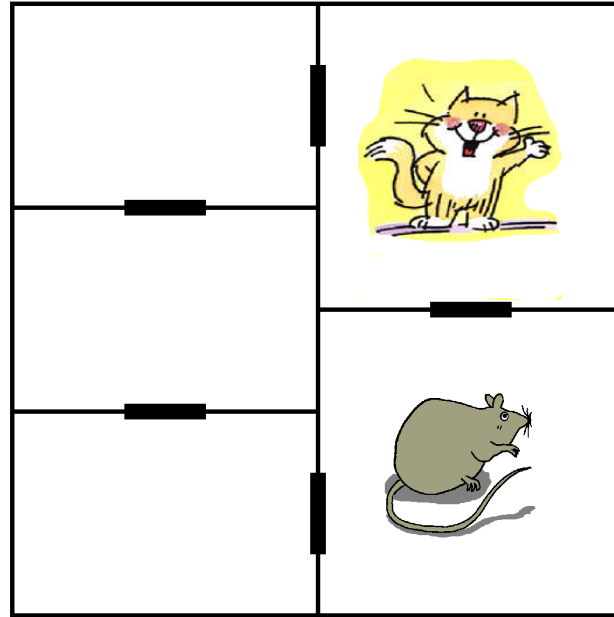
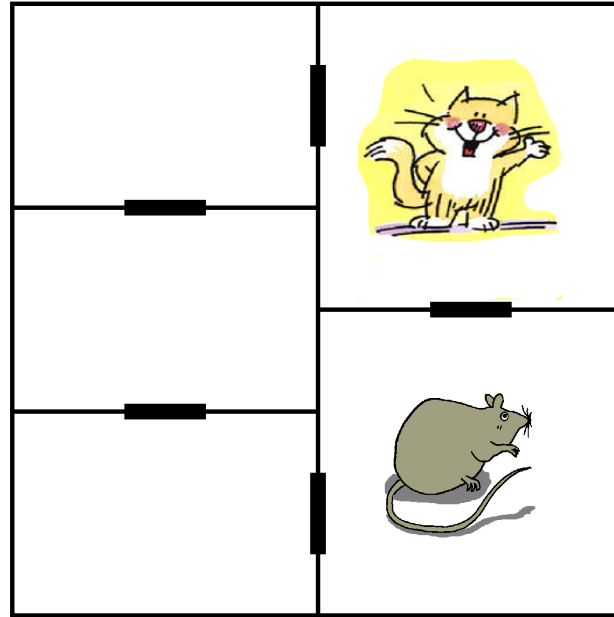

Distributed synthesis with distributed games

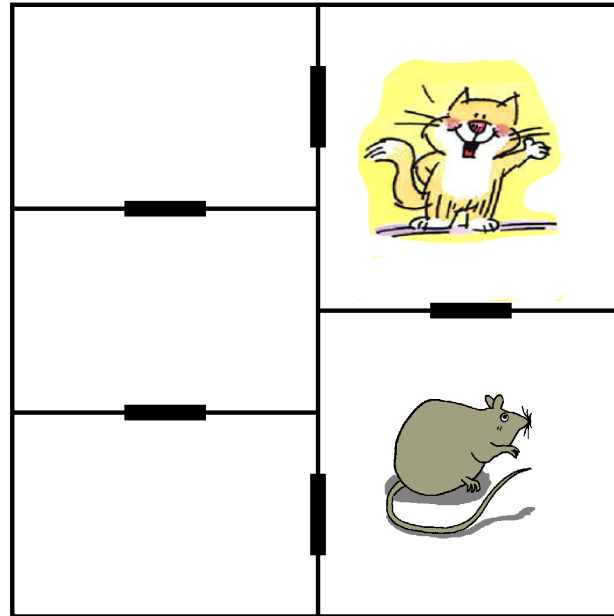
Igor Walukiewicz
LaBRI, Bordeaux University



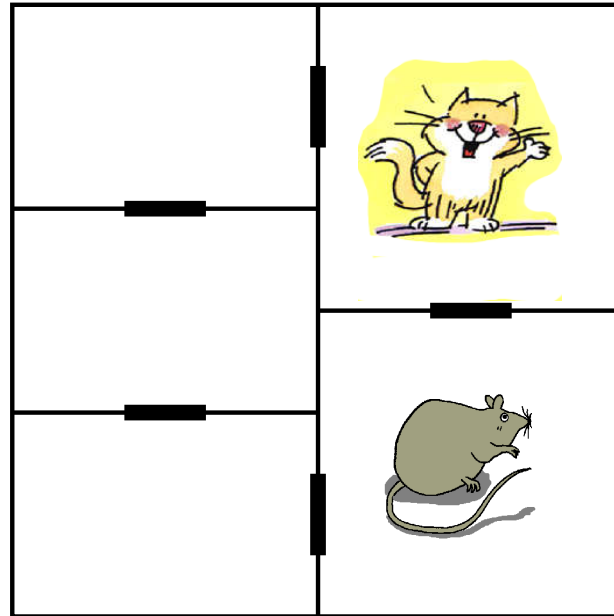
- Control of the doors.



- Control of the doors.
- Observation of the cat.



- Control of the doors.
- Observation of the cat.
- Allow the animals to move as freely as possible.



- Control of the doors.
- Observation of the cat.
- Allow the animals to move as freely as possible.
- Speed of the animals.

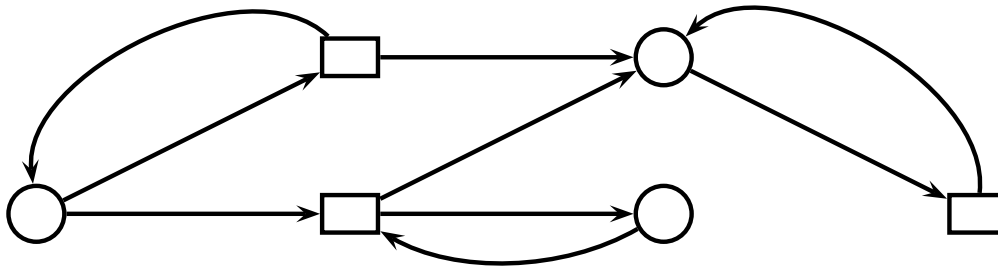
- Synthesis of one controller.
- Distributed synthesis.
- Undecidability.
- Distributed games and their basic properties.
- Examples of coding synthesis problems into games.
 - Pipeline.
 - Communicating systems with local specifications.
- Two theorems to solve these problems.

Part I

Synthesis of one controller

- Parity games.
- A simple synthesis problem.
- Synthesis for a given plant.
- Controllable and observable actions.
- Minimal/maximal controllers.

$$\mathcal{G} = \langle V_0, V_1, T, Acc \subseteq (V_0 \cup V_1)^\omega \rangle$$



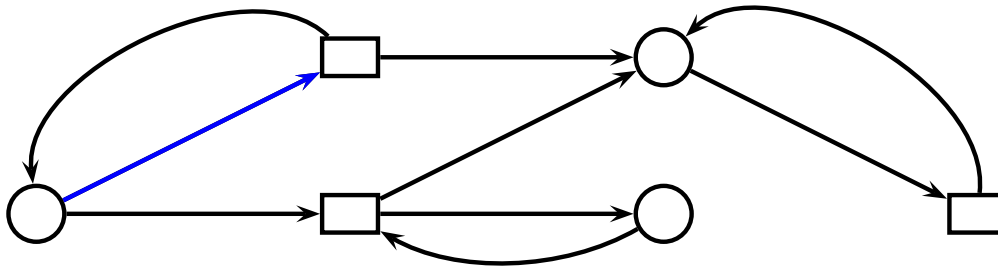
- Player 0 wins if the path is in Acc . (There is an edge from every node.)

Def: **Parity game** is a game where Acc is given by a function $\Omega : V \rightarrow \{0, \dots, d\}$.

$\vec{v} \in Acc$ iff $\min(\text{Inf}_\Omega(\vec{v}))$ is even.

$\text{Inf}_\Omega(v_0v_1 \dots) = \{n : n \text{ appears infinitely often in } \Omega(v_0)\Omega(v_1) \dots\}$

$$\mathcal{G} = \langle V_0, V_1, T, Acc \subseteq (V_0 \cup V_1)^\omega \rangle$$



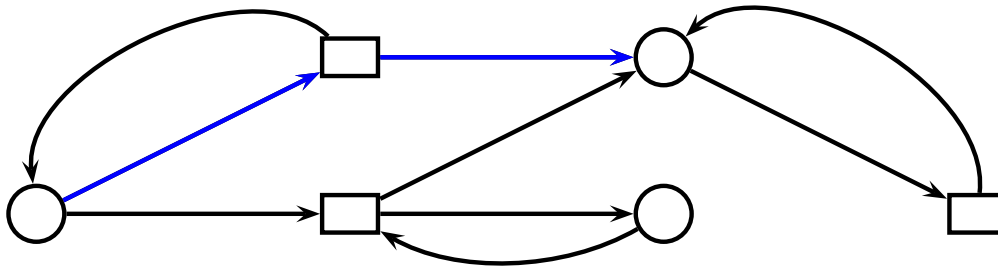
- Player 0 wins if the path is in Acc . (There is an edge from every node.)

Def: **Parity game** is a game where Acc is given by a function $\Omega : V \rightarrow \{0, \dots, d\}$.

$\vec{v} \in Acc$ iff $\min(\text{Inf}_\Omega(\vec{v}))$ is even.

$\text{Inf}_\Omega(v_0v_1 \dots) = \{n : n \text{ appears infinitely often in } \Omega(v_0)\Omega(v_1) \dots\}$

$$\mathcal{G} = \langle V_0, V_1, T, Acc \subseteq (V_0 \cup V_1)^\omega \rangle$$



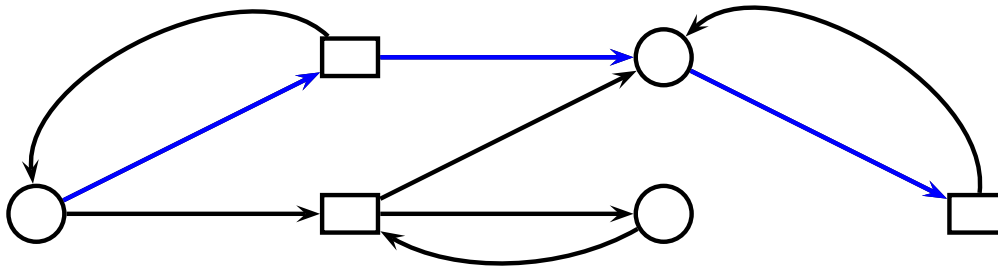
- Player 0 wins if the path is in Acc . (There is an edge from every node.)

Def: **Parity game** is a game where Acc is given by a function $\Omega : V \rightarrow \{0, \dots, d\}$.

$\vec{v} \in Acc$ iff $\min(\text{Inf}_\Omega(\vec{v}))$ is even.

$\text{Inf}_\Omega(v_0v_1 \dots) = \{n : n \text{ appears infinitely often in } \Omega(v_0)\Omega(v_1) \dots\}$

$$\mathcal{G} = \langle V_0, V_1, T, Acc \subseteq (V_0 \cup V_1)^\omega \rangle$$



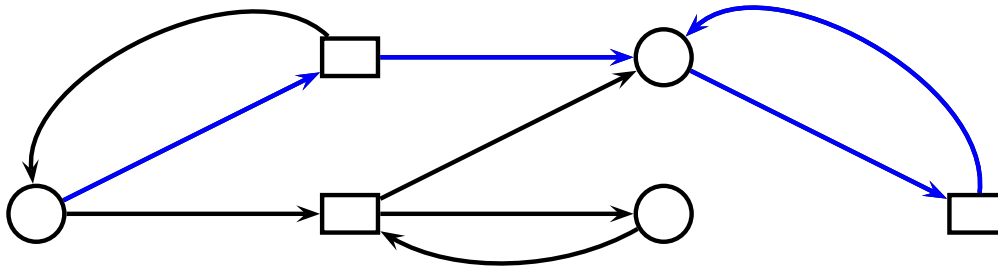
- Player 0 wins if the path is in Acc . (There is an edge from every node.)

Def: **Parity game** is a game where Acc is given by a function $\Omega : V \rightarrow \{0, \dots, d\}$.

$\vec{v} \in Acc$ iff $\min(\text{Inf}_\Omega(\vec{v}))$ is even.

$\text{Inf}_\Omega(v_0v_1 \dots) = \{n : n \text{ appears infinitely often in } \Omega(v_0)\Omega(v_1) \dots\}$

$$\mathcal{G} = \langle V_0, V_1, T, Acc \subseteq (V_0 \cup V_1)^\omega \rangle$$



- Player 0 wins if the path is in Acc . (There is an edge from every node.)

Def: **Parity game** is a game where Acc is given by a function $\Omega : V \rightarrow \{0, \dots, d\}$.

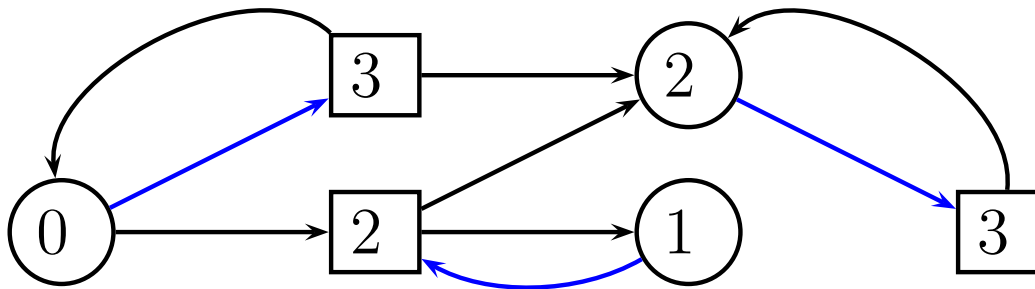
$\vec{v} \in Acc$ iff $\min(\text{Inf}_\Omega(\vec{v}))$ is even.

$\text{Inf}_\Omega(v_0v_1 \dots) = \{n : n \text{ appears infinitely often in } \Omega(v_0)\Omega(v_1) \dots\}$

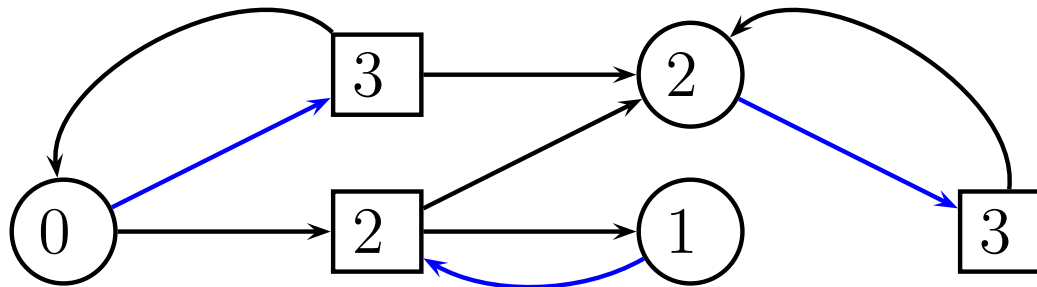
$$\mathcal{G} = \langle V_0, V_1, T, \Omega : (V_0 \cup V_1) \rightarrow \{0, \dots, d\} \rangle$$

Def: **Strategy** for player 0 is $\sigma : V^* \times V_0 \rightarrow V$.

Def: **Memoryless strategy** for player 0 is a function $\sigma : V_0 \rightarrow V$ such that $(v, \sigma(v)) \in T$.



Def: A strategy σ for player 0 is **winning from** v if all plays from v respecting the strategy are winning for player 0.



Thm: In a parity game, from every vertex one of the players has a memoryless winning strategy.

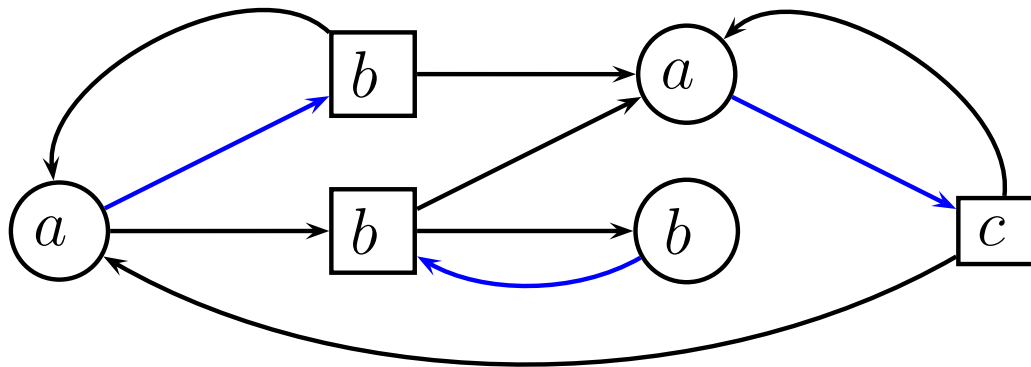
● To **solve a game** means to decide in each vertex who has a winning strategy.

Thm: Solving a parity game can be done in time $n^{d/2+1}$ where n is the number of vertices and $\{0, 1, \dots, d\}$ is the winning condition. This is an $\text{NP} \cap \text{co-NP}$ problem.

Digression: Muller conditions

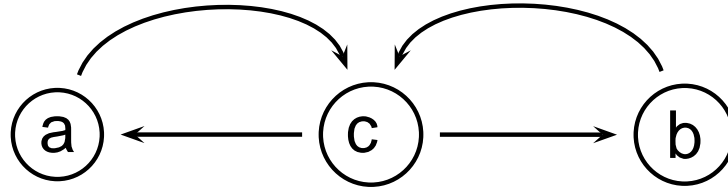
- Let *Colours* be a finite set of colours.
- Muller condition** is given by a set $\mathcal{F} \subseteq \mathcal{P}(\text{Colours})$.

$$\vec{v} \in \text{Acc}_{\mathcal{F}} \text{ iff } \text{Inf}_{\lambda}(\vec{v}) \in \mathcal{F}.$$



- Example: $\mathcal{F} = \{\{a, b\}, \{a, c\}\}$.

Winning with Muller conditions



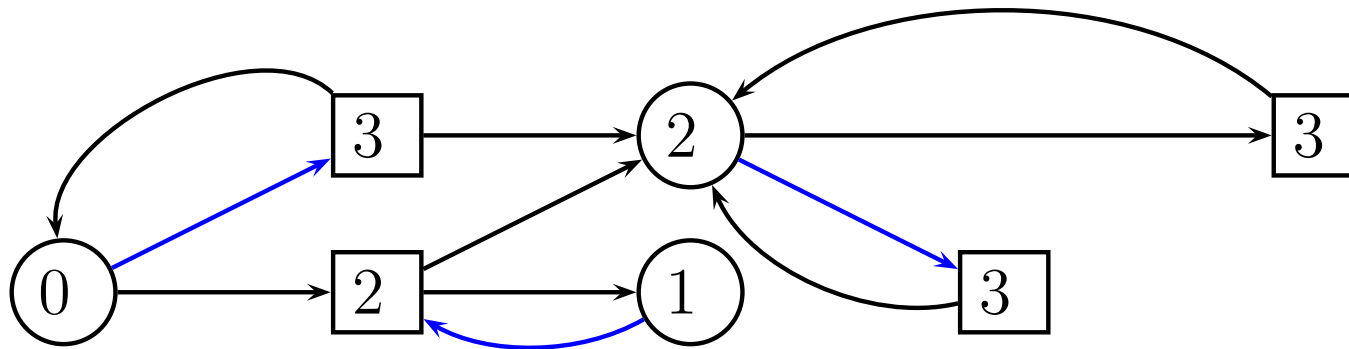
- Muller condition: $\mathcal{F} = \{\{a, b, c\}\}$.
(Both a and b appear infinitely often.)

- Player 0 has a winning strategy in this game but no memoryless winning strategy.

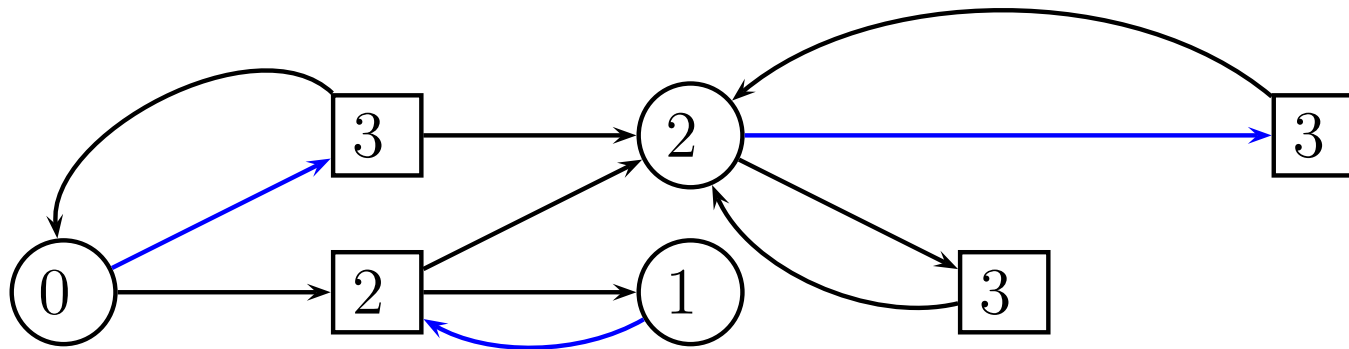
- Parity condition is:

- The only Muller condition that guarantees existence of memoryless strategies.
- Closed by negation (the negation of a parity condition is a parity condition).
- Universal, in the sense that every game with a Muller condition can be reduced to a game with a parity condition.

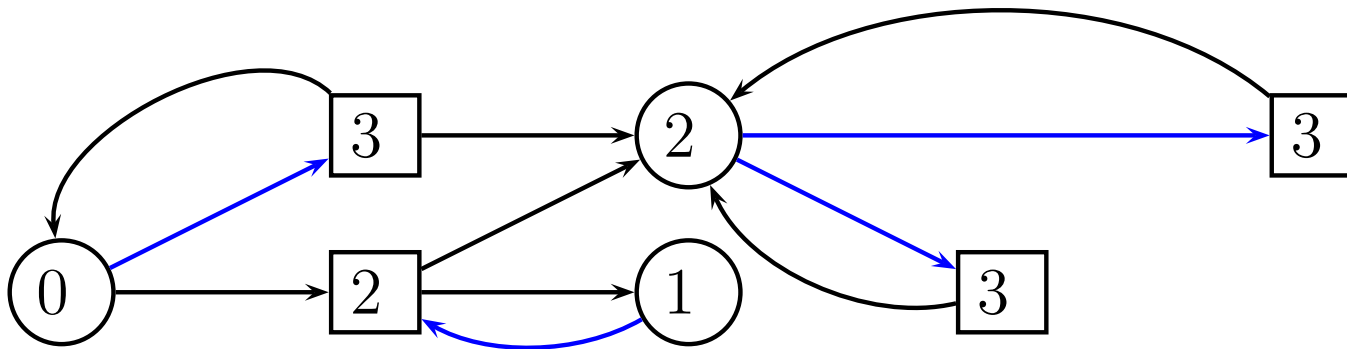
- Games can be considered as specifications and strategies as programs.
- In this view game solving is synthesis of a program for a given specification.



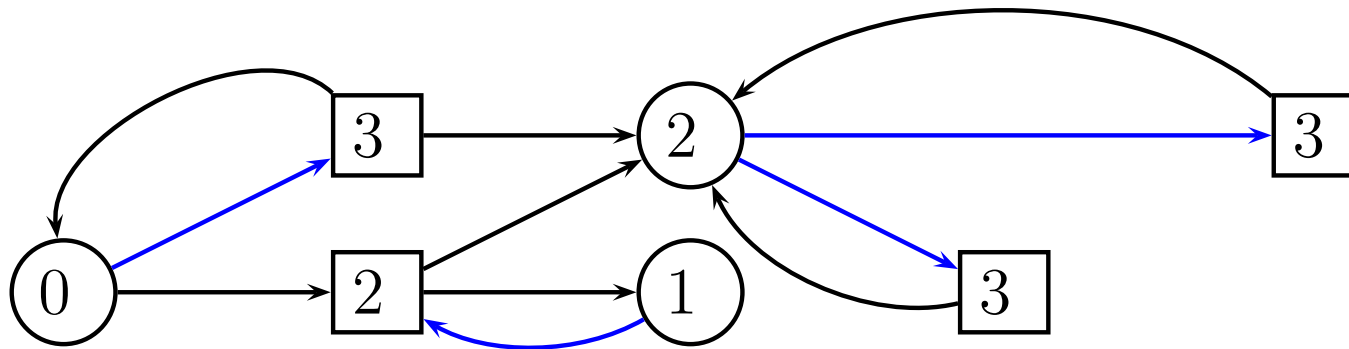
- Games can be considered as specifications and strategies as programs.
- In this view game solving is synthesis of a program for a given specification.



- Games can be considered as specifications and strategies as programs.
- In this view game solving is synthesis of a program for a given specification.



- Games can be considered as specifications and strategies as programs.
- In this view game solving is synthesis of a program for a given specification.



- In this kind of setting we can vary only the shape of a graph and the rest of a specification is fixed.

All centralized control problems are reducible to this one

Part Ib

Increasing the power of specifications

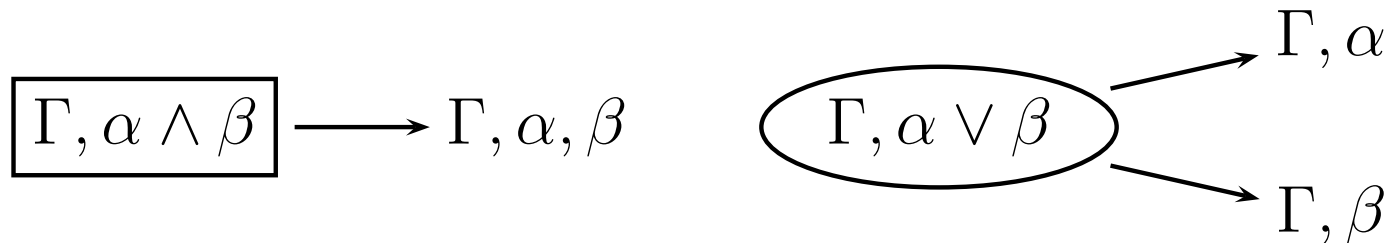
Given a specification α , find a system satisfying α .

