Synthèse de systèmes distribués ouverts

Nathalie Sznajder

LSV, ENS Cachan & CNRS & INRIA Saclay IdF

12 Novembre 2009
Need for formal methods
Need for formal methods

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

- Test
- Computer-aided proofs
- Model-checking
Principles of Model-checking

a system

a specification
Principles of Model-checking

Does a system satisfy a specification?
Principles of Model-checking

Does a system satisfy a specification?
Principles of Model-checking

Does a system satisfy a specification?
Principles of Model-checking

Does a system satisfy a specification?

\[ \phi \]

model

formula
Principles of Model-checking

Does a system satisfy a specification?

model \models Model-checking algorithm \phi formula
Synthesis

a specification
Synthesis

Find a system satisfying a specification
Synthesis

Find a system satisfying a specification
Synthesis

Find a system satisfying a specification

\( \phi \)

formula
Synthesis

Find a system satisfying a specification

\[ \models \phi \]

model \quad \models \quad \phi \quad \text{formula}
Synthesis of open and reactive systems

Open system $S$

inputs from $E$  outputs to $E$
Synthesis of open and reactive systems

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

For reasonable systems and specifications, the problems are decidable.
Synthesis of open reactive systems different from satisfiability
Synthesis of open reactive systems different from satisfiability
Synthesis of open reactive systems different from satisfiability

Eventually $u = 1$

$x = 1$ if and only if eventually $u = 1$

$u$ not controllable!
Synthesis of open reactive systems different from satisfiability

- Open system
- Eventually \( u = 1 \) not controllable!
- \( x = 1 \) if and only if eventually \( u = 1 \) the system cannot foresee what will happen!
Synthesis of distributed open systems

Open distributed system $S$

Specification $\varphi$

input of $E$  output to $E$
Synthesis of distributed open systems

Two problems

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

Main parameters

- Which semantics? synchronous, asynchronous
- What kind of specification?
- What kind of memory for the programs? local memory bounded or unbounded memory
Outline

Introduction

Synthesis of synchronous distributed systems
  Model and motivations
  Uncomparable information
  Uniformly well connected architectures
  Well connected architectures

Synthesis of asynchronous distributed systems
  Model
  Specifications
  Decidability Results

Conclusion
Outline

Introduction

Synthesis of synchronous distributed systems
  Model and motivations
  Uncomparable information
  Uniformly well connected architectures
  Well connected architectures

Synthesis of asynchronous distributed systems
  Model
  Specifications
  Decidability Results

Conclusion
Distributed systems with shared variables

Architecture

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

where \(S^I = \prod_{v \in I} S^v\) for \(I \subseteq V\).
Distributed systems with shared variables

Architecture

1. \((\text{Proc} \cup V, E)\) bipartite graph, where \(E \subseteq (\text{Proc} \times V) \cup (V \times \text{Proc})\).
2. \(V_i \subseteq V\) input values from the environment, and \(V_o \subseteq V\) output values from the system, read by the environment.
3. \(S^v\) (finite) domain for each variable \(v \in V\).
4. \(s_0 \in S^V\) initial state

where \(S^I = \prod_{v \in I} S^v\) for \(I \subseteq V\).
Parameters

- Which semantics?

synchronous behaviors
Synthesis of synchronous distributed systems

Parameters

- Which semantics?
  - synchronous behaviors

$s_0s_1s_2 \cdots$ where $s_n \in S^V$ are global states.
Synthesis of synchronous distributed systems

Parameters

- Which semantics?
  - synchronous behaviors
  
  \[ s_0 s_1 s_2 \cdots \text{ where } s_n \in S^V \text{ are global states.} \]

- With or without delays?
Synthesis of synchronous distributed systems

Parameters

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

Parameters

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

Parameters

- Which semantics?
  - synchronous behaviors
  - \( s_0 s_1 s_2 \cdots \) where \( s_n \in S^V \) are global states.

- With or without delays?

- What kind of memory for the program?
  - local memory
  - \( f_p : (S^{E^{-1}(p)})^* \rightarrow S^E(p) \) for all \( p \in P \).

