

How to get decidability of distributed synthesis for asynchronous systems

Nathalie Sznajder

Joint work with Thomas Chatain and Paul Gastin

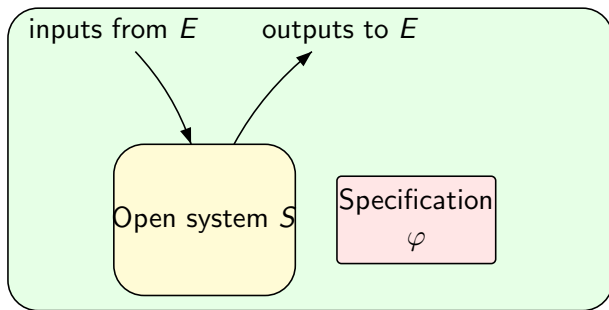
March 05-06, 2009

GASICS Meeting

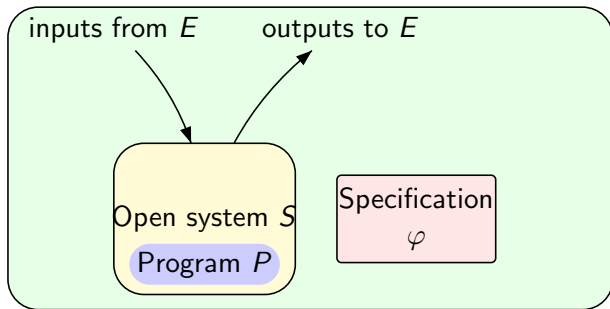
Outline

- 1 Introduction
- 2 Model
- 3 Specification
- 4 Decidability Results

Synthesis of a reactive system



Synthesis of a reactive system

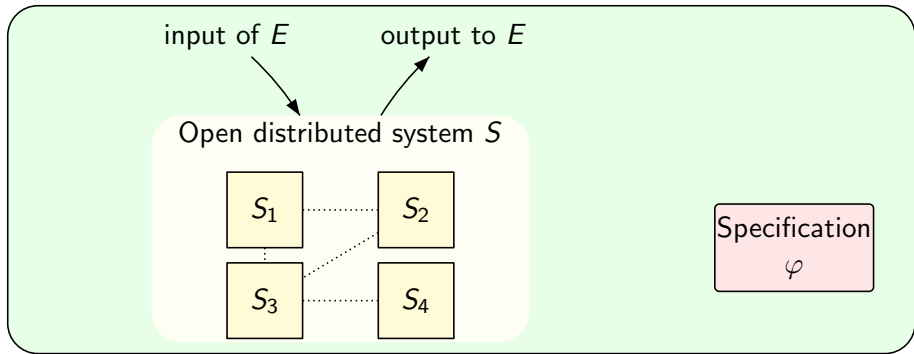


Two problems

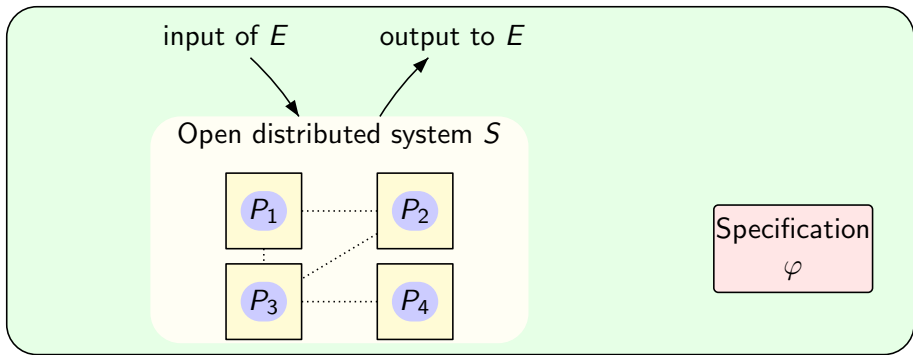
- Decide whether there exists a program st. $P \parallel E \models \varphi, \quad \forall E.$
- Synthesis: If so, compute such a program.

For reasonable systems and specifications, the problems are decidable.

Distributed synthesis



Distributed synthesis



Two problems

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

Distributed synthesis

Synchronous or asynchronous semantics?

Synchronous semantics

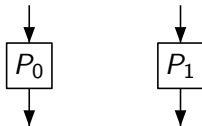
- At each tick of a global clock, all processes and the environment output their new value
- Introduced in [PnueliRosner90].
- In general undecidable.

Distributed synthesis

Synchronous or asynchronous semantics?

Synchronous semantics

- At each tick of a global clock, all processes and the environment output their new value
- Introduced in [PnueliRosner90].
- In general undecidable.



Distributed synthesis

Synchronous or asynchronous semantics?

Synchronous semantics

- At each tick of a global clock, all processes and the environment output their new value
- Introduced in [PnueliRosner90].
- In general undecidable.



Asynchronous semantics

Our model

- Processes evolve asynchronously for **local** actions (i.e., communications with the environment)

Asynchronous semantics

Our model

- Processes evolve asynchronously for **local** actions (i.e., communications with the environment)
- They can synchronize by **signals** = common actions initiated by only one process. A process cannot refuse reception of a signal.

Asynchronous semantics

Our model

- Processes evolve asynchronously for **local** actions (i.e., communications with the environment)
- They can synchronize by **signals** = common actions initiated by only one process. A process cannot refuse reception of a signal.
- Specifications :
 - ▶ over **partial orders**

Asynchronous semantics

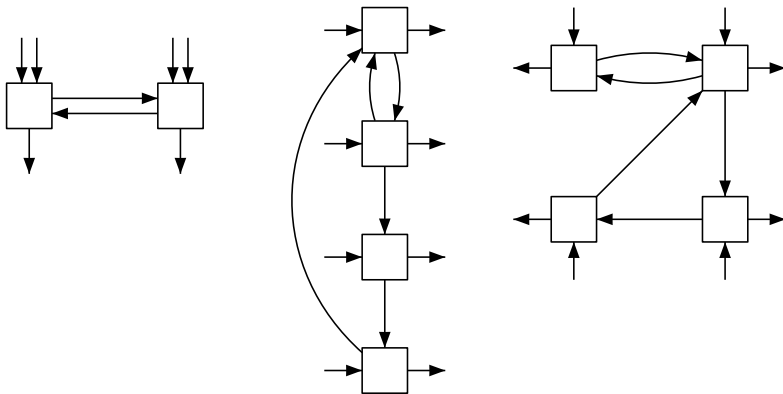
Our model

- Processes evolve asynchronously for **local** actions (i.e., communications with the environment)
- They can synchronize by **signals** = common actions initiated by only one process. A process cannot refuse reception of a signal.
- Specifications :
 - ▶ over **partial orders**
 - ▶ will not restrain **communication abilities**

Decidability Results

Theorem

Synthesis problem is decidable for strongly-connected architectures



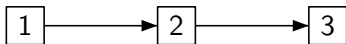
Outline

- 1 Introduction
- 2 Model**
- 3 Specification
- 4 Decidability Results

The model

Architectures

- Communication graph ($Proc, E$)

