How to get decidability of distributed synthesis?

Paul Gastin
Joint work with Thomas Chatain and Nathalie Sznajder

March 12, 2009
Séminaire Bordeaux
Outline

1. Introduction
2. Model
3. Specification
4. Decidability Results
Synthesis of a reactive system

inputs from $E$  outputs to $E$

Open system $S$

Specification $\phi$
Synthesis of a reactive system

Two problems

1. Decide whether there exists a program st. \( P \parallel E \models \varphi, \quad \forall E \).
2. Synthesis: If so, compute such a program.

For reasonable systems and specifications, the problems are decidable.
Distributed synthesis

input of $E$  

output to $E$

Open distributed system $S$

$S_1$  

$S_2$  

$S_3$  

$S_4$

Specification $\varphi$
Distributed synthesis

Two problems

- Decide the existence of a distributed program such that their joint behavior $P_1 \parallel P_2 \parallel P_3 \parallel P_4 \parallel E$ satisfies $\varphi$, for all $E$.
- Synthesis: If it exists, compute such a distributed program.
Distributed synthesis: Undecidable in general!? 

**Synchronous semantics: Introduced by Pnueli Rosner ’90**

- At each tick of a global clock, all processes and the environment output their new value
Distributed synthesis: Undecidable in general!? 

Synchronous semantics: Introduced by Pnueli Rosner ’90

- At each tick of a global clock, all processes and the environment output their new value
- Undecidable with global specifications.
Distributed synthesis: Undecidable in general!?

Synchronous semantics: Introduced by Pnueli Rosner ’90

- At each tick of a global clock, all processes and the environment output their new value.
- Undecidable with global specifications.
- Undecidable with constraints on internal channels.
Distributed synthesis: Undecidable in general!? 

Synchronous semantics: Introduced by Pnueli Rosner ’90

- At each tick of a global clock, all processes and the environment output their new value.
- Undecidable with global specifications.
- Undecidable with constraints on internal channels.
- Undecidable with bandwidth constraints.

\[ P_0 \quad P_1 \]
Distributed synthesis: Undecidable in general!?

**Synchronous semantics: Introduced by Pnueli Rosner ‘90**

- At each tick of a global clock, all processes and the environment output their new value
- Undecidable with global specifications.
- Undecidable with constraints on internal channels.
- Undecidable with bandwidth constraints.
- Decidable for some architectures, e.g., pipelines.
Asynchronous semantics

Behaviors are Mazurkiewicz traces
Players = controllable actions
Causal memory
Specification : regular over Mazurkiewicz traces
Asynchronous semantics

Behaviors are Mazurkiewicz traces
Players = controllable actions
Causal memory
Specification : regular over Mazurkiewicz traces

Theorem
Synthesis problem is decidable for co-graph dependence alphabets, i.e., for series-parallel systems.
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.
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,
Decidability Results

Theorem

Synthesis problem is decidable for
- strongly-connected architectures,
- disjoint unions of decidable architectures.
Outline

1. Introduction
2. Model
3. Specification
4. Decidability Results
The model

Architectures

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

Architectures

- Communication graph \((Proc, E)\)
- For each process \(i\), sets \(In_i\) and \(Out_i\) of input and output signals:
  \[
  \Gamma = \bigcup_{i \in Proc} In_i \cup \bigcup_{i \in Proc} Out_i
  \]
The model

Architectures

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

Architectures

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

- For each process \(i\), sets \(\text{In}_i\) and \(\text{Out}_i\) of input and output signals:
  \[
  \Gamma = \bigcup_{i \in Proc} \text{In}_i \cup \bigcup_{i \in Proc} \text{Out}_i
  \]

- 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 \(i\),
  \[
  \Sigma_i^c = \text{Out}_i \cup \bigcup_{j, (i,j) \in E} \Sigma_{i,j}
  \]
  is the set of signals it can send (control),

  \[
  \Sigma_i = \text{In}_i \cup \Sigma_i^c
  \]
  is its alphabet.
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

**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 $b$.
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: strategies

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

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. ____________________________ $f_1 : b$
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.

1. $a \quad f_1 : b'$
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 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.
Strategies

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

\[
\begin{align*}
1 & \quad a & b' \\
2 & \quad a' & a' & h \\
3 & \quad f_1 : g & f_2 : i & f_3 : d
\end{align*}
\]
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  b'  g
   a'  a'  h
  f1 : j

2  a  b'  g
   a'  a'  h
  f2 : i

3  a  b'  g
   a'  a'  h
  f3 : 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 $f_i$. 