- **Specification:** a propositional formula α ; with negations only before constants.
- **System:** Valuation of variables.
- **Solution:** For α construct a valuation satisfying α .
- This setting is very simple but already can code some funny problems:

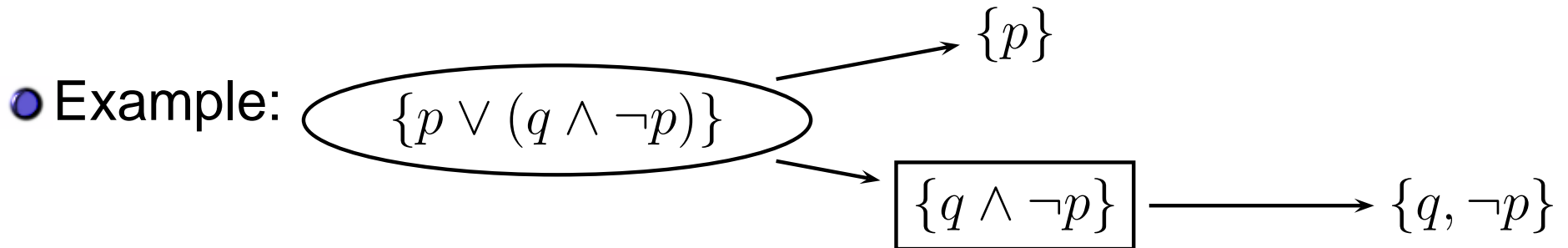
“A peasant wants to transport a cabbage, a goat and a wolf to the other side of the river. He can only transport one thing at a time in his boat. If he leaves, for example, cabbage and goat then goat will eat the cabbage. Design the transport schedule.”

- Given α we want to find a model for α .

- Rules of the game:



- Player 0 wins when there is nothing to reduce and there is no literal and its negation in the set.



- Every satisfying valuation is obtained as from some strategy.

- Labelled graph (aka **transition system**) is a structure:

$$\mathcal{M} = \langle V, T \subseteq V \times V, (P_a^{\mathcal{M}})_{(a \in \Sigma)} \rangle$$

- **Modal logic:**

$$P_a \mid \neg\alpha \mid \alpha \vee \beta \mid \langle \cdot \rangle \alpha$$

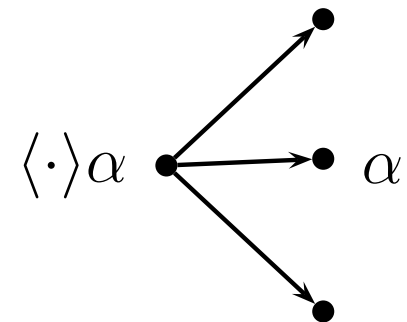
- **Semantics** $\llbracket \alpha \rrbracket^{\mathcal{M}}$:

$$\llbracket P_a \rrbracket^{\mathcal{M}} = P_a^{\mathcal{M}}$$

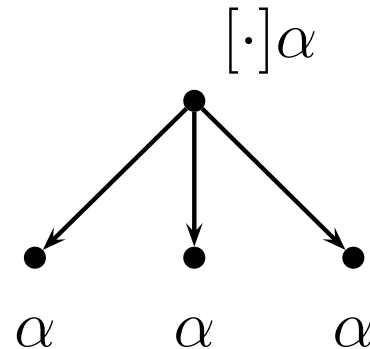
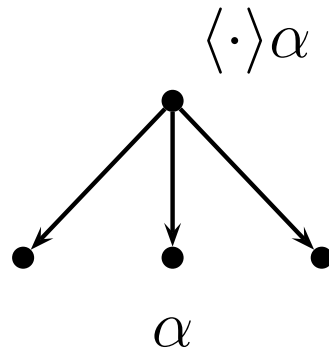
$$\llbracket \neg\alpha \rrbracket^{\mathcal{M}} = V \setminus \llbracket \alpha \rrbracket^{\mathcal{M}}$$

$$\llbracket \alpha \vee \beta \rrbracket^{\mathcal{M}} = \llbracket \alpha \rrbracket^{\mathcal{M}} \cup \llbracket \beta \rrbracket^{\mathcal{M}}$$

$$\llbracket \langle \cdot \rangle \alpha \rrbracket^{\mathcal{M}} = \{v : \exists v'. T(v, v') \wedge v' \in \llbracket \alpha \rrbracket^{\mathcal{M}}\}$$



● $[\cdot]\alpha \equiv \neg\langle\cdot\rangle\neg\alpha$



● Each modal formula looks only finitely many steps from the origin.

+ $v \in \llbracket \alpha \rrbracket$ depends only on a neighbourhood of v of some distance $\leq |\alpha|$.

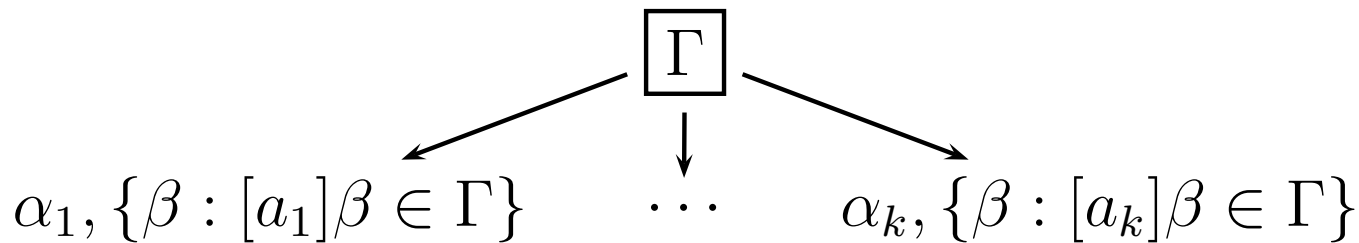
● Still the logic is more difficult than the propositional logic.

+ Satisfiability problem is PSPACE-complete (vs. NP-complete)

● Modal logic is not very useful for verification because transitive closure is not expressible.

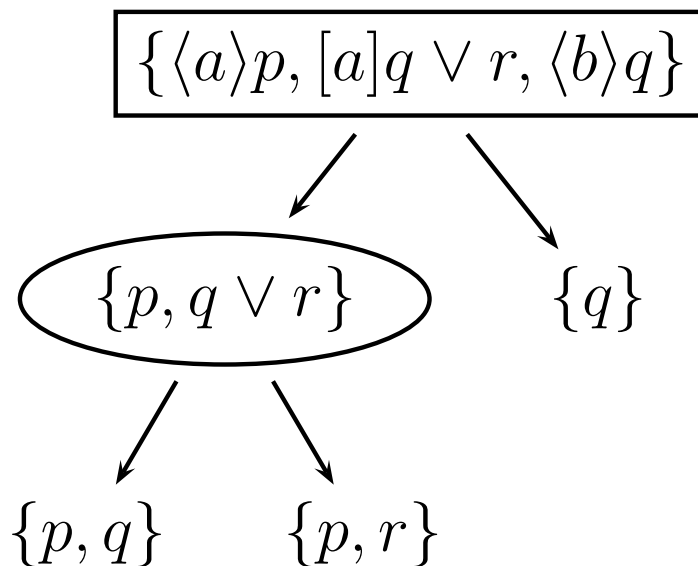
$tt \mid X \mid \neg\alpha \mid \alpha \wedge \beta \mid \langle a \rangle\alpha \mid [a]\alpha$

● Modal rule:



for all $\langle a_i \rangle \alpha_i \in \Gamma$.

● Example:



Given a specification α , find a system satisfying α .

- **System**: deterministic transition system over a set A of actions:

$$P = \langle S, A, s_I, e : S \times A \rightarrow S \rangle$$

- **Specification**: a mu-calculus formula:

$$tt \mid X \mid \neg\alpha \mid \alpha \wedge \beta \mid \langle a \rangle\alpha \mid \mu X.\alpha(X)$$

- **Solution**: For α construct an equivalent nondeterministic automaton \mathcal{A}_α and then find a system accepted by \mathcal{A}_α .

$tt \mid X \mid \neg\alpha \mid \alpha \wedge \beta \mid \langle a \rangle\alpha \mid \mu X.\alpha(X)$

- Over deterministic trees: μ -calculus \equiv MSOL.
- Over transition systems: μ -calculus \equiv MSOL/bisim.
- Decidable in EXPTIME.
- Quite direct translations between μ -calculus and (alternating) automata.
- **In summary:** expressive and manageable.

- We can add the constructor $\langle \cdot \rangle^* \alpha$:

$$\llbracket \langle \cdot \rangle^* \alpha \rrbracket^{\mathcal{M}} = \{v : \exists v'. T^*(v, v') \wedge v' \in \llbracket \alpha \rrbracket^{\mathcal{M}}\}$$

- Why not to check some properties on the path? $\exists(P_a \cup \alpha)$:

$$\llbracket \exists(P_a \cup \alpha) \rrbracket^{\mathcal{M}} = \{v : \exists \vec{v}. Path(\vec{v}, v) \wedge last(\vec{v}) \in \llbracket \alpha \rrbracket^{\mathcal{M}} \wedge \forall v_i \in \vec{v}. P_a(v_i)\}$$

- Why not the dual? $\forall(P_a \cup \alpha)$

- Why not alternating reachability?

- We introduce a set Var of propositional variables.

- Syntax: $P_a \mid X \mid \neg\alpha \mid \alpha \vee \beta \mid \langle \cdot \rangle \alpha \mid \mu X. \alpha$

+ We require that X appears in α only under even number of negations.

- Semantics; we need $Val : Var \rightarrow \mathcal{P}(V)$

$$\llbracket P_a \rrbracket_{Val}^{\mathcal{M}} = P_a$$

$$\llbracket X \rrbracket_{Val}^{\mathcal{M}} = Val(X)$$

$$\llbracket \langle \cdot \rangle \alpha \rrbracket_{Val}^{\mathcal{M}} = \{v : \exists v'. E(v, v') \wedge v' \in \llbracket \alpha \rrbracket_{Val}^{\mathcal{M}}\}$$

The meaning of the fix-point

- In $\mu X.\alpha(X)$ we require X to appear only positively in $\alpha(X)$.

The meaning of the fix-point

- In $\mu X.\alpha(X)$ we require X to appear only positively in $\alpha(X)$.
- The formula $\alpha(X)$ defines a function:

$$\lambda S. \llbracket \alpha(S) \rrbracket_{Val}^{\mathcal{M}} : \mathcal{P}(V) \rightarrow \mathcal{P}(V)$$

The meaning of the fix-point

- In $\mu X.\alpha(X)$ we require X to appear only positively in $\alpha(X)$.
- The formula $\alpha(X)$ defines a function:

$$\lambda S. \llbracket \alpha(S) \rrbracket_{Val}^{\mathcal{M}} : \mathcal{P}(V) \rightarrow \mathcal{P}(V)$$

- The function is monotonic:

$$\llbracket \alpha(S_1) \rrbracket_{Val}^{\mathcal{M}} \subseteq \llbracket \alpha(S_2) \rrbracket_{Val}^{\mathcal{M}} \quad \text{if } S_1 \subseteq S_2.$$

The meaning of the fix-point

- In $\mu X.\alpha(X)$ we require X to appear only positively in $\alpha(X)$.
- The formula $\alpha(X)$ defines a function:

$$\lambda S. \llbracket \alpha(S) \rrbracket_{Val}^{\mathcal{M}} : \mathcal{P}(V) \rightarrow \mathcal{P}(V)$$

- The function is monotonic:

$$\llbracket \alpha(S_1) \rrbracket_{Val}^{\mathcal{M}} \subseteq \llbracket \alpha(S_2) \rrbracket_{Val}^{\mathcal{M}} \quad \text{if } S_1 \subseteq S_2.$$

- There is the least fix-point of this function:

$$\llbracket \mu X.\alpha(X) \rrbracket_{Val}^{\mathcal{M}} = \bigcap \{ S \subseteq V : \llbracket \alpha(S) \rrbracket_{Val}^{\mathcal{M}} \subseteq S \}$$

- The greatest fix-point: $\nu X.\alpha(X) \equiv \neg\mu X.\neg\alpha(\neg X)$

$$\neg \bigcap \{S \subseteq \{0, 1\}^* : \llbracket \neg\alpha(\bar{S}) \rrbracket_V^{\mathcal{M}} \subseteq S\}$$

$$\neg \bigcap \{S \subseteq \{0, 1\}^* : \bar{S} \subseteq \llbracket \alpha(\bar{S}) \rrbracket_V^{\mathcal{M}}\}$$

$$\bigcup \{\bar{S} \subseteq \{0, 1\}^* : \bar{S} \subseteq \llbracket \alpha(\bar{S}) \rrbracket_V^{\mathcal{M}}\}$$

Def: A formula is in a **positive normal form** if negations are only before variables and propositional constants.

Fact: Every formula is equivalent to the one in the normal form.

$$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta \quad \neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$$

$$\neg\langle \cdot \rangle \alpha \equiv [\cdot] \neg\alpha \quad \neg[\cdot] \alpha \equiv \langle \cdot \rangle \neg\alpha$$

$$\neg\mu X.\alpha(X) \equiv \nu X.\neg\alpha(\neg X)$$

$$\neg\nu X.\alpha(X) \equiv \mu X.\neg\alpha(\neg X)$$

- Reachability: $\langle \cdot \rangle^* \alpha \equiv \mu X. \alpha \vee \langle \cdot \rangle X.$
- Existential until: $\exists(P_a \mathbb{U} \alpha) \equiv \mu X. \alpha \vee (P_a \wedge \langle \cdot \rangle X)$
- Universal until: $\forall(P_a \mathbb{U} \alpha) \equiv \mu X. \alpha \vee (P_a \wedge [\cdot] X \wedge \langle \cdot \rangle tt).$
- Alternating reachability

$$\mu X. \alpha \vee (P_{or} \wedge \langle \cdot \rangle X) \vee (P_{and} \wedge [\cdot] X)$$

$$\mu X. \beta(X) = \bigcup_{\tau \in Ord} \mu^\tau X. \beta(X)$$

$$\llbracket \mu^0 X. \beta(X) \rrbracket_V^{\mathcal{M}} = \emptyset$$

$$\llbracket \mu^{\tau+1} X. \beta(X) \rrbracket_V^{\mathcal{M}} = \llbracket \beta(X) \rrbracket_V^{\mathcal{M}} \llbracket \llbracket \mu^\tau X. \beta(X) \rrbracket_V^{\mathcal{M}} / X \rrbracket$$

$$\llbracket \mu^\tau X. \beta(X) \rrbracket_V^{\mathcal{M}} = \bigcup_{\tau' < \tau} \llbracket \mu^{\tau'} X. \beta(X) \rrbracket_V^{\mathcal{M}} \quad \text{if } \tau \text{ is a limit ordinal}$$

- $\mu X. P_b \vee \langle \cdot \rangle X$ holds in v whenever there is a reachable vertex labelled by b .
- $\nu X. P_b \wedge \langle \cdot \rangle X$ holds in v if there is an infinite path every node of which is labelled by b .

$$\nu X. \beta(X) = \bigcap_{\tau \in \text{Ord}} \nu^\tau X. \beta(X)$$

$$\llbracket \nu^0 X. \beta(X) \rrbracket_V^{\mathcal{M}} = S$$

$$\llbracket \nu^{\tau+1} X. \beta(X) \rrbracket = \llbracket \beta(X) \rrbracket_V^{\mathcal{M}} \llbracket \llbracket \nu^\tau X. \beta(X) \rrbracket_V^{\mathcal{M}} / X \rrbracket$$

$$\llbracket \nu^\tau X. \beta(X) \rrbracket_V^{\mathcal{M}} = \bigcap_{\tau' < \tau} \llbracket \nu^{\tau'} X. \beta(X) \rrbracket_V^{\mathcal{M}} \quad \text{if } \tau \text{ is a limit ordinal}$$

Examples (alternating fixpoints)

- Almost always P_a on some path

$$\mu Y. \nu X. (P_a \wedge \langle \cdot \rangle X) \vee \langle \cdot \rangle Y$$

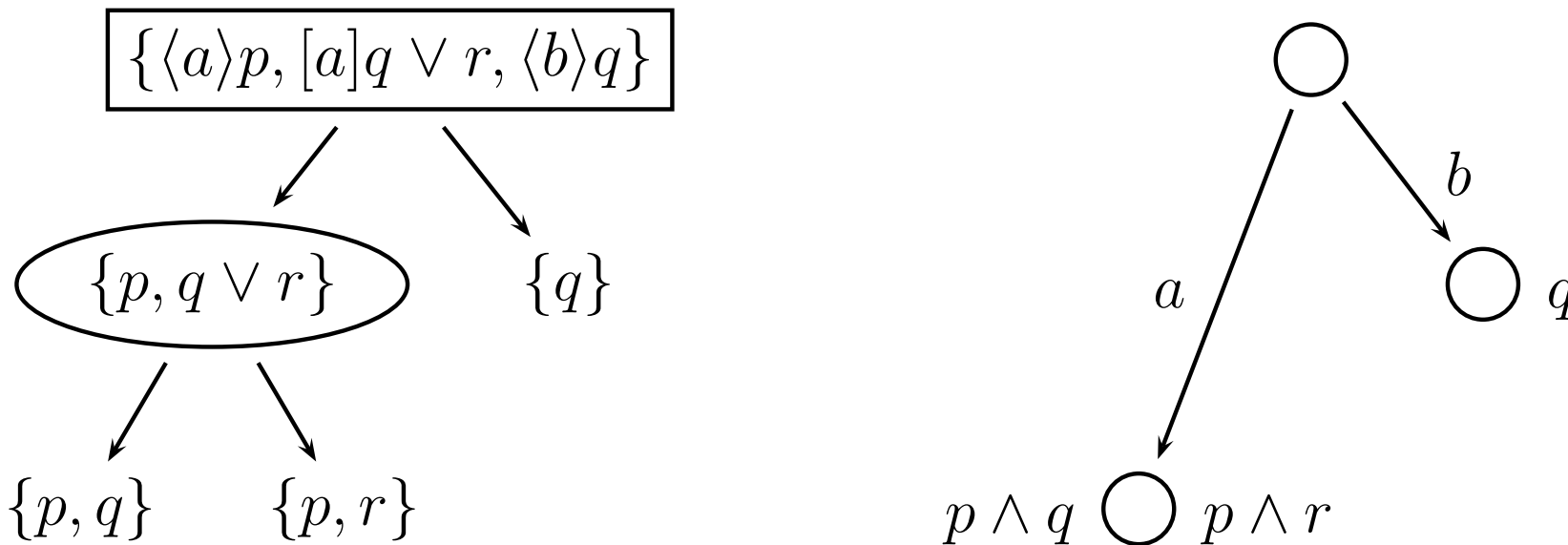
- Infinitely often P_a on some path

$$\nu X. \mu Y. (P_a \wedge \langle \cdot \rangle X) \vee \langle \cdot \rangle Y$$

Why branching time specifications?

- The μ -calculus talks about the properties of the “view” from the given node, i.e. unwinding of the the transition system from a given node. This unwinding is a **tree**.
- Linear specification is the one that talks only about all the possible paths from the node. Here we consider only **set of paths**
- In the later case one cannot express possibility:
From every state it is possible to reach a reset state.

- For modal logic we have constructed a game where winning strategies corresponded to models:



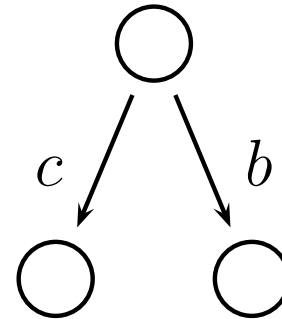
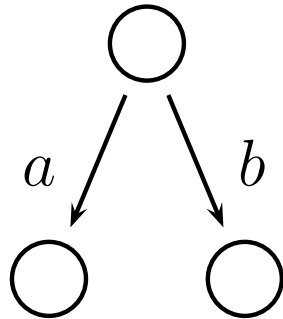
- This game defines a very simple automaton accepting all the models of the formula.

$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$
$$\text{Moves}(A, Q) = A \rightarrow (Q \cup \{\rightarrow, \nrightarrow\})$$

Interpretation of a transition $f \in \text{Moves}(A, Q)$:

- $f(a) = \rightarrow$ means: there must be a successor.
- $f(a) = \nrightarrow$ means: there cannot be a successor.
- $f(a) = q'$ means: the automaton should go to a successor and change the state to q' .

- Language consisting of the following two trees.



- Automaton $\mathcal{A} = \langle Q = \{q_0, q_1\}, A = \{a, b, c\}, \delta, Acc \rangle$

$$\delta(q_0) = \{f_a, f_c\}, \quad \delta(q_1) = g$$

$$f_a(a) = q_1$$

$$f_a(b) = q_1$$

$$f_a(c) = \rightarrow$$

$$f_a(a) = \rightarrow$$

$$f_a(b) = q_1$$

$$f_a(c) = q_1$$

$$g(a) = \rightarrow$$

$$g(b) = \rightarrow$$

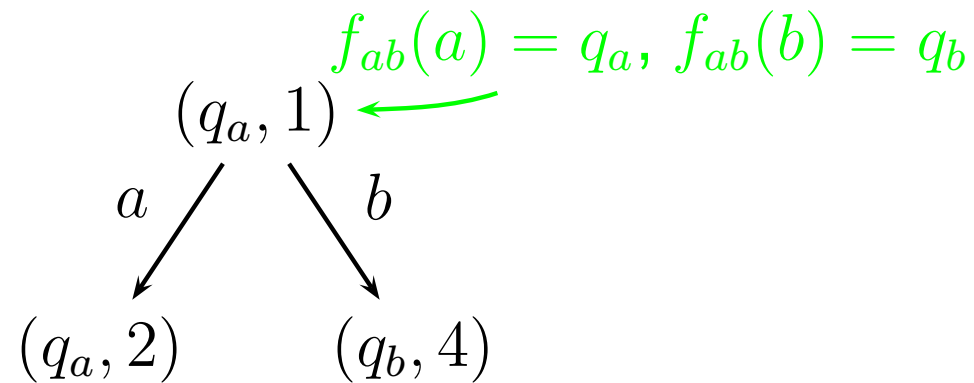
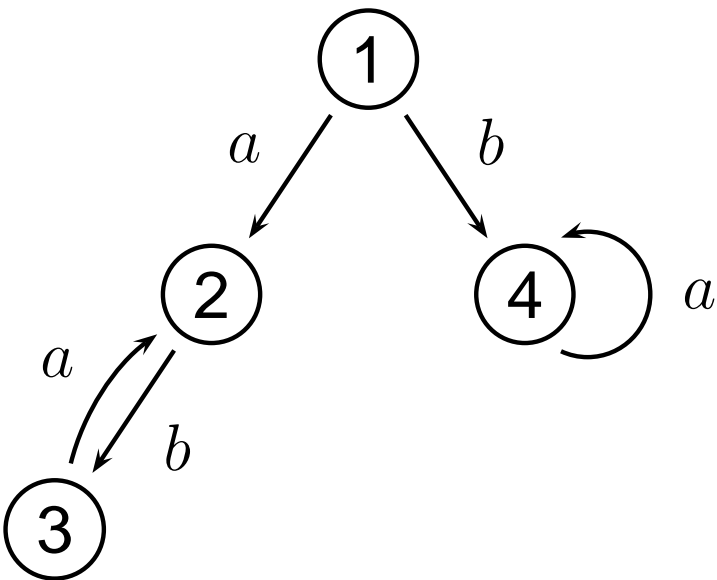
$$g(c) = \rightarrow$$

- $L = \text{"inf many } b \text{ on every infinite path"}$

$$\mathcal{A} = \langle \{q_a, q_b\}, \{a, b\}, q_a, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\delta(q_a) = \delta(q_b) = \{f : f(a) \in \{q_a, \dashrightarrow\}, f(b) \in \{q_b, \dashrightarrow\}\}$$

$$\text{Acc} = \{q_0 q_1 \dots : \exists^\infty i. q_i = q_b\}$$

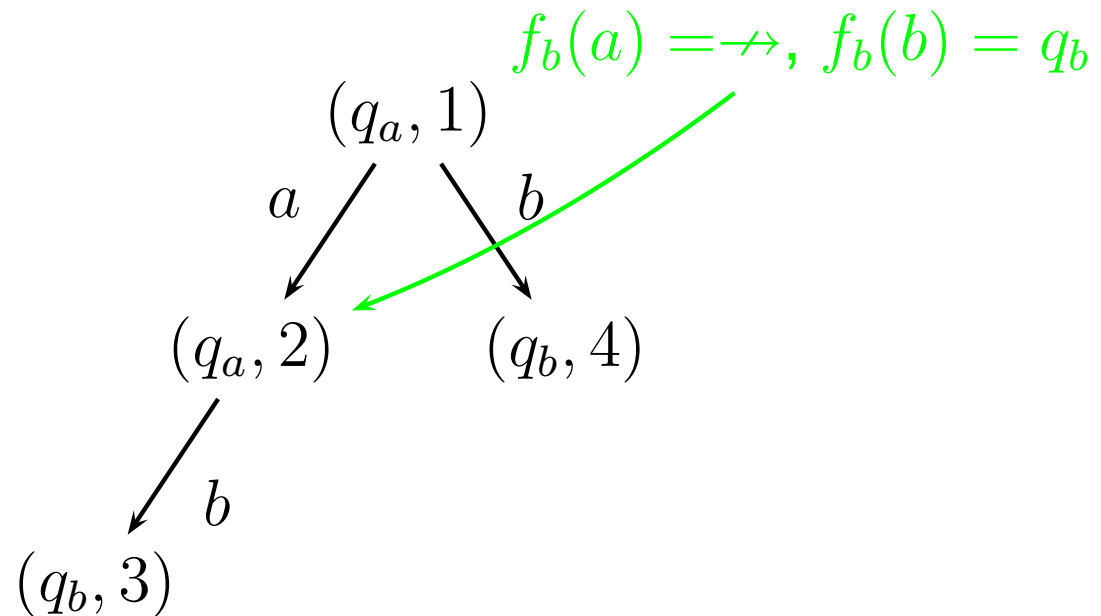
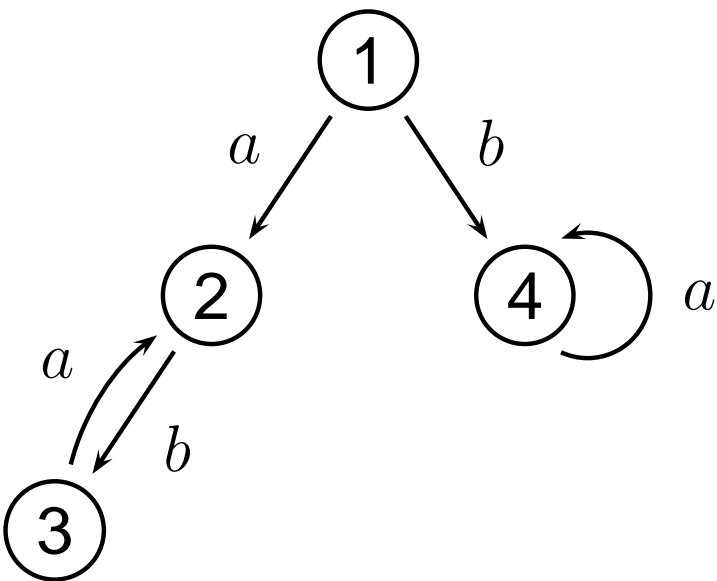


