Distributed synthesis: synchronous and asynchronous semantics

Paul Gastin

LSV

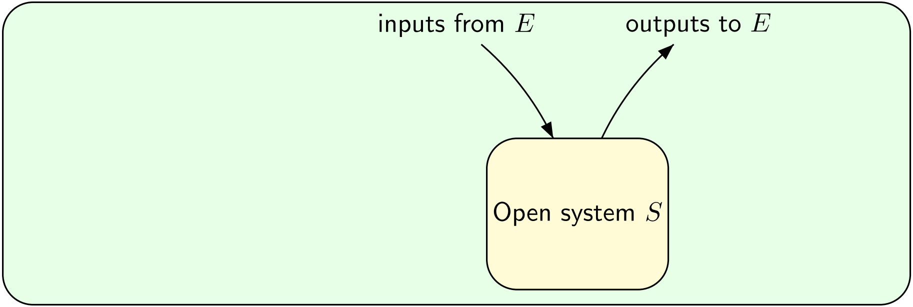
ENS de Cachan & CNRS

Paul.Gastin@lsv.ens-cachan.fr

EPIT, May 31st, 2006

Outline

Open / Reactive system



Model for the open system

- Transitions system $\mathcal{A} = (Q, \Sigma, q_0, \delta)$
  - Q: finite or infinite set of states,
  - δ: 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:

	Σ_o	Σ_{uo}
Σ_{uc}	call level i	enter/exit cabin
Σ_c	open/close door i move 1 level up/down	

- We get easily a finite and deterministic transition system.

Specification

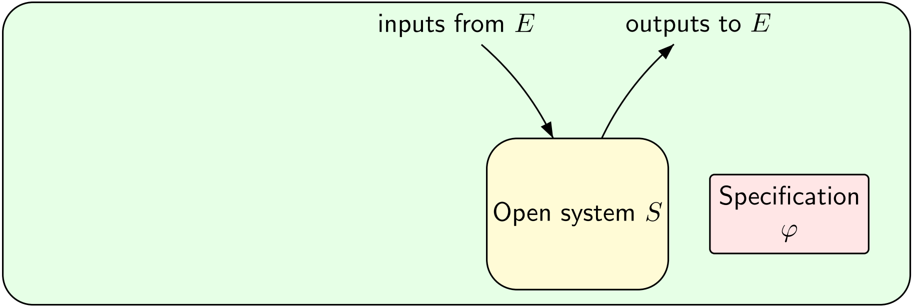
Linear time: LTL, FO, MSO, regular, ...

- Safety: $\mathsf{G}(\texttt{level} \neq i \longrightarrow \texttt{is\_closed}_i)$
- Liveness: $\mathsf{G}(\texttt{is\_called}_i \longrightarrow \mathsf{F}(\texttt{level} = i \wedge \texttt{is\_open}_i))$

Branching time: CTL, CTL*, μ-calculus, ...

- $\mathsf{AG}\langle\texttt{call}_i\rangle\top$ ($\texttt{call}_i$ is uncontrollable)
- $\mathsf{AG}\,\mathsf{EF}(\texttt{level} = 0 \wedge \texttt{is\_open}_0)$

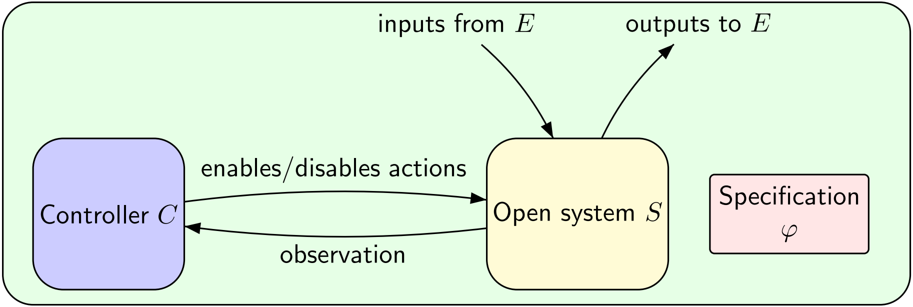
Control problem



Two problem

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

Control problem



Two problem

- Control: Given a system S and a specification φ, decide whether there exists a controller C such that $S \otimes C \models \varphi$.
- Synthesis: Given a system S and a specification φ, builda 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 $a_i \in f(q_0 a_1 q_1 \cdots q_{i-1})$ for all $i > 0$.
- Controlled execution tree: $t : D^* \to \Sigma \times Q$ with
  - $t(\varepsilon) = (a, q_0)$ $\quad$ ($a \in \Sigma$ 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 $t(\mathtt{sons}(x)) = \{(a, q) \mid a \in f(q_0 a_1 q_1 \cdots a_n q_n)$ 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: $f : Q(\Sigma Q)^* \to 2^\Sigma$ 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 $a_i \in f(q_0 a_1 q_1 \cdots q_{i-1})$ for all $i > 0$.
- Controlled execution tree: $t : D^* \to \Sigma \times Q$ with
  - $t(\varepsilon) = (a, q_0)$ $\qquad$ ($a \in \Sigma$ 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 $t(\mathtt{sons}(x)) = \{(a, q) \mid a \in f(q_0 a_1 q_1 \cdots a_n q_n) \text{ and } (q_n, a, q) \in \delta\}$.

Under full event observation

- Controller: $f : \Sigma^* \to 2^\Sigma$ 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: $f : Q(\Sigma Q)^* \to 2^\Sigma$ 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 $a_i \in f(q_0 a_1 q_1 \cdots q_{i-1})$ for all $i > 0$.
- Controlled execution tree: $t : D^* \to \Sigma \times Q$ with
  - $t(\varepsilon) = (a, q_0)$ $\qquad$ ($a \in \Sigma$ 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 $t(\mathtt{sons}(x)) = \{(a, q) \mid a \in f(q_0 a_1 q_1 \cdots a_n q_n)$ 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: $f : Q(\Sigma Q)^* \to 2^\Sigma$ 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 $a_i \in f(q_0 a_1 q_1 \cdots q_{i-1})$ for all $i > 0$.
- Controlled execution tree: $t : D^* \to \Sigma \times Q$ with
  - $t(\varepsilon) = (a, q_0)$ $\qquad$ ($a \in \Sigma$ 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 $t(\mathtt{sons}(x)) = \{(a, q) \mid a \in f(q_0 a_1 q_1 \cdots a_n q_n)$ 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 $i > 0$.

Control versus Game

Correspondance

Transition system	$=$	Game arena (graph).
Controllable events	$=$	Actions of player 1 (controller).
Uncontrollable events	$=$	Action of player 0 (opponent, environment).
Behavior	$=$	Play.
Controller	$=$	Strategy.
Specification	$=$	Winning condition.
Finding a controller	$=$	finding a winning strategy.

Control problem

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

Theorem

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

Moreover, when (S, φ) is controllable, we can synthesize a finite state controller.

Control versus Game

Correspondance

Transition system	$=$	Game arena (graph).
Controllable events	$=$	Actions of player 1 (controller).
Uncontrollable events	$=$	Action of player 0 (opponent, environment).
Behavior	$=$	Play.
Controller	$=$	Strategy.
Specification	$=$	Winning condition.
Finding a controller	$=$	finding a winning strategy.

Control problem

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

Theorem

If the system is finite state and the specification is regular then the control problem is decidable.
Moreover, when (S, φ) is controllable, we can synthesize a finite state controller.

Ramadge - Wonham 87$\longrightarrow$

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

- $(S, \mathrm{Pref}(K))$ is controllable iff $\mathrm{Pref}(K) \cdot \Sigma_{uc} \cap \mathrm{Pref}(\mathcal{L}(S)) \subseteq \mathrm{Pref}(K)$.
- (S, K) is controllable without deadlock iff
  - $\mathrm{Pref}(K) \cdot \Sigma_{uc} \cap \mathrm{Pref}(\mathcal{L}(S)) \subseteq \mathrm{Pref}(K)$
  - $\mathrm{Pref}(K) \cap \mathcal{L}(S) = K$.
- 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)
- ...

Ramadge - Wonham 87→

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

- $(S, \mathrm{Pref}(K))$ is controllable iff $\mathrm{Pref}(K) \cdot \Sigma_{uc} \cap \mathrm{Pref}(\mathcal{L}(S)) \subseteq \mathrm{Pref}(K)$.
- (S, K) is controllable without deadlock iff
  - $\mathrm{Pref}(K) \cdot \Sigma_{uc} \cap \mathrm{Pref}(\mathcal{L}(S)) \subseteq \mathrm{Pref}(K)$
  - $\mathrm{Pref}(K) \cap \mathcal{L}(S) = K$.
- 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)
- ...

