MAD

Models & **A**lgorithms for **D**istributed systems

-- 3/5 --

download slides at

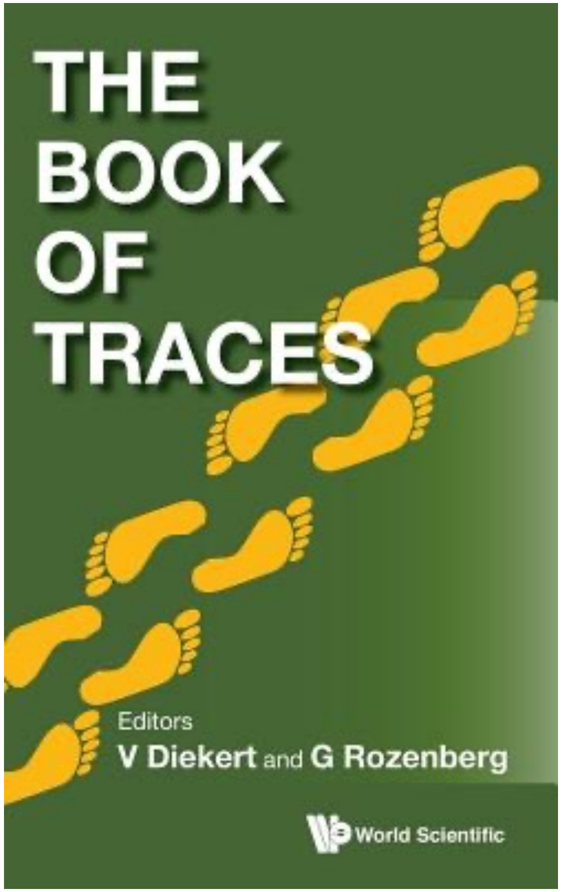
http://people.rennes.inria.fr/Eric.Fabre/

Today…

- A first formal model for distributed systems:
  networks of automata

- We recall the basics of automata and formal languages…
- …then introduce
  - the product of automata
  - Mazurkiewicz traces as a first true concurrency semantics for these systems
- …and start studying
  - algebraic properties of languages of networks of automata
  - distributed computations on traces

1994-95



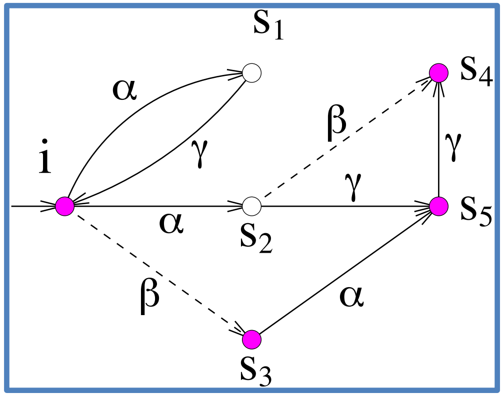
Traces and trace languages

- the counterpart of formal languages, handling runs as partial orders of events instead of sequences
- recognizability/rationality : asynchronous automata by Zielonka
- event structures as a central object
- adequate logics
- Antoni Mazurkiewicz as leading contributor

Preliminaries

Automaton $\mathcal{A} = (S, T, \Sigma, s_o, S_F)$

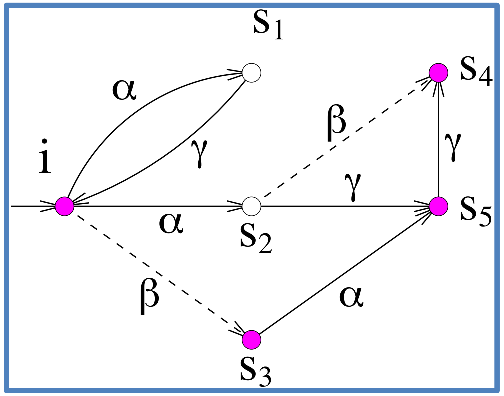
- finite state set S, initial state s_0, final/marked states S_F (optional)
- finite label set (alphabet) Σ
- transition set $T \subseteq S \times \Sigma \times S$
  notation for transitions $t = (s, \alpha, s') = ({}^\bullet t, \sigma(t), t^\bullet)$
- trajectory/run $\omega = t_1 t_2 \ldots t_n$
  - $t_i^\bullet = {}^\bullet t_{i+1}, \quad 1 \leq i < n$
  - ${}^\bullet t_1 = s_0, \quad t_n^\bullet \in S_F$



Preliminaries

Automaton $\mathcal{A} = (S, T, \Sigma, s_o, S_F)$

- finite state set S, initial state s_0, final/marked states S_F (optional)
- finite label set (alphabet) Σ
- transition set $T \subseteq S \times \Sigma \times S$
  notation for transitions $t = (s, \alpha, s') = ({}^{\bullet}t, \sigma(t), t^{\bullet})$
- trajectory/run $\omega = t_1 t_2 ... t_n$
  - $t_i^{\bullet} = {}^{\bullet}t_{i+1}, \quad 1 \leq i < n$
  - ${}^{\bullet}t_1 = s_0, \quad t_n^{\bullet} \in S_F$



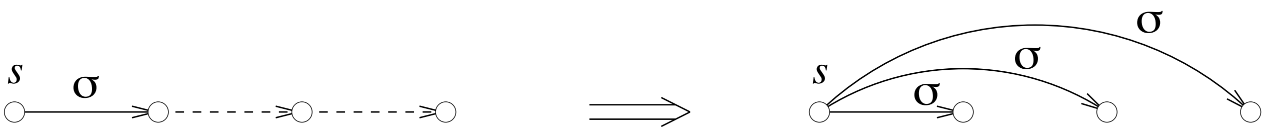
- $\mathcal{A}$ is deterministic iff $\forall s, \alpha, \quad |\{(s, \alpha, s') \in T\}| \leq 1$
- language of $\mathcal{A}$: $\mathcal{L}(\mathcal{A}) = \{\sigma(\omega) : \omega \text{ run of } \mathcal{A}\}$
  where $\sigma(t_1 ... t_n) = \sigma(t_1) ... \sigma(t_n) \in \Sigma^*$
- A language $\mathcal{L} \subseteq \Sigma^*$ is regular iff it is the language of some automaton.
- **Thm** : there exists a unique minimal deterministic automaton recognizing a given regular language.

