# Logic, games and synthesis – ESSV, Univ. Bordeaux 1, 2012/13

## Contents

- Anca Muscholl: 2-player games on graphs, logic and automata on infinte trees, controller synthesis, distributed games.
- Hugo Gimbert: Markov chains, stochastic games, probabilistic automata.

# Plan

# Plan

- Two-player games can model open systems, i.e., where a system faces a (hostile) environment. The question is whether the system has a strategy for satisfying a given specification, no matter what the environment does. Synthesizing a strategy (or a program) amounts to construct correct programs.
- Two-player games are connected to logics ($\forall/\exists$). They can be used for showing (in)expressivity results (see first-order logic and Ehrenfeucht-Fraissé games).
- More complex types of games arise in specific applications, like in stochastic, timed or distributed systems.

- Two-player games can model open systems, i.e., where a system faces a (hostile) environment. The question is whether the system has a strategy for satisfying a given specification, no matter what the environment does. Synthesizing a strategy (or a program) amounts to construct correct programs.
- Two-player games are connected to logics ($\forall/\exists$). They can be used for showing (in)expressivity results (see first-order logic and Ehrenfeucht-Fraissé games).
- More complex types of games arise in specific applications, like in stochastic, timed or distributed systems.

## Some vocabulary

- Games are played on arenas, which can be finite or infinite (words, trees, graphs, ...).
- Solving games means finding winning positions as well as winning strategies for the players.

## Notations

- $\Sigma$ finite alphabet, $\Sigma^*$ set of finite words over $\Sigma$,
- $\Sigma^\omega$ set of infinite words over $\Sigma$ (mappings $\mathbb{N} \to \Sigma$)

## Gale-Stewart games

- 2 players, $P_0, P_1$, play alternatively ($P_0$ starts). Each player has his own alphabet: $\Sigma_0, \Sigma_1$. A move of $P_i$ consists of the choice of some symbol in $\Sigma_i$.
- A play is a (infinite) sequence of moves: $w \in (\Sigma_0\Sigma_1)^\omega$.
- A Gale-Stewart game is given by a winning condition

$$\text{Win} \subseteq (\Sigma_0\Sigma_1)^\omega$$

- Play $w$ is winning for $P_0$ if $w \in \text{Win}$. Otherwise it is winning for $P_1$.
- A strategy for $P_0$ is a mapping $\sigma : \Sigma_1^* \to \Sigma_0$ (or $\sigma : (\Sigma_0\Sigma_1)^* \to \Sigma_0$). Extends to $\overline{\sigma} : \Sigma_1^\omega \to \Sigma_0^\omega$ by

$$\overline{\sigma}(b_1 b_2 \cdots) = a_0 a_1 \cdots \text{ if } a_n = \sigma(b_1 \cdots b_n), \quad n \geq 0$$

## Playing the game

- If $P_1$ plays $y = b_1 b_2 \cdots \in \Sigma_1^\omega$ and $P_0$ applies $\sigma$ by playing $\overline{\sigma}(y) = a_0 a_1 \cdots$: resulting play is $a_0 b_1 a_1 b_2 \cdots$, denoted as $\overline{\sigma}(y) \mid y$.
- $P_0$ wins the game with strategy $\sigma$ if for every $y \in \Sigma_1^\omega$ we have $\overline{\sigma}(y) \mid y \in \mathrm{Win}$.

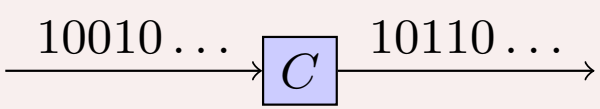
## Determinacy

- We define in the same way strategies for $P_1$. The winning condition for $P_1$ is $\mathrm{Win}^{co} = \Sigma^\omega \setminus \mathrm{Win}$.
- The game is called determined if either $P_0$ or $P_1$ has a winning strategy.
- Martin's theorem says that Gale-Stewart games are determined when Win is a Borel set.

## Regular games

- $\omega$-regular: defined by automata (Büchi, Muller, parity, ...), or logics (monadic second-order), or $\omega$-expressions, ....
- Regular (infinite) games: winning conditions are $\omega$-regular. In particular, they are Borel. Regular games are therefore determined.

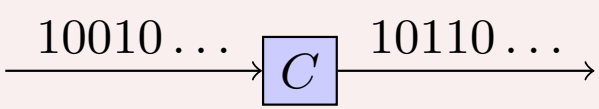
## Church's problem (1957)




Church's problem:
given an input/output relation $R \subseteq (\{0,1\}^2)^\omega$, construct a circuit $C$ implementing $R$.

- Example 1: if the current input is 1, then the corresponding output is 1; if the input is 0 then the output is the parity of the number of 0's seen so far. A finite memory circuit $C$ (automaton) can do this job.
- Example 2: if the current input is 1, then the output is 0; and if there are infinitely many inputs equal to 0, then infinitely mnay outputs are equal to 0.
  The circuit that outputs only 0's does the job.
- Example 3: Infinitely many 1's in the input if and only if finitely many 1's in the output.
  No circuit can do this (why?)

## Church's problem (1957)




Church's problem:
given an input/output relation $R \subseteq (\{0,1\}^2)^\omega$, construct a circuit $C$ implementing $R$.

- Example 1: if the current input is 1, then the corresponding output is 1; if the input is 0 then the output is the parity of the number of 0's seen so far. A finite memory circuit $C$ (automaton) can do this job.
- Example 2: if the current input is 1, then the output is 0; and if there are infinitely many inputs equal to 0, then infinitely mnay outputs are equal to 0.
  The circuit that outputs only 0's does the job.
- Example 3: Infinitely many 1's in the input if and only if finitely many 1's in the output.
  No circuit can do this (why?)

## Remark

Church's problem is about non-terminating computations. The specifications are MSOL (monadic second-order logic). Church's problem was solved by Büchi-Landweber in 1969.

## Arenas

$$\mathcal{A} = (V_0, V_1, E)$$

- Two players $P_0$ et $P_1$. The set of vertices is partitioned into two disjoint subsets: $V_0$ belongs to $P_0$ and $V_1$ to $P_1$.
- Set of edges $E \subseteq V \times V$, $V = V_0 \cup V_1$.

## Plays

A play $\pi$ over $\mathcal{A}$ is a maximal path in $\mathcal{A}$:

- either an infinite path $\pi = v_0, v_1, \ldots$
- or a finite path $\pi = v_0, v_1, \ldots, v_n$ such that $v_n$ has no successor.

## Strategies

- A strategy for $P_0$ is a mapping $\sigma : V^* V_0 \to V$ s.t. $\sigma(\pi v) \in vE := \{w \in V \mid (v, w) \in E\}$, with $\pi \in V^*$, $v \in V$.
- A play $\pi = v_0, v_1, \ldots$ is consistent with the strategy $\sigma$ if $v_{i+1} = \sigma(v_0 \cdots v_i)$ for all $i$ s.t. $v_i \in V_0$.
- Different types of strategies (non-randomized): positional, finite-memory, ...

## Game

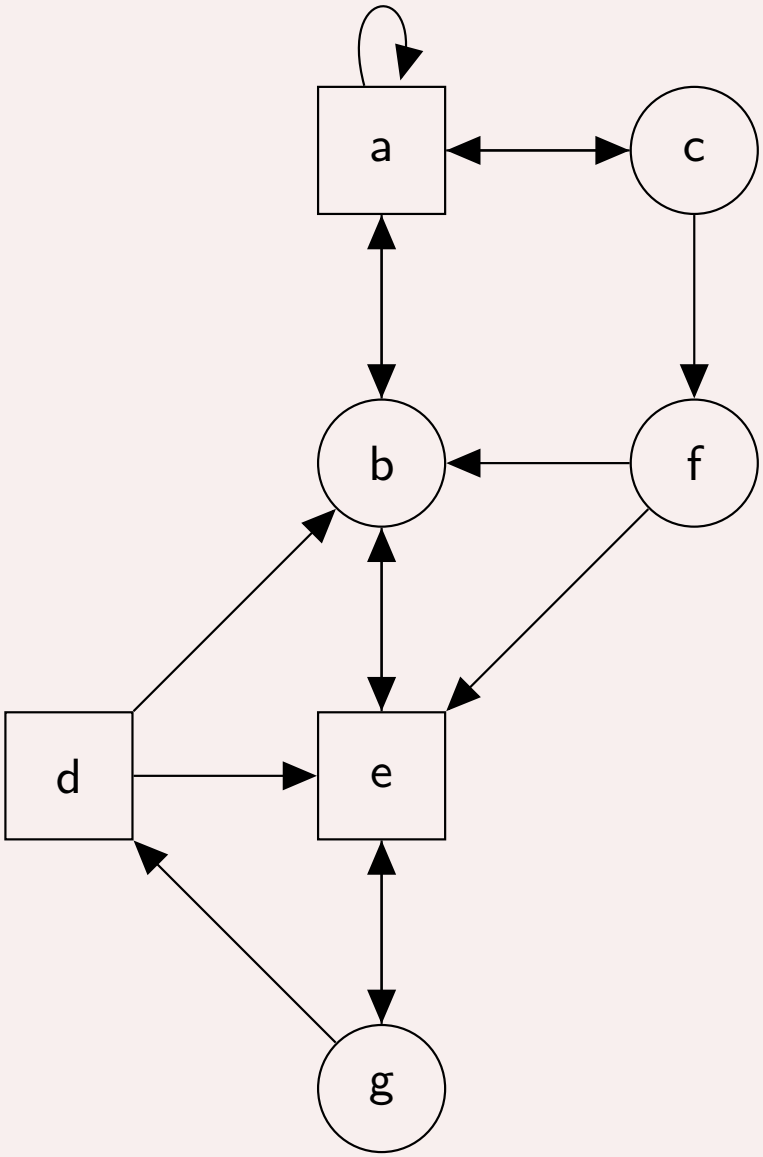
A game is given by a pair $(\mathcal{A}, \text{Win})$, where $\mathcal{A}$ is a graph arena and $\text{Win} \subseteq V^\omega$ is the winning condition.
Some examples ($G, B \subseteq V$, $\mathcal{F} \subseteq \mathcal{P}(V)$) :

- reachability games: $\text{Win} = V^* G V^\omega$
- safety games: $\text{Win} = V^\omega \setminus V^* B V^\omega$
- obligation games: the set of states visited during a play should belong to $\mathcal{F}$
- Büchi games: $\text{Win} = (V^* G)^\omega$, Muller games, parity games...

## Gale-Stewart games and graph games

- Let $\mathcal{A}$ be a finite arena such that $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$ (turn-based) and no vertex is a dead end. Such a graph game corresponds to a Gale-Stewart game where $\Sigma_0 = \{(u, v) \in V_0 \times V_1 \mid (u, v) \in E\}$ and $\Sigma_1 = \{(u, v) \in V_1 \times V_0 \mid (u, v) \in E\}$. A regular winning condition can in particular enforce that every play corresponds to a path in $\mathcal{A}$.
- A Gale-Stewart game $(\Sigma_0, \Sigma_1)$ basically corresponds to a graph game with vertices $V_0 = \{\mathit{start}\} \cup \Sigma_0 \times \Sigma_1 \times \{0\}$, $V_1 = \Sigma_0 \cup \Sigma_0 \times \Sigma_1 \times \{1\}$.
  Plays are of the form $\mathit{start}, a_0, (a_0, b_0), (a_1, b_0), (a_1, b_1), \ldots$ and $P_0$ wins if $a_0 b_0 a_1 b_1 \cdots \in$ Win.
  If Win is regular we can translate the Gale-Stewart game into an equivalent graph game with winning condition of type: Muller, Rabin, parity, ...

## $P_0$: circle, $P_1$: square




Reachability game:
$F = \{a\}$: $W_0 = \{a, b, c, f\}$
$F = \{e\}$: $W_0 = V \setminus \{a\}$

Büchi games: same $W_0$ as above

## Attractors

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

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

$\mathsf{Attr}_0^i(F)$ 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 $V \setminus \mathsf{Attr}_0^{|V|}(F)$ is the winning region of $P_1$.

## Strategies

Reachability games are determined and have positional winning strategies (for both players), which can be computed in polynomial time.
Positional strategy (for $P_0$): mapping $\sigma : V_0 \to V$.

## Strategies

Büchi games are determined and have positional winning strategies (for both players), which can be computed in polynomial time.

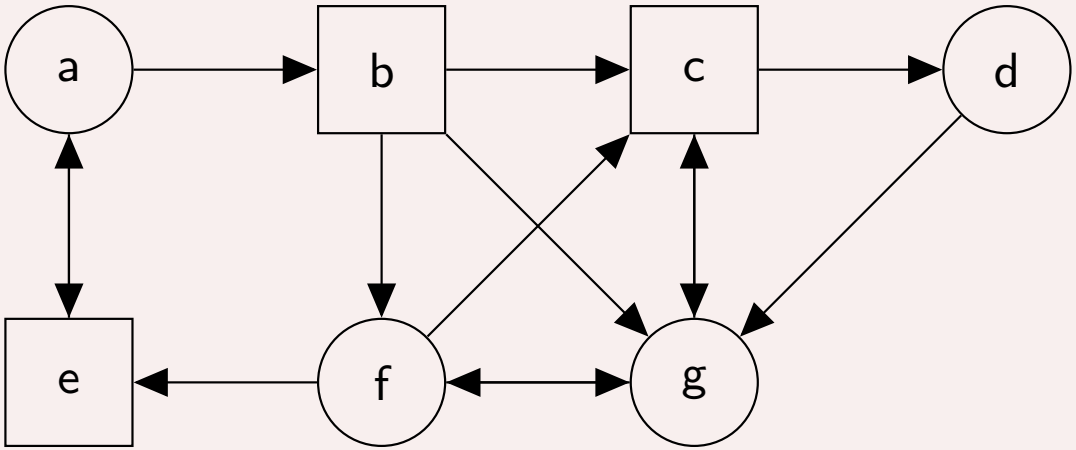
## Algorithm

- $\mathsf{Attr}_0^+(F)$: 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).

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

  Notice that $X^{(i+1)} \subseteq X^{(i)}$. So there exists $k$ with $W_0 = X^{(k)} = X^{(k+1)}$: $W_0$ is a largest fixpoint.
- Strategy for $P_0$ in the Büchi game: positional strategy used by $P_0$ from $W_0 = \mathsf{Attr}_0^+(W_0 \cap F)$.

**Node coloring $\chi : V \to \{0, \ldots, k\}$. Winning condition: maximal visited color is even.**




$\chi(a) = \chi(e) = 0$, $\chi(b) = \chi(f) = 1$, $\chi(c) = \chi(g) = 2$, $\chi(d) = 3$.