- $L = \text{"inf many } b \text{ on every infinite path"}$

$$\mathcal{A} = \langle \{q_a, q_b\}, \{a, b\}, q_a, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\delta(q_a) = \delta(q_b) = \{f : f(a) \in \{q_a, \dashrightarrow\}, f(b) \in \{q_b, \dashrightarrow\}\}$$

$$\text{Acc} = \{q_0 q_1 \dots : \exists^\infty i. q_i = q_b\}$$

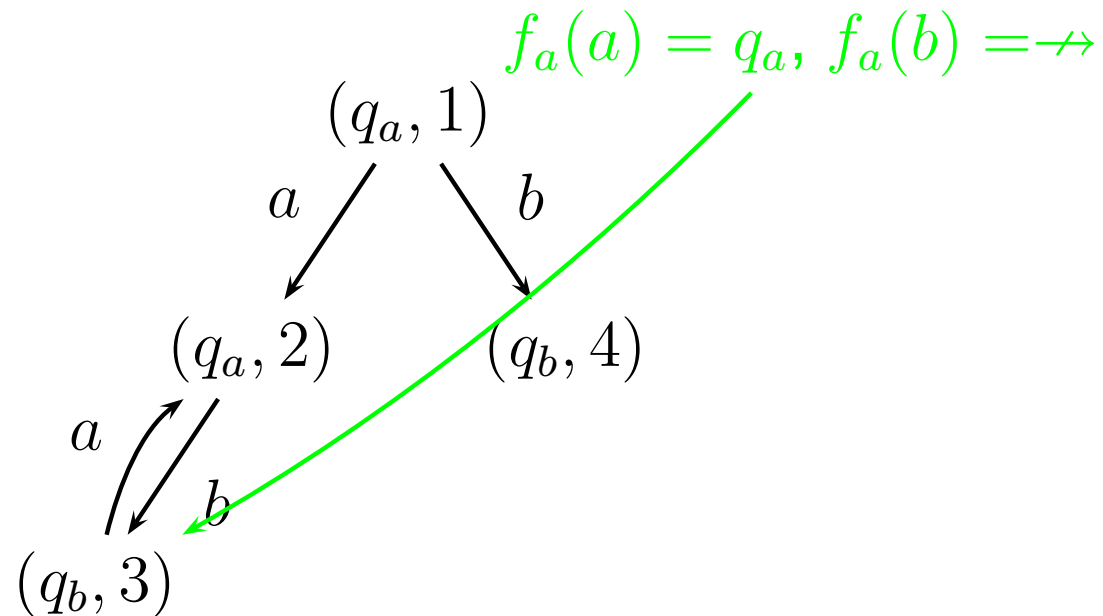
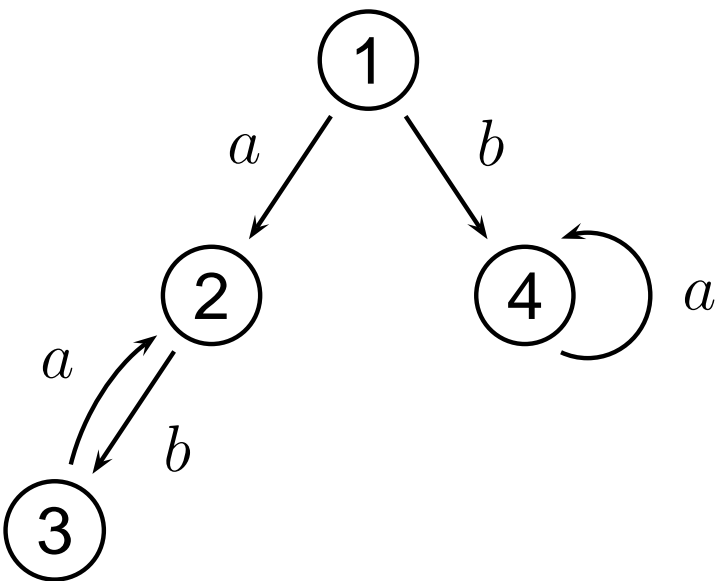


- $L = \text{"inf many } b \text{ on every infinite path"}$

$$\mathcal{A} = \langle \{q_a, q_b\}, \{a, b\}, q_a, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\delta(q_a) = \delta(q_b) = \{f : f(a) \in \{q_a, \dashrightarrow\}, f(b) \in \{q_b, \dashrightarrow\}\}$$

$$\text{Acc} = \{q_0 q_1 \dots : \exists^\infty i. q_i = q_b\}$$

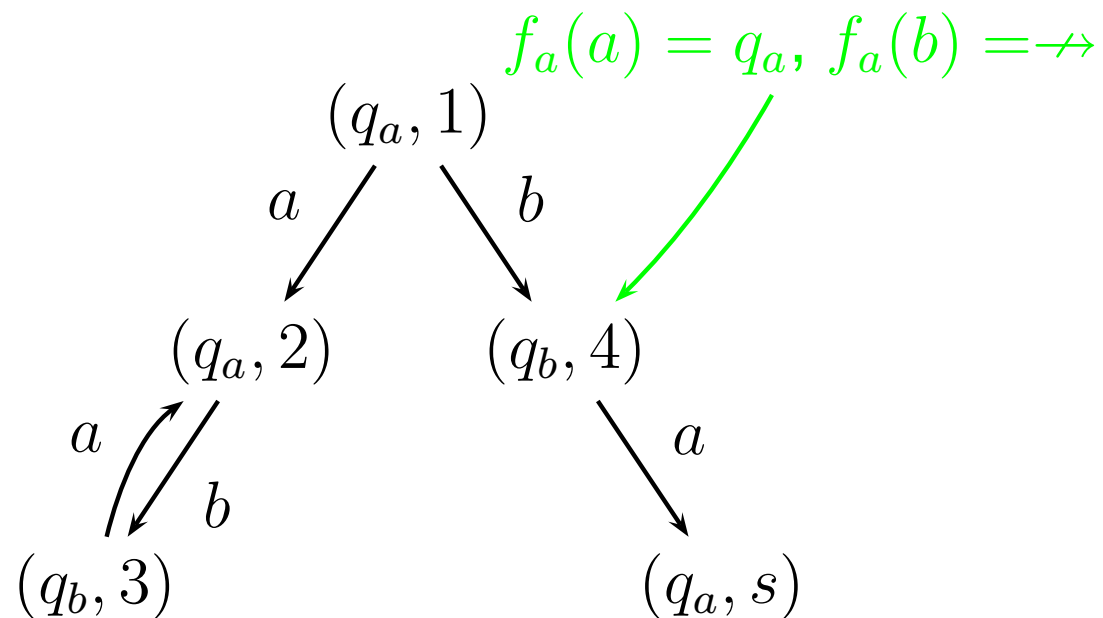
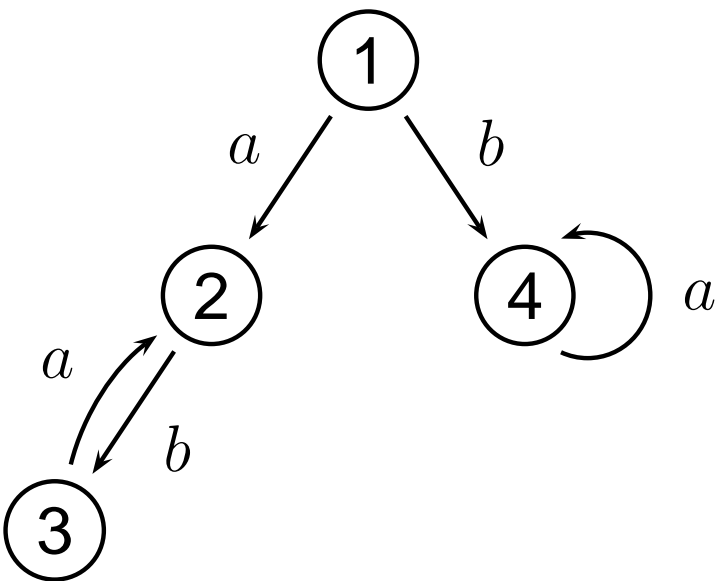


- $L = \text{"inf many } b \text{ on every infinite path"}$

$$\mathcal{A} = \langle \{q_a, q_b\}, \{a, b\}, q_a, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\delta(q_a) = \delta(q_b) = \{f : f(a) \in \{q_a, \dashrightarrow\}, f(b) \in \{q_b, \dashrightarrow\}\}$$

$$\text{Acc} = \{q_0 q_1 \dots : \exists^\infty i. q_i = q_b\}$$

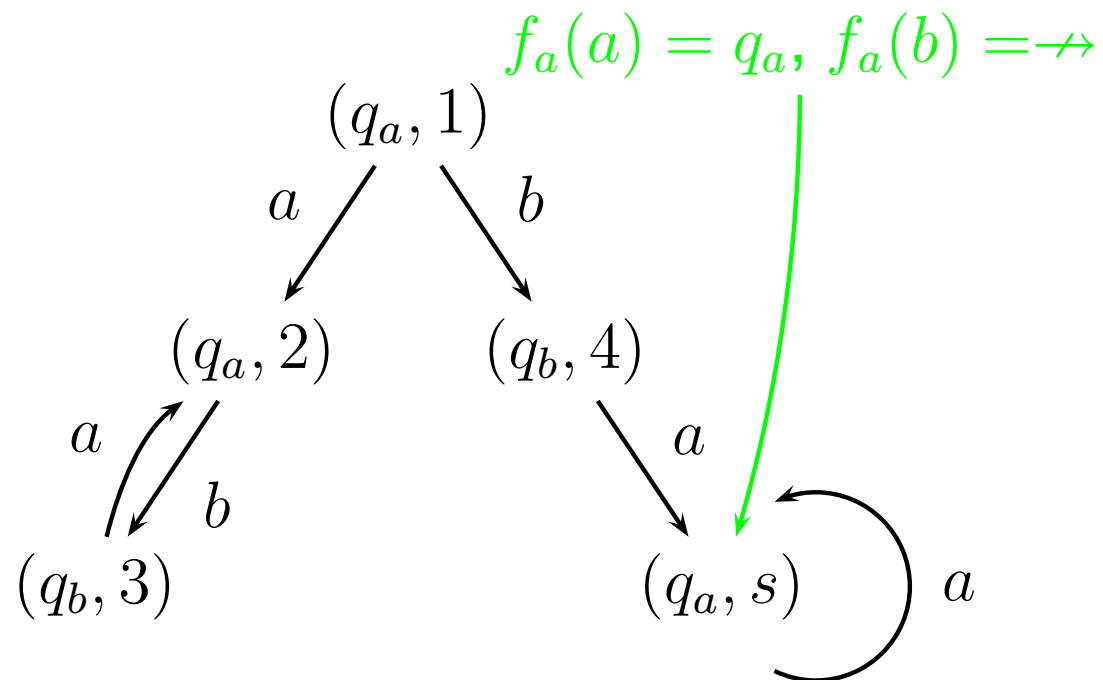
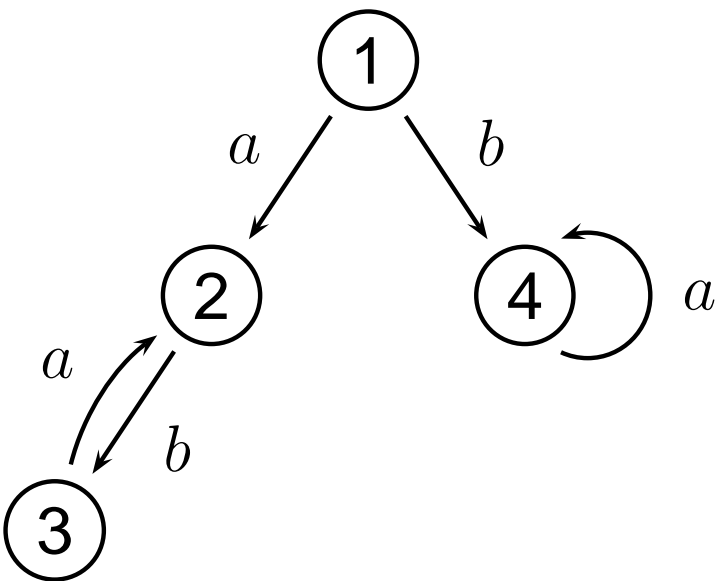


- $L = \text{"inf many } b \text{ on every infinite path"}$

$$\mathcal{A} = \langle \{q_a, q_b\}, \{a, b\}, q_a, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\delta(q_a) = \delta(q_b) = \{f : f(a) \in \{q_a, \rightarrow\}, f(b) \in \{q_b, \rightarrow\}\}$$

$$\text{Acc} = \{q_0 q_1 \dots : \exists^\infty i. q_i = q_b\}$$



$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\text{Moves}(A, Q) = A \rightarrow (Q \cup \{\rightarrow, \nrightarrow\})$$

● Game $G(\mathcal{A}, P)$:

- $V_0 = Q \times S \cup \{\perp\}$
- $V_1 = \text{Moves}(A, Q) \cup \{\top\}$
- $v_I = (q_I, s_I)$
- $(q, s) \rightarrow (f, s)$ for every $f \in \delta(q)$,
- $(f, s) \rightarrow (q_a, s_a)$ if $f(a) = q_a$ and $s \rightarrow_a s_a$,
- $(f, s) \rightarrow \top$ if $f(a) = \rightarrow$ and $s \rightarrow_a s_a$ exists,
- $(f, s) \rightarrow \perp$ if $f(a) = \nrightarrow, q_a$ and no $s \rightarrow_a s_a$.

$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\text{Moves}(A, Q) = A \rightarrow (Q \cup \{\rightarrow, \nrightarrow\})$$

● Game $G(\mathcal{A}, P)$:

- $V_0 = Q \times S \cup \{\perp\}$
- $V_1 = \text{Moves}(A, Q) \cup \{\top\}$
- $v_I = (q_I, s_I)$
- $(q, s) \rightarrow (f, s)$ for every $f \in \delta(q)$,
- $(f, s) \rightarrow (q_a, s_a)$ if $f(a) = q_a$ and $s \rightarrow_a s_a$,
- $(f, s) \rightarrow \top$ [\perp] if $f(a) \Rightarrow \rightarrow$ [\nrightarrow] and $s \rightarrow_a s_a$ exists,
- $(f, s) \rightarrow \perp$ [\top] if $f(a) \Rightarrow \rightarrow$, q_a [\nrightarrow] and no $s \rightarrow_a s_a$.

$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\text{Moves}(A, Q) = A \rightarrow (Q \cup \{\rightarrow, \nrightarrow\})$$

● Game $G(\mathcal{A}, P)$:

- $V_0 = Q \times S \cup \{\perp\}$
 - $V_1 = \text{Moves}(A, Q) \cup \{\top\}$
 - $v_I = (q_I, s_I)$
 - $(q, s) \rightarrow (f, s)$ for every $f \in \delta(q)$,
 - $(f, s) \rightarrow (q_a, s_a)$ if $f(a) = q_a$ and $s \rightarrow_a s_a$,
 - $(f, s) \rightarrow \top$ [\perp] if $f(a) \Rightarrow \rightarrow$ [\nrightarrow] and $s \rightarrow_a s_a$ exists,
 - $(f, s) \rightarrow \perp$ [\top] if $f(a) \Rightarrow \rightarrow, q_a$ [\nrightarrow] and no $s \rightarrow_a s_a$.
- P0 wins if the sequence of states is in Acc or P1 is stuck.

$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\text{Moves}(A, Q) = A \rightarrow (Q \cup \{\rightarrow, \nrightarrow\})$$

● Game $G(\mathcal{A}, P)$:

- $V_0 = Q \times S \cup \{\perp\}$
 - $V_1 = \text{Moves}(A, Q) \cup \{\top\}$
 - $v_I = (q_I, s_I)$
 - $(q, s) \rightarrow (f, s)$ for every $f \in \delta(q)$,
 - $(f, s) \rightarrow (q_a, s_a)$ if $f(a) = q_a$ and $s \rightarrow_a s_a$,
 - $(f, s) \rightarrow \top$ [\perp] if $f(a) \Rightarrow \rightarrow$ [\nrightarrow] and $s \rightarrow_a s_a$ exists,
 - $(f, s) \rightarrow \perp$ [\top] if $f(a) \Rightarrow \rightarrow$, q_a [\nrightarrow] and no $s \rightarrow_a s_a$.
- P0 wins if the sequence of states is in Acc or P1 is stuck.
- $P \in L(\mathcal{A})$ if player 0 has a winning strategy in $G(\mathcal{A}, P)$.

$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

● Game $G(\mathcal{A}, P)$:

- $V_0 = Q \times S \cup \{\perp\}$
- $V_1 = \text{Moves}(A, Q) \cup \{\top\}$
- $v_I = (q_I, s_I)$
- $(q, s) \rightarrow (f, s)$ for every $f \in \delta(q)$,
- $(f, s) \rightarrow (q_a, s_a)$ if $f(a) = q_a$ and $s \rightarrow_a s_a$,
- $(f, s) \rightarrow \top$ if $f(a) = \rightarrow$ and $s \rightarrow_a s_a$ exists,
- $(f, s) \rightarrow \perp$ if $f(a) = \rightarrow$ and no $s \rightarrow_a s_a$.
- P0 wins if the sequence of states is in Acc or P1 is stuck.

$$\mathcal{A} = \langle Q, A, q_I, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

● Game $G(\mathcal{A})$:

● $V_0 = Q$

● $V_1 = \text{Moves}(A, Q)$

● $v_I = q_I$

● $q \rightarrow f$ for every $f \in \delta(q)$,

● $f \rightarrow q_a$ if $f(a) = q_a$

● P0 wins if the sequence of states is in Acc or P1 is stuck.

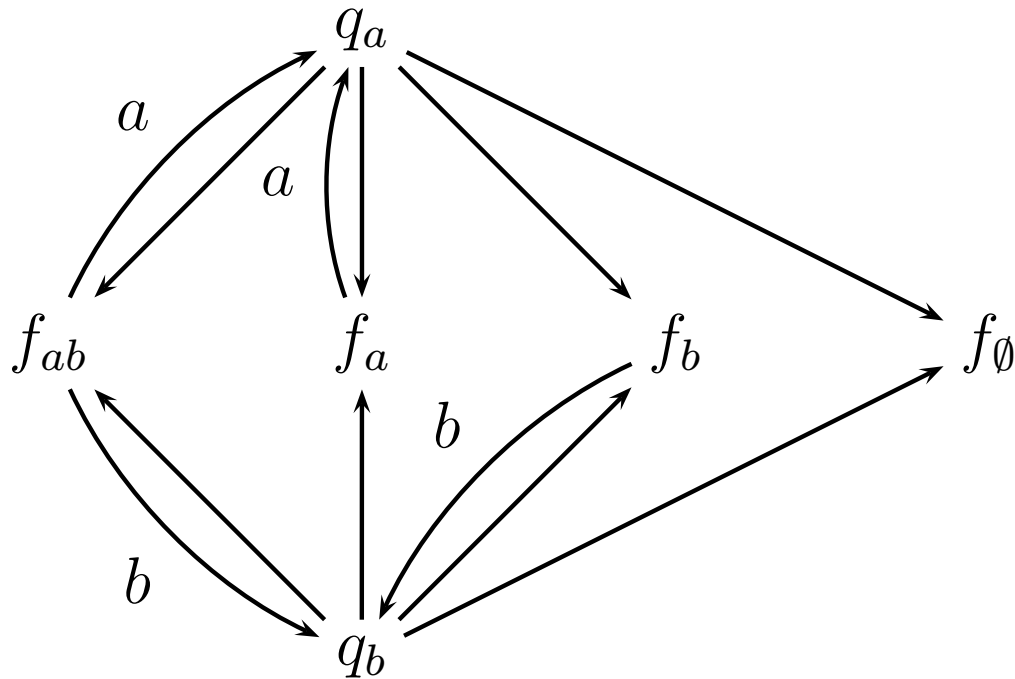
Fact: $L(\mathcal{A}) \neq \emptyset$ iff player 0 has a winning strategy in $G(\mathcal{A})$. A winning strategy defines a system P s.t. $P \in L(\mathcal{A})$.

- Infinitely many b 's on every infinite path.

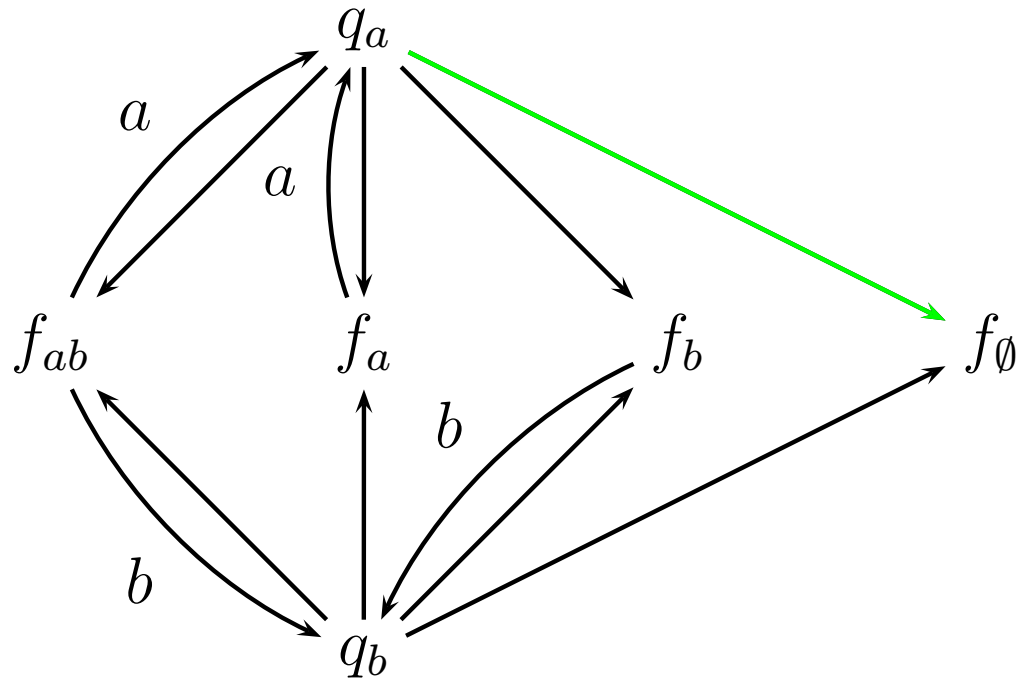
$$\mathcal{A} = \langle \{q_a, q_b\}, \{a, b\}, q_a, \delta : Q \rightarrow \mathcal{P}(\text{Moves}(A, Q)), \text{Acc} \subseteq Q^\omega \rangle$$

$$\begin{aligned} \delta(q_a) = \delta(q_b) &= \{f : f(a) \in \{q_a, \dashrightarrow\}, f(b) \in \{q_b, \dashrightarrow\}\} \\ \text{Acc} &= \{q_0q_1 \dots : \exists^\infty i. q_i = q_b\} \end{aligned}$$

- Infinitely many b 's on every infinite path.

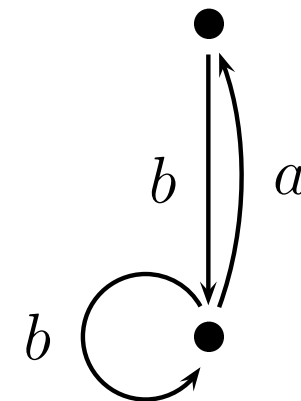
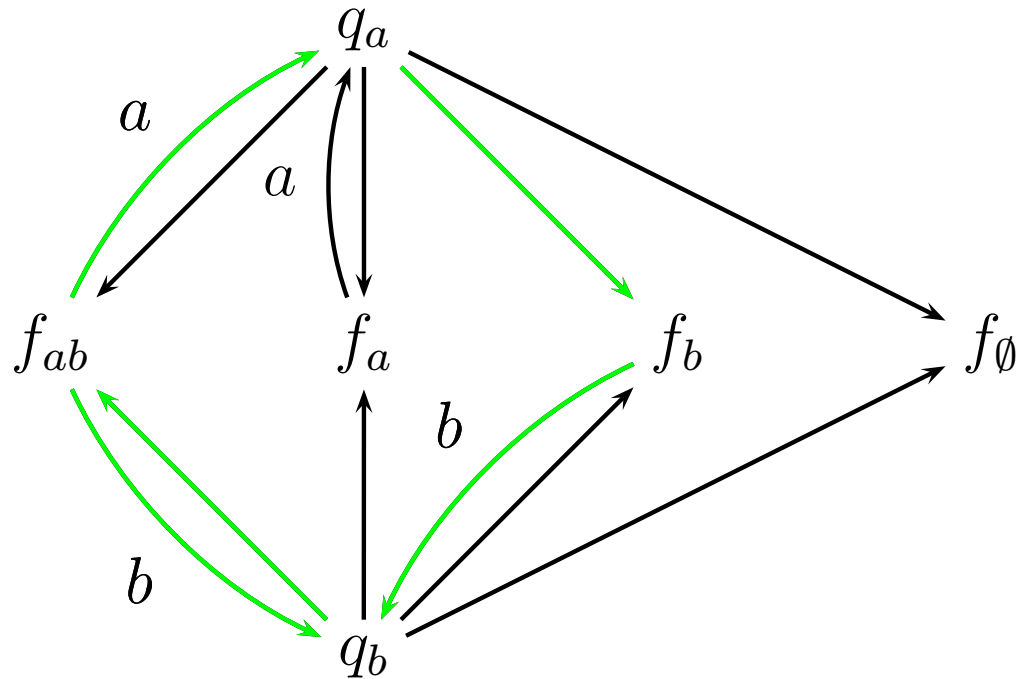


- Infinitely many b 's on every infinite path.