Projection of an automaton

Projection of $\mathcal{A} = (S, T, \Sigma, s_o, S_F)$ on sub-alphabet $\Sigma' \subseteq \Sigma$

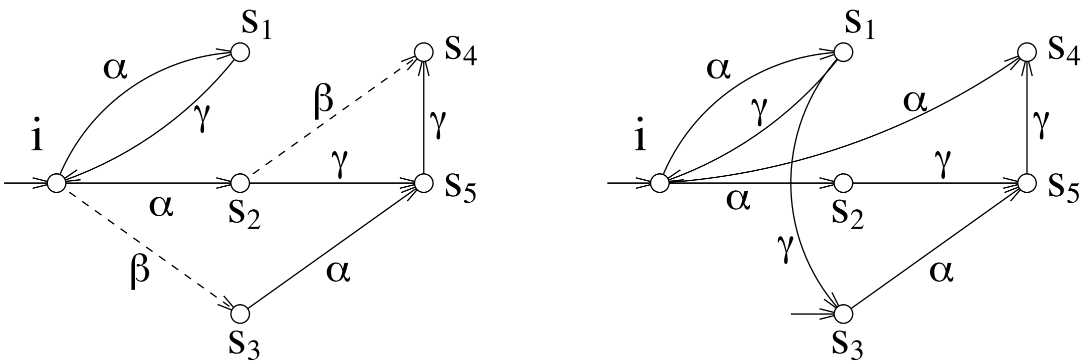
$$\mathcal{A}' = \Pi_{\Sigma'}(\mathcal{A}) = (S, T', \Sigma', s_0, S_F)$$

- in transitions, replace each label $\alpha \in \Sigma \setminus \Sigma'$ by ϵ (empty word)
- perform ε-reduction (or ε-closure), to the right or to the left



- one may then determinize and minimize the result

Example $\mathcal{A}' = \Pi_{\{\alpha,\gamma\}}(\mathcal{A})$

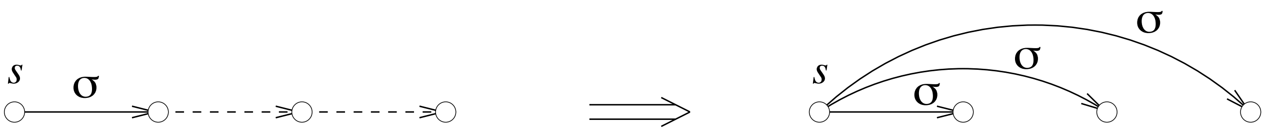


Projection of an automaton

Projection of $\mathcal{A} = (S, T, \Sigma, s_o, S_F)$ on sub-alphabet $\Sigma' \subseteq \Sigma$

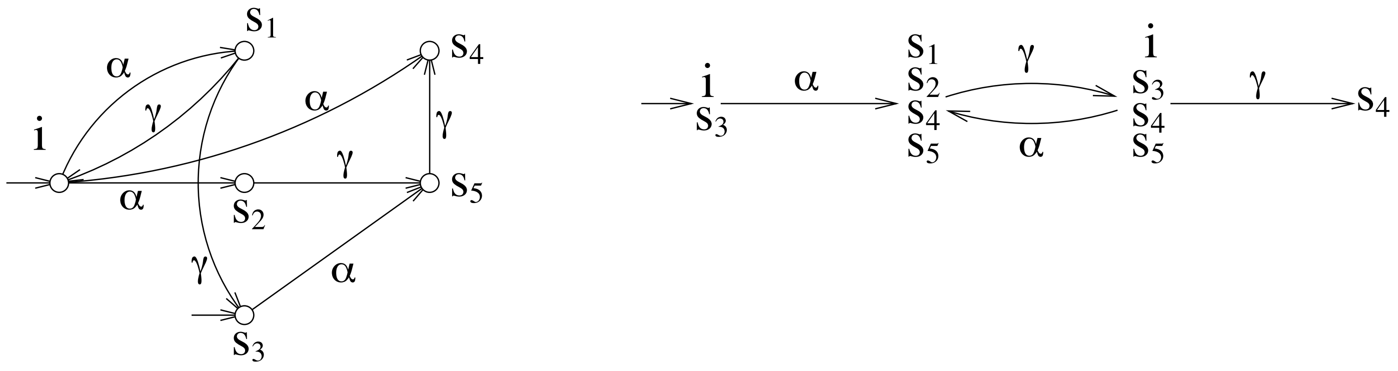
$$\mathcal{A}' = \Pi_{\Sigma'}(\mathcal{A}) = (S, T', \Sigma', s_0, S_F)$$

- in transitions, replace each label $\alpha \in \Sigma \setminus \Sigma'$ by ϵ (empty word)
- perform ε-reduction (or ε-closure), to the right or to the left



- one may then determinize and minimize the result

Example $\mathcal{A}' = \Pi_{\{\alpha,\gamma\}}(\mathcal{A})$



Projection of a language

Projection of $\mathcal{L} \subseteq \Sigma^*$ on sub-alphabet $\Sigma' \subseteq \Sigma$

$$\mathcal{L}' = \Pi_{\Sigma'}(\mathcal{L}) \subseteq (\Sigma')^*$$

- on letters $\Pi_{\Sigma'}(\alpha) = \alpha$ if $\alpha \in \Sigma'$ and $\Pi_{\Sigma'}(\alpha) = \epsilon$ otherwise
- extension to words : $\Pi_{\Sigma'}(uv) = \Pi_{\Sigma'}(u)\Pi_{\Sigma'}(v)$
- extension to languages, *i.e.* sets of words
- amounts to erasing letters of $\Sigma \setminus \Sigma'$ in words of $\mathcal{L}$

