# MAD

## **M**odels & **A**lgorithms for **D**istributed systems

-- 4/5 --

download slides at

http://people.rennes.inria.fr/Eric.Fabre/

# Today…

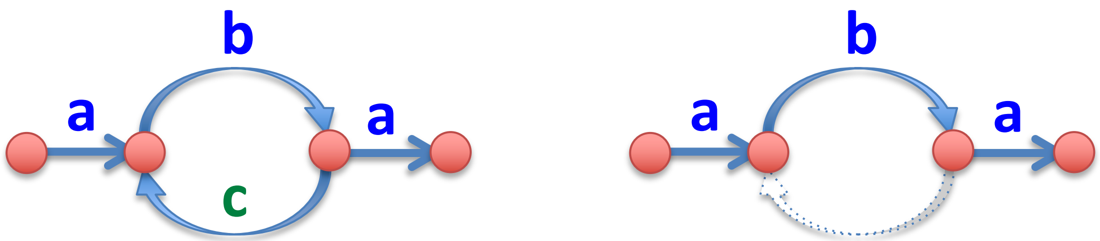
- Playing with networks of automata

- A recap of their algebraic properties
- Extra properties, enabling distributed computations
- Applications
  - distributed diagnosis
  - distributed planning

# What do we have so far ?

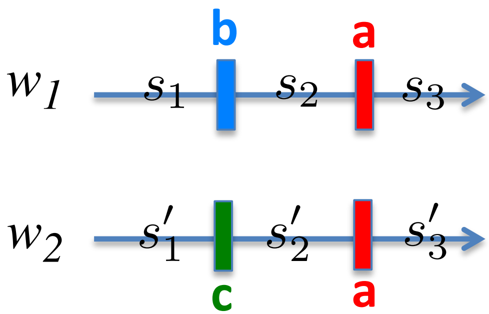
## Projection operators

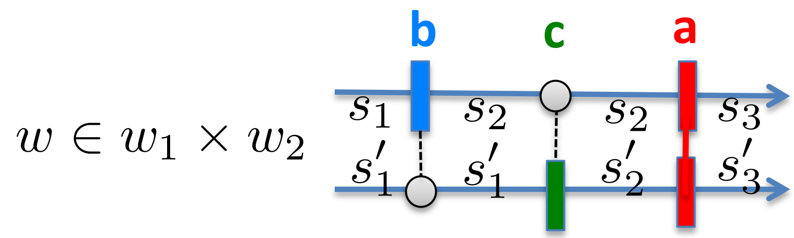
- on words and languages $\Pi_{\{a,b\}}$( **abcbcba** ) = **abbba**
- on automata




**Thm** $$\Pi_{\Sigma'}[\,\mathcal{L}(\mathcal{A})\,] \;=\; \mathcal{L}\,[\,\Pi_{\Sigma'}(\mathcal{A})\,]$$

# **Product operators** on languages and on automata




When 2 words $w_1$, $w_2$ match on common letters, any word $w$ in their product is also a word of the product automaton.

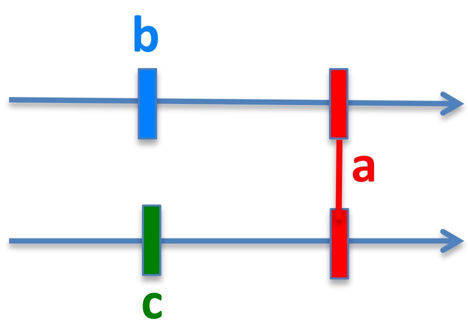
**Thm** $\mathcal{L}(\mathcal{A}_1 \times \ldots \times \mathcal{A}_N) = \mathcal{L}(\mathcal{A}_1) \times \ldots \times \mathcal{L}(\mathcal{A}_N)$

**Consequence :** computations on languages (infinite objects) can be turned into computations on automata (finite objects)

**Traces** = runs seen as partial orders, equivalent to a product of local words
they can thus be encoded/represented as tuples of (local) words

$$[w] \quad = \quad w_1 \times \ldots \times w_N \quad \text{where} \quad w_i \in \Sigma_i^*$$

| | |
|---|---|
| multiple (equivalent) words | single tuple of words |
| concurrent events interleaved | factored form of a trace : more compact |
| partial ordering not visible | partial order easily readable |




**Remark** : $\forall w \in w_1 \times \ldots \times w_N$ one has $\Pi_i(w) = w_i$

# More algebraic properties

## Reduced languages

- a distributed/modular automaton $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$
- one has $\mathcal{L}(\mathcal{A}_i) \subseteq \Sigma_i^*$ and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times ... \times \mathcal{L}(\mathcal{A}_N) \subseteq \Sigma^*$

- by definition, one has $\mathcal{L}'_i = \Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$
- these words represent behaviors of $A_i$ that remain possible once this component is connected to the rest of the system

