Verifying distributed systems with unbounded channels

Régis Gascon & Éric Madelaine

INRIA Sophia Antipolis

SAFA Workshop - September 23rd, 2009

VERCORS in a nutshell

- Platform for specification of distributed applications.
- Based on the semantics features of the ProActive library.

  `http://www-sop.inria.fr/oasis/ProActive/`
- Generation of intermediate finite model.
- Various tools can then operate on these models:
  static analysis, model checking, code generation...
- The aim is to integrate the platform in a development environment, used by non-specialists.

Formal verification of pNets

- Basically, pNets are made of LTSs synchronized by mean of transducer (synchronization vector).
- Verifying pNets remains to verify systems:
  - manipulating unbounded data,
  - having a parameterized topology,
  - using unbounded communication queues.
- Numerous sources of infinity
  $\Leftrightarrow$ numerous complications for formal verification.
- Current platform uses only finite-sate based model-checkers (through finite abstraction).
- We want to apply infinite state model-checking techniques.

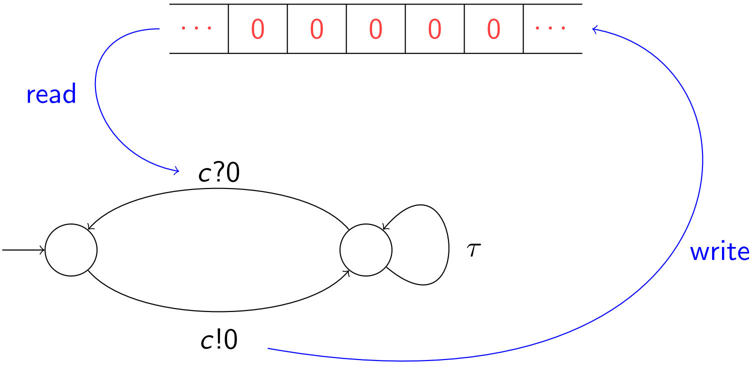
Infinite-state system verification

- Well studied theory:
  - counter systems,
  - pushdown systems,
  - parameterized systems,
  - ...
- Few implementations for unbounded queue systems:
  - LASH (Boigelot et al.),
  - TReX (Bouajjani et al.).
- Difficult to find a tool that fits our goals
  - integration to VERCORS
  - possibility of extensions

Outline

Communicating finite state machines

Basically a finite state machine augmented with a set of queues.



Communicating finite state machines

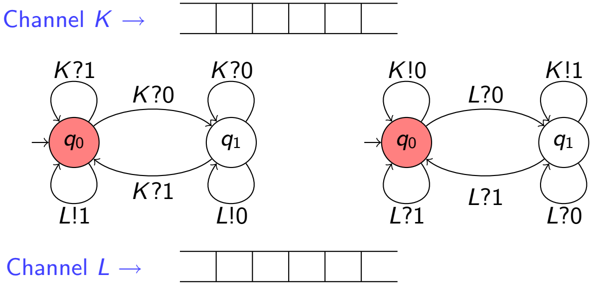
Formally, a communicating finite state machines (CFSM) is a tuple

$$\mathcal{M} = (Q, q_0, C, \Sigma, A, \delta) \quad \text{such that}$$

- $Q =$ is a finite set of states,
- $q_0 \in Q$ is the initial state,
- C is a set of communicating channels/queues,
- Σ is the alphabet of messages,
- A is a finite set of internal actions,
- $\delta \subset Q \times ((C \times \{?, !\} \times \Sigma) \cup A) \times Q$ is the transition relation.

Short Example

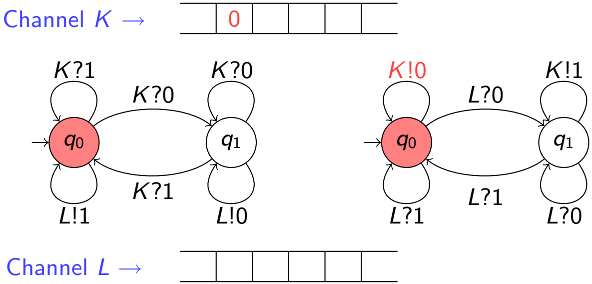
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon \rangle$