Projection of a language

Projection of $\mathcal{L} \subseteq \Sigma^*$ on sub-alphabet $\Sigma' \subseteq \Sigma$

$$\mathcal{L}' = \Pi_{\Sigma'}(\mathcal{L}) \subseteq (\Sigma')^*$$

- on letters $\Pi_{\Sigma'}(\alpha) = \alpha$ if $\alpha \in \Sigma'$ and $\Pi_{\Sigma'}(\alpha) = \epsilon$ otherwise
- extension to words : $\Pi_{\Sigma'}(uv) = \Pi_{\Sigma'}(u)\Pi_{\Sigma'}(v)$
- extension to languages, *i.e.* sets of words
- amounts to erasing letters of $\Sigma \setminus \Sigma'$ in words of $\mathcal{L}$

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

Proof: exercise

Networks of automata

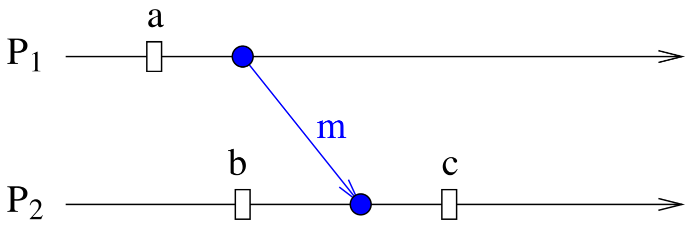
Objectives

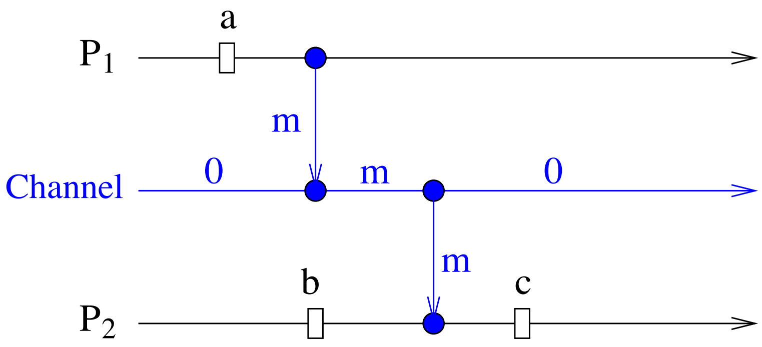
- so far, processes were abstract machines, computing and communicating
- towards a formal model of distributed system:
  let’s put behaviors/purposes into processes
- we want to be able to verify, analyze, control, diagnose, etc. such systems
- **idea** : a (local) process becomes an automaton

Simplification

Let’s get rid of channels !

- we add processes that represent channels
- writing/reading on the channel becomes instantaneous
- the process “channel” can delay the messages
- it can also have behaviors (FIFO, lossy,…)

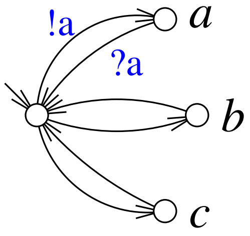


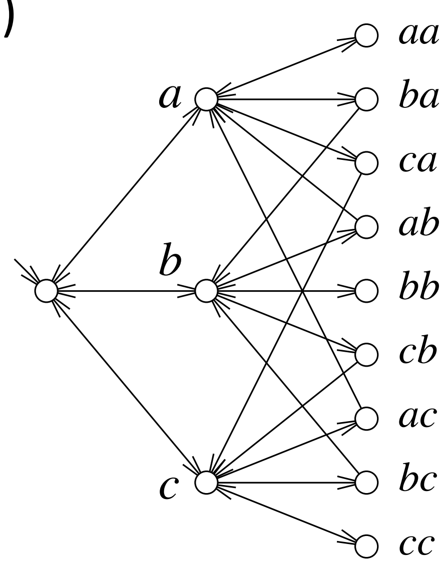


Simplification

Let’s get rid of channels !

- we add processes that represent channels
- writing/reading on the channel becomes instantaneous
- the process “channel” can delay the messages
- it can also have behaviors (FIFO, lossy,…)

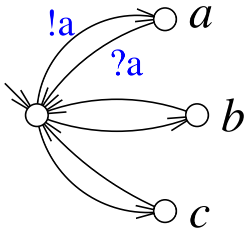


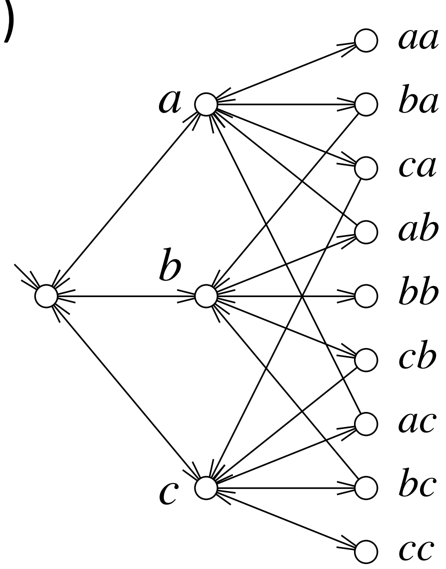


Simplification

Let's get rid of channels !

