

# LOGIC IN COMPUTER SCIENCE - CONTROL AND SYNTHESIS, FROM A DISTRIBUTED PERSPECTIVE

Anca Muscholl

Nordic Logic School, Stockholm, August 2017

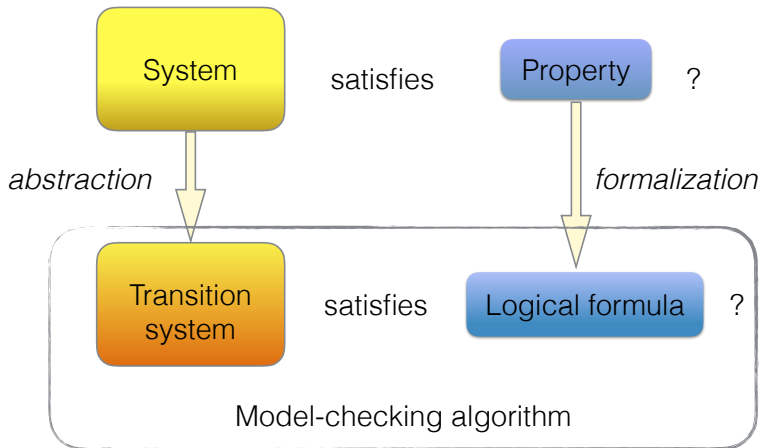
## OUTLINE

- ① Basics on automata and logic
- ② Basics on synthesis and control: Church's problem
- ③ Distributed synthesis: Pnueli/Rosner model
- ④ Control for distributed automata. New decidability results.

# I. Basics on automata and logic

# VERIFICATION

## MODEL-CHECKING



# VERIFICATION

## SYNTHESIS

From a logical formula  $F$ : construct a transition system that is equivalent to  $F$ .

*“Write programs that are correct by construction.”*

# AUTOMATA, LOGIC AND VERIFICATION

## TRANSITION SYSTEMS

$$\mathcal{S} = \langle S, AP, S_0, \longrightarrow, \lambda \rangle$$

- $S$  is the set of states
- $AP$  is a (finite) set of atomic propositions
- $S_0 \subseteq S$  is the set of initial states
- $\longrightarrow \subseteq S \times S$  is the transition relation
- $\lambda : S \rightarrow 2^{AP}$  labels states by sets of atomic propositions

## FINITE AUTOMATA

$$\mathcal{A} = \langle S, \Sigma, S_0, (\overset{a}{\longrightarrow})_{a \in \Sigma}, \text{Acc} \rangle$$

- $S$  is the finite set of states
- $\Sigma$  is a (finite) alphabet
- $S_0 \subseteq S$  is the set of initial states
- $(\overset{a}{\longrightarrow})_{a \in \Sigma} \subseteq S \times S$  is the transition relation (function = **deterministic**)
- $\text{Acc}$  is the acceptance condition

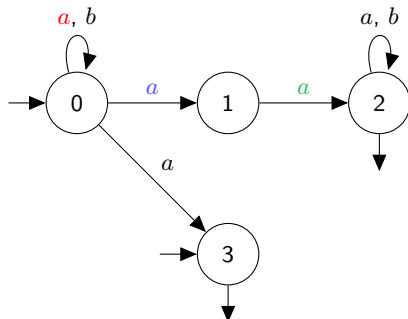
# AUTOMATA

## REGULAR LANGUAGE

$$\mathcal{A} = \langle S, \Sigma, S_0, (\xrightarrow{a})_{a \in \Sigma}, \text{Acc} \rangle$$

- Word  $w = a_0 a_1 \dots$ : possibly infinite sequence of symbols from a finite alphabet  $\Sigma$ . Set of finite words  $\Sigma^*$ , set of infinite words  $\Sigma^\omega$ .
- Run  $\rho : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots$  of  $\mathcal{A}$  on  $w$ .
- Successful run:  $s_0 \in S_0$  and  $\rho$  satisfies  $\text{Acc}$ .
- Accepted language  $L(\mathcal{A}) = \{w \mid \exists \text{ some successful run of } \mathcal{A} \text{ on } w\}$ .

## EXAMPLE (AUTOMATON)



- $S = \{0, 1, 2, 3\}$
- $\Sigma = \{a, b\}$
- $S_0 = \{0, 3\}$ ,  $F = \{2, 3\}$   
( $\text{Acc}$ : end in  $F$ )

$$a a a \in L(\mathcal{A})$$

# AUTOMATA

## REGULAR

A word language  $L \subseteq \Sigma^*$  (resp.  $L \subseteq \Sigma^\omega$ ) is regular (resp.  $\omega$ -regular) if it is accepted by some finite automaton.

## ACCEPTANCE

Acc is a set of states  $F$ .

- Finite words: a successful run must end in  $F$ .
- Infinite words: a successful run must visit  $F$  infinitely often (Büchi automaton).

## DETERMINISM

- Over finite words, deterministic and non-deterministic automata are equi-expressive.
- Over infinite words, deterministic Büchi automata are less expressive than non-deterministic ones.
- More powerful acceptance conditions are required for deterministic automata over infinite words, e.g. the parity condition: states have priorities and a run is successful if the highest priority visited infinitely often is even.



## LOGICS

- Temporal logics: Linear Temporal Logic (LTL), Computation Tree Logic (CTL, CTL<sup>\*</sup>),  $\mu$ -calculus
- Monadic Second-Order Logic (MSO)

## MODEL-CHECKING

Basic paradigm for checking whether a transition system  $\mathcal{S}$  satisfies a formula  $\varphi$ :

- Translate the formula  $\varphi$  (or its negation) into some (equivalent) automaton  $\mathcal{A}_\varphi$ .
- Build the product between  $\mathcal{S}$  and  $\mathcal{A}_\varphi$ , and check for non-emptiness.

## MONADIC SECOND-ORDER LOGIC (MSO): SYNTAX

- First-order variables  $x, y, \dots$  and second-order variables  $X, Y, \dots$
- Atomic propositions  $P_a(x)_{a \in \Sigma}$ ,  $S(x)$ ,  $x < y$ ,  $x \in X$ .
- Boolean connectors  $\neg, \wedge, \vee, \dots$ , quantifiers  $\exists, \forall$ .

## SEMANTICS

Relational structure associated with a word  $w = a_1 a_2 \dots$  over  $\Sigma$ , with  $\text{dom}(w) = \{1, 2, \dots\}$ :

$$\langle \text{dom}(w), \text{succ}, <, (P_a)_{a \in \Sigma} \rangle$$

- $\text{succ}$  is successor relation on  $\text{dom}(w)$
- $<$  is linear order on  $\text{dom}(w)$
- $P_a = \{k \in \text{dom}(w) \mid a_k = a\}$

Second-order variables  $X, Y, \dots =$  sets of positions (subsets of  $\text{dom}(w)$ )

## LANGUAGE OF $\varphi$

$w \models \varphi$	$w$ models $\varphi$
$L(\varphi) = \{w \in \Sigma^+ \mid w \models \varphi\}$	finitary language of $\varphi$
$L(\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}$	infinitary language of $\varphi$

## EXAMPLES

$$\begin{aligned} \exists X_0 \exists X_1 \quad & \left( \forall x \left( (x \in X_0 \vee x \in X_1) \wedge (x \in X_0 \Leftrightarrow x \notin X_1) \right) \wedge 1 \in X_1 \wedge \right. \\ & \forall x \left( (x \in X_0 \Leftrightarrow \text{succ}(x) \in X_1) \wedge (x \in X_1 \Leftrightarrow \text{succ}(x) \in X_0) \right) \\ & \left. \forall x \left( x \in X_1 \Rightarrow P_a(x) \right) \right) =: \text{ODD}_a \end{aligned}$$

*Every odd position carries an  $a$ .*

$$\varphi := \text{ODD}_a \wedge \text{EVEN}_b \qquad L(\varphi) = (ab)^+$$

## LANGUAGE OF $\varphi$

$w \models \varphi$	$w$ models $\varphi$
$L(\varphi) = \{w \in \Sigma^+ \mid w \models \varphi\}$	finitary language of $\varphi$
$L(\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}$	infinitary language of $\varphi$

## EXAMPLES

$$\begin{aligned} \exists X_0 \exists X_1 \quad & \left( \forall x \left( (x \in X_0 \vee x \in X_1) \wedge (x \in X_0 \Leftrightarrow x \notin X_1) \right) \wedge 1 \in X_1 \wedge \right. \\ & \forall x \left( (x \in X_0 \Leftrightarrow \text{succ}(x) \in X_1) \wedge (x \in X_1 \Leftrightarrow \text{succ}(x) \in X_0) \right) \\ & \left. \forall x (x \in X_1 \Rightarrow P_a(x)) \right) =: \text{ODD}_a \end{aligned}$$

*Every odd position carries an  $a$ .*

$$\varphi := \text{ODD}_a \wedge \text{EVEN}_b \qquad L(\varphi) = (ab)^+$$

$$\varphi' := \forall x \left( P_a(x) \Rightarrow P_b(\text{succ}(x)) \wedge P_b(x) \Rightarrow P_a(\text{succ}(x)) \wedge P_a(\min) \wedge P_b(\max) \right)$$

$(ab)^*$  is definable in first-order logic (FO)

## BÜCHI-ELGOT-TRAKHTENBROT THEOREM

A language of finite words is regular iff it is definable in monadic second-order logic (MSO). Both conversions are effective.

### PROOF SKETCH.

- ① direct implication: describe accepting runs in MSO
  - Partition  $\text{dom}(w)$  into sets  $X_s$ , one for each state  $s$ .
  - First position belongs to  $\bigcup_{s \in S_0} X_s$ . Last one (resp. infinitely many positions) belongs to  $\bigcup_{f \in F} X_f$ .
  - Consistency of automaton transitions: for each  $k \in \text{dom}(w)$ ,  $s \in S$ ,  $a \in \Sigma$ ,

$$k \in X_s \wedge P_a(k) \implies \bigcup_{s \xrightarrow{a} s'} (k+1) \in X_{s'}$$

- ② reverse implication: regular languages are closed under union (disjunction), projection (existential quantification) and complement (negation). Closure under complementation is easy, using determinization.

# AUTOMATA OVER INFINITE WORDS

Recall: deterministic Büchi automata are less expressive than non-deterministic ones.

More powerful acceptance conditions are required for deterministic automata, e.g. the **parity condition** (“Rabin chain condition”, Mostowski’85, Emerson-Jutla’91):

## Parity automaton

$$\mathcal{A} = \langle S, \Sigma, S_0, (\xrightarrow{a})_{a \in \Sigma}, \ell : S \rightarrow \{0, \dots, d\} \rangle$$

where  $\ell(s)$  is called *priority* of state  $s$ . A run is accepting if the maximal priority visited infinitely often is even.

## DETERMINIZATION

- Classical determinization constructions, from Büchi to deterministic Muller/Rabin acceptance: McNaughton (1966), Safra (1988), Muller-Schupp (1995).
- Piterman (2006), Kähler-Wilke (2008), Schewe (2009) and Liu-Wang (2009) provide single exponential construction from non-deterministic Büchi automata to deterministic parity automata.

# MSO OVER TREES

## BINARY TREES

- Finite binary tree over alphabet  $\Sigma$  = partial mapping

$$t : \{0, 1\}^* \rightarrow \Sigma$$

such that  $\text{dom}(t)$  is finite, prefix-closed and  $x1 \in \text{dom}(t)$  iff  $x0 \in \text{dom}(t)$ , for all  $x$ .

$\epsilon$ : root, 0, 1: children of the root, etc.

- Infinite binary tree over  $\Sigma$  = total mapping  $t : \{0, 1\}^* \rightarrow \Sigma$ .

## MSO

Two successors left/right:

- First-order variables  $x, y, \dots$  and second-order variables  $X, Y, \dots$
- Atomic propositions  $P_a(x)_{a \in \Sigma}$ ,  $\text{succ}_0(x)$ ,  $\text{succ}_1(x)$ ,  $x < y$ ,  $x \in X$ .
- Boolean connectors  $\neg, \wedge, \vee, \dots$ , quantifiers  $\exists, \forall$ .

## TREE AUTOMATA

$$\mathcal{A} = \langle S, \Sigma, S_0, (\xrightarrow{a})_{a \in \Sigma}, \text{Acc} \rangle$$

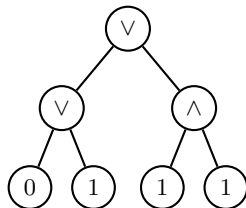
- a finite set of states  $S$ ,
- a finite alphabet  $\Sigma$ ,
- set of initial states  $S_0 \subseteq S$ ,
- a transition relation  $(\xrightarrow{a})_{a \in \Sigma} \subseteq S \times (S^2 \cup S)$ ,
- an acceptance condition  $\text{Acc}$ .

**Deterministic:**  $\xrightarrow{a}$  is function  $S \rightarrow (S^2 \cup S)$



# AUTOMATA OVER FINITE TREES

## EXAMPLE



Language of all trees that evaluate to 1 (at the root).

Acc is a set  $F \subseteq S$  of final states. Run is successful if it ends on all leaves in a final state.

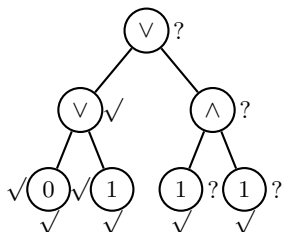
- $S = \{?, \sqrt{\phantom{x}}\}$
- $S_0 = \{?\}$ ,  $F = \{\sqrt{\phantom{x}}\}$
- $\xrightarrow{V}$   
 $= \{(\sqrt{\phantom{x}}, (\sqrt{\phantom{x}}, \sqrt{\phantom{x}})), (?, (? , *)), (?, (*, ?))\}$
- $\xrightarrow{\wedge}$   
 $= \{(\sqrt{\phantom{x}}, (\sqrt{\phantom{x}}, \sqrt{\phantom{x}})), (?, (? , ?))\}$
- $\xrightarrow{0}$   
 $= \{(\sqrt{\phantom{x}}, \sqrt{\phantom{x}})\}$
- $\xrightarrow{1}$   
 $= \{(\sqrt{\phantom{x}}, \sqrt{\phantom{x}}), (?, \sqrt{\phantom{x}})\}$

## DETERMINIZATION

Deterministic bottom-up tree automata are an equivalent model.

# AUTOMATA OVER FINITE TREES

## EXAMPLE



Language of all trees that evaluate to 1 (at the root).

Acc is a set  $F \subseteq S$  of final states. Run is successful if it ends on all leaves in a final state.

- $S = \{?, \sqrt{\phantom{x}}\}$
- $S_0 = \{?\}, F = \{\sqrt{\phantom{x}}\}$
- $\xrightarrow{V} = \{(\sqrt{\phantom{x}}, (\sqrt{\phantom{x}}, \sqrt{\phantom{x}})), (?, (?,*)), (?, (*,?))\}$   
 $\xrightarrow{\wedge} = \{(\sqrt{\phantom{x}}, (\sqrt{\phantom{x}}, \sqrt{\phantom{x}})), (?, (?,*))\}$   
 $\xrightarrow{0} = \{(\sqrt{\phantom{x}}, \sqrt{\phantom{x}})\},$   
 $\xrightarrow{1} = \{(\sqrt{\phantom{x}}, \sqrt{\phantom{x}}), (?, \sqrt{\phantom{x}})\}$

## DETERMINIZATION

Deterministic bottom-up tree automata are an equivalent model.

# AUTOMATA OVER INFINITE TREES

## BÜCHI

Acc is a set of (final) states  $F \subseteq S$ . Run is successful if on every path,  $F$  is visited infinitely often.

## PARITY

Acc is a labeling of states by priorities from  $\{0, \dots, p\}$ . Run is successful if on every path, the highest priority seen infinitely often is even.

