Partial Order Techniques for Distributed Discrete Event Systems

Wodes 2006

Eric Fabre and Albert Benveniste

IRISA-INRIA Rennes

An industrial experience with Alcatel

Fault management and alarm correlation in telecom nets:

- Edge equipement of long haul submarine optical line
- Radio access network (GSM)
- Optical networks SONET/SDH/WDM
- ALcatel MAnagement Plateform (ALMAP)

A telecom network system is made of a number of hardware and software components. Each component possesses its own monitoring system that detects anomalies and propagates deny of service information to the neighbours, through alarm messages. This causes thousands of causally related ("correlated") messages to travel across the network and reach the supervision system.

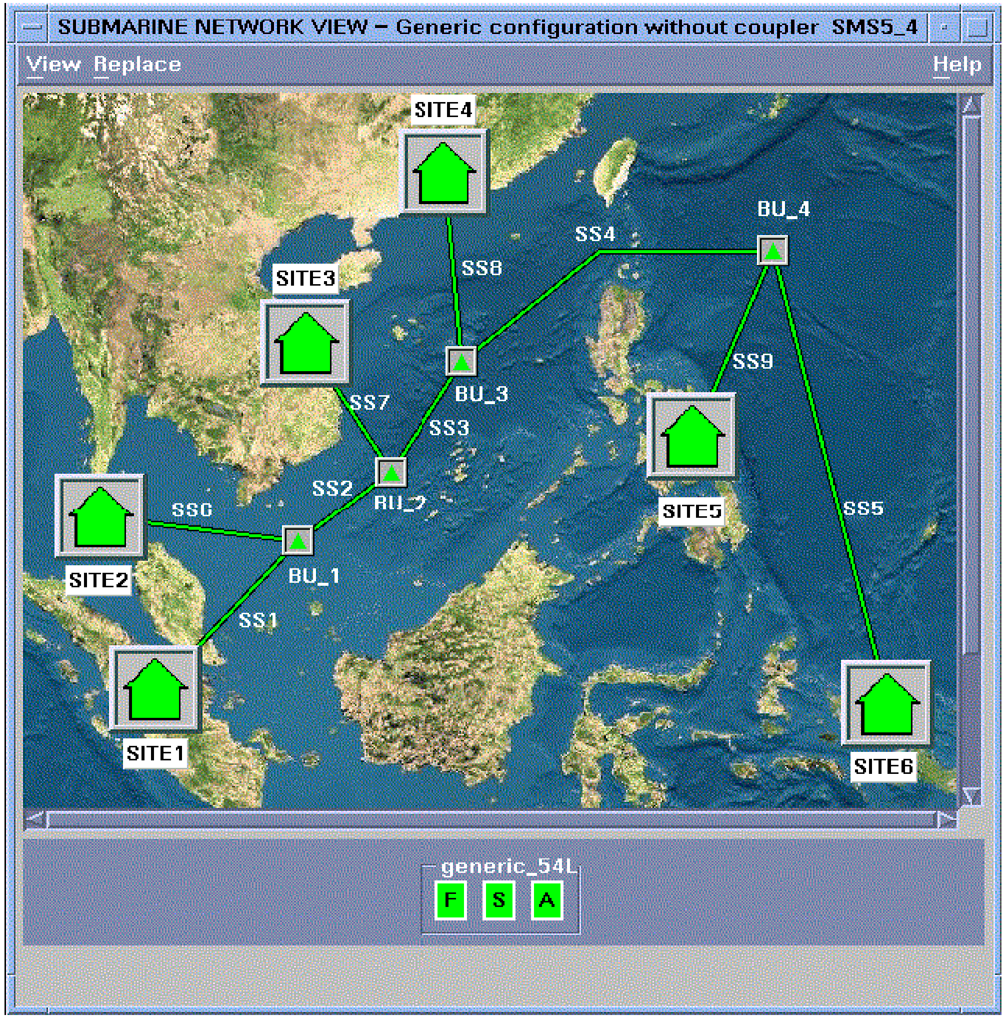


Figure A.1: the submarine optical telecommunication system considered for the trial with Alcatel Optical Systems business division and Alcatel Research and Innovation.

