MAD

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

-- 5/5 --

download slides at

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

Today…

- A new model for distributed systems: Petri nets

- Main features
  - concurrency naturally (graphically) encoded
  - runs easily encoded as partial orders of events
  - languages encoded as branching processes and unfoldings (tightly related to the formal notion of event structure)

What do we have so far ?

Model

- network of automata $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$
- language = set of runs, a run = a sequence of events
- factorization $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times \ldots \times \mathcal{L}(\mathcal{A}_N) \subseteq \Sigma^*$
- a Mazurkiewicz trace :
  - one way to recover concurrency, a run becomes a partial order of events
  - encoding of traces as tuples of local words $w \in w_1 \times \ldots \times w_N$

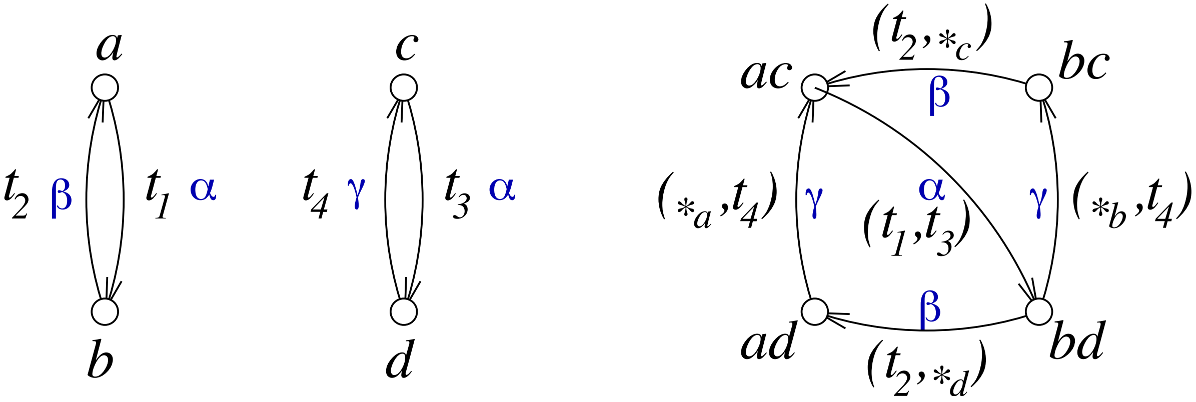
Algebra

- projection & product on (networks of) automata and languages
- rich properties $\Rightarrow$ distributed/modular computations in this algebra
- working with factorized forms is like working with traces
- application: distributed diagnosis, distributed planning

Limitations

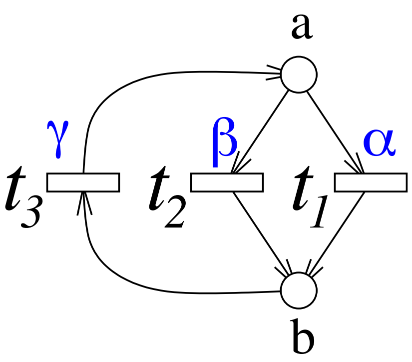
- the product of automata does not make concurrency visible (creates concurrency diamonds), and leads to state explosion
- the natural sequential semantics (runs as sequences of events) does not capture well concurrency
- traces are an indirect way to recover a true concurrency semantics from sequences, where “$a \prec b$ and $b \prec a$” is made equivalent to “$b \perp a$”; one may need to distinguish these situations :

“I can go first” $\wedge$ “you can go first” $\not\Rightarrow$ “we can go at the same time”



Petri nets

change of notation



- automaton $\mathcal{A} = (S, T, \Sigma, s_0, S_F)$
    - transitions set $T \subseteq S \times \Sigma \times S$
    - one transition $t = (s, \alpha, s') = ({}^\bullet t, \sigma(t), t^\bullet)$

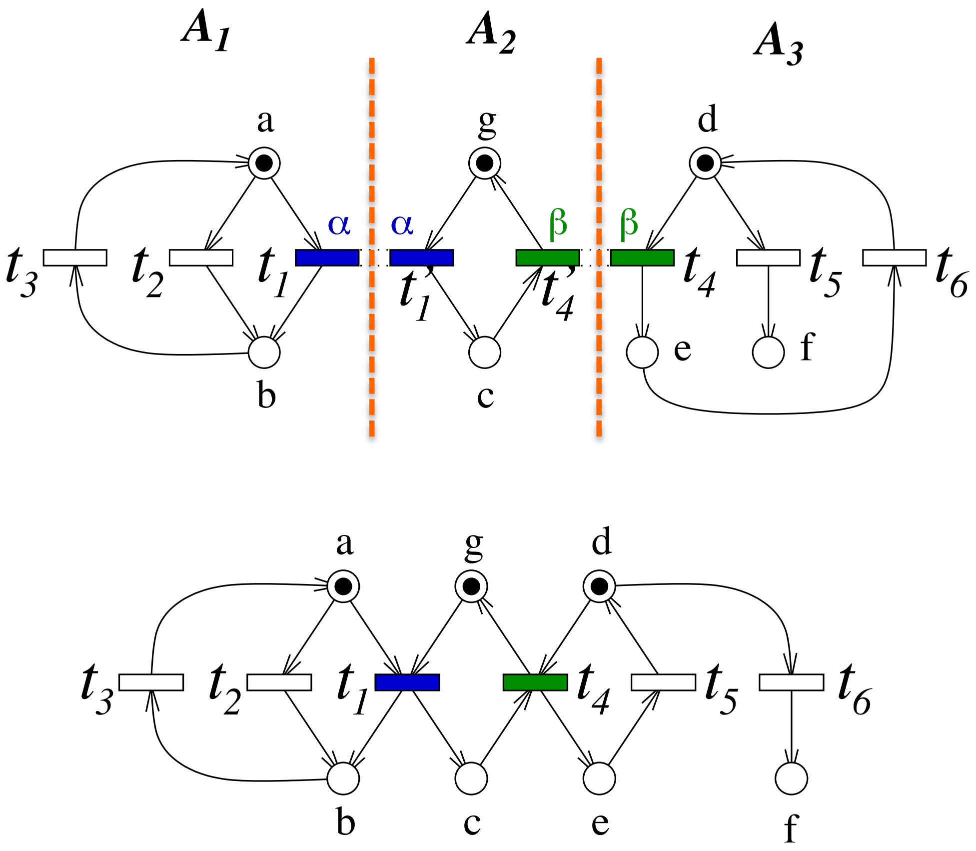
- new notation (Petri Net inspired) $\mathcal{A} = (S, T, \rightarrow, s_0, \lambda, \Lambda)$
    - S, T, Λ are finite sets of states (places), transitions, labels
    - flow connects transitions and states $\rightarrow \subseteq (S \times T) \cup (T \times S)$
    - preset $\forall x \in S \cup T, \; {}^\bullet x = \{y \in S \cup T \; : \; y \rightarrow x\}$ and sym. for postset $x^\bullet$
    - labeling of transitions $\lambda : T \rightarrow \Lambda$

Product

$$\mathcal{N} = \mathcal{A}_1 \times \mathcal{A}_2 = (P, T, \rightarrow, P_0, \lambda, \Lambda) \text{ where } \mathcal{A}_i = (S_i, T_i, \rightarrow_i, s_{0,i}, \lambda_i, \Lambda_i)$$