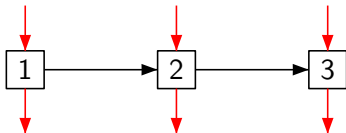


The model

Architectures

- Communication graph $(Proc, E)$
- Sets of input and output signals for each process :

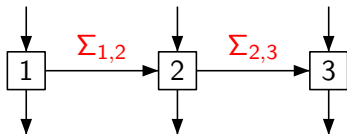
$$\bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i = \Gamma$$



The model

Architectures

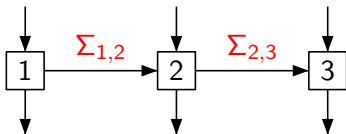
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process :
$$\bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i = \Gamma$$
- Processes **choose** sets $\Sigma_{i,j}$ for $(i,j) \in E$



The model

Architectures

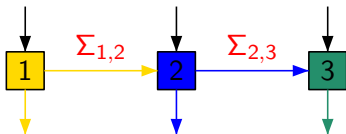
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process :
$$\bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i = \Gamma$$
- Processes **choose** sets $\Sigma_{i,j}$ for $(i,j) \in E$
- $\Sigma = \Gamma \cup \bigcup_{(i,j) \in E} \Sigma_{i,j}$



The model

Architectures

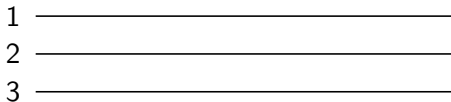
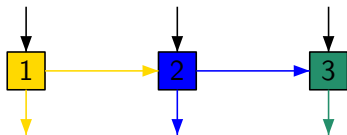
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process :
$$\bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i = \Gamma$$
- Processes **choose** sets $\Sigma_{i,j}$ for $(i,j) \in E$
- $\Sigma = \Gamma \cup \bigcup_{(i,j) \in E} \Sigma_{i,j}$
- For each process i , Σ_i is the set of signals it can send or receive, and
$$\Sigma_i^c = Out_i \cup \bigcup_{j, (i,j) \in E} \Sigma_{i,j}$$



The model: runs

Runs

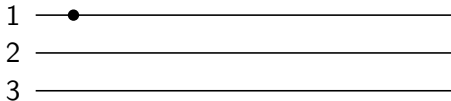
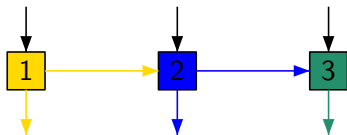
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

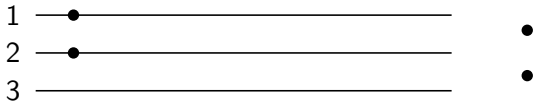
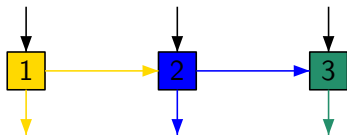
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

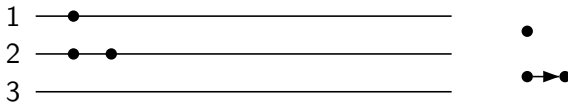
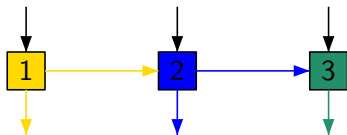
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

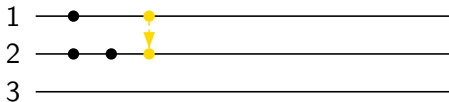
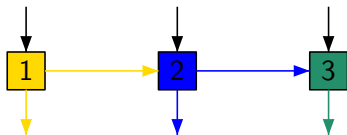
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

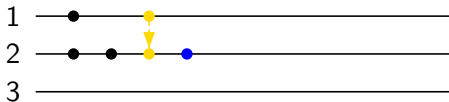
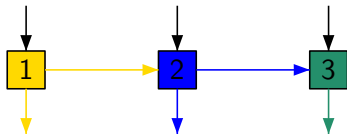
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

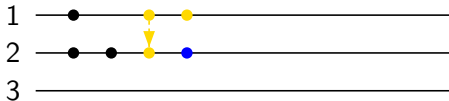
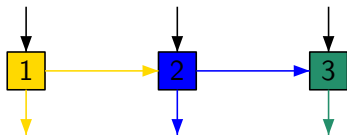
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

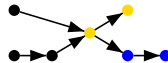
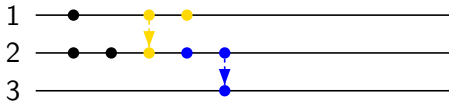
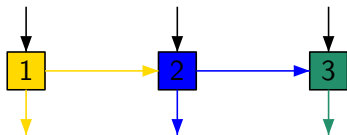
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

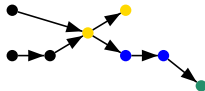
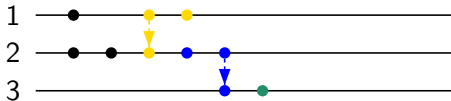
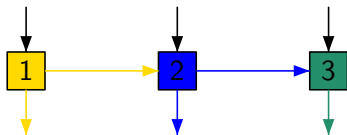
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where $a D b$ if there is a process that takes part both in a and b



The model: strategies

Strategies

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

The model: strategies

Strategies

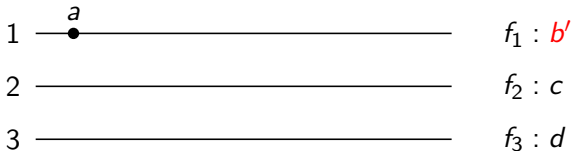
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.

1	_____	$f_1 : b$
2	_____	$f_2 : c$
3	_____	$f_3 : d$

The model: strategies

Strategies

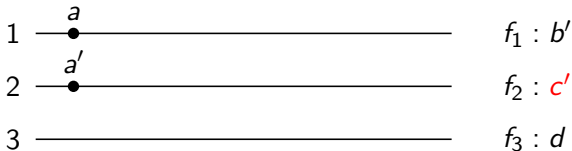
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.



The model: strategies

Strategies

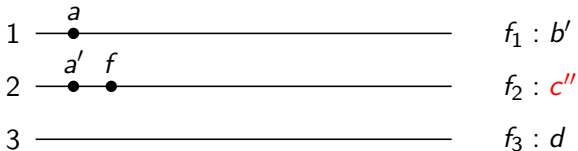
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.



The model: strategies

Strategies

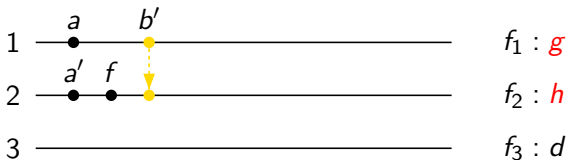
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.



