Control and Synthesis, from a Distributed Perspective

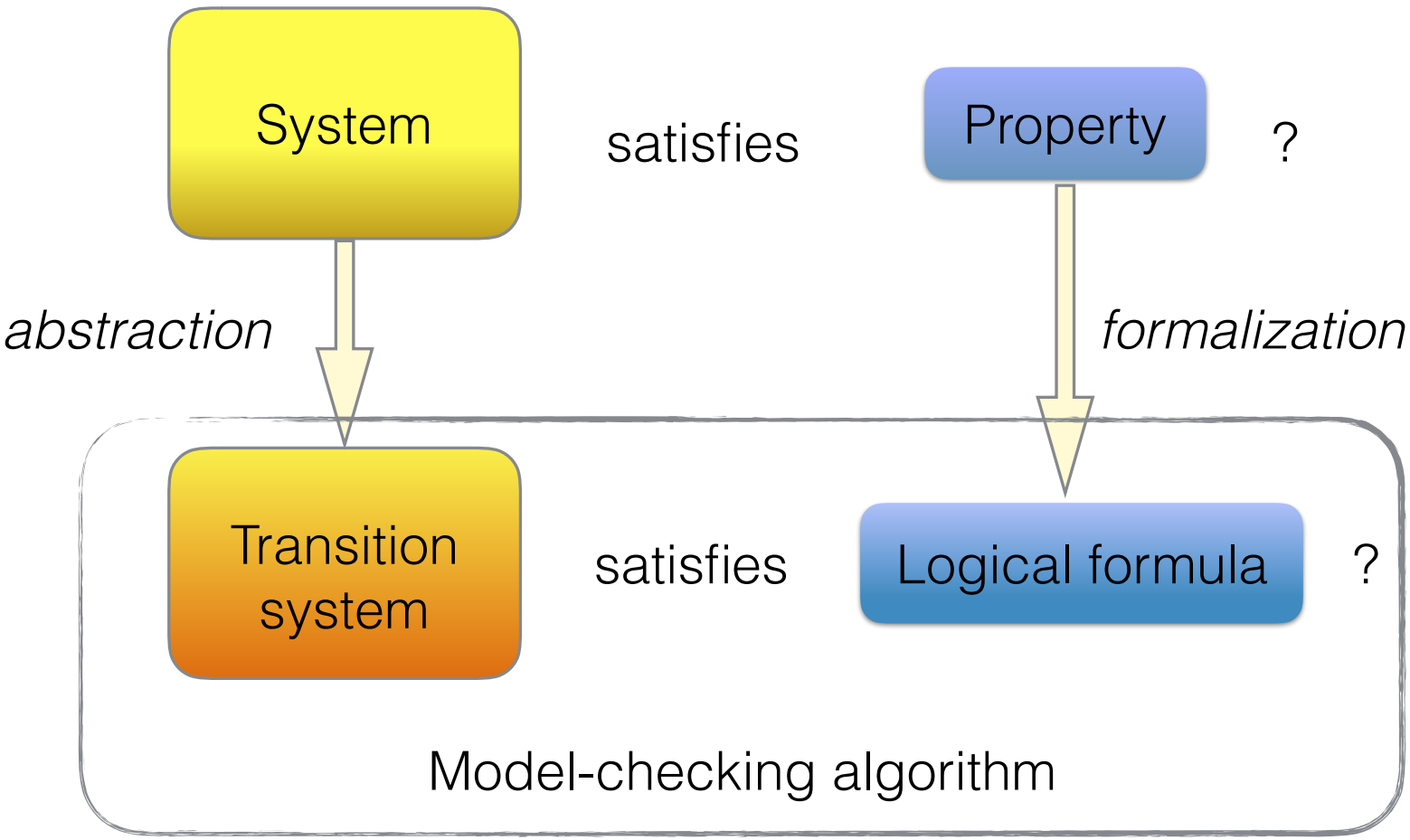
Anca Muscholl

VTSA Summer School, Nancy, August 2018

Outline

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

Model-checking



Synthesis

"Write programs that are correct by construction."

Given a logical formula, does an equivalent transition system exist? If yes, construct one.

Closed/open synthesis

- Closed synthesis: no environment.
- Open synthesis: the system is reactive, it evolves "against" the environment.

Synthesis

"Write programs that are correct by construction."

Given a logical formula, does an equivalent transition system exist? If yes, construct one.

Closed/open synthesis

- Closed synthesis: no environment.
- Open synthesis: the system is reactive, it evolves "against" the environment.

Synthesis of reactive systems has wide applicability. Think about writing a module that will be part of a larger system: the remaining system can be abstracted as the "environment" that interacts with the module. The module needs to be correct for all possible interactions.

Synthesis of closed, centralized 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 implementable in a distributed model.

Some history

Synthesis of open, sequential systems: games

- 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

Synthesis of open, sequential systems: games

- 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.*

Synthesis of 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.*

I. Automata and logic: back to basics

Automata, logic and verification

Transition systems

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

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

Finite automata

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

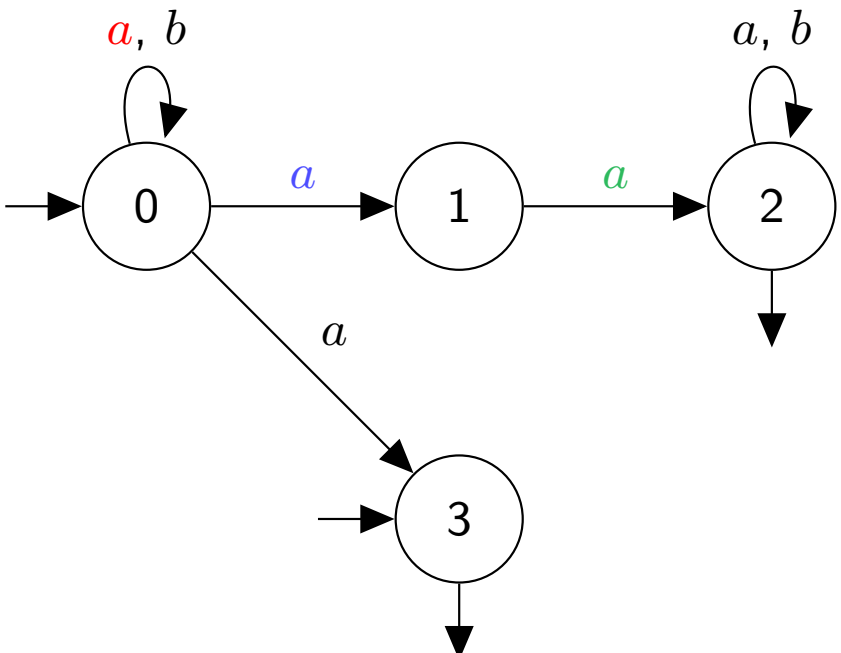
- S is the finite set of states
- Σ is a (finite) alphabet
- $S_0 \subseteq S$ is th set of initial states
- $(\stackrel{a}{\longrightarrow})_{a \in \Sigma} \subseteq S \times S$ is the transition relation (function = deterministic)
- Acc is the acceptance condition

Regular language

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

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

Example (automaton)



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

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

Automata

Regular

A word language $L \subseteq \Sigma^*$ (resp. $L \subseteq \Sigma^\omega$) is regular (resp. ω-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 condition).

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*), μ-calculus
- Monadic Second-Order Logic (MSO)

Model-checking

Roadmap to check whether a transition system $\mathcal{S}$ satisfies a formula φ:

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

Semantics

Relational structure associated with a word $w = a_1 a_2 \cdots$ over Σ, with $\mathrm{dom}(w) = \{1, 2, \ldots\}$:

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

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

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

Language of φ

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

Examples

- *Every odd position carries an a.*

$$\exists X_0 \exists X_1 \quad \Big(\forall x \, \big((x \in X_0 \vee x \in X_1) \wedge (x \in X_0 \Leftrightarrow x \notin X_1) \big) \wedge 1 \in X_1 \wedge$$
$$\forall x \, \big((x \in X_0 \Leftrightarrow \mathsf{succ}(x) \in X_1) \wedge (x \in X_1 \Leftrightarrow \mathsf{succ}(x) \in X_0) \big)$$
$$\forall x \, (x \in X_1 \Rightarrow P_a(x)) \Big) =: \mathsf{ODD}_a$$

Language of φ

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

Examples

- *Every odd position carries an a.*

$$\exists X_0 \exists X_1 \quad \Big(\forall x \left((x \in X_0 \vee x \in X_1) \wedge (x \in X_0 \Leftrightarrow x \notin X_1)\right) \wedge 1 \in X_1 \wedge$$
$$\forall x \left((x \in X_0 \Leftrightarrow \mathsf{succ}(x) \in X_1) \wedge (x \in X_1 \Leftrightarrow \mathsf{succ}(x) \in X_0)\right)$$
$$\forall x \left(x \in X_1 \Rightarrow P_a(x)\right)\Big) =: \mathsf{ODD}_a$$

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

Language of φ

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

Examples

- *Every odd position carries an a.*

$$\exists X_0 \exists X_1 \quad \Big(\forall x \left((x \in X_0 \vee x \in X_1) \wedge (x \in X_0 \Leftrightarrow x \notin X_1)\right) \wedge 1 \in X_1 \wedge$$
$$\forall x \left((x \in X_0 \Leftrightarrow \mathsf{succ}(x) \in X_1) \wedge (x \in X_1 \Leftrightarrow \mathsf{succ}(x) \in X_0)\right)$$
$$\forall x \left(x \in X_1 \Rightarrow P_a(x)\right)\Big) =: \mathsf{ODD}_a$$

- $\varphi := \mathsf{ODD}_a \wedge \mathsf{EVEN}_b \qquad L(\varphi) = (ab)^+$
- $(ab)^+$ is also definable in first-order logic (FO)

$$\forall x \Big(P_a(x) \Rightarrow P_b(\mathsf{succ}(x)) \wedge P_b(x) \Rightarrow P_a(\mathsf{succ}(x)) \wedge P_a(\min) \wedge P_b(\max)\Big)$$

Büchi-Elgot-Trakhtenbrot theorem ($\sim$ 1960)

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 $\mathrm{dom}(w)$ 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 $k \in \mathrm{dom}(w)$, $s \in S$, $a \in \Sigma$,

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

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

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 1985, Emerson-Jutla 1991):

Parity automaton

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

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.

Summary

- Model-checking linear-time properties (LTL, MSO) requires automata & logic over infinite words.
- Both model-checking branching-time properties (CTL*, mu-calculus) and synthesis require automata & logic over infinite trees.

MSO over trees

Binary trees

- Finite binary tree over alphabet Σ = partial mapping

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

  such that $\mathrm{dom}(t)$ is finite, prefix-closed and $x1 \in \mathrm{dom}(t)$ iff $x0 \in \mathrm{dom}(t)$, for all x.
  ϵ: root, $0, 1$: children of the root, etc.
- Infinite binary tree over Σ = total mapping $t : \{0,1\}^* \to \Sigma$.

MSO

Two successors left/right:

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

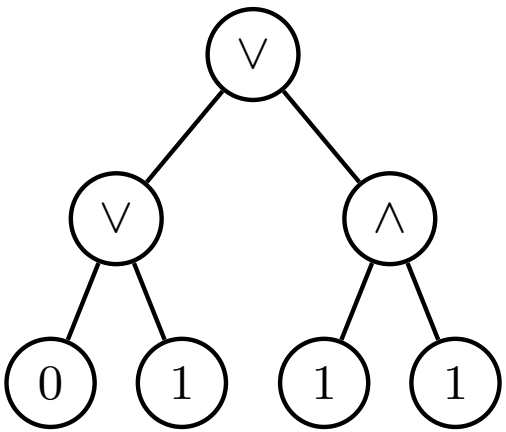
Tree automata

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

- a finite set of states S,
- a finite alphabet Σ,
- 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 Acc.

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

Example



Trees that evaluate to 1 at the root.

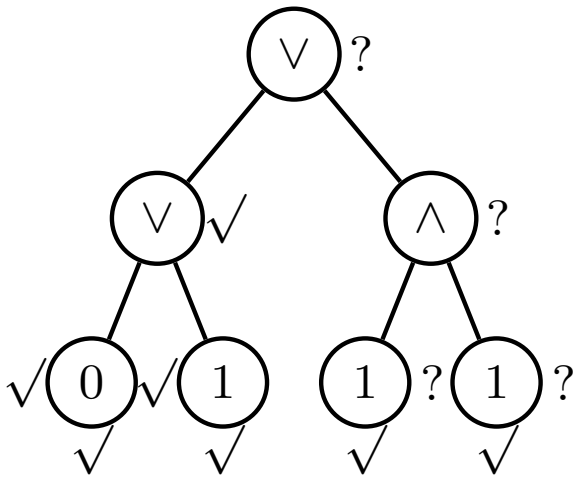
Acc is a set $F \subseteq S$ of final states. Run is successful if it ends on all leaves in a final state.

- $S = \{?, \surd\}$
- $S_0 = \{?\}$, $F = \{\surd\}$
- $\xrightarrow{\vee}$
  $=\{(\surd, (\surd, \surd)), (?, (?, *)), (?, (*, ?))\}$
  $\xrightarrow{\wedge} =\{(\surd, (\surd, \surd)), (?, (?, ?))\}$
  $\xrightarrow{0}= \{(\surd, \surd)\}$,
  $\xrightarrow{1}= \{(\surd, \surd), (?, \surd)\}$

Determinization

Deterministic bottom-up tree automata are an equivalent model.

Example



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.

- $S = \{?, \surd\}$
- $S_0 = \{?\}$, $F = \{\surd\}$
- $\xrightarrow{\vee}$
  $=\{(\surd, (\surd, \surd)), (?, (?, *)), (?, (*, ?))\}$
  $\xrightarrow{\wedge} =\{(\surd, (\surd, \surd)), (?, (?, ?))\}$
  $\xrightarrow{0} = \{(\surd, \surd)\}$,
  $\xrightarrow{1} = \{(\surd, \surd), (?, \surd)\}$

Determinization

Deterministic bottom-up tree automata are an equivalent model.

Büchi condition

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

Parity condition

Acc is a labeling of states by priorities from $\{0, \ldots, p\}$. 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 1968, Doner 1970

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 1969

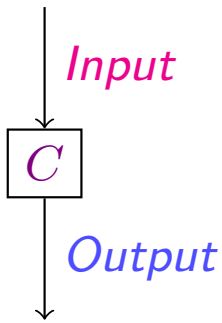
A language of infinite trees is accepted by some parity tree automaton iff it is definable in MSO. Both conversions are effective.

Cor.

MSO over infinite, binary trees is decidable.

II. Basics on games and controller synthesis

Church's problem (1963) "Logic, arithmetic and automata"

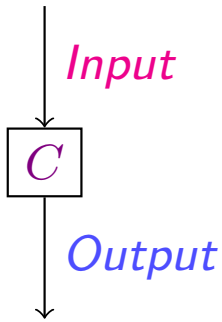


Problem

- *Given*: specification $\mathcal{R} \subseteq (\{0,1\} \times \{0,1\})^\omega$ relating inputs/outputs.
- *Output*: I/O device $C : \{0,1\}^* \to \{0,1\}$ 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

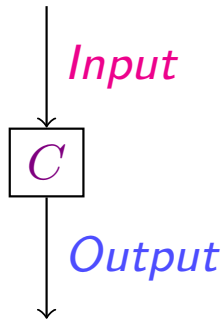
- *Given*: specification $\mathcal{R} \subseteq (\{0,1\} \times \{0,1\})^\omega$ relating inputs/outputs.
- *Output*: I/O device $C : \{0,1\}^* \to \{0,1\}$ s.t. $(x, C(x)) \in \mathcal{R}$ for all inputs x.

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 $C : \{0,1\}^* \to \{0,1\}$ s.t. $(x, C(x)) \in \mathcal{R}$ for all inputs x.

Controller C must react correctly on every input.

Remarks

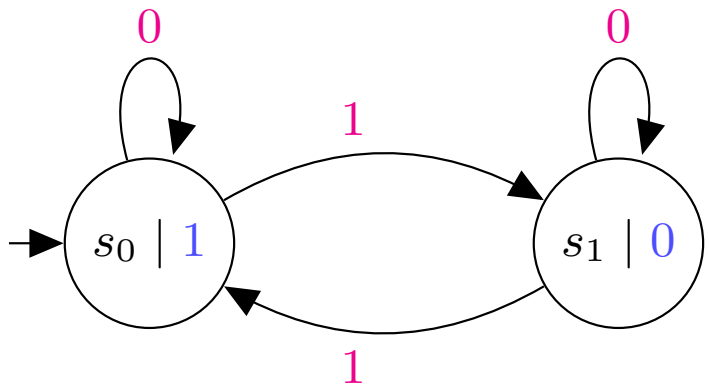
- The specification $\mathcal{R}$ is provided in an effective way, by an MSO formula or a Büchi automaton.
- The problem is more complicated than just requiring $\forall x \, \exists y \, .(x, y) \in \mathcal{R}$: controller C must react *continuously* on inputs.

Church's problem:
Synthesis of open systems: systems reacting on input from environment.

Examples

Ex. 1

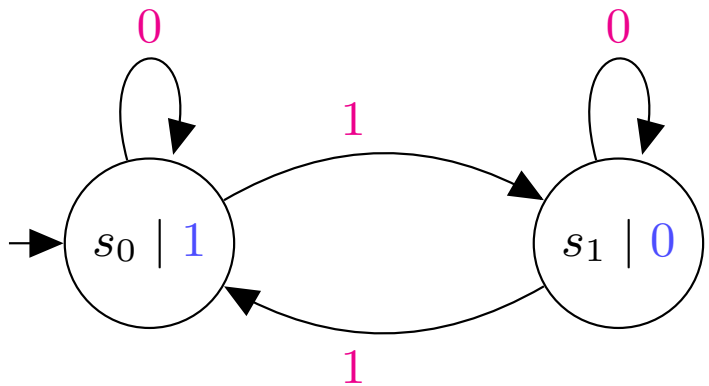
$\mathcal{R}$: "*the output is 1 iff the number of previous inputs equal to 1, is even*"



Examples

Ex. 1

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



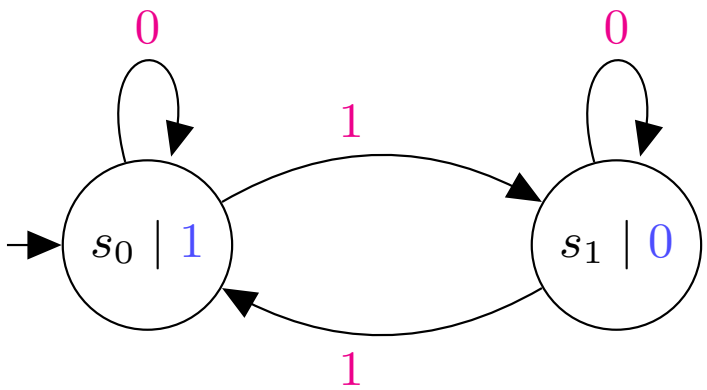
Ex. 2

$\mathcal{R}$: "*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

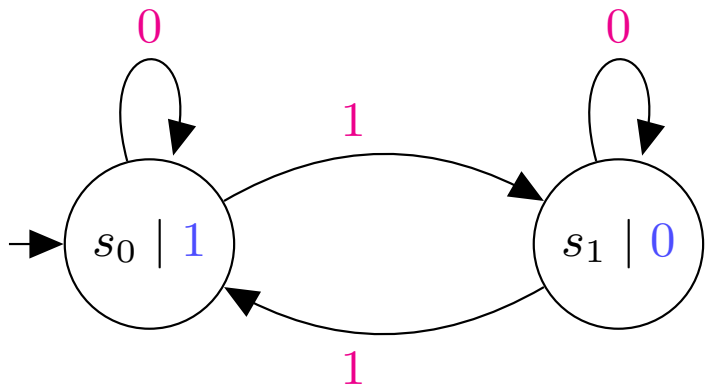
$\mathcal{R}$: "*the output is 1 iff some future input is 1*"

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.

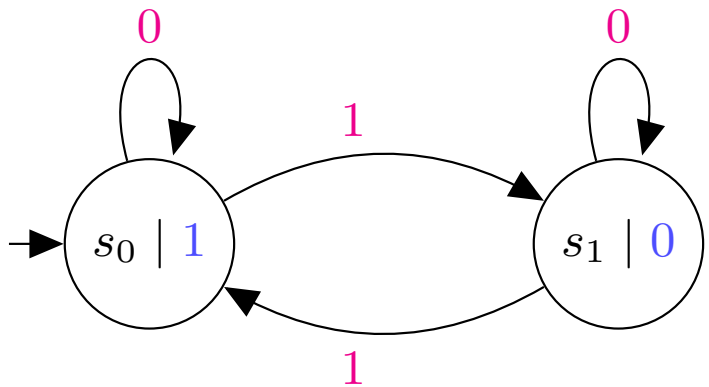
Ex. 3

$\mathcal{R}$: "*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^ω 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^ω 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.

Specifications

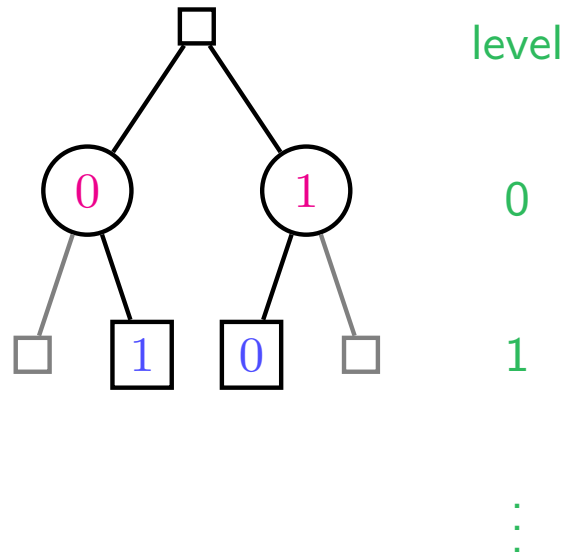
Specification $\mathcal{R} \subseteq (\{0,1\}\{0,1\})^\omega$: finite description, e.g. Büchi automaton or MSO.