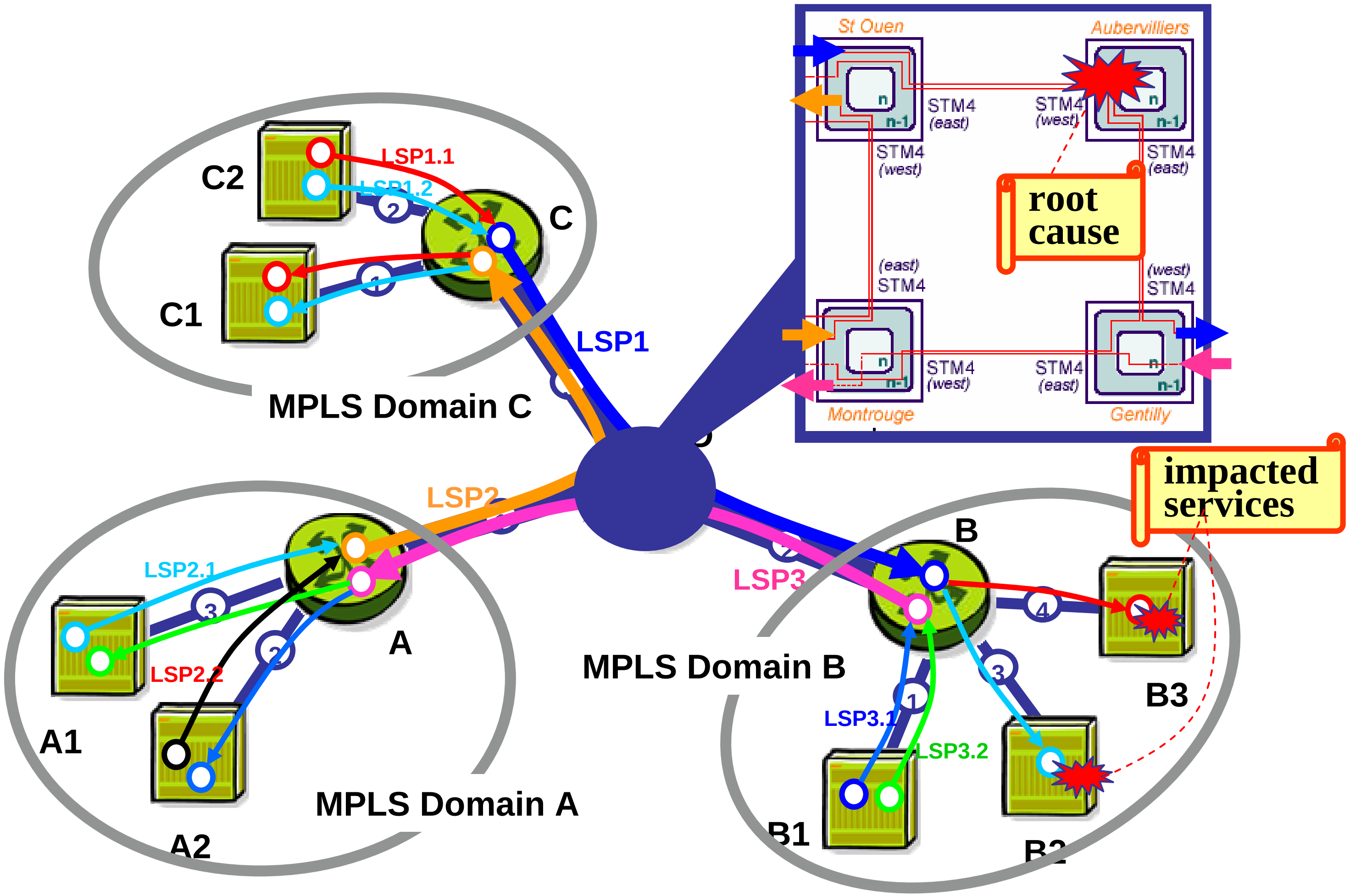


Figure A.2: failure impact analysis.