The model: strategies

Strategies

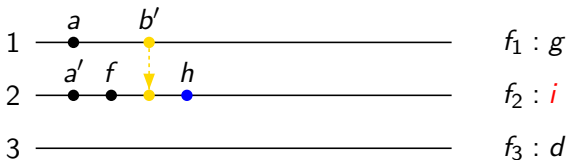
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.



The model: strategies

Strategies

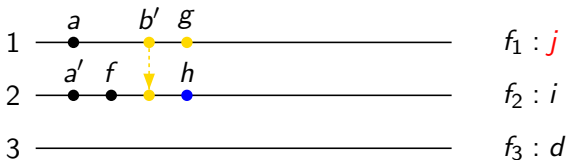
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.



The model: strategies

Strategies

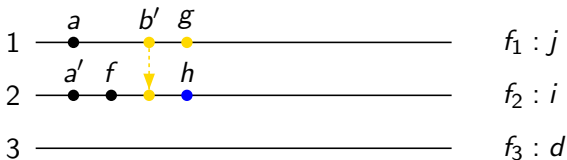
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.



The model: strategies

Strategies

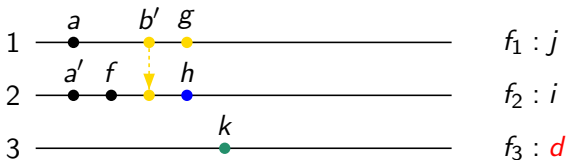
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.
- A run respects a strategy $f = (f_i)_{i \in \text{Proc}}$ (is an **f -run**) if each event of process i labelled with a controllable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



The model: strategies

Strategies

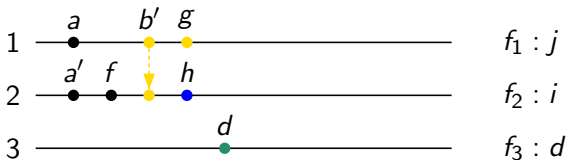
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.
- A run respects a strategy $f = (f_i)_{i \in \text{Proc}}$ (is an **f -run**) if each event of process i labelled with a controllable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



The model: strategies

Strategies

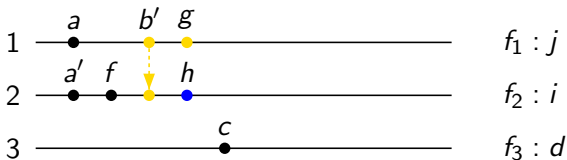
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.
- A run respects a strategy $f = (f_i)_{i \in \text{Proc}}$ (is an **f -run**) if each event of process i labelled with a controllable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



The model: strategies

Strategies

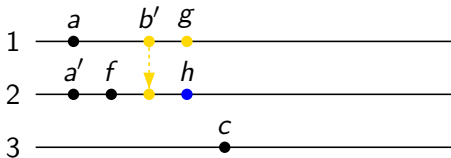
- Strategies are partial functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with **local** memory.
- Signal semantics implies **reactivity** of processes to events.
- A run respects a strategy $f = (f_i)_{i \in \text{Proc}}$ (is an **f -run**) if each event of process i labelled with a controllable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



The model: strategies

Strategies

- 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 .
- A run $t = (V, \lambda, \leq)$ is **f -maximal** if for each process i either $V_i = \lambda^{-1}(\Sigma_i)$ is infinite, or f_i is undefined on the maximal event of V_i .



The model

Observable runs

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

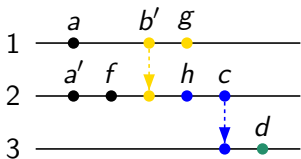
$$\pi_{\Gamma}(t) = (\Gamma, \lambda|_{\Gamma}, \leq \cap (\Gamma \times \Gamma))$$

The model

Observable runs

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

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

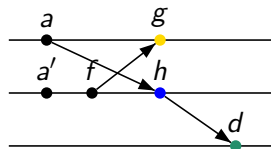
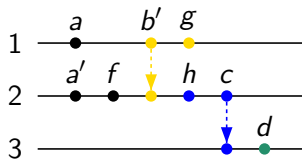


The model

Observable runs

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

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



The synthesis problem

Given

- $\mathcal{A} = (\text{Proc}, E, \Gamma)$

The synthesis problem

Given

- $\mathcal{A} = (\text{Proc}, E, \Gamma)$
- φ a specification over Γ -labelled partial orders (observable runs)

The synthesis problem

Given

- $\mathcal{A} = (\text{Proc}, E, \Gamma)$
- φ a specification over Γ -labelled partial orders (observable runs)

Do there exist

- sets $\Sigma_{i,j}$ for each $(i,j) \in E$

The synthesis problem

Given

- $\mathcal{A} = (\text{Proc}, E, \Gamma)$
- φ a specification over Γ -labelled partial orders (observable runs)

Do there exist

- sets $\Sigma_{i,j}$ for each $(i,j) \in E$
- and strategies $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ for each $i \in \text{Proc}$

The synthesis problem

Given

- $\mathcal{A} = (\text{Proc}, E, \Gamma)$
- φ a specification over Γ -labelled partial orders (observable runs)

Do there exist

- sets $\Sigma_{i,j}$ for each $(i,j) \in E$
- and strategies $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ for each $i \in \text{Proc}$

such that every f -maximal f -run t is such that $\pi_\Gamma(t) \models \varphi$?

The synthesis problem

Given

- $\mathcal{A} = (\text{Proc}, E, \Gamma)$
- φ a specification over Γ -labelled partial orders (observable runs)

Do there exist

- sets $\Sigma_{i,j}$ for each $(i,j) \in E$
- and strategies $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ for each $i \in \text{Proc}$

such that every f -maximal f -run t is such that $\pi_\Gamma(t) \models \varphi$?

If so, compute them

Outline

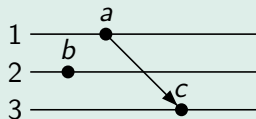
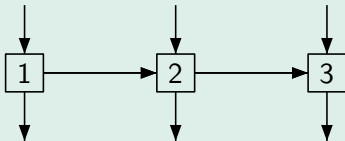
- 1 Introduction
- 2 Model
- 3 Specification**
- 4 Decidability Results

Specifications

Communication induces order relation

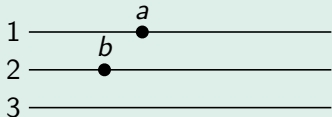
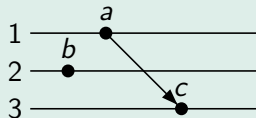
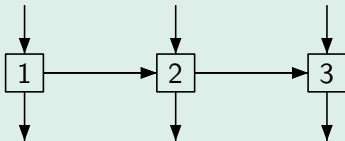
Specifications

Communication induces order relation



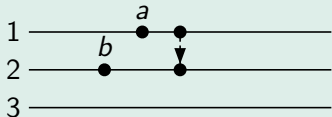
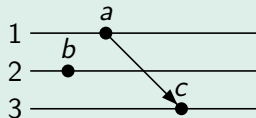
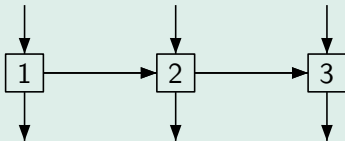
Specifications