Ramadge - Wonham 87$\longrightarrow$

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

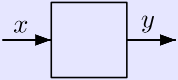
- $(S, \mathrm{Pref}(K))$ is controllable iff $\mathrm{Pref}(K) \cdot \Sigma_{uc} \cap \mathrm{Pref}(\mathcal{L}(S)) \subseteq \mathrm{Pref}(K)$.
- (S, K) is controllable without deadlock iff
  - $\mathrm{Pref}(K) \cdot \Sigma_{uc} \cap \mathrm{Pref}(\mathcal{L}(S)) \subseteq \mathrm{Pref}(K)$
  - $\mathrm{Pref}(K) \cap \mathcal{L}(S) = K$.
- 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)
- ...

Synthesis of reactive programs

Pnueli-Rosner 89



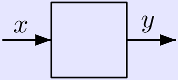
- Q_x: domain for input variable x
- Q_y: domain for output variable y
- Program: $f : Q_x^+ \to Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3) \cdots$ with $y_n = f_1(x_1 \cdots x_n)$ for all $n > 0$.

Implementability problem

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

Synthesis of reactive programs

Pnueli-Rosner 89



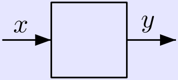
- Q_x: domain for input variable x
- Q_y: domain for output variable y
- Program: $f : Q_x^+ \to Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3) \cdots$ with $y_n = f_1(x_1 \cdots x_n)$ for all $n > 0$.

Implementability problem

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

Synthesis of reactive programs

Pnueli-Rosner 89



- Q_x: domain for input variable x
- Q_y: domain for output variable y
- Program: $f : Q_x^+ \to Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- 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 φ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program f such that all f-behaviors satisfy φ?
- Given a branching time specification φ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program f such that its run-tree satisfies φ?

Synthesis of reactive programs

Implementability problem

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

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi = \mathsf{F}(x = 1)$
- φ is satisfiable: $(1, 0)^\omega \models \varphi$
- φ 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 $\varphi = (y = 1) \longleftrightarrow \mathsf{F}(x = 1)$
- $\forall \vec{x}\ \exists \vec{y}\ \varphi$ is valid.
- φ is not implementable by a reactive program.

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

Synthesis of reactive programs

Implementability problem

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

Implementability $\neq$ Satisfiability

- $Q_x = \{0,1\}$ and $\varphi = \mathsf{F}(x=1)$
- φ is satisfiable: $(1,0)^\omega \models \varphi$
- φ 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 $\varphi = (y=1) \longleftrightarrow \mathsf{F}(x=1)$
- $\forall\vec{x}\ \exists\vec{y}\ \varphi$ is valid.
- φ is not implementable by a reactive program.

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

Synthesis of reactive programs

Implementability problem

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

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi = \mathsf{F}(x = 1)$
- φ is satisfiable: $(1, 0)^\omega \models \varphi$
- φ 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 $\varphi = (y = 1) \longleftrightarrow \mathsf{F}(x = 1)$
- $\forall \vec{x}\ \exists \vec{y}\ \varphi$ is valid.
- φ is not implementable by a reactive program.

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

Synthesis of reactive programs

Implementability problem

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

Theorem (Pnueli-Rosner 89)

- The specification $\varphi \in \mathrm{LTL}$ is implementable iff the formula

$$\mathcal{A}\varphi \wedge \mathsf{AG}(\bigwedge_{a \in Q_x} \mathsf{EX}(x = a))$$

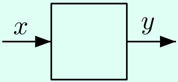
  is satisfiable.

- When φ is implementable, we can construct a finite state implementation (program) in time doubly exponential in φ.

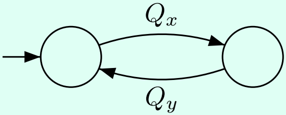
Program synthesis versus System control

Equivalence

The implementability problem for



is equivalent to the control problem for the system



Outline

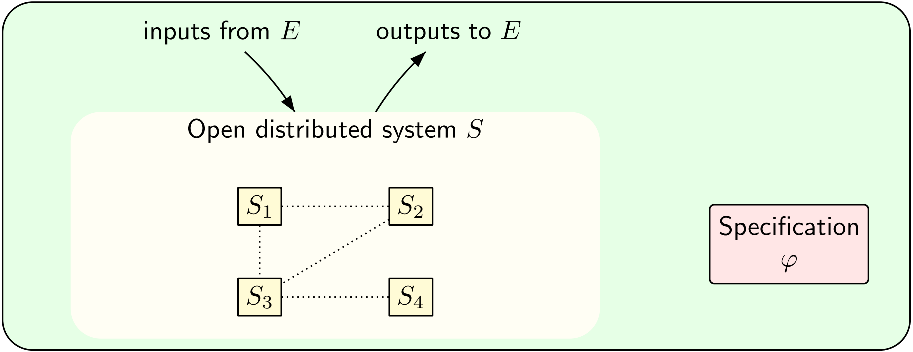
Control for sequential systems

2 Control for distributed systems

Synchronous semantics

Asynchronous semantics

Distributed control



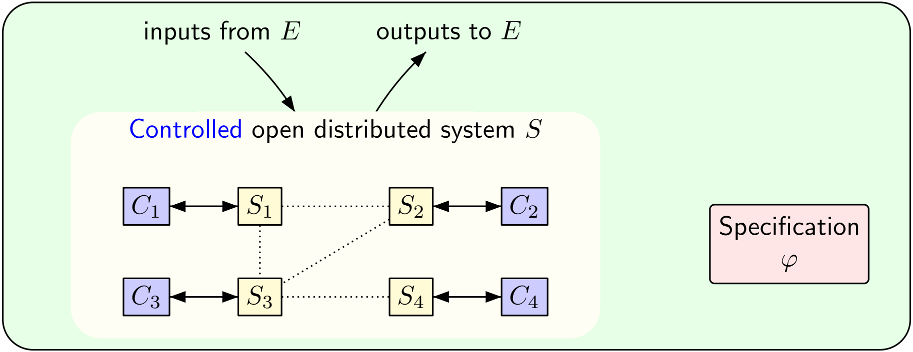
Two problems, again

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

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problems are undecidable.

Distributed control



Two problems, again

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

Peterson-Reif 1979, Pnueli-Rosner 1990

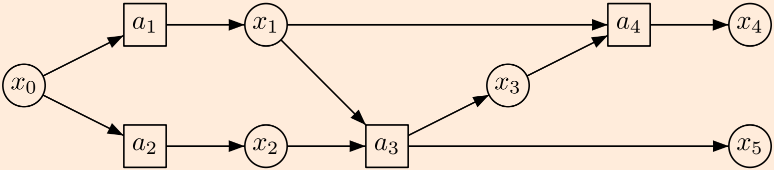
In general, the problems are undecidable.

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



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 our 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 our 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 our 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 our 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 our 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

Outline

Control for sequential systems

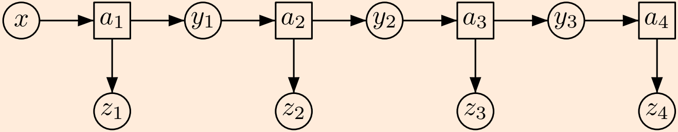
Control for distributed systems

3. Synchronous semantics

Asynchronous semantics

Pnueli-Rosner (FOCS'90)

Pipeline

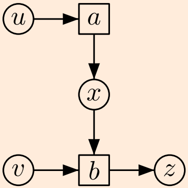


Restrictions

- 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: $f_a : Q^*_{R(a)} \to Q_{W(a)}$ for all $a \in \mathcal{P}$.
- Specification: LTL over input and output variables only.
  - Input variables: $\mathrm{In} = W(\mathrm{environment})$
  - output variables: $\mathrm{Out} = R(\mathrm{environment})$

0-delay synchronous semantics

Example



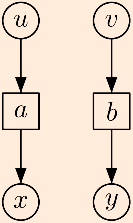
Programs: $f_x : Q_u^* \longrightarrow Q_x$ and $f_z : (Q_x \times Q_v)^* \longrightarrow Q_z$.

- 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 $\begin{cases} x_n = f_x(u_1 \cdots u_n) \\ z_n = f_z((x_1, v_1) \cdots (x_n, v_n)) \end{cases}$ for all $n > 0$.

