Games on Asynchronous Transition Systems

Nehul Jain

IIT Bombay

Feb 2026

Advisor: Prof. Bharat Adsul

Motivation: Distributed Synthesis

- Reactive synthesis: given a specification, synthesize a system that meets the spec. Active research area.
- Distributed synthesis: given a distributed spec, synthesize a distributed system that meets the spec.
  1. First proposed by Pnueli and Rosner
     - ★ in Synchronous systems
     - ★ with message passing
     - ★ essentially undecidable beyond pipeline architecture
  2. Two actively studied models based on causal information
     - ★ Petri games by Finkbeiner et al and others [FO17]
     - ★ Asynchronous control games by Muscholl, Gimbert and others. [GGMW13]
     - ★ Several interesting (acyclic architectures) decidable cases. But undecidable in general!

Contributions

- A new model of distributed games: played on asynchronous transition systems
- Analysis for two processes/NFAs.
  - Addressed global safety, local reachability, and global reachability objectives.
  - Fixpoint-based algorithms.
  - Memory and computational complexity of the algorithm.
  - Hardness results
- Analysis of games with a Central Decision Maker(CDM)
  - Global safety and local parity condition on CDM process
  - Extraction of distributed strategies via special linearization from simple sequential game
  - Memory and computational complexity of the algorithm.
  - Hardness results
- Safety games with two decision makers are undecidable.
- Asynchronous control games and our games are equivalent.

An ATS game

An ATS game

- Distributed system is a **team** of processes vs Environment

Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

An ATS game

- Distributed system is a **team** of processes vs Environment
- Played on NFAs owned by the each processes which **communicate** via shared/joint actions

Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

An ATS game

- Distributed system is a **team** of processes vs Environment
- Played on NFAs owned by the each processes which **communicate** via shared/joint actions
- Actions

Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

An ATS game

- Distributed system is a **team** of processes vs Environment
- Played on NFAs owned by the each processes which **communicate** via shared/joint actions
- Actions
  - joint - a fixed set of processes/NFAs participate.

Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

An ATS game

- Distributed system is a **team** of processes vs Environment
- Played on NFAs owned by the each processes which **communicate** via shared/joint actions
- Actions
  - joint - a fixed set of processes/NFAs participate.
  - local - only one process participates.

Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

An ATS game

- Distributed system is a **team** of processes vs Environment
- Played on NFAs owned by the each processes which **communicate** via shared/joint actions
- Actions
  - joint - a fixed set of processes/NFAs participate.
  - local - only one process participates.
- Initial (global) state

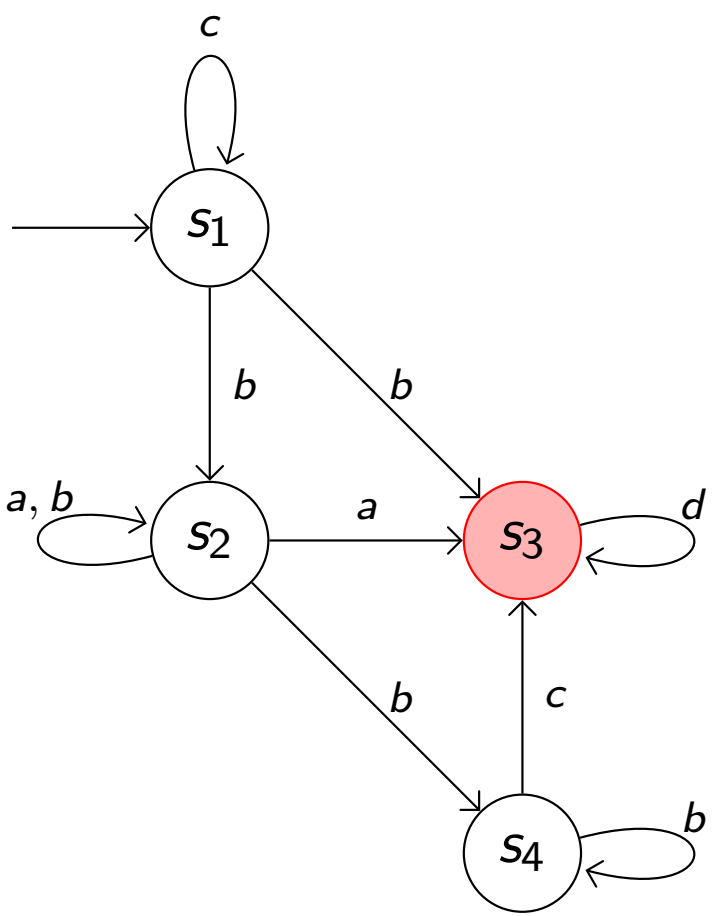
Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

An ATS game

- Distributed system is a **team** of processes vs Environment
- Played on NFAs owned by the each processes which **communicate** via shared/joint actions
- Actions
  - joint - a fixed set of processes/NFAs participate.
  - local - only one process participates.
- Initial (global) state
- A winning condition

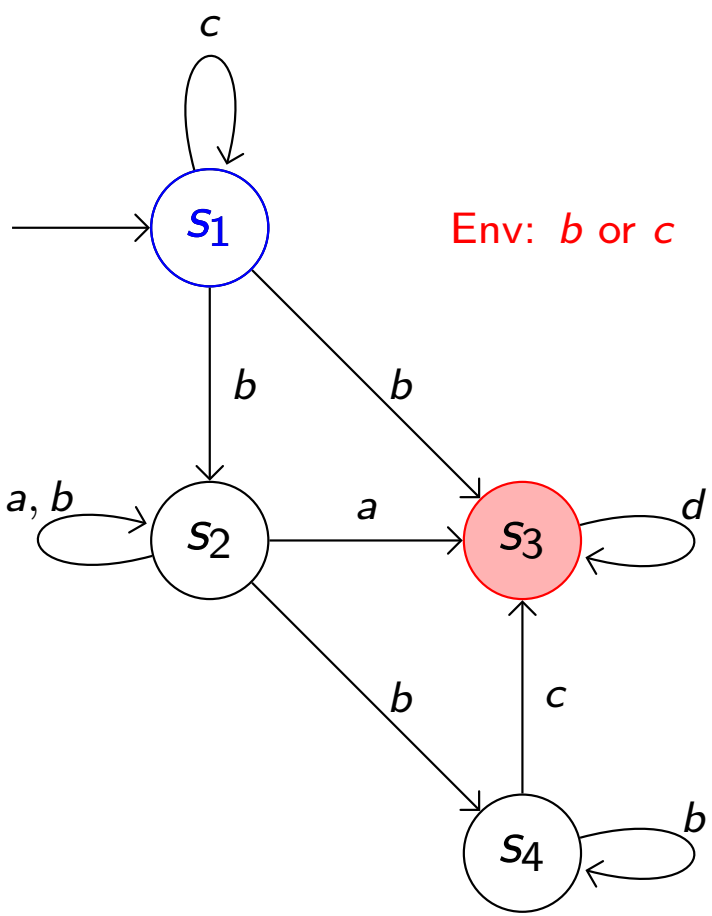
Does there exist a "distributed strategy" or advice function for the NFA components to win no matter what the environment moves are?

Safety Game: Played on a single NFA



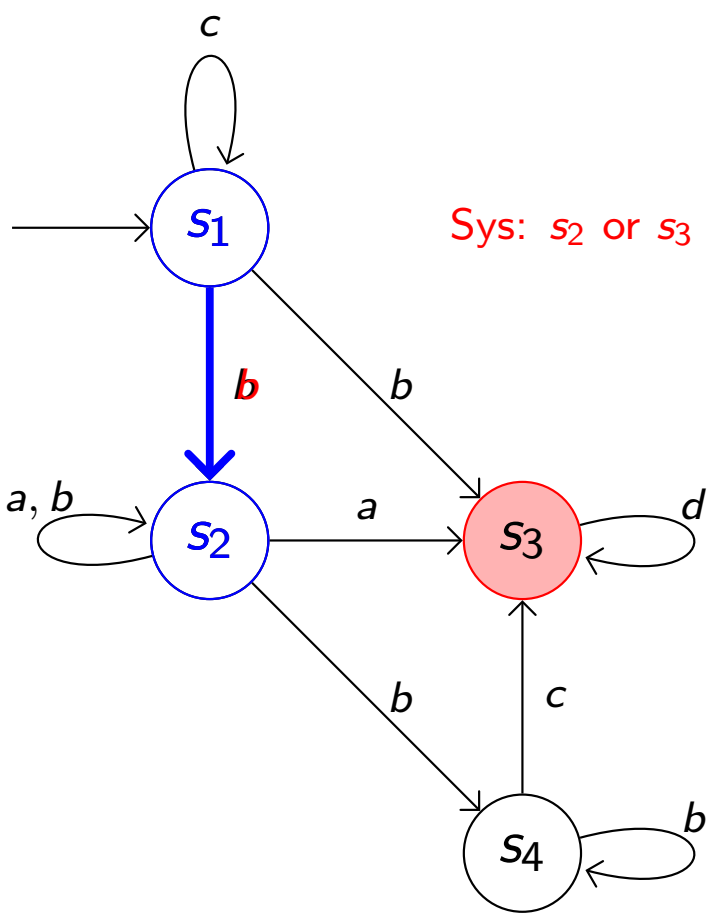
- Play = Run from initial state

Safety Game: Played on a single NFA



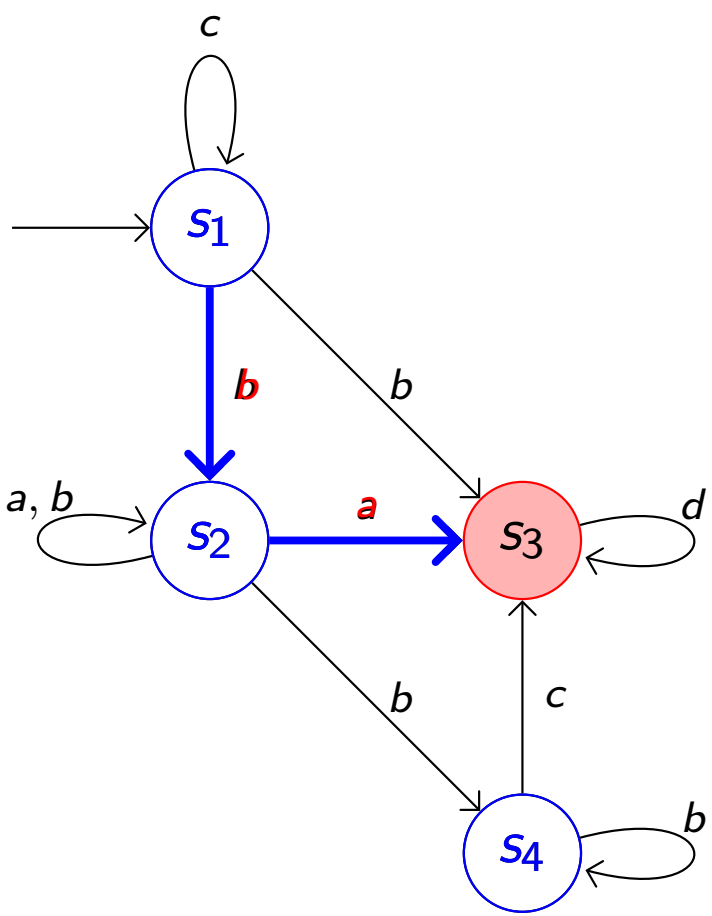
- Play = Run from initial state

Safety Game: Played on a single NFA



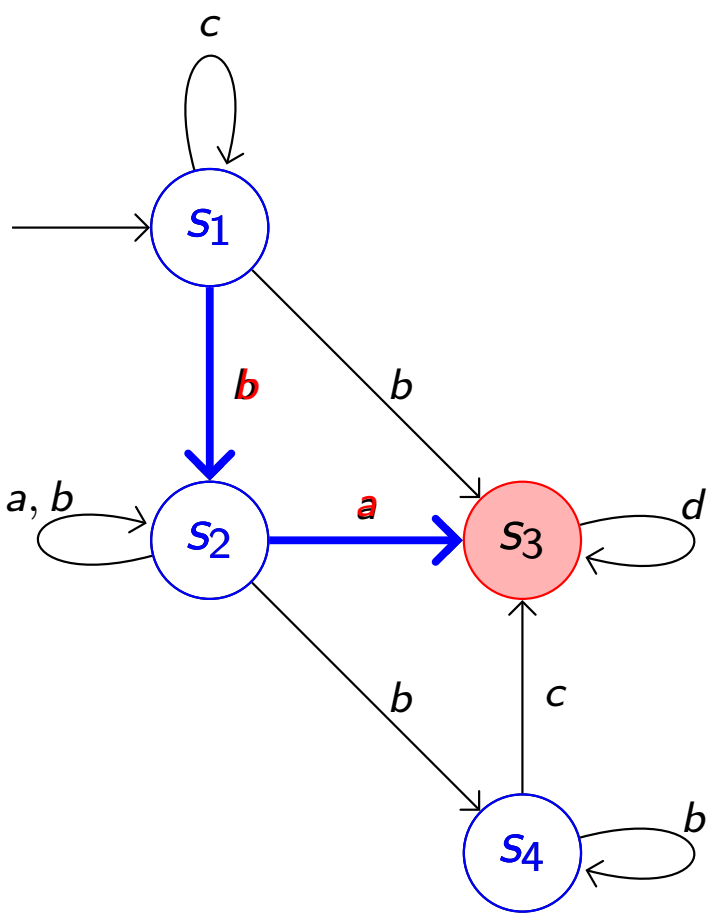
- Play = Run from initial state

Safety Game: Played on a single NFA



- Play = Run from initial state
- Unsafe State = System looses

Safety Game: Played on a single NFA



- Play = Run from initial state
- Unsafe State = System looses

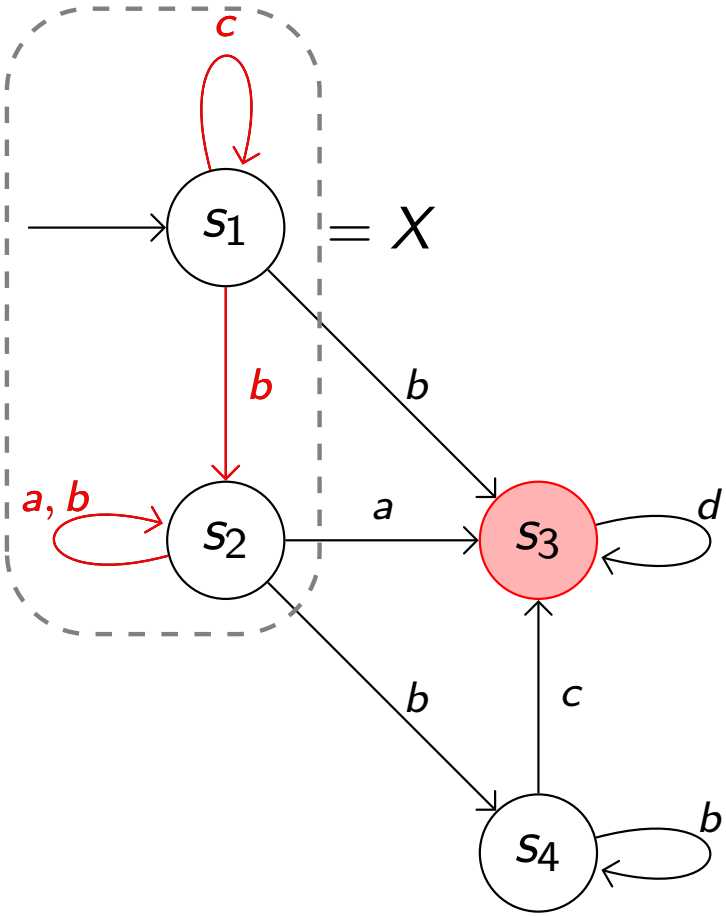
Objective

Given a safety game, find an advice function for the system such that all resulting plays always remain in safe set.

- Single process games boil down to standard two player graph games with **zero memory, polytime, fixpoint** solutions.

Key element: Trap- System traps the environment in X

Safety Game: Played on a single NFA



- Play = Run from initial state
- Unsafe State = System looses

Objective

Given a safety game, find an advice function for the system such that all resulting plays always remain in safe set.

- Single process games boil down to standard two player graph games with **zero memory, polytime, fixpoint** solutions.

Key element: Trap- System traps the environment in X

Two process ATS games

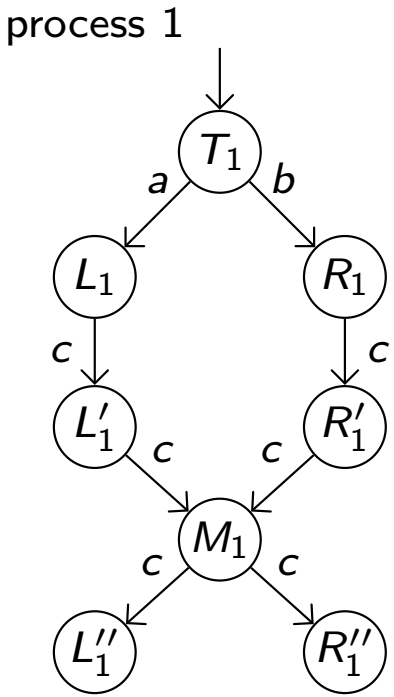
2 process Safety Game - An Example

- System consists of team of processes 1 and 2

2 process Safety Game - An Example

- System consists of team of processes 1 and 2
- Played on 2 NFAs

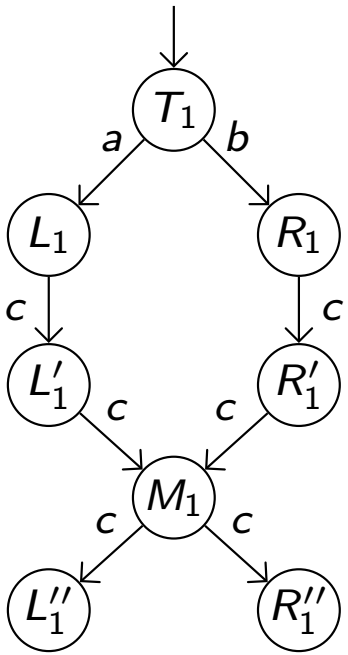
2 process Safety Game - An Example



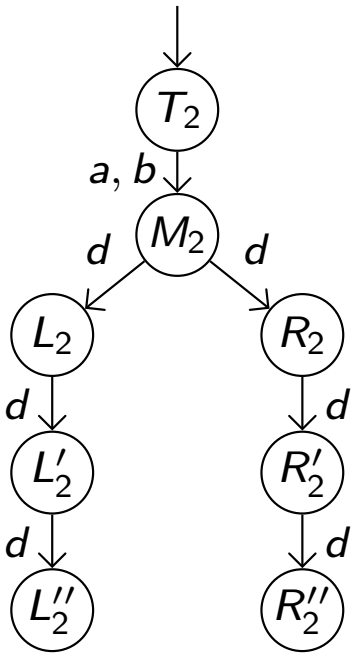
- System consists of team of processes 1 and 2
- Played on 2 NFAs

2 process Safety Game - An Example

process 1

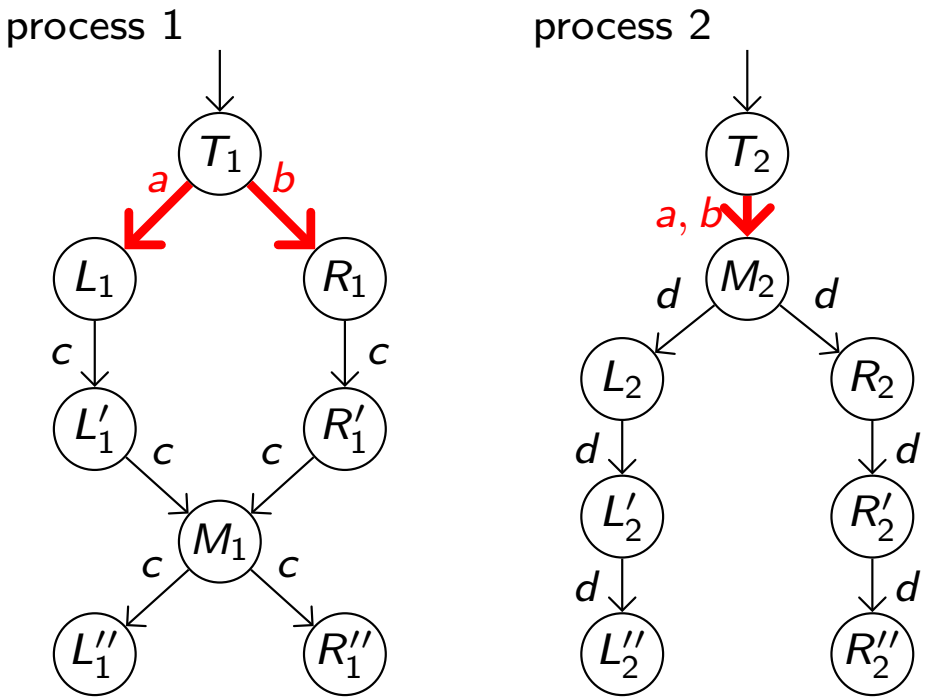


process 2



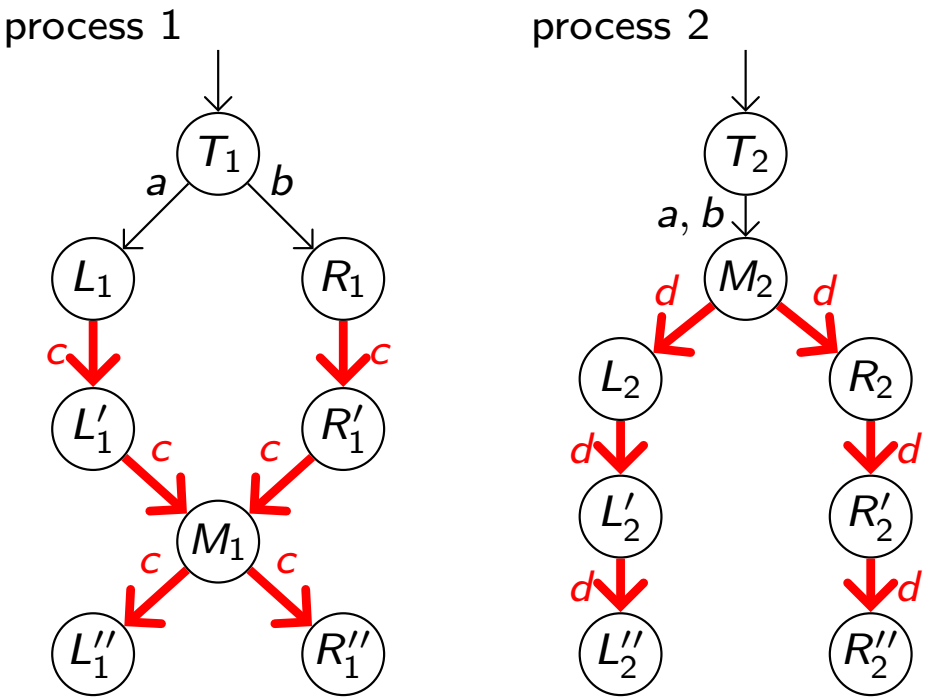
- System consists of team of processes 1 and 2
- Played on 2 NFAs

2 process Safety Game - An Example



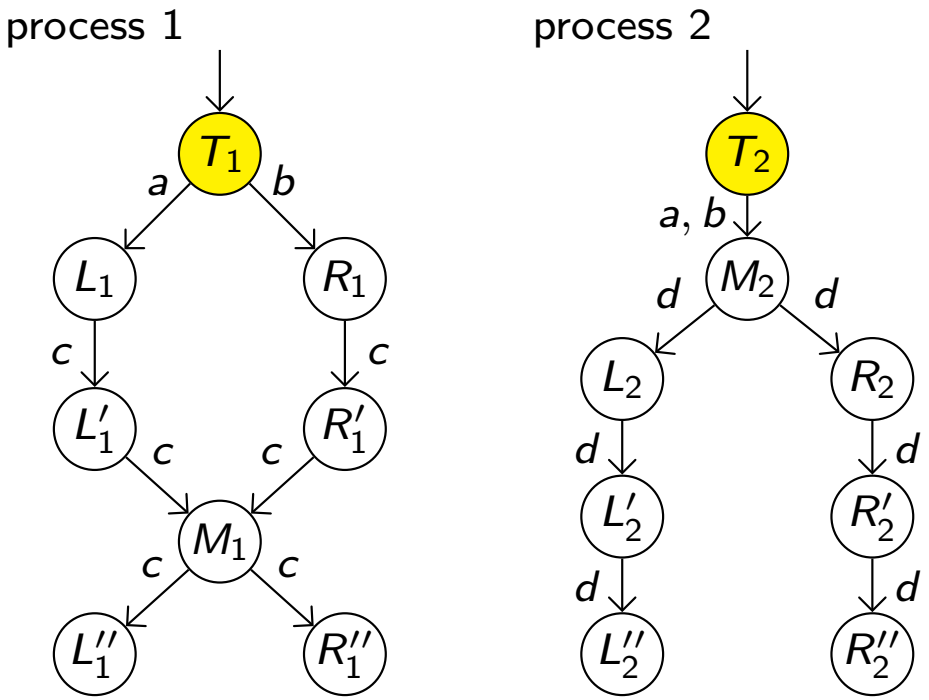
- System consists of team of processes 1 and 2
- Played on 2 NFAs
- Actions
  - **Joint**: a, b

2 process Safety Game - An Example



- System consists of team of processes 1 and 2
- Played on 2 NFAs
- Actions
  - **Joint**: a, b
  - **Local**: c (P1), d (P2)

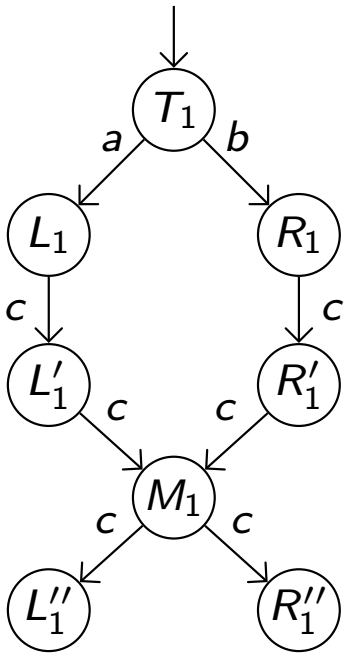
2 process Safety Game - An Example



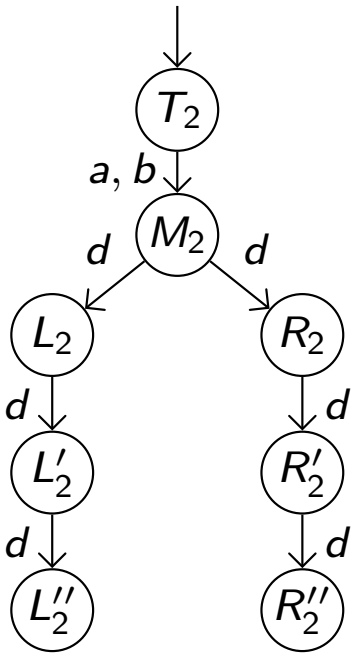
- System consists of team of processes 1 and 2
- Played on 2 NFAs
- Actions
  - **Joint**: a, b
  - **Local**: c (P1), d (P2)
- Initial state: (T_1, T_2)

2 process Safety Game - An Example

process 1

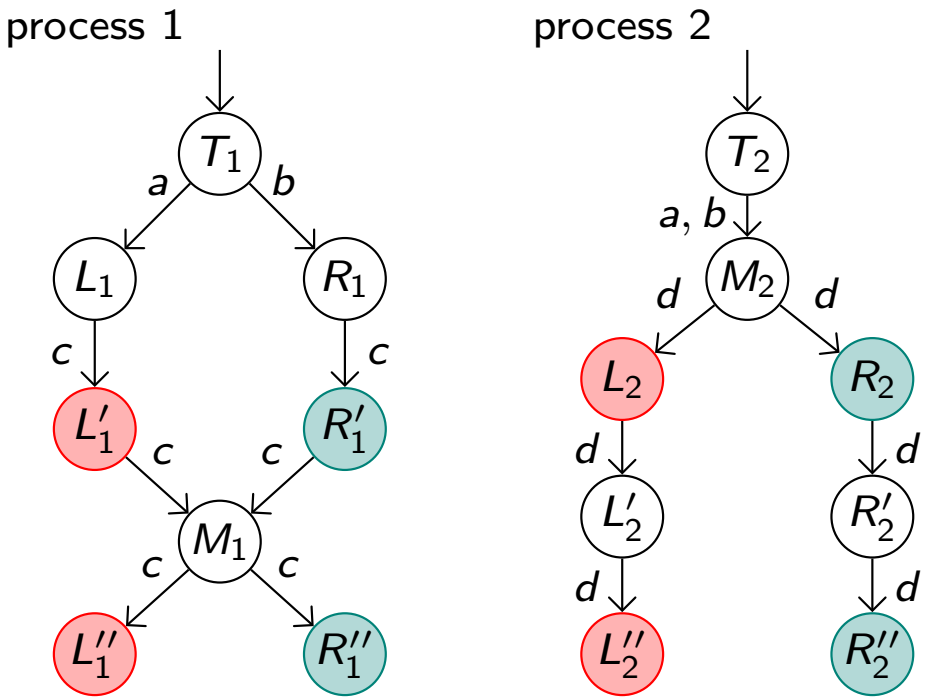


process 2



- System consists of team of processes 1 and 2
- Played on 2 NFAs
- Actions
  - **Joint**: a, b
  - **Local**: c (P1), d (P2)
- Initial state: (T_1, T_2)
- Causal memory / past

2 process Safety Game - An Example

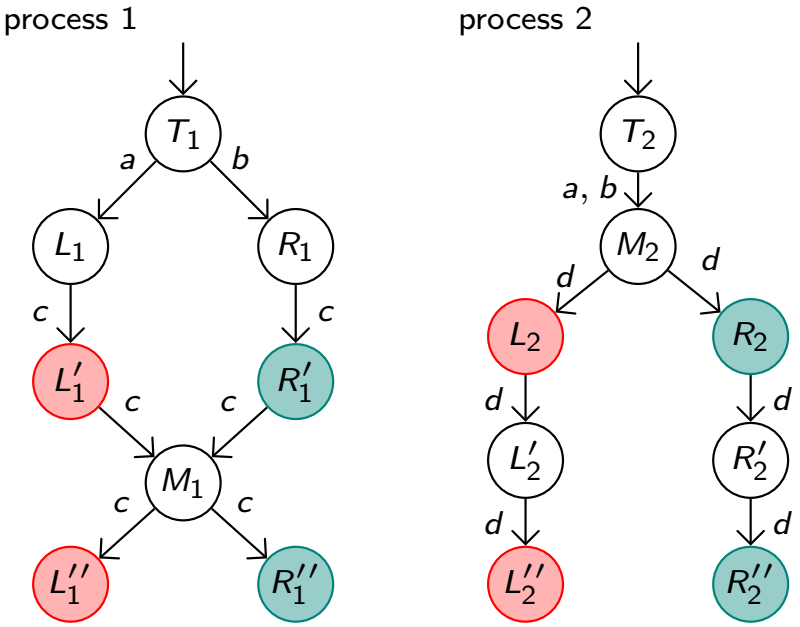


- System consists of team of processes 1 and 2
- Played on 2 NFAs
- Actions
  - **Joint**: a, b
  - **Local**: c (P1), d (P2)
- Initial state: (T_1, T_2)
- Causal memory / past
- Unsafe: $L \times R$ or $R \times L$

Does there exist a "distributed strategy" or advice function for the two NFA components 1 and 2 to avoid $L \times R$ or $R \times L$ at the same time?

Avoid $\{L'_1, L''_1\} \times \{R_2, R''_2\} \cup \{R'_1, R''_1\} \times \{L_2, L''_2\}$ concurrently.

