

On the synthesis of distributed controllers

Paul Gastin

LSV

ENS de Cachan & CNRS

`Paul.Gastin@lsv.ens-cachan.fr`

Perspectives in Verification, November 18th, 2005

Outline

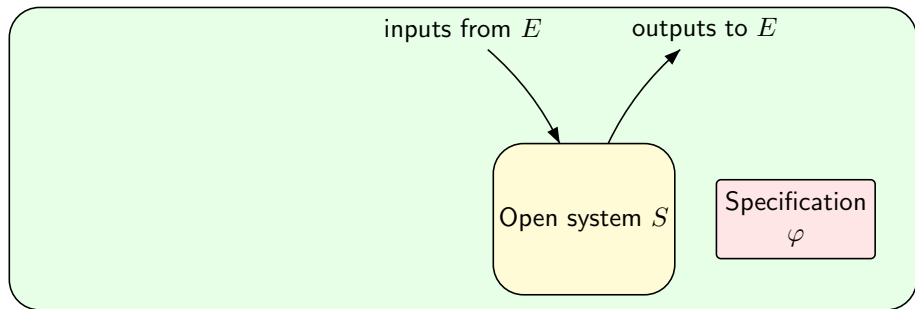
Motivations and context

A general framework

Asynchronous semantics

Synchronous semantics

Control synthesis

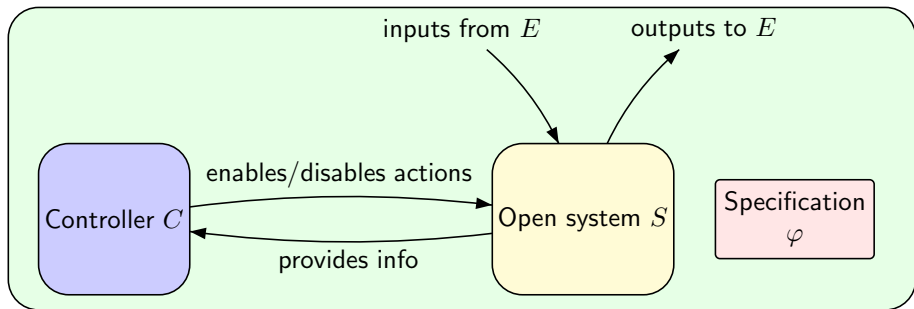


Two problems

- ▶ Decide whether there exists a controller st. $(S \otimes C) \parallel E \models \varphi, \quad \forall E.$
- ▶ Synthesis: If so, compute such a controller.

For reasonable systems and specifications, the problems are decidable.

Control synthesis

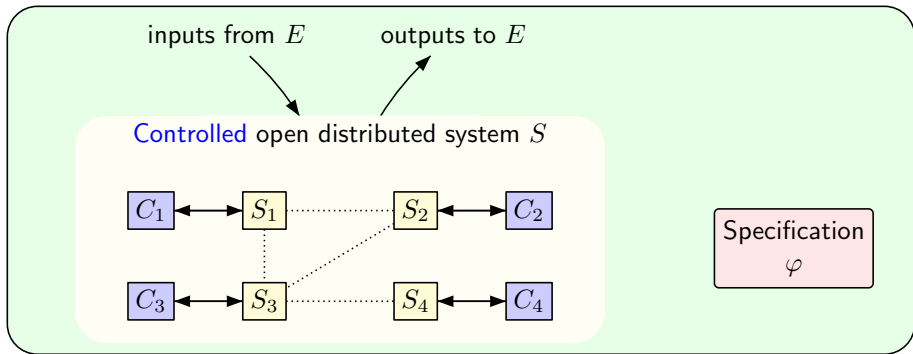


Two problems

- ▶ Decide whether there exists a controller st. $(S \otimes C) \parallel E \models \varphi$, $\forall E$.
- ▶ Synthesis: If so, compute such a controller.

For reasonable systems and specifications, the problems are decidable.

Distributed control synthesis



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**.