# More algebraic properties

## Reduced languages

- a distributed/modular automaton $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$
- one has $\mathcal{L}(\mathcal{A}_i) \subseteq \Sigma_i^*$ and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times ... \times \mathcal{L}(\mathcal{A}_N) \subseteq \Sigma^*$
- by definition, one has $\mathcal{L}'_i = \Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$
- these words represent behaviors of $A_i$ that remain possible once this component is connected to the rest of the system

**Thm**

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times ... \times \mathcal{L}(\mathcal{A}_N)$$
$$= \mathcal{L}'_1 \times ... \times \mathcal{L}'_N$$

← *minimal factored form*

Proof : $\supseteq$ is obvious, so only $\subseteq$ must be proved
any word $w \in \mathcal{L}(\mathcal{A})$ satisfies $w \in w_1 \times ... \times w_N$
for some $w_i \in \mathcal{L}(\mathcal{A}_i)$
and one has $w_i = \Pi_i(w)$ so $w_i \in \mathcal{L}'_i$

## Example

$L_1$ = {abb, ababa, baba}   $L_2$ = {cc, cac, aca}

$L'_1$ = {abb, baba}   $L'_2$ = {cac, aca}

$L = L_1 \times L_2 = L'_1 \times L'_2 =$ { cabbc, cabcb, cacbb, babca, bacba }

## Objective

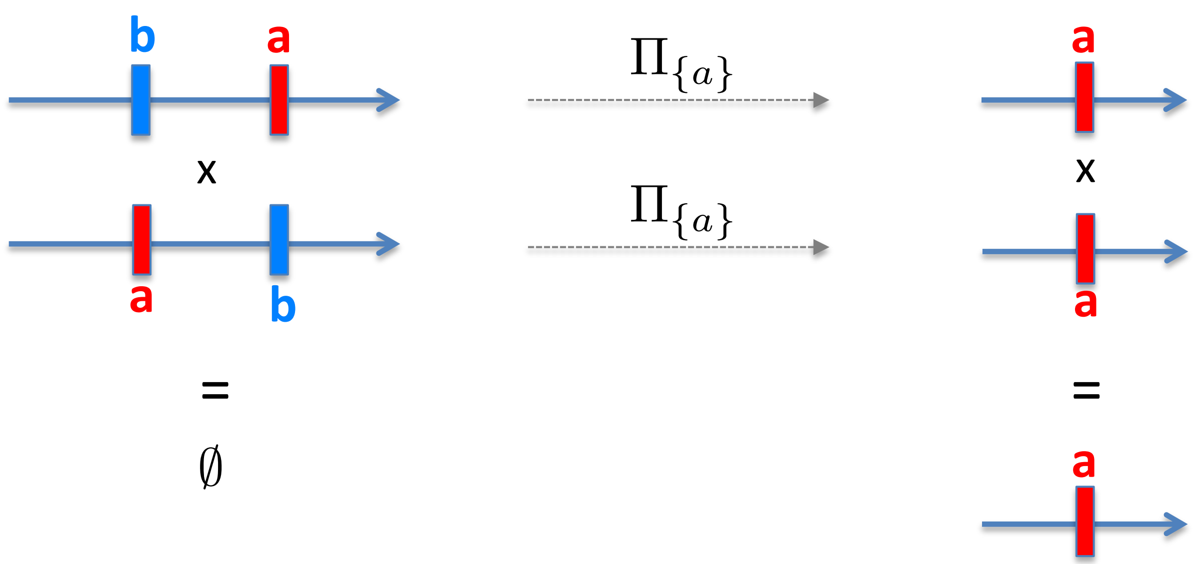
- given the distributed automaton $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$
- we want to compute the reduced languages $\mathcal{L}'_i = \Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$
- without computing $\mathcal{A}$ nor $\mathcal{L}(\mathcal{A})$ which are huge objects
- **Interest**
  - check system design more easily (deadlocks/liveness, reachability, safety…)
  - eliminate spurious behaviors, debugging
  - select runs that match some property (e.g. use in diagnosis and planning)

# A central property

**Thm** let $\mathcal{L}_i \subseteq \Sigma_i^*$ , i=1,2, let $\Sigma' \subseteq \Sigma$
if $\Sigma' \supseteq \Sigma_1 \cap \Sigma_2$ then

$$\Pi_{\Sigma'}(\mathcal{L}_1 \times \mathcal{L}_2) = \Pi_{\Sigma'}(\mathcal{L}_1) \times \Pi_{\Sigma'}(\mathcal{L}_2)$$

- **Proof** : exercise, by double inclusion ; assume first that $\Sigma' = \Sigma_1 \cap \Sigma_2$
- **necessity of** $\Sigma' \supseteq \Sigma_1 \cap \Sigma_2$



# Consequence 1

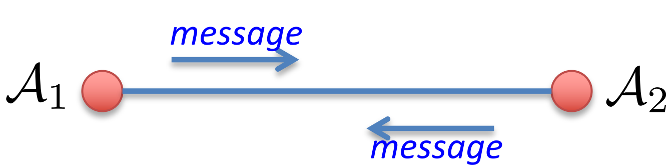
**Cor** : take $\Sigma' = \Sigma_1 \supseteq \Sigma_1 \cap \Sigma_2$, one has

$$\Pi_{\Sigma_1}(\mathcal{L}_1 \times \mathcal{L}_2) \;=\; \mathcal{L}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{L}_2)$$

