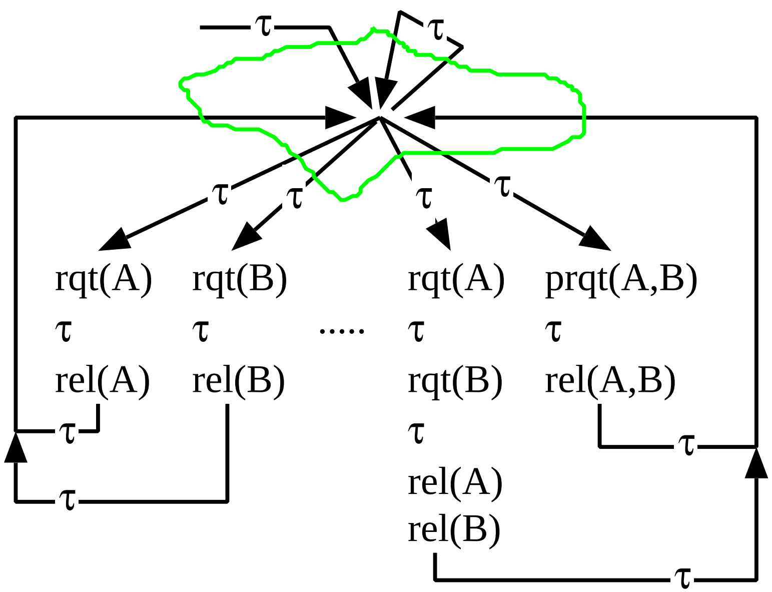
9. Fair Behaviour of (RT)R$_{PRIO}$ (4/5)

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



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.**



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

9. Fair Behaviour of (RT)R$_{PRIO}$ (5/5)

❖ **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$_{FIFO}$ or (RT)R$_{PRIO}$ family $\mathcal{F}$, it suffices to examine the system** $S_{max(m,2)} \in \mathcal{F}$ **where $m = |R|$.**

- **The proof is simple for RTR$_{FIFO}$ 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$_{FIFO}$ or (RT)R$_{PRIO}$ family $\mathcal{F}$, it suffices to examine the system $S_{max(m,2)} \in \mathcal{F}$ where $m = |R|$.

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

- In (RT)R$_{PRIO}$, 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 (1/7)

- **In 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 RTR$_{FIFO}$: the EF fragment of ICTL* with only global as well as only local process quantification**
  - ***even for (RT)R$_{FIFO}$*: the LTL$\setminus$X fragment of ICTL* based on atomic formulae of the kind $\forall_p \, .p = q$**

11. Some Undecidability Results (1/7)

❖ **In 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 RTR$_{FIFO}$: the EF fragment of ICTL* with only global as well as only local process quantification**
- ***even for (RT)R$_{FIFO}$*: the LTL$\backslash$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.**

11. Some Undecidability Results (2/7)

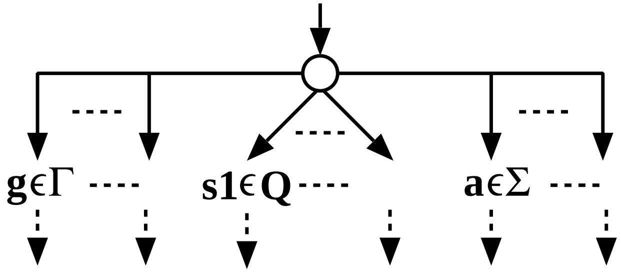
❖ **proof idea:**

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

11. Some Undecidability Results (2/7)

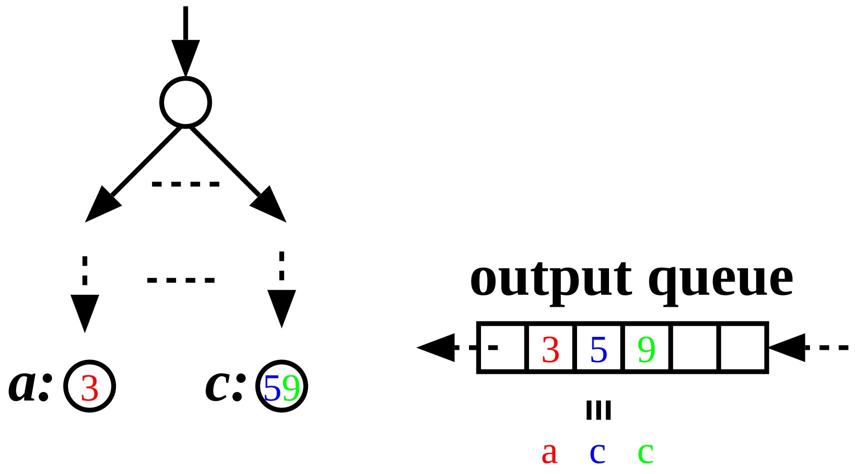
- **proof idea:**
  - We show how to simulate **PDAs** in a way that can easily be generalized to using two stacks.