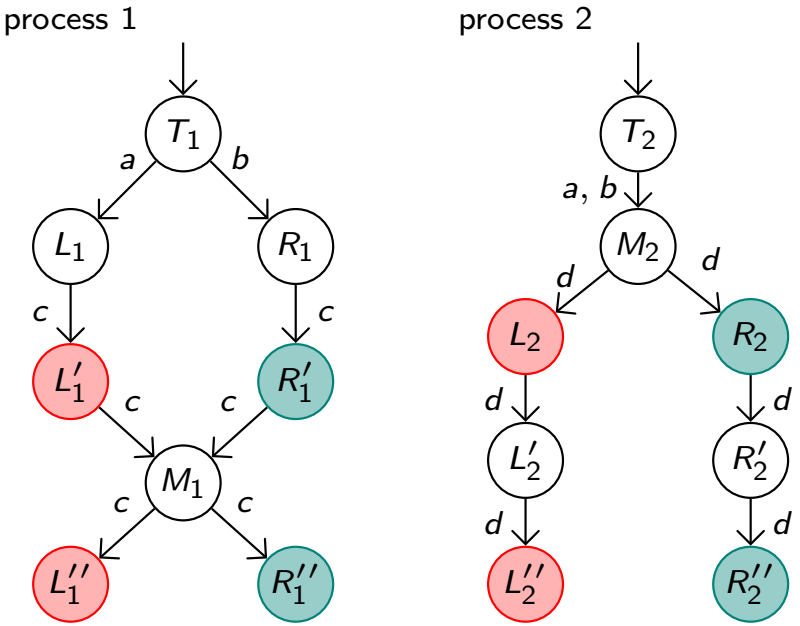
2 process Safety Game- Play dynamics

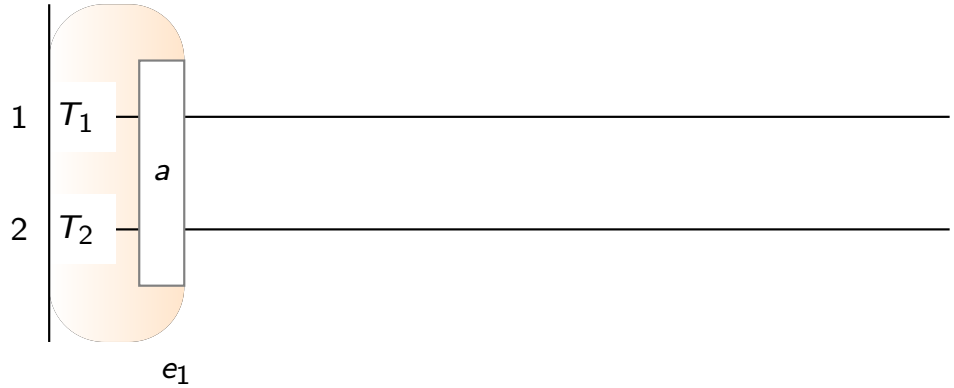


Play begins in the top states

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

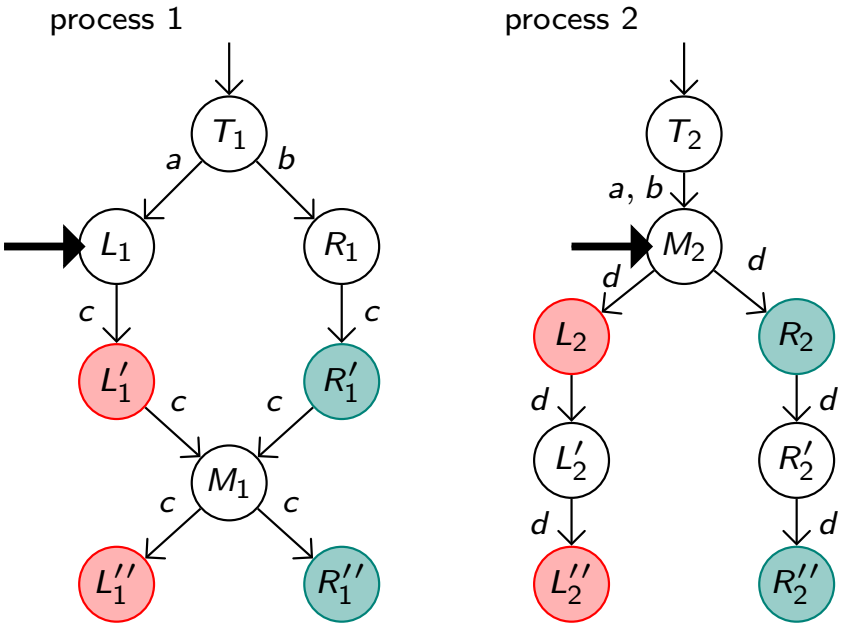


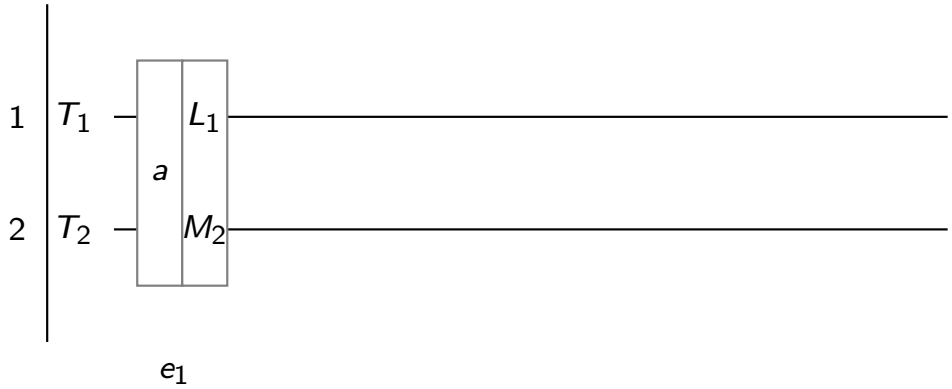


Environment can choose between a or b.
Here it schedules an a

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

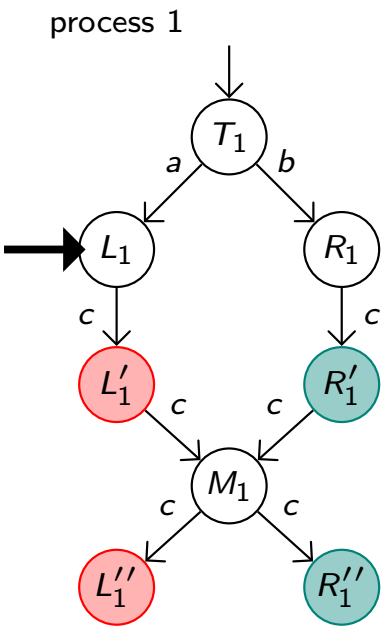


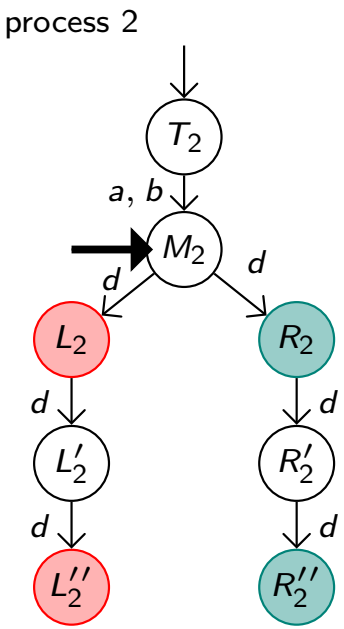


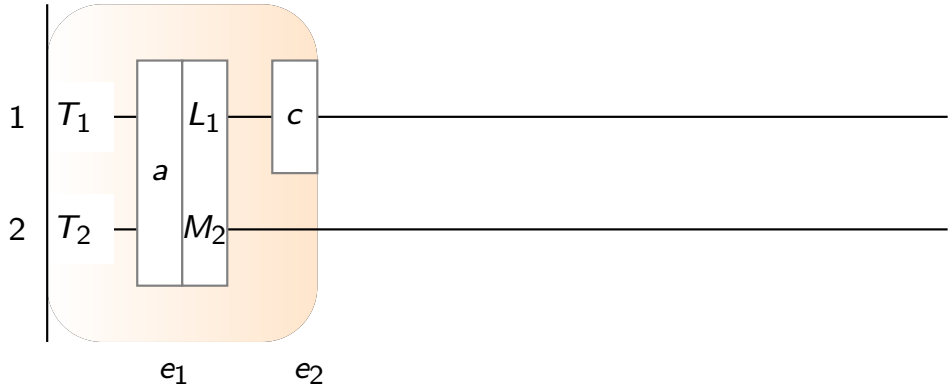
Process 1 and 2 move to their left and middle states

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics



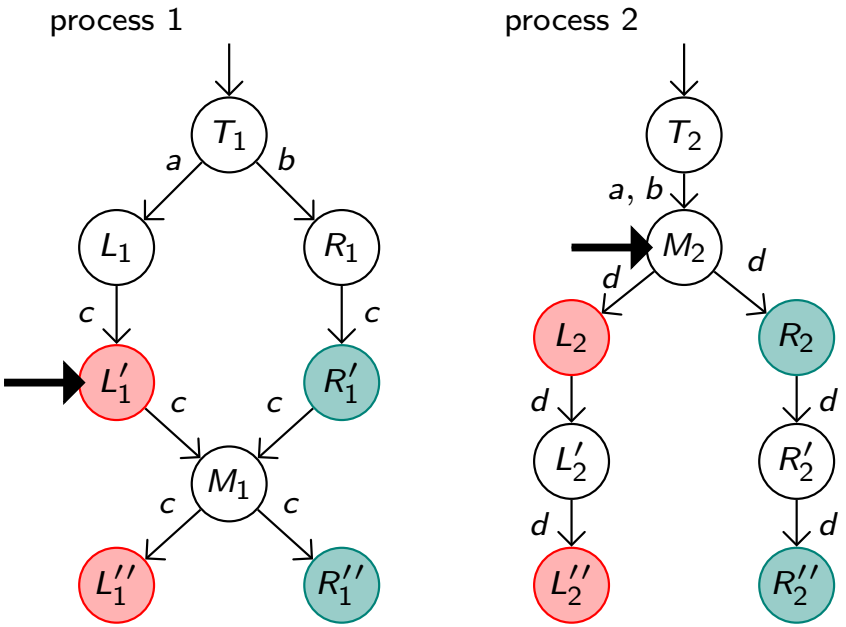


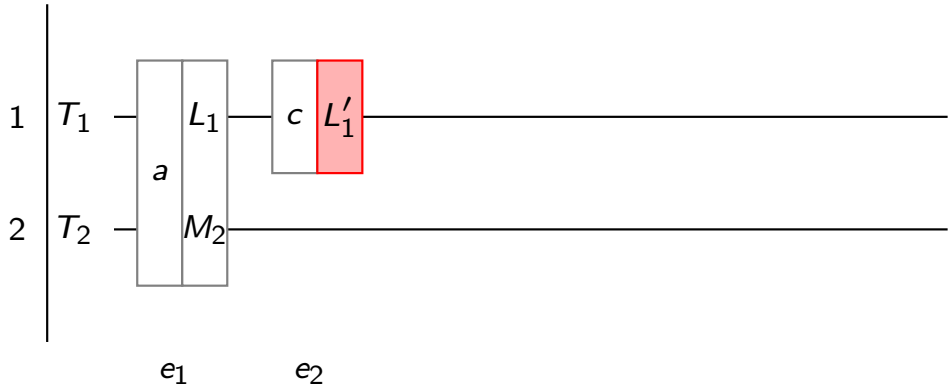


Environment schedules a c

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

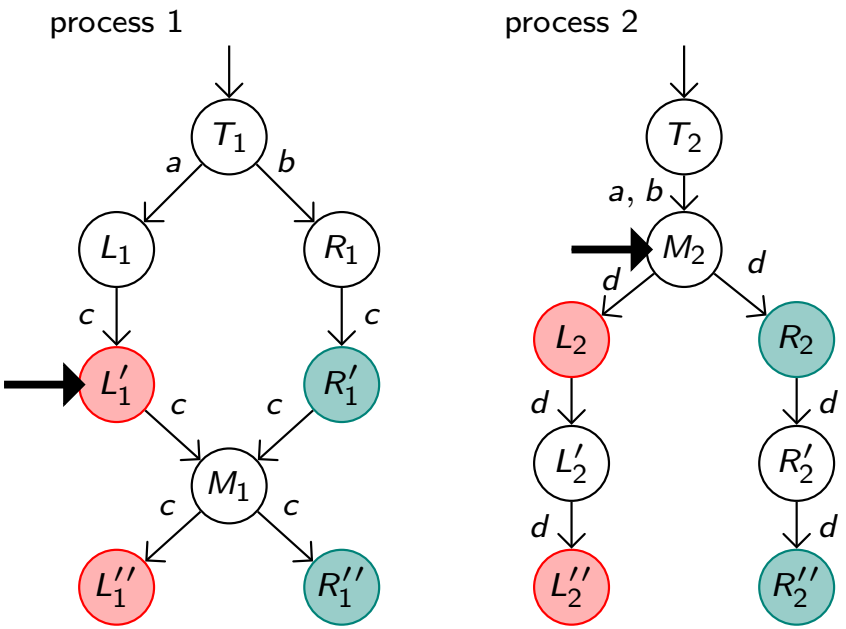


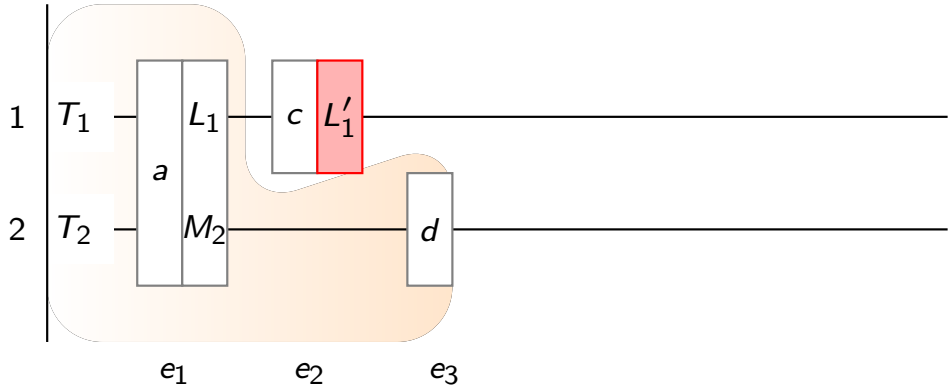


Process 1 moves to the next left state

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

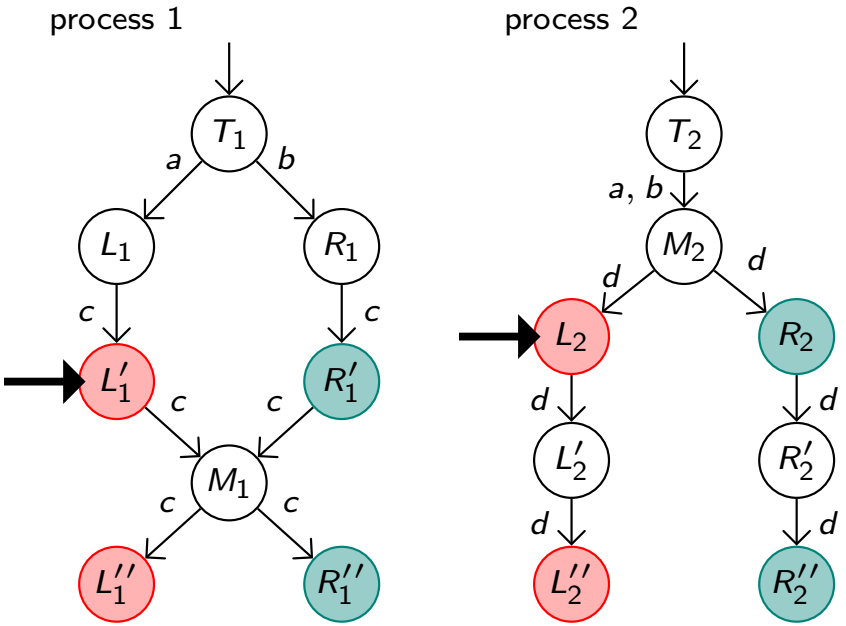


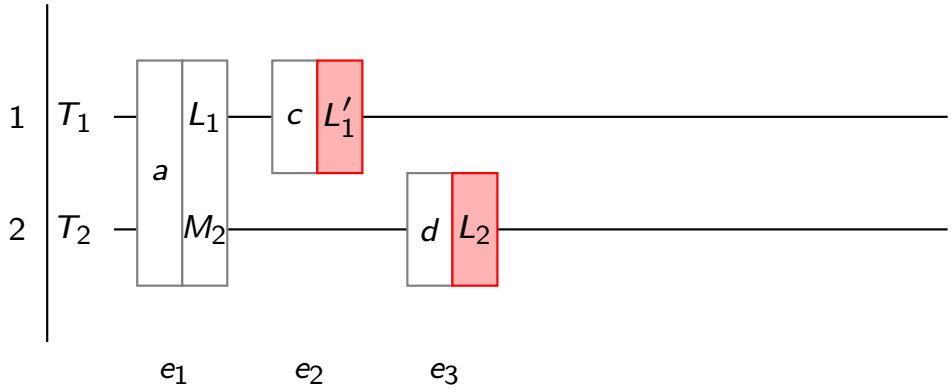


Environment schedules a d
System can chooses between left or right based on its causal past- highlighted in orange

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

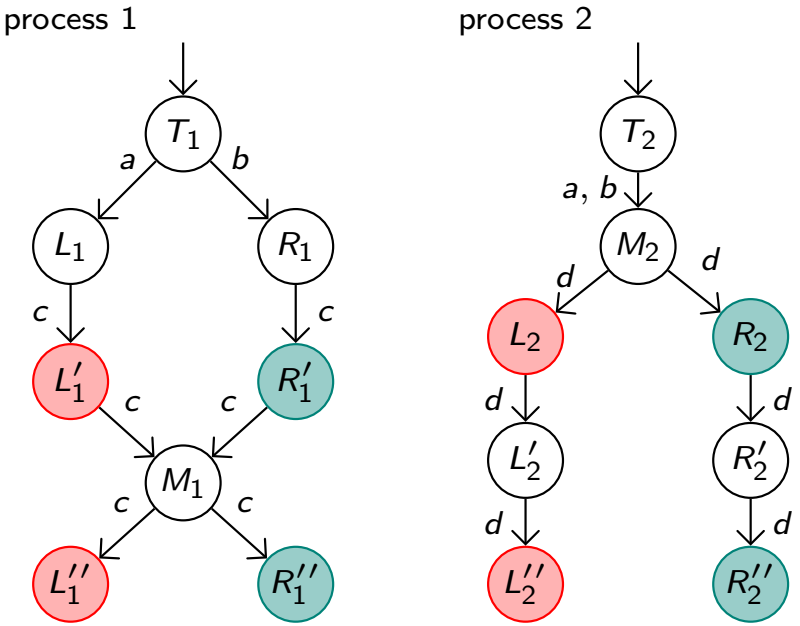


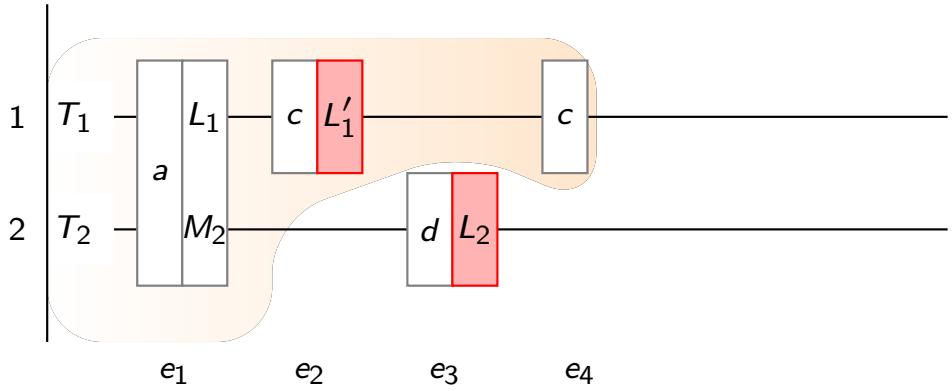


Process 2 chooses to go to left state

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

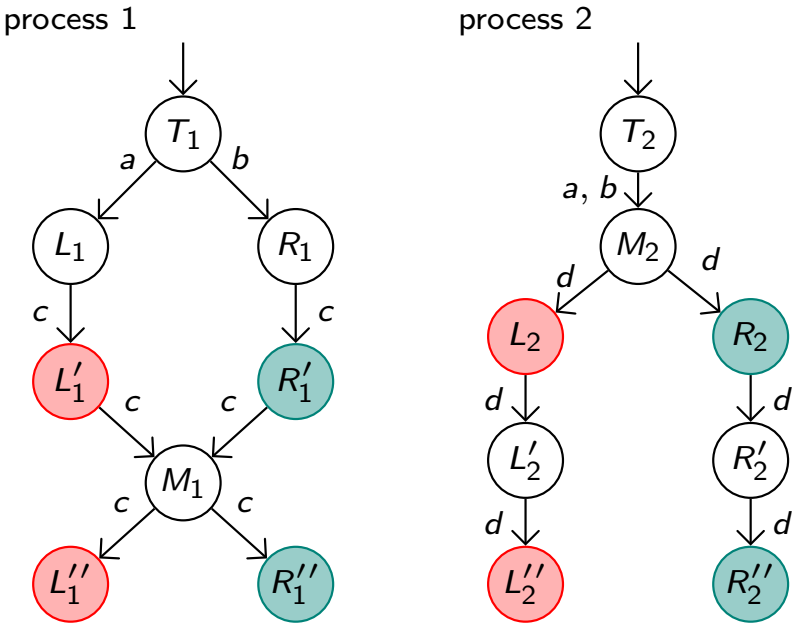


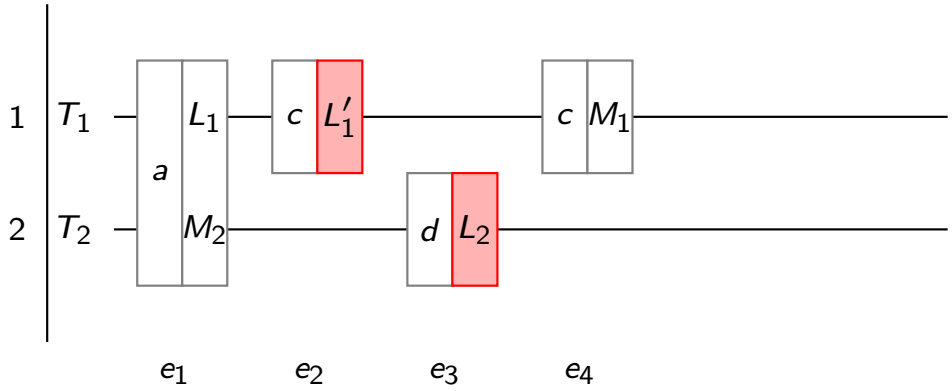


Environment schedules a c

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

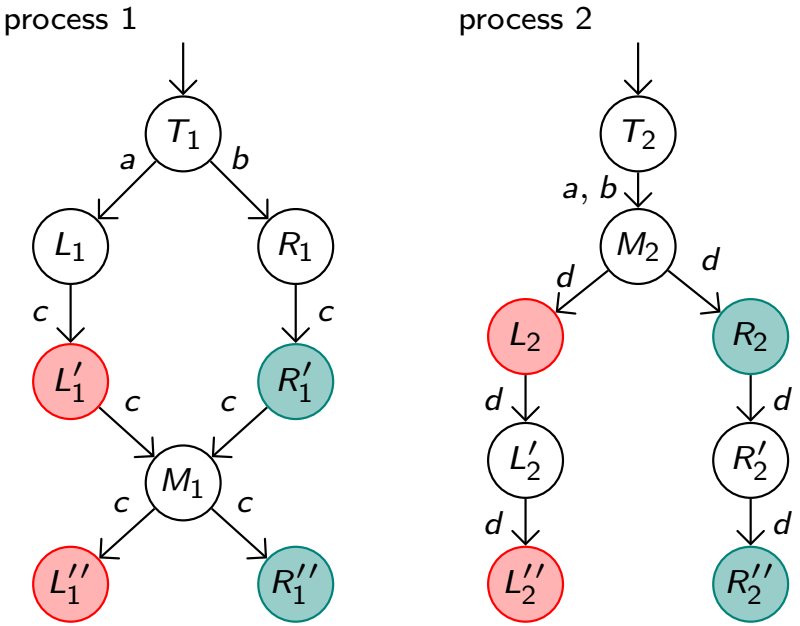


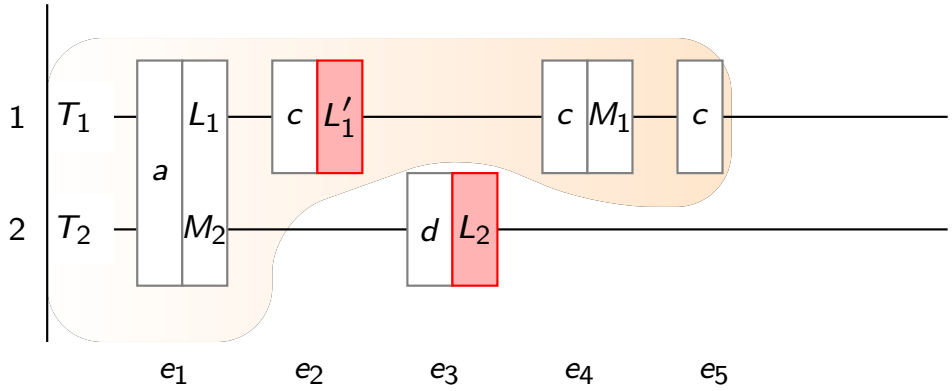


Process 1 moves to the middle state

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

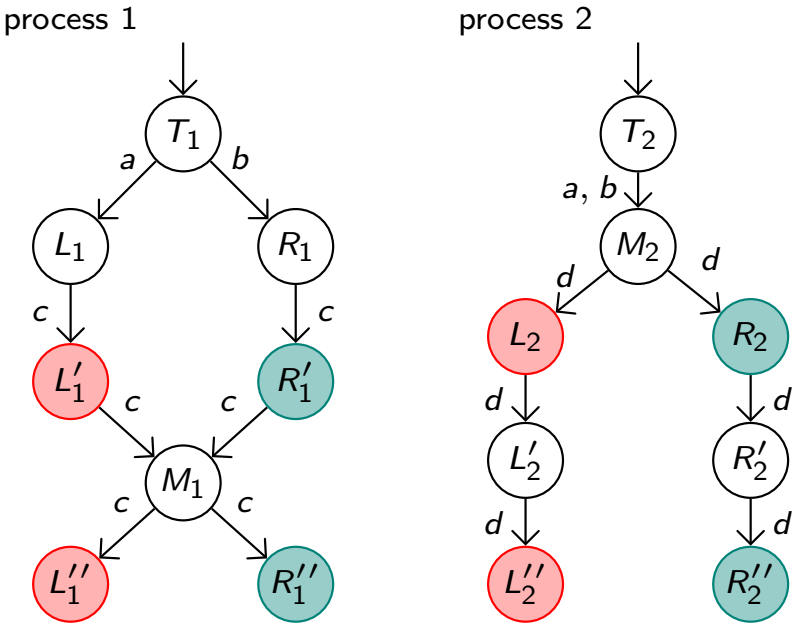


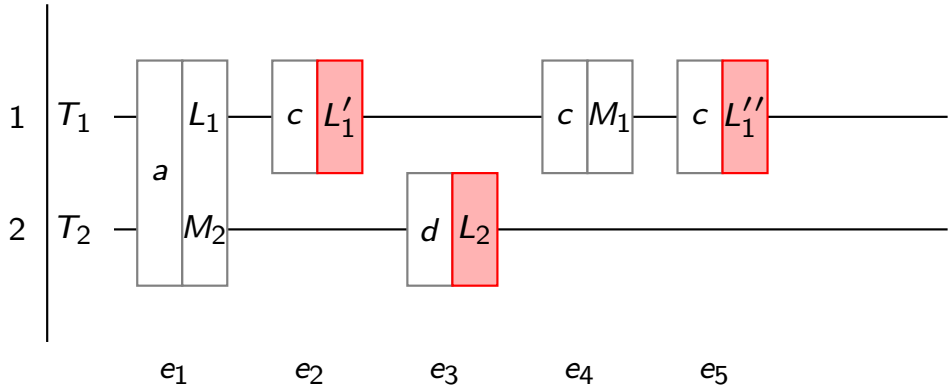


Environment schedules a c

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

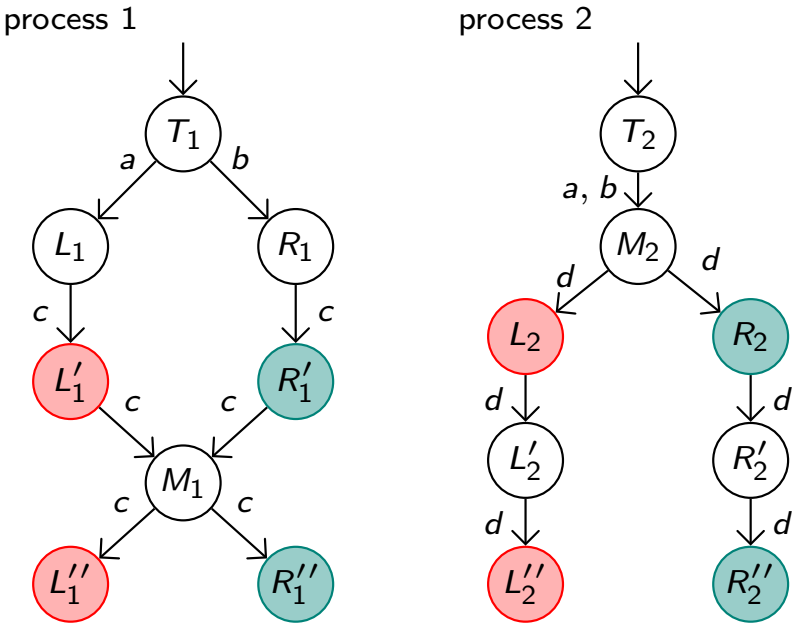


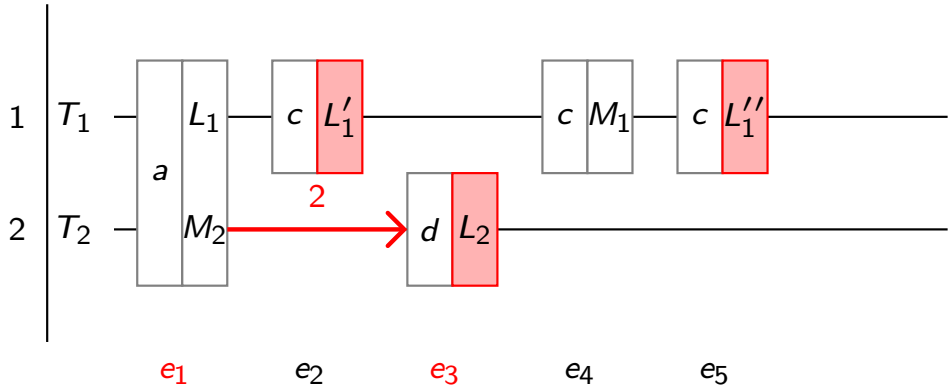


Based on its causal past
Process 1 chooses to go left again

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

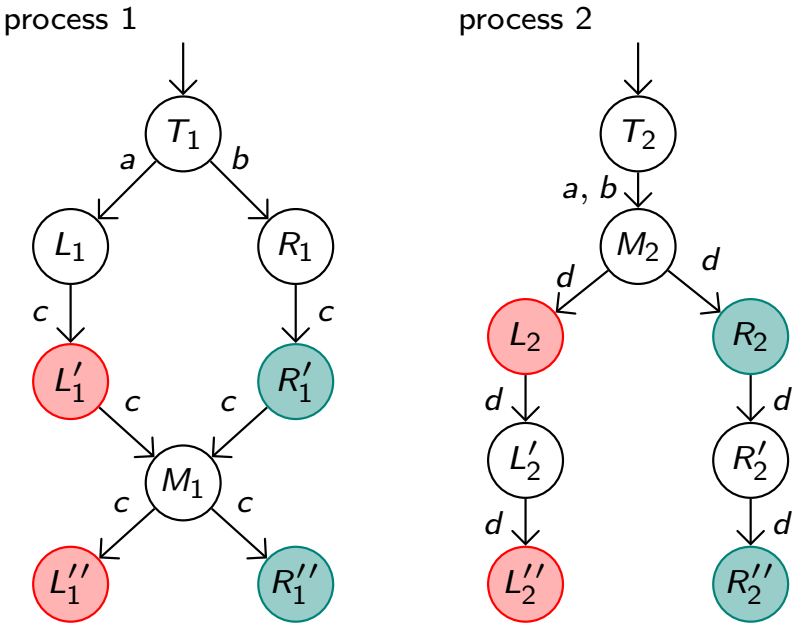


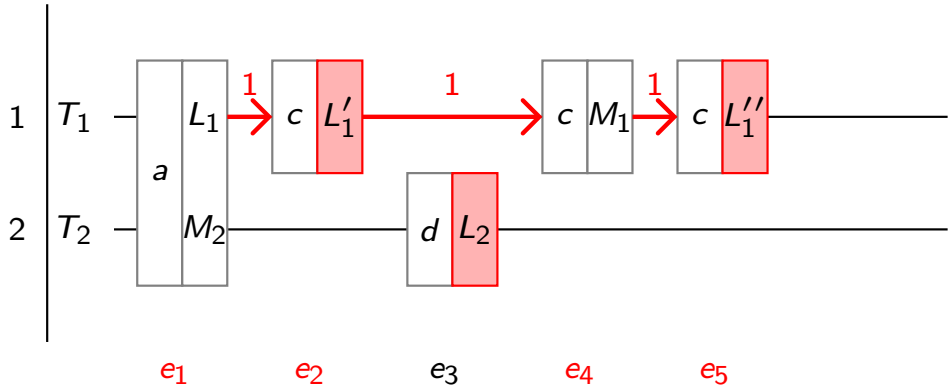


Causal past of e_3 contains e_1

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics

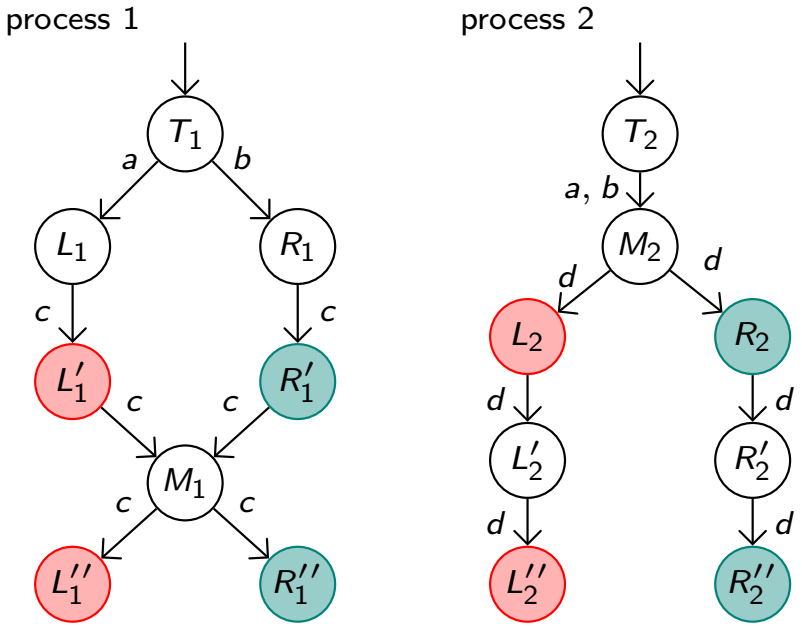


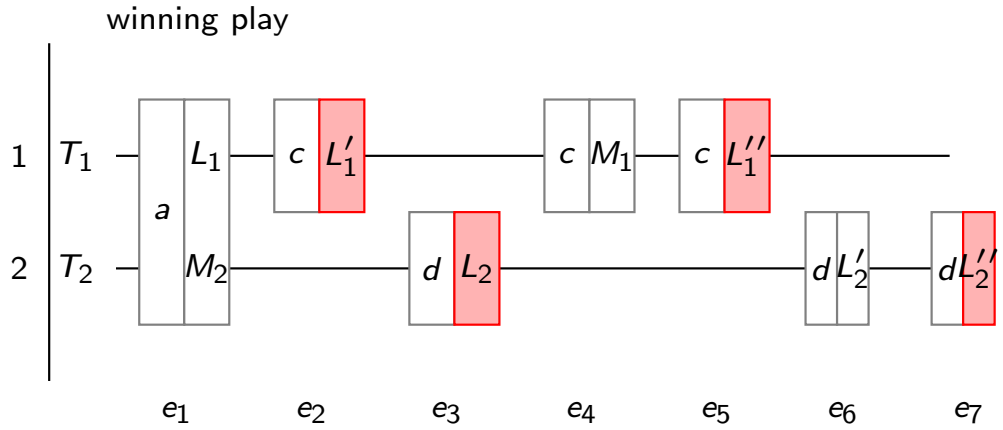


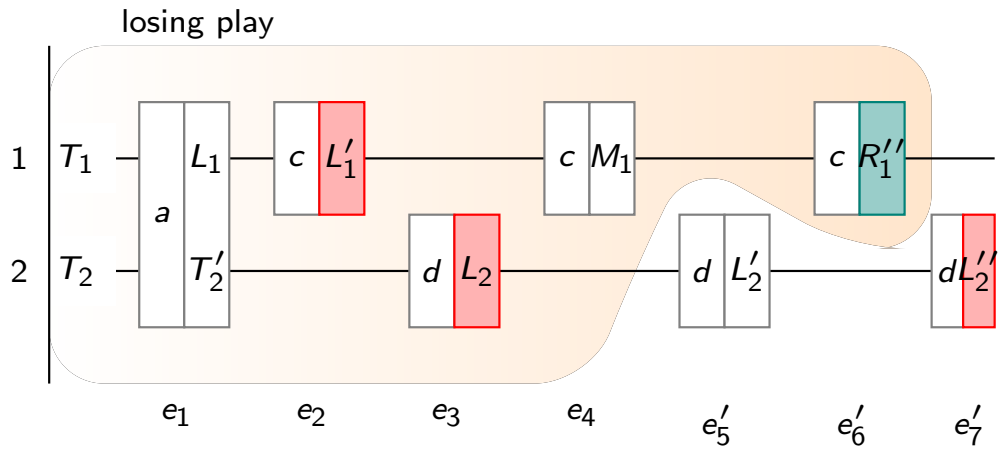
All other events are linearly ordered by process 1

Example of a play from initial state (T_1, T_2)

2 process Safety Game- Play dynamics







Example of a play from initial state (T_1, T_2)

From Single-Process Games to Distributed ATS Games

Single-process safety games

- **Players**: Environment vs one agent system
- Equivalent to classical **turn-based graph games**
- **Play**: linear sequence of states (a run)
- **Information**: Perfect information

Distributed ATS games (multi-process)

- **Players**: Multiple distributed processes vs environment
- Not turn-based: actions may occur **concurrently**
- **Play**: partial-ordered events (Mazurkiewicz trace)
- **Information**: **Partial** — each process sees only its causal past

From Single-Process Games to Distributed ATS Games

Single-process safety games

- **Players**: Environment vs one agent system
- Equivalent to classical **turn-based graph games**
- **Play**: linear sequence of states (a run)
- **Information**: Perfect information
- Strategy depends on the **entire global history**
- **Memoryless (zero-memory) winning strategies suffice** for safety and reachability

Distributed ATS games (multi-process)

- **Players**: Multiple distributed processes vs environment
- Not turn-based: actions may occur **concurrently**
- **Play**: partial-ordered events (Mazurkiewicz trace)
- **Information**: **Partial** — each process sees only its causal past
- Strategies are **distributed**: each process decides locally
- In general, **memory is required**;

A winning distributed strategy

A distributed strategy is an advice function for the processes to decide matching transitions.

