

# Synthèse de systèmes distribués ouverts

Nathalie Sznajder

LSV, ENS Cachan & CNRS & INRIA Saclay IdF

12 Novembre 2009

# Need for formal methods



# Need for formal methods



Need for **formal tools** to check behaviors of critical programs:

- ▶ Test
- ▶ Computer-aided proofs
- ▶ Model-checking

# Principles of Model-checking

a system

a specification

# Principles of Model-checking

Does a system satisfy a specification ?

# Principles of Model-checking

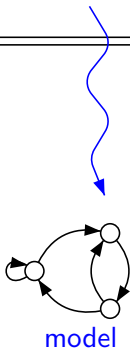
Does a system satisfy a specification ?

---

---

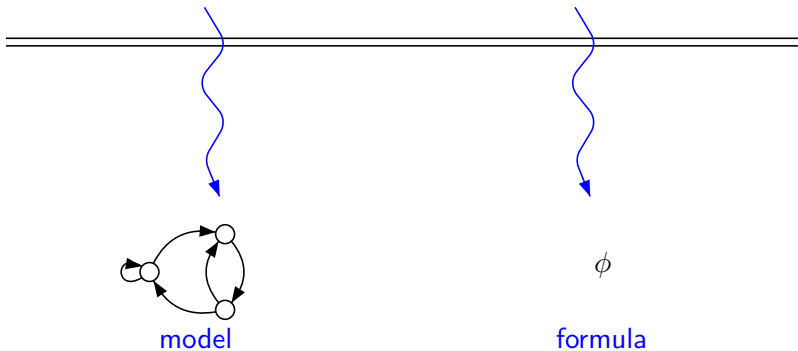
# Principles of Model-checking

Does a system satisfy a specification ?



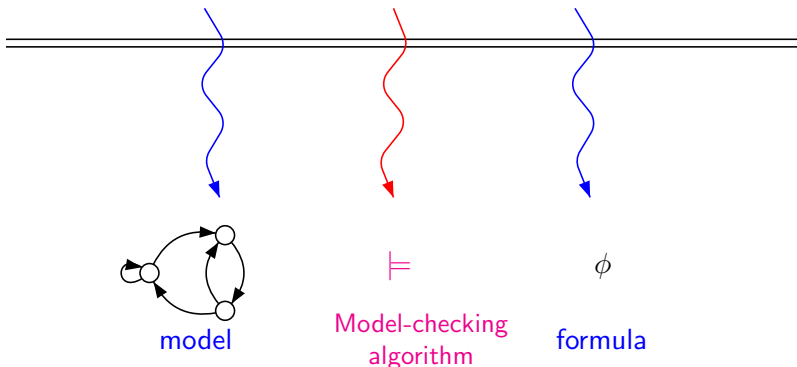
# Principles of Model-checking

Does a system satisfy a specification ?



# Principles of Model-checking

Does a system satisfy a specification ?



# Synthesis

a specification

# Synthesis

Find a system satisfying a specification

# Synthesis

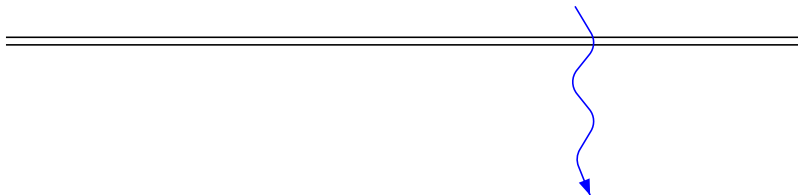
Find a system satisfying a specification

---

---

# Synthesis

Find a system satisfying a specification

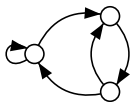


$\phi$

formula

# Synthesis

Find a system satisfying a specification



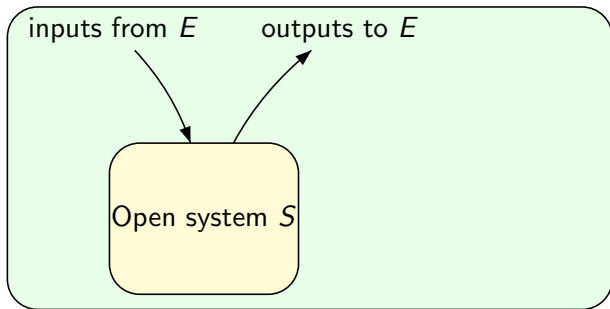
model

$\models$

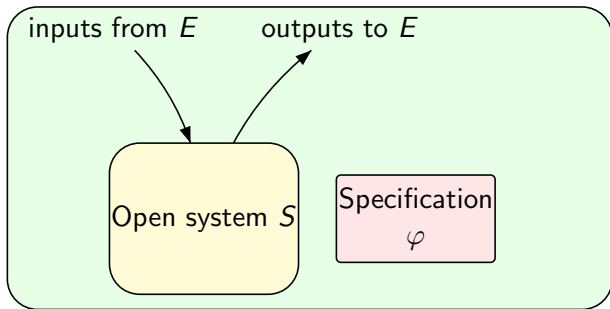
$\phi$

formula

# Synthesis of open and reactive systems



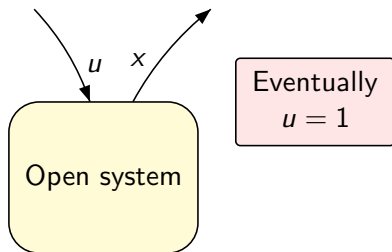
# Synthesis of open and reactive systems



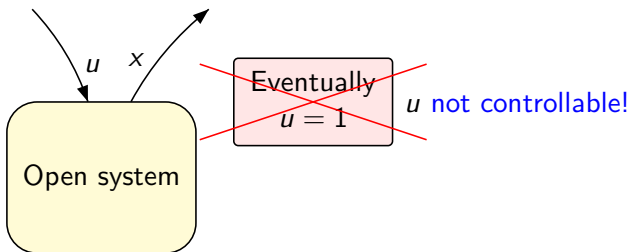
- ▶ Decide whether there exists a program st.  $P \parallel E \models \varphi$ ,  $\forall E$ .
- ▶ Synthesis: If so, compute such a program.

For reasonable systems and specifications, the problems are decidable.

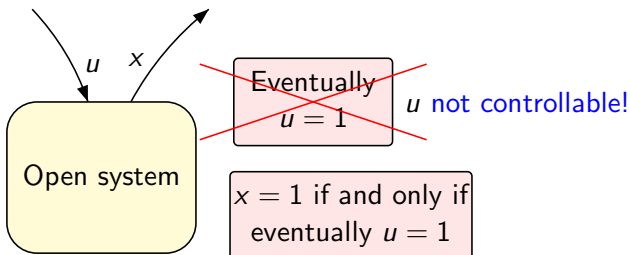
# Synthesis of open reactive systems different from satisfiability



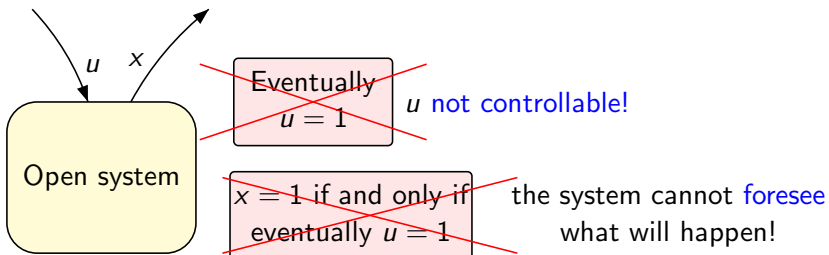
# Synthesis of open reactive systems different from satisfiability



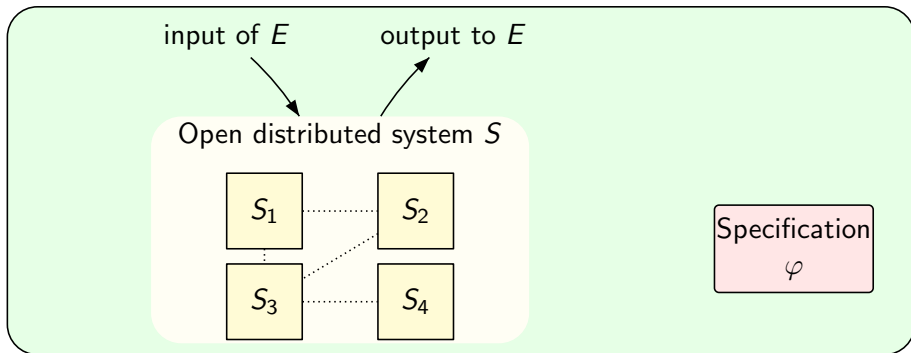
# Synthesis of open reactive systems different from satisfiability



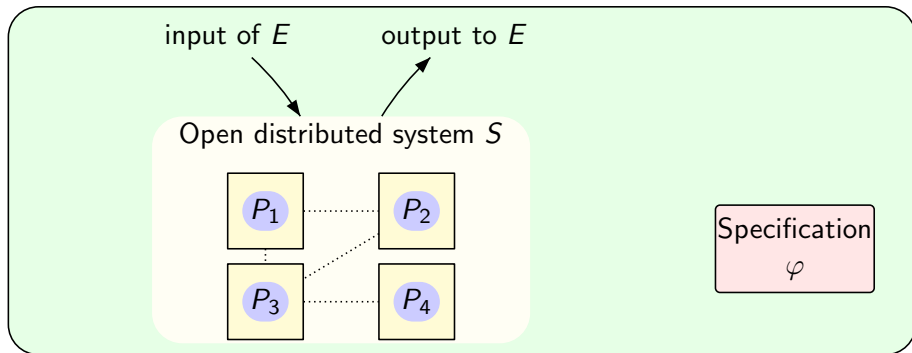
# Synthesis of open reactive systems different from satisfiability



# Synthesis of distributed open systems



# Synthesis of distributed open systems



## Two problems

- ▶ Decide the existence of a **distributed** program such that the **joint behavior**  $P_1 || P_2 || P_3 || P_4 || E$  satisfies  $\varphi$ , for all  $E$ .
- ▶ Synthesis: If it exists, compute such a **distributed** program.

