Distributed synthesis: synchronous and asynchronous semantics

Paul Gastin

LSV
ENS de Cachan & CNRS
Paul.Gastin@lsv.ens-cachan.fr

EPIT, May 31st, 2006
Outline

1. Control for sequential systems

Control for distributed systems

Synchronous semantics

Asynchronous semantics
Open / Reactive system

Model for the open system

- Transitions system \( A = (Q, \Sigma, q_0, \delta) \)
  - \( Q \): finite or infinite set of states,
  - \( \delta \): deterministic or non deterministic transition function.
- \( \Sigma = \Sigma_c \uplus \Sigma_{uc} \) Controllable / Uncontrollable events.
- \( \Sigma = \Sigma_o \uplus \Sigma_{uo} \) Observable / Unobservable events.
Example: Elevator

Transition system

- **States:**
  - position of the cabin
  - flag is_open for each door
  - flag is_called for each level
  - number of persons in the cabin

- **Events:**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma_o$</td>
<td>call level $i$</td>
<td>enter/exit cabin</td>
</tr>
<tr>
<td>$\Sigma_{uc}$</td>
<td>open/close door $i$</td>
<td>move 1 level up/down</td>
</tr>
</tbody>
</table>

- We get easily a finite and deterministic transition system.
Specification

Linear time: LTL, FO, MSO, regular, ...

- Safety: $G(\text{level} \neq i \rightarrow \text{is\_closed}_i)$
- Liveness: $G(\text{is\_called}_i \rightarrow F(\text{level} = i \land \text{is\_open}_i))$

Branching time: CTL, CTL*, $\mu$-calculus, ...

- $\text{AG}\langle\text{call}_i\rangle \top$ (call$_i$ is uncontrollable)
- $\text{AG\ EF(level} = 0 \land \text{is\_open}_0)$
Control problem

Two problem

- **Control**: Given a system $S$ and a specification $\varphi$, decide whether there exists a controller $C$ such that $S \otimes C \models \varphi$.

- **Synthesis**: Given a system $S$ and a specification $\varphi$, build a controller $C$ (if one exists) such that $S \otimes C \models \varphi$.
Control problem

Two problem

- Control: Given a system $S$ and a specification $\varphi$, decide whether there exists a controller $C$ such that $S \otimes C \models \varphi$.
- Synthesis: Given a system $S$ and a specification $\varphi$, build a controller $C$ (if one exists) such that $S \otimes C \models \varphi$. 
Controller

Under full state-event observation

- Controller: \( f : Q(\Sigma Q)^* \to 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in Q(\Sigma Q)^* \).
- Controlled behavior: \( q_0, a_1, q_1, a_2, q_2, \ldots \) with \( (q_{i-1}, a_i, q_i) \in \delta \) and \( a_i \in f(q_0a_1q_1\cdots q_{i-1}) \) for all \( i > 0 \).
- Controlled execution tree: \( t : D^* \to \Sigma \times Q \) with
  - \( t(\varepsilon) = (a, q_0) \) (\( a \in \Sigma \) fixed arbitrarily)
  - for all \( x = d_1 \cdots d_n \in D^* \) with \( t(d_1 \cdots d_i) = (a_i, q_i) \), we have
    \[
    t(\text{sons}(x)) = \{(a, q) \mid a \in f(q_0a_1q_1\cdots a_nq_n) \text{ and } (q_n, a, q) \in \delta\}.
    \]

Under full event observation

- Controller: \( f : \Sigma^* \to 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in \Sigma^* \).
  Remark: same as full state-event observation if the system is deterministic.

Under partial event observation

- Controller: \( f : \Sigma_o^* \to 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in \Sigma^* \).
- Controlled behavior: \( q_0, a_1, q_1, a_2, q_2, \ldots \) with \( (q_{i-1}, a_i, q_i) \in \delta \) and \( a_i \in f \circ \Pi_{\Sigma_o}(a_1 \cdots a_{i-1}) \) for all \( i > 0 \).
Controller

Under full state-event observation

- Controller: \( f: Q(\Sigma Q)^* \rightarrow 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in Q(\Sigma Q)^* \).
- Controlled behavior: \( q_0, a_1, q_1, a_2, q_2, \ldots \) with \( (q_{i-1}, a_i, q_i) \in \delta \) and \( a_i \in f(q_0 a_1 q_1 \cdots q_{i-1}) \) for all \( i > 0 \).
- Controlled execution tree: \( t: D^* \rightarrow \Sigma \times Q \) with
  - \( t(\varepsilon) = (a, q_0) \) \( (a \in \Sigma \) fixed arbitrarily)\)
  - for all \( x = d_1 \cdots d_n \in D^* \) with \( t(d_1 \cdots d_i) = (a_i, q_i) \), we have
    \( t(\text{sons}(x)) = \{(a, q) \mid a \in f(q_0 a_1 q_1 \cdots a_{n-1} q_n) \text{ and } (q_n, a, q) \in \delta\} \).

Remark: same as full state–event observation if the system is deterministic.

Under partial event observation

- Controller: \( f: \Sigma^* \rightarrow 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in \Sigma^* \).

Under partial event observation

- Controller: \( f: \Sigma_o^* \rightarrow 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in \Sigma^* \).
- Controlled behavior: \( q_0, a_1, q_1, a_2, q_2, \ldots \) with \( (q_{i-1}, a_i, q_i) \in \delta \) and \( a_i \in f \circ \Pi_{\Sigma_o}(a_1 \cdots a_{i-1}) \) for all \( i > 0 \).
Controller

Under full state-event observation

- Controller: $f : Q(\Sigma Q)^* \rightarrow 2^\Sigma$ with $\Sigma_{uc} \subseteq f(x)$ for all $x \in Q(\Sigma Q)^*$.
- Controlled behavior: $q_0, a_1, q_1, a_2, q_2, \ldots$ with $(q_{i-1}, a_i, q_i) \in \delta$ and $a_i \in f(q_0 a_1 q_1 \cdots q_{i-1})$ for all $i > 0$.
- Controlled execution tree: $t : D^* \rightarrow \Sigma \times Q$ with
  - $t(\varepsilon) = (a, q_0)$ ($a \in \Sigma$ fixed arbitrarily)
  - for all $x = d_1 \cdots d_n \in D^*$ with $t(d_1 \cdots d_i) = (a_i, q_i)$, we have
    $t(\text{sons}(x)) = \{(a, q) \mid a \in f(q_0 a_1 q_1 \cdots a_n q_n) \text{ and } (q_n, a, q) \in \delta\}$.

Under full event observation

- Controller: $f : \Sigma^* \rightarrow 2^\Sigma$ with $\Sigma_{uc} \subseteq f(x)$ for all $x \in \Sigma^*$.
  Remark: same as full state-event observation if the system is deterministic.

Under partial event observation

- Controller: $f : \Sigma_o^* \rightarrow 2^\Sigma$ with $\Sigma_{uc} \subseteq f(x)$ for all $x \in \Sigma^*$.
- Controlled behavior: $q_0, a_1, q_1, a_2, q_2, \ldots$ with $(q_{i-1}, a_i, q_i) \in \delta$ and $a_i \in f \circ \Pi_{\Sigma_o}(a_1 \cdots a_{i-1})$ for all $i > 0$. 
Controller

Under full state-event observation

- Controller: \( f : Q(\Sigma Q)^* \rightarrow 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in Q(\Sigma Q)^* \).
- Controlled behavior: \( q_0, a_1, q_1, a_2, q_2, \ldots \) with \( (q_{i-1}, a_i, q_i) \in \delta \) and \( a_i \in f(q_0a_1q_1\cdots q_{i-1}) \) for all \( i > 0 \).
- Controlled execution tree: \( t : D^* \rightarrow \Sigma \times Q \) with
  - \( t(\varepsilon) = (a, q_0) \) (\( a \in \Sigma \) fixed arbitrarily)
  - for all \( x = d_1 \cdots d_n \in D^* \) with \( t(d_1 \cdots d_i) = (a_i, q_i) \), we have \( t(\text{sons}(x)) = \{(a, q) \mid a \in f(q_0a_1q_1\cdots a_nq_n) \text{ and } (q_n, a, q) \in \delta\} \).

