

ASR : an introduction to Distributed Systems and Algorithms (Algorithmes et Systèmes Répartis)

1/3

M. Raynal¹, A. Mostefaoui², E. Fabre³

¹Pr., ASAP team

²MC, ASAP team

³CR INRIA, DistribCom team

Master 2, research in Computer Science, '07

Download the **lecture slides** from my web-page :

http://www.irisa.fr/distribcom/Personal_Pages/fabre/fabre.html

Outline

- 1 Introduction
 - What has been seen so far
 - Novelties
 - Contents of the 3 lectures
- 2 Models of a distributed application
 - Communicating automata
 - Channels as processes
 - Distributed automaton
 - Mazurkiewicz traces
 - Petri nets
- 3 Next time...

Objectives

- **accessibility** of a (global) state, or of an event in the application
- liveness, existence of **deadlocks**, fairness
- supervision/**diagnosis**:
i.e. inference of processes states from observations (*careful, different meaning*)
 - observations = partial orders
 - inference by distributed computations

Outline

- 1 Introduction
 - What has been seen so far
 - Novelties
 - Contents of the 3 lectures
- 2 **Models of a distributed application**
 - Communicating automata
 - Channels as processes
 - Distributed automaton
 - Mazurkiewicz traces
 - Petri nets
- 3 Next time...

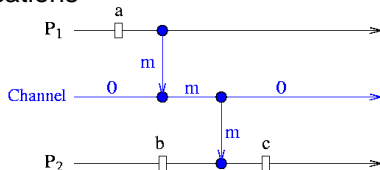
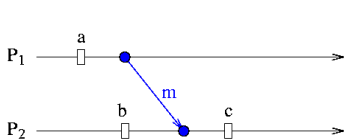
Notice:

- yields an infinite state system, because of the unbounded channels
- a Turing powerful model
- We limit ourselves to bounded channels, and simplify this model.

Channels as processes

Objective: Model messages content, and channel behaviors.

1st idea: instantaneous communications



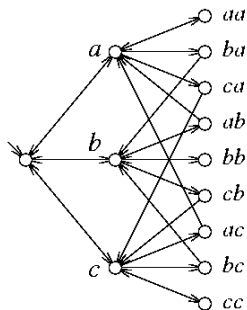
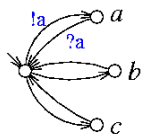
- we add processes that represent the channels
- writing/reading on the channel becomes instantaneous
- the process “channel” can delay the messages
- channels have a behavior, for ex. FIFO

2nd idea: a channel becomes an automaton, i.e. a process

Example:

$M = \{a, b, c\} =$ *nite* set of possible messages

FIFO channel of size 1, or of size 2



Summary:

it is equivalent to say that we only have processes (no channels), with instantaneous/synchronous interactions.

Distributed automaton

Automaton: (new formulation, more convenient)

$$\mathcal{A} = (\mathcal{S}, T, \rightarrow, s_0, \lambda, \Lambda)$$

- flow $\rightarrow \subseteq (\mathcal{S} \times T) \cup (T \times \mathcal{S})$, and $\forall t \in T, |{}^\bullet t| = |t^\bullet| = 1$,
- ${}^\bullet x = \{y \in \mathcal{S} \cup T : y \rightarrow x\}$ and sym. for t^\bullet
- $\lambda : T \rightarrow \Lambda$, transition labels operate as actions σ

Synchronization: product of automata (synchronous, parallel,...)

$$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \text{ (or } \mathcal{A}_1 \parallel \mathcal{A}_2 \text{) defined by}$$

- states $S = S_1 \times S_2$, $s_0 = (s_{1,0}, s_{2,0})$
- transitions

$$\begin{aligned}
 T = & \{(t_1, t_2) : \lambda_1(t_1) = \lambda_2(t_2)\} && \text{synchro. by shared labels} \\
 & \cup \{(t_1, \star_{s_2}) : \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} && \text{private to } \mathcal{A}_1 \\
 & \cup \{(\star_{s_1}, t_2) : \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} && \text{private to } \mathcal{A}_2
 \end{aligned}$$

Distributed automaton

Automaton: (new formulation, more convenient)

$$\mathcal{A} = (\mathcal{S}, T, \rightarrow, s_0, \lambda, \Lambda)$$

- flow $\rightarrow \subseteq (\mathcal{S} \times T) \cup (T \times \mathcal{S})$, and $\forall t \in T, |{}^\bullet t| = |t^\bullet| = 1$,
- ${}^\bullet x = \{y \in \mathcal{S} \cup T : y \rightarrow x\}$ and sym. for t^\bullet
- $\lambda : T \rightarrow \Lambda$, transition labels operate as actions σ