- What kind of specification?
  - Temporal logic formulae, total or external over words/trees over alphabet \( S^V \).
Synthesis of synchronous distributed systems

Synchronous runs

\[
\begin{align*}
  u_1 & \quad u_2 & \quad u_3 & \quad \ldots \\
  v_1 & \quad v_2 & \quad v_3 & \quad \ldots \\
  t_1 & \quad t_2 & \quad t_3 & \quad \ldots \\
  x_1 & \quad x_2 & \quad x_3 & \quad \ldots \\
\end{align*}
\]

- **0-delay:**
  \[
  t_i = f_t(u_1 \cdots u_i) \\
  x_i = f_x((t_1, v_1) \cdots (t_i, v_i))
  \]

- **1-delay:**
  \[
  t_i = f_t(u_1 \cdots u_{i-1}) \\
  x_i = f_x((t_1, v_1) \cdots (t_{i-1}, v_{i-1}))
  \]
Synthesis of synchronous distributed systems

Undecidable and decidable architectures

Pnueli-Rosner (FOCS’90)

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

Undecidable and decidable architectures

Pnueli-Rosner (FOCS’90)

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

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

Synthesis problem for pipeline architectures is decidable for CTL* total specifications.
Synthesis of synchronous distributed systems

Undecidable and decidable architectures

Pnueli-Rosner (FOCS’90)

Synthesis problem for synchronous distributed systems is undecidable for LTL or CTL \textit{external} or \textit{total} specifications.

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

Synthesis problem for pipeline architectures is decidable for CTL* \textit{total} specifications.
Total specifications: Information fork criterion

Finkbeiner-Schewe (LICS’05)

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

\[ \begin{align*}
u & \rightarrow a \\
a & \rightarrow x \\
x & \rightarrow b \\
b & \rightarrow y \\
\end{align*} \]

\[ \begin{align*}
v & \rightarrow a \\
\end{align*} \]

\[ \begin{align*}
v & \rightarrow a \\
c & \rightarrow z \\
\end{align*} \]
Back to external specifications

Finkbeiner-Schewe (LICS’05)

Synthesis problem is undecidable over this architecture with LTL total specifications.
Back to external specifications

Finkbeiner-Schewe (LICS’05)
Synthesis problem is undecidable over this architecture with LTL \emph{total} specifications.

Pnueli-Rosner (FOCS’90)
Synthesis problem is decidable over this architecture with LTL \emph{external} specifications.
What if we consider external specifications?
Architectures with uncomparable information

**View of a variable**

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

**Uncomparable information (FSTTCS’06)**

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

View of a variable

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

Uncomparable information (FSTTCS’06)

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

Otherwise it is said to have **linearly preordered information**.
Architectures with uncomparable information

**View of a variable**

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

**Uncomparable information (FSTTCS’06)**

An architecture has **uncomparable information** if there exist $x, y$ output variables such that $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$ and $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$. Otherwise it is said to have **linearly preordered information**.
Architectures with uncomparable information

**View of a variable**

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

**Uncomparable information (FSTTCS’06)**

An architecture has uncomparable information if there exist $x, y$ output variables such that $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$ and $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$. Otherwise it is said to have linearly preordered information.
Architectures with uncomparable information

View of a variable

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

Uncomparable information (FSTTCS’06)

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

Otherwise it is said to have linearly preordered information.
Architectures with uncomparable information

**View of a variable**

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

**Uncomparable information (FSTTCS’06)**

An architecture has uncomparable information if there exist $x, y$ output variables such that View($x$) \ View($y$) ≠ ∅ and View($y$) \ View($x$) ≠ ∅.

Otherwise it is said to have linearly preordered information.
Uncomparable information yields undecidability

**Theorem (FSTTCS’06,FMSD’09)**

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

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

![Diagram of uniformly well connected architecture]
Uniformly well connected architectures

**Definition**

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

Definition

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

![Diagram showing the architecture with nodes labeled $u$, $v$, $w$, $s$, $t$, $p_1$, $p_2$, $p_3$, $p_4$, $p_5$, $x$, $y$, $z$. The diagram illustrates the connections between these nodes, demonstrating the unique routing property.]
Uniformly well connected architectures

Definition

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

Definition

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

![Diagram showing uniformly well connected architecture](diagram.png)
Decidability results

Proposition

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

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

**Theorem (FSTTCS’06, FMSD’09)**
Synthesis problem is decidable for UWC architectures with linearly preordered information and external specifications (branching or linear time).