Outline

Motivations and context

A general framework

Asynchronous semantics

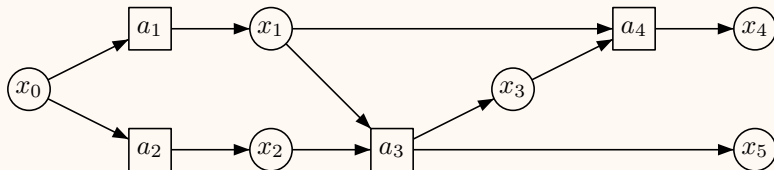
Synchronous semantics

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 $I \subseteq \mathcal{V}$.

Outline

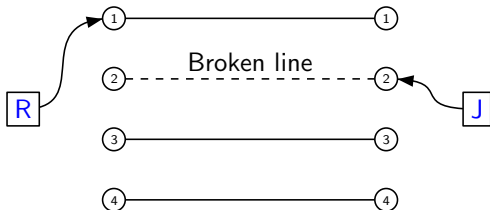
Motivations and context

A general framework

Asynchronous semantics

Synchronous semantics

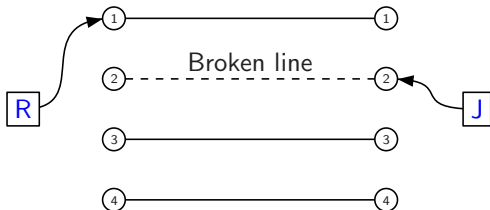
An example: Romeo and Juliet



Romeo and Juliet against the environment

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

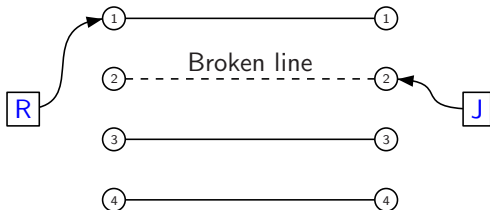
An example: Romeo and Juliet



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.

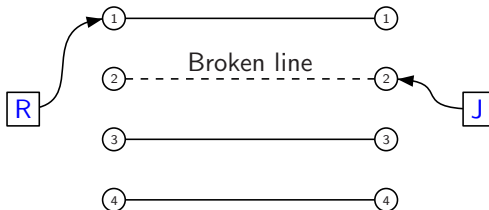
An example: Romeo and Juliet



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.
- ▶ x_2 : broken line
- ▶ x_3 : Juliet's current line.

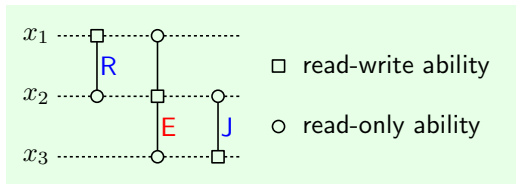
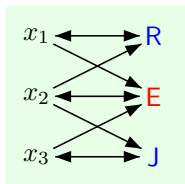
$$Q_1 = \{1, 2, 3, 4\}$$

$$Q_2 = \{1, 2, 3, 4\}$$

$$Q_3 = \{1, 2, 3, 4\}$$

- ▶ Agents: **R**omeo, **J**uliet and **E**nvironment.
- ▶ Read/Write table

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



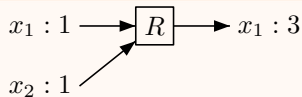
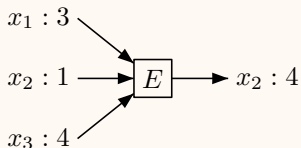
Distributed Game/Control

Game over $(\mathcal{P}, \mathcal{V}, R, W)$

- ▶ $G = (\mathcal{P}_0, \mathcal{P}_1, (Q_x)_{x \in \mathcal{V}}, (\delta_a)_{a \in \mathcal{P}}, q^0, \mathcal{W})$.
- ▶ $\mathcal{P} = \mathcal{P}_0 \uplus \mathcal{P}_1$: partition of the set of players/processes in teams **0** and **1**.
Players of team **0** cooperate together against team **1**.
- ▶ **Rules** of the game for player a given by $\delta_a \subseteq Q_{R(a)} \times Q_{W(a)}$.