Short Example

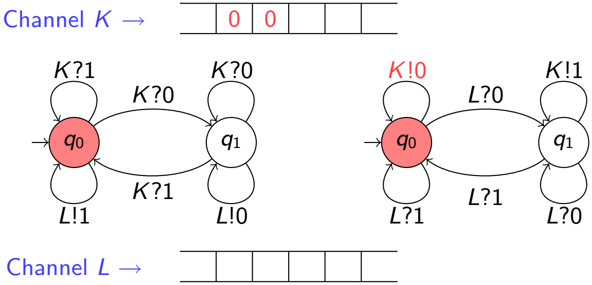
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon\rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon\rangle$

Short Example

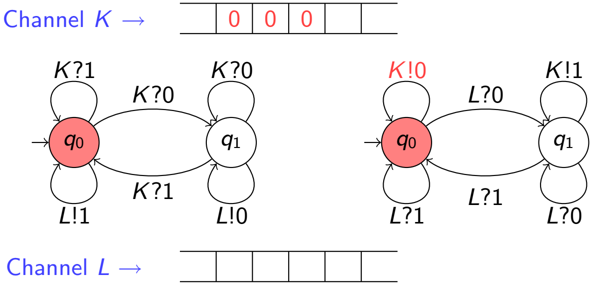
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon \rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon \rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0}$

Short Example

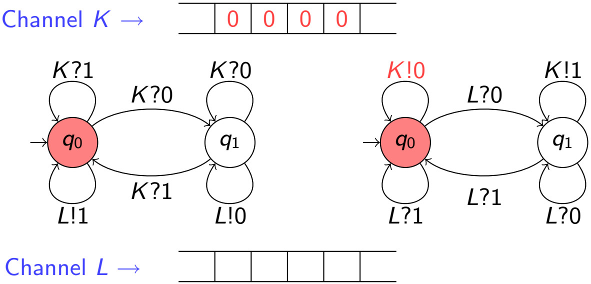
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon \rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon \rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0}$

Short Example

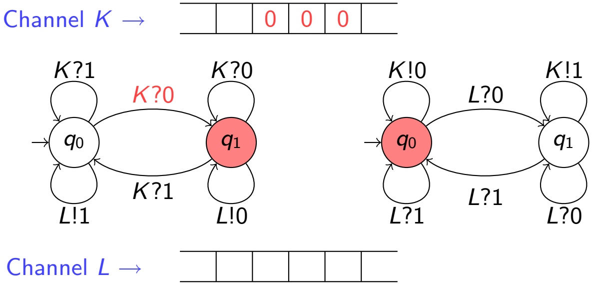
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon \rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon \rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0} \langle q_0, q_0, 0000, \varepsilon \rangle$

Short Example

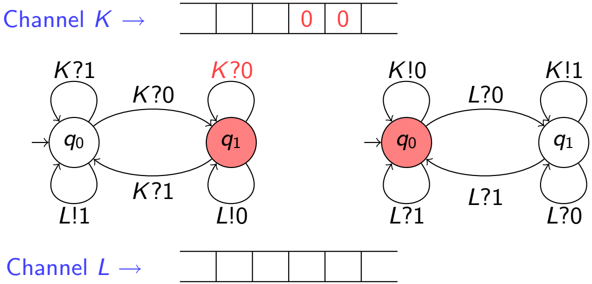
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon\rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon\rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0} \langle q_0, q_0, 0000, \varepsilon\rangle \xrightarrow{K?0} \langle q_1, q_0, 000, \varepsilon\rangle$

Short Example

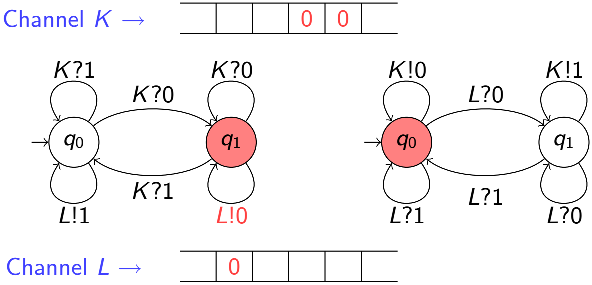
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon \rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon \rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0} \langle q_0, q_0, 0000, \varepsilon \rangle \xrightarrow{K?0}$
$\langle q_1, q_0, 000, \varepsilon \rangle \xrightarrow{K?0} \langle q_1, q_0, 00, \varepsilon \rangle$