**Proof idea**
Routing is used for memoryless internal strategies.
Robust specifications

Definition (FSTTCS’06, FMSD’09)

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

Definition (FSTTCS’06, FMSD’09)

A formula $\varphi$ in CTL* is **robust** if $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \psi_z$ where $\psi_z$ only depends on $\text{View}(z) \cup \{z\}$.
Definition (FSTTCS’06, FMSD’09)

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

**Definition (FSTTCS’06, FMSD’09)**

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

**Theorem (FSTTCS’06, FMSD’09)**

Synthesis problem is decidable for uniformly well connected architectures and external, robust $\text{CTL}^*$ specifications.
Undecidable and decidable architectures -
Total specifications
Undecidable and decidable architectures -

External specifications

Uncomparable information

\[ u \rightarrow v \rightarrow w \rightarrow y \rightarrow z \]

\[ x \rightarrow a_1 \rightarrow y_1 \rightarrow a_2 \rightarrow y_2 \rightarrow a_3 \]

\[ z_1 \rightarrow z_2 \rightarrow z_3 \]
Undecidable and decidable architectures -

External specifications

Uncomparable information

UWC architectures
Well Connected Architectures

Definition

An architecture is **well connected** if, for each output variable \( v \), the subarchitecture formed by its ‘ancestors’ is UWC.

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

An architecture is well connected if, for each output variable \( v \), the subarchitecture formed by its ‘ancestors’ is UWC.

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

**Definition**

An architecture is *well connected* if, for each output variable \( v \), the subarchitecture formed by its ‘ancestors’ is UWC.

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

Definition

An architecture is well connected if, for each output variable \( v \), the subarchitecture formed by its ‘ancestors’ is UWC.

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

**Definition**

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

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

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

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

**Definition**

An architecture is **well connected** if, for each output variable \( v \), the subarchitecture formed by its ‘ancestors’ is UWC.

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

![Diagram of well-connected architecture]

Nathalie Sznajder PhD defense - November 12th, 2009, p.25
Undecidability for well connected architectures with linearly preordered information
Undecidability for well connected architectures with linearly preordered information
Undecidability for well connected architectures with linearly preordered information

Process $p_6$ knowing no value of $u$ yields undecidability
Undecidability for well connected architectures with linearly preordered information

Process $p_6$ knowing all values of $u$ yields decidability
Undecidability for well connected architectures with linearly preordered information

Process $p_6$ missing one bit of $u$ yields undecidability
Synthesis of synchronous distributed systems

Undecidable and decidable architectures - External specifications

Uncomparable information

WC architectures

UWC architectures

$\begin{array}{c}
\mathbb{U} \\
\mathbb{V} \\
\mathbb{a} \\
\mathbb{b} \\
\mathbb{y} \\
\mathbb{z}
\end{array}$

$\begin{array}{c}
\mathbb{X} \\
\mathbb{a_1} \\
\mathbb{v_1} \\
\mathbb{a_2} \\
\mathbb{v_2} \\
\mathbb{a_3}
\end{array}$

$\begin{array}{c}
\mathbb{z_1} \\
\mathbb{z_2} \\
\mathbb{z_3}
\end{array}$
Related work

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

Introduction

Synthesis of synchronous distributed systems
  Model and motivations
  Uncomparable information
  Uniformly well connected architectures
  Well connected architectures

Synthesis of asynchronous distributed systems
  Model
  Specifications
  Decidability Results

Conclusion
## Distributed Synthesis

### Parameters

- **Which semantics?** asynchronous executions are partial orders (Mazurkiewicz traces)
- **What kind of memory for the programs?** local memory
- **What kind of specification?** external, over partial orders
Asynchronous semantics: communication through common actions

- Rendez-vous: two processes agree on a common action.
Asynchronous semantics: communication through common actions