Synchronization: product of automata (synchronous, parallel,...)

$$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \text{ (or } \mathcal{A}_1 \parallel \mathcal{A}_2 \text{) defined by}$$

- states $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$, $s_0 = (s_{1,0}, s_{2,0})$
- transitions

$$\begin{aligned}
 T = & \{(t_1, t_2) : \lambda_1(t_1) = \lambda_2(t_2)\} && \text{synchro. by shared labels} \\
 & \cup \{(t_1, \star_{s_2}) : \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} && \text{private to } \mathcal{A}_1 \\
 & \cup \{(\star_{s_1}, t_2) : \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} && \text{private to } \mathcal{A}_2
 \end{aligned}$$

- flow \rightarrow in $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$, given by

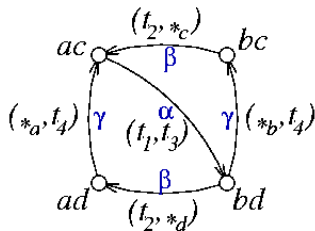
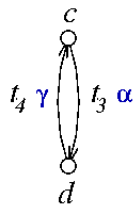
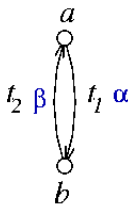
$$(s_1, s_2) \rightarrow (t_1, t_2) \rightarrow (s'_1, s'_2) \text{ iff } s_i \rightarrow_i t_i \rightarrow_i s'_i$$

$$(s_1, s_2) \rightarrow (t_1, \star_{s_2}) \rightarrow (s'_1, s_2) \text{ iff } s_1 \rightarrow_1 t_1 \rightarrow_1 s'_1$$

$$(s_1, s_2) \rightarrow (\star_{s_1}, t_2) \rightarrow (s_1, s'_2) \text{ iff } s_2 \rightarrow_2 t_2 \rightarrow_2 s'_2$$

- labels: $\Lambda = \Lambda_1 \cup \Lambda_2$, $\lambda : T \rightarrow \Lambda$ obvious

Example:



Remarks:

- The product of automata is associative; so $\mathcal{A} = \times_{i \in I} \mathcal{A}_i$ makes sense.

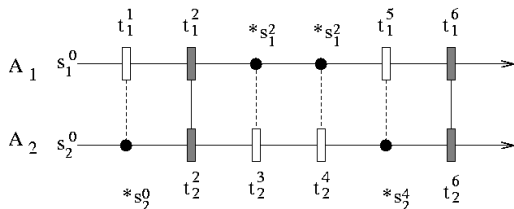
A **distributed automaton** : $\mathcal{A} = \times_{i \in I} \mathcal{A}_i$, I finite

- For label $\alpha \in \Lambda$, all components \mathcal{A}_i having α in their label set Λ_i must provide a transition $t_j \neq \star_{s_j}$, $\lambda_i(t_j) = \alpha$, to the synchronisation.
- Synchro may involve more than 2 components: more general than talking to a channel.

Language $\mathcal{L}(\mathcal{A})$: (sequential semantics)

Run $\omega = t^1 t^2 \dots t^n$ as usual, $t^k = (t_i^k)_{i \in I}$, poss. $t_i^k = \star s_i$

Example: for $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$



One would like to make this run a **partial order!**

Projection of a language on a (set of) component(s) \mathcal{A}_i

$$\mathcal{A} = \times_{i \in I} \mathcal{A}_i$$

$$\Pi_k(t) = \Pi_k[(t_i)_{i \in I}] = \begin{cases} \emptyset & \text{if } t_k = \star_{s_k} \\ t_k & \text{otherwise} \end{cases}$$

$$\Pi_k(\omega t) = \Pi_k(\omega) \Pi_k(t) \quad \omega \in T^*$$

$$\Pi_k(L) = \{\Pi_k(\omega), \omega \in L\}$$

Lemma

$$\Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$$

- Trivial, by definition of \mathcal{A} .
- Inclusion is strict in general (exo: make an example).
- $\Pi_i[\mathcal{L}(\mathcal{A})]$ represents runs of \mathcal{A}_i that remain possible in \mathcal{A} , given interactions with all the other \mathcal{A}_j , $j \neq i$.
- Definition and lemma extend trivially to $\Pi_J(\omega)$, projection on $\mathcal{A}_J = \times_{j \in J} \mathcal{A}_j$ for $J \subseteq I$.