Trees

Synthesis is concerned with trees:

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

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



Church's problem and logic

Specifications

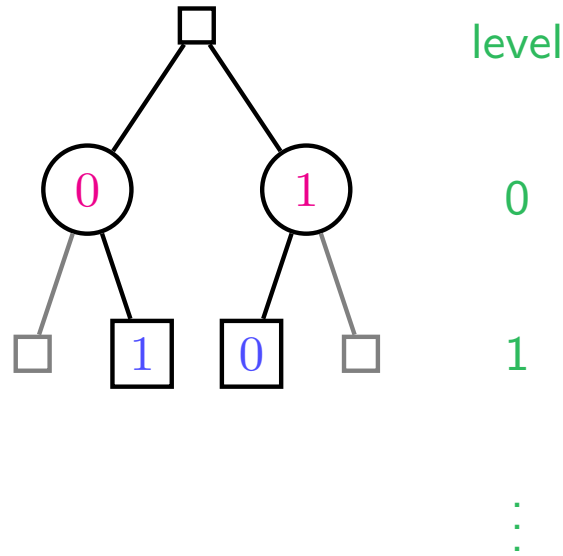
Specification $\mathcal{R} \subseteq (\{0,1\}\{0,1\})^\omega$: finite description, e.g. Büchi automaton or MSO.

Trees

Synthesis is concerned with trees:

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

Controller $C : \{0,1\}^* \to \{0,1\}$ 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.

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 φ is satisfiable, then it has a regular tree model: tree that is the unfolding of a finite automaton. □

Example

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

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

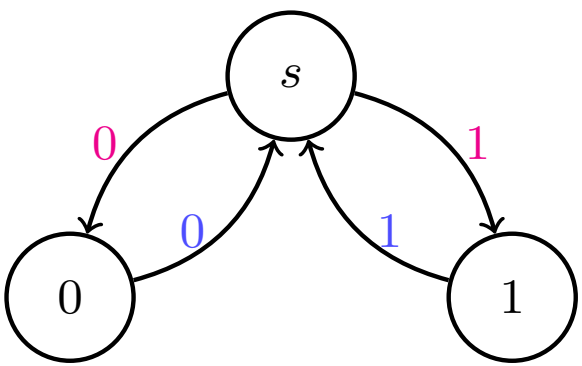
$X_0 =$ even level nodes, $X_1 =$ odd level nodes,

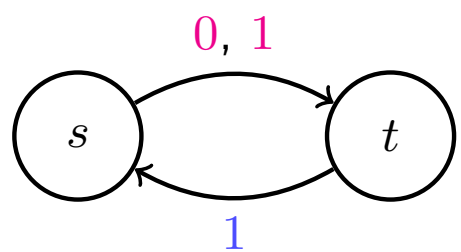
root $\in Z$,

$\forall x \in Z$ (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 of plays in G.
  Parity game: priorities $p : V \to \{0, \ldots, d\}$. A play is winning if the highest priority visited infinitely often is even.
- Strategies: $\sigma_0 : V^* V_0 \to V$, $\sigma_1 : V^* V_1 \to V$.
- Strategy σ_0 is winning for P_0 from $v \in V$ if every play from v that is consistent with σ_0 is winning.
- 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.

Game solution

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

Strategies

"Nice" strategies are

- positional (= memoryless)

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

- or finite-memory

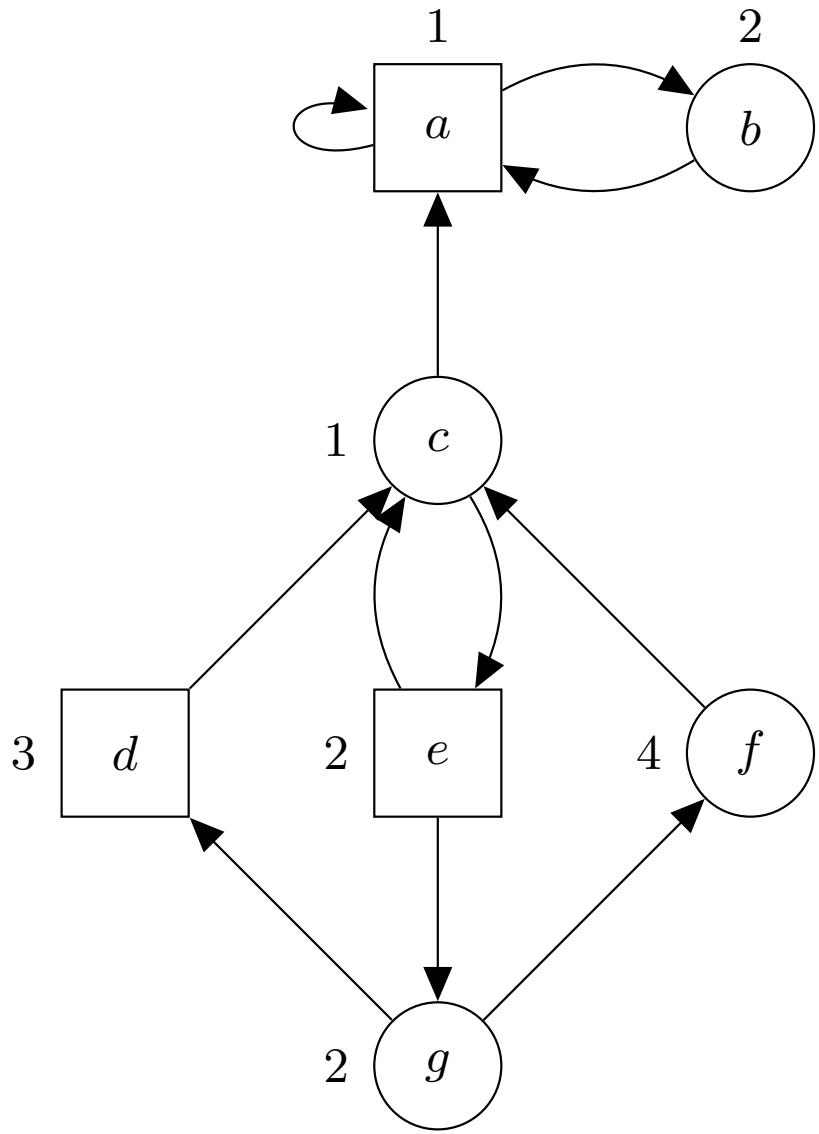
$$\sigma_0 : (V_0 \times M) \to V, \quad \sigma_1 : (V_1 \times M) \to V\,,$$

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

Determined games

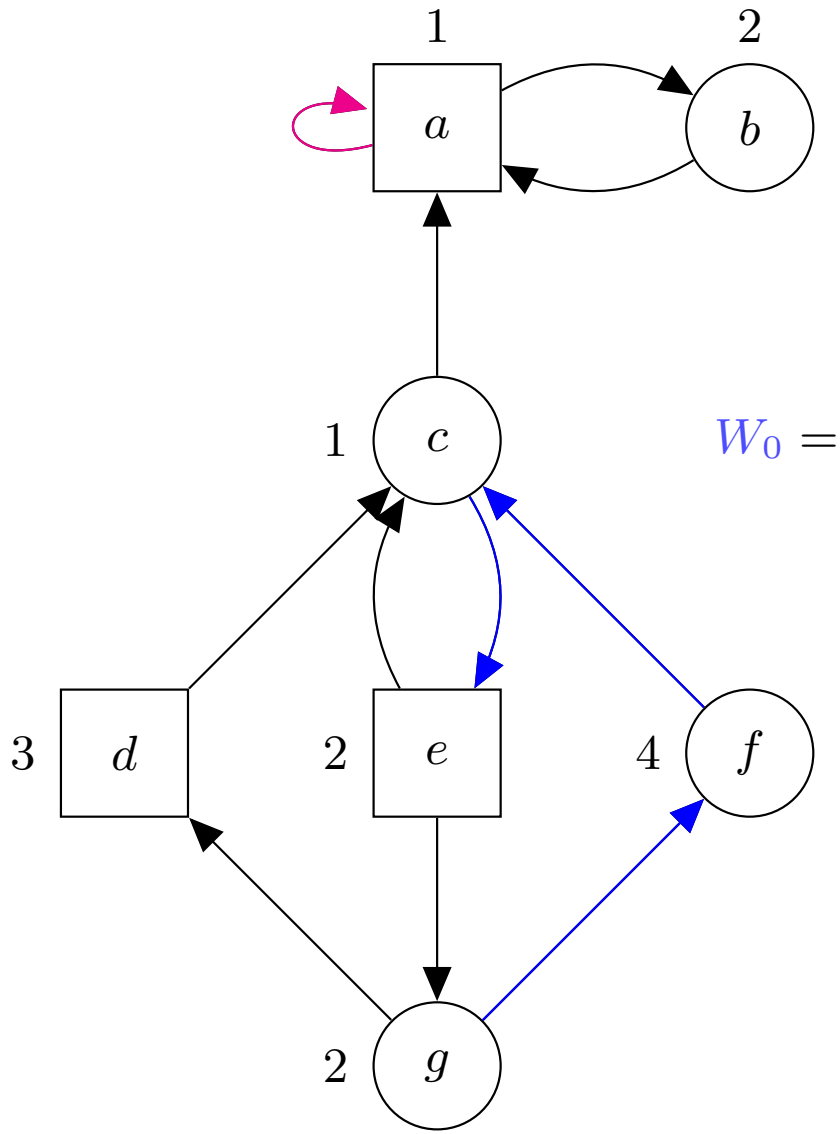
A (graph) game is determined if $V = W_0 \cup W_1$ (this actually partitions V if the game is zero-sum).

Example (parity)



Plays won by P_0: $ababa\ldots$, $cegfcegf\ldots$, $cece\ldots$.
Plays won by P_1: $aa\ldots$, $cegdcegd\ldots$

Example (parity)



$W_0 = \{c, d, e, f, g\}$, $W_1 = \{a, b\}$

Plays won by P_0: $ababa\ldots$, $cegfcegf\ldots$, $cece\ldots$.
Plays won by P_1: $aa\ldots$, $cegdcegd\ldots$

Church's problem as graph game (McNaughton 1966, Büchi-Landweber 1967)

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 $S = S_\bigcirc \cup S_\Box$:
  from $S_\bigcirc$ only transitions with $\{0,1\}$, from $S_\Box$ only transitions with $\{0,1\}$.
  Initial state $s_0 \in S_\Box$.
- Player P_0 owns $V_0 = S_\bigcirc$, player P_1 owns $V_1 = S_\Box$.
- Play $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ = maximal path in $\mathcal{A}_\mathcal{R}$.
- A play is winning for P_0 iff the path satisfies the parity condition $\Longrightarrow$ 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).

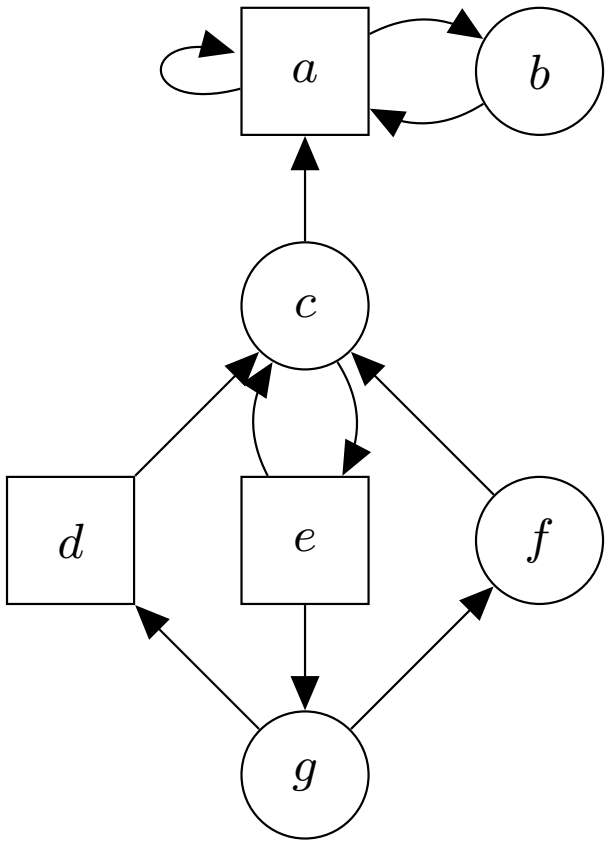
Parity games: complexity

- Parity games are determined, and winning strategies are positional (memoryless). On finite graphs, deciding the winner is in NP $\cap$ 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 (McNaughton-Zielonka): $O(n^{d+O(1)})$.
- Recent breakthrough: $O(n^{\log(d)+O(1)})$ (quasi-polyonomial) [Calude-Jain-Khoussainov-Li-Stephan 2016, Jurdzinski-Lazic 2017].

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

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

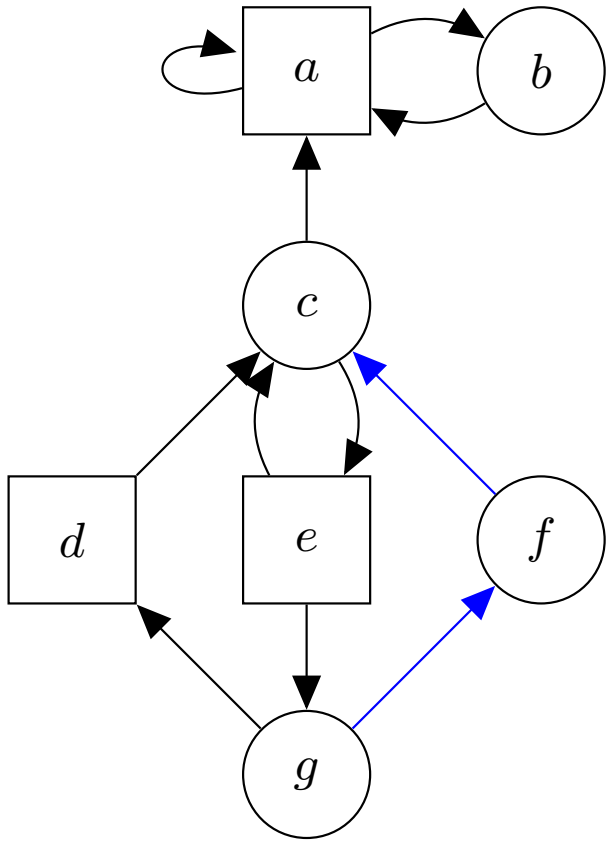
$$\mathsf{Attr}_0^0(F) \subseteq \mathsf{Attr}_0^1(F) \subseteq \cdots \subseteq \mathsf{Attr}_0^{|V|}(F)$$

$\mathsf{Attr}_0^i(F)$ is the set of vertices from which P_0 can reach F after at most i moves.
$W_0 = \mathsf{Attr}_0^{|V|}(F)$ is the winning region of P_0 (smallest fixpoint), and
$W_1 = V \setminus \mathsf{Attr}_0^{|V|}(F)$ is the winning region of P_1 (trap for P_0).

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.

Example



$F = \{b, c\}$
$\mathsf{Attr}_0^0(F) = \{b, c, f\}$
$\mathsf{Attr}_0^1(F) = \{b, c, f, g, d\}$
$\mathsf{Attr}_0^2(F) = \{b, c, f, g, d, e\}$

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

Büchi games

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

- $\mathsf{Attr}_0^+(F)$: set of states from which P_0 can reach F in at least one move. We can compute $\mathsf{Attr}_0^+(F)$, 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).

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

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

Strategies

Büchi games are determined and have positional winning strategies: Attr_0^+ strategy for P_0 and trap strategy for P_1 (positional).
Both winning regions / strategies can be computed in PTime.

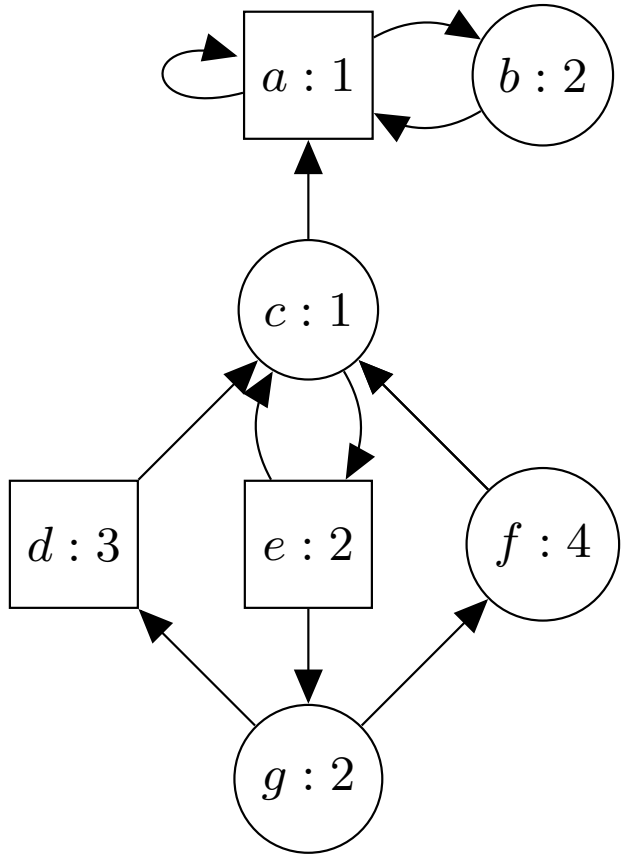
McNaughton-Zielonka recursive algorithm

Input: Parity game $G = (V_0, V_1, E)$, $p : V \to \{0, \ldots, k\}$
Output: Parity$(G) = (W_0, W_1)$

if $V = \emptyset$ **then**
  return $(\emptyset, \emptyset)$

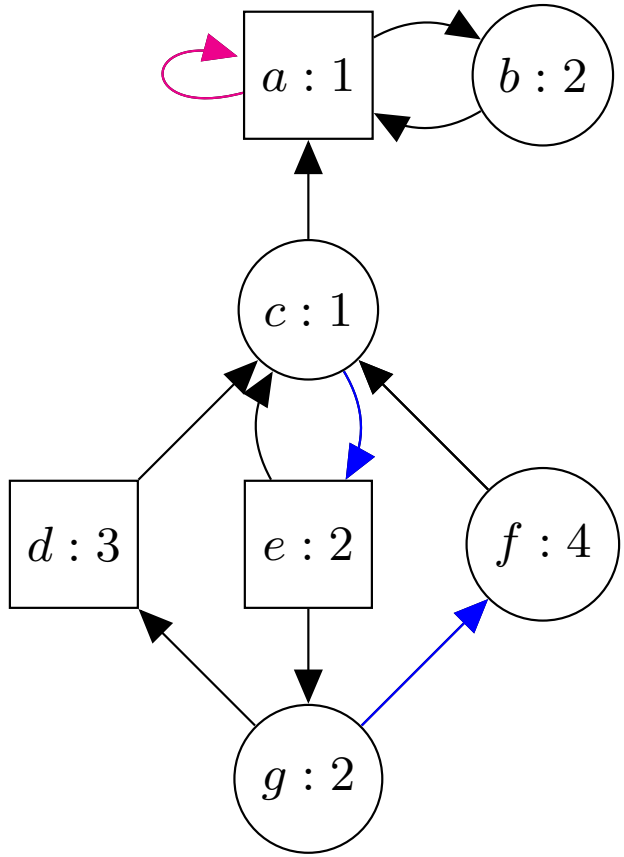
$i := k \bmod 2$; /* parity of maximal priority */
$U := \{v \in V : p(v) = k\}$; /* vertices of maximal priority */
$A :=$ Attr$_i(U)$;
$(W'_i, W'_{1-i}) =$ Parity$(G \setminus A)$;
if $W'_{1-i} = \emptyset$ **then**
  $W_i := W'_i \cup A$;
  $W_{1-i} := \emptyset$;
  return (W_i, W_{1-i});
else
  $B :=$ Attr$_{1-i}(W'_{1-i})$; /* Attractor in G */
  $(W''_i, W''_{1-i}) =$ Parity$(G \setminus B)$;
  $W_i := W''_i$;
  $W_{1-i} := B \cup W''_{1-i}$;
  return (W_i, W_{1-i});
end

Example



- $A = \mathsf{Attr}_0(f) = \{f, g\}$
- Recursive call on $G \setminus A$ yields $W_0' = \{c, d, e\}$ and $W_1' = \{a, b\}$.
- $B = \mathsf{Attr}_1(\{a, b\}) = \{a, b\}$
- Recursive call on $G \setminus B$ yields $W_0'' = \{c, d, e, f, g\}$ and $W_1'' = \emptyset$, so $W_0 = W_0''$ and $W_1 = B$.

Example



- $A = \mathsf{Attr}_0(f) = \{f, g\}$
- Recursive call on $G \setminus A$ yields $W_0' = \{c, d, e\}$ and $W_1' = \{a, b\}$.
- $B = \mathsf{Attr}_1(\{a, b\}) = \{a, b\}$
- Recursive call on $G \setminus B$ yields $W_0'' = \{c, d, e, f, g\}$ and $W_1'' = \emptyset$, so $W_0 = W_0''$ and $W_1 = B$.

In practice

Complexity: parity games

- Recursive algorithm: $n = |V|$, $m = |E|$, $k =$ number of priorities

  Running time of Parity:

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

  O. Friedmann, Recursive algorithm for parity games requires exponential time. RAIRO - Theor. Inf. and Applic. 45(4): 449-457 (2011)

- Current algorithms (Khoussainov et al., Jurdzinski et al.): quasi-polynomial time, polynomial space

Synthesis

- LTL and CTL* games: **2ExpTime**-c.
- CTL games: **ExpTime**-c.
- GR(1) games (e.g. "infinitely often request $\longrightarrow$ infinitely often grant"): **ExpTime**

Tools

GAVS+ (TU Munich), Acacia+ (U. Bruxelles), BoSy (bounded synthesis, U. Saarbrücken)

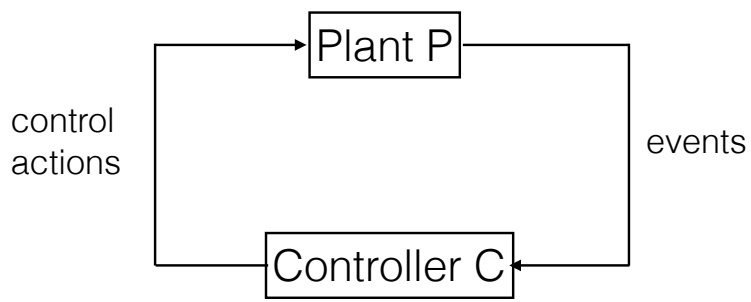
Supervisory control: Ramadge/Wonham

Setting

We are given

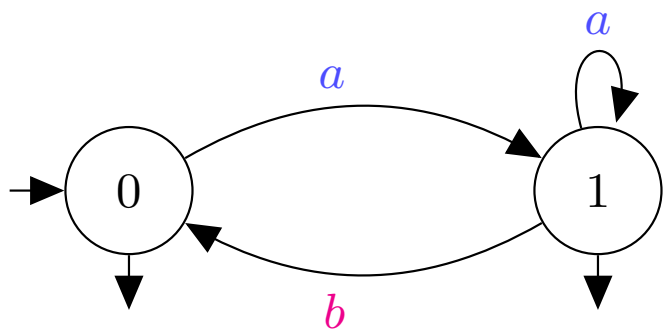
- a "plant" P (deterministic finite automaton),
- a partition of the set Σ of actions into controllable actions from Σ^{sys} and uncontrollable actions from Σ^{env},
- a (regular) specification $Spec$.