- $d \in W_1$ and $\mathsf{Attr}_1(d) = \{d, c, b\} \subseteq W_1$
- $V' = V \setminus \{b, c, d\}$ is a trap for $P_1$, defining a subarena for $P_0$. So in $V'$, $P_0$ can avoid color 3, and $\mathsf{Attr}_0(g) \restriction_{V'} = \{g, f\} \subseteq W_0$
- $V'' = V' \setminus \{f, g\}$ is a trap for $P_0$, so subarena for $P_1$. $W_1 \restriction_{V''} = \emptyset$, $W_0 \restriction_{V''} = \{a, e\}$

Conclusion: $W_0 = \{a, e, f, g\}$, $W_1 = \{b, c, d\}$. Positional winning strategies for both players (attractor strategies).

- The complement of an attractor $C$ for $P_i$ is a trap pour $P_i$: $P_i$ has no outgoing transition from $C$, so $P_{1-i}$ can force the play to stay in $V \setminus C$.
- A trap $C$ for $P_i$ is a subarena for $P_{1-i}$: every winning strategy for $P_{1-i}$ in $C$ is winning for $P_{1-i}$ in $V$.

**Algorithm (arena $(V, E)$)**

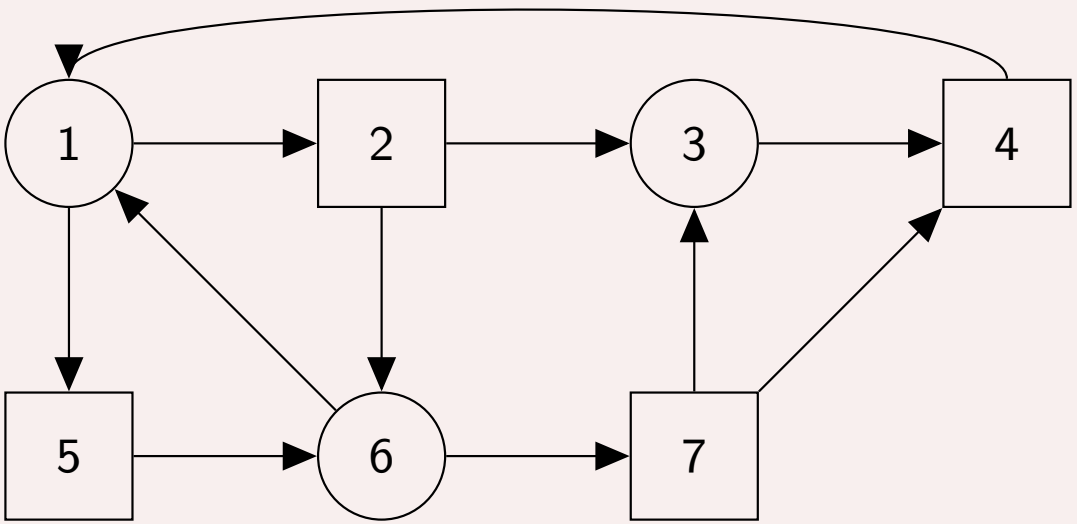
$V^{(k)} := \chi^{-1}(k)$.

1. Suppose that the maximal color $k$ is odd (otherwise exchange $P_0, P_1$). Compute $\mathrm{Attr}_1(V^{(k)})$ (a subset of $W_1$).
2. $V' := V \setminus \mathrm{Attr}_1(V^{(k)})$ is subarena for $P_0$ and the maximal color in $V'$ is at most $k-1$. Compute inductively the winning region $W_0'$ of $P_0$ in $V'$. Every vertex in $W_0'$ is winning for $P_0$ in $V$ (a subset of $W_0$, see above).
3. Every vertex in $V' \setminus W_0'$ is winning for $P_1$ in $V$: $P_1$ applies his winning strategy from $V'$ while staying in $V'$, and the attractor strategy as soon as the game enters $\mathrm{Attr}_1(V^{(k)})$.
4. Strategies for both players are positional.

## Winning condition

Win is given as a set $\mathcal{F} \subseteq \mathcal{P}(V)$ of sets of states. Player $P_0$ wins a play $\pi$ if the set $\mathrm{Occ}(\pi)$ of states visited in $\pi$ belongs to $\mathcal{F}$.
Obligation games are thus Boolean combinations of reachability games.

## Example: conjunction of reachability conditions




Play $\pi \in \mathrm{Win}$ iff $\{2, 7\} \subseteq \mathrm{Occ}(\pi)$. Player $P_0$ needs 1 bit of memory in state 1: he needs to visit alternatively states 2 and 5.

## Solution

Reduce the obligation game to a simpler (and larger) weak parity game.

- Recall: $P_0$ wins in a weak parity game if the maximal visited color is even.
- General idea of the reduction: start with an obligation game $G = (\mathcal{A}, \mathcal{F})$ and define a weak parity game $G' = (\mathcal{A}', \chi)$ such that:
  1. $\mathcal{A}' = (V_0', V_1', E')$, $V_i' = V_i \times M$, with $M$ finite (memory).
  2. Transitions: $(\langle v, m\rangle, \langle w, m'\rangle) \in E'$ if $(v, w) \in E$ and $m' = f(m, v, w)$ ($f =$ memory update function).
  3. With each play $\pi$ in $G$ starting in $v$, and each initial value $m_0 \in M$, we associate a unique play $\pi'$ in $G'$ from $(v, m_0)$. In addition, $\pi$ and $\pi'$ are won by the same player.
- Since weak parity games have positional strategies, such a reduction then provides finite memory strategies in the obligation game.

## Solution

Reduce the obligation game to a simpler (and larger) weak parity game.

- Recall: $P_0$ wins in a weak parity game if the maximal visited color is even.
- General idea of the reduction: start with an obligation game $G = (\mathcal{A}, \mathcal{F})$ and define a weak parity game $G' = (\mathcal{A}', \chi)$ such that:
  1. $\mathcal{A}' = (V_0', V_1', E')$, $V_i' = V_i \times M$, with $M$ finite (memory).
  2. Transitions: $(\langle v, m\rangle, \langle w, m'\rangle) \in E'$ if $(v, w) \in E$ and $m' = f(m, v, w)$ ($f$ = memory update function).
  3. With each play $\pi$ in $G$ starting in $v$, and each initial value $m_0 \in M$, we associate a unique play $\pi'$ in $G'$ from $(v, m_0)$. In addition, $\pi$ and $\pi'$ are won by the same player.
- Since weak parity games have positional strategies, such a reduction then provides finite memory strategies in the obligation game.

## Reduction

- $M = 2^V$, $E' = \{(v, R) \to (w, R \cup \{v\}) \mid (v, w) \in E\}$.
- Coloring: $\chi(v, R) = 2|R|$ if $R \in \mathcal{F}$ ; $\chi(v, R) = 2|R| - 1$ if $R \notin \mathcal{F}$.
- For every play $\pi' = (v_0, R_0), (v_1, R_1), \ldots$ in $G'$ we have $R_0 \subseteq R_1 \subseteq R_2 \cdots$, so for some $i \geq 0$ we have $R_j = R_i$ for all $j \geq i$. For $R_0 = \emptyset$, the set $R_i$ equals $\mathrm{Occ}(\pi)$, where $\pi = v_0, v_1, \ldots$.

Let $\mathsf{Inf}(v_0 v_1 \cdots) = \{v \in V \mid \exists^\infty i : v = v_i\}$. For simplicity, we assume that there are no dead-ends in $(V, E)$.

## Parity games

Node coloring $\chi : V \to \{1, \ldots, k\}$.

$$\mathsf{Win} = \{\pi \mid \max(\chi(\mathsf{Inf}(\pi))) \text{ is even}\}.$$

## Muller games

Let $\mathcal{F} \subseteq \mathcal{P}(V)$.

$$\mathsf{Win} = \{\pi \mid \mathsf{Inf}(\pi) \in \mathcal{F}\}$$

## Rem.

Every game $(\mathcal{A}, \mathsf{Win})$ with $\mathsf{Win} \subseteq V^\omega$ regular reduces to a Muller game. We will show how to reduce Muller games to parity games, then how to solve parity games.

## (Dziembowski, Jurdzinski, Walukiewicz)

- $V_0 = \{A, B, C, D\}$, $V_1 = \{1, 2, 3, 4\}$
- $E = V_0 \times V_1 \cup V_1 \times V_0$
- Winning condition for $P_0$:

$$|\mathsf{Inf}(\pi) \cap V_0| = \max(\mathsf{Inf}(\pi) \cap V_1)$$

How can $P_0$ win? solution: Last Appearance Record (LAR).

## (Dziembowski, Jurdzinski, Walukiewicz)

- $V_0 = \{A, B, C, D\}$, $V_1 = \{1, 2, 3, 4\}$
- $E = V_0 \times V_1 \cup V_1 \times V_0$
- Winning condition for $P_0$:

$$|\mathsf{Inf}(\pi) \cap V_0| = \max(\mathsf{Inf}(\pi) \cap V_1)$$

How can $P_0$ win? solution: Last Appearance Record (LAR).

## LAR

The order of visited $V_0$ states is recorded: the last visited state is moved to the front of LAR, and the position where it occurred previously (if any) is underlined – the hit position.
Example:

$$\epsilon \xrightarrow{A} A \xrightarrow{D} DA \xrightarrow{C} CDA \xrightarrow{C} \underline{C}DA \xrightarrow{D} D\underline{C}A \xrightarrow{D} \underline{D}CA \xrightarrow{C} C\underline{D}A \cdots$$

If $\mathsf{Inf}(\pi) \cap V_0 = \{C, D\}$ then infinitely often, the hit position in the LAR is 2 (we start to count from 1). $P_0$'s (winning) strategy is to play the hit position of the LAR.

## Reduction

- Let $(G, \mathcal{F})$ be a Muller game with $G = (V_0, V_1, E)$ and $\mathcal{F} \subseteq 2^V$.
- LAR: pairs $\langle v_1 \cdots v_r, h \rangle$, where $0 \leq h \leq r$ and $v_1, \ldots, v_r$ are distinct states from $V$.
  Let $Q$ be the set of LAR's. The initial LAR is $q_0 = \langle \epsilon, 0 \rangle$, and the transition function $\delta : Q \times V \to Q$ is given by:

$$\delta(\langle v_1 \cdots v_r, h \rangle, v) = \begin{cases} \langle v v_1 \cdots v_r, 0 \rangle & \text{if } v \neq v_p \text{ for all } p \\ \langle v v_1 \cdots v_{p-1} v_{p+1} \cdots v_r, p \rangle & \text{if } v = v_p \end{cases}$$

  Hit set of $\langle v_1 \cdots v_r, h \rangle$: $\{v_1, \ldots, v_h\}$.
- For an infinite play $\pi$, let $\pi = \pi_0 \pi'$ be such that $\mathrm{Inf}(\pi)$ is the set of states occurring in $\pi'$. If $h$ is the maximal hit position repeated infinitely often, then $\mathrm{Inf}(\pi)$ equals the hit set for the LAR's with hit position $h$ in $\pi'$.
-

$$\chi(\langle v_1 \cdots v_r, h \rangle) = \begin{cases} 2h & \text{if } \{v_1, \ldots, v_h\} \in \mathcal{F} \\ 2h - 1 & \text{else} \end{cases}$$

  $\mathrm{Inf}(\pi) \in \mathcal{F}$ iff the maximal LAR-color repeated infinitely often is even. Thus, Muller games can be simulated by parity games (with $n! \cdot n$ additional memory), so they can be solved with finite-memory strategies.

## Inductive solution

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

## Inductive solution

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

## Complexity

The above algorithm has exponential running time. Since strategies are positional, computing the winning regions is in $\mathrm{NP} \cap \text{co-NP}$.

# Plan

# Games & complementing Büchi $\omega$-automata

Ch. 4 in *Automata, Logics and Infinite Games (Eds. Grädel, W. Thomas, Th. Wilke)*

## Methods

- Büchi (1962): combinatorial arguments
- McNaughton (1966), Safra (1988): determinization
- Klarlund (1991): progress measures
- Kupferman/Vardi (1997): alternating automata

## Complexity (upper/lower bounds)

- Büchi: doubly exponential upper bound
- Safra: upper bound $2^{\Theta(n \log(n))}$
- M. Michel (1988): lower bound $n!$
- Kupferman/Vardi (1997,...): upper bound $(c_1 \cdot n)^n$ $(c < 1)$
- Yan (2008): lower bound $(c_2 \cdot n)^n$
- Schewe (2009): matching upper bound

## Run graphs

- Büchi automaton $\mathcal{A} = \langle S, \Sigma, s_{in}, \delta, F \rangle$, $\omega$-word $w$
- Run graph $G(w)$ of $\mathcal{A}$ on $w = a_0 a_1 \ldots$:

$$G(w) = \langle V, E, C \rangle$$

  - vertices $V \subseteq S \times \mathbb{N}$, where

$$\begin{array}{rcl} (s, 0) & \in & V \text{ iff } s = s_{in} \\ (s', n+1) & \in & V \text{ iff } s' \in \delta(s, a_n) \text{ for some } (s, n) \in V \end{array}$$

  - edge set:

$$(s, n) \rightarrow (s', n+1) \text{ if } s' \in \delta(s, a_n)$$

  - marked vertices:

$$C = \{(s, n) \in V \mid s \in F\}$$

- $w \notin L(\mathcal{A})$ iff no infinite path in $G(w)$ has infinitely many marked vertices (we say that $G(w)$ is finitely marked).

## Idea

- Partition the vertex set $V$ of $G(w)$ in a (infinite) sequence of sets $V_0, V_1, \ldots$:
  - $V_0 \subseteq V$ contains all vertices from which we can reach only finitely many vertices,
  - $V_1 \subseteq V \setminus V_0$ contains all vertices from which we cannot (non-trivially) reach any marked vertex,
  - $V_2 \subseteq V \setminus (V_0 \cup V_1)$ contains all vertices from which we can reach only finitely many vertices,
  - $\ldots$
- We show that $w \notin L(\mathcal{A})$ iff there exists some $0 \leq k \leq 2|S|$ such that $V_k = \emptyset$ for all $k > k_0$.
- Büchi automaton for $\Sigma^\omega \setminus L(\mathcal{A})$ guesses the existence of a partition.

## Progress measures

Let $m \in \mathbb{N}$ and $G = \langle V, E, C \rangle$ be a DAG with $C \subseteq V$.
A progress measure of size $m$ for $G$ is a mapping $\mu : V \to \{1, \ldots, 2m+1\}$ satisfying:

- $\mu(u) \geq \mu(v)$ for all $(u, v) \in E$,
- if $\mu(u) = \mu(v)$ and $(u, v) \in E$, then either $\mu(u)$ is odd or $v \notin C$,
- there is no infinite path in $G$ with constant $\mu$-value odd.

## Notations

$G = \langle V, E, C \rangle$ marked graph. For $v \in V$ let $R(v) = \{w \in V \mid v \xrightarrow{*} w\}$ denote the set of vertices reachable from $v$.