Short Example

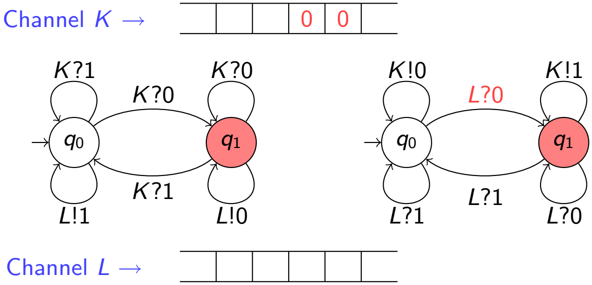
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon\rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon\rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0} \langle q_0, q_0, 0000, \varepsilon\rangle \xrightarrow{K?0}$ $\langle q_1, q_0, 000, \varepsilon\rangle \xrightarrow{K?0} \langle q_1, q_0, 00, \varepsilon\rangle \xrightarrow{L!0} \langle q_1, q_0, 00, 0\rangle$

Short Example

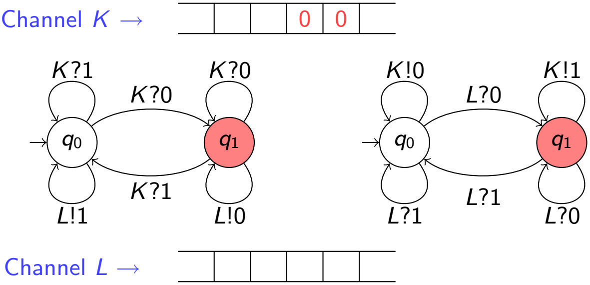
- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon\rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon\rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0} \langle q_0, q_0, 0000, \varepsilon\rangle \xrightarrow{K?0}$
  $\langle q_1, q_0, 000, \varepsilon\rangle \xrightarrow{K?0} \langle q_1, q_0, 00, \varepsilon\rangle \xrightarrow{L!0} \langle q_1, q_0, 00, 0\rangle \xrightarrow{L!0}$
  $\langle q_1, q_1, 00, \varepsilon\rangle$

Short Example

- Execution: Sequence respecting the transition relation.



- $\langle q_0, q_0, \varepsilon, \varepsilon \rangle \xrightarrow{K!0} \langle q_0, q_0, 0, \varepsilon \rangle \xrightarrow{K!0} \cdots \xrightarrow{K!0} \langle q_0, q_0, 0000, \varepsilon \rangle \xrightarrow{K?0}$ $\langle q_1, q_0, 000, \varepsilon \rangle \xrightarrow{K?0} \langle q_1, q_0, 00, \varepsilon \rangle \xrightarrow{L!0} \langle q_1, q_0, 00, 0 \rangle \xrightarrow{L!0}$ $\langle q_1, q_1, 00, \varepsilon \rangle \rightarrow \cdots$

Operational Semantics

- We consider unbounded FIFO queues.
- Consider a set of CFSM sharing a set of queues $\{K, L\}$.
- Configuration: $\langle q_1, q_2, w_K, w_L \rangle$ (for a pair of CFSM)

  Global state + Queue contents
- Operations:
  - Send (non-blocking).

    if $\langle q_1, K!a, q_1' \rangle \in \delta_1$ then

    $$\langle q_1, q_2, w_K, w_L \rangle \xrightarrow{K!a} \langle q_1', q_2, w_K \cdot a, w_L \rangle$$
  - Receive (blocking).
  - Internal Action.

Operational Semantics

- We consider unbounded FIFO queues.
- Consider a set of CFSM sharing a set of queues $\{K, L\}$.
- Configuration: $\langle q_1, q_2, w_K, w_L \rangle$ (for a pair of CFSM)

  Global state + Queue contents
- Operations:
  - Send (non-blocking).
  - Receive (blocking).

    if $\langle q_1, K?a, q_1' \rangle \in \delta_1$ then

    $$\langle q_1, q_2, a \cdot w_K, w_L \rangle \xrightarrow{K!a} \langle q_1', q_2, w_K, w_L \rangle$$
  - Internal Action.