Remark: same as full state-event observation if the system is deterministic.

Under full event observation

- Controller: \( f : \Sigma^* \rightarrow 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in \Sigma^* \).

Under partial event observation

- Controller: \( f : \Sigma_o^* \rightarrow 2^\Sigma \) with \( \Sigma_{uc} \subseteq f(x) \) for all \( x \in \Sigma^* \).
- Controlled behavior: \( q_0, a_1, q_1, a_2, q_2, \ldots \) with \( (q_{i-1}, a_i, q_i) \in \delta \) and \( a_i \in f \circ \Pi_{\Sigma_o}(a_1 \cdots a_{i-1}) \) for all \( i > 0 \).
Control versus Game

## Correspondance

<table>
<thead>
<tr>
<th>Transition system</th>
<th>= Game arena (graph).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllable events</td>
<td>= Actions of player 1 (controller).</td>
</tr>
<tr>
<td>Uncontrollable events</td>
<td>= Action of player 0 (opponent, environment).</td>
</tr>
<tr>
<td>Behavior</td>
<td>= Play.</td>
</tr>
<tr>
<td>Controller</td>
<td>= Strategy.</td>
</tr>
<tr>
<td>Specification</td>
<td>= Winning condition.</td>
</tr>
<tr>
<td>Finding a controller</td>
<td>= finding a winning strategy.</td>
</tr>
</tbody>
</table>

## Control problem

Given a system $S$ and a specification $\varphi$, does there exist a controller $C$ such that $L(C \otimes S) \subseteq L(\varphi)$?

## Theorem

If the system is finite state and the specification is regular then the control problem is decidable.

Moreover, when $(S, \varphi)$ is controllable, we can synthesize a finite state controller.
## Control versus Game

### Correspondance

<table>
<thead>
<tr>
<th>Transition system</th>
<th>= Game arena (graph).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllable events</td>
<td>= Actions of player 1 (controller).</td>
</tr>
<tr>
<td>Uncontrollable events</td>
<td>= Action of player 0 (opponent, environment).</td>
</tr>
<tr>
<td>Behavior</td>
<td>= Play.</td>
</tr>
<tr>
<td>Controller</td>
<td>= Strategy.</td>
</tr>
<tr>
<td>Specification</td>
<td>= Winning condition.</td>
</tr>
<tr>
<td>Finding a controller</td>
<td>= finding a winning strategy.</td>
</tr>
</tbody>
</table>

### Control problem

Given a system $S$ and a specification $\varphi$, does there exist a controller $C$ such that $L(C \otimes S) \subseteq L(\varphi)$?

### Theorem

If the system is **finite state** and the specification is **regular** then the control problem is **decidable**.
Moreover, when $(S, \varphi)$ is controllable, we can synthesize a **finite state** controller.
Control problem (Exact)

Given a system \( S \) (with accepting states) and a specification \( K \subseteq \Sigma^* \), does there exist a controller \( C \) such that \( \mathcal{L}(C \otimes S) = K \)?

Theorem

- \((S, \text{Pref}(K))\) is controllable iff \( \text{Pref}(K) \cdot \Sigma_{uc} \cap \text{Pref}(\mathcal{L}(S)) \subseteq \text{Pref}(K) \).
- \((S, K)\) is controllable without deadlock iff
  - \( \text{Pref}(K) \cdot \Sigma_{uc} \cap \text{Pref}(\mathcal{L}(S)) \subseteq \text{Pref}(K) \)
  - \( \text{Pref}(K) \cap \mathcal{L}(S) = K \).
- If \( S \) is finite state and \( K \) regular then the control problem is decidable. When \((S, K)\) is controllable, we can synthesize a finite state controller.

Other results

- control under partial observation
- maximal controllable sub-specification
- generalization to infinite behaviors (Thistle - Wonham)
- ...
Control problem (Exact)

Given a system $S$ (with accepting states) and a specification $K \subseteq \Sigma^*$, does there exist a controller $C$ such that $L(C \otimes S) = K$?

Theorem

- $(S, \text{Pref}(K))$ is controllable iff $\text{Pref}(K) \cdot \Sigma_{uc} \cap \text{Pref}(L(S)) \subseteq \text{Pref}(K)$.
- $(S, K)$ is controllable without deadlock iff
  - $\text{Pref}(K) \cdot \Sigma_{uc} \cap \text{Pref}(L(S)) \subseteq \text{Pref}(K)$
  - $\text{Pref}(K) \cap L(S) = K$.
- If $S$ is finite state and $K$ regular then the control problem is decidable. When $(S, K)$ is controllable, we can synthesize a finite state controller.

Other results

- control under partial observation
- maximal controllable sub-specification
- generalization to infinite behaviors (Thistle - Wonham)
- ...
Control problem (Exact)

Given a system $S$ (with accepting states) and a specification $K \subseteq \Sigma^*$, does there exist a controller $C$ such that $L(C \otimes S) = K$?

Theorem

- $(S, \text{Pref}(K))$ is controllable iff $\text{Pref}(K) \cdot \Sigma_{uc} \cap \text{Pref}(L(S)) \subseteq \text{Pref}(K)$.
- $(S, K)$ is controllable without deadlock iff
  - $\text{Pref}(K) \cdot \Sigma_{uc} \cap \text{Pref}(L(S)) \subseteq \text{Pref}(K)$
  - $\text{Pref}(K) \cap L(S) = K$.
- If $S$ is finite state and $K$ regular then the control problem is decidable. When $(S, K)$ is controllable, we can synthesize a finite state controller.

Other results

- control under partial observation
- maximal controllable sub-specification
- generalization to infinite behaviors (Thistle - Wonham)
- ...
Synthesis of reactive programs

Pnueli-Rosner 89

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1x_2\cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3)\cdots$ with $y_n = f_1(x_1\cdots x_n)$ for all $n > 0$.

Implementability problem

- Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?
- Given a branching time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that its run-tree satisfies $\varphi$?
Synthesis of reactive programs

Pnueli-Rosner 89

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1x_2\cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3)\cdots$ with $y_n = f_1(x_1\cdots x_n)$ for all $n > 0$.

Implementability problem

- Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?
- Given a branching time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that its run-tree satisfies $\varphi$?
Synthesis of reactive programs

Pnueli-Rosner 89

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1x_2\cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3)\cdots$ with $y_n = f_1(x_1\cdots x_n)$ for all $n > 0$.

Implementability problem

- Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?
- Given a branching time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that its run-tree satisfies $\varphi$?
Synthesis of reactive programs

Implementability problem

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi = F(x = 1)$
- $\varphi$ is satisfiable: $(1, 0)^\omega \models \varphi$
- $\varphi$ is not implementable since the input is not controllable.

Implementability $\neq$ Validity of $\forall \vec{x} \exists \vec{y} \varphi$

- $Q_x = Q_y = \{0, 1\}$ and $\varphi = (y = 1) \leftrightarrow F(x = 1)$
- $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
- $\varphi$ is not implementable by a reactive program.

For non-reactive terminating programs, Implementability $= \text{Validity of } \forall \vec{x} \exists \vec{y} \varphi$
Synthesis of reactive programs

Implementability problem
Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Implementability $\neq$ Satisfiability
- $Q_x = \{0, 1\}$ and $\varphi = F(x = 1)$
- $\varphi$ is satisfiable: $(1, 0)^\omega \models \varphi$
- $\varphi$ is not implementable since the input is not controllable.

Implementability $\neq$ Validity of $\forall \vec{x} \exists \vec{y} \varphi$
- $Q_x = Q_y = \{0, 1\}$ and $\varphi = (y = 1) \leftrightarrow F(x = 1)$
- $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
- $\varphi$ is not implementable by a reactive program.

For non-reactive terminating programs, Implementability $= \text{Validity of } \forall \vec{x} \exists \vec{y} \varphi$
Synthesis of reactive programs

Implementability problem

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi = F(x = 1)$
- $\varphi$ is satisfiable: $(1, 0)^\omega \models \varphi$
- $\varphi$ is not implementable since the input is not controllable.