Romeo and Juliet (continued)

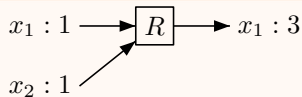
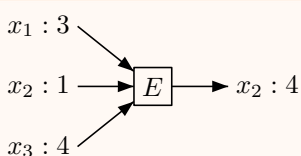
Legal moves: $\delta_a \subseteq Q_{R(a)} \times Q_{W(a)}$



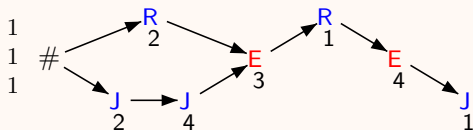
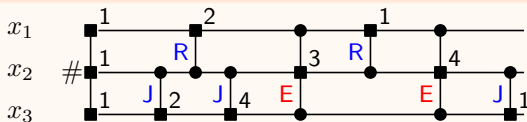
A distributed play of the asynchronous system, R&J against E

Romeo and Juliet (continued)

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

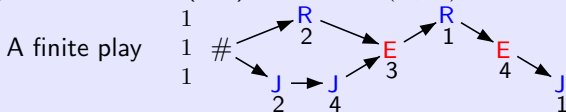


A distributed play of the asynchronous system, R&J against E



Distributed Behaviors

- ▶ $\Sigma = \{(a, q) \mid a \in \mathcal{P}, q \in Q_{W(a)}\} \cup \{(\#, q^0)\}$,
- ▶ $(a, p) D (b, q) \Leftrightarrow R(a) \cap W(b) \neq \emptyset \Leftrightarrow R(b) \cap W(a) \neq \emptyset$.
- ▶ Play: Mazurkiewicz (real) traces over (Σ, D) .



- ▶ 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 $\mathbb{R}(\Sigma, D) \setminus \mathcal{W}$.

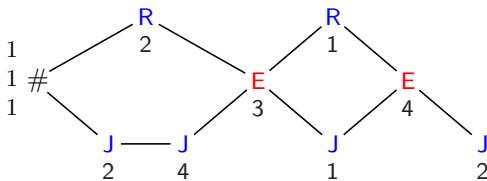
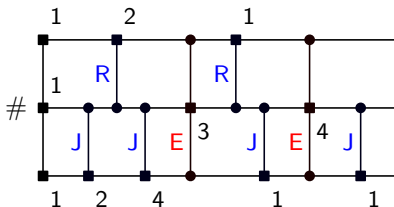
Romeo and Juliet

\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.



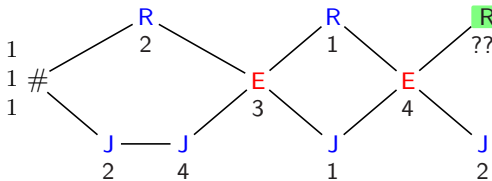
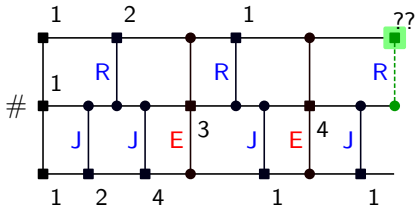
Causal memory (intuitively, maximal history a player can observe)

- ▶ 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.



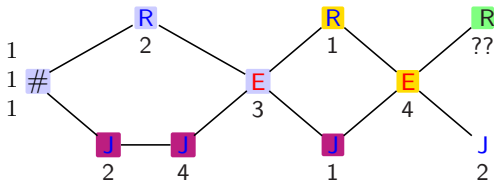
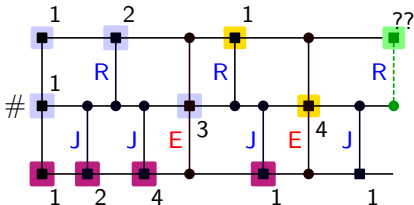
Causal memory (intuitively, maximal history a player can observe)

- ▶ 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.