![Diagram](image-url)
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 $V_i$.

```
1  a      b'  g  a
 a' a'    h  a'
2  a'' a''
3  a'' a''
```

$f_1 : \bot$

$f_2 : \bot$

$f_3 : \bot$
The model

Observable runs

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

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

where

$$\Gamma = \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i$$
Given a run $t = (V, \lambda, \leq)$, we define the observable run by

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

where

$$\Gamma = \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i$$
The model

Observable runs

Given a run \( t = (V, \lambda, \leq) \), we define the \textit{observable} run by

\[
\pi_\Gamma(t) = (\Gamma, \lambda|_\Gamma, \leq \cap (\Gamma \times \Gamma))
\]

where

\[
\Gamma = \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i
\]
The synthesis problem

Given

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

Given

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

Given

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

Do there exist
- sets \( \Sigma_{i,j} \) for each \( (i,j) \in E \)
- 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 \( f_i : \Sigma_i^* \rightarrow \Sigma_i^c \) 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

1 2 3
1 2 3

1 2 3
Specifications

Communication induces order relation

1
2
3

b
a
c

1
2
3

a
b

1
2
3

Specifications
Communication induces order relation
Specifications

Communication induces order relation
Communication induces order relation
Specifications

Communication induces order relation
Specifications

Communication induces order relation

1 2 3

\[ \begin{array}{c}
1 \\
2 \\
3
\end{array} \quad \begin{array}{c}
a \\
b \\
c
\end{array} \]
Specifications

Communication induces order relation
Specifications

Communication induces order relation

Diagram: 1 → 2 → 3
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
Input events are not controllable by processes
Specifications

Input events are not controllable by processes

![Diagram showing input events and 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

Diagram showing the sequence of events from 1 to 3 with input and grant actions, indicating the uncontrollable nature of input events.
Input events are not controllable by processes

Specifications
Specifications

Input events are not controllable by processes

1
2
3

req
grant
req'

1
2
3

req

1
2
3

---

1
2
3

---
Specifications

Input events are not controllable by processes
Specifications

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 "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

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

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

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

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

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**

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

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

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

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

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 \( i, j \in \text{Proc} \)

---

**Diagram**

```plaintext
\( \varphi U_1 \psi \)

1
\[ \bullet \quad \bullet \quad \bullet \quad \bullet \]

2
\[ \text{--------------------------} \]

3
\[ \text{--------------------------} \]
```
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**

