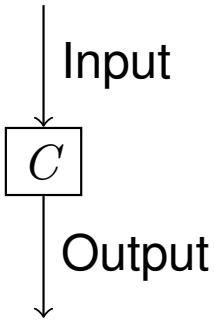
Synthesis for acyclic architectures

Igor Walukiewicz

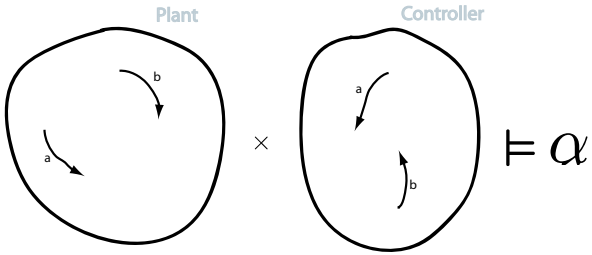
CNRS, LaBRI Bordeaux

Amsterdam
Sept 2012

Motivation



- **Church, "Applications of recursive arithmetics to the problem of circuit synthesis", 1957**
  Controlling finite automata with input/output.
- **Emerson & Clarke, "Using branching time temporal logic to synthesize synchronization skeletons" 1982**



- **Ramadge & Wonham, "The control of discrete event systems", 1987-89**
  Finite automata from control theory perspective.

Goal: A satisfactory framework for distributed synthesis

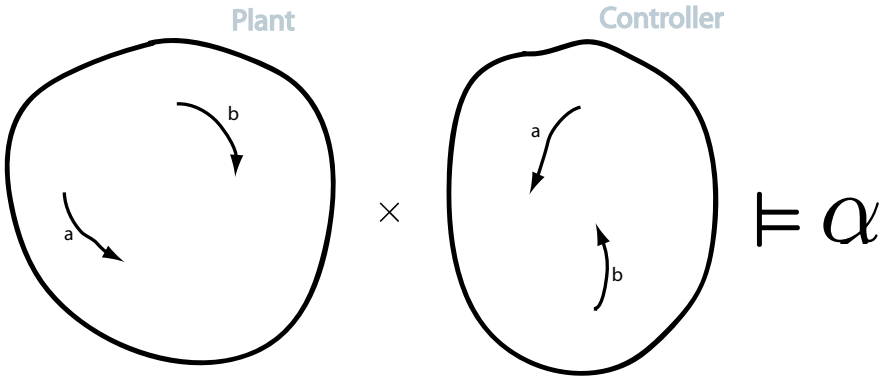
Plan

- Ramadge and Wonham setting for control.
- Ramadge & Wonham for asynchronous automata.
- Decidability and complexity for the acyclic case (and reachability conditions).

Part I

Ramadge and Wonham formulation

Ramadge and Wonham formulation



Definition

A plant is a deterministic automaton over an alphabet A of actions:

$$\mathcal{A}_P = \langle A, S_P, s_P^0, \delta_P : S \times A \dot{\to} S \rangle$$

or equivalently a regular prefix-closed language P.

Definition (Control problem)

Given $\mathcal{A}_P$, a specification α, and constraint β find a controller $\mathcal{A}_C$ such that $\mathcal{A}_C \models \beta$ and $\mathcal{A}_P \times \mathcal{A}_C \models \alpha$.

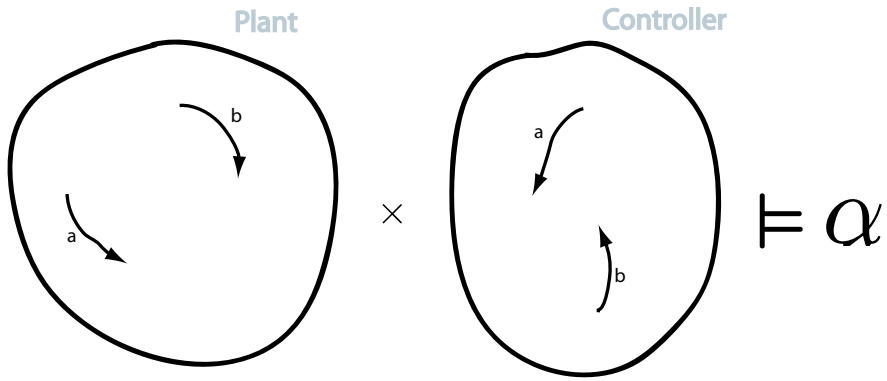
Uncontrollable actions

Definition (Uncontrollable actions)

We can divide A into A_{ctr} and A_{uctr} of controllable and uncontrollable actions.
We have additional condition:

θ_{uctr} : there are always transitions on all actions in A_{uctr}

$\delta(s, a)$ defined for all $s \in S$ and $a \in A_{uctr}$



Separation between plant and controller makes it possible to state controlability property.

"The true" Ramadge and Wonham

Formulation

- Fix an alphabet A, divided into A_{ctr} and A_{uctr}.
- We are given $P, K \subseteq A^*$ that are respectively plant language and specification language (prefix closed).
- We want $C \subseteq A^*$ such that :

$$P \cap C \subseteq K$$

This would be too simple

$$C \subseteq K \cup (A^* - P).$$

Additional requirements on C

prefix C is prefix closed.

control If $w \in C$, $a \in A_{uctr}$ then $wa \in C$.

Solution

- Take a deterministic automaton $\mathcal{A}_P$ for P.
- Suppose that K is determined by a subset E of states of $\mathcal{A}_P$.
- The controller will be a sub-automaton of $\mathcal{A}_P$ obtained by removing E and all states from which E can be reached by a sequence of uncontrollable actions.
- (We may also need to add some uncontrollable actions)

Fact

- $C = L(\mathcal{A}_C)$ is a solution to the control problem.
  $P \cap C \subseteq K$ and C satisfies (CONTROL).
- Moreover it is the biggest solution. For every other solution C': $P \cap C' \subseteq C$.

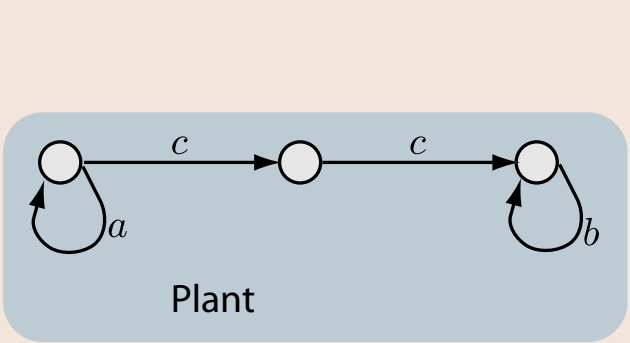
Blocking

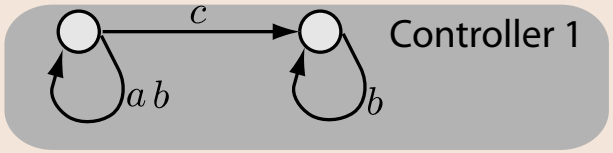
Problem

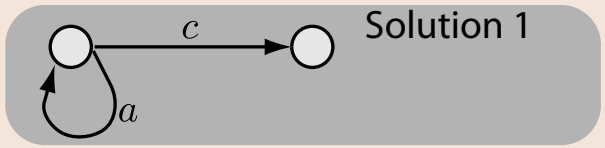
In $P \cap C$ we can have behaviours that block.

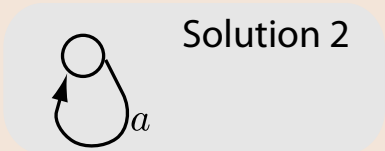
Example

Suppose a, c are controllable, but b is not.









- Second solution is avoids blocking.

Some more conditions on a controller

non-blocking Every word in $C \cap P$ has a prolongation in $C \cap P$.

marking We fix a marked language $P_M \subseteq P$. We require that every word in $C \cap P$ can be (strictly) prolonged to a word in $C \cap P_M$.

Solution avoiding blocking

Solution

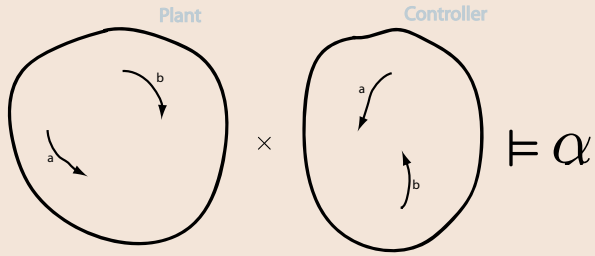
- Take automaton $\mathcal{A}_C$ constructed for control.
- A state p is blocked if there are no transitions from p.
- Remove all blocked states. Remove all uncontrollable states.
- Repeat the last step until no new states are removed.
- Automaton $\mathcal{A}_C^b$ is the restriction of $\mathcal{A}_C$ to states that are not removed.