- On local actions, the local process can use its entire **causal past**
- On joint actions, participating processes exchange causal pasts and use this *entire* shared information.

A (finite-state) memory-based distributed strategy

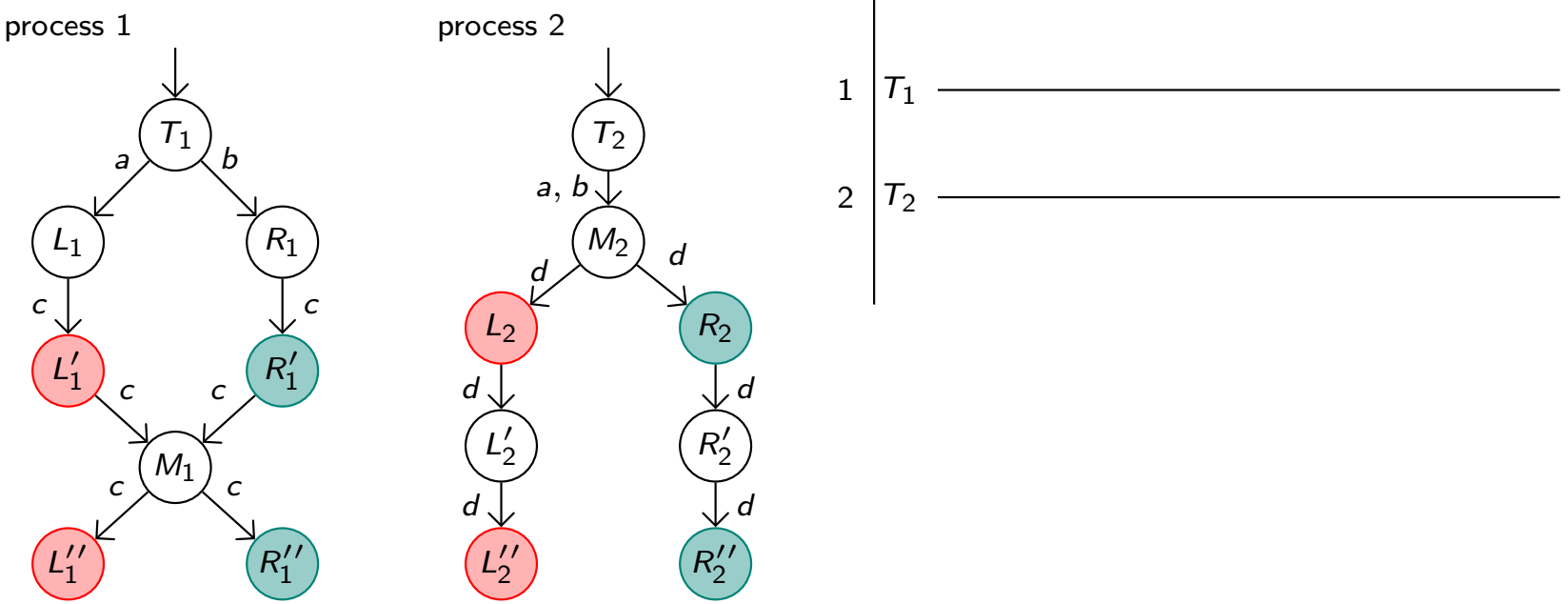
- maintains and updates the relevant key past *distributed* information
- the decisions are completely based on this distributed memory.

Problem Statement

Given an ATS game, decide if there exists a winning distributed strategy. Further, construct a memory-based winning strategy if one exists.

A winning memory strategy: an example

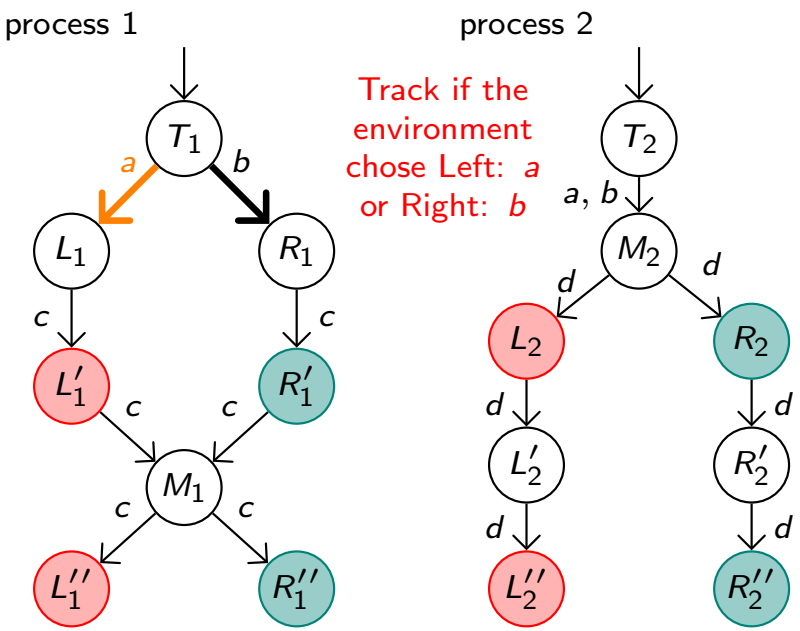
A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.

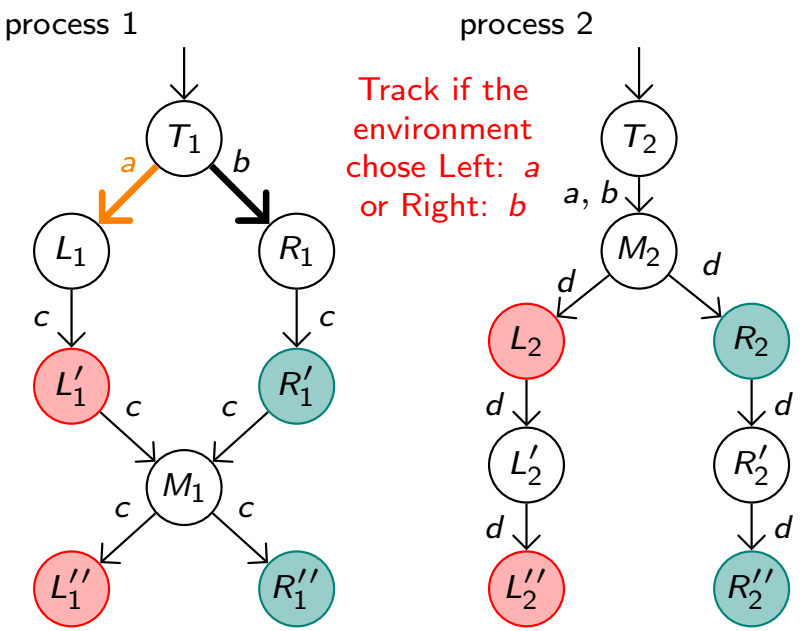


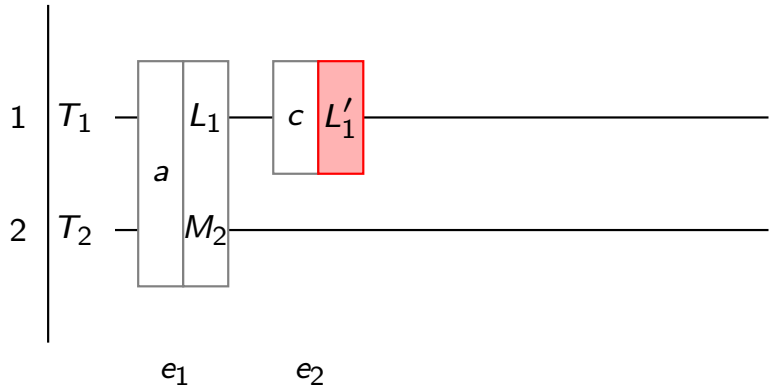
Both processes remember if the environment chose Left: a or Right: b

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



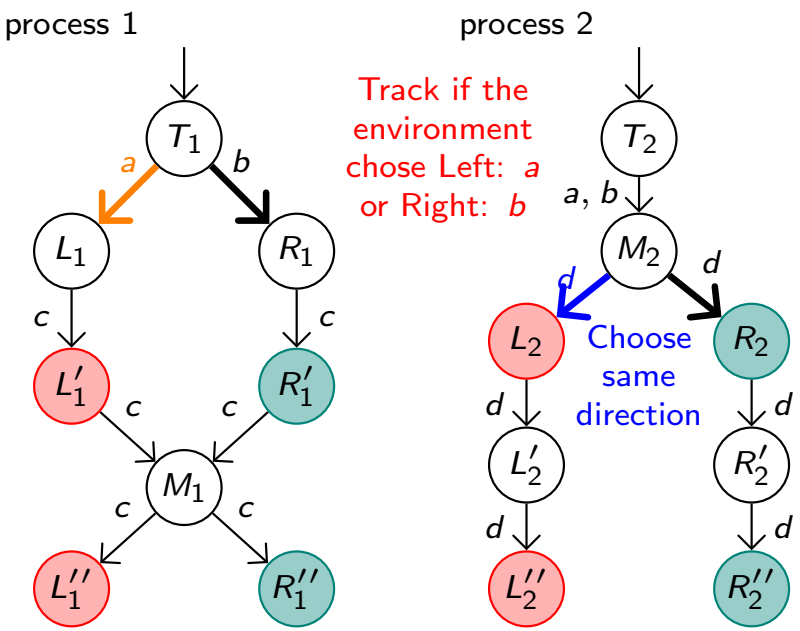


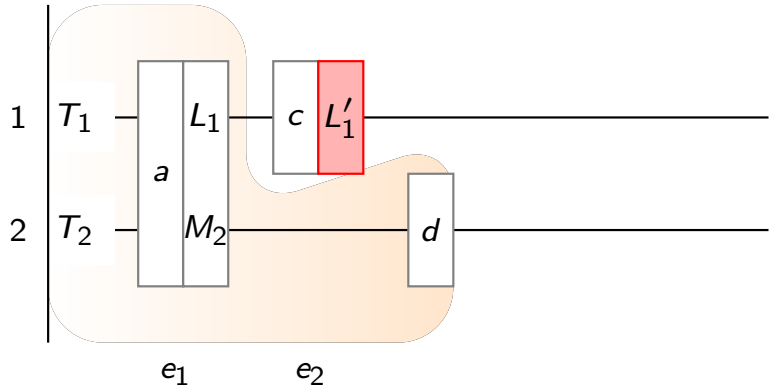
Both processes remember if the environment chose Left: a or Right: b

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



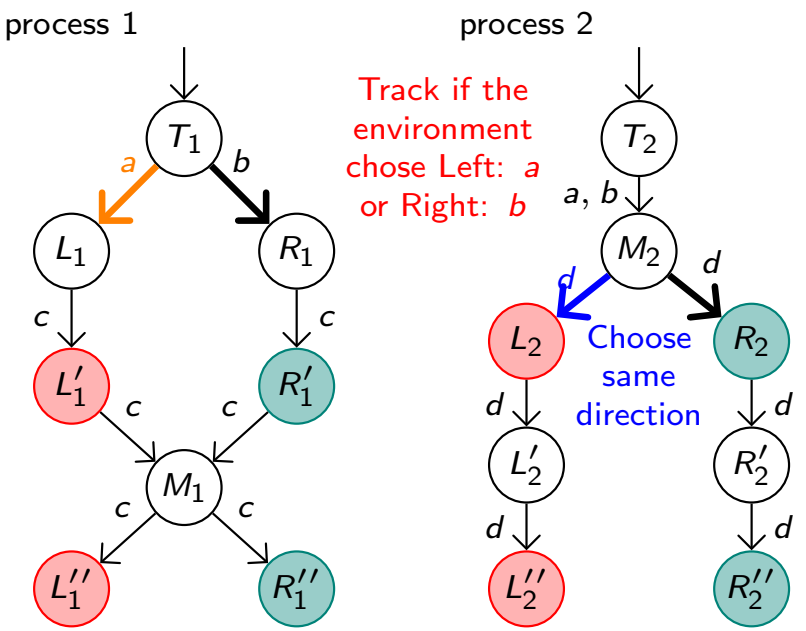


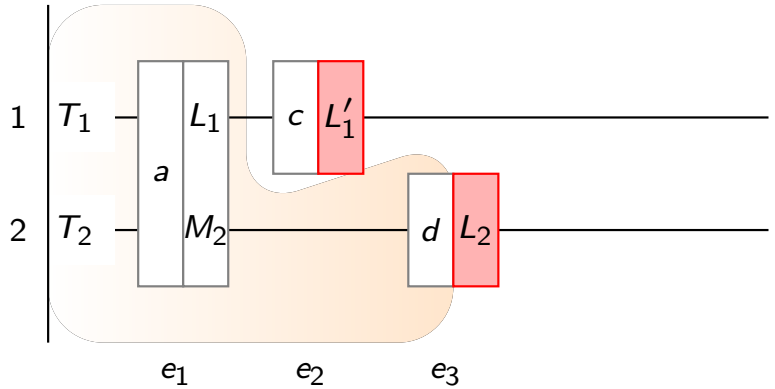
Both processes remember if the environment chose Left: a or Right: b

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



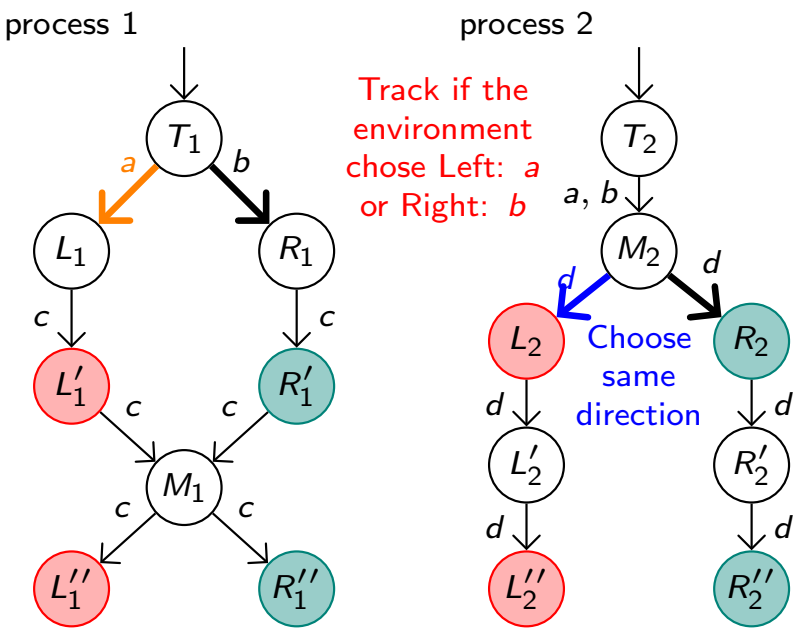


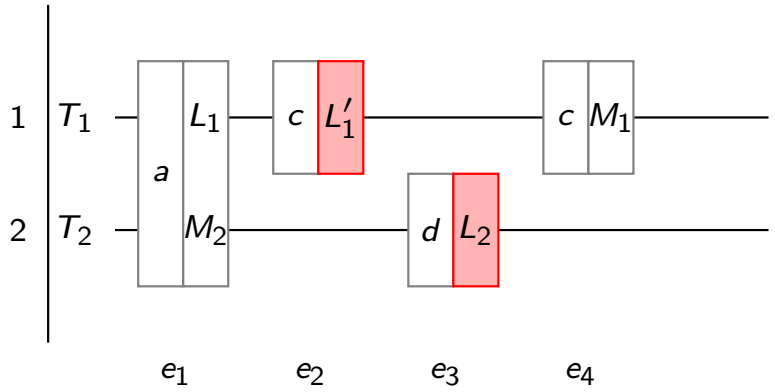
Both processes remember if the environment chose Left: a or Right: b
Process 2 goes left if the environment chose a

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



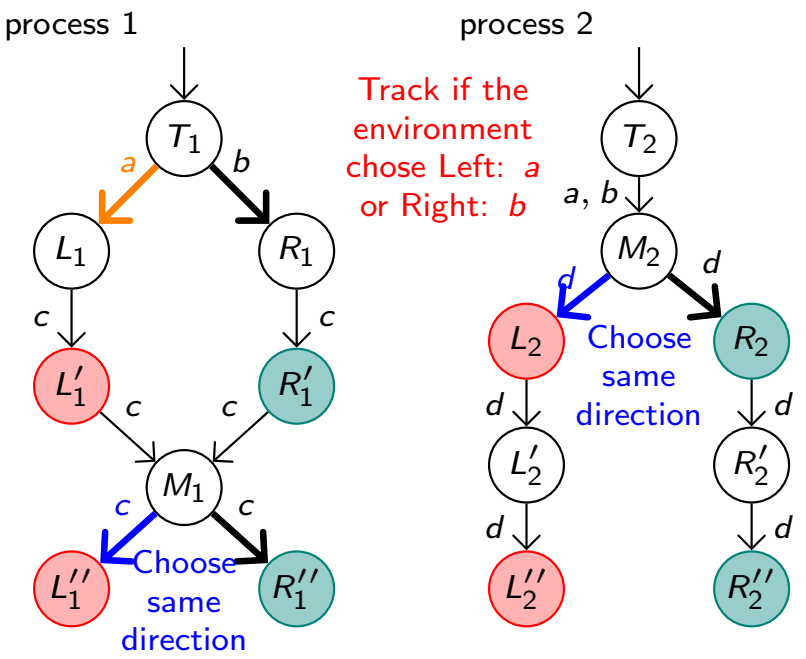


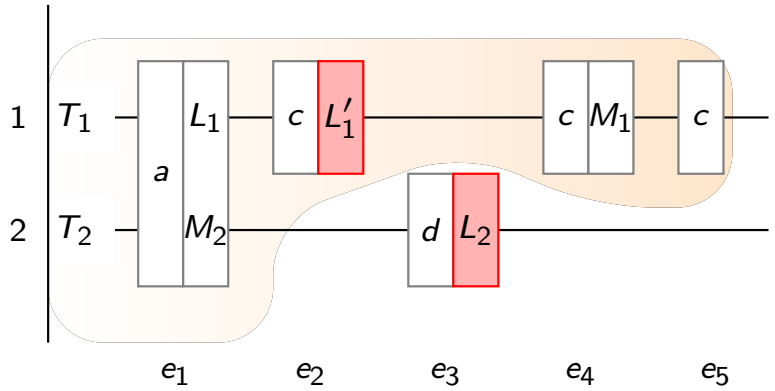
Both processes remember if the environment chose Left: a or Right: b
Process 2 goes left if the environment chose a

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



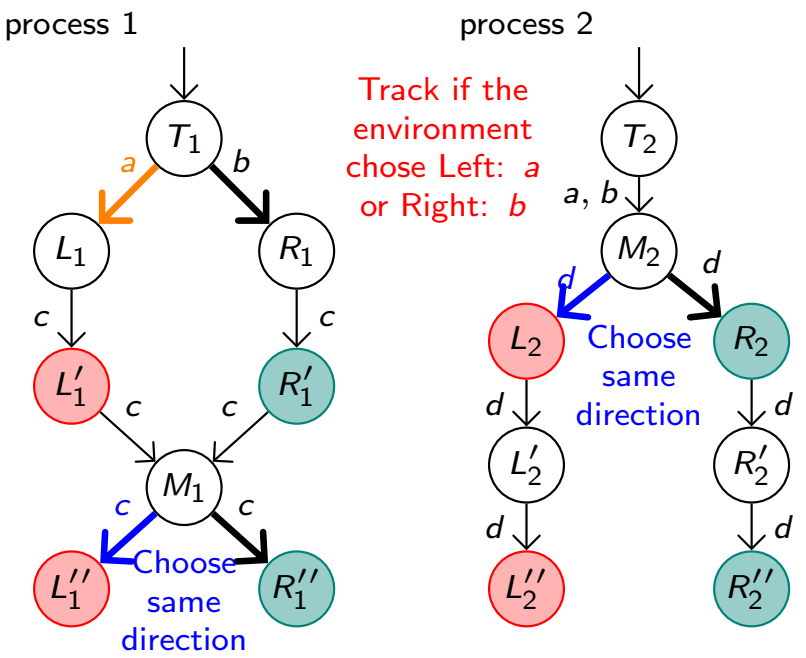


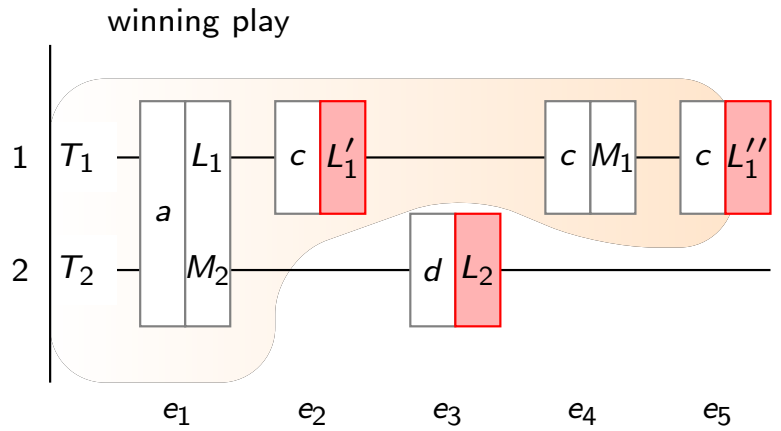
Both processes remember if the environment chose Left: a or Right: b
Process 2 goes left if the environment chose a

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.



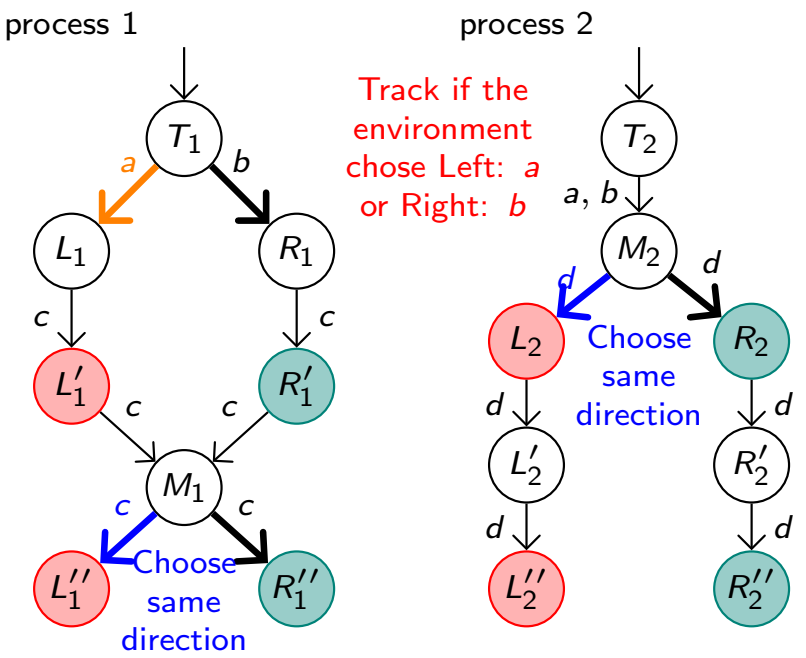


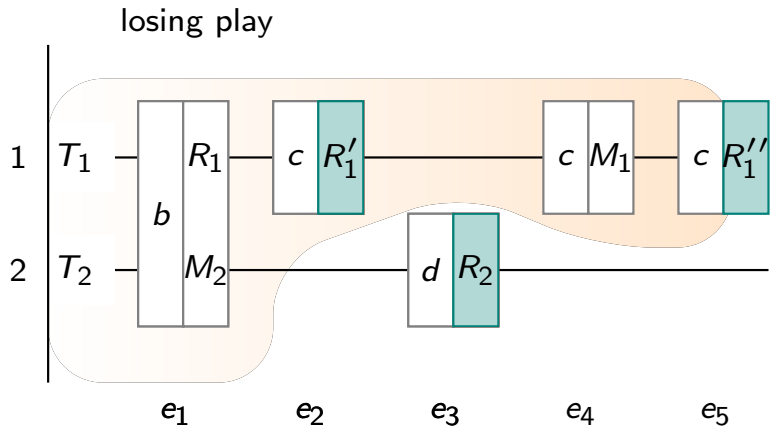
Both processes remember if the environment chose Left: a or Right: b
Process 2 goes left if the environment chose a
Process 1 goes left if the environment chose a instead of right

The processes must remember information exchanged during synchronization.

A winning memory strategy: an example

A strategy is an advice function for the processes that tells them their next moves based on information gathered from their causal past.





Had the environment chosen right, so would have both processes

The processes must remember information exchanged during synchronization.

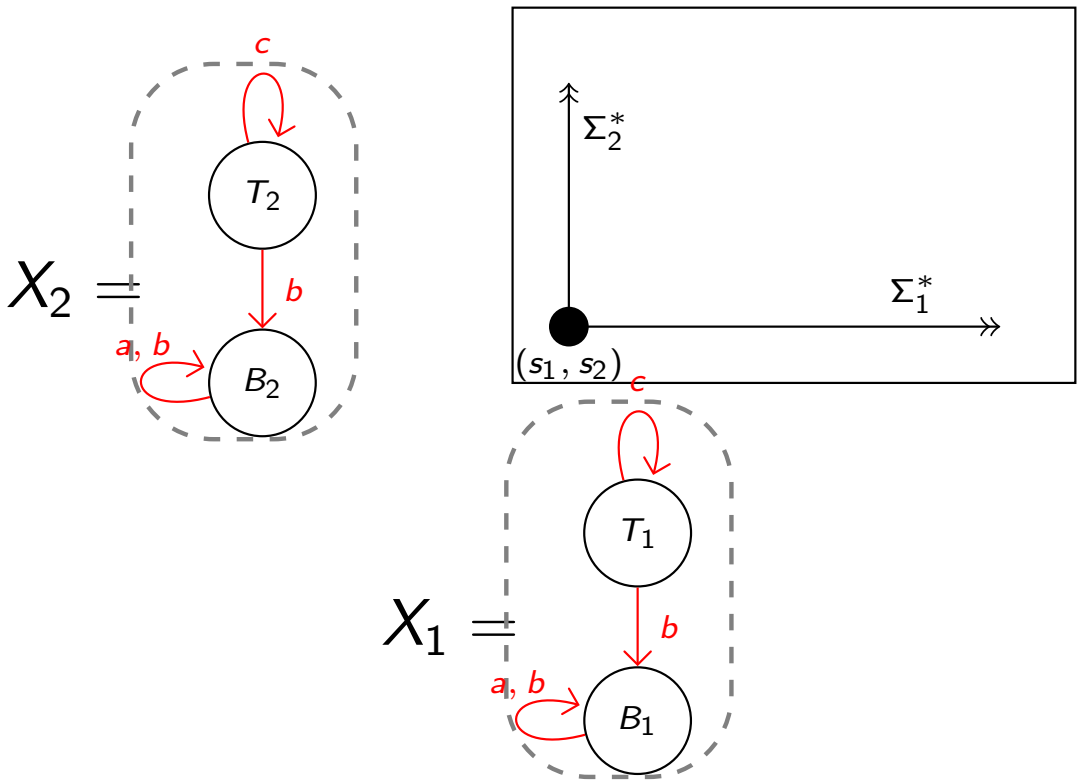
Global Safety for two process:

A fixpoint Approach

Environment Does Not Play Any Synchronizations: Trap Pair

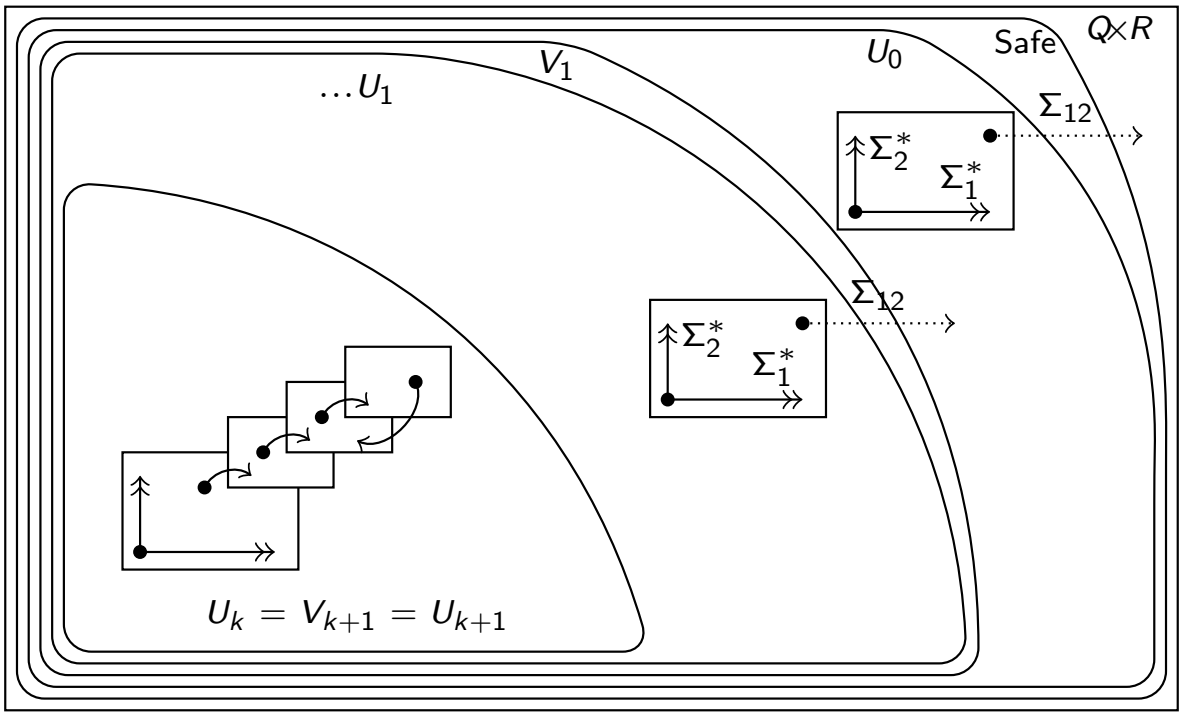
A key element

i-trap: a subset of NFA i states such that, for every local action, there is a successor inside it.



- pair of i-traps $= X_1, X_2$
- Rectangle $R = X_1 \times X_2$
- $R \subseteq$ Safe
- Environment does not play any synchronizations
- $U_0 = \bigcup_{R \subseteq \text{Safe}} R$

Environment plays synchronization: fixpoint approach



- $V_0 = \text{Safe}$,
- $U_0 = \bigcup_{R \subseteq V_0} R$
- $V_{j+1} = U_j \setminus \{s \mid \exists a \in \Sigma_{12}\ \exists s \xrightarrow{a} s', s' \notin U_j\}$
- $U_j = \bigcup_{R \subseteq V_j} R$

It turns out that:

$U_j = \{s \in \text{Safe} \mid$ from s system can win against an environment which plays atmost 'j' joint actions $\}$

$\text{Safe} = V_0 \supseteq U_0 \supseteq V_1 \supseteq U_1 \ldots$ and $U_k = V_{k+1} = U_{k+1}$

Memory in 2 process

Process i remembers -

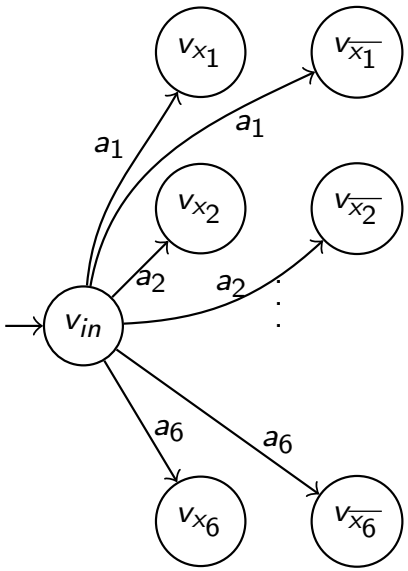
- Joint state after last joint action
- local state of process i

Each joint state reached after a synchronization $\longmapsto$ Rectangle

- This problem is in NP as
  - If there is a strategy there is a polynomial size strategy
  - Given a strategy it can be verified in polynomial time.