Compute controller (supervisor) C that restricts only controllable actions, while satisfying $Spec$.



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:

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

- 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)

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

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

Safety specifications

Given:

- A finite-state automaton (*plant*) $P = \langle Q_P, \Sigma, \longrightarrow_P, q_{0,P}, Q_P \rangle$ over alphabet Σ partitioned into controllable actions Σ^{sys} and uncontrollable actions Σ^{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 Σ partitioned into controllable actions Σ^{sys} and uncontrollable actions Σ^{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:

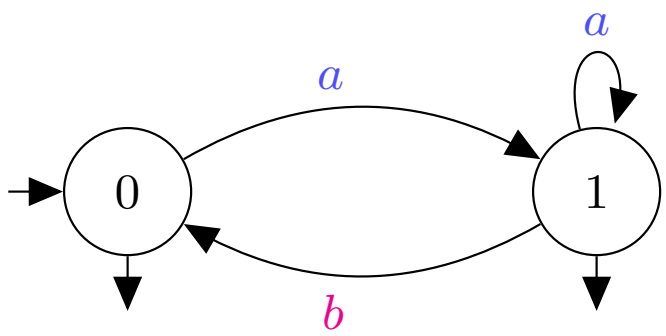
- $P \times C \subseteq S$,
- $w \in C$ and $a \in \Sigma^{env}$ implies $wa \in C$,
- Other possible requirements: C is non-blocking, maximal permissive, ...

Solution

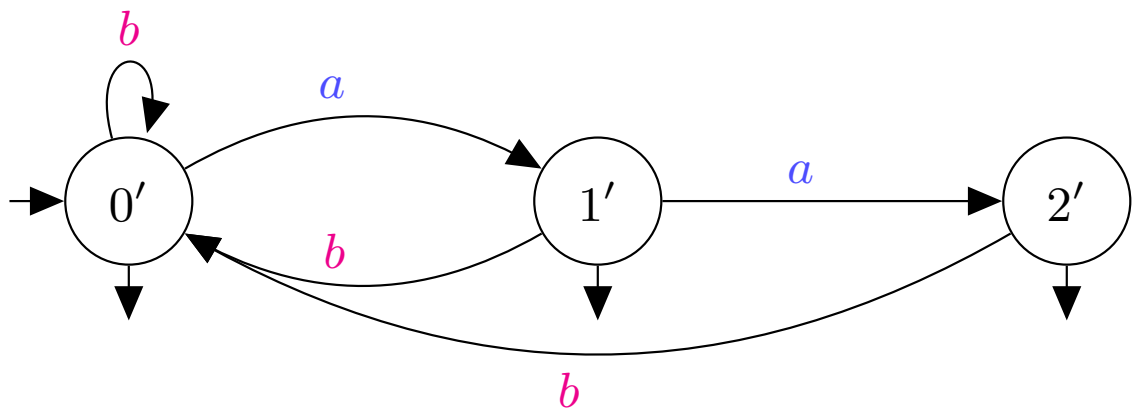
- Build the product $P \times S$.
- Remove all states (q_P, q_S) such that for some $w \in (\Sigma^{env})^*$: $q_P \xrightarrow{w}_P \cdot$ is defined, but $q_S \xrightarrow{w}_S \cdot$ is undefined.
- Add self-loops with Σ^{env}, if necessary. The output is the most permissive controller C.

Example 1

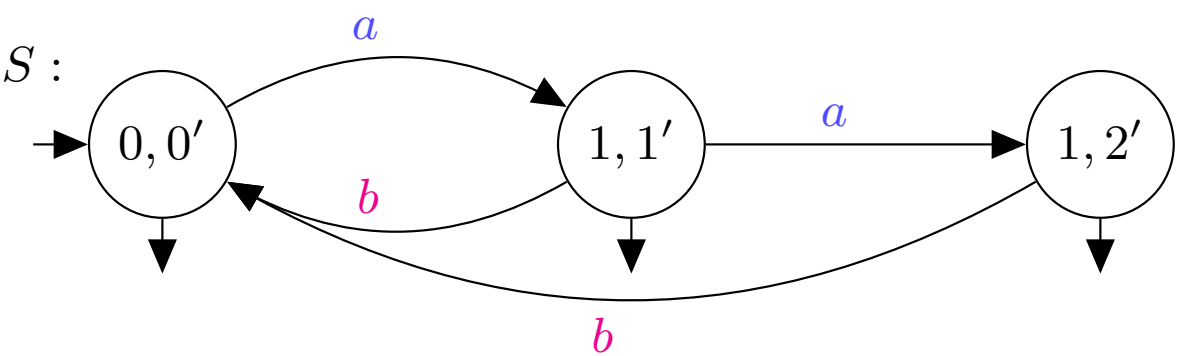
P :



S :

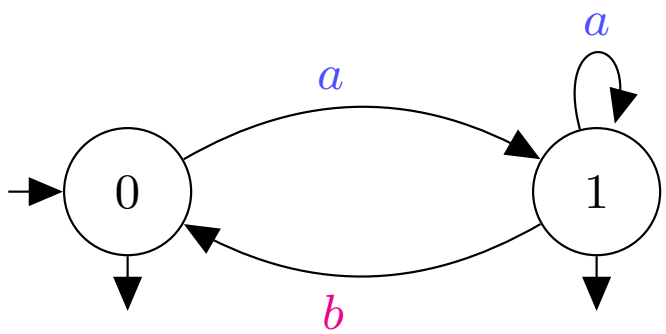


$C = P \times S$:

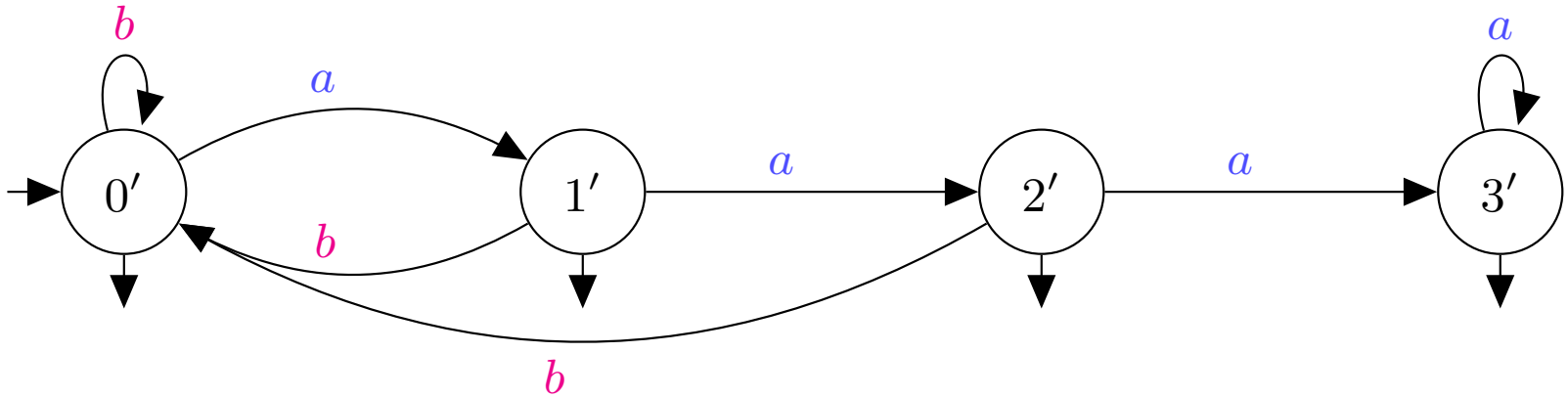


Example 2

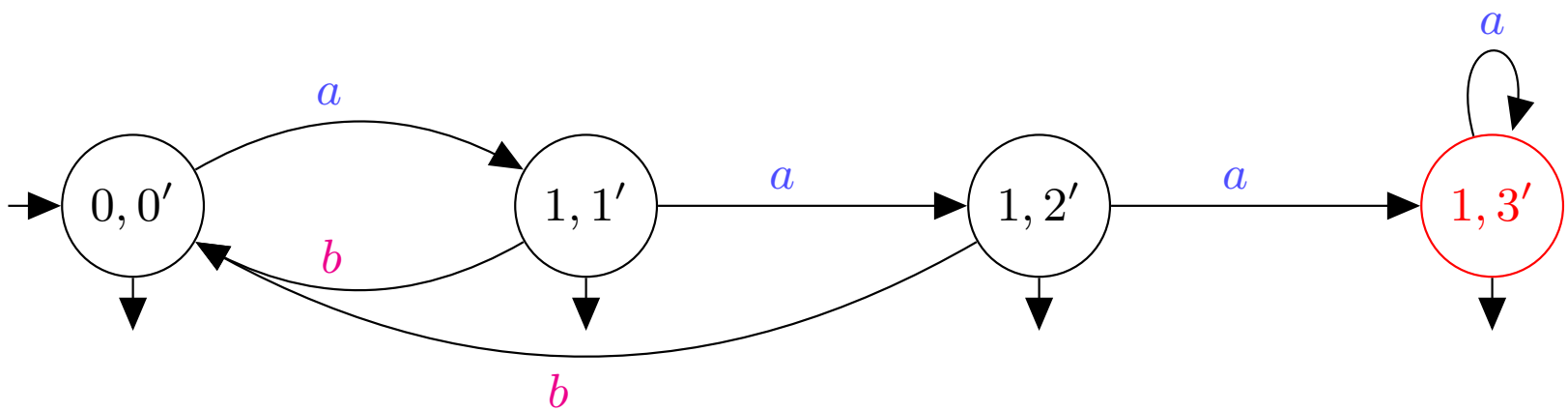
P :



S :

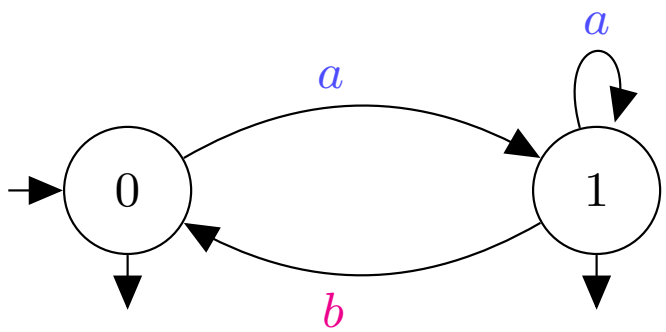


$P \times S$:

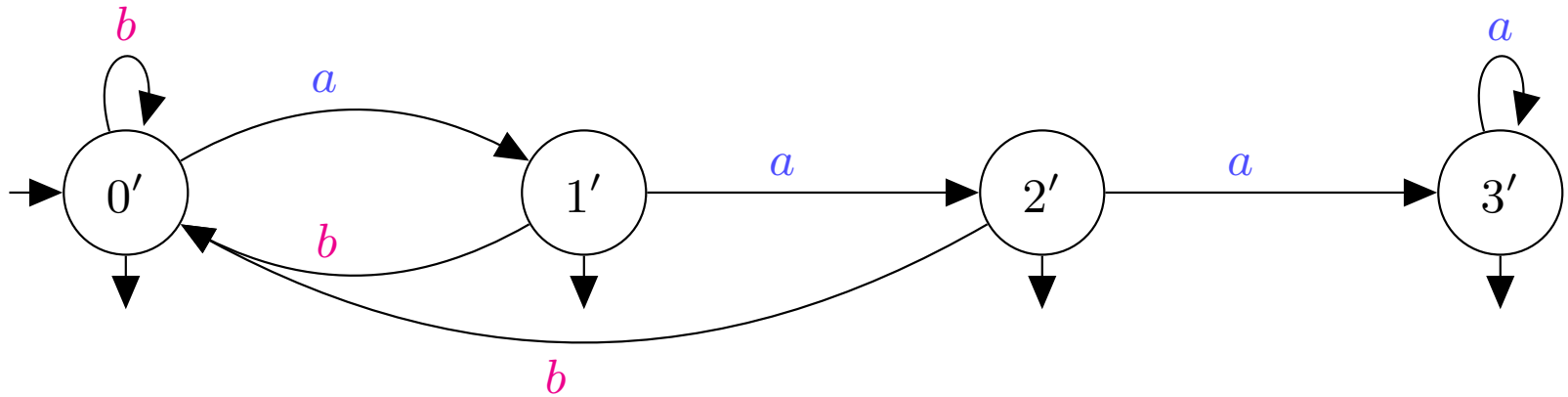


Example 2

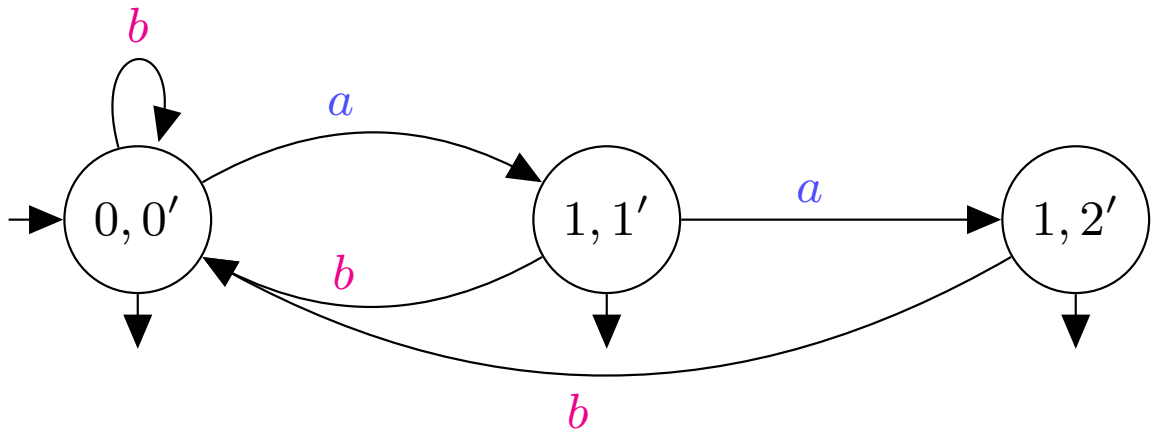
P :



S :



C :



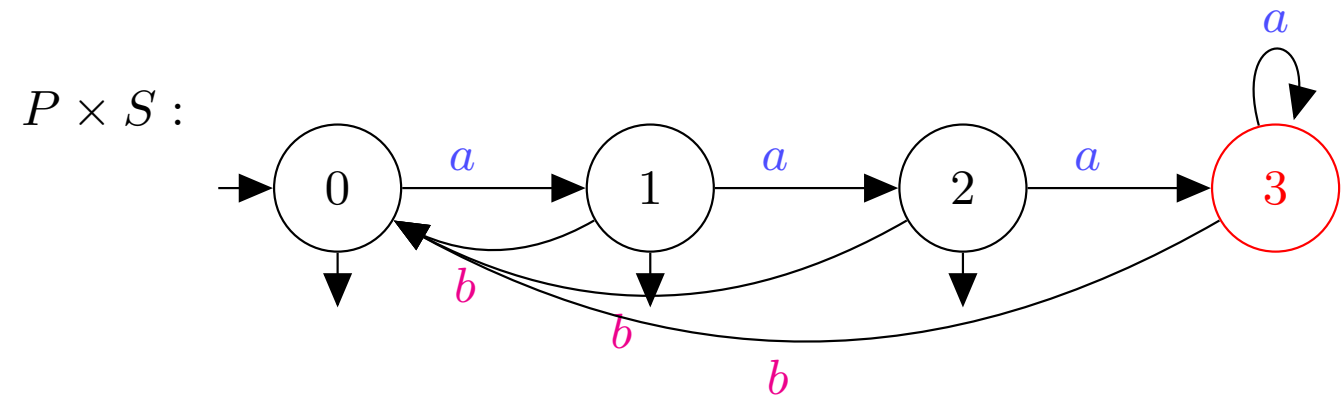
From supervisory control to games

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

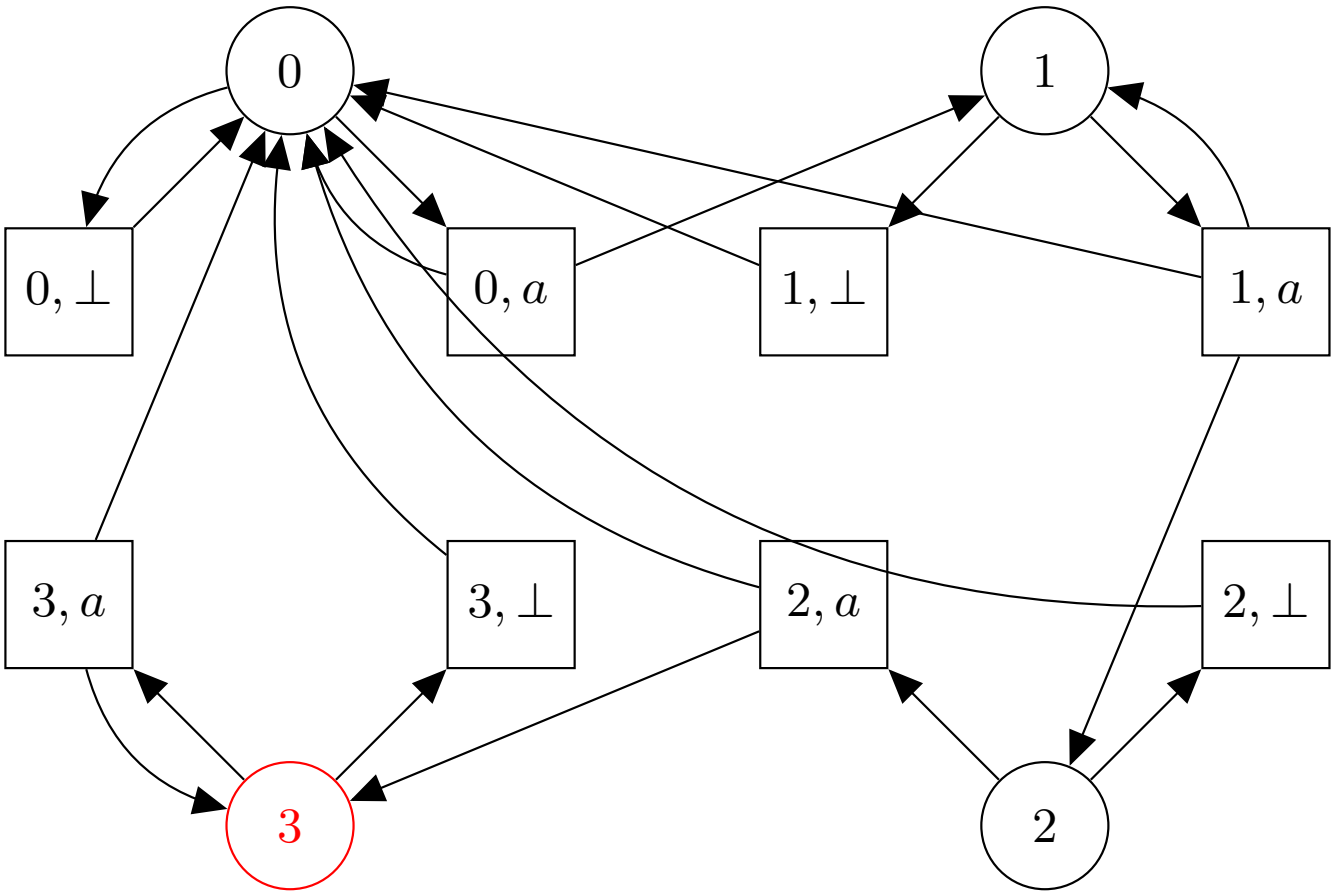
Build game arena $(V_0, V_1, \longrightarrow)$:

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

From supervisory control to games



≤ 2 consecutive a

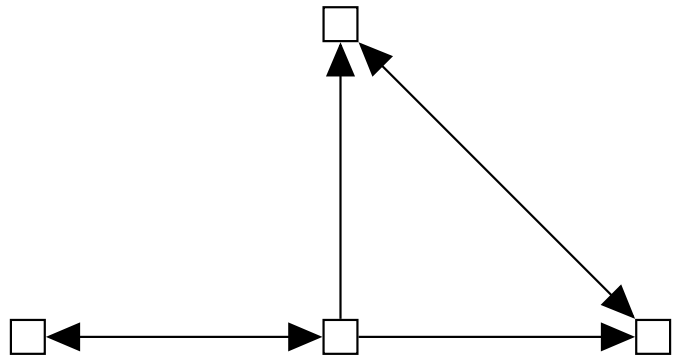


Avoid state 3

II. Distributed synthesis

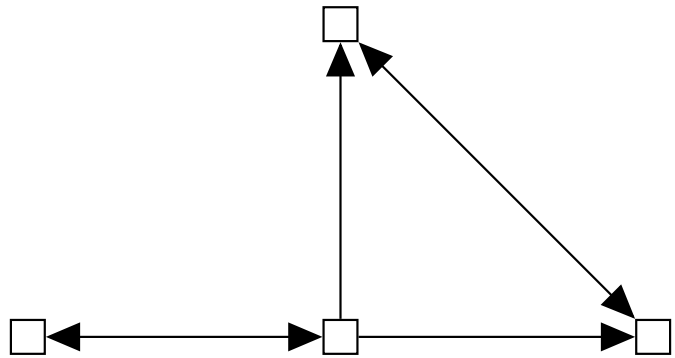
Models

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



Models

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

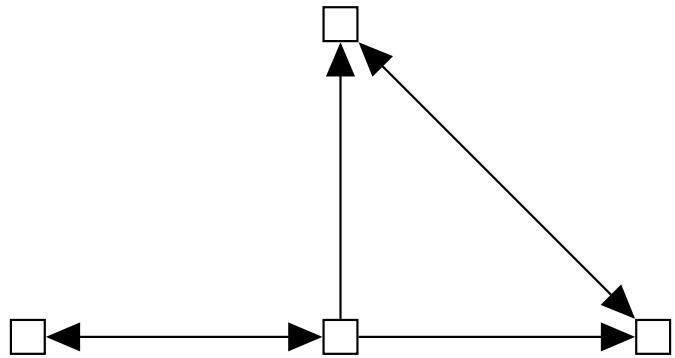


Links as channels

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

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.