Proposition

$\mathcal{C}_b = L(\mathcal{A}_C^b)$ is a solution to the problem and it is the biggest one.

Summing up

Ramadge and Wonham formulation



- Uncontrollable actions: controller can see them but cannot prevent them.
- Avoiding blocking: controller should not block the system.
- Solutions in polynomial time using simple graph algorithms.

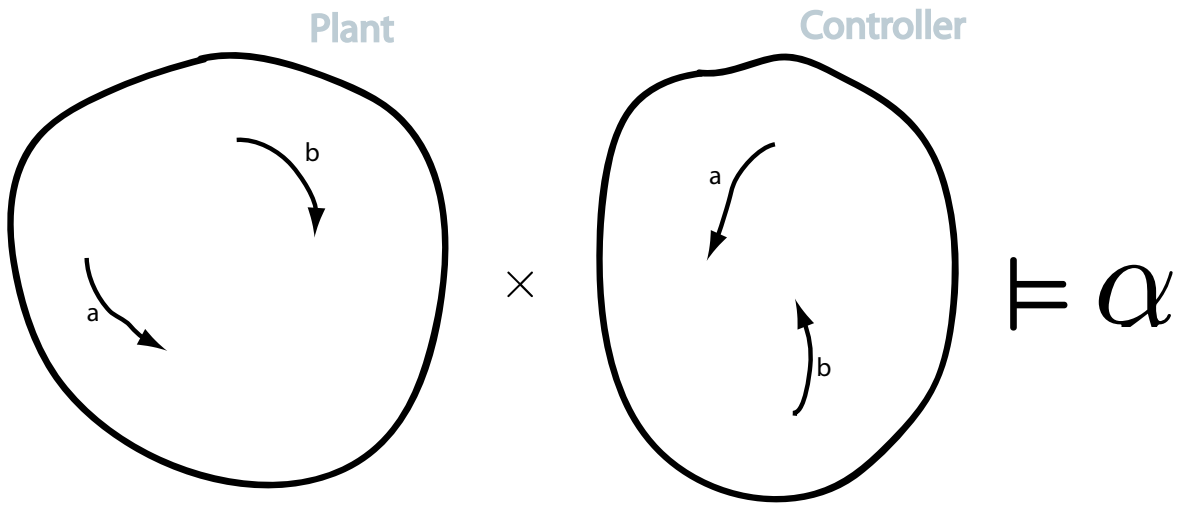
Extensions

- Unobservable action.
- With unobservable actions, the complexity of finding the biggest controller becomes exponential.
- Many other extensions for particular properties or particular restrictions.
- A more general approach through a division operation on the mu-calculus formulas. Extension of the mu-calculus with $\circlearrowleft_a$ and $\downdownarrows_{a,b}$ to talk about unobservable and indistinguishable actions.

Part II

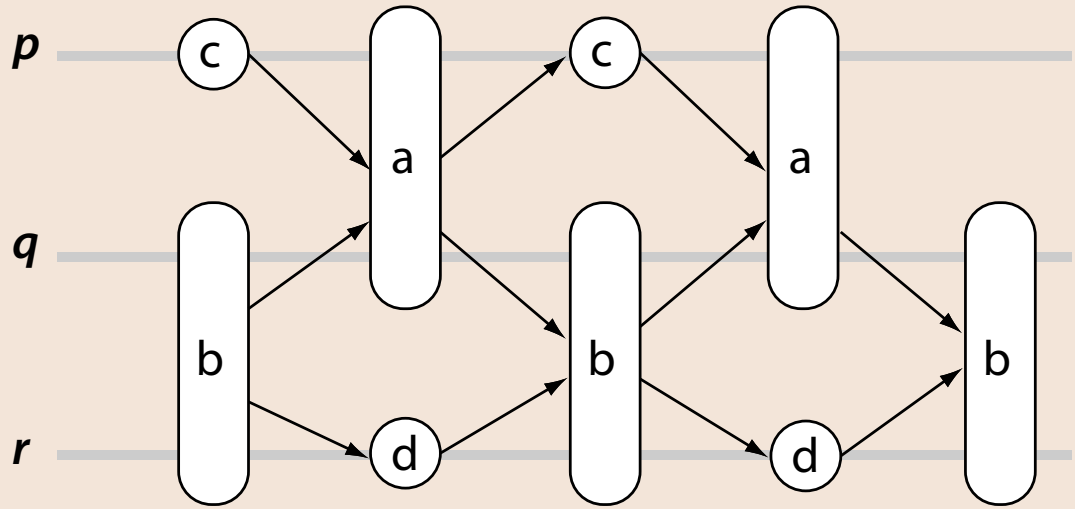
Ramadge and Wonham for asynchronous automata

Ramadge and Wonham for distributed systems



We want the plant and the controller to be distributed devices

Asynchronous automaton



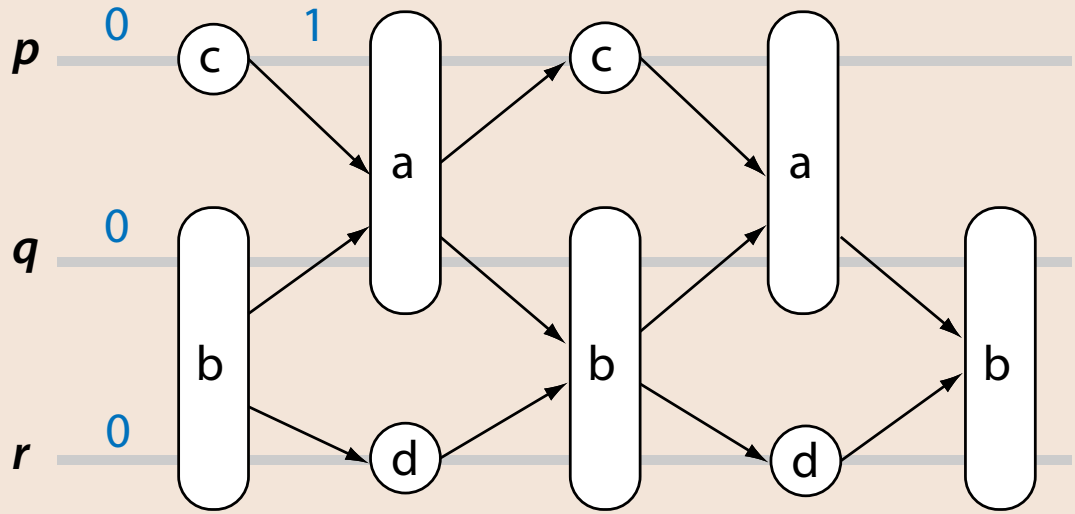
- Local states sets S_p, S_q, S_r.
- Local transitions

  $\delta_c : S_p \to S_p$

  $\delta_b : S_q \times S_r \to S_q \times S_r, \ldots$

  Process p executes local action c,

Asynchronous automaton



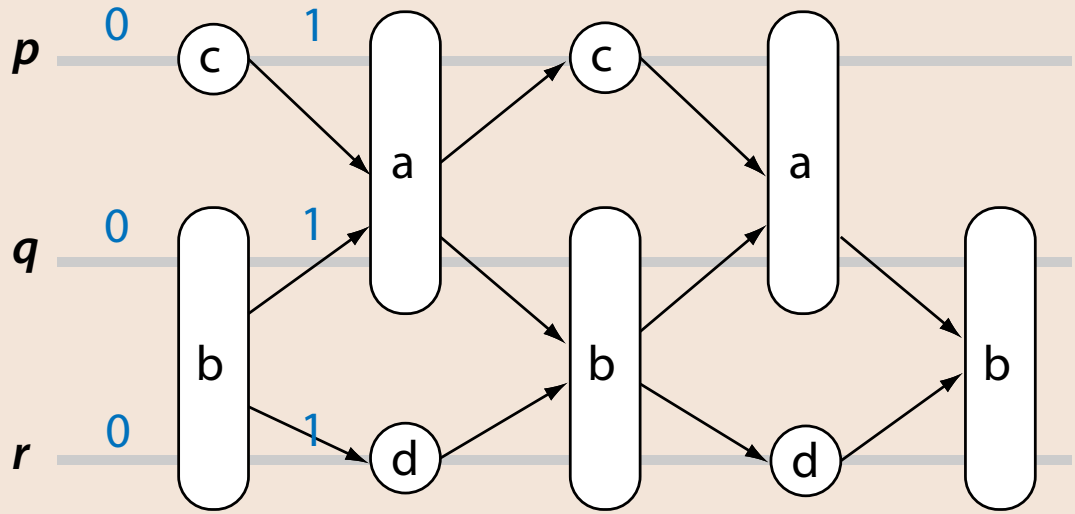
- Local states sets S_p, S_q, S_r.
- Local transitions

  $\delta_c : S_p \to S_p$

  $\delta_b : S_q \times S_r \to S_q \times S_r, \ldots$

  Process p executes local action c,
  Processes q, r synchronize on action b (and update states). . . .