Undecidability

Architecture $\mathcal{A}_0$



Theorem (Pnueli-Rosner FOCS'90)

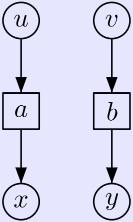
The synthesis problem for architecture $\mathcal{A}_0$ and LTL (or CTL) specifications is undecidable.

Proof

Reduction from the halting problem on the empty tape.

Undecidability proof 1

SPEC$_1$: processes a and b must output configurations



$0^q 1^p 0\cdots \quad : n(v) = p$

$\#^{q+p} C \#^\omega \quad$: where $C \in \Gamma^* Q \Gamma^+$

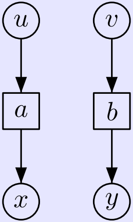
$$(v = 0 \land y = \#)\,\mathsf{W}\,\Big(v = 1 \land (v = 1 \land y = \#)\,\mathsf{W}\,(v = 0 \land y \in \Gamma^* Q \Gamma^+ \#^\omega)\Big)$$

where

$$y \in \Gamma^* Q \Gamma^+ \#^\omega \quad \stackrel{\text{def}}{=} \quad y \in \Gamma\,\mathsf{U}\,\Big(y \in Q \land \mathsf{X}\big(y \in \Gamma\,\mathsf{U}\,(y \in \Gamma \land \mathsf{X}\,\mathsf{G}\,y = \#)\big)\Big)$$

Undecidability proof 1

SPEC$_1$: processes a and b must output configurations



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

$\#^{q+p} C \#^{\omega}$: where $C \in \Gamma^* Q \Gamma^+$

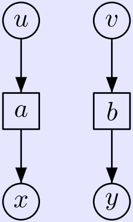
$$(v = 0 \wedge y = \#) \,\mathsf{W}\, \Big(v = 1 \wedge (v = 1 \wedge y = \#) \,\mathsf{W}\, (v = 0 \wedge y \in \Gamma^* Q \Gamma^+ \#^{\omega})\Big)$$

where

$$y \in \Gamma^* Q \Gamma^+ \#^{\omega} \quad \stackrel{\text{def}}{=} \quad y \in \Gamma \,\mathsf{U}\, \Big(y \in Q \wedge \mathsf{X}\big(y \in \Gamma \,\mathsf{U}\, (y \in \Gamma \wedge \mathsf{X}\, \mathsf{G}\, y = \#)\big)\Big)$$

Undecidability proof 2

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



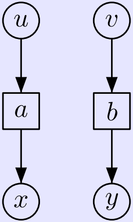
$0^q10\cdots \quad : n(v) = 1$

$\#^{q+1}C_1\#^\omega$

$$v = 0 \,\mathsf{W} \Big(v = 1 \wedge \mathsf{X}\big(v = 0 \longrightarrow y \in C_1\#^\omega\big)\Big)$$

Undecidability proof 2

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



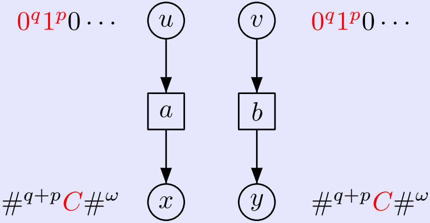
$0^q 1 0 \cdots \quad : n(v) = 1$

$\#^{q+1} C_1 \#^\omega$

$$v = 0 \,\mathsf{W} \Big(v = 1 \wedge \mathsf{X}\big(v = 0 \longrightarrow y \in C_1 \#^\omega\big)\Big)$$

Undecidability proof 3

SPEC$_3$: if $n(u) = n(v)$ are synchronized then $x = y$



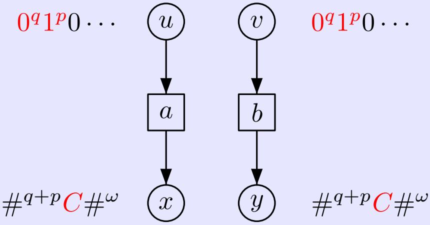
$$n(u) = n(v) \longrightarrow \mathsf{G}(x = y)$$

where

$$n(u) = n(v) \quad \stackrel{\text{def}}{=} \quad (u = v = 0)\ \mathsf{U}\ (u = v = 1 \wedge (u = v = 1\ \mathsf{U}\ u = v = 0))$$

Undecidability proof 3

SPEC$_3$: if $n(u) = n(v)$ are synchronized then $x = y$



$$n(u) = n(v) \longrightarrow \mathsf{G}(x = y)$$

where

$$n(u) = n(v) \quad \stackrel{\text{def}}{=} \quad (u = v = 0)\ \mathsf{U}\ (u = v = 1 \wedge (u = v = 1\ \mathsf{U}\ u = v = 0))$$

Undecidability proof 4

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

$$
\begin{tikzcd}
0^q 1^{p+1} 0 \cdots & u \arrow[d] & v \arrow[d] & 0^{q+1} 1^p 0 \cdots \\
& \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
\#^{q+p+1} C_x \#^\omega & x & y & \#^{q+p+1} C_y \#^\omega
\end{tikzcd}
$$

$$n(u) = n(v) + 1 \quad \longrightarrow \quad x = y \, \mathsf{U} \left(\mathrm{Trans}(y, x) \wedge \mathsf{X}^3 \, \mathsf{G} \, x = y \right)$$

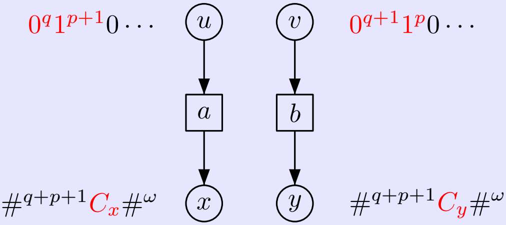
where $\mathrm{Trans}(y, x)$ is defined by

$$\bigvee_{(p,a,q,b,\leftarrow) \in T, c \in \Gamma} (y = cpa \wedge x = qcb) \quad \vee \bigvee_{(p,a,q,b,\rightarrow) \in T, c \in \Gamma} (y = pac \wedge x = bqc)$$

$$\vee \bigvee_{(p,a,q,b,\rightarrow) \in T} (y = pa\# \wedge x = bq\Box)$$

Undecidability proof 4

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



$$n(u) = n(v) + 1 \quad \longrightarrow \quad x = y \,\mathsf{U}\, \Big(\mathrm{Trans}(y, x) \wedge \mathsf{X}^3\, \mathsf{G}\, x = y\Big)$$

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

$$\bigvee_{(p,a,q,b,\leftarrow)\in T, c\in\Gamma} (y = cpa \wedge x = qcb) \quad \vee \quad \bigvee_{(p,a,q,b,\rightarrow)\in T, c\in\Gamma} (y = pac \wedge x = bqc)$$

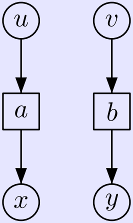
$$\vee \quad \bigvee_{(p,a,q,b,\rightarrow)\in T} (y = pa\# \wedge x = bq\square)$$

Undecidability proof 5

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

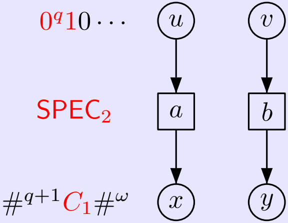
Specifications 1-4 and 5: G $x \neq \mathrm{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.

Undecidability proof 5

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

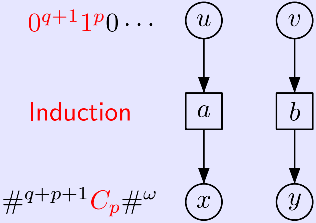
Specifications 1-4 and 5: G $x \neq$ stop are implementable iff the Turing machine does not halt starting from the empty tape.

Undecidability proof 5

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

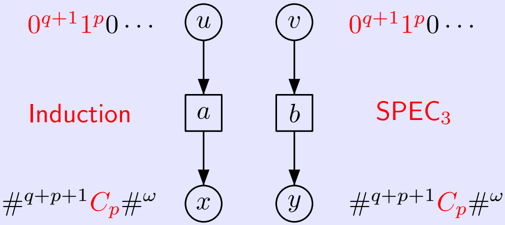
Specifications 1-4 and 5: $\mathsf{G}\, x \neq \mathrm{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.

Undecidability proof 5

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

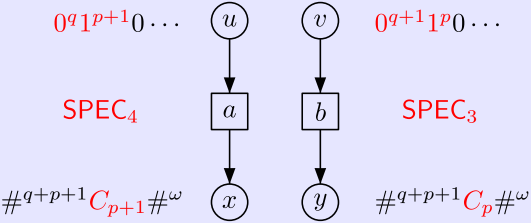
Specifications 1-4 and 5: G $x \neq \mathrm{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.