- **Places:**
  - disjoint union (not the product !) $P = S_1 \uplus S_2$
  - initial places $P_0 = \{s_{0,1}, s_{0,2}\}$
- **Transitions**: a single copy of each private transition
  - synchro on common labels $T = \{(t_1, t_2) \ : \ \lambda_1(t_1) = \lambda_2(t_2)\}$
  - private transitions in 1st comp. $\cup \ \{(t_1, \star) \ : \ \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\}$
  - private transitions in 2nd comp. $\cup \ \{(\star, t_2) \ : \ \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\}$
- **Flow**:
  - $\rightarrow$ is defined by ${}^\bullet(t_1, t_2) = {}^\bullet t_1 \uplus {}^\bullet t_2$ and $(t_1, t_2)^\bullet = t_1^\bullet \uplus t_2^\bullet$
  - where $\star^\bullet = \emptyset$ and ${}^\bullet\star = \emptyset$

Example



Remarks

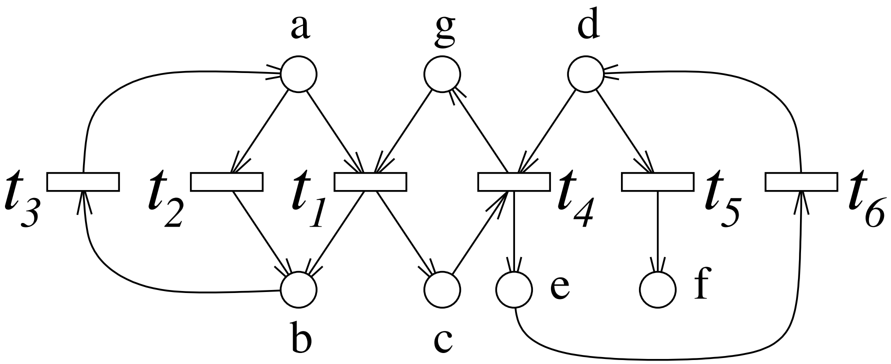
- in general, as for the product of automata, the association of transitions is not one to one
- this definition of product extends to (safe) Petri nets…
- …and makes the product associative

Dynamics

in a Petri net $\mathcal{N} = \mathcal{A}_1 \times \mathcal{A}_2 = (P, T, \rightarrow, P_0, \lambda, \Lambda)$

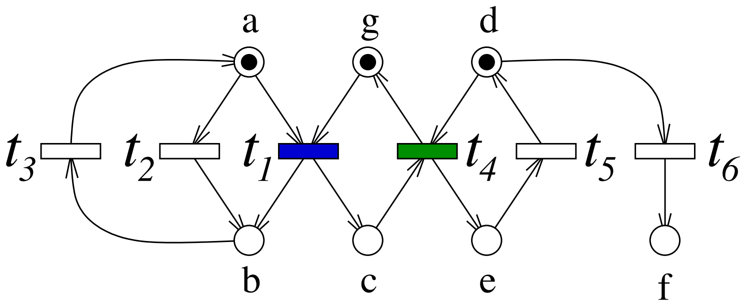
- **Marking:**
  - a function $m : P \rightarrow \mathbb{N}$
  - assigns a number of tokens to each place
  - notation : $m \subseteq P$ if places contain at most one token (safe net)

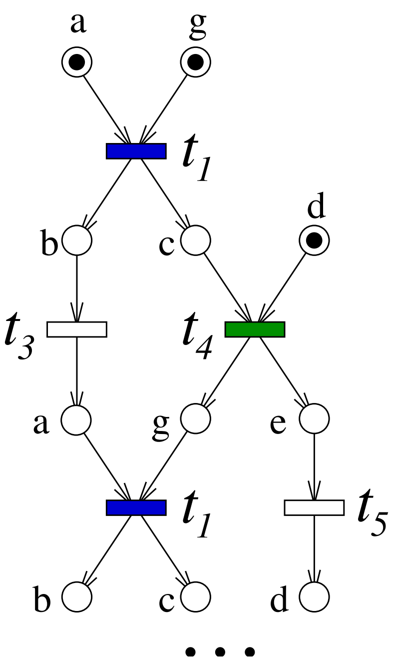
- **Enabling** of a transition
  - transition $t \in T$ is enabled at marking $m \subseteq P$ iff $^\bullet t \subseteq m$
  - the resources/tokens needed by t are present in the current marking
- **Firing** of a transition
  - it changes the current marking m into m' with $m' = m - {}^\bullet t + t^\bullet$
  - t consumes tokens in its present, and produces some in its postset



True concurrency semantics

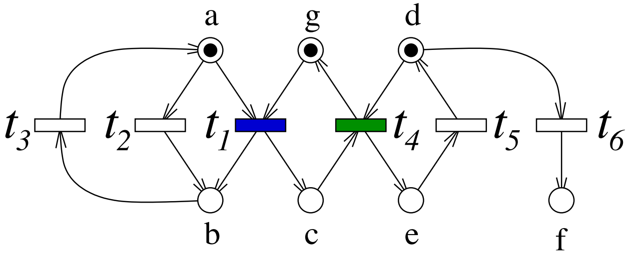
- **sequential semantics**
  - a run = a sequence of transition firings, rooted at $m_0 = P_0$
  - imposes the interleaving of concurrent events
  - different interleavings = different runs
- **true concurrency semantics**
  - a run is a partial order of events
  - encoded as another Petri net, without circuits, called a **configuration**