- Finite reach:
$$B(G) = \{v \in V \mid R(v) \text{ is finite}\}$$
- Unmarked reach:
$$U(G) = \{v \in V \mid C \cap (R(v) \setminus \{v\}) = \emptyset\}$$
- If $G$ is layered DAG (edges from layer $n$ go to layer $n+1$):
width$(G)$ = maximal number occurring infinitely often as size of a layer.

## Partition of $G(w)$

Define sequence of subgraphs $(G_n)_n$ and sets of vertices $(V_n)_n$:

- $G_0 = G(w)$ and $V_0 = B(G_0)$,
- $G_{2n+1} = G_{2n} \setminus V_{2n}$ and $V_{2n+1} = U(G_{2n+1})$,
- $G_{2n+2} = G_{2n+1} \setminus V_{2n+1}$ and $V_{2n+2} = B(G_{2n+1})$.

We have that $(V_k)_{k \geq 0}$ partitions $V$.

## Progress measure for $G(w)$

- Claim: If $G(w)$ is finitely marked and $G_{2n+1} \neq \emptyset$:

$$\mathsf{width}(G_{2n+2}) < \mathsf{width}(G_{2n+1})\,.$$

  By construction $U(G_{2n+1}) \neq \emptyset$. Moreover, if $v \in U(G_{2n+1})$ then using $G_{2n+1} = G_{2n} \setminus V_{2n}$ we obtain that there is an infinite path starting in $v$ and lying in $G_{2n+1}$. Since this path is not preserved in $G_{2n+2}$, we obtain the claim.
- Let $n = |S|$ $(= \mathsf{width}(G_0))$. Then $G_{2n+1} = \emptyset$.
- $G(w) = \langle V, E, C\rangle$. Recall: $w \notin L(\mathcal{A})$ iff $G(w)$ finitely marked. Define

$$\mu(v) = k + 1 \text{ with } k = \min\{n \mid v \in G_n \setminus G_{n+1}\}$$

  Then $\mu$ is a progress measure for $G(w)$ of size $m = |S|$.

## Complementation

A Büchi automaton of size $2^{O(n \log(m))}$ can check the existence of a progress measure of size $m$.

## $\mathcal{B}^+(X)$: set of positive formulas over $X$

- contains $X$ and constants 0,1,
- built using $\vee, \wedge$.

Model $M$ for $\phi \in \mathcal{B}^+(X)$: subset of $X$ s.t.. $\phi$ evaluates to 1 if $x \in M$ is replaced by 1, and $x \notin M$ by 0. Set of minimal models of $\phi$: $\mathrm{Mod}(\phi)$.

## Weak alternating parity automata (WAPA)

$\mathcal{A} = \langle S, \Sigma, \delta, s_{in}, \chi \rangle$ with $\chi : S \to \{1, \ldots, d\}$ and $\delta : S \times \Sigma \to \mathcal{B}^+(S)$.

## Runs of WAPA

Runs of WAPA $\mathcal{A}$ on $w = a_0 a_1 \cdots$ are DAGs $G$ with vertex set $V \subseteq S \times \mathbb{N}$, such that:

- $(s_{in}, 0) \in V$,
- $(s, m) \to (t, n)$ implies $n = m + 1$,
- for every $(s, n) \in V$: $\{s' \in S \mid (s, n) \to (s', n+1)\} \in \mathrm{Mod}(\delta(s, a_n))$.

An infinite path of $G$ is accepting if the minimal color seen is even. A finite path is accepting if the last node $(s, n)$ is such that $\delta(s, a_n) = 1$. A run DAG is accepting if all paths are accepting. An $\omega$-word $w$ is accepted if there is some accepting run DAG on $w$.

## Weak parity game

Given a WAPA $\mathcal{A}$ and $w = a_0 a_1 \cdots$ we define a weak parity game $\mathcal{G}(\mathcal{A}, w) = \langle V_0, V_1, E, \chi \rangle$:

- $V_0 = S \times \mathbb{N}$, $V_1 = S \times \mathcal{P}(S) \times \mathbb{N}$
- $(s, m) \to (s, M, n)$ if $n = m + 1$ and $M \in \mathsf{Mod}(\delta(s, a_m))$
- $(s, M, m) \to (t, n)$ if $m = n$ and $t \in M$
- $\chi(s, n) = \chi(s, M, n)$ is the color of $s$ in $\mathcal{A}$.

## WAPA & weak parity game

$w \in L(\mathcal{A})$ iff $P_0$ wins the weak parity game $\mathcal{G}(\mathcal{A}, w)$.

## Dual WAPA

- Dual formula $\widehat{\phi}$ obtained from $\phi \in \mathcal{B}^+(X)$ by exchanging $0, 1$ and $\vee, \wedge$.
- Dual WAPA $\widehat{\mathcal{A}} = \langle S, \Sigma, \widehat{\delta}, s_{in}, \widehat{\chi} \rangle$, $\widehat{\delta}(s, a) = \widehat{\delta(s, a)}$, $\widehat{\chi}(s) = \chi(s) + 1$.

By determinacy of weak parity games: $L(\widehat{\mathcal{A}}) = \Sigma^\omega \setminus L(\mathcal{A})$.

## Büchi → WAPA

From a Büchi automaton $\mathcal{B}$ with $n$ states construct a WAPA $\mathcal{A}$ with $O(n^2)$ states:

- $\mathcal{B} = \langle S, \Sigma, \delta, s_{in}, F \rangle$, $\mathcal{A} = \{S', \Sigma, \delta', s'_{in}, \chi\}$
- $S' = S \times \{0, \ldots, 2|S|\}$, $\chi(s, i) = i$,
- $s'_{in} = (s_{in}, 2|S|)$,

$$\delta'((s,i),a) = \begin{cases} \bigvee_{s' \in \delta(s,a)} (s', 0) & \text{if } i = 0 \\ \bigvee_{s' \in \delta(s,a)} ((s', i) \wedge (s', i-1)) & \text{if } i > 0 \text{ and } i \text{ even} \\ \bigvee_{s' \in \delta(s,a)} (s', i) & \text{if } s \notin F \text{ and } i \text{ odd} \\ \bigvee_{s' \in \delta(s,a)} (s', i-1) & \text{if } s \in F \text{ and } i \text{ odd} \end{cases}$$

Show that $L(\mathcal{B}) = L(\mathcal{A})$.

## WAPA → Büchi

From a WAPA with $n$ states one can construct an equivalent Büchi automaton with $2^{O(n^2)}$ states.

# Plan

# Applying parity games: automata over infinite trees

Ch. 8 in *Automata, Logics and Infinite Games (Eds. Grädel, W. Thomas, Th. Wilke)*

## Structures

- The infinite binary tree is the set $\{0,1\}^*$ of all words over $\{0,1\}$: root $\epsilon$, children of node $u$: $u0, u1$.
- A $\Sigma$-labeled infinite binary tree is a mapping $T : \{0,1\}^* \to \Sigma$.

## Automata

On infinite trees: top-down automata $\mathcal{A} = \langle S, \Sigma, \delta, s_0, \text{Acc} \rangle$, where $\delta \subseteq S \times \Sigma \times S \times S$.

Run $\rho : \{0,1\}^* \to S$ on tree $T$:

- $\rho(\epsilon) = s_0$,
- $(\rho(u), T(u), \rho(u0), \rho(u1)) \in \delta$ for all $u \in \{0,1\}^*$.

A run $\rho$ is accepted if *all* infinite paths of $\rho$ satisfy Acc (Acc can be Büchi, Muller, parity etc.).
An infinite tree $T$ is accepted by $\mathcal{A}$ is there is at least one accepting run $\rho$ on $T$. We denote by $\mathcal{L}(\mathcal{A})$ the set of accepted trees.

## Büchi tree automata

Büchi tree automata are strictly weaker than parity tree automata.

Example: "each path of the tree has finitely many $a$-nodes" cannot be accepted by a Büchi automaton, even non-deterministic (why?).
Parity tree automaton:

- $S = \{s_a, s_b\}$, $\Sigma = \{a, b\}$,
- $\delta = \{s_a\} \times \{a\} \times S^2 \cup \{s_b\} \times \{b\} \times S^2$.
- $\chi(s_b) = 0$, $\chi(s_a) = 1$.

## Expressive power

Parity, Muller, Rabin, Streett automata all recognize the same languages. The deterministic versions are strictly weaker (why?).

## Closure operations

Parity tree automata are closed under union, intersection and projection.

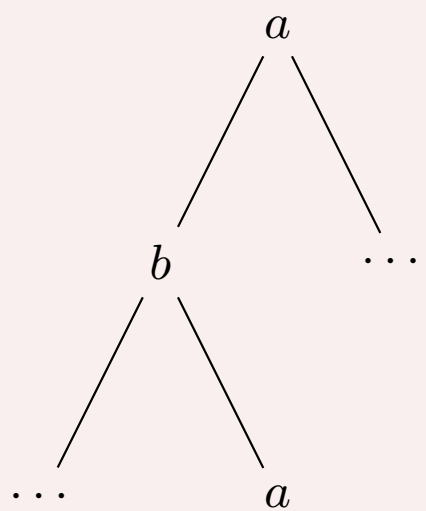
## Complementation (Rabin 1972)

Parity tree automata can be effectively complemented.

## Complementation via games (Gurevich/Harrington 1982)

Two players, Automaton and Pathfinder. Player Automaton chooses a transition, player Pathfinder chooses a direction (left/right) to proceed. The "outcome" is an infinite sequence from $(\Sigma \times S)^\omega$. Automaton wins the game if each outcome satisfies the parity condition.

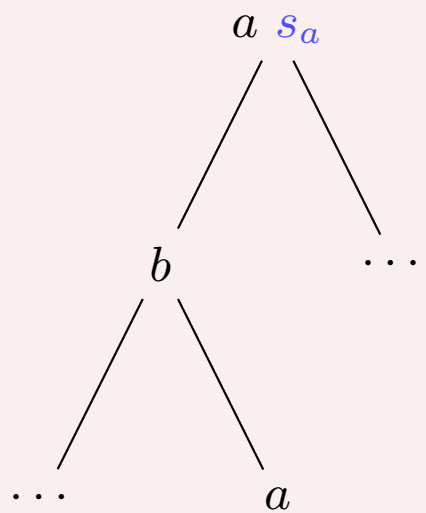
## Example



## Complementation via games (Gurevich/Harrington 1982)

Two players, Automaton and Pathfinder. Player Automaton chooses a transition, player Pathfinder chooses a direction (left/right) to proceed. The "outcome" is an infinite sequence from $(\Sigma \times S)^\omega$. Automaton wins the game if each outcome satisfies the parity condition.

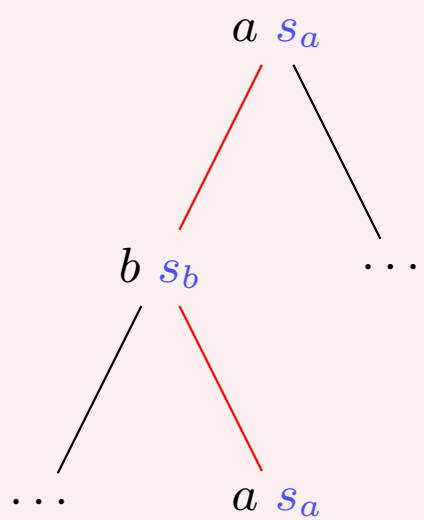
## Example



## Complementation via games (Gurevich/Harrington 1982)

Two players, Automaton and Pathfinder. Player Automaton chooses a transition, player Pathfinder chooses a direction (left/right) to proceed. The "outcome" is an infinite sequence from $(\Sigma \times S)^\omega$. Automaton wins the game if each outcome satisfies the parity condition.

## Example




Outcome $(a, s_a)(b, s_b)(a, s_a) \cdots$

## Complementation game (1/3)

- The above game (on tree $T$) is a *parity game* $\mathcal{G}_T$ on an infinite graph:
  - Automaton positions: $V_0 = \{(u, s) \mid u \in \{0,1\}^*, s \in S\}$,
  - Pathfinder positions: $V_1 = \{(u, \tau) \mid u \in \{0,1\}^*, \tau \in S \times T(u) \times S^2 \cap \delta\}$.

  From position $(u, s)$, Automaton can choose a transition $\tau = (s, T(u), s_0, s_1)$ and move to $(u, \tau)$. From position $(u, \tau)$, Pathfinder can choose $i \in \{0, 1\}$ and move to $(ui, s_i)$. Color $(u, s)$ and $(u, (s, a, s_0, s_1))$ by $\chi(s)$.

## Complementation game (1/3)

- The above game (on tree $T$) is a *parity game* $\mathcal{G}_T$ on an infinite graph:
  - Automaton positions: $V_0 = \{(u, s) \mid u \in \{0,1\}^*, s \in S\}$,
  - Pathfinder positions: $V_1 = \{(u, \tau) \mid u \in \{0,1\}^*, \tau \in S \times T(u) \times S^2 \cap \delta\}$.

  From position $(u, s)$, Automaton can choose a transition $\tau = (s, T(u), s_0, s_1)$ and move to $(u, \tau)$. From position $(u, \tau)$, Pathfinder can choose $i \in \{0, 1\}$ and move to $(ui, s_i)$. Color $(u, s)$ and $(u, (s, a, s_0, s_1))$ by $\chi(s)$.
- Automaton has a winning strategy in $\mathcal{G}_T$ (from $(\epsilon, s_{\text{in}})$) iff $T \in \mathcal{L}(\mathcal{A})$.

## Complementation game (1/3)

- The above game (on tree $T$) is a *parity game* $\mathcal{G}_T$ on an infinite graph:
  - Automaton positions: $V_0 = \{(u, s) \mid u \in \{0,1\}^*, s \in S\}$,
  - Pathfinder positions: $V_1 = \{(u, \tau) \mid u \in \{0,1\}^*, \tau \in S \times T(u) \times S^2 \cap \delta\}$.

  From position $(u, s)$, Automaton can choose a transition $\tau = (s, T(u), s_0, s_1)$ and move to $(u, \tau)$. From position $(u, \tau)$, Pathfinder can choose $i \in \{0, 1\}$ and move to $(ui, s_i)$. Color $(u, s)$ and $(u, (s, a, s_0, s_1))$ by $\chi(s)$.
- Automaton has a winning strategy in $\mathcal{G}_T$ (from $(\epsilon, s_{\text{in}})$) iff $T \in \mathcal{L}(\mathcal{A})$.
- The game $\mathcal{G}_T$ is determined. Hence, Pathfinder has a winning strategy in $\mathcal{G}_T$ (from $(\epsilon, s_{\text{in}})$) iff $T \notin \mathcal{L}(\mathcal{A})$.