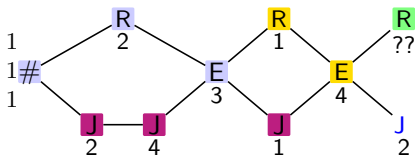
Winning strategies

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

Memoryless $f_a : Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop}$

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

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



f -maximal f -plays

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

- ▶ **consistent** with f : all a -moves played according to f_a (f -play).
- ▶ **maximal**: f predicts to **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 $\mu : \mathbb{M}(\Sigma, D) \rightarrow M$ 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

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

the associated strategy is defined by

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

If M is finite then f_a^μ is a distributed strategy with finite memory.

If $|M| = 1$ then f_a^μ is memoryless.

(Un)deciding games

Proposition: (Folklore)

Deciding whether team 0 has a **causal** distributed (memoryless) WS 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 winning **local** distributed strategy 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 **causal** distributed (memoryless) WS 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 winning **local** distributed strategy is **undecidable** even:

- ▶ for **reachability** or **safety** winning conditions.
- ▶ with 3 players against the environment.

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:

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

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

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

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

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.

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.

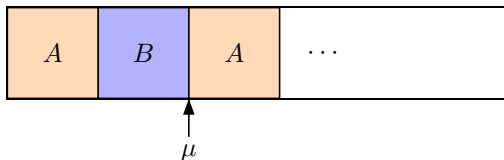
- ▶ \mathcal{A} is a **parallel product** if
 $\mathcal{P} = A \uplus B$ with $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in A \times B$.
- ▶ \mathcal{A} is a **serial product** if
 $\mathcal{P} = A \uplus B$ with $R(a) \cap W(b) \neq \emptyset$ for all $(a, b) \in A \times B$.
- ▶ \mathcal{A} is **series-parallel** 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.

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 $(\tilde{G}, \tilde{\mathcal{W}})$ st. the following are equivalent:

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

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

Naive idea Consider the game on the global transition system.

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

Solution

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

From distributed to sequential games

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

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

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

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

Naive idea Consider the game on the global transition system.

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

Solution

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

From distributed to sequential games

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

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

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

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

Naive idea Consider the game on the global transition system.

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

Solution

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

Open problems

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

Outline

Motivations and context

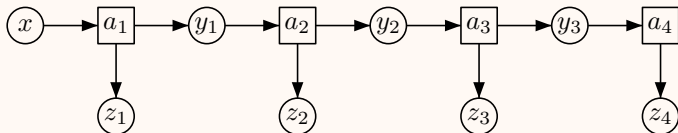
A general framework

Asynchronous semantics

Synchronous semantics

Pnueli-Rosner (FOCS'90)

Pipeline

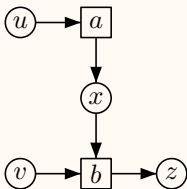


Restrictions

- ▶ 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: $q^0 q^1 q^2 \dots$ where $q^n \in Q_{\mathcal{V}}$ are global states.
- ▶ program with **local memory**: $f_a : Q_{R(a)}^* \rightarrow Q_{W(a)}$ for all $a \in \mathcal{P}$.
- ▶ Specification: LTL over input and output variables only.
 - ▶ Input variables: $\text{In} = W(\text{environment})$
 - ▶ output variables: $\text{Out} = R(\text{environment})$

0-delay synchronous semantics

Example



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

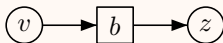
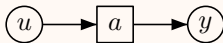
▶ Input: $\begin{pmatrix} u_1 & u_2 & u_3 & \cdots \\ v_1 & v_2 & v_3 & \cdots \end{pmatrix} \in (Q_u \times Q_v)^\omega$.

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

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

Undecidability

Architecture \mathcal{A}_0

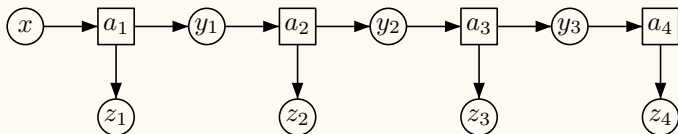


Pnueli-Rosner (FOCS'90)

The synthesis problem for architecture \mathcal{A}_0 and LTL specifications is undecidable.

