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
Synthesis of a reactive system

inputs from $E$ \hspace{1cm} \text{outputs to } E

Open system $S$

Specification $\varphi$
Synthesis of a reactive system

Two problems
- 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.
Distributed synthesis

Open distributed system $S$

input of $E$  

output to $E$

$S_1$  

$S_2$  

$S_3$  

$S_4$  

Specification $\varphi$
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 $\varphi$, 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
The model

Architectures

- Communication graph \((Proc, E)\)
The model

Architectures

- Communication graph \((\text{Proc}, E)\)
- Sets of input and output signals for each process:
  \[
  \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i = \Gamma
  \]
The model

Architectures

- Communication graph \((\text{Proc}, E)\)
- Sets of input and output signals for each process:
  \[ \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i = \Gamma \]
- Processes choose sets \(\Sigma_{i,j}\) for \((i,j) \in E\)
The model

Architectures

- Communication graph \((\text{Proc}, E)\)
- Sets of input and output signals for each process:
  \[ \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i = \Gamma \]
- 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 \((\text{Proc}, E)\)
- Sets of input and output signals for each process:
  \[
  \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i = \Gamma
  \]
- 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 \(i\), \(\Sigma_i\) is the set of signals it can send or receive, and
  \[
  \Sigma_i^c = \text{Out}_i \cup \bigcup_{j,(i,j) \in E} \Sigma_{i,j}
  \]
A run is a Mazurkiewicz trace \( t = (V, \lambda, \leq) \) over \((\Sigma, D)\) where \( a D b \) if there is a process that takes part both in \( a \) and \( b \)
A run is a Mazurkiewicz trace \( t = (V, \lambda, \leq) \) over \((\Sigma, D)\) where \( a D b \) if there is a process that takes part both in \( a \) and \( b \).
The model: runs

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

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

A run is a Mazurkiewicz trace $t = (V, \lambda, \preceq)$ over $(\Sigma, D)$ where $a D b$ if there is a process that takes part both in $a$ and $b$
A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over $(\Sigma, D)$ where $a D b$ if there is a process that takes part both in $a$ and $b$.
The model: runs

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

```
1  a
   ---------------------
   f_1 : b'

2
   ----------------------------------
   f_2 : c

3
   ----------------------------------
   f_3 : d
```
The model: 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.

1. $a$  
   $a' \ f$  
   $f_1 : b'$

2. $f$  
   $f_2 : c''$

3. $f_3 : d$
The model: 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.

1. $a \quad b'$
2. $a' \quad f \quad h$
3. $f_1 : g$
4. $f_2 : i$
5. $f_3 : d$
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 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 $f_i$. 

\begin{itemize}
  \item 1 \hspace{2cm} a \hspace{1cm} b' \hspace{1cm} g \\
  \item 2 \hspace{2cm} a' \hspace{1cm} f \hspace{1cm} h \\
  \item 3 \hspace{2cm} f_1 : j \\
  \item 4 \hspace{2cm} f_2 : i \\
  \item 5 \hspace{2cm} f_3 : d
\end{itemize}
The model: strategies

Strategies

- Strategies are partial functions $f_i : \Sigma_i^* \to \Sigma_i^c$ 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 $f_i$. 

![Diagram of strategies and processes](image-url)
The model: 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 $f_i$. 

```
1    a  b'  g
     \__________
2    a'  f  h
     \__________
3    d
```

$f_1 : j$

$f_2 : i$

$f_3 : d$
The model: 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 \( f_i \).
Strategies

- Strategies are partial functions $f_i : \Sigma^*_i \rightarrow \Sigma^c_i$ 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 $V_i$. 