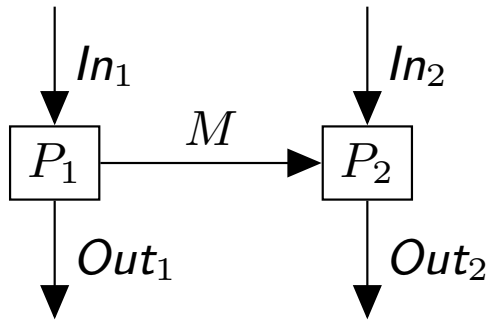
Links as synchronization

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

Pnueli & Rosner, 1990

Synthesis setting

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



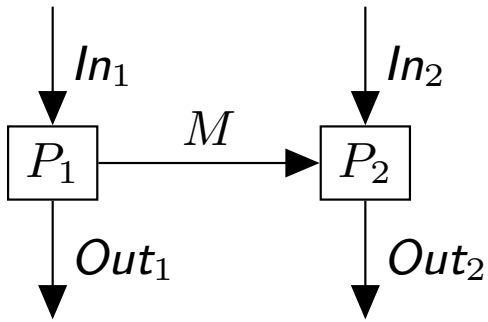
specification $\mathcal{R} \subseteq A^\omega$
$A = In_1 \times In_2 \times M \times Out_1 \times Out_2$

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

Pnueli & Rosner, 1990

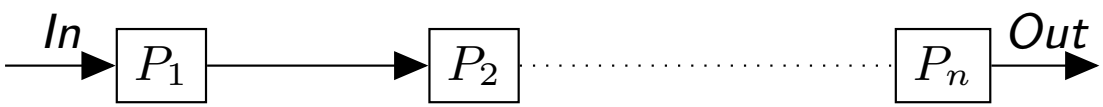
Synthesis setting

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



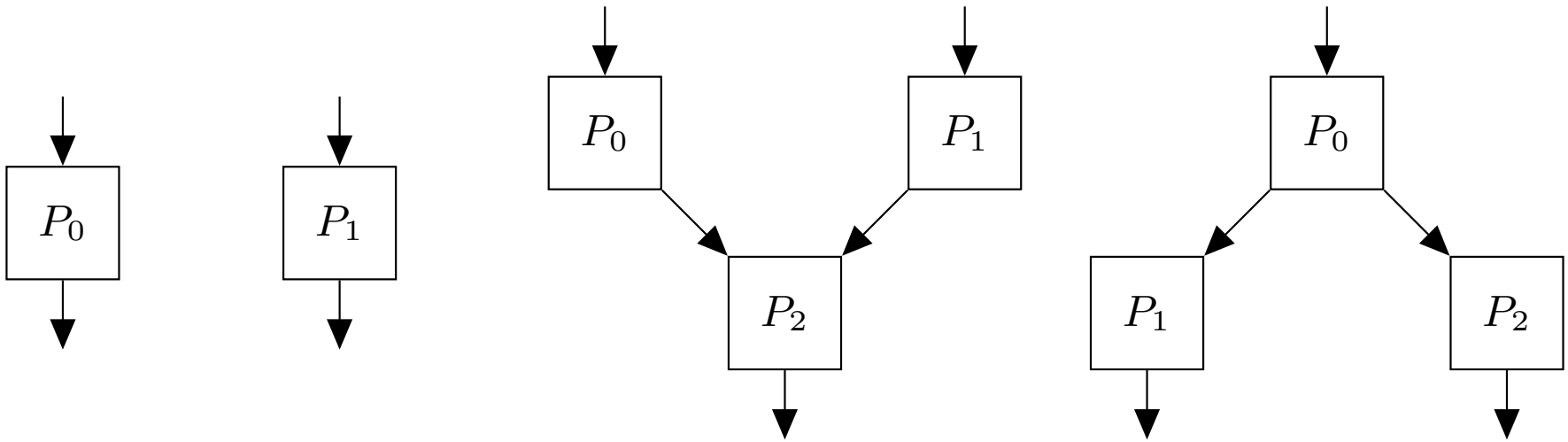
specification $\mathcal{R} \subseteq A^\omega$
$A = In_1 \times In_2 \times M \times Out_1 \times Out_2$

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

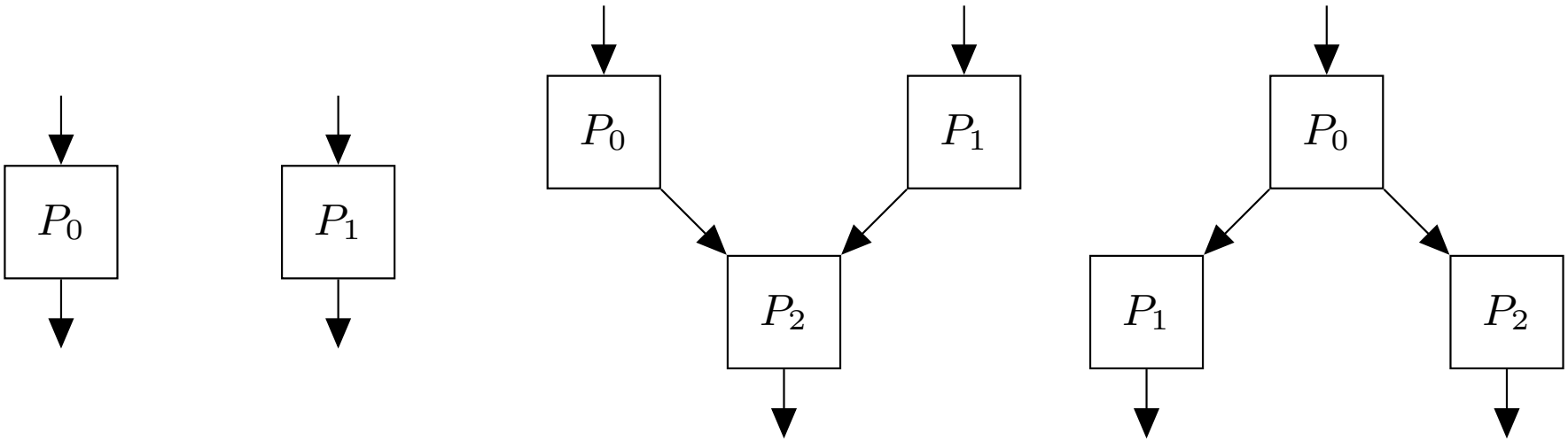


Complexity: non-elementary.

Undecidable architectures



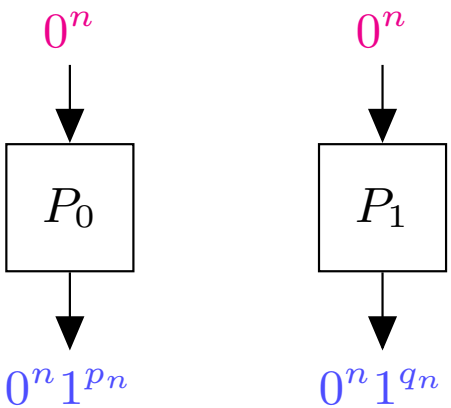
Undecidable architectures



Undecidability: reasons

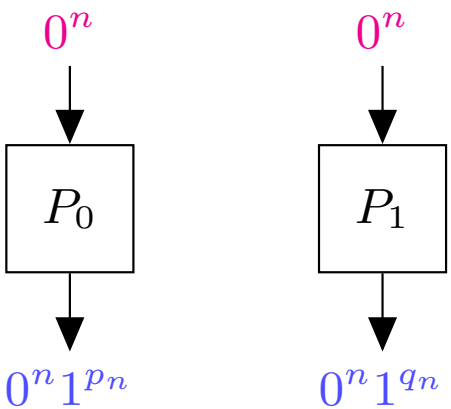
- Processes have different knowledge about the moves of the (global) environment. Left example: P_0, P_1 have incomparable information. Information fork (Finkbeiner/Schewe 2005).
- No compatibility required between architecture and specification.

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?

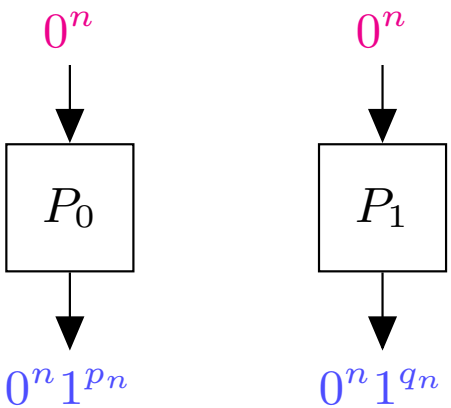
Undecidability



- 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

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

$$
\begin{aligned}
S &= S_1 \cup S_2 \\
S_1 &= \{(0^n, 0^n 1^p, 0^n, 0^n 1^q) : n \geq 0,\ p = q\} \\
S_2 &= \{(0^n, 0^n 1^p, 0^{n+1}, 0^{n+1} 1^q) : n \geq 0,\ q = p + 1\}
\end{aligned}
$$

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

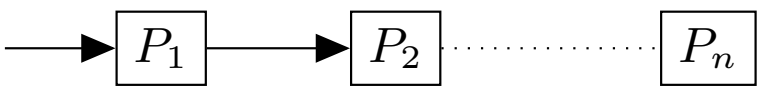
Information fork (Finkbeiner/Schewe 2005)

- 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.

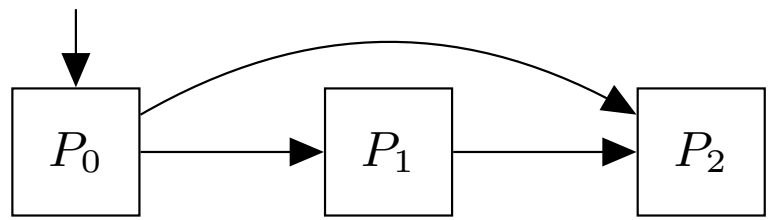
Information fork (Finkbeiner/Schewe 2005)

- 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



P_k is better informed than P_{k+1}.

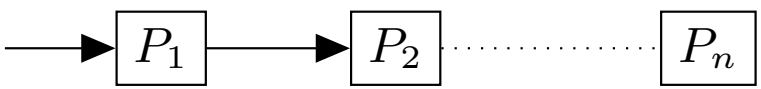


P_1 and P_2 have incomparable information.

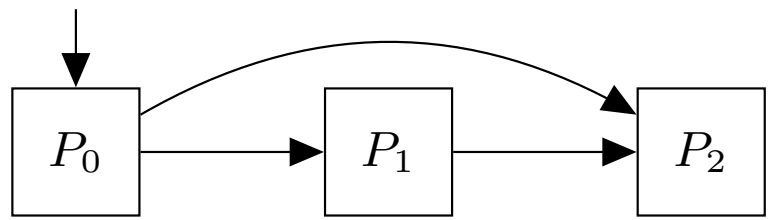
Information fork (Finkbeiner/Schewe 2005)

- 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



P_k is better informed than P_{k+1}.



P_1 and P_2 have incomparable information.

Finkbeiner/Schewe 2005

Synchronous synthesis is decidable iff there is no information fork.

Distributed synthesis: synchronous case

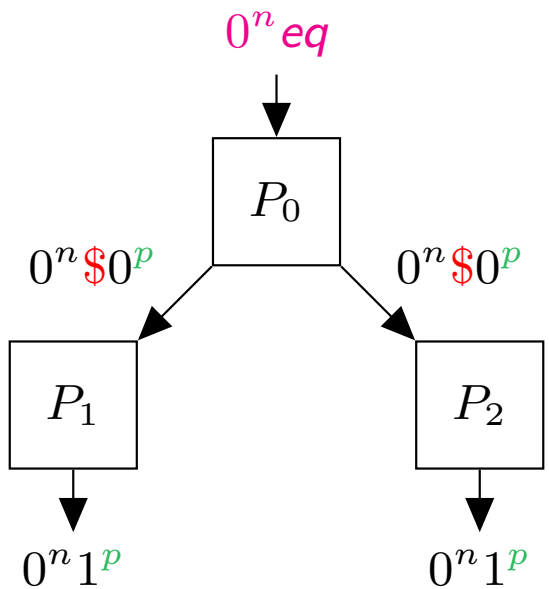
Local specifications (Madhusudan/Thiagarajan 2001)

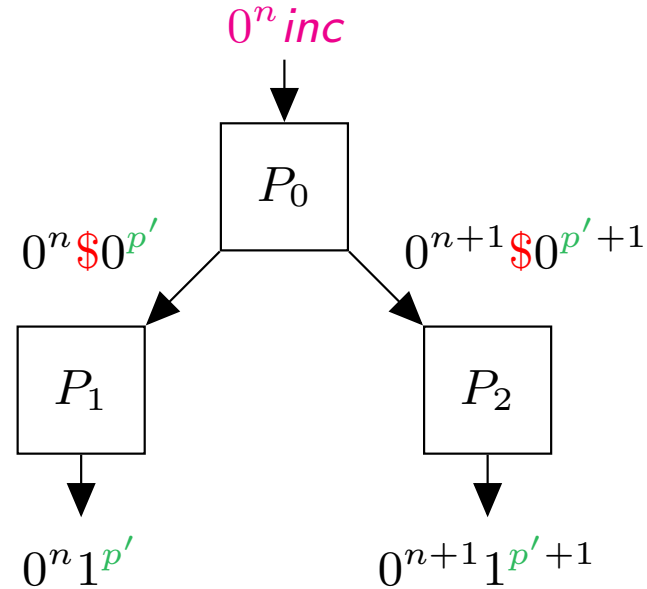
Undecidability for synchronous case due to global specifications? Not only.

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:

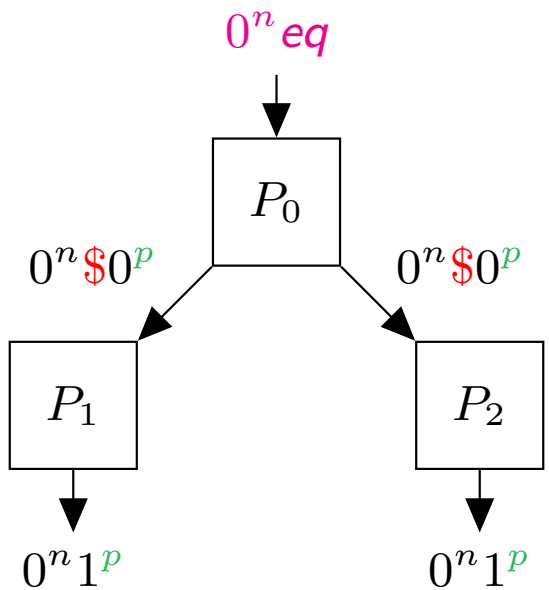


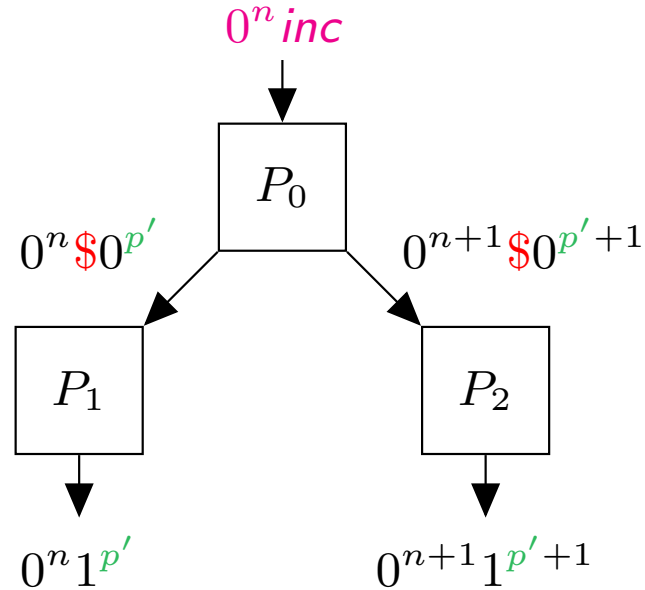


Local specifications (Madhusudan/Thiagarajan 2001)

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:



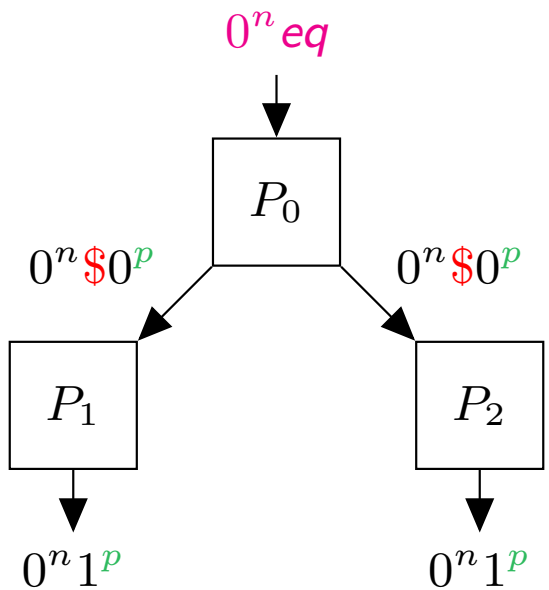


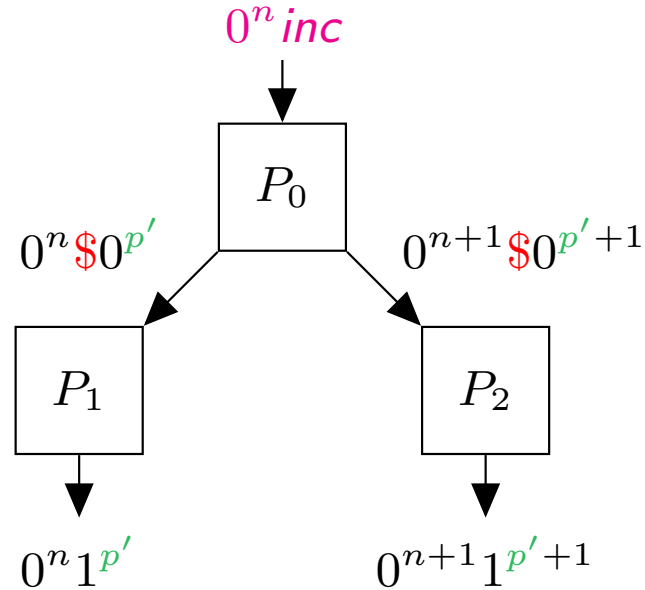
Why is P_0 forced to output $p = p'$ on given n?

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:

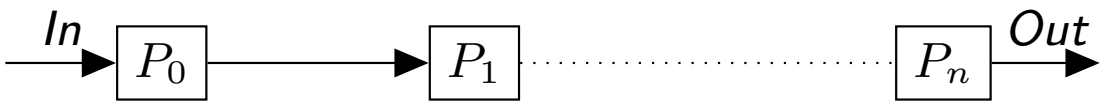




The specification $\{(0^n\$0^p, 0^n\, 1^p) : n, p\}$ forces P_1 to "accept" from P_0 only one value of p, for given n.

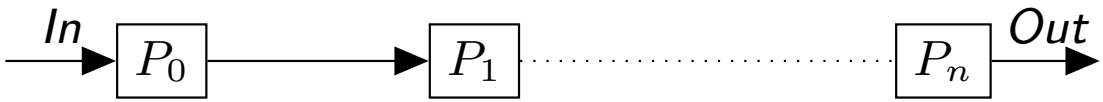
Pnueli/Rosner 1990

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

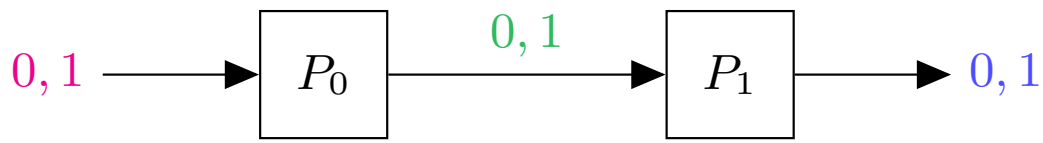


Pnueli/Rosner 1990

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



Proof idea



$$P_0 : \{0,1\}^* \to \{0,1\}$$
$$P_1 : \{0,1\}^* \to \{0,1\}$$
$$\overline{P_0} : \{0,1\}^* \to \{0,1\}^*$$

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

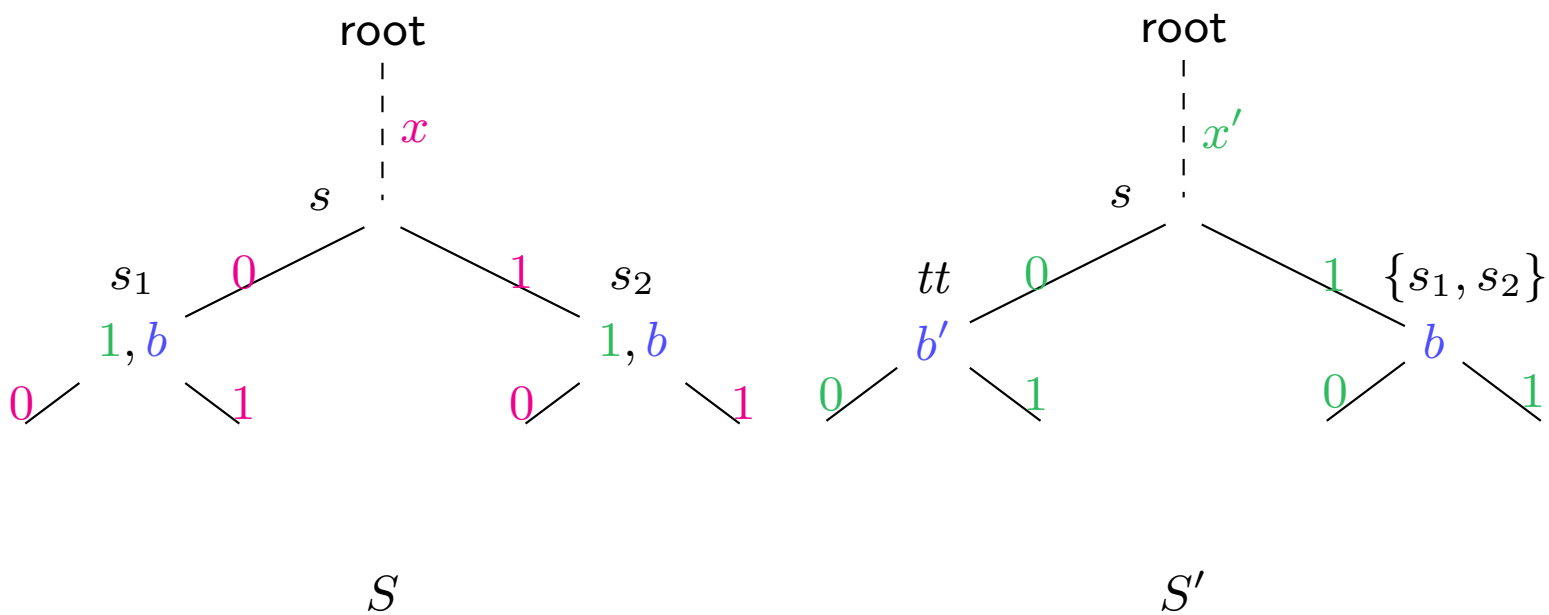
If S is a regular tree language defining a set of functions $\{0,1\}^* \to \{0,1\}$, then there is a regular tree language S' defining a set of functions $\{0,1\}^* \to \{0,1\}$ such that

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

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.