Communication induces order relation



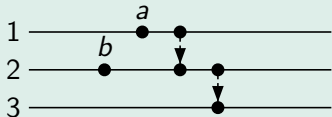
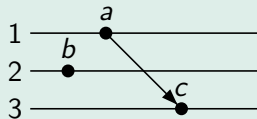
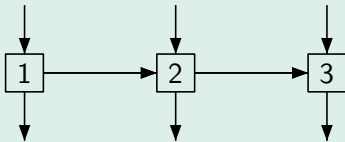
Specifications

Communication induces order relation



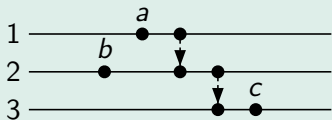
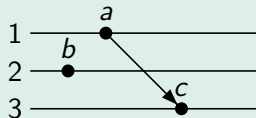
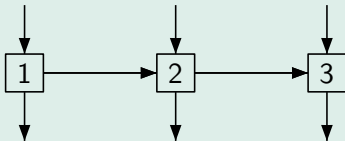
Specifications

Communication induces order relation



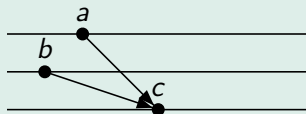
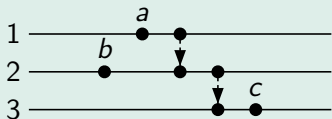
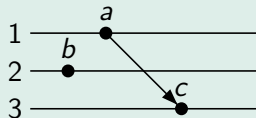
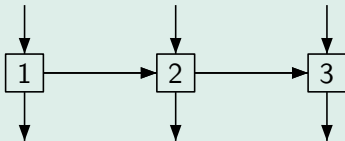
Specifications

Communication induces order relation



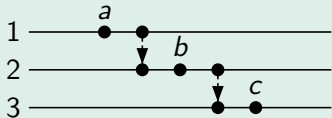
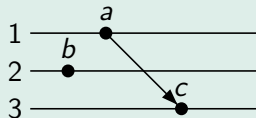
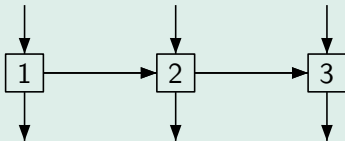
Specifications

Communication induces order relation



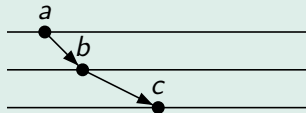
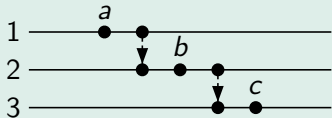
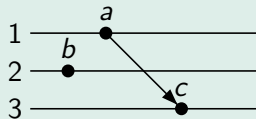
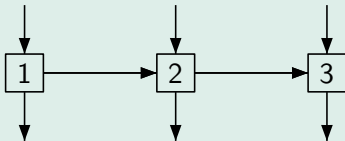
Specifications

Communication induces order relation



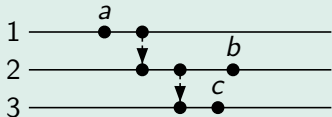
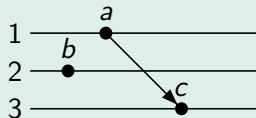
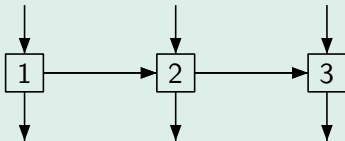
Specifications

Communication induces order relation



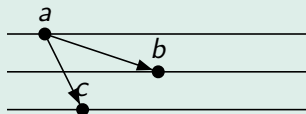
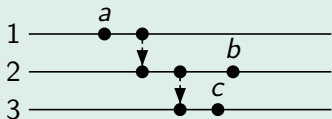
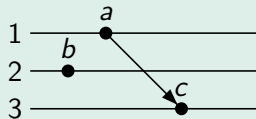
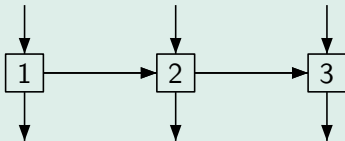
Specifications

Communication induces order relation



Specifications

Communication induces order relation



Specifications

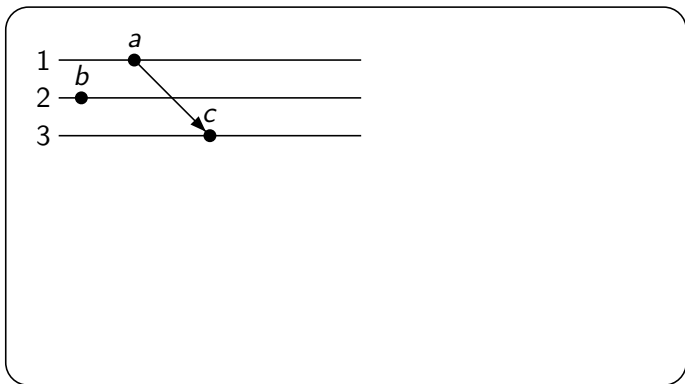
Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions

Specifications

Restrictions on specifications

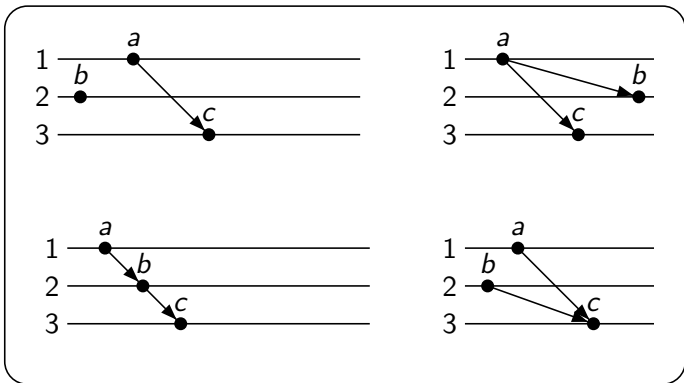
- Specifications should not discriminate between a partial order and its order extensions



Specifications

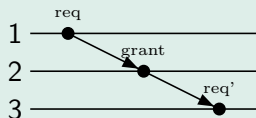
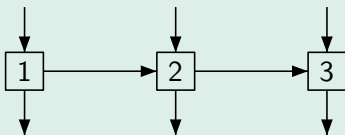
Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions



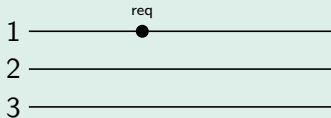
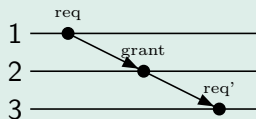
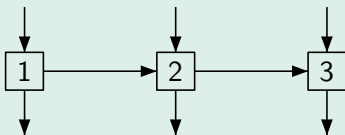
Specifications