Diagram:

```
1    a   b'   g
    \downarrow
a'   f
2    \downarrow
    h
    \downarrow
3    c
```
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

$$\pi_{\Gamma}(t) = (\Gamma, \lambda|_{\Gamma}, \leq \cap (\Gamma \times \Gamma))$$
Given a run $t = (V, \lambda, \leq)$, we define the observable run by

$$\pi_t(t) = (\Gamma, \lambda|_\Gamma, \leq \cap (\Gamma \times \Gamma))$$
The synthesis problem

Given

\[ A = (\text{Proc}, E, \Gamma) \]
The synthesis problem

Given

- $A = (\text{Proc}, E, \Gamma)$
- $\varphi$ a specification over $\Gamma$-labelled partial orders (observable runs)
The synthesis problem

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

Do there exist
- sets $\Sigma_{i,j}$ for each $(i, j) \in E$
The synthesis problem

Given
- $A = (\text{Proc}, E, \Gamma)$
- $\varphi$ a specification over $\Gamma$-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
- \( A = (\text{Proc}, E, \Gamma) \)
- \( \varphi \) a specification over \( \Gamma \)-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

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

Do there exist

- sets $\Sigma_{i,j}$ for each $(i,j) \in E$
- and strategies $f_i : \Sigma^*_i \to \Sigma^c_i$ 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
Communication induces order relation
Specifications

Communication induces order relation

Diagram:

1 \rightarrow 2 \rightarrow 3

1

2

3

1 

2

3

1

2

3

a

b

c
Specifications

Communication induces order relation

\[
\begin{array}{c}
\text{1} \quad \text{2} \quad \text{3} \\
\text{1} \quad \text{2} \quad \text{3} \\
\text{1} \quad \text{2} \quad \text{3}
\end{array}
\]
Communication induces order relation

Specifications
Specifications

Communication induces order relation

1 → 2 → 3

1 → a
2 → b → 3
2 → c
3
Specifications

Communication induces order relation
Specifications

Communication induces order relation

1 -> 2 -> 3

1  2  3

1  b  2  c  3

1  a

1  2  3
Specifications

Communication induces order relation

1 -> 2 -> 3

1  2  3

1  2  3

1  2  3

1  2  3

1  2  3

1  2  3

1  2  3

1  2  3

1  2  3
Specifications

Communication induces order relation

Diagram: [Diagram showing the order relation with nodes and arrows]
Specifications

Communication induces order relation

\[
\begin{align*}
&\begin{array}{c}
1 \\
\downarrow \\
\downarrow \\
2 \\
\downarrow \\
\downarrow \\
3
\end{array} \\
\begin{array}{c}
a \\
\downarrow \\
b \\
\downarrow \\
c
\end{array} \\
\begin{array}{c}
1 \\
\downarrow \\
2 \\
\downarrow \\
3
\end{array}
\end{align*}
\]

\[
\begin{align*}
&\begin{array}{c}
a \\
\downarrow \\
b \\
\downarrow \\
c
\end{array} \\
\begin{array}{c}
1 \\
\downarrow \\
2 \\
\downarrow \\
3
\end{array}
\end{align*}
\]
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

![Diagram showing a partial order with points a, b, and c, and edges between them.](image)
Specifications

Restrictions on specifications

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

Input events are not controllable by processes

1 → 2 → 3
Specifications

Input events are not controllable by processes

1 req
2
3

1
2
3

1 req
2 grant
3 req'
Input events are not controllable by processes
Specifications

Input events are not controllable by processes

1. req
2. grant
3. req'
Specifications

Input events are not controllable by processes

1
2
3

1
2
3

req
grant
req'
Specifications

Input events are not controllable by processes

![Diagram showing the flow of requests and grants between processes 1, 2, and 3. The diagram illustrates the non-controllability of input events by processes.]
Input events are not controllable by processes
Input events are not controllable by processes
Specifications

Input events are not controllable by processes

1. req
2. grant
3. req'

Diagram:

- Process 1
- Process 2
- Process 3
- req
- grant
- req'

1. req
2. grant
3. req'
Input events are not controllable by processes
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 "weakensings".
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

<table>
<thead>
<tr>
<th>AlocTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi ::= a</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(</td>
</tr>
</tbody>
</table>

with \( a \in \Gamma \) and \( i, j \in \text{Proc} \)
Example of a logic closed by extension and weakening

**AlocTL**

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

\[ \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \]

\[ \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \]

\[ \mid F_{i,j}(\text{Out} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi \]