  - 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).**



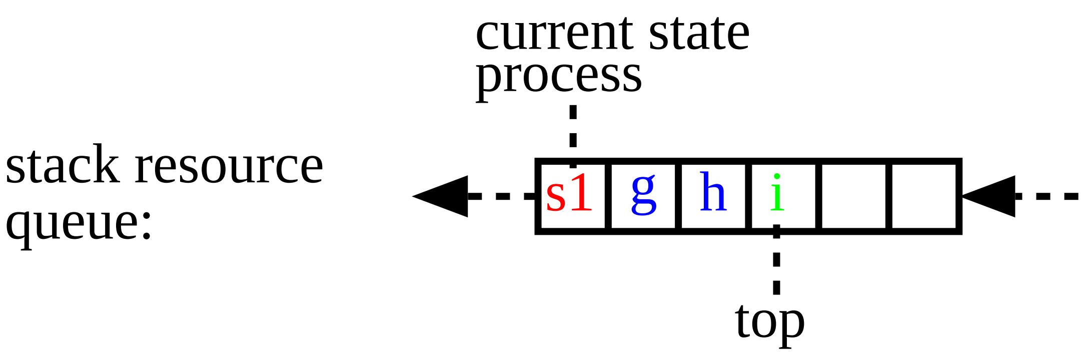
11. Some Undecidability Results (4/7)

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

11. Some Undecidability Results (4/7)

❖ **proof idea (continued):**

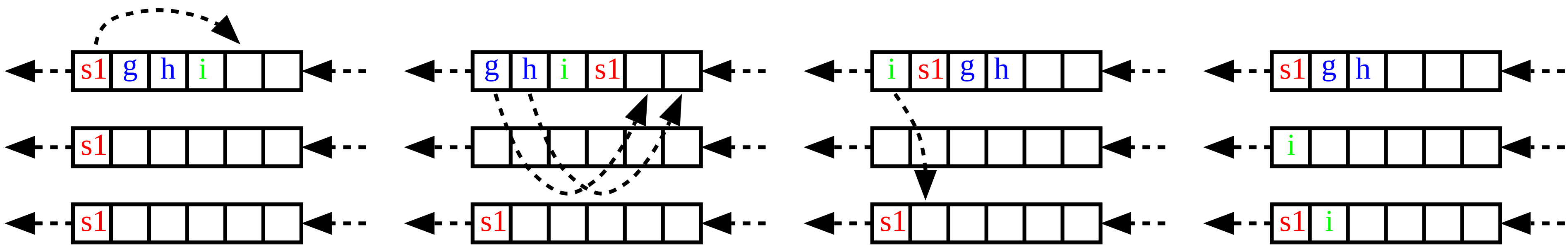
- **The simulation is controlled by state processes; if the current-state process p deadlocks, the whole system deadlocks.**
- **The stack is simulated by a resource that is normally owned by p and stack-symbol processes wait in its queue; the top of the stack corresponds to the tail of the queue:**



11. Some Undecidability Results (5/7)

❖ **proof idea (continued):**

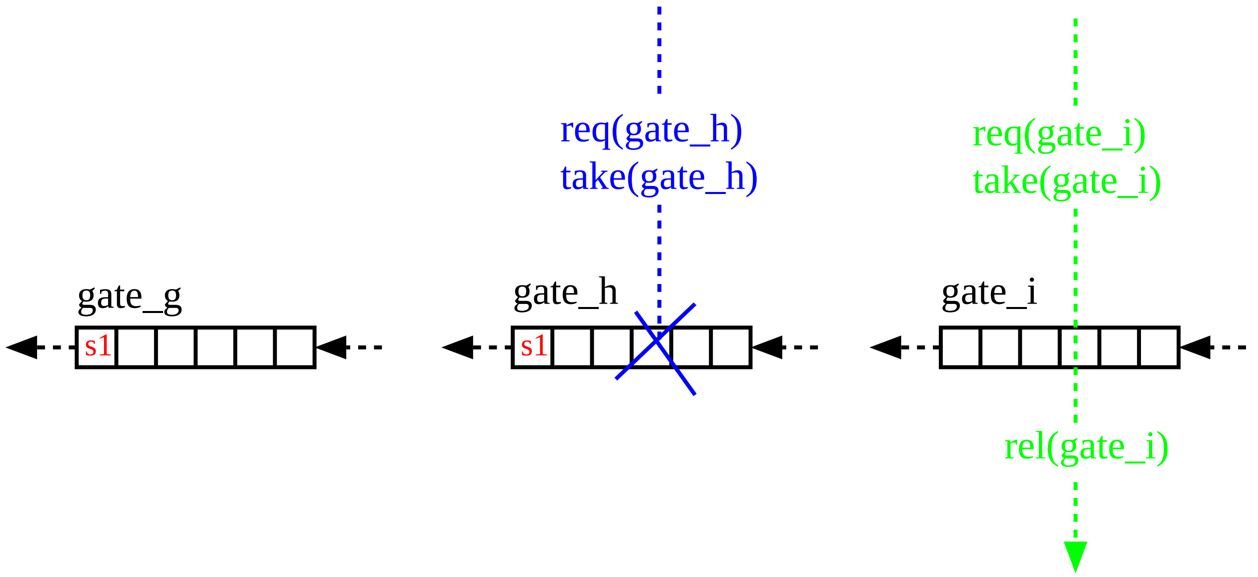
- **Reading of the top symbol can be implemented by p releasing a stack, taking it again, and ensuring that after some symbol process releases the stack and does not take it back, all further symbol processes will block before releasing the stack.**



11. Some Undecidability Results (6/7)

❖ **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.**



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.**

$$\ldots\ Q\ \ \underline{\Gamma Q}\ \ \underline{\Sigma Q}\ \ \underline{\Gamma Q\ \ \Gamma Q}\ \ Q'\underline{Q}\ \ \ldots$$

	pop	out	push	next state
$\ldots$ Q	ΓQ	ΣQ	ΓQ ΓQ	$Q'Q$ $\ldots$

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.**

  ... Q ΓQ ΣQ ΓQ ΓQ Q'Q ...

  pop out push 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