Input events are not controllable by processes



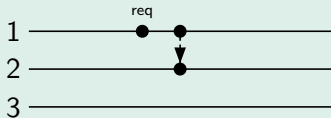
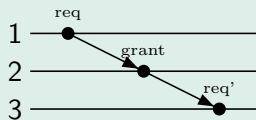
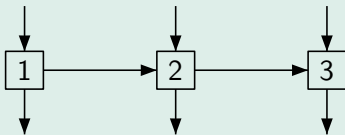
Specifications

Input events are not controllable by processes



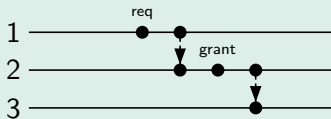
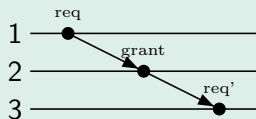
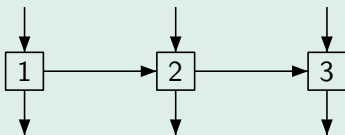
Specifications

Input events are not controllable by processes



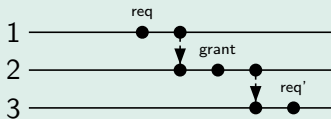
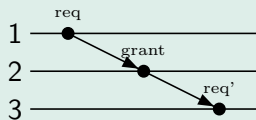
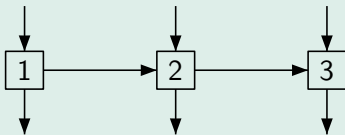
Specifications

Input events are not controllable by processes



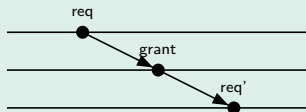
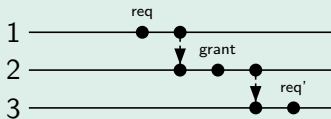
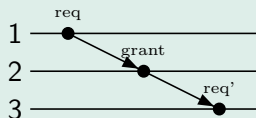
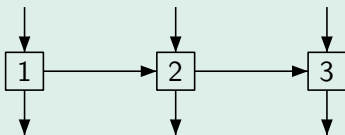
Specifications

Input events are not controllable by processes



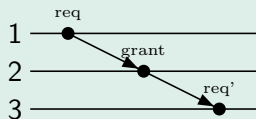
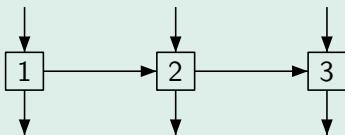
Specifications

Input events are not controllable by processes



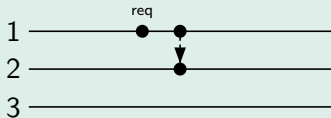
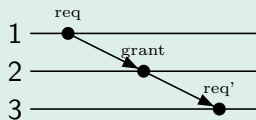
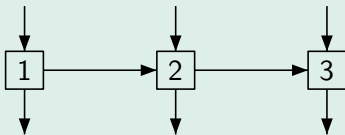
Specifications

Input events are not controllable by processes



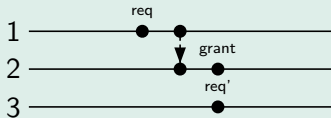
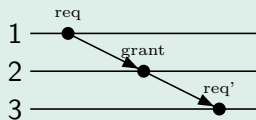
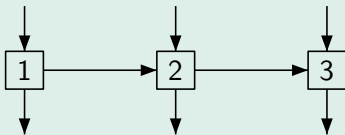
Specifications

Input events are not controllable by processes



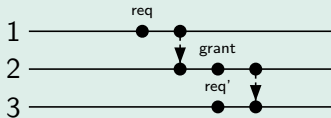
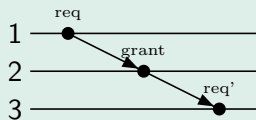
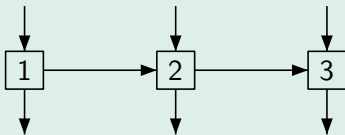
Specifications

Input events are not controllable by processes



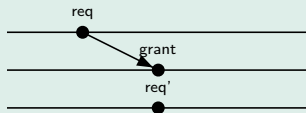
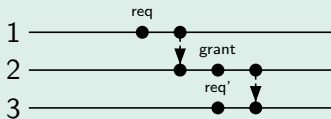
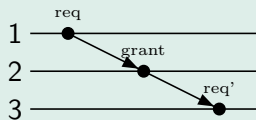
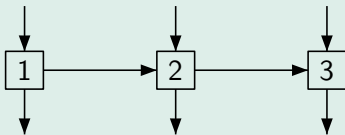
Specifications

Input events are not controllable by processes



Specifications

Input events are not controllable by processes



Specifications

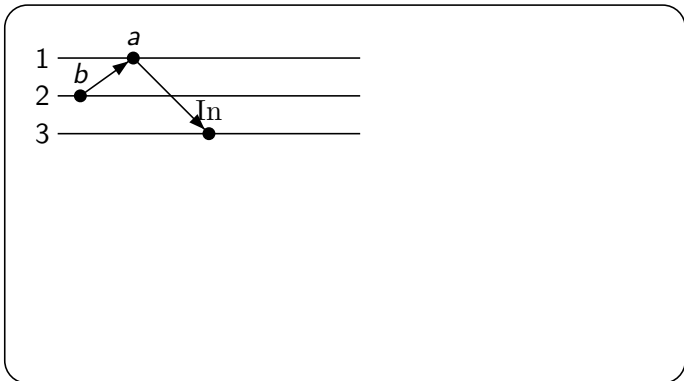
Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions
- Specifications should not discriminate between a partial order and its "weakenings"

Specifications

Restrictions on specifications

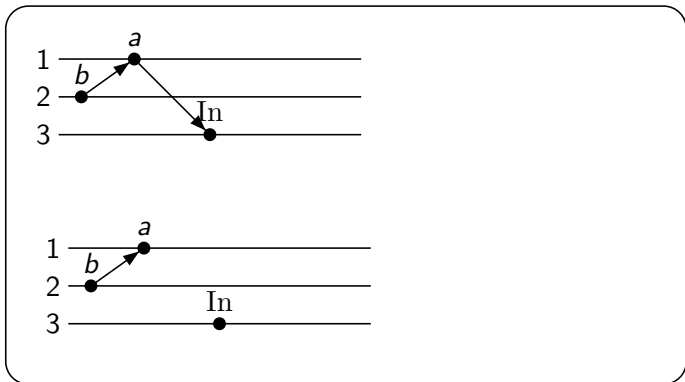
- Specifications should not discriminate between a partial order and its order extensions
- Specifications should not discriminate between a partial order and its "weakenings"



Specifications

Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions
- Specifications should not discriminate between a partial order and its "weakenings"



Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

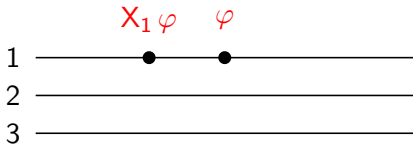
with $a \in \Gamma$ and $i, j \in \text{Proc}$

Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

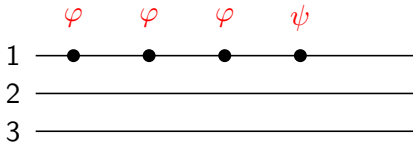


Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

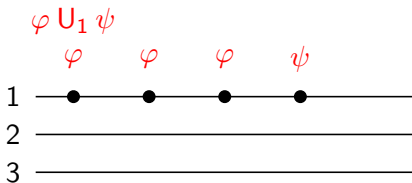


Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

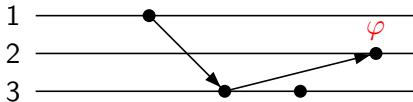


Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

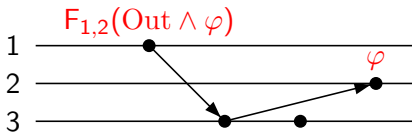


Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

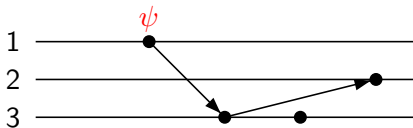


Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

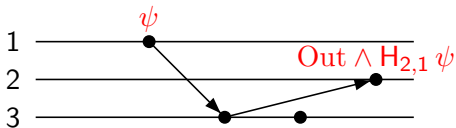


Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$



Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned} \varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi \end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

Formulae

- $G_1(\text{request} \longrightarrow F_{1,2}(\text{Out} \wedge \text{grant}))$
- $G_2(\text{grant} \longrightarrow (\text{Out} \wedge H_{2,1} \text{request}))$

Example of a logic closed by extension and weakening

AlocTL

$$\begin{aligned}\varphi ::= & a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \\ & \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \\ & \mid F_{i,j}(\text{Out} \wedge \varphi) \mid \text{Out} \wedge H_{i,j} \varphi\end{aligned}$$

with $a \in \Gamma$ and $i, j \in \text{Proc}$

Formulae

- $G_1(\text{request} \longrightarrow F_{1,2}(\text{Out} \wedge \text{grant}))$
- $G_2(\text{grant} \longrightarrow (\text{Out} \wedge H_{2,1} \text{request}))$

Theorem

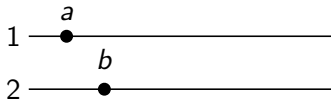
AlocTL is closed under extension and weakening

Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!

Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!



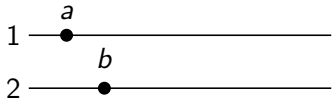
$$a \wedge \neg F_{1,2} b$$

OK

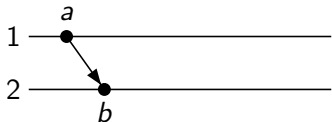
Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!

$a \wedge \neg F_{1,2} b$



OK



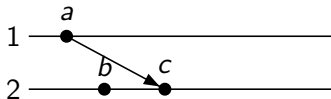
KO

Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!
- $X_{i,j} \varphi$ forbidden!

Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!
- $X_{i,j} \varphi$ forbidden!

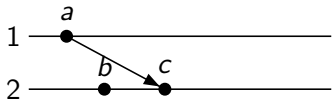


$a \wedge X_{1,2} c$

OK

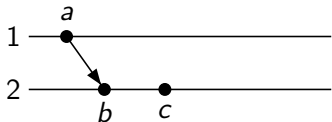
Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!
- $X_{i,j} \varphi$ forbidden!



$a \wedge X_{1,2} c$

OK



KO

Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!
- $X_{i,j} \varphi$ forbidden!

Specification is not allowed to require **concurrency**

Closure by extension

- $\neg F_{i,j} \varphi$ forbidden!
- $X_{i,j} \varphi$ forbidden!

Specification is not allowed to require **concurrency**

Closure by weakening

Ensured by $F_{i,j} \wedge \mathbf{Out}$ and $\mathbf{Out} \wedge H_{i,j} \varphi$.

Outline

- 1 Introduction
- 2 Model
- 3 Specification
- 4 Decidability Results**

Decidability Results

Theorem

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

Decidability Results

Theorem

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

Theorem

The distributed synthesis problem over strongly connected architectures is decidable for $AlocTL$ specifications.

Decidability Results

Theorem

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

Theorem

The distributed synthesis problem over strongly connected architectures is decidable for $AlocTL$ specifications.

Proof

By reduction to the singleton case.

Strongly connected architectures (2)

Proposition

If there are communication sets $\Sigma_{i,j}$ for $(i,j) \in E$ and a winning distributed strategy on the strongly connected architecture, then there is a winning strategy on the singleton.

Proof

Easy.

Strongly connected architectures

Proposition

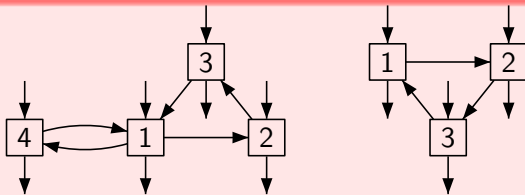
If there is a winning strategy f over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected architecture.

Strongly connected architectures

Proposition

If there is a winning strategy f over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected architecture.

Proof



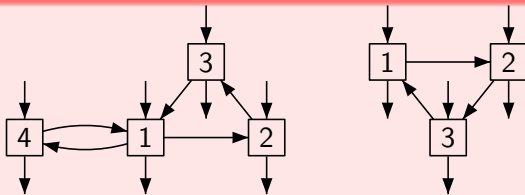
- We select a master process and a cycle.

Strongly connected architectures

Proposition

If there is a winning strategy f over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected architecture.

Proof



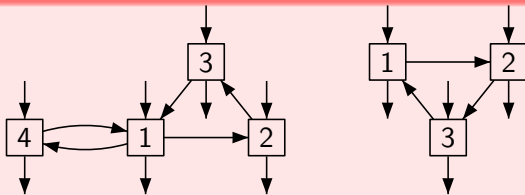
- We select a master process and a cycle.
- The master process will centralize information in order to simulate f and tell other processes which value to output

Strongly connected architectures

Proposition

If there is a winning strategy f over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected architecture.