## DETERMINISM, COMPLEMENTATION

- Over infinite trees, deterministic automata are strictly weaker. So complementation is a challenge.
- Büchi tree automata are less expressive than parity tree automata.
- Parity tree automata can be complemented (games!). This is the crucial step in Rabin's theorem, cf. next slide.

## THATCHER-WRIGHT'68, DONER'70

A language of finite trees is accepted by some tree automaton iff it is definable in MSO. Both conversions are effective.

The following result is deeply intertwined with the theory of infinite 2-player games:

## RABIN'69

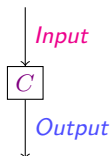
A language of infinite trees is accepted by some parity tree automaton iff it is definable in MSO. Both conversions are effective.

## COR.

MSO over infinite, binary trees is decidable.

## II. Basics on games and controller synthesis

# CHURCH'S PROBLEM (1963) “LOGIC, ARITHMETIC AND AUTOMATA”

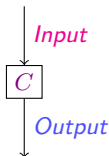


## PROBLEM

- *Given:* specification  $\mathcal{R} \subseteq (\{0, 1\} \times \{0, 1\})^\omega$  relating inputs/outputs.
- *Output:* I/O device  $C : \{0, 1\}^* \rightarrow \{0, 1\}$  s.t.  $(x, C(x)) \in \mathcal{R}$  for all inputs  $x$ .

*Controller  $C$  must react correctly on every input.*

# CHURCH'S PROBLEM (1963) “LOGIC, ARITHMETIC AND AUTOMATA”



## PROBLEM

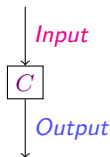
- *Given:* specification  $\mathcal{R} \subseteq (\{0, 1\} \times \{0, 1\})^\omega$  relating inputs/outputs.
- *Output:* I/O device  $C : \{0, 1\}^* \rightarrow \{0, 1\}$  s.t.  $(x, C(x)) \in \mathcal{R}$  for all inputs  $x$ .

*Controller  $C$  must react correctly on every input.*

## REMARKS

- The specification  $\mathcal{R}$  is provided in an effective way, by an MSO formula or a Büchi automaton.
- The problem is more complicated than just requiring  $\forall x \exists y. (x, y) \in \mathcal{R}$ : controller  $C$  must react *continuously* on inputs.

# CHURCH'S PROBLEM (1963) “LOGIC, ARITHMETIC AND AUTOMATA”



## PROBLEM

- *Given:* specification  $\mathcal{R} \subseteq (\{0, 1\} \times \{0, 1\})^\omega$  relating inputs/outputs.
- *Output:* I/O device  $C : \{0, 1\}^* \rightarrow \{0, 1\}$  s.t.  $(x, C(x)) \in \mathcal{R}$  for all inputs  $x$ .

*Controller  $C$  must react correctly on every input.*

## REMARKS

- The specification  $\mathcal{R}$  is provided in an effective way, by an MSO formula or a Büchi automaton.
- The problem is more complicated than just requiring  $\forall x \exists y . (x, y) \in \mathcal{R}$ : controller  $C$  must react *continuously* on inputs.

Church's problem:

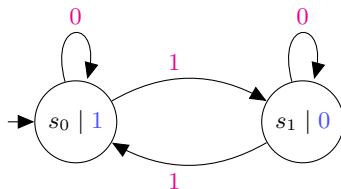
*Synthesis of **open** systems: systems **reacting** on input from environment.*



## EXAMPLES

### Ex. 1

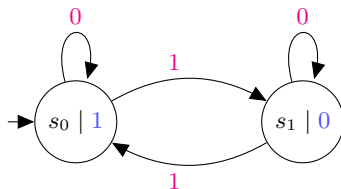
$\mathcal{R}$ : "the output is 1 iff the number of previous inputs equal to 1, is even"



## EXAMPLES

### Ex. 1

$\mathcal{R}$ : "the output is 1 iff the number of previous inputs equal to 1, is even"



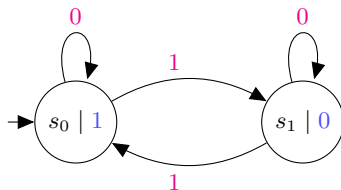
### Ex. 2

$\mathcal{R}$ : "the output is 1 iff some future input is 1"

## EXAMPLES

### Ex. 1

$\mathcal{R}$ : "the output is 1 iff the number of previous inputs equal to 1, is even"



### Ex. 2

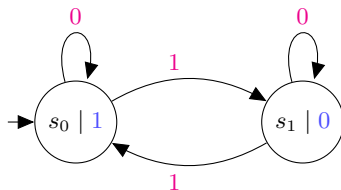
$\mathcal{R}$ : "the output is 1 iff some future input is 1"

No solution.

## EXAMPLES

### Ex. 1

$\mathcal{R}$ : "the output is 1 iff the number of previous inputs equal to 1, is even"



### Ex. 2

$\mathcal{R}$ : "the output is 1 iff some future input is 1"

No solution.

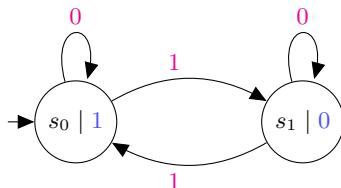
### Ex. 3

$\mathcal{R}$ : "The output has infinitely many 1's if the input has infinitely many 1's".

## EXAMPLES

### Ex. 1

$\mathcal{R}$ : "the output is 1 iff the number of previous inputs equal to 1, is even"



### Ex. 2

$\mathcal{R}$ : "the output is 1 iff some future input is 1"

No solution.

### Ex. 3

$\mathcal{R}$ : "The output has infinitely many 1's if the input has infinitely many 1's".

Various solutions (e.g. copying the input, outputting always 1, ...).

## EXAMPLES (CONT.)

Ex. 4

$\mathcal{R}$ : “*The output has finitely many 1’s iff the input has infinitely many 1’s*”.

## EXAMPLES (CONT.)

### Ex. 4

$\mathcal{R}$ : “*The output has finitely many 1's iff the input has infinitely many 1's*”.

- on  $0^\omega$  the output should contain at least one 1, say after  $k_1$  steps;
- on  $0^{k_1}10^\omega$  the output should contain at least one more 1, say after another  $k_2$  steps;
- In the limit: on  $0^{k_1}10^{k_2}1\cdots$  the output will contain infinitely many 1's.

## EXAMPLES (CONT.)

### Ex. 4

$\mathcal{R}$ : “*The output has finitely many 1’s iff the input has infinitely many 1’s*”.

- on  $0^\omega$  the output should contain at least one 1, say after  $k_1$  steps;
- on  $0^{k_1}10^\omega$  the output should contain at least one more 1, say after another  $k_2$  steps;
- In the limit: on  $0^{k_1}10^{k_2}1\cdots$  the output will contain infinitely many 1’s.

No solution.



# CHURCH'S PROBLEM AND LOGIC

## SPECIFICATIONS

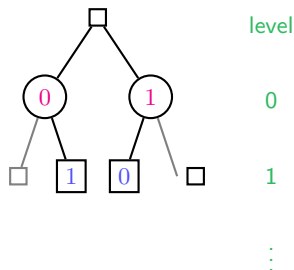
Specification  $\mathcal{R} \subseteq (\{0, 1\}^{\omega})^{\omega}$ : finite description, Büchi automaton or MSO.

## TREES

Synthesis is concerned with **trees**:

$$t : \{0, 1\}^{\omega} \rightarrow \Sigma$$

$C : \{0, 1\}^* \rightarrow \{0, 1\}$  is a subset of the tree.



# CHURCH'S PROBLEM AND LOGIC

## SPECIFICATIONS

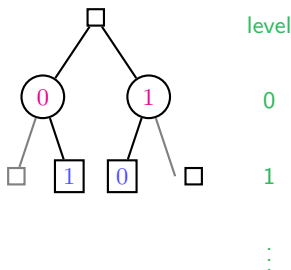
Specification  $\mathcal{R} \subseteq (\{0, 1\}^{\omega})^{\omega}$ : finite description, Büchi automaton or MSO.

## TREES

Synthesis is concerned with trees:

$$t : \{0, 1\}^{\omega} \rightarrow \Sigma$$

$C : \{0, 1\}^* \rightarrow \{0, 1\}$  is a subset of the tree.



## CONTROLLER SYNTHESIS

The existence of a controller  $C$  satisfying property  $\mathcal{R}$  can be expressed by an MSO formula over the infinite binary tree. Rabin's theorem on the decidability of MSO provides the decidability of controller synthesis. If a controller  $C$  exists, then it is a finite automaton.

## CHURCH'S PROBLEM AND LOGIC

The existence of a controller  $C$  satisfying property  $\mathcal{R}$  can be expressed by an MSO formula over the infinite binary tree. Rabin's theorem on the decidability of MSO provides the decidability of controller synthesis. If a controller  $C$  exists, then it is a finite-state automaton.

### PROOF.

Construct MSO formula  $\varphi_{\mathcal{R}}$  that is satisfiable over the infinite binary tree iff there exists a controller  $C$  satisfying  $\mathcal{R}$ :

- Using a monadic quantifier  $\exists Z$ : guess the successor of each node at even level (circle nodes, choosing as output either 0 or 1).
- $Z$  induces a subtree: take all successors of square nodes, and only one  $Z$ -successor of circle nodes.
- Using Büchi-Elgot-Trakhtenbrot's theorem, express that  $\mathcal{R}$  is satisfied along every infinite path in the subtree induced by  $Z$ .

If MSO formula  $\varphi$  is satisfiable, then it has a regular tree model: tree that is the unfolding of a finite automaton. □

# CHURCH'S PROBLEM AND LOGIC

## EXAMPLE

*The output has infinitely many 1's if the input has infinitely many 1's.*

$\exists X_0 \exists X_1 \exists Z :$

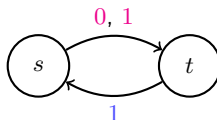
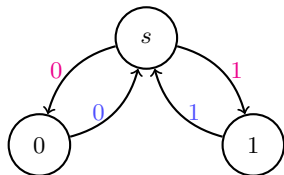
$X_0 =$  even level nodes,  $X_1 =$  odd level nodes,

$\text{root} \in Z$ ,

$\forall x \in Z$  (if  $x$  on **odd** level, then both children in  $Z$ ),

$\forall x \in Z$  (if  $x$  on **even** level, then exactly one child in  $Z$ ),

$\forall P \subseteq Z$  (if  $P$  is infinite path starting at root with infinitely many right edges from **odd** nodes, then it has infinitely many right edges from **even** nodes)



# CHURCH'S PROBLEM AND GAMES

## GRAPH GAMES

- Game arena: graph  $G = (V, E)$  with vertex set  $V$ , edge set  $E$ .
- Two players  $P_0$  (system) et  $P_1$  (environment). The set of vertices is partitioned into two disjoint subsets:  $V_0$  belongs to  $P_0$  and  $V_1$  to  $P_1$ .
- Play = path in the graph  $G$ . Owner of the current vertex chooses the outgoing edge.
- Winning condition = set Win of plays in  $G$ .  
Parity game: priorities  $p : V \rightarrow \{0, \dots, d\}$ . Win consists of all paths where the maximal priority visited infinitely often is even.
- Strategies:  $\sigma_0 : V^*V_0 \rightarrow V$ ,  $\sigma_1 : V^*V_1 \rightarrow V$ .
- Strategy  $\sigma_0$  is winning for  $P_0$  from  $v \in V$  if every play from  $v$  that is consistent with  $\sigma_0$  belongs to Win.
- Vertex  $v \in V$  is winning for  $P_0$  if  $P_0$  has a winning strategy from  $v$ .
- $W_0$  = set of winning vertices of  $P_0$  ( $P_0$ 's winning region). Symmetric:  $W_1$  for  $P_1$ .

## GAME SOLUTION

Solving a game means computing the winning regions  $W_0$ ,  $W_1$  and corresponding winning strategies.

# CHURCH'S PROBLEM AND GAMES

## GAME SOLUTION

Solving a game means computing the winning regions  $W_0$ ,  $W_1$  and corresponding winning strategies.

## STRATEGIES

“Ideal” strategies are

- positional (= memoryless)

$$\sigma_0 : V_0 \rightarrow V, \quad \sigma_1 : V_1 \rightarrow V,$$

- or finite-memory

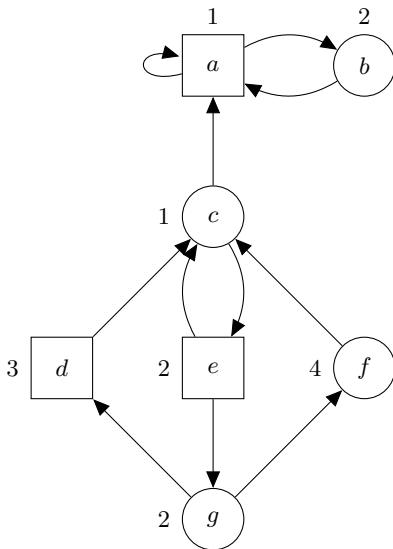
$$\sigma_0 : (V_0 \times M) \rightarrow V, \quad \sigma_1 : (V_1 \times M) \rightarrow V,$$

for some finite set  $M$  (with suitable update function).

## DETERMINED GAMES

A (graph) game is determined if  $V = W_0 \cup W_1$  (this actually partitions  $V$  if the game is zero-sum).

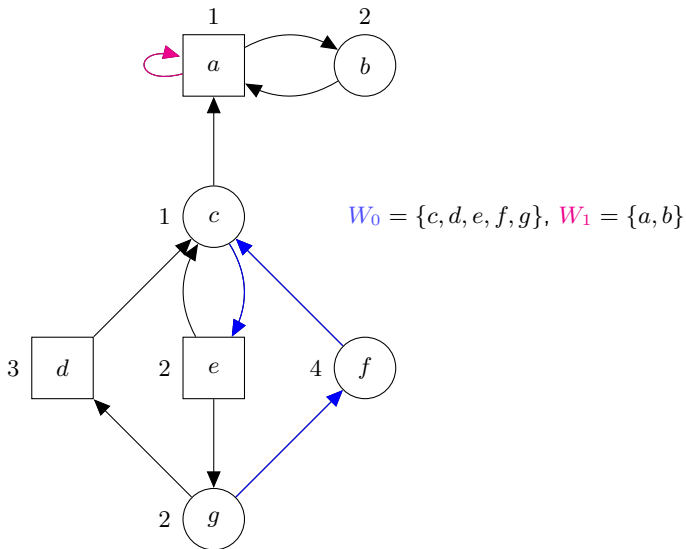
## EXAMPLE



Plays won by  $P_0$ :  $ababa \dots, cegfceg f \dots, cece \dots$

Plays won by  $P_1$ :  $aa \dots, cegdcegd \dots$

# EXAMPLE



Plays won by  $P_0$ :  $ababa \dots, cegfceg f \dots, cece \dots$

Plays won by  $P_1$ :  $aa \dots, cegdcegd \dots$



## CHURCH'S PROBLEM AS GRAPH GAME (McNAUGHTON'66, BÜCHI-LANDWEBER'67)

Recall: specification  $\mathcal{R} \subseteq (\{0, 1\}\{0, 1\})^\omega$  described as (non-deterministic) Büchi automaton.