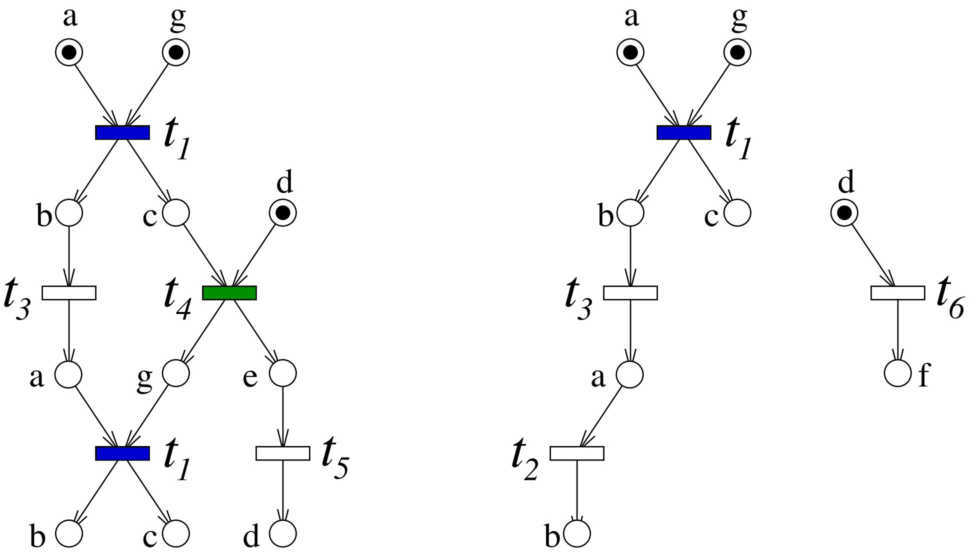


Unfoldings

A safe Petri net…

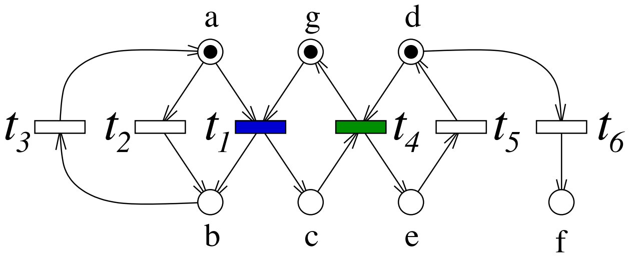


…and two of its configurations (runs), as partially ordered events

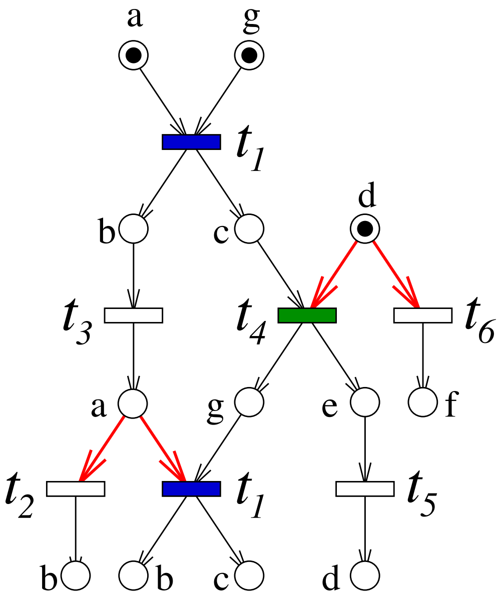


Unfoldings

A safe Petri net...



merging common prefixes yields an **occurrence net**



Occurrence net

- a special Petri net $\mathcal{O} = (C, E, \rightarrow, C_0, \lambda, \Lambda)$
- places are called conditions, transitions are called events
- the flow $\rightarrow$ is acyclic (partial ordering)
- and this partial order is well founded

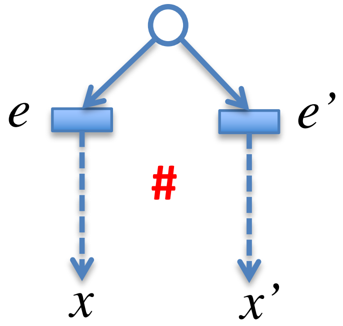
$$\forall x \in C \cup E, \quad |\{y \in C \cup E \, : y \rightarrow^* x\}| < \infty$$

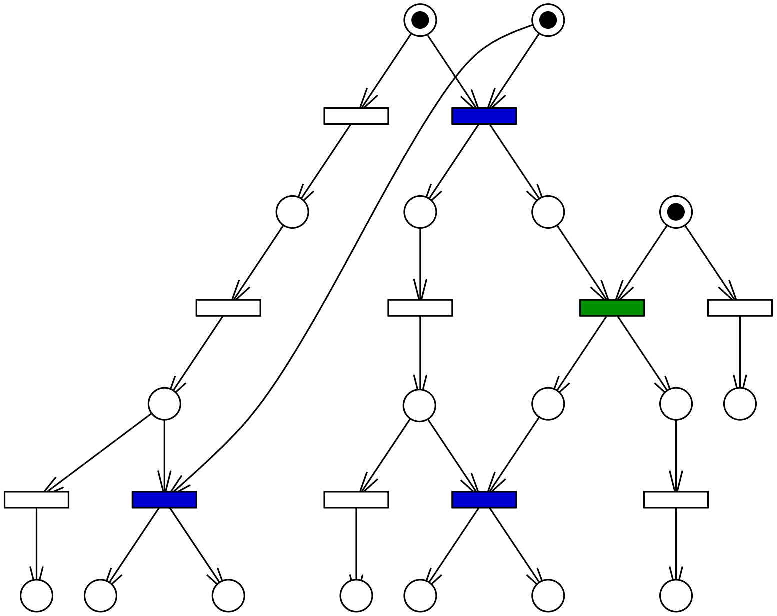
- every condition has a unique cause or is minimal

$$\forall c \in C, \quad |{}^\bullet c| \leq 1 \qquad \text{and} \qquad C_0 = \{c \in C, {}^\bullet c = \emptyset\}$$

- no event is in self-conflict

$$x \# x' \quad \Leftrightarrow \quad \exists e \neq e' \in E, \;\; {}^\bullet e \cap {}^\bullet e' \neq \emptyset, \;\; e \rightarrow^* x, \;\; e' \rightarrow^* x'$$





concurrency $x \perp y \;\Leftrightarrow\; \neg(x \rightarrow^* y) \wedge \neg(y \rightarrow^* x) \wedge \neg(x \# y)$

represents nodes that can lie in the same configuration

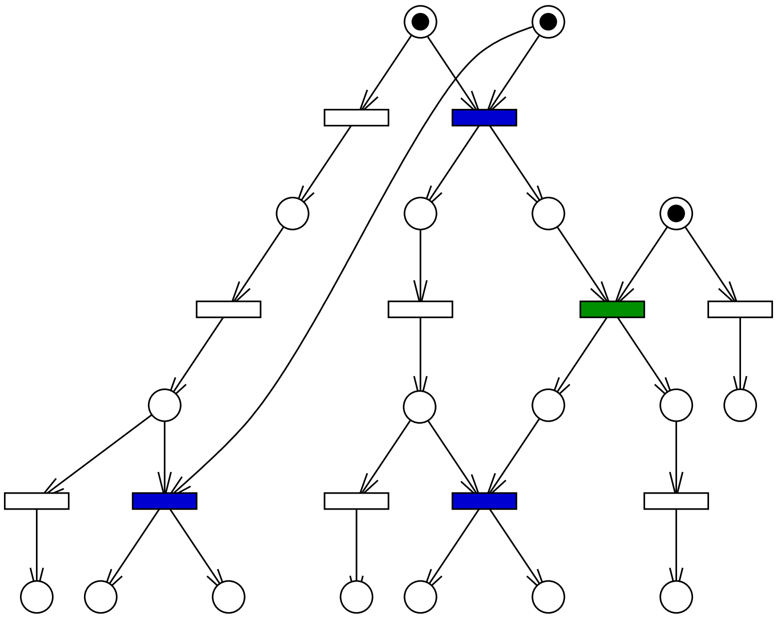
co-set : $X \subseteq C$ such that $\forall c, c' \in X, \;\; c \perp c'$

represents resources (tokens) that are available at the same time in some run/configuration

cut : a maximal co-set for $\subseteq$

prefix : $\mathcal{O}' = (C', E', \rightarrow', C_0, \lambda', \Lambda) \sqsubseteq \mathcal{O}$

iff $\mathcal{O}'$ is a causally closed sub-net of $\mathcal{O}$, containing C_0 and $E'^{\bullet}$