NP-hardness

$s_{SAT} = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_3})$

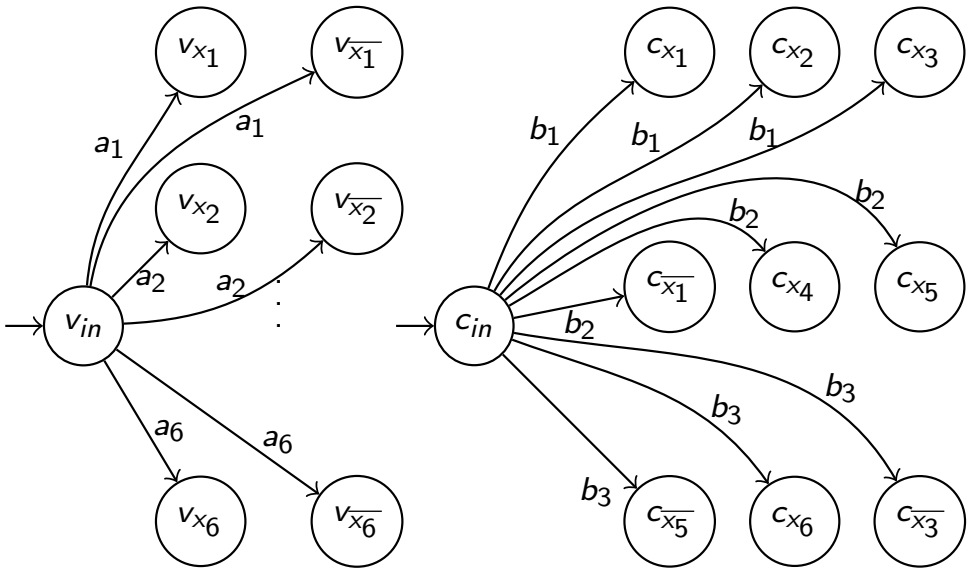


$P_{variable}$

- $P_{variable}$
  - Environment's choice of action a_i asks value of v_i
  - System's choice of next state assigns value of v_i

NP-hardness

$$s_{SAT} = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_3})$$

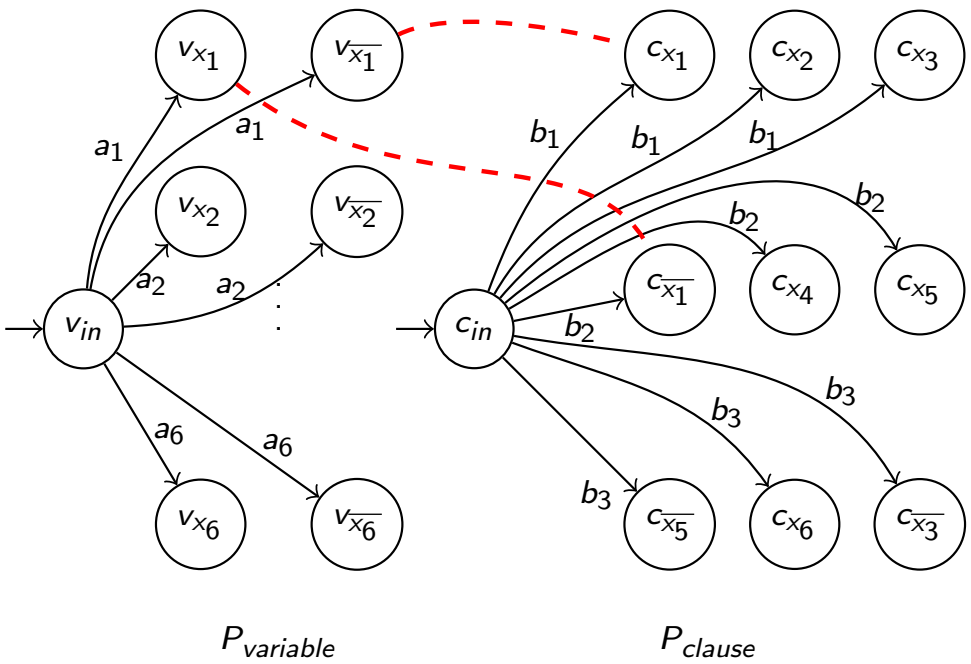


$P_{variable}$ P_{clause}

- $P_{variable}$
  - Environment's choice of action a_i asks value of v_i
  - System's choice of next state assigns value of v_i
- P_{clause}
  - Environment's choice of action b_j asks for term in c_j that makes it true
  - System's choice of next state chooses term in c_j that makes it true

NP-hardness

$s_{SAT} = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_3})$



Two of the unsafe states are depicted by red dashed lines.

Satisfying assignment $\Leftrightarrow$ Winning strategy

- $P_{variable}$
  - Environment's choice of action a_i asks value of v_i
  - System's choice of next state assigns value of v_i
- P_{clause}
  - Environment's choice of action b_j asks for term in c_j that makes it true
  - System's choice of next state chooses term in c_j that makes it true

Global Safety Objective

Problem Statement

Given a 2 process safety game, find the states from where system has a winning memory strategy.

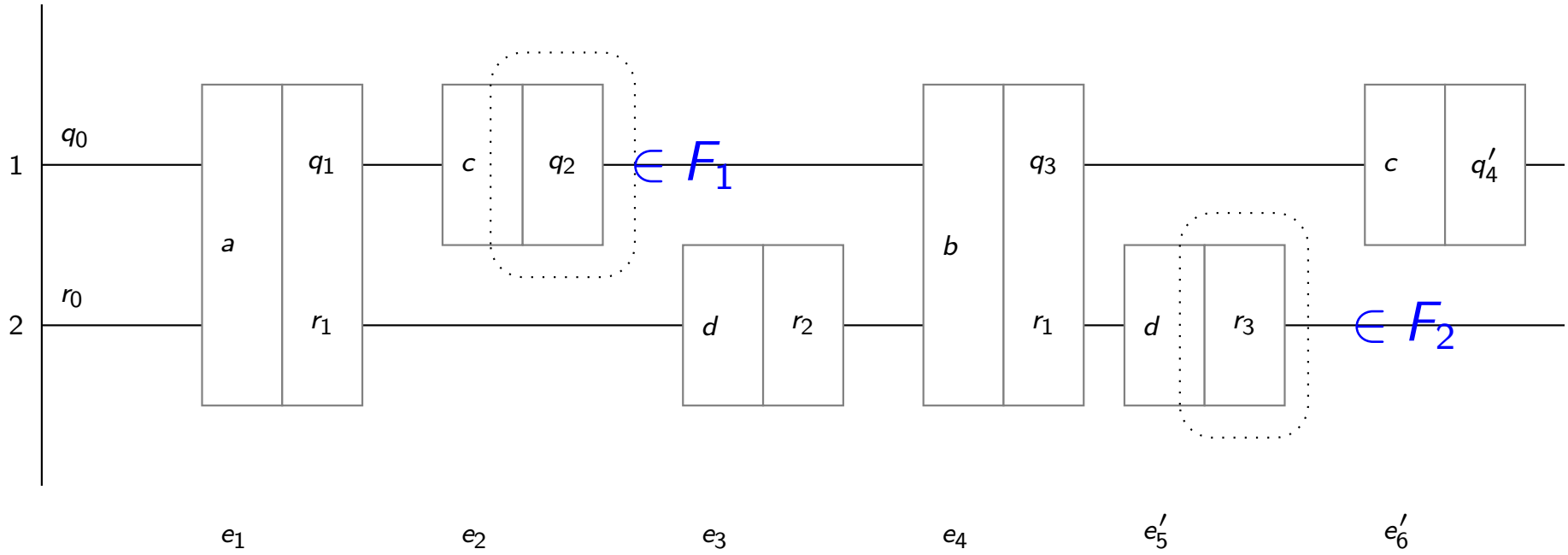
- Objective: Avoid unsafe states throughout the play.

Theorem

Two process global safety games are NP-complete.

Local Reachability Objective

- Given by set $F_1 \subseteq$ NFA 1 states, $F_2 \subseteq$ NFA 2 states.
- Objective: Each process observes a state in the goal set.



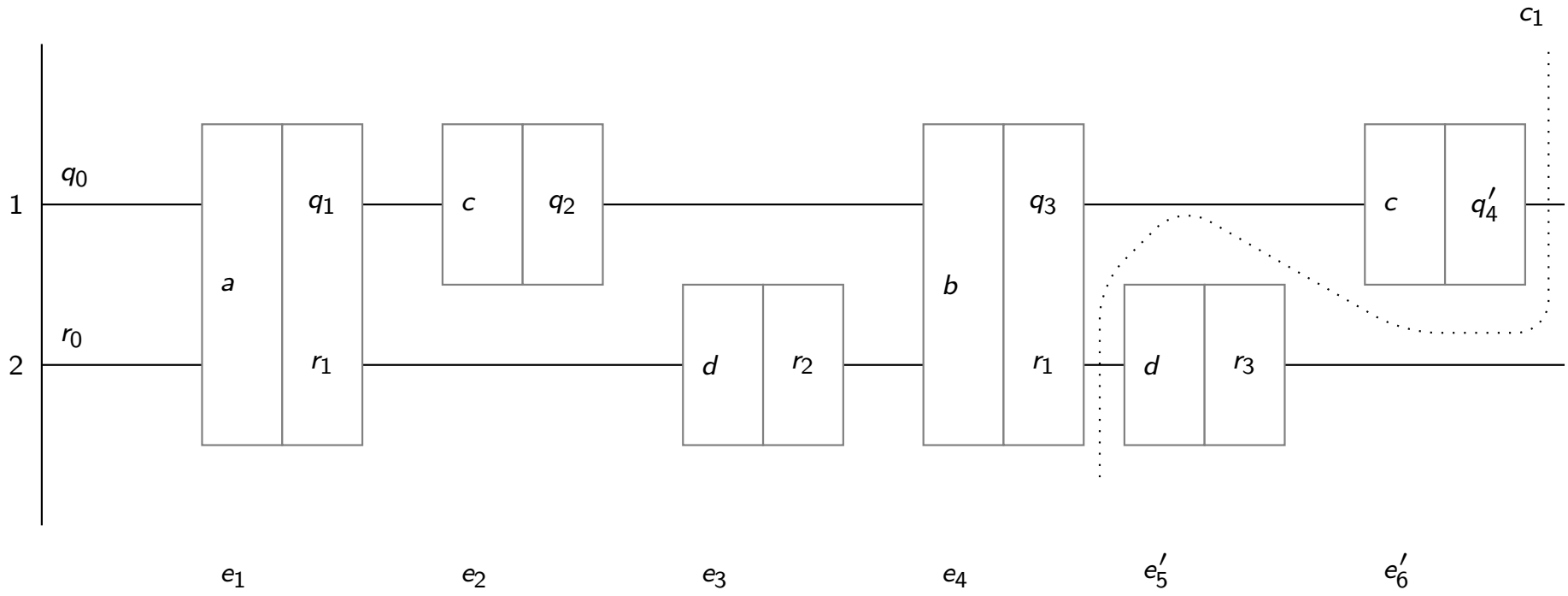
Example of a play from initial state (q_0, r_0)

Theorem

Two process local reachability games are NP-complete.

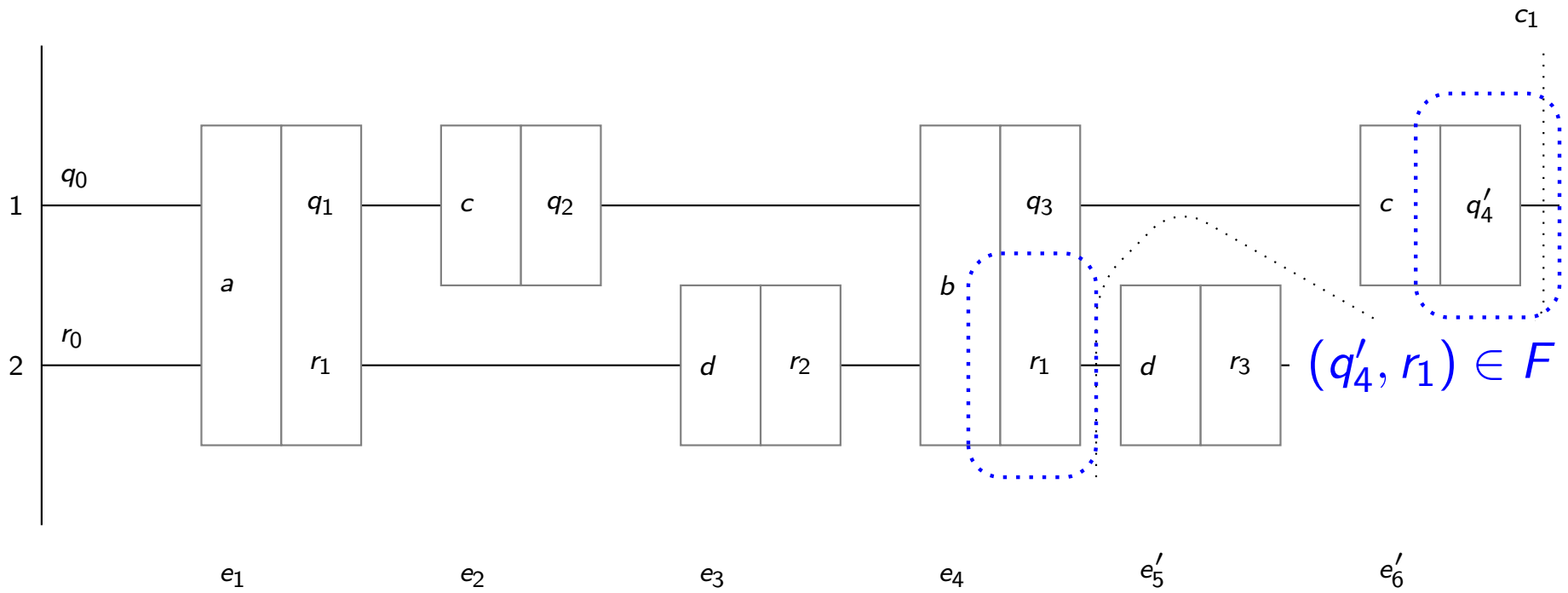
Global Reachability Objective

- Given by set $F \subseteq$ NFA 1 states $\times$ NFA 2 states.
- Objective: Team reaches a target global state.
- Complexity: PSPACE-hard
- In NEXPTIME



Global Reachability Objective

- Given by set $F \subseteq$ NFA 1 states $\times$ NFA 2 states.
- Objective: Team reaches a target global state.
- Complexity: PSPACE-hard
- In NEXPTIME



One vs. Two Processes: Summary So Far

One NFA

Winning condition	Decidability Results	Information needed to decide next move
Safety	Poly time	Current state
Reachability	Poly time	Current state

Two NFAs

Winning condition	Decidability Results	Information needed by each process to decide next move
Global Safety	NP-complete	Joint state after last joint action + current state
Local Reachability	NP-complete	Joint state after last joint action +current state + local goal seen flag
Global Reachability	PSPACE hard, in NEXPTIME	Joint state after last joint action + local states seen since

- Introduced ATS games for 2 processes.
- Developed algorithms and characterized computational complexity.

ATS games with a Central Decision Maker

CDM systems

Distributed settings with a designated process, **c**entral **d**ecision **m**aker, which participates in all key decisions

- Server-client architectures – server maintains overall integrity
- Distributed version control systems which maintain a single master copy; Users make concurrent changes to local copies; Changes to the master copy are made by acquiring an exclusive access.
- Working of an organization with multiple agents but a designated head; Several committees are formed and work concurrently; Head is a member of all the decision-making committees.

The cdm participates in all decision making activities.

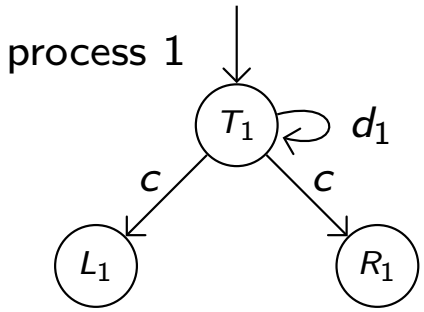
CDM Game: Example

CDM games

A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.

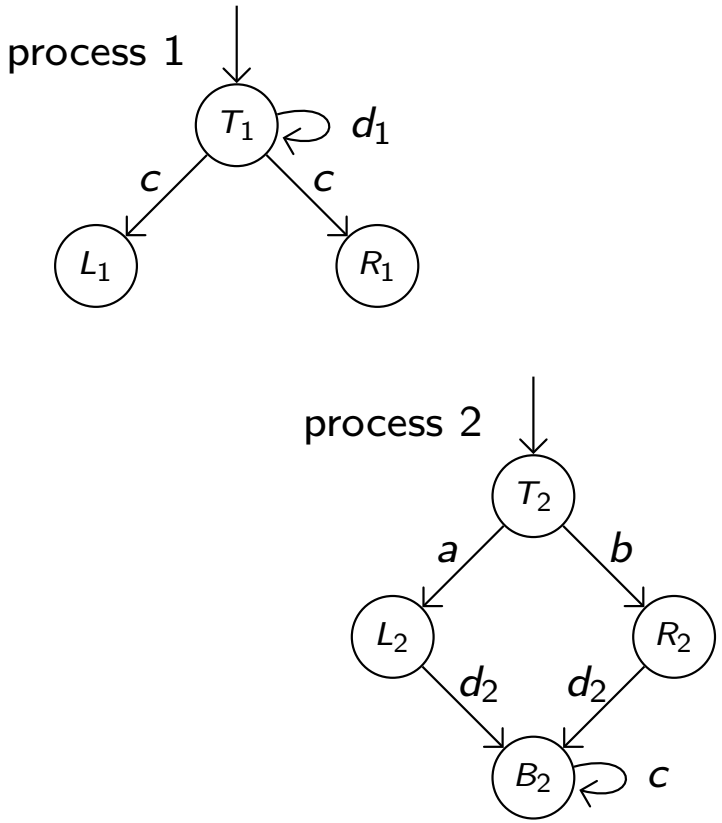
CDM games

A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



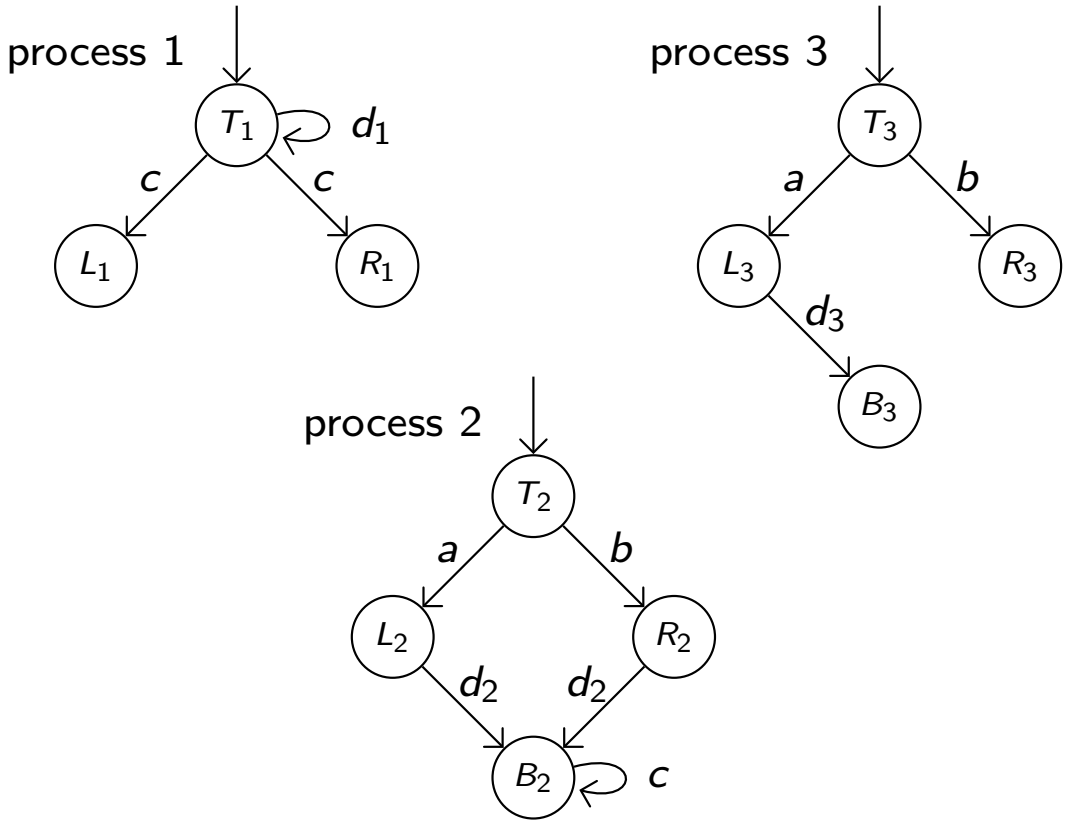
CDM games

A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



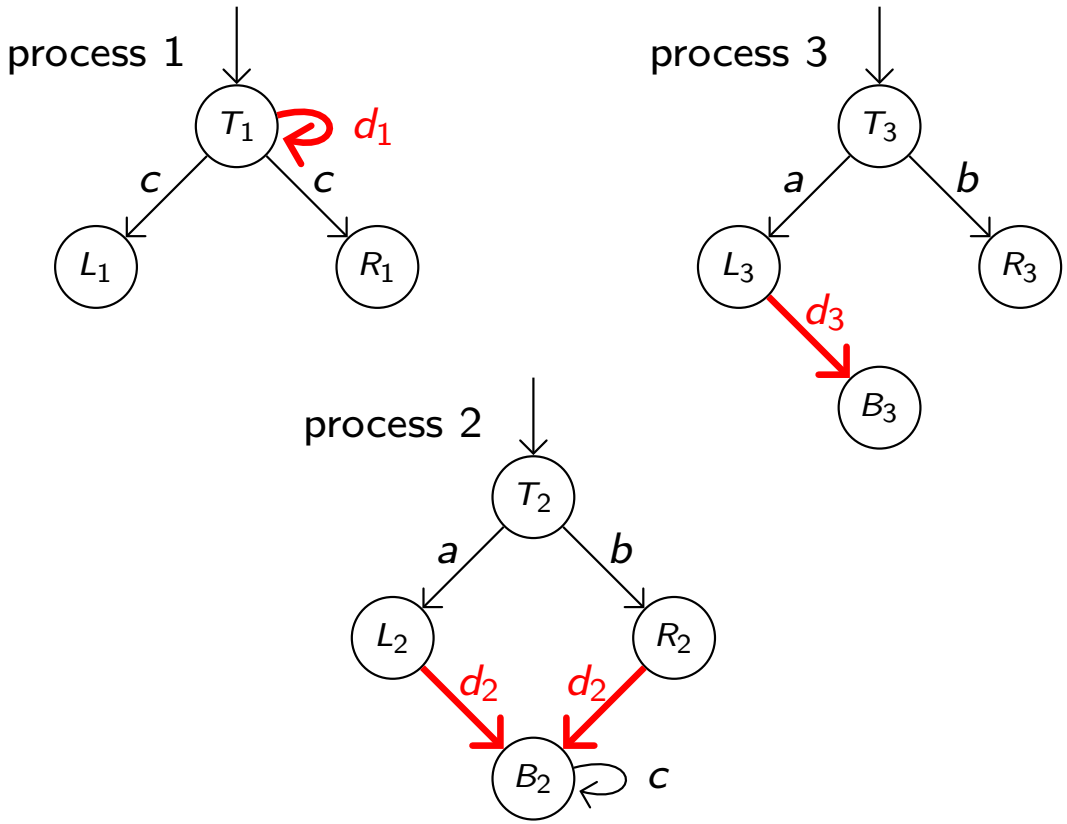
CDM games

A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



CDM games

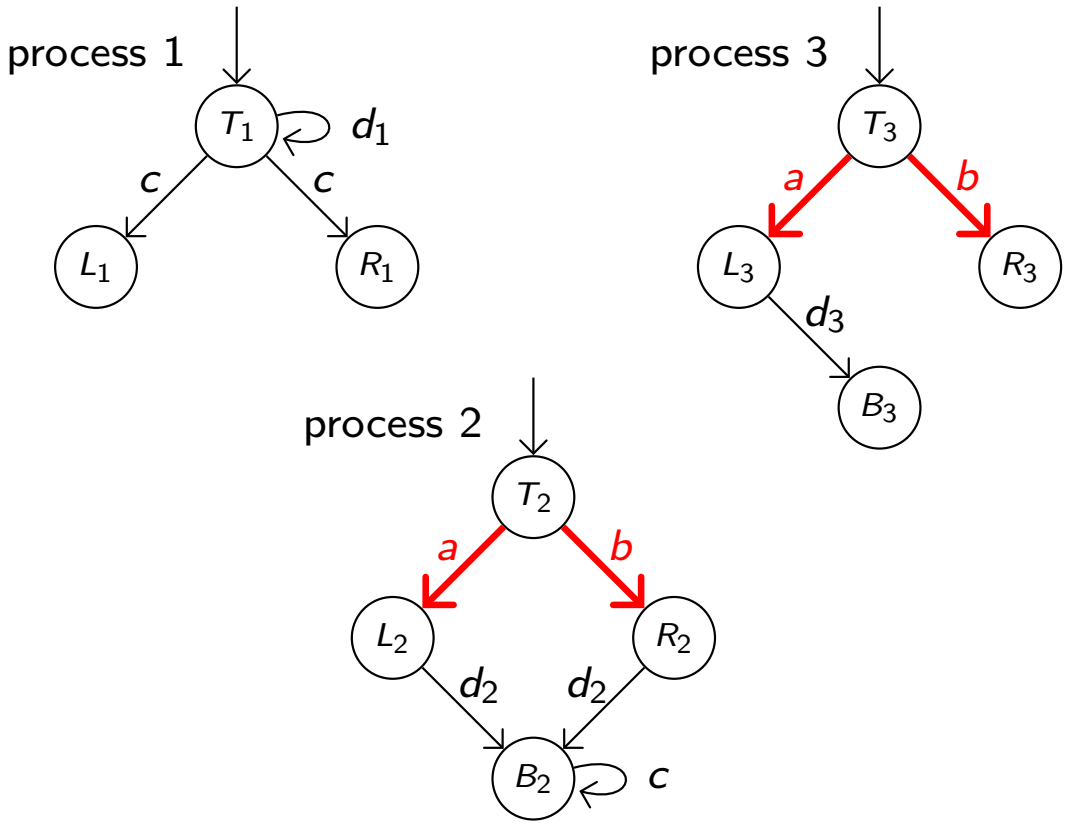
A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



- d_i is a local action of process i

CDM games

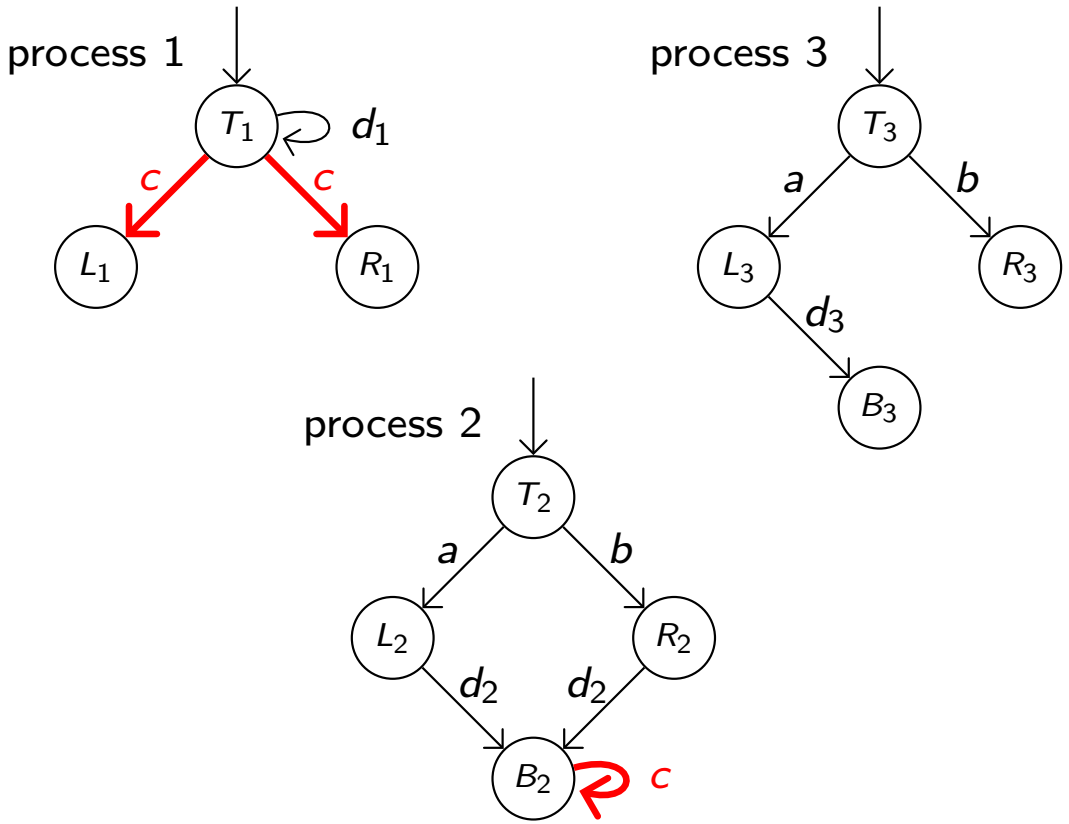
A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



- d_i is a local action of process i
- processes 2 and 3 share deterministic actions a and b

CDM games

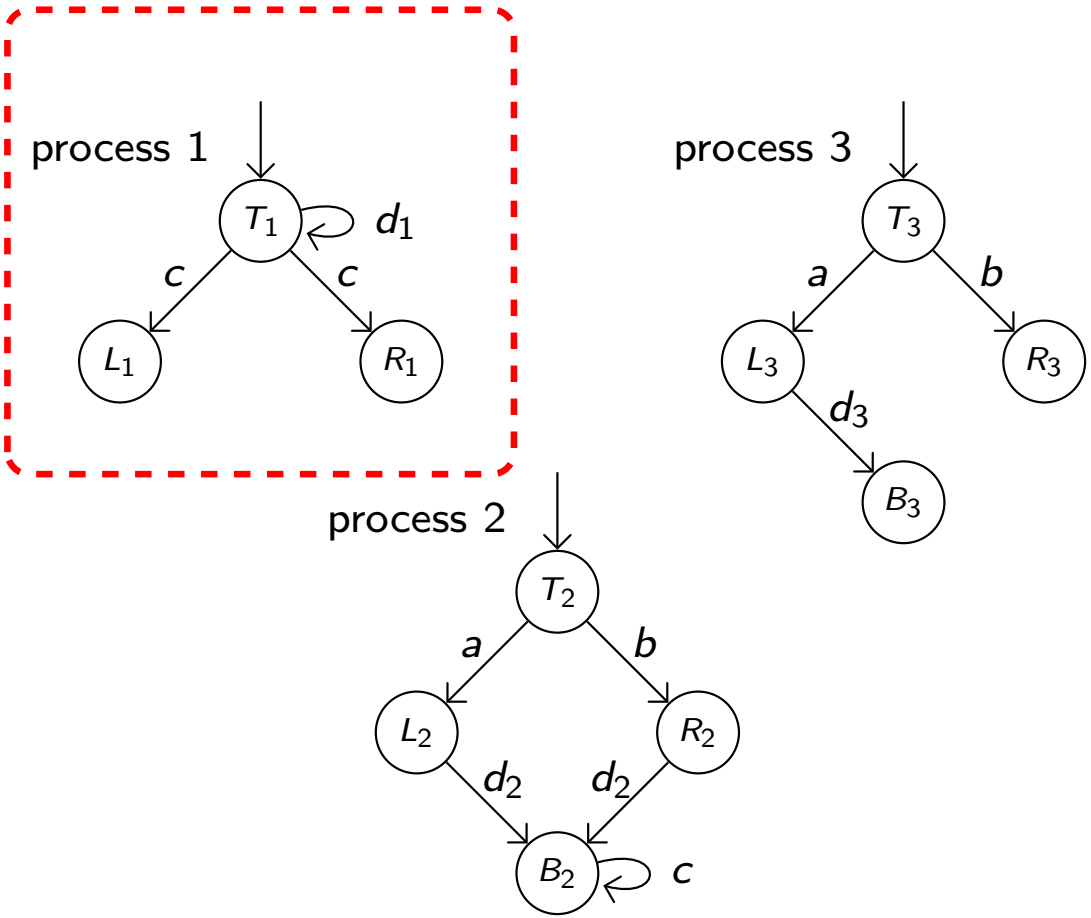
A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



- d_i is a local action of process i
- processes 2 and 3 share deterministic actions a and b
- processes 1 and 2 share non deterministic action c
  $(T_1, B_2) \xrightarrow{c} (L_1, B_2)$,
  $(T_1, B_2) \xrightarrow{c} (R_1, B_2)$

CDM games

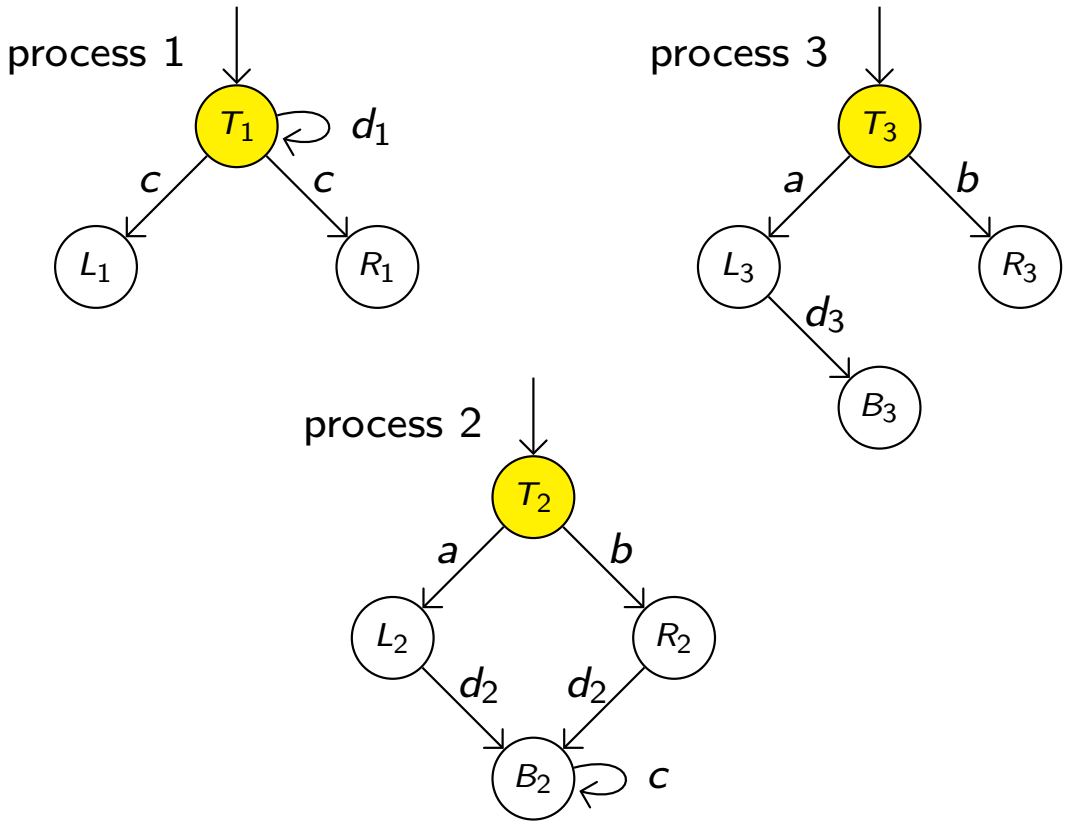
A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



- d_i is a local action of process i
- processes 2 and 3 share deterministic actions a and b
- processes 1 and 2 share non deterministic action c
  $(T_1, B_2) \xrightarrow{c} (L_1, B_2)$,
  $(T_1, B_2) \xrightarrow{c} (R_1, B_2)$
- 1 is the designated cdm

CDM games

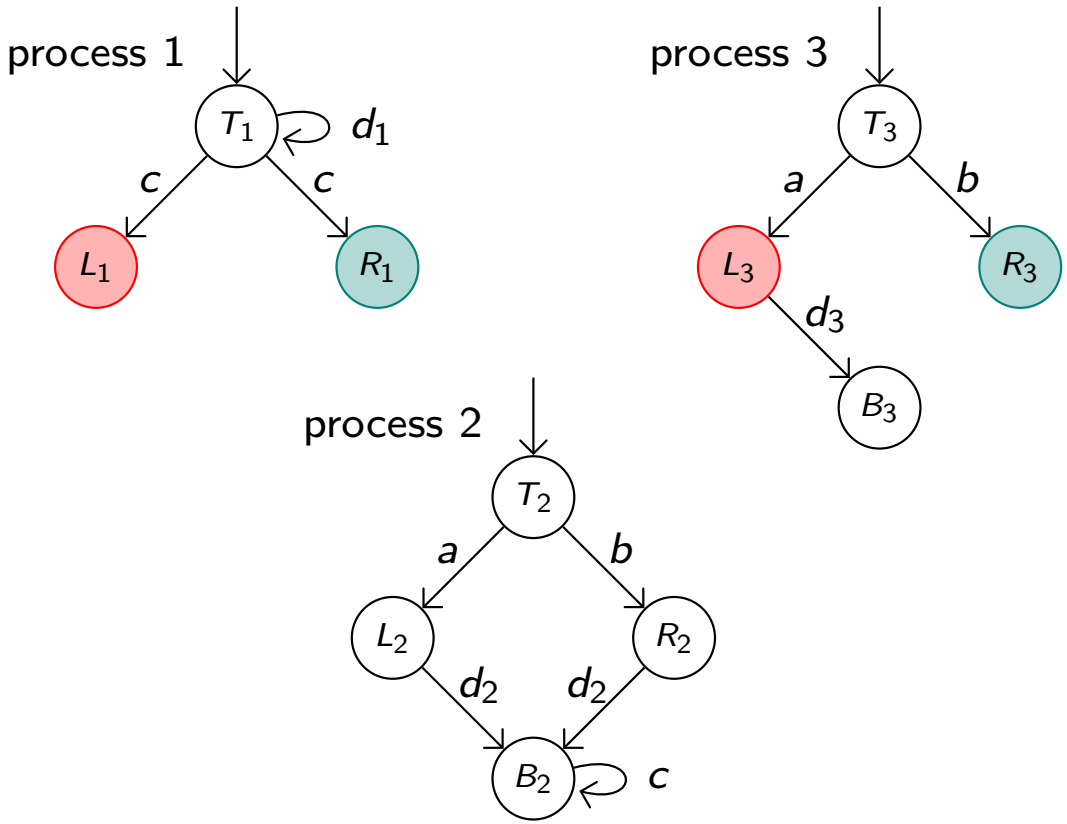
A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.