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

1

2

3

\( X_1 \varphi \quad \varphi \)
Example of a logic closed by extension and weakening

### AlocTL

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

\[ \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \]

\[ \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \]

\[ \mid F_{i,j}(\text{Out} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi \]

with \( a \in \Gamma \) and \( i, j \in \text{Proc} \)
Example of a logic closed by extension and weakening

AlocTL

\( \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \)

\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} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi

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

\( \varphi \ U_1 \psi \)

1

2

3
Example of a logic closed by extension and weakening

**AlocTL**

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \Tilde{U}_i \varphi \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \Tilde{S}_i \varphi \mid F_{i,j}(\text{Out} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi \]

with \( a \in \Gamma \) and \( i, j \in \text{Proc} \)
Example of a logic closed by extension and weakening

\textbf{AlocTL}

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

\[ \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \]

\[ \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \]

\[ \mid F_{i,j} (\text{Out} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi \]

with \( a \in \Gamma \) and \( i, j \in \text{Proc} \)
Example of a logic closed by extension and weakening

**AlocTL**

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \]
\[ \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \]
\[ \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \]
\[ \mid F_{i,j}(\text{Out} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi \]

with \( a \in \Gamma \) and \( i,j \in \text{Proc} \)
Example of a logic closed by extension and weakening

**AlocTL**

\[ \varphi ::= a | \neg a | \varphi \lor \varphi | \varphi \land \varphi \]

\[ | X_i \varphi | \varphi U_i \varphi | \neg X_i \top | \varphi \tilde{U}_i \varphi \]

\[ | Y_i \varphi | \varphi S_i \varphi | \neg Y_i \top | \varphi \tilde{S}_i \varphi \]

\[ | F_{i,j}(\text{Out} \land \varphi) | \text{Out} \land H_{i,j} \varphi \]

with \( a \in \Gamma \) and \( i, j \in \text{Proc} \)
Example of a logic closed by extension and weakening

**AllocTL**

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

\[ \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \]

\[ \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \]

\[ \mid F_{i,j}(\text{Out} \land \varphi) \mid \text{Out} \land H_{i,j} \varphi \]

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

**Formulae**

- \( G_1(\text{request} \rightarrow F_{1,2}(\text{Out} \land \text{grant})) \)
- \( G_2(\text{grant} \rightarrow (\text{Out} \land H_{2,1} \text{request})) \)
Example of a logic closed by extension and weakening

AlocTL

\[ \varphi ::= a \mid \neg a \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

\[ \mid X_i \varphi \mid \varphi U_i \varphi \mid \neg X_i \top \mid \varphi \tilde{U}_i \varphi \]

\[ \mid Y_i \varphi \mid \varphi S_i \varphi \mid \neg Y_i \top \mid \varphi \tilde{S}_i \varphi \]

\[ \mid F_{i,j}(Out \land \varphi) \mid Out \land H_{i,j} \varphi \]

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

Formulae

- \( G_1(\text{request} \rightarrow F_{1,2}(Out \land \text{grant})) \)
- \( G_2(\text{grant} \rightarrow (Out \land H_{2,1} \text{request})) \)

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!

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

OK
Closure by extension

\[ \neg F_{i,j} \varphi \text{ forbidden!} \]

\[ a \land \neg F_{1,2} b \]

1. 1 $\rightarrow a$
   2 $\rightarrow b$

   OK

2. 1 $\rightarrow a$
   2 $\rightarrow b$

   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!

\[ a \land X_{1,2} c \]

OK
Closure by extension

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

For the given example:

- \( a \land X_{1,2} c \)
  - OK
- \( a \land X_{1,2} c \)
  - 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 $F_{i,j} \land \text{Out}$ and $\text{Out} \land H_{i,j} \varphi$. 
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 \textit{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.
**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.
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.
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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master collect information by sending a signal \( \text{Msg} \) through the cycle

\[
\begin{align*}
1 & \quad 2 \\
\text{t} & \quad 3 \\
\text{t}' &
\end{align*}
\]
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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master collect information by sending a signal \( \text{Msg} \) through the cycle

\[
\begin{align*}
&1 \quad a \quad a \\
&\text{t:} \quad 2 \quad c \\
&\quad \quad \text{req}_3 \\
&3 \\
&t':
\end{align*}
\]
Centralize information