Decidability

Pipeline



Pnueli-Rosner (FOCS'90)

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

Peterson-Reif (FOCS'79)

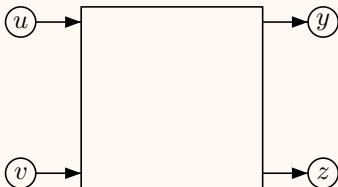
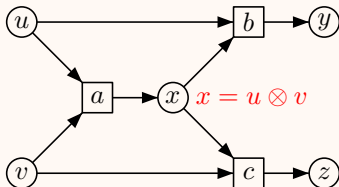
multi-person games with incomplete information.

\implies non-elementary lower bound for the synthesis problem.

Decidability

Adequately connected sub-architecture

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



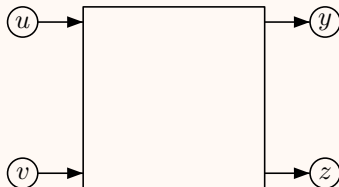
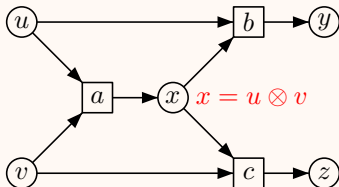
Pnueli-Rosner (FOCS'90)

- ▶ 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.

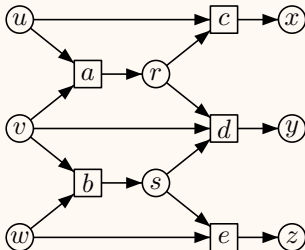
Causally well-connected

Definitions

- ▶ **Visibility**(z): set of input variables u such that there is a path from u to z .
- ▶ The architecture is **causally well-connected** if there is a (uniform) way to route variables in **Visibility**(z) to z for each output variable z .

Example

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



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

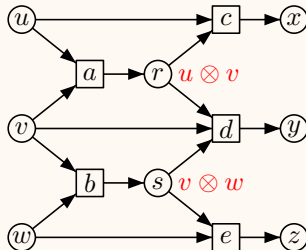
Causally well-connected

Definitions

- ▶ **Visibility**(z): set of input variables u such that there is a path from u to z .
- ▶ The architecture is **causally well-connected** if there is a (uniform) way to route variables in **Visibility**(z) to z for each output variable z .

Example

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



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

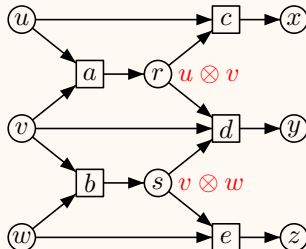
Causally well-connected

Definitions

- ▶ **Visibility**(z): set of input variables u such that there is a path from u to z .
- ▶ The architecture is **causally well-connected** if there is a (uniform) way to route variables in $\text{Visibility}(z)$ to z for each output variable z .

Example

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



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

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

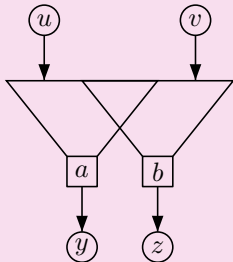
Proof. If part

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. If part

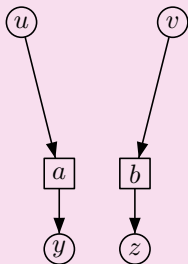
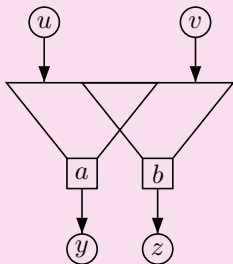


Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. If part

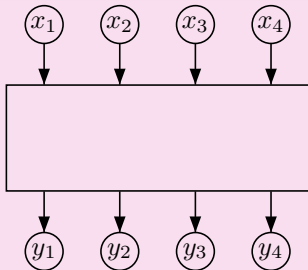


Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part



Theorem: Kupferman-Vardi (LICS'01)

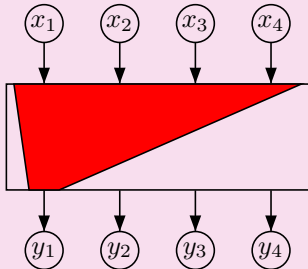
The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part



Theorem: Kupferman-Vardi (LICS'01)