- d_i is a local action of process i
- processes 2 and 3 share deterministic actions a and b
- processes 1 and 2 share non deterministic action c
  $(T_1, B_2) \xrightarrow{c} (L_1, B_2)$,
  $(T_1, B_2) \xrightarrow{c} (R_1, B_2)$
- 1 is the designated cdm
- Initial state: (T_1, T_2, T_3)

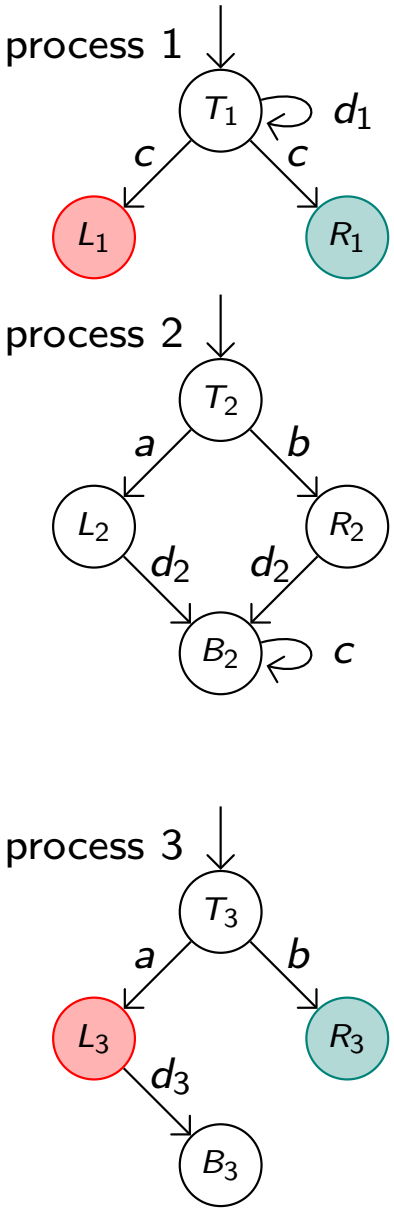
CDM games

A CDM game is a distributed game with a designated cdm process ℓ such that if an action a is not deterministic, then ℓ participates in a.

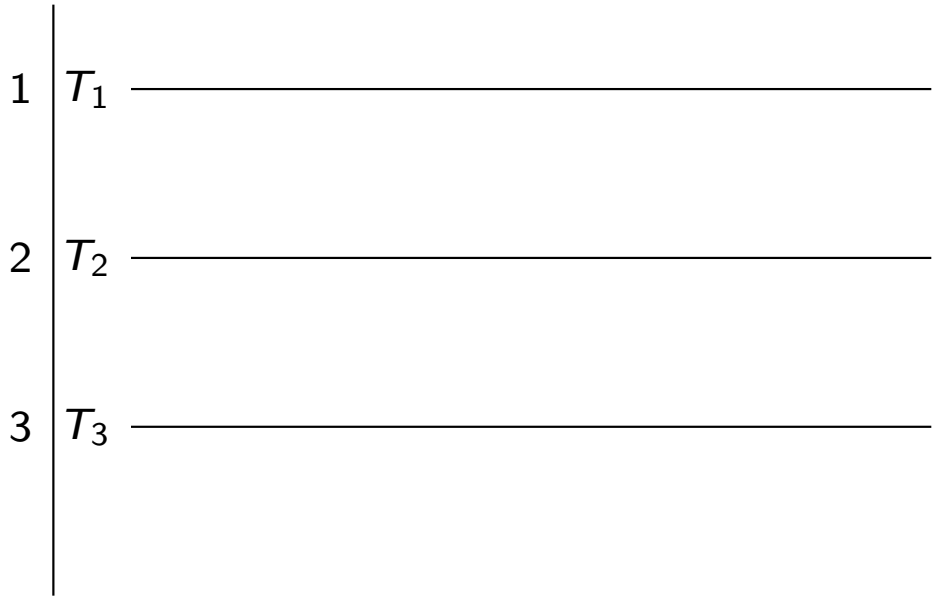


- d_i is a local action of process i
- processes 2 and 3 share deterministic actions a and b
- processes 1 and 2 share non deterministic action c
  $(T_1, B_2) \xrightarrow{c} (L_1, B_2)$,
  $(T_1, B_2) \xrightarrow{c} (R_1, B_2)$
- 1 is the designated cdm
- Initial state: (T_1, T_2, T_3)
- Unsafe set $\{(L_1, B_2, R_3), (R_1, B_2, L_3)\}$

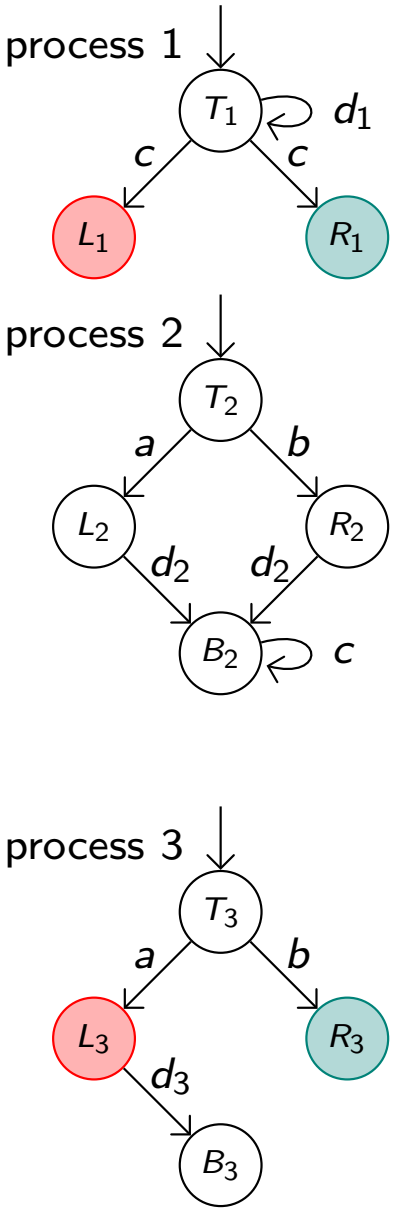
Example: A play in this game



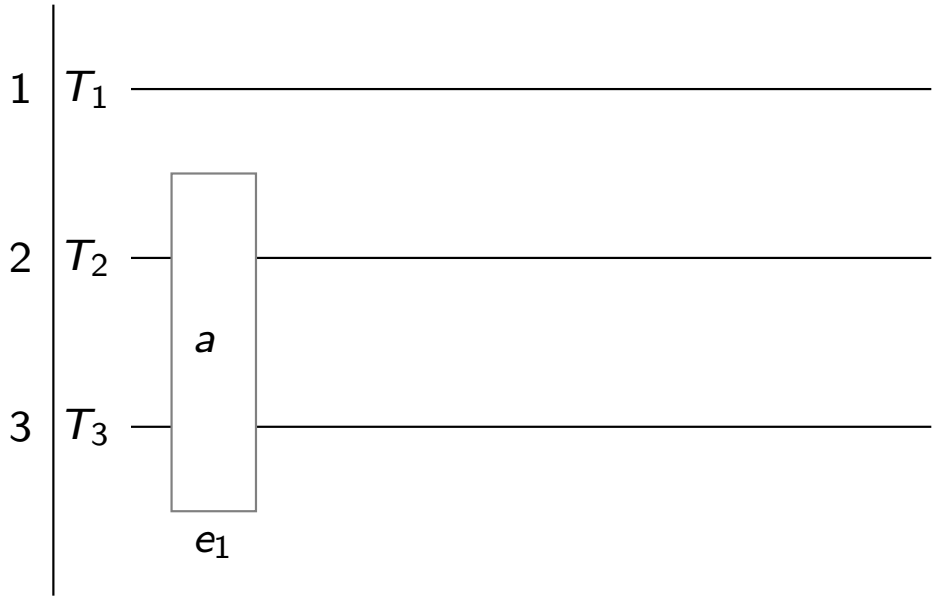
Example of a play from initial state (T_1, T_2, T_3)



Example: A play in this game

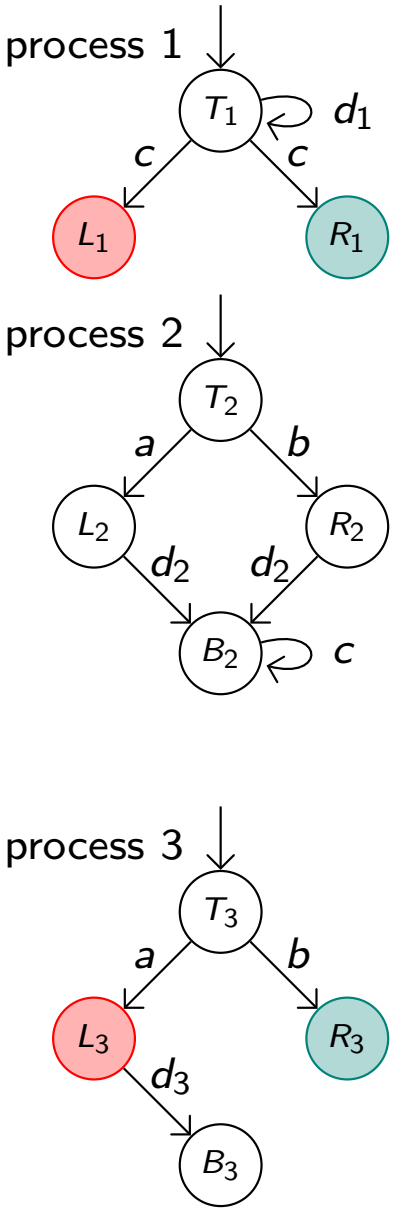


Example of a play from initial state (T_1, T_2, T_3)

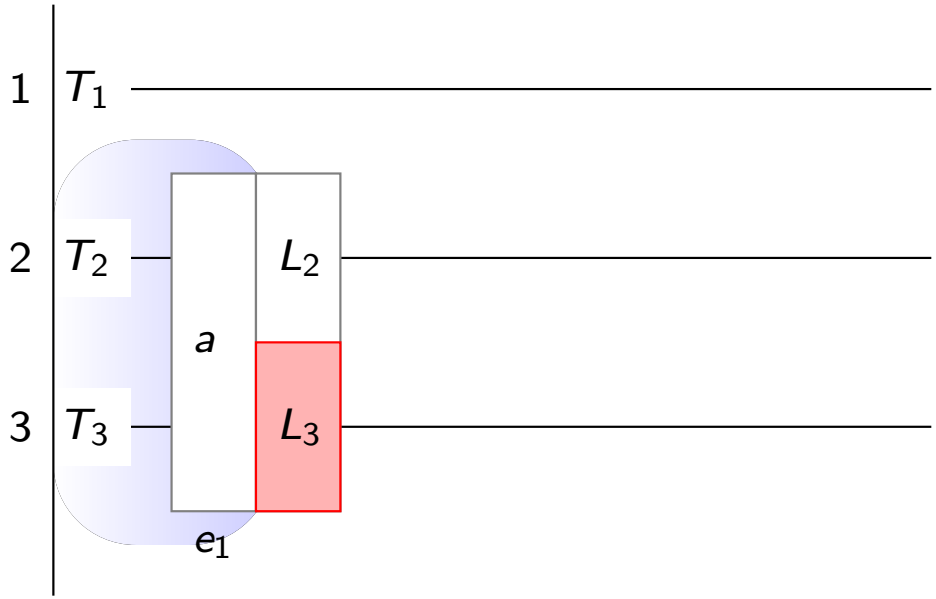


From the initial state environment can play an a or b. Say it plays action a

Example: A play in this game

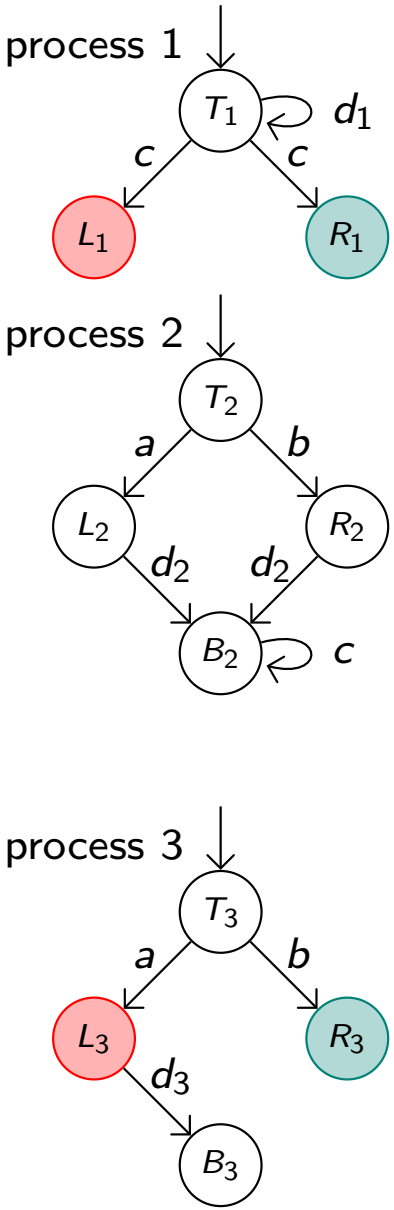


Example of a play from initial state (T_1, T_2, T_3)

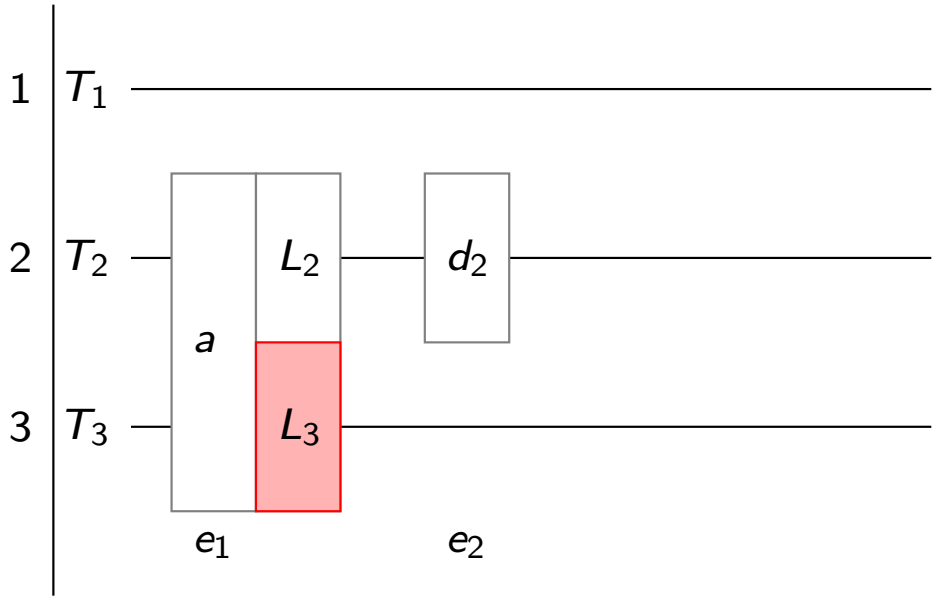


The participating processes respond by moving to the next state

Example: A play in this game

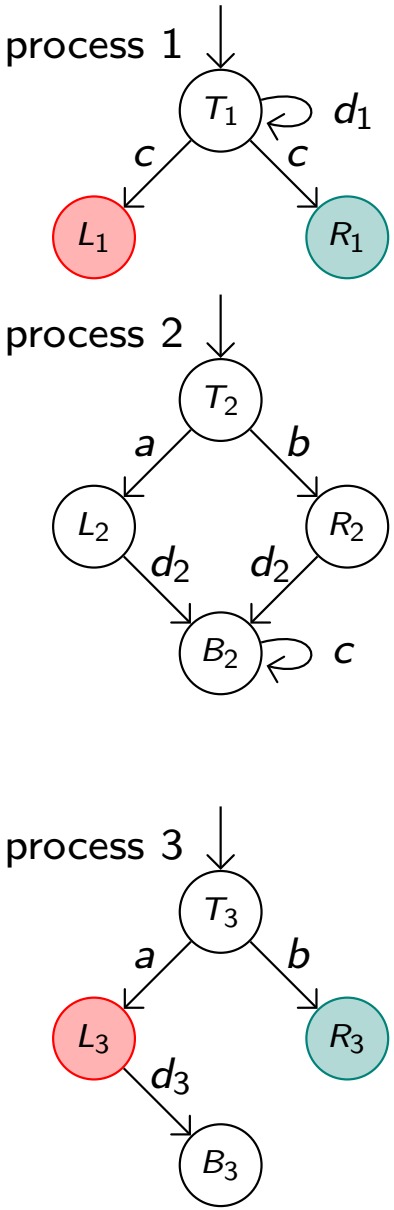


Example of a play from initial state (T_1, T_2, T_3)

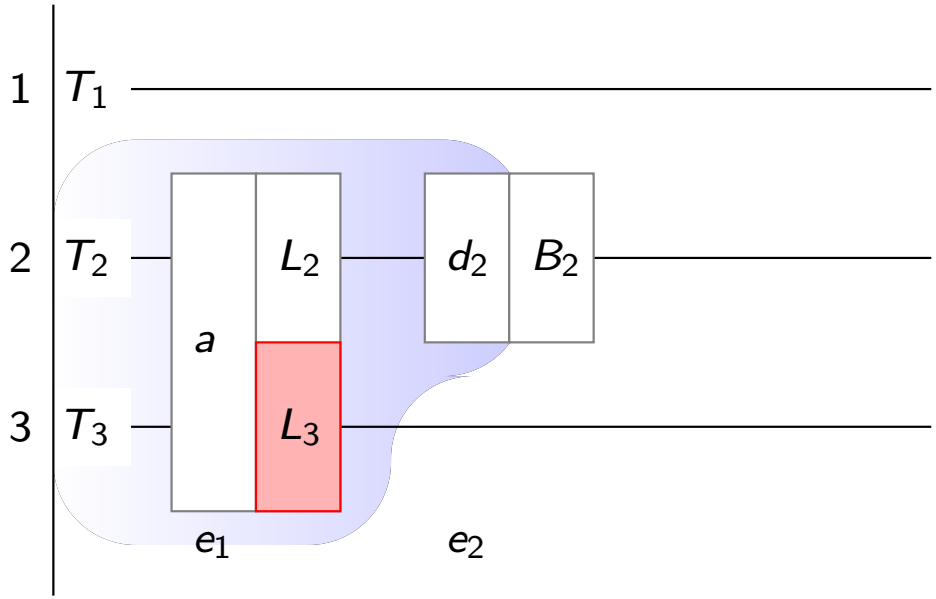


Next environment plays action d_2.

Example: A play in this game

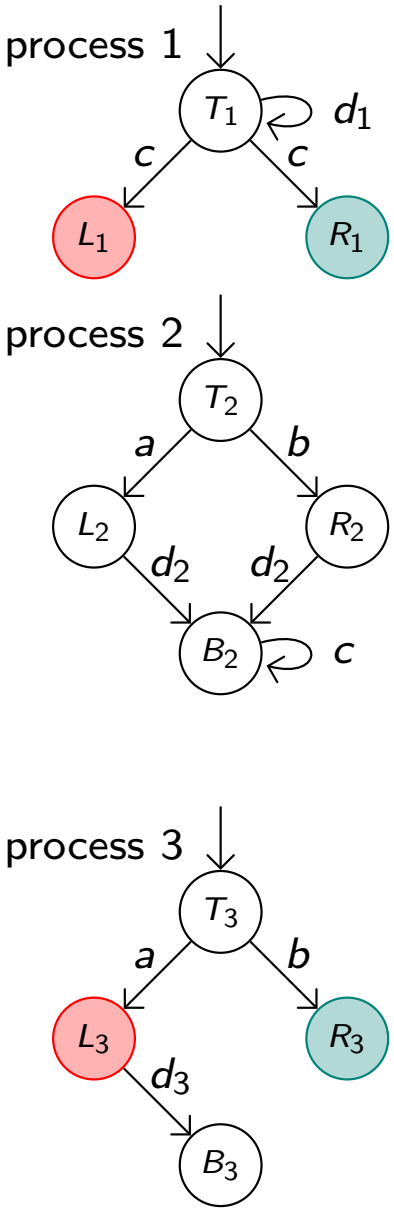


Example of a play from initial state (T_1, T_2, T_3)

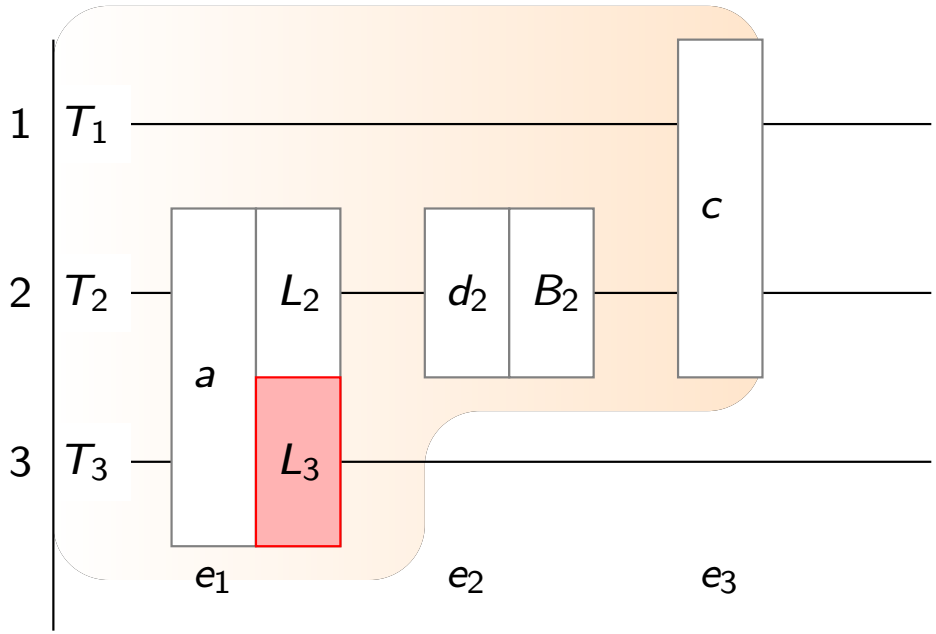


and process 2 moves to its bottom state

Example: A play in this game

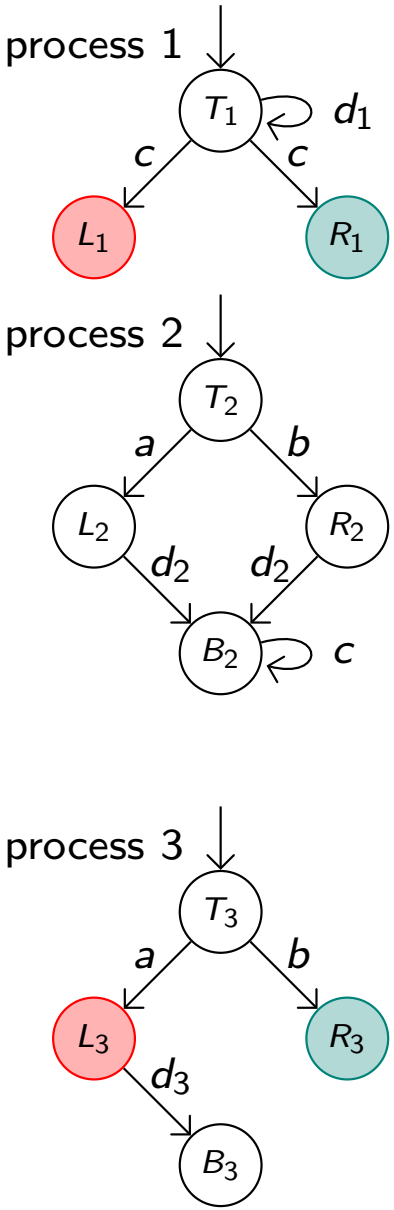


Example of a play from initial state (T_1, T_2, T_3)

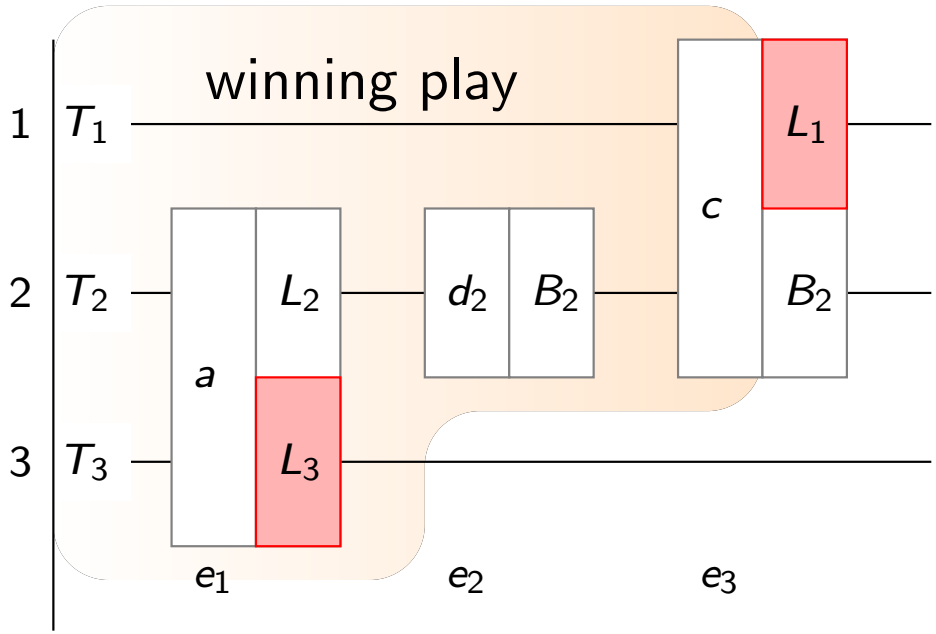


Next environemnt schedules a c

Example: A play in this game

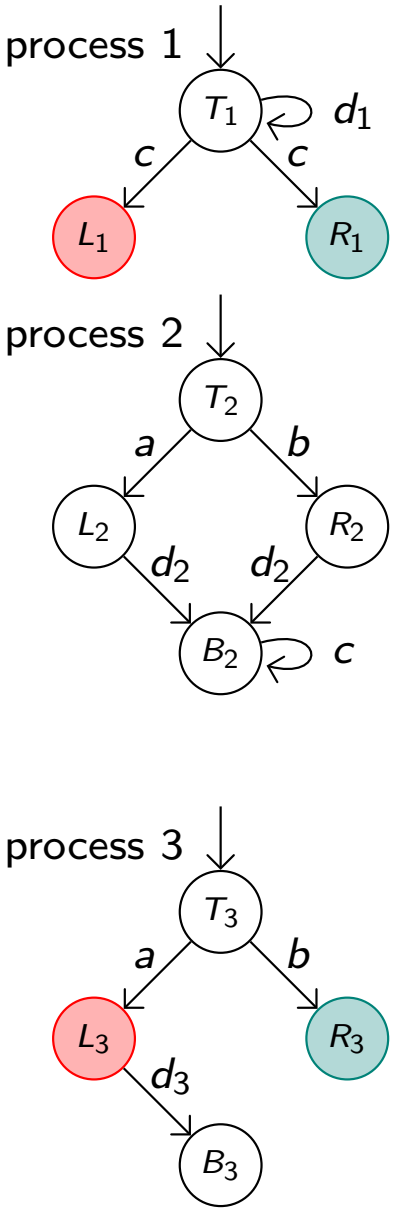


Example of a play from initial state (T_1, T_2, T_3)

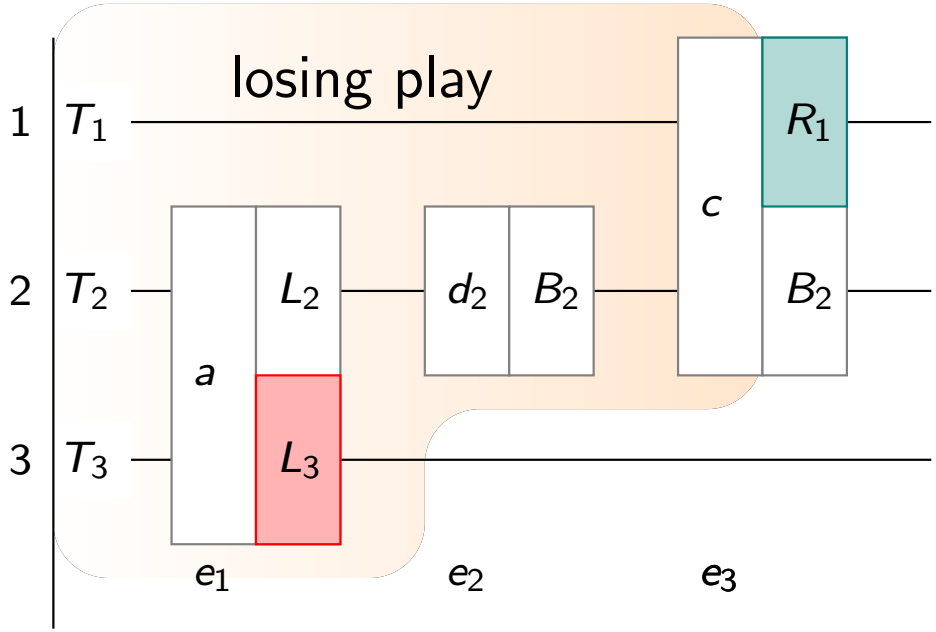


Process 1 and 2 can choose between left

Example: A play in this game

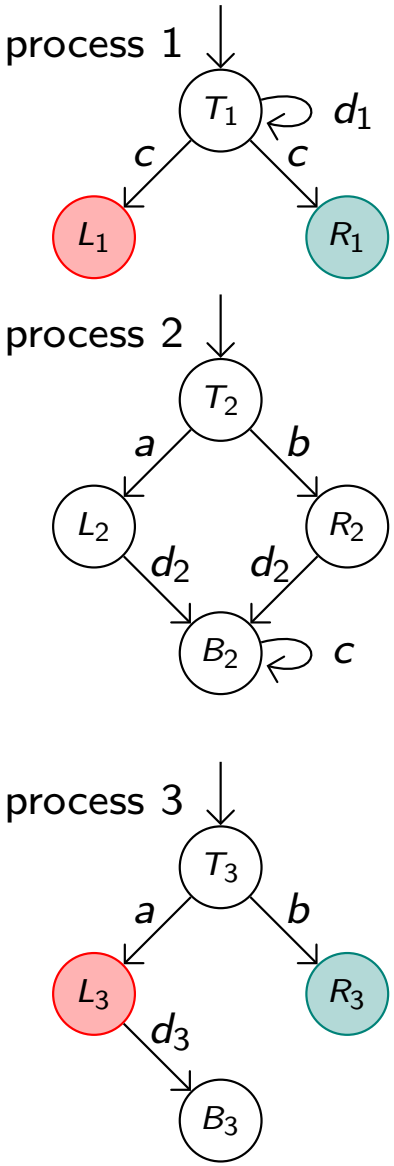


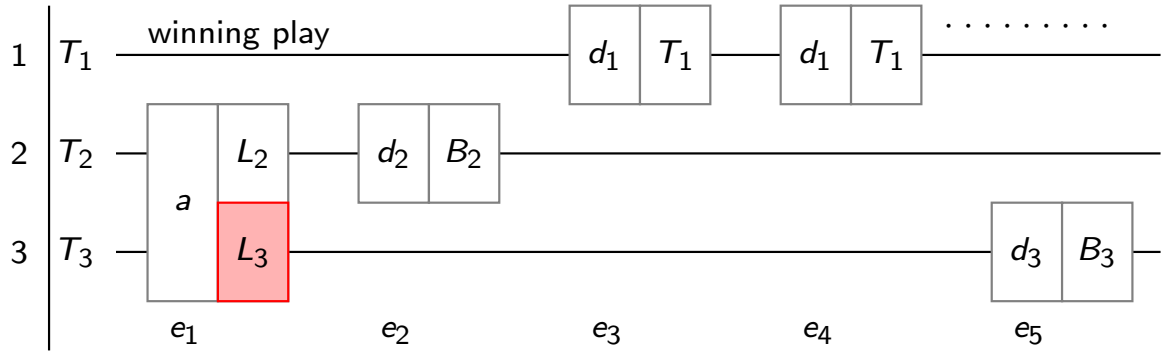
Example of a play from initial state (T_1, T_2, T_3)