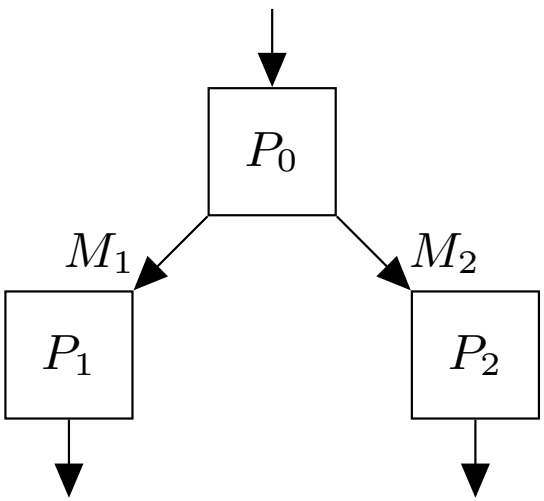
III. Distributed control:
asynchronous case

Synchronous/asynchronous

- Pnueli & Rosner model has synchronous communication: at each step all controllers make a transition. Good for hardware systems.
- 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.



Rem.: Adding information to the messages sent by P_0 to P_1, P_2 (beyond M_1, M_2) makes the synthesis problem decidable here.

Distributed automata

- Finite set of processes $\mathbb{P}$
- Process p has finite set of states S_p.
- Distributed alphabet of actions $\langle \Sigma, \mathsf{dom} : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset) \rangle$

  Action a synchronizes *only* processes in $\mathsf{dom}(a)$:

$$\text{Transition relations } \xrightarrow{a} \subseteq \prod_{p \in \mathsf{dom}(a)} S_p \times \prod_{p \in \mathsf{dom}(a)} S_p$$

Asynchronous model? Which one?

Distributed automata

- Finite set of processes $\mathbb{P}$
- Process p has finite set of states S_p.
- Distributed alphabet of actions $\langle \Sigma, \mathsf{dom} : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset) \rangle$

  Action a synchronizes *only* processes in $\mathsf{dom}(a)$:

  $$\text{Transition relations } \xrightarrow{a} \subseteq \prod_{p \in \mathsf{dom}(a)} S_p \times \prod_{p \in \mathsf{dom}(a)} S_p$$

  $\longrightarrow$ exchange of information among processes in $\mathsf{dom}(a)$ while executing a (rendez-vous synchronization)

Example

Compare-and-swap

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

If the value of x is old, then replace it by new, and return 1; otherwise do nothing with x, and return 0.

Example

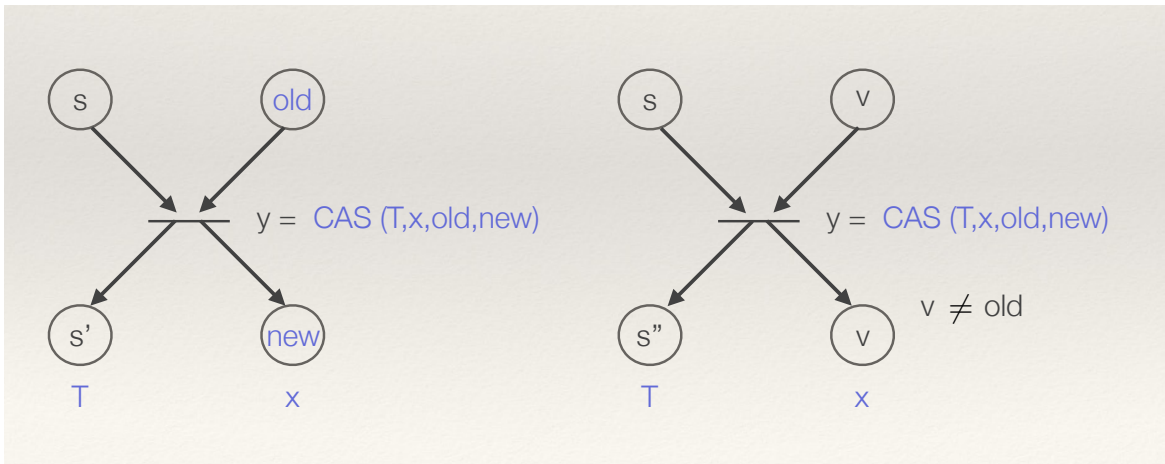
Compare-and-swap

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

If the value of x is old, then replace it by new, and return 1; otherwise do nothing with x, and return 0.

Multi-threaded programs as distributed automata

One process per thread T and per shared variable x.



Exchange of information:
in state s' we have $y = 1$; in state s'' we have $y = 0$.

The language of the automaton

The (regular) language of the product automaton.

The language of the automaton

The (regular) language of the product automaton.

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

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

The language of the automaton

The (regular) language of the product automaton.

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

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

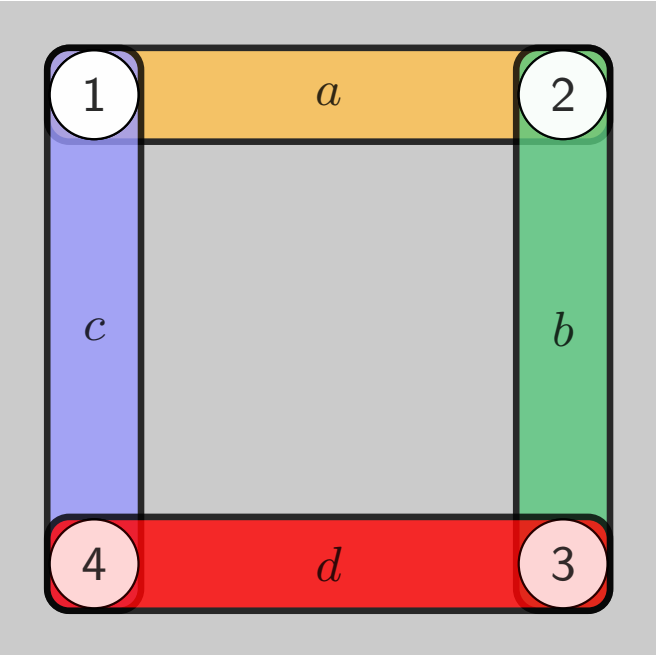
Regular trace languages

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

$$u\,ab\,v \in L \quad \text{iff} \quad u\,ba\,v \in L\,,$$

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

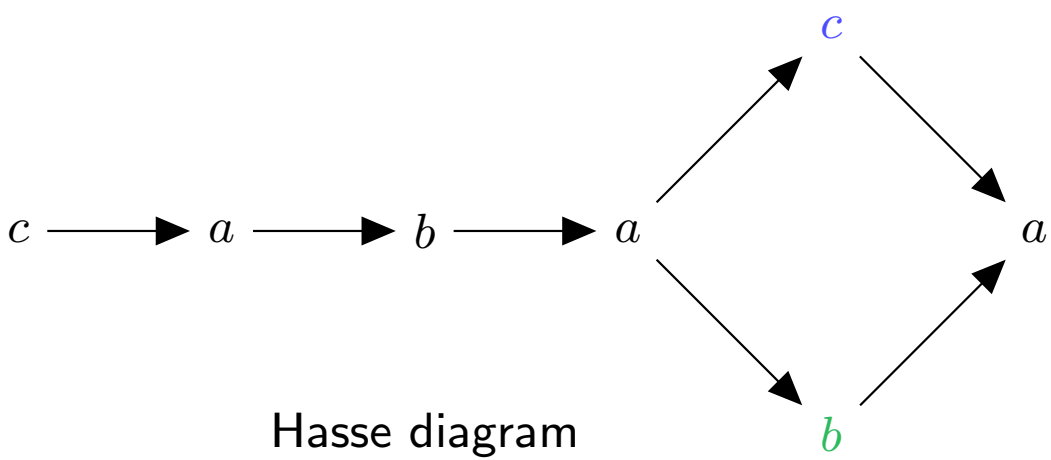
Mazurkiewicz traces



Distributed alphabet

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

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



Hasse diagram

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

Mazurkiewicz trace = labelled partial order

Zielonka's Theorem

[Zielonka 1989]

Construction of deterministic distributed automaton for every regular comm-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]

Example

- SDN (software defined networking): given a network and a specification, synthesize local rules for routing messages such that all behaviours complying with the rules satisfy the specification. For example, depending on failures a node can decide to forward messages to a subset of its neighbors, only.
- Abstract problem:
  Given a distributed automaton $\mathcal{A}$ ("network") and a (regular) specification S, look for another distributed automaton $\mathcal{C}$ ("local rules") such that

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

Warning...

The above problem is undecidable, unless S is comm-closed (Stefanescu, Esparza, M., 2003). For comm-closed S use Zielonka's theorem for constructing equivalent $\mathcal{C}$.

Distributed automata: not that easy to construct

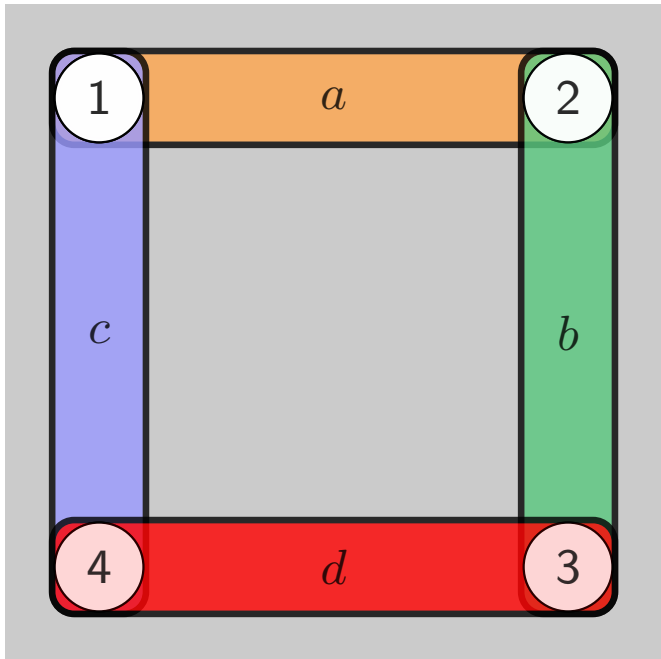
Zielonka (1987)

Every regular, comm-closed language can be recognized by some deterministic, distributed automaton.

Distributed automata: not that easy to construct

Zielonka (1987)

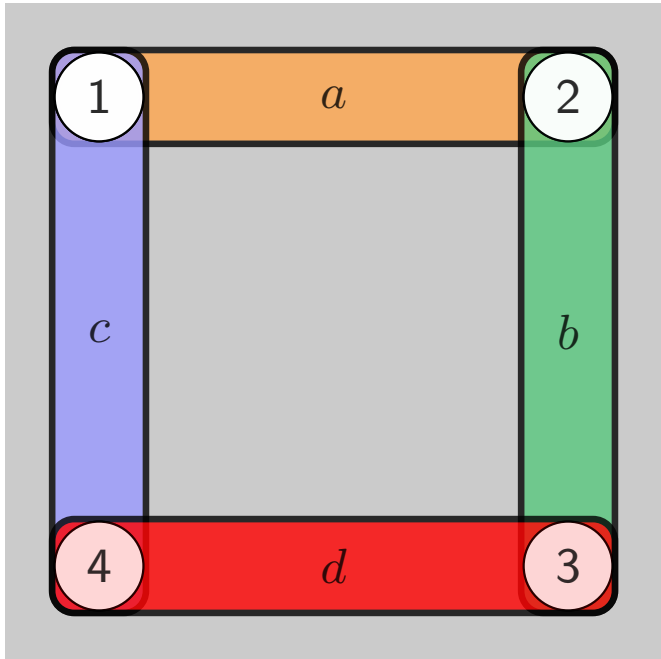
Every regular, comm-closed language can be recognized by some deterministic, distributed automaton.



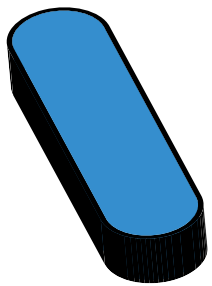
Distributed automata: not that easy to construct

Zielonka (1987)

Every regular, comm-closed language can be recognized by some deterministic, distributed automaton.

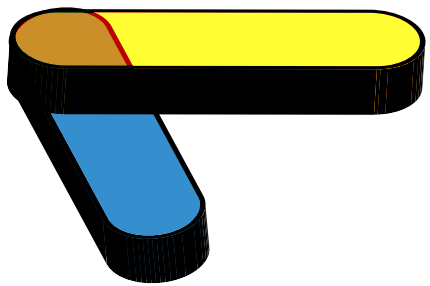


Build a distributed automaton for the trace language $((b + c)(a + d))^*$.
A deterministic finite-state automaton needs only 3 states.



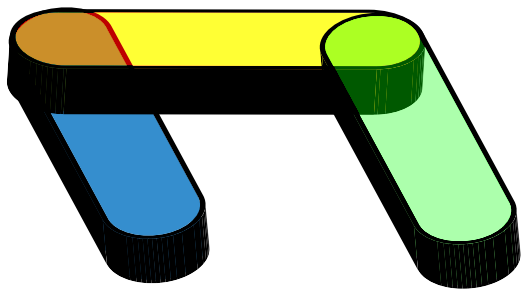
c

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



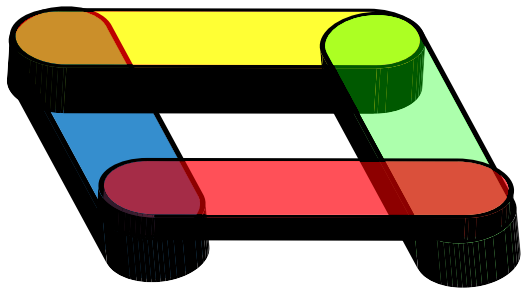
ca

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



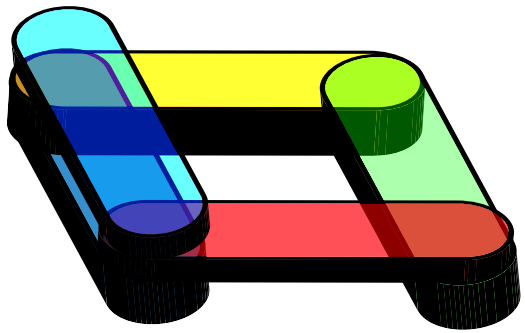
cab

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



$cabd$

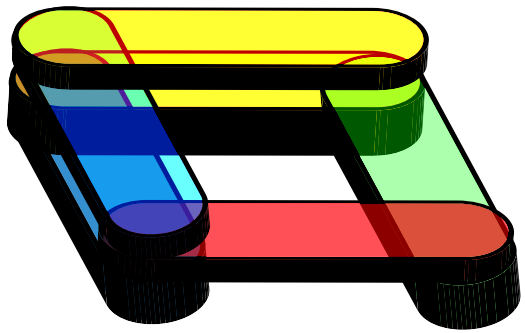
$$((b + c)(a + d))^*$$



$cabdc$

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

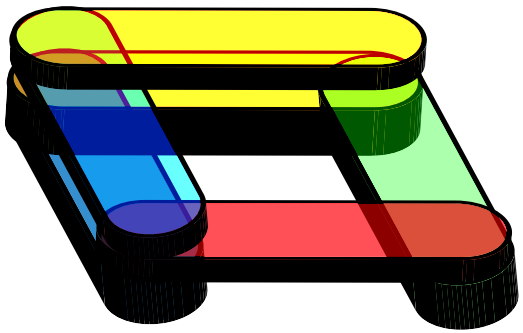
Good



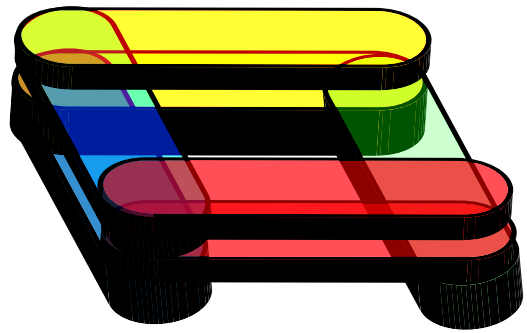
$cabdca$

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

Good



Bad

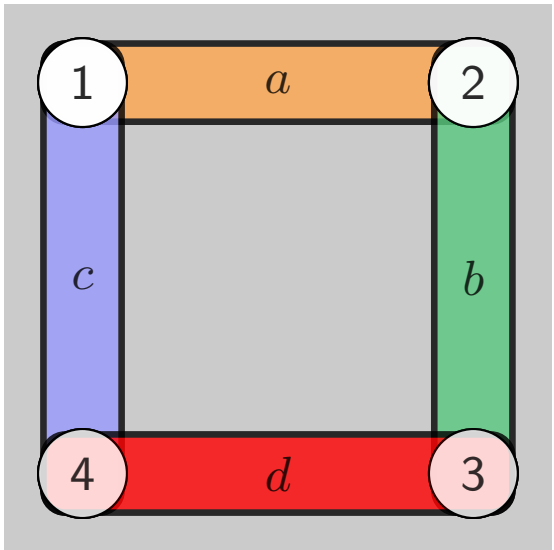


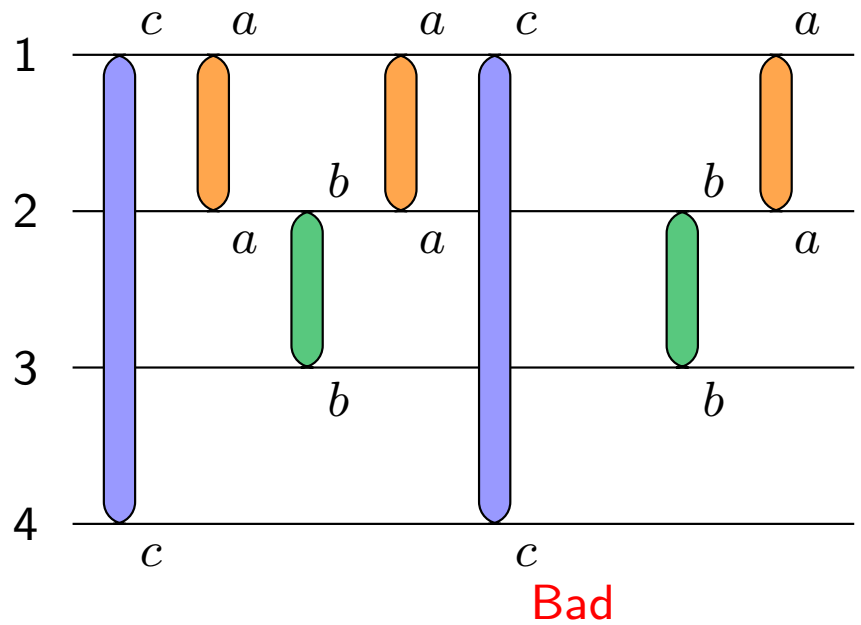
$cabdc\ a\ d$

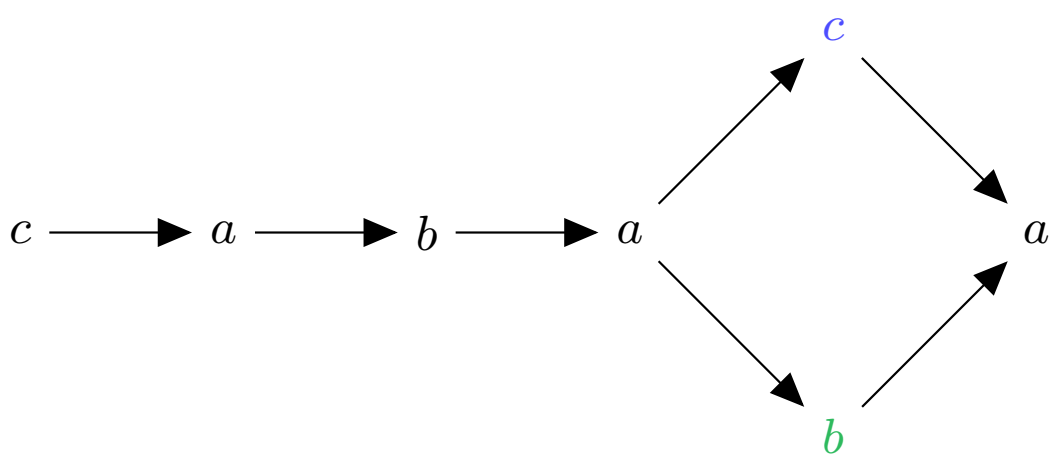
$((b+c)(a+d))^*$

Distributed automata: not that easy to construct

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



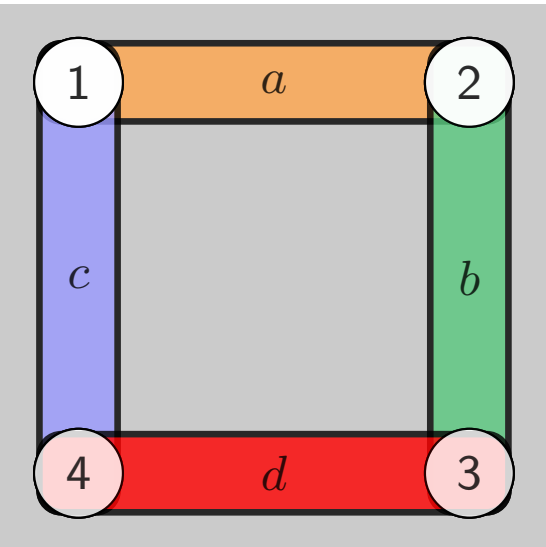


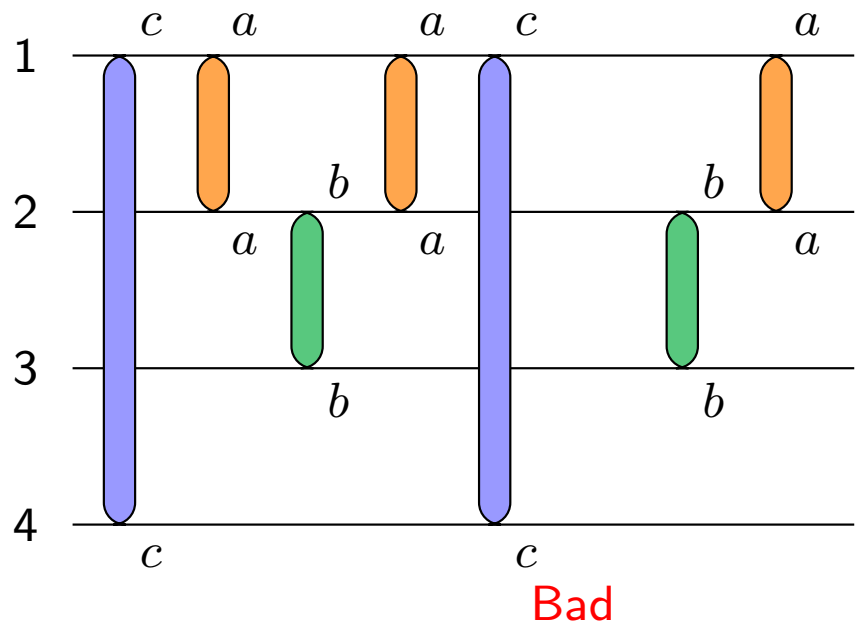


Last a sees bad b, c: both are in the view of processes 1 and 2.