Asynchronous automaton



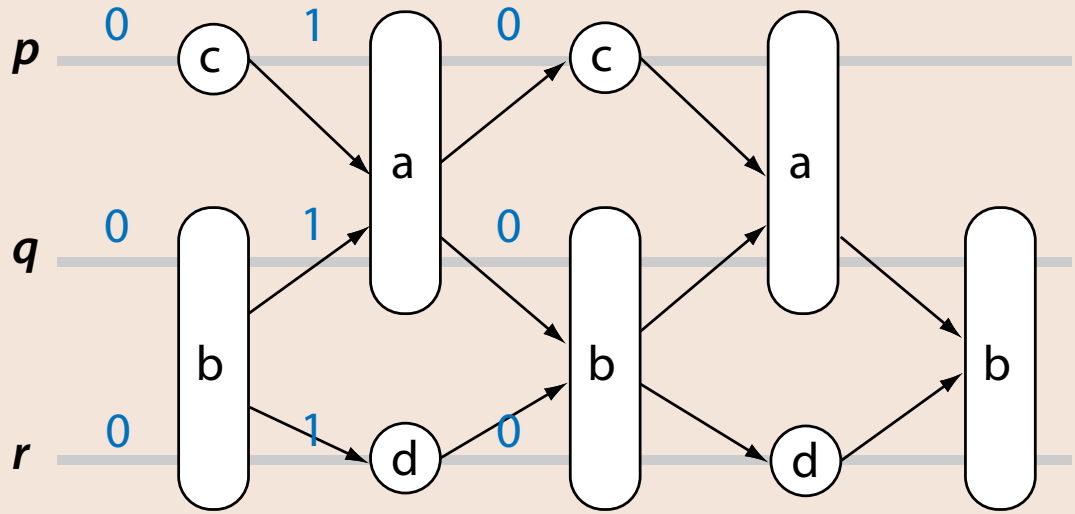
- Local states sets S_p, S_q, S_r.
- Local transitions

  $\delta_c : S_p \to S_p$

  $\delta_b : S_q \times S_r \to S_q \times S_r, \ldots$

  Process p executes local action c,
  Processes q, r synchronize on action b (and update states). . . .

Asynchronous automaton



- Local states sets S_p, S_q, S_r.
- Local transitions

  $\delta_c : S_p \to S_p$

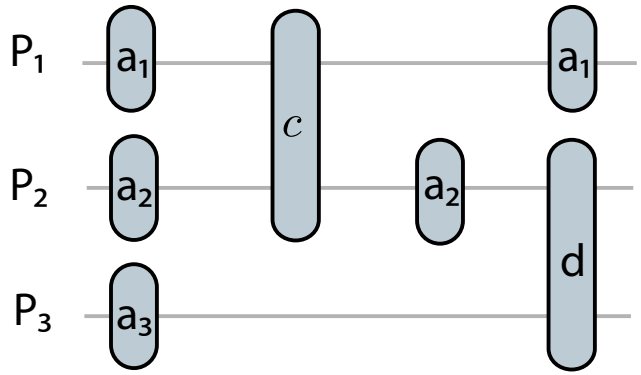
  $\delta_b : S_q \times S_r \to S_q \times S_r, \ldots$

  Process p executes local action c,
  Processes q, r synchronize on action b (and update states). . . .
- Processes evolve asynchronously.

Asynchronous automaton

Distributed alphabet

- $\mathbb{P}$: finite set of processes.
- A: finite set of letters.
- $loc : A \to (2^{\mathbb{P}} \setminus \emptyset)$: distribution of letters over processes.

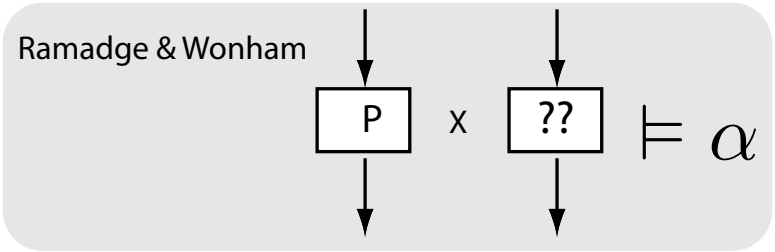


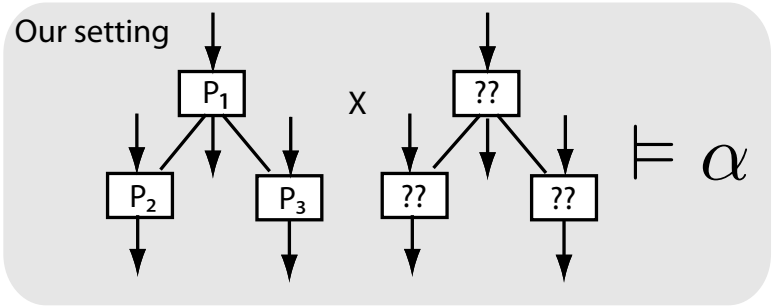
A (deterministic) asynchronous automaton

$$\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a\}_{a\in A}, F \rangle$$

- S_p states of process p
- $s_{in} \in \prod_{p\in\mathbb{P}} S_p$ is a (global) initial state,
- $\delta_a : \prod_{p\in loc(a)} S_p \dot{\to} \prod_{p\in loc(a)} S_p$ is a transition relation. Information exchange!
- $F \subseteq \prod_{p\in\mathbb{P}} S_p$ is a set of (global) final states.

Ramadge & Wonham for asynchronous automata





Ramadge & Wonham for asynchronous automata

Alphabet

Fix an alphabet $(\mathbb{P}, A, loc : A \to (2^{\mathbb{P}} \setminus \emptyset))$ with A divided into A_{ctr} and A_{uctr}.
We require that every $a \in A_{uctr}$ is local: $loc(a)$ is a singleton.

Formulation

- We are given implementable languages $P, K \subseteq A^*$ that are respectively plant language and specification language.
- We want a language $C \subseteq A^*$ such that

$$P \cap C \subseteq K \quad \text{or equivalently} \quad C \subseteq K \cup (A^* - P)$$

Additional requirements on C

implementable C is implementable.

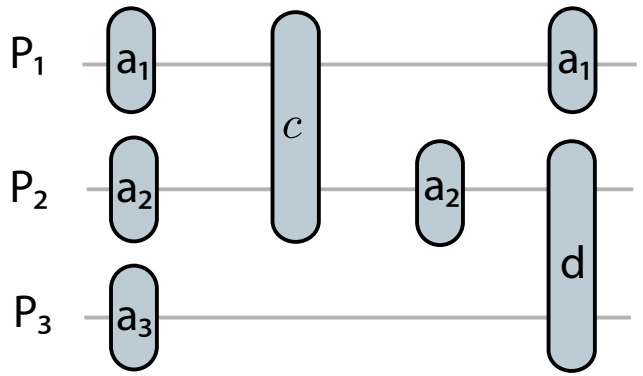
control If $w \in C$, $a \in A_{uctr}$ then $wa \in C$.

Remark: "Implementable" plays the role of "prefix-closed".

Language of an asynchronous automaton

The language of the automaton

The (regular) language of the product automaton.



Independence

- Function $loc : A \to (2^{\mathbb{P}} \setminus \emptyset)$ implies some independence on letters:

$$(a, b) \in I \quad \text{iff} \quad loc(a) \cap loc(b) = \emptyset$$

- So the language is a closed under permutations of independent letters:

$$wabv \in L(\mathcal{A}) \quad \text{implies} \quad wbav \in L(\mathcal{A})$$

Zielonka's Theorem

Fix a distribution $loc : A \to (2^{\mathbb{P}} \setminus \emptyset)$
Distribution induces closure under permutations of independent letters:

$$\text{if} \quad aIb \text{ and } wabv \in L(\mathcal{A}) \quad \text{then} \quad wbav \in L(\mathcal{A})$$

A language is trace closed if it is closed under permutations of independent letters.