- McNaughton's theorem: non-deterministic Büchi automaton for  $\mathcal{R}$  can be transformed into a **deterministic** parity automaton over  $\Sigma = \{0, 1, 0, 1\}$ :

$$\mathcal{A}_{\mathcal{R}} = \langle S, \Sigma, s_0, (\xrightarrow{a})_{a \in \Sigma}, \ell \rangle, \quad \mathcal{R} = L(\mathcal{A}_{\mathcal{R}})$$

- Wlog state set  $S$  partitioned into  $S = S_{\bigcirc} \cup S_{\square}$ :  
from  $S_{\bigcirc}$  only transitions with  $\{0, 1\}$ , from  $S_{\square}$  only transitions with  $\{0, 1\}$ .  
Initial state  $s_0 \in S_{\bigcirc}$ .
- Player  $P_0$  owns  $V_0 = S_{\bigcirc}$ , player  $P_1$  owns  $V_1 = S_{\square}$ .
- Play  $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots =$  maximal path in  $\mathcal{A}_{\mathcal{R}}$ .
- A play is winning for  $P_0$  iff the path satisfies the parity condition  $\implies$  **parity game!**

## PARITY GAMES: REFERENCES

- An excellent survey with a simplified proof (over countable graphs):  
W. Zielonka, "Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees".  
Theor. Comp. Sci. 1998(200):135-183.
- References: Büchi-Landweber (1969), Rabin (1969), Gurevich-Harrington (1982), Muchnik (1984), Emerson-Jutla (1988), Mostowski (1991), McNaughton (1993), Muller-Schupp (1995).

## PARITY GAMES: COMPLEXITY

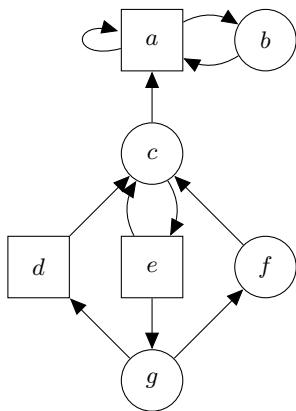
- Parity games are determined, and winning strategies are positional (memoryless). On finite graphs, deciding the winner is in  $\text{NP} \cap \text{co-NP}$ .
- Still open: are parity games in PTime? It is so for restricted classes of graphs, like bounded tree-width, bounded clique-width graphs.
- Classical algorithm (Zielonka):  $O(n^{d+O(1)})$ .
- Recent papers:  $O(n^{\log(d)+O(1)})$  [Calude-Jain-Khoussainov-Li-Stephan 2016, Jurdzinski-Lazic 2017].

# SIMPLE GAMES ON FINITE GRAPHS

## REACHABILITY GAMES

A reachability game  $G = (V = V_0 \cup V_1, E)$  has winning condition described by a set  $F \subseteq S$  of final nodes. A path is winning for  $P_0$  if it visits  $F$  at least once.

## EXAMPLE



$$F = \{b, d\}$$

$$W_0 = \{b, d, g\}$$

$$F = \{a, f\}$$

$$W_0 = V$$

# REACHABILITY GAMES

## ATTRACTORS

$$\begin{aligned}\text{Attr}_0^0(F) &= F \\ \text{Attr}_0^{n+1}(F) &= \text{Attr}_0^n(F) \cup \\ &\quad \{v \in V_0 \mid \exists w \in \text{Attr}_0^n(F), (v, w) \in E\} \cup \\ &\quad \{v \in V_1 \mid \forall w \text{ t.q. } (v, w) \in E : w \in \text{Attr}_0^n(F)\} \\ \text{Attr}_0^0(F) &\subseteq \text{Attr}_0^1(F) \subseteq \dots \subseteq \text{Attr}_0^{|V|}(F)\end{aligned}$$

$\text{Attr}_0^i(F)$  is the set of vertices from which  $P_0$  can reach  $F$  after at most  $i$  moves.

$W_0 = \text{Attr}_0^{|V|}(F)$  is the **winning region** of  $P_0$  (smallest fixpoint), and

$W_1 = V \setminus \text{Attr}_0^{|V|}(F)$  is the **winning region** of  $P_1$ .

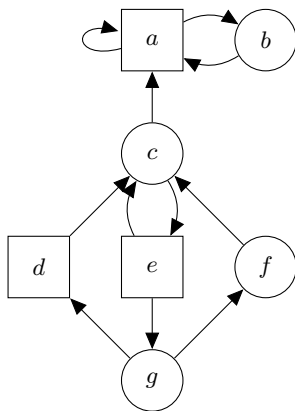
## STRATEGIES

Reachability games are determined and have **positional** winning strategies: attractor strategy for  $P_0$  and trap strategy for  $P_1$ .

Both the winning regions and winning strategies can be computed in polynomial time.

# REACHABILITY GAMES

## EXAMPLE



$$F = \{a, f\}$$

$$\text{Attr}_0^0(F) = \{a, f\}$$

$$\text{Attr}_0^1(F) = \{a, f, b, c, g\}$$

$$\text{Attr}_0^2(F) = \{a, f, b, c, g, e, d\}$$

$$\sigma_0(b) = \sigma_0(c) = a, \sigma_0(g) = f,$$

$$\sigma_0(f) = c$$

A Büchi game  $G = (V = V_0 \cup V_1, E)$  has winning condition described by a set  $F \subseteq S$  of final nodes. A path is winning for  $P_0$  if it visits  $F$  infinitely often.

## ALGORITHM

- $\text{Attr}_0^+(F)$ : set of states from which  $P_0$  can reach  $F$  in **at least** one move. We can compute  $\text{Attr}_0^+(F)$ , as well as a positional strategy, in polynomial time.
- $X^{(i)}$ : set of states from which  $P_0$  can go through  $F$  **at least**  $i$  times (without counting the starting state).

$$\begin{aligned} X^{(0)} &= V, & X^{(i+1)} &= \text{Attr}_0^+(X^{(i)} \cap F) \\ W_0 &= \bigcap_{i \geq 1} X^{(i)} \end{aligned}$$

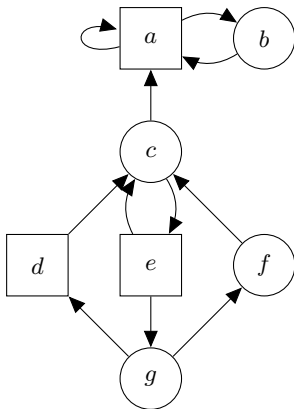
$X^{(0)} \supseteq X^{(1)} \supseteq \dots$ : some  $k$  with  $X^{(k)} = X^{(k+1)} =: W_0$ .  
 $W_0$  is a largest fixpoint and  $W_0 = \text{Attr}_0^+(W_0 \cap F)$ .

## STRATEGIES

Büchi games are determined and have **positional** winning strategies:  $\text{Attr}_0^+$  strategy for  $P_0$  and trap strategy for  $P_1$  (positional).

Both winning regions / strategies can be computed in PTime.

## EXAMPLE



$$F = \{b, d, e\}$$

$$X^{(1)} = \text{Attr}_0^+(F) = \{a, c, d, e, f, g\}$$

$$X^{(2)} = \text{Attr}_0^+(\{d, e\}) = \\ \{c, d, e, f, g\}$$

$$X^{(3)} = X^{(2)}$$

$$\sigma_0(c) = e, \sigma_0(f) = c, \sigma_0(g) = d$$

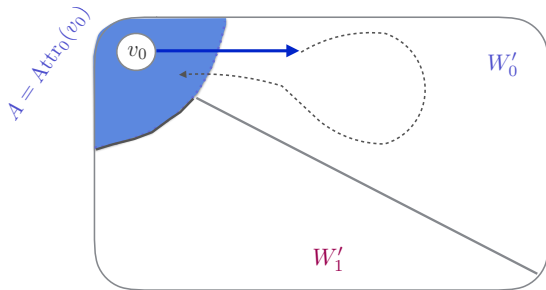
## ZIELONKA/MCNAUGHTON ALGORITHM

Inductive solution:

- Let  $G = (V_0, V_1, E)$ ,  $p : V \rightarrow \{0, \dots, k\}$  be a parity game.
- Induction over  $|V|$ . Suppose wlog. that the maximal priority  $k$  is even and let  $v_0$  be some state with  $p(v) = k$ .
- Let  $A_0 = \text{Attr}_0(v_0)$ , so  $V \setminus A_0$  is a subgame for  $P_1$ . By induction,  $V \setminus A_0$  is partitioned into  $W'_0$  and  $W'_1$ , and  $P_i$  has a positional winning strategy on  $W'_i$ .
- 2 cases (and exactly one applies):
  - ① From  $v_0$ , player  $P_0$  can ensure to be in  $A_0 \cup W'_0$  at the next step.
  - ② From  $v_0$ , player  $P_1$  can ensure to be in  $W'_1$  at the next step.
- Case 1:  $W_0 = W'_0 \cup A_0$ ,  $W_1 = W'_1$ .
- Case 2:  $v_0 \in \text{Attr}_1(W'_1)$ . Let  $A_1 = \text{Attr}_1(W'_1 \cup \{v_0\})$ .  
 $V \setminus A_1$  is a subgame for  $P_0$ . Let  $W''_i$  be the winning region of  $P_i$  in this subgame (with positional strategies).  
 $W_0 = W''_0$  and  $W_1 = W''_1 \cup A_1$ .

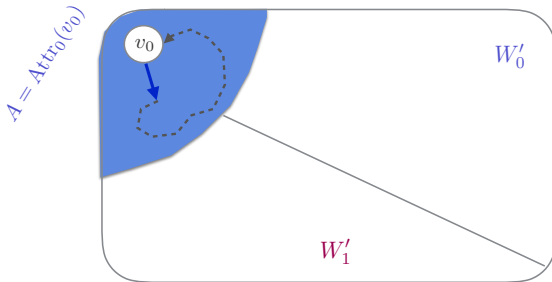


From  $v_0$ , player  $P_0$  can enforce to stay in  $A = A_0$  or go to  $W'_0$ :

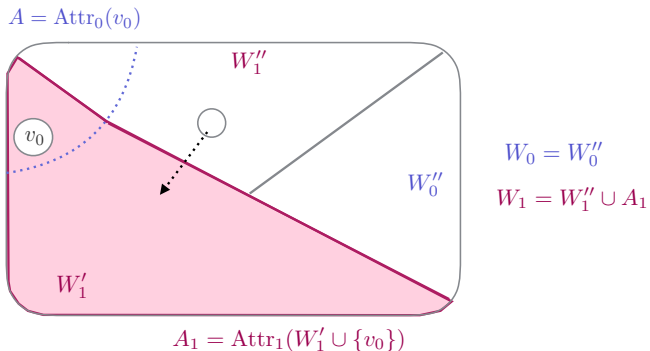


$$W_0 = W'_0 \cup A$$

$$W_1 = W'_1$$



From  $v_0$ , player  $P_1$  can enforce to go to  $W_1'$ :



$W_0'', W_1''$  are winning regions in game  $V \setminus A_1$

## COMPLEXITY

$n = |V|$ ,  $m = |E|$ ,  $k = \text{number of priorities}$

- At each recursive call, there is (at least) one vertex of highest priority less.
- Running time:

$$T_{n,m}(k) \leq n \cdot (T_{n,m}(k-1) + O(m+n)) \implies T_{n,m}(k) \leq n^{k+O(1)}$$

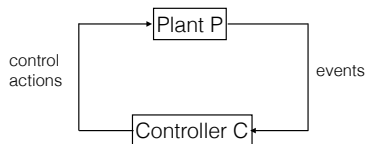
# SUPERVISORY CONTROL: RAMADGE/WONHAM

## SETTING

We are given

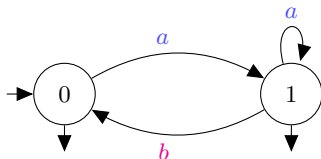
- a “plant”  $P$  (deterministic finite automaton),
- a partition of the set  $\Sigma$  of actions into **controllable** actions from  $\Sigma^{sys}$  and **uncontrollable** actions from  $\Sigma^{env}$ ,
- a (regular) specification  $Spec$ .

Compute controller (supervisor)  $C$  that restricts only **controllable** actions s.t.  $Spec$  is satisfied.



## EXAMPLE

Plant  $P$  with  $\Sigma^{env} = \{b\}$ :



$Spec$ : at most 2 consecutive  $a$ 's

- Controller: observes the dynamics of the plant. Cannot restrict **uncontrollable** actions:

$$C : \text{Path}(P) \rightarrow 2^{\Sigma} \text{ s.t. } \Sigma^{env} \subseteq C(w) \text{ for all } w$$

- Controlled plant:  $P \times C$  must satisfy  $Spec$ .
- Examples:  $C_1$  counts  $a$  up to 2 and  $P \times C_1 = ((a + aa)b)^*(\epsilon + a + aa)$ .  
Or  $C_2$  never allows  $a$ , so  $P \times C_2 = \emptyset$ .

## $P \times C$ (SYNCHRONIZED PRODUCT)

$$P = \langle Q, \Sigma, \longrightarrow_P, q_0, Q \rangle, C : \text{Path}(P) \rightarrow 2^{\Sigma}$$

$$\begin{aligned}
 P \times C &= \langle Q \times \Sigma^*, \Sigma, \longrightarrow, (q_0, \epsilon), F \times \Sigma^* \rangle \\
 (q, w) &\xrightarrow{a} (q', wa) \quad \text{if } q \xrightarrow{a}_P q', a \in C(w)
 \end{aligned}$$

## SAFETY SPECIFICATIONS

Given:

- A finite-state automaton (*plant*)  $P = \langle Q_P, \Sigma, \longrightarrow_P, q_{0,P}, Q_P \rangle$  over alphabet  $\Sigma$  partitioned into controllable actions  $\Sigma^{sys}$  and uncontrollable actions  $\Sigma^{env}$ .
- A finite-state automaton (specification)  $S = \langle Q_S, \Sigma, \longrightarrow_S, q_{0,S}, Q_S \rangle$ , *all states are final* (safety).

Compute  $C$  such that:

- $P \times C \subseteq S$ ,
- $w \in C$  and  $a \in \Sigma^{env}$  implies  $wa \in C$ ,
- Other possible requirements:  $C$  is non-blocking, maximal permissive, . . . .

## SAFETY SPECIFICATIONS

Given:

- A finite-state automaton (*plant*)  $P = \langle Q_P, \Sigma, \longrightarrow_P, q_{0,P}, Q_P \rangle$  over alphabet  $\Sigma$  partitioned into controllable actions  $\Sigma^{sys}$  and uncontrollable actions  $\Sigma^{env}$ .
- A finite-state automaton (specification)  $S = \langle Q_S, \Sigma, \longrightarrow_S, q_{0,S}, Q_S \rangle$ , *all states are final* (safety).

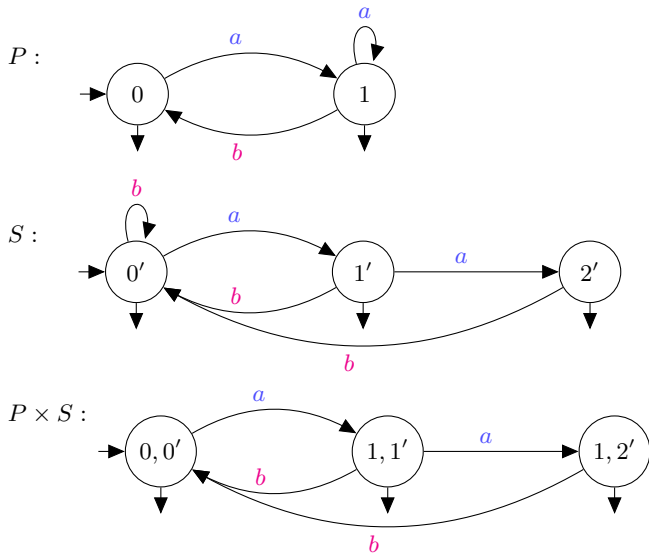
Compute  $C$  such that:

- $P \times C \subseteq S$ ,
- $w \in C$  and  $a \in \Sigma^{env}$  implies  $wa \in C$ ,
- Other possible requirements:  $C$  is non-blocking, maximal permissive, . . . .