Undecidability proof 5

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

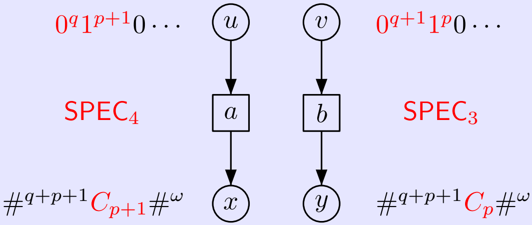
Specifications 1-4 and 5: G $x \neq \mathrm{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.

Undecidability proof 5

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

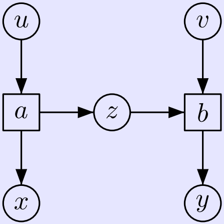


Corollary

Specifications 1-4 and 5: $\mathsf{G}\, x \neq \mathrm{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.

Communication allows to cheat

Architecture with communication



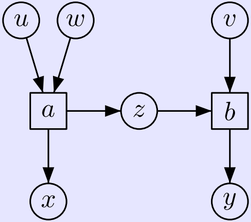
- Strategy for a:
  - copy u to z
  - if $u = 0^q 1^p 0 \cdots$ then $x = \begin{cases} \#^{p+q} C_1 \#^\omega & \text{if } p = 1 \text{ (for SPEC}_2\text{)} \\ \#^{p+q} C_2 \#^\omega & \text{othewise (for SPEC}_4\text{).} \end{cases}$
- Strategy for b: if $z = 0^{q'} 1^{p'} 0 \cdots$ and $v = 0^q 1^p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

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

More undecidable architectures

Exercices

1. Show that the architecture below is undecidable.



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

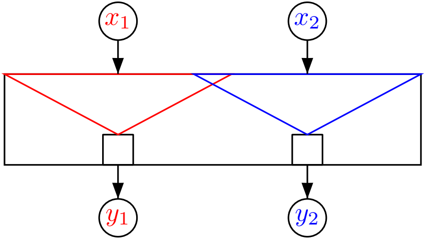
Uncomparable information

Definition

For an output variable y, 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.



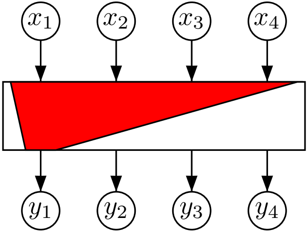
Uncomparable information

Definition

For an output variable y, 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 View$(y_2) \setminus$ View$(y_1) \neq \emptyset$ and View$(y_1) \setminus$ View$(y_2) \neq \emptyset$.
Otherwise it is said to have preordered information.



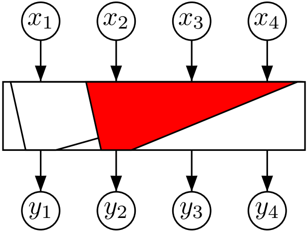
Uncomparable information

Definition

For an output variable y, $\mathsf{View}(y)$ 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 $\mathsf{View}(y_2) \setminus \mathsf{View}(y_1) \neq \emptyset$ and $\mathsf{View}(y_1) \setminus \mathsf{View}(y_2) \neq \emptyset$.
Otherwise it is said to have preordered information.



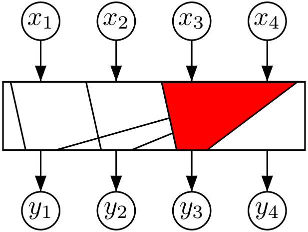
Uncomparable information

Definition

For an output variable y, 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 View(y_2) $\setminus$ View(y_1) $\neq \emptyset$ and View(y_1) $\setminus$ View(y_2) $\neq \emptyset$.
Otherwise it is said to have preordered information.



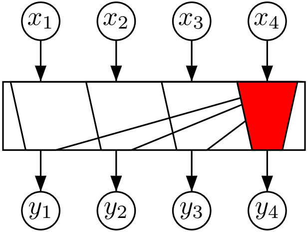
Uncomparable information

Definition

For an output variable y, 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 View$(y_2) \setminus$ View$(y_1) \neq \emptyset$ and View$(y_1) \setminus$ View$(y_2) \neq \emptyset$.
Otherwise it is said to have preordered information.

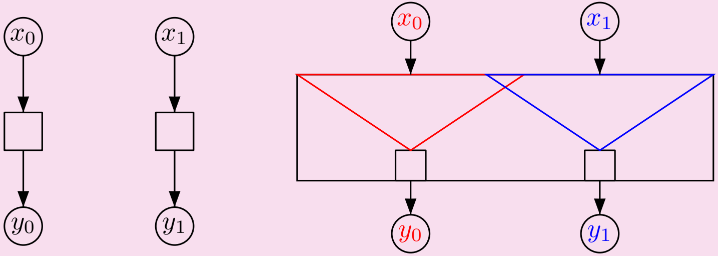


Uncomparable information yields undecidability

Theorem

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

Proof for LTL specifications

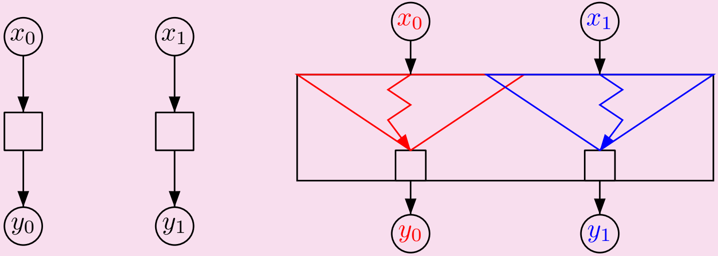


Uncomparable information yields undecidability

Theorem

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

Proof for LTL specifications

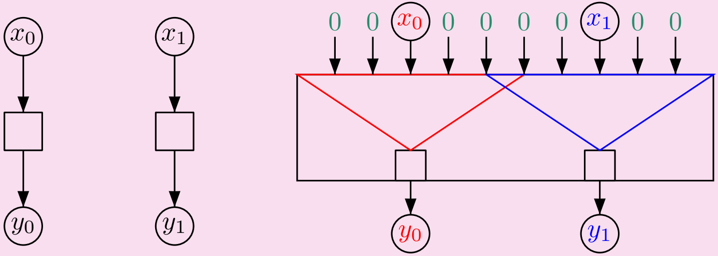


Uncomparable information yields undecidability

Theorem

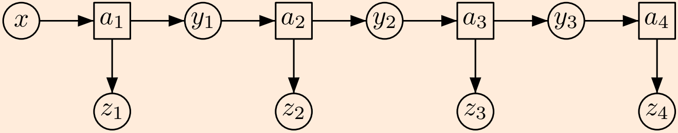
Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

Proof for LTL specifications



Decidability

Pipeline

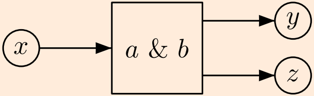


Pnueli-Rosner (FOCS'90)

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

Decidability proof 1

Pipeline



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

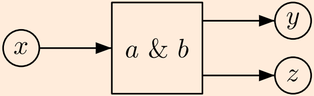
$$h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \qquad \text{where} \qquad y_i = f_y(x_1, \cdots, x_i).$$

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 $f_y : Q_x^+ \to Q_y$ 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

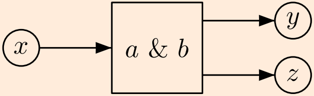
$$h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \qquad \text{where} \qquad y_i = f_y(x_1, \cdots, x_i).$$

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 $f_y : Q_x^+ \to Q_y$ 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

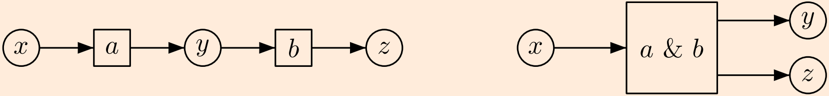
$$h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \qquad \text{where} \qquad y_i = f_y(x_1, \cdots, x_i).$$