Operational Semantics

- We consider unbounded FIFO queues.
- Consider a set of CFSM sharing a set of queues $\{K, L\}$.
- Configuration: $\langle q_1, q_2, w_K, w_L \rangle$ (for a pair of CFSM)

  Global state + Queue contents
- Operations:
  - Send (non-blocking).
  - Receive (blocking).
  - Internal Action.

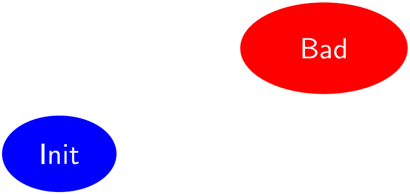
    if $\langle q_1, \tau, q_1' \rangle \in \delta_1$ with $\tau \in A$ then

$$\langle q_1, q_2, w_K, w_L \rangle \xrightarrow{\tau} \langle q_1', q_2, w_K, w_L \rangle$$

Outline

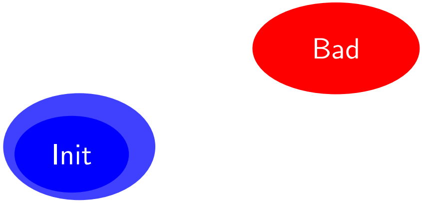
Reachability Problem

We consider the following problem:



Reachability Problem

We consider the following problem:

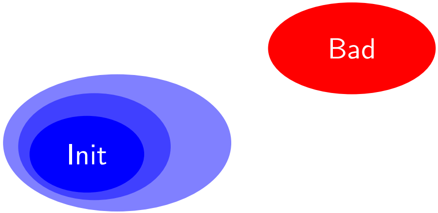


We note:

- $\mathrm{Post}(X) = \{x \mid \exists x' \in X \text{ s.t. } x \rightarrow x'\}$.

Reachability Problem

We consider the following problem:

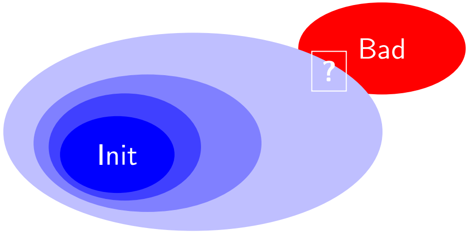


We note:

- $\mathrm{Post}(X) = \{x \mid \exists x' \in X \text{ s.t. } x \rightarrow x'\}$.
- $\mathrm{Post}^i(X) = \mathrm{Post}(\mathrm{Post}(\cdots \mathrm{Post}(X)))$.

Reachability Problem

We consider the following problem:

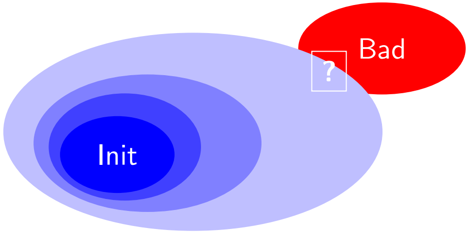


We note:

- $\mathrm{Post}(X) = \{x \mid \exists x' \in X \text{ s.t. } x \to x'\}$.
- $\mathrm{Post}^i(X) = \mathrm{Post}(\mathrm{Post}(\cdots \mathrm{Post}(X)))$.
- $\mathrm{Post}^*(X) = \bigcup_{i \geq 0} \mathrm{Post}^i(X)$.

Reachability Problem

We consider the following problem:

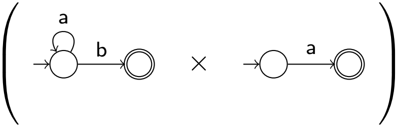


We note:

- $\mathrm{Post}(X) = \{x \mid \exists x' \in X \text{ s.t. } x \to x'\}$.
- $\mathrm{Post}^i(X) = \mathrm{Post}(\mathrm{Post}(\cdots \mathrm{Post}(X)))$.
- $\mathrm{Post}^*(X) = \bigcup_{i \geq 0} \mathrm{Post}^i(X)$. UNDECIDABLE (semi-algorithm)

Representing Sets of Configurations

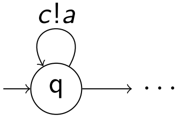
- We need to represent possibly infinite sets of configurations.
- We associate to each tuple of states of the CFSM a set of finite state automata (FUDFA) over Σ.
- The set of configurations corresponds to the (regular) language associated to each state.
- Ex: $\langle q_1, q_2 \rangle +$



represents the set of configurations $\langle q_1, q_2, a^* b, a \rangle$.