Example
Specification: \( \text{req}_3 \rightarrow F_{32}(\neg \text{Y}_2 \text{ alert} \leftrightarrow \text{grant}) \)
Strategy for the singleton: \( f(\sigma) = \text{grant} \) iff \( \sigma \) contains \( \text{req}_3 \) but no alert

Master collect information by sending a signal \( \text{Msg} \) through the cycle

\[
\begin{align*}
1 & \quad a & \quad a & \quad \text{Msg} \\
2 & \quad c & & \\
3 & \quad \text{req}_3 & & \\
\end{align*}
\]

\( t: \)  
\( t': \)
Centralize information

Example

Specification: \( \text{req}_3 \rightarrow \text{F}_{32}(\neg \text{Y}_2 \text{ alert} \leftrightarrow \text{grant}) \)

Strategy for the singleton: \( f(\sigma) = \text{grant} \) iff \( \sigma \) contains \( \text{req}_3 \) but no \text{alert}

Master collect information by sending a signal \text{Msg} through the cycle

\[
\begin{align*}
1 & \quad a \quad a \quad \text{Msg} \\
2 & \quad c \quad c \quad (\text{Msg}, c \cdot c) \\
3 & \quad \text{req}_3 \\
\end{align*}
\]
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 $\sigma$ contains $\text{req}_3$ but no alert

Master collect information by sending a signal $\text{Msg}$ 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 $\sigma$ contains $\text{req}_3$ but no alert

Master collect information by sending a signal $\text{Msg}$ through the cycle

$t$: 1
   a a a
   c c (Msg, c·c)
   req3 b (Msg, c·c·req3·b)

$t'$: a a
Centralize information

**Example**

**Specification:** \( req_3 \rightarrow F_{32}(\neg Y_2 \text{ alert } \leftrightarrow \text{ grant}) \)

**Strategy for the singleton:** \( f(\sigma) = \text{ grant iff } \sigma \text{ contains req}_3 \text{ but no alert} \)

Master collect information by sending a signal \( \text{Msg} \) through the cycle

\[
\begin{align*}
&\text{1} & \text{a} & \text{a} & \text{Msg} & \text{a} & \text{a} \\
&\text{2} & \text{c} & \text{c} & & \text{c} & \text{c} \\
&\text{3} & \text{req}_3 & & \text{b} & \text{b} & \text{b} \\
&\text{t':} & \text{a} & \text{a} & \text{c} & \text{c} & \text{req}_3 & \text{b} & \text{b} & \text{a} & \text{a} & \text{a} & \text{a}
\end{align*}
\]
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: \( f(\sigma) = \text{grant} \text{ iff } \sigma \text{ contains req}_3 \text{ but no alert} \)

Master sends orders to other processes to simulate the strategy \( f \)

\[
\begin{align*}
\text{t:} & \quad 1 & 2 & 3 \\
\text{t':} & \quad a & a & c & c & \text{req}_3 & b & a \\
\end{align*}
\]
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 $\sigma$ contains $\text{req}_3$ but no alert

Master sends orders to other processes to simulate the strategy $f$

$t$: $a \ a \ c \ \triangledown \ c \ \text{req}_3 \ b \ a$

$t'$: $a \ a \ c \ c \ \text{req}_3 \ b \ a$

$f$ : 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 \( \sigma \) contains \( \text{req}_3 \) 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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master sends orders to other processes to simulate the strategy \( f \)

\[
\begin{align*}
\text{t:} & & 1 & a & a & a & (\text{Ord}_2, \text{grant}) \\
& & 2 & c & i & c & \text{grant} \\
& & 3 & \text{req}_3 & b & \text{a} \\
\text{t':} & & a & a & c & c & \text{req}_3 & b & \text{a} & f : \text{grant}
\end{align*}
\]
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 $\sigma$ contains $\text{req}_3$ but no alert