- we add processes that represent channels
- writing/reading on the channel becomes instantaneous
- the process "channel" can delay the messages
- it can also have behaviors (FIFO, lossy,…)





- what do we gain:
  - homogeneity (1 object type instead of 2)
  - synchrony of interactions
  - without losing the global asynchrony of behaviors
- what do we lose:
  - (finite number of messages) + one reading action per possible message
  - channels are not anymore "passive" objects
  - need to recall that actions of a component "channel" can not be enforced
  - and that their state needs not be observable (one may have to estimate it from outside)

Synchronous composition of processes

- **Automata** $\mathcal{A}_i = (S_i, T_i, \Sigma_i, s_{i,0}, S_{i,F})$ for i=1,2
- **Product** $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 = (S, T, \Sigma, s_0, S_F)$ where
  - states $S = S_1 \times S_2, \quad s_0 = (s_{1,0}, s_{2,0}), \quad S_F = S_{1,F} \times S_{2,F}$
  - labels $\Sigma = \Sigma_1 \cup \Sigma_2$

    shared labels $\Sigma = \Sigma_1 \cap \Sigma_2$ define synchronized actions

Synchronous composition of processes

- **Automata** $\mathcal{A}_i = (S_i, T_i, \Sigma_i, s_{i,0}, S_{i,F})$ for i=1,2
- **Product** $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 = (S, T, \Sigma, s_0, S_F)$ where
  - states $S = S_1 \times S_2, \quad s_0 = (s_{1,0}, s_{2,0}), \quad S_F = S_{1,F} \times S_{2,F}$
  - labels $\Sigma = \Sigma_1 \cup \Sigma_2$

    shared labels $\Sigma = \Sigma_1 \cap \Sigma_2$ define synchronized actions
  - transitions, for $t_i \in T_i, \quad s_i \in S_i$

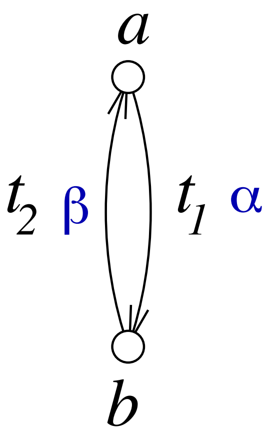
    $T = \{(t_1, t_2) \ : \ \sigma_1(t_1) = \sigma_2(t_2) \in \Sigma_1 \cap \Sigma_2\}$ synchronized actions

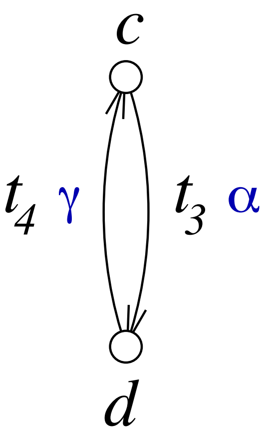
    $\biguplus \ \{(t_1, \star_{s_2}) \ : \ \sigma_1(t_1) \in \Sigma_1 \setminus \Sigma_2\}$ private moves in $\mathcal{A}_1$

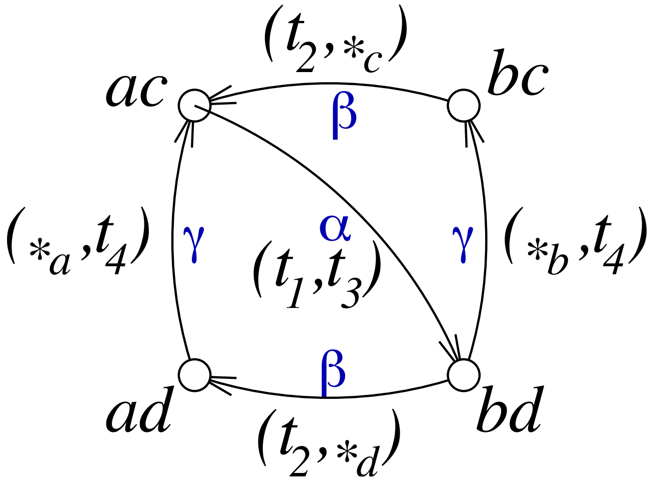
    $\biguplus \ \{(\star_{s_1}, t_2) \ : \ \sigma_2(t_2) \in \Sigma_2 \setminus \Sigma_1\}$ private moves in $\mathcal{A}_2$

  - flow relation given by ${}^\bullet(t_1, t_2) = ({}^\bullet t_1, {}^\bullet t_2)$ and $(t_1, t_2)^\bullet = (t_1^\bullet, t_2^\bullet)$
    where one can have $t_i = \star_{s_i}$ and ${}^\bullet(\star_{s_i}) = s_i = (\star_{s_i})^\bullet$

Example







Network of automata

(or distributed automaton)

- We call a network of automata a system $\mathcal{A}$ defined as

$$\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$$

- interaction graph of a distributed automaton: $G = (V = \{1, \ldots, N\}, E)$
  - each node *i* stands for component $\mathcal{A}_i$
  - edge *i—j* exists iff $\Sigma_i \cap \Sigma_j \neq \emptyset$
- caution:
  - this model allows synchronous actions with more than 2 components
  - if $\alpha \in \Sigma_i \cap \Sigma_j \cap \Sigma_k$ then action α **must** be performed jointly by $\mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_k$
  - in general, all components declaring some shared label **must** contribute to fire it
- The factorized form is a more compact description of the system (exponential state space explosion with number of components)

Product of languages

- Let $\mathcal{L}_1, \mathcal{L}_2$ be languages, with $\mathcal{L}_i \subseteq \Sigma_i^*$
- let $\Sigma = \Sigma_1 \cup \Sigma_2$ be the union of their alphabets
- let $\Pi_i : \Sigma^* \to \Sigma_i^*$ be the canonical projections
- The product $\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2$ is defined as

$$\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 = \Pi_1^{-1}(\mathcal{L}_1) \cap \Pi_2^{-1}(\mathcal{L}_2)$$

- it consists of words over Σ which projections through Π_1, Π_2 lie in $\mathcal{L}_1, \mathcal{L}_2$ respectively

- Example $\Sigma_1 = \{a, b\}, \Sigma_2 = \{a, c, o\}$

$$\mathcal{L}_1 = \{abba\}, \mathcal{L}_2 = \{cacao\}$$

$$\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 = \{cabbcao, cabcbao, cacbbao\}$$

Remarks (and homework)