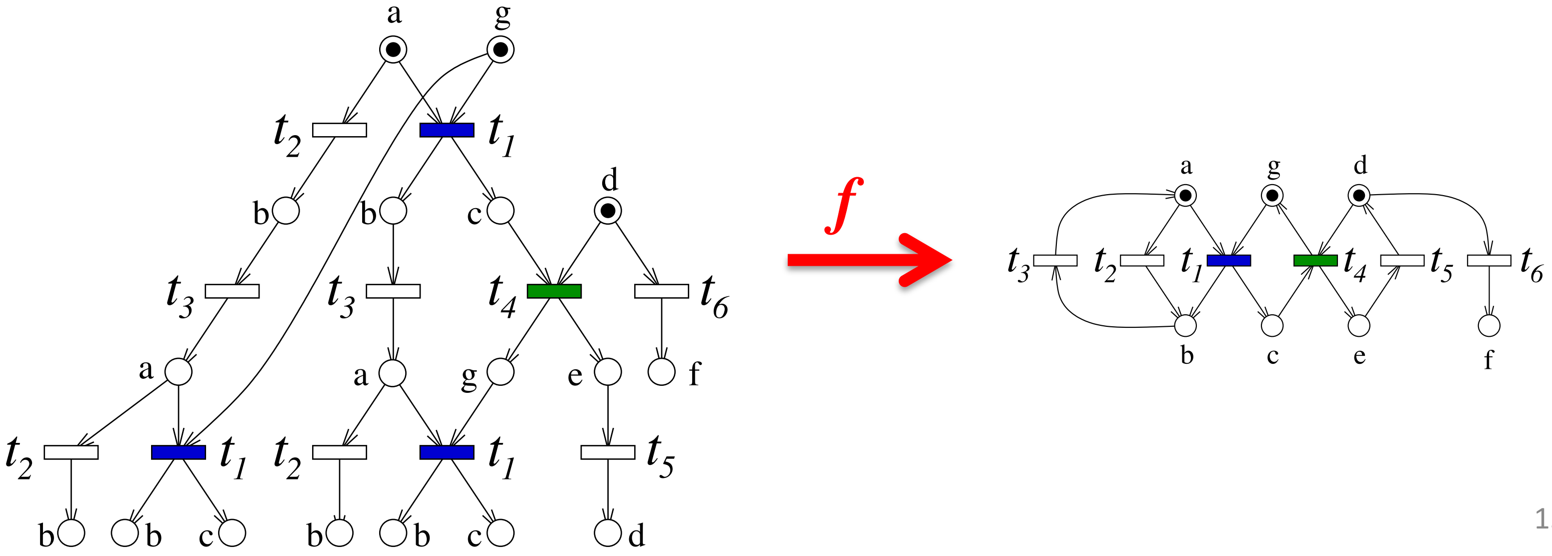
configuration : denoted κ , a conflict-free prefix of $\mathcal{O}$

local configuration : [e] = smallest configuration containing event e,
= causal past of e

Lem : relating cuts and configurations
X is a cut of $\mathcal{O}$ $\Leftrightarrow$ $\exists \kappa = (C', E', \ldots)$ such that $X = max(C')$

Branching process

- a branching process of net $\mathcal{N}$ is a pair $(\mathcal{O}, f)$
  where $\mathcal{O}$ is an occurrence net,
  and $f : \mathcal{O} \to \mathcal{N}$ a morphism of nets (a total function)
- f "labels" conditions/events of $\mathcal{O}$ by places/transitions of $\mathcal{N}$
  it turns a configuration of $\mathcal{O}$ into a run of $\mathcal{N}$
- parsimony : $\forall e, e' \in E, \quad {}^\bullet e = {}^\bullet e' \ \wedge \ f(e) = f(e') \ \Rightarrow \ e = e'$
- if X = maximal conditions in configuration κ (X forms a cut)
  then $f(X)$ is the marking of $\mathcal{N}$ produced by run κ



Unfolding

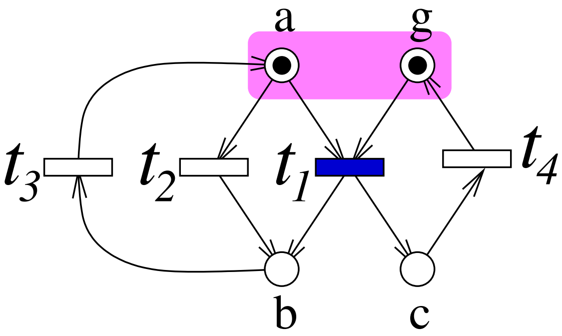
Thm : there exists a unique branching process $(\mathcal{U}_\mathcal{N}, f_\mathcal{N})$ of $\mathcal{N}$ maximal for prefix inclusion $\sqsubseteq$, it is called the unfolding of $\mathcal{N}$

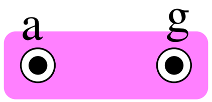
Proof : main idea is to define the union of branching processes, a little technical, but not difficult (see refs. at the end of the lesson).

Algorithm (unfolding)

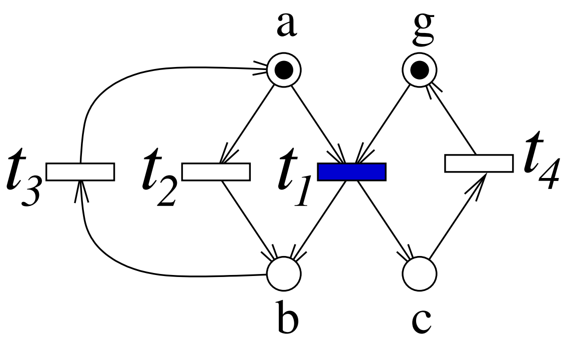
- init
  - $C = C_0$, isomorphic to P_0 through f
  - $E = \emptyset, \quad \rightarrow = \emptyset$
- repeat until stability (extension with a new event)
  - for a coset $X \subseteq C$ and transition t such that $f(X) = {}^\bullet t$
  - create event $e \in E$ (if it does not already exist) such that ${}^\bullet e = X,\ f(e) = t$
  - create new conditions $X' = e^\bullet \subset C$ and extend f so that $f(X') = t^\bullet$