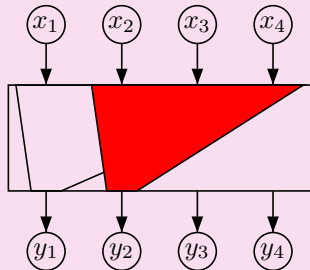
The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part



Theorem: Kupferman-Vardi (LICS'01)

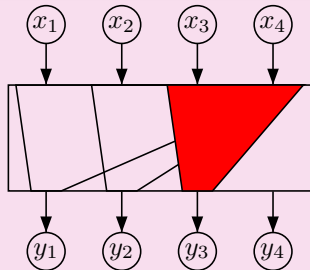
The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part



Theorem: Kupferman-Vardi (LICS'01)

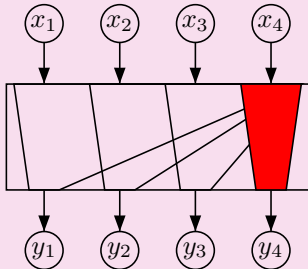
The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part



Theorem: Kupferman-Vardi (LICS'01)

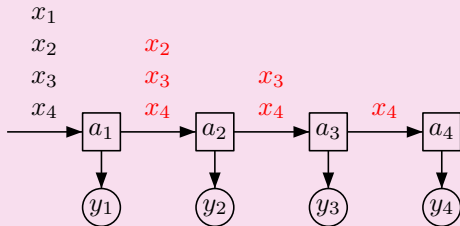
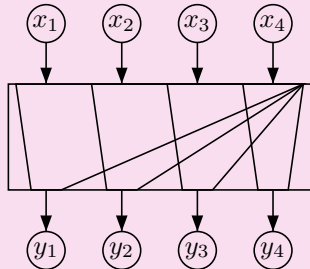
The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part



Theorem: Kupferman-Vardi (LICS'01)

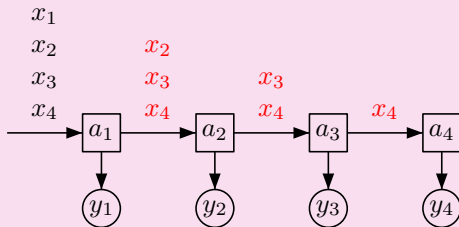
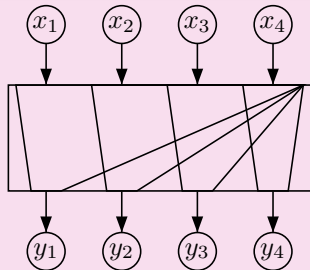
The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Causally well-connected

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

The synthesis problem for a **causally well-connected** architecture is **undecidable** if and only if we can find two output variables $z, z' \in \text{Out}$ such that the sets $\text{Visibility}(z)$ and $\text{Visibility}(z')$ are **incomparable**.

Proof. Only if part

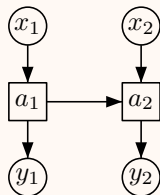


Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem with **local** strategies is decidable for pipeline architectures and CTL* specifications (or tree-automata specifications) **on all variables**.

Decidable architectures

Examples

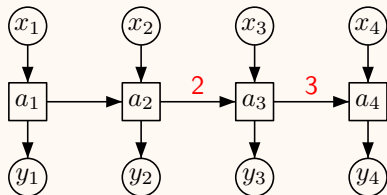


Finkbeiner-Schewe (LICS'05)

All these architectures are undecidable if we allow specifications on **all variables** (information fork criterion).

Decidable architectures

Examples

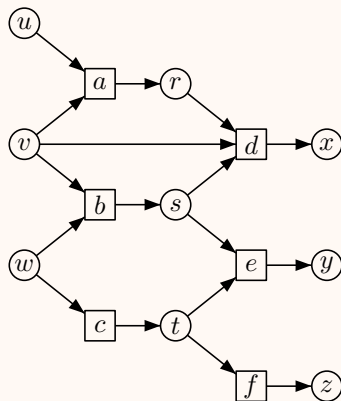
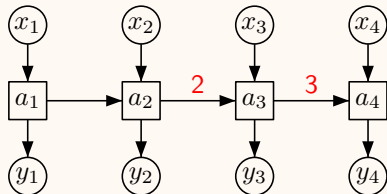


Finkbeiner-Schewe (LICS'05)

All these architectures are undecidable if we allow specifications on **all variables** (information fork criterium).