Implementability $\neq$ Validity of $\forall \vec{x} \exists \vec{y} \varphi$

- $Q_x = Q_y = \{0, 1\}$ and $\varphi = (y = 1) \iff F(x = 1)$
- $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
- $\varphi$ is not implementable by a reactive program.

For non-reactive terminating programs, Implementability $= \text{Validity of } \forall \vec{x} \exists \vec{y} \varphi$
Synthesis of reactive programs

Implementability problem

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Theorem (Pnueli-Rosner 89)

- The specification $\varphi \in \text{LTL}$ is implementable iff the formula
  \[
  A \varphi \land AG(\bigwedge_{a \in Q_x} EX(x = a))
  \]
  is satisfiable.

- When $\varphi$ is implementable, we can construct a finite state implementation (program) in time doubly exponential in $\varphi$. 
Program synthesis versus System control

Equivalence

The implementability problem for

\[ \begin{array}{c}
  x \\
  \uparrow \quad \downarrow \\
  \text{Box} \\
  \downarrow \quad \uparrow \\
  y
\end{array} \]

is equivalent to the control problem for the system

\[ \begin{array}{c}
  Q_x \\
  \rightarrow \\
  \rightarrow \quad \rightarrow \\
  Q_y \\
  \leftarrow \\
  \rightarrow \\
  Q_y
\end{array} \]
Outline

Control for sequential systems

Control for distributed systems

Synchronous semantics

Asynchronous semantics
Distributed control

Two problems, again

- Decide whether there exists a distributed controller st.
  \((S_1 \otimes C_1) \parallel \cdots \parallel (S_n \otimes C_n) \parallel E \models \phi\).
- Synthesis: If so, compute such a distributed controller.

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problems are undecidable.
Distributed control

inputs from $E$  outputs to $E$

Controlled open distributed system $S$

$C_1 \rightarrow S_1 \rightarrow S_2 \rightarrow C_2$

$C_3 \rightarrow S_3 \rightarrow S_4 \rightarrow C_4$

Two problems, again

- Decide whether there exists a distributed controller st.
  $(S_1 \otimes C_1) \parallel \cdots \parallel (S_n \otimes C_n) \parallel E \models \varphi$.
- Synthesis: If so, compute such a distributed controller.

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problems are undecidable.
Architectures with shared variables

**Architecture** $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$

- $\mathcal{P}$ finite set of processes/agents.
- $\mathcal{V}$ finite set of Variables.
- $R \subseteq \mathcal{P} \times \mathcal{V}$: $(a, x) \in R$ iff $a$ reads $x$.
  - $R(a)$ variables read by process $a \in \mathcal{P}$,
  - $R^{-1}(x)$ processes reading variable $x \in \mathcal{V}$.
- $W \subseteq \mathcal{P} \times \mathcal{V}$: $(a, x) \in W$ iff $a$ writes to $x$.
  - $W(a)$ variables written by process $a \in \mathcal{P}$,
  - $W^{-1}(x)$ processes writing to variable $x \in \mathcal{V}$.

**Example**

```
<table>
<thead>
<tr>
<th>$x_0$</th>
<th>$a_1$</th>
<th>$x_1$</th>
<th>$a_2$</th>
<th>$x_2$</th>
<th>$a_3$</th>
<th>$x_3$</th>
<th>$a_4$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x0 ---- a1 ---- x1 ---- a2 ---- x2 ---- a3 ---- x3 ---- a4 ---- x4
x0 ---- a1 ---- x1 ---- a2 ---- x2 ---- a3 ---- x3 ---- a4 ---- x4
x0 ---- a1 ---- x1 ---- a2 ---- x2 ---- a3 ---- x3 ---- a4 ---- x4
```
Distributed systems with shared variables

Distributed system/plant/arena

- \( \mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W) \) architecture.
- \( Q_x \) (finite) domain for each variable \( x \in \mathcal{V} \).
- \( \delta_a \subseteq Q_{R(a)} \times Q_{W(a)} \) legal actions/moves for process/player \( a \in \mathcal{P} \).
- \( q^0 \in Q_{\mathcal{V}} \) initial state

where \( Q_I = \prod_{x \in I} Q_x \) for \( I \subseteq \mathcal{V} \).
# Distributed Synthesis

## Problem

Given a distributed system and a specification

Problem existence/synthesis of programs/strategies for the processes/players such that the system satisfies the specification (whatever the environment/opponent does).

## Main parameters

- Which subclass of architectures?
- Which semantics?
  - synchronous (with or without delay), asynchronous
- What kind of specification?
  - LTL, CLT*, \(\mu\)-calculus
  - Rational, Recognizable word/tree
- What kind of memory for the programs?
  - memoryless, local memory, causal memory
  - finite or infinite memory
## Distributed Synthesis

### Problem

**Given** a distributed system and a specification

**Problem** existence/synthesis of programs/strategies for the processes/players such that the system satisfies the specification (whatever the environment/opponent does).

### Main parameters

- Which subclass of architectures?
- Which semantics?
  - synchronous (with or without delay), asynchronous
- What kind of specification?
  - LTL, CLT*, μ-calculus
  - Rational, Recognizable
  - word/tree
- What kind of memory for the programs?
  - memoryless, local memory, causal memory
  - finite or infinite memory
Distributed Synthesis

**Problem**

Given a distributed system and a specification, existence/synthesis of programs/strategies for the processes/players such that the system satisfies the specification (whatever the environment/opponent does).

**Main parameters**

- Which subclass of architectures?
- Which semantics?
  
  synchronous (with or without delay), asynchronous

- What kind of specification?
  
  LTL, CLT*, μ-calculus
  
  Rational, Recognizable
  
  word/tree

- What kind of memory for the programs?
  
  memoryless, local memory, causal memory
  
  finite or infinite memory
Distributed Synthesis

Problem

- Given: a distributed system and a specification

Problem: existence/synthesis of programs/strategies for the processes/players such that the system satisfies the specification (whatever the environment/opponent does).

Main parameters

- Which subclass of architectures?
- Which semantics?
  - synchronous (with or without delay), asynchronous
- What kind of specification?
  - LTL, CLT*, μ-calculus
  - Rational, Recognizable
  - word/tree
- What kind of memory for the programs?
  - memoryless, local memory, causal memory
  - finite or infinite memory
**Distributed Synthesis**

### Problem

Given a distributed system and a specification existencesynthesis of programsstrategies for the processesplayers such that the system satisfies the specification (whatever the environmentopponent does).

### Main parameters

- Which subclass of architectures?
- Which semantics?
  - synchronous (with or without delay), asynchronous
- What kind of specification?
  - LTL, CLT*, $\mu$-calculus
  - Rational, Recognizable
  - word/tree
- What kind of memory for the programs?
  - memoryless, local memory, causal memory
  - finite or infinite memory
Outline

Control for sequential systems

Control for distributed systems

3 Synchronous semantics

Asynchronous semantics
Pnueli-Rosner (FOCS’90)

Pipeline

Restrictions

- Unique writer: $|W^{-1}(x)| = 1$ for all $x \in \mathcal{V}$
- Unique reader: $|R^{-1}(x)| = 1$ for all $x \in \mathcal{V}$
- Acyclic graph (0-delay)
- No restrictions on moves: $\delta_a = Q_{R(a)} \times Q_{W(a)}$ for all $a \in \mathcal{P}$.
- Synchronous behaviors: $q_0 q_1 q^2 \cdots$ where $q^n \in Q_{\mathcal{V}}$ are global states.
- Program with local memory: $f_a : Q_{R(a)}^* \rightarrow Q_{W(a)}$ for all $a \in \mathcal{P}$.
- Specification: LTL over input and output variables only.
  - Input variables: $\text{In} = W(\text{environment})$
  - Output variables: $\text{Out} = R(\text{environment})$
0-delay synchronous semantics

Example

Programs: $f_x : Q_u^* \rightarrow Q_x$ and $f_z : (Q_x \times Q_v)^* \rightarrow Q_z$.