- the product of two words can be several words (interleaving of private letters)
- homework: what is the size of $\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2$ when $\Sigma_1 \cap \Sigma_2 = \emptyset$?
- homework: find an NSC for $w_1 \times w_2 = \emptyset$, with words $w_i \in \Sigma_i^*$
- homework: design an algorithm to compute $w_1 \times w_2$
- **Thm** $\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 = \emptyset$ iff $\Pi_{1,2}(\mathcal{L}_1) \cap \Pi_{1,2}(\mathcal{L}_2) = \emptyset$
  proof : homework

Remarks (and homework)

- the product of two words can be several words
  (interleaving of private letters)
- homework: what is the size of $\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2$ when $\Sigma_1 \cap \Sigma_2 = \emptyset$?
- homework: find an NSC for $w_1 \times w_2 = \emptyset$, with words $w_i \in \Sigma_i^*$
- homework: design an algorithm to compute $w_1 \times w_2$
- **Thm** $\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 = \emptyset$ iff $\Pi_{1,2}(\mathcal{L}_1) \cap \Pi_{1,2}(\mathcal{L}_2) = \emptyset$
  proof : homework

Thm let $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_N$ be a network of automata, then

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

proof : it is enough to check it for N=2
then proceed by double inclusion (exercise)

Product of two words

- w_1 = babbab, w_2 = acac over $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$ resp.

$$w_1 \times w_2 = \{w \in \Sigma^* \ :\ \Pi_{\Sigma_1}(w) = w_1,\ \Pi_{\Sigma_2}(w) = w_2\}$$

- one has $w_1 \times w_2 \neq \emptyset \ \Leftrightarrow\ \Pi_{\Sigma_2}(w_1) = \Pi_{\Sigma_1}(w_2)$

- **Algorithm** to build one such w : repeat until end of w_1 and w_2
  - interleave private parts of both words, until next synchro
  - place next synchro action of both words, if they match, otherwise return $\emptyset$

w_1 = babbab

w = …

w_2 = acac

Product of two words

- w_1 = babbab, w_2 = acac over $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$ resp.

$$w_1 \times w_2 = \{w \in \Sigma^* \ :\ \Pi_{\Sigma_1}(w) = w_1,\ \Pi_{\Sigma_2}(w) = w_2\}$$

- one has $w_1 \times w_2 \neq \emptyset \quad \Leftrightarrow \quad \Pi_{\Sigma_2}(w_1) = \Pi_{\Sigma_1}(w_2)$

- **Algorithm** to build one such w : repeat until end of w_1 and w_2
  - interleave private parts of both words, until next synchro
  - place next synchro action of both words, if they match, otherwise return $\emptyset$

w_1 = **babbab**

w = **b**…

w_2 = acac

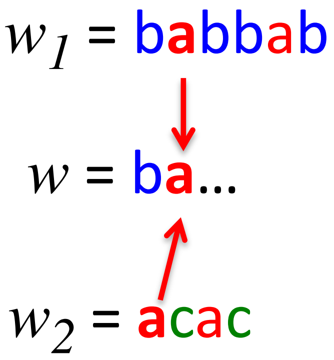
Product of two words

- w_1 = babbab, w_2 = acac over $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$ resp.

$$w_1 \times w_2 = \{w \in \Sigma^* \ :\ \Pi_{\Sigma_1}(w) = w_1,\, \Pi_{\Sigma_2}(w) = w_2\}$$

- one has $w_1 \times w_2 \neq \emptyset \ \Leftrightarrow\ \Pi_{\Sigma_2}(w_1) = \Pi_{\Sigma_1}(w_2)$

- **Algorithm** to build one such w : repeat until end of w_1 and w_2
  - interleave private parts of both words, until next synchro
  - place next synchro action of both words, if they match, otherwise return $\emptyset$



w_1 = babbab

w = ba...

w_2 = acac

Product of two words

- w_1 = babbab, w_2 = acac over $\Sigma_1 = \{a, b\}, \ \Sigma_2 = \{a, c\}$ resp.

$$w_1 \times w_2 = \{w \in \Sigma^* \ : \ \Pi_{\Sigma_1}(w) = w_1, \ \Pi_{\Sigma_2}(w) = w_2\}$$

- one has $w_1 \times w_2 \neq \emptyset \ \Leftrightarrow \ \Pi_{\Sigma_2}(w_1) = \Pi_{\Sigma_1}(w_2)$

- **Algorithm** to build one such w : repeat until end of w_1 and w_2
  - interleave private parts of both words, until next synchro
  - place next synchro action of both words, if they match, otherwise return $\emptyset$

w_1 = ba**bb**ab

w = ba**bbc**...

w_2 = a**c**ac

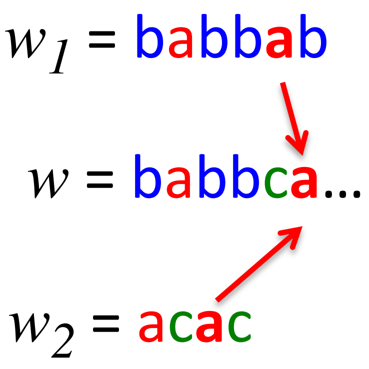
Product of two words

- w_1 = babbab, w_2 = acac over $\Sigma_1 = \{a, b\}, \ \Sigma_2 = \{a, c\}$ resp.

$$w_1 \times w_2 = \{ w \in \Sigma^* \ : \ \Pi_{\Sigma_1}(w) = w_1, \ \Pi_{\Sigma_2}(w) = w_2 \}$$

- one has $w_1 \times w_2 \neq \emptyset \ \Leftrightarrow \ \Pi_{\Sigma_2}(w_1) = \Pi_{\Sigma_1}(w_2)$

- **Algorithm** to build one such w : repeat until end of w_1 and w_2
  - interleave private parts of both words, until next synchro
  - place next synchro action of both words, if they match, otherwise return $\emptyset$



w_1 = babb**a**b

w = babbc**a**...

w_2 = ac**a**c

Product of two words

- w_1 = babbab, w_2 = acac over $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$ resp.

$$w_1 \times w_2 = \{w \in \Sigma^* \ : \ \Pi_{\Sigma_1}(w) = w_1,\ \Pi_{\Sigma_2}(w) = w_2\}$$