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 2

Pipeline



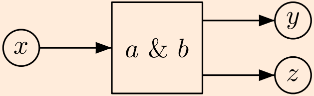
Proof

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 $\mathcal{A}$

Decidability proof 2

Pipeline

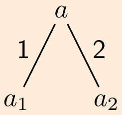


Proof

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 $\mathcal{A}$

Tree automata

non deterministic transitions

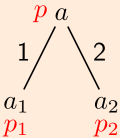


Alternating transitions

or

Tree automata

non deterministic transitions

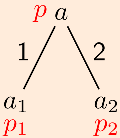


Alternating transitions

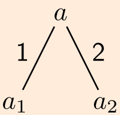
or

Tree automata

non deterministic transitions



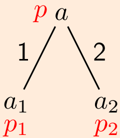
Alternating transitions



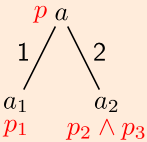
or

Tree automata

non deterministic transitions



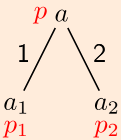
Alternating transitions



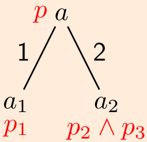
or

Tree automata

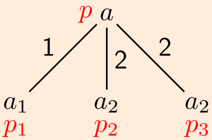
non deterministic transitions



Alternating transitions

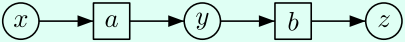


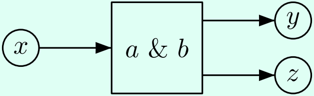
or



Decidability proof 3

Proof

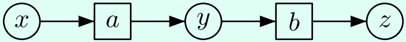


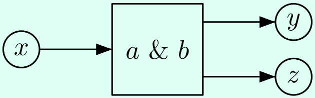


1. We first solve the global game: We obtain an ND tree-automaton $\mathcal{A}$ accepting the global strategies $h : Q_x^+ \to Q_y \times Q_z$ that implement the specification.
   Easily obtained from a ND tree automaton for the specification.
2. We build from $\mathcal{A}$ an alternating tree automaton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

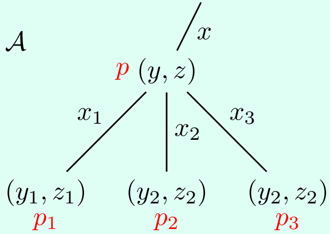
Decidability proof 3

Proof



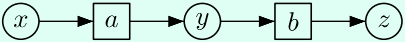


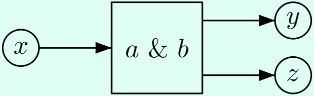
1. We first solve the global game: We obtain an ND tree-automaton $\mathcal{A}$ accepting the global strategies $h : Q_x^+ \to Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.
2. We build from $\mathcal{A}$ an alternating tree automaton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



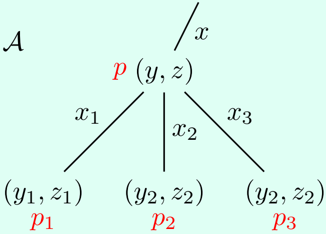
Decidability proof 3

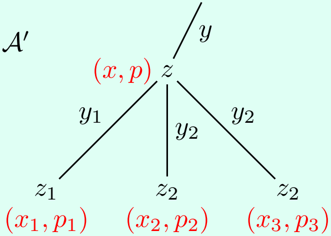
Proof





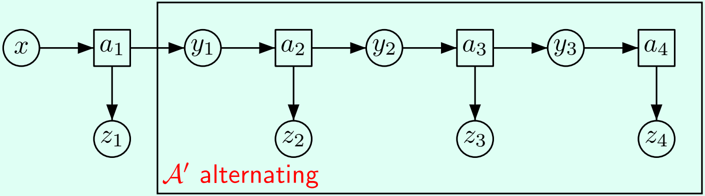
1. We first solve the global game: We obtain an ND tree-automaton $\mathcal{A}$ accepting the global strategies $h : Q_x^+ \to Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.
2. We build from $\mathcal{A}$ an alternating tree automaton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





Decidability proof 4

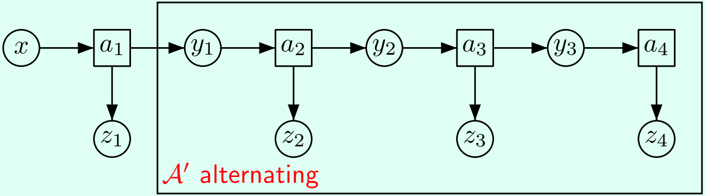
Proof



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.

Decidability proof 4

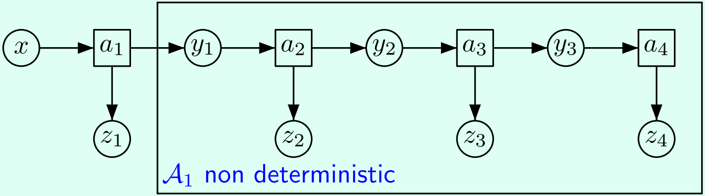
Proof



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.

Decidability proof 4

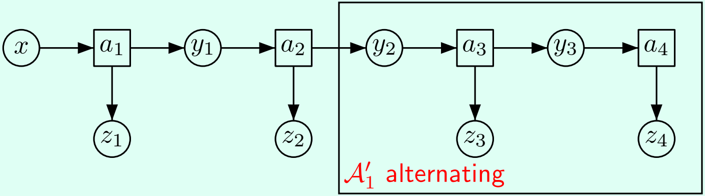
Proof



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.

Decidability proof 4

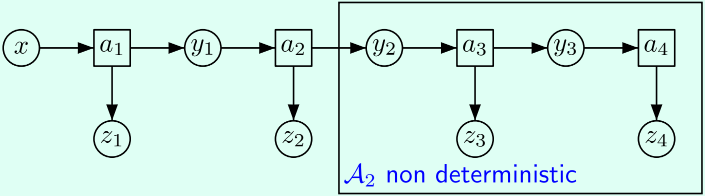
Proof



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.

Decidability proof 4

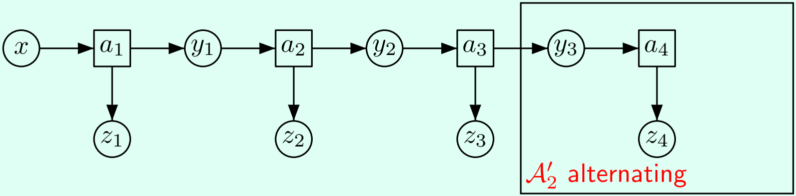
Proof



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.

Decidability proof 4

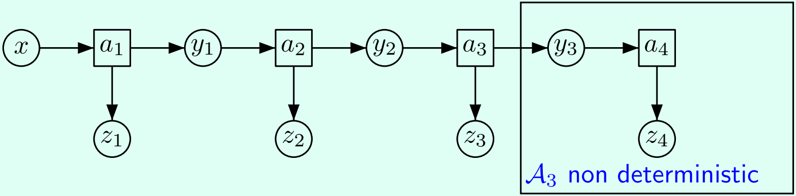
Proof



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.

Decidability proof 4

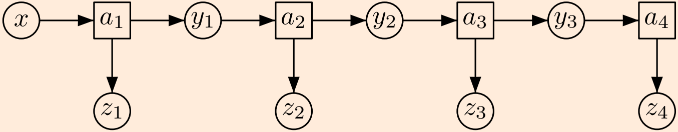
Proof



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.

Decidability

Pipeline



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.

$\Longrightarrow$ non-elementary lower bound for the synthesis problem.

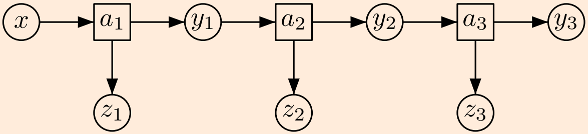
Decidability

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



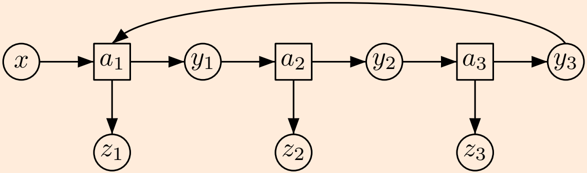
Decidability

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



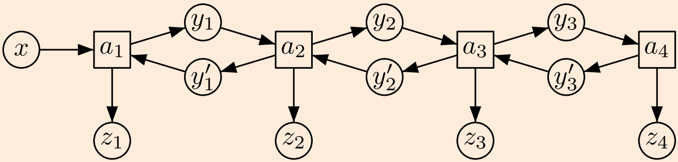
Decidability

Kupferman-Vardi (LICS'01)

The synthesis problem is non elementary decidable for

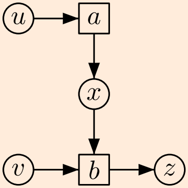
- one-way chain, one-way ring, two-way chain and two-way ring,
- CTL* specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.

two-way chain



1-delay synchronous semantics

Example



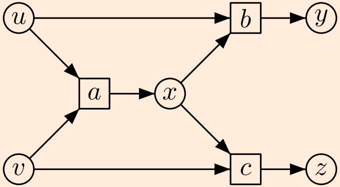
Programs: $f_x : Q_u^* \longrightarrow Q_x$ and $f_z : (Q_x \times Q_v)^* \longrightarrow Q_z$.