- $f_a(a) = q_a$ $f_a(b) = q_b$.

- Infinitely many b 's on every infinite path.



- $f_a(a) = q_a$ $f_a(b) = \rightarrow$.

Thm: For every μ -calculus formula α there is an equivalent automaton \mathcal{A}_α with a parity acceptance condition s.t.:

$$P \models \alpha \quad \text{iff} \quad P \in L(\mathcal{A}_\alpha)$$

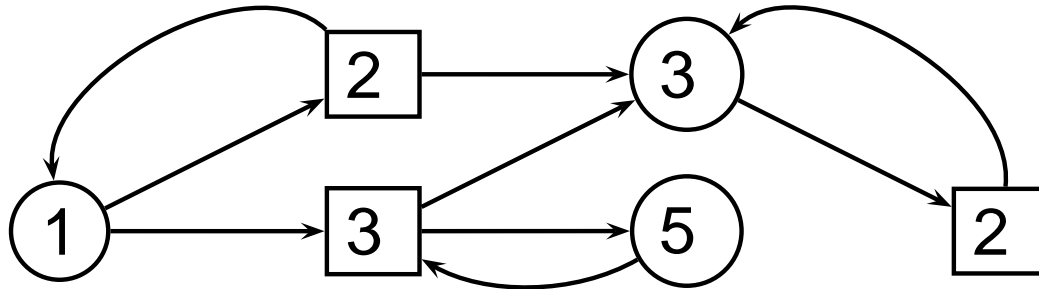
- Solving the synthesis problem:
- given α construct \mathcal{A}_α ;
- check if player 0 has a winning strategy in $G(\mathcal{A}_\alpha)$;
- a winning strategy defines a system $P \models \alpha$.

Every system $P \models \alpha$ comes from a winning player 0 strategy in $G(\mathcal{A}_\alpha)$.

Part Ic

Complicating the problem further

- Specification via games:



- Specification via formulas:

Given a formula α find a model for α .

Synthesis problem for a given plant

- A **plant** is a deterministic transition system over Σ .

$$P = \langle A, S^p, s_I^p, e^p : S \times A \rightarrow S \rangle$$

Given P and α , find a controller C (deterministic transition system) s.t. $P \times C \models \alpha$.

$$P \times C = \langle S = S^p \times S^c, A, (s_I^p, s_I^c), e : S \times A \rightarrow S \rangle$$
$$e((s_p, s_c), a) = (e^p(s_p, a), e^c(s_c, a))$$

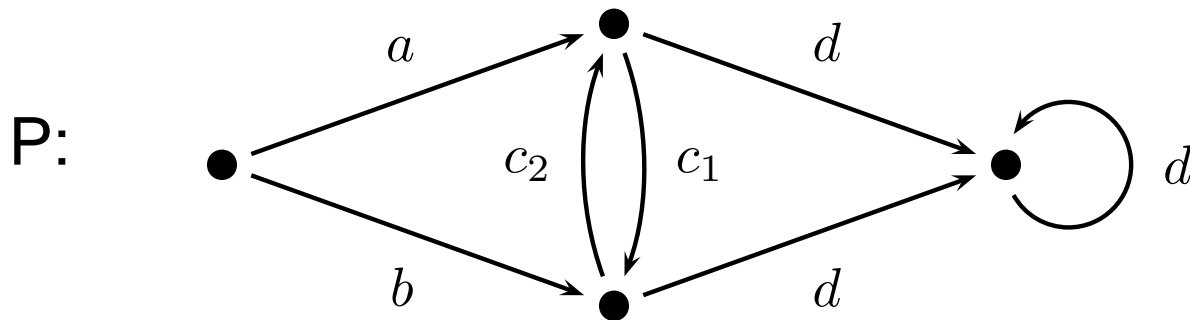
- **Solution:**

- Define an operation \mathcal{A}/P such that:

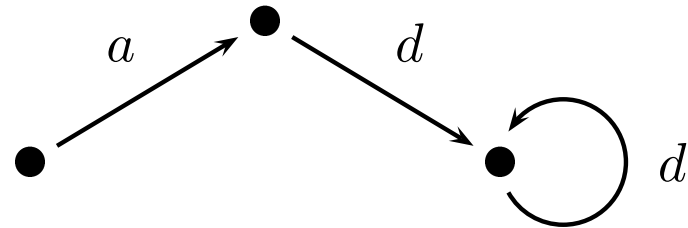
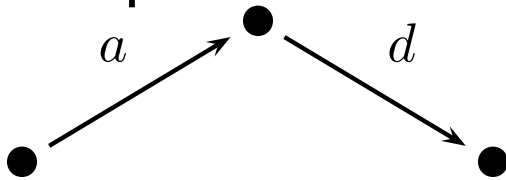
$$C \in L(\mathcal{A}/P) \quad \text{iff} \quad P \times C \in L(\mathcal{A})$$

- Find a model $C \in L(\mathcal{A}_\alpha/P)$.

$\alpha \equiv \text{execute } d \text{ action}$



- Some possible controllers:



- The second solution is non-blocking.

$$C \in L(\mathcal{A}_\alpha/P) \cap L(\mathcal{A}_{nonblock})$$

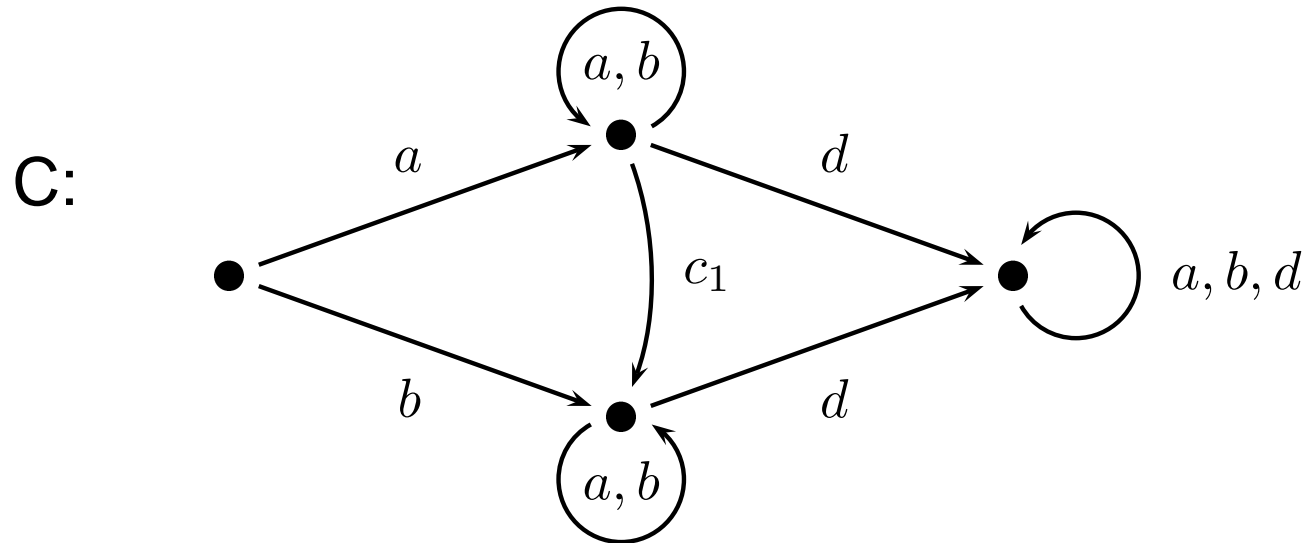
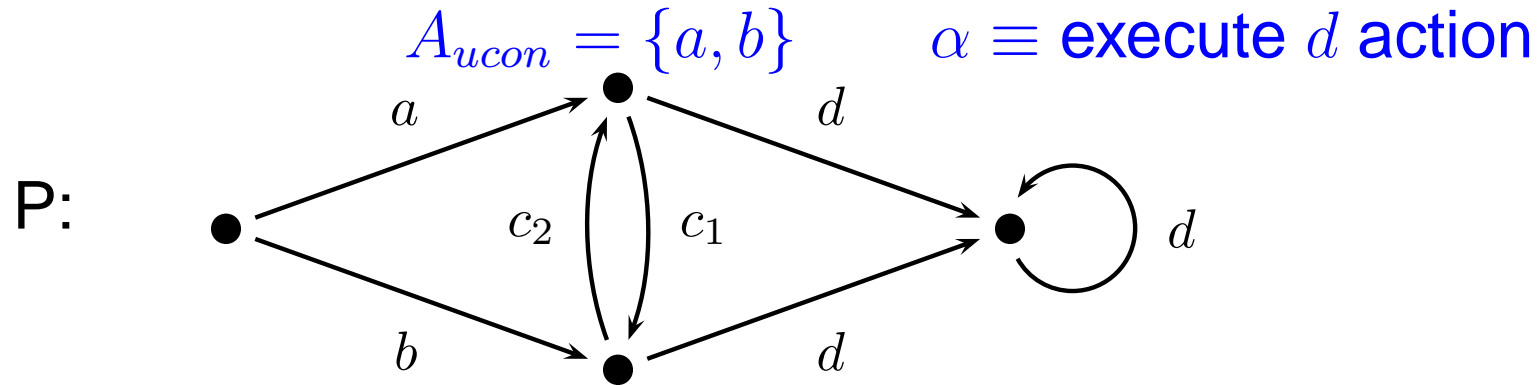
- So we can require additional properties from the controller.

- Divide A into A_{con} and A_{ucon} of **controllable** and **uncontrollable** actions.

- Additional condition:

$$\begin{aligned}\theta_{ucon} : & \quad C \text{ cannot forbid actions from } A_{ucon} \\ & \equiv \forall s \in S. \quad \forall a \in A_{ucon}. \quad e(s, a) \text{ defined} \\ & \equiv \nu X. \quad \left(\bigwedge_{a \in A} [a]X \right) \quad \wedge \quad \left(\bigwedge_{a \in A_{ucon}} \langle a \rangle tt \right)\end{aligned}$$

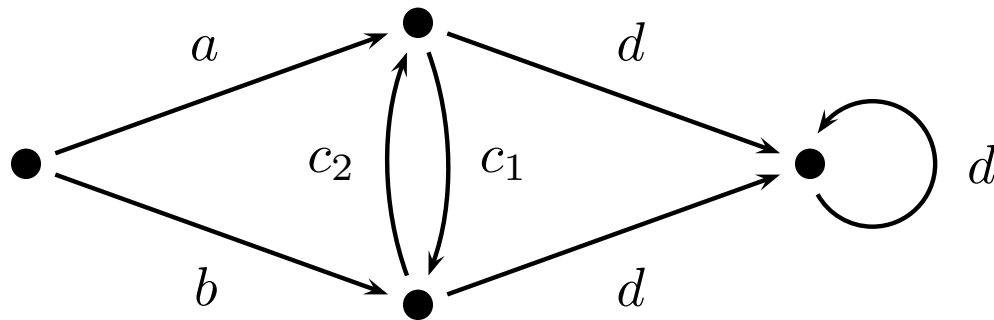
- **Solution:** $C \in L(\mathcal{A}_\alpha/P) \cap L(\mathcal{A}_{\theta_{ucon}})$



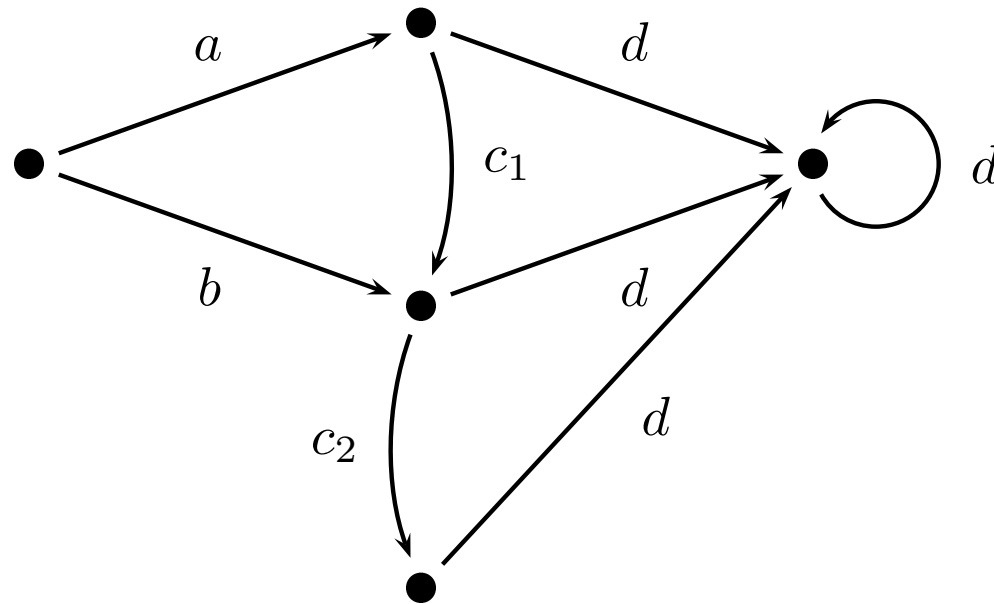
- There are infinitely many controllers satisfying the specification.

$A_{ucon} = \{a, b\}$ $\alpha \equiv \text{execute } d \text{ action}$

P:



C:



- Divide A into A_{obs} and A_{uobs} of **observable** and **unobservable** actions.

- New condition

θ_{uobs} : C cannot change state on actions in A_{uobs}

(C cannot observe actions from A_{uobs})

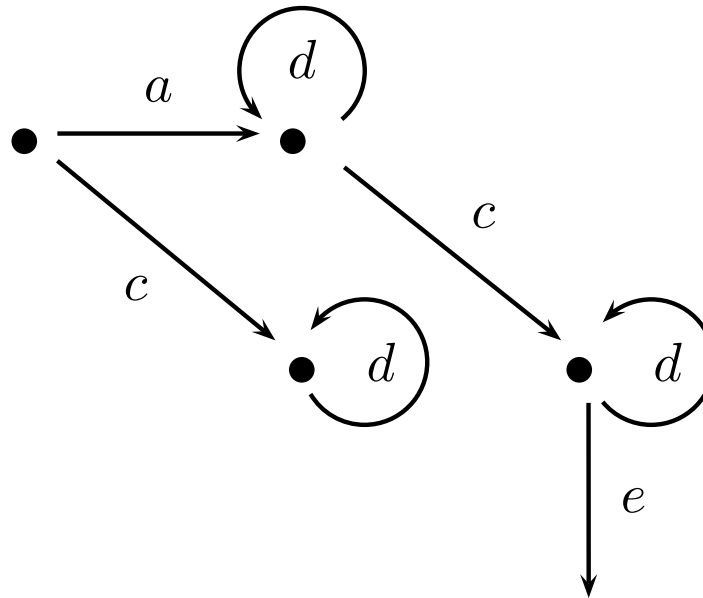
$$\equiv \forall s \in S. \quad \forall a \in A_{uobs}. \quad e(s, a) = s$$

$$\equiv \nu X. \quad \left(\bigwedge_{a \in A} [a]X \right) \quad \wedge \quad \left(\bigwedge_{a \in A_{uobs}} \bigcirc_a \right)$$

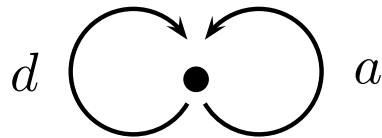
- A new operation \bigcirc_a is need in the μ -calculus.

- **Solution:** $C \in L(\mathcal{A}_\alpha/P) \cap L(\mathcal{A}_{\theta_{ucon}}) \cap L(\mathcal{A}_{\theta_{uobs}})$

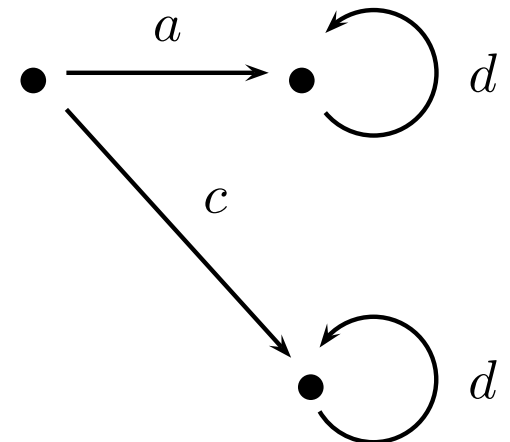
$A_{uobs} = \{a\}$, $A_{ucon} = \{e\}$ and the goal is to avoid e



● The solution:



Wrong:



Why new operation $\overset{\circ}{\circ}_a$ is needed?

- $\overset{\circ}{\circ}_a$ holds in v iff $e(s, a) = s$
- The property θ_{uobs} is not bisimulation invariant, hence not expressible in the μ -calculus or by automata.
- The μ -calculus and automata extended with $\overset{\circ}{\circ}_a$ still have the same good algorithmic properties:
 - $\mu(\overset{\circ}{\circ})$ -calculus is translatable to parity nondeterministic parity automata with $2^{\mathcal{O}(|\alpha| \log(|\alpha|))}$ blowup.
 - Emptiness of an automaton of size 2^n with a parity condition of size $\mathcal{O}(n)$ can be solved in $\mathcal{O}(2^n)$ time.

- The conditions θ_{ucon} and θ_{uobs} are expressible in the specification language. This allows to consider their refinements.

- After f appears action c becomes uncontrollable:

$$\nu x. \langle a \rangle x \wedge [c]x \wedge \langle f \rangle \nu y. (\langle a \rangle y \wedge \langle c \rangle y \wedge \langle f \rangle y).$$

- Only one of the actions c_1, c_2 is controllable at a time:

$$\nu x. \langle a \rangle x \wedge \left((\langle c_1 \rangle x \wedge [c_2]x) \vee ([c_1]x \wedge \langle c_2 \rangle x) \right)$$

- The general setting is always the same:

Given P and α, β :

find C such that $C \models \beta$ and $C \times P \models \alpha$.

- There is a number of possible variations/extensions of the synthesis problem.

- All these problems reduce to

Given P and α, β :

find C such that $C \models \beta$ and $C \times P \models \alpha$.

- Hence: C is a controller iff $C \in L(\mathcal{A}_\beta) \cap (\mathcal{A}_\alpha/P)$.

- Every C comes from a strategy in the emptiness checking game $G(\mathcal{A}_\beta \cap (\mathcal{A}_\alpha/P))$.

- The notion of observability is central and causes many problems.

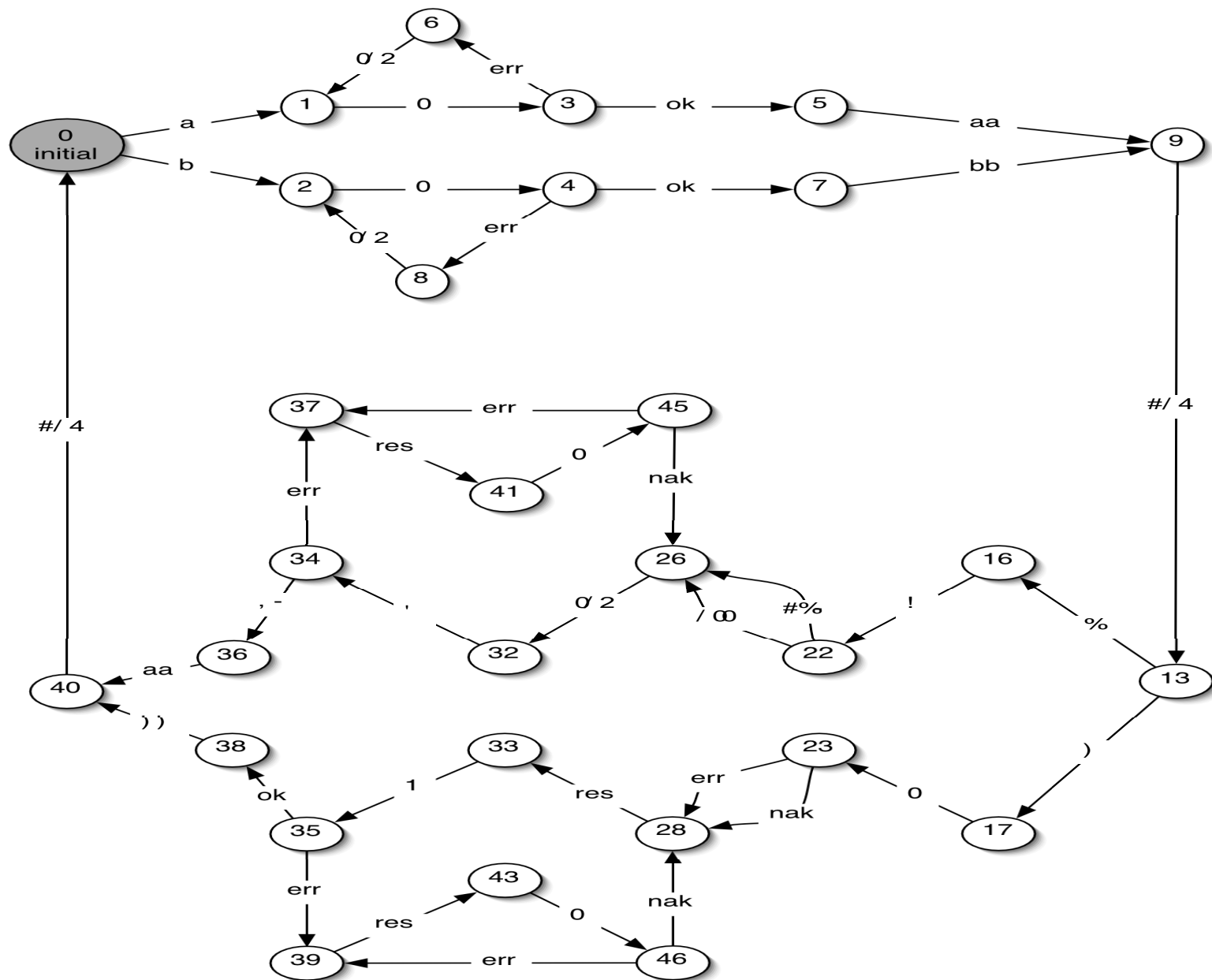
- One can ask for a controller with least number of states. It is possible to find one by exhaustive search.
- Another possibility is to compare controllers by the set of behaviours they allow.
- μ -specifications do not have maximal controllers, ex., “finitely many a ”.
- There exist behaviourally maximal controllers only for ν -specifications. Deterministic ν -specifications have the biggest controller.

Practice:

Practice:

Still long way to go.

Example: alternating bit protocol



name phi1; x = nu -> ERR.([a,b,aa,bb,0,1,new,res,ok,err,nak]x);
<initial=x>. **No error.**

name psi; x = nu -> <a,b,aa,bb,ok,err,nak>x. [0,1,new,res]x;
<initial=x>. **0,1 new, res are controllable.**

name phi2;

x= mu -> (<a>y+ y + <aa>x + <bb>x+ <0>x +<1>x+ <new>x+
<res>x+
<ok>x + <err>y+ <nak>x).

[a]y.[b]y. [aa]x. [bb]x. [0]x. [1]x. [new]x. [res]x. [ok]x. [err]y
.[nak]x

y= nu -> (<a>y+ y + <aa>x + <bb>x+ <0>x +<1>x+ <new>x+
<res>x+
<ok>x + <err>y+ <nak>x).

[a]y.[b]y. [aa]x. [bb]x. [0]x. [1]x. [new]x. [res]x. [ok]x. [err]y
.[nak]x

<initial=x> **Not blocking and one of a, b, err infinitely often**

Part IIa

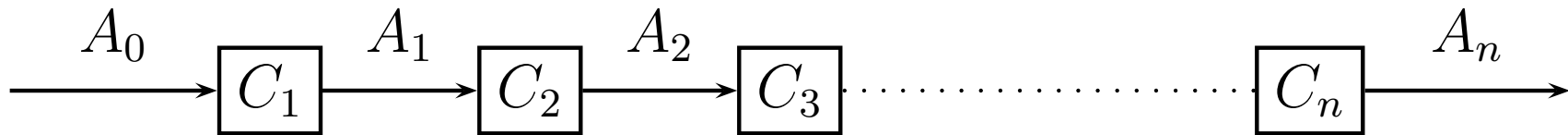
Distributed synthesis

Given a plant P and formulas α, β_1, β_2 do there exist controllers C_1, C_2 such that:

$$C_1 \models \beta_1, \quad C_2 \models \beta_2 \quad \text{and} \quad P \times C_1 \times C_2 \models \alpha.$$

(Each controller has its own A_{ucon}^i and A_{uobs}^i .)

Synthesis for given architectures:



Given a plant P and formulas α, β_1, β_2 do there exist controllers C_1, C_2 such that:

$$C_1 \models \beta_1, \quad C_2 \models \beta_2 \quad \text{and} \quad P \times C_1 \times C_2 \models \alpha.$$

(Each controller has its own A_{ucon}^i and A_{uobs}^i .)

- Define new operation \mathcal{A}/\mathcal{B} with the property:

$$P \in L(\mathcal{A}/\mathcal{B}) \text{ iff there is } C \text{ such that } C \in L(\mathcal{B}) \text{ and } P \times C \in L(\mathcal{A})$$

- The operation \mathcal{A}/\mathcal{B} works only if \mathcal{B} does not use \odot .

- We have:

$$\begin{aligned} P \in L((\mathcal{A}/\mathcal{B}_1)/\mathcal{B}_2) & \text{ iff there is } C_2 \text{ with } P \times C_2 \in L(\mathcal{A}/\mathcal{B}_1) \\ & \text{ iff there are } C_1, C_2 \text{ with } P \times C_1 \times C_2 \in L(\mathcal{A}). \end{aligned}$$

- If $P \in L((\mathcal{A}/\mathcal{B}_1)/\mathcal{B}_2)$ then controllers exists but where they are?

$P \in L(\mathcal{A}/\mathcal{B})$ iff there is C such that
 $C \in L(\mathcal{B})$ and $P \times C \in L(\mathcal{A})$.

(1) Find $C_2 \in L((\mathcal{A}/\mathcal{B}_1)/P) \cap L(\mathcal{B}_2)$

+ We have $C_2 \models \mathcal{B}_2$ and $P \times C_2 \models L(\mathcal{A}/\mathcal{B}_1)$.

(2) Find $C_1 \in L(\mathcal{A}/(P \times C_2)) \cap L(\mathcal{B}_1)$

+ We have $C_1 \models \mathcal{B}_1$ and $P \times C_1 \times C_2 \models \mathcal{A}$.