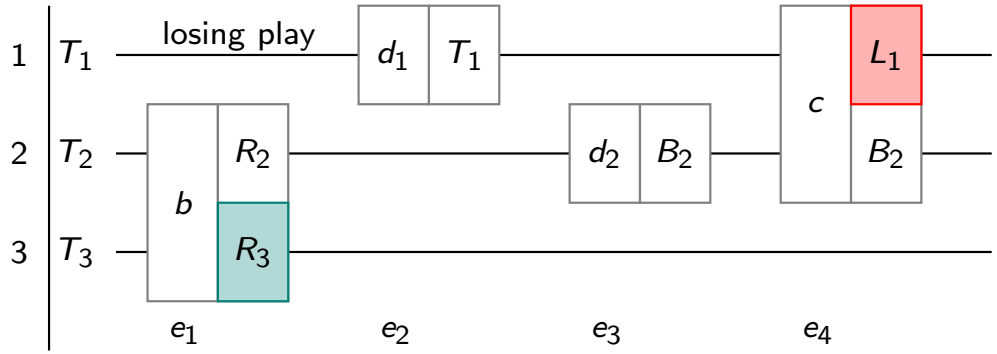


or right

CDM game plays: Winning condition

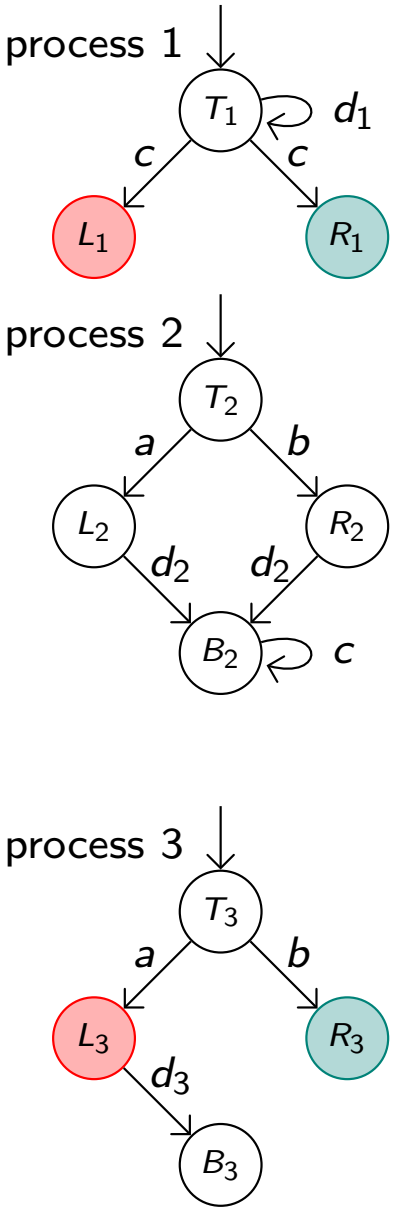


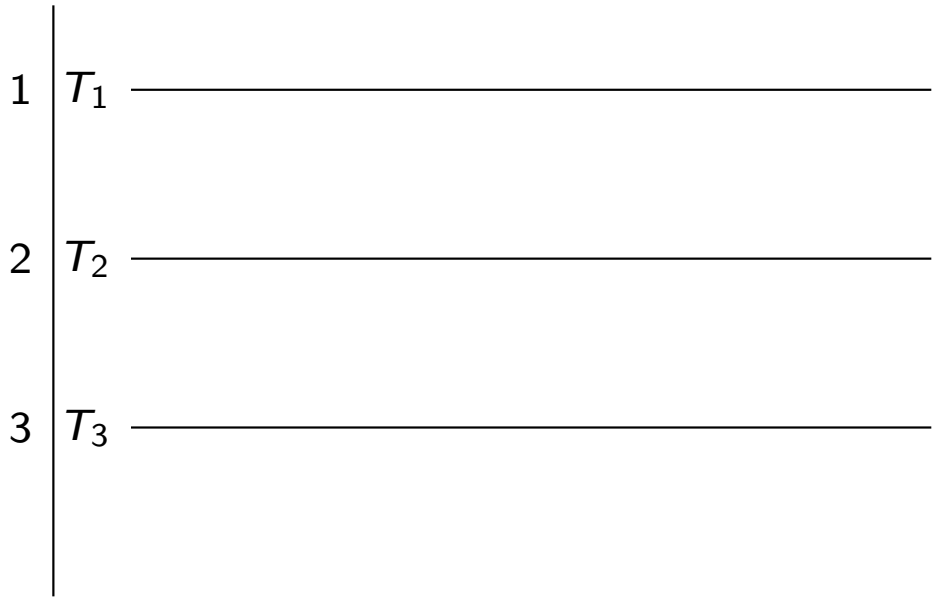




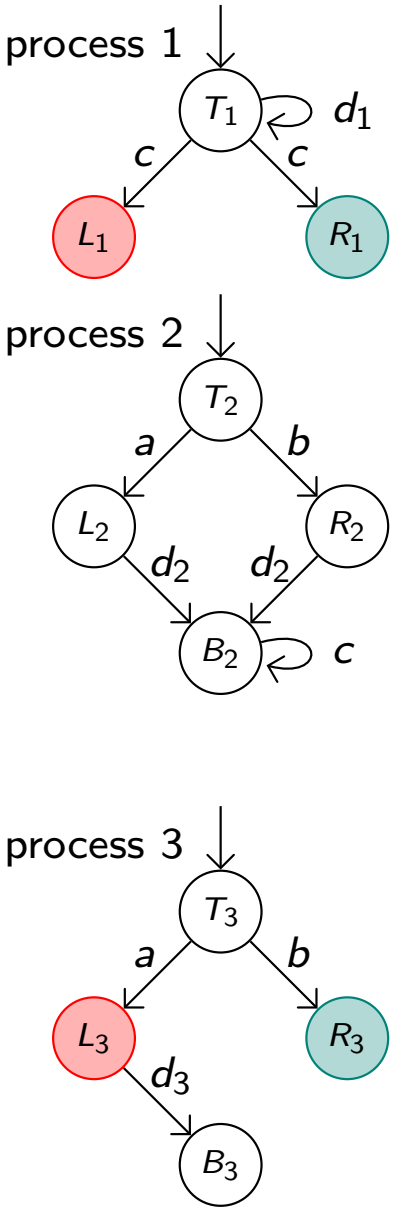
A winning strategy: Example

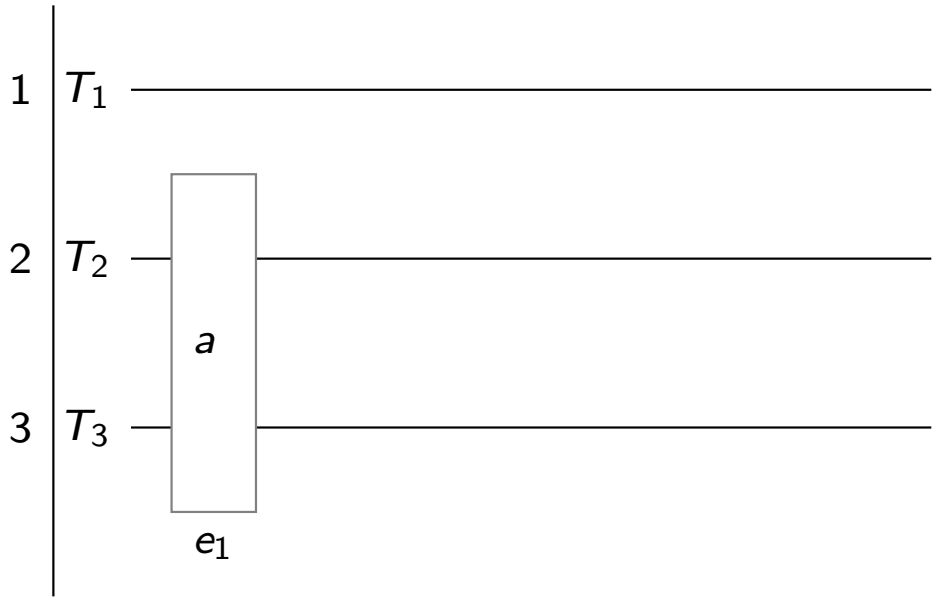
Example: A winning memory-based strategy



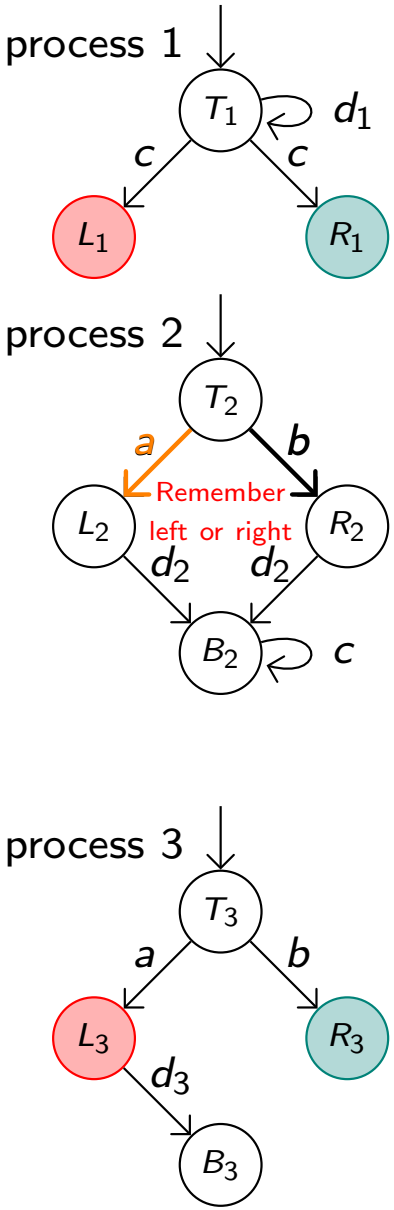


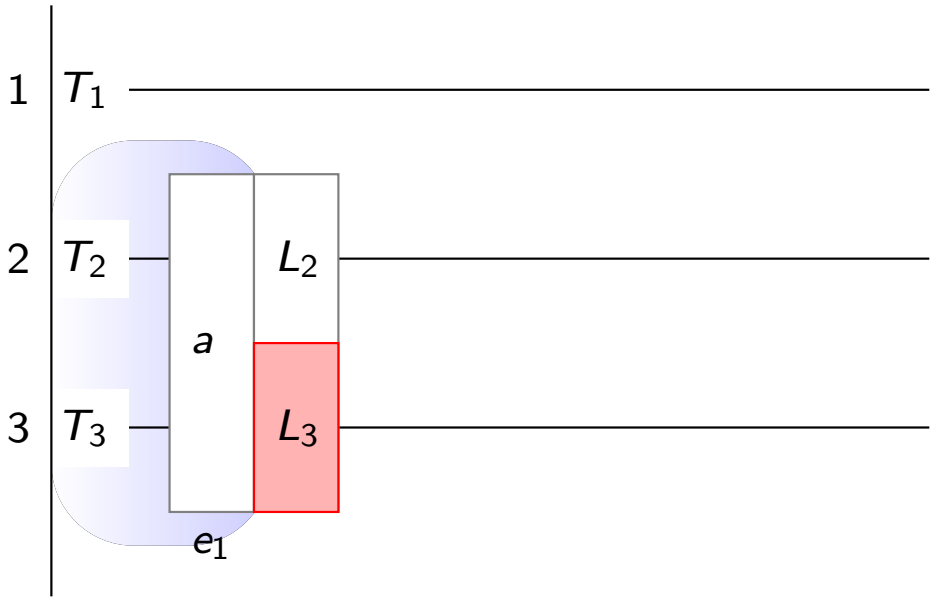
Example: A winning memory-based strategy





Example: A winning memory-based strategy

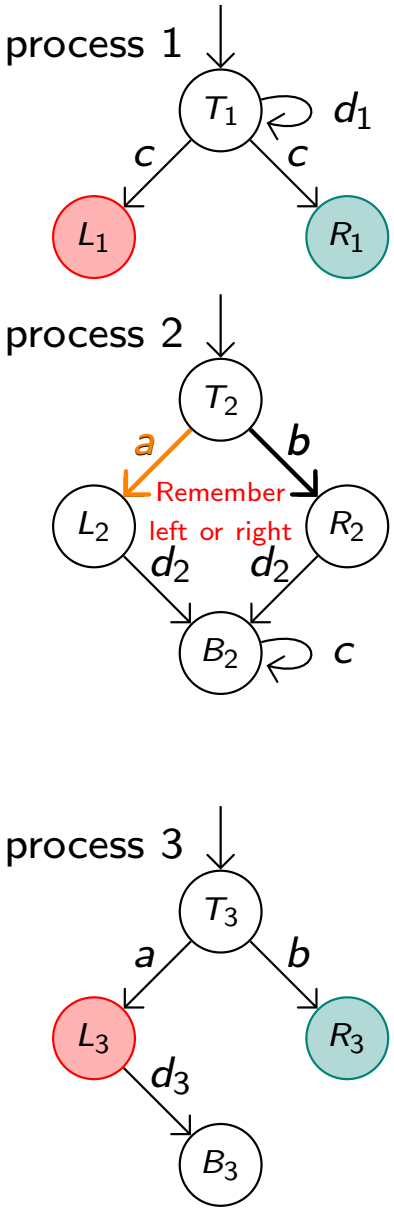


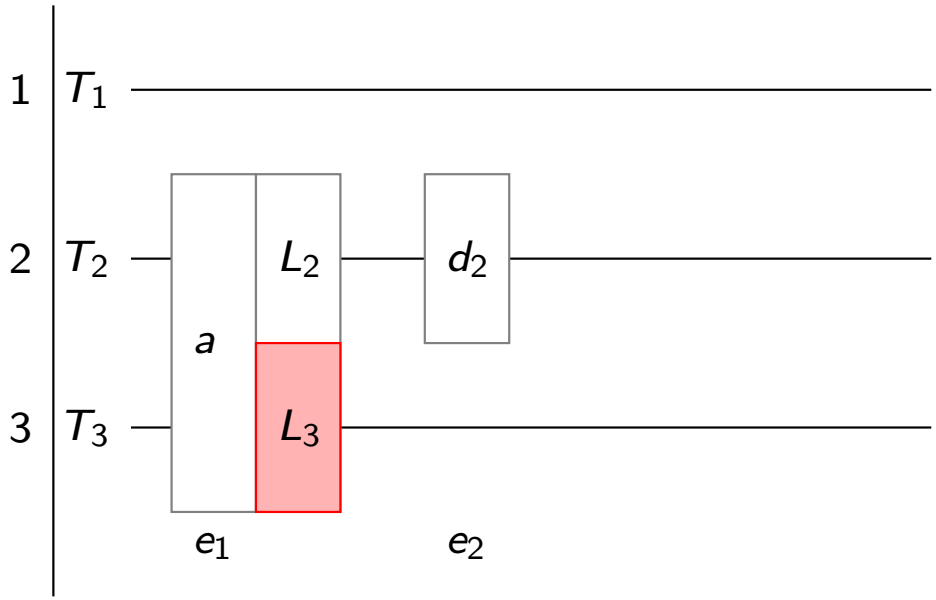


Process 2 remembers if the environment chose Left: a or Right: b

(non CDM process remembers)

Example: A winning memory-based strategy

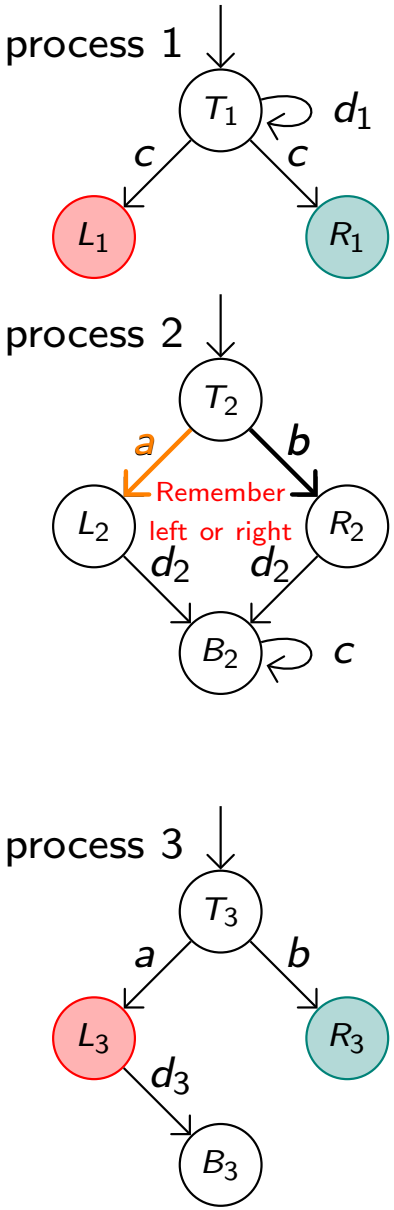


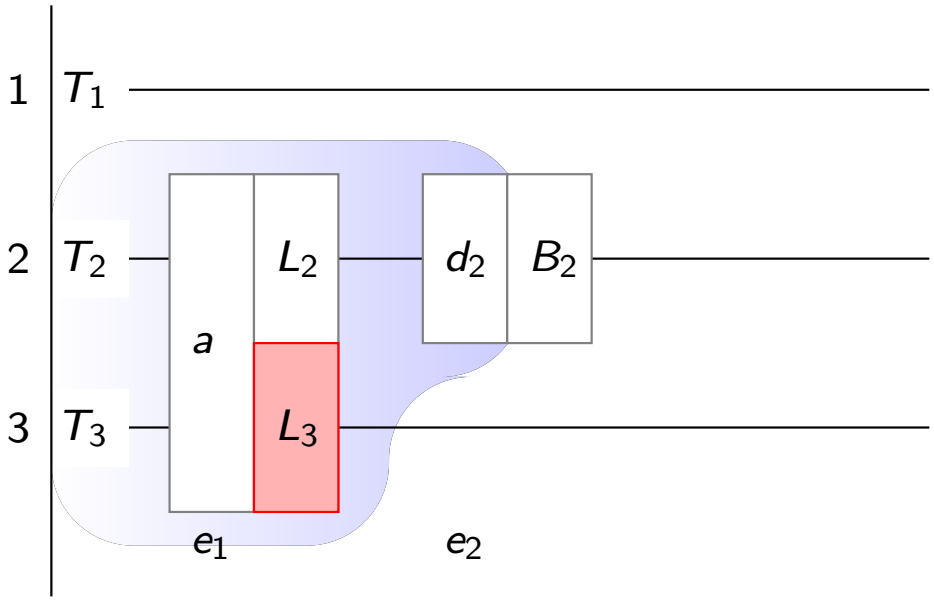


Process 2 remembers if the environment chose Left: a or Right: b

(non CDM process remembers)

Example: A winning memory-based strategy

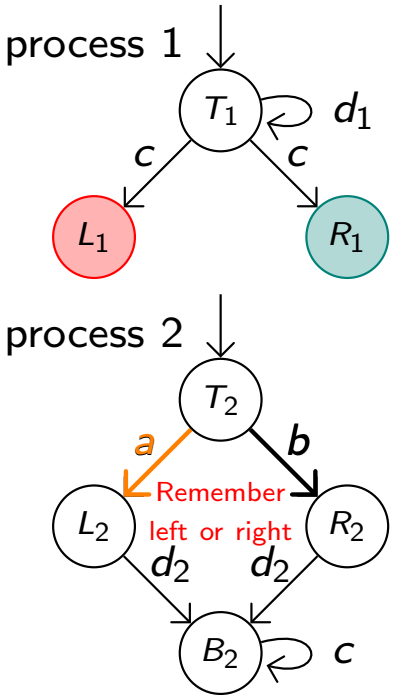


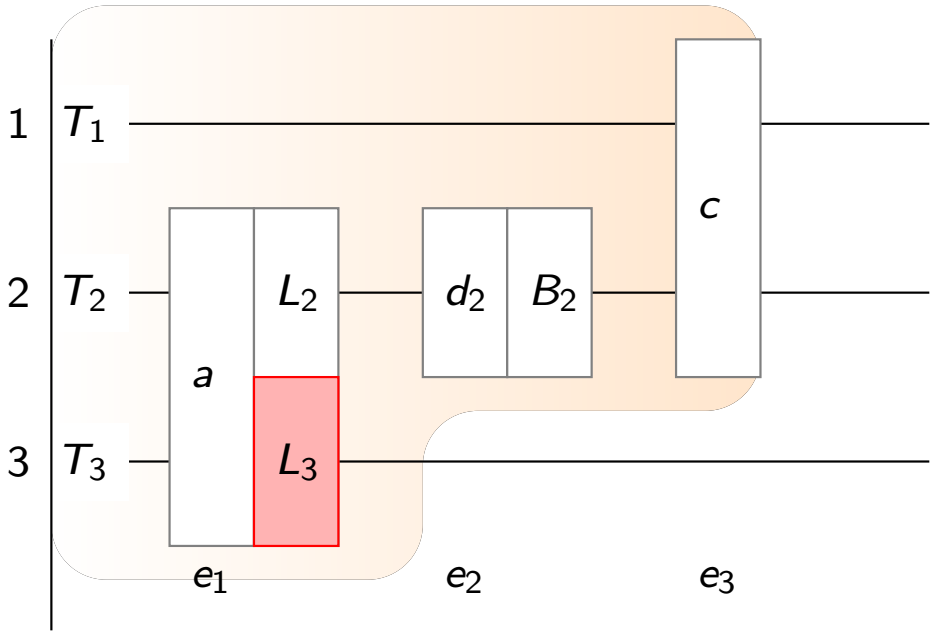


Process 2 remembers if the environment chose Left: a or Right: b

(non CDM process remembers)

Example: A winning memory-based strategy

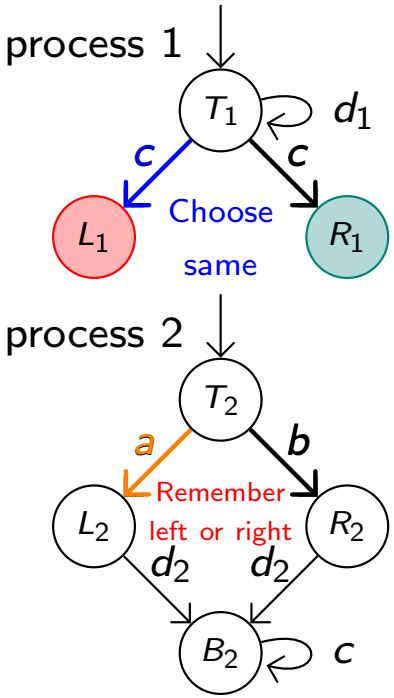


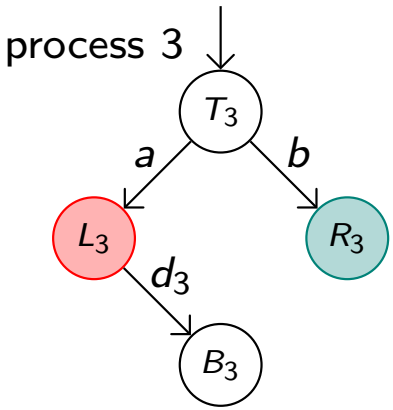


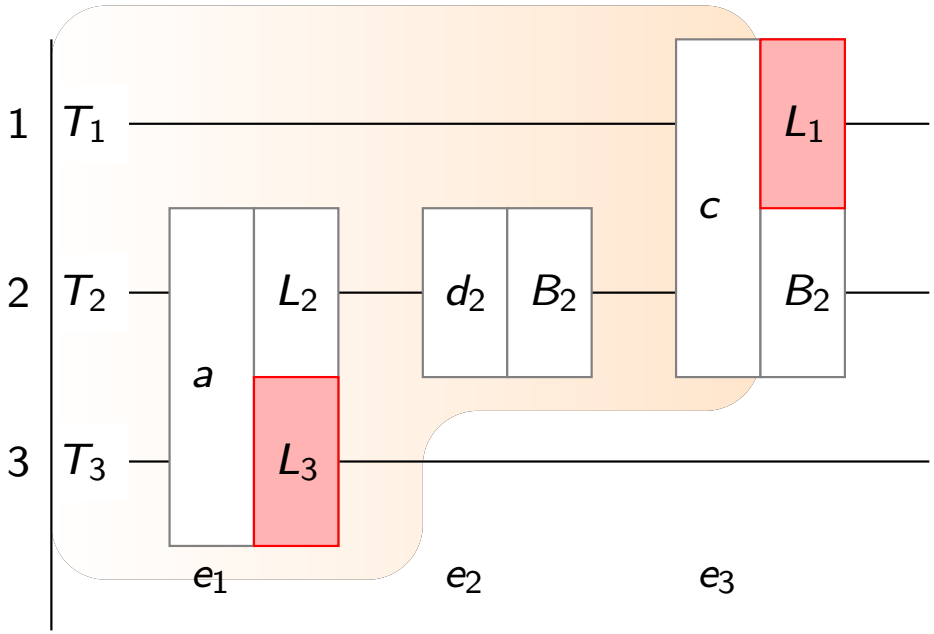
Process 2 remembers if the environment chose Left: a or Right: b

(non CDM process remembers)

Example: A winning memory-based strategy







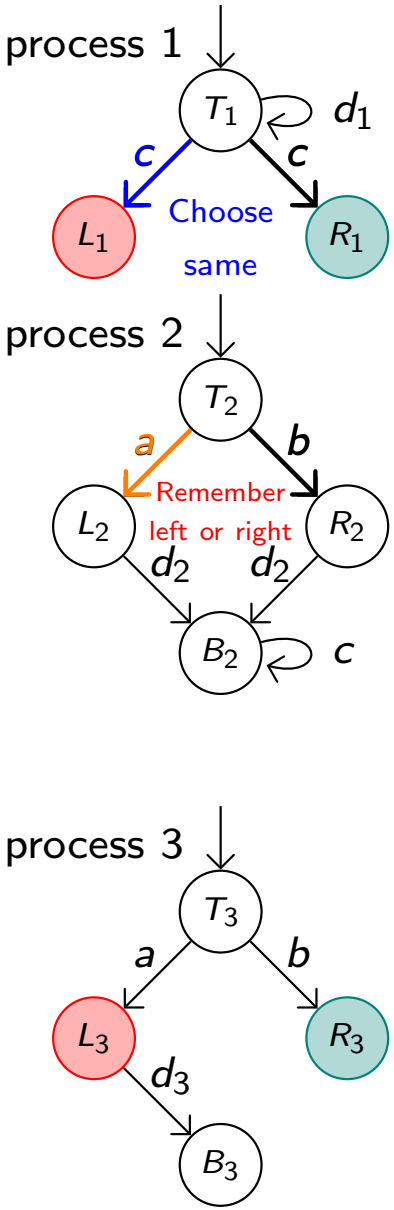
Process 2 remembers if the environment chose Left: a or Right: b

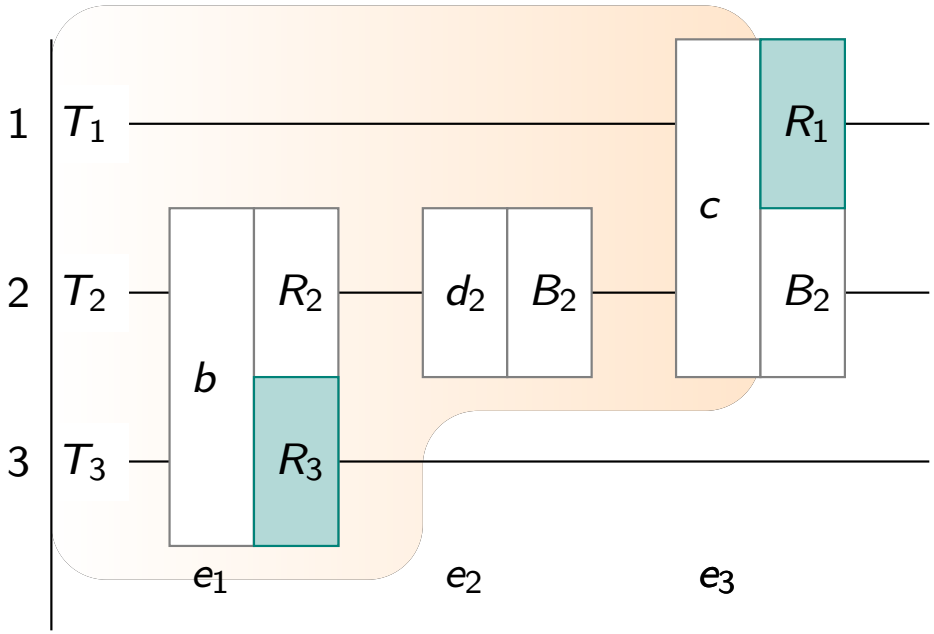
(non CDM process remembers)

Process 1 goes left if the environment chose a, right if b

(CDM process chooses)

Example: A winning memory-based strategy





Process 2 remembers if the environment chose Left: a or Right: b

(non CDM process remembers)

Process 1 goes left if the environment chose a, right if b

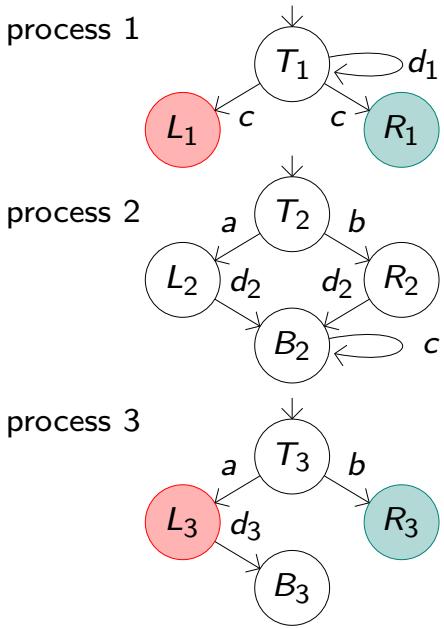
(CDM process chooses)

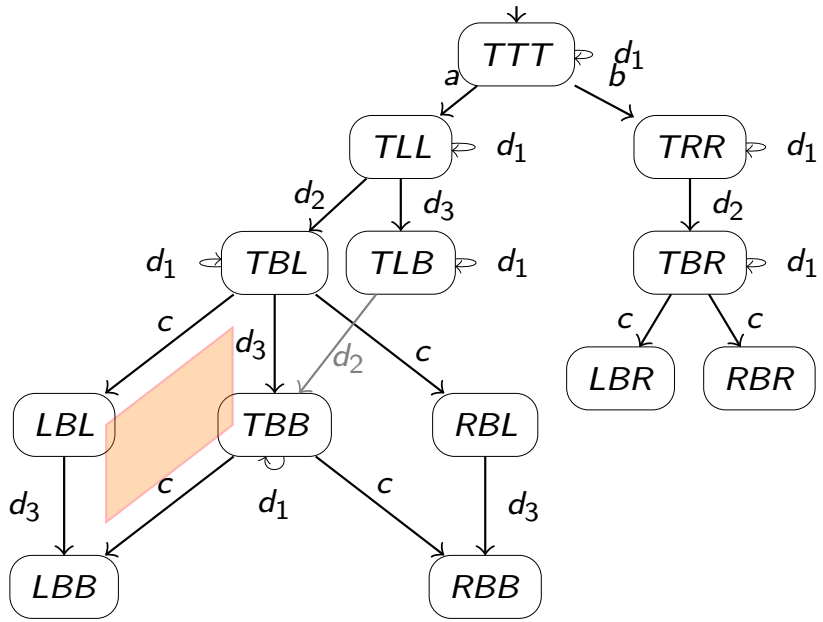
Solution approach for CDM

Solution approach - sequential game

Given a safety CDM game G, construct a sequential safety game G_{seq} on the global-states of G

Derived sequential game G_{seq}

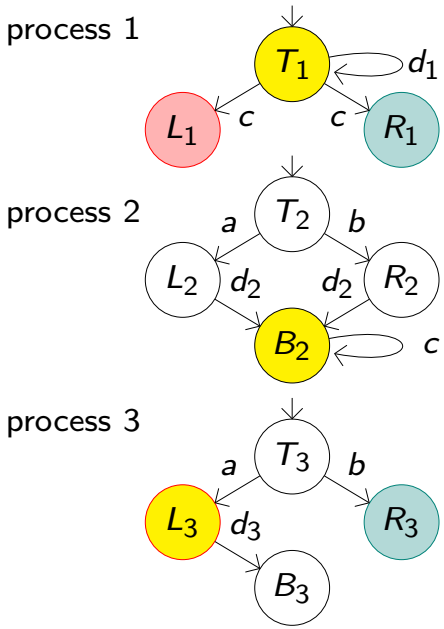


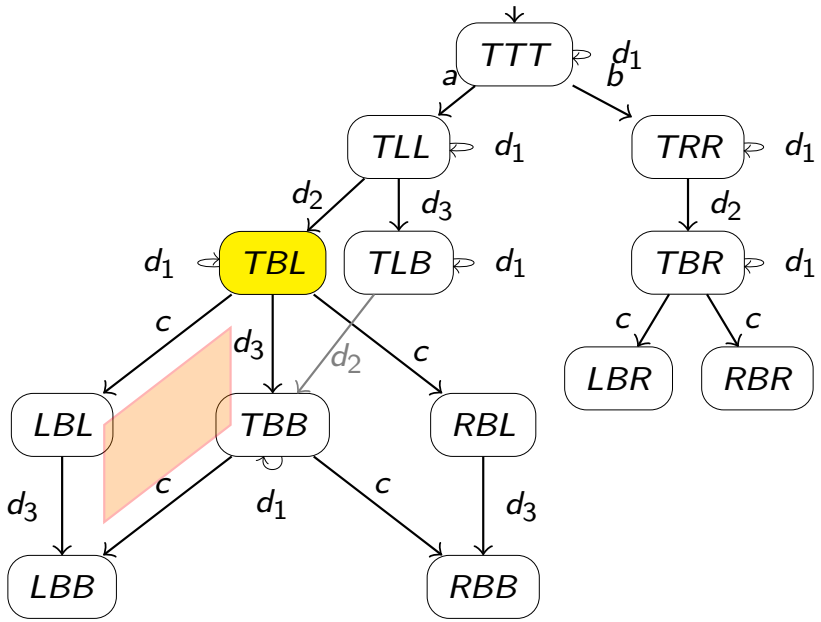


Solution approach - sequential game

Given a safety CDM game G, construct a sequential safety game G_{seq} on the global-states of G