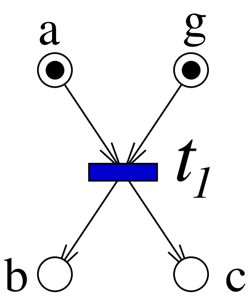
Example



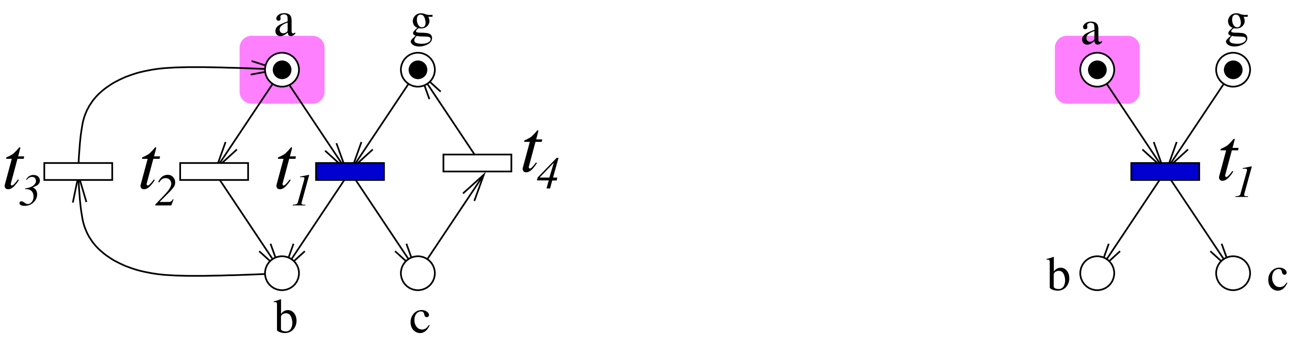


Example

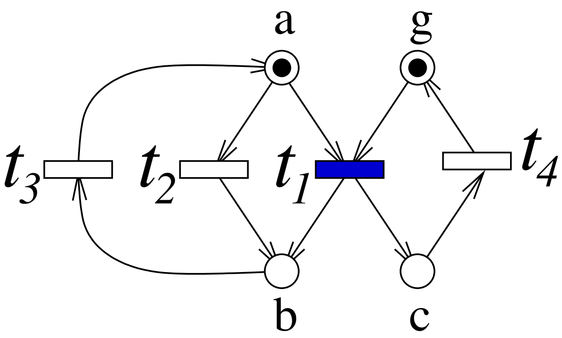


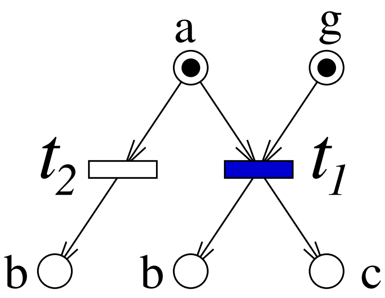


Example

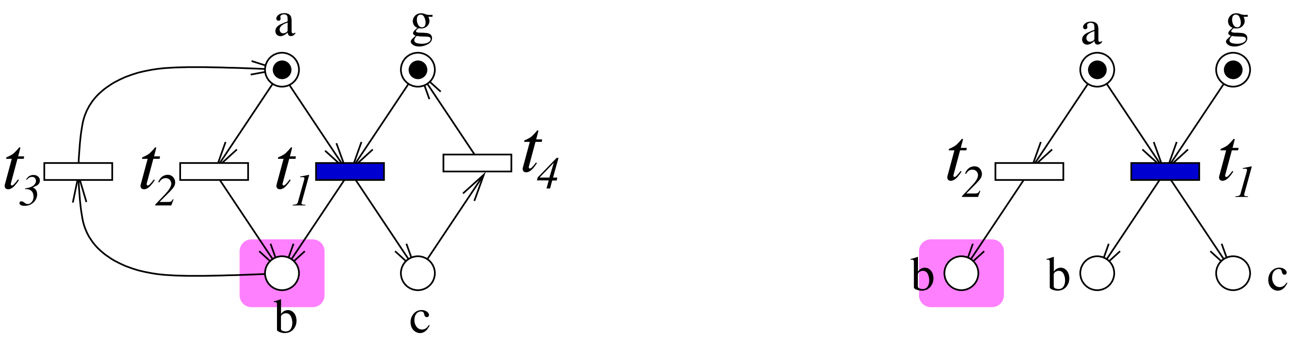


Example

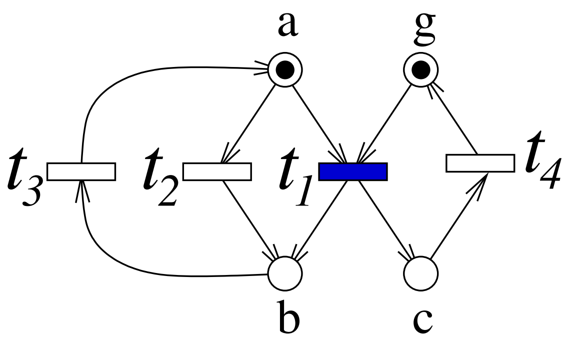


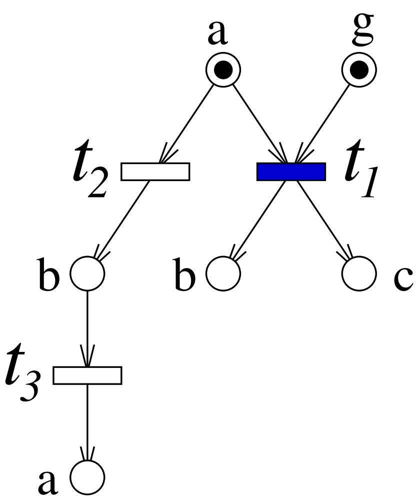


Example

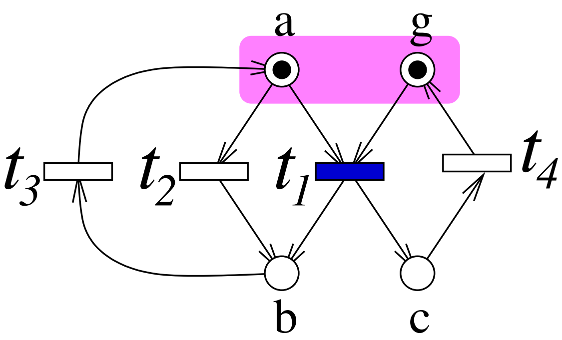


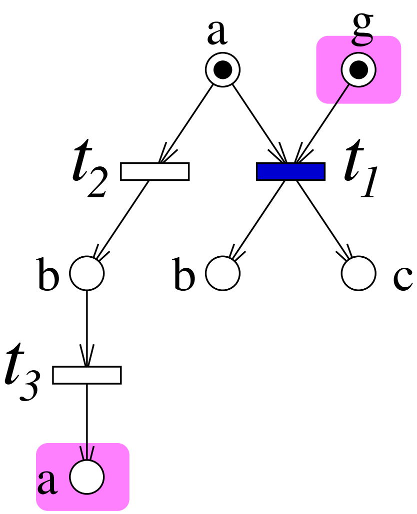
Example



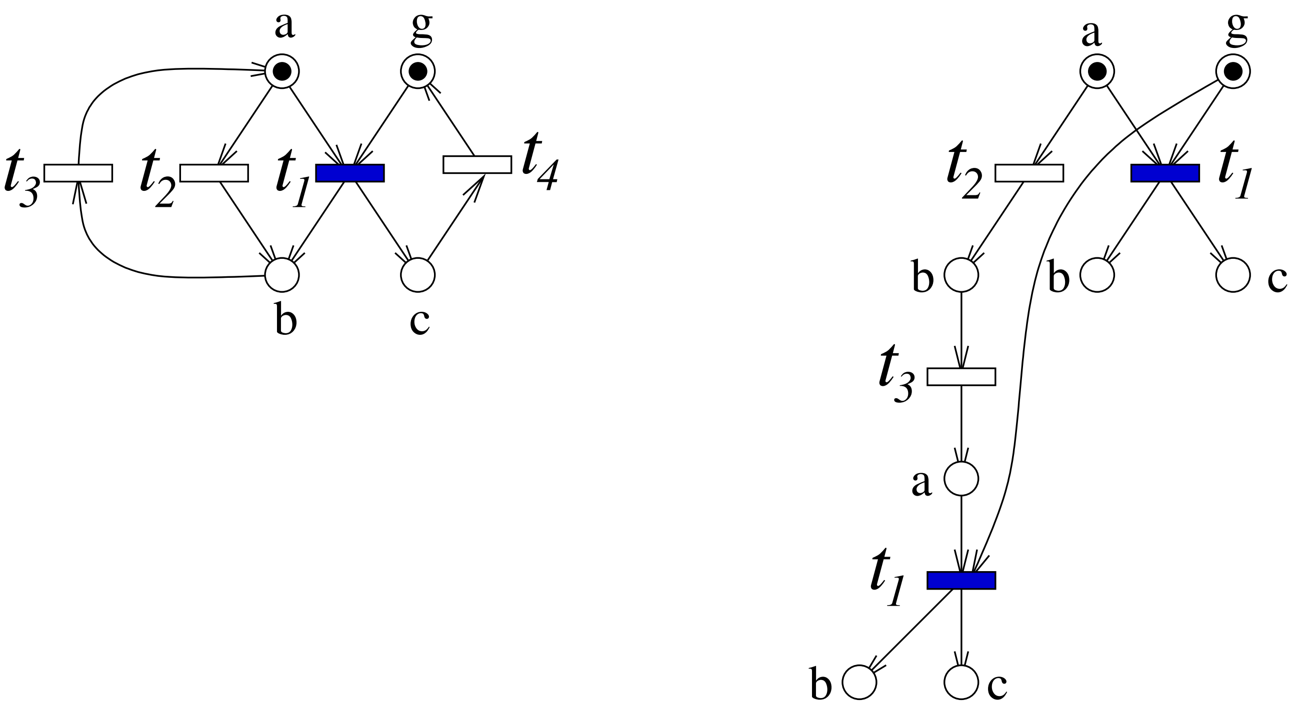


Example

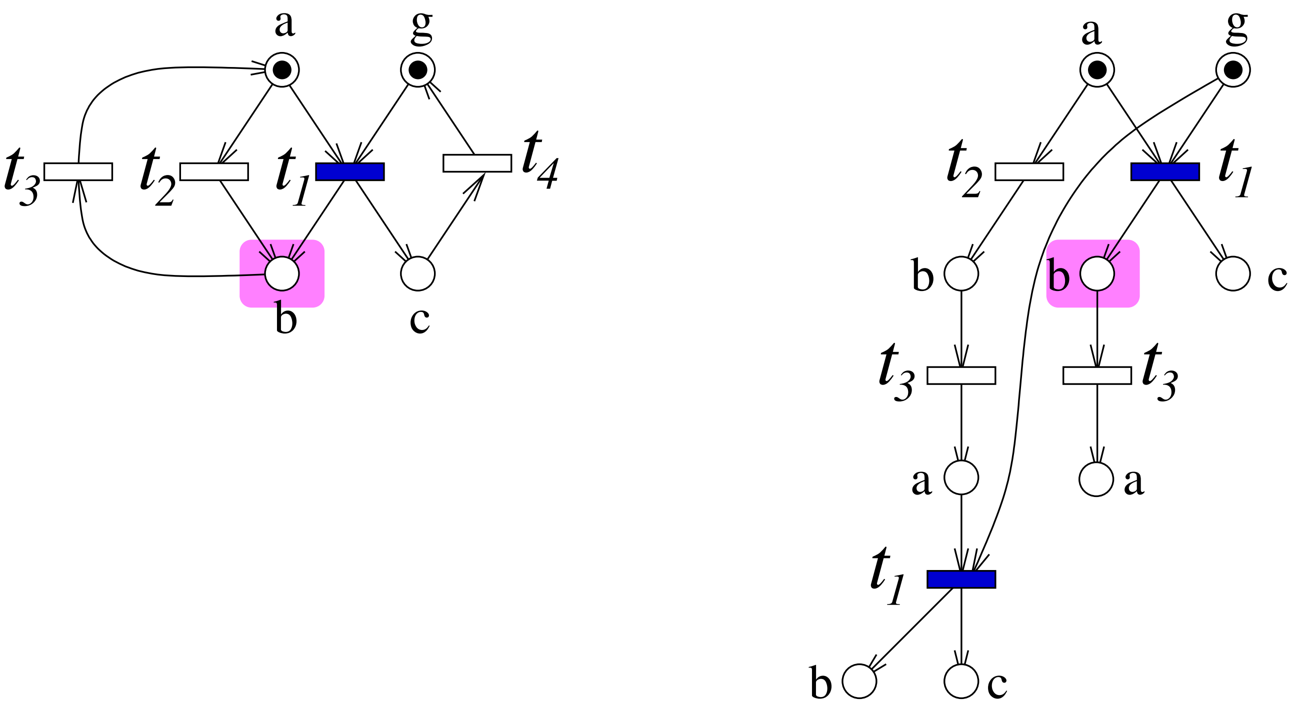




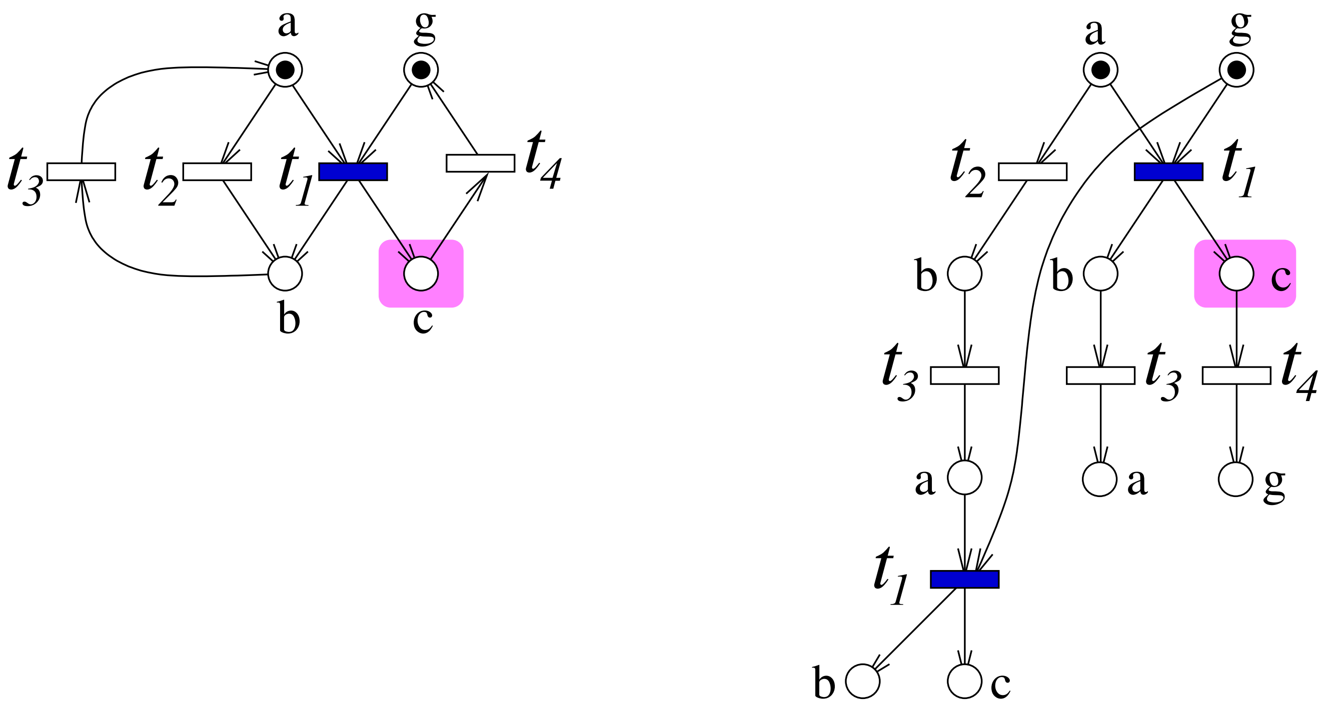
Example



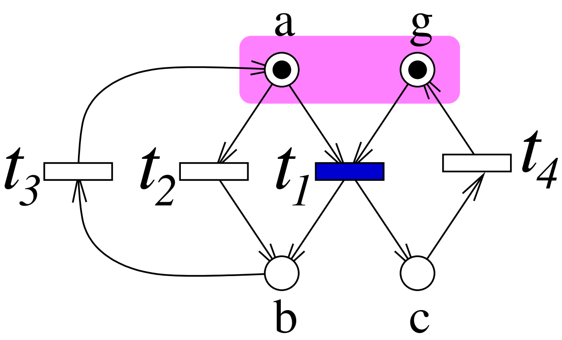
Example

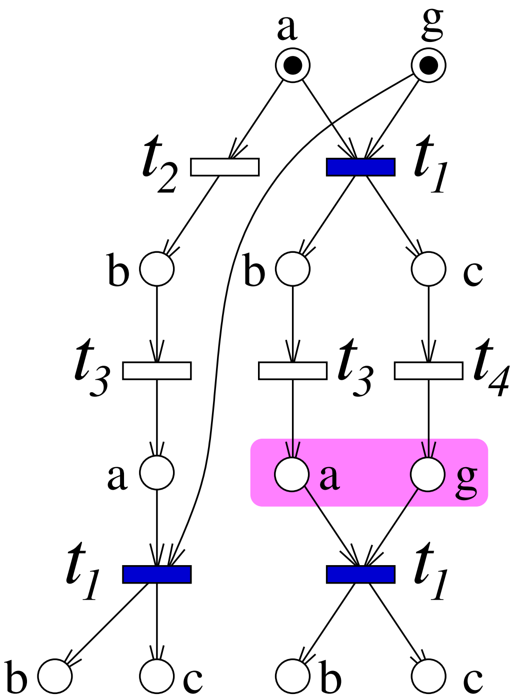


Example



Example





Application of the unfolding

Reachability/coverability test

- one wishes to know if there exists an accessible marking m in net $\mathcal{N}$ where each place of $Q \subseteq P$ holds a token, i.e. $Q \subseteq m$