Question

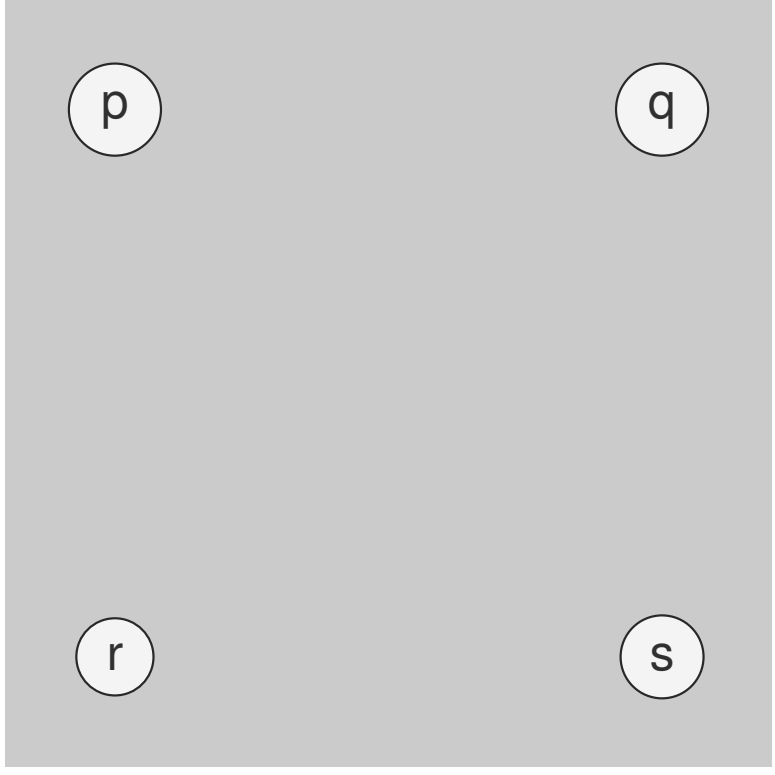
Can every trace closed regular language be recognized by an asynchronous automaton (with the same distribution)?

Zielonka's Theorem

Yes. The proof gives a finite asynchronous automaton.

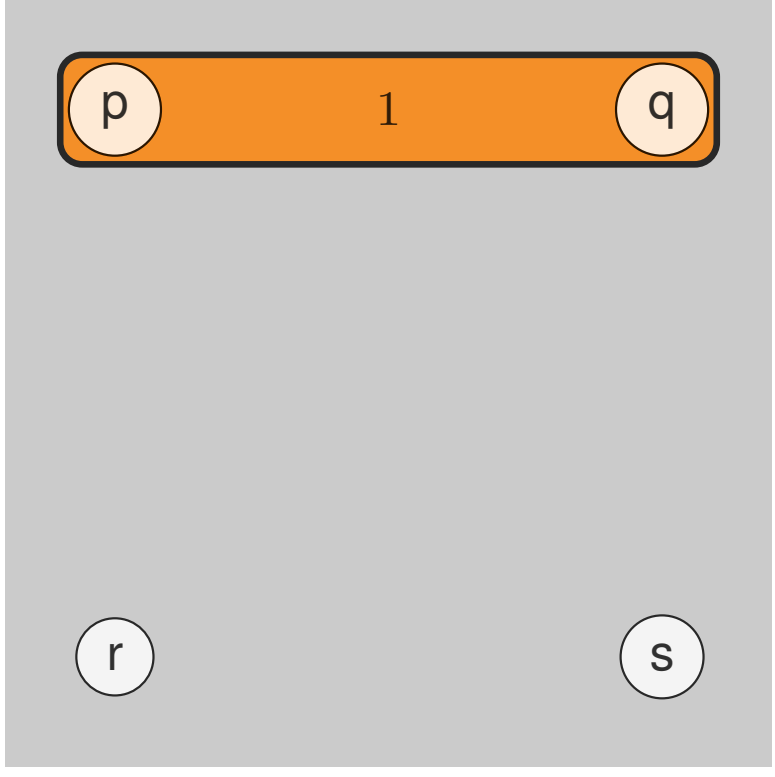
Trombones

- Four automata: p, q, r, s.
- Communicating on common actions: $\{p, q\}, \{p, r\}, \{q, s\}, \{r, s\}$.
- We want sequences $w_1 w_2 \ldots$ with $w_i \cap w_{i+1} \neq \emptyset$.



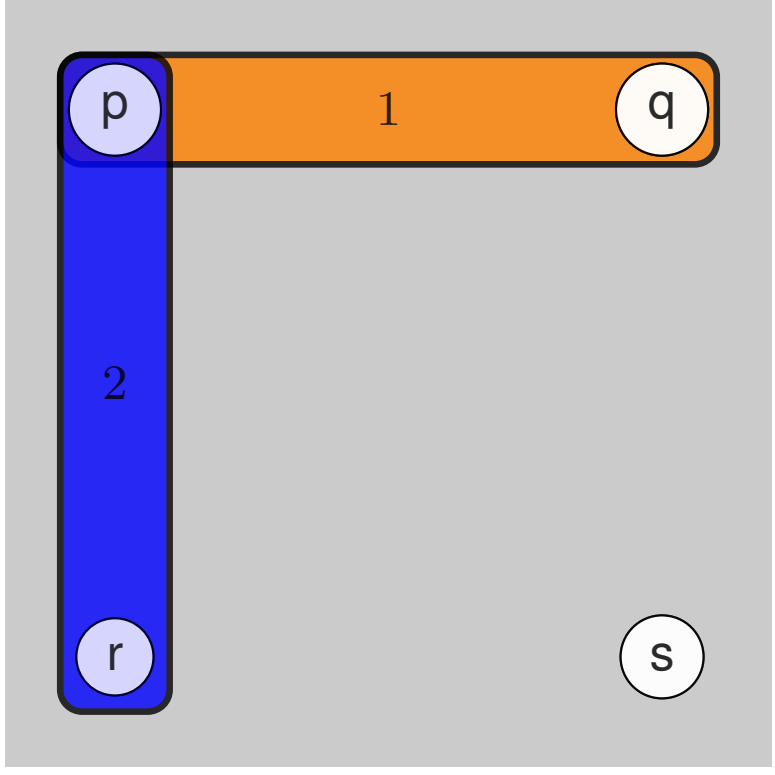
Trombones

- Four automata: p, q, r, s.
- Communicating on common actions: $\{p, q\}, \{p, r\}, \{q, s\}, \{r, s\}$.
- We want sequences $w_1 w_2 \ldots$ with $w_i \cap w_{i+1} \neq \emptyset$.



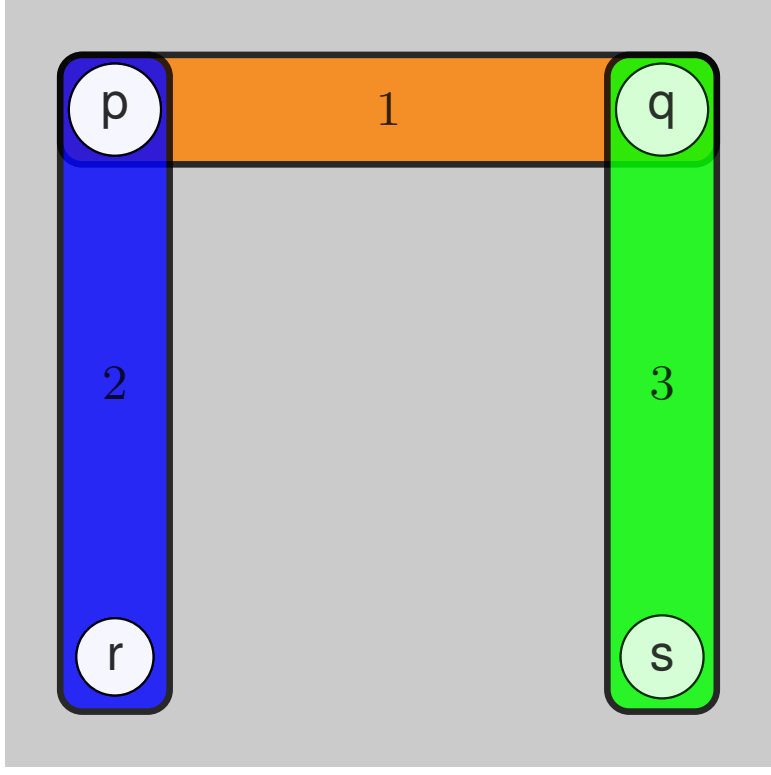
Trombones

- Four automata: p, q, r, s.
- Communicating on common actions: $\{p, q\}, \{p, r\}, \{q, s\}, \{r, s\}$.
- We want sequences $w_1 w_2 \ldots$ with $w_i \cap w_{i+1} \neq \emptyset$.