- one has $w_1 \times w_2 \neq \emptyset \ \Leftrightarrow \ \Pi_{\Sigma_2}(w_1) = \Pi_{\Sigma_1}(w_2)$

- **Algorithm** to build one such w : repeat until end of w_1 and w_2
  - interleave private parts of both words, until next synchro
  - place next synchro action of both words, if they match, otherwise return $\emptyset$

w_1 = babba**b**

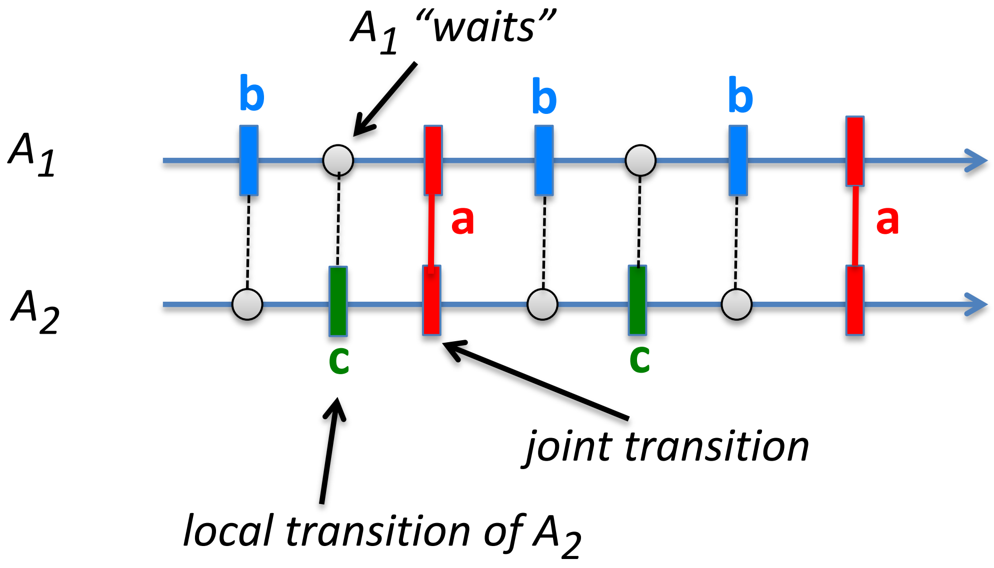
w = babbca**bc**

w_2 = aca**c**

Towards true concurrency semantics

Problem for a distributed system runs/words are still sequences of events.
How to model the fact that private events in the
could occur in any order ?

Example $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ with $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$

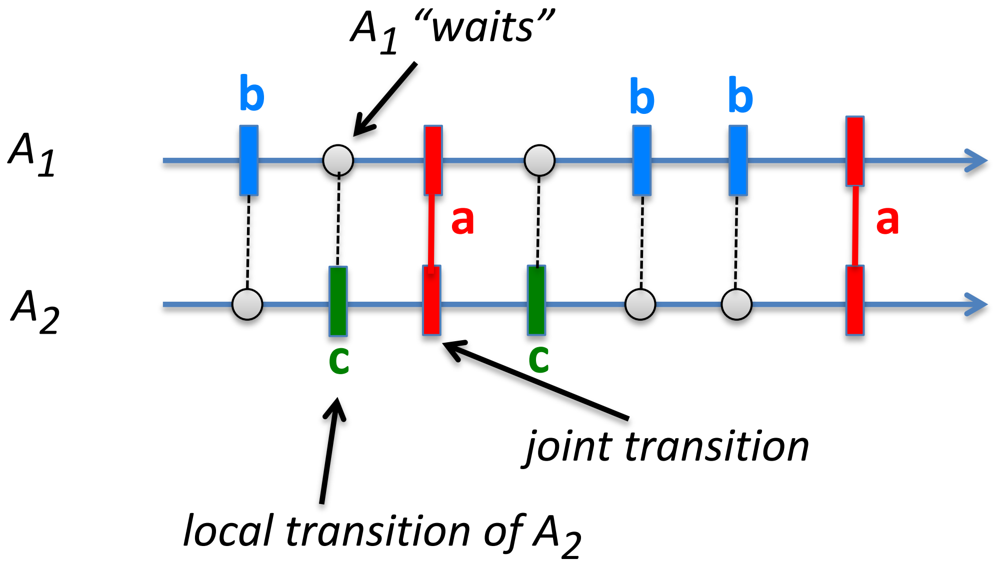


For the sequential semantics, runs **bcabcba** and **bcacbba** are different !

Towards true concurrency semantics

Problem for a distributed system runs/words are still sequences of events.
How to model the fact that private events in the
could occur in any order ?

Example $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ with $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$



For the sequential semantics, runs **bcabcba** and **bcacbba** are different !

Mazurkiewicz traces

Idea: define runs as equivalence relations of sequences, *i.e.* allow the permutation of successive events that live on different components

Dependency : on letters of $\Sigma = \cup_i \Sigma_i$

$$\alpha \, D \, \beta \quad \Leftrightarrow \quad \exists i, \ \alpha, \beta \in \Sigma_i$$

in any run of $\mathcal{A}$, these letters will be ordered by at least one component $\mathcal{A}_i$

Independence : complement of the dependency relation, denoted $\alpha \ I \ \beta$

Mazurkiewicz traces

Idea: define runs as equivalence relations of sequences, *i.e.* allow the permutation of successive events that live on different components

Dependency : on letters of $\Sigma = \cup_i \Sigma_i$

$$\alpha \, D \, \beta \quad \Leftrightarrow \quad \exists i, \; \alpha, \beta \in \Sigma_i$$

in any run of $\mathcal{A}$, these letters will be ordered by at least one component $\mathcal{A}_i$

Independence : complement of the dependency relation, denoted $\alpha \; I \; \beta$