- Input: \[
\begin{pmatrix}
    u_1 & u_2 & u_3 & \cdots \\
    v_1 & v_2 & v_3 & \cdots
\end{pmatrix} \in (Q_u \times Q_v)^\omega.
\]

- Behavior: \[
\begin{pmatrix}
    u_1 & u_2 & u_3 & \cdots \\
    v_1 & v_2 & v_3 & \cdots \\
    x_1 & x_2 & x_3 & \cdots \\
    z_1 & z_2 & z_3 & \cdots
\end{pmatrix}
\]

with \[
\begin{align*}
x_n &= f_x(u_1 \cdots u_n) \\
z_n &= f_z((x_1, v_1) \cdots (x_n, v_n))
\end{align*}
\] for all $n > 0$. 

The synthesis problem for architecture $A_0$ and LTL (or CTL) specifications is undecidable.

Proof
Reduction from the halting problem on the empty tape.
**Undecidability proof 1**

**SPEC\textsubscript{1}:** processes \(a\) and \(b\) must output configurations

\[
\begin{align*}
0^q1^p0 \cdots & : n(v) = p \\
#^{q+p}C#^\omega & : \text{where } C \in \Gamma^*Q\Gamma^+
\end{align*}
\]

\[
(v = 0 \land y = \#) \mathcal{W} \left( v = 1 \land (v = 1 \land y = \#) \mathcal{W} (v = 0 \land y \in \Gamma^*Q\Gamma^+ #^\omega) \right)
\]

where

\[
y \in \Gamma^*Q\Gamma^+ #^\omega \overset{\text{def}}{=} y \in \Gamma \cup \left( y \in Q \land X(y \in \Gamma \cup (y \in \Gamma \land XG y = \#)) \right)
\]
**SPEC$_1$:** processes $a$ and $b$ must output configurations

\[
0^q 1^p 0 \cdots : n(v) = p
\]

\[
\#^{q+p} C \#^\omega : \text{where } C \in \Gamma^* \Gamma^+\
\]

\[
(v = 0 \land y = \#) \mathcal{W} \left( v = 1 \land (v = 1 \land y = \#) \mathcal{W} (v = 0 \land y \in \Gamma^* \Gamma^+ \#^\omega) \right)
\]

where

\[
y \in \Gamma^* \Gamma^+ \#^\omega \overset{\text{def}}{=} y \in \Gamma \cup \left( y \in Q \land X(y \in \Gamma \cup (y \in \Gamma \land X G y = \#)) \right)
\]
**Undecidability proof 2**

**SPEC$_2$:** processes $a$ and $b$ must start with the first configuration

- $u \xrightarrow{a} x$
- $v \xrightarrow{b} y$

0$q$10$\cdots$ : $n(v) = 1$

$v = 0 W \left(v = 1 \land X(v = 0 \rightarrow y \in C_1\#^\omega)\right)$
Undecidability proof 2

**SPEC**₂: processes \(a\) and \(b\) must start with the first configuration

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

\(0^q10\cdots : n(v) = 1\)

\(\#^{q+1}C_1 \#^\omega\)

\[v = 0 \mathcal{W}(v = 1 \land \mathcal{X}(v = 0 \rightarrow y \in C_1 \#^\omega))\]
**SPEC₃:** if \( n(u) = n(v) \) are synchronized then \( x = y \)

\[
0^q1^p0 \cdots \xrightarrow{a} x \quad \text{and} \quad 0^q1^p0 \cdots \xrightarrow{b} y
\]

\[
\#^{q+p}C \#^\omega \quad \xrightarrow{\text{G}} \quad \#^{q+p}C \#^\omega
\]

where

\[
n(u) = n(v) \overset{\text{def}}{=} (u = v = 0) \cup (u = v = 1 \land (u = v = 1 \cup u = v = 0))
\]
SPEC\(_3\): if \(n(u) = n(v)\) are synchronized then \(x = y\)

\[
n(u) = n(v) \quad \xrightarrow{\text{def}} \quad (u = v = 0) \lor (u = v = 1 \land (u = v = 1 \lor u = v = 0))
\]
Undecidability proof 4

**SPEC$_4$:** if $n(u) = n(v) + 1$ are synchronized then $C_y \models C_x$

\[
\begin{align*}
0^q1^{p+1}0\ldots &\quad u \\
&\quad \Downarrow \quad a \\
&\quad \Downarrow \quad x \\
0^q1^{p+1}0\ldots &\quad v \\
&\quad \Downarrow \quad b \\
&\quad \Downarrow \quad y \\
\#q+p+1C_x\#^\omega &\quad \Rightarrow \\
\#q+p+1C_y\#^\omega
\end{align*}
\]

\[
n(u) = n(v) + 1 \quad \rightarrow \quad x = y \cup \left(\text{Trans}(y, x) \land X^3 G x = y\right)
\]

where $\text{Trans}(y, x)$ is defined by

\[
\bigvee_{(p, a, q, b, \leftarrow) \in T, c \in \Gamma} (y = cpa \land x = qcb) \quad \lor \quad \bigvee_{(p, a, q, b, \rightarrow) \in T, c \in \Gamma} (y = pac \land x = bqc) \quad \lor \quad \bigvee_{(p, a, q, b, \rightarrow) \in T} (y = pa\# \land x = bq\Box)
\]
Undecidability proof 4

**SPEC₄**: if \( n(u) = n(v) + 1 \) are synchronized then \( C_y \vdash C_x \)

\[
0^q 1^{p+1} 0 \ldots \quad u \quad 0^q 1^{p+1} 0 \ldots
\]

\[
\begin{array}{cc}
\text{a} & \text{b} \\
\downarrow & \downarrow \\
\text{x} & \text{y} \\
\end{array}
\]

\[
\#^{q+p+1} C_x \#^\omega \quad \text{and} \quad \#^{q+p+1} C_y \#^\omega
\]

\[
n(u) = n(v) + 1 \quad \longrightarrow \quad x = y \cup \left( \text{Trans}(y, x) \land X^3 G x = y \right)
\]

where \( \text{Trans}(y, x) \) is defined by

\[
\bigvee_{(p,a,q,b,\leftarrow) \in T, c \in \Gamma} (y = cp a \land x = q c b) \quad \bigvee_{(p,a,q,b,\rightarrow) \in T, c \in \Gamma} (y = p a c \land x = b q c)
\]

\[
\bigvee_{(p,a,q,b,\leftarrow) \in T} (y = p a \# \land x = b q \Box)
\]
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Corollary

Specifications 1-4 and 5: $G \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Corollary

Specifications 1-4 and 5: $G \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Induction

Corollary

Specifications 1-4 and 5: $G x \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

$0^{q+1}1^{p}0\ldots$ $u$ $v$ $0^{q+1}1^{p}0\ldots$

Induction $\#^{q+p+1}C_p\#^{\omega}$ $a$ $b$ $\#^{q+p+1}C_p\#^{\omega}$

SPEC$_3$

Corollary

Specifications 1-4 and 5: $Gx \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
**Lemma: winning strategies must simulate the Turing machine**

For each \( p \geq 1 \), if \( n(u) = p \) then \( C_x = C_p \) is the \( p \)-th configuration of the Turing machine starting from the empty tape.