## Complementation game (1/3)

- The above game (on tree $T$) is a *parity game* $\mathcal{G}_T$ on an infinite graph:
  - Automaton positions: $V_0 = \{(u, s) \mid u \in \{0,1\}^*, s \in S\}$,
  - Pathfinder positions: $V_1 = \{(u, \tau) \mid u \in \{0,1\}^*, \tau \in S \times T(u) \times S^2 \cap \delta\}$.

  From position $(u, s)$, Automaton can choose a transition $\tau = (s, T(u), s_0, s_1)$ and move to $(u, \tau)$. From position $(u, \tau)$, Pathfinder can choose $i \in \{0, 1\}$ and move to $(ui, s_i)$. Color $(u, s)$ and $(u, (s, a, s_0, s_1))$ by $\chi(s)$.
- Automaton has a winning strategy in $\mathcal{G}_T$ (from $(\epsilon, s_{\text{in}})$) iff $T \in \mathcal{L}(\mathcal{A})$.
- The game $\mathcal{G}_T$ is determined. Hence, Pathfinder has a winning strategy in $\mathcal{G}_T$ (from $(\epsilon, s_{\text{in}})$) iff $T \notin \mathcal{L}(\mathcal{A})$.
- Turn Pathfinder's winning, memoryless strategy into a parity tree automaton (accepting the complementary tree language):
  Memoryless strategy of Pathfinder = mapping $\sigma : \{0,1\}^* \times \delta \to \{0,1\}$.
  Equivalently, $\sigma : \{0,1\}^* \to (\delta \to \{0,1\})$.

## Complementation game (2/3)

Strategy of Pathfinder: $\sigma : \{0,1\}^* \to (\delta \to \{0,1\})$.

- Trees labeled by $\delta \to \{0,1\}$ are called *strategy trees*. A strategy tree in *winning* for $T$ if it corresponds to a winning strategy of Pathfinder in $\mathcal{G}_T$.

## Complementation game (2/3)

Strategy of Pathfinder: $\sigma : \{0,1\}^* \to (\delta \to \{0,1\})$.

- Trees labeled by $\delta \to \{0,1\}$ are called *strategy trees*. A strategy tree in *winning* for $T$ if it corresponds to a winning strategy of Pathfinder in $\mathcal{G}_T$.
- There exists a winning tree $\mathcal{S}$ iff $T \notin \mathcal{L}(\mathcal{A})$.

## Complementation game (2/3)

Strategy of Pathfinder: $\sigma : \{0,1\}^* \to (\delta \to \{0,1\})$.

- Trees labeled by $\delta \to \{0,1\}$ are called *strategy trees*. A strategy tree in *winning* for $T$ if it corresponds to a winning strategy of Pathfinder in $\mathcal{G}_T$.
- There exists a winning tree $\mathcal{S}$ iff $T \notin \mathcal{L}(\mathcal{A})$.
- Given the tree $T$ and a strategy tree $\mathcal{S}$ we consider the "product" tree $T \times \mathcal{S}$. Then $\mathcal{S}$ is winning iff every path of $T \times \mathcal{S}$ violates the parity condition of $\mathcal{A}$.

## Complementation game (2/3)

Strategy of Pathfinder: $\sigma : \{0,1\}^* \to (\delta \to \{0,1\})$.

- Trees labeled by $\delta \to \{0,1\}$ are called *strategy trees*. A strategy tree in *winning* for $T$ if it corresponds to a winning strategy of Pathfinder in $\mathcal{G}_T$.
- There exists a winning tree $\mathcal{S}$ iff $T \notin \mathcal{L}(\mathcal{A})$.
- Given the tree $T$ and a strategy tree $\mathcal{S}$ we consider the "product" tree $T \times \mathcal{S}$. Then $\mathcal{S}$ is winning iff every path of $T \times \mathcal{S}$ violates the parity condition of $\mathcal{A}$.
- Paths in $T \times \mathcal{S}$:

  $$(a_0, f_0, i_0), (a_1, f_1, i_1), \ldots$$

  $a_j \in \Sigma$, $f_j : \delta \to \{0,1\}$, $i_j \in \{0,1\}$.
- A *non-deterministic* parity $\omega$-automaton $\mathcal{C}$ with state set $S$ and coloring $\chi$ can check that a path in $T \times \mathcal{S}$ satisfies the parity condition (non-determinism: guess the transitions of $\mathcal{A}$).

## Complementation game (3/3)

Let $\mathcal{D} = \langle S', \Sigma', \delta', s'_{\mathsf{in}}, \chi' \rangle$ be a deterministic parity automaton accepting $(\Sigma')^\omega \setminus \mathcal{L}(\mathcal{C})$, with $\Sigma' = \Sigma \times \{0,1\}^\delta \times \{0,1\}$.

- Define a tree automaton $\mathcal{B} = \langle S', \Sigma, \Delta', s'_{\mathsf{in}}, \chi' \rangle$ that runs $\mathcal{D}$ on every path of $T$, guessing $\Sigma'$ at the same time:

$$(s', a, s'_0, s'_1) \in \Delta' \quad \text{iff} \quad s_0 = \delta'(s', (a, f, 0)) \text{ and } s_1 = \delta'(s', (a, f, 1))$$
$$\text{for some } f : \delta \to \{0,1\}$$

## Complementation game (3/3)

Let $\mathcal{D} = \langle S', \Sigma', \delta', s'_{\mathsf{in}}, \chi' \rangle$ be a deterministic parity automaton accepting $(\Sigma')^\omega \setminus \mathcal{L}(\mathcal{C})$, with $\Sigma' = \Sigma \times \{0,1\}^\delta \times \{0,1\}$.

- Define a tree automaton $\mathcal{B} = \langle S', \Sigma, \Delta', s'_{\mathsf{in}}, \chi' \rangle$ that runs $\mathcal{D}$ on every path of $T$, guessing $\Sigma'$ at the same time:

$$(s', a, s'_0, s'_1) \in \Delta' \quad \text{iff} \quad s_0 = \delta'(s', (a, f, 0)) \text{ and } s_1 = \delta'(s', (a, f, 1)) \text{ for some } f : \delta \to \{0,1\}$$

- $\mathcal{B}$ accepts the complement of $\mathcal{A}$.

# Plan

# Games & control

## Problem statement

- Given a plant $P$, i.e., transition system with set of actions $\Sigma$ partitioned into controlable actions $\Sigma_c$ and uncontrolable actions $\Sigma_u$, together with a specification $Sp$.

## Problem statement

- Given a plant $P$, i.e., transition system with set of actions $\Sigma$ partitioned into controlable actions $\Sigma_c$ and uncontrolable actions $\Sigma_u$, together with a specification $Sp$.
- Path$(P)$ = set of executions of $P$, not necessarily accepting;
  Path$(P) \supseteq L(P)$ = set of accepting executions of $P$

## Problem statement

- Given a plant $P$, i.e., transition system with set of actions $\Sigma$ partitioned into controlable actions $\Sigma_c$ and uncontrolable actions $\Sigma_u$, together with a specification $Sp$.
- Path$(P)$ = set of executions of $P$, not necessarily accepting;
  Path$(P) \supseteq L(P)$ = set of accepting executions of $P$
- Controller $C$ for plant $P$:

$$\text{Partial mapping } C : \text{Path}(P) \to 2^{\Sigma} \text{ such that}$$
$$\Sigma_u \subseteq C(w) \text{ for all } w \in \text{Path}(P).$$

## Problem statement

- Given a plant $P$, i.e., transition system with set of actions $\Sigma$ partitioned into controlable actions $\Sigma_c$ and uncontrolable actions $\Sigma_u$, together with a specification $Sp$.
- Path$(P)$ = set of executions of $P$, not necessarily accepting;
  Path$(P) \supseteq L(P)$ = set of accepting executions of $P$
- Controller $C$ for plant $P$:

  $$\text{Partial mapping } C : \text{Path}(P) \to 2^{\Sigma} \text{ such that}$$
  $$\Sigma_u \subseteq C(w) \text{ for all } w \in \text{Path}(P).$$

- Controled behavior $P(C) \subseteq \Sigma^*$ of plant $P$ (under $C$): defined inductively
  - $\epsilon \in P(C)$
  - For every $w \in P(C)$ and $a \in \Sigma$:
    $wa \in P(C)$ iff $wa \in \text{Path}(P)$ and $a \in C(w)$.

  The controller $C$ cannot forbid uncontrolable actions, and an action is possible in the controled behavior $P(C)$ iff it is possible in $P$ and allowed by $C$.

## Problem statement

- Given a plant $P$, i.e., transition system with set of actions $\Sigma$ partitioned into controlable actions $\Sigma_c$ and uncontrolable actions $\Sigma_u$, together with a specification $Sp$.
- Path$(P)$ = set of executions of $P$, not necessarily accepting;
  Path$(P) \supseteq L(P)$ = set of accepting executions of $P$
- Controller $C$ for plant $P$:

  Partial mapping $C$ : Path$(P) \to 2^{\Sigma}$ such that
  $\Sigma_u \subseteq C(w)$ for all $w \in$ Path$(P)$.
- Controled behavior $P(C) \subseteq \Sigma^*$ of plant $P$ (under $C$): defined inductively
  - $\epsilon \in P(C)$
  - For every $w \in P(C)$ and $a \in \Sigma$:
    $wa \in P(C)$ iff $wa \in$ Path$(P)$ and $a \in C(w)$.

  The controller $C$ cannot forbid uncontrolable actions, and an action is possible in the controled behavior $P(C)$ iff it is possible in $P$ and allowed by $C$.
- Controled language of $P$ (under $C$): $L_C(P) = L(P) \cap P(C)$.

We want to know whether a controller $C$ exists such that $L_C(P) = Sp$, and construct one (it it exists).

## Setting

- The controled plant is non-blocking if for every $u \in P(C)$ there is some extension $uv \in L_C(P)$ (i.e., leading to a final state).
- $P = \langle S, \Sigma, \delta, s_0, F \rangle$ is finite-state and $\emptyset \neq Sp$ is regular.

## Setting

- The controled plant is non-blocking if for every $u \in P(C)$ there is some extension $uv \in L_C(P)$ (i.e., leading to a final state).
- $P = \langle S, \Sigma, \delta, s_0, F \rangle$ is finite-state and $\emptyset \neq Sp$ is regular.

## Prop.

Suppose that $Sp \subseteq \mathrm{Path}(\mathcal{A})$ is prefix-closed (safety property). There exists some controller $C$ with $P(C) = Sp$ iff $Sp\,\Sigma_u \cap \mathrm{Path}(P) \subseteq Sp$. If $C$ exists, then a finite-memory one exists.

## Remark

- $Sp = \Sigma^*$ and $Sp = \mathrm{Pref}(P) := \{u \mid u \leq v \text{ for some } v \in \mathrm{Path}(P)\}$ satisfy the property above.
- The above result does not refer to final states.

## $Sp$ safety (prefix-closed), $P$ controllable

$L(P) = (a^+b)^*a^*$, $Sp =$ "at most two consecutive $a$'s":

$$Sp = (ab + aab)^*(\epsilon + a + aa)$$

$Sp$ satisfies $Sp\Sigma_u \cap \mathsf{Path}(P) \subseteq Sp$ iff $a \in \Sigma_c$. If $a \in \Sigma_c$, then controlling $P$ means counting the number of $a$'s seen since the last $b$ (up to 2).

## $Sp$ safety, $P$ uncontrollable

$L(P) = (a^+b)^*a^*$, $Sp$ "at most two consecutive $a$'s before *last* $b$":

$$Sp = (ab + aab)^*a^*$$

If $b \in \Sigma_u$ then $aaa \in Sp$, but $aaab \in \mathsf{Path}(P) \setminus Sp$.

## $P$ controllable, $Sp$ regular (reachability)

$L(P) = (a^+b)^*$, $S = a^+b$.
If $\Sigma_c = \{a\}$ then $P$ is controlable (don't allow any $a$ after $b$). We have $\mathsf{Pref}(Sp)\Sigma_u \cap \mathsf{Path}(P) \subseteq \mathsf{Pref}(Sp)$ and $\mathsf{Pref}(Sp) \cap L(P) \subseteq Sp$.

## Safety case

- Build the product between $P = \langle S, \Sigma, \delta, s_0 \rangle$ and a DFA $\mathcal{A} = \langle Q, \Sigma, \Delta, q_0 \rangle$ for $Sp$.
- Repeat the following:

  If there is some $(s, q) \in S \times Q$ and some $a \in \Sigma_u$ such that $\delta(s, a)$ is defined but $\Delta(q, a)$ is not: remove $(s, q)$ and all outgoing edges.

The condition $Sp\Sigma_u \cap \mathsf{Path}(P) \subseteq Sp$ means that no state is removed. The obtained (sub)automaton $C$ is the maximal controller for $P$ wrt. $Sp$.

## Maximal controller

- Suppose $Sp \subseteq \mathsf{Path}(P)$. If $Sp$ does not satisfy the property $Sp\Sigma_u \cap \mathsf{Path}(P) \subseteq Sp$ then we define the maximal subset $Sp_{\max} \subseteq Sp$ that satisfies it: $Sp_{\max}$ is the union of all $K \subseteq Sp$ satisfying $\mathsf{Pref}(K)\Sigma_u \cap \mathsf{Path}(P) \subseteq \mathsf{Pref}(K)$.
- We can compute $S_{\max}$ as a greatest fixpoint of the following operator $\mathcal{F} : 2^{\Sigma^*} \to 2^{\Sigma^*}$:

$$\mathcal{F}(K) = \max\{J \subseteq K \mid \mathsf{Pref}(J)\Sigma_u \cap \mathsf{Path}(P) \subseteq \mathsf{Pref}(K)\}$$

**Prop.**

Assume every state in $P$ is co-accessible from the accepting states and $Sp \subseteq L(P)$. There exists some controller $C$ with $L_C(P) = Sp$ and such that the controlled plant is non-blocking iff

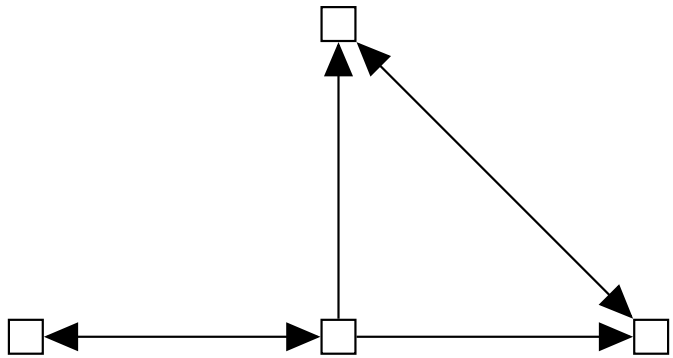
- $\mathsf{Pref}(Sp)\Sigma_u \cap \mathsf{Path}(P) \subseteq \mathsf{Pref}(Sp)$, and
- $\mathsf{Pref}(Sp) \cap L(P) = Sp$.