Proof



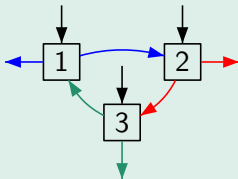
- We select a master process and a cycle.
- The master process will centralize information in order to simulate f and tell other processes which value to output
- Aim: create a run that will be a **weakening** of some f -run over the singleton

Centralize information

Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal `Msg` through the cycle

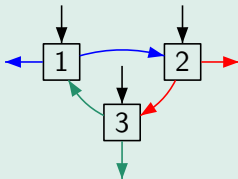
1 _____
 t : 2 _____
3 _____
 t' : _____

Centralize information

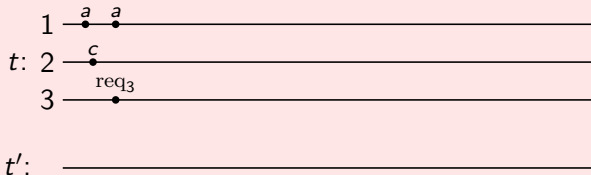
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal Msg through the cycle

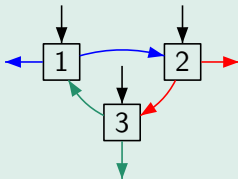


Centralize information

Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal Msg through the cycle

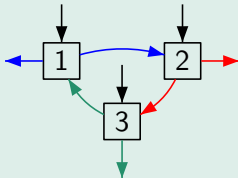


Centralize information

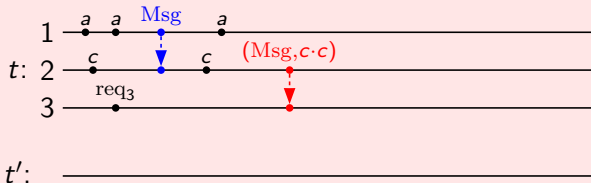
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal Msg through the cycle

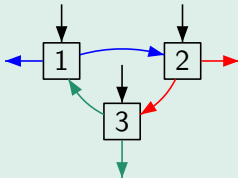


Centralize information

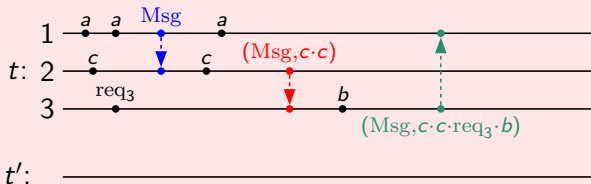
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal Msg through the cycle

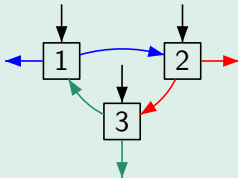


Centralize information

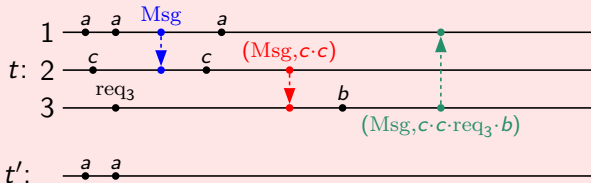
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal Msg through the cycle

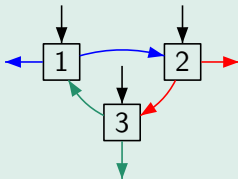


Centralize information

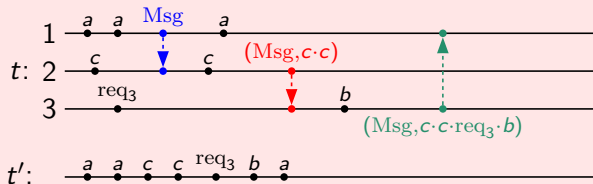
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master collect information by sending a signal Msg through the cycle

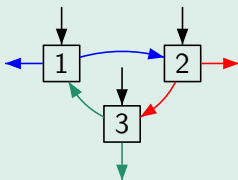


Tell processes what to output

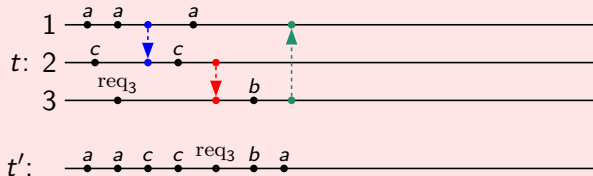
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f

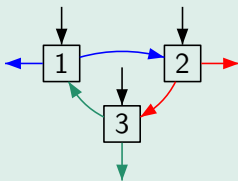


Tell processes what to output

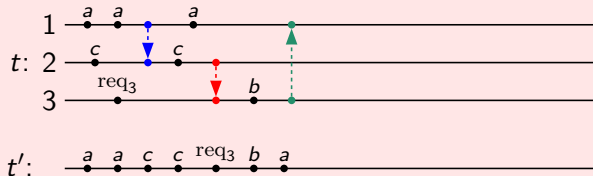
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



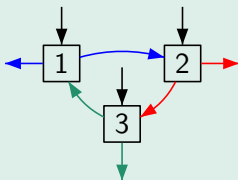
$f : \text{grant}$

Tell processes what to output

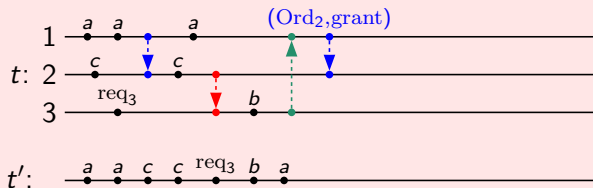
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



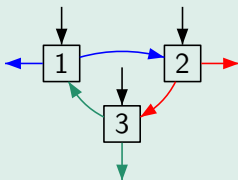
$f : \text{grant}$

Tell processes what to output

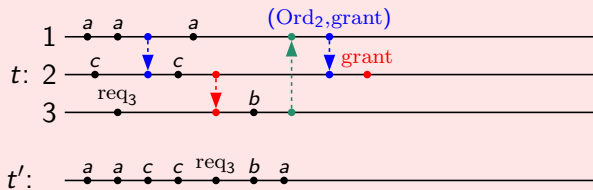
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



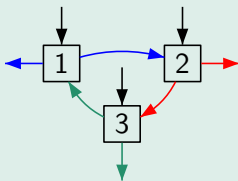
$f : \text{grant}$

Tell processes what to output

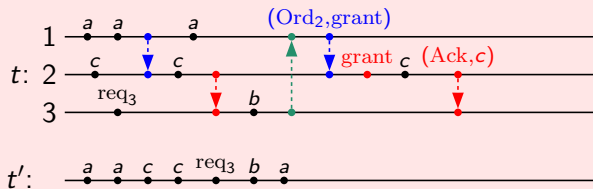
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



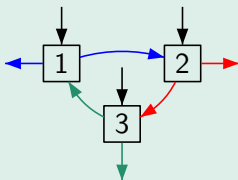
$f : \text{grant}$

Tell processes what to output

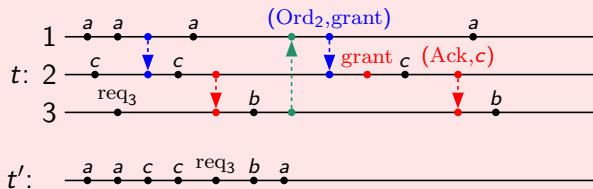
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



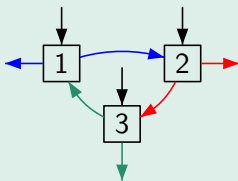
$f : \text{grant}$

Tell processes what to output

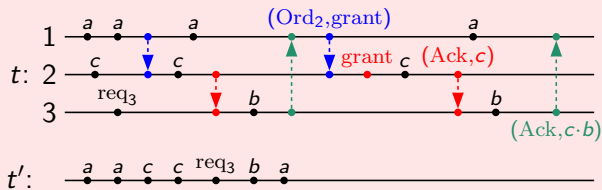
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