# Synthesis of distributed systems

## Main parameters

- ▶ Which semantics?

synchronous, asynchronous

- ▶ What kind of specification?

- ▶ What kind of memory for the programs?

local memory

bounded or unbounded memory

# Outline

## Introduction

## Synthesis of synchronous distributed systems

- Model and motivations

- Uncomparable information

- Uniformly well connected architectures

- Well connected architectures

## Synthesis of asynchronous distributed systems

- Model

- Specifications

- Decidability Results

## Conclusion

# Outline

## Introduction

## Synthesis of synchronous distributed systems

- Model and motivations

- Uncomparable information

- Uniformly well connected architectures

- Well connected architectures

## Synthesis of asynchronous distributed systems

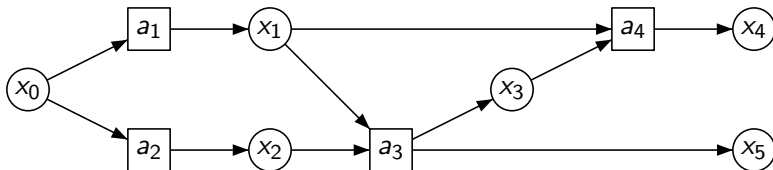
- Model

- Specifications

- Decidability Results

## Conclusion

# Distributed systems with shared variables

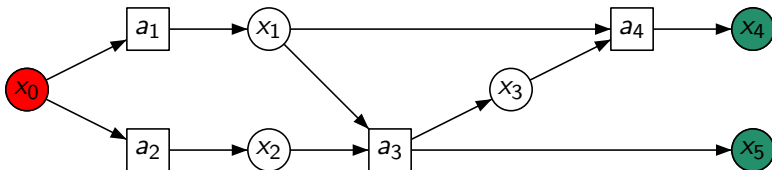


## Architecture

- ▶  $(\text{Proc} \uplus V, E)$  bipartite graph, where  $E \subseteq (\text{Proc} \times V) \cup (V \times \text{Proc})$ .
- ▶  $V_I \subseteq V$  input values from the environment, and  $V_O \subseteq V$  output values from the system, read by the environment.
- ▶  $S^v$  (finite) domain for each variable  $v \in V$ .
- ▶  $s_0 \in S^V$  initial state

where  $S^I = \prod_{v \in I} S^v$  for  $I \subseteq V$ .

# Distributed systems with shared variables

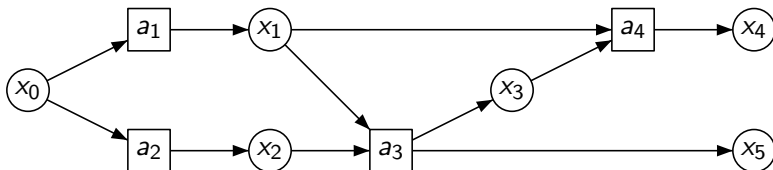


## Architecture

- ▶  $(\text{Proc} \uplus V, E)$  bipartite graph, where  $E \subseteq (\text{Proc} \times V) \cup (V \times \text{Proc})$ .
- ▶  $V_I \subseteq V$  input values from the environment, and  $V_O \subseteq V$  output values from the system, read by the environment.
- ▶  $S^v$  (finite) domain for each variable  $v \in V$ .
- ▶  $s_0 \in S^V$  initial state

where  $S^I = \prod_{v \in I} S^v$  for  $I \subseteq V$ .

# Synthesis of synchronous distributed systems

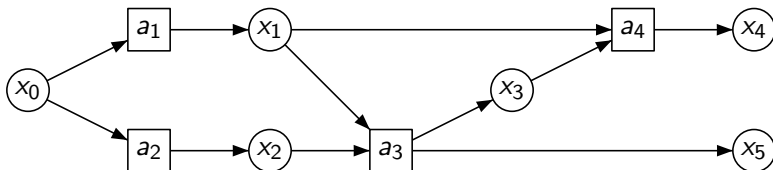


## Parameters

- ▶ Which semantics?

synchronous behaviors

# Synthesis of synchronous distributed systems



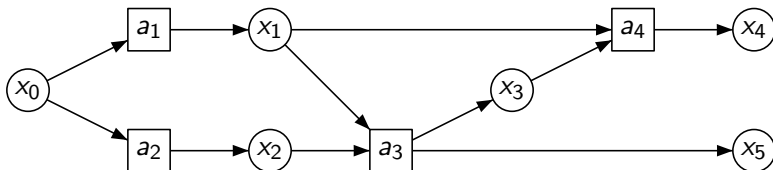
## Parameters

- ▶ Which semantics?

synchronous behaviors

$s_0 s_1 s_2 \dots$  where  $s_n \in S^V$  are global states.

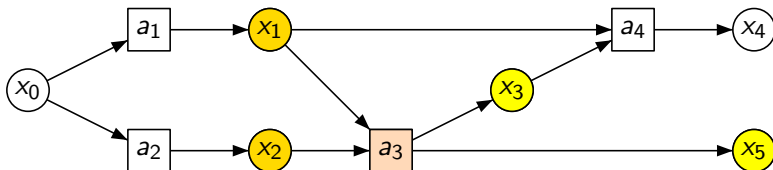
# Synthesis of synchronous distributed systems



## Parameters

- ▶ Which semantics? synchronous behaviors  
 $s_0 s_1 s_2 \dots$  where  $s_n \in S^V$  are global states.
- ▶ With or without delays?

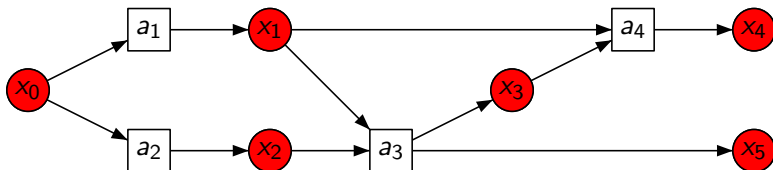
# Synthesis of synchronous distributed systems



## Parameters

- ▶ Which semantics? synchronous behaviors  
 $s_0 s_1 s_2 \dots$  where  $s_n \in S^V$  are global states.
- ▶ With or without delays?
- ▶ What kind of memory for the program? local memory  
 $f^p : (S^{E^{-1}(p)})^* \rightarrow S^{E(p)}$  for all  $p \in P$ .

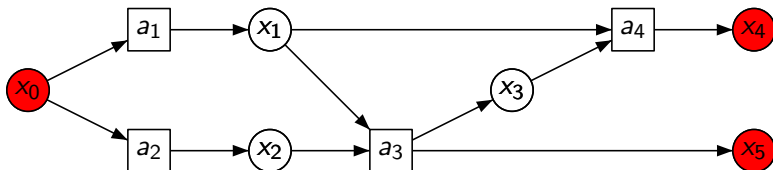
# Synthesis of synchronous distributed systems



## Parameters

- ▶ Which semantics? synchronous behaviors  
 $s_0 s_1 s_2 \dots$  where  $s_n \in S^V$  are global states.
- ▶ With or without delays?
- ▶ What kind of memory for the program? local memory  
 $f^p : (S^{E^{-1}(p)})^* \rightarrow S^{E(p)}$  for all  $p \in P$ .
- ▶ What kind of specification? Temporal logic formulae, total or external over words/trees over alphabet  $S^V$ .

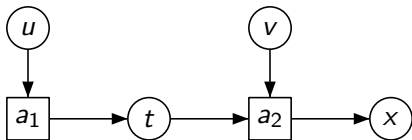
# Synthesis of synchronous distributed systems



## Parameters

- ▶ Which semantics? synchronous behaviors  
 $s_0 s_1 s_2 \dots$  where  $s_n \in S^V$  are global states.
- ▶ With or without delays?
- ▶ What kind of memory for the program? local memory  
 $f^p : (S^{E^{-1}(p)})^* \rightarrow S^{E(p)}$  for all  $p \in P$ .
- ▶ What kind of specification? Temporal logic formulae, total or external over words/trees over alphabet  $S^V$ .

## Synchronous runs



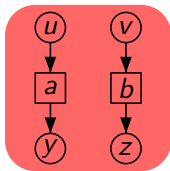
$u_1$	$u_2$	$u_3$	...
$v_1$	$v_2$	$v_3$	...
$t_1$	$t_2$	$t_3$	...
$x_1$	$x_2$	$x_3$	...

- ▶ 0-delay:
  - $t_i = f_t(u_1 \cdots u_i)$
  - $x_i = f_x((t_1, v_1) \cdots (t_i, v_i))$
- ▶ 1-delay:
  - $t_i = f_t(u_1 \cdots u_{i-1})$
  - $x_i = f_x((t_1, v_1) \cdots (t_{i-1}, v_{i-1}))$

# Undecidable and decidable architectures

## Pnueli-Rosner (FOCS'90)

Synthesis problem for synchronous distributed systems is undecidable for LTL or CTL **external** or **total** specifications.