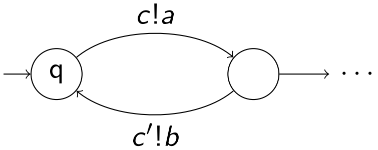
Improving convergence

- FUDFA allows to compute directly the result of infinitely iterating some cycles:



$\rightsquigarrow \quad \langle q, a^* \rangle$

- Pb: Cycles can induce non-regular sets of queue contents:



$\rightsquigarrow \quad \langle q, a^n, b^n \rangle$

- Need for characterization of accelerable loops.

Algorithm with accelerations

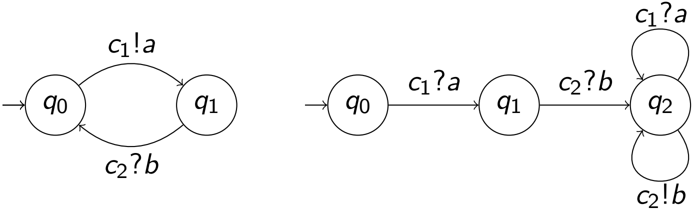
- $F[s]$ is the FUDFA associated to global state s.
- We apply a depth-first exploration method.

While $S \neq \emptyset$ **do**
  Choose and remove some $s \in S$

  Acceleration:
  For all cycle θ from s
    If θ can be accelerated **then**
      Compute the effect of θ^* on $F[s]$

  OneStep successors:
  For all possible transition $s \xrightarrow{\mathrm{op}} s'$
    Compute the effect of op on $F[s]$
    Add new reached configurations to $F[s']$.

Complete example



$\langle q_0, q_0 \rangle$		$\langle q_0, q_1 \rangle$	
$\langle q_0, q_2 \rangle$		$\langle q_1, q_0 \rangle$	
$\langle q_1, q_1 \rangle$		$\langle q_1, q_2 \rangle$	

Important issues for the implementation

1. Data structure,
2. Adaptability/Modularity (cannot use LASH has a blackbox),
3. Selection of cycles for acceleration,
   - global cycles or local cycles,
   - heuristics.
4. Exploration strategy,
5. Using the result of the computation.

Outline

Implementation

- Algorithm implemented in JAVA.
- Input: A set of CFSMs sharing a set of channels: text format or graphical editor (eclipse plugin).
- Computes successively the set of reachable states step by step + acceleration (at each iteration).
- A FUDFA is associated to each global state and the main loop of the algorithm can be executed.
- The algorithm follows strictly the method described.

Exploring the statespace

We have concentrated on useful functionalities (exploration, utilisation of the result).

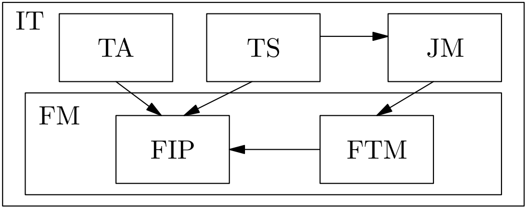
- If the computation converge $\rightsquigarrow$ OK
- Otherwise, the user can specify:
  - a set of final configurations,
  - a timeout (number of iterations),
  - a bound on the size of representations ($\neq$ bounding the size of the queues).

Performance scale

- On the other hand, there is no fine tunning of the implementation for the moment:
  - data structure quite big (naive implementation of DFAs),
  - possible improvements in data manipulation.
- In this context, we have checked the implementation w.r.t. the utilisation of the computation.
  $\rightsquigarrow$ no evaluation in terms of computation performance.
- Objective: giving a readable diagnosis of the analysis.

Example: Integrated toolkit

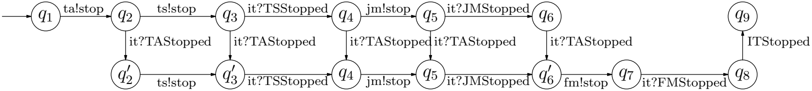
- Hierarchical component example.
- Arrows represents dependencies.