Conclusion: If everything is visible then we are done.

- Problem:

Given two languages $K, M \subseteq A^*$ and a plant P , find C_1, C_2 :

$$K \subseteq L(P \times C_1 \times C_2) \subseteq M$$

each C_i has its own A_{con}^i and A_{obs}^i .

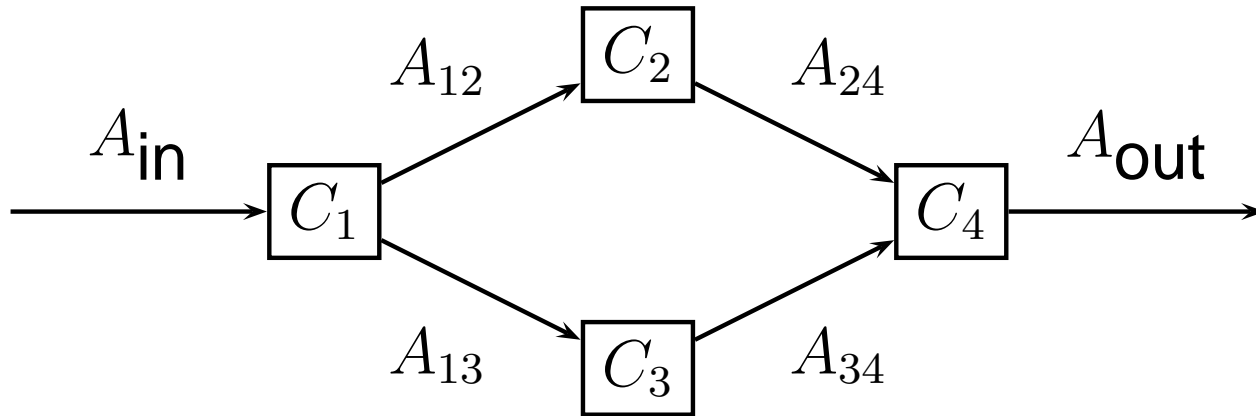
- Find the smallest C_1 such that $K \subseteq L(C_1)$.
Find the smallest C_2 such that $K \subseteq L(C_2)$.
- Check if $L(C_1) \cap L(C_2) = L(C_1 \times C_2) \subseteq M$.
If it fails then there is no solution.

If

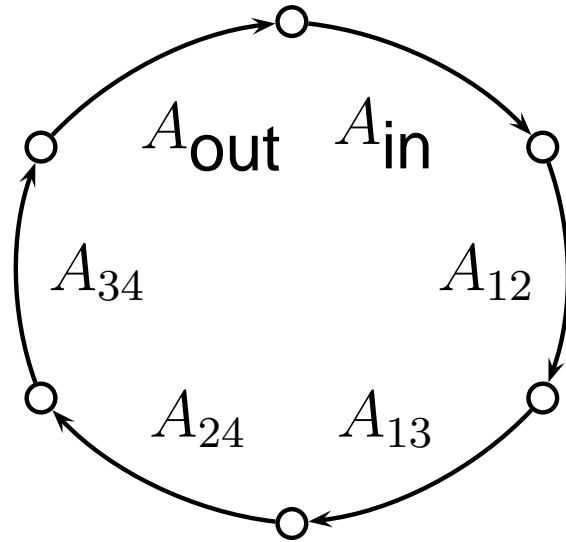
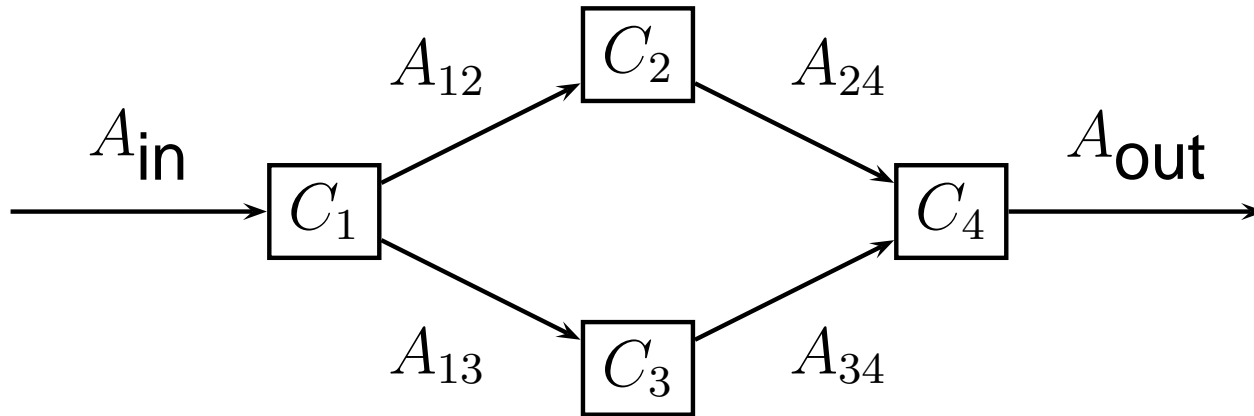
$$K \subseteq L(P \times C'_1 \times C'_2) \subseteq M$$

then

$$K \subseteq L(P \times C_1 \times C_2) \subseteq L(P \times C'_1 \times C'_2) \subseteq M$$

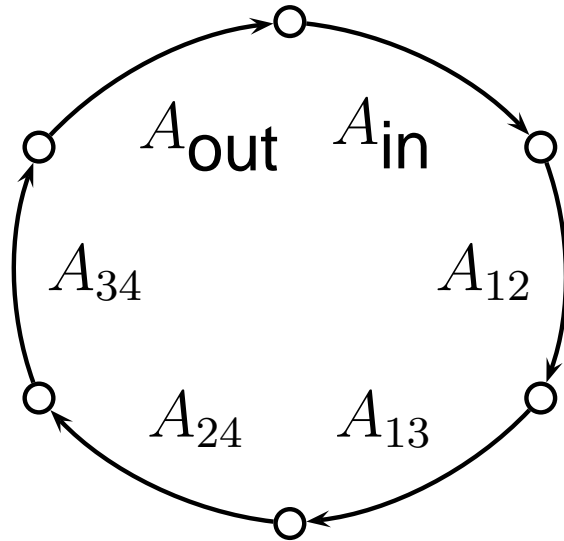
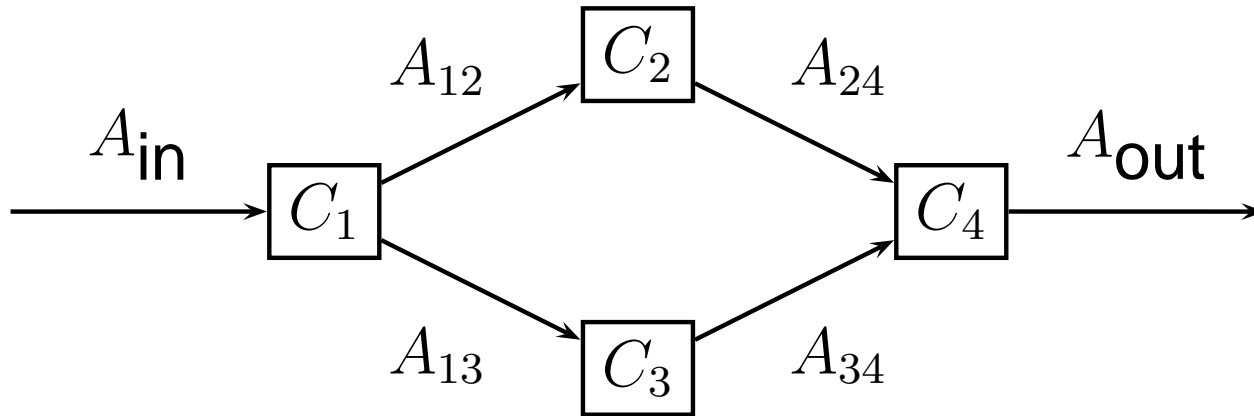


Pnueli-Rosner setting



$$\times \boxed{C_1} \times \dots \times \boxed{C_4}$$

Pnueli-Rosner setting

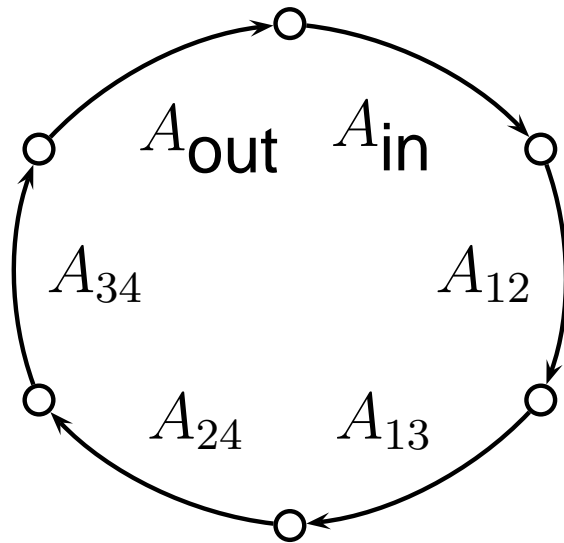
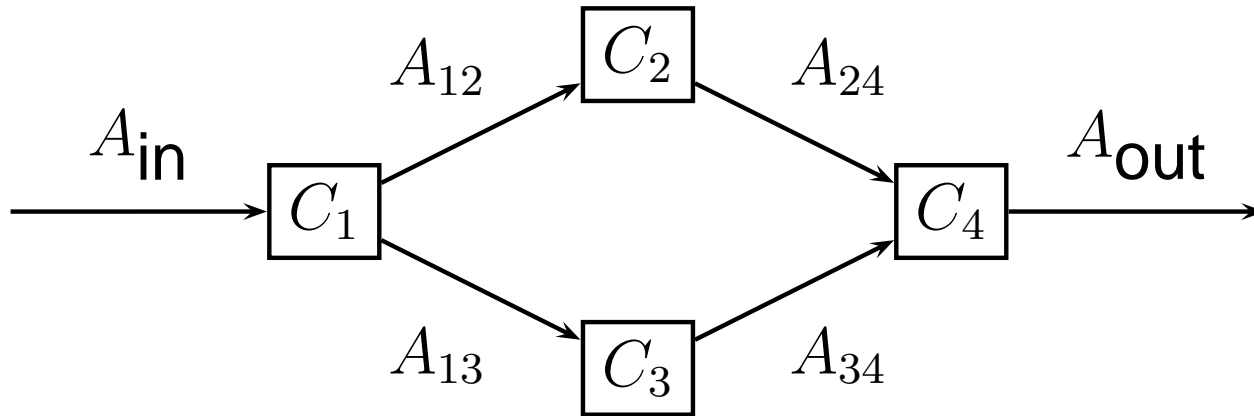


$$\times \boxed{C_1} \times \dots \times \boxed{C_4}$$

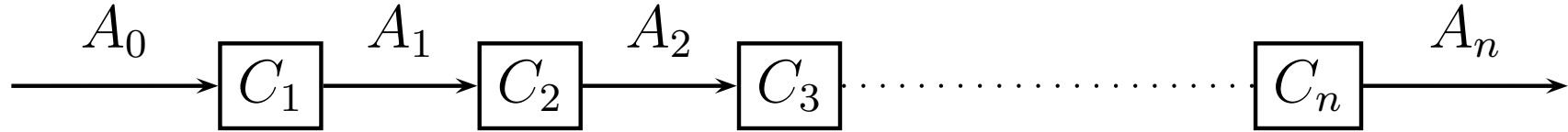
$$A_o^1 = A_{in} \cup A_{12} \cup A_{13}$$

$$A_o^4 = A_{out} \cup A_{24} \cup A_{34}$$

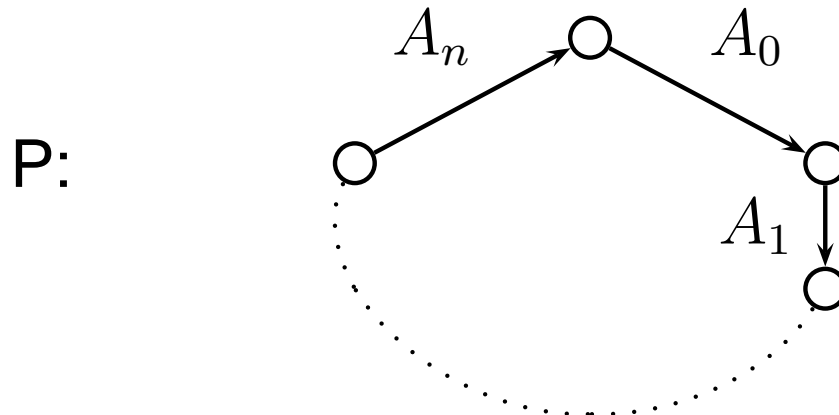
Pnueli-Rosner setting



$$A_c^4 = A_{out}$$
$$A_c^1 = A_{12} \cup A_{13}$$
$$\times C_1 \times \dots \times C_4$$
$$A_o^1 = A_{in} \cup A_{12} \cup A_{13}$$
$$A_o^4 = A_{out} \cup A_{24} \cup A_{34}$$



- Formalization in our setting:



Each C_i can see only $A_{i-1} \cup A_i$ and can control only A_i .

$$P \times C_1 \times \cdots \times C_n \models \varphi$$

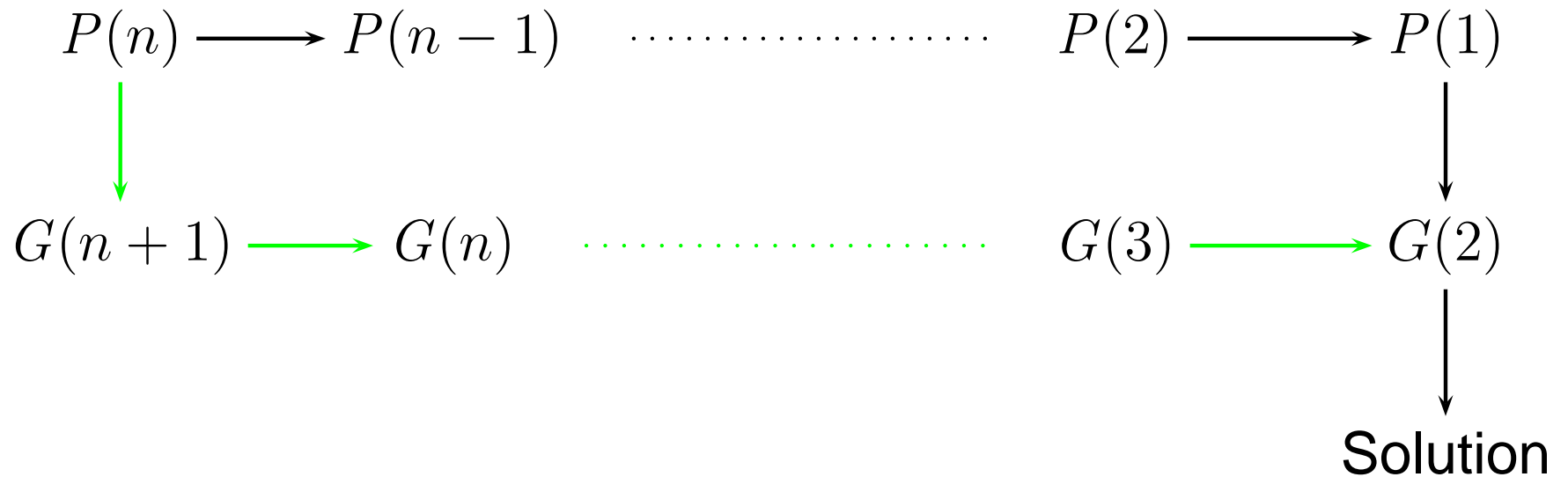
Thm [Pnueli& Rosner]: This synthesis problem is decidable.

- The operation \mathcal{A}/\mathcal{B} is only defined when \mathcal{B} does not use \circlearrowleft .
- This does not suffice to solve the synthesis problem for pipelines.
- An extension of \mathcal{A}/\mathcal{B} permitting some use of \circlearrowleft is possible.
- This extension is difficult to prove and seems rather particular to pipeline problem.
- The operation \mathcal{A}/\mathcal{B} is nice but exists for very few formalisms.

$P \in L(\mathcal{A}/\mathcal{B})$ iff there is C such that
 $C \in L(\mathcal{B})$ and $P \times C \in L(\mathcal{A})$.

- It would be easier if instead of trying to eliminate one controller we could reduce to games with many players.





- The game setting can be:
 - more general,
 - combinatorially easier to handle.

Part IIb

Undecidability

Distributed synthesis is difficult

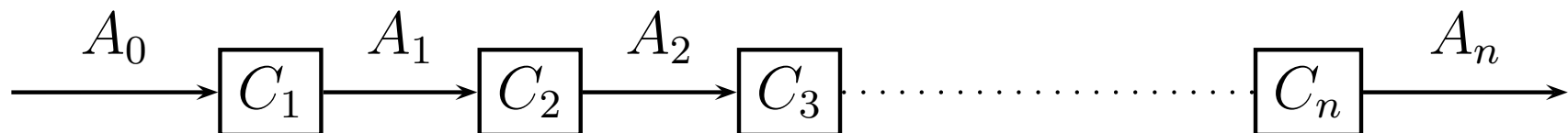
Fact: The following problem is undecidable:

Given α, β_1, β_2 are there C_1, C_2 such that $C_1 \times C_2 \models \alpha$ and $C_1 \models \beta_1, C_2 \models \beta_2$.

Thm[Pnueli & Rosner]: The problem:

For a fixed architecture, given α are there controllers that make the system satisfy α .

is decidable only for pipelines.



Distributed synthesis is difficult

Fact: The following problem is undecidable:

Given α, β_1, β_2 are there C_1, C_2 such that $C_1 \times C_2 \models \alpha$ and $C_1 \models \beta_1, C_2 \models \beta_2$.

Thm[Pnueli & Rosner]: The problem:

For a fixed architecture, given α are there controllers that make the system satisfy α .

is decidable only for pipelines.



An undecidable control problem

Given \mathcal{A} , \mathcal{B}_1 and \mathcal{B}_2 check if there exists C_1, C_2 such that $C_i \in L(\mathcal{B}_i)$ and $C_1 \times C_2 \in L(\mathcal{A})$.

Thm: The problem is undecidable if \mathcal{B}_1 and \mathcal{B}_2 are allowed to mention \circlearrowleft operation.

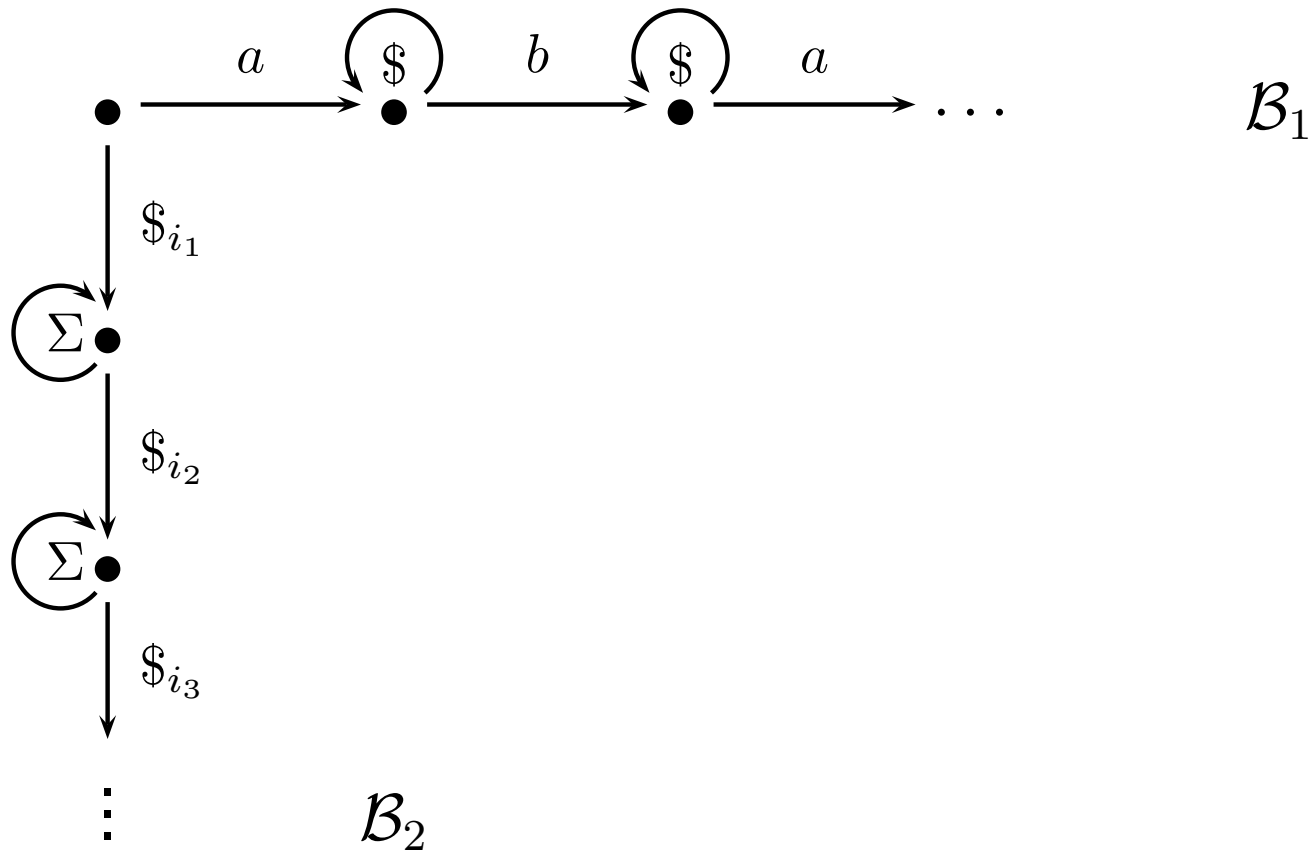
- Post correspondence problem
- Given $\{(u_1, v_1), \dots, (u_k, v_k)\} \subseteq \Sigma^* \times \Sigma^*$.
- Find i_1, i_2, \dots, i_n such that:

$$u_{i_1} u_{i_2} \dots u_{i_n} = v_{i_1} v_{i_2} \dots v_{i_n}$$

- We construct \mathcal{A} , \mathcal{B}_1 and \mathcal{B}_2 such that the synthesis problem has a solution iff the Post correspondence problem has one.

- Post correspondence problem

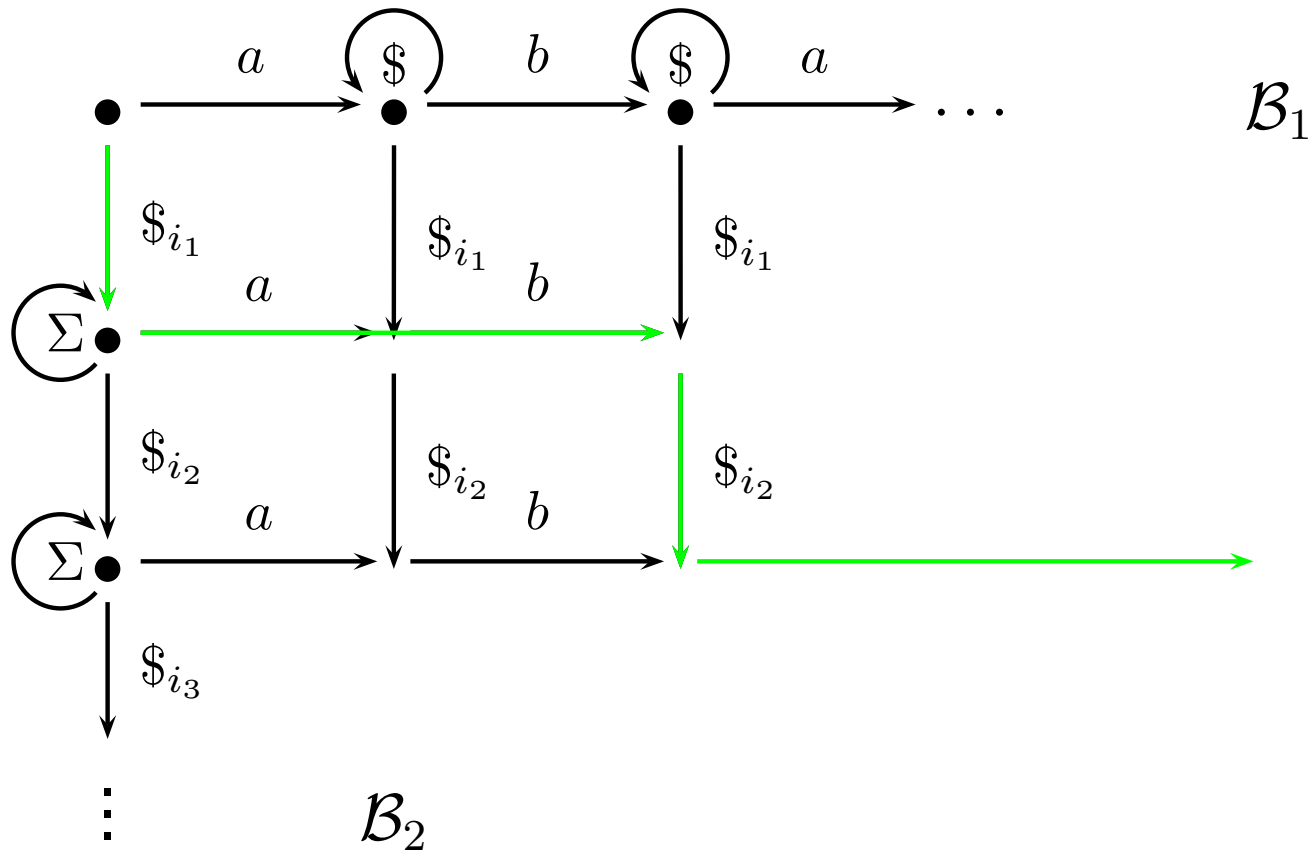
$$\{(u_1, v_1), \dots, (u_k, v_k)\} \subseteq \Sigma^* \times \Sigma^*.$$



- $L_1 = (\bigcup \$_i u_i)^* \#$
 $L_2 = (\bigcup \$_i v_i)^* \#$
 $\mathcal{A} = \exists L_1 \wedge \exists L_2.$