- Rendez-vous: two processes agree on a common action.
Asynchronous semantics: communication through common actions

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

![Diagram showing asynchronous communication through common actions]
Asynchronous semantics: communication through common actions

- Rendez-vous: two processes agree on a common action.
Asynchronous semantics: communication through common actions

- Rendez-vous: two processes agree on a common action.
Asynchronous semantics: communication through common actions

- Rendez-vous: two processes agree on a common action.
Asynchronous semantics: communication through common actions

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

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

Architectures

- Communication graph \((Proc, E)\)
Communication by signals

Architectures

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

Architectures

- Communication graph \((\text{Proc}, E)\)
- For each process \(i\), sets \(\text{In}_i\) and \(\text{Out}_i\) of input and output signals:
  \[ \Gamma = \bigcup_{i \in \text{Proc}} \text{In}_i \cup \bigcup_{i \in \text{Proc}} \text{Out}_i \]
- For each process \(i\),
  \(\Sigma_i^c\) is the set of signals it can send (control),
  \(\Sigma_i\) is the set of signals it can observe.
Programs

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

1. \[ \]
f_1 : b

2. \[ \]
f_2 : c

3. \[ \]
f_3 : d
Programs

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

1 \hfill \hfill \hfill f_1 : b \\
2 \hfill \hfill \hfill f_2 : c \\
3 \hfill \hfill \hfill f_3 : d
Programs

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

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

1. $a \quad f_1 : b'$
2. $a' \quad f_2 : c'$
3. $f_3 : d$
Programs

- 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. \( a' \) — \( f_2 : c \)
3. \( a' \) — \( f_3 : d \)
Programs

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

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

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

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

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

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

$f_1 : j$
$f_2 : i$
$f_3 : d$
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 $f_i$. 

![Diagram of a run with labeled events and transitions labeled with strategies $f_1 : j$, $f_2 : i$, and $f_3 : d$.]
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{align*}
1 & a \quad b' \quad g \\
2 & a' \quad a' \quad h \\
3 & a''
\end{align*}

$\begin{align*}
f_1 : j \\
f_2 : i \\
f_3 : d
\end{align*}$
Fairness conditions
Fairness conditions

\[ G(\text{req}_1 \rightarrow (F \text{grant}_1)) \land G(\text{req}_2 \rightarrow (F \text{grant}_2)) \]
Fairness conditions

\[ G(req_1 \rightarrow (F \text{grant}_1)) \land G(req_2 \rightarrow (F \text{grant}_2)) \]

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

Fairness has to be **distributed**.
Fairness conditions

Some runs are unfair for the processes.

Fairness has to be distributed.
Models of an external specification

Parameters

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

Parameters

- What kind of specification? external, over partial orders
Observable runs

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

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

where \( \Gamma \) is the set of external actions.
Models of an external specification

Observable runs

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

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

where \( \Gamma \) is the set of external actions.
Given a run $t = (V, \lambda, \leq)$, we define the observable run by

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

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

Communication induces order relation
Acceptable Specifications

Communication induces order relation

1  2  3

1  b  a
2   c  3
Acceptable Specifications

Communication induces order relation
Acceptable Specifications

Communication induces order relation
Acceptable Specifications

Communication induces order relation
Acceptable Specifications

Communication induces order relation

1. a
2. b
3. c
Acceptable Specifications

Communication induces order relation

Diagram showing the relationships between nodes 1, 2, and 3, with arrows indicating communication and order.

1. Node 1 sends a message to node 2.
2. Node 2 sends a message to node 3.
3. Node 3 receives a message from node 2.

Nodes and messages are labeled as follows:
- Node 1: a
- Node 2: b
- Node 3: c
Acceptable Specifications

Communication induces order relation
Acceptable Specifications

Communication induces order relation
Acceptable Specifications

Communication induces order relation

1
2
3

1
2
3

a
b
c

Nathalie Sznajder PhD defense - November 12th, 2009, p.37
Acceptable Specifications

Communication induces order relation

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure}
\end{figure}
Acceptable Specifications

Restrictions on specifications

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

Restrictions on specifications

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

\[ \begin{align*}
1 & \quad b \\
2 & \quad a \\
3 & \quad c
\end{align*} \]
Acceptable Specifications