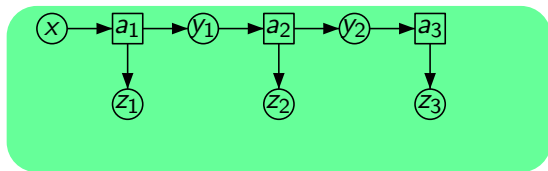
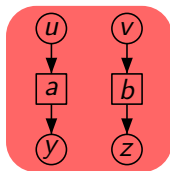
# Undecidable and decidable architectures

## Pnueli-Rosner (FOCS'90)

Synthesis problem for synchronous distributed systems is undecidable for LTL or CTL **external** or **total** specifications.

## Pnueli-Rosner (FOCS'90), Kupferman-Vardi (LICS'01)

Synthesis problem for pipeline architectures is decidable for CTL\* **total** specifications.



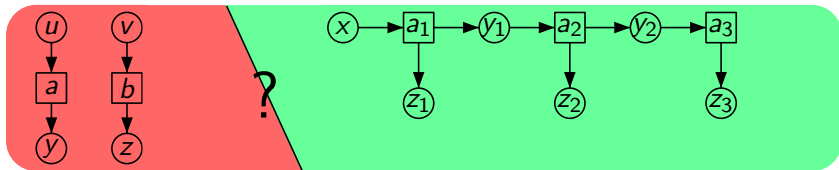
# Undecidable and decidable architectures

## Pnueli-Rosner (FOCS'90)

Synthesis problem for synchronous distributed systems is undecidable for LTL or CTL **external** or **total** specifications.

## Pnueli-Rosner (FOCS'90), Kupferman-Vardi (LICS'01)

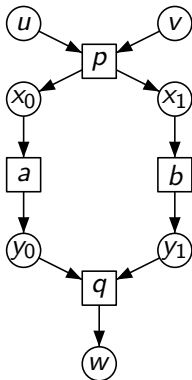
Synthesis problem for pipeline architectures is decidable for CTL\* **total** specifications.



# Total specifications: Information fork criterion

## Finkbeiner-Schewe (LICS'05)

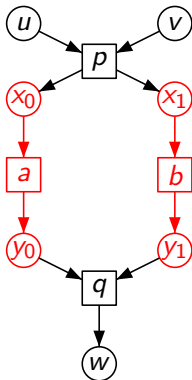
Synthesis problem is decidable for a given architecture if and only if there is no information fork.



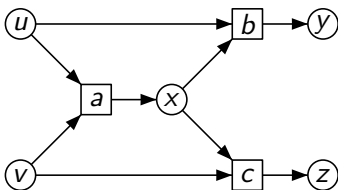
# Total specifications: Information fork criterion

## Finkbeiner-Schewe (LICS'05)

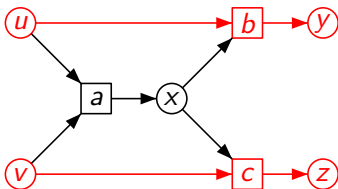
Synthesis problem is decidable for a given architecture if and only if there is no information fork.



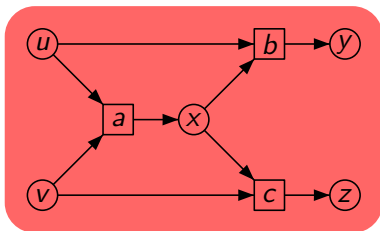
## Back to external specifications



## Back to external specifications



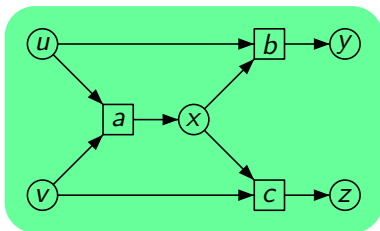
## Back to external specifications



Finkbeiner-Schewe (LICS'05)

Synthesis problem is undecidable over this architecture with LTL **total** specifications.

## Back to external specifications



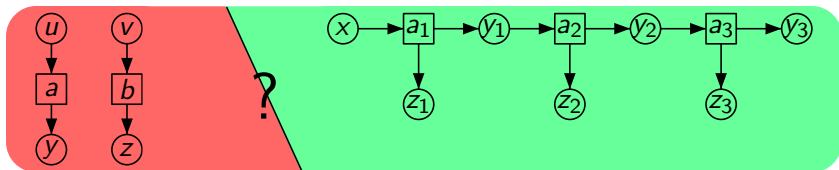
### Finkbeiner-Schewe (LICS'05)

Synthesis problem is undecidable over this architecture with LTL **total** specifications.

### Pnueli-Rosner (FOCS'90)

Synthesis problem is decidable over this architecture with LTL **external** specifications.

# What if we consider external specifications?



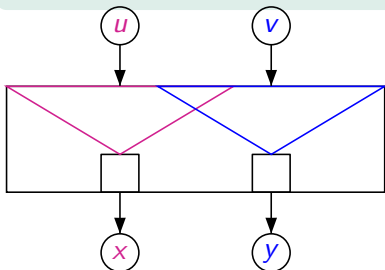
# Architectures with uncomparable information

## View of a variable

For an output variable  $x$ ,  $\text{View}(x)$  is the set of input variables  $u$  such that  $x$  is accessible from  $u$ .

## Uncomparable information (FSTTCS'06)

An architecture has **uncomparable information** if there exist  $x, y$  output variables such that  $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$  and  $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$ .



# Architectures with uncomparable information

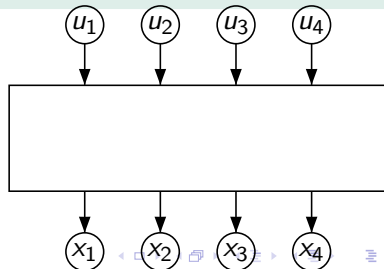
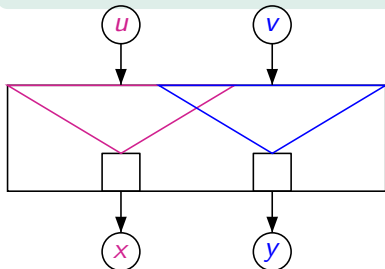
## View of a variable

For an output variable  $x$ ,  $\text{View}(x)$  is the set of input variables  $u$  such that  $x$  is accessible from  $u$ .

## Uncomparable information (FSTTCS'06)

An architecture has **uncomparable information** if there exist  $x, y$  output variables such that  $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$  and  $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$ .

Otherwise it is said to have **linearly preordered information**.



# Architectures with uncomparable information

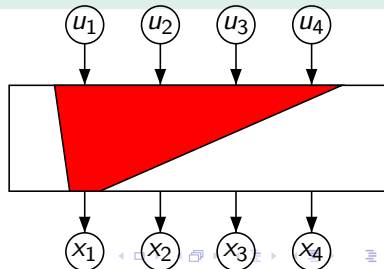
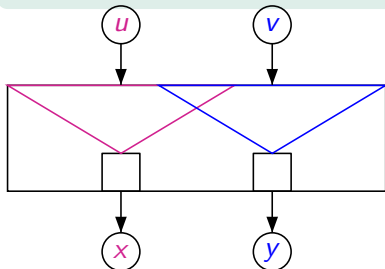
## View of a variable

For an output variable  $x$ ,  $\text{View}(x)$  is the set of input variables  $u$  such that  $x$  is accessible from  $u$ .

## Uncomparable information (FSTTCS'06)

An architecture has **uncomparable information** if there exist  $x, y$  output variables such that  $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$  and  $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$ .

Otherwise it is said to have **linearly preordered information**.



# Architectures with uncomparable information

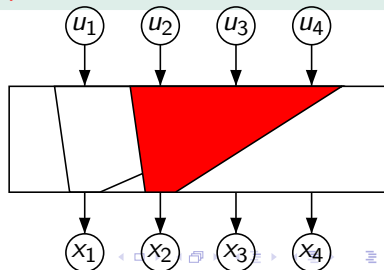
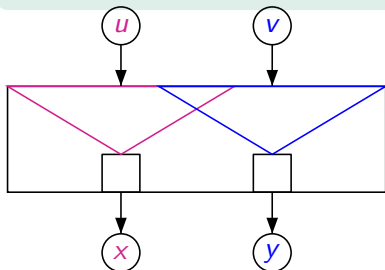
## View of a variable

For an output variable  $x$ ,  $\text{View}(x)$  is the set of input variables  $u$  such that  $x$  is accessible from  $u$ .

## Uncomparable information (FSTTCS'06)

An architecture has **uncomparable information** if there exist  $x, y$  output variables such that  $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$  and  $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$ .

Otherwise it is said to have **linearly preordered information**.



# Architectures with uncomparable information

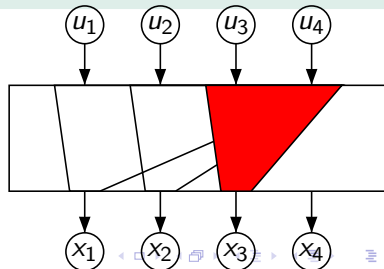
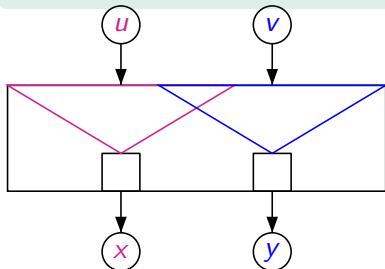
## View of a variable

For an output variable  $x$ ,  $\text{View}(x)$  is the set of input variables  $u$  such that  $x$  is accessible from  $u$ .

## Uncomparable information (FSTTCS'06)

An architecture has **uncomparable information** if there exist  $x, y$  output variables such that  $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$  and  $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$ .

Otherwise it is said to have **linearly preordered information**.