**Proof**

\[
\begin{align*}
0^q1^{p+1}0\cdots & \quad u \quad v \quad 0^q1^{p+1}1^p0\cdots \\
\text{SPEC}_4 & \quad \text{SPEC}_3 \\
\#^{q+p+1}C_{p+1}\#^\omega & \quad \#^{q+p+1}C_p\#^\omega
\end{align*}
\]

**Corollary**

Specifications 1-4 and 5: \( G_x \neq \text{stop} \) are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Specifications 1-4 and 5: $Gx \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Communication allows to cheat

Architecture with communication

- **Strategy for $a$:**
  - copy $u$ to $z$
  - if $u = 0^q 1^p 0 \cdots$ then $x = \begin{cases} 
  \#^p q C_1 \#^\omega & \text{if } p = 1 \text{ (for SPEC}_2) \\
  \#^p q C_2 \#^\omega & \text{otherwise (for SPEC}_4). 
\end{cases}$

- **Strategy for $b$:** if $z = 0^{q'} 1^{p'} 0 \cdots$ and $v = 0^q 1^p 0 \cdots$ then

  $y = \begin{cases} 
  \#^p q C_1 \#^\omega & \text{if } p = 1 \text{ (for SPEC}_2) \\
  \#^p q C_2 \#^\omega & \text{if } p = p' > 1 \text{ and } q = q' \text{ (for SPEC}_3) \\
  \#^p q C_1 \#^\omega & \text{otherwise (for SPEC}_4). 
\end{cases}$
More undecidable architectures

Exercices

1. Show that the architecture below is undecidable.

![Diagram](attachment:image.png)

2. Show that the undecidability results also hold for CTL specifications
Uncomparable information

**Definition**

For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

**Definition**

An architecture has **uncomparable information** if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$. Otherwise it is said to have **preordered information**.
**Uncomparable information**

**Definition**

For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

**Definition**

An architecture has **uncomparable information** if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$. Otherwise it is said to have **preordered information**.
**Uncomparable information**

**Definition**
For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

**Definition**
An architecture has **uncomparable information** if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$. Otherwise it is said to have **preordered information**.
Uncomparable information

Definition

For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

Definition

An architecture has uncomparable information if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$. Otherwise it is said to have preordered information.
Definition

For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

Definition

An architecture has uncomparable information if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$. Otherwise it is said to have preordered information.
Uncomparable information yields undecidability

Theorem
Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

Proof for LTL specifications
Theorem

Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

Proof for LTL specifications

\[ x_0 \rightarrow y_0 \]
\[ x_1 \rightarrow y_1 \]

\[ x_0 \rightarrow y_0 \rightarrow x_1 \rightarrow y_1 \]
Uncomparable information yields undecidability

**Theorem**

Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

**Proof for LTL specifications**
The synthesis problem for pipeline architectures and LTL specifications is non-elementary decidable.

Pnueli-Rosner (FOCS’90)
Decidability proof 1

### From distributed to global

If \( f_y : Q^+_x \rightarrow Q_y \) and \( f_z : Q^+_y \rightarrow Q_z \) are local (distributed) strategies then we can define an equivalent global strategy \( h = f_y \otimes f_z : Q^+_x \rightarrow Q_y \times Q_z \) by

\[
h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \quad \text{where} \quad y_i = f_y(x_1, \cdots, x_i).
\]

### From global to distributed

\( z \) should only depend on \( y \).
We cannot transmit \( x \) to \( y \) if \( |Q_y| < |Q_x| \).
We have to check whether there exists a global strategy that can be distributed.
Decidability proof 1

Pipeline

From distributed to global

If \( f_y : Q_x^+ \rightarrow Q_y \) and \( f_z : Q_y^+ \rightarrow Q_z \) are local (distributed) strategies then we can define an equivalent global strategy \( h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z \) by

\[
h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \quad \text{where} \quad y_i = f_y(x_1, \cdots, x_i).
\]

From global to distributed

\( z \) should only depend on \( y \).
We cannot transmit \( x \) to \( y \) if \( |Q_y| < |Q_x| \).
We have to check whether there exists a global strategy that can be distributed.
Decidability proof 1

From distributed to global
If \( f_y : Q_x^+ \to Q_y \) and \( f_z : Q_y^+ \to Q_z \) are local (distributed) strategies then we can define an equivalent global strategy \( h = f_y \otimes f_z : Q_x^+ \to Q_y \times Q_z \) by

\[
h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \quad \text{where} \quad y_i = f_y(x_1, \cdots, x_i).
\]

From global to distributed
\( z \) should only depend on \( y \).
We cannot transmit \( x \) to \( y \) if \(|Q_y| < |Q_x|\).
We have to check whether there exists a global strategy that can be distributed.
Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$. 
1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$. 
Tree automata

Non deterministic transitions

Alternating transitions

or
Tree automata

non deterministic transitions

Alternating transitions

or
Tree automata

**Non deterministic transitions**

```
    p  a
   / \
  1   2
 / \  /  \
a1  a  a1  a2
/ \  /  \
 p1 p2 p1 p2
```

**Alternating transitions**

```
    a
   / \
  1   2
 / \  /  \
 a1 a2 a1 a2
```

or
Tree automata

**Non deterministic transitions**

\[
p a
\]

\[
1 \quad 2
\]

\[
a_1 p_1 a_2 p_2
\]

**Alternating transitions**

\[
p a
\]

\[
1 \quad 2
\]

\[
a_1 p_1 a_2 p_2 \wedge p_3
\]

or
Tree automata

**Non Deterministic Transitions**

```
  p a
   1  2
  a_1 a_2
 p_1 p_2
```

**Alternating Transitions**

```
  p a
   1 2
  a_1 a_2
 p_1 p_2 \lor p_3
```

or

```
  p a
   1 2 2
  a_1 a_2 a_2
 p_1 p_2 p_3
```
1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$. 

Proof

\[
\begin{array}{cccc}
\text{x} & \xrightarrow{a} & \text{y} & \xrightarrow{b} \text{z} \\
\text{x} & \xrightarrow{a \ & b} \text{y} & \text{z}
\end{array}
\]
Decidability proof 3

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$.
Decidability proof 3

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q^+_x \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q^+_y \rightarrow Q_z$ iff there exists a local strategy $f_y : Q^+_x \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q^+_x \rightarrow Q_y \times Q_z$ is accepted by $A$. 

\[
A \\
p \ (y, z) \\
/ \ x \\
x_1 \\
(y_1, z_1) \\
p_1 \\
x_2 \\
(y_2, z_2) \\
p_2 \\
x_3 \\
(y_2, z_2) \\
p_3 \\
A' \\
(x, p) \ (y, z) \\
/ y \\
y_1 \\
(z_1) \\
(x_1, p_1) \\
y_2 \\
(z_2) \\
(x_2, p_2) \\
y_2 \\
(z_2) \\
(x_3, p_3)
Decidability proof 4

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$.

3. Transform the alternating TA $A'$ to an equivalent non determinisitic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
Decidability proof 4

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$.

3. Transform the alternating TA $A'$ to an equivalent non deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$.

3. Transform the alternating TA $A'$ to an equivalent non deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
Decidability proof 4

Proof

1. We first solve the global game: We obtain an ND tree-automaton $\mathcal{A}$ accepting the global strategies $h : Q_x^+ \to Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $\mathcal{A}$ an alternating tree automaton $\mathcal{A}'$ accepting a local strategy $f_z : Q_y^+ \to Q_z$ iff there exists a local strategy $f_y : Q_x^+ \to Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \to Q_y \times Q_z$ is accepted by $\mathcal{A}$

3. Transform the alternating TA $\mathcal{A}'$ to an equivalent non determinisitic TA $\mathcal{A}_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q^+_x \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q^+_y \rightarrow Q_z$ iff there exists a local strategy $f_y : Q^+_x \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q^+_x \rightarrow Q_y \times Q_z$ is accepted by $A$

3. Transform the alternating TA $A'$ to an equivalent non determinisitc TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
Decidability proof 4

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$

3. Transform the alternating TA $A'$ to an equivalent non deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^+ \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^+ \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_x^+ \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_x^+ \rightarrow Q_y \times Q_z$ is accepted by $A$

3. Transform the alternating TA $A'$ to an equivalent non deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
Decidability

**Pnueli-Rosner (FOCS’90)**

The synthesis problem for pipeline architectures and LTL specifications is non-elementary decidable.