Restrictions on specifications

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

Input events are not controllable by processes
Acceptable Specifications

Input events are not controllable by processes

Diagram showing the flow of events from 1, 2, and 3, with arrows indicating the flow of requests (req) and grants (grant), and a note on req'.
Acceptable Specifications

Input events are not controllable by processes
Acceptable Specifications

Input events are not controllable by processes

1
2
3

req
grant
req'
Acceptable Specifications

Input events are not controllable by processes
Acceptable Specifications

Input events are not controllable by processes

1
2
3

req
grant
req'

1
2
3

req
grant
req'

1
2
3

req
grant
req'

Nathalie Sznajder PhD defense - November 12th, 2009, p.39
Acceptable Specifications

Input events are not controllable by processes
Acceptable Specifications

Restrictions on specifications

- **Communication induces order relation**: specifications should not discriminate between a partial order and its order extensions.
- **Input events are not controllable**: specifications should not discriminate between a partial order and its "weakenings".
Acceptable Specifications

Restrictions on specifications

- **Communication induces order relation**: specifications should not discriminate between a partial order and its order extensions.
- **Input events are not controllable**: specifications should not discriminate between a partial order and its "weakenings".
Acceptable Specifications

Restrictions on specifications

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

- **Input events are not controllable**: specifications should not discriminate between a partial order and its "weakenings".

![Diagram](attachment:image.png)
Fair synthesis problem
Fair synthesis problem

Given an architecture
a specification
Fair synthesis problem

Given an architecture
a specification

Decide whether there exists a distributed program such that all its fair runs meet the specification.
Fair synthesis problem

**Given** an architecture
a specification

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

**Theorem (SOFSEM’09 + PhD)**

The fair synthesis problem over singleton architectures is decidable for regular specifications.
Synthesis of asynchronous distributed systems

Fair synthesis problem

Given an architecture
a specification

Decide whether there exists a distributed program such that
all its fair runs meet the specification.

Theorem (SOFSEM’09 + PhD)
The fair synthesis problem over singleton architectures is decidable
for regular specifications.

Theorem (SOFSEM’09 + PhD)
The fair synthesis problem over strongly connected architectures is
decidable for acceptable specifications.

Proof idea
By reduction to the singleton
We select a cycle.

The processes will use a token to play one at a time and collect information on what happened in their past.

Aim: create a run that will be a weakening of some $f$-run over the singleton.
Token passing

**Example**

![Diagram showing token passing example]

**Process 1 has the token at the beginning**

<table>
<thead>
<tr>
<th>Step</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(a) ((\text{Token}, a \cdot a))</td>
</tr>
<tr>
<td>t: 2</td>
<td>c ((\text{Token}, a \cdot a \cdot c \cdot c))</td>
</tr>
<tr>
<td>3</td>
<td>\text{req}_3 (b)</td>
</tr>
<tr>
<td>(t'_1)</td>
<td>(a) (a)</td>
</tr>
<tr>
<td>(t'_2)</td>
<td>(a) (a) (c) (c)</td>
</tr>
<tr>
<td>(t'_3)</td>
<td>(a) (a) (c) (c) (\text{req}_3) (b)</td>
</tr>
</tbody>
</table>
Examples of strongly connected architectures
Related Work

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

Introduction

Synthesis of synchronous distributed systems
  Model and motivations
  Uncomparable information
  Uniformly well connected architectures
  Well connected architectures

Synthesis of asynchronous distributed systems
  Model
  Specifications
  Decidability Results

Conclusion
Summary

Synthesis of synchronous systems
► Necessary condition for decidability for external specifications
► Exhibition of a new class of architectures for which it becomes a sufficient condition
► New undecidability proof giving new insights

Synthesis of asynchronous systems
► Definition of a realistic model for synthesis of asynchronous systems
► Decidability of a class which is undecidable in the synchronous case
Open problems

- Synchronous case
  - Definition of a general decidability criterion for external specifications in the synchronous case

- Asynchronous case
  - Obtain decidability of the problem on all architectures

- Fault-tolerant synthesis
Thank you for your attention!