**Consequence**

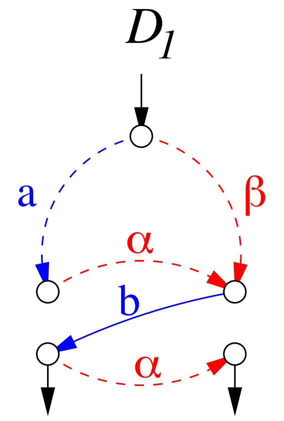
$$\begin{aligned}\mathcal{L}'_1 = \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A})] &= \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)] \\ &= \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A}_1) \times \mathcal{L}(\mathcal{A}_2)] \\ &= \mathcal{L}(\mathcal{A}_1) \times \underbrace{\Pi_{\Sigma_1 \cap \Sigma_2}[\mathcal{L}(\mathcal{A}_2)]}_{\textit{message}}\end{aligned}$$

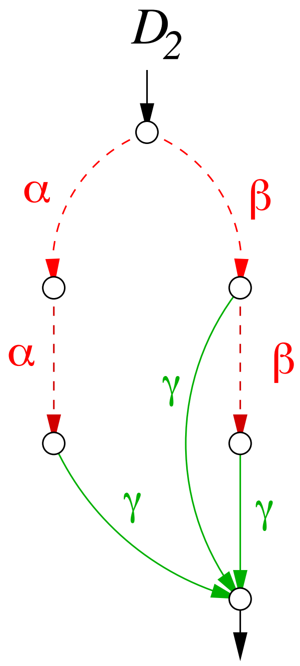
- the reduced language of $A_1$ combines its local language with a message from the other component $A_2$
- the message contains information about possible actions of $A_2$ on shared letters
- these synchronization possibilities are used to filter out behaviors of $A_1$ that are not compatible with any run of $A_2$



## Example

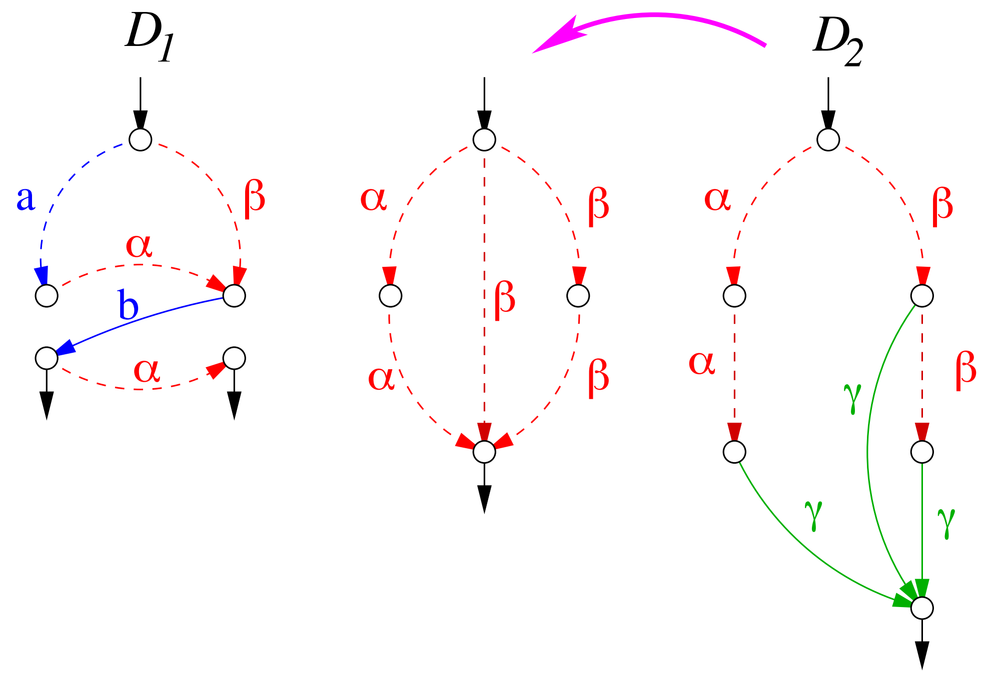
- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages