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

Decidability

Adequately connected sub-architecture $Q_x = Q$ for all $x \in \mathcal{V}$

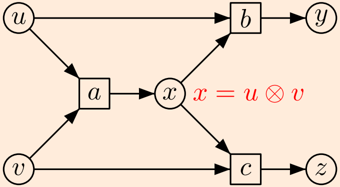


Pnueli-Rosner (FOCS'90)

- 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

Adequately connected sub-architecture $Q_x = Q$ for all $x \in \mathcal{V}$



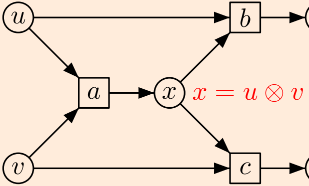
Pnueli-Rosner (FOCS'90)

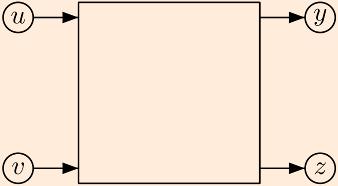
- An adequately connected architecture is equivalent to a singleton architecture.
- The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.

Decidability

Adequately connected sub-architecture

$Q_x = Q$ for all $x \in \mathcal{V}$



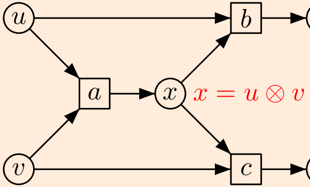


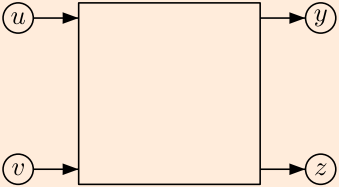
Pnueli-Rosner (FOCS'90)

- 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

Adequately connected sub-architecture $\quad Q_x = Q$ for all $x \in \mathcal{V}$

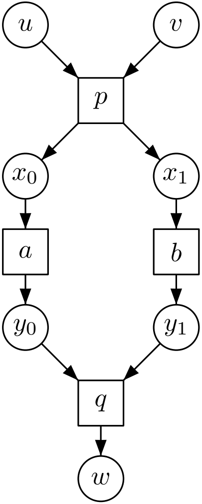




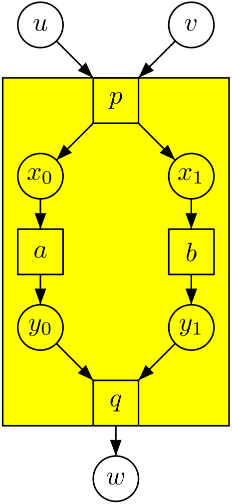
Pnueli-Rosner (FOCS'90)

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

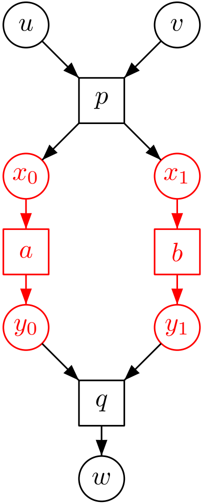
Information fork criterion (Finkbeiner–Schewe LICS '05)



Information fork criterion (Finkbeiner–Schewe LICS '05)



Information fork criterion (Finkbeiner–Schewe LICS '05)

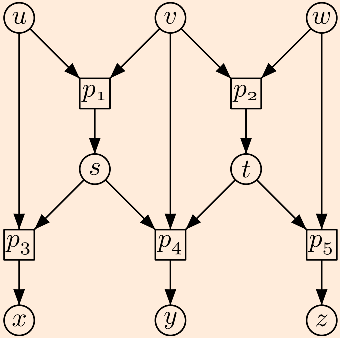


Uniformly well connected architectures

Definition

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

Example

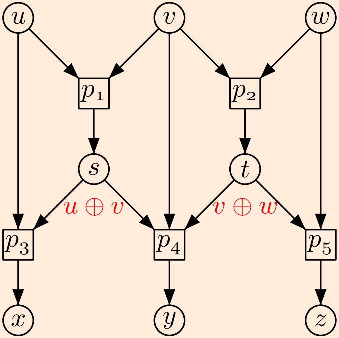


Uniformly well connected architectures

Definition

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

Example



Uniformly well connected architectures

Definition

An architecture is uniformly well connected if there is a uniform way to route variables in 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 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

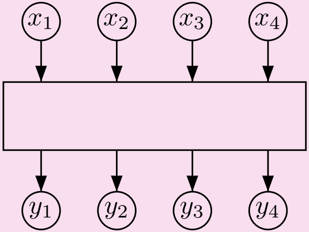
Reduction to the multicast problem in Network Information Flow.
The multicast problem is NP-complete (Rasala Lehman-Lehman 2004).

Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

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

Proof.



Theorem: Kupferman-Vardi (LICS'01)

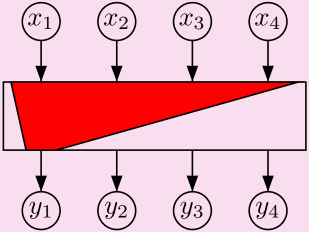
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

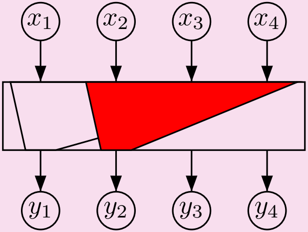
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

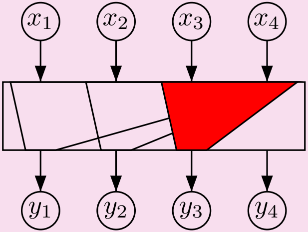
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

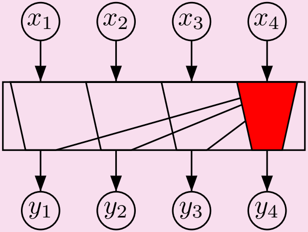
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

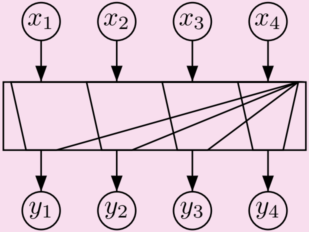
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

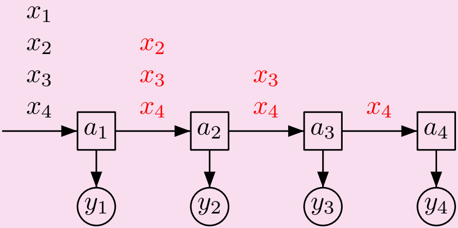
Uniformly well connected architectures

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)

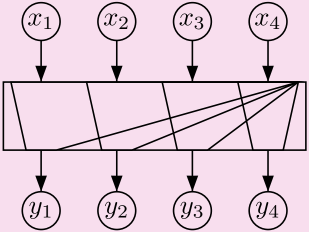
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

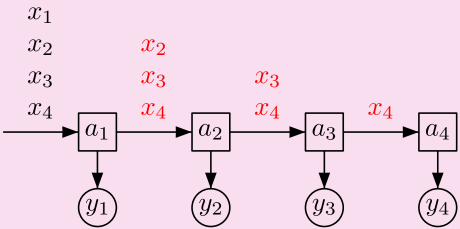
Uniformly well connected architectures

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.

Robust specifications

Definition

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

Theorem

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

Proof.

Robust specifications

Definition

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

Theorem

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

Proof.

Robust specifications

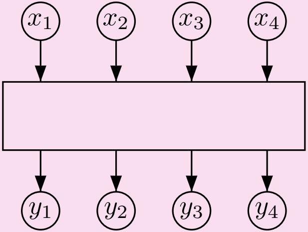
Definition

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

Theorem

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

Proof.



Robust specifications

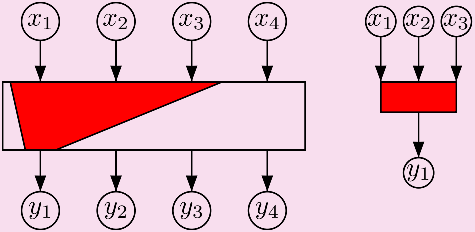
Definition

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

Theorem

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

Proof.