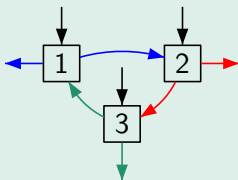
$f : \text{grant}$

Tell processes what to output

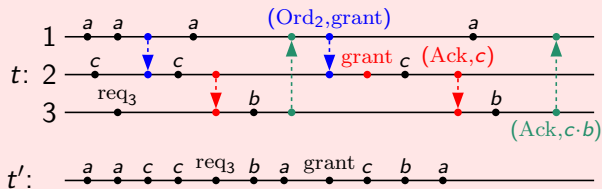
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f

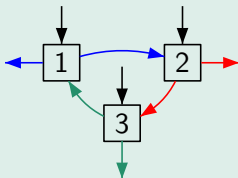


Tell processes what to output (2)

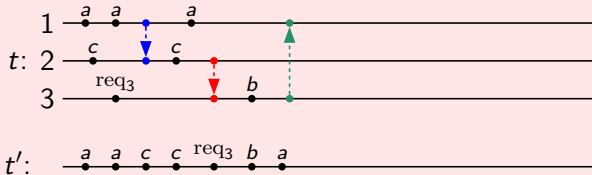
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



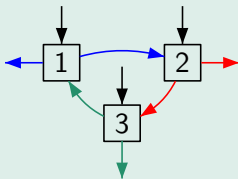
$f : \text{grant}$

Tell processes what to output (2)

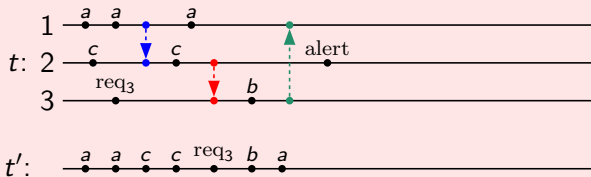
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



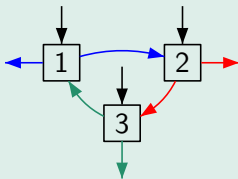
$f : \text{grant}$

Tell processes what to output (2)

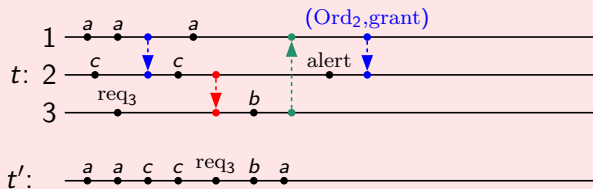
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



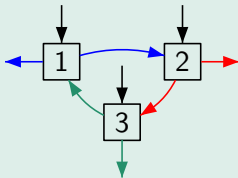
$f : \text{grant}$

Tell processes what to output (2)

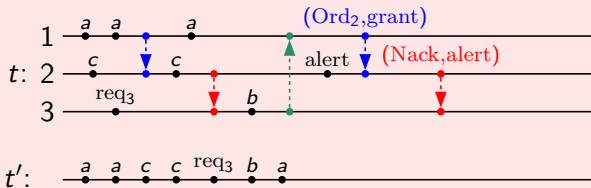
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



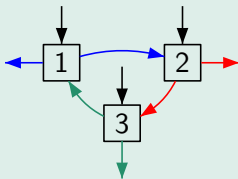
$f : \text{grant}$

Tell processes what to output (2)

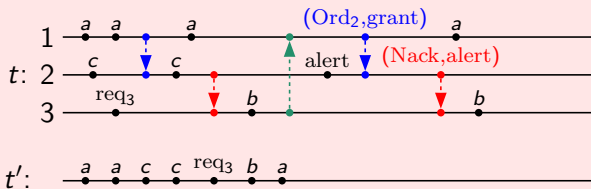
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



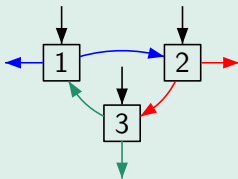
$f : \text{grant}$

Tell processes what to output (2)

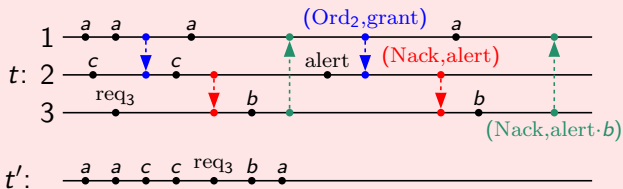
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



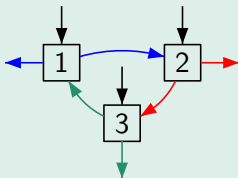
$f : \text{grant}$

Tell processes what to output (2)

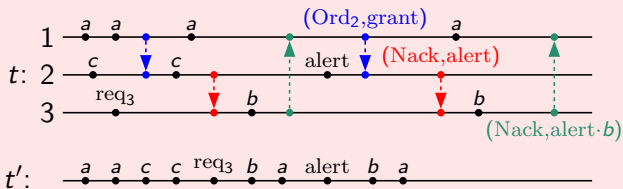
Example

Specification: $\text{req}_3 \rightarrow F_{32}(\neg Y_2 \text{ alert} \leftrightarrow \text{grant})$

Strategy for the singleton: $f(\sigma) = \text{grant}$ iff σ contains req_3 but no alert



Master sends orders to other processes to simulate the strategy f



Proof - end

Lemma

t' is an extension of $\pi_{\Gamma}(t)$.

Proof - end

Lemma

t' is an extension of $\pi_{\Gamma}(t)$.

Lemma

t' is an f -maximal f -run.

Proof - end

Lemma

t' is an extension of $\pi_{\Gamma}(t)$.

Lemma

t' is an f -maximal f -run.

Lemma

If $x <' y$ in t' and $x \parallel y$ in $\pi_{\Gamma}(t)$ then $\lambda(y) \in \text{In}$.

Proof - end

Lemma

t' is an extension of $\pi_\Gamma(t)$.

Lemma

t' is an f -maximal f -run.

Lemma

If $x <' y$ in t' and $x \parallel y$ in $\pi_\Gamma(t)$ then $\lambda(y) \in \text{In}$.

Corollary

$\pi_\Gamma(t)$ is a weakening of t' .

Proof - end

Lemma

t' is an extension of $\pi_{\Gamma}(t)$.

Lemma

t' is an f -maximal f -run.

Lemma

If $x <' y$ in t' and $x \parallel y$ in $\pi_{\Gamma}(t)$ then $\lambda(y) \in \text{In}$.

Corollary

$\pi_{\Gamma}(t)$ is a weakening of t' .

Conclusion

Then $t' \models \varphi$ and, by closure property $\pi_{\Gamma}(t) \models \varphi$.

Conclusion

- Asynchrony removes undecidability causes
- We have defined a new model of communication
- We have identified a class of decidable architectures
- Hopefully, many more to come!