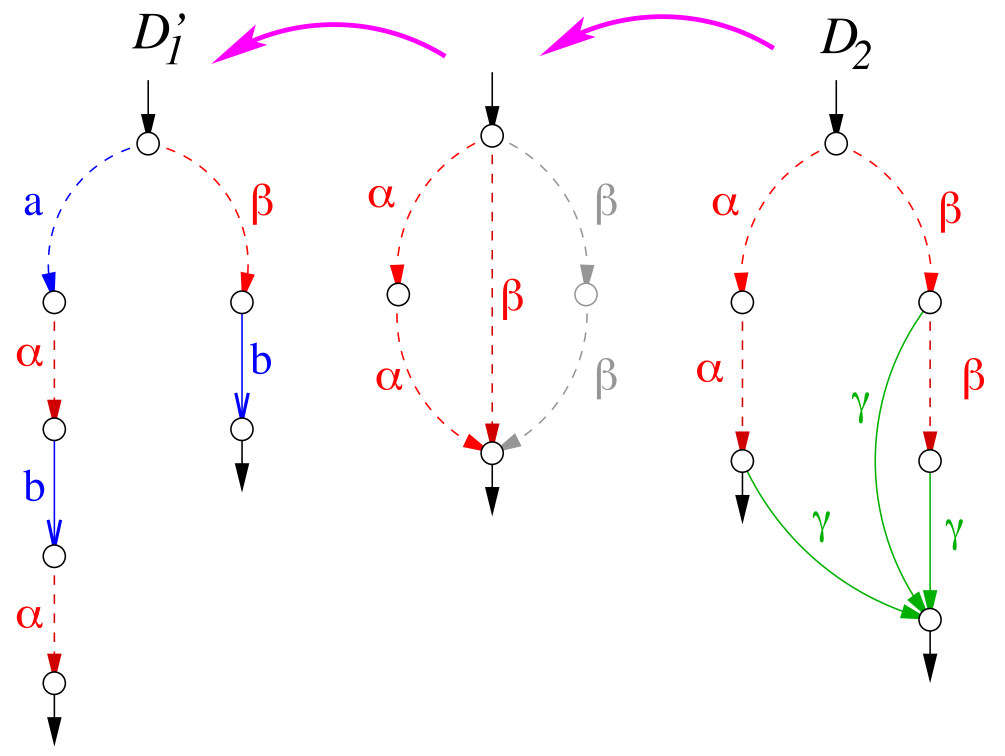
## Example

- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages



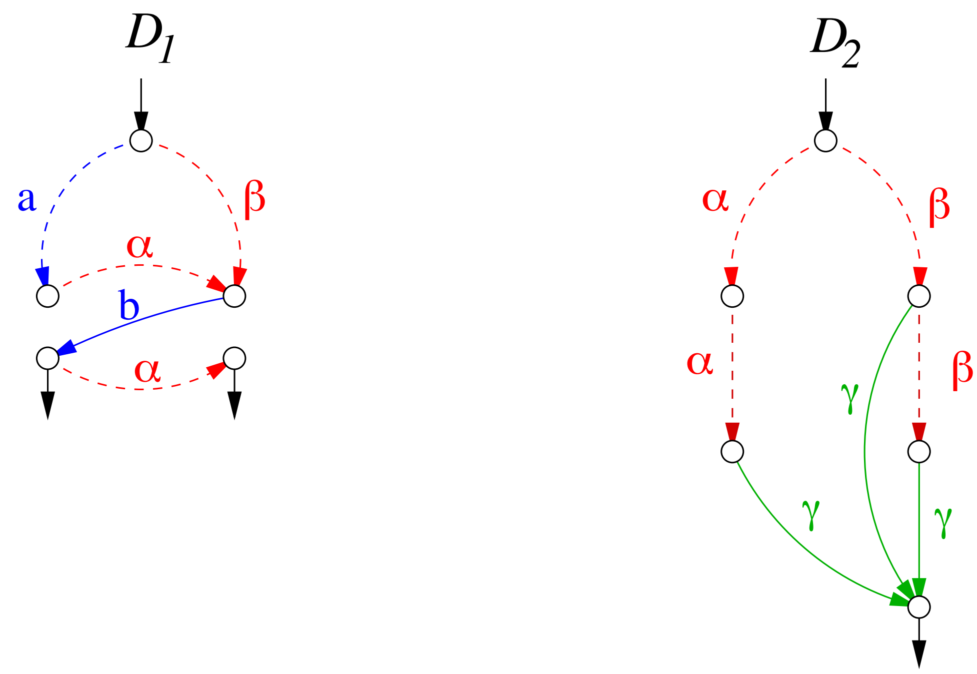
## Example

- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages



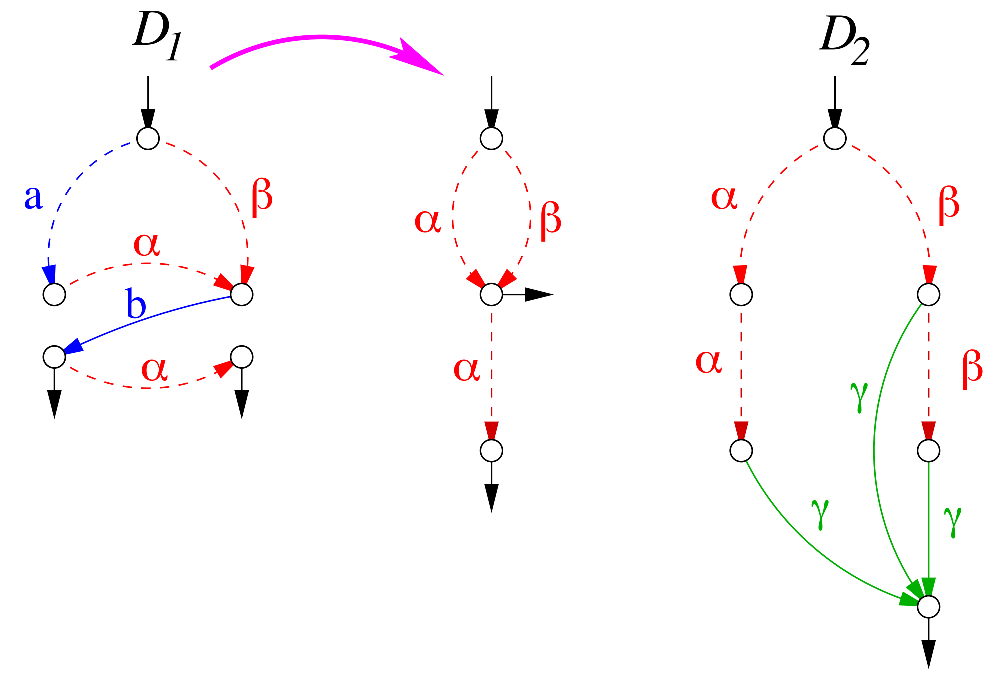
## Example

- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages



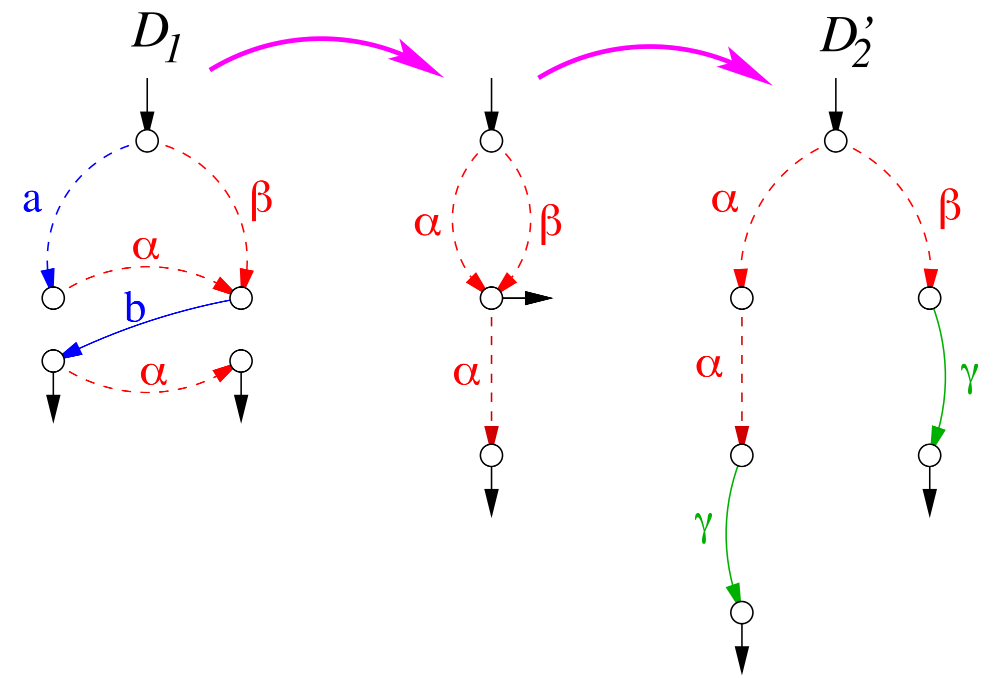
## Example

- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages



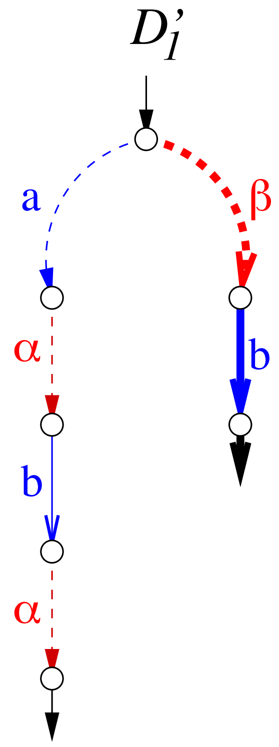
**Example**

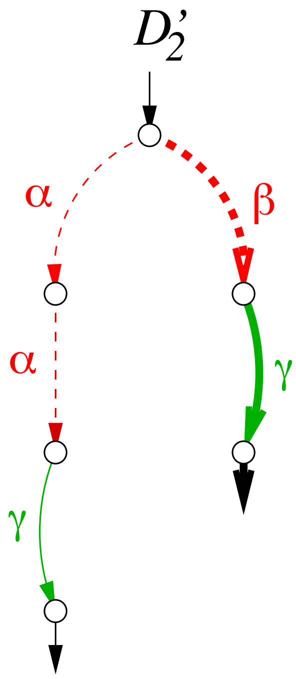
- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages



## Example

- $\Sigma_1 = \{a, b, \alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \beta, \gamma\}$
- computations performed on automata instead of languages

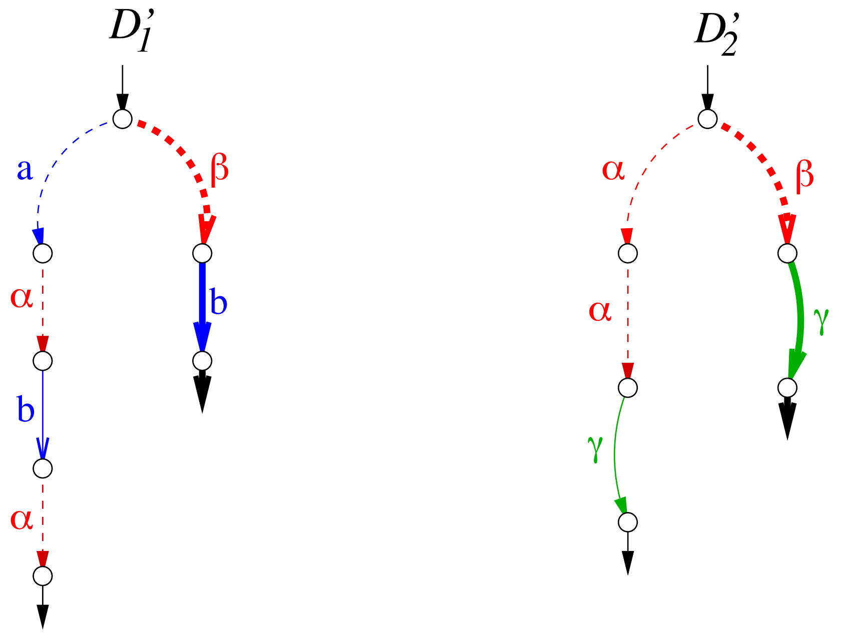


# Application 1

## Distributed planning

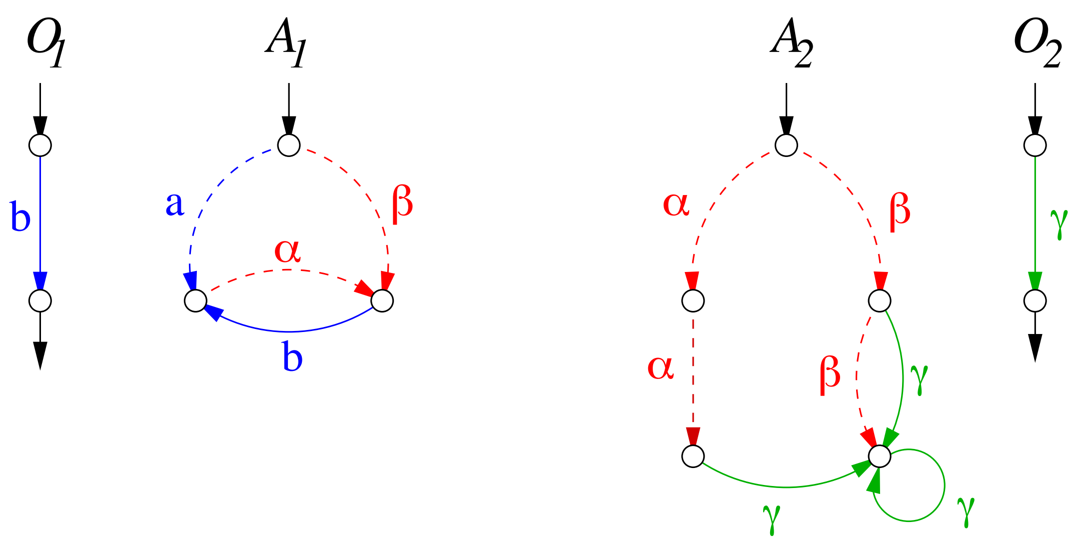
- compute a pair/tuple of compatible words/runs/sequences of actions, one per component
- computations are distributed, by message passing
- the resulting global plan is a tuple of local plans, i.e. a Mazurkiewicz trace, i.e. a partial order of actions, where actors sync. by *rendez-vous*
- the resulting plan can be executed in a distributed manner



# Application 2

## Distributed diagnosis

- some actions are observable in each component : $\Sigma_{i,o} \subseteq \Sigma_i$
- the global system $A = A_1 \times \dots \times A_N$ performs some hidden run $w$
  one only observes its signature in each component
- compute a pair/tuple of compatible words/runs/sequences of actions, one per component $o_i = \Pi_{\Sigma_{i,o}}(w)$
- objective = recover all global runs matching distributed observations $o_1, \dots, o_N$