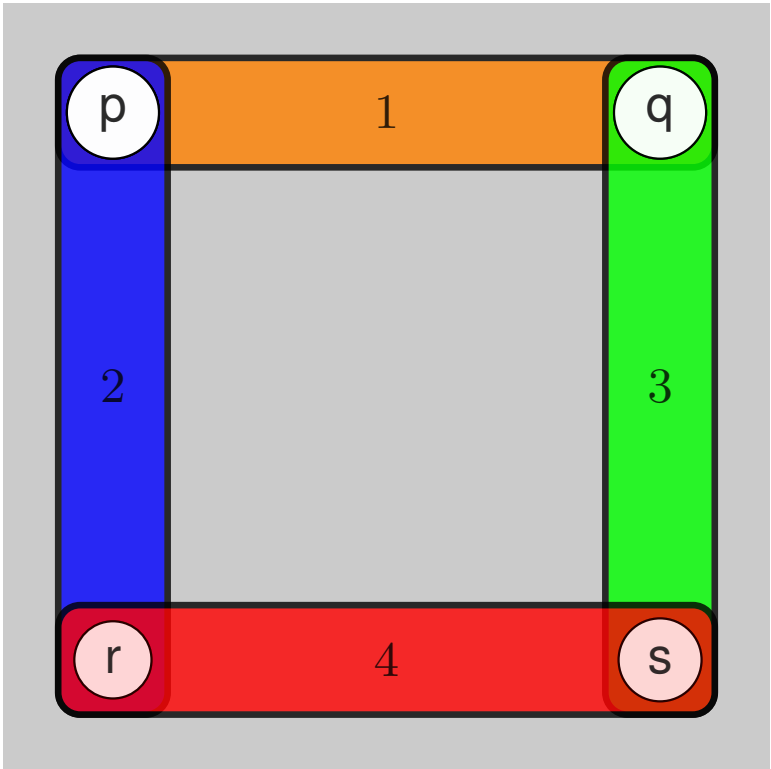
Trombones

- Four automata: p, q, r, s.
- Communicating on common actions: $\{p, q\}, \{p, r\}, \{q, s\}, \{r, s\}$.
- We want sequences $w_1 w_2 \ldots$ with $w_i \cap w_{i+1} \neq \emptyset$.



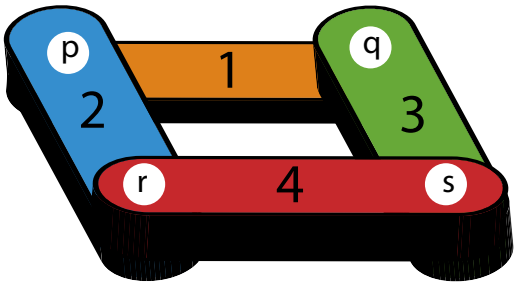
Trombones

- Four automata: p, q, r, s.
- Communicating on common actions: $\{p, q\}, \{p, r\}, \{q, s\}, \{r, s\}$.
- We want sequences $w_1 w_2 \ldots$ with $w_i \cap w_{i+1} \neq \emptyset$.

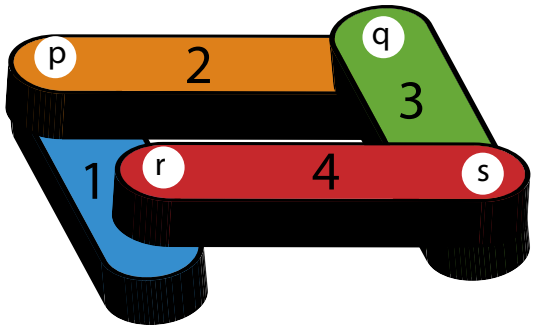


Information flow

Bad



Good



If no additional information is transmitted process r and s cannot detect a bad situation

Additional information (like time-stamps on blocks) makes it possible.

Characterizing languages of implementations

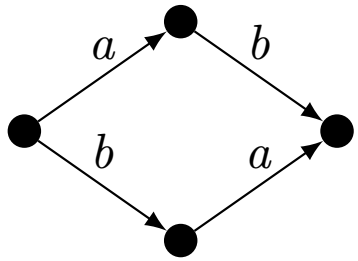
Implementations

We want to take asynchronous automata as our notion of implementation. What are languages of such automata?

Closure properties for aIb

diamond If $wabv \in L$ then $wbav \in L$.

FD If $wa, wb \in L$ then $wab \in L$.



Fact

L is recognizable by a (det.) asynchronous automaton with all states accepting

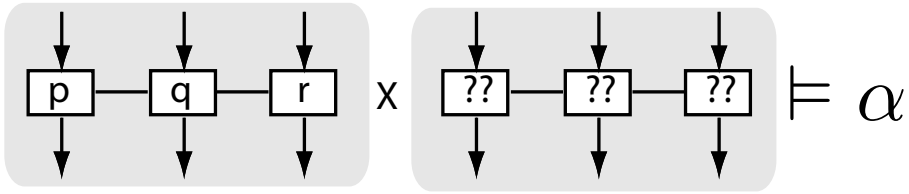
iff

L is FD, diamond and prefix closed.

Implementable language

We say that a language is implementable iff it is FD, trace and prefix closed.

Ramadge & Wonham for asynchronous automata



Alphabet

Fix an alphabet $(\mathbb{P}, A, loc : A \to (2^{\mathbb{P}} \setminus \emptyset))$ with A divided into A_{ctr} and A_{uctr}.
We require that every $a \in A_{uctr}$ is local: $loc(a)$ is a singleton.

Formulation

- We are given implementable languages $P, K \subseteq A^*$ that are respectively plant language and specification language.
- We want a language $C \subseteq A^*$ such that

$$P \cap C \subseteq K \quad \text{or equivalently} \quad C \subseteq K \cup (A^* - P)$$

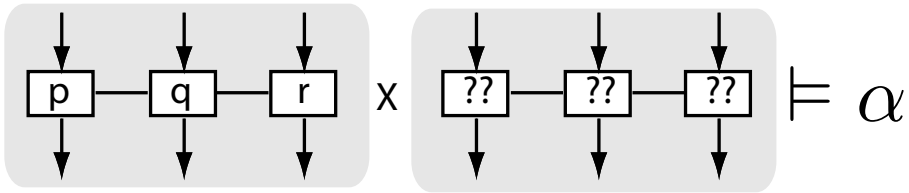
Additional requirements on C

implementable C is implementable.

control If $w \in C$, $a \in A_{uctr}$ then $wa \in C$.

Remark: "Implementable" plays the role of "prefix-closed".

Solution



Solution

- Take deterministic (standard) automata for $\mathcal{A}_P$ and $\mathcal{A}_K$, for P and K respectively. (All states accepting.)
- Construct $\mathcal{A}_C$ as for the word case.
- Use Zielonka's construction.

Fact

$C = L(\mathcal{A}_c)$ is implementable, and control closed.

Remark

If K is not implementable then this construction does not work. Moreover there is no biggest solution.

Blocked on blocking

Blocking

Some more conditions on a controller

occurrence Every letter from A appears in some word of $C \cap P$.

non-blocking Every word in $C \cap P$ has a prolongation in $C \cap P$.

marking We give a marked language $P_M \subseteq P$. We require that every word in $C \cap P$ can be (strictly) prolonged to a word in $C \cap P_M$.

Remark

The first condition is monotone, so there is a solution if C from the previous slide is a solution (i.e. if it is implementable).

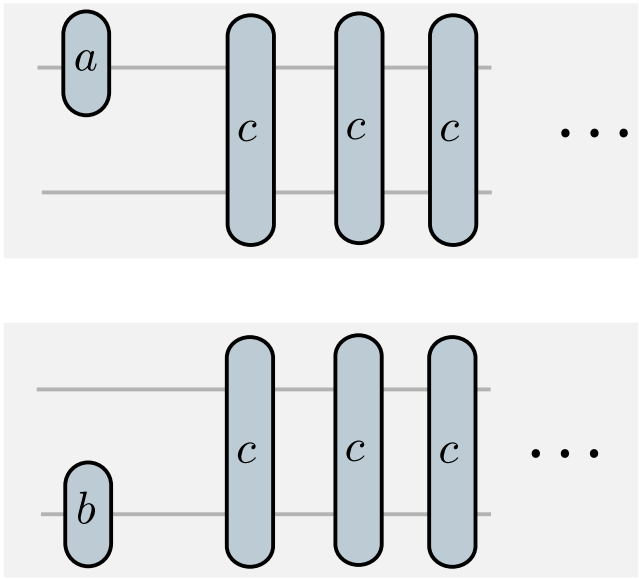
Remark

The other two conditions take us outside implementable languages.