If $C$ exists, then a finite-memory one exists.

# Distributed control

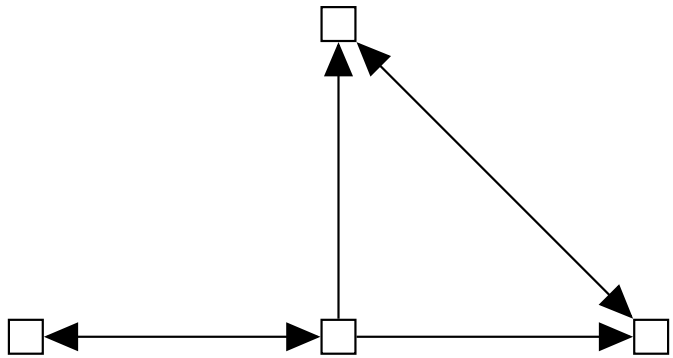
## Models

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



## Models

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

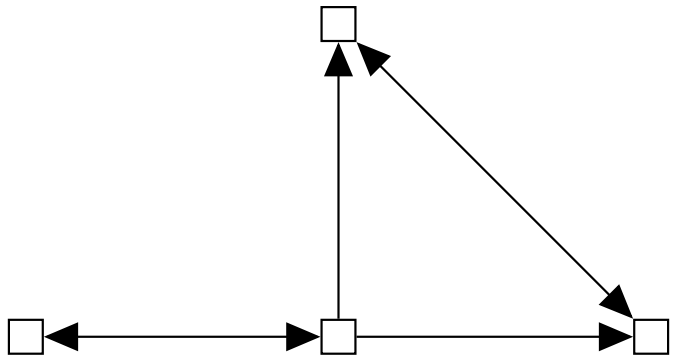


## Links as channels

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

## Models

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



## Links as channels

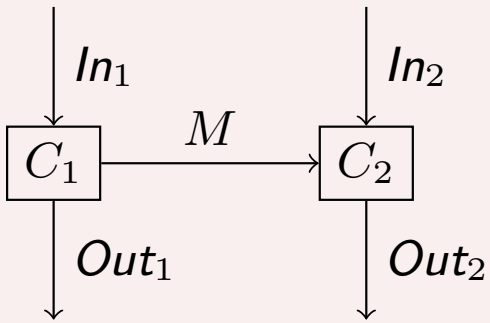
Links are channels and processes have send and receive operations:
communicating automata, message sequence charts. Turing powerful.

## Links as synchronization

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

## Distributed control: Pnueli & Rossner

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

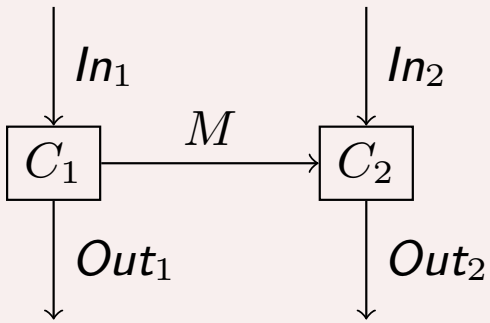



specification $S \subseteq A^*$
$A = In_1 \times In_2 \times M \times Out_1 \times Out_2$

- Control problem: given an architecture over $n$ processes and a regular tree language $S$ over $A$, decide if there exist devices $C_1, \ldots, C_n$ such that the tree of all behaviors is in $S$.

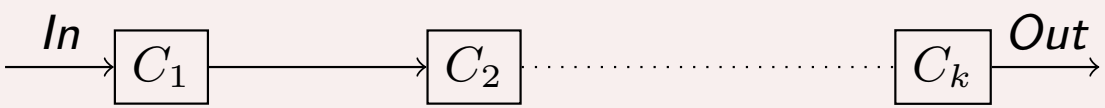
## Distributed control: Pnueli & Rossner

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

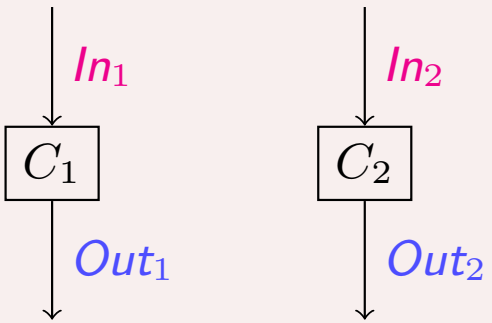



specification $S \subseteq A^*$
$A = In_1 \times In_2 \times M \times Out_1 \times Out_2$

- Control problem: given an architecture over $n$ processes and a regular tree language $S$ over $A$, decide if there exist devices $C_1, \ldots, C_n$ such that the tree of all behaviors is in $S$.
- Control problem is decidable iff the architecture is a pipeline.
  Complexity: non-elementary.

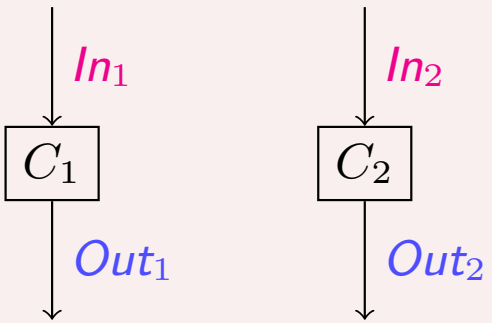


## Undecidability




- Partial information: each process knows only its input. Specification talks about $In/Out$ of both processes.
- Specification fixes the information exchanged by controllers.
- Specification does not take into account the architecture.

## Undecidability



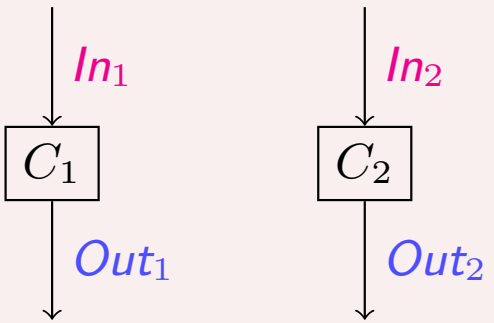

$a_1 - 0 - b_1 - 1 \cdots$

$a_2 - 1 - b_2 - 0 \cdots$

- Partial information: each process knows only its input. Specification talks about *In*/*Out* of both processes.
- Specification fixes the information exchanged by controllers.
- Specification does not take into account the architecture.

$a_1 a_2 01 b_1 b_2 10 \ldots + a_1 0 a_2 b_1 11 a_3 b_2 \ldots$

## Undecidability




$a_1 - 0 - b_1 - 1 \cdots$

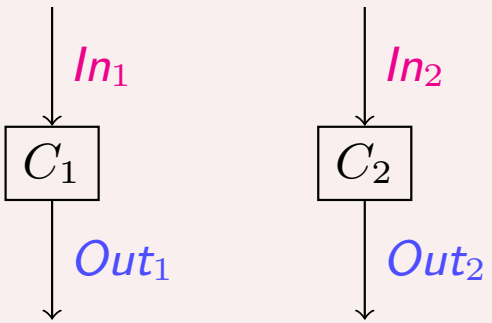
$a_2 - 1 - b_2 - 0 \cdots$

- Partial information: each process knows only its input. Specification talks about *In*/*Out* of both processes.
- Specification fixes the information exchanged by controllers.
- Specification does not take into account the architecture.

$$a_1 a_2 01 b_1 b_2 10 \ldots + a_1 0 a_2 b_1 11 a_3 b_2 \ldots$$

- Undecidable even for local specifications [Madhusudan & Thiagarajan 2001].

## Undecidability




- Partial information: each process knows only its input. Specification talks about $In/Out$ of both processes.
- Specification fixes the information exchanged by controllers.
- Specification does not take into account the architecture.
- Undecidable even for local specifications [Madhusudan & Thiagarajan 2001].

## Pipeline

Decidability: tree automata. Important: controllers exchange fixed amount of information.

## Proof

- Reduction from the question whether a TM $M$ has an infinite run from the empty tape. Configurations of $M$ are encoded by words from $\Gamma^*(\Gamma \times Q)\Gamma^*$ ($\Gamma$ tape alphabet, $Q$ set of states).
- Alphabets $In_i = \{\#, \$\}$ and $Out_i = \Gamma \cup (\Gamma \times Q) \cup \{\#, \$\}$, $i = 1, 2$.
- Specification $S = S_= \cup S_\vdash$:
  - $S_=$: consists of 4-tuples $(x_1, x_2, y_1, y_2) \in In_1^* \times In_2^* \times Out_1^* \times Out_2^*$ of the form below, with $i \geq 0$ and $\alpha$ configuration of $M$,

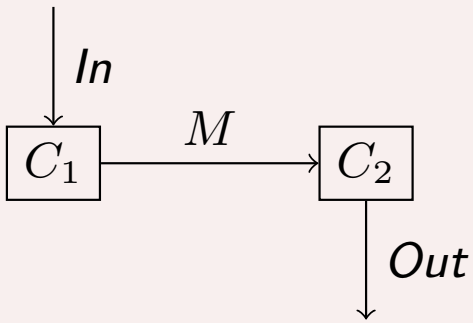
    $$x_1 = x_2 = \#^i\$^\omega, \quad y_1 = y_2 = \#^i\alpha\,\$^\omega.$$

    In addition, if $i = 0$ then $\alpha$ is the initial configuration.
  - $S_\vdash$: consists of 4-tuples $(x_1, x_2, y_1, y_2) \in In_1^* \times In_2^* \times Out_1^* \times Out_2^*$, with $i \geq 0$ and $\alpha, \beta$ configurations of $M$ with $\alpha \vdash \beta$,

    $$x_1 = \#^i\$^\omega,\ x_2 = \#^{i+1}\$^\omega,\ y_1 = \#^i\alpha\,\$^\omega,\ y_2 = \#^{i+1}\beta\,\$^\omega.$$

  Note that $S$ is a regular $\omega$-language over $In_1 \times In_2 \times Out_1 \times Out_2$.
- The only way for $C_1, C_2$ to behave according to $S$ is to produce an infinite run $\alpha_0 \vdash \alpha_1 \vdash \cdots$ of $M$: let $\alpha_i$ be the configuration produced by $C_1$ on $\#^i\$^\omega$, and $\beta_i$ the configuration produced by $C_2$ on $\#^i\$^\omega$. Then $\alpha_0$ must be initial (because of $S_=$). Moreover $\alpha_i = \beta_i$ (because of $S_=$) and $\alpha_i \vdash \beta_{i+1}$ (because of $S_\vdash$), for every $i$.

## $n = 2$




- $C_1 : In^* \to M$, $C_2 : M^* \to Out$.
- Joint behavior $(C_1, C_2) : In^* \to (M \times Out)$ is the composition of $C_1, C_2$ ($w = a_0 a_1 \cdots a_n$):

$$(C_1, C_2)(w) = (C_1(w), C_2(C_1(a_0)C_1(a_0 a_1) \cdots C_1(a_0 \cdots a_n)))$$

- Search for $C_2 : M^* \to Out$, $C_1 : In^* \to M$ such that $(C_1, C_2)$ satisfy some regular tree specification $S$ for functions $f : In^* \to (M \times Out)$.

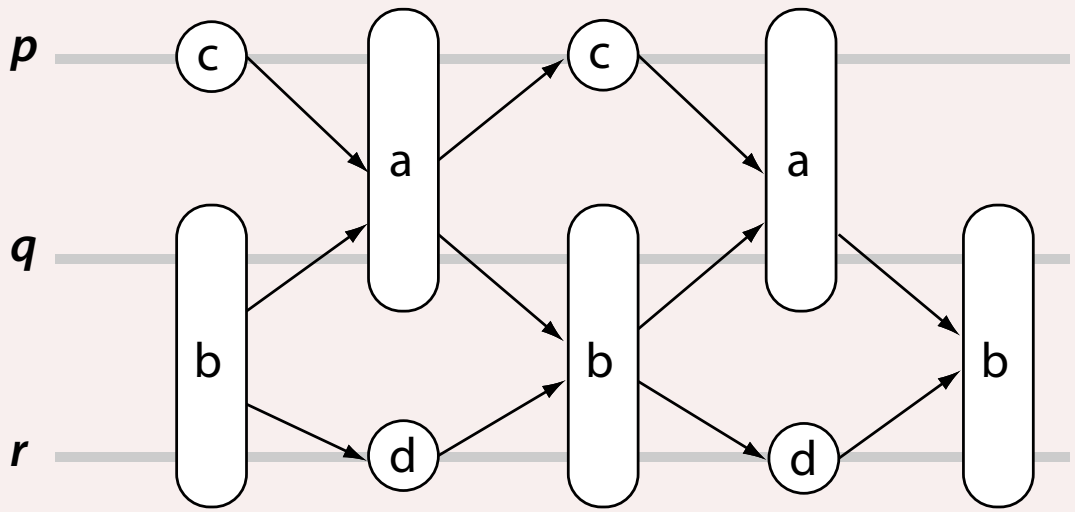
## Solution

- If $S : In^* \to (M \times Out)$ is a regular tree specification for functions $C : In^* \to (M \times Out)$, then

$$\{C_2 : M^* \to Out \mid \exists C_1 : In^* \to M \text{ s.t. } (C_1, C_2) \in S\}$$

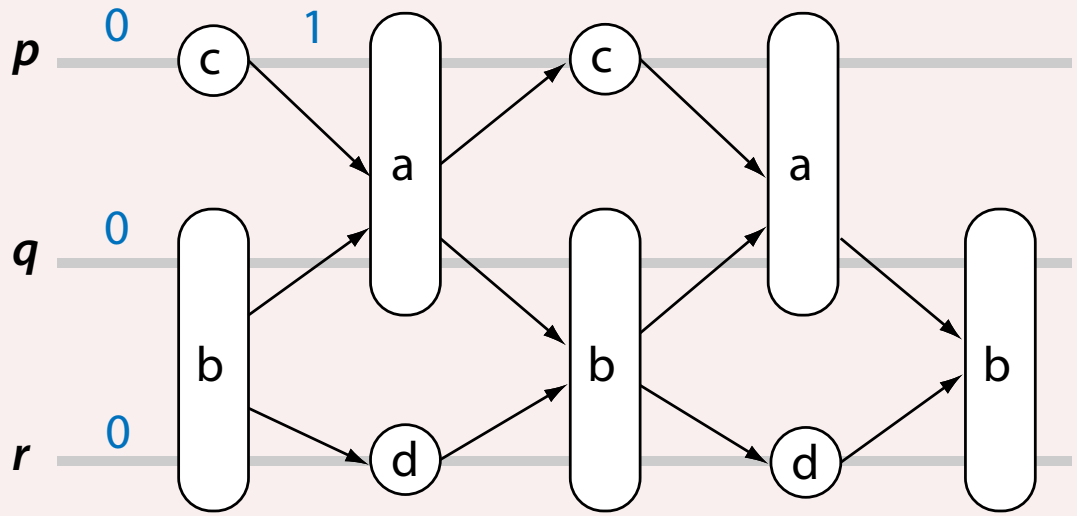
  is a regular tree specification $S'$ for functions $C_2 : M^* \to Out$ and a tree automaton for $S'$ can be constructed effectively from a tree automaton for $S$. Exponential blow-up from $S$ to $S'$ (projection).
- Test emptiness for tree automaton accepting $S'$.