Projection of a language on a (set of) component(s) \mathcal{A}_i

$$\mathcal{A} = \times_{i \in I} \mathcal{A}_i$$

$$\Pi_k(t) = \Pi_k[(t_i)_{i \in I}] = \begin{cases} \emptyset & \text{if } t_k = \star_{s_k} \\ t_k & \text{otherwise} \end{cases}$$

$$\Pi_k(\omega t) = \Pi_k(\omega) \Pi_k(t) \quad \omega \in T^*$$

$$\Pi_k(L) = \{\Pi_k(\omega), \omega \in L\}$$

Lemma

$$\Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$$

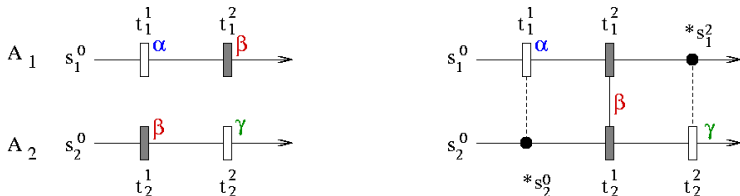
- Trivial, by definition of \mathcal{A} .
- Inclusion is strict in general (exo: make an example).
- $\Pi_i[\mathcal{L}(\mathcal{A})]$ represents runs of \mathcal{A}_i that remain possible in \mathcal{A} , given interactions with all the other \mathcal{A}_j , $j \neq i$.
- Definition and lemma extend trivially to $\Pi_J(\omega)$, projection on $\mathcal{A}_J = \times_{j \in J} \mathcal{A}_j$ for $J \subseteq I$.

Product of languages: $L_i \subseteq \mathcal{L}(\mathcal{A}_i)$ and $\mathcal{A} = \times_{i \in I} \mathcal{A}_i$

$$\times_{i \in I}^L L_i = \bigcap_{i \in I} \Pi_i^{-1}(L_i)$$

Remarks:

- \times^L is associative (exo).
- The product of two words is a *language*, that can be empty.



- There exists a recursive algorithm to compute the product of 2 words (exo).

Proposition (Factorization property on languages)

$$\mathcal{L}(\mathcal{A}) = \times_{i \in I}^L \mathcal{L}(\mathcal{A}_i)$$

- Remains true with a product of $\mathcal{L}(\mathcal{A}_J)$ when the J form a partition of I .

Proposition (Minimal factors)

$$\mathcal{L}(\mathcal{A}) = \times_{i \in I}^L \Pi_i[\mathcal{L}(\mathcal{A})]$$

- $\Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$ is a minimal factor of $\mathcal{L}(\mathcal{A})$ in $\mathcal{L}(\mathcal{A}_i)$,
- removing any word in $\Pi_i[\mathcal{L}(\mathcal{A})]$ prevents recovering $\mathcal{L}(\mathcal{A})$ in the product.
- Remains true with projections Π_J when the J form a partition of I .

Proposition (Factorization property on languages)

$$\mathcal{L}(\mathcal{A}) = \times_{i \in I}^L \mathcal{L}(\mathcal{A}_i)$$

- Remains true with a product of $\mathcal{L}(\mathcal{A}_J)$ when the J form a partition of I .

Proposition (Minimal factors)

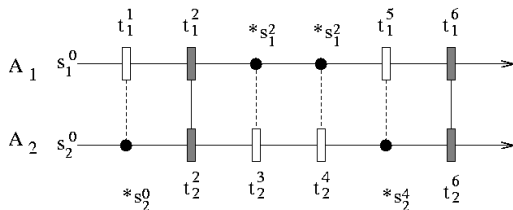
$$\mathcal{L}(\mathcal{A}) = \times_{i \in I}^L \Pi_i[\mathcal{L}(\mathcal{A})]$$

- $\Pi_i[\mathcal{L}(\mathcal{A})] \subseteq \mathcal{L}(\mathcal{A}_i)$ is a minimal factor of $\mathcal{L}(\mathcal{A})$ in $\mathcal{L}(\mathcal{A}_i)$,
- removing any word in $\Pi_i[\mathcal{L}(\mathcal{A})]$ prevents recovering $\mathcal{L}(\mathcal{A})$ in the product.
- Remains true with projections Π_J when the J form a partition of I .