**Peterson-Reif (FOCS’79)**

multi-person games with incomplete information.

\[ \Rightarrow \text{non-elementary lower bound for the synthesis problem.} \]
Decidability

Kupferman-Vardi (LICS’01)

The synthesis problem is non elementary decidable for

- one-way chain, one-way ring, two-way chain and two-way ring,
- CTL* specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.

one-way chain

```
x -> a1 -> y1 -> a2 -> y2 -> a3 -> y3
   |     |     |     |
  z1   z2   z2   z3
```
Decidability

Kupferman-Vardi (LICS’01)

The synthesis problem is non elementary decidable for

- one-way chain, one-way ring, two-way chain and two-way ring,
- CTL* specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.

one-way ring

\[
\begin{array}{ccccc}
  x & \rightarrow & a_1 & \rightarrow & y_1 \\
  & \downarrow & & \downarrow & \\
  & z_1 & & & \\
  y_1 & \rightarrow & a_2 & \rightarrow & y_2 \\
  & \downarrow & & \downarrow & \\
  y_2 & \rightarrow & a_3 & \rightarrow & y_3 \\
  & \downarrow & & \downarrow & \\
  & z_3 & & & \\
\end{array}
\]
Decidability

Kupferman-Vardi (LICS’01)

The synthesis problem is non elementary decidable for

- one-way chain, one-way ring, two-way chain and two-way ring,
- $\text{CTL}^*$ specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.

two-way chain

\[
\begin{array}{c}
\xrightarrow{} \quad a_1 \quad \xleftarrow{} \\
\downarrow \quad y_1 \quad \downarrow \\
\quad \quad z_1 \\
\xrightarrow{} \quad a_2 \quad \xleftarrow{} \\
\downarrow \quad y_1' \quad \downarrow \\
\quad \quad z_2 \\
\xrightarrow{} \quad a_3 \quad \xleftarrow{} \\
\downarrow \quad y_2 \quad \downarrow \\
\quad \quad z_3 \\
\xrightarrow{} \quad a_4 \quad \xleftarrow{} \\
\downarrow \quad y_3 \quad \downarrow \\
\quad \quad z_4
\end{array}
\]
Example

Programs: $f_x : Q_u^* \rightarrow Q_x$ and $f_z : (Q_x \times Q_v)^* \rightarrow Q_z$.

- Input: \[
\begin{pmatrix}
  u_1 & u_2 & u_3 & \cdots \\
  v_1 & v_2 & v_3 & \cdots 
\end{pmatrix}
\in (Q_u \times Q_v)\omega.
\]

- Behavior: \[
\begin{pmatrix}
  u_1 & u_2 & u_3 & \cdots \\
  v_1 & v_2 & v_3 & \cdots \\
  x_1 & x_2 & x_3 & \cdots \\
  z_1 & z_2 & z_3 & \cdots 
\end{pmatrix}
\]

with \[
\begin{cases}
  x_{n+1} = f_x(u_1 \cdots u_n) \\
  z_{n+1} = f_z((x_1, v_1) \cdots (x_n, v_n))
\end{cases}
\text{ for all } n > 0.
Decidability

Adequately connected sub-architecture

\[ Q_x = Q \text{ for all } x \in \mathcal{V} \]

\begin{tikzpicture}
  \node (a) at (0,0) [shape=rectangle] \(a\);
  \node (b) at (1,1) [shape=rectangle] \(b\);
  \node (c) at (1,-1) [shape=rectangle] \(c\);
  \node (d) at (0,2) [shape=circle] \(u\);
  \node (e) at (2,1) [shape=circle] \(y\);
  \node (f) at (2,-1) [shape=circle] \(z\);
  \node (g) at (-2,1) [shape=circle] \(v\);

  \draw[->] (d) -- (a);
  \draw[->] (a) -- (b);
  \draw[->] (b) -- (e);
  \draw[->] (a) -- (f);
  \draw[->] (b) -- (f);
  \draw[->] (c) -- (g);
  \draw[->] (c) -- (f);
  \draw[->] (g) -- (d);
\end{tikzpicture}

Pnueli-Rosner (FOCS’90)

- An adequately connected architecture is equivalent to a singleton architecture.
- The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.
Decidability

An adequately connected sub-architecture is equivalent to a singleton architecture.

The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.

Pnueli-Rosner (FOCS‘90)

\[ \mathcal{Q}_x = \mathcal{Q} \text{ for all } x \in \mathcal{V} \]
Decidability

An adequately connected architecture is equivalent to a singleton architecture.

The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.
Decidability

An adequately connected architecture is equivalent to a singleton architecture.

The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.

\[ Q_x = Q \text{ for all } x \in \mathcal{V} \]
Information fork criterion
(Finkbeiner–Schewe LICS ’05)
Information fork criterion
(Finkbeiner–Schewe LICS ’05)
Information fork criterion
(Finkbeiner–Schewe LICS ’05)
Uniformly well connected architectures

Definition

An architecture is uniformly well connected if there is a uniform way to route variables in \( \text{View}(y) \) to \( y \) for each output variable \( y \).

Example

[Diagram showing a network with nodes labeled 'u', 'v', 'w', 'p1', 's', 'p2', 't', 'p3', 'p4', 'p5', 'x', 'y', 'z']
Definition

An architecture is uniformly well connected if there is a uniform way to route variables in $\text{View}(y)$ to $y$ for each output variable $y$.

Example

![Diagram of uniformly well connected architectures](attachment:image.png)
Uniformly well connected architectures

Definition
An architecture is uniformly well connected if there is a uniform way to route variables in \( \text{View}(v) \) to \( v \) for each output variable \( v \).

- If the capacity of internal variables is big enough then the architecture is uniformly well-connected.
- If the architecture is uniformly well-connected then we can use causal strategies instead of local ones.

Proposition
Checking whether a given architecture is uniformly well connected is NP-complete.

Proof
Reduction to the multicast problem in Network Information Flow. The multicast problem is NP-complete (Rasala Lehman-Lehman 2004).
Uniformly well connected architectures

**Definition**
An architecture is uniformly well connected if there is a uniform way to route variables in $\text{View}(v)$ to $v$ for each output variable $v$.

- If the **capacity of internal variables is big enough** then the architecture is uniformly well-connected.
- If the architecture is **uniformly well-connected** then we can use causal strategies instead of local ones.

**Proposition**
Checking whether a given architecture is uniformly well connected is NP-complete.

**Proof**
Reduction to the multicast problem in Network Information Flow. The multicast problem is NP-complete (Rasala Lehman-Lehman 2004).
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

Theorem: Kupferman-Vardi (LICS'01)
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

\[ \begin{array}{cccc}
    x_1 & x_2 & x_3 & x_4 \\
    \downarrow & \downarrow & \downarrow & \downarrow \\
    y_1 & y_2 & y_3 & y_4 \\
\end{array} \]

Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Robust specifications

**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**


## Definition

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

## Theorem

The synthesis problem for **uniformly well-connected** architectures and external and **robust** $\text{CTL}^*$ specifications is decidable.

## Proof.
Robust specifications

Definition
A specification $\varphi$ is robust if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

Theorem
The synthesis problem for uniformly well-connected architectures and external and robust $\text{CTL}^*$ specifications is decidable.

Proof.

\[
\begin{array}{cccc}
  x_1 & x_2 & x_3 & x_4 \\
  \downarrow & \downarrow & \downarrow & \downarrow \\
  \text{ } & \text{ } & \text{ } & \text{ } \\
  \downarrow & \downarrow & \downarrow & \downarrow \\
  y_1 & y_2 & y_3 & y_4 \\
\end{array}
\]
Robust specifications

**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

![Diagram](attachment:image.png)
**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

![Diagram](image)
Robust specifications

**Definition**
A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**
The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

![Diagram showing the robustness of specifications with nodes and arrows illustrating the connection between $x$ and $y$ variables.](image)
Robust specifications

**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

![Diagram](https://via.placeholder.com/150)
Open problem

- Decidability of the distributed control/synthesis problem for robust and external specifications.
Outline

Control for sequential systems

Control for distributed systems

Synchronous semantics

Asynchronous semantics
An example: Romeo and Juliet

Romeo and Juliet against the environment

- Want to communicate through the same communication line.
- At any time, one line is broken.
- Environment looks where R&J are connected, and then, atomically, changes (possibly) the broken line.
- Romeo/Juliet looks status of lines and, atomically, chooses where to connect.
Romeo and Juliet against the environment

- Want to communicate through the same communication line.
- At any time, one line is broken.
- Environment looks where R&J are connected, and then, atomically, changes (possibly) the broken line.
- Romeo/Juliet looks status of lines and, atomically, chooses where to connect.
An example: Romeo and Juliet

Romeo and Juliet against the environment

- Want to communicate through the same communication line.
- At any time, one line is broken.
- Environment looks where R&J are connected, and then, atomically, changes (possibly) the broken line.
- Romeo/Juliet looks status of lines and, atomically, chooses where to connect.
An example: Romeo and Juliet

Romeo and Juliet against the environment

- Want to communicate through the same communication line.
- At any time, one line is broken.
- Environment looks where R&J are connected, and then, atomically, changes (possibly) the broken line.
- Romeo/Juliet looks status of lines and, atomically, chooses where to connect.
Romeo and Juliet (continued)

Architecture

- Variables:
  - $x_1$: Romeo’s current line.
  - $x_2$: broken line
  - $x_3$: Juliet’s current line.