Robust specifications

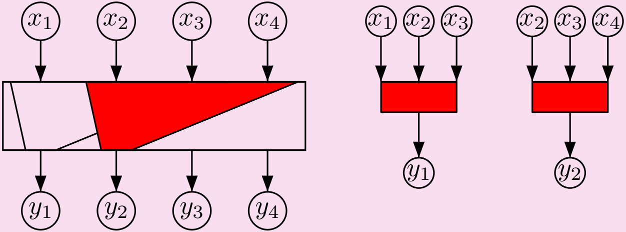
Definition

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

Theorem

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

Proof.



Robust specifications

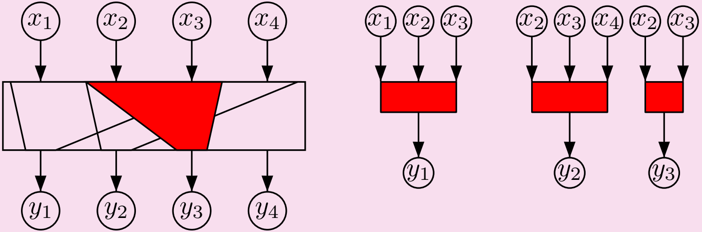
Definition

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

Theorem

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

Proof.



Robust specifications

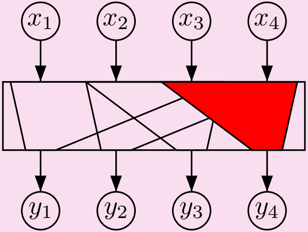
Definition

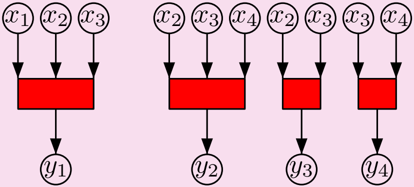
A specification φ is robust if it can be written $\varphi = \bigvee \bigwedge_{z\in\mathrm{Out}} \varphi_z$ where φ_z depends only on $\mathsf{View}(z) \cup \{z\}$.

Theorem

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

Proof.



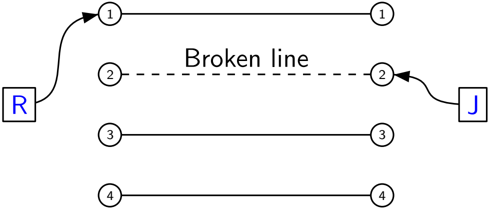


Open problem

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

Outline

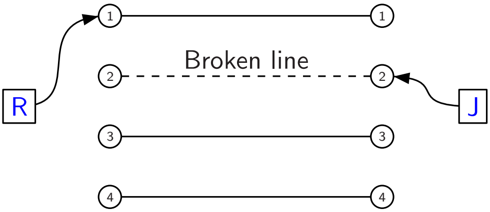
An example: Romeo and Juliet



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.

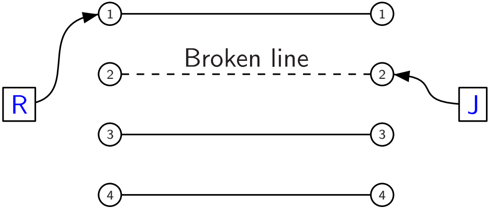
An example: Romeo and Juliet



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.

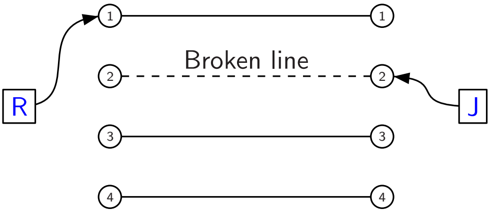
An example: Romeo and Juliet



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

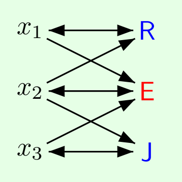
- Want to communicate through the same communication line.
- At any time, one line is broken.
- Environment looks where R&J are connected, and then, atomically, changes (possibly) the broken line.
- Romeo/Juliet looks status of lines and, atomically, chooses where to connect.

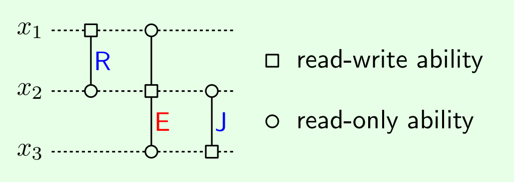
Romeo and Juliet (continued)

Architecture

- Variables:
  - x_1: Romeo's current line. $Q_1 = \{1, 2, 3, 4\}$
  - x_2: broken line $Q_2 = \{1, 2, 3, 4\}$
  - x_3: Juliet's current line. $Q_3 = \{1, 2, 3, 4\}$
- Agents: Romeo, Juliet and Environment.
- Read/Write table

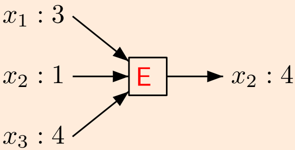
	Romeo	Juliet	Environment
Read	$\{x_1, x_2\}$	$\{x_2, x_3\}$	$\{x_1, x_2, x_3\}$
Write	$\{x_1\}$	$\{x_3\}$	$\{x_2\}$

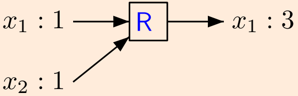




Romeo and Juliet (continued)

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

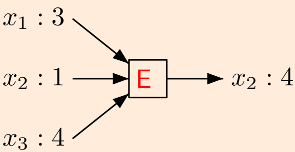


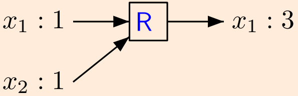


A distributed play of the asynchronous system, R & J against E

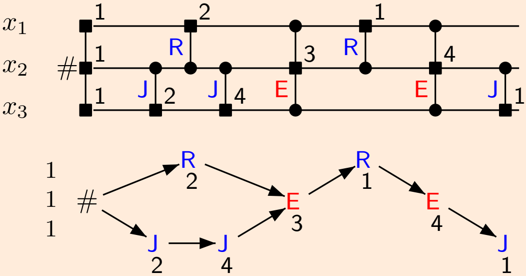
Romeo and Juliet (continued)

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





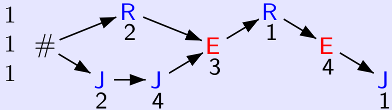
A distributed play of the asynchronous system, R & J against E



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 $\mathcal{W} \subseteq \mathbb{R}(\Sigma, D)$.
  Team 0 wins plays of $\mathcal{W}$ and loses plays of $\mathbb{R}(\Sigma, D) \setminus \mathcal{W}$.

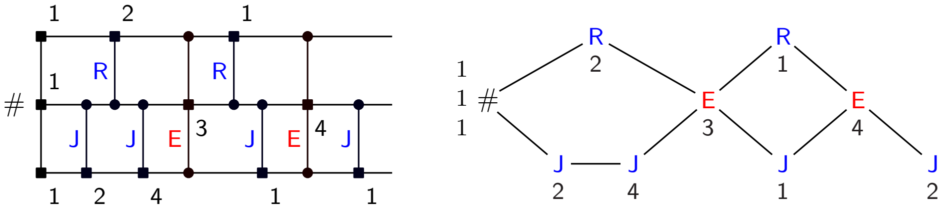
Romeo and Juliet

$\mathcal{W}$ 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.



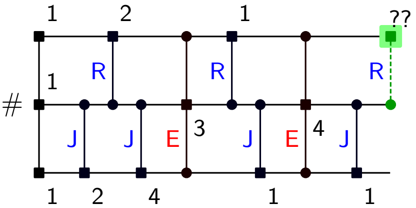
Causal memory (intuitively, maximal history a player can observe)

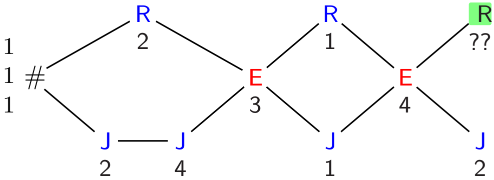
- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.

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.