Mazurkiewicz traces

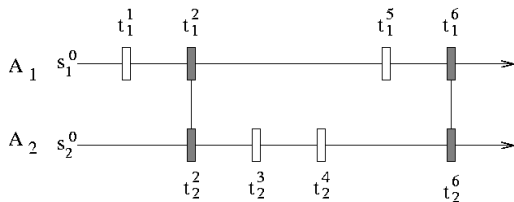
A first model of true concurrency semantics on trajectories of

$$\mathcal{A} = \times_{i \in I} \mathcal{A}_i$$



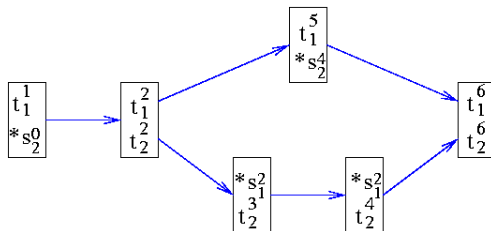
Mazurkiewicz traces

A first model of true concurrency semantics on trajectories of $\mathcal{A} = \times_{i \in I} \mathcal{A}_i$



Mazurkiewicz traces

A first model of true concurrency semantics on trajectories of $\mathcal{A} = \times_{i \in I} \mathcal{A}_i$



Idea: allow the permutation of successive events that live on different components.

(In)Dependency: on transitions of T

$$t D t' \Leftrightarrow \exists i \in I, \Pi_i(t) \neq \emptyset \neq \Pi_i(t')$$

i.e. the firing of t and t' will be ordered by some component \mathcal{A}_i

Equivalent runs:

- We define relation R on runs of \mathcal{A}

$$\omega_1 t t' \omega_2 R \omega_1 t' t \omega_2 \Leftrightarrow \neg(t D t')$$

- \equiv is the equivalence relation generated by R on runs of \mathcal{A} .
- A **trace** of \mathcal{A} is an equivalence class $[\omega]$ of runs for \equiv
- \equiv is the smallest congruence in T^* s.t. $\neg(t D t') \Rightarrow t t' \equiv t' t$

Idea: allow the permutation of successive events that live on different components.

(In)Dependency: on transitions of T

$$t D t' \Leftrightarrow \exists i \in I, \Pi_i(t) \neq \emptyset \neq \Pi_i(t')$$

i.e. the firing of t and t' will be ordered by some component \mathcal{A}_i

Equivalent runs:

- We define relation R on runs of \mathcal{A}

$$\omega_1 t t' \omega_2 R \omega_1 t' t \omega_2 \Leftrightarrow \neg(t D t')$$

- \equiv is the equivalence relation generated by R on runs of \mathcal{A} .
- A **trace** of \mathcal{A} is an equivalence class $[\omega]$ of runs for \equiv
- \equiv is the smallest congruence in T^* s.t. $\neg(t D t') \Rightarrow t t' \equiv t' t$

Properties: in $\omega = t^1 t^2 \dots t^n$,

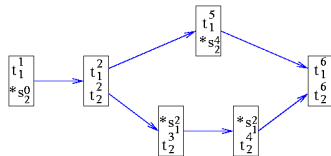
- **Concurrency** of events

$$t^i \perp t^{i+j} \Leftrightarrow \neg(t^i D t^{i+k}), \quad 1 \leq k \leq j$$

- **Causal dependence** of events

$$t^i < t^{i+j} \Leftrightarrow \neg(t^i \perp t^{i+j})$$

- **Hasse diagram**, on events $\{t^1, \dots, t^n\}$ of ω :
the transitive reduction of the partial order $(\{t^1, \dots, t^n\}, <)$



- $[\omega]$ is the set of all linear extensions of $(\{t^1, \dots, t^n\}, <)$

Properties: in $\omega = t^1 t^2 \dots t^n$,

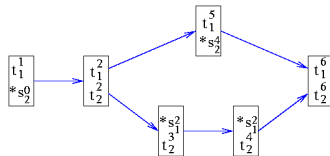
- **Concurrency** of events

$$t^i \perp t^{i+j} \Leftrightarrow \neg(t^i D t^{i+k}), \quad 1 \leq k \leq j$$

- **Causal dependence** of events

$$t^i < t^{i+j} \Leftrightarrow \neg(t^i \perp t^{i+j})$$

- **Hasse diagram**, on events $\{t^1, \dots, t^n\}$ of ω :
the transitive reduction of the partial order $(\{t^1, \dots, t^n\}, <)$



- $[\omega]$ is the set of all linear extensions of $(\{t^1, \dots, t^n\}, <)$

Properties: in $\omega = t^1 t^2 \dots t^n$,

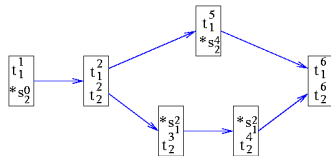
- **Concurrency** of events

$$t^i \perp t^{i+j} \Leftrightarrow \neg(t^i D t^{i+k}), \quad 1 \leq k \leq j$$

- **Causal dependence** of events

$$t^i < t^{i+j} \Leftrightarrow \neg(t^i \perp t^{i+j})$$

- **Hasse diagram**, on events $\{t^1, \dots, t^n\}$ of ω :
the transitive reduction of the partial order $(\{t^1, \dots, t^n\}, <)$