# Distributed control:
## asynchronous case



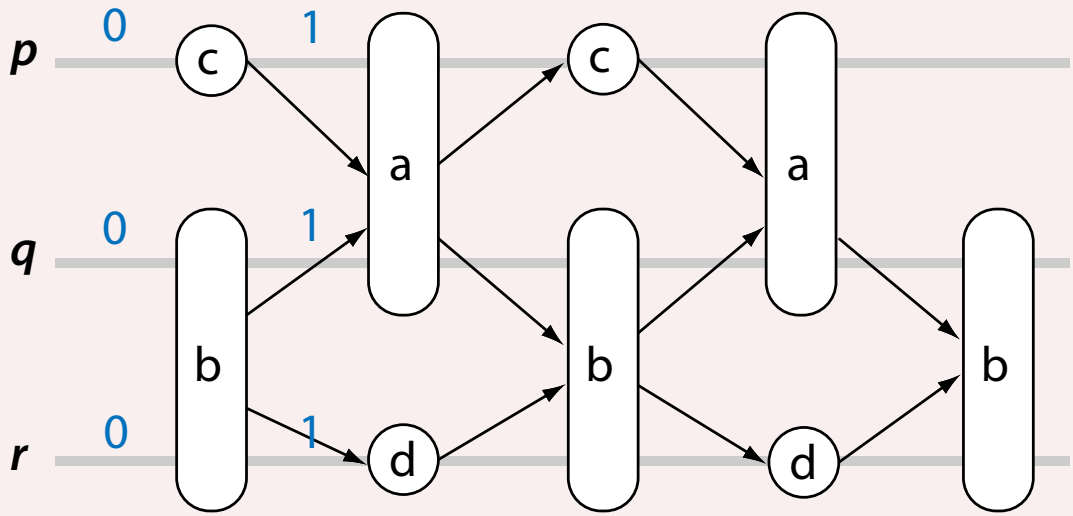

- Local states sets $S_p, S_q, S_r$.
- Local transitions $\delta_b : S_q \times S_r \to S_q \times S_r$, ...

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



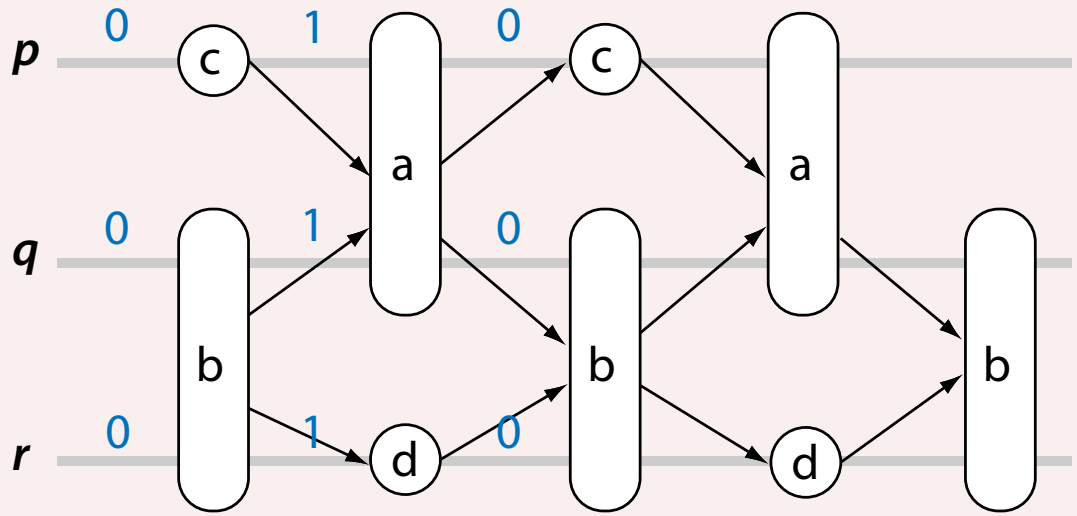

- Local states sets $S_p, S_q, S_r$.
- Local transitions $\delta_b : S_q \times S_r \to S_q \times S_r$, ...

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




- Local states sets $S_p, S_q, S_r$.
- Local transitions $\delta_b : S_q \times S_r \to S_q \times S_r$, ...

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




- Local states sets $S_p, S_q, S_r$.
- Local transitions $\delta_b : S_q \times S_r \to S_q \times S_r$, ...

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

## Distributed automata

- $\mathbb{P}$: finite set of processes, each $p \in \mathbb{P}$ has its set of states $S_p$
- distributed alphabet $\langle \Sigma, \mathrm{dom} : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset) \rangle$
  $\mathrm{dom}(a)$ = set of processes involved in action $a$
- transition functions $\delta_a : \prod_{p \in \mathrm{dom}(a)} S_p \to \prod_{p \in \mathrm{dom}(a)} S_p$
- Let $I = \{(a,b) \mid \mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset\}$ (independence relation).
  Example: $I = \{(a,d),(d,a),(b,c),(c,b),(c,d),(d,c)\}$.
- Mazurkiewicz trace $[u]$ = set of all words that can be obtained from $u \in \Sigma^*$ by commuting independent adjacent letters ($I$-commutations).
  Example: $[adbc] = \{adbc, dabc, dacb, adcb\}$.

## Product automaton

Given a distributed automaton $\mathcal{A} = \langle (S_p)_{p \in \mathbb{P}}, \Sigma, \mathrm{dom}, (\delta_a)_{a \in \Sigma} \rangle$, we associate the product automaton $\mathcal{B} = \langle \prod_{p \in \mathbb{P}} S_p, \Sigma, \Delta \rangle$:

$$(s_p)_{p \in \mathbb{P}} \xrightarrow{a}_{\Delta} (s'_p)_{p \in \mathbb{P}} \text{ if } (s'_p)_{p \in \mathrm{dom}(a)} \in \delta_a((s_p)_{p \in \mathrm{dom}(a)}) \text{ and } s'_q = s_q \text{ for all } q \notin \mathrm{dom}(a).$$

In addition, $\mathcal{A}$ has a set of final states $F \subseteq \prod_{p \in \mathbb{P}} S_p$.

## Language

The language $L(\mathcal{A})$ of a distributed automaton $\mathcal{A}$ is the (regular) language of the product automaton $\mathcal{B}$.

Note that if $[u] = [v]$ then $u \in L(\mathcal{B})$ iff $v \in L(\mathcal{B})$. So $L(\mathcal{A})$ can be viewed either as a language of pomsets (labelled partial orders corresponding to Mazurkiewicz traces), or as a word language closed under $I$-commutations.

## Language

The language $L(\mathcal{A})$ of a distributed automaton $\mathcal{A}$ is the (regular) language of the product automaton $\mathcal{B}$.

Note that if $[u] = [v]$ then $u \in L(\mathcal{B})$ iff $v \in L(\mathcal{B})$. So $L(\mathcal{A})$ can be viewed either as a language of pomsets (labelled partial orders corresponding to Mazurkiewicz traces), or as a word language closed under $I$-commutations.

## Zielonka's theorem

Each regular language closed under $I$-commutations can be accepted by a deterministic distributed automaton (of exponential size in $\mathbb{P}$).

## $I$-diamond property

- A DFA has the $I$-diamond property if for every state $s$, and every $(a, b) \in I$: if $\delta(s, ab)$ is defined, then $\delta(s, ba)$ is also defined and equal to $\delta(s, ab)$.
- The minimal DFA of a regular language closed under $I$-commutations has the $I$-diamond property.

## Lemma

Given an $I$-diamond DFA $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$, there is a function $\mathcal{D} : S^3 \times 2^{\mathbb{P}} \to S$ such that for every three states $s_0, s_1, s_2 \in S$ and every $X \subseteq \mathbb{P}$, the state $s = \mathcal{D}(s_0, s_1, s_2, X)$ satisfies the following property:

> *For all $u, v, w \in \Sigma^*$ with dom$(v) \subseteq X$ and dom$(w) \subseteq \mathbb{P} \setminus X$, if $\mathcal{D}(s^0, u) = s_0$, $\mathcal{D}(s^0, uv) = s_1$ and $\mathcal{D}(s^0, uw) = s_2$, then $\mathcal{D}(s^0, uvw) = s$.*

## Setting

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

  *Vertex set* $= \mathbb{P}$*; edges* $\{p, q\}$ *if there is some* $a \in \Sigma$ *with* $dom(a) = \{p, q\}$.

  Wlog. $CG$ is a tree.
- Input: the minimal DFA $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$ for a regular language $L$ closed under $I$-commutations.
- We will build a distributed automaton $\mathcal{B} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma} \rangle$ with $L(\mathcal{B}) = L(\mathcal{A})$ and each $|S_p| = |S|^2$.

## Construction

- For every process $p \in \mathbb{P}$: the set of local states is $S^2$. We write $\langle s, s' \rangle_p$ to denote a local state of $p$.
  *Invariant:* the first state stored by $p$ is the state at which it last synchronized with its parent in $CG$; the second state of $p$ stores the state reached by the automaton $\mathcal{A}$ on the current $p$-view.

- The start state of $p$ is $(s_{in})_p = \langle s^0, s^0 \rangle_p$.
- The transition function $\delta_a$ for each action $a \in \Sigma$ is defined as follows:
  - if $\mathrm{dom}(a) = \{p\}$ then $\delta_a(\langle s, s' \rangle_p) = \langle s, \Delta(s', a) \rangle_p$.
  - if $\mathrm{dom}(a) = \{p, q\}$ and $p$ is the parent of $q$, then

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

    where $s' = \Delta(s, a)$, $s = \mathcal{D}(s_2, s_2', s_1', X(q))$ and $X(q) \subseteq \mathbb{P}$ is the subtree of $CG$ rooted at $q$.
- Final states $F \subseteq \prod_{p \in \mathbb{P}} S_p$: later.

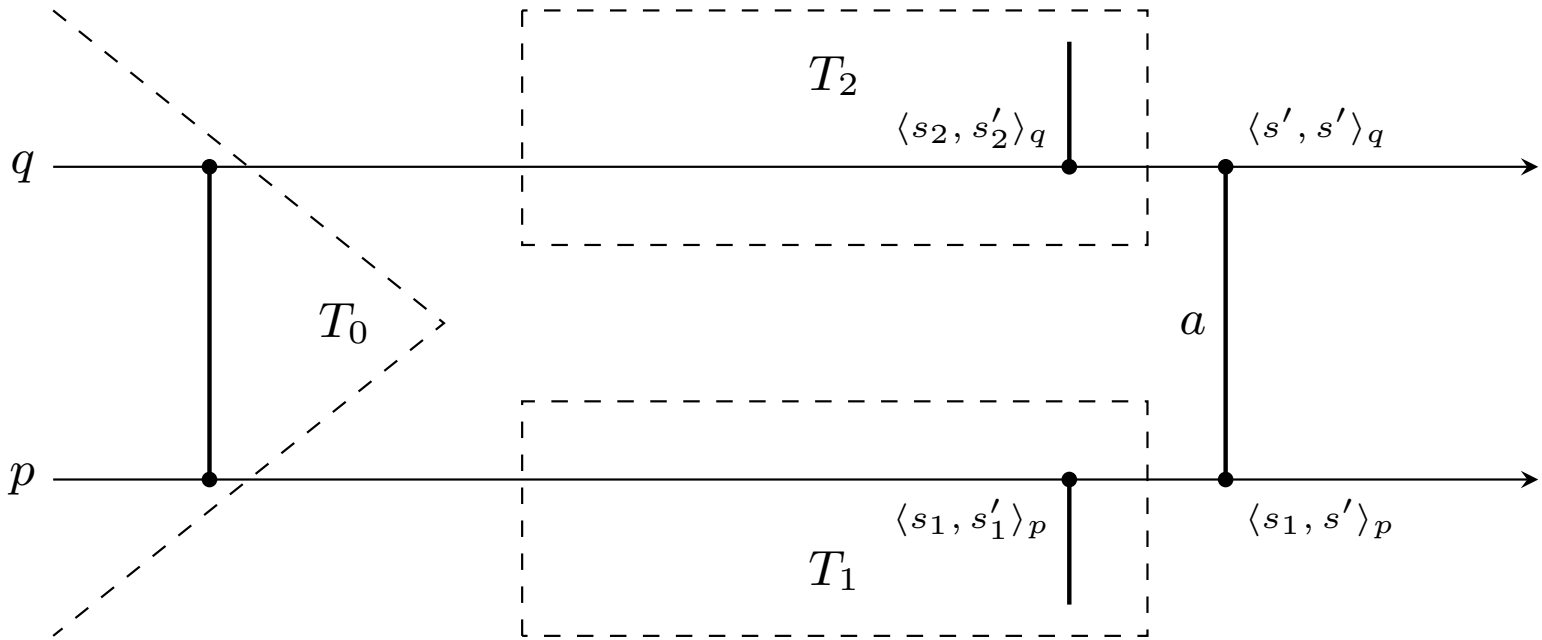



Figure: Inductive step: $s' = \Delta(s, a)$, $s = \mathcal{D}(s_2, s_2', s_1', X(q))$.

## Final states

- If $\mathbb{P} = \{p\}$ then $\langle s^0, s \rangle_p$ is final if $s \in F$.
- Otherwise, start at the root $p$, and apply the following process recursively to each subtree. Then if $p$ is in state $\langle s_1, s_1' \rangle_p$ and its child $q$ in state $\langle s_2, s_2' \rangle_q$, update the state $s_1'$ to be the state $s_3 = \mathcal{D}(s_2, s_2', s_1', X(q))$, where $X(q) \subseteq \mathbb{P}$ is the subtree of $CG$ rooted at $q$. When this is done for all the children of $p$, look at its state $\langle s, s' \rangle_p$ and call the original state of $\mathcal{B}$ accepting if and only if $s' \in F$.