Distributed automata: not that easy to construct

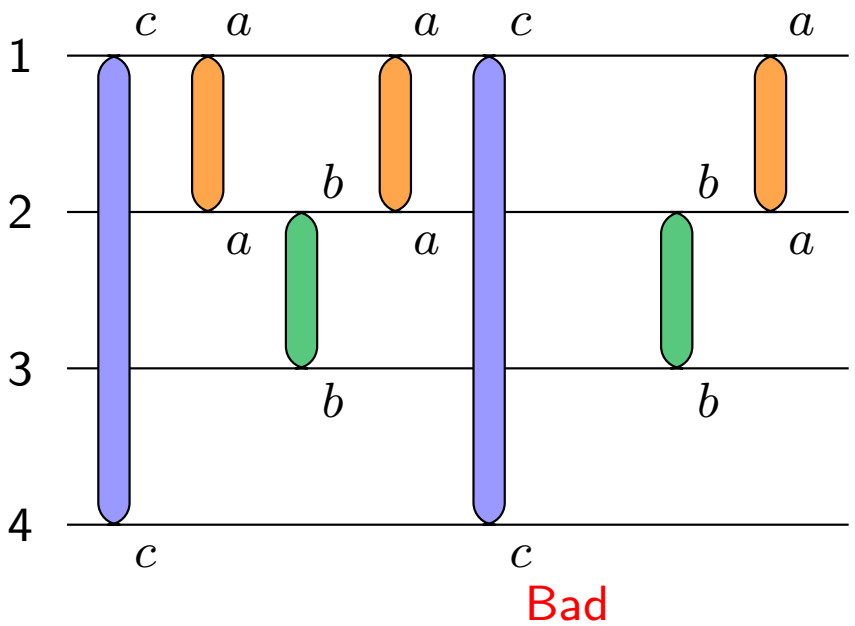
Example: $((b+c)(a+d))^*$

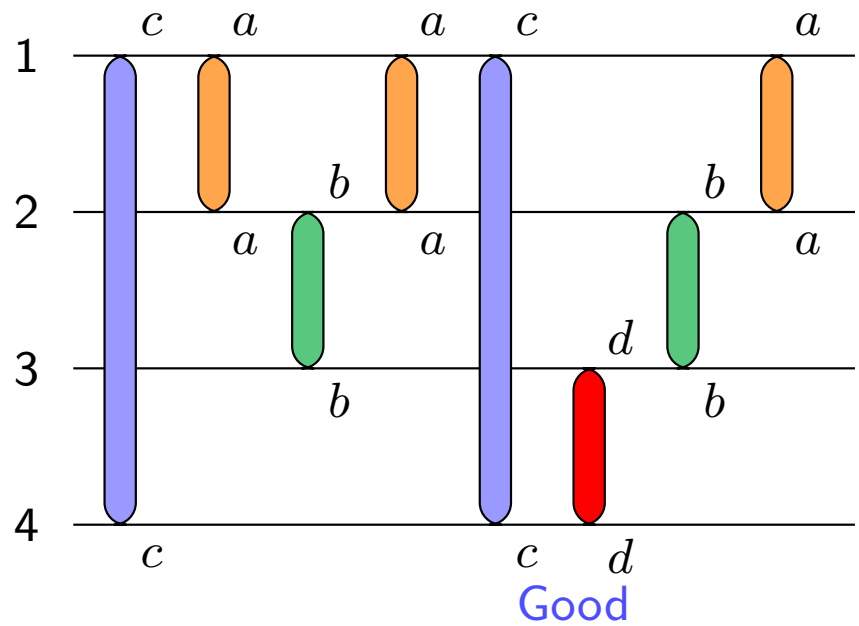




- First idea: each process remembers its *last action*.

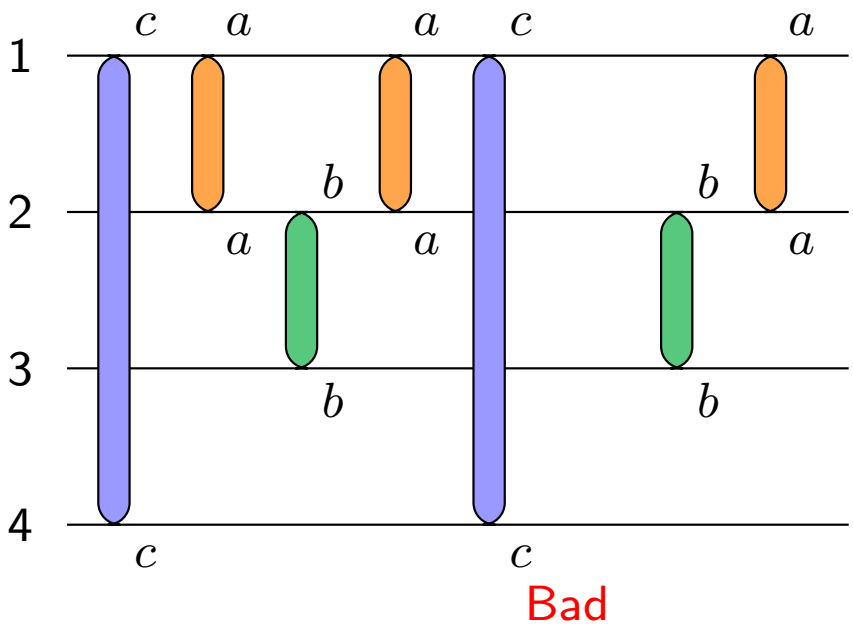
Example: $((b+c)(a+d))^*$

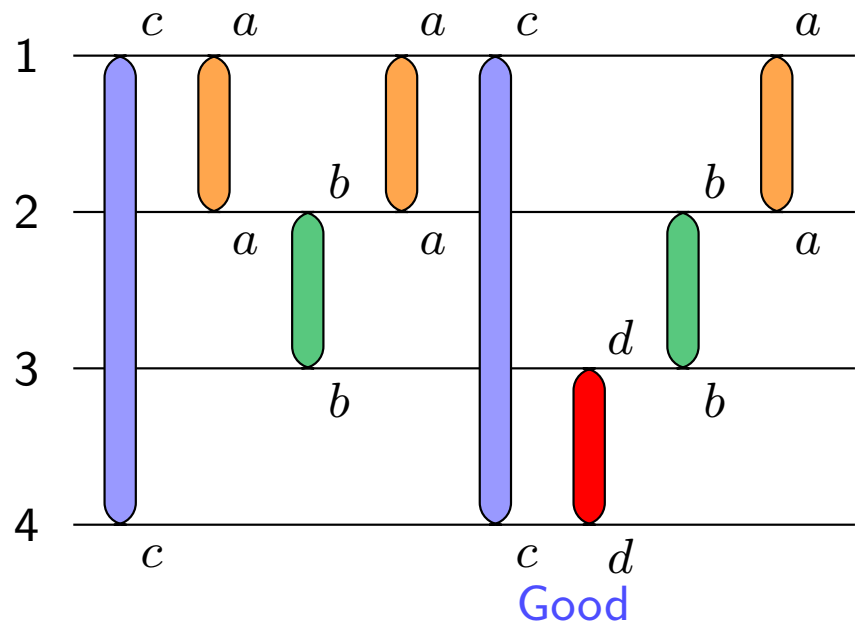




- First idea: each process remembers its *last action*. When synchronizing, processes know about the *previous* action of the other process.
- How can process 1 know that between its first two a's there was a b? By communicating with process 2: the second a is only possible because process 2 did a b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

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

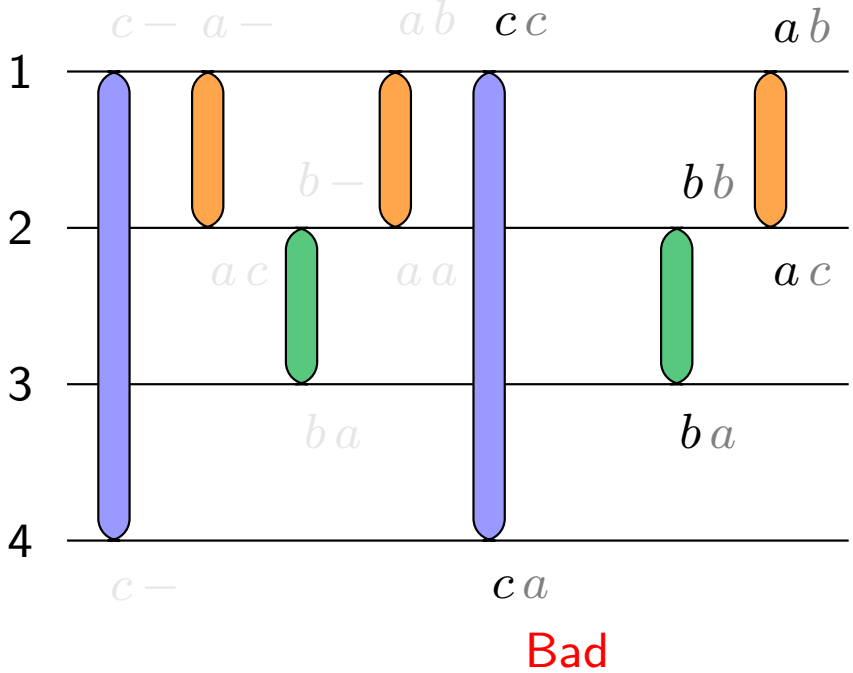




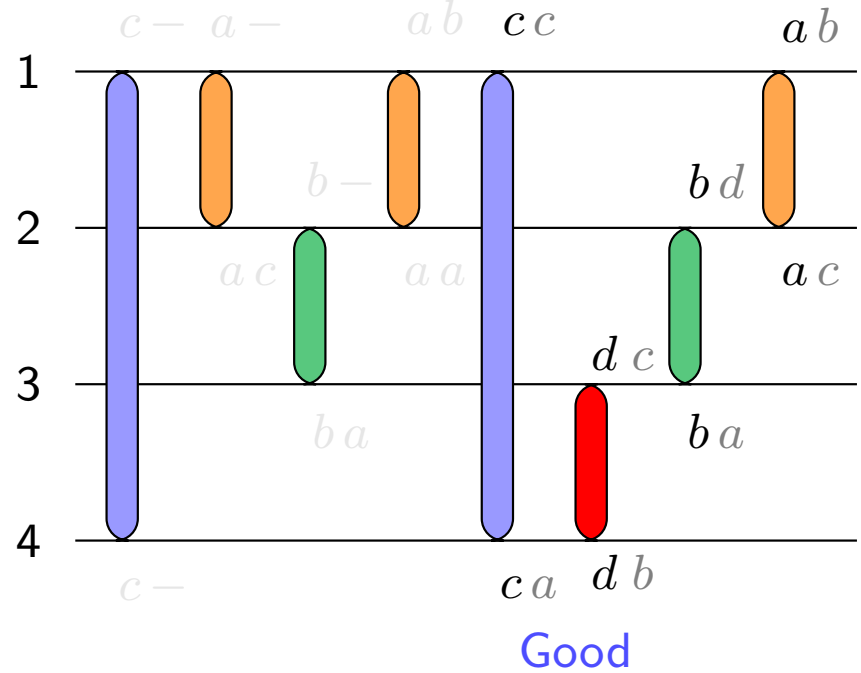
- First idea: each process remembers its *last action*. When synchronizing, processes know about the *previous* action of the other process.
- How can process 1 know that between its first two a's there was a b? By communicating with process 2: the second a is only possible because process 2 did a b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

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



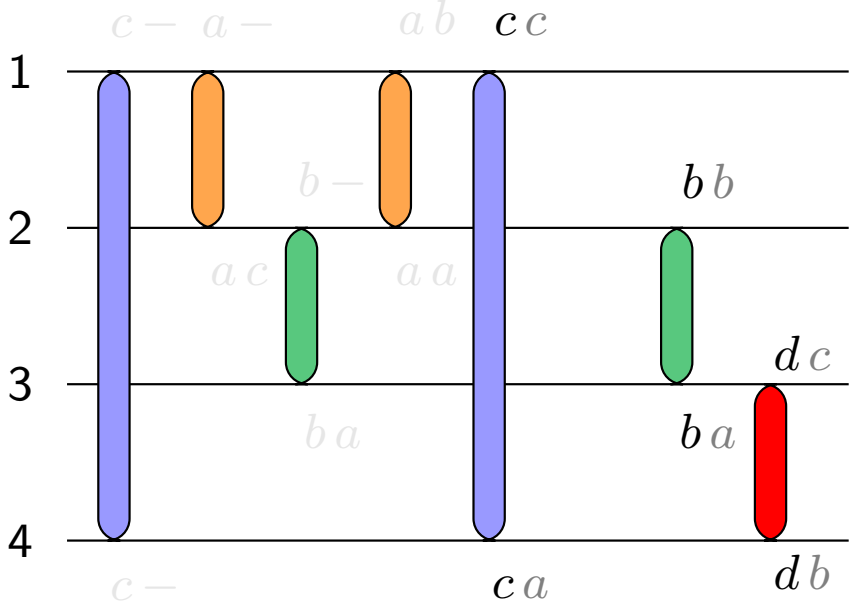
Bad



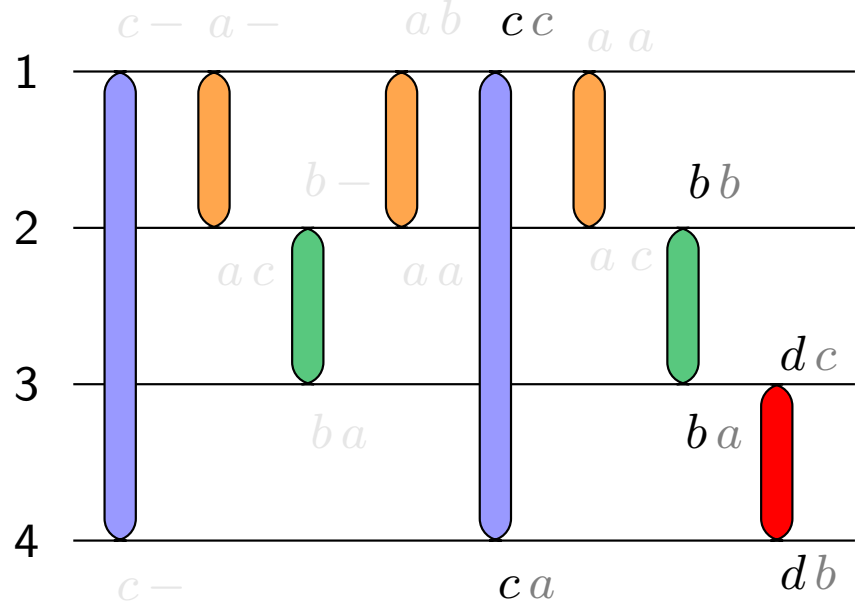
Good

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

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



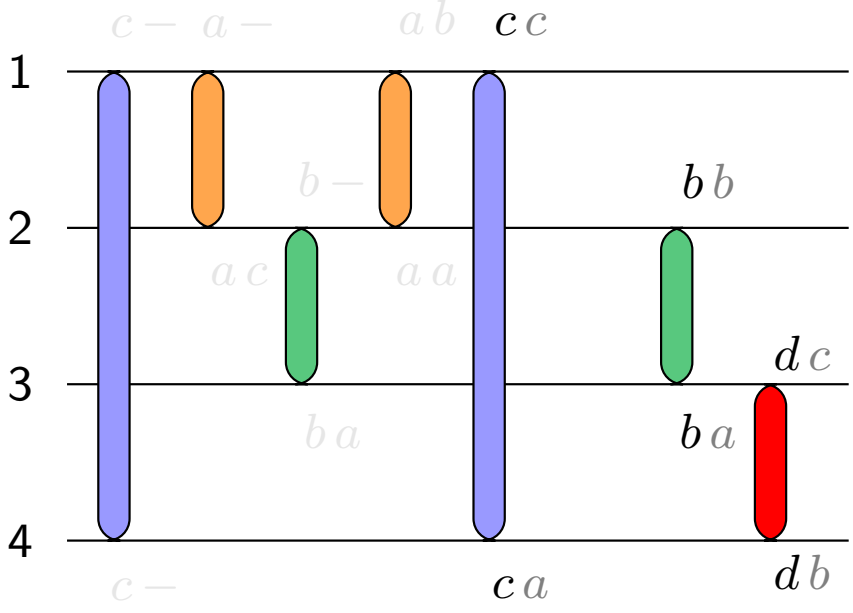
Bad



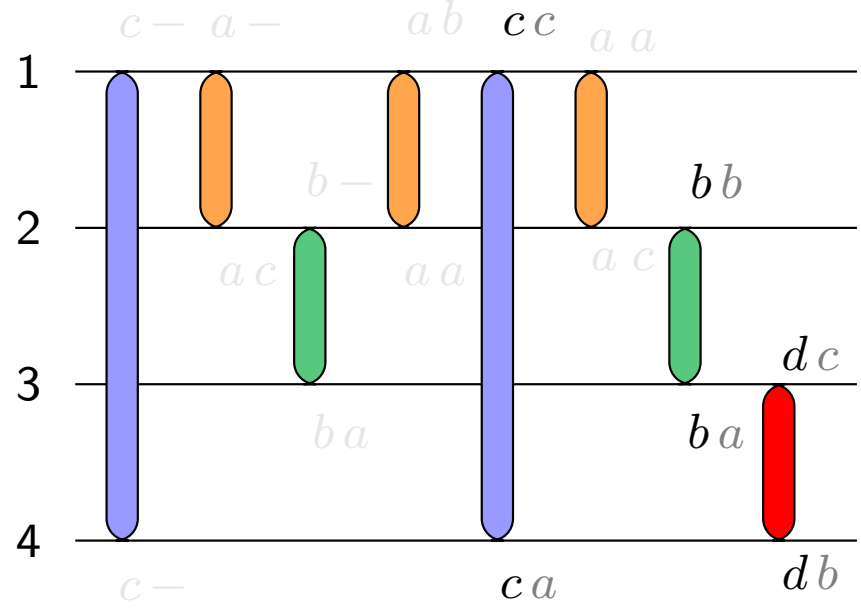
Good

- 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 3 and 4 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 3 and 4 do not have different information upon executing last d.

... the solution here is actually as complicated as the general case: clever finite-memory time-stamping (sort of finite version of Lamport's happened-before relation)

Acyclic case

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

  CG: undirected graph where vertices = processes, and edges between processes that share some action

  Wlog. CG is a tree.

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

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

[S. Krishna, M.: A quadratic construction for Zielonka automata with acyclic communication structure. Theor. Comput. Sci. 503: 109-114 (2013)]

Diamond property

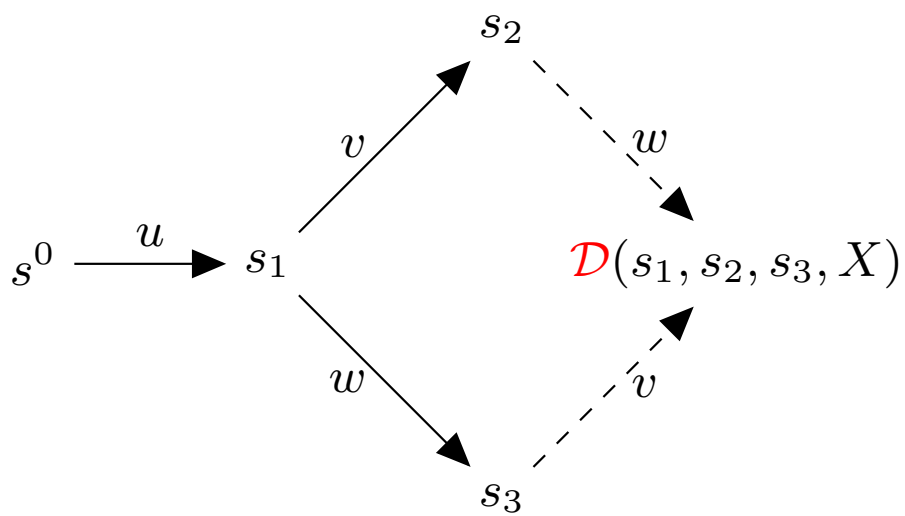
- A finite-state automaton is diamond if for every state s, and every $a, b \in \Sigma$ such that $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$:

$$s \stackrel{ab}{\Longrightarrow} s' \qquad \text{iff} \qquad s \stackrel{ba}{\Longrightarrow} s'$$

- The minimal automaton of a regular, comm-closed language is diamond.

Lemma

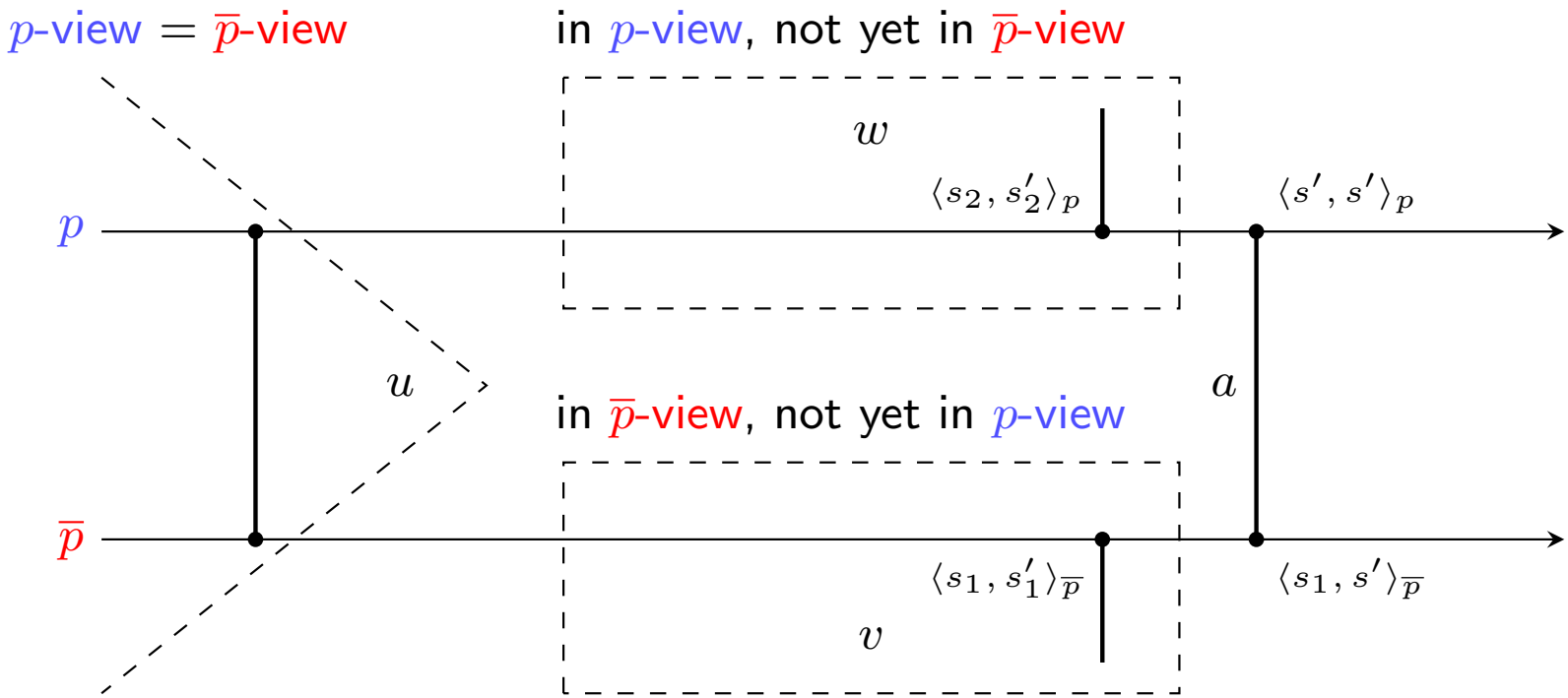
Given a diamond automaton $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$, we can compute a table $\mathcal{D} : S^3 \times 2^{\mathbb{P}} \to S$ such that for all states $s_1, s_2, s_3 \in S$, every set of processes $X \subseteq \mathbb{P}$ and all u, v, w with $\mathsf{dom}(v) \subseteq X$, $\mathsf{dom}(w) \cap X = \emptyset$:



Zielonka's theorem: acyclic case

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.