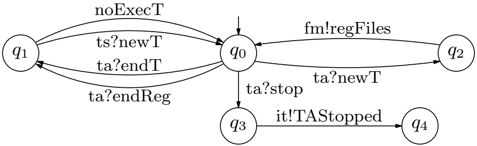
- Each box has an associated CFSM and queue.

Example: Integrated toolkit

IT (stop procedure)



TA



What does happen when the system is stopped?

Experimental scenario

When trying to compute the set configuration where IT is stopped, computation does not converge.

- 15 iterations $\leadsto$ 2460 DFAs and 19096 states
- $+$ Size Limit $\leadsto$ 78 DFAs and 275 states
- Result: $ta = (\mathrm{NewT}^* \cdot \mathrm{EndReg}^* \cdot \mathrm{EndT}^*)^*$
- Then one can check that a configuration where *ta* is not empty can be reached.

So TA can left requests unsatisfied.

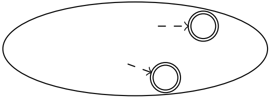
Future Work

- Improvements of exploration techniques.
- Comparisons with existing tools (LASH, TReX,...).
- Extension of the representation:
  - representation of non-regular sets of queues,
  - addition of datas (ex: queues + counters)
- Combination with other techniques (parametrized sytems)

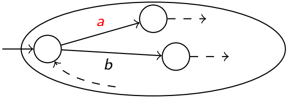
QUESTIONS ???

Basic Operations

- Add a letter (!a):



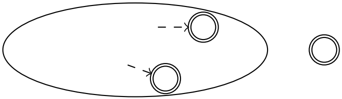
- Remove a letter (?a):



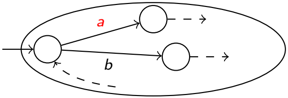
- Nothing to do with internal actions.
- Generalisation to sequences: just iterate!

Basic Operations

- Add a letter (!a):



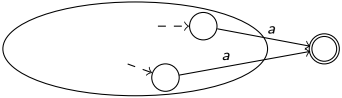
- Remove a letter (?a):



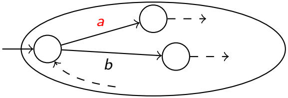
- Nothing to do with internal actions.
- Generalisation to sequences: just iterate!

Basic Operations

- Add a letter ($!a$):



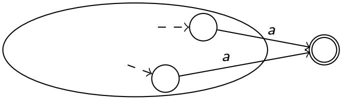
- Remove a letter ($?a$):



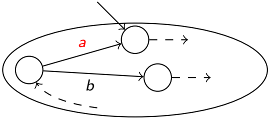
- Nothing to do with internal actions.
- Generalisation to sequences: just iterate!

Basic Operations

- Add a letter ($!a$):



- Remove a letter ($?a$):



- Nothing to do with internal actions.
- Generalisation to sequences: just iterate!

Cycle selection and acceleration

- All the material needed can be adapted from Boigelot's thesis.
  - exact characterisation of accelerable cycles,
  - computation of the acceleration.
- For every sequence of operations σ,
  - $\sharp_!(\sigma)$ is the number of send operations,
  - $\sharp_?(\sigma)$ is the number of receive operations.
- A sequence involving only one queue is counting iff
  - $|\Sigma| = 1$ and $\sharp_!(\theta) > \sharp_?(\theta)$,
  - $|\Sigma| > 1$ and $\sharp_!(\theta) > 0$.
- Given a system with queues $\{c_1, \ldots, c_n\}$ and a cycle θ, $\theta_{|i}$ is the sub-sequence of transitions manipulating c_i.

Fundamental Results for acceleration

- For systems with only one queue, the result is the following.

Theorem (Single-queue systems)

For every set of configurations X and cycle θ, the set $\mathrm{Post}_\theta^*(X)$ is FUDFA representable.

- The result for systems with several queues is more restrictive.

Theorem (Multi-queue systems)

For every set of configurations X and cycle θ, the set $\mathrm{Post}_\theta^*(X)$ is FUDFA representable iff there do not exist i and j s.t $\theta_{|i}$ and $\theta_{|j}$ are counting.