Equivalence relation on words in $\mathcal{L}(\mathcal{A})$

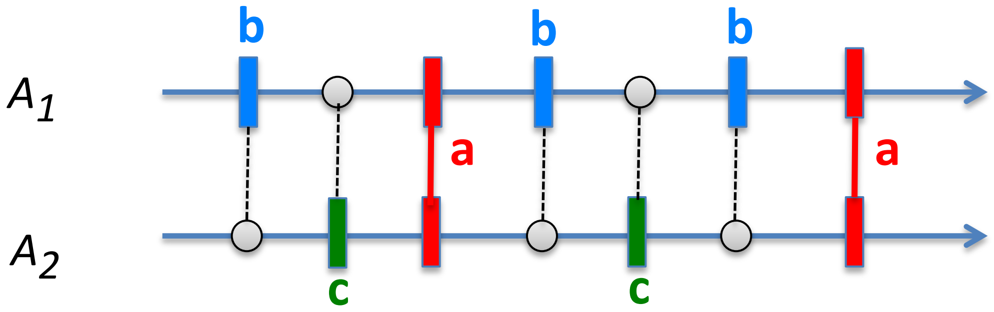
$$w\alpha\beta w' \;\equiv\; w\beta\alpha w' \quad \Leftarrow \quad \alpha I \beta$$

we consider the equivalence relation on words generated by this property

Trace of a word w, denoted *[w]*: it is the equivalence class of w for $\equiv$

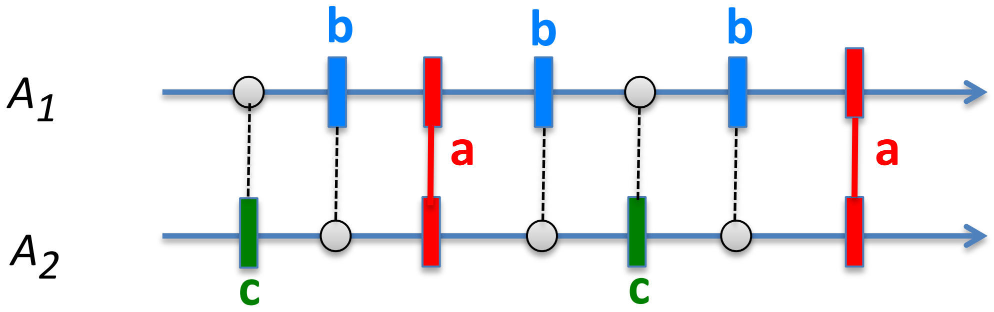
it is also the set of sequences obtained by successively permuting consecutive independent letters

Example $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$ one has $b\ I\ c$



[bcabcba]= { bcabcba, cbabcba, bcacbba, cbacbba, bcabbca, cbabbca }

Example $\Sigma_1 = \{a, b\},\ \Sigma_2 = \{a, c\}$ one has $b\ I\ c$



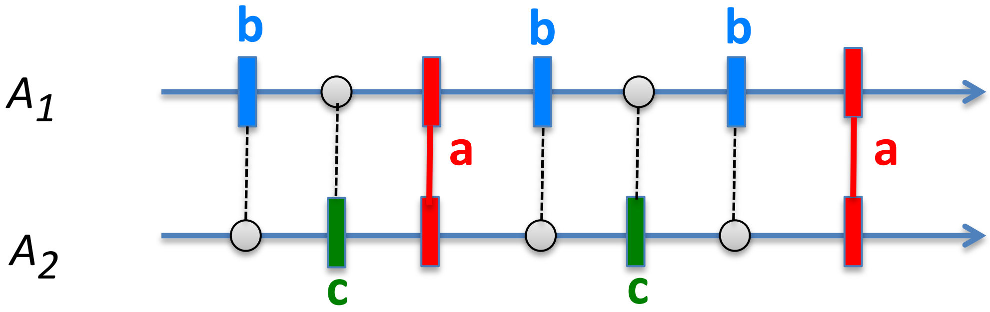
[bcabcba]= { bcabcba, cbabcba, bcacbba, cbacbba, bcabbca, cbabbca }

Concurrency: consider <u>events</u> α and β in word $w = u\,\alpha\, v\, v'\,\beta\, u'$ where u,v,u',v' are subwords

α and β are <u>concurrent</u> events in *w*, denoted $\alpha \perp \beta$, iff

$$\begin{aligned} w &= u\,\alpha\, v\, v'\,\beta\, u' \\ &\equiv u\, v\,\alpha\,\beta\, v'\, u' \\ &\equiv u\, v\,\beta\,\alpha\, v'\, u' \end{aligned}$$

Causality …otherwise, α and β are causally related, denoted $\alpha \prec \beta$

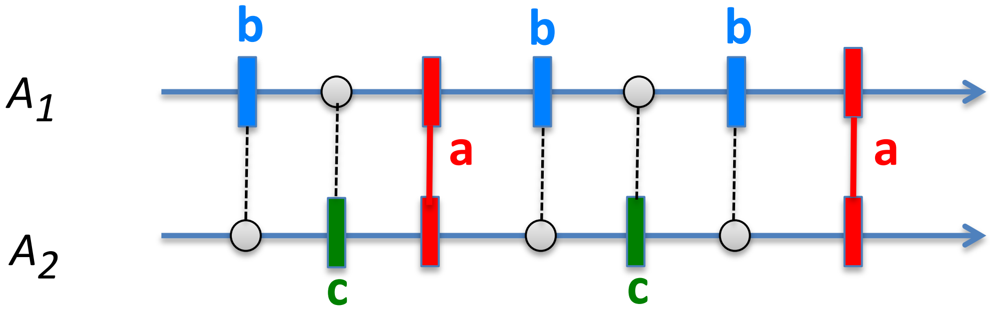


Concurrency: consider <u>events</u> α and β in word $w = u\,\alpha\,v\,v'\,\beta\,u'$ where u,v,u',v' are subwords