# Architectures with uncomparable information

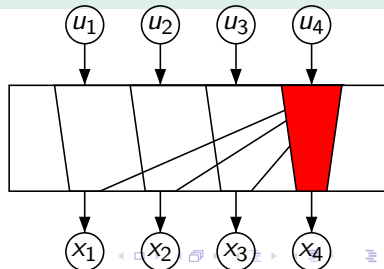
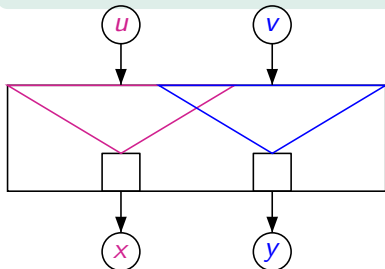
## View of a variable

For an output variable  $x$ ,  $\text{View}(x)$  is the set of input variables  $u$  such that  $x$  is accessible from  $u$ .

## Uncomparable information (FSTTCS'06)

An architecture has **uncomparable information** if there exist  $x, y$  output variables such that  $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$  and  $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$ .

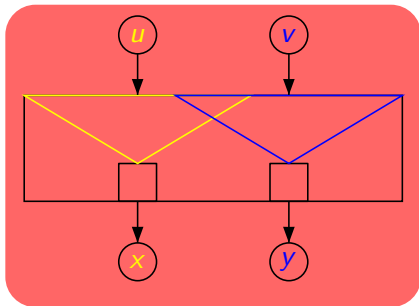
Otherwise it is said to have **linearly preordered information**.



# Uncomparable information yields undecidability

## Theorem (FSTTCS'06,FMDS'09)

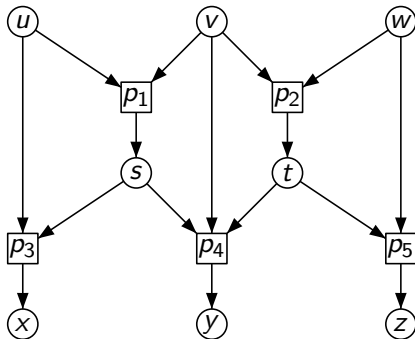
Synthesis problem is undecidable for architectures with uncomparable information and LTL or CTL **external** specifications.



# Uniformly well connected architectures

## Definition

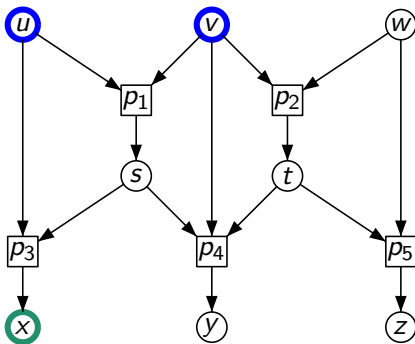
An architecture is **uniformly well connected (UWC)** if there is a unique routing which, for each output variable  $x$ , sends variables in  $\text{View}(x)$  to  $x$ .



# Uniformly well connected architectures

## Definition

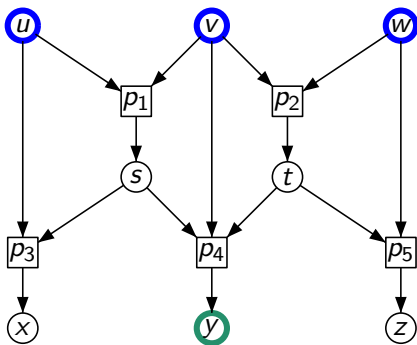
An architecture is **uniformly well connected (UWC)** if there is a unique routing which, for each output variable  $x$ , sends variables in  $\text{View}(x)$  to  $x$ .



# Uniformly well connected architectures

## Definition

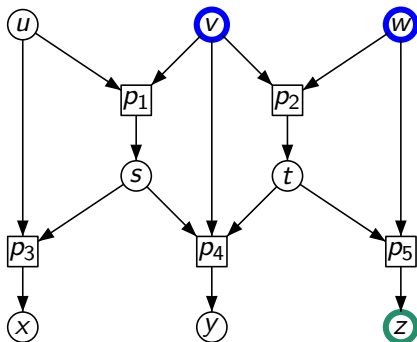
An architecture is **uniformly well connected (UWC)** if there is a unique routing which, for each output variable  $x$ , sends variables in  $\text{View}(x)$  to  $x$ .



# Uniformly well connected architectures

## Definition

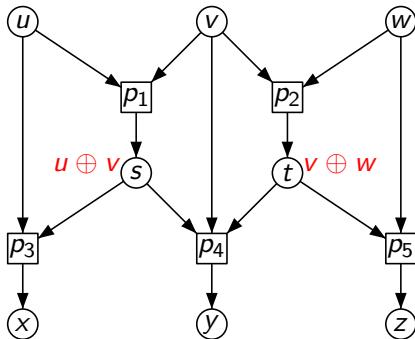
An architecture is **uniformly well connected (UWC)** if there is a unique routing which, for each output variable  $x$ , sends variables in  $\text{View}(x)$  to  $x$ .



# Uniformly well connected architectures

## Definition

An architecture is **uniformly well connected (UWC)** if there is a unique routing which, for each output variable  $x$ , sends variables in  $\text{View}(x)$  to  $x$ .



# Decidability results

## Proposition

Checking if an architecture is UWC is decidable, in 2-NEXPTIME and NP-hard.

# Decidability results

## Proposition

Checking if an architecture is UWC is decidable, in 2-NEXPTIME and NP-hard.

## Theorem (FSTTCS'06, FMSSD'09)

Synthesis problem is decidable for **UWC architectures** with **linearly preordered information** and **external** specifications (branching or linear time).

## Proof idea

Routing is used for memoryless internal strategies.

# Robust specifications

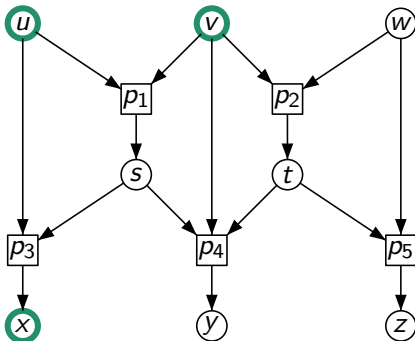
## Definition (FSTTCS'06, FMDS'09)

A formula  $\varphi$  in CTL\* is **robust** if  $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \psi_z$  where  $\psi_z$  only depends on  $\text{View}(z) \cup \{z\}$ .

# Robust specifications

## Definition (FSTTCS'06, FMDS'09)

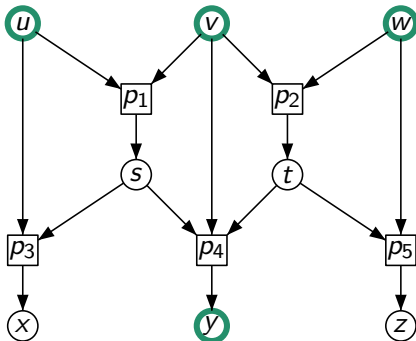
A formula  $\varphi$  in CTL\* is **robust** if  $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \psi_z$  where  $\psi_z$  only depends on  $\text{View}(z) \cup \{z\}$ .



# Robust specifications

## Definition (FSTTCS'06, FMDS'09)

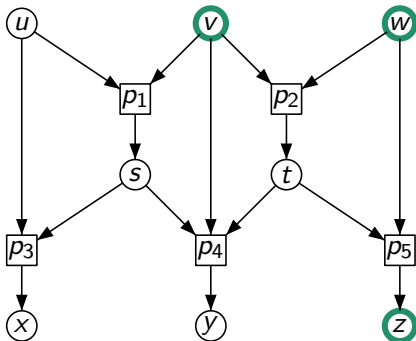
A formula  $\varphi$  in CTL\* is **robust** if  $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \psi_z$  where  $\psi_z$  only depends on  $\text{View}(z) \cup \{z\}$ .



# Robust specifications

## Definition (FSTTCS'06, FMDS'09)

A formula  $\varphi$  in CTL\* is **robust** if  $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \psi_z$  where  $\psi_z$  only depends on  $\text{View}(z) \cup \{z\}$ .



# Robust specifications

## Definition (FSTTCS'06,FMSD'09)

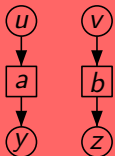
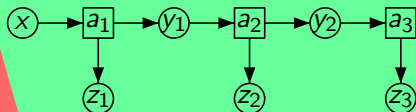
A formula  $\varphi \in \text{CTL}^*$  is **robust** if  $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \psi_z$  where  $\psi_z$  only depends on  $\text{View}(z) \cup \{z\}$ .

## Theorem (FSTTCS'06, FMSD'09)

Synthesis problem is decidable for **uniformly well connected architectures** and **external, robust** CTL\* specifications.

# Undecidable and decidable architectures - Total specifications

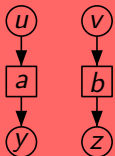
Information fork



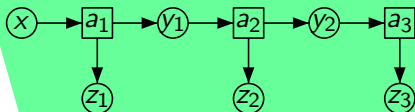
# Undecidable and decidable architectures -

## External specifications

Uncomparable  
information



?

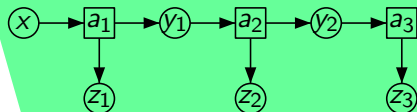


## Undecidable and decidable architectures -

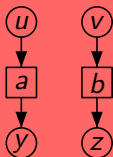
## External specifications

Uncomparable  
information

?