- Post correspondence problem

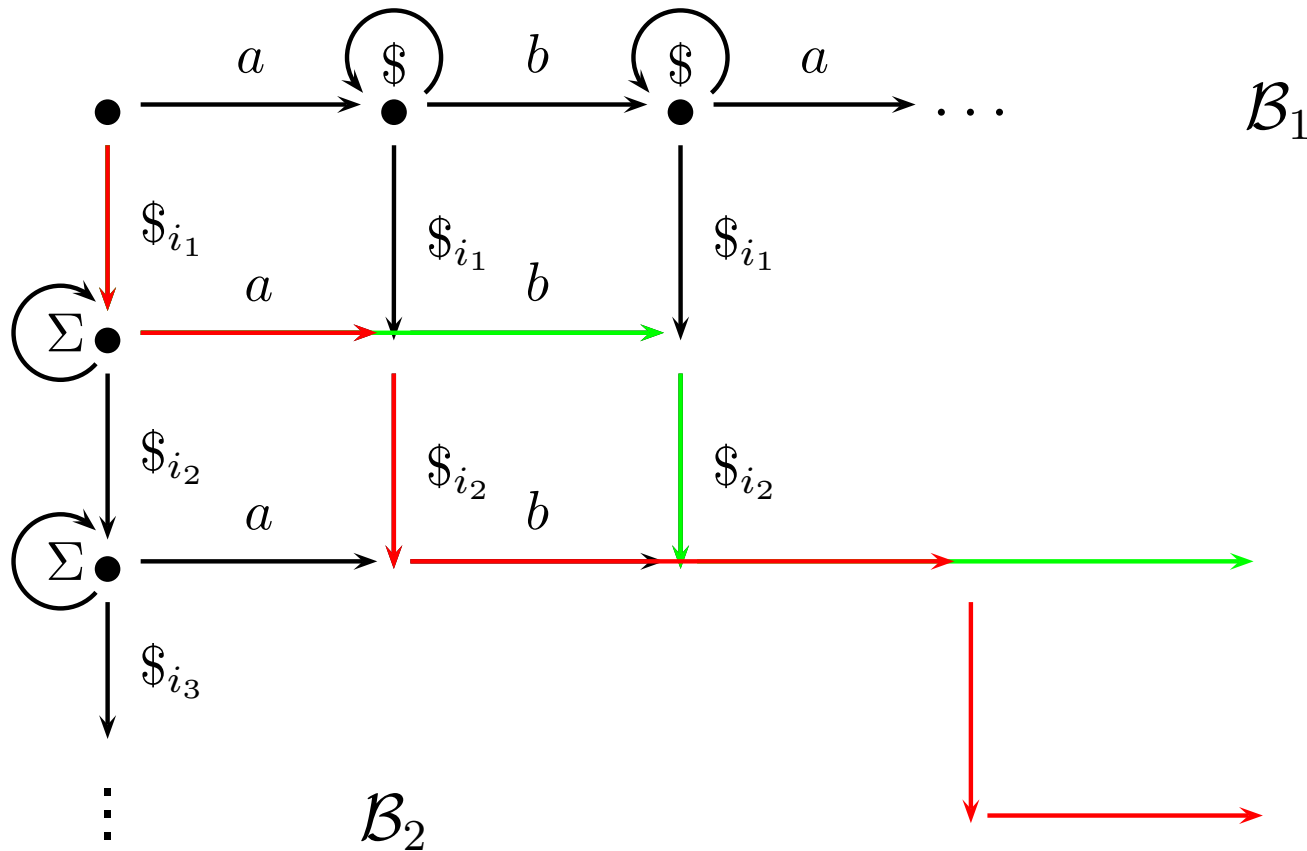
$$\{(u_1, v_1), \dots, (u_k, v_k)\} \subseteq \Sigma^* \times \Sigma^*.$$



- $L_1 = \left(\bigcup \$_i u_i \right)^* \#$
 $L_2 = \left(\bigcup \$_i v_i \right)^* \#$
 $\mathcal{A} = \exists L_1 \wedge \exists L_2.$

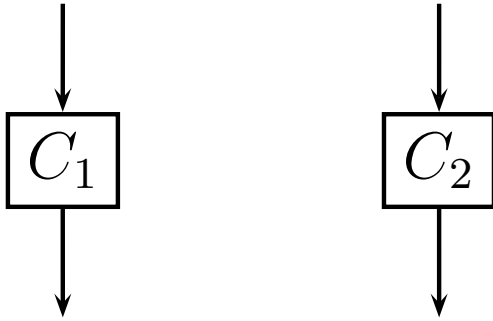
- Post correspondence problem

$$\{(u_1, v_1), \dots, (u_k, v_k)\} \subseteq \Sigma^* \times \Sigma^*.$$



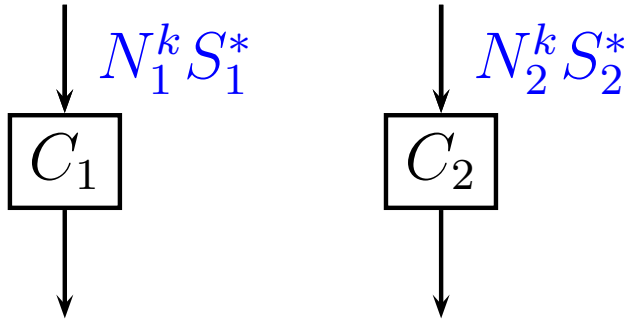
- $L_1 = (\bigcup \$_i u_i)^* \#$
 $L_2 = (\bigcup \$_i v_i)^* \#$
 $\mathcal{A} = \exists L_1 \wedge \exists L_2.$

Undecidability for architectures



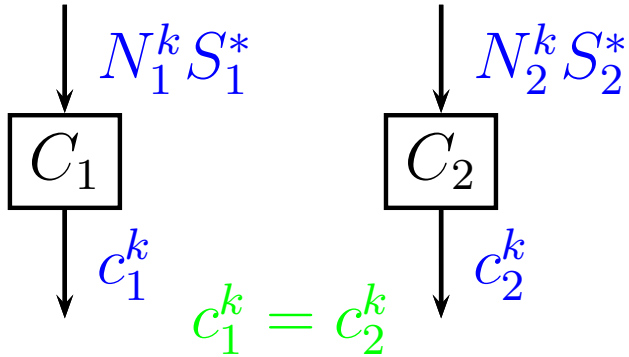
- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.

Undecidability for architectures



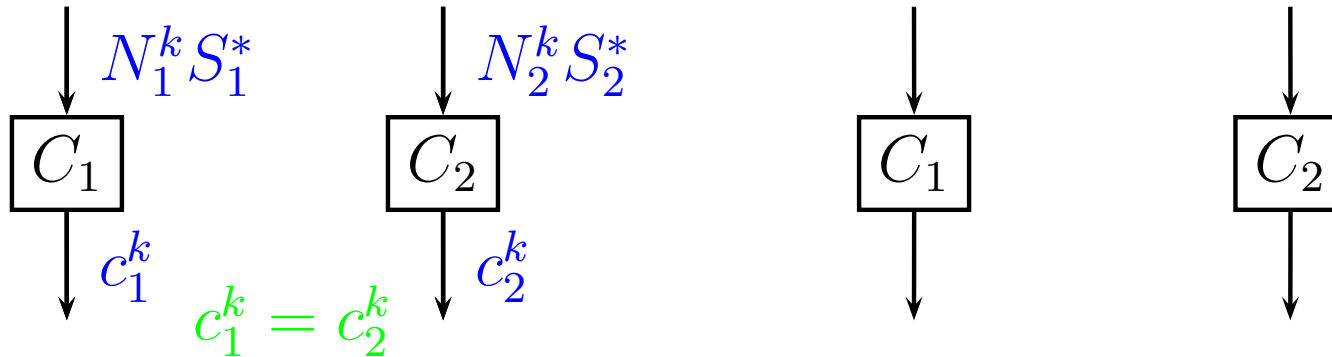
- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.

Undecidability for architectures



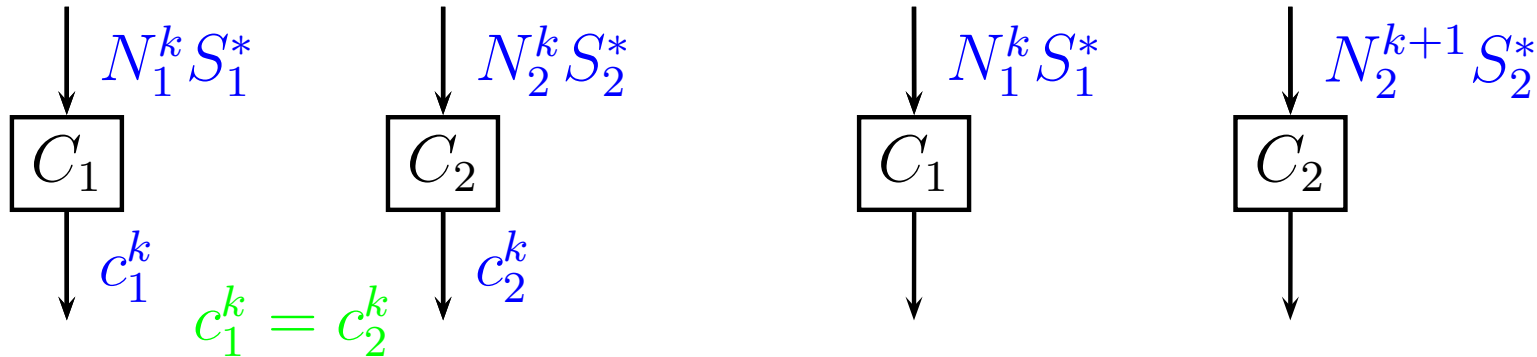
- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.

Undecidability for architectures



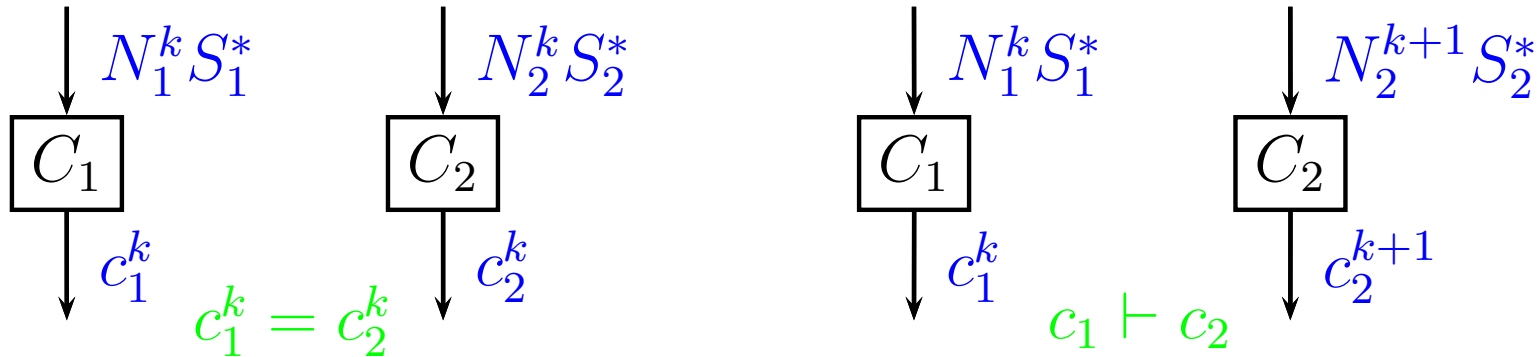
- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.

Undecidability for architectures



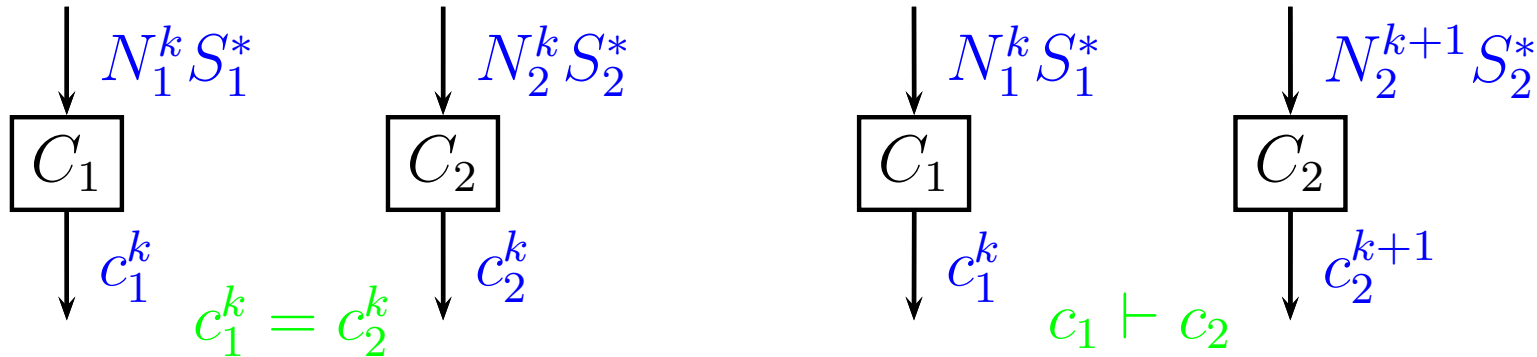
- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.

Undecidability for architectures



- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.

Undecidability for architectures



- The synthesis problem for this architecture is undecidable.
- Consider a Turing machine M . We construct a specification α such that there is a controller for α iff M stops on empty tape.
- N_1, N_2 – next configuration, S_1, S_2 – next letter.
- The specification implies that for all k :

$$c_1^k = c_2^k \text{ and } c_1^k \vdash c_2^{k+1}.$$

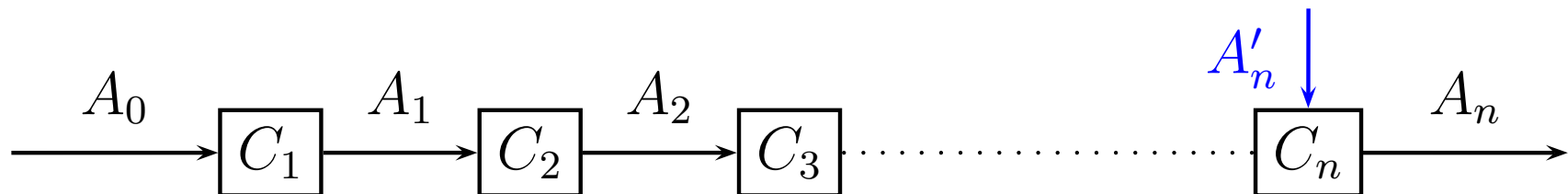
Distributed synthesis is difficult (2)

● A specification is **local** if it is a conjunction of requirements on each controller.

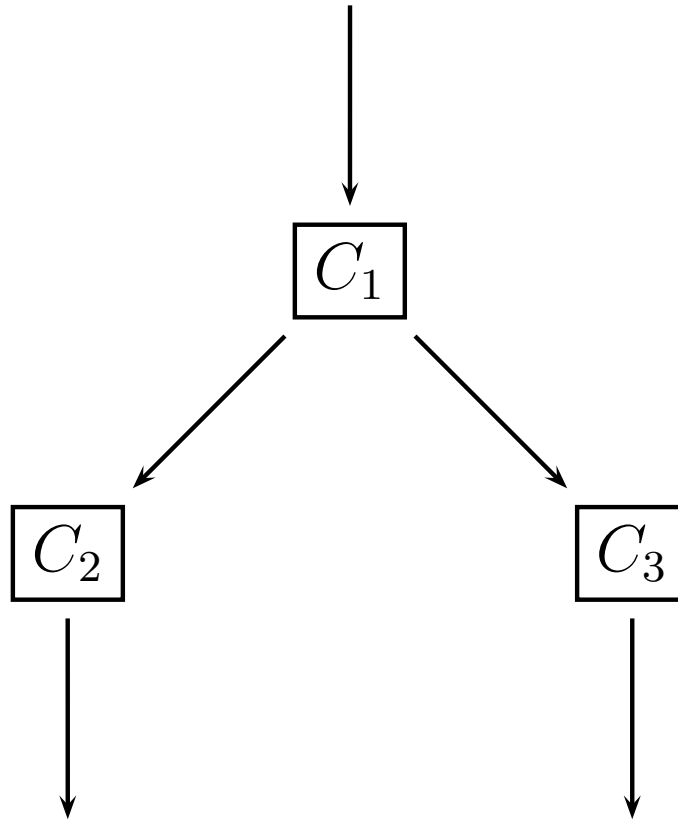
Thm[Madhusudan]: The problem:

For a fixed architecture given a **local** specification, are there controllers that make the system satisfy the specification.

is decidable only for double flanked pipelines.

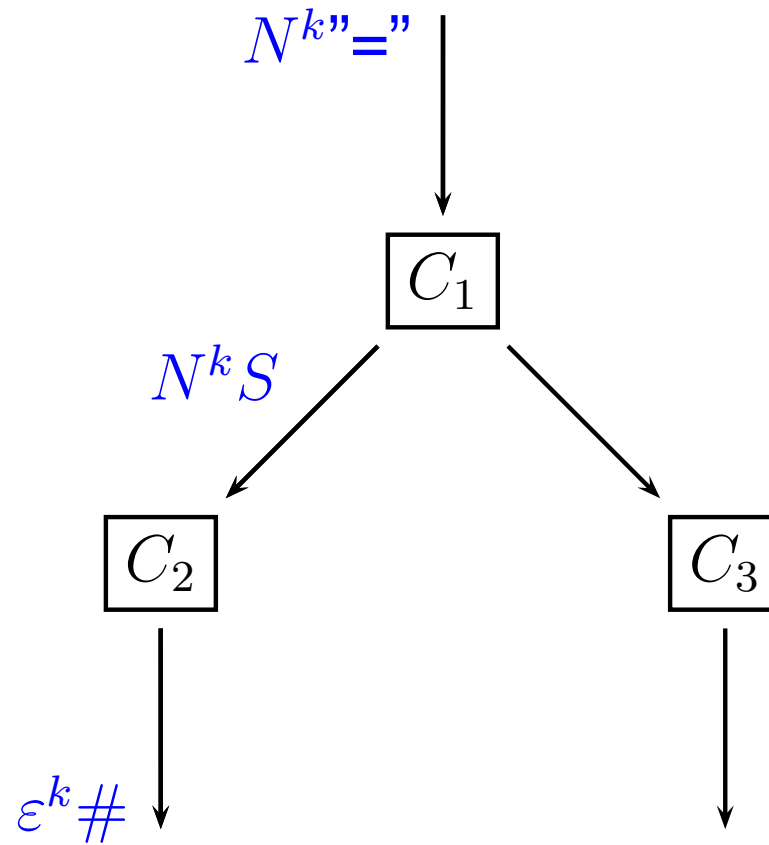


Undecidability (local specifications)

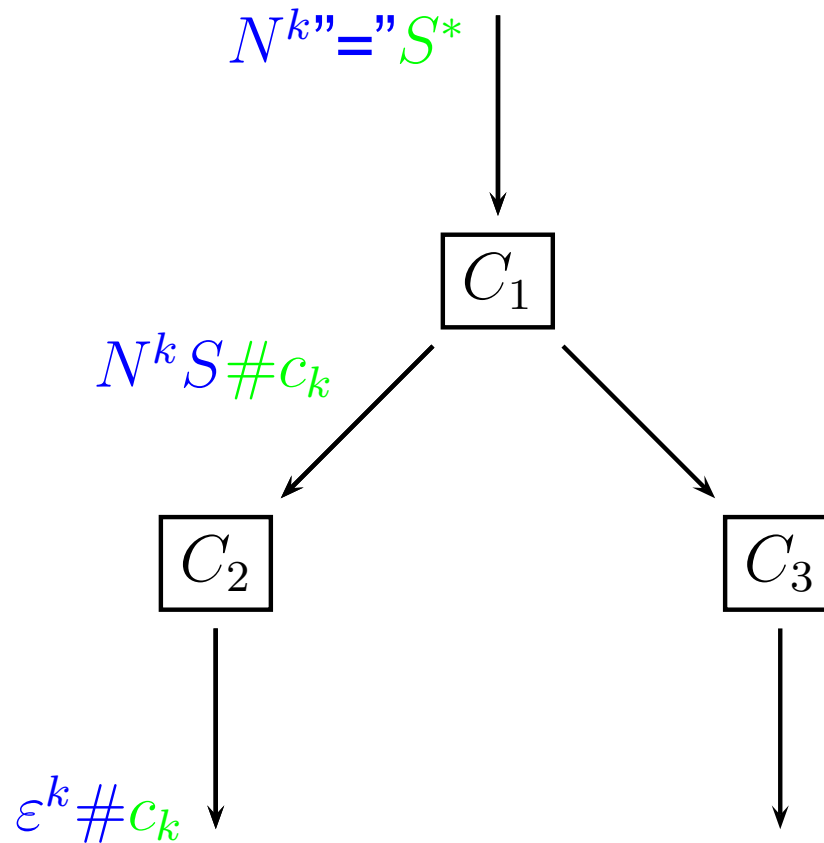


- For a given Turing machine M we find a specification that is realizable iff M halts on empty input.

Undecidability (local specifications)

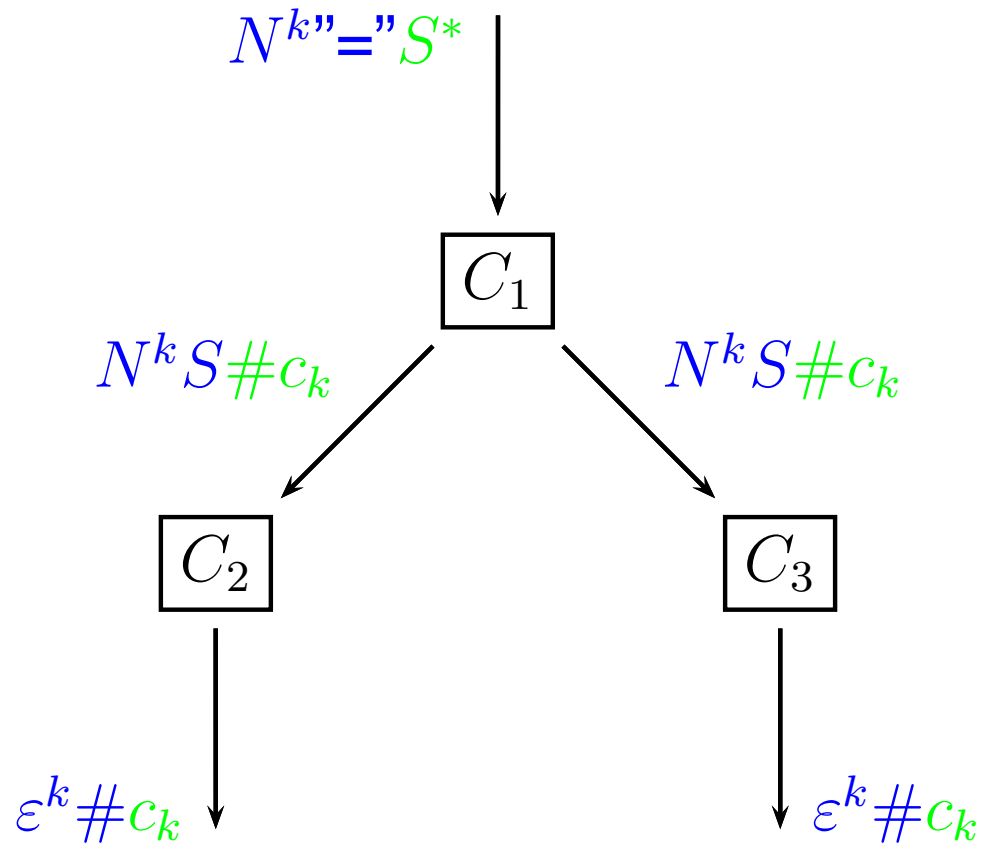


Undecidability (local specifications)



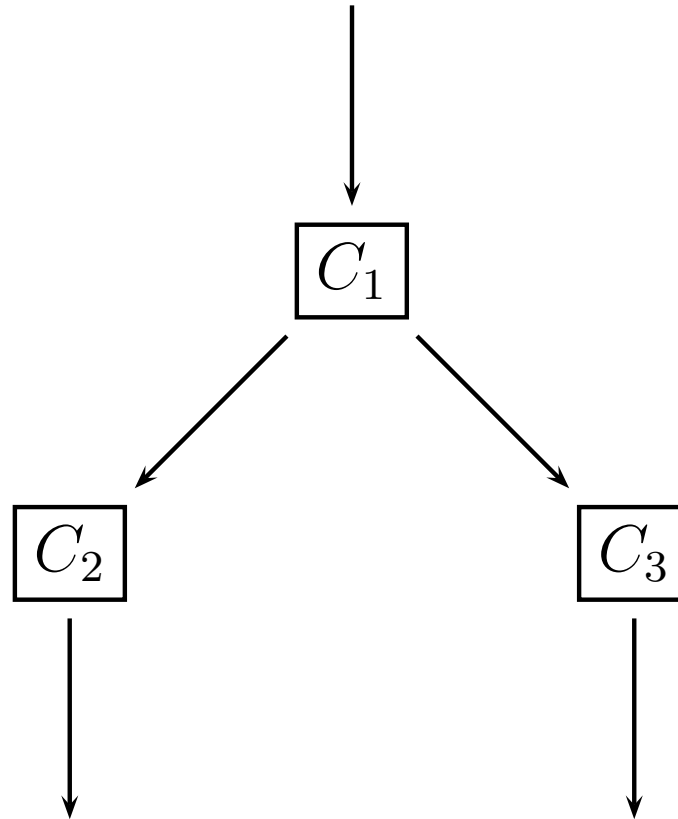
- This way after $N^k S$ there is only one possible sequence of actions

Undecidability (local specifications)