- Agents: Romeo, Juliet and Environment.

- Read/Write table

<table>
<thead>
<tr>
<th></th>
<th>Romeo</th>
<th>Juliet</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>${x_1, x_2}$</td>
<td>${x_2, x_3}$</td>
<td>${x_1, x_2, x_3}$</td>
</tr>
<tr>
<td>Write</td>
<td>${x_1}$</td>
<td>${x_3}$</td>
<td>${x_2}$</td>
</tr>
</tbody>
</table>

- $Q_1 = \{1, 2, 3, 4\}$
- $Q_2 = \{1, 2, 3, 4\}$
- $Q_3 = \{1, 2, 3, 4\}$

- □ read-write ability
- ○ read-only ability
Legal moves: $\delta_a \subseteq Q_{R(a)} \times Q_{W(a)}$

$x_1 : 3$
$x_2 : 1 \rightarrow E \rightarrow x_2 : 4$
$x_3 : 4$

$x_1 : 1 \rightarrow R \rightarrow x_1 : 3$
$x_2 : 1$

A distributed play of the asynchronous system, R & J against E
Romeo and Juliet (continued)

Legal moves: $\delta_a \subseteq Q_{R(a)} \times Q_{W(a)}$

$x_1: 3$  
$x_2: 1$  
$x_3: 4$

A distributed play of the asynchronous system, $R$ & $J$ against $E$

$x_1$  
$x_2$  
$x_3$
Distributed Behaviors

A play is a Mazurkiewicz (real) trace

- A finite play:

- Move: extension of the current Mazurkiewicz trace following the rules.
- The game is not “position based”, nor “turn based”.
- Winning condition: set of finite or infinite Mazurkiewicz traces $W \subseteq R(\Sigma, D)$. Team $0$ wins plays of $W$ and loses plays of $R(\Sigma, D) \setminus W$.

Romeo and Juliet

$\mathcal{W}$ imposes fairness conditions to the environment.
Memory for strategies

Memory

- Each player only has a partial view of the global history.
- Memoryless: move can depend only on the current state.
- Local memory: a player can remember its read history.

Causal memory (intuitively, maximal history a player can observe)

- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.
Memory for strategies

Memory

- Each player only has a partial view of the global history.
- Memoryless: move can depend only on the current state.
- Local memory: a player can remember its read history.

Causal memory (intuitively, maximal history a player can observe)

- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.
Memory for strategies

Memory

- Each player only has a partial view of the global history.
- Memoryless: move can depend only on the current state.
- Local memory: a player can remember its read history.

Causal memory (intuitively, maximal history a player can observe)

- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.
Memory for strategies

Memory

- Each player only has a partial view of the global history.
- Memoryless: move can depend only on the current state.
- Local memory: a player can remember its read history.

Causal memory (intuitively, maximal history a player can observe)

- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.
Memory for strategies

Memory

- Each player only has a partial view of the global history.
- Memoryless: move can depend only on the current state.
- Local memory: a player can remember its read history.

Causal memory (intuitively, maximal history a player can observe)

- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.
Winning strategies

Tuple \((f_a)_{a \in \mathcal{P}_0}\) where \(f_a\) tells player \(a \in \mathcal{P}_0\) how to play.

- **Memoryless**
  \[ f_a : Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

- **Local memory**
  \[ f_a : (Q_{R(a)})^* Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

- **Causal memory**
  \[ f_a : \mathcal{M}(\Sigma, D) \times Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

### Winning strategies

A strategy \(f\) is winning in \(G\) if all \(f\)-maximal \(f\)-plays in \(G\) are in \(\mathcal{W}\).

---

**f-maximal f-plays**

Given a strategy \(f = (f_a)_{a \in \mathcal{P}_0}\), one looks at plays \(t\) which are

- **consistent** with \(f\): all \(a\)-moves played according to \(f_a\) (\(f\)-play).
- **maximal**: \(f\) predicts to Stop for all \(a\)-moves enabled at \(t\) with \(a \in \mathcal{P}_0\).
Winning strategies

Tuple \((f_a)_{a \in \mathcal{P}_0}\) where \(f_a\) tells player \(a \in \mathcal{P}_0\) how to play.

- **Memoryless**
  \[ f_a : Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

- **Local memory**
  \[ f_a : (Q_{R(a)})^* Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

- **Causal memory**
  \[ f_a : \mathbb{M}(\Sigma, D) \times Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

---

**\(f\)-maximal \(f\)-plays**

Given a strategy \(f = (f_a)_{a \in \mathcal{P}_0}\), one looks at plays \(t\) which are

- **consistent** with \(f\): all \(a\)-moves played according to \(f_a\) (\(f\)-play).
- **maximal**: \(f\) predicts to \text{Stop} for all \(a\)-moves enabled at \(t\) with \(a \in \mathcal{P}_0\).

---

**Winning strategies**

A strategy \(f\) is winning in \(G\) if all \(f\)-maximal \(f\)-plays in \(G\) are in \(\mathcal{W}\).
Finite abstraction of the causal memory

Distributed memory

A distributed memory is a mapping $\mu : M(\Sigma, D) \rightarrow M$ satisfying the following equivalent properties:

1. $\mu^{-1}(m)$ is recognizable for each $m \in M$,
2. $\mu$ is an abstraction of an asynchronous mapping (cf. Zielonka),
3. $\mu$ can be computed in a distributed way (allowing additional contents inside existing communications (piggy-backing), but no extra communications).

Strategy with memory $\mu$

\[ f_a : M \times Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \]

the associated strategy is defined by

\[ f_a^\mu(t, q) = f_a(\mu(t), q) \]

If $M$ is finite then $f^\mu$ is a distributed strategy with finite memory.

If $|M| = 1$ then $f^\mu$ is memoryless.
Proposition: PG-Lerman-Zeitoun (LATIN’04)

For a distributed game $G$ and a distributed memory $\mu$, one can build a game $G^\mu$ such that

\[
\text{team 0 has a WDS in } G \text{ with memory } \mu
\]

iff

\[
\text{team 0 has a memoryless WDS in } G^\mu.
\]

Proof.

\[
G^\mu = G \times \mu
\]
Given a finite distributed game \((G, W)\), we can effectively build a finite sequential 2-players game \((\tilde{G}, \tilde{W})\) st. the following are equivalent:

- There exists a memoryless distributed WS for team 0 in \((G, W)\).
- There exists a memoryless WS for player 0 in \((\tilde{G}, \tilde{W})\).
- There exists a WS for player 0 in \((\tilde{G}, \tilde{W})\).

Moreover, if \(W\) is recognizable then so is \(\tilde{W}\).

Naive idea: Consider the game on the global transition system.

Main problem: The controller has more information than its causal memory.