Example

Example

- $A = \{a, b, c\}$ with aIb. All actions are controllable
- Let $P = A^*$ and $K = (a + b)c^* + ab$. Both languages are implementable.
- The maximal non-blocking controller (for sequential setting): $C = (a + b)c^*$.
- But C is not FD.
- Maximal FD non-blocking controllers: $C_a = ac^*$ and $C_b = bc^*$.



Open problem

Is it possible to find algorithmically an implementable controller satisfying control and non-blocking conditions.

Theorem (Stefanescu, Esparza, Muscholl)

*Given a regular language (not necessary trace closed) it is not decidable if it contains a (implementable) trace closed language satisfying (*OCCURRENCE*) condition.*

Last year's status

Control

- Fix an alphabet A and a distribution $loc : A \to 2^{\mathbb{P}}$.
- Given implementable P and K find an implementable C satisfying the control condition and:

$$P \cap C \subseteq K$$

- Solution: take as C the solution in the word case. It is implementable.

+nonblocking

- The solution from words, $\mathcal{C}_b$, is not implementable as it is not forward-diamond.
  - Goal: Find a smaller solution that is implementable (by a deterministic device),

- This approach is equivalent to deciding the winner in asynchronous games. Shown decidable for co-graph alphabets [Gastin & Lerman & Zeitoun].
- Solving such games is also known decidable for special cases of automata [Madhusudan & Thiagarajan].

Summing up

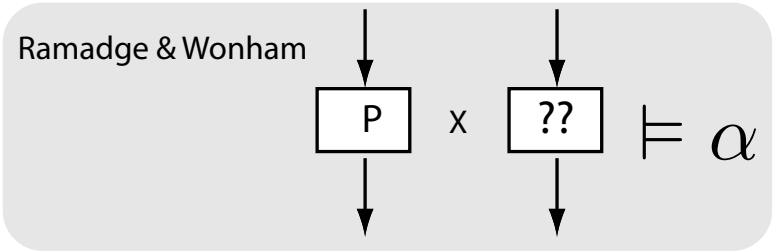
- A framework for distributed control using asynchronous automata in Ramadge and Wonham formulation.
- Decidability of the case disregarding blocking thanks to Zielonka's Thm.
- Trombones example shows why passing additional information with a communication is important.
- The blocking condition adds new dimension to the problem.

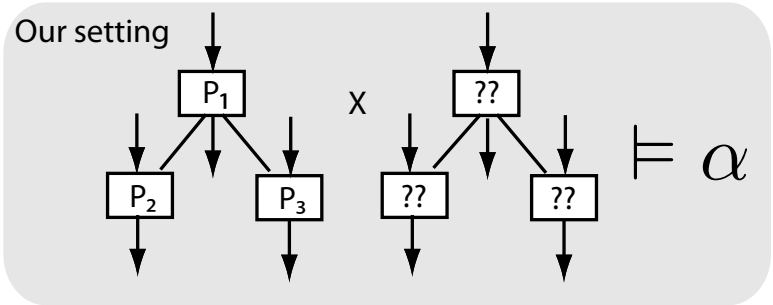
Decidability for acyclic architectures

Main result

Theorem

The control problem is decidable for acyclic architectures and (proper) reachability conditions. The complexity of the problem is non-elementary.

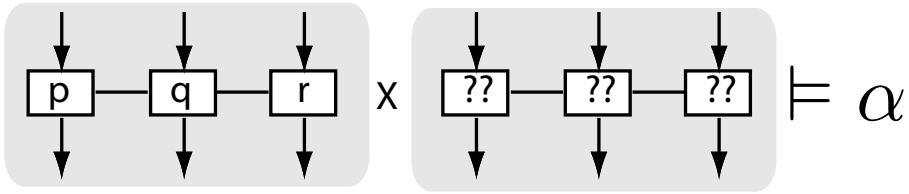




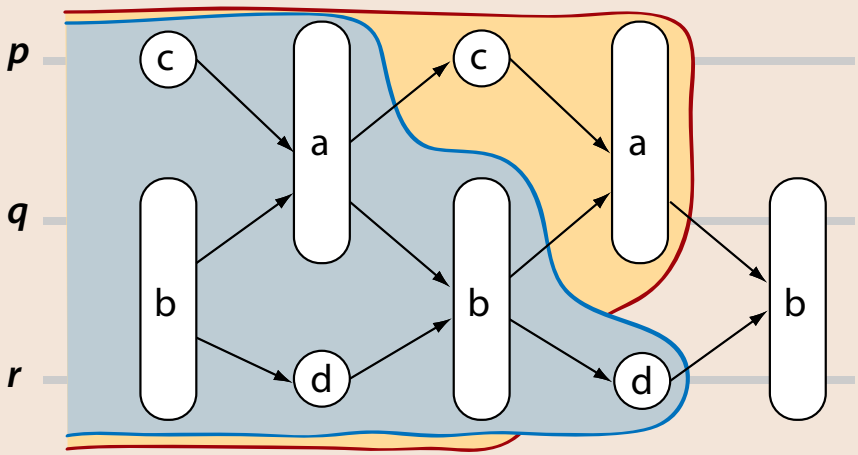
Acyclic architecture: Communication graph has no cycles.

Recall restrictions: uncontrollable actions are local, communications only binary.

Upper bound (1/2)

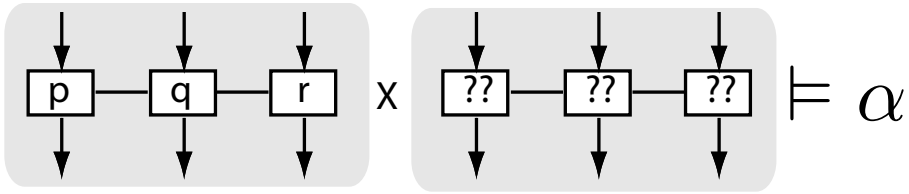


How to play the control game

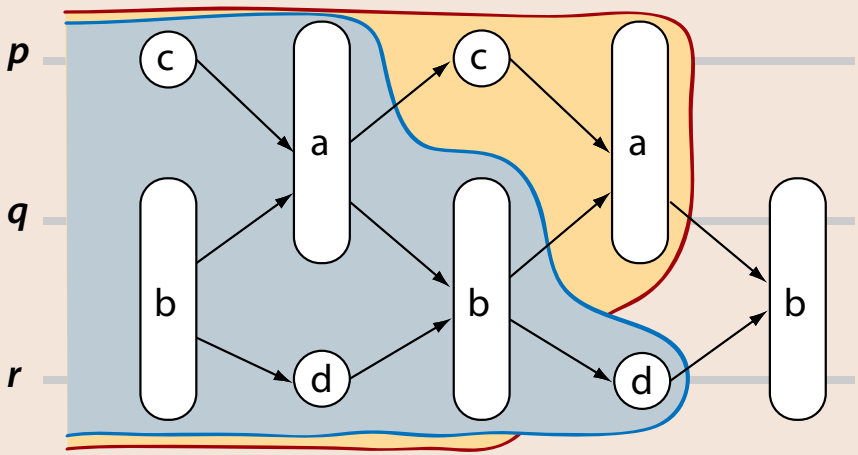


- Control strategies σ_p : $\text{Views}_p \to 2^{\Sigma_p}$.
- Environment chooses $b \in \sigma_q(\text{View}_q(x)) \cap \sigma_r(\text{View}_r(x))$.
- Uncontrollable (local) actions are always possible.

Upper bound (1/2)



How to play the control game



- Control strategies σ_p : $\text{Views}_p \to 2^{\Sigma_p}$.
- Environment chooses $b \in \sigma_q(\text{View}_q(x)) \cap \sigma_r(\text{View}_r(x))$.
- Uncontrollable (local) actions are always possible.