- by creating in $\mathcal{N}$ a new transition t with $Q = {}^\bullet t$ this amounts to checking if t is accessible

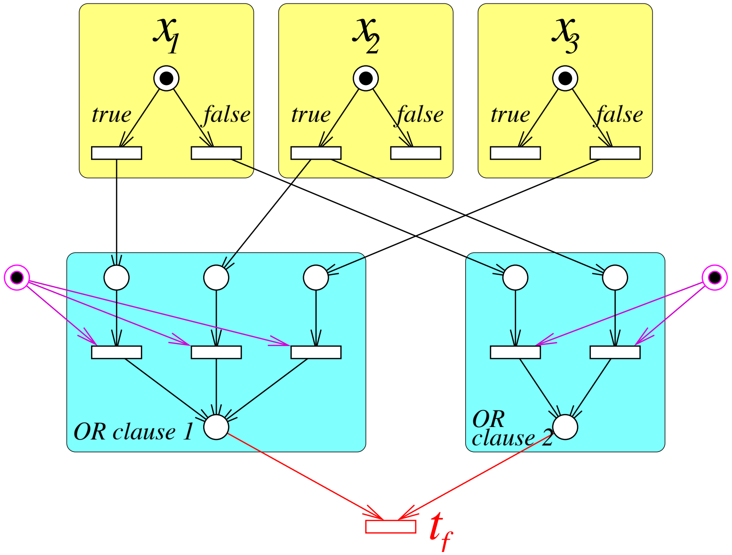
Questions

1. what is the complexity of this test ?
2. how far should one go in the computation of the unfolding ?

Thm : the reachability/coverability test (co-set construction) is NP-complete.

Proof: by reduction of SAT problems (at least 3-SAT)

example : encoding SAT problem $(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2)$



- The complexity of unfolding this net (before t_f) is polynomial, so the complexity of finding a co-set where t_f is firable is NP-hard.
- As building an unfolding requires finding co-sets, one must rely on SAT solvers (which modern unfolders do).

Finite complete prefix

Idea: A prefix $\mathcal{O} \sqsubseteq \mathcal{U}_{\mathcal{N}}$ is said to be complete if all reachable markings in $\mathcal{N}$ are represented as (the image of) a cut in $\mathcal{O}$.