UWC architectures

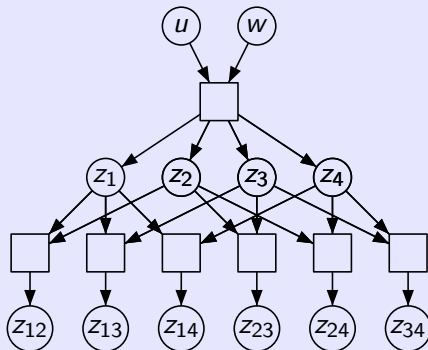


# Well Connected Architectures

## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)

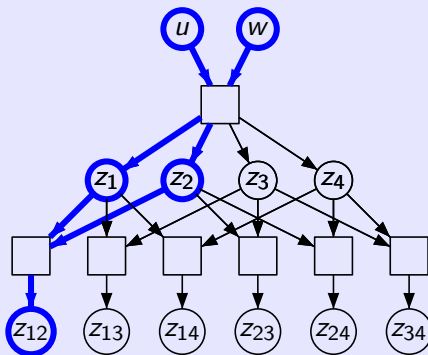


# Well Connected Architectures

## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)

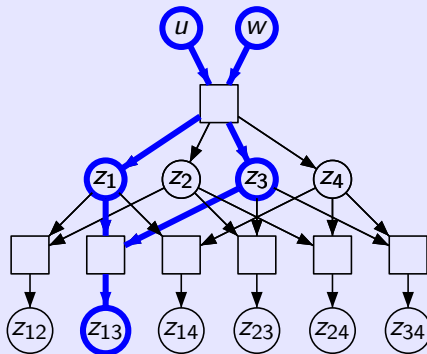


# Well Connected Architectures

## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)

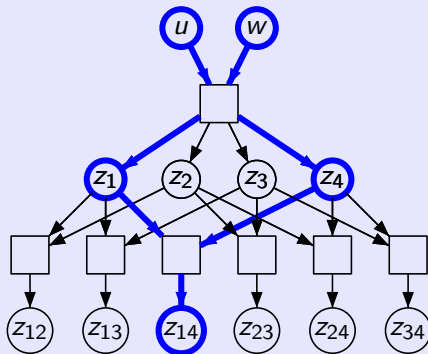


# Well Connected Architectures

## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)

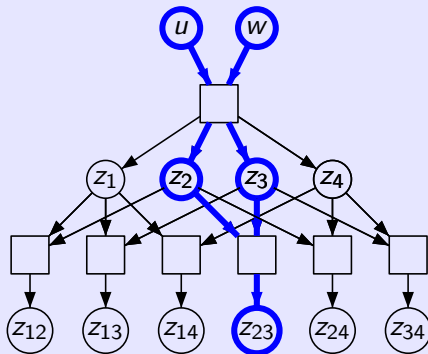


# Well Connected Architectures

## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)

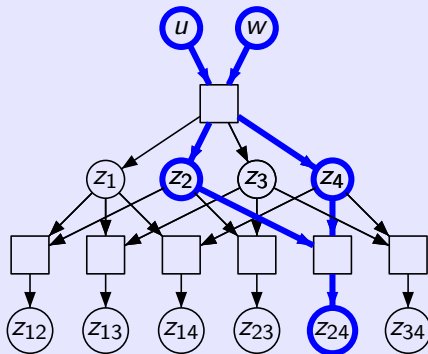


# Well Connected Architectures

## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)

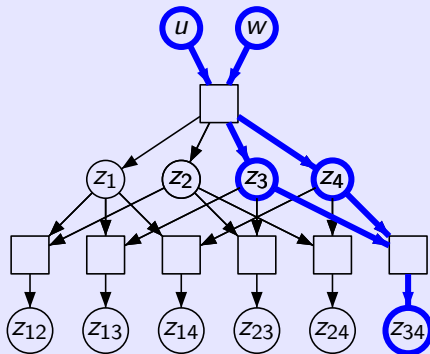


# Well Connected Architectures

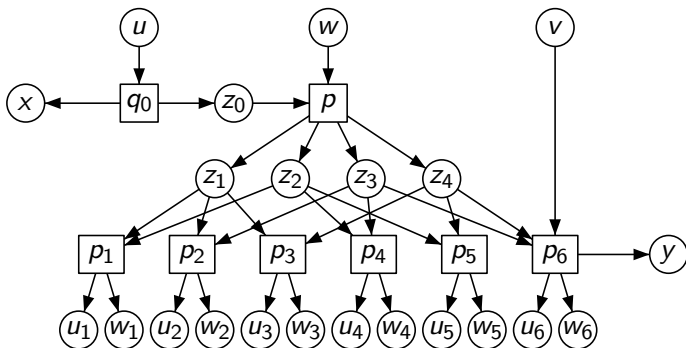
## Definition

An architecture is **well connected** if, for each output variable  $v$ , the subarchitecture formed by its 'ancestors' is UWC.

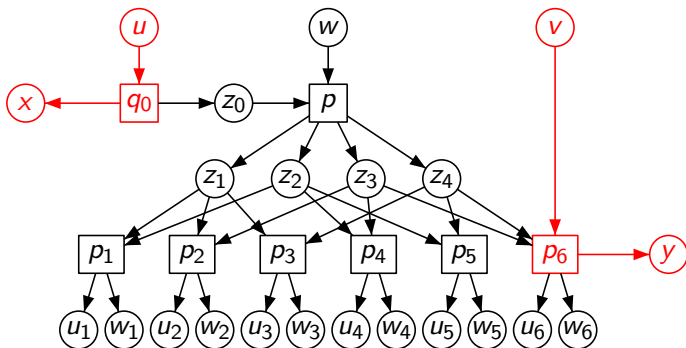
A well-connected architecture, but not UWC (from Rasala Lehman-Lehman, SODA'04)



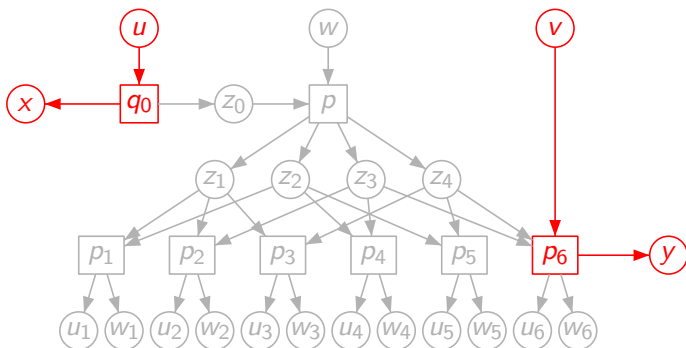
# Undecidability for well connected architectures with linearly preordered information



# Undecidability for well connected architectures with linearly preordered information

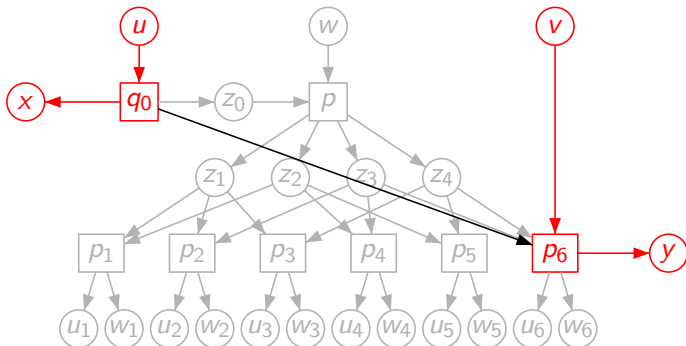


# Undecidability for well connected architectures with linearly preordered information



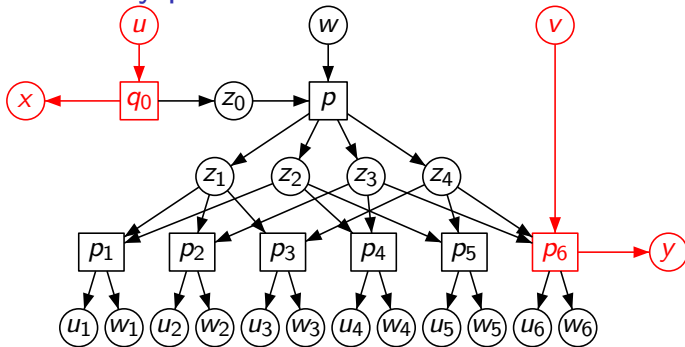
Process  $p_6$  knowing **no value** of  $u$  yields **undecidability**

# Undecidability for well connected architectures with linearly preordered information



Process  $p_6$  knowing **all values** of  $u$  yields **decidability**

# Undecidability for well connected architectures with linearly preordered information

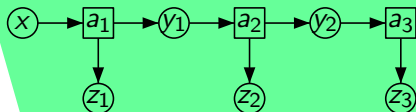


Process  $p_6$  missing **one bit** of  $u$  yields **undecidability**

# Undecidable and decidable architectures - External specifications

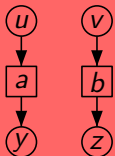
Uncomparable information

?



WC architectures

UWC architectures



## Related work

- ▶ **Total specifications:** [Kupferman-Vardi, LICS'01], [Finkbeiner-Schewe, LICS'05]
- ▶ **External specifications:** [Pnueli-Rosner, FOCS'90], [S., PhD'09]
- ▶ **Local specifications:** [Madhusudan-Thiagarajan, ICALP'01]
- ▶ **Distributed games framework:** [Peterson-Reif, FOCS'79], [Mohalik-Walukiewicz, FSTTCS'03], [Bernet-Janin, FCT'05]

# Outline

## Introduction

## Synthesis of synchronous distributed systems

- Model and motivations

- Uncomparable information

- Uniformly well connected architectures

- Well connected architectures

## Synthesis of asynchronous distributed systems

- Model

- Specifications

- Decidability Results

## Conclusion

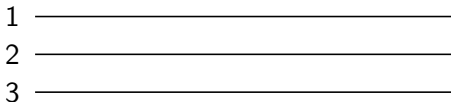
# Distributed Synthesis

## Parameters

- ▶ Which semantics? **asynchronous**  
executions are partial orders (Mazurkiewicz traces)
- ▶ What kind of memory for the programs? **local memory**
- ▶ What kind of specification? **external**, over **partial orders**

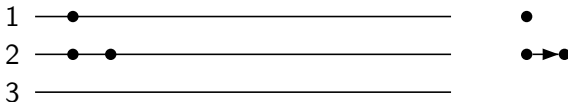
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.