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

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

**AlocTL**

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

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

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

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

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

---

1
2
3

\( \psi \)
Example of a logic closed by extension and weakening

```
AlocTL

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

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

**AlocTL**

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

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

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

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

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

**Formulae**

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

**AlocTL**

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

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

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

**Formulae**

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

**Theorem**

AlocTL is closed under extension and weakening
Closure by extension

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

¬F_{i,j} \varphi \text{ forbidden!}

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

OK
Closure by extension

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

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

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!

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

OK
Closure by extension

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

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

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 $F_{i,j} \land Out$ and $Out \land H_{i,j} \varphi$. 
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 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.

Note that we do not need to change the specification since it is closed under extension.
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.
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.
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
Centralize information

Example

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

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

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

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

\[
\begin{align*}
t: & \quad 2 \\
& \quad \rightarrow \quad \text{req}_3 \\
\end{align*}
\]
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

- \( t: 2 \)
- \( t': \)
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 \( \sigma \) contains \( req_3 \) but no alert

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

1. \( \text{Msg} \)
2. \( \text{req}_3 \)
3. \( \text{Msg, c} \cdot c \)
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 \( \sigma \) contains \( req_3 \) but no alert

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

\[
\begin{align*}
  t: & \quad 1 \quad a \quad a \quad \text{Msg} \quad a \quad 2 \\
  & \quad c \quad c \quad (\text{Msg}, c \cdot c) \\
  t': & \quad 3 \quad \text{req}_3 \quad b \quad (\text{Msg}, c \cdot c \cdot \text{req}_3 \cdot b)
\end{align*}
\]
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

1. $a \quad a \quad \text{Msg} \quad a$
2. $c \quad c \quad (\text{Msg},c \cdot c)$
3. $\text{req}_3 \quad c \quad b \quad (\text{Msg},c \cdot c \cdot \text{req}_3 \cdot b)$

$t'$: $a \quad a$
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
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: req_3 \rightarrow F_{32}(\neg Y_2 \text{alert} \leftrightarrow \text{grant})

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

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

\[
\begin{align*}
1 & \quad a \quad a \\
\text{t:} \quad 2 & \quad c \quad \downarrow \quad c \\
3 & \quad \text{req}_3 \quad \downarrow \quad b
\end{align*}
\]

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

\[
\begin{align*}
t: & \quad \text{a} \quad \text{a} \quad \text{a} \quad (\text{Ord}_2, \text{grant}) \\
1 & \quad \text{c} \quad \text{c} \quad \text{req}_3 \\
2 & \quad \text{b} \\
3 & \quad \text{a} \\
\end{align*}
\]

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

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

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 \( f \)

\[ t: 1 \quad 2 \quad 3 \]

\[ t': 1 \quad 2 \quad 3 \]

\[ a \quad a \quad a \quad (\text{Ord}_2, \text{grant}) \]

\[ a \quad a \quad a \quad (\text{Ord}_2, \text{grant}) \]

\[ c \quad c \quad c \quad \text{grant} \quad c \quad (\text{Ack}, c) \]

\[ \text{req}_3 \quad b \quad \text{req}_3 \quad b \quad a \]

\[ f : \text{grant} \]
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 \( f \)

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

\[
\begin{align*}
t' &: 3 \\
1 & \quad a \quad a \quad \text{grant} \quad \text{Ord}_2, \text{grant} \quad a \\
2 & \quad c \quad b \\
3 & \quad \text{req}_3 \quad b \\
\end{align*}
\]

\[
\begin{align*}
& f : \text{grant}
\end{align*}
\]
**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** \( f \)

\( t: 1 \)  
\( 2 \)  
\( 3 \)  
\( t': \)  
\( f: \text{grant} \)
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: \( f(\sigma) = \text{grant} \iff \sigma \text{ contains req}_3 \) but no alert

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

\[
\begin{array}{c}
1 \quad a \quad a \quad a \\
\downarrow \quad \downarrow \quad \downarrow \\
2 \quad c \quad c \quad c \\
\downarrow \quad \downarrow \quad \downarrow \\
3 \quad \text{req}_3 \quad b \quad b \\
\end{array}
\]

\( t: \quad (\text{Ord}_2, \text{grant}) \)

\( t': \quad (\text{Ack}, c \cdot b) \)

\( f : \text{grant} \)
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 \( f \)

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

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

\( f : \text{grant} \)
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 \( f \)

\[ t: \quad 1 \quad \xrightarrow{a} \quad 2 \quad \xrightarrow{c} \quad 3 \quad \xrightarrow{\text{req}_3} \quad 2 \quad \xrightarrow{\text{alert}} \quad 1 \]

\[ t': \quad 1 \quad \xrightarrow{a} \quad 2 \quad \xrightarrow{c} \quad 3 \quad \xrightarrow{\text{req}_3} \quad 2 \quad \xrightarrow{b} \quad 3 \quad \xrightarrow{a} \]

\[ f : \text{grant} \]
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 \( \text{alert} \)

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

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

\[ t': \begin{array}{l}
1 \quad a \quad a \\
2 \quad c \quad c \\
3 \quad \text{req}_3 \quad b \quad a \\
\end{array} \]

\( f: \text{grant} \)
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 \( f \)

\[
\begin{align*}
\text{Master} & : t' \\
\text{Process 1} & : a, a, a \quad \text{(Ord}_2, \text{grant)} \\
\text{Process 2} & : c, c, c \quad \text{alert} \quad \text{(Nack, alert)} \\
\text{Process 3} & : \text{req}_3, b, a \\
\end{align*}
\]

\( f : \text{grant} \)
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 $f$

1. $a\ a\ a$ (Ord2,grant)
2. $c\ \downarrow\ c$
3. $\text{req}_3\ \downarrow\ b$

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

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

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

1  
\(\begin{array}{cccc}
1 & a & a & a \\
2 & c & c & c \\
3 & \text{req}_3 & \text{req}_3 & \text{req}_3 \\
\end{array}\) 

\(\begin{array}{cccc}
2 & \text{alert} & \text{alert} & \text{alert} \\
3 & b & b & b \\
\end{array}\) 

\((\text{Ord}_2,\text{grant})\) 

\((\text{Nack},\text{alert})\)

\((\text{Nack},\text{alert} \cdot b)\)

\(t'\):
\(\begin{array}{cccccc}
a & a & c & c & \text{req}_3 & b & a \\
\end{array}\) 

\(f : \text{grant}\)
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 \( f \)
Lemma

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

Proof - end
Lemma

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

Lemma

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

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

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

If \( x \prec y \) 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 \( f \)-maximal \( f \)-run.
Lemma
$t'$ is an extension of $\pi_\Gamma(t)$.

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 $f$-maximal $f$-run.

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

Proposition
If the strategy $f$ over the singleton has finite memory, then we can distribute the strategy for the strongly connected architecture using

- finite alphabets $\Sigma_{i,j}$
- local strategies with finite memories
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!