## Ramadge and Wonham formulation

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

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

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

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

- Each process $p$ has its control strategy $\sigma_p$.
- Views of $p$: for $u \in \Sigma^*$ let $\mathit{view}_p(u)$ be such that

$$[u] = [\mathit{view}_p(u)\, v]$$

for some $v \in \Sigma^*$ containing no action $a$ from $\Sigma_p = \{b \mid p \in \mathrm{dom}(b)\}$.

$$\mathit{Views}_p(\mathcal{A}) = \{\mathit{view}_p(u) \mid u \in L(\mathcal{A})\}$$

- $p$'s strategy $\sigma_p$ depends on the history (view) of $p$:

$$\sigma_p : \mathit{Views}_p(\mathcal{A}) \to 2^{\Sigma_p^{sys}}$$

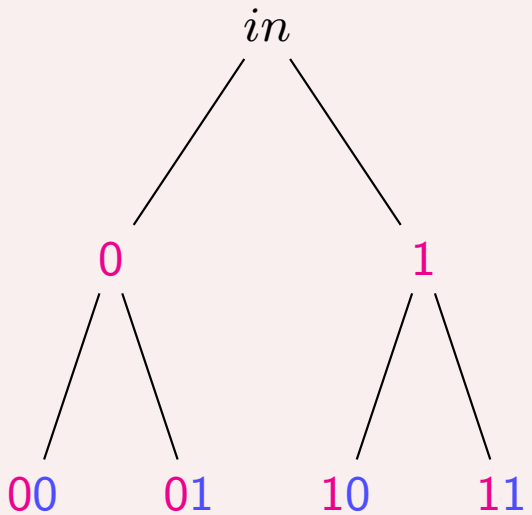
## $\sigma$-plays on $\mathcal{A}$

- If $t$ is a $\sigma$-play and $ta \in L(\mathcal{A})$ with $a$ uncontrollable, then $ta$ is a $\sigma$-play.
- If $t$ is a $\sigma$-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 $\sigma$-play.

## Winning condition: local reachability $(F_p)_{p \in \mathbb{P}}$

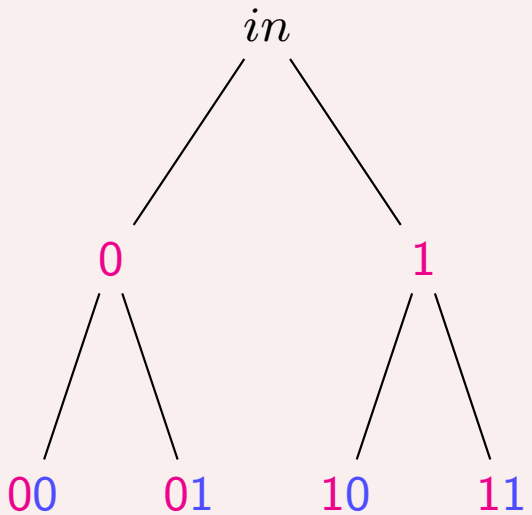
A maximal $\sigma$-play is winning if every process (ultimately) reaches a final state.
A strategy $\sigma$ is winning if all maximal $\sigma$-plays are winning.

## Game




- Two processes $P_0, P_1$.
- $P_i$ receives an uncontrolable bit $u_i$ and has to produce a controlable bit $c_i$.
- Winning equals $c_1 = u_0$ or $c_0 = u_1$.

## Game



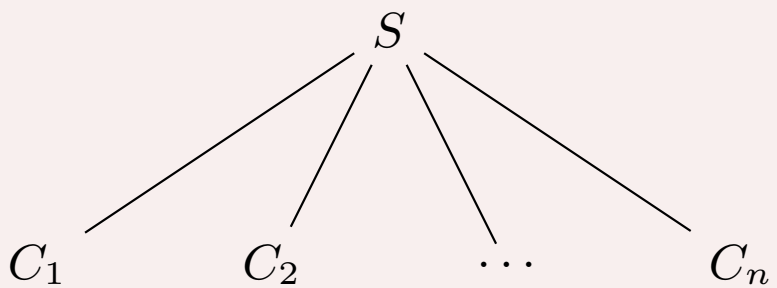

- Two processes $P_0, P_1$.
- $P_i$ receives an uncontrolable bit $u_i$ and has to produce a controlable bit $c_i$.
- Winning equals $c_1 = u_0$ or $c_0 = u_1$.

## How to win

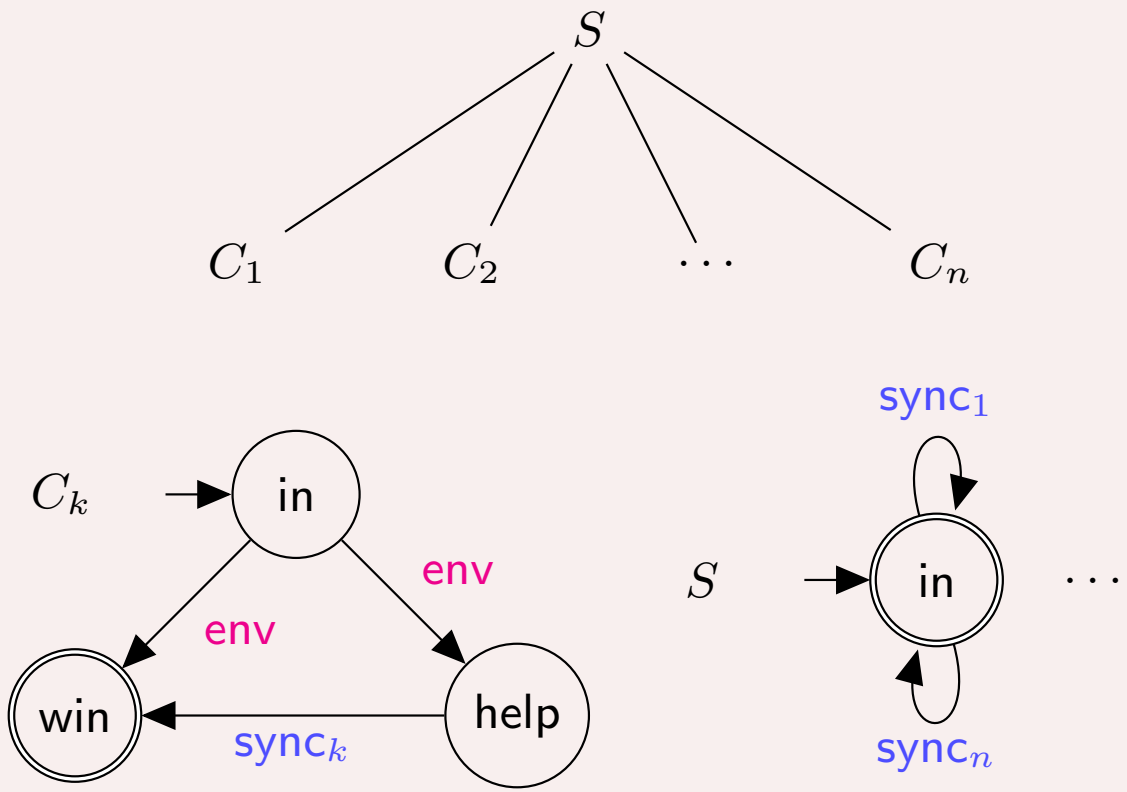
Distributed strategy: $P_0$ plays $c_0 = u_0$ and $P_1$ plays $c_1 = 1 - u_1$.
Winning, since either $u_0 = u_1$, so $u_1 = c_0$. Or $u_1 = 1 - u_0$, so $u_0 = c_1$.

# Example 2

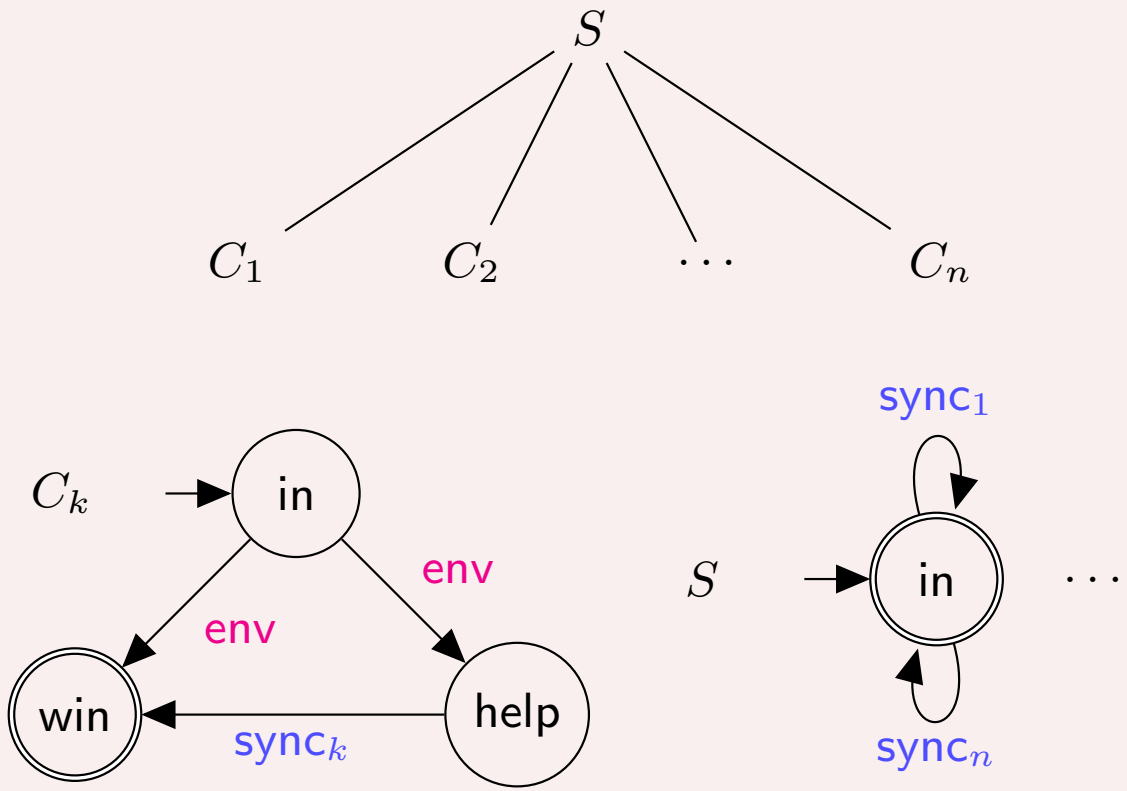
## Game



## Game



## Game




Winning strategy of $S$: offer synchronization with every $C_k$

## [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.

With the above restrictions strategies can be represented by Mazurkiewicz traces.

## [Gastin & Lerman & Zeitoun 2004]

Decidability for restricted alphabets of actions: co-graphs. Induction over alphabet.

## [Madhusudan & Thiagarajan 2005]

- Decidability for Zielonka automata satisfying a strong synchronization restriction ("there is no loop with a concurrent event").
- The above restriction suffices for getting decidability of MSO over the branching structure of the automaton. In this case, the structure is very much "tree-like".

## Communication graph

- We restrict $\Sigma$ to actions which are either local or binary.
- Communication graph:
  Vertex set $= \mathbb{P}$; edges $\{p, q\}$ if there is some $a \in \Sigma$ with $\mathsf{dom}(a) = \{p, q\}$.

## Theorem (Genest, Gimbert, M., Walukiewicz, 2011)

- The distributed control problem is decidable if the communication graph is acyclic.
- Exact complexity: non-elementary in the depth of the architecture.

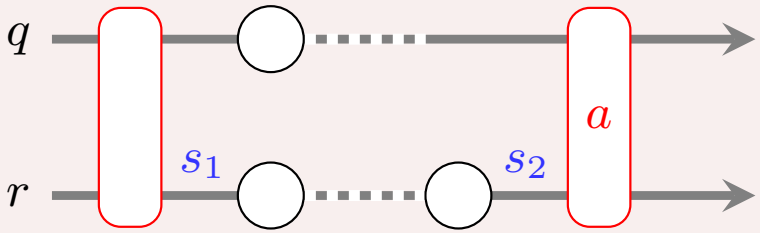
## Game simulation: reduce the number of processes

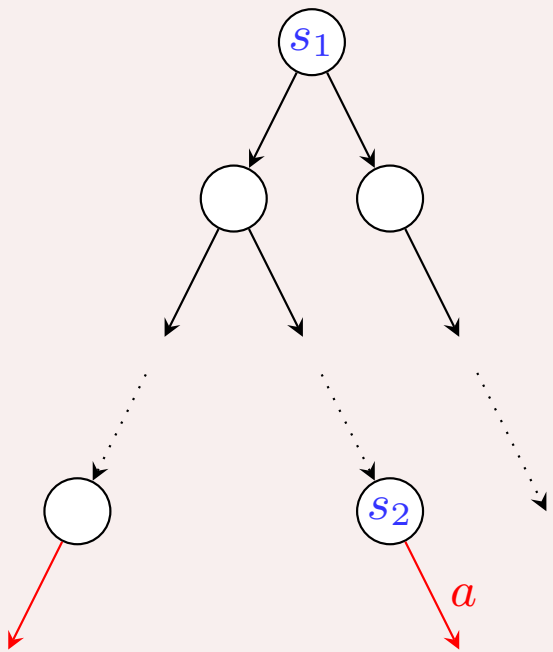
- Fix a leaf process $r$. The parent process $q$ guesses a local strategy $T$ for $r$ resuming $r$'s local behavior until the next synchronization with $q$.
- Uncontrolable actions can challenge the guess.

## Game simulation: reduce the number of processes

- Fix a leaf process $r$. The parent process $q$ guesses a local strategy $T$ for $r$ resuming $r$'s local behavior until the next synchronization with $q$.
- Uncontrolable actions can challenge the guess.

## Simulate $r$ by $q$




Process $q$ guesses $T : S_r \to 2^{\Sigma_{q,r}}$.
Uncontrolable actions $(a, s_2)$, where $a \in T(s_2)$.

## Setting

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

## Lemma

Any winning strategy $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ is such that $\sigma_p$ always proposes either one local $p$-action, or a set of communications of $p$ with its neighbors.

## Setting

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

## Lemma

Any winning strategy $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ is such that $\sigma_p$ always proposes either one local $p$-action, or a set of communications of $p$ with its neighbors.

## Local plays and strategies

- $\Sigma_r^{loc}$ = set of actions $a$ with $\mathrm{dom}(a) = \{r\}$.
- Local $r$-play: word from $(\Sigma_r^{loc})^*$.
- $r$-context: play ending in $\Sigma_{q,r}$.
- Local $r$-strategy $\sigma_r[t](\Sigma_r^{loc})^* \to 2^{\Sigma_r}$ from $r$-context $t$:

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

## Setting

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

## Lemma

Any winning strategy $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ is such that $\sigma_p$ always proposes either one local $p$-action, or a set of communications of $p$ with its neighbors.