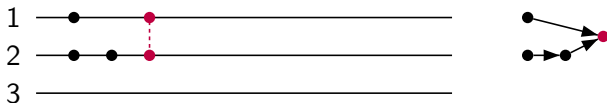
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.



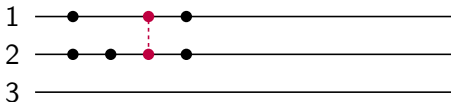
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.



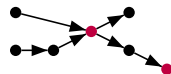
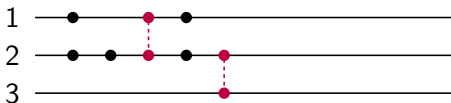
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.



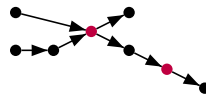
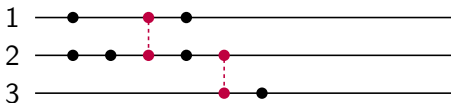
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.



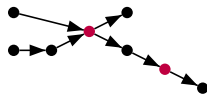
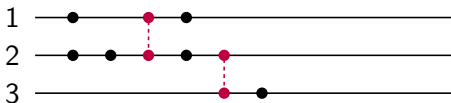
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.



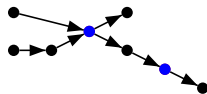
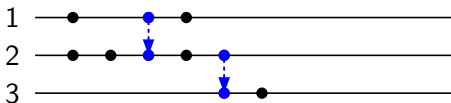
# Asynchronous semantics : communication through common actions

- ▶ Rendez-vous: two processes agree on a common action.  
**Drawback:** For an action to be played, the two processes have to take the same decision, maybe with different knowledge.



# Asynchronous semantics : communication through common actions

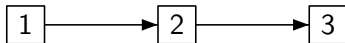
- ▶ Rendez-vous: two processes agree on a common action.  
**Drawback:** For an action to be played, the two processes have to take the same decision, maybe with different knowledge.
- ▶ **Signal:** asymmetric rendez-vous. A common action is initiated by only one process.



# Communication by signals

## Architectures

- ▶ Communication graph ( $Proc, E$ )

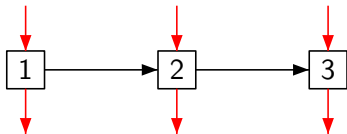


# Communication by signals

## Architectures

- ▶ Communication graph  $(Proc, E)$
- ▶ For each process  $i$ , sets  $In_i$  and  $Out_i$  of input and output signals:

$$\Gamma = \bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i$$



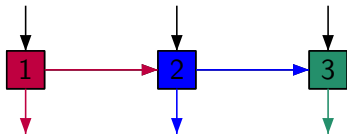
# Communication by signals

## Architectures

- ▶ Communication graph  $(Proc, E)$
- ▶ For each process  $i$ , sets  $In_i$  and  $Out_i$  of input and output signals:

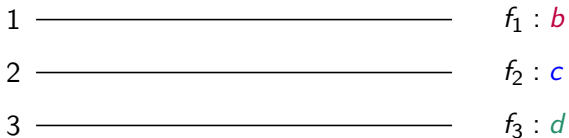
$$\Gamma = \bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i$$

- ▶ For each process  $i$ ,  
 $\Sigma_i^c$  is the set of signals it can send (control),  
 $\Sigma_i$  is the set of signals it can observe.



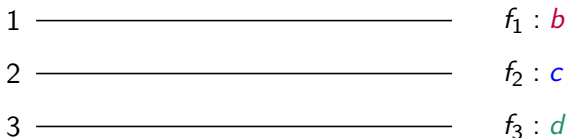
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.



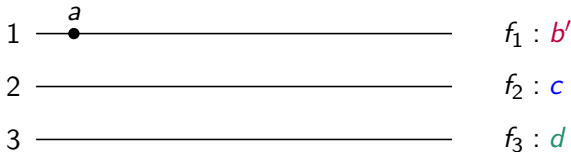
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



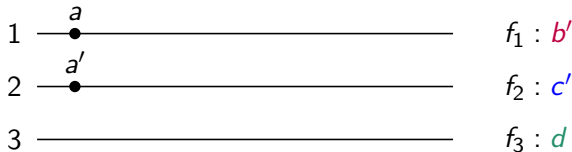
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



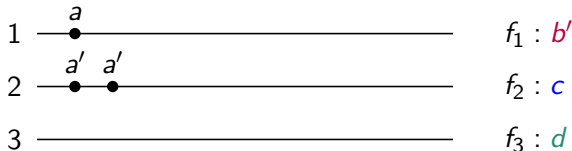
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



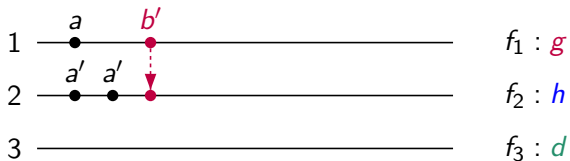
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



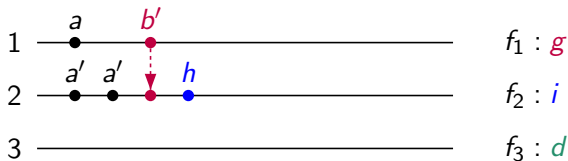
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



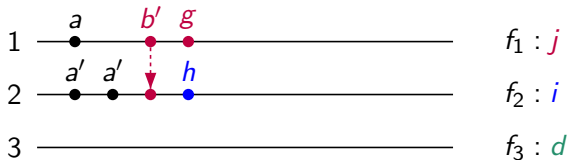
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



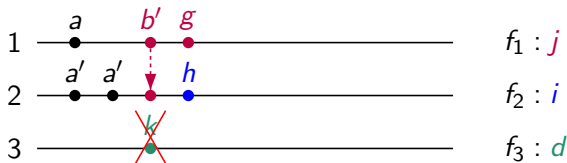
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.



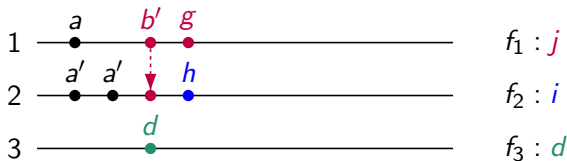
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.
- ▶ A run respects a strategy  $f = (f_i)_{i \in \text{Proc}}$  (is an  **$f$ -run**) if each event of process  $i$  labelled with a **controllable action** respects the strategy  $f_i$ .



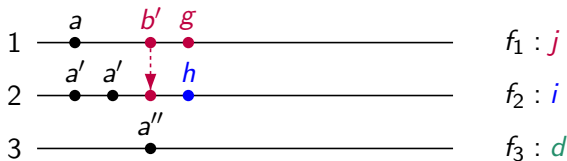
# Programs

- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.
- ▶ A run respects a strategy  $f = (f_i)_{i \in \text{Proc}}$  (is an  **$f$ -run**) if each event of process  $i$  labelled with a **controllable action** respects the strategy  $f_i$ .

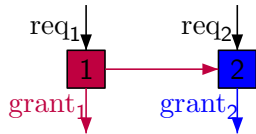


# Programs

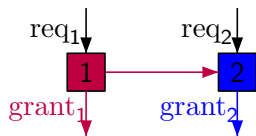
- ▶ Strategies are partial functions  $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$  with **local** memory.
- ▶ Signal semantics implies **reactivity** of processes to events.
- ▶ A run respects a strategy  $f = (f_i)_{i \in \text{Proc}}$  (is an  **$f$ -run**) if each event of process  $i$  labelled with a **controllable action** respects the strategy  $f_i$ .



## Fairness conditions

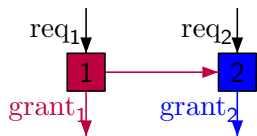


## Fairness conditions

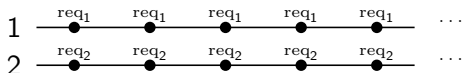


$$G(\text{req}_1 \rightarrow (F \text{grant}_1)) \wedge G(\text{req}_2 \rightarrow (F \text{grant}_2))$$

## Fairness conditions



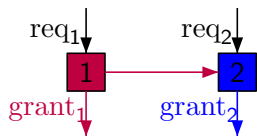
$$G(\text{req}_1 \rightarrow (F \text{grant}_1)) \wedge G(\text{req}_2 \rightarrow (F \text{grant}_2))$$



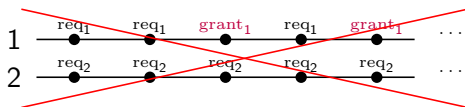
- Some runs are **unfair** for the processes.



## Fairness conditions



$$G(\text{req}_1 \rightarrow (F \text{grant}_1)) \wedge G(\text{req}_2 \rightarrow (F \text{grant}_2))$$



UNFAIR

$$f_1(\sigma) = \text{grant}_1$$

$$f_2(\sigma) = \text{grant}_2$$

- ▶ Some runs are **unfair** for the processes.
- ▶ Fairness has to be **distributed**.