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



Construction

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

$$\langle s, s' \rangle_p \xrightarrow{a} \langle s, \Delta(s', a) \rangle_p$$

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

$$(\langle s_1, s_1' \rangle_p, \langle s_2, s_2' \rangle_{\overline{p}}) \xrightarrow{a} (\langle s', s' \rangle_p, \langle s_2, s' \rangle_{\overline{p}}),$$

    where $s' = \Delta(s, a)$, $s = \mathcal{D}(s_2, s_1', s_2', X(p))$ and $X(p) \subseteq \mathbb{P}$ is the subtree of CG rooted at p.
- Final states $F \subseteq \prod_{p \in \mathbb{P}} S_p$: apply diamond lemma to combine information up to the root process and determine the final state.

Control problem for distributed automata

SDN example

Given a network and a specification, synthesize local rules for routing messages such that all behaviours complying with the rules satisfy the specification - no matter which nodes or links may fail ($\longrightarrow$ uncontrollable events).

Control problem: statement

Ramadge & Wonham

- Given: distributed automaton P ("plant") with two kinds of actions, controllable actions (or system actions, Σ^{sys}) and uncontrollable actions (or environment actions, Σ^{env}); and a specification S.
- Compute local controllers, one for each process. A local controller must allow every uncontrollable action. It can disallow controllable actions only.

  In essence: we look for a distributed controller $\mathcal{C}$.
- Notice: local controllers exchange information (as in a distributed automaton).

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.

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.

The product $P \times C$ is the usual synchronized product of automata, here process-wise.

Remark

Decidability status: open.

Example 1

Shared bit game

- Two processes P_0, P_1, that do not communicate.
- P_i receives an uncontrolable bit u_i and has to produce a controlable bit c_i.
- Winning condition: $c_1 = u_0$ or $c_0 = u_1$.

Example 1

Shared bit game

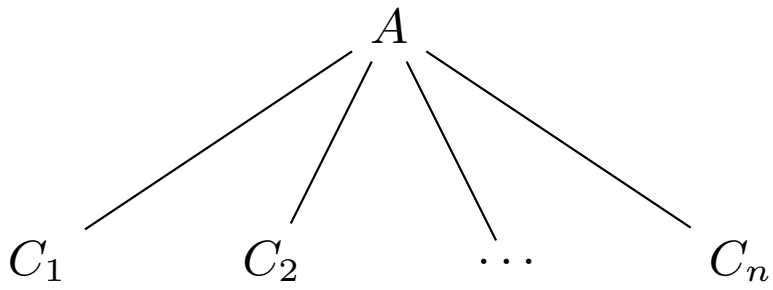
- Two processes P_0, P_1, that do not communicate.
- P_i receives an uncontrolable bit u_i and has to produce a controlable bit c_i.
- Winning condition: $c_1 = u_0$ or $c_0 = u_1$.

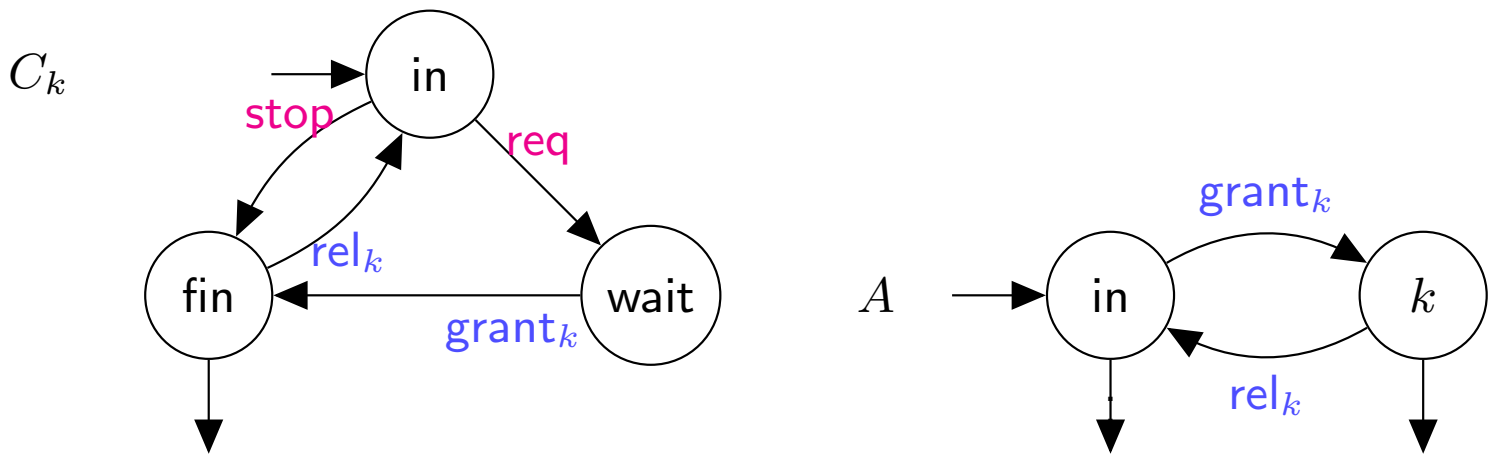
How to win

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

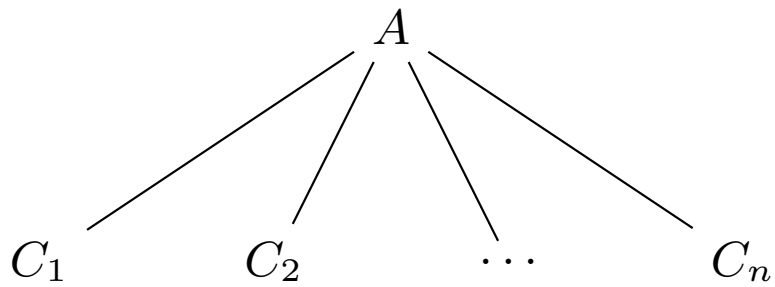


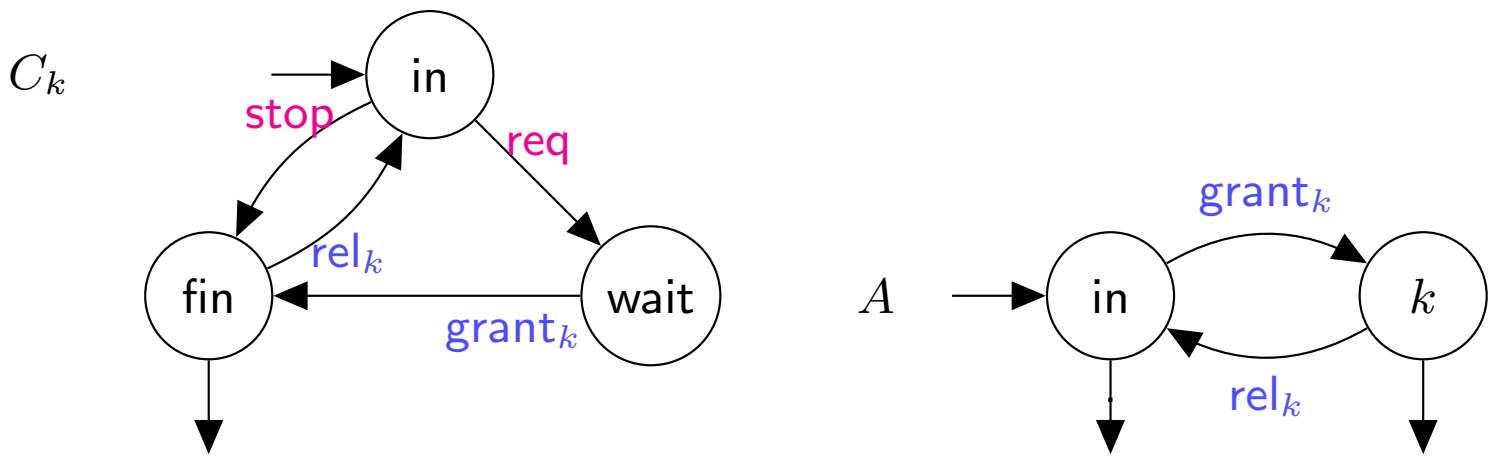


req,stop: local uncontrolable actions, grant,rel: shared controlable actions.

Example 2

Arbiter game



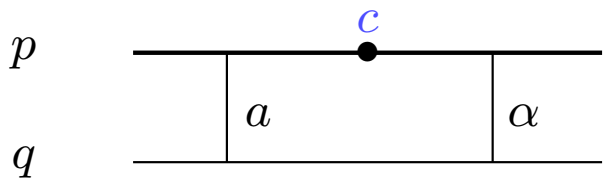


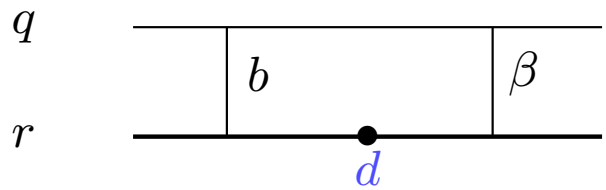
req,stop: local uncontrolable actions, grant,rel: shared controlable actions.

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

Example 3

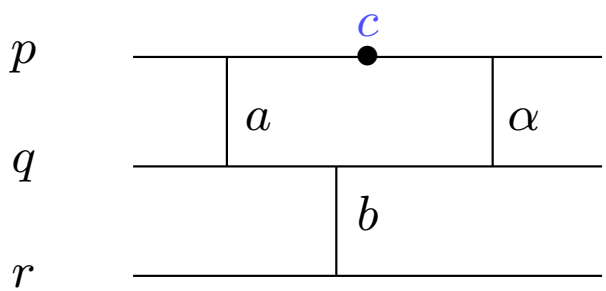
- Plant:

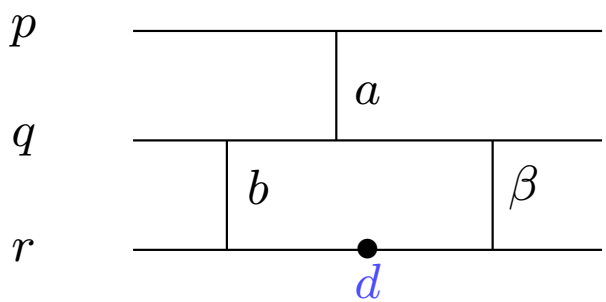




Process q: $(ab + ba)(\alpha + \beta)$

- Controllable actions: c and d
- Specification:

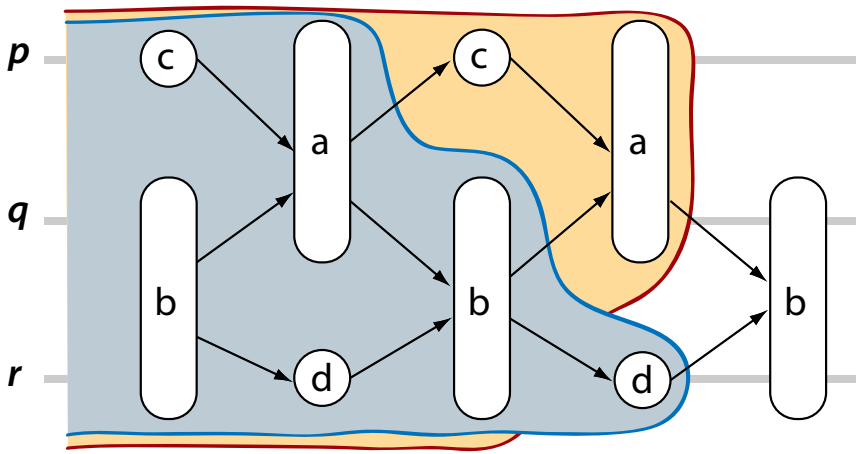




Plant is controllable: through communication with q, both processes p and r can learn about the order between a and b. If e.g. a preceded b, then p's controller allows c and q's controller disallows d.

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

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



- $\Sigma_p^{sys} = \Sigma^{sys} \cap \{a : p \in \mathrm{dom}(a)\}$: set of controlable actions involving p
- Views_p: set of p-views of process p (causal memory)

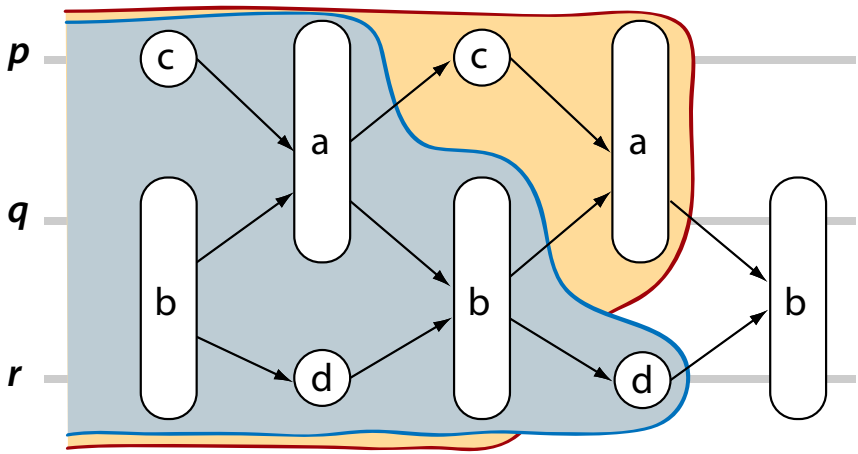
Before last b:

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

Control for distributed automata: game version

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

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



- $\Sigma_p^{sys} = \Sigma^{sys} \cap \{a : p \in \mathrm{dom}(a)\}$: set of controlable actions involving p
- Views_p: set of p-views of process p (causal memory)

Before last b:

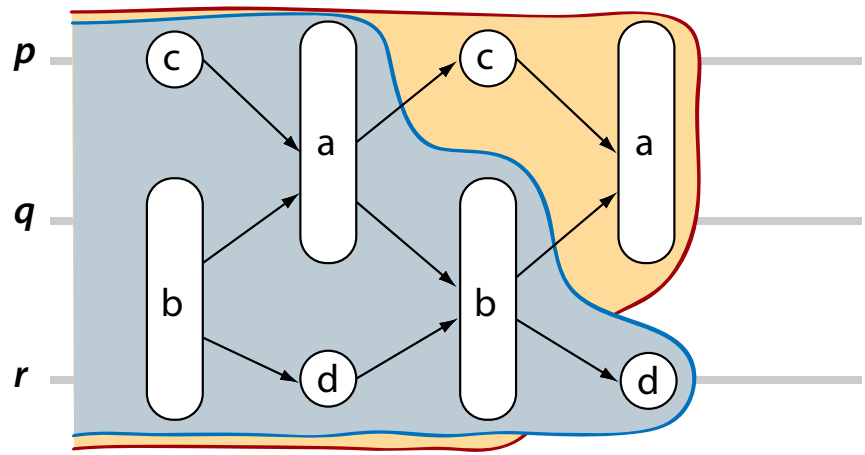
$$\mathit{view}_p = \mathit{view}_q = [cbadcba], \quad \mathit{view}_r = [cbadbd]$$

σ-plays on $\mathcal{A}$

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

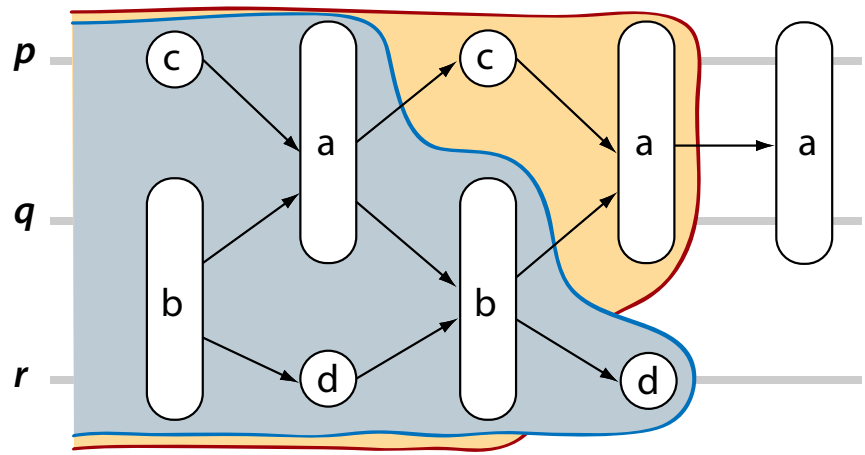
Example

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



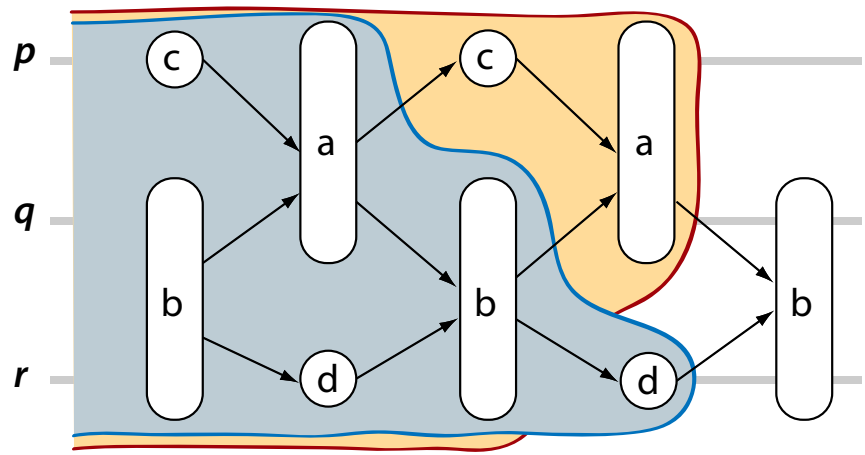
Example

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



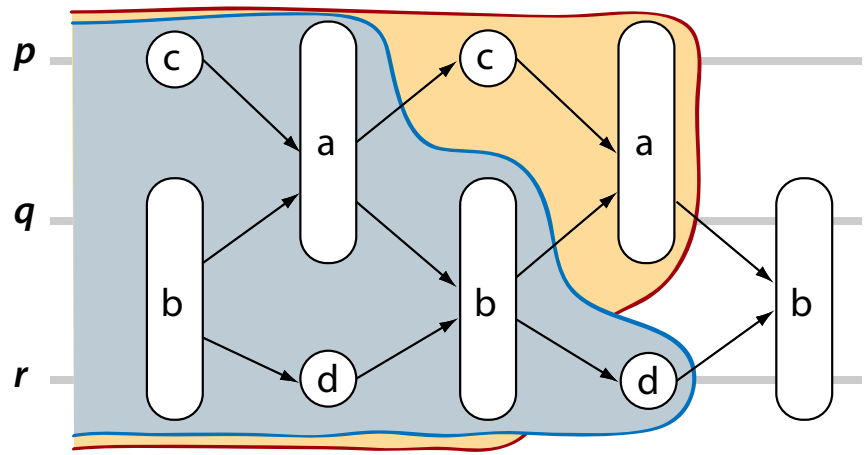
Example

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



Example

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



Winning conditions

- Most general: ω-regular, comm-closed $\rightsquigarrow$ 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.
- Warning: distributed control problem gets undecidable if...
  - controllers do not exchange full information (loosely cooperating), or
  - their strategies are based only on local histories, or
  - the specification is not comm-closed.

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.

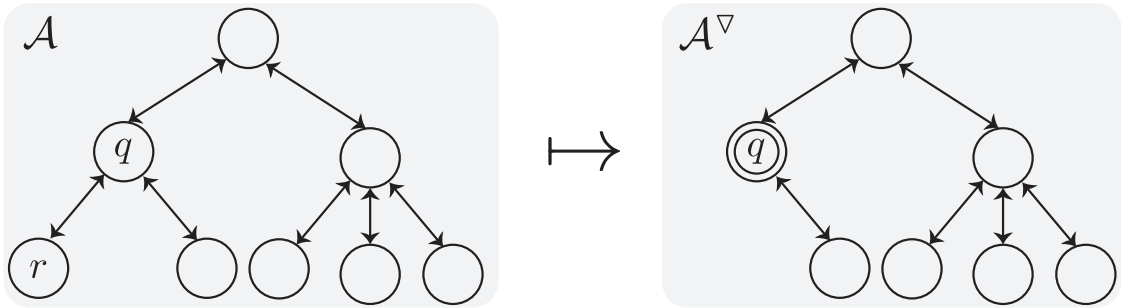
Theorem (M. & Walukiewicz, 2014)

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.

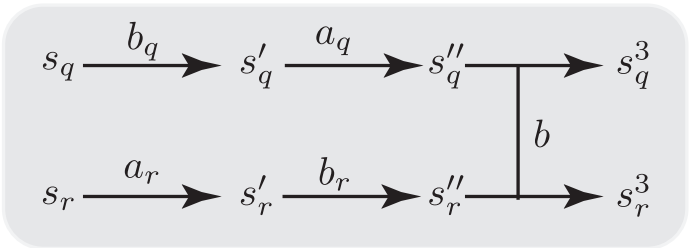
  Argument: if $\mathcal{A}$ can be controlled to satisfy a local parity specification, then the controller doesn't need to visit twice the same r-state during an r-local run.