## SOLUTION

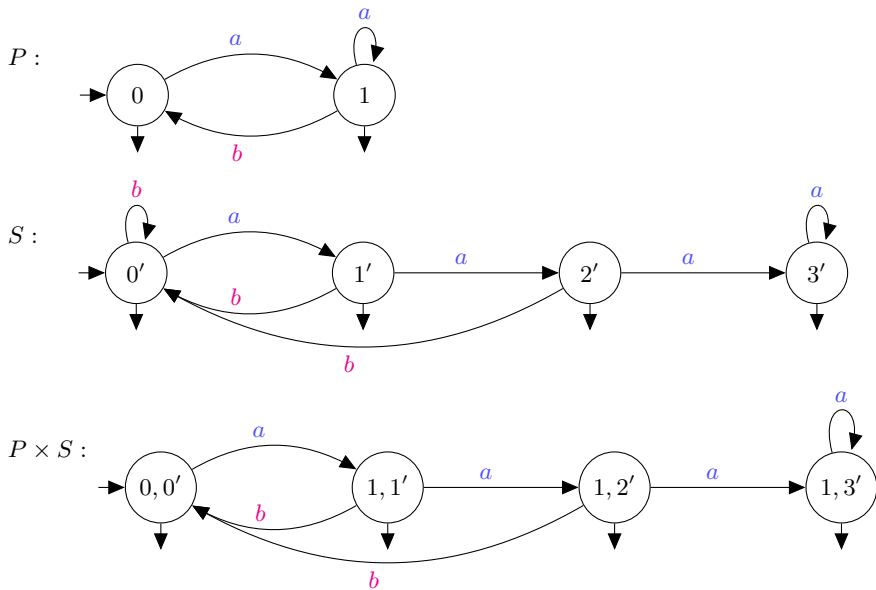
- Build the product  $P \times S$ .
- Remove all states  $(q_P, q_S)$  such that for some  $w \in (\Sigma^{env})^*$ :  $q \xrightarrow{w}_P \cdot$  is defined, but  $q_S \xrightarrow{w}_S \cdot$  is undefined.
- Add self-loops with  $\Sigma^{env}$ , if necessary. The output is the most permissive controller  $C$ .

# EXAMPLE 1

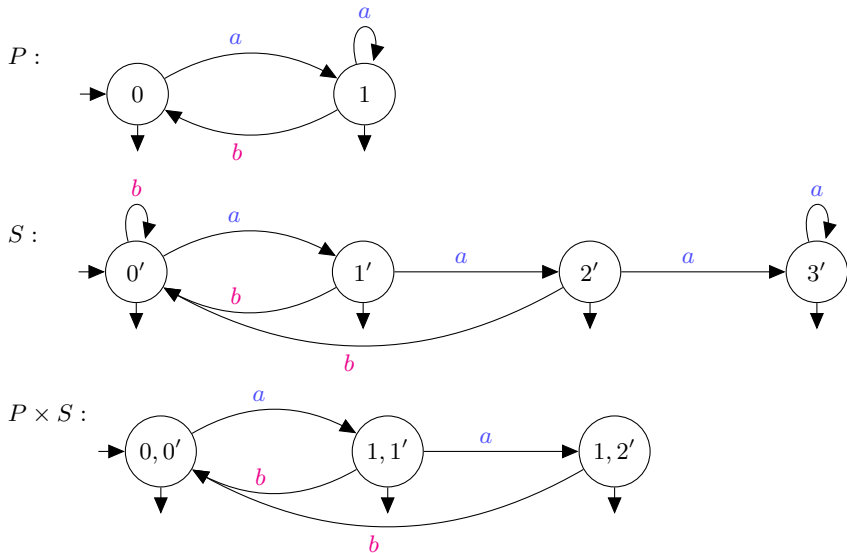




## EXAMPLE 2



## EXAMPLE 2



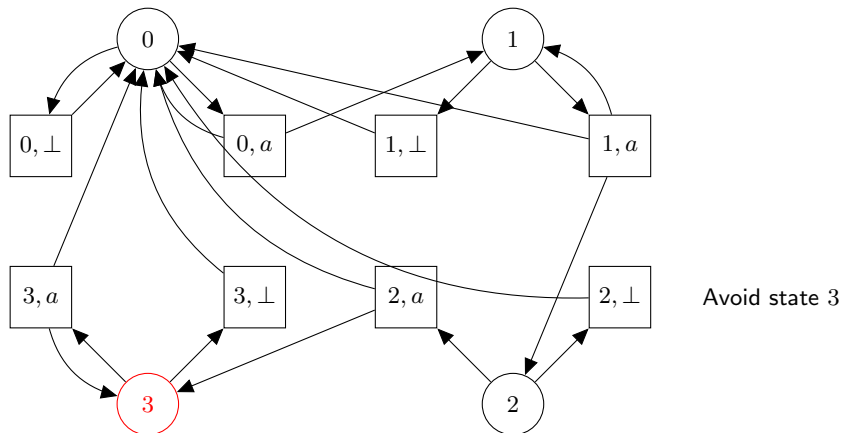
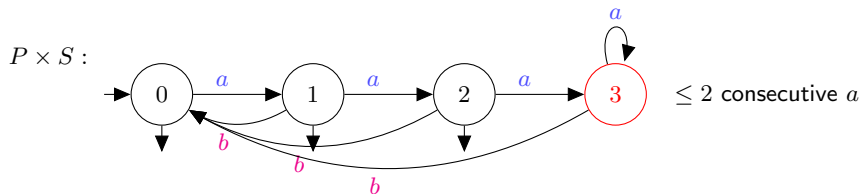
# FROM SUPERVISORY CONTROL TO GAMES

Given: plant  $P = \langle Q, \Sigma, \longrightarrow, q_0, Q \rangle$  over alphabet  $\Sigma = \Sigma^{sys} \dot{\cup} \Sigma^{env}$ .

Game arena

- Node set  $V_0 = Q$ ,  
 $V_1 = \{(q, a) \mid a \in \Sigma^{sys} \text{ and } q \xrightarrow{a} \text{ is defined}\} \cup Q \times \{\perp\}$ .
- Edge set:
  - $q \longrightarrow (q, a)$  if  $q \xrightarrow{a}$  is defined,
  - $q \longrightarrow (q, \perp)$ ,
  - $(q, a) \longrightarrow q'$  if either  $q \xrightarrow{a} q'$ , or  $q \xrightarrow{b} q'$  for some  $b \in \Sigma^{env}$ ,
  - $(q, \perp) \longrightarrow q'$  if  $q \xrightarrow{b} q'$  for some  $b \in \Sigma^{env}$ .
- Winning condition: specification  $S$ .

# FROM SUPERVISORY CONTROL TO GAMES

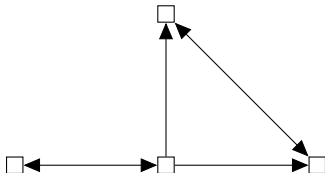


## II. Distributed synthesis

# DISTRIBUTED SYSTEMS

## MODELS

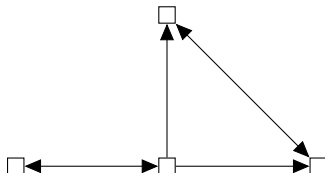
Processes with links. A process is e.g. finite-state automaton.



# DISTRIBUTED SYSTEMS

## MODELS

Processes with links. A process is e.g. finite-state automaton.



## LINKS AS CHANNELS

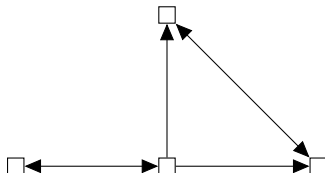
Links are channels and processes have send and receive operations:  
communicating automata, message sequence charts.

Turing powerful.

# DISTRIBUTED SYSTEMS

## MODELS

Processes with links. A process is e.g. finite-state automaton.



## LINKS AS CHANNELS

Links are channels and processes have send and receive operations:  
communicating automata, message sequence charts.

Turing powerful.

## LINKS AS SYNCHRONIZATION

Links are shared variables and processes can synchronize (rendez-vous):  
distributed automata, Mazurkiewicz traces, event structures.

Regular languages.



# SOME HISTORY

## SYNTHESIZING CLOSED, GLOBAL SYSTEMS

- Clarke/Emerson 1982, *"Using branching time temporal logic to synthesize synchronization skeletons"*: *We present a method of constructing concurrent programs in which the synchronization skeleton of the program is automatically synthesized from a (branching time) temporal logic specification.*
- Manna/Wolper 1984, *"Synthesis of communicating processes from temporal logic"*: *In this paper, we apply Propositional Temporal Logic (PTL) to the specification and synthesis of the synchronization part of communicating processes. To specify a process, we give a PTL formula that describes its sequence of communications.*

In both settings the systems are **closed** (no environment). "Concurrent" programs means here: **product** transition system. The synthesized programs are not guaranteed to be **distributedly** implementable.

# SOME HISTORY

## SYNTHESIZING OPEN, SEQUENTIAL SYSTEMS

- Pnueli/Rosner 1989, **"On the synthesis of a reactive module"**: We consider the synthesis of a reactive module with input  $x$  and output  $y$ , which is specified by the linear temporal formula  $\varphi(x, y)$ .
- Kupferman/Vardi 1999, **"Church's problem revisited"**: We consider linear and branching settings with complete and incomplete information. [...] In particular, we prove that independently of the presence of incomplete information, the synthesis problems for CTL and CTL\* are complete for EXPTIME and 2EXPTIME, respectively.

# SOME HISTORY

## SYNTHESIZING OPEN, SEQUENTIAL SYSTEMS

- Pnueli/Rosner 1989, *"On the synthesis of a reactive module"*: We consider the synthesis of a reactive module with input  $x$  and output  $y$ , which is specified by the linear temporal formula  $\varphi(x, y)$ .
- Kupferman/Vardi 1999, *"Church's problem revisited"*: We consider linear and branching settings with complete and incomplete information. [...] In particular, we prove that independently of the presence of incomplete information, the synthesis problems for CTL and CTL\* are complete for EXPTIME and 2EXPTIME, respectively.

## SYNTHESIZING OPEN, DISTRIBUTED (?) SYSTEMS

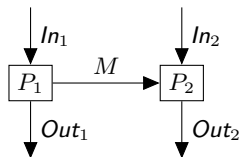
Pnueli/Rosner 1990, *"Distributed reactive systems are hard to synthesize"*. The limitation (of [CE82,MW84]) is that all the synthesis algorithms produce a program for a single module [...]. This is particularly embarrassing in cases that the problem we set out to solve is meaningful only in a distributed context, such as the mutual exclusion problem [...]. The somewhat ad-hoc solution [...] is to use first the general algorithm to produce a single module program, and then to decompose this program into a set of programs, one for each distributed component of the system.

NB: the last task is **undecidable**.

Distributed synthesis:  
open, synchronous systems  
(Pnueli & Rossner)

## SYNTHESIS SETTING

- **Synchronous** processes (global clock), exchange finite information.



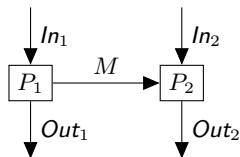
specification  $\mathcal{R} \subseteq A^\omega$

$$A = In_1 \times In_2 \times M \times Out_1 \times Out_2$$

- **Problem:** given an architecture over  $n$  processes and a regular language  $\mathcal{R} \subseteq A^\omega$ , decide if there exist devices  $P_1, \dots, P_n$  such that all executions are in  $\mathcal{R}$ .

## SYNTHESIS SETTING

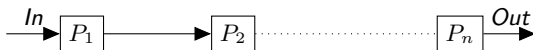
- **Synchronous** processes (global clock), exchange finite information.



specification  $\mathcal{R} \subseteq A^\omega$

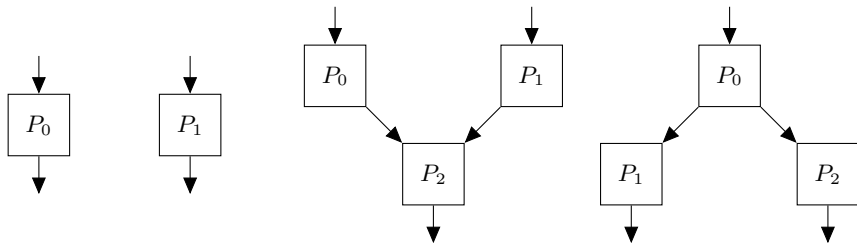
$$A = In_1 \times In_2 \times M \times Out_1 \times Out_2$$

- **Problem:** given an architecture over  $n$  processes and a regular language  $\mathcal{R} \subseteq A^\omega$ , decide if there exist devices  $P_1, \dots, P_n$  such that all executions are in  $\mathcal{R}$ .
- Synthesis problem is **decidable** iff the architecture is a pipeline.  
Complexity: **non-elementary**.



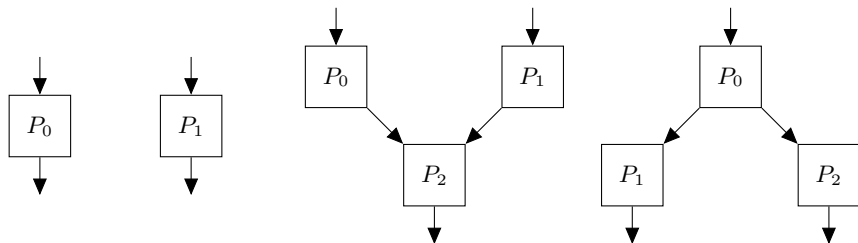
# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## UNDECIDABLE ARCHITECTURES



# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## UNDECIDABLE ARCHITECTURES



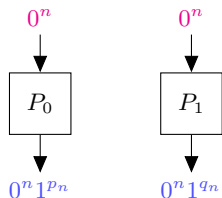
## UNDECIDABILITY: REASONS

- Processes have **different knowledge** about the moves of the (global) environment. Left example:  $P_0$ ,  $P_1$  have incomparable information. **Information fork** (Finkbeiner, Schewe).
- No compatibility** required between architecture and specification.



# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

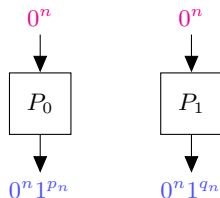
## UNDECIDABILITY



- On input  $0^n$  the specification will force  $P_0, P_1$  to output  $0^n 1^n$ .
- How can we enforce this with a **regular** specification  $S$ ?

# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## UNDECIDABILITY

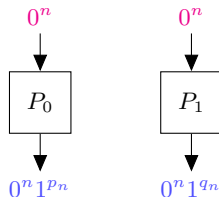


- On input  $0^n$  the specification will force  $P_0, P_1$  to output  $0^n 1^n$ .
- How can we enforce this with a **regular** specification  $S$ ?

Trick: using synchronicity,  $S$  can relate the outputs of  $P_0$  and  $P_1$

# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## UNDECIDABILITY



- On input  $0^n$  the specification will force  $P_0, P_1$  to output  $0^n 1^n$ .
- How can we enforce this with a **regular** specification  $S$ ?

Trick: using synchronicity,  $S$  can relate the outputs of  $P_0$  and  $P_1$

$$S = S_1 \cup S_2$$

$$S_1 = \{(0^n, 0^n 1^p, 0^n, 0^n 1^q) : n \geq 0, p = q\}$$

$$S_2 = \{(0^n, 0^n 1^p, 0^{n+1}, 0^{n+1} 1^q) : n \geq 0, q = p + 1\}$$

If in addition,  $P_0$  and  $P_1$  must output  $p_0 = q_0 = 0$ , we get  $p_n = q_n = n$  for all  $n \geq 0$ .

# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## INFORMATION FORK (FINKBEINER/SCHWE)

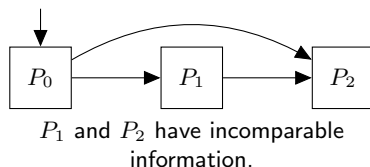
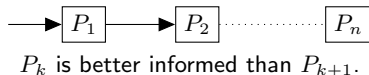
- Process  $P$  is (at least) as well **informed** as process  $P'$  if the environment cannot transmit information to  $P'$  without  $P$  knowing about it.
- **Information fork**: two processes with incomparable information.

# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## INFORMATION FORK (FINKBEINER/SCHWE)

- Process  $P$  is (at least) as well **informed** as process  $P'$  if the environment cannot transmit information to  $P'$  without  $P$  knowing about it.
- **Information fork**: two processes with incomparable information.

## EXAMPLE

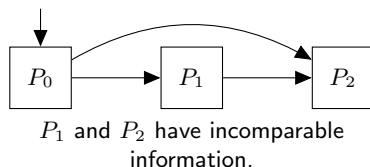
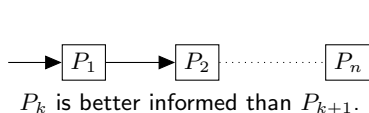


# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## INFORMATION FORK (FINKBEINER/SCHWE)

- Process  $P$  is (at least) as well **informed** as process  $P'$  if the environment cannot transmit information to  $P'$  without  $P$  knowing about it.
- **Information fork**: two processes with incomparable information.

## EXAMPLE



## FINKBEINER/SCHWE 2005

Synchronous synthesis is decidable iff there is no information fork.

## DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

LOCAL SPECIFICATIONS (MADHUSUDAN/THIAGARAJAN 2001)

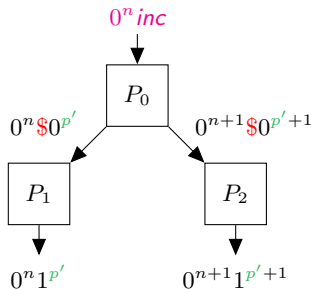
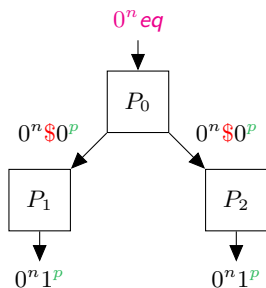
Undecidability for synchronous case due to global specifications? Not only.

# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## LOCAL SPECIFICATIONS (MADHUSUDAN/THIAGARAJAN 2001)

Undecidability for synchronous case due to global specifications? **Not only.**

- Same as before,  $P_0$  and  $P_1$  should output  $0^n 1^{p_n}$  and  $0^n 1^{q_n}$ , with  $p_n = q_n = n$ .
- “Checking”  $p_n = q_n$  and  $q_{n+1} = p_n + 1$  is now done by the choice of the environment:



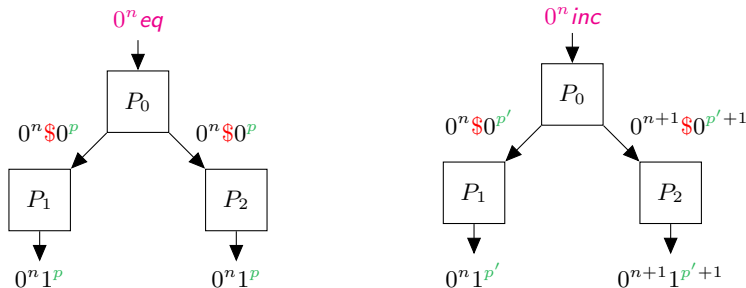


# DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

## LOCAL SPECIFICATIONS (MADHUSUDAN/THIAGARAJAN 2001)

Undecidability for synchronous case due to **global** specifications? **Not only.**

- Same as before,  $P_0$  and  $P_1$  should output  $0^n 1^{p_n}$  and  $0^n 1^{q_n}$ , with  $p_n = q_n = n$ .
- “Checking”  $p_n = q_n$  and  $q_{n+1} = p_n + 1$  is now done by the choice of the environment:



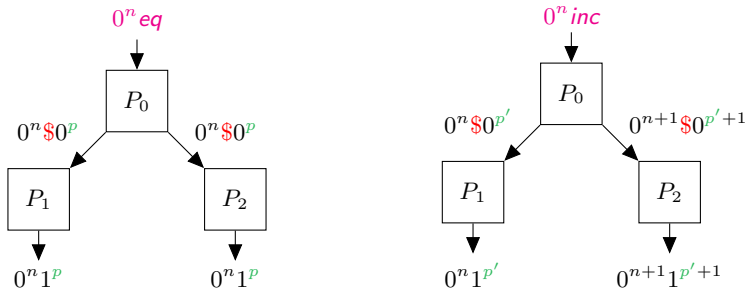
Why is  $P_0$  forced to output  $p = p'$ ?

## DISTRIBUTED SYNTHESIS: SYNCHRONOUS CASE

### LOCAL SPECIFICATIONS (MADHUSUDAN/THIAGARAJAN 2001)

Undecidability for synchronous case due to global specifications? **Not only.**

- Same as before,  $P_0$  and  $P_1$  should output  $0^n 1^{p_n}$  and  $0^n 1^{q_n}$ , with  $p_n = q_n = n$ .
- “Checking”  $p_n = q_n$  and  $q_{n+1} = p_n + 1$  is now done by the choice of the environment:



Why is  $P_0$  forced to output  $p = p'$ ?

The specification  $\{(0^n \$0^p, 0^n 1^p) : n, p\}$  forces  $P_1$  to “accept” only one value of  $p$ , for given  $n$ .

# SYNCHRONOUS CASE: DECIDABILITY

PNUELI/ROSNER 1990

Synthesis is decidable on pipelines, with non-elementary complexity.



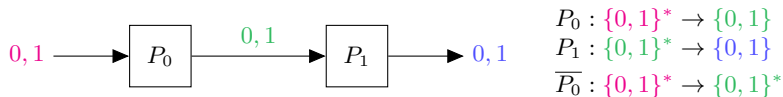
# SYNCHRONOUS CASE: DECIDABILITY

PNUELI/ROSNER 1990

Synthesis is decidable on pipelines, with non-elementary complexity.



PROOF IDEA



$$P_0 \circ P_1 : \{0,1\}^* \rightarrow \{0,1\} \quad P_0 \circ P_1(w) = P_1(\overline{P_0}(w))$$

If  $S$  is a regular tree language defining a set of functions  $\{0,1\}^* \rightarrow \{0,1\}$ , then there is a regular tree language  $S'$  defining a set of functions  $\{0,1\}^* \rightarrow \{0,1\}$  such that

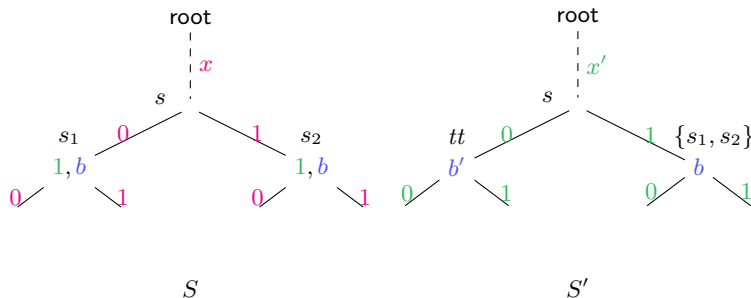
$$P_1 \in S' \quad \text{iff} \quad \exists P_0 : \{0,1\}^* \rightarrow \{0,1\} : P_0 \circ P_1 \in S$$

## PIPELINE: PROOF

### AUTOMATA CONSTRUCTION (KUPFERMAN/VARDI)

From a **non-deterministic** parity tree automaton accepting  $S$  one constructs an **alternating** parity tree automaton accepting  $S'$ .

Strategy tree: binary tree labelled by strategy outputs.



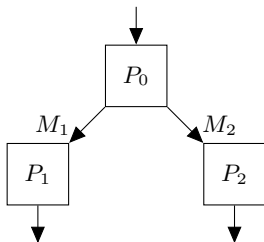
### III. Distributed control: asynchronous case

## SYNCHRONOUS/ASYNCHRONOUS

- Pnueli & Rosner model has **synchronous** communication: at each step all controllers make a transition.
- **Asynchronous** communication: each controller progresses at own speed.

## INFORMATION

In the Pnueli & Rosner model: controllers do **not exchange** information beyond the amount allowed by the specification.



Adding information to the messages sent by  $P_0$  to  $P_1$ ,  $P_2$  (beyond  $M_1$ ,  $M_2$ ) would make the synthesis problem decidable here.

## DISTRIBUTED AUTOMATA

- Finite set of processes  $\mathbb{P}$
- One finite-state automaton per process.
- **Distributed alphabet**  $\langle \Sigma, \text{dom} : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \emptyset) \rangle$

Each action  $a$  synchronizes all processes in  $\text{dom}(a)$ :

$$\text{Transition functions } \delta_a : \prod_{p \in \text{dom}(a)} S_p \rightarrow \prod_{p \in \text{dom}(a)} S_p$$



## DISTRIBUTED AUTOMATA

- Finite set of processes  $\mathbb{P}$
- One finite-state automaton per process.
- **Distributed alphabet**  $\langle \Sigma, \text{dom} : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \emptyset) \rangle$

Each action  $a$  synchronizes all processes in  $\text{dom}(a)$ :

Transition functions $\delta_a : \prod_{p \in \text{dom}(a)} S_p \rightarrow \prod_{p \in \text{dom}(a)} S_p$
---

→ **exchange of information** among  $\text{dom}(a)$  while executing  $a$

## EXAMPLE

### COMPARE-AND-SWAP

CAS ( $T$ : thread,  $x$ : variable; *old*, *new*: int).

If the value of  $x$  is *old*, then replace it by *new*; otherwise do nothing.

## EXAMPLE

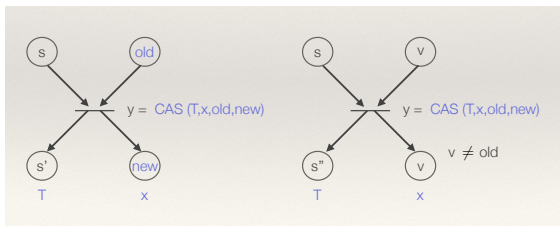
### COMPARE-AND-SWAP

CAS ( $T$ : thread,  $x$ : variable; *old*, *new*: int).

If the value of  $x$  is *old*, then replace it by *new*; otherwise do nothing.

### MULTI-THREADED PROGRAMS AS DISTRIBUTED AUTOMATA

One process per thread  $T$  and per shared variable  $x$ .



**Exchange of information:**

in state  $s'$  we have  $y = i$ ; in state  $s''$  we have  $y = j$ .

# DISTRIBUTED AUTOMATA

## THE LANGUAGE OF THE AUTOMATON

The (regular) language of the product automaton.

# DISTRIBUTED AUTOMATA

## THE LANGUAGE OF THE AUTOMATON

The (regular) language of the product automaton.

$$(s_p)_{p \in \mathbb{P}} \xrightarrow{a} (s'_p)_{p \in \mathbb{P}} \text{ if}$$

- $(s_p)_{p \in \text{dom}(a)} \rightarrow_a (s'_p)_{p \in \text{dom}(a)}$ , and
- $s'_q = s_q$  for  $q \notin \text{dom}(a)$ .

## THE LANGUAGE OF THE AUTOMATON

The (regular) language of the product automaton.

$(s_p)_{p \in \mathbb{P}} \xrightarrow{a} (s'_p)_{p \in \mathbb{P}}$  if

- $(s_p)_{p \in \text{dom}(a)} \rightarrow_a (s'_p)_{p \in \text{dom}(a)}$ , and
- $s'_q = s_q$  for  $q \notin \text{dom}(a)$ .

## REGULAR TRACE LANGUAGES

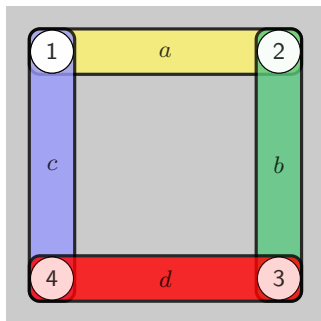
A regular, **trace-closed** language  $L \subseteq \Sigma^*$ :

$$uabv \in L \quad \text{iff} \quad ubav \in L,$$

for all  $u, v \in \Sigma^*$ ,  $\text{dom}(a) \cap \text{dom}(b) = \emptyset$ .

# TRACE LANGUAGES

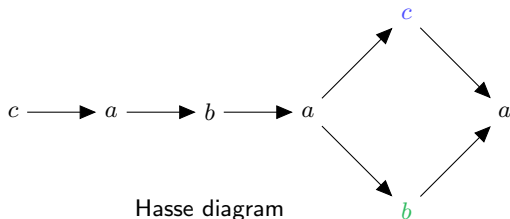
## MAZURKIEWICZ TRACES



### Distributed alphabet

$$\langle \Sigma, \text{dom} : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \emptyset) \rangle$$

- $\mathbb{P} = \{1, 2, 3, 4\}$
- $\Sigma = \{a, b, c, d\}$
- $\text{dom}(a) = \{1, 2\},$   
 $\text{dom}(b) = \{2, 3\}, \dots$



$$= [cabacba]$$
$$= [cababca]$$

trace = labelled partial  
order

# ZIELONKA'S THEOREM

[ZIELONKA 1989]

Construction of deterministic distributed automaton for every **regular** trace-closed language.

## CRUX

**Finite** gossiping (= knowledge exchange between processes).

## COMPLEXITY

From a deterministic finite-state automaton of size  $s$ , an equivalent distributed automaton on  $p$  processes with  $4p^4 \cdot s^{p^2}$  states can be constructed.

[Genest, Gimbert, M., Walukiewicz 2010]



# DISTRIBUTED AUTOMATA: NOT THAT EASY TO CONSTRUCT

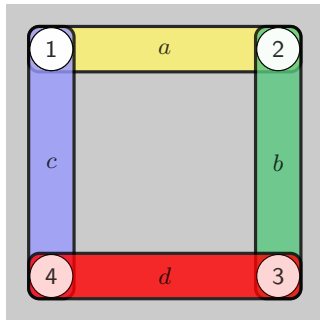
ZIELONKA (1987)

Each regular, **trace-closed** language can be recognized by a deterministic, distributed automaton.

# DISTRIBUTED AUTOMATA: NOT THAT EASY TO CONSTRUCT

ZIELONKA (1987)

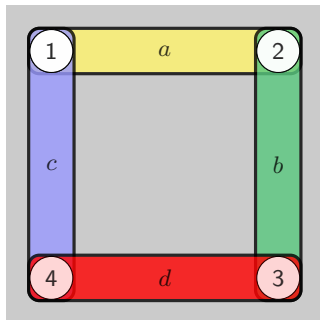
Each regular, **trace-closed** language can be recognized by a deterministic, distributed automaton.



# DISTRIBUTED AUTOMATA: NOT THAT EASY TO CONSTRUCT

ZIELONKA (1987)

Each regular, **trace-closed** language can be recognized by a deterministic, distributed automaton.

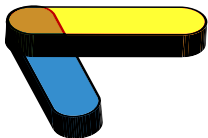


Build a distributed automaton for the trace language  $((b+c)(a+d))^*$ .  
A deterministic finite-state automaton needs only 3 states.