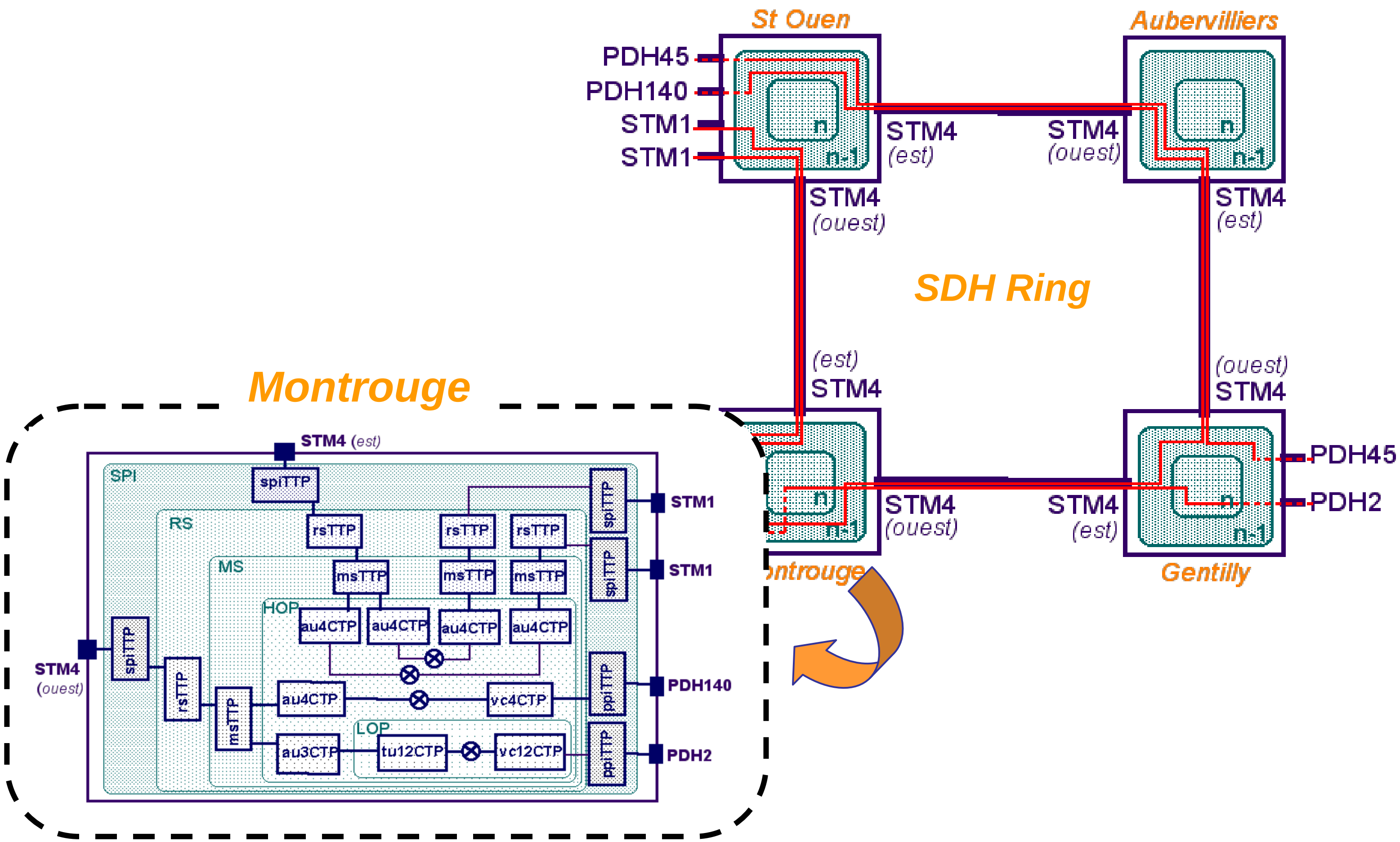


Figure A.3: the SDH/SONET optical ring of the Paris area, with its four nodes. The diagram on the left zooms on the structure of the management software, and shows its Managed Objects