Main idea

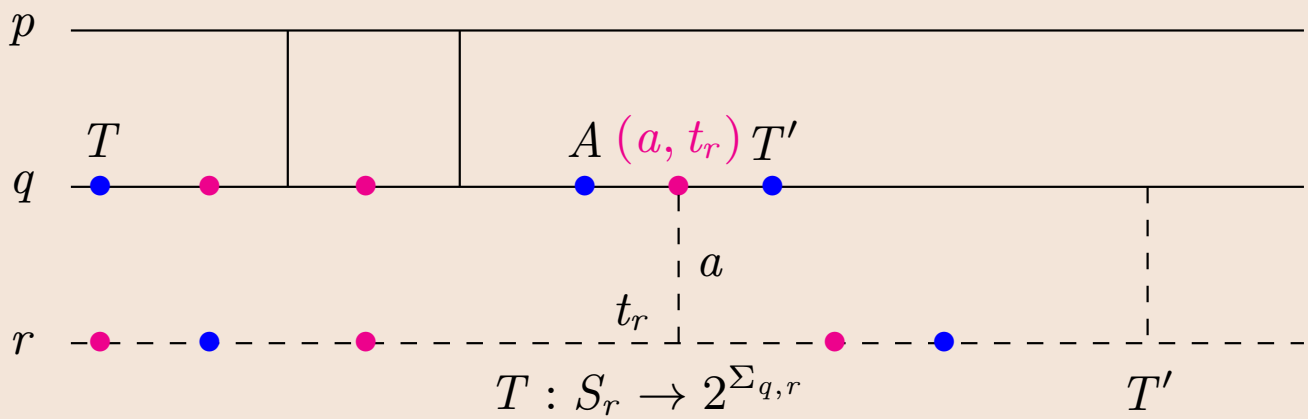
Process q "simulates" leaf process r, by choosing r's strategy for the r-local run until the next synchronization with q.
New state space of q: exponential in $|S_r|$.

Upper bound (2/2)

Proof idea

- Fix a leaf r with parent q.
- Wlog. q's strategy proposes one of the following:
  1. only local actions,
  2. only synchronizations with r,
  3. only synchronizations with processes other than r.

Simulate r by q



Environment action (a, t_r): choose $t_r \in \mathrm{dom}(T)$ and $a \in A \cap T(t_r)$.

Acyclic architectures: proof ideas

Lower bound

Nested counters:

- level 1: $0, 1, \ldots, 2^n - 1$

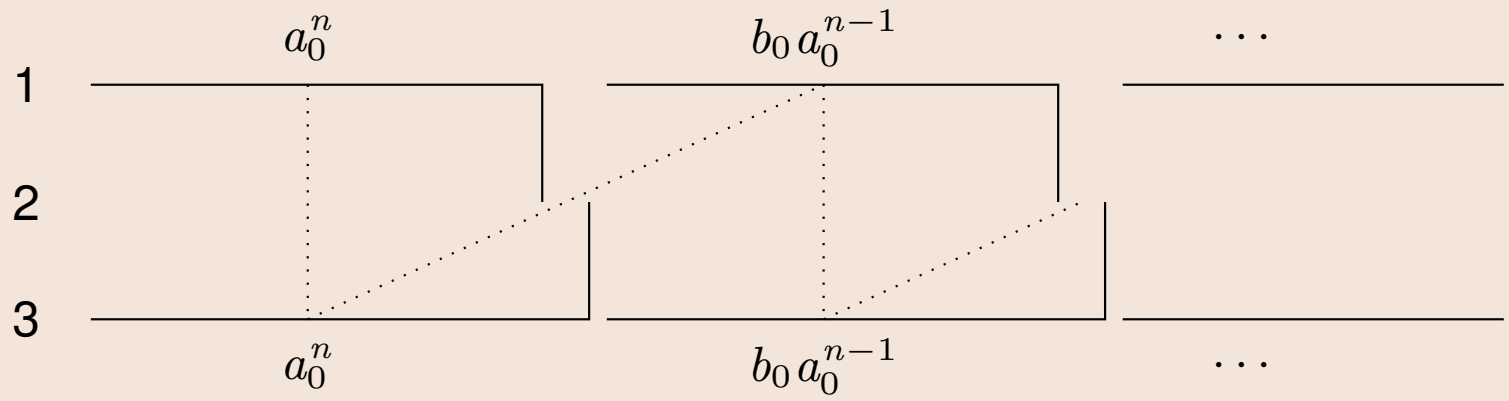
$$\underbrace{a_0 \cdots a_0}_{n} \# b_0 \underbrace{a_0 \cdots a_0}_{n-1} \# a_0 b_0 \underbrace{a_0 \cdots a_0}_{n-2} \# \cdots \# \underbrace{b_0 \cdots b_0}_{n} \#$$

- level 2: $0, 1, \ldots, 2^{2^n} - 1$

$$a_1 \mathsf{bin}(0)\, a_1 \mathsf{bin}(1) \cdots a_1 \mathsf{bin}(2^n - 1) \# b_1 \mathsf{bin}(0)\, a_1 \mathsf{bin}(1) \cdots a_1 \mathsf{bin}(2^n - 1) \# \cdots$$

- level k: ...

Level 1



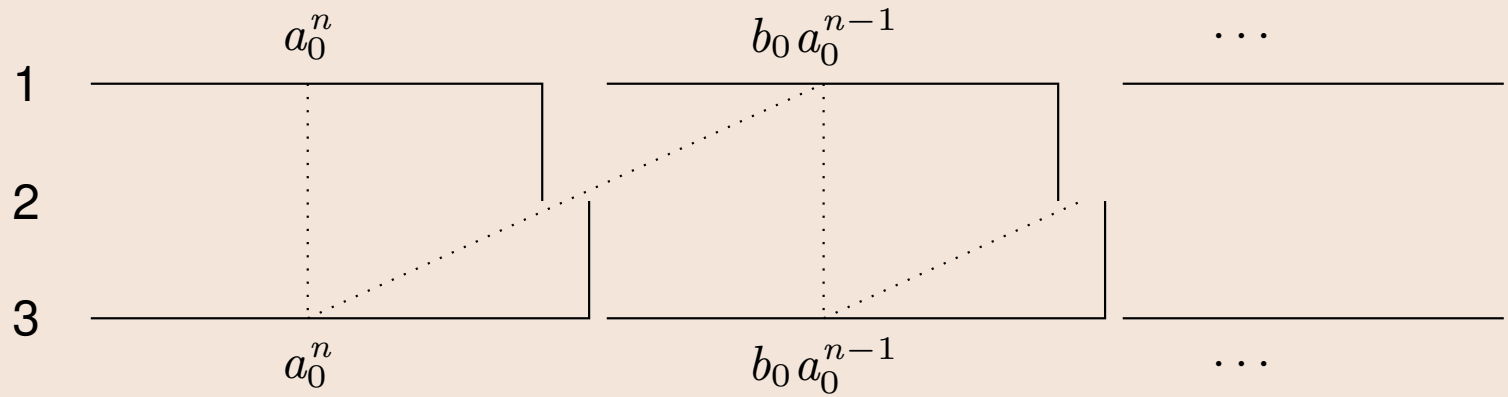
Processes 1,2,3.

Environment can ask for a pair of bits: either $\uparrow$ (above) or $\nearrow$.

If test initiated, then processes $1, 3$ synchronize over 2 and check (e.g. for $\uparrow$):

Positions are unequal or bits are equal.

Level 1



Processes 1,2,3.

Environment can ask for a pair of bits: either $\uparrow$ (above) or $\nearrow$.

If test initiated, then processes $1, 3$ synchronize over 2 and check (e.g. for $\uparrow$):

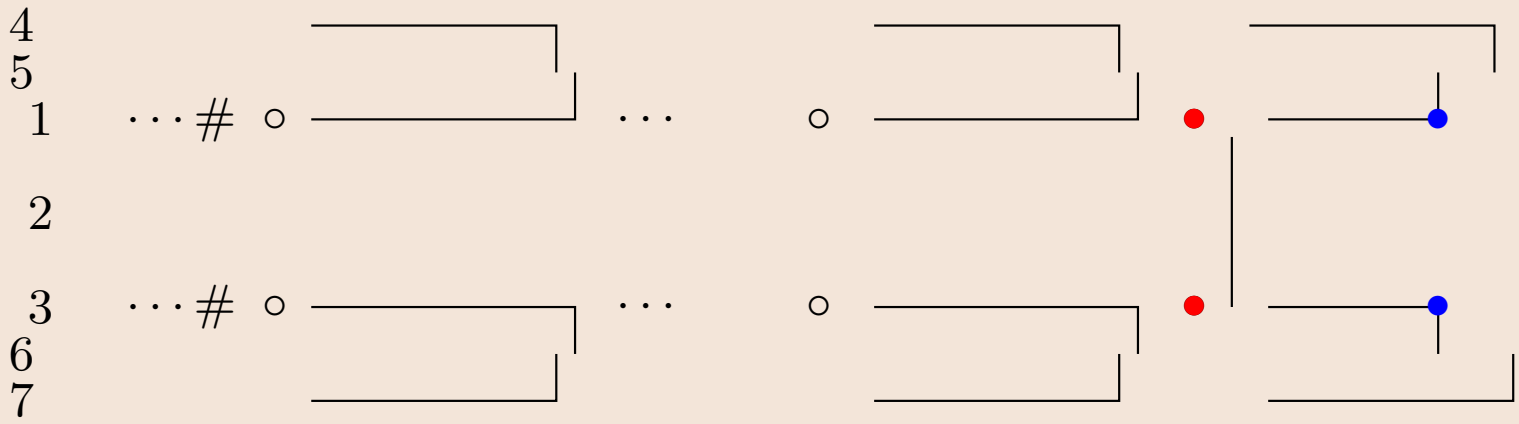
Positions are unequal or bits are equal.