Master sends orders to other processes to simulate the strategy $f$

\[ t: \begin{array}{ccccccc}
1 & a & a & a & (\text{Ord}_2, \text{grant}) \\
2 & c & i & c & \text{grant} & c & (\text{Ack}, c) \\
3 & \text{req}_3 & b & \text{grant} & c & \text{grant} & \text{grant} \\
\end{array} \]

\[ t': \begin{array}{ccccccc}
 a & a & c & c & \text{req}_3 & b & a \\
\end{array} \]

$f : \text{grant}$
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: \( f(\sigma) = \text{grant} \text{ iff } \sigma \text{ contains req}_3 \text{ but no alert} \)

Master sends orders to other processes to simulate the strategy \( f \)

\( t: 2 \)
\( a \quad a \quad a \quad c \quad c \quad b \quad (\text{Ord}_2, \text{grant}) \)
\( t': 3 \)
\( 1 \quad a \quad a \quad a \quad (\text{Ord}_2, \text{grant}) \)
\( 2 \quad a \quad a \quad a \quad b \quad (\text{Ack}, c) \)
\( 3 \quad 1 \quad 2 \quad 3 \quad \text{req}_3 \quad b \quad a \quad f: \text{grant} \)
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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master sends orders to other processes to simulate the strategy \( f \)

\( t: \frac{\text{a a a}}{\text{c c c}} \)

\( t': \frac{\text{a a c c req}_3 b a}{\text{a a c c req}_3 b a} \quad f: \text{grant} \)
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: \( f(\sigma) = \text{grant} \text{ iff } \sigma \text{ contains req}_3 \text{ but no alert} \)

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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master sends orders to other processes to simulate the strategy \( f \)

\[
\begin{array}{c}
\text{t:} & 1 & a & a & a & c & c & c & \text{req}_3 & b & a \\
\text{t':} & a & a & c & c & \text{req}_3 & b & a & f : \text{grant} \\
\text{1} & \text{2} & \text{3} & \\
& 1 & 2 & 3
\end{array}
\]
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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master sends orders to other processes to simulate the strategy \( f \)

\[
\begin{align*}
\text{t': } & a \quad a \quad c \quad c \quad \text{req}_3 \quad b \quad a \\
\end{align*}
\]

\( f : \text{grant} \)
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 \( \sigma \) contains \( \text{req}_3 \) 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 \( \sigma \) contains \( \text{req}_3 \) but no alert

Master sends orders to other processes to simulate the strategy \( f \)

\[ t: \begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & (\text{Ord}_2,\text{grant}) \\
2 & c & c & c & c & \text{alert} \quad (\text{Nack},\text{alert}) \\
3 & \text{req}_3 & b & b & b & \\
\end{array} \]

\[ t': \begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 & \text{req}_3 \\
2 & c & c & c & c & \text{alert} \quad (\text{Nack},\text{alert}) \\
3 & b & b & b & b & \\
\end{array} \]

\( f : \text{grant} \)
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 $\sigma$ contains $\text{req}_3$ but no alert

Master sends orders to other processes to simulate the strategy $f$

$\text{Master sends orders to other processes to simulate the strategy } f$

$\text{Master sends orders to other processes to simulate the strategy } f$
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 \( \sigma \) contains \( \text{req}_3 \) 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: \( f(\sigma) = \text{grant} \text{ iff } \sigma \text{ contains } \text{req}_3 \text{ but no alert} \)

Master sends orders to other processes to simulate the strategy \( f \)
Lemma

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

t′ is an extension of $\pi_I(t)$.

### Lemma

t′ is an f-maximal f-run.
**Proof - end**

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t'$ is an extension of $\pi_\Gamma(t)$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t'$ is an $f$-maximal $f$-run.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $x &lt;' y$ in $t'$ and $x \parallel y$ in $\pi_\Gamma(t)$ then $\lambda(y) \in \text{In}$.</td>
</tr>
</tbody>
</table>
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 $t'$.
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!