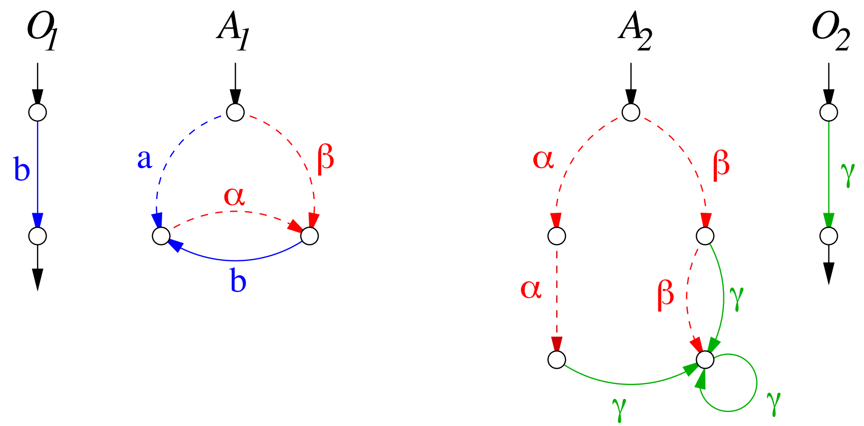

$$\Sigma_{1,o} = \{b\} \subseteq \{a, b, \alpha, \beta\} = \Sigma_1 \qquad \Sigma_{2,o} = \{\gamma\} \subseteq \{\alpha, \beta, \gamma\} = \Sigma_2$$

# Application 2

## Method

- synchronize runs of the distributed system with distributed observations
- observe that observations are a partial order, as well as runs of the system (they are handled in factorized form)
- **idea** : compute local diagnoses, then reduce them !

$$\begin{aligned}\mathcal{D} &= \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) \times (o_1 \times o_2) \\ &= [\mathcal{L}(\mathcal{A}_1) \times o_1] \times [\mathcal{L}(\mathcal{A}_2) \times o_2] \\ &= \mathcal{D}_1 \times \mathcal{D}_2 \\ &= \mathcal{D}'_1 \times \mathcal{D}'_2\end{aligned}$$

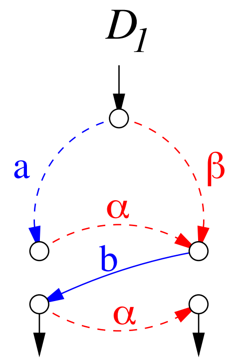


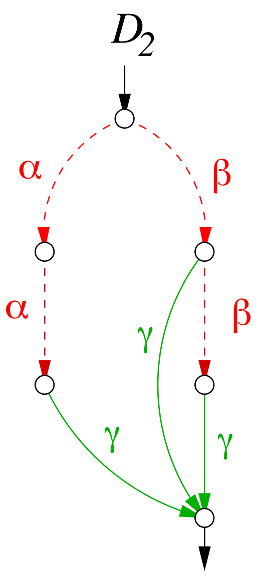
# Application 2

## Method

- synchronize runs of the distributed system with distributed observations
- observe that observations are a partial order, as well as runs of the system (they are handled in factorized form)
- **idea** : compute local diagnoses, then reduce them !

$$
\begin{aligned}
\mathcal{D} &= \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) \times (o_1 \times o_2) \\
&= [\mathcal{L}(\mathcal{A}_1) \times o_1] \times [\mathcal{L}(\mathcal{A}_2) \times o_2] \\
&= \mathcal{D}_1 \times \mathcal{D}_2 \\
&= \mathcal{D}'_1 \times \mathcal{D}'_2
\end{aligned}
$$



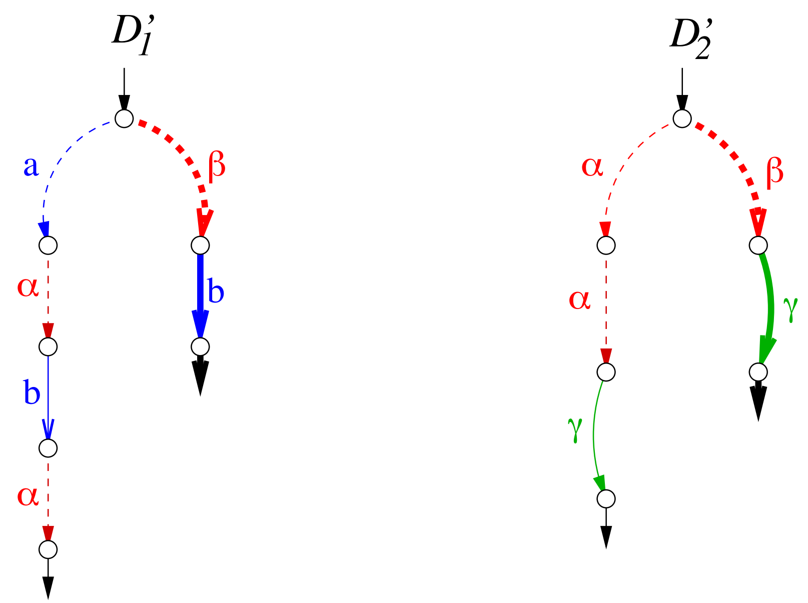



# Application 2

## Method

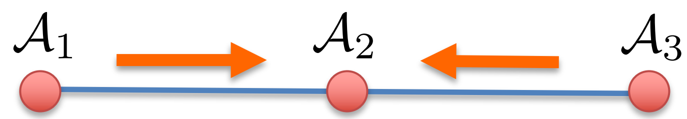
- synchronize runs of the distributed system with distributed observations
- observe that observations are a partial order, as well as runs of the system (they are handled in factorized form)
- **idea** : compute local diagnoses, then reduce them !