# Models of an external specification

## Parameters

- ▶ Which semantics? **asynchronous**  
executions are partial orders (Mazurkiewicz traces)
- ▶ What kind of memory for the programs? **local memory**
- ▶ What kind of specification? **external**, over **partial orders**

# Models of an external specification

## Parameters

- ▶ What kind of specification? **external**, over **partial orders**

# Models of an external specification

## Observable runs

Given a run  $t = (V, \lambda, \leq)$ , we define the **observable run** by

$$\pi_{\Gamma}(t) = (\Gamma, \lambda|_{\Gamma}, \leq \cap (\Gamma \times \Gamma))$$

where  $\Gamma$  is the set of external actions.

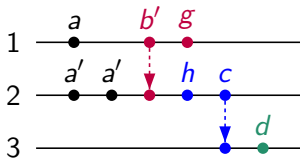
# Models of an external specification

## Observable runs

Given a run  $t = (V, \lambda, \leq)$ , we define the **observable run** by

$$\pi_{\Gamma}(t) = (\Gamma, \lambda|_{\Gamma}, \leq \cap (\Gamma \times \Gamma))$$

where  $\Gamma$  is the set of external actions.



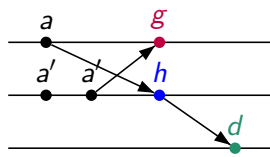
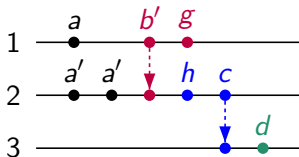
# Models of an external specification

## Observable runs

Given a run  $t = (V, \lambda, \leq)$ , we define the **observable run** by

$$\pi_{\Gamma}(t) = (\Gamma, \lambda|_{\Gamma}, \leq \cap (\Gamma \times \Gamma))$$

where  $\Gamma$  is the set of external actions.

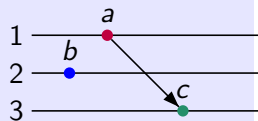
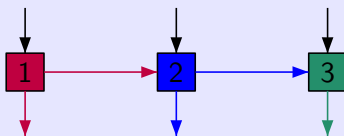


# Acceptable Specifications

Communication induces order relation

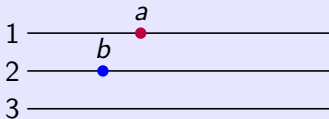
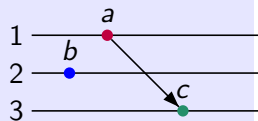
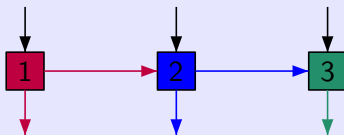
# Acceptable Specifications

## Communication induces order relation



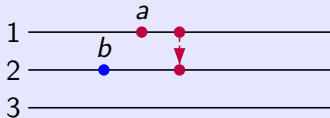
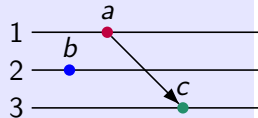
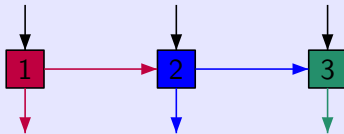
# Acceptable Specifications

## Communication induces order relation



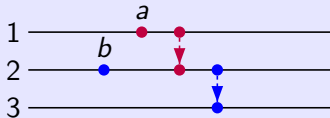
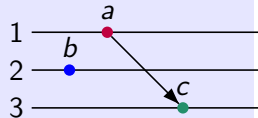
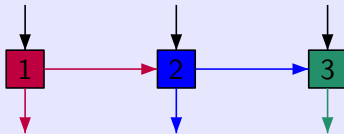
# Acceptable Specifications

## Communication induces order relation



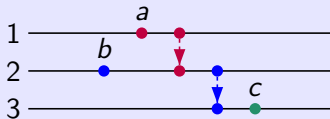
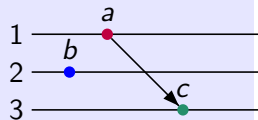
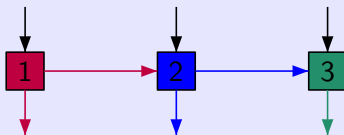
# Acceptable Specifications

## Communication induces order relation



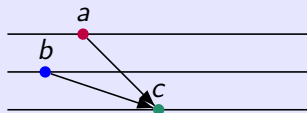
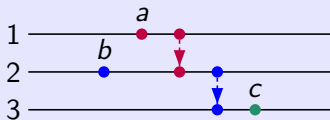
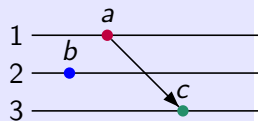
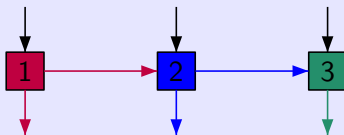
# Acceptable Specifications

## Communication induces order relation



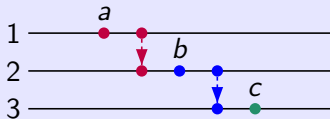
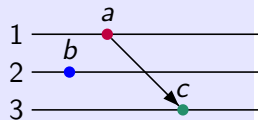
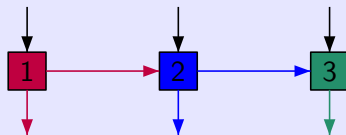
# Acceptable Specifications

## Communication induces order relation



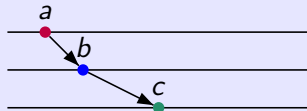
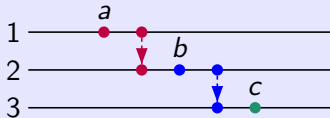
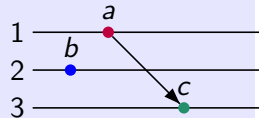
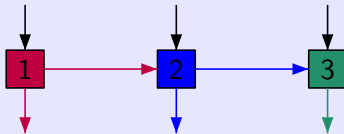
# Acceptable Specifications

## Communication induces order relation



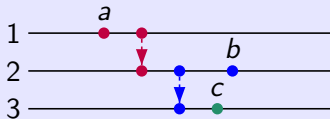
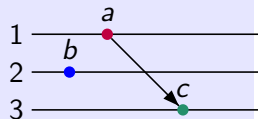
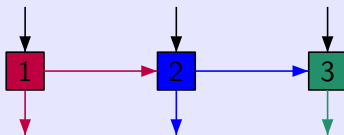
# Acceptable Specifications

## Communication induces order relation



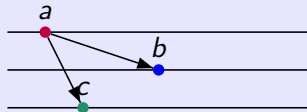
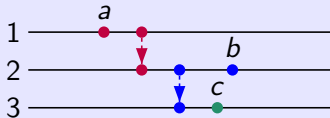
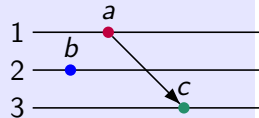
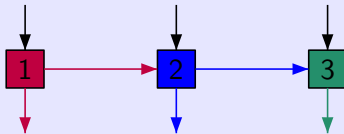
# Acceptable Specifications

## Communication induces order relation



# Acceptable Specifications

## Communication induces order relation



# Acceptable Specifications

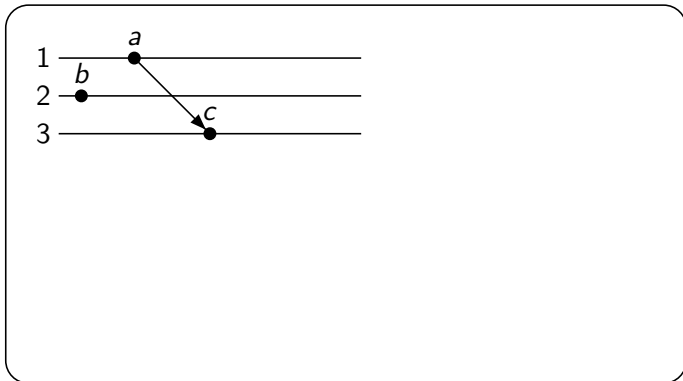
## Restrictions on specifications

- ▶ **Communication induces order relation:** specifications should not discriminate between a partial order and its **order extensions**

# Acceptable Specifications

## Restrictions on specifications

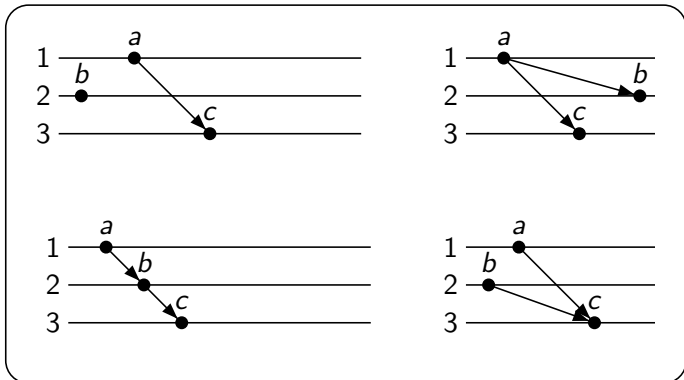
- **Communication induces order relation:** specifications should not discriminate between a partial order and its **order extensions**



# Acceptable Specifications

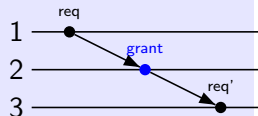
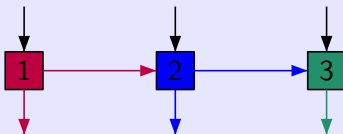
## Restrictions on specifications

- ▶ **Communication induces order relation:** specifications should not discriminate between a partial order and its **order extensions**