Decidable architectures

Examples

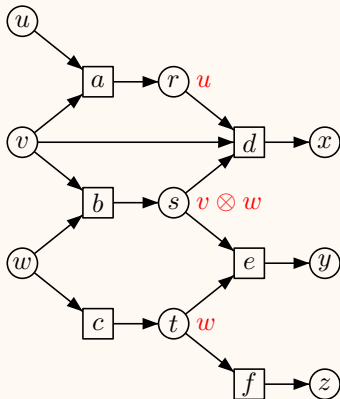
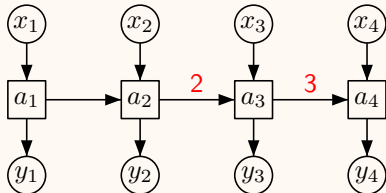


Finkbeiner-Schewe (LICS'05)

All these architectures are undecidable if we allow specifications on **all variables** (information fork criterium).

Decidable architectures

Examples

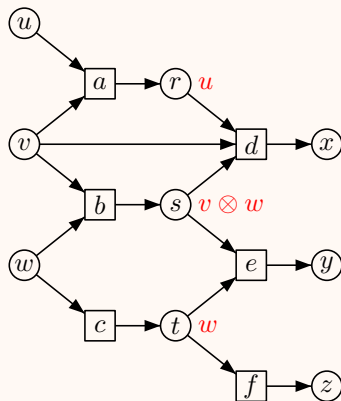
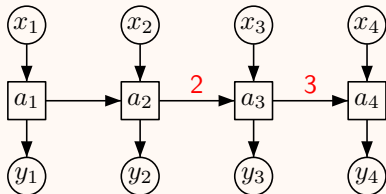


Finkbeiner-Schewe (LICS'05)

All these architectures are undecidable if we allow specifications on **all variables** (information fork criterium).

Decidable architectures

Examples



Finkbeiner-Schewe (LICS'05)

All these architectures are undecidable if we allow specifications on **all variables** (information fork criterium).

Open problems

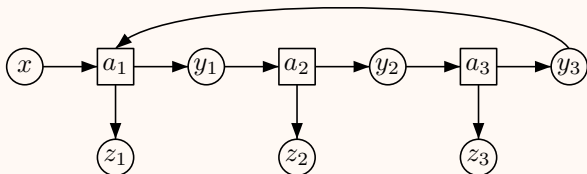
- ▶ Characterize decidable architectures for
 - ▶ input/output specifications
 - ▶ without the causally well-connectedness assumption
- ▶ Investigate **Robust specifications** for **synchronous** architectures.

Decidable with asynchronous semantics

Undecidable with synchronous semantics

Kupferman-Vardi (LICS'01)

Ring

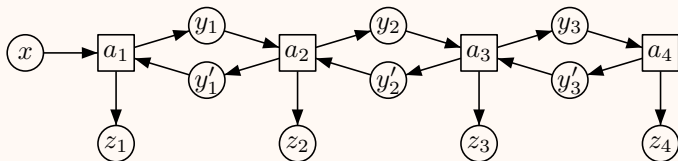


Restrictions

- ▶ Unique writer: $|W^{-1}(x)| = 1$ for all $x \in \mathcal{V}$
- ▶ Possibly several readers.
- ▶ Possibly cyclic graph (1-delay).
- ▶ No restrictions on moves: $\delta_a = Q_{R(a)} \times Q_{W(a)}$ for all $a \in \mathcal{P}$.
- ▶ Synchronous behaviors: $q^0 q^1 q^2 \dots$ where $q^n \in Q_{\mathcal{V}}$ are global states.
- ▶ program with **local** memory: $f_a : Q_{R(a)}^* \rightarrow Q_{W(a)}$ for all $a \in \mathcal{P}$.
- ▶ Specification: **CTL*** over **all** variables.

Decidability

two-way chain



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.