(One wishes to avoid useless repetitions of similar patterns in $\mathcal{O}$)

More formally: $\mathcal{O} \sqsubseteq \mathcal{U}_{\mathcal{N}}$ is complete iff

- $\forall m$ reachable marking in $\mathcal{N}$, it appears in the prefix

$$\exists \kappa \in \mathcal{O} \ : \ m = Mark(\kappa) = f_{\mathcal{N}}(\max(\kappa))$$

- $\forall t \in T, \ m[t\rangle m'$, i.e. t firable from m, it appears as an event on top of marking m

$$\exists \kappa, \kappa' \in \mathcal{O} \ : \ m = Mark(\kappa), \ \kappa' = \kappa \oplus \{e\}, \ f_{\mathcal{N}}(e) = t$$

How to build a finite complete prefix ?

Naive idea:

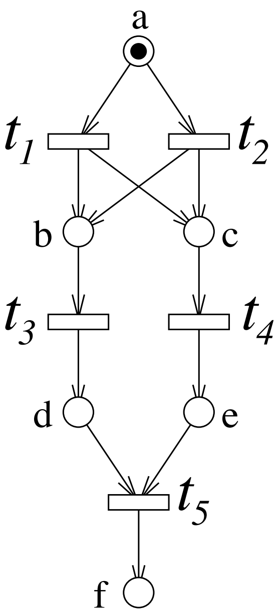
- apply the unfolding algorithm, and stop at event e when the marking produced by *[e]* is already present in the prefix:

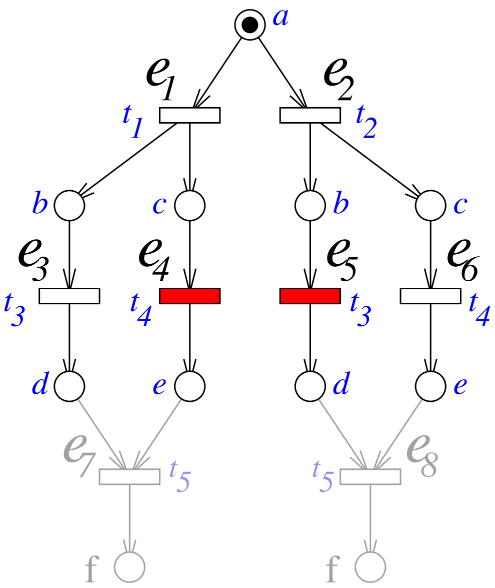
$$\exists \kappa \in \mathcal{O} \ : \ \mathrm{Mark}(\kappa) = \mathrm{Mark}([e])$$