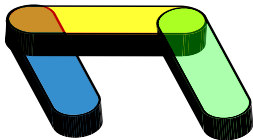
$c$

$$((b + c)(a + d))^*$$



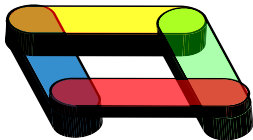
$ca$

$$((b + c)(a + d))^*$$



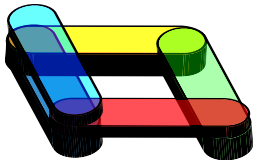
*cab*

$$((b + c)(a + d))^*$$



*cabd*

$$((b + c)(a + d))^*$$

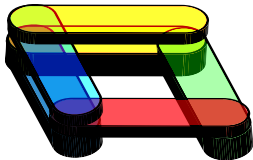


*cabdc*

$$((b + c)(a + d))^*$$



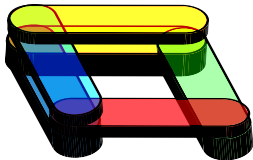
Good



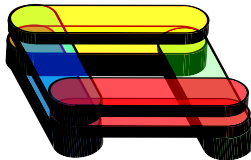
*cabdca*

$$((b + c)(a + d))^*$$

Good



Bad

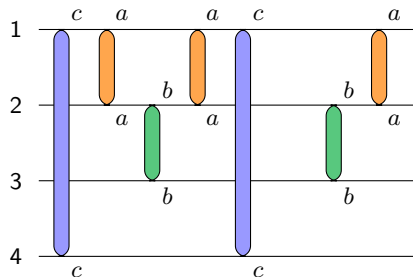
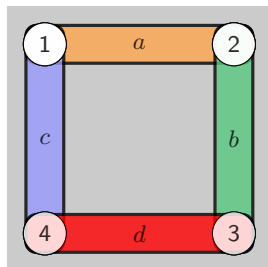


*cabdcad*

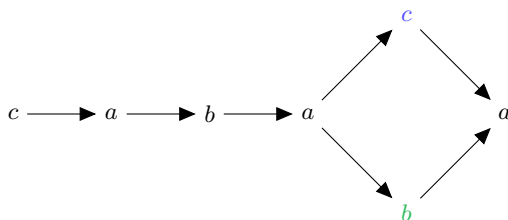
$$((b + c)(a + d))^*$$

# DISTRIBUTED AUTOMATA: NOT THAT EASY TO CONSTRUCT

EXAMPLE:  $((b + c)(a + d))^*$



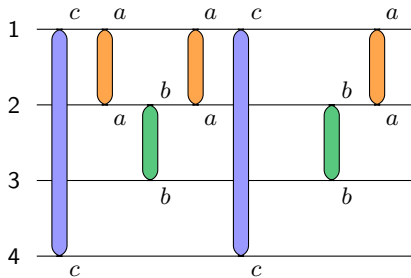
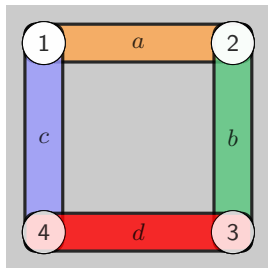
Bad



Last  $a$  sees bad  $b, c$ :  
they are in the view of  
processes 1 and 2.

# DISTRIBUTED AUTOMATA: NOT THAT EASY TO CONSTRUCT

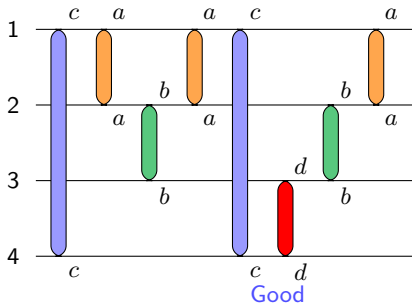
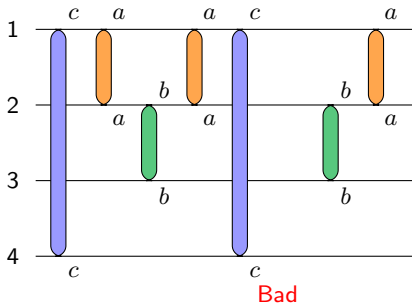
EXAMPLE:  $((b + c)(a + d))^*$



Bad

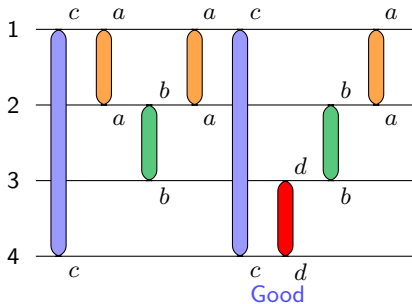
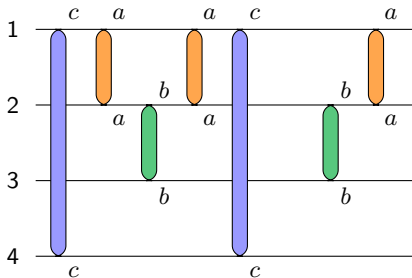
- First idea: each process remembers its *last action*.

EXAMPLE:  $((b + c)(a + d))^*$



- First idea: each process remembers its *last action*. When synchronizing, processes know about the *previous* action of the other process.
- How can process 1 know that between its first two  $a$ 's there was a  $b$ ? By communicating with process 2: the second  $a$  is only possible because process 2 did a  $b$  since the last  $a$ .

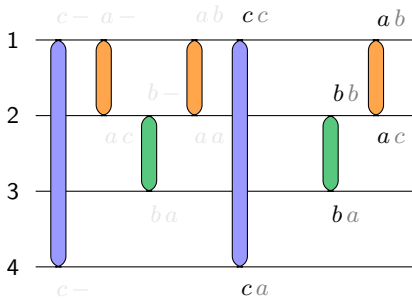
EXAMPLE:  $((b + c)(a + d))^*$



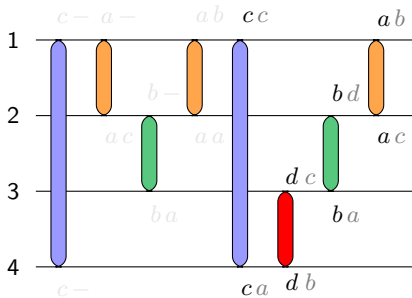
- First idea: each process remembers its *last action*. When synchronizing, processes know about the *previous* action of the other process.
- How can process 1 know that between its first two  $a$ 's there was a  $b$ ? By communicating with process 2: the second  $a$  is only possible because process 2 did a  $b$  since the last  $a$ .
- **Not sufficient:**  $d$  changes **Bad** into **Good**, but...

*In both cases, upon executing the last  $a$ , the last action of process 1 was  $c$ , and the last action of process 2 was  $b$ . Action  $d$  is "invisible".*

EXAMPLE:  $((b + c)(a + d))^*$



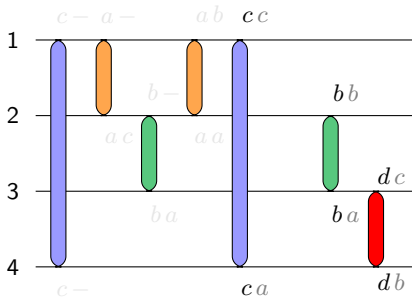
Bad



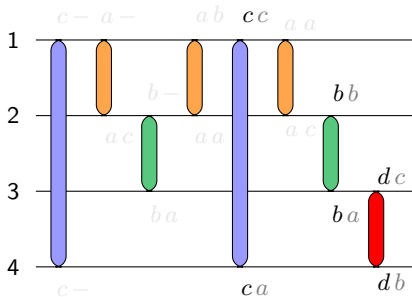
Good

- Second idea: each process  $p$  remembers its *last action* and, after a sync with process  $q$ , the previous action of  $q$ .

EXAMPLE:  $((b + c)(a + d))^*$



Bad

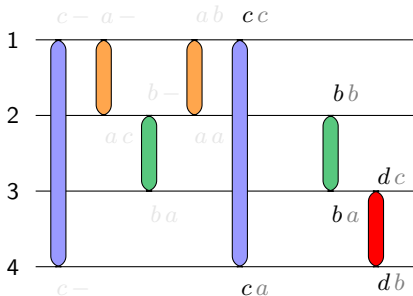


Good

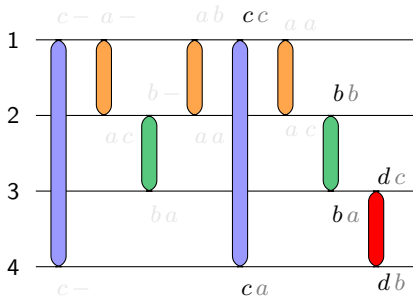
- Second idea: each process  $p$  remembers its *last action* and, after a sync with process  $q$ , the previous action of  $q$ .
- Does not work either: processes 1 and 2 do not have different information upon executing last  $d$ .



EXAMPLE:  $((b + c)(a + d))^*$



Bad



Good

- Second idea: each process  $p$  remembers its *last action* and, after a sync with process  $q$ , the previous action of  $q$ .
- Does not work either: processes 1 and 2 do not have different information upon executing last  $d$ .

... the solution here is actually as complicated as the general case:  
keep ordering of the actions of all processes...

# ZIELONKA'S THEOREM: ACYCLIC CASE

## ACYCLIC CASE

- Assume that  $|\text{dom}(a)| \leq 2$  for every  $a \in \Sigma$  and that the **communication graph**  $CG$  is acyclic:

**CG**: undirected graph with vertices = processes, edges between processes that share some action

Wlog. **CG** is a tree.

- Input: diamond finite-state automaton  $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$  recognizing a regular, trace-closed language  $L \subseteq \Sigma^*$ .
- We build a **polynomial-size** distributed automaton  $\mathcal{B} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma} \rangle$  with  $L(\mathcal{B}) = L(\mathcal{A})$ :

$$|S_p| = |S|^2, \quad \text{for each process } p$$

## DIAMOND PROPERTY

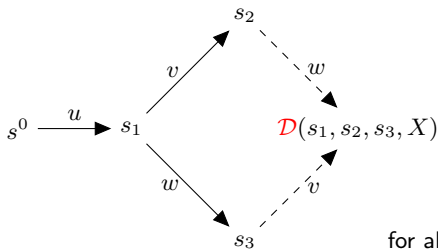
- A finite-state automaton has the **diamond property** if for every state  $s$ , and every  $a, b \in \Sigma$  such that  $\text{dom}(a) \cap \text{dom}(b) = \emptyset$ :

*if  $\delta(s, ab)$  is defined, then  $\delta(s, ba)$  is also defined and  $\delta(s, ab) = \delta(s, ba)$ .*

- The minimal automaton of a regular, trace-closed language has the diamond property.

## LEMMA

Given a diamond automaton  $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$ , there is a function  $\mathcal{D} : S^3 \times 2^{\mathbb{P}} \rightarrow S$  such that for every three states  $s_1, s_2, s_3 \in S$  and every  $X \subseteq \mathbb{P}$ , the state  $s = \mathcal{D}(s_1, s_2, s_3, X)$  satisfies:



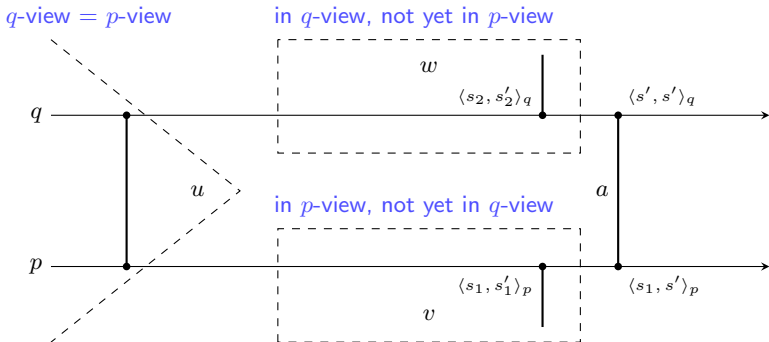
for all  $u, v, w, X$

# ZIELONKA'S THEOREM: ACYCLIC CASE

## CONSTRUCTION

Each process  $p \in \mathbb{P}$  stores a pair of states of  $\mathcal{A}$ : we write  $\langle s, s' \rangle_p$  to denote a pair stored by process  $p$ .

*Invariant:* the first state stored by process  $q$  is the state at which it synchronized the last time with its parent  $p$  in *CG*;  
the second state of  $q$  stores the state reached by the automaton  $\mathcal{A}$  on the current  $q$ -view.



# ZIELONKA'S THEOREM: ACYCLIC CASE

## CONSTRUCTION

- Starting state of  $p$ :  $(s_{in})_p = \langle s^0, s^0 \rangle_p$ .
- Transition function  $\delta_a$  for  $a \in \Sigma$ :
  - if  $\text{dom}(a) = \{p\}$ : easy, only local update

$$\delta_a(\langle s, s' \rangle_p) = \langle s, \Delta(s', a) \rangle_p$$

- if  $\text{dom}(a) = \{p, q\}$  and  $p$  is the parent of  $q$  in  $CG$ : apply diamond lemma to combine information stored by  $p, q$

$$\delta_a(\langle s_1, s'_1 \rangle_p, \langle s_2, s'_2 \rangle_q) = (\langle s_1, s' \rangle_p, \langle s' , s' \rangle_q),$$

where  $s' = \Delta(s, a)$ ,  $s = \mathcal{D}(s_2, s'_2, s'_1, X(q))$  and  $X(q) \subseteq \mathbb{P}$  is the subtree of  $CG$  rooted at  $q$ .

- Final states  $F \subseteq \prod_{p \in \mathbb{P}} S_p$ : apply diamond lemma to combine information up to the root process.

# Control problem for distributed automata

## CONTROL PROBLEM: STATEMENT

### RAMADGE & WONHAM

- Given: distributed automaton  $\mathcal{A}$  (“plant”) with two kinds of actions, **controllable** actions (or system actions,  $\Sigma^{sys}$ ) and **uncontrollable** actions (or environment actions,  $\Sigma^{env}$ ); and a specification  $S$ .
- Compute **local controllers**, one for each process. A local controller must allow every **uncontrollable** action.  
Local controllers exchange information as in a distributed automaton: so we look for a distributed automaton  $\mathcal{C}$  such that

$$\mathcal{A} \times \mathcal{C} \models S$$

The product  $\mathcal{A} \times \mathcal{C}$  is the usual synchronized product of automata, here **process-wise**.

## SUPERVISOR CONTROL PROBLEM FOR DISTRIBUTED AUTOMATA

- Given a distributed automaton  $P$  (plant) with two kinds of actions: **controllable** (system) and **uncontrollable** (environment), and a specification  $S$ .



## SUPERVISOR CONTROL PROBLEM FOR DISTRIBUTED AUTOMATA

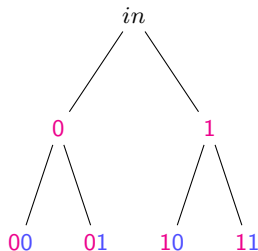
- Given a distributed automaton  $P$  (plant) with two kinds of actions: **controllable** (system) and **uncontrollable** (environment), and a specification  $S$ .
- Find a distributed automaton (controller)  $C$  such that  $P \times C \subseteq S$ .  
Controller must allow every uncontrollable action.

### REMARK

Decidability status: **open**.

## EXAMPLE 1

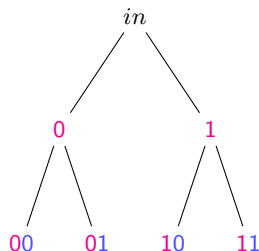
### SHARED BIT GAME



- Two processes  $P_0, P_1$ .
- $P_i$  receives an **uncontrollable** bit  $u_i$  and has to produce a **controllable** bit  $c_i$ .
- Winning:  $c_1 = u_0$  or  $c_0 = u_1$ .

## EXAMPLE 1

### SHARED BIT GAME



- Two processes  $P_0, P_1$ .
- $P_i$  receives an **uncontrollable** bit  $u_i$  and has to produce a **controllable** bit  $c_i$ .
- Winning:  $c_1 = u_0$  or  $c_0 = u_1$ .