Solution:
- The opponent controls the linearization to be played.
- Using reset moves, he can replay different linearizations for the same play.
- The winning condition \(\tilde{W}\) makes sure that the strategy followed by the controller is indeed distributed.
From distributed to sequential games

Theorem: PG-Lerman-Zeitoun (LATIN’04)

Given a finite distributed game \((G, W)\), we can effectively build a finite sequential 2-players game \((\tilde{G}, \tilde{W})\) st. the following are equivalent:

- There exists a **memoryless distributed** WS for team 0 in \((G, W)\).
- There exists a memoryless WS for player 0 in \((\tilde{G}, \tilde{W})\).
- There exists a WS for player 0 in \((\tilde{G}, \tilde{W})\).

Moreover, if \(W\) is recognizable then so is \(\tilde{W}\).

**Naive idea** Consider the game on the global transition system.

**Main problem** The controller has more information than its causal memory.

**Solution**

- The opponent controls the linearization to be played.
- Using reset moves, he can replay different linearizations for the same play.
- The winning condition \(\tilde{W}\) makes sure that the strategy followed by the controller is indeed distributed.
From distributed to sequential games

**Theorem:** PG-Lerman-Zeitoun (LATIN’04)

Given a finite distributed game \((G, \mathcal{W})\), we can effectively build a finite sequential 2-players game \((\tilde{G}, \tilde{\mathcal{W}})\) st. the following are equivalent:

- There exists a memoryless distributed WS for team 0 in \((G, \mathcal{W})\).
- There exists a memoryless WS for player 0 in \((\tilde{G}, \tilde{\mathcal{W}})\).
- There exists a WS for player 0 in \((\tilde{G}, \tilde{\mathcal{W}})\).

Moreover, if \(\mathcal{W}\) is recognizable then so is \(\tilde{\mathcal{W}}\).

**Naive idea** Consider the game on the global transition system.

**Main problem** The controller has more information than its causal memory.

**Solution**

- The opponent controls the linearization to be played.
- Using reset moves, he can replay different linearizations for the same play.
- The winning condition \(\tilde{\mathcal{W}}\) makes sure that the strategy followed by the controller is indeed distributed.
Proposition: (Folklore)
Deciding whether team 0 has a distributed WS with causal memory is undecidable for rational winning conditions.

Proof. Simple reduction of the universality problem for rational trace languages.

Deciding whether team 0 has a distributed WS with local memory is undecidable even:
- for reachability or safety winning conditions.
- with 3 players against the environment.
Proposition: (Folklore)

Deciding whether team $0$ has a distributed WS with causal memory is undecidable for rational winning conditions.

Proof. Simple reduction of the universality problem for rational trace languages.

Deciding whether team $0$ has a distributed WS with local memory is undecidable even:

- for reachability or safety winning conditions.
- with 3 players against the environment.
**Series-parallel architectures**

**Theorem:** PG-Lerman-Zeitoun (FSTTCS’04)

Distributed games with **recognizable** winning conditions are decidable for **series-parallel** systems and **causal** memory strategies.

**Definition:** let $A = (P, V, R, W)$ be some architecture.

- $A$ is a **parallel product** if $P = A \cup B$ with $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in A \times B$.

- $A$ is a **serial product** if $P = A \cup B$ with $R(a) \cap W(b) \neq \emptyset$ for all $(a, b) \in A \times B$.

- $A$ is **series-parallel** if it can be obtained from singletons ($|P| = 1$) using serial and parallel compositions.

- $A$ is series-parallel iff the associated dependence relation does not contain a $P_4$: $a \rightarrow b \rightarrow c \rightarrow d$ as induced subgraph.

- Behaviors of series parallel architectures are series-parallel posets.
Series-parallel architectures

Theorem: PG-Lerman-Zeitoun (FSTTCS’04)

Distributed games with recognizable winning conditions are decidable for series-parallel systems and causal memory strategies.

Definition: let \( A = (\mathcal{P}, \mathcal{V}, R, W) \) be some architecture.

- \( A \) is a **parallel product** if \( \mathcal{P} = A \uplus B \) with \( R(a) \cap W(b) = \emptyset \) for all \( (a, b) \in A \times B \).
- \( A \) is a **serial product** if \( \mathcal{P} = A \uplus B \) with \( R(a) \cap W(b) \neq \emptyset \) for all \( (a, b) \in A \times B \).
- \( A \) is **series-parallel** if it can be obtained from singletons \( |\mathcal{P}| = 1 \) using serial and parallel compositions.
- \( A \) is series-parallel iff the associated dependence relation does not contain a \( P_4: a \rightleftarrows b \rightleftarrows c \rightleftarrows d \) as induced subgraph.
- Behaviors of series parallel architectures are series-parallel posets.
**Proof outline**

Team 0 has a WDS $\Rightarrow$ it has a WDS with a “small” distributed memory.

**Induction on $\Sigma$.**

Difficult case: serial product.

1. A WS on $A \sqcup B$ induces WS on the restrictions of the game to $A$ and $B$.
2. Replace the WS on $A$, $B$ by WS with small memory (induction).
3. Finally, glue together these WS on $A$ and $B$ to obtain a WS on $A \sqcup B$ using small memory.

**Main problem**

- Team 0 must know on which small game it is playing.
- Team 0 has to compute this information in a distributed way.
Setting

- Architecture: $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$ with $R(a) = W(a)$ for all $a \in \mathcal{P}$.
- Moves: $\delta_a$ are built from local moves for variables $\delta_{a,x} \subseteq Q_x \times Q_x$:
  \[
  \delta_a = \prod_{x \in R(a)} \delta_{a,x}
  \]
- Strategies with local memory: associated with variables and not with agents, and only predict the next actions and not the next state:
  \[
  f_x : Q_x^* \rightarrow 2^{R^{-1}(x)}
  \]
  action $a$ is enabled by $(f_x)_{x \in \mathcal{V}}$ at some finite play $t$ if
  \[
  \forall x \in R(a), \quad a \in f_x(\pi_{Q_x}(t))
  \]
- The environment decides which $a$-transition should be taken among the actions $a$ enabled by the strategies.
Restricted control synthesis problem

Given a distributed system and a recognizable specification,

Question existence of a clocked and com-rigid non-blocking winning distributed strategy with local memory.

- **clocked**: $f_x(w)$ only depends on $|w|$.
- **com-rigid**: $a, b \in f_x(w)$ implies $R(a) = R(b)$.

Theorem

1. The restricted control synthesis problem is decidable.
2. It becomes undecidable if one of the red condition is dropped.
Restricted control synthesis problem

Given a distributed system and a recognizable specification,

Question existence of a clocked and com-rigid non-blocking winning distributed strategy with local memory.

- **clocked**: $f_x(w)$ only depends on $|w|$.
- **com-rigid**: $a, b \in f_x(w)$ implies $R(a) = R(b)$.

Theorem

1. The **restricted** control synthesis problem is decidable.
2. It becomes undecidable if one of the **red** condition is dropped.
Restrictions

- Controllable actions: \( R(a) = W(a) \) is a singleton for all \( a \in \mathcal{P}_0 \).
- Environment actions: \( R(e) = W(e) = \mathcal{V} \) and \( \mathcal{P}_1 = \{e\} \).
- Moves: \( \delta_e \subseteq Q_{\mathcal{V}} \times Q_{\mathcal{V}} \).
- Strategies: local memory with stuttering reduction so that a player \( a \in \mathcal{P}_0 \) cannot see how long it has been idle.

Theorem

- Previous settings with local memory can be encoded.
- Two constructions to solve the distributed control problem subsuming previously known decidable cases with local memory.
Open problems

- Generalization to arbitrary symmetric architectures.
- Generalization to non-symmetric architectures.
- Reasonable upper bounds for synthesis?
Symmetric architecture

Architecture \( \mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W) \)

- Restrictions:
  \[
  \forall a \in \mathcal{P} \quad \emptyset \neq W(a) \subseteq R(a) \\
  \forall a, b \in \mathcal{P} \quad R(a) \cap W(b) \neq \emptyset \iff R(b) \cap W(a) \neq \emptyset
  \]

- Dependence:
  \( a \ D \ b \iff R(a) \cap W(b) \neq \emptyset \iff R(b) \cap W(a) \neq \emptyset \)

Legal and forbidden architectures

- OK
- Forbidden (not symmetric)