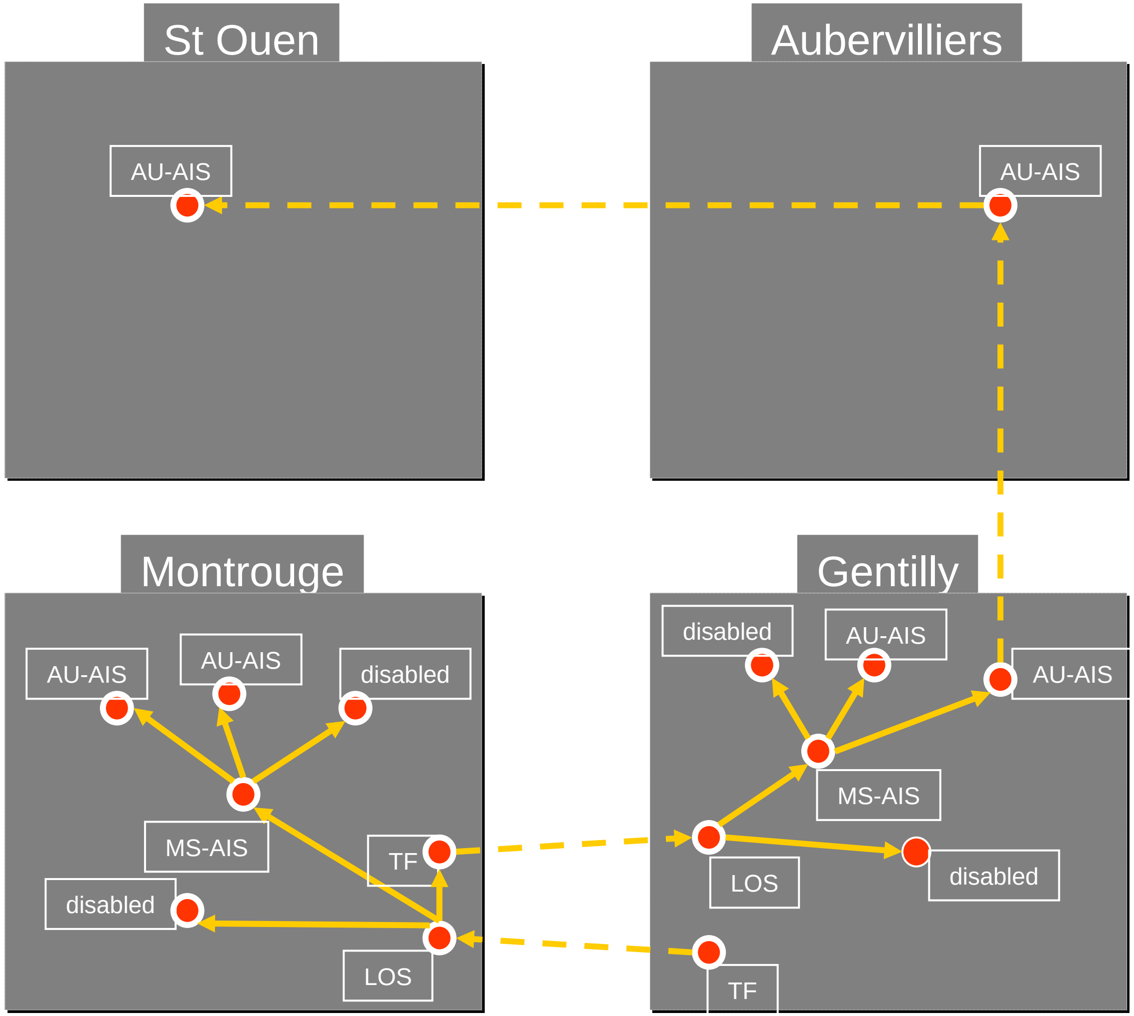


Figure A.4: showing a failure propagation scenario, across management layers (vertically) and network nodes (horizontally).