Derived sequential game G_{seq}

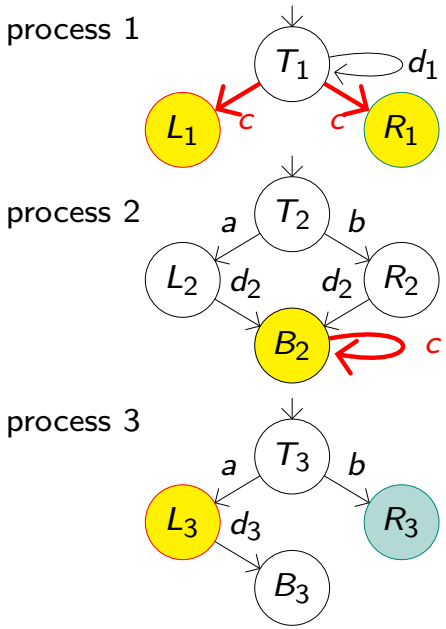


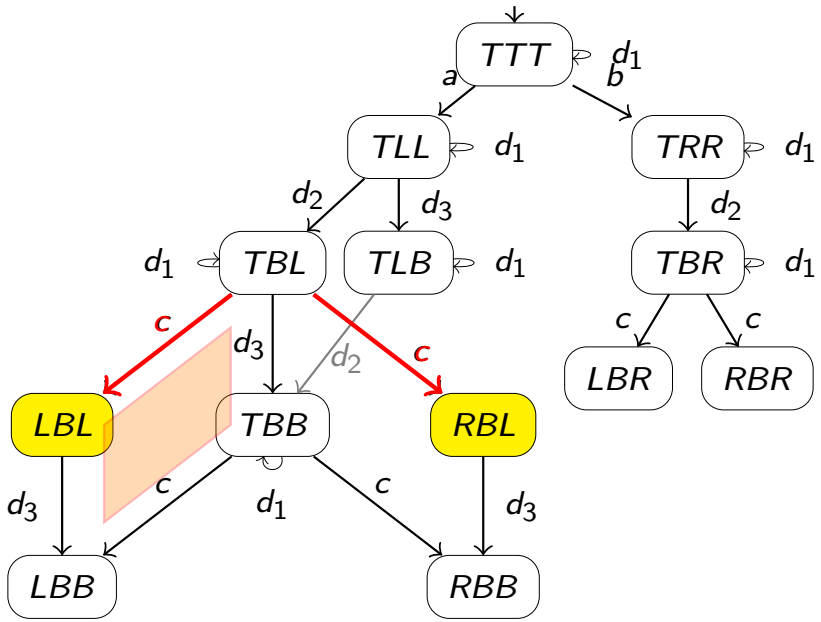


Solution approach - sequential game

Given a safety CDM game G, construct a sequential safety game G_{seq} on the global-states of G

Derived sequential game G_{seq}

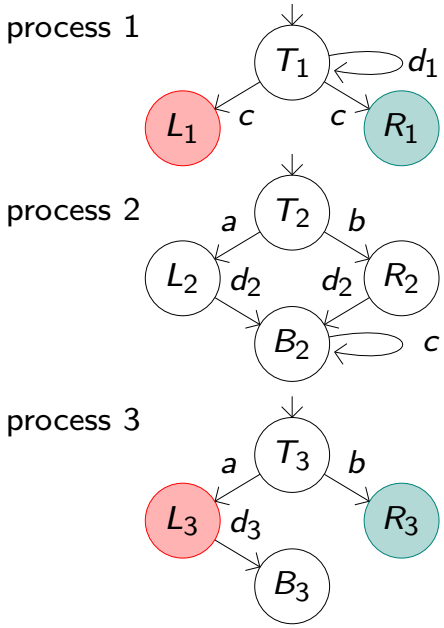


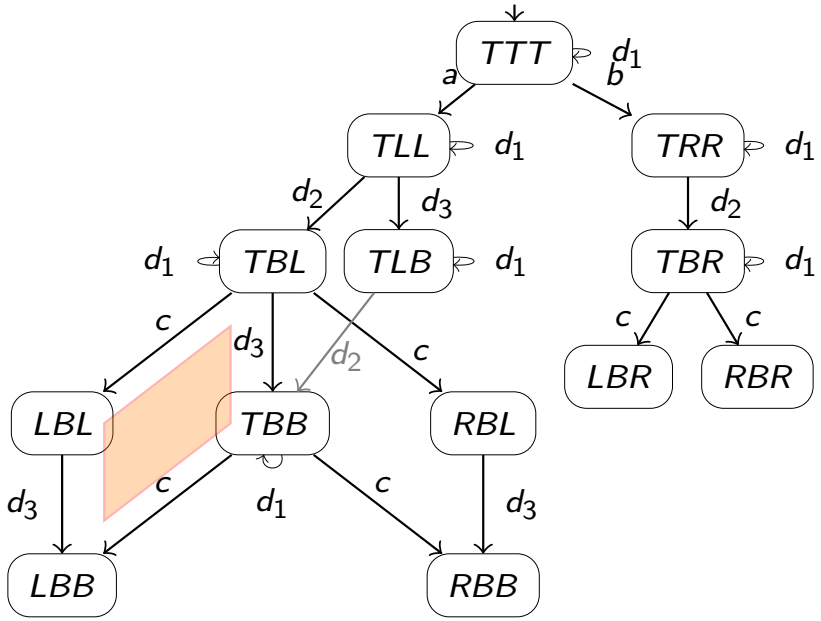


Solution approach - sequential game

Given a safety CDM game G, construct a sequential safety game G_{seq} on the global-states of G

Derived sequential game G_{seq}



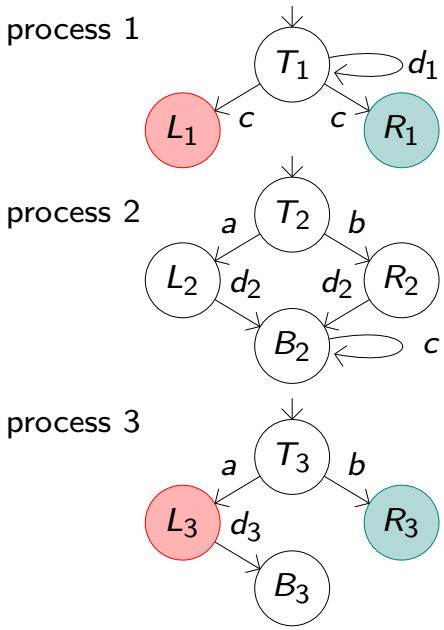


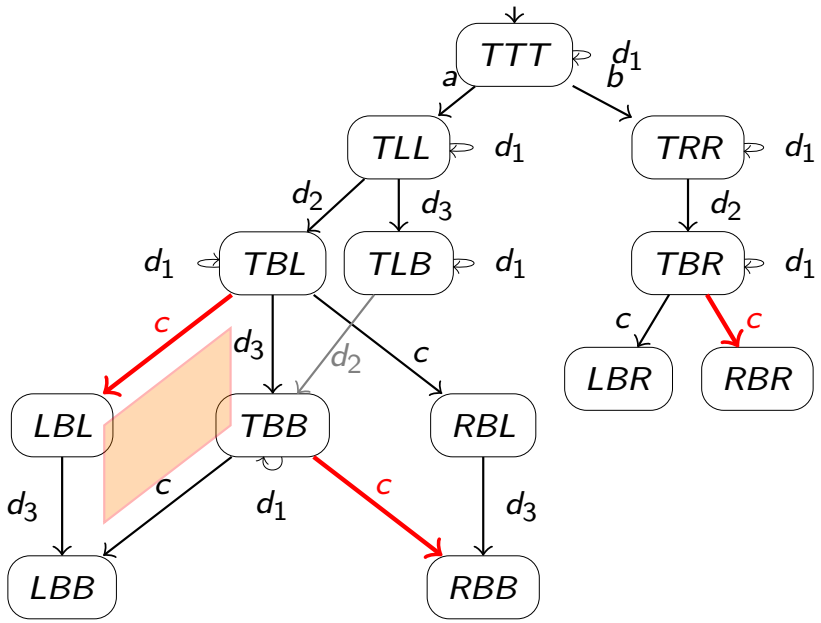
easy: Distributed strategy ——> Sequential strategy

Solution approach - sequential game

Given a safety CDM game G, construct a sequential safety game G_{seq} on the global-states of G

Derived sequential game G_{seq}





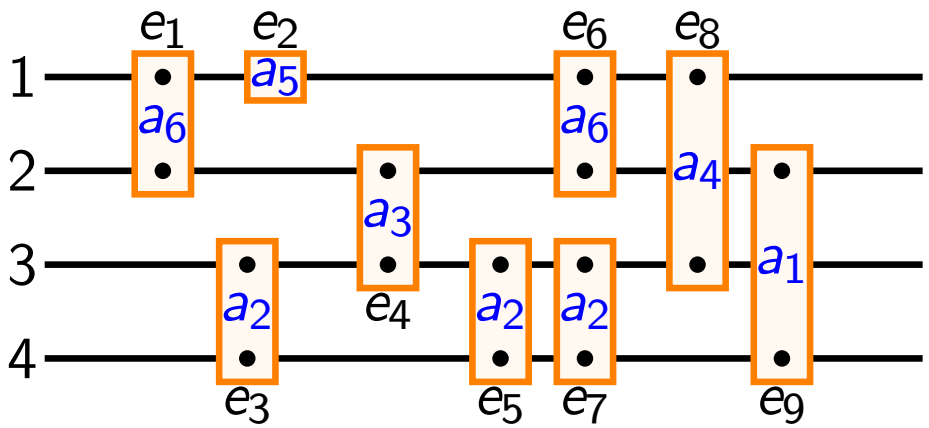
????: Distributed strategy <—— Sequential strategy

Solution approach - Linearization

Given a safety CDM game G, construct a sequential safety game G_{seq} on the global-states of G

There is a distributed winning strategy in G iff there is a sequential winning strategy in G_{seq}.

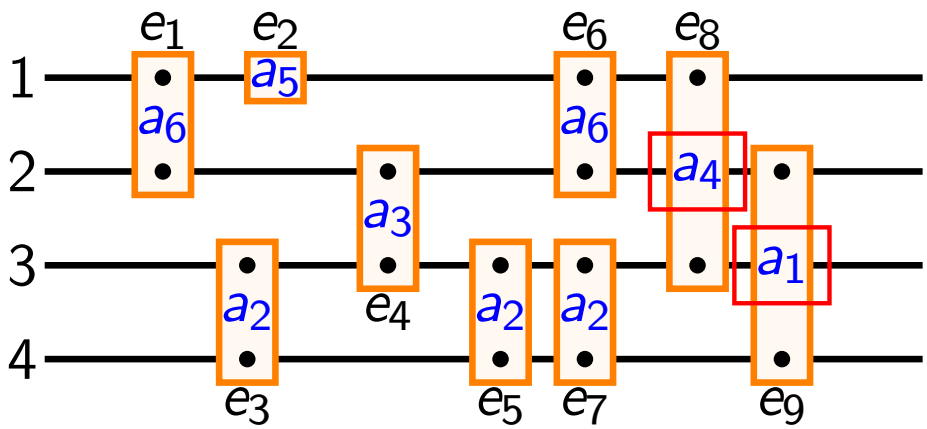
Key idea: given a seq winning strategy in G_{seq}, extract a distributed winning strategy in G. To achieve this, we develop special linearizations of plays/process-diagrams/traces in which cdm events appear the earliest and on any trace we copy the system move on this linearization.



$$e_1\, e_2\, e_3\, e_4\, e_6\, e_5\, e_7\, e_8\, e_9 \;=\; a_6\, a_5\, a_2\, a_3\, a_6\, a_2\, a_2\, a_4\, a_1$$

Special Linearization - Definition

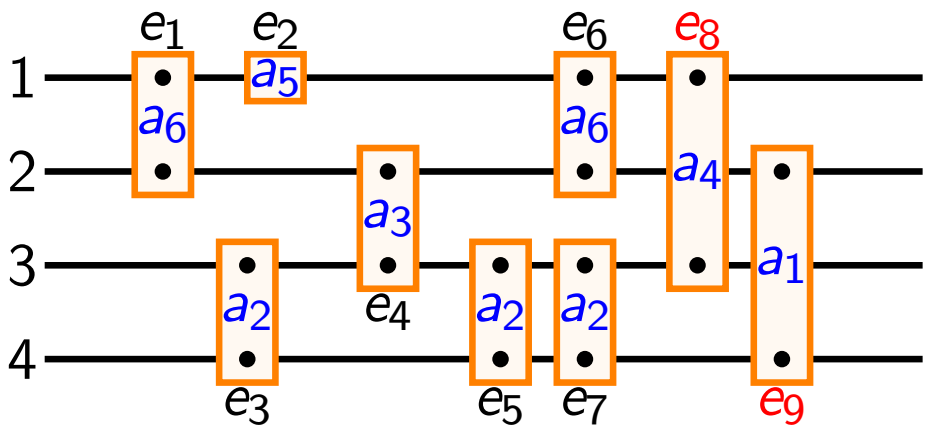
- Fix a total order $\lhd_\Sigma$ on Σ such that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$.



Process 1 participates in a_4 but not in a_1: $a_1 \lhd_\Sigma a_4$

Special Linearization - Definition

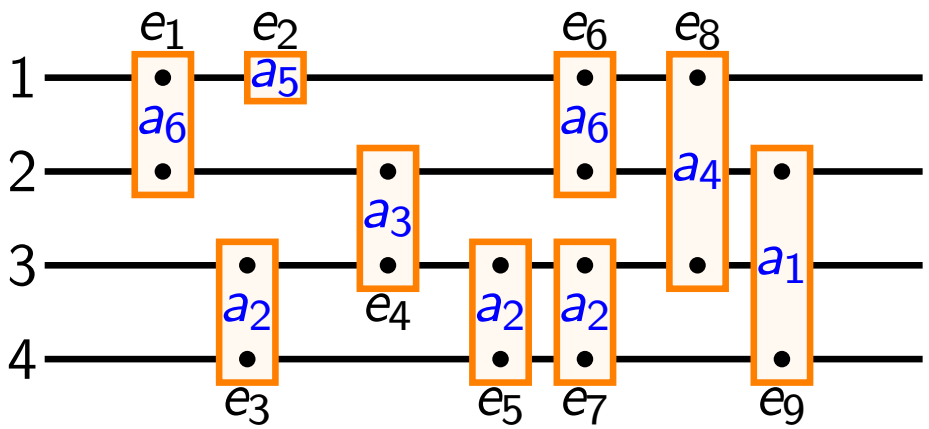
- Fix a total order $\lhd_\Sigma$ on Σ such that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$.
- To compute linearization $\mathrm{Lin}(t)$ of a play t, start from the **maximal** (last) actions:



maximal events: e_8, e_9

Special Linearization - Definition

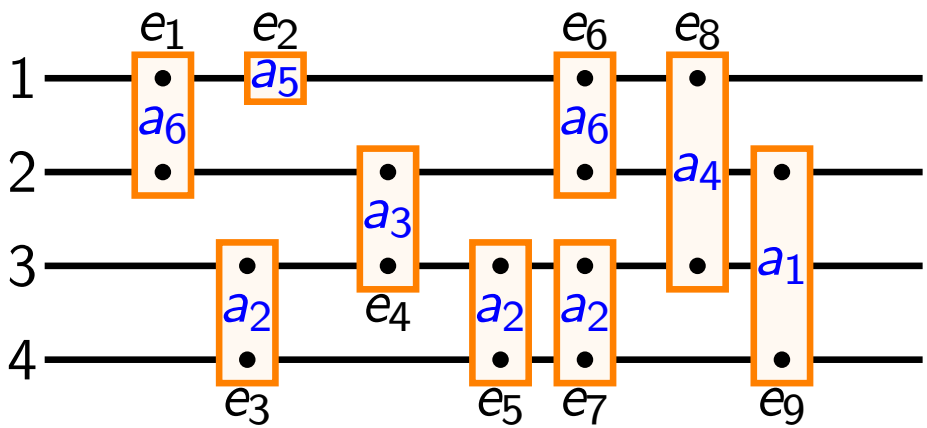
- Fix a total order $\lhd_\Sigma$ on Σ such that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$.
- To compute linearization $\mathrm{Lin}(t)$ of a play t, start from the **maximal** (last) actions:
- Peel off the $\lhd_\Sigma$-**least** event e_9 among the maximal events.



$\lhd_\Sigma$-least among the maximal events e_9

Special Linearization - Definition

- Fix a total order $\lhd_\Sigma$ on Σ such that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$.
- To compute linearization $\mathrm{Lin}(t)$ of a play t, start from the **maximal** (last) actions:
- Peel off the $\lhd_\Sigma$-**least** event e_9 among the maximal events.
- Define the linearization recursively: $\mathrm{Lin}(t) = \mathrm{Lin}(t \setminus e_9)\, \lambda(e_9)$



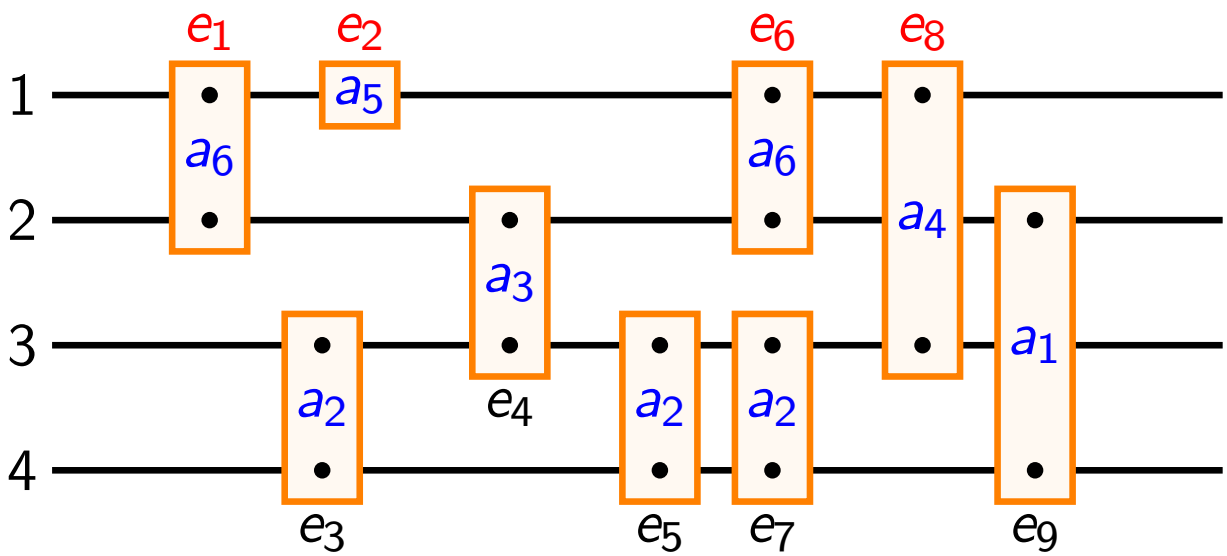
Special Linearization: Property

Resulting property

- Let t be a play and e its event where process 1 participates. Then

$$\mathrm{Lin}(\textbf{causal past of } e) \sqsubseteq \mathrm{Lin}(t)$$

process 1 is the cdm



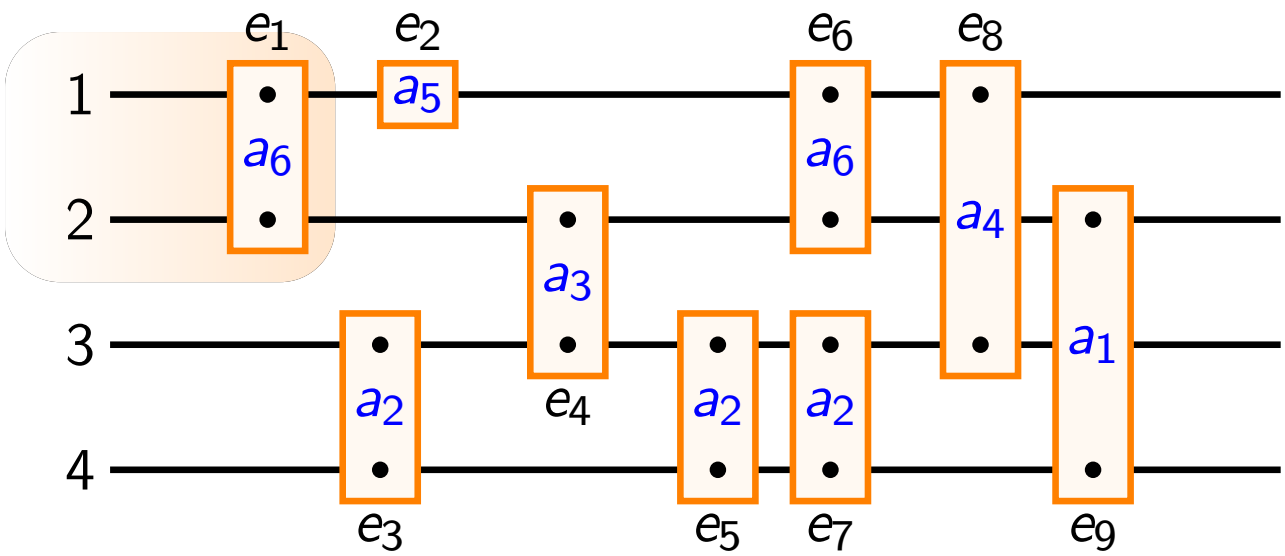
Special Linearization: Property

Resulting property

- Let t be a play and e its event where process 1 participates. Then

$$\mathrm{Lin}(\textbf{causal past of } e) \sqsubseteq \mathrm{Lin}(t)$$

process 1 is the cdm



e_1

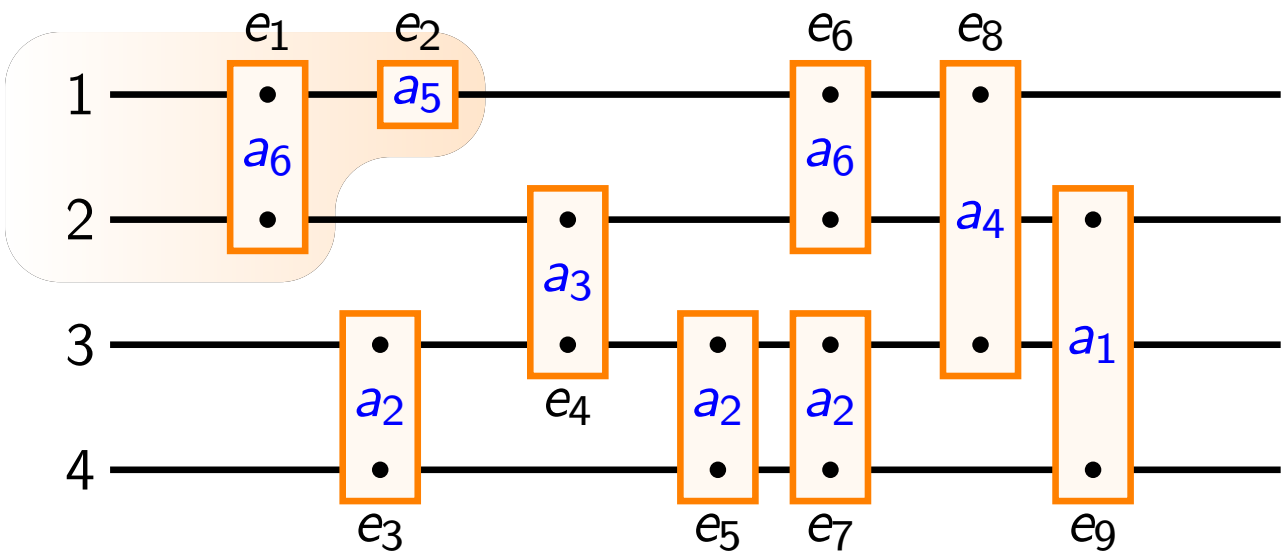
Special Linearization: Property

Resulting property

- Let t be a play and e its event where process 1 participates. Then

$$\mathrm{Lin}(\textbf{causal past of } e) \sqsubseteq \mathrm{Lin}(t)$$

process 1 is the cdm



e_1 e_2

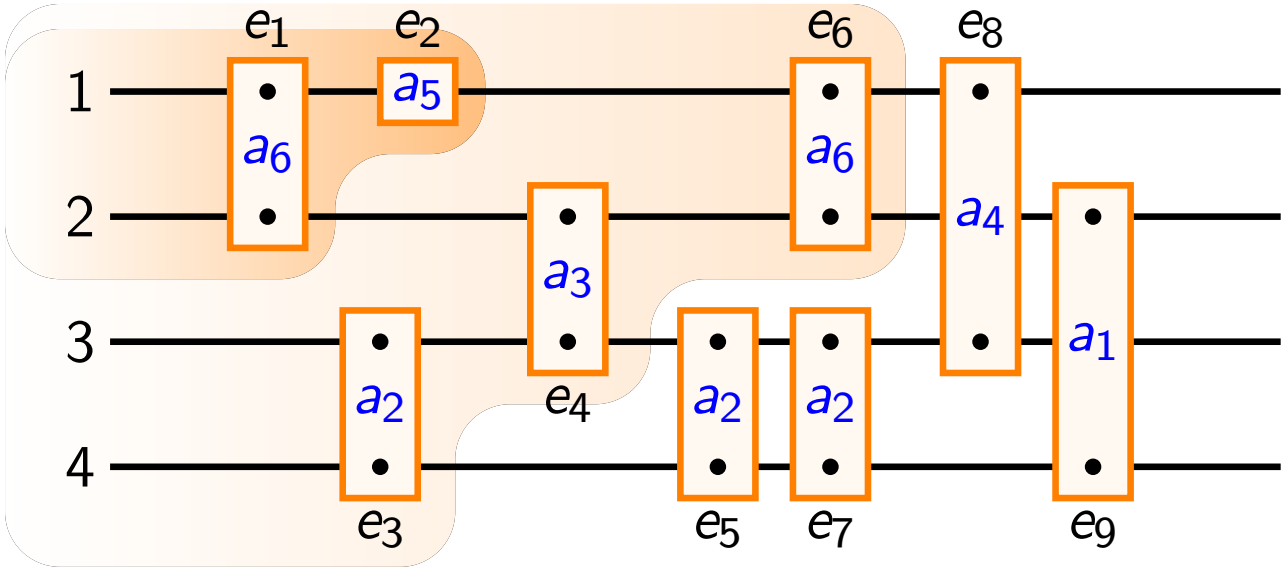
Special Linearization: Property

Resulting property

- Let t be a play and e its event where process 1 participates. Then

$$\mathrm{Lin}(\textbf{causal past of } e) \sqsubseteq \mathrm{Lin}(t)$$

process 1 is the cdm



e_1 e_2 e_3 e_4 e_6

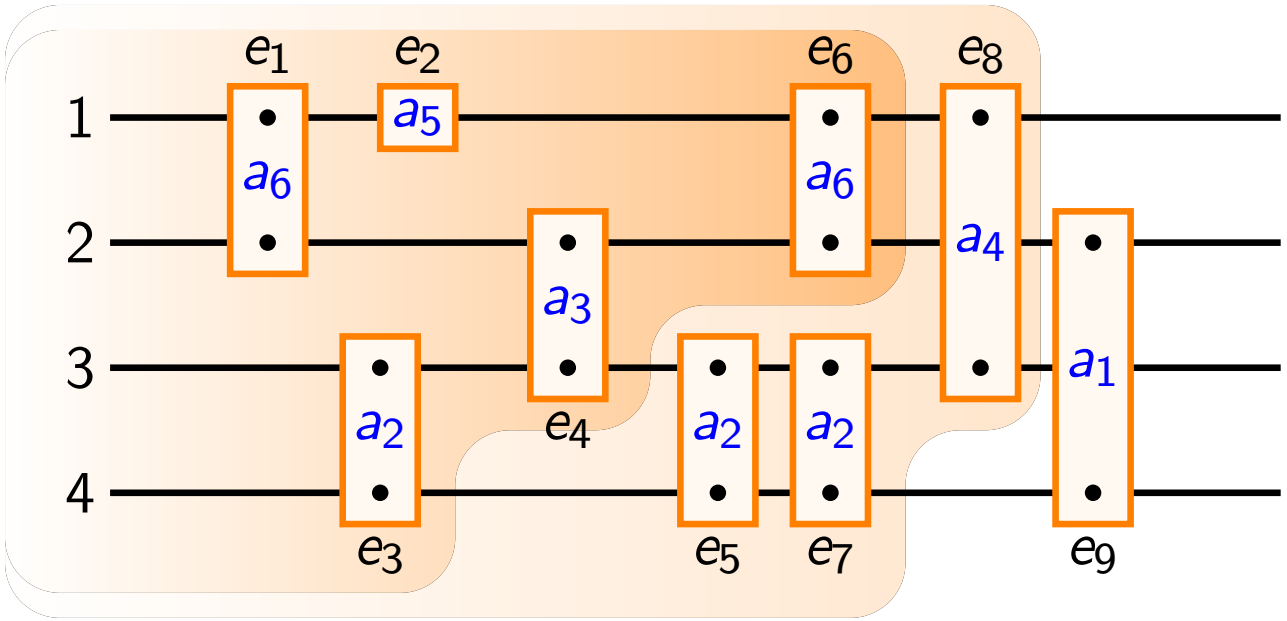
Special Linearization: Property

Resulting property

- Let t be a play and e its event where process 1 participates. Then

$$\mathrm{Lin}(\textbf{causal past of } e) \sqsubseteq \mathrm{Lin}(t)$$

process 1 is the cdm



$e_1\ e_2\ e_3\, e_4\, e_6\ e_5\, e_7\, e_8$

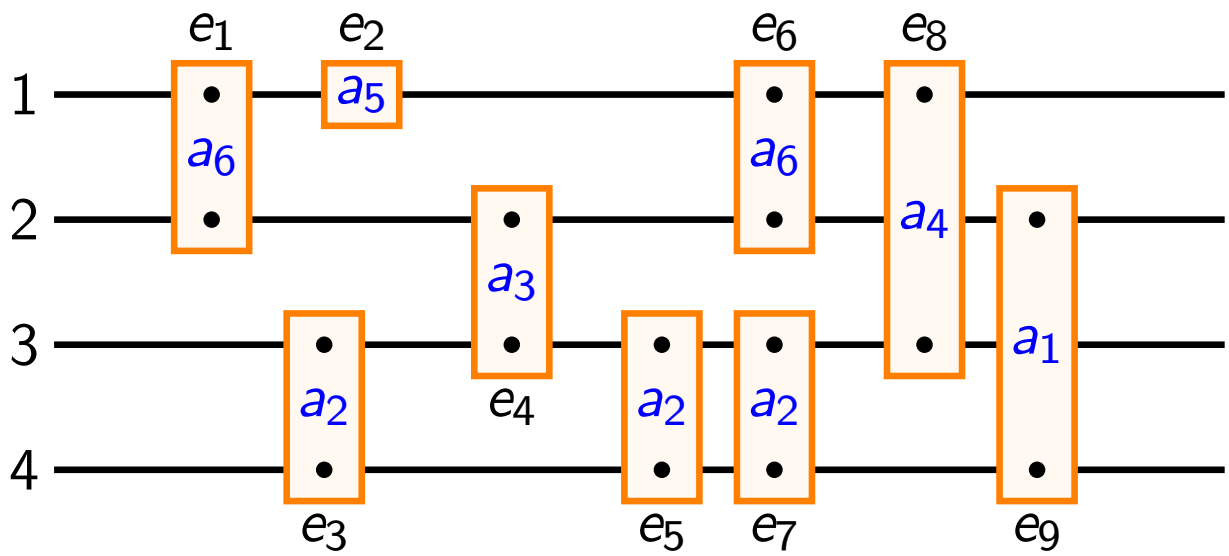
Special Linearization: Property

Resulting property

- Let t be a play and e its event where process 1 participates. Then

$$\mathrm{Lin}(\textbf{causal past of } e) \sqsubseteq \mathrm{Lin}(t)$$

process 1 is the cdm



$e_1\ e_2\ e_3 e_4 e_6\ e_5 e_7 e_8\ e_9$

Distributed strategy and Why this works

Given τ in G_{seq} we define $\hat{\tau}$ in G at trace t as:

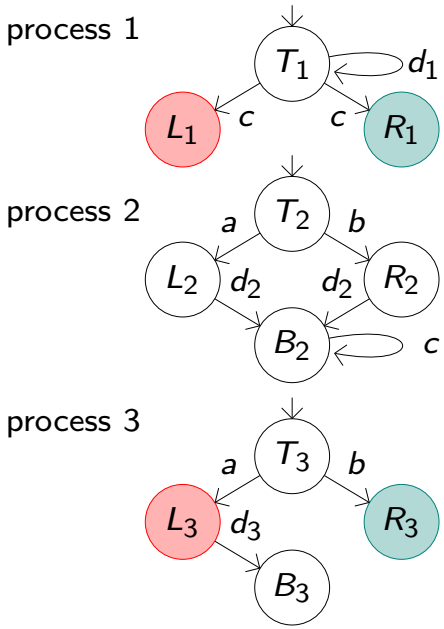
$$\hat{\tau}(t) = \tau(\text{Lin}(t))$$

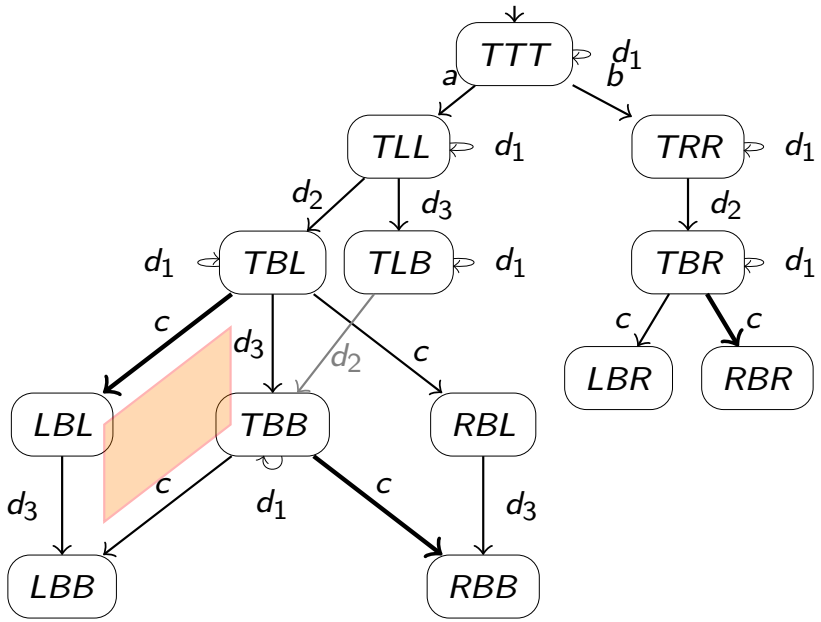
We prove that the system choices still respect the transition system of the distributed game.

- The cdm events appear the earliest, so on cdm events distributed strategy uses linearization of the causal past that is actually available.
- The CDM choices on each CDM event get extended as the plays continues
- When a trace gets extended by a
  - deterministic action: no matter what information the sequential game and distributed game offer the same unique next move is available
  - non deterministic action: the choice made by the distributed strategy copied from sequential strategy remains valid even after some concurrent actions have been played.