Thm.

The control problem over the architecture 1 ——— 2 ——— 3 is EXPTIME-complete.

Level 2

- Processes $1, 2, \ldots, 7$.



- Processes 1,5,4 and 3,6,7 resp., check level-1 counters.
- Processes 1,2,3 check level-2 counters.
- Level-2 test:
  - Environment chooses bits.
  - Processes 1,3 synchronize over 2 and accept if bits are equal.
  - If level-1 positions are unequal processes 1,2 choose a distinguishing position • and the game continues as if the environment had chosen that position on level 1.

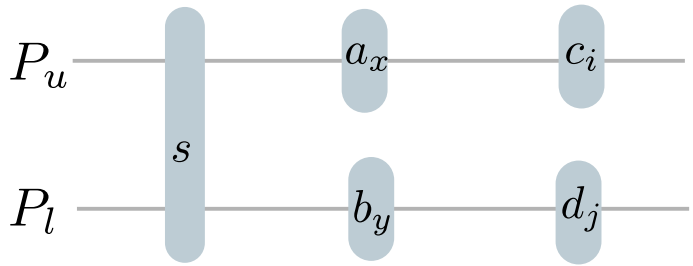
Conclusions

- Synthesis of finite automata is relatively well-understood $\mapsto$ MSOL theory of trees.
- Despite some effort, distributed synthesis is much less understood.
- There are two families of distributed synthesis problems: based on boxes and based on asynchronous automata.
- They differ in the way information is transmitted in the system.
- In the box approach we know of very few decidable cases.
- The main result shows that in the asynchronous automata approach the problem is decidable for acyclic architectures.
- The decidability for all architectures is still open.

An example

Take the alphabet $\{s\} \cup \{a_i, b_i, c_i, d_i : i = 0, 1\}$, where a's and b's are uncontrollable.

Consider linearizations of traces of the form



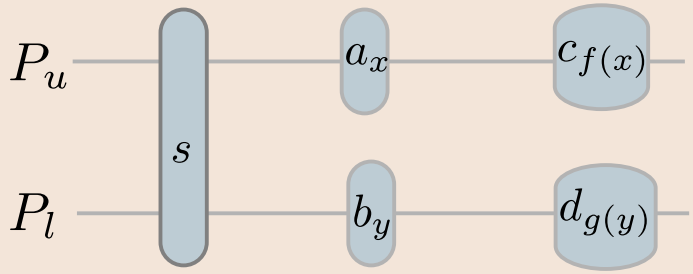
where $x = j$ or $y = i$.

The prefix closure of this language is a trace language.
It is controllable when linearizations are concerned:

$$sa_x \ldots d_j \ldots \quad \text{or} \quad sb_y \ldots c_i \ldots$$

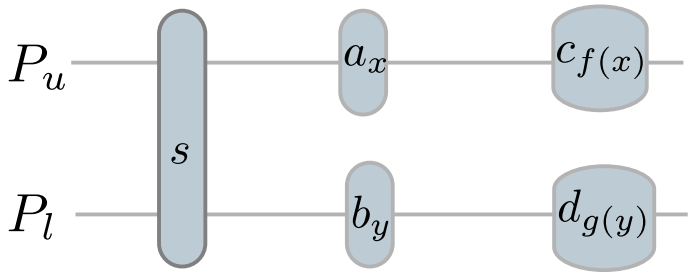
Can it be implemented without blocking?

Two strategies $f, g : \{0, 1\} \to \{0, 1\}$.



Question: does there exist a winning pair of strategies?
I.e. $y = f(x)$ or $x = g(y)$ for every $x, y \in \{0, 1\}$.

Example (cont.)

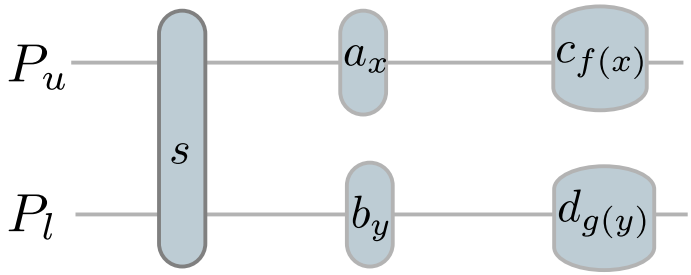


Question: Do there exist functions $f, g : \{0,1\} \to \{0,1\}$ such that for every for every $x, y \in \{0,1\}$ either: $y = f(x)$ or $x = g(y)$.

First attempt

$f(x) = x$ and $g(x) = x$ then
when taking $x = 0$, $y = 1$ we have $y \neq f(x)$ and $x \neq g(y)$.Wrong

Example (cont.)



Question: Do there exist functions $f, g : \{0,1\} \to \{0,1\}$ such that for every for every $x, y \in \{0,1\}$ either: $y = f(x)$ or $x = g(y)$.

First attempt

$f(x) = x$ and $g(x) = x$ then
when taking $x = 0$, $y = 1$ we have $y \neq f(x)$ and $x \neq g(y)$.Wrong

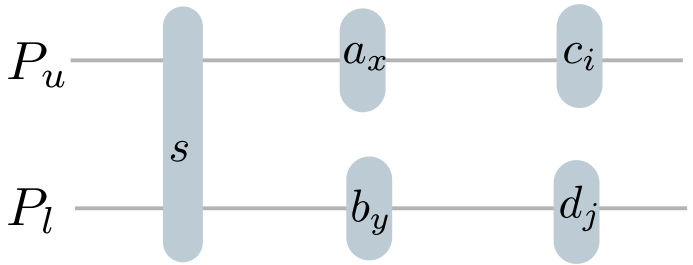
Second attempt

$f(x) = x$ and $g(y) = 1 - y$.
We have "$y = f(x)$ or $x = g(y)$" is equivalent to "$y = x$ or $x = 1 - y$". OK

Modified example

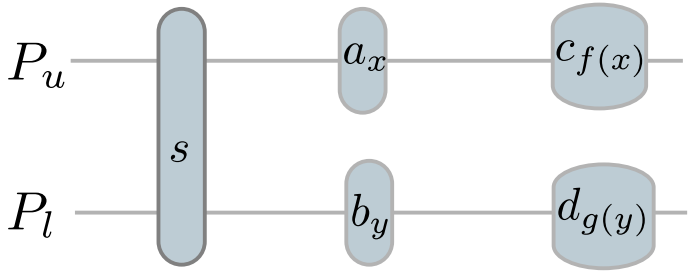
Take the alphabet $\{s\} \cup \{a_i, b_i, c_i, d_i : i = 0, 1, 2\}$.

Consider linearizations of traces of the form



where $x = j$ or $y = i$. We assume that a_x and b_y are uncontrollable.

Two strategies $f, g : \{0, 1, 2\} \to \{0, 1, 2\}$.

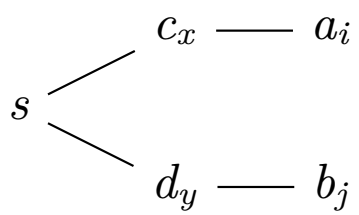


Question: does there exist a winning pair of strategies?
I.e. $y = f(x)$ or $x = g(y)$ for every $x, y \in \{0, 1, 2\}$.

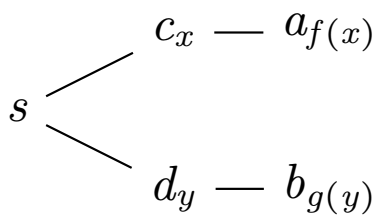
No: Take y such that there is at most one x with $f(x) = y$.
There are two x such that $x \neq g(y)$.
Take the one with $f(x) \neq y$.

Example (cont.)

Take the alphabet $\{s\} \cup \{a_i, b_i, c_i, d_i : i = 0, 1, 2\}$.
Consider language of linearizations of traces of the form



where $x = j$ or $y = i$. We assume that c_x and d_y are uncontrollable.
Two strategies $f, g : \{0, 1, 2\} \to \{0, 1, 2\}$.
Executions respecting strategies:



Question: does there exist a winning pair of strategies? I.e. $y = f(x)$ or $x = g(y)$ for every $x, y \in \{0, 1\}$.
No: Take y such that there is at most one x with $f(x) = y$ There are two x such that $x \neq g(y)$. Take the one with $f(x) \neq y$.