- $[\omega]$ is the set of all linear extensions of $(\{t^1, \dots, t^n\}, <)$

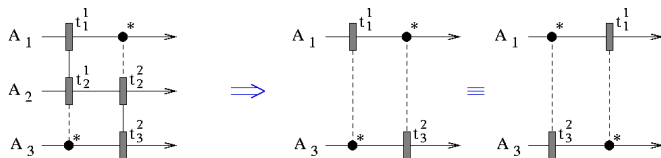
Language of traces of \mathcal{A} :

- a sub-language \bar{L} of $\mathcal{L}(\mathcal{A})$ stable by \equiv
- any sub-language of $\mathcal{L}(\mathcal{A}_i)$ is a trace language, by def. of D

Projection:

- Does the projection defined on words preserve trace languages?

NO ! Causality relations may be lost. The projection of an equivalence classe may not be an equivalence class.



- Projection of a trace language

$$\bar{\Pi}_J(L) = \cup_{\omega \in L} [\Pi_J(\omega)]$$

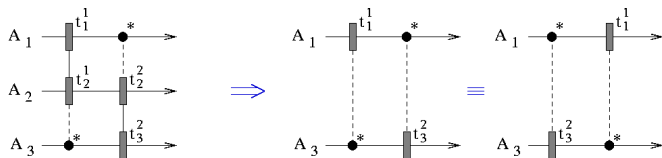
Language of traces of \mathcal{A} :

- a sub-language \bar{L} of $\mathcal{L}(\mathcal{A})$ stable by \equiv
- any sub-language of $\mathcal{L}(\mathcal{A}_i)$ is a trace language, by def. of D

Projection:

- Does the projection defined on words preserve trace languages?

NO ! Causality relations may be lost. The projection of an equivalence classe may not be an equivalence class.



- Projection of a trace language

$$\bar{\Pi}_J(L) = \cup_{\omega \in L} [\Pi_J(\omega)]$$

Product of trace languages:

- \bar{L}_J, \bar{L}_K trace languages of $\mathcal{A}_J, \mathcal{A}_K$ resp., $J \cap K = \emptyset$

$$\bar{L}_J \times^T \bar{L}_K = \bar{\Pi}_J^{-1}(\bar{L}_J) \cap \bar{\Pi}_K^{-1}(\bar{L}_K)$$

- \times^T is associative
- we have the same results as before

$$\begin{aligned} \mathcal{L}(\mathcal{A}) &= \times_{i \in I}^T \mathcal{L}(\mathcal{A}_i) \\ &= \times_{i \in I}^T \bar{\Pi}_i[\mathcal{L}(\mathcal{A})] \end{aligned}$$

and this extends to projections on groups of components \mathcal{A}_J

Drawbacks of this approach...

- Traces are not convenient to handle...
- We have too many transitions:
 $(t_1, \star_{s_2}), (t_1, \star_{s'_2}), \dots$ instead of simply “ t_1 ”

We examine now a simpler framework that has this property.

Petri nets

Another way of composing automata $\mathcal{A}_i = (S_i, T_i, \rightarrow_i, s_{i,0}, \lambda_i, \Lambda_i)$, that yields a **Petri net**.

Product: $\mathcal{N} = \mathcal{A}_1 \times \mathcal{A}_2 = (P, T, \rightarrow, P_0, \lambda, \Lambda)$

- Places: $P = S_1 \uplus S_2$ *disjoint union*
- Transitions:

$$\begin{aligned}
 T &= \{(t_1, t_2) : \lambda_1(t_1) = \lambda_2(t_2)\} && \text{synchro. by shared labels} \\
 &\cup \{(t_1, \star) : \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} && \text{private to } \mathcal{A}_1 \\
 &\cup \{(\star, t_2) : \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} && \text{private to } \mathcal{A}_2
 \end{aligned}$$

- Flow: \rightarrow defined by

$$\begin{aligned}
 \bullet(t_1, t_2) &= \bullet t_1 \uplus \bullet t_2 \\
 (t_1, t_2)^\bullet &= t_1^\bullet \uplus t_2^\bullet
 \end{aligned}$$

Defining and computing with these objects: next time !