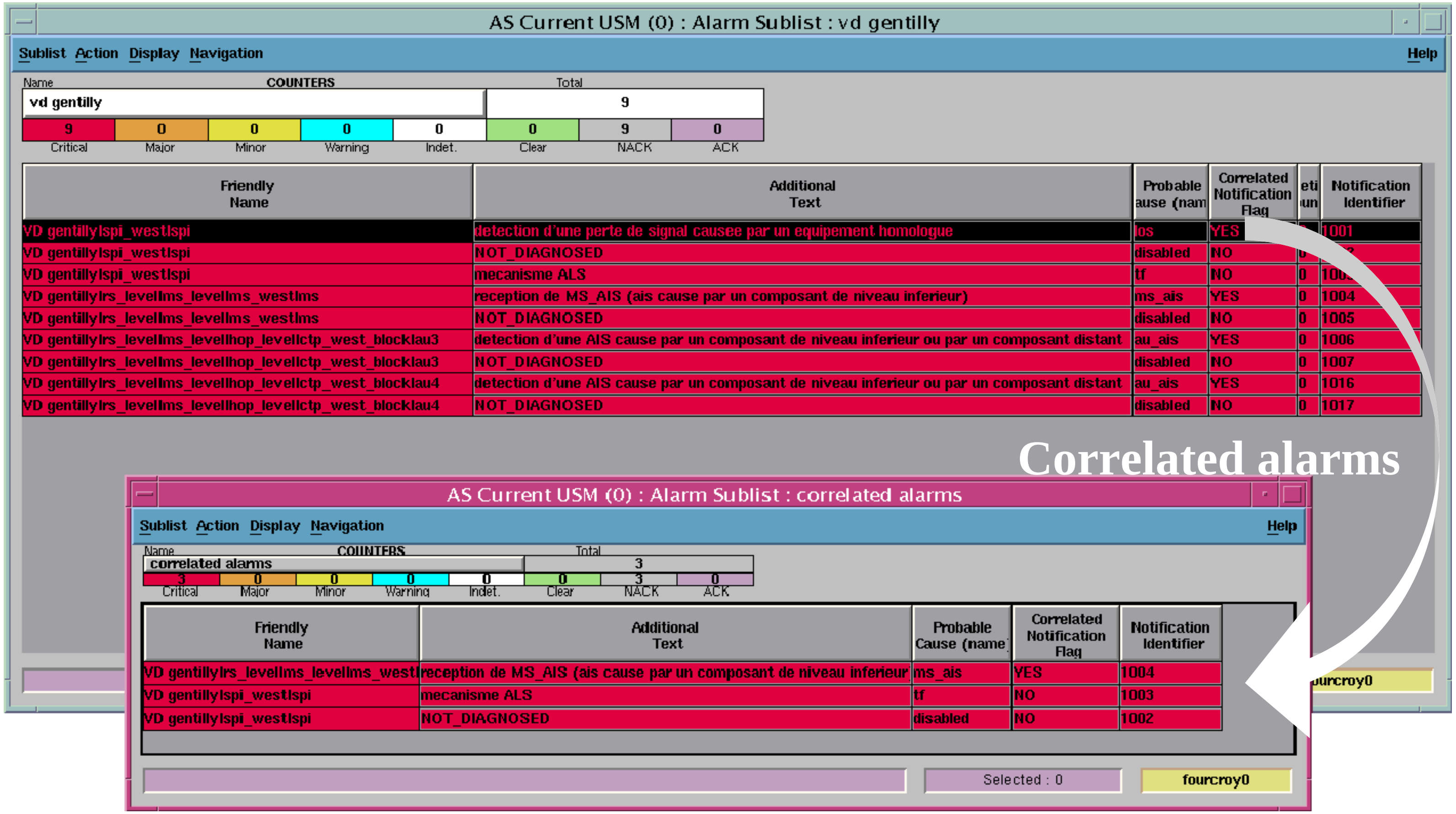