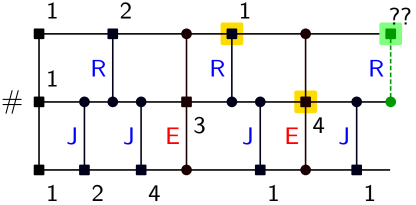
Causal memory (intuitively, maximal history a player can observe)

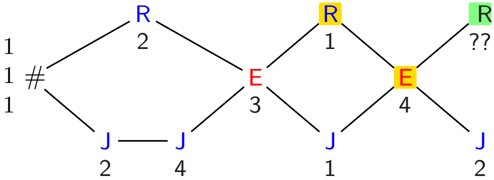
- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.

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.





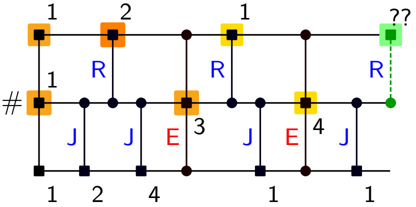
Causal memory (intuitively, maximal history a player can observe)

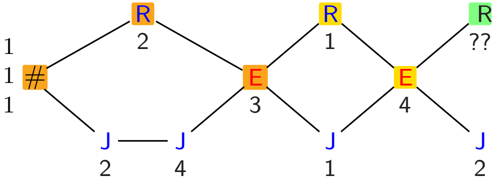
- Players gather and forward as much information as possible.
- but no global view, the choice for an action cannot depend on a concurrent event.

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.





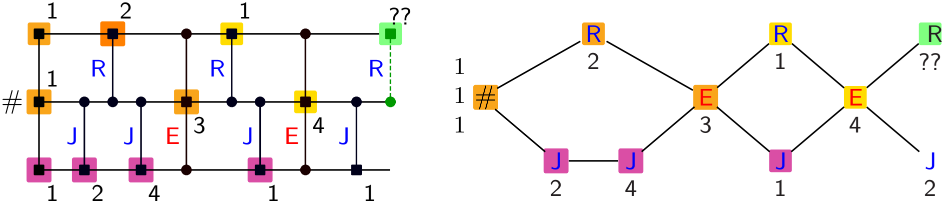
Causal memory (intuitively, maximal history a player can observe)

- 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.



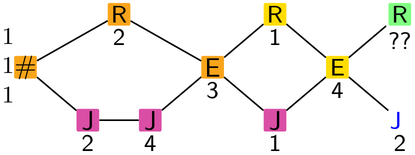
Causal memory (intuitively, maximal history a player can observe)

- 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	$f_a : Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$
Local memory	$f_a : (Q_{R(a)})^* Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$
Causal memory	$f_a : \mathbb{M}(\Sigma, D) \times Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$



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 Stop for all a-moves enabled at t with $a \in \mathcal{P}_0$.

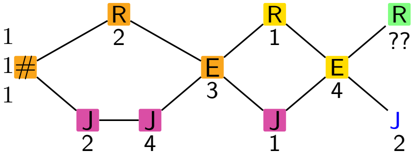
Winning strategies

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

Winning strategies

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

Memoryless	$f_a : Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$
Local memory	$f_a : (Q_{R(a)})^* Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$
Causal memory	$f_a : \mathbb{M}(\Sigma, D) \times Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$



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 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 $\mu : \mathbb{M}(\Sigma, D) \to M$ satisfying the following equivalent properties:

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

Strategy with memory μ

$$f_a : M \times Q_{R(a)} \to Q_{W(a)} \cup \text{Stop}$$

the associated strategy is defined by

$$f_a^{\mu}(t, q) = f_a(\mu(t), q)$$

If M is finite then f^{μ} is a distributed strategy with finite memory.
If $|M| = 1$ then f^{μ} is memoryless.

Embedding causal memory inside games

Proposition: PG-Lerman-Zeitoun (LATIN'04)

For a distributed game G and a distributed memory μ, one can build a game G^μ such that

team 0 has a WDS in G with memory μ

iff

team 0 has a memoryless WDS in G^μ.

Proof.

$$G^\mu = G \times \mu$$

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 $(\widetilde{G}, \widetilde{\mathcal{W}})$ 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 $(\widetilde{G}, \widetilde{\mathcal{W}})$.
- There exists a WS for player 0 in $(\widetilde{G}, \widetilde{\mathcal{W}})$.

Moreover, if $\mathcal{W}$ is recognizable then so is $\widetilde{\mathcal{W}}$

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 $\widetilde{\mathcal{W}}$ 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 $(\widetilde{G}, \widetilde{\mathcal{W}})$ 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 $(\widetilde{G}, \widetilde{\mathcal{W}})$.
- There exists a WS for player 0 in $(\widetilde{G}, \widetilde{\mathcal{W}})$.

Moreover, if $\mathcal{W}$ is recognizable then so is $\widetilde{\mathcal{W}}$

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 $\widetilde{\mathcal{W}}$ 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 $(\widetilde{G}, \widetilde{\mathcal{W}})$ 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 $(\widetilde{G}, \widetilde{\mathcal{W}})$.
- There exists a WS for player 0 in $(\widetilde{G}, \widetilde{\mathcal{W}})$.

Moreover, if $\mathcal{W}$ is recognizable then so is $\widetilde{\mathcal{W}}$

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 $\widetilde{\mathcal{W}}$ makes sure that the strategy followed by the controller is indeed distributed.

(Un)deciding games

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.

Peterson-Reif Madhusudan–Thiagarajan Bernet–Janin–Walukiewicz

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.

(Un)deciding games

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.

Peterson-Reif Madhusudan–Thiagarajan Bernet–Janin–Walukiewicz

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

- $\mathcal{A}$ is a parallel product if
  $\mathcal{P} = A \uplus B$ with $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in A \times B$.
- $\mathcal{A}$ is a serial product if
  $\mathcal{P} = A \uplus B$ with $R(a) \cap W(b) \neq \emptyset$ for all $(a, b) \in A \times B$.
- $\mathcal{A}$ is series-parallel if it can be obtained from singletons ($|\mathcal{P}| = 1$) using serial and parallel compositions.
- $\mathcal{A}$ is series-parallel iff the associated dependence relation does not contain a P_4: $a — b — c — d$ 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 $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$ be some architecture.

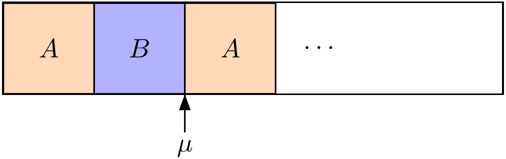
- $\mathcal{A}$ is a parallel product if
  $\mathcal{P} = A \uplus B$ with $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in A \times B$.
- $\mathcal{A}$ is a serial product if
  $\mathcal{P} = A \uplus B$ with $R(a) \cap W(b) \neq \emptyset$ for all $(a, b) \in A \times B$.
- $\mathcal{A}$ is series-parallel if it can be obtained from singletons ($|\mathcal{P}| = 1$) using serial and parallel compositions.
- $\mathcal{A}$ is series-parallel iff the associated dependence relation does not contain a P_4: $a — b — c — d$ 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 Σ.

Difficult case: serial product.



1. A WS on $A \uplus B$ 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 $A \cup B$ 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.

Madhusudan and Thiagarajan (Concur'02)

Setting

- Architecture: $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$ with $R(a) = W(a)$ for all $a \in \mathcal{P}$.
- Moves: δ_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:

$$f_x : Q_x^* \to 2^{R^{-1}(x)}$$

action a is enabled by $(f_x)_{x \in \mathcal{V}}$ at some finite play t if

$$\forall x \in R(a), \qquad a \in f_x(\pi_{Q_x}(t))$$

- The environment decides which a-transition should be taken among the actions a enabled by the strategies.

Madhusudan and Thiagarajan (Concur'02)

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.

Madhusudan and Thiagarajan (Concur'02)

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.

Mohalik and Walukiewicz (FSTTCS'03)

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

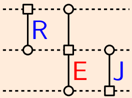
- Generalization to arbitrary symmetric architectures.
- Generalization to non-symmetric architectures.
- Reasonable upper bounds for synthesis?

Symmetric architecture

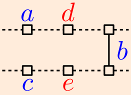
Architecture $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$

- Restrictions: $\begin{cases} \forall a \in \mathcal{P} & \emptyset \neq W(a) \subseteq R(a) \\ \forall a, b \in \mathcal{P} & R(a) \cap W(b) \neq \emptyset \iff R(b) \cap W(a) \neq \emptyset \end{cases}$
- Dependence: $a \; D \; b \iff R(a) \cap W(b) \neq \emptyset \iff R(b) \cap W(a) \neq \emptyset$

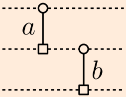
Legal and forbidden architectures



OK



OK



Forbidden (not symmetric)