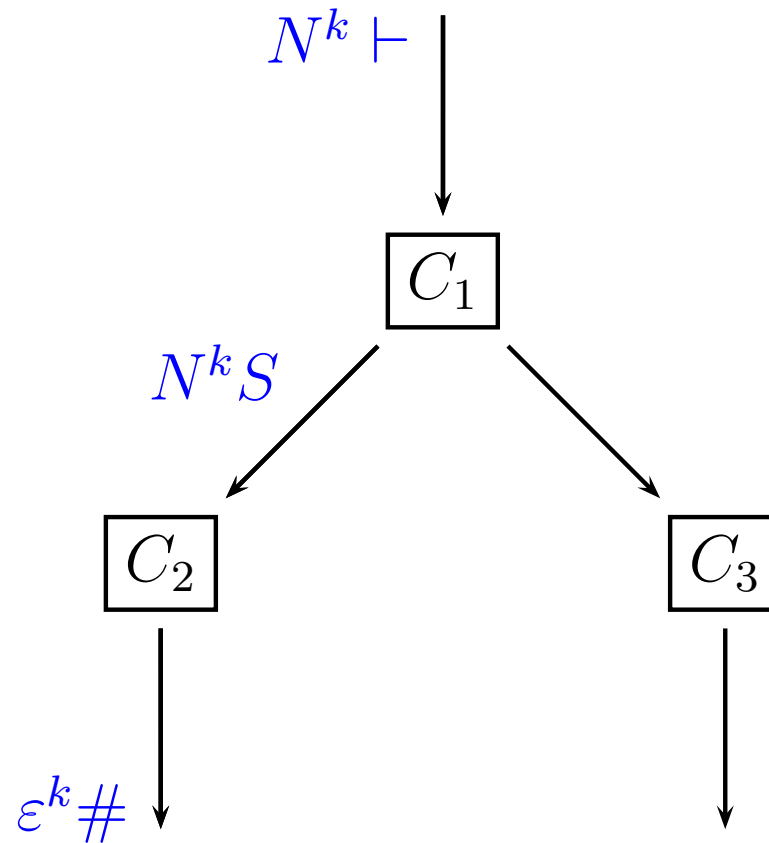


- This way after $N^k S$ there is only one possible sequence of actions

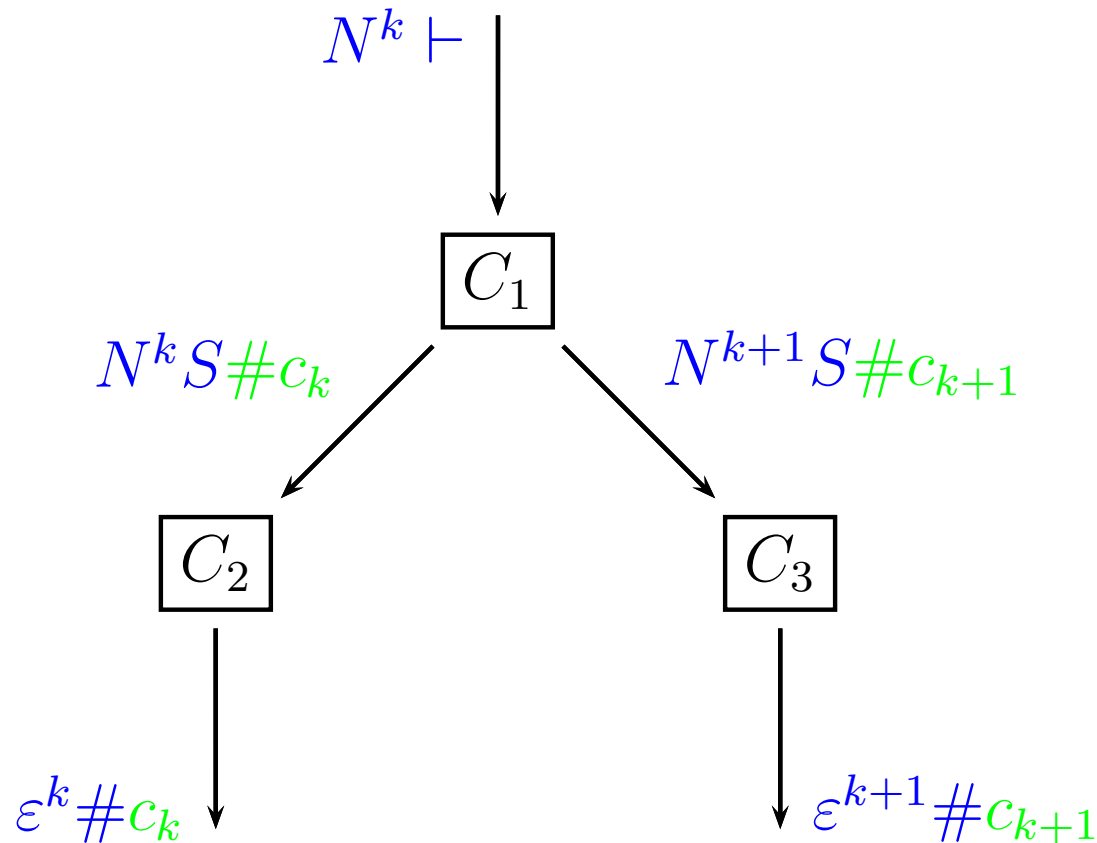
Undecidability (local specifications)



Undecidability (local specifications)



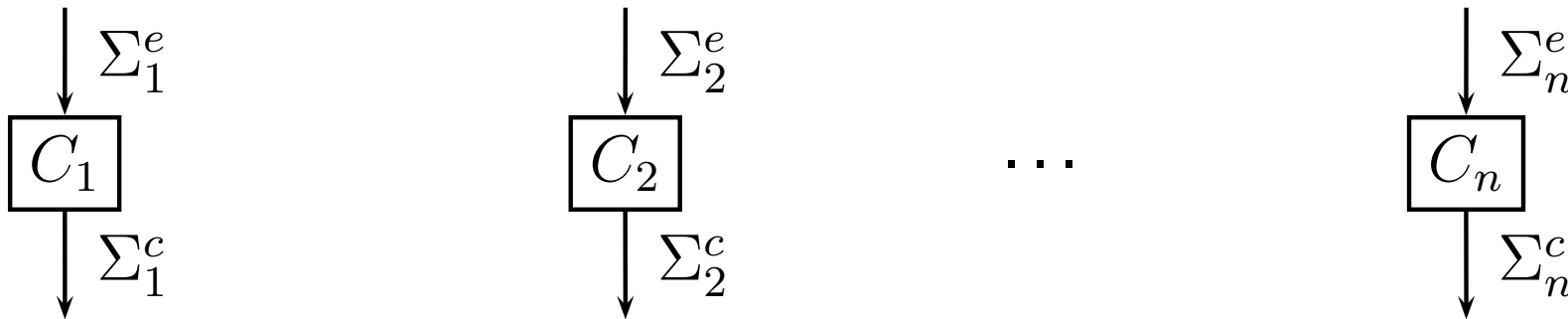
Undecidability (local specifications)



- This way we guarantee that $C_k \vdash C_{k+1}$ by just putting the conditions on C_1 .

Distributed synthesis is difficult (3)

Def: Communicating system:



Processes can synchronize on controllable actions.

Def: A strategy is **clocked** if it depends only on time and not on letters of the input. A strategy is **com-rigid** if it fixes communication.

Thm[Madhusudan & Thiagarajan]: It is decidable if for a given trace-closed specification there is a clocked and com-rigid specification realizing it.

- Unobservability may lead to undecidability.
- For most architectures there are specifications that make the problem undecidable.
- It may be more fruitful to take a specification into account and look for which pairs (architecture, specification) the problem is decidable.
- Idea: Compile (architecture, specification) pair into a game and use tools developed there.
- Problem: Compiling to two player games does not make much sense.
- Distributed games to distributed strategies as standard games to centralized strategies.

Part IIc

Distributed games

A naive parallel composition

- Take two transition systems $\mathcal{M}_1 = \langle V_1, T_1 \rangle$ and $\mathcal{M}_2 = \langle V_2, T_2 \rangle$

- A parallel composition of the two is:

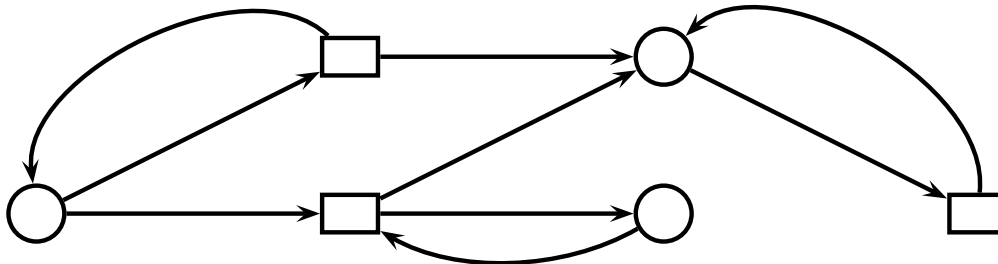
$$\mathcal{M}_1 \times \mathcal{M}_2 = \langle V_1 \times V_2, T_{\times} \rangle$$

where $(s_1, s_2) \rightarrow (t_1, t_2) \in T_{\times}$ iff $s_1 \rightarrow t_1 \in T_1$ and $s_2 \rightarrow t_2 \in T_2$.

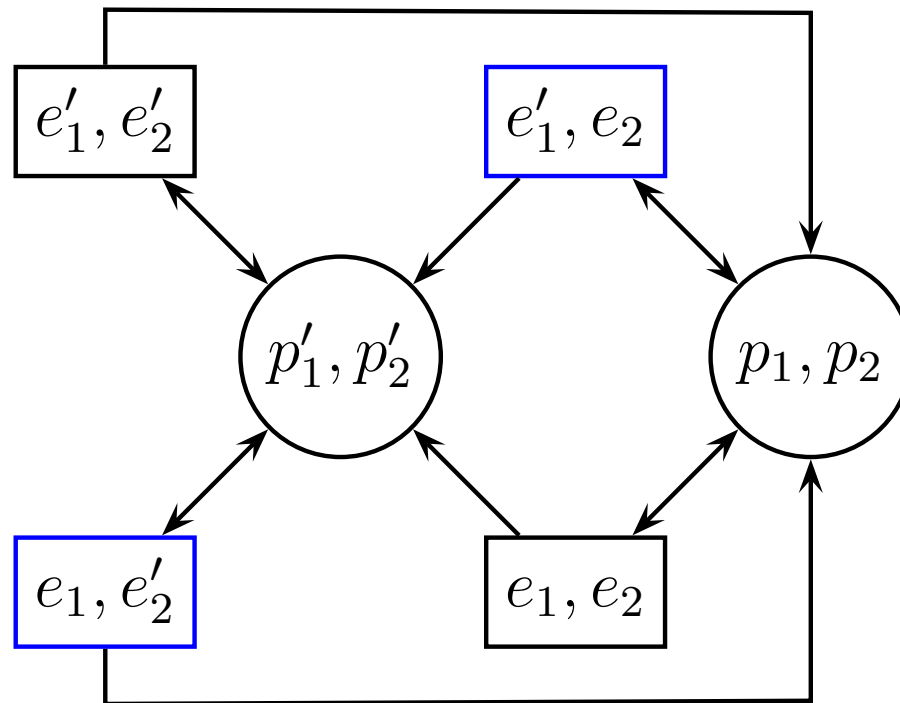
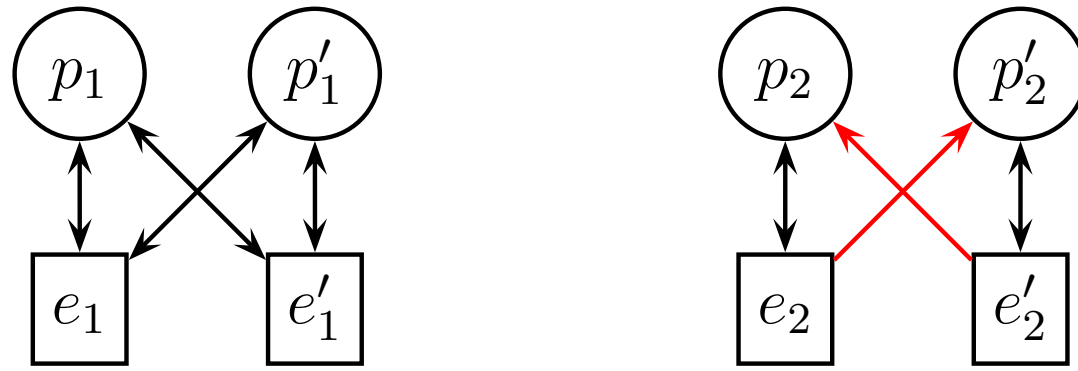
- Eventually some more transitions in $\mathcal{M}_1 \times \mathcal{M}_2$ are suppressed.

- We can also have “asynchronous product” $(s_1, s) \rightarrow (t_1, s) \in T_{\times}$ if $s_1 \rightarrow t_1 \in T_1$

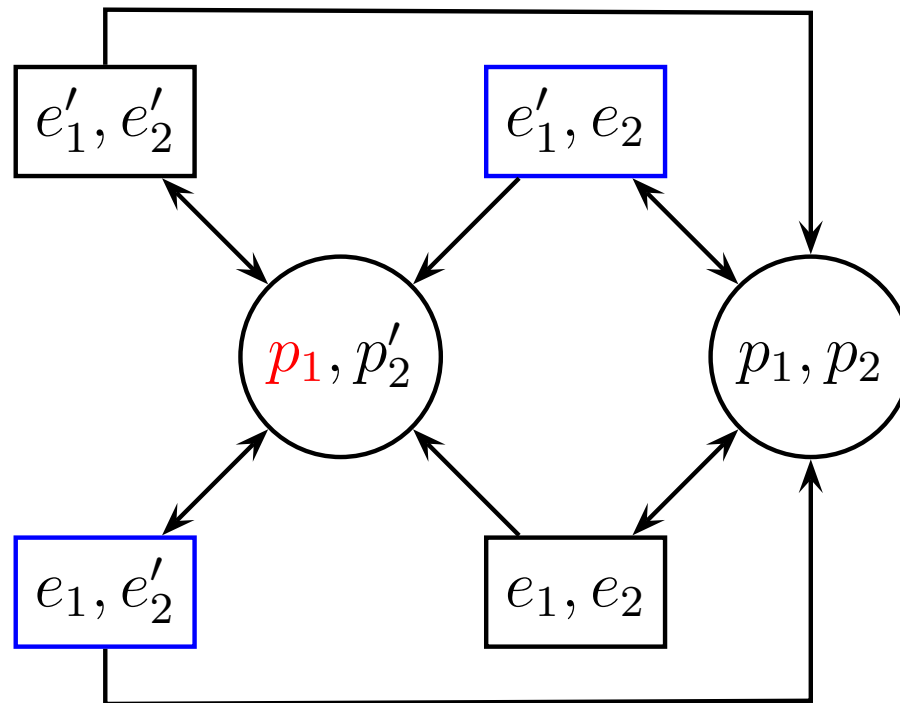
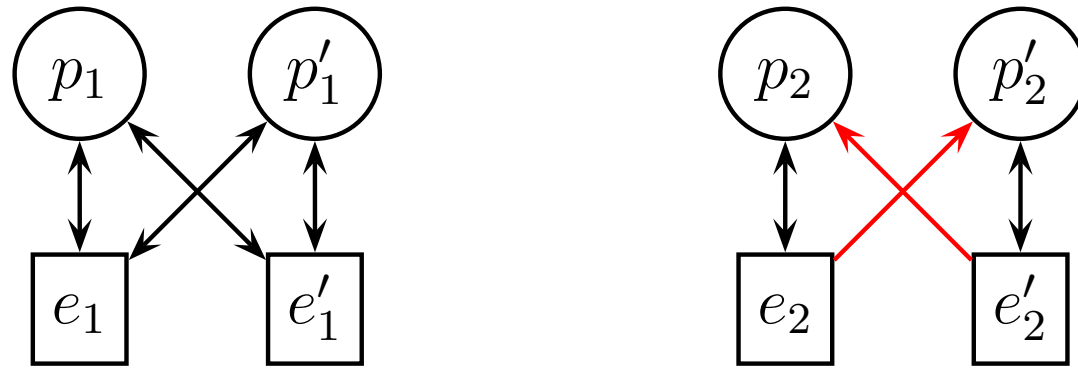
- We try to do the same with games:



- Take n “local” games $G_i = \langle P_i, E_i, T_i \rangle$. (bipartite)
- Distributed game $\mathcal{G} = \langle P, E, T, Acc \subseteq (E \cup P)^\omega \rangle$.
- $E = E_1 \times \cdots \times E_n$,
- $P \subseteq (P_1 \cup E_1) \times \cdots \times (P_n \cup E_n) \setminus E$.
- Environment moves: $[e_1, \dots, e_n] \rightarrow (x_1, \dots, x_n)$ with
 $x_i = e_i$ or $e_i \rightarrow x_i$.
Some of these transitions can be suppressed.
- Player moves: $(x_1, \dots, x_n) \rightarrow [e_1, \dots, e_n]$ with
 $x_i = e_i$ or $x_i \rightarrow e_i$.
Every such transition must be present.



- Goal: Avoid blue positions.



- Goal: Avoid blue positions.

- Given a play \vec{v} in \mathcal{G} , a **view** of player i is $view_i(v) \in (E_i \cdot P_i)^\omega$.

$$(e_1, e_2, \dots, e_n) \quad e_1$$

$$(e_1, p_2, \dots, e_n)$$

$$(e_1, e'_2, \dots, e_n)$$

$$(p_1, e_2, \dots, e_n) \quad p_1$$

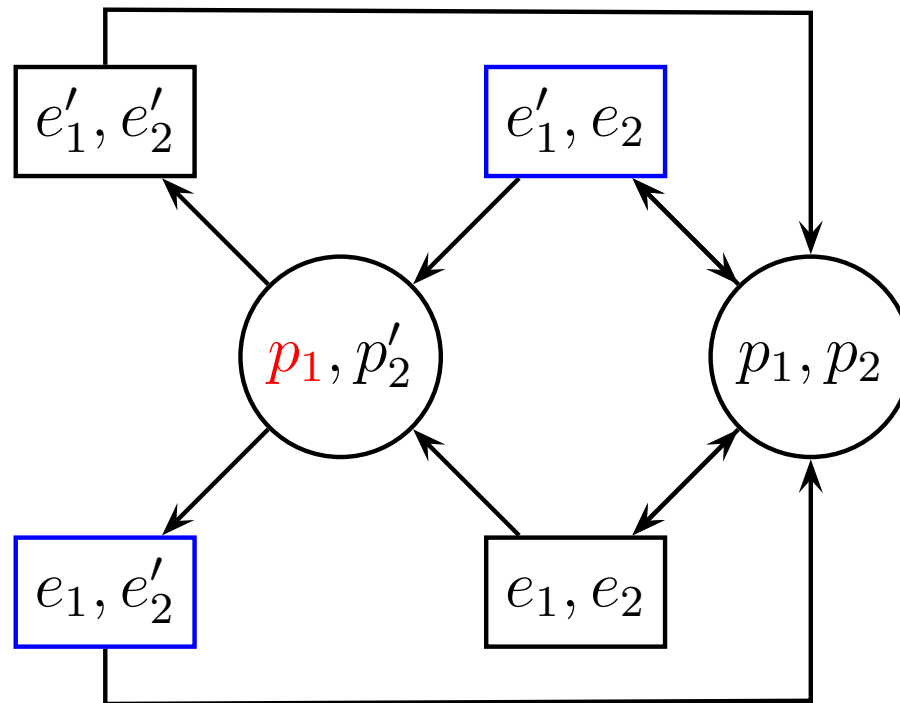
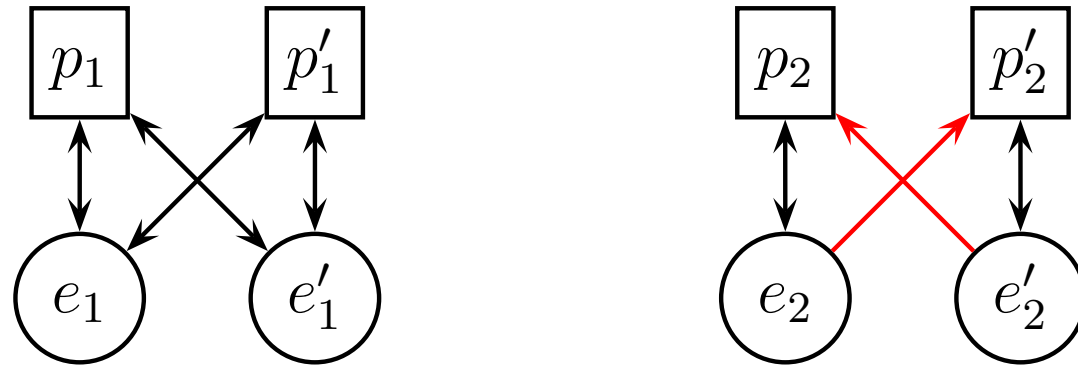
$$(e'_1, e'_2, \dots, e_n) \quad e_1$$

- An **i -local strategy** is a strategy in the game G_i .
- Distributed strategy** is a tuple $\langle \sigma_1, \dots, \sigma_n \rangle$ of local strategies.

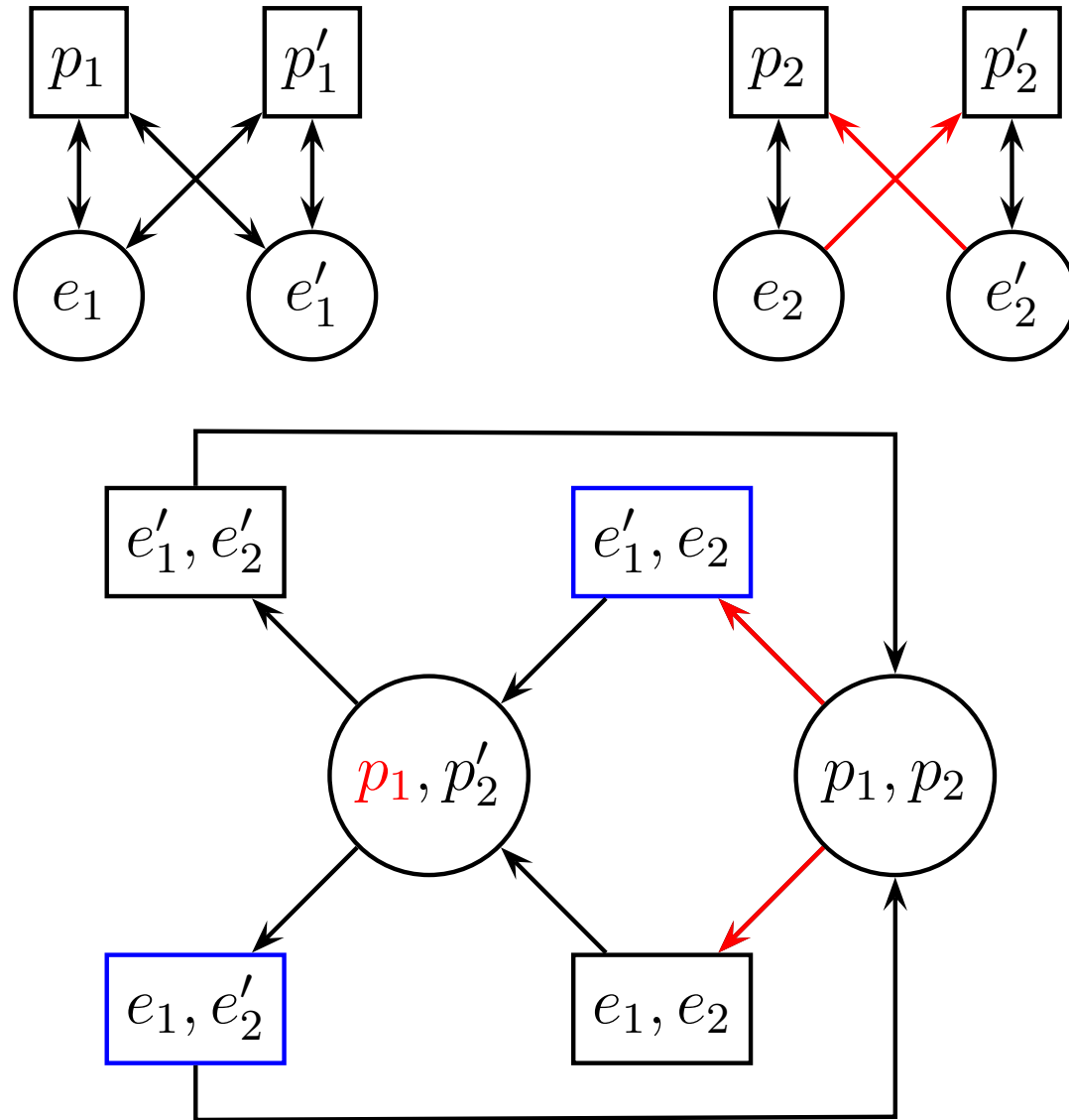
$$\sigma(\vec{v} \cdot (x_1, \dots, x_n)) = (e_1, \dots, e_n)$$

where $e_i = x_i$ or $e_i = \sigma_i(view_i(\vec{v} \cdot x_i))$.

- Players may have a global strategy in a game but not a distributed one. (Distributed games are not determined).
- It is not decidable if there is a distributed winning strategy in a given distributed game.
- There may be a positional global strategy but all distributed strategies may require memory.



- Goal: Avoid blue positions.

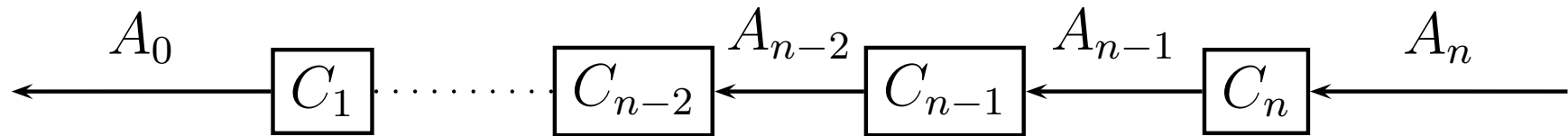


- Goal: Avoid blue positions.

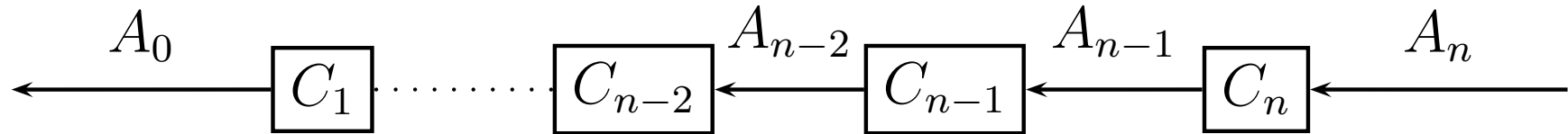
- The game is **deterministic for the environment** iff every environment position has at most one successor.

- If player has a global strategy in such a game then he has a local strategy.

Cor: Environment deterministic distributed games are solvable.
(Existence of distributed strategies is decidable).



- Execution is a word from $(A_n A_{n-1} \dots A_0)^\omega$.
- For given controllers the set of executions forms a tree.
- We specify the set of allowed execution paths (linear-time specifications).