- this makes e a cut-off event, on top of which no more event will be added

Problem: it generally yields an incomplete prefix…

<u>example</u> : stop events in red, firing of t_5 not seen





Solution: break the symmetry, by favoring some configurations for extension

Adequate order : $\prec$ on (local) configurations *[e]*

- well founded partial order (finite number of predecessors)
- refines prefix inclusion : $\kappa \sqsubset \kappa' \Rightarrow \kappa \prec \kappa'$
- preserved by isomorphic extensions :

  $$\kappa \prec \kappa' \wedge \mathrm{Mark}(\kappa) = \mathrm{Mark}(\kappa') \Rightarrow \kappa \oplus e \prec \kappa' \oplus e' \text{ where } f_{\mathcal{N}}(e) = f_{\mathcal{N}}(e')$$

Examples

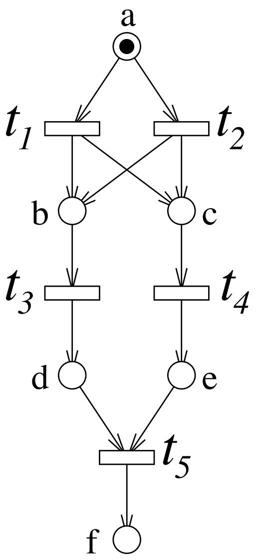
1. take for $\prec$ the prefix inclusion $\sqsubset$
2. $\prec$ defined by the number of events (total order, proposed by McMillan)
3. take for $\prec$ the lexicographic order, when net $\mathcal{N}$ is made of several components,
   by ordering components, and counting events in each component,
   as in Mattern's vector clocks (partial order, proposed by Esparza)

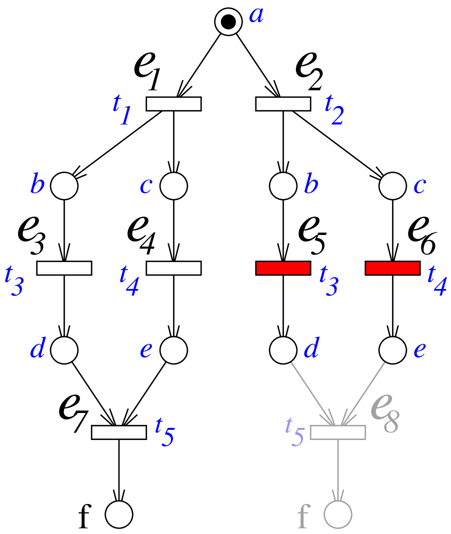
Cut-off event : event e is a cut-off event in the BP $(\mathcal{O}, f)$ of $\mathcal{N}$ iff there exists another event e' in $(\mathcal{O}, f)$ such that

$$\mathrm{Mark}([e']) = \mathrm{Mark}([e]) \quad \wedge \quad [e'] \prec [e]$$

Example (continued)

- assume $[e_3] \prec [e_5]$, which makes e_5 a cut-off event
- this entails $[e_3, e_4] \prec [e_5, e_6]$, by isomorphic extension
- $[e_6] \prec [e_4]$, which would make e_4 a cut-off event, is false, as this would entail $[e_6, e_5] \prec [e_4, e_3]$ by isomorphic extension, which is false





Thm the prefix $\mathcal{O} \sqsubseteq \mathcal{U}_{\mathcal{N}}$ obtained by stopping the unfolding algorithm at cut-off events is finite and complete [McMillan, Esparza].

<u>Proof</u> : see references at the end of the lesson ;
finiteness and completeness are proved separately,
and heavily rely on properties of adequate orders.

Thm the prefix $\mathcal{O} \sqsubseteq \mathcal{U}_{\mathcal{N}}$ obtained by stopping the unfolding algorithm at cut-off events is finite and complete [McMillan, Esparza].

Proof : see references at the end of the lesson ;
finiteness and completeness are proved separately,
and heavily rely on properties of adequate orders.

Thm if the adequate order $\prec$ used to buid the FCP $\mathcal{O} \sqsubseteq \mathcal{U}_{\mathcal{N}}$ is a total order, then the number of non-cut-off events in $\mathcal{O}$ is bounded by the number of reachable markings in $\mathcal{N}$.

Proof : for two events $e, e' \in \mathcal{O}$ such that $\mathrm{Mark}([e]) = \mathrm{Mark}([e'])$
either $[e] \prec [e']$ or $[e'] \prec [e]$ holds,
so one of these events is a cut-off

Application to deadlock checking

Deadlock : a marking of $\mathcal{N}$ where no more transition can be fired

Thm Let $\mathcal{O} \sqsubseteq \mathcal{U}_{\mathcal{N}}$ be a finite complete prefix.
There is no deadlock in $\mathcal{N}$ iff every configuraion $\kappa \sqsubseteq \mathcal{O}$ can be extended into a configuration $\kappa \sqsubseteq \kappa' \sqsubseteq \mathcal{O}$ that contains a cut-off event. [McMillan]

Proof : (sketch of)

- a maximal configuration with no cut-off can't be extended :
  the terminal marking is a dead-end
- conversely, at a cut-off, one reaches a marking that is present and extended elsewhere in the prefix, which means that a continuation is possible

Take home messages

(Safe) Petri nets

- are a natural model for concurrent systems
- can be built by product, as networks of automata
- admit natural (built in) true concurrency semantics for their runs
- sets of runs can be handled by branching processes (unfoldings) instead of languages

Extra results

- by restricting branching processes to events, one gets (prime) event structures $\mathcal{E} = (E, \rightarrow, \#)$ a complete theory of event structures exists
- one can define a product on unfoldings, and

$$\mathcal{U}(\mathcal{N}_1 \times ... \times \mathcal{N}_K) \;=\; \mathcal{U}(\mathcal{N}_1) \times ... \times \mathcal{U}(\mathcal{N}_K)$$

projections exists as well, which enables distributed computations based branching processes, or on other structures (e.g. event structures).

References

About prefix construction

1. An unfolding algorithm for synchronous products of transition systems, Esparza and Romer, proceedings of CONCUR'99, pp 2-20
2. An improvement of McMillan's unfolding algorithm, Esparza and Romer, LNCS 1055, pp 87-106, 1996
3. Canonical prefixes of Petri net unfoldings,Khomenko, Koutny and Vogler, Acta Informatica 40, pp 95-118, 2003

More oriented to model-checking applications

1. Model checking using net unfoldings, Esparza, Science of Computer Programming 23, pp 151-195, 1994
2. Reachability analysis unsing net unfoldings,Schroter and Esparza
3. Deadlock checking using net unfoldings, Melzer and Romer, LNCS 1254, pp 352-363, 1997
4. Using net unfoldings to avoid the state explosion problem in the verification of asynchronous circuits, McMillan, LNCS 663, pp 164-174, 1992