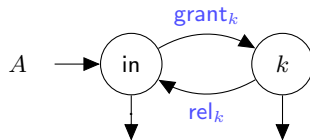
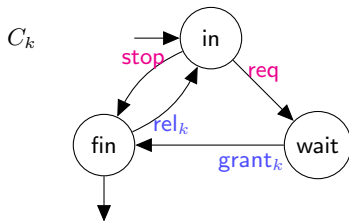
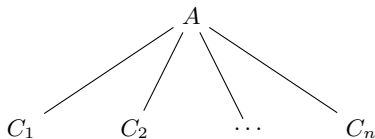
### HOW TO WIN

Distributed strategy:  $P_0$  plays  $c_0 = u_0$  and  $P_1$  plays  $c_1 = 1 - u_1$ .

Winning, since either  $u_0 = u_1$ , so  $u_1 = c_0$ . Or  $u_1 = 1 - u_0$ , so  $u_0 = c_1$ .

## EXAMPLE 2

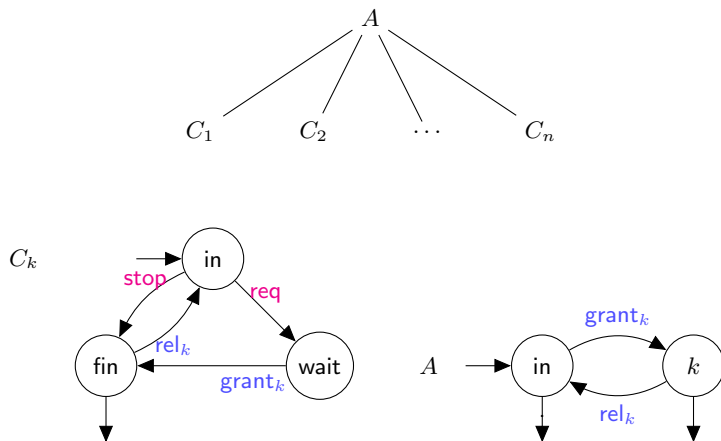
### ARBITER GAME



$req, stop$ : local uncontrollable actions,  $grant, rel$ : shared controllable actions.

## EXAMPLE 2

### ARBITER GAME

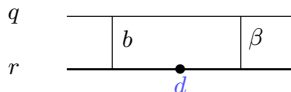
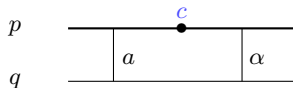


$req, stop$ : local uncontrollable actions,  $grant, rel$ : shared controllable actions.

Strategy of  $A$ : propose synchronization with every  $C_k$ .  
Winning, if the scheduler is fair.

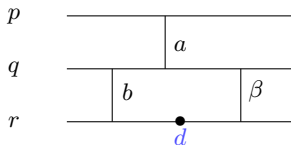
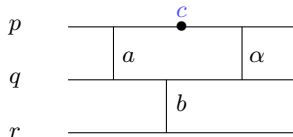
## EXAMPLE 3

- Plant:



Process  $q$ :  $(ab + ba)(\alpha + \beta)$

- Controllable actions:  $c$  and  $d$
- Specification:

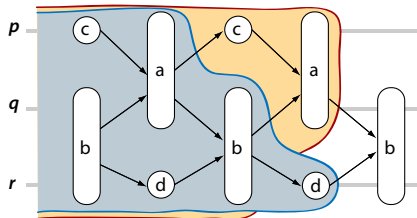


Plant is controllable: through communication with  $q$ , both processes  $p$  and  $r$  can learn about the order between  $a$  and  $b$ . If e.g.  $a$  preceded  $b$ , then  $p$ 's controller allows  $c$  and  $q$ 's controller disallows  $d$ .

# CONTROL FOR DISTRIBUTED AUTOMATA: GAME VERSION

STRATEGIES  $\sigma = (\sigma_p)_{p \in \mathbb{P}}$

Strategy of process  $p$ : mapping  $\sigma_p : \text{Views}_p \rightarrow 2^{\Sigma_p^{sys}}$ .



Before last  $b$ :

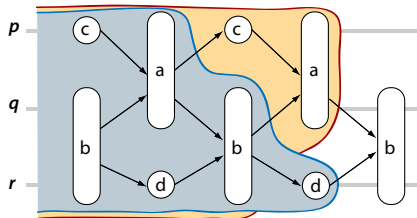
$$\text{view}_p = \text{view}_q = [cbadcba], \quad \text{view}_r = [cbadbd]$$

- $\Sigma_p^{sys} = \Sigma^{sys} \cap \{a : p \in \text{dom}(a)\}$ : set of controllable actions involving  $p$
- $\text{Views}_p$ : set of  $p$ -views of process  $p$

# CONTROL FOR DISTRIBUTED AUTOMATA: GAME VERSION

STRATEGIES  $\sigma = (\sigma_p)_{p \in \mathbb{P}}$

Strategy of process  $p$ : mapping  $\sigma_p : \text{Views}_p \rightarrow 2^{\Sigma_p^{sys}}$ .



- $\Sigma_p^{sys} = \Sigma^{sys} \cap \{a : p \in \text{dom}(a)\}$ : set of controllable actions involving  $p$
- $\text{Views}_p$ : set of  $p$ -views of process  $p$

Before last  $b$ :

$$\text{view}_p = \text{view}_q = [\text{cbad} \text{ cba}], \quad \text{view}_r = [\text{cbad} \text{ bd}]$$

## $\sigma$ -PLAYS ON $\mathcal{A}$

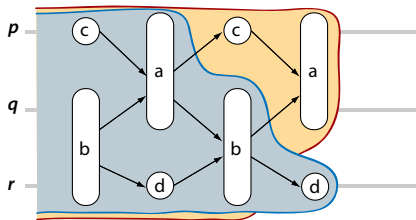
- If  $t$  is a  $\sigma$ -play and  $t \text{ a} \in L(\mathcal{A})$  with  $a$  uncontrollable, then  $t \text{ a}$  is a  $\sigma$ -play.
- If  $t$  is a  $\sigma$ -play,  $t \text{ a} \in L(\mathcal{A})$  with  $a$  controllable and  $a \in \sigma_p(t)$  for all  $p \in \text{dom}(a)$ , then  $t \text{ a}$  is a  $\sigma$ -play.



# DISTRIBUTED GAMES: PLAYS

## EXAMPLE

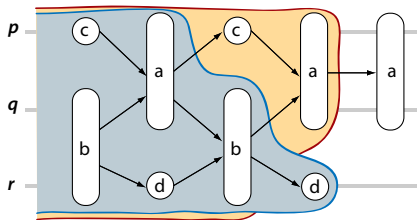
$$\sigma_q(T) = \{a, b\}, \sigma_p(T) = \{a\}, \sigma_r(T') = \{b\}.$$



# DISTRIBUTED GAMES: PLAYS

## EXAMPLE

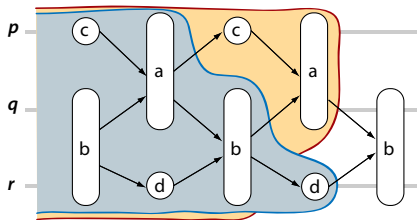
$$\sigma_q(T) = \{a, b\}, \sigma_p(T) = \{a\}, \sigma_r(T') = \{b\}.$$



# DISTRIBUTED GAMES: PLAYS

## EXAMPLE

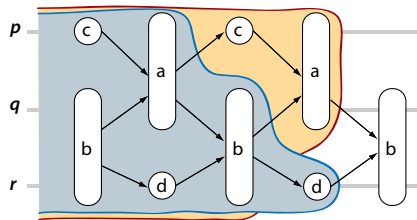
$$\sigma_q(T) = \{a, b\}, \sigma_p(T) = \{a\}, \sigma_r(T') = \{b\}.$$



# DISTRIBUTED GAMES: PLAYS

## EXAMPLE

$$\sigma_q(T) = \{a, b\}, \sigma_p(T) = \{a\}, \sigma_r(T') = \{b\}.$$



## WINNING CONDITIONS

- Most general:  $\omega$ -regular, trace-closed  $\leadsto$  repeating **global** states.
- Here: **local** conditions, one for each process (reachability, Büchi, ...).

A maximal play is winning if every process satisfies its local condition.

# DISTRIBUTED CONTROL VS. PNUELI & ROSNER

## PARTIAL INFORMATION

- In both cases: games with **partial knowledge**.
- In distributed control, partial knowledge is limited to **concurrency**: two synchronizing processes get **full** information about the other one.
- Distributed control: local controllers can exchange *a priori* **unbounded** knowledge (= process views). Unlike Pnueli & Rosner, where the specification tells what they are allowed to exchange. The crux here is to show that there is a bound on the additional knowledge exchanged by controllers.

# DISTRIBUTED CONTROL VS. PNUELI & ROSNER

## PARTIAL INFORMATION

- In both cases: games with **partial knowledge**.
- In distributed control, partial knowledge is limited to **concurrency**: two synchronizing processes get **full** information about the other one.
- Distributed control: local controllers can exchange *a priori* **unbounded** knowledge (= process views). Unlike Pnueli & Rosner, where the specification tells what they are allowed to exchange. The crux here is to show that there is a bound on the additional knowledge exchanged by controllers.

## (UN)DECIDABILITY?

- Unlikely to get undecidability of distributed control. Reason: the game is as **honest** as possible.
- But: distributed control problem gets undecidable if controllers do not exchange full information (loosely cooperating) or if their strategies are based only on local histories.

## DECIDABILITY: PARTIAL RESULTS

[MADHUSUDAN & THIAGARAJAN 2002]

Decidability for restricted local strategies:

- clocked: depending only on time, not history
- synchronization-rigid: each local strategy proposes either local actions or communication with the same process.

[GASTIN & LERMAN & ZEITOUN 2004]

Decidability for restricted communication architecture: co-graphs.

[MADHUSUDAN & THIAGARAJAN & YANG 2005]

Decidability for restricted distributed automata: every process misses only bounded knowledge. MSO specifications.

[GENEST & GIMBERT & M & WALUKIEWICZ 2013]

Decidability for acyclic process communication and local reachability conditions (blocking). Shared actions controllable.

Complexity: non-elementary (complete). EXPTIME-complete for depth one.

# DECIDABILITY FOR ACYCLIC PROCESS COMMUNICATION

## SETTING

- Shared actions are binary. Communication graph is **acyclic**.
- Shared actions are **uncontrollable**. Not a restriction.
- Each process has its own parity specification.

## THEOREM (M. & WALUKIEWICZ, 2014)

For a given plant (distributed automaton)  $\mathcal{A}$  and local parity specification, it is decidable whether a controller (distributed automaton)  $\mathcal{C}$  exists s.t. the controlled plant  $\mathcal{A} \times \mathcal{C}$  satisfies the parity specification.

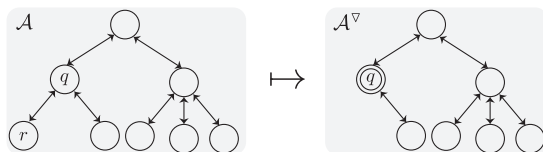
Complexity is **non-elementary**, EXPTIME-complete for depth one.



# PROOF

## MAIN IDEA

Induction over the processes: simulate a leaf process by its parent.



## PROOF IDEAS

- We can assume that  $\mathcal{A}$  has a **bound** on the number of local actions of process  $r$  between two consecutive synchronizations with  $q$ .

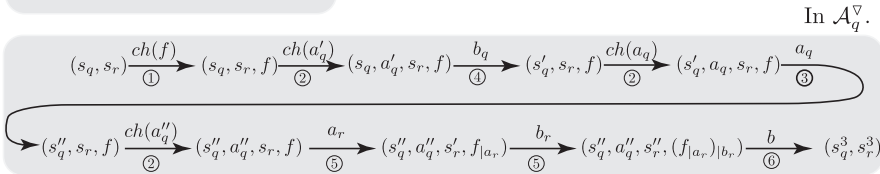
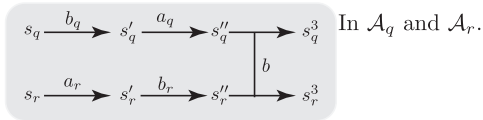
*Argument:* if  $\mathcal{A}$  can be controlled to satisfy a local parity specification, then the controller doesn't need to visit twice the same  $r$ -state during an  $r$ -local run.

- In  $\mathcal{A}^\nabla$ , process  $q$  simulates process  $r$  by **choosing an  $r$ -local strategy**, until simulating the next synchronization between  $q$  and  $r$ .

$$r\text{-local strategy: } f : (S_r)^* \rightarrow \Sigma_r^{sys}$$

# PROOF

## SIMULATING PROCESS $r$ BY PROCESS $q$



- $a_q, a'_q$  controllable  $q$ -actions,  $b_q$  uncontrollable  $q$ -action
- $f : (S_r)^* \rightarrow \Sigma_r^{sys}$  is  $r$ -local strategy
- only  $q$ -actions  $ch(\cdot)$  are controllable in  $\mathcal{A}^\nabla$
- $ch(f)$ : process  $q$  chooses  $r$ -local strategy

# DISTRIBUTED CONTROL: DEPTH 1

## SETTING

We assume here that the communication is over a tree of depth one: call the root  $q$ , and its children  $r_1, \dots, r_k$ .

## REM.

Recall: only local actions can be controllable. So any strategy  $\sigma = (\sigma_p)_{p \in \mathbb{P}}$  is such that  $\sigma_p$  always proposes one local  $p$ -action.

# DISTRIBUTED CONTROL: DEPTH 1

## SETTING

We assume here that the communication is over a tree of depth one: call the root  $q$ , and its children  $r_1, \dots, r_k$ .

## REM.

Recall: only local actions can be controllable. So any strategy  $\sigma = (\sigma_p)_{p \in \mathbb{P}}$  is such that  $\sigma_p$  always proposes one local  $p$ -action.

## LOCAL PLAYS AND STRATEGIES

- $\Sigma_r^{loc} = \{a \in \Sigma : \text{dom}(a) = \{r\}\}$ , set of local  $r$ -actions.
- $\Sigma_{q,r} = \{a \in \Sigma : \text{dom}(a) = \{q, r\}\}$ .
- Local  $r$ -play: word from  $(\Sigma_r^{loc})^*$ .
- $r$ -context: play ending in  $\Sigma_{q,r}$ .
- Local  $r$ -strategy  $\sigma_r[t] : (\Sigma_r^{loc})^* \rightarrow \Sigma_r^{loc}$  from  $r$ -context  $t$ :

$$\sigma_r[t](x) := \sigma_r(tx) \text{ for all } x \in (\Sigma_r^{loc})^*$$

# DISTRIBUTED CONTROL: DEPTH 1

## SETTING

We assume here that the communication is over a tree of depth one: call the root  $q$ , and its children  $r_1, \dots, r_k$ .

## REM.

Recall: only local actions can be controllable. So any strategy  $\sigma = (\sigma_p)_{p \in \mathbb{P}}$  is such that  $\sigma_p$  always proposes one local  $p$ -action.

## LOCAL PLAYS AND STRATEGIES

- $\Sigma_r^{loc} = \{a \in \Sigma : \text{dom}(a) = \{r\}\}$ , set of local  $r$ -actions.
- $\Sigma_{q,r} = \{a \in \Sigma : \text{dom}(a) = \{q, r\}\}$ .
- Local  $r$ -play: word from  $(\Sigma_r^{loc})^*$ .
- $r$ -context: play ending in  $\Sigma_{q,r}$ .
- Local  $r$ -strategy  $\sigma_r[t] : (\Sigma_r^{loc})^* \rightarrow \Sigma_r^{loc}$  from  $r$ -context  $t$ :

$$\sigma_r[t](x) := \sigma_r(tx) \text{ for all } x \in (\Sigma_r^{loc})^*$$

## KEY LEMMA

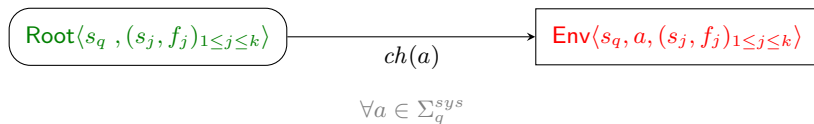
We can assume that each local  $r$ -strategy  $\sigma_r[t]$  is **positional**.