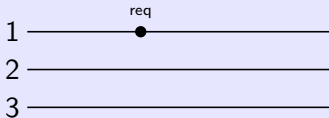
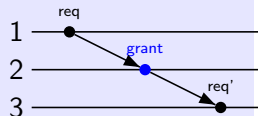
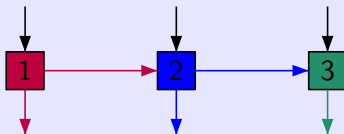
# Acceptable Specifications

Input events are not controllable by processes



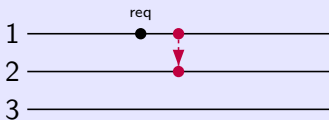
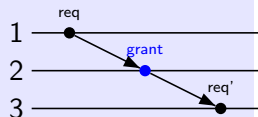
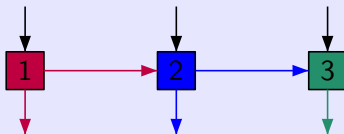
# Acceptable Specifications

Input events are not controllable by processes



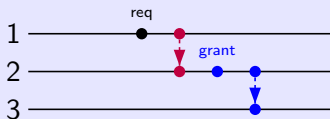
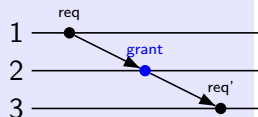
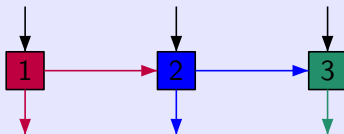
# Acceptable Specifications

Input events are not controllable by processes



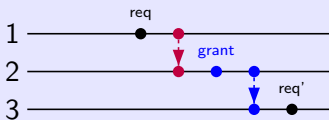
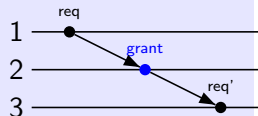
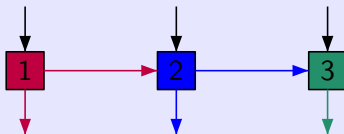
# Acceptable Specifications

Input events are not controllable by processes



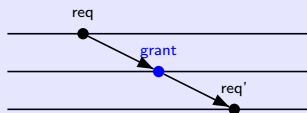
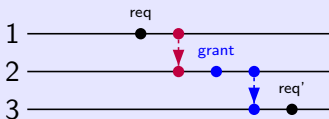
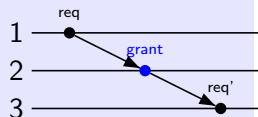
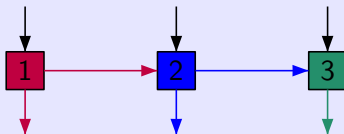
# Acceptable Specifications

Input events are not controllable by processes



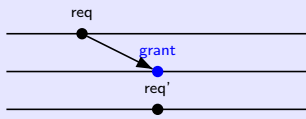
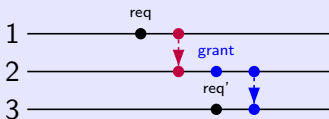
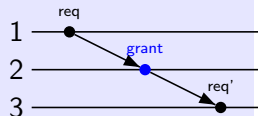
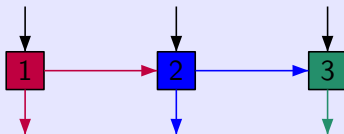
# Acceptable Specifications

Input events are not controllable by processes



# Acceptable Specifications

Input events are not controllable by processes



# Acceptable Specifications

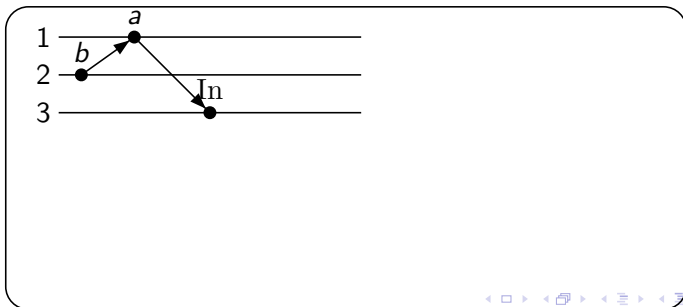
## Restrictions on specifications

- ▶ **Communication induces order relation**: specifications should not discriminate between a partial order and its order extensions
- ▶ **Input events are not controllable**: specifications should not discriminate between a partial order and its “weakenings”

# Acceptable Specifications

## Restrictions on specifications

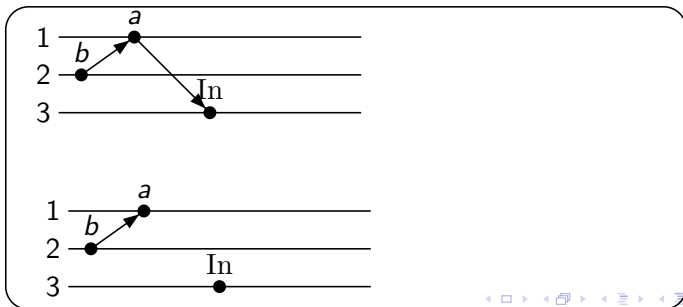
- ▶ **Communication induces order relation**: specifications should not discriminate between a partial order and its order extensions
- ▶ **Input events are not controllable**: specifications should not discriminate between a partial order and its “weakenings”



# Acceptable Specifications

## Restrictions on specifications

- ▶ **Communication induces order relation:** specifications should not discriminate between a partial order and its order extensions
- ▶ **Input events are not controllable:** specifications should not discriminate between a partial order and its “weakenings”



# Fair synthesis problem

# Fair synthesis problem

**Given** an architecture  
a specification

## Fair synthesis problem

- Given** an architecture  
a specification
- Decide** whether there exists a distributed program such that  
all its fair runs meet the specification.

## Fair synthesis problem

- Given** an architecture  
a specification
- Decide** whether there exists a distributed program such that  
all its fair runs meet the specification.

### Theorem (SOFSEM'09 + PhD)

The fair synthesis problem over singleton architectures is decidable for regular specifications.

## Fair synthesis problem

- Given** an architecture  
a specification
- Decide** whether there exists a distributed program such that  
all its fair runs meet the specification.

### Theorem (SOFSEM'09 + PhD)

The fair synthesis problem over singleton architectures is decidable for regular specifications.

### Theorem (SOFSEM'09 + PhD)

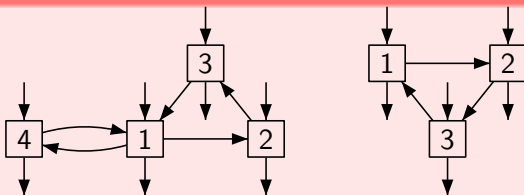
The fair synthesis problem over strongly connected architectures is decidable for acceptable specifications.

### Proof idea

By reduction to the singleton

# Distributing a centralized strategy

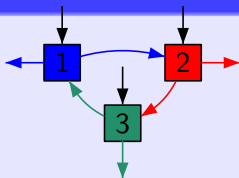
## Proof



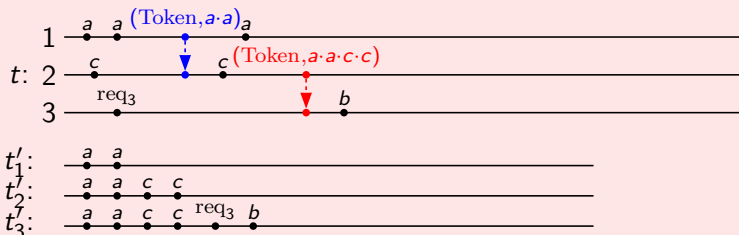
- ▶ We select a cycle.
- ▶ The processes will use a token to play one at a time and collect information on what happened in their past
- ▶ Aim: create a run that will be a **weakening** of some  $f$ -run over the singleton

## Token passing

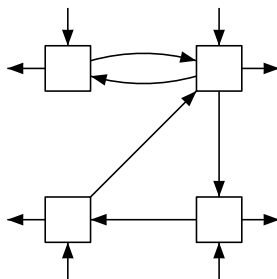
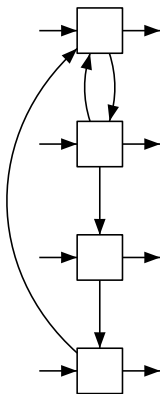
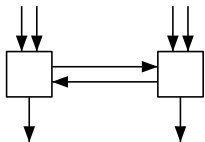
## Example



## Process 1 has the token at the beginning



# Examples of strongly connected architectures



## Related Work

- ▶ **Causal memory:** [Gastin-Lerman-Zeitoun, FSTTCS'04], [Madhusudan-Thiagarajan-Yang, FSTTCS'05]
- ▶ **Local memory:** [Madhusudan-Thiagarajan, CONCUR'02], [S., PhD'09]
- ▶ **Distributed games framework:** [Mohalik-Walukiewicz, FSTTCS'03]

# Outline

## Introduction

### Synthesis of synchronous distributed systems

- Model and motivations

- Uncomparable information

- Uniformly well connected architectures

- Well connected architectures

### Synthesis of asynchronous distributed systems

- Model

- Specifications

- Decidability Results

## Conclusion

# Summary

## Synthesis of synchronous systems

- ▶ Necessary condition for decidability for external specifications
- ▶ Exhibition of a new class of architectures for which it becomes a sufficient condition
- ▶ New undecidability proof giving new insights

## Synthesis of asynchronous systems

- ▶ Definition of a realistic model for synthesis of asynchronous systems
- ▶ Decidability of a class which is undecidable in the synchronous case

# Open problems

- ▶ Synchronous case
  - ▶ Definition of a general decidability criterion for external specifications in the synchronous case
- ▶ Asynchronous case
  - ▶ Obtain decidability of the problem on all architectures
- ▶ Fault-tolerant synthesis

Thank you for your attention!