An illustration: When a trace gets extended by a non deterministic action

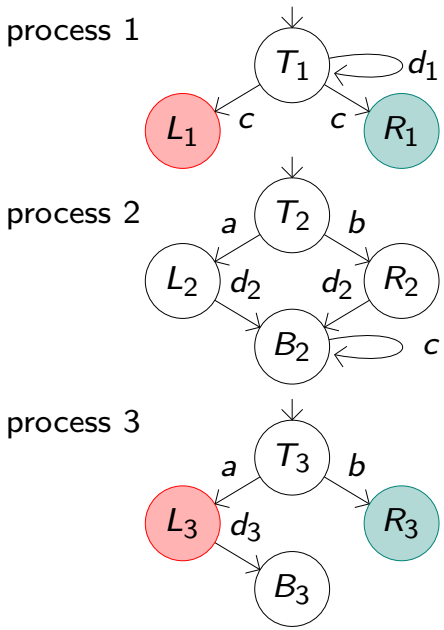
Derived sequential game G_{seq}

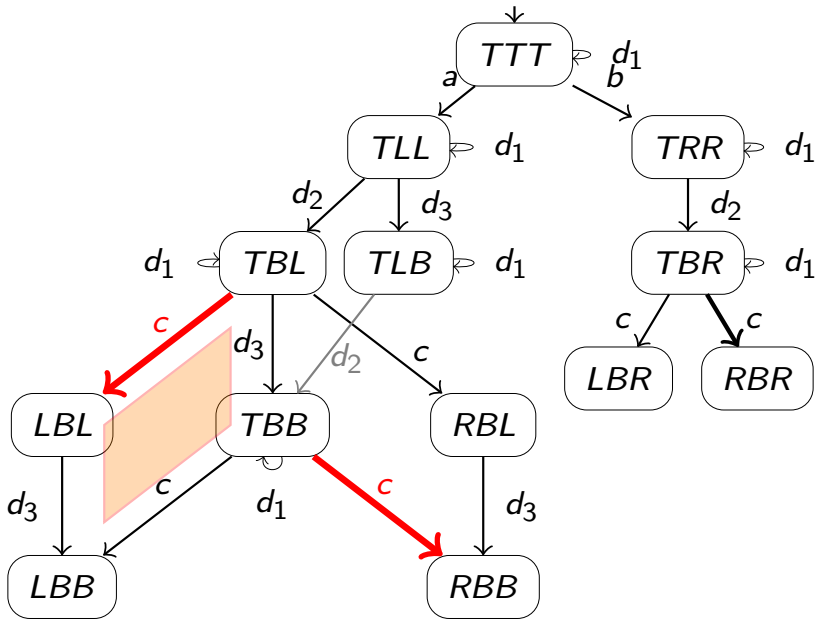




An illustration: When a trace gets extended by a non deterministic action

Derived sequential game G_{seq}

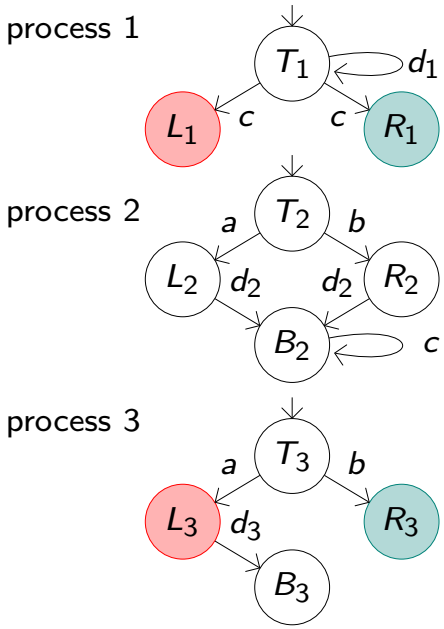


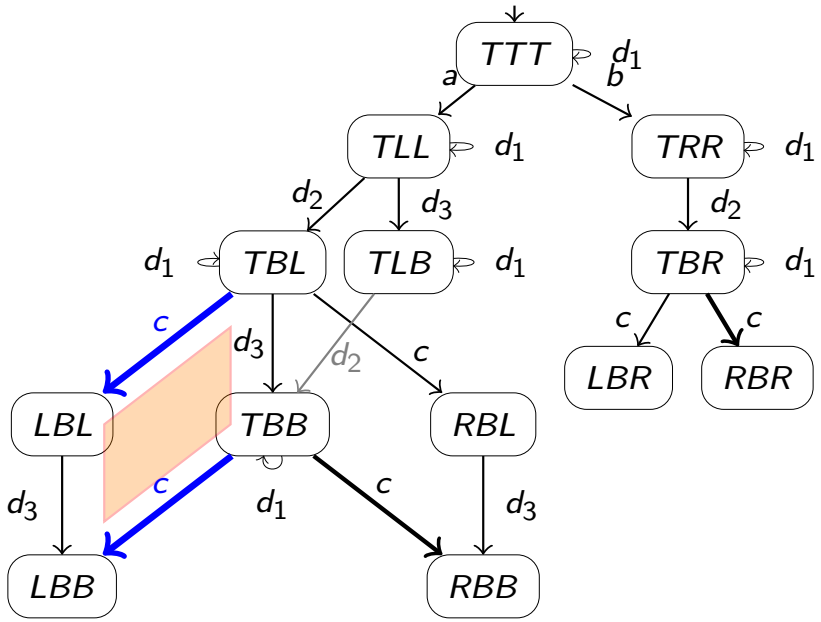


Sequential strategy

An illustration: When a trace gets extended by a non deterministic action

Derived sequential game G_{seq}





Derived distributed strategy: On non deterministic action c the choice made at state TBL remains valid even after concurrent action d_3 have been played.

Implementation as a finite-state distributed strategy

Finite-state distributed strategies

Let us fix a sequential zero-memory strategy τ in G_{seq}. What is the distributed memory required by the extracted distributed strategy $\hat{\tau}$ in G?

When cdm needs to respond, it computes the effect of τ on the special linearization of its strict causal past. As τ is zero-memory, this amounts to the computation of the best global-state that the cdm is aware of.

So, every process keeps track of the best global-state that they are aware of. In order to update this information on a synchronization, they need to decide, for every other process k, which of the synchronizing process has the latest information about process k.

Finite-state distributed strategies

Let us fix a sequential zero-memory strategy τ in G_{seq}. What is the distributed memory required by the extracted distributed strategy $\hat{\tau}$ in G?

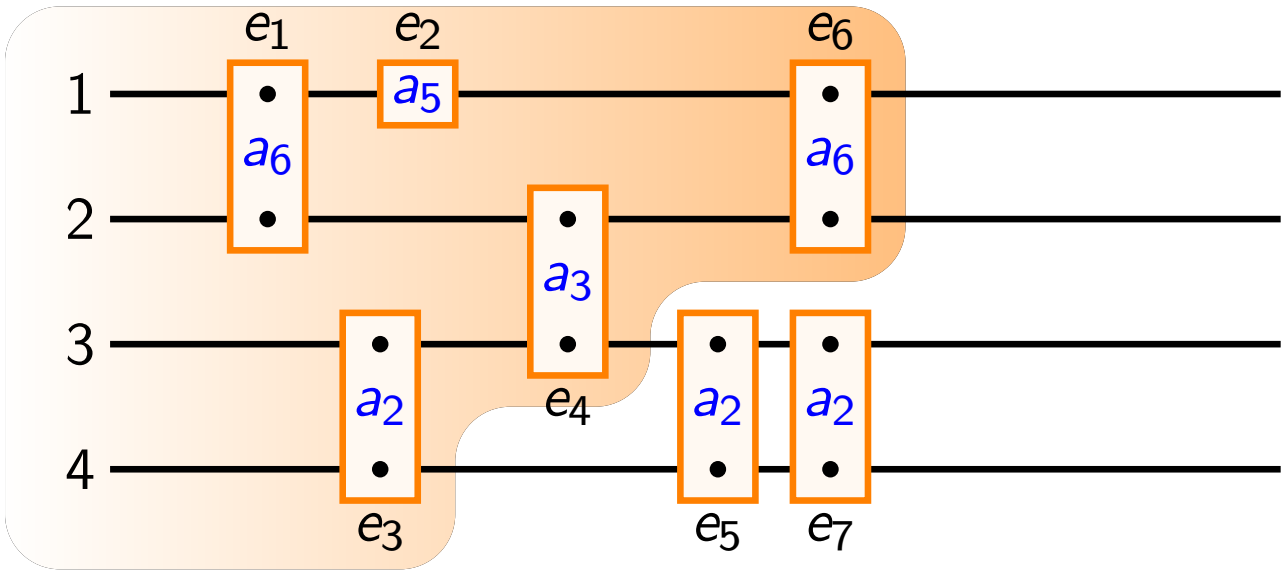
When cdm needs to respond, it computes the effect of τ on the special linearization of its strict causal past. As τ is zero-memory, this amounts to the computation of the best global-state that the cdm is aware of.

So, every process keeps track of the best global-state that they are aware of. In order to update this information on a synchronization, they need to decide, for every other process k, which of the synchronizing process has the latest information about process k.

The gossip asynchronous automaton (Mukund-Sohoni)[MS97] essentially solves the same problem and can be used to update the latest global-state

Computing the latest global state

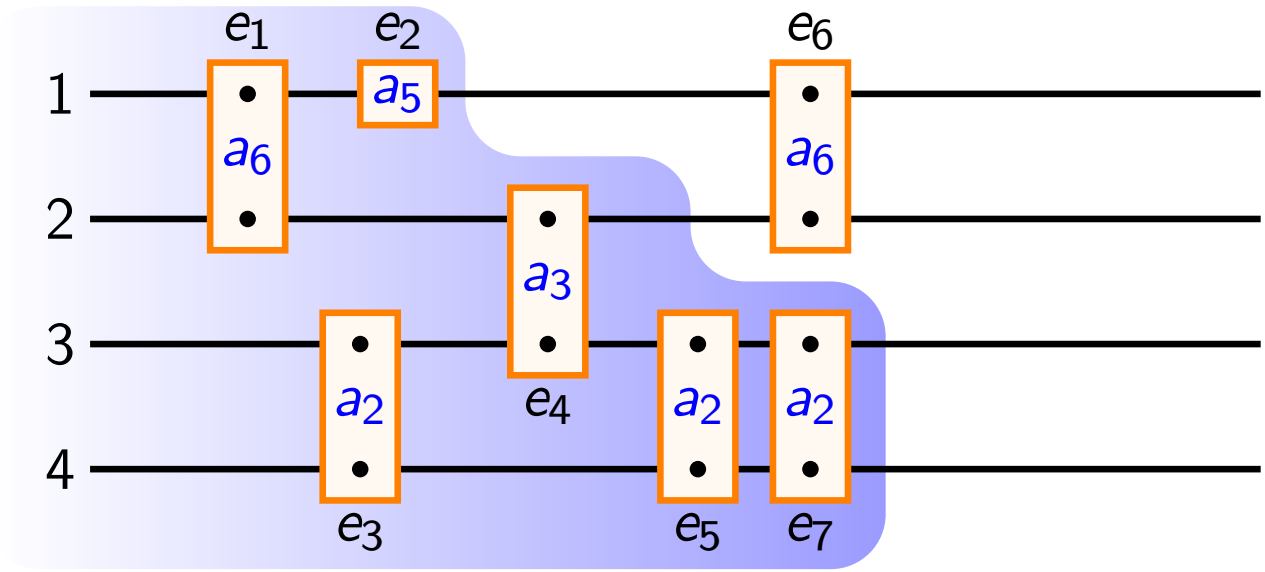
process 1 is the cdm



In an ongoing play
Causal past of process 1
Process 1 tracks global state here

Computing the latest global state

process 1 is the cdm

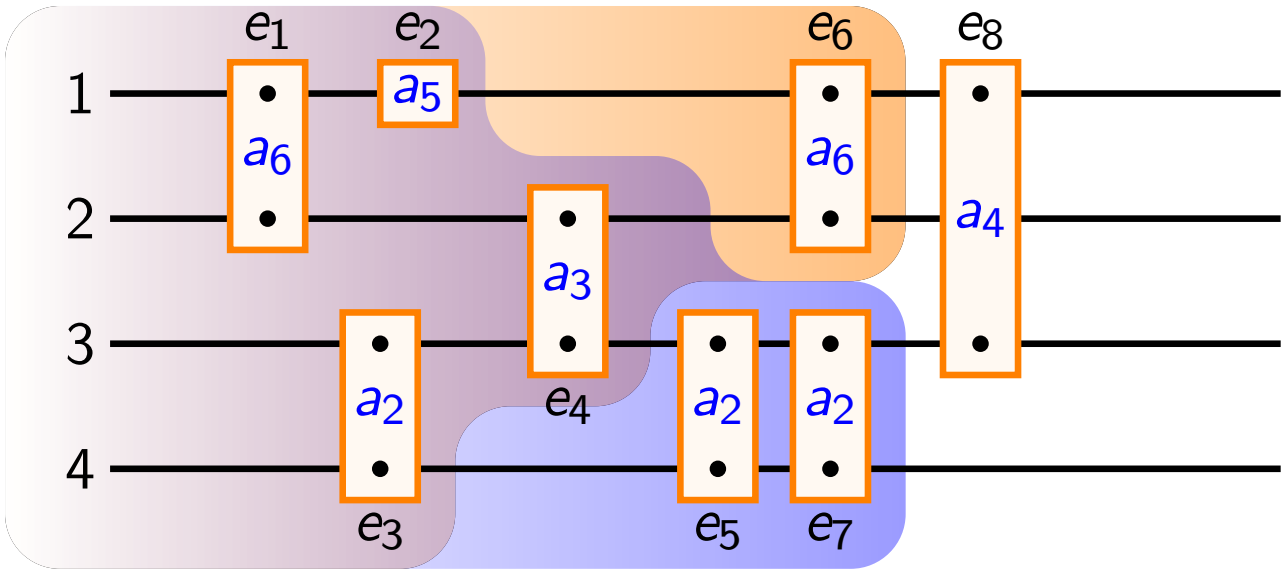


Causal past of process 3

Process 3 tracks global state here

Computing the latest global state

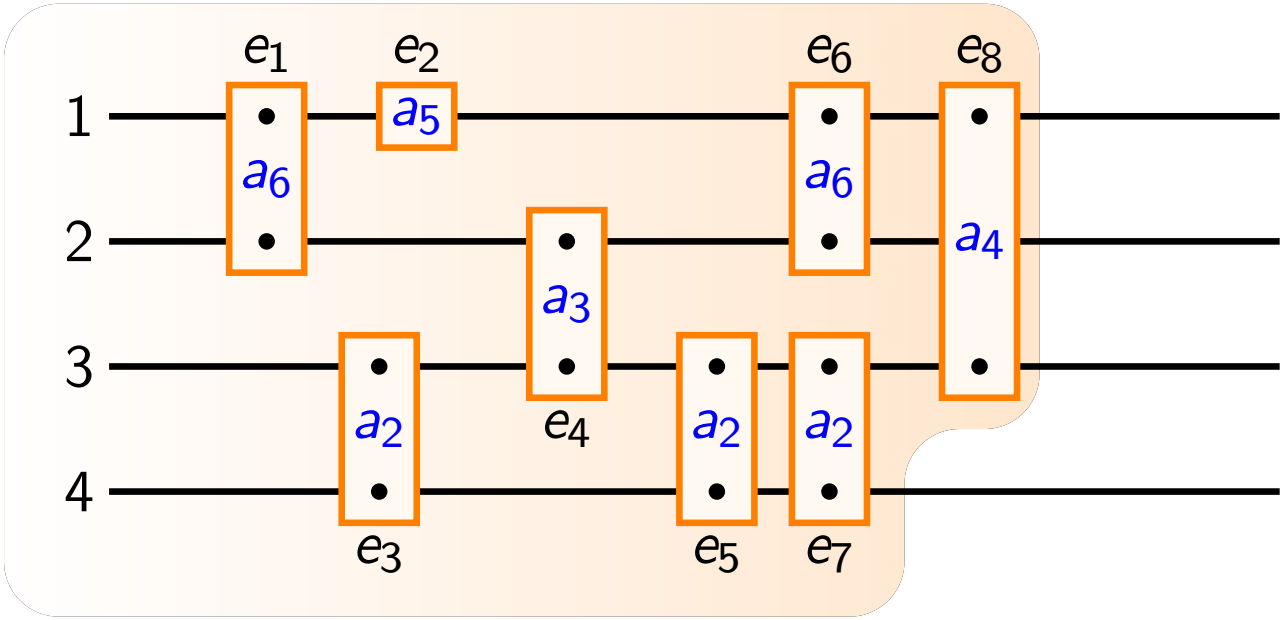
process 1 is the cdm



Who knows better about process 2: process 1(e_6)
Who knows better about process 4: process 3(e_7)

Computing the latest global state

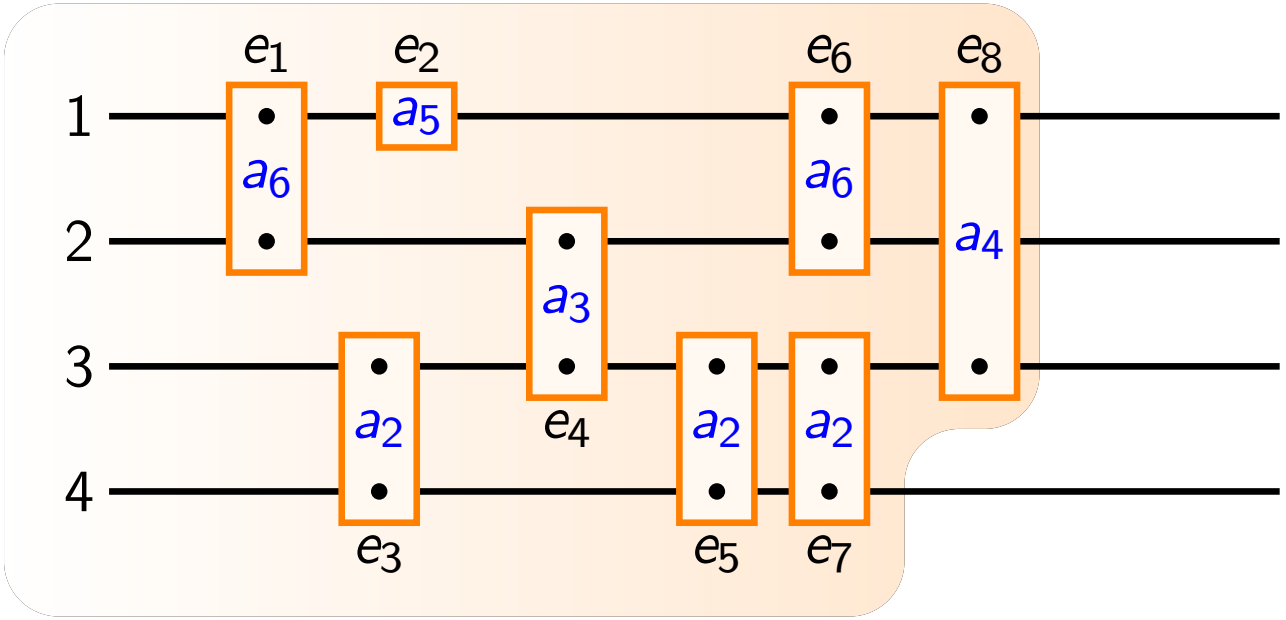
process 1 is the cdm



The gossip asynchronous automaton (Mukund-Sohoni)[MS97] essentially solves this problem and can be used to update the latest global-state

Computing the latest global state

process 1 is the cdm



Thus, Process 1 and 3 update to the next global state they keep track of

Safety CDM games results

Decision complexity

Safety CDM games are EXPTIME-complete

Memory complexity

For a YES-instance, there exists a finite-state distributed winning strategy wherein each process essentially keeps track of the best global-state that it is aware of.

Suppose each process has atmost m local states and there are n processes. So, overall there are m^n global-states and each process needs exponential (in n) many local memory-states.

We show that this exponential dependence on the number of processes is necessary.

The Decidability Frontier

- **Central Decision Maker (CDM or 1DM):**
  - Actions concurrent to the DM are **deterministic**.
  - Problem is Decidable (EXPTIME-complete).
- **2-Decision Maker (2DM):**
  - Two decision makers make choices **concurrently**.
  - Determining a winning strategy is undecidable.

2DM undecidability: builds over (Gimbert, 2022) [Gim22]

Determining a winning strategy for 2DM ATS games is undecidable, even for basic safety conditions.

Insight: Adapting to Two Decision Makers

Background. Existing constructions (Gimbert, 2022) [Gim22] shows undecidability for six processes. A straightforward adaptation yields the following:

- **3DM + 3 deterministic processes:** undecidability holds even when the environment schedules at most **two non-deterministic actions per play**.

Insight: Adapting to Two Decision Makers

Background. Existing constructions (Gimbert, 2022) [Gim22] shows undecidability for six processes. A straightforward adaptation yields the following:

- **3DM + 3 deterministic processes:** undecidability holds even when the environment schedules at most **two non-deterministic actions per play**.

Key Question: Can we reduce this to **2 Decision Makers**?

Insight: Adapting to Two Decision Makers

Background. Existing constructions (Gimbert, 2022) [Gim22] shows undecidability for six processes. A straightforward adaptation yields the following:

- **3DM + 3 deterministic processes:** undecidability holds even when the environment schedules at most **two non-deterministic actions per play**.

Key Question: Can we reduce this to **2 Decision Makers**? **Our**

Solution: Oracle Architecture

- **Oracle Pair Strategy:**
  - Introduce **two external Oracle processes**.
  - Each process in a s split into a **pair**.
  - Each member of the pair synchronizes with exactly one Oracle.
- **Outcome:** All non-deterministic choices are funneled through the two Oracles, effectively reducing the system to **2DM**.

Results for the CDM Setting

- Analysis of games with a Central Decision Maker (CDM)
  - **Objectives considered:**
    - ★ Global safety
    - ★ Parity condition on the decision-maker process
  - **Main results:**
    - ★ Decision problem is **EXPTIME-complete**
    - ★ **EXPTIME** lower bound on local memory of a process
- Safety games with two decision makers are undecidable.

Equivalence with Control games

Equivalence: Control Games vs. ATS Games

(State-based conditions)

Control games

- Processes decide their move locally first
- Each enables controllable actions from its causal past
- Information is shared after an action is scheduled.

ATS games

- Environment schedules an action first
- Participating processes learn the full causal past
- They jointly choose their next state
- Decision is made after information is shared

Key insight

- Under state-based conditions,

$$\text{decide} \rightarrow \text{share} \equiv \text{share} \rightarrow \text{decide}$$

Result

- Strategies and winning conditions are preserved

Equivalence under State-Based Conditions

Control: decide $\rightarrow$ share

ATS: share $\rightarrow$ decide

State-based conditions make the order irrelevant.

Conclusion

- **Framework:** ATS games unify classical control and asynchronous models
- **Equivalence:** ATS games $\equiv$ control games (state-based conditions)
- **Two-process ATS:**
  - Decidability for global safety, local reachability, global reachability
  - **Algorithms:** Fixpoint-based computation of winning regions
  - **Strategies:** Tight upper and lower bounds on memory
- **CDM games:** EXPTIME-complete decision problems; optimal strategy memory in global safety and local parity using special linearization
- **2DM Games:** Undecidable even for safety winning condition

Future Work

- **Two-process ATS games:** Extension from safety and reachability to parity objectives
  - **Expectation:** Decidability with structured memory strategies
  - **Techniques:** Reduction to single-process game constructions
- **CDM games:** Extension from local parity to ω-regular winning conditions
  - **Expectation:** Büchi–Landweber–type results, ensuring existence of finite-memory strategies
  - **Techniques:** Extension of existing linearization constructions

References I

- Bernd Finkbeiner and Ernst-Rüdiger Olderog, *Petri games: Synthesis of distributed systems with causal memory*, Information and Computation **253** (2017), 181–203.
- Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz, *Asynchronous games over tree architectures*, Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II (Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, eds.), Lecture Notes in Computer Science, vol. 7966, Springer, 2013, pp. 275–286.
- Hugo Gimbert, *Distributed asynchronous games with causal memory are undecidable*, Logical Methods in Computer Science **Volume 18, Issue 3** (2022).

References II

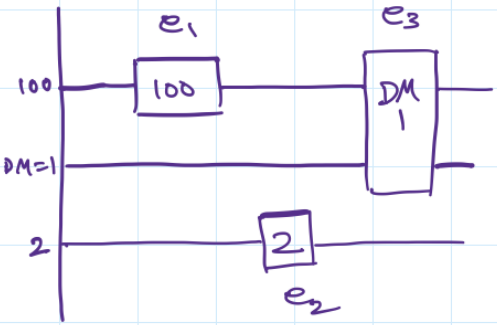

Madhavan Mukund and Milind A. Sohoni, *Keeping track of the latest gossip in a distributed system*, Distributed Computing **10** (1997), no. 3, 137–148.

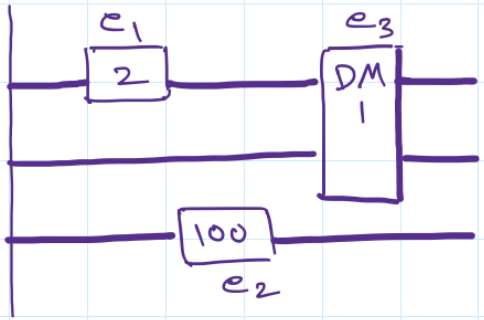
Thank you

Linearization

1. The current definition of special linearization takes a trace and proceed by picking the last event in the linearization (least among the maximal). Can we instead proceed by picking the first event in the linearization (max among the minimal)?

Proceed by picking the first event in the linearization





Σmin among (causal) minimal

LNF = $e_2\, e_1\, e_3$

Desired $e_1\, e_3\, e_2$

Σmax among (causal) minimal

result = $e_2\, e_1\, e_3$

Desired $e_1\, e_3\, e_2$

It is an ordering of events in causal past st the last [causally maximal] CDM event is seen earliest in the linearization.

CDM model generalization

2. Suppose in an ATS game, non-detrministic choice events do not occur concurrently (i.e. they are always sequentially ordered). Can these games also be solved similar to the CDS game ?
Yes

- In fact, one can introduce a pseudo–CDM process that participates in every non-deterministic transition. Since this process does not affect the enabling conditions or the causal structure of the game, its addition does not change the game dynamics; it merely serves to collect the sequentially ordered decisions in a single coordinating process.
- Therefore, a strategy in the augmented game can be lifted to the original game, since the added pseudo-CDM process does not introduce any new information to the processes and does not enable any additional actions.

Other models

3. Would our results carry-over to letter games played on message-passing systems like communicating finite-state machines (CFM) ?
Not directly. Main concerns-

- What is an appropriate notion of a **CFM game**? In particular, what is the role of the environment: does it choose the communicating (sender and receiver) processes, after which the sender selects the message to transmit? (This aspect can be handled by adding message passing processes.)
- If the processes communicate on their own volition (or initiative), what is an appropriate notion of the environment? Does it disallow a set of messages??
- If a winning strategy in an ATS game uses some information from the causal past, is the **message size** in the CFM game sufficient to encode or accommodate the required information?

Lower memory bound

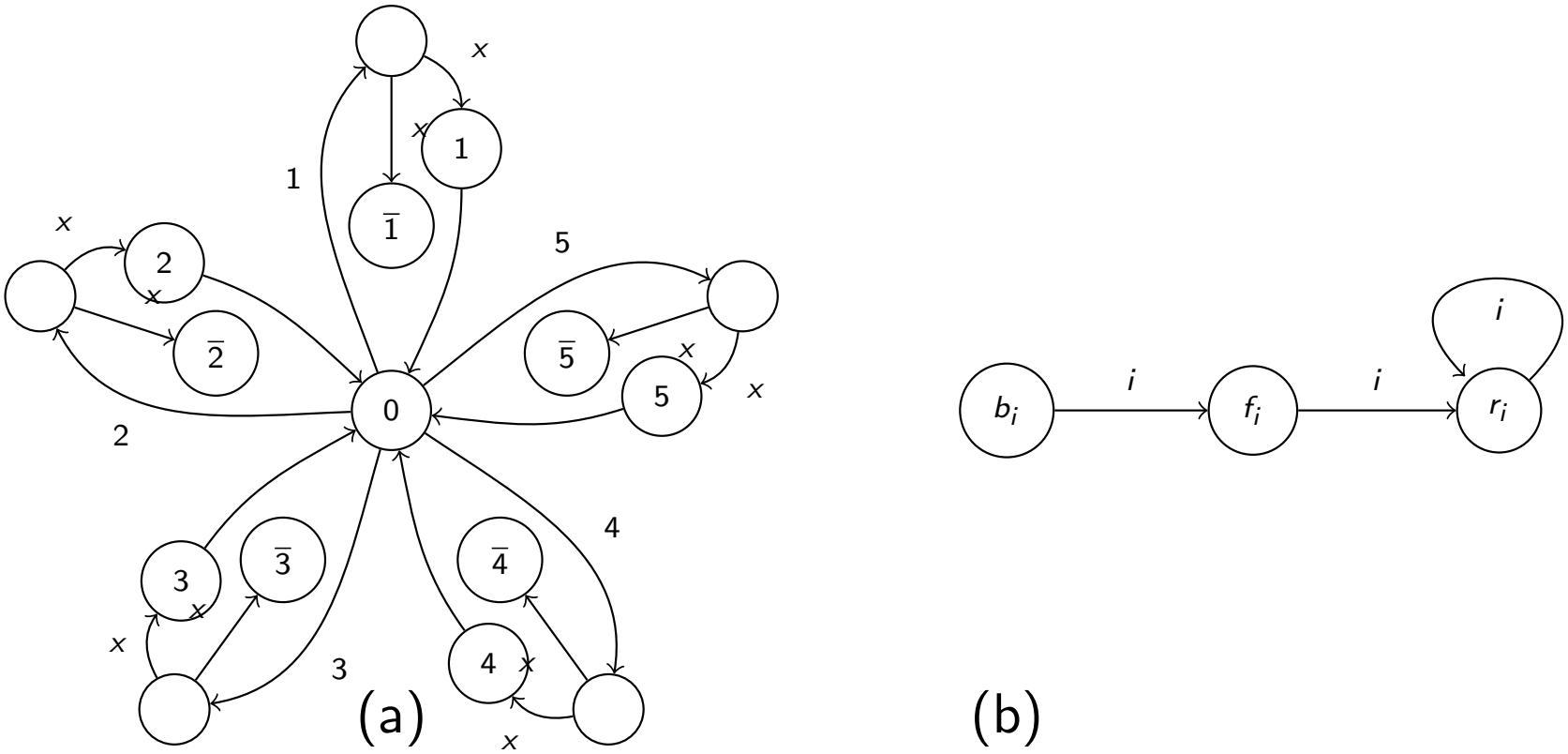
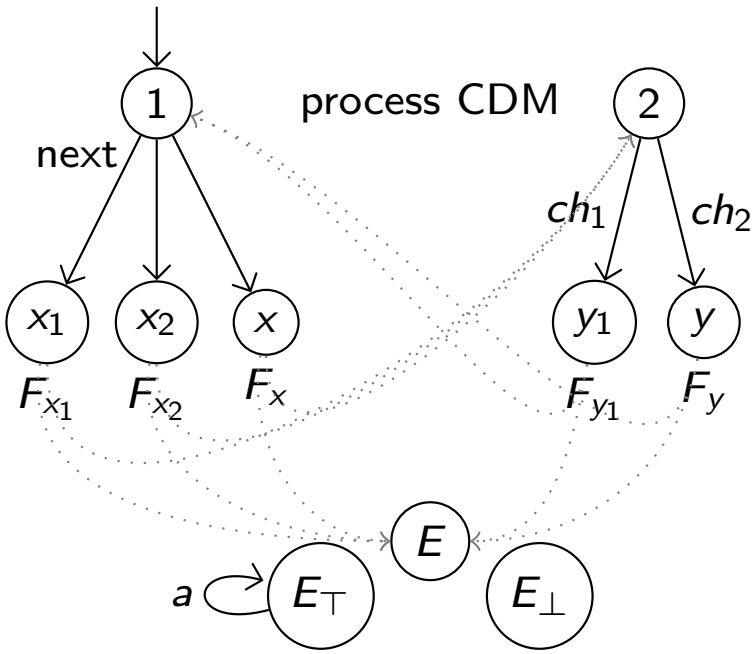
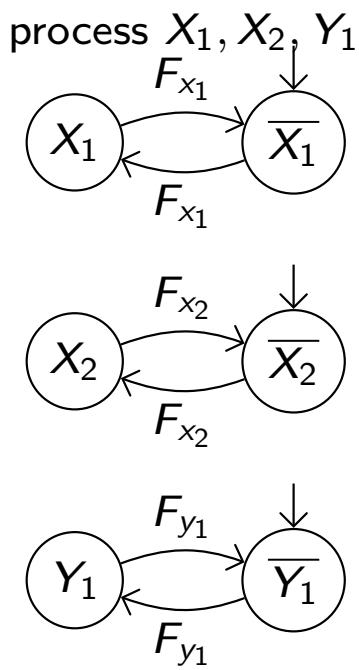


Figure 1: Memory lower bound: The pairing $(\bar{i}, f_i)$ and (i, r_i) in any global state makes it unsafe

EXP lower bound for CDM global safety and local parity

G_5: A position is a triple $(\tau, F(X, Y), \alpha)$ where $\tau \in 1, 2$, F is a formula in CNF whose variables have been partitioned into disjoint sets X, Y and a is n assignment for the set of variables $V(F)$ in F. Player $I(II)$ moves by changing at most one variable in $X(Y)$: passing is allowed. Player I wins if the formula F is ever true. we define game $\overline{G_5}$ when Player I wins if the formula F is never true.





$$(E, x_1, y_1, \vee_0) \xrightarrow{\vee} (E, x_1, y_1, \vee)$$
$$(E, \overline{x_1}, y_1, \vee_0) \xrightarrow{\vee} (E, \overline{x_1}, y_1, \vee)$$
$$(E, x_1, \overline{y_1}, \vee_0) \xrightarrow{\vee} (E, x_1, \overline{y_1}, \vee)$$
$$(E, \overline{x_1}, \overline{y_1}, \vee_0) \xrightarrow{\vee} (E, \overline{x_1}, \overline{y_1}, \overline{\vee})$$

$$(E, \vee, x_2, \wedge_0) \xrightarrow{\wedge} (E_\top, \vee, x_2, \wedge)$$
$$(E, \overline{\vee}, x_2, \wedge_0) \xrightarrow{\wedge} (E_\bot, \overline{\vee}, x_2, \overline{\wedge})$$
$$(E, \vee, \overline{x_2}, \wedge_0) \xrightarrow{\wedge} (E_\bot, \vee, \overline{x_2}, \overline{\wedge})$$
$$(E, \overline{\vee}, \overline{x_2}, \wedge_0) \xrightarrow{\wedge} (E_\bot, \overline{\vee}, \overline{x_2}, \overline{\wedge})$$

Figure 2: *$\overline{G_5}$ reduces to global safety game with local unsafe state $E_\top$ and G_5 reduces to parity game with coloring $\chi(E_\top) = 2$ and if $s_1 \neq E_\top$ then $\chi(s_1) = 1$. Formula is $(x_1 \vee y_1) \wedge x_2$.*