- For each component G_i :
+ $P_i = A_i$, $E_i = (A_i \rightarrow A_{i-1})$, full graph of transitions.

- Spec. give by a deterministic automaton

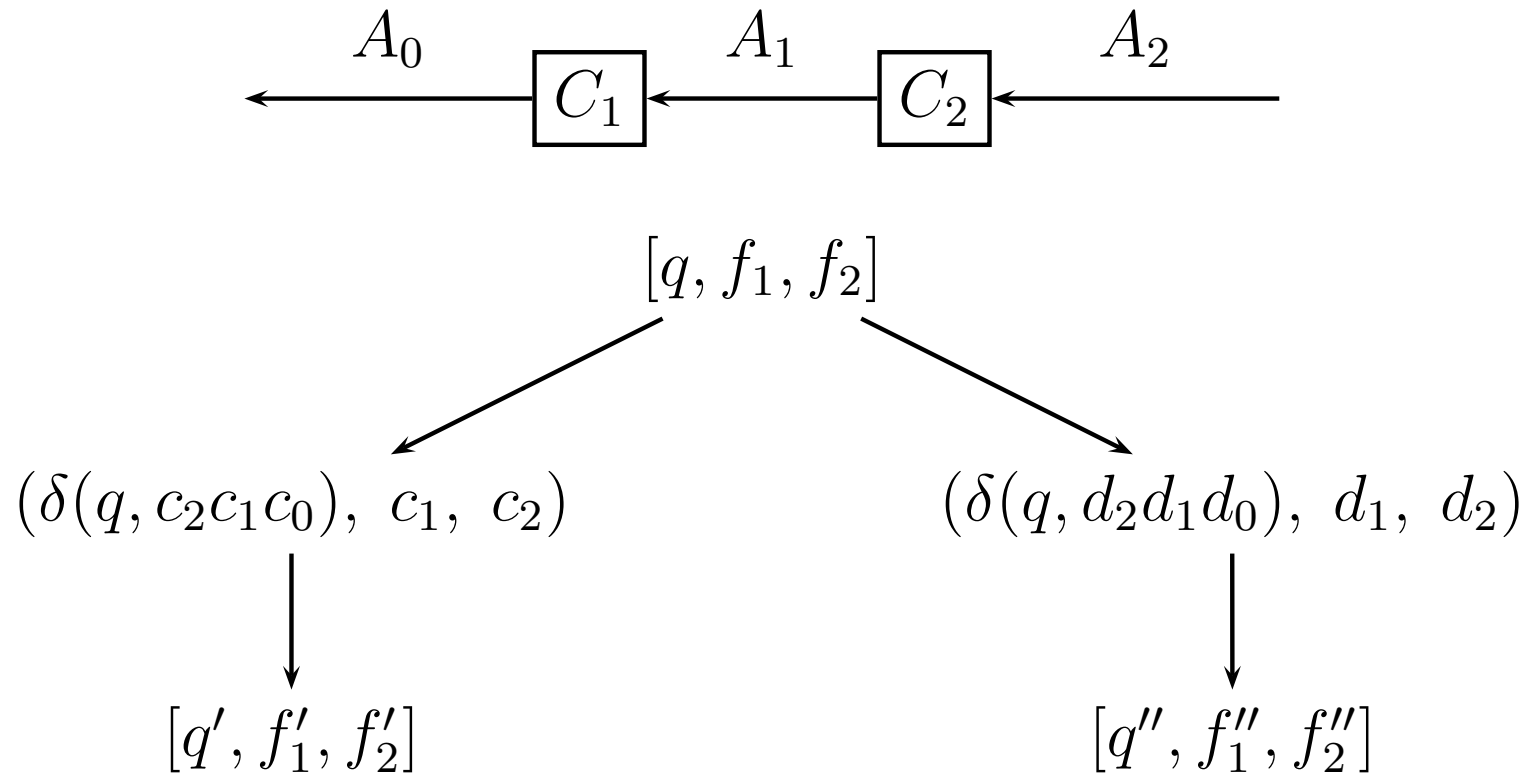
$$\mathcal{A} = \langle Q, \Sigma = A_0 \times \dots \times A_n, q^0, \delta, Acc \rangle$$

- Global transitions:

$$[q, f_1, \dots, f_n] \rightarrow (\delta(q, a_n \dots a_0), a_1, \dots, a_n)$$

where $a_{i-1} = f_i(a_i)$. So all is determined by a_n !

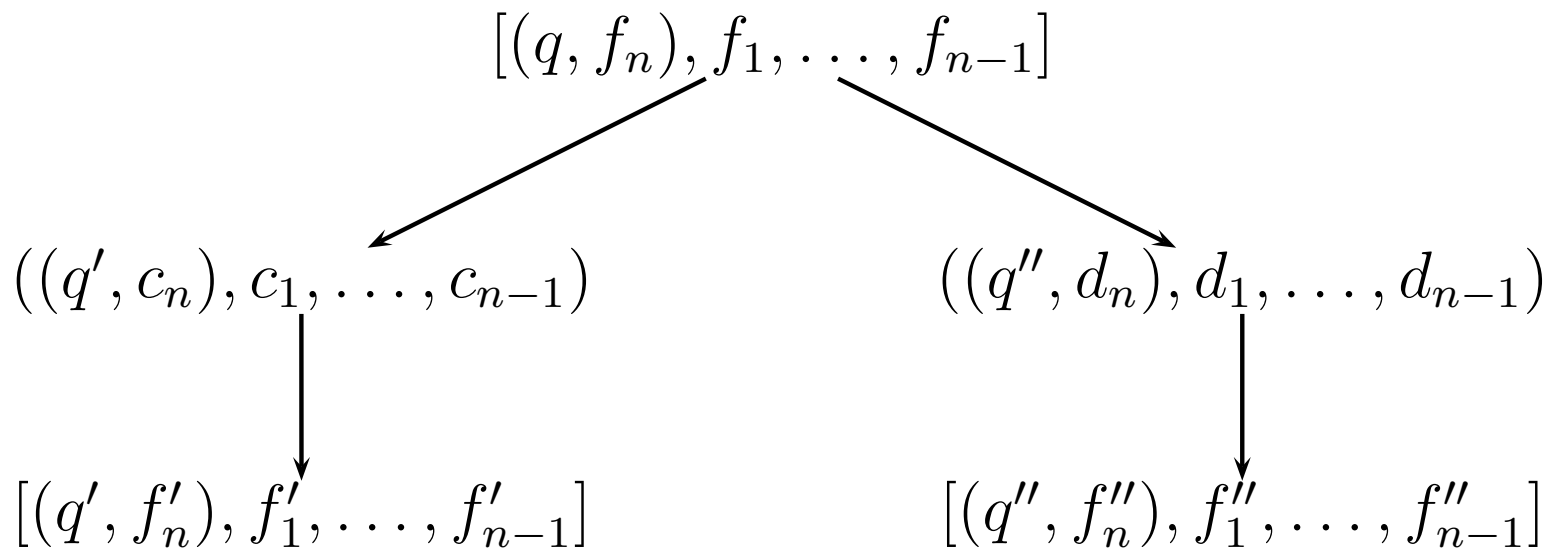
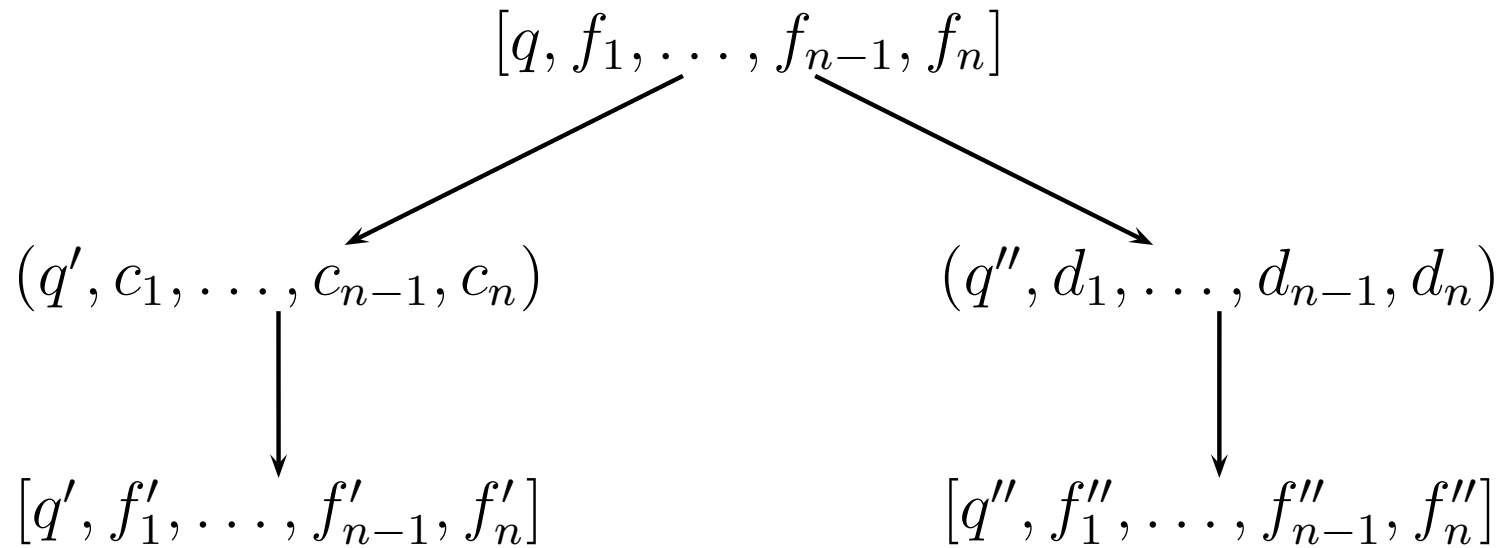
Coding pipeline into game: example



$$c_1 = f_2(c_2), c_0 = f_1(c_1)$$

$$d_1 = f_2(d_2), d_0 = f_1(d_1)$$

Dividing out last player



- Game is **k -deterministic** if for every environment position η :
when $\eta \rightarrow \pi_1, \pi_2$ then $\pi_1[k] \neq \pi_2[k]$.

- Game is **dividable** if it is 0 and n -deterministic.

- **Division operation:**

$$(x_0, x_1, \dots, x_{n-1}, x_n) \mapsto ((x_0, x_n), x_1, \dots, x_{n-1})$$

Thm: For every dividable game \mathcal{G} of $n + 1$ players, the distributed $\text{DIVIDE}(\mathcal{G})$ is a game with n players such that:

there is a distributed strategy from η in \mathcal{G} iff there is a distributed strategy from $\tilde{\eta}$ in $\text{DIVIDE}(\mathcal{G})$.

- Game is **k -deterministic** if for every environment position η :
when $\eta \rightarrow \pi_1, \pi_2$ then $\pi_1[k] \neq \pi_2[k]$.

- Game is **dividable** if it is 0 and n -deterministic.

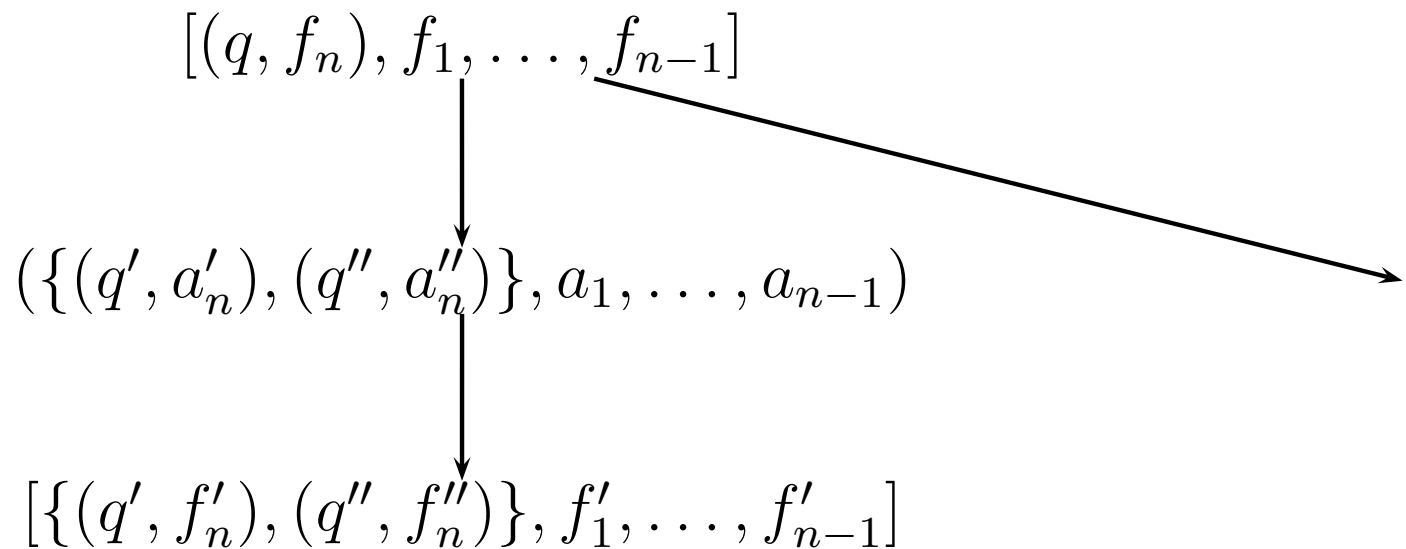
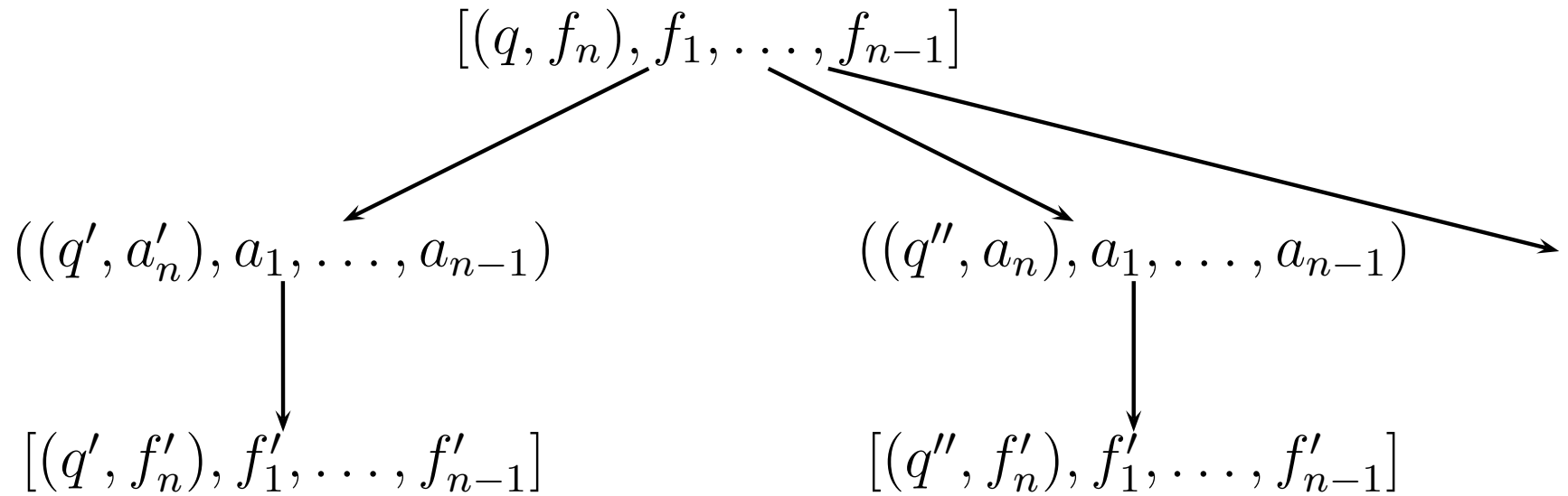
- **Division operation:**

$$(\underset{\eta}{x_0, x_1, \dots, x_{n-1}, x_n}) \mapsto ((\underset{\tilde{\eta}}{x_0, x_n}), x_1, \dots, x_{n-1})$$

Thm: For every dividable game \mathcal{G} of $n + 1$ players, the distributed $\text{DIVIDE}(\mathcal{G})$ is a game with n players such that:

there is a distributed strategy from η in \mathcal{G} iff there is a distributed strategy from $\tilde{\eta}$ in $\text{DIVIDE}(\mathcal{G})$.

- $\tilde{P}_i = P_i$, $\tilde{E}_i = E_i$ and $\tilde{T}_i = T_i$ for all $i = 1, \dots, n - 1$;
- $\tilde{P}_0 = P_0 \times P_n$ and $\tilde{E}_0 = E_0 \times E_n$;
- $(p_0, p_n) \rightarrow (e_0, e_n) \in \tilde{T}_0$ provided $p_0 \rightarrow e_0 \in T_0$ and $p_n \rightarrow e_n \in T_n$;
- $[(e_0, e_n), e_1, \dots, e_{n-1}] \rightarrow ((p_0, p_n), p_1, \dots, p_{n-1}) \in \tilde{T}$ if $[e_0, e_1, \dots, e_{n-1}, e_n] \rightarrow (p_0, p_1, \dots, p_{n-1}, p_n) \in T$
- $\widetilde{Acc} = \{flat^{-1}(\vec{v}) \mid \vec{v} \in Acc\}$.



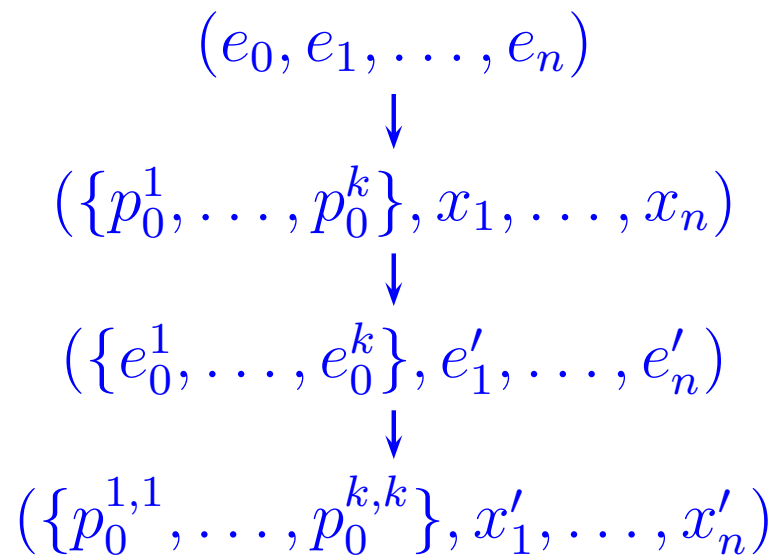
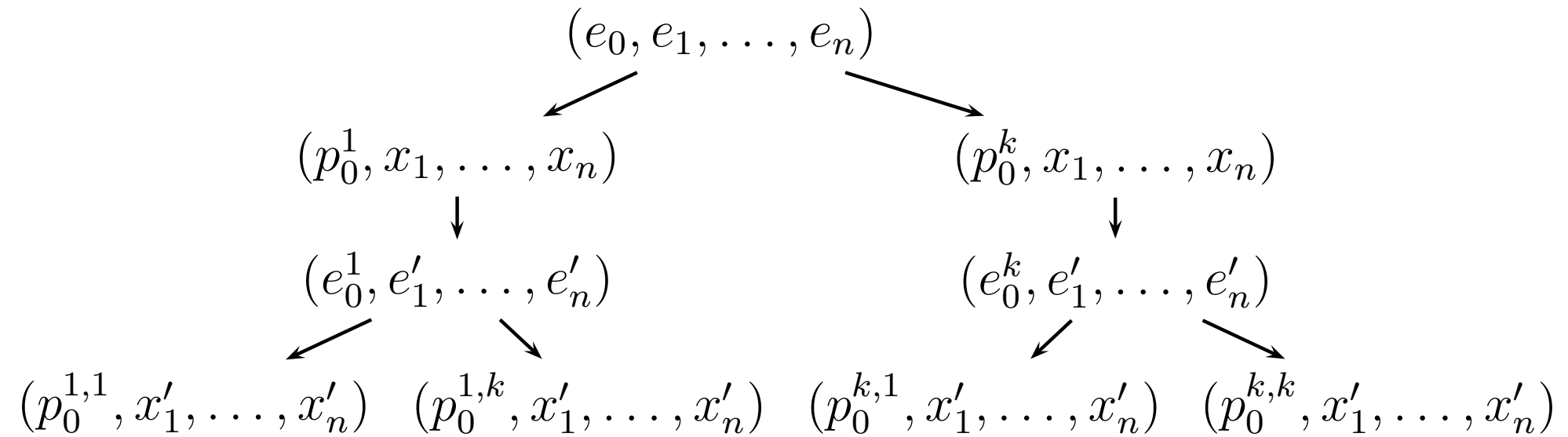
- The game is **0-active** if all player positions have player node on 0-th component.
- The game is **gluable** if:
 - it is 0-deterministic,
 - 0-active,
 - with a parity condition on the states of player 0.

Thm: Let \mathcal{G} be a gluable game. For every position η in \mathcal{G} there is a position $\hat{\eta}$ in $\text{GLUE}(\mathcal{G})$ such that:

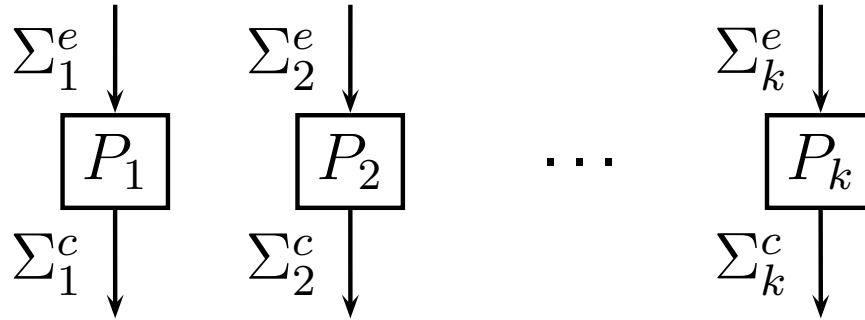
there is a distributed strategy from η in \mathcal{G} iff there is a distributed strategy from $\hat{\eta}$ in $\text{GLUE}(\mathcal{G})$.

The game $\text{GLUE}(\mathcal{G})$ is $(1 \cdots n)$ -deterministic.

- $\tilde{P}_i = P_i$, $\tilde{E}_i = E_i$ and $\tilde{T}_i = T_i$ for all $i = 1, \dots, n$;
- $\tilde{P}_0 = \mathcal{P}(E_0 \times P_0)$ and $\tilde{E}_0 = \mathcal{P}(P_0 \times E_0)$;
- $\tilde{p} \rightarrow_0 \tilde{e}$ if for every $(e, p) \in \tilde{p}$ there is $(p, e') \in \tilde{e}$ with $p \rightarrow e' \in T_0$.
- $(\tilde{e}_0, \bar{e}) \rightarrow (\tilde{p}_0, \bar{x}) \in \tilde{T}$ for $\tilde{x}_0 \neq \emptyset$, where
 $\tilde{p}_0 = \{(e_0, p_0) \in \tilde{P} : \exists (p', e_0) \in \tilde{e}_0. (e_0, \bar{e}) \rightarrow (p_0, \bar{x})\}$.
- \widetilde{Acc} is defined on traces.



- We start with $(n + 1)$ -player a game G that is 0 and n -deterministic.
- The game $\text{DIVIDE}(G)$ is n -player game that is 0-deterministic.
- The game $\text{GLUE}(\text{DIVIDE}(G))$ is n -player game that is 0 and $(n - 1)$ -deterministic.
- This way we reduce to a game with one player.



- $\Sigma_i^e \cap \Sigma_j^e = \emptyset$ but not necessary for Σ^c : $\theta(a) = \{i : a \in \Sigma_i^c\}$.
- Local strategy $\sigma_i : \Sigma_i^* \times \Sigma_i^e \rightarrow \mathcal{P}(\Sigma_i^c)$.
- Specification: $L \subseteq \bigcup \Sigma_i$, trace closed.
- Strategy is **com-rigid** if for every value R of σ_i and every $a, b \in R$: $\theta(a) = \theta(b)$.
- Strategy is **clocked** if for every $u, v \in \Sigma_i$, $\sigma_i(u) = \sigma_i(v)$ whenever $|u| = |v|$.

Thm [M. & T.]: It is decidable to check if there is a com-rigid and clocked strategy for a given trace-closed spec.

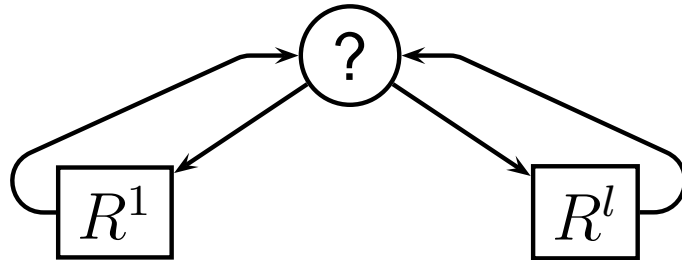
- We are given a system S of n components and a trace closed specification L .

- We can think that $L \subseteq \Sigma_{step}^\omega$ where

$$\Sigma_{step} = \Pi(\Sigma_i^e \cup \{\varepsilon\}) \cdot \Pi(\Sigma_i^c \cup \{\varepsilon\}).$$

- Hence automaton \mathcal{A} recognizing L reads a letter from Σ_{step} at each move.

- We can also assume that $L(\mathcal{A}) \subseteq L(S)$.



$$R^i \subseteq \Sigma_p^c \text{ and com-rigid}$$

- $E_0 = AP \times Q^A$
- $P_0 = E_0 \times \Sigma_{step}$

$$\begin{array}{c}
 ((ap, q), R_1, \dots, R_n) \\
 \downarrow \\
 ((ap, q, \zeta), ?, R_2, \dots, ?, R_n) \\
 \downarrow \\
 ((ap', q'), R'_1, R_2, \dots, R'_{n-1}, R_n)
 \end{array}$$

- After one application of gluing operation we get an environment deterministic game.

- Distributed games are in general neither determined nor algorithmically solvable.
- Many known settings of distributed synthesis are representable in distributed games.
 - Pipelines.
 - Local specification and double flanked pipelines.
 - Madhusudan & Thiagarajan setting.
 - Rudie & Wonham distributed control.
- The solutions require some coding and two theorems.
- Distributed games can be hopefully as useful for distributed synthesis problem as two player games are for the centralized synthesis problem.