α and β are <u>concurrent</u> events in *w*, denoted $\alpha \perp \beta$, iff

$$\begin{aligned} w &= u\,\alpha\,v\,v'\,\beta\,u' \\ &\equiv u\,v\,\alpha\,\beta\,v'\,u' \\ &\equiv u\,v\,\beta\,\alpha\,v'\,u' \end{aligned}$$

Causality …otherwise, α and β are causally related, denoted $\alpha \prec \beta$



A trace as a partial order : let $w = e_1 e_2 \ldots e_n$

then $\prec$ defines a partial order on $\{e_1, \ldots, e_n\}$

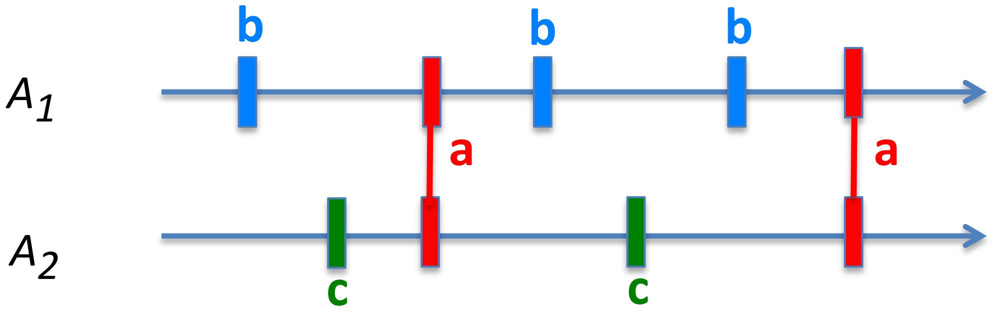
[different occurrences of the same letter are distinguished]

Concurrency: consider events α and β in word *w = u* α *v v'* β *u'*
where *u,v,u',v'* are subwords

α and β are concurrent events in *w*, denoted $\alpha \perp \beta$, iff

$$\begin{aligned} w &= u\,\alpha\,v\,v'\,\beta\,u' \\ &\equiv u\,v\,\alpha\,\beta\,v'\,u' \\ &\equiv u\,v\,\beta\,\alpha\,v'\,u' \end{aligned}$$

Causality ...otherwise, α and β are causally related, denoted $\alpha \prec \beta$

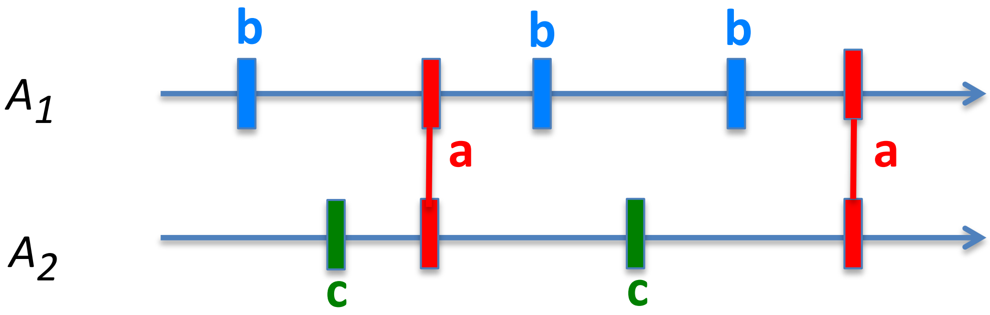


A trace as a partial order : let $w = e_1 e_2 \ldots e_n$
then $\prec$ defines a partial order on $\{e_1, \ldots, e_n\}$
[different occurrences of the same letter are distinguished]

Thm : let $w = e_1 e_2 ... e_n$ then *[w]* is obtained as the set of all linear extensions of $(\{e_1, ..., e_n\}, \prec)$

Proof : exercise (almost by construction/definition)

Consequence: a (Mazurkiewicz) trace is equivalently described as a partial order of events



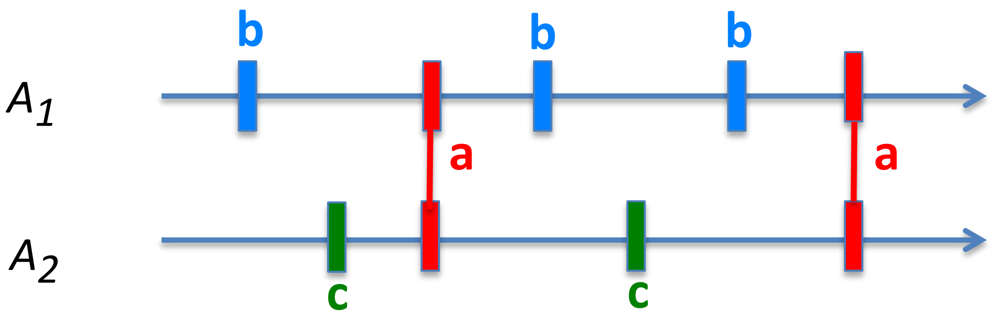
intuitively, one can consider it as a necklace with several threads, one per process, and pearls placed on either one or several threads, and free to move along it

Thm : Consider the network of automata $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$
let the $w_i \in \mathcal{L}(\mathcal{A}_i)$ be words in each component,
let $w \in w_1 \times \ldots \times w_N$, then

$$[w] = w_1 \times \ldots \times w_N \subseteq \mathcal{L}(\mathcal{A})$$

Proof : exercise (hint: proceed by double inclusion)

Consequence: *a (Mazurkiewicz) trace $[w]$ is equivalently described as a tuple of local words $(w_1,\ldots,w_N)$, one per component*



[bcabcba] = { bcabcba, cbabcba, bcacbba, cbacbba, bcabbca, cbabbca }
= { babba } x { caca }

The encoding of a trace as a tuple is similar to Mattern's vector clock !

Take home messages

In a network of automata

- one can define true concurrency semantics, where runs are partial orders of events
- encoding of these runs as products of sequences
- factorized representations are more compact

Next time

- distributed/modular algorithms to compute with these partial orders
- applications to multi-agent diagnosis & planning