## Local plays and strategies

- $\Sigma_r^{loc}$ = set of actions $a$ with $\mathrm{dom}(a) = \{r\}$.
- Local $r$-play: word from $(\Sigma_r^{loc})^*$.
- $r$-context: play ending in $\Sigma_{q,r}$.
- Local $r$-strategy $\sigma_r[t](\Sigma_r^{loc})^* \to 2^{\Sigma_r}$ from $r$-context $t$:

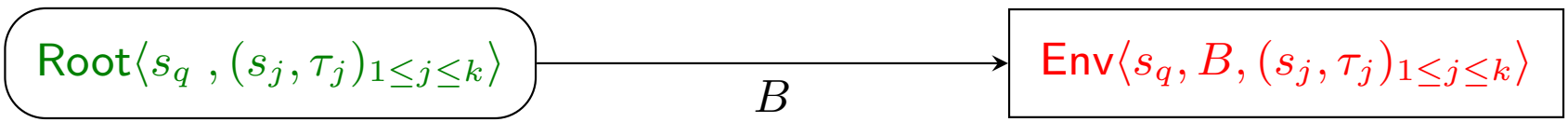
$$\sigma_r[t](x) := \sigma_r(tx) \text{ for all } x \in (\Sigma_r^{loc})^*$$

## Key lemma

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

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

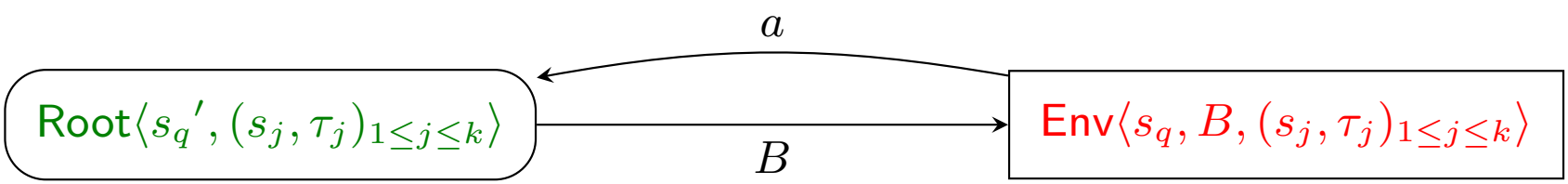
- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$




$\forall B \subseteq \Sigma_q^{sys}$

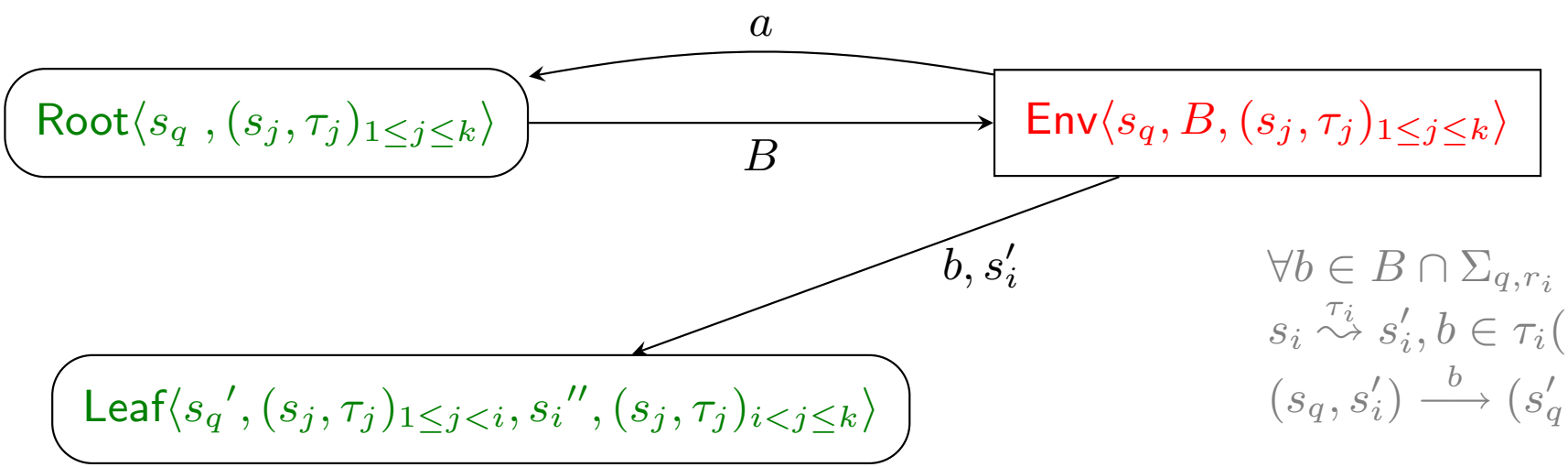
- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$

$$\forall a \in (B \cap \Sigma_q^{loc}) \cup \Sigma_q^{env} : s_q \xrightarrow{a} s'_q$$



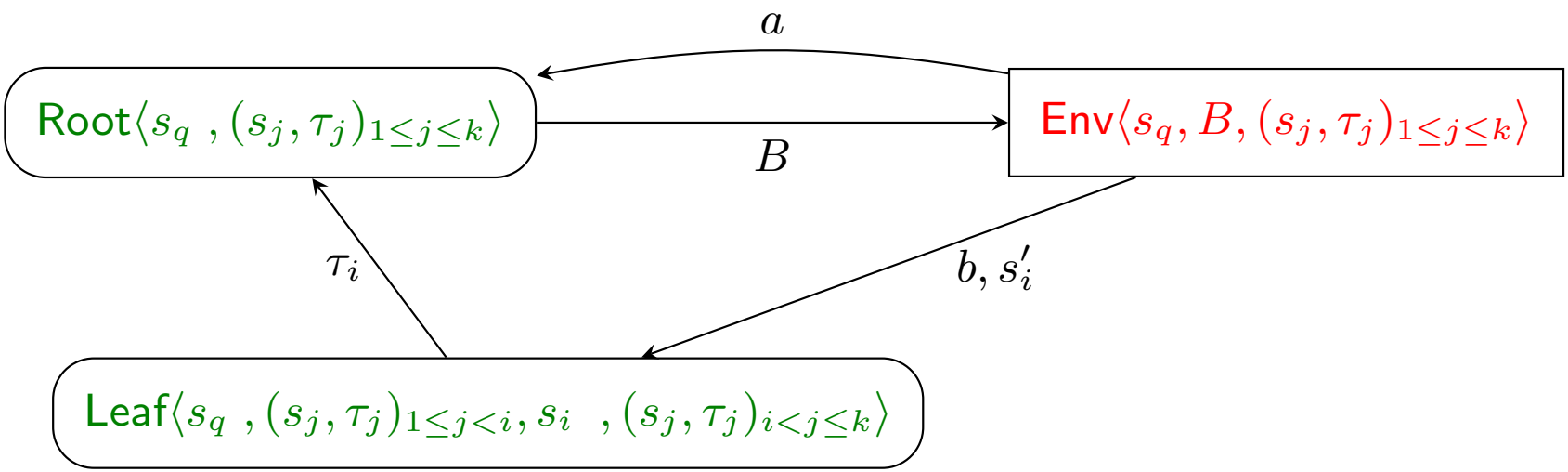

Environment either chooses a local action $a$ for $q$

- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$

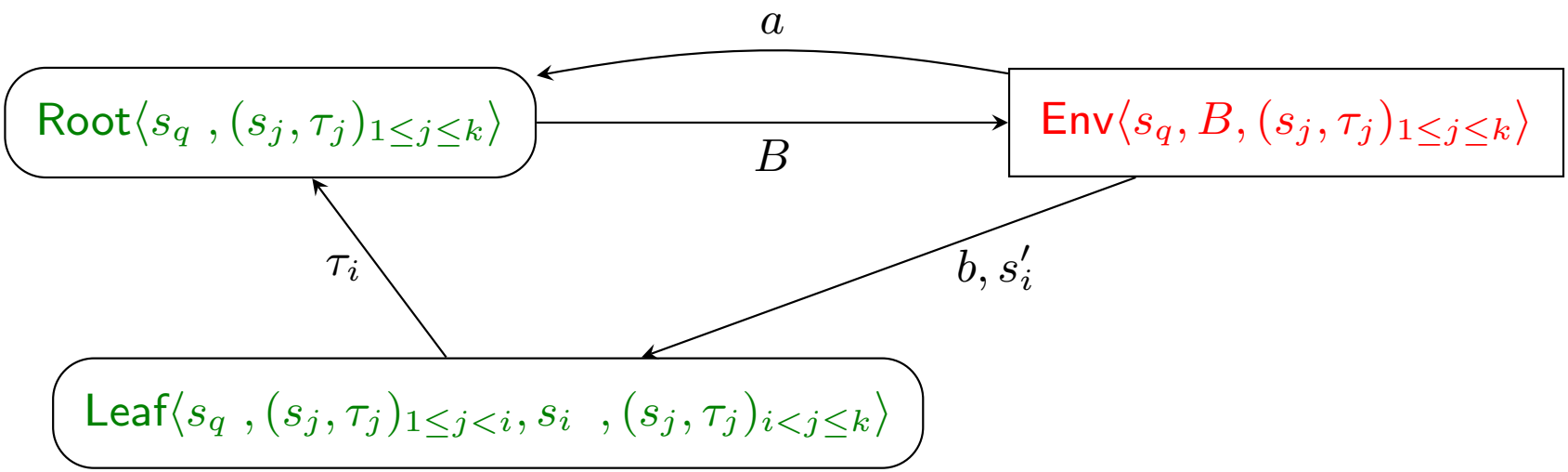



Or chooses a synchronization action $b$ between $q$ and $r_i$.

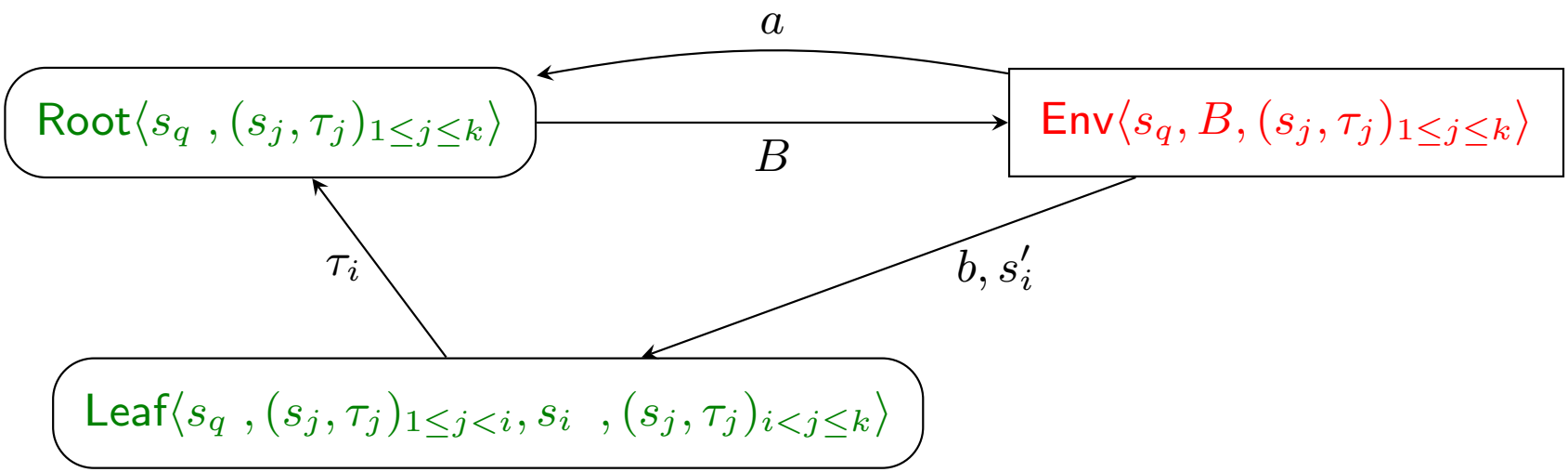
- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$



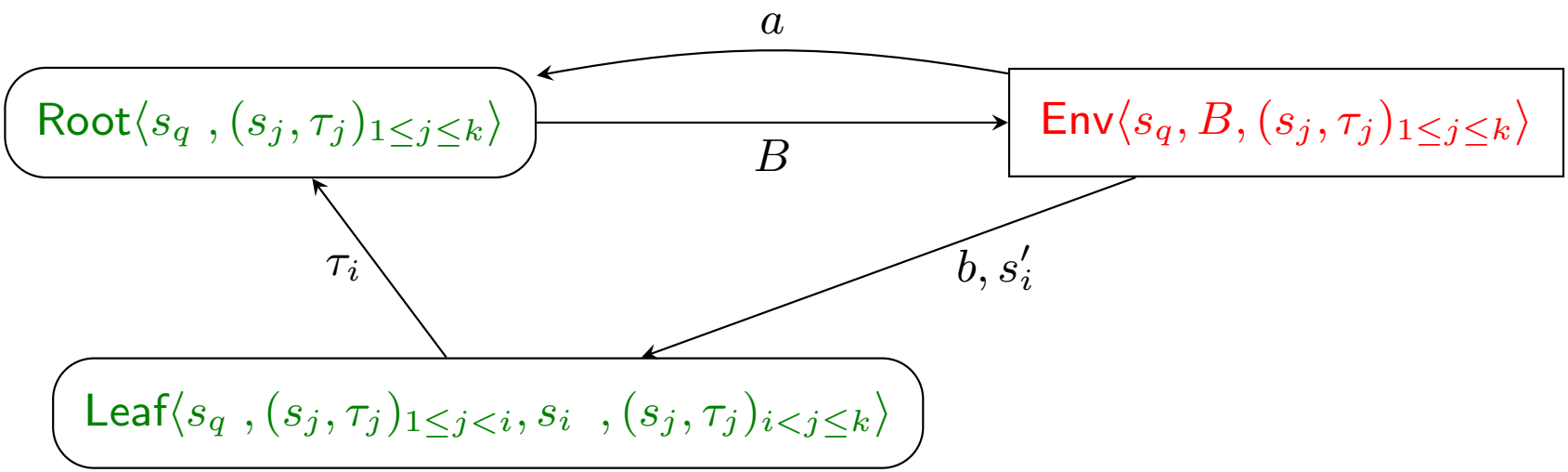

- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$




- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$
- Good $\tau_i$ : every infinite local play from $s_i$ ultimately in $F_{r_i}$.




- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$
- Good $\tau_i$ : every infinite local play from $s_i$ ultimately in $F_{r_i}$.
- Final states: $s_q \in F_q$, all local maximal finite $\tau_i$-plays from $s_i$ end in $F_{r_i}$.




- $s_q \in S_q$, $s_i \in S_{r_i}$, $\tau_i$ local positional strategy for $r_i$
- Good $\tau_i$ : every infinite local play from $s_i$ ultimately in $F_{r_i}$.
- Final states: $s_q \in F_q$, all local maximal finite $\tau_i$-plays from $s_i$ end in $F_{r_i}$.
- Forbid environment states that are non-final deadlocks: (1) $q$ cannot progress locally and (2) each $r_i$ can get into a state (compatible with $\tau_i$) where it cannot synchronize with $q$.