# FROM THE DISTRIBUTED CONTROL PROBLEM TO A SEQUENTIAL GAME

$\text{Root}\langle s_q, (s_j, f_j)_{1 \leq j \leq k} \rangle$

- $s_q$  is  $q$ -state,  $s_i$  is  $r_i$ -state,  $f_i$  is local positional  $r_i$ -strategy

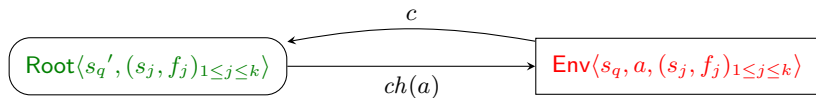
# FROM THE DISTRIBUTED CONTROL PROBLEM TO A SEQUENTIAL GAME



- $s_q$  is  $q$ -state,  $s_i$  is  $r_i$ -state,  $f_i$  is local positional  $r_i$ -strategy

# FROM THE DISTRIBUTED CONTROL PROBLEM TO A SEQUENTIAL GAME

$$\forall c \in \{a\} \cup \Sigma_q^{env} : s_q \xrightarrow{c} s'_q$$

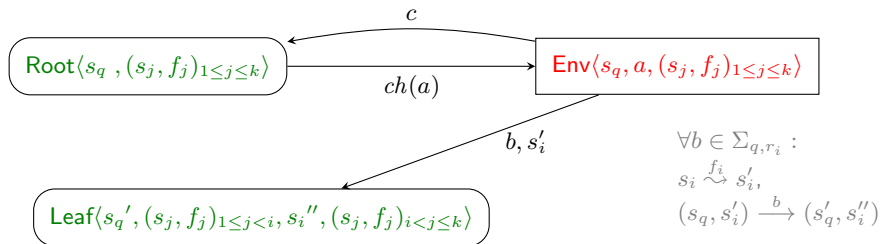


Environment either chooses a local action  $c$  for  $q$

- $s_q$  is  $q$ -state,  $s_i$  is  $r_i$ -state,  $f_i$  is local positional  $r_i$ -strategy



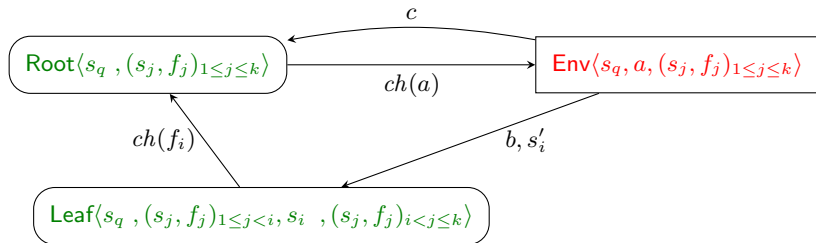
# FROM THE DISTRIBUTED CONTROL PROBLEM TO A SEQUENTIAL GAME



Or chooses a synchronization action  $b$  between  $q$  and  $r_i$ .

- $s_q$  is  $q$ -state,  $s_i$  is  $r_i$ -state,  $f_i$  is local positional  $r_i$ -strategy

# FROM THE DISTRIBUTED CONTROL PROBLEM TO A SEQUENTIAL GAME



- $s_q$  is  $q$ -state,  $s_i$  is  $r_i$ -state,  $f_i$  is local positional  $r_i$ -strategy
- Some more bookkeeping to record the maximal  $r_i$ -priority seen on the local  $r_i$  runs

## COMPLEXITY (GENERAL CASE)

### UPPER BOUND

The size of  $\mathcal{A}^\nabla$  is exponential in the size of  $\mathcal{A}$ : every reduction step increases the size of the plant by an exponential. Overall complexity is non-elementary in the *depth*  $n$  of the tree:

$$Tower(n) = 2^{Tower(n-1)}$$

# COMPLEXITY (GENERAL CASE)

## UPPER BOUND

The size of  $\mathcal{A}^\nabla$  is exponential in the size of  $\mathcal{A}$ : every reduction step increases the size of the plant by an exponential. Overall complexity is non-elementary in the *depth*  $n$  of the tree:

$$\text{Tower}(n) = 2^{\text{Tower}(n-1)}$$

## MATCHING LOWER BOUND: NESTED COUNTERS

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

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

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

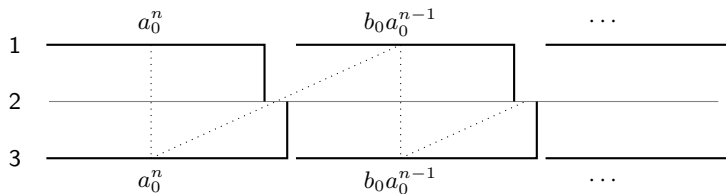
$$a_1 \text{bin}(0) a_1 \text{bin}(1) \cdots a_1 \text{bin}(2^n - 1) \# b_1 \text{bin}(0) a_1 \text{bin}(1) \cdots a_1 \text{bin}(2^n - 1) \# \cdots$$

- level  $k$ : ...

Turing machine with  $\text{Tower}(n)$  space bound.

# LEVEL 1

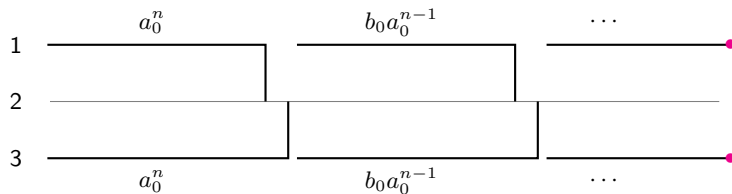
Processes 1,2,3.



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

## LEVEL 1

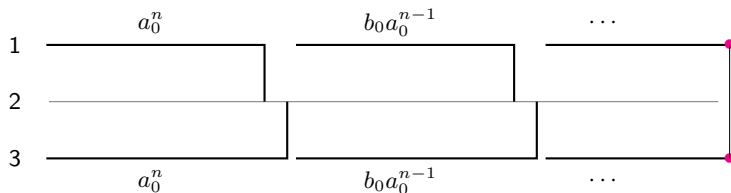
Processes 1,2,3.



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

## LEVEL 1

Processes 1,2,3.



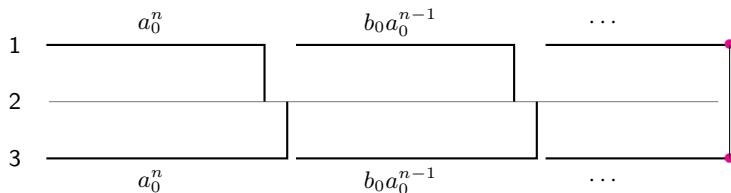
**Environment** can ask for a pair of bits: either  $\uparrow$  (above) or  $\nearrow$ .

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.



# WHAT MAKES DISTRIBUTED CONTROL SO DIFFICULT?

## BRANCHING AND DISTRIBUTED AUTOMATA

Branching behavior of distributed automata: event structures.

# WHAT MAKES DISTRIBUTED CONTROL SO DIFFICULT?

## BRANCHING AND DISTRIBUTED AUTOMATA

Branching behavior of distributed automata: event structures.

A prefix-closed trace language  $L$  defines a  $\Sigma$ -labeled event structure:

- Nodes: traces from  $L$  with *one* maximal element (prime traces).
- Partial order: trace prefix relation.
- Conflict relation: no common extension.
- Label: maximal element of the trace.

# WHAT MAKES DISTRIBUTED CONTROL SO DIFFICULT?

## BRANCHING AND DISTRIBUTED AUTOMATA

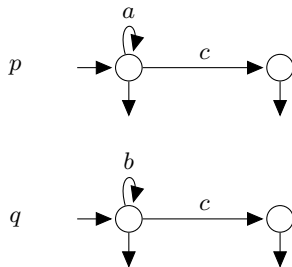
Branching behavior of distributed automata: **event structures**.

A prefix-closed trace language  $L$  defines a  $\Sigma$ -labeled event structure:

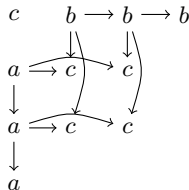
- Nodes: traces from  $L$  with *one* maximal element (**prime** traces).
- Partial order: trace prefix relation.
- Conflict relation: no common extension.
- Label: maximal element of the trace.

## EXAMPLE

$$\Sigma = \{a, b, c\}, L = (a + b)^* c$$



Event structure:



# WHAT MAKES DISTRIBUTED CONTROL SO DIFFICULT?

## EVENT STRUCTURES - A SOLUTION?

The control problem for a distributed plant  $P$  reduces to the satisfiability of a monadic second-order formula over the event structure of  $P$  [Madhusudan et al.].

# WHAT MAKES DISTRIBUTED CONTROL SO DIFFICULT?

## EVENT STRUCTURES - A SOLUTION?

The control problem for a distributed plant  $P$  reduces to the satisfiability of a monadic second-order formula over the event structure of  $P$  [Madhusudan et al.].

## PROOF

MSO formula  $\exists X_A \exists X_B \cdots \varphi$ , with quantifiers ranging over all subsets  $A \subseteq \Sigma_p^{sys}$ ,  $B \subseteq \Sigma_q^{sys}$ , ....

- Set  $X_A \cup X_B \cup \cdots$  consists of all prime traces ("histories") that are compatible with the strategy  $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ .
- Prime trace belongs to  $X_A$ ,  $A \subseteq \Sigma_p^{sys}$ , if its maximal element has label from  $\{a \in \Sigma : p \in \text{dom}(a)\}$  and  $\sigma_p([w]) = A$ .
- Formula  $\varphi$  expresses that (1) every uncontrollable action is allowed by  $\sigma$  and (2) that the winning condition is satisfied by every trace representing a  $\sigma$ -play.

# EVENT STRUCTURES - A SOLUTION?

## UNFORTUNATELY NOT:

- There exist distributed automata s.t. the associated event structure has **undecidable** MSO theory.
- Thiagarajan's conjecture: the event structure of a distributed automaton  $\mathcal{A}$  has decidable MSO theory iff  $\mathcal{A}$  has no concurrent loops.
- But: Decidability of MSO is **not** necessary for deciding the control problem.  
Example: the control problem for distributed automata over 2 processes is decidable.

# WHAT MAKES ASYNCHRONOUS CONTROL SO DIFFICULT?

From



to



... and back?

# WHAT MAKES ASYNCHRONOUS CONTROL SO DIFFICULT?

## PARTIAL KNOWLEDGE

A process  $p$  has only **partial** knowledge about other processes. What happens in “parallel” to  $p$  **may** affect  $p$ 's future.

## TAMING PARTIAL KNOWLEDGE: TWO EXAMPLES

- **Acyclic case**: parent of the leaf process  $r$  knows everything about  $r$ , except for local behavior that can be resumed.
- **Missing knowledge** is bounded:

*Every event has at most  $n$  concurrent events (unless their processes never meet again in the future).*



MADHUSUDAN, THIAGARAJAN 2005

Assume that the distributed plant  $P$  is such that in every run, every event has at most  $n$  concurrent events (unless their processes never meet again in the future). Then the MSO theory of the event structure of  $P$  is decidable.

# BOUNDED MISSING KNOWLEDGE

MADHUSUDAN, THIAGARAJAN 2005

Assume that the distributed plant  $P$  is such that in every run, every event has at most  $n$  concurrent events (unless their processes never meet again in the future). Then the MSO theory of the event structure of  $P$  is decidable.

PROOF

Idea: work on words/trees.

### MADHUSUDAN, THIAGARAJAN 2005

Assume that the distributed plant  $P$  is such that in every run, every event has at most  $n$  concurrent events (unless their processes never meet again in the future). Then the MSO theory of the event structure of  $P$  is decidable.

### PROOF

Idea: work on words/trees.

- Fix a total order on the set  $\Sigma$  of actions and consider only the set  $\mathcal{L}(P)$  of runs of  $P$  that (1) are lexicographically minimal and (2) have a unique maximal event.

Both (1) and (2) are MSO-expressible.

## MADHUSUDAN, THIAGARAJAN 2005

Assume that the distributed plant  $P$  is such that in every run, every event has at most  $n$  concurrent events (unless their processes never meet again in the future). Then the MSO theory of the event structure of  $P$  is decidable.

## PROOF

Idea: work on words/trees.

- Fix a total order on the set  $\Sigma$  of actions and consider only the set  $\mathcal{L}(P)$  of runs of  $P$  that (1) are lexicographically minimal and (2) have a unique maximal event.

Both (1) and (2) are MSO-expressible.

- *Key property:* if  $x, y \in \mathcal{L}(P)$  are such that  $[x] < [y]$  then  $x = zx'$  and  $y = zy'$  for some  $z, x', y' \in \Sigma^*$  with  $|x'| \leq n$ .

## MADHUSUDAN, THIAGARAJAN 2005

Assume that the distributed plant  $P$  is such that in every run, every event has at most  $n$  concurrent events (unless their processes never meet again in the future). Then the MSO theory of the event structure of  $P$  is decidable.

## PROOF

Idea: work on words/trees.

- Fix a total order on the set  $\Sigma$  of actions and consider only the set  $\mathcal{L}(P)$  of runs of  $P$  that (1) are lexicographically minimal and (2) have a unique maximal event.

Both (1) and (2) are MSO-expressible.

- *Key property:* if  $x, y \in \mathcal{L}(P)$  are such that  $[x] < [y]$  then  $x = zx'$  and  $y = zy'$  for some  $z, x', y' \in \Sigma^*$  with  $|x'| \leq n$ .

Proof: Write  $y' = y_1x_1y_2 \cdots y_k$ , with  $x_1 \cdots x_{k-1} \sim x$  and  $\text{dom}(y_j) \cap \text{dom}(x_j \cdots x_{k-1}) = \emptyset$ . If  $y_1 \neq \epsilon$  then  $|x| \leq n$ .

$\leadsto$  MSO (FO) interpretation of the trace-prefix order into word-prefix order.

# CONCLUSIONS

## SYNTHESIS

- We saw Church's formulation of the synthesis problem in the 50's and the interplay with logic on trees (Rabin's theorem about MSO over the infinite binary tree).
- Alternative setting: control/supervisory theory (Ramadge & Wonham).
- We saw some simple 2-player games, and McNaughton's algorithm for parity games.
- We saw Pnueli and Rosner's version of distributed synthesis and we discussed why it is almost always undecidable: games with (very) partial information - no communication between controllers.
- We saw a second version of distributed synthesis, this time **with information exchange**: control of **distributed automata**. Decidability of control holds if the communication is acyclic.

# CONCLUSIONS

## CONTROL OF DISTRIBUTED AUTOMATA: OPEN QUESTIONS

- Decidability of the general asynchronous control problem? Open.
- When distributed control is decidable, we finally reason on trees. Is there anything beyond?
- Which parameters make the control problem difficult? How can we capture missing knowledge in a systematic way?
- We lack a good understanding of branching in the distributed case. Event structures are too much of a jungle.
- Show Thiagarajan's conjecture.

# CONCLUSIONS

## CONTROL OF DISTRIBUTED AUTOMATA: OPEN QUESTIONS

- Decidability of the general asynchronous control problem? Open.
- When distributed control is decidable, we finally reason on trees. Is there anything beyond?
- Which parameters make the control problem difficult? How can we capture missing knowledge in a systematic way?
- We lack a good understanding of branching in the distributed case. Event structures are too much of a jungle.
- Show Thiagarajan's conjecture.

Thank you for listening!