$$
\begin{aligned}
\mathcal{D} &= \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) \times (o_1 \times o_2) \\
&= [\mathcal{L}(\mathcal{A}_1) \times o_1] \times [\mathcal{L}(\mathcal{A}_2) \times o_2] \\
&= \mathcal{D}_1 \times \mathcal{D}_2 \\
&= \mathcal{D}_1' \times \mathcal{D}_2'
\end{aligned}
$$



# Consequence 2

$$\Sigma' \supseteq \Sigma_1 \cap \Sigma_2 \quad \Rightarrow \quad \Pi_{\Sigma'}(\mathcal{L}_1 \times \mathcal{L}_2) \;=\; \Pi_{\Sigma'}(\mathcal{L}_1) \times \Pi_{\Sigma'}(\mathcal{L}_2)$$




**Case of 3 components**, with $\Sigma_1 \cap \Sigma_3 \subseteq \Sigma_2$ ( or even $\Sigma_1 \cap \Sigma_3 = \emptyset$ )

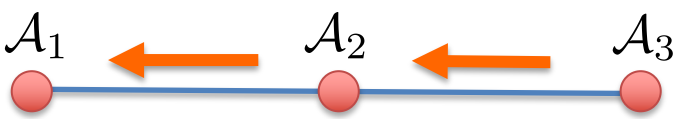
- **Merge rule** :

$$\begin{aligned}
\mathcal{L}'_2 = \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A})] &= \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_1) \times \mathcal{L}(\mathcal{A}_2) \times \mathcal{L}(\mathcal{A}_3)] \\
&= \mathcal{L}(\mathcal{A}_2) \times \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_1) \times \mathcal{L}(\mathcal{A}_3)] \\
&= \mathcal{L}(\mathcal{A}_2) \times \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_1)] \times \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_3)] \\
&= \mathcal{L}(\mathcal{A}_2) \times \Pi_{\Sigma_1 \cap \Sigma_2}[\mathcal{L}(\mathcal{A}_1)] \times \Pi_{\Sigma_2 \cap \Sigma_3}[\mathcal{L}(\mathcal{A}_3)]
\end{aligned}$$

  - combines messages of lateral components with local language
  - messages inform about possible words on shared letters

# Consequence 2

$$\Sigma' \supseteq \Sigma_1 \cap \Sigma_2 \quad \Rightarrow \quad \Pi_{\Sigma'}(\mathcal{L}_1 \times \mathcal{L}_2) \;=\; \Pi_{\Sigma'}(\mathcal{L}_1) \times \Pi_{\Sigma'}(\mathcal{L}_2)$$




**Case of 3 components**, with $\Sigma_1 \cap \Sigma_3 \subseteq \Sigma_2$ ( or even $\Sigma_1 \cap \Sigma_3 = \emptyset$ )

- **Propagation rule** : uses $(\Sigma_1 \cup \Sigma_2) \cap (\Sigma_2 \cup \Sigma_3) = \Sigma_2$

$$\begin{aligned}
\mathcal{L}'_1 = \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A})] &= \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A}_1) \times \mathcal{L}(\mathcal{A}_2) \times \mathcal{L}(\mathcal{A}_3)] \\
&= \mathcal{L}(\mathcal{A}_1) \times \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A}_2) \times \mathcal{L}(\mathcal{A}_3)] \\
&= \mathcal{L}(\mathcal{A}_1) \times \Pi_{\Sigma_1}[\Pi_{\Sigma_1 \cup \Sigma_2}[\mathcal{L}(\mathcal{A}_2) \times \mathcal{L}(\mathcal{A}_3)]] \\
&= \mathcal{L}(\mathcal{A}_1) \times \Pi_{\Sigma_1}[\Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_2) \times \mathcal{L}(\mathcal{A}_3)]] \\
&= \mathcal{L}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}[\mathcal{L}(\mathcal{A}_2) \times \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_3)]] \\
&= \mathcal{L}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}[\mathcal{L}(\mathcal{A}_2) \times \Pi_{\Sigma_2 \cap \Sigma_3}[\mathcal{L}(\mathcal{A}_3)]]
\end{aligned}$$

  - messages propagate from extremities
  - they are progressively combined to local component, reduced and forwarded

# Take home messages

## Computing on runs of a distributed system

- should be done on the factorized form
  (captures concurrency, more compact)
- this can be done in a distributed/modular way, by message passing

## Next time

- Petri nets : a new model for distributed/concurrent systems
- unfoldings/event structures : a new representation for (sets of) runs in a true concurrency semantics