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

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

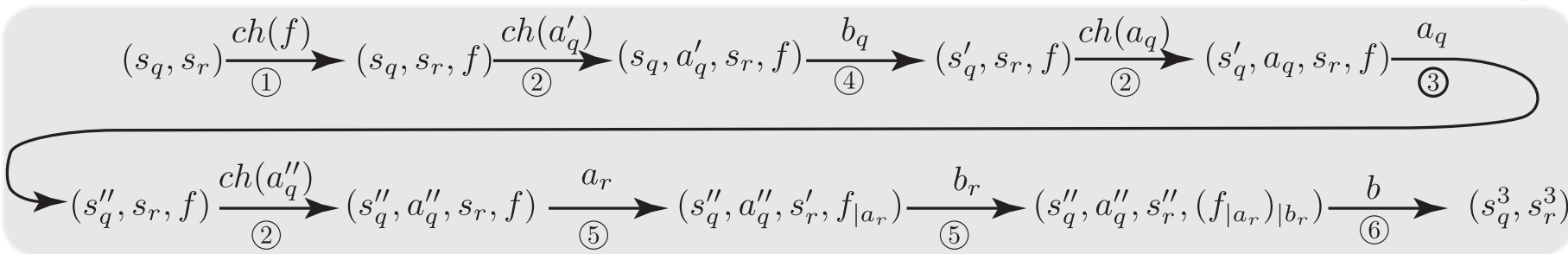
Proof

Simulating process r by process q



In $\mathcal{A}_q$ and $\mathcal{A}_r$.

In $\mathcal{A}_q^{\triangledown}$.



- a_q, a'_q controllable q-actions, b_q uncontrollable q-action
- $f : (S_r)^* \to \Sigma_r^{sys}$ is r-local strategy
- only q-actions $ch(\cdot)$ are controllable in $\mathcal{A}^{\triangledown}$
- $ch(f)$: 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 $r_1, \ldots, r_k$.

Rem.

Recall: only local actions can be controllable. So any strategy $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ is such that σ_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 $r_1, \ldots, r_k$.

Rem.

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

Local plays and strategies

- $\Sigma_r^{loc} = \{a \in \Sigma : \mathrm{dom}(a) = \{r\}\}$, set of local r-actions.
- $\Sigma_{q,r} = \{a \in \Sigma : \mathrm{dom}(a) = \{q, r\}\}$.
- Local r-play: word from $(\Sigma_r^{loc})^*$.
- r-context: play ending in $\Sigma_{q,r}$.
- Local r-strategy $\sigma_r[t] : (\Sigma_r^{loc})^* \to \Sigma_r^{loc}$ 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 $r_1, \ldots, r_k$.

Rem.

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

Local plays and strategies

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

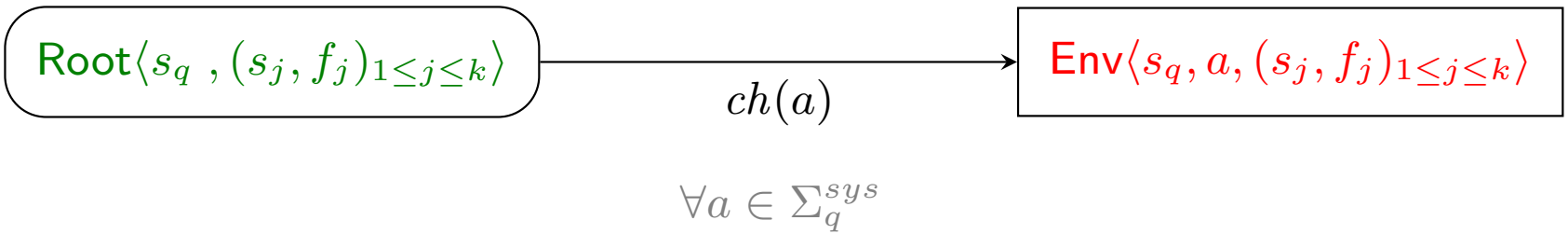
Root process q, leaves $r_1, \ldots, r_k$

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

- s_q is state of root q, s_i is state of leaf r_i,
- f_i is local positional r_i-strategy

From the distributed control problem to a sequential game

Root process q, leaves $r_1, \ldots, r_k$



$\forall a \in \Sigma_q^{sys}$

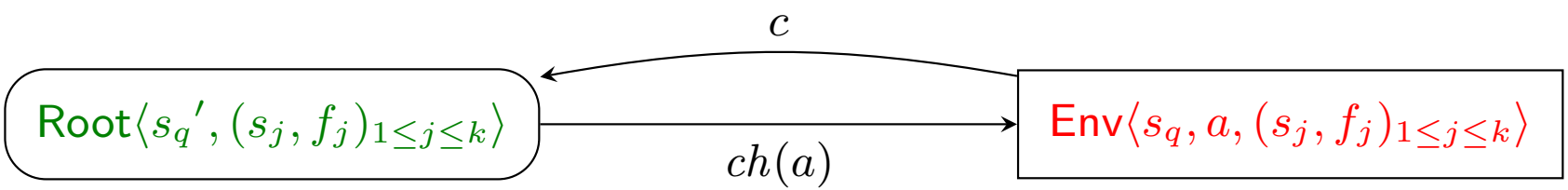
Root q proposes a local, controllable action a

- s_q is state of root q, s_i is state of leaf r_i,
- f_i is local positional r_i-strategy

From the distributed control problem to a sequential game

Root process q, leaves $r_1, \ldots, r_k$

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

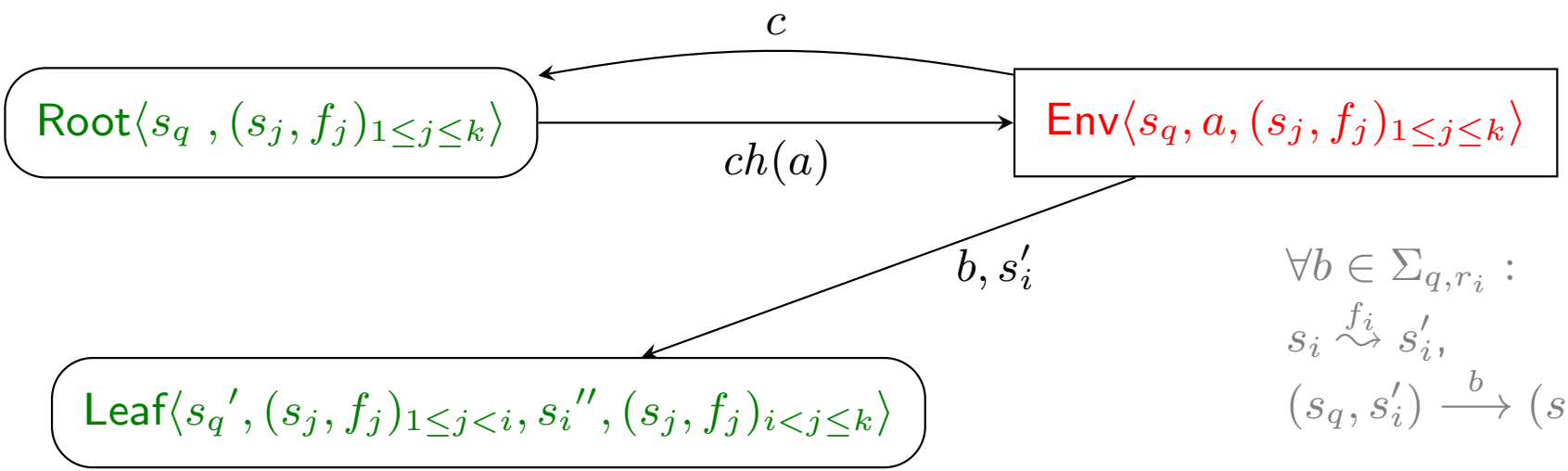


Environment either chooses a local action c for q

- s_q is state of root q, s_i is state of leaf r_i,
- f_i is local positional r_i-strategy

From the distributed control problem to a sequential game

Root process q, leaves $r_1, \ldots, r_k$

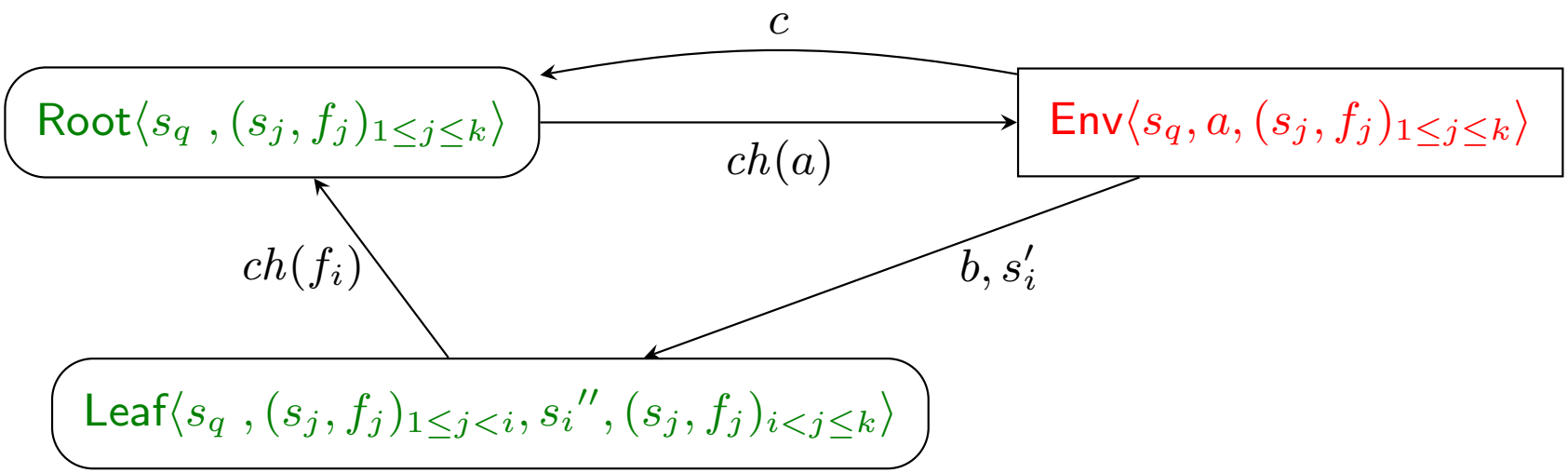


Or chooses a synchronization action b between q and r_i.

- s_q is state of root q, s_i is state of leaf r_i,
- f_i is local positional r_i-strategy

From the distributed control problem to a sequential game

Root process q, leaves $r_1, \ldots, r_k$



- s_q is state of root q, s_i is state of leaf r_i,
- 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:

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

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:

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

Matching lower bound: nested counters

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

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

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

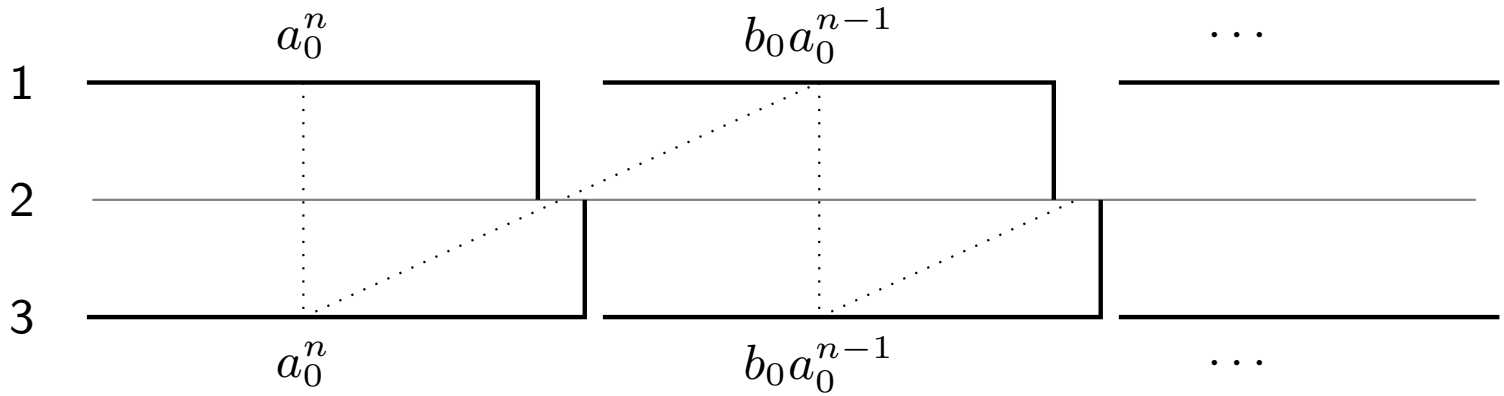
$$a_1 \mathsf{bin}(0) a_1 \mathsf{bin}(1) \cdots a_1 \mathsf{bin}(2^n - 1) \# b_1 \mathsf{bin}(0) a_1 \mathsf{bin}(1) \cdots a_1 \mathsf{bin}(2^n - 1) \# \cdots$$

- level k: $\ldots$

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

Level 1

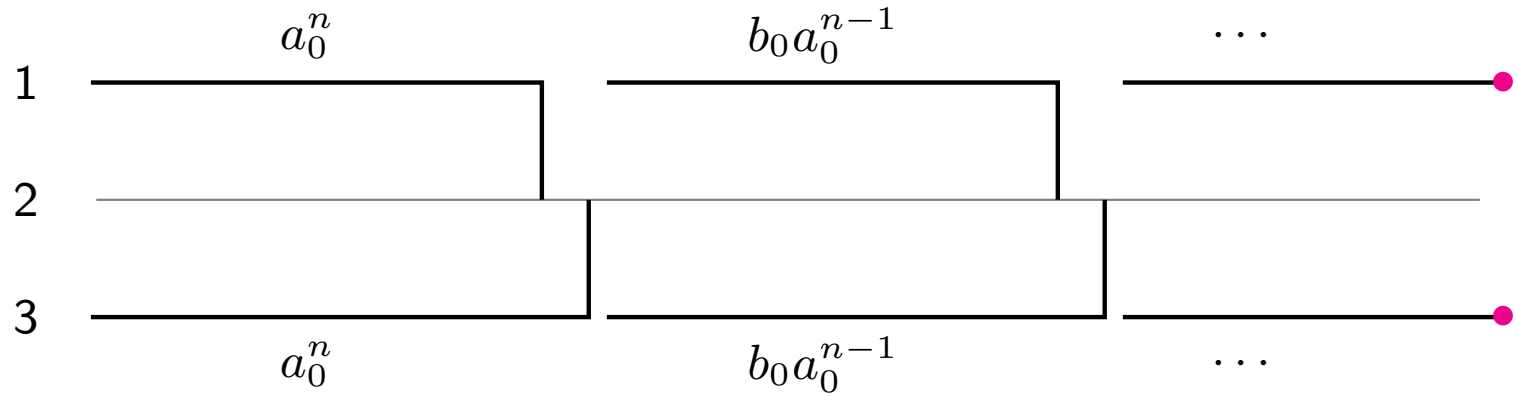
Processes 1,2,3.



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

Level 1

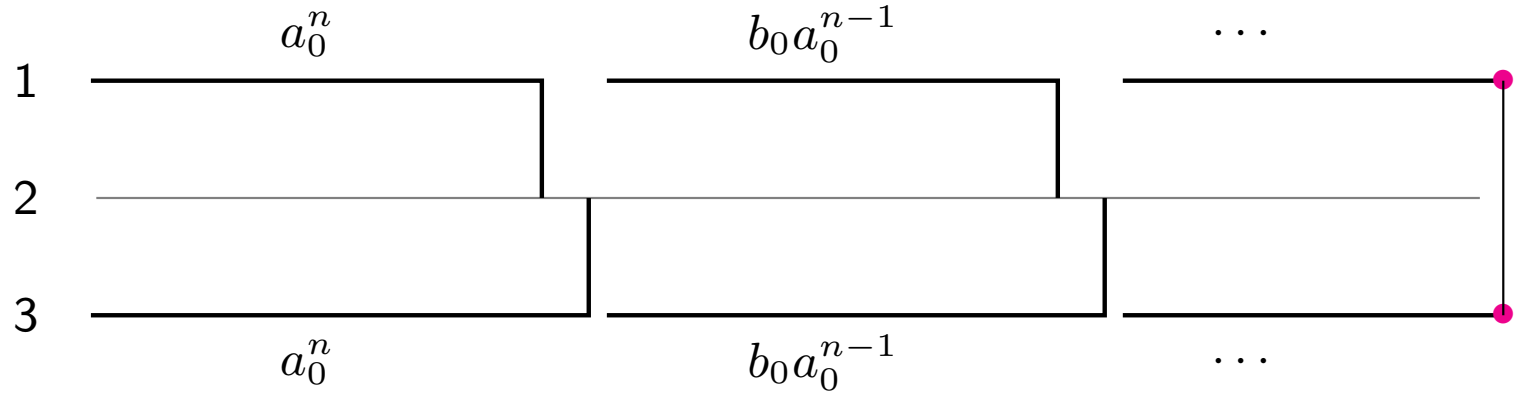
Processes 1,2,3.



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

Level 1

Processes 1,2,3.



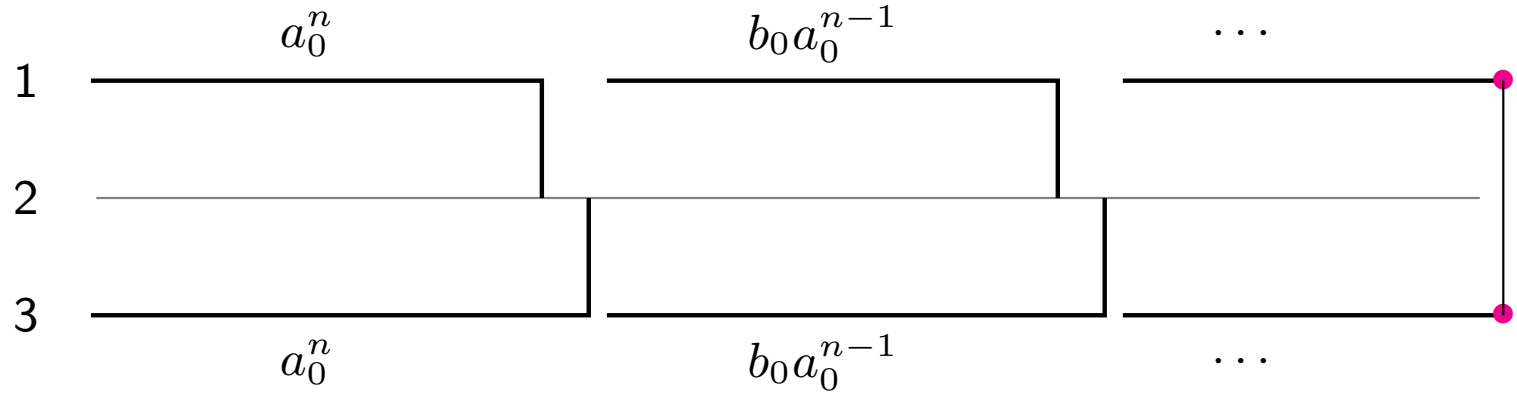
Environment can ask for a pair of bits: either $\uparrow$ (above) or $\nearrow$.

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 $1 \text{ ——— } 2 \text{ ——— } 3$ is Exptime-complete.

What makes distributed control so difficult?

Branching and distributed automata

Unfolding of distributed automata: event structures.

What makes distributed control so difficult?

Branching and distributed automata

Unfolding of distributed automata: event structures.

A prefix-closed trace language L defines a Σ-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 action of the trace.

What makes distributed control so difficult?

Branching and distributed automata

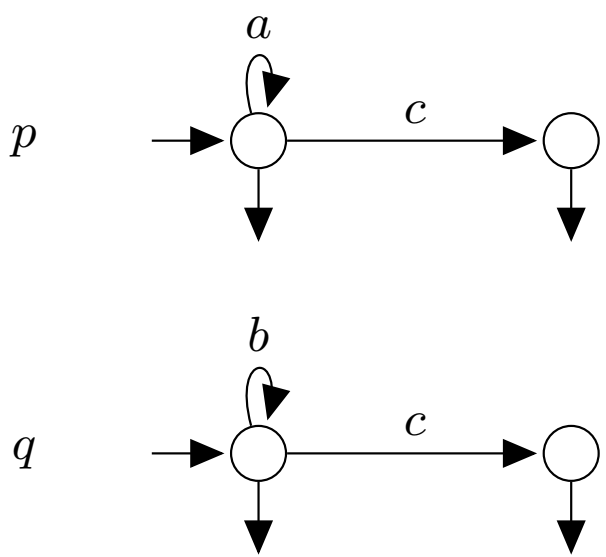
Unfolding of distributed automata: event structures.

A prefix-closed trace language L defines a Σ-labeled event structure:

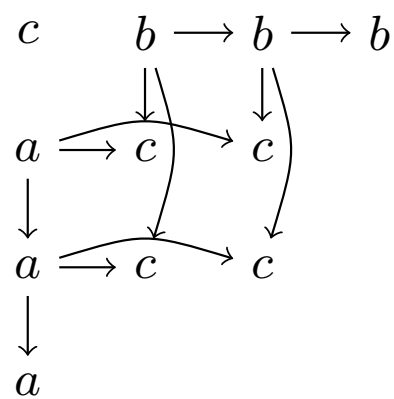
- Nodes: traces from L with *one* maximal element (prime traces).
- Partial order: trace prefix relation.
- Conflict relation: no common extension.
- Label: maximal action of the trace.

Example

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



Event structure:



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.].

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 $A \subseteq \Sigma_p^{sys}$, $B \subseteq \Sigma_q^{sys}, \ldots$.

- 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 $\{a \in \Sigma : p \in \mathrm{dom}(a)\}$ and $\sigma_p([w]) = A$.
- Formula φ expresses that (1) every uncontrollable action is allowed by σ and (2) that the winning condition is satisfied by every trace representing a σ-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 (implies "grid-free" unfolding).
  Very recently disproved by Chalopin/Chepoi (August 2018), by exhibiting $\mathcal{A}$ with grid-free unfolding, yet unbounded tree-width, hence undecidable MSO theory.
- Warning: Decidability of MSO is not necessary for deciding the control problem.
  Example: the control problem for distributed automata over 2 processes is decidable, yet the MSO theory of the unfolding is undecidable.

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).

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).

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).

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.

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.

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, general case is open.
  Note: related game model $\longrightarrow$ Petri games (Finkbeiner, Olderog 2014).

Control of distributed automata: open questions

- Decidability of the general asynchonous 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 (cf. Chalopin/Chepoi recent paper).

Control of distributed automata: open questions

- Decidability of the general asynchonous 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 (cf. Chalopin/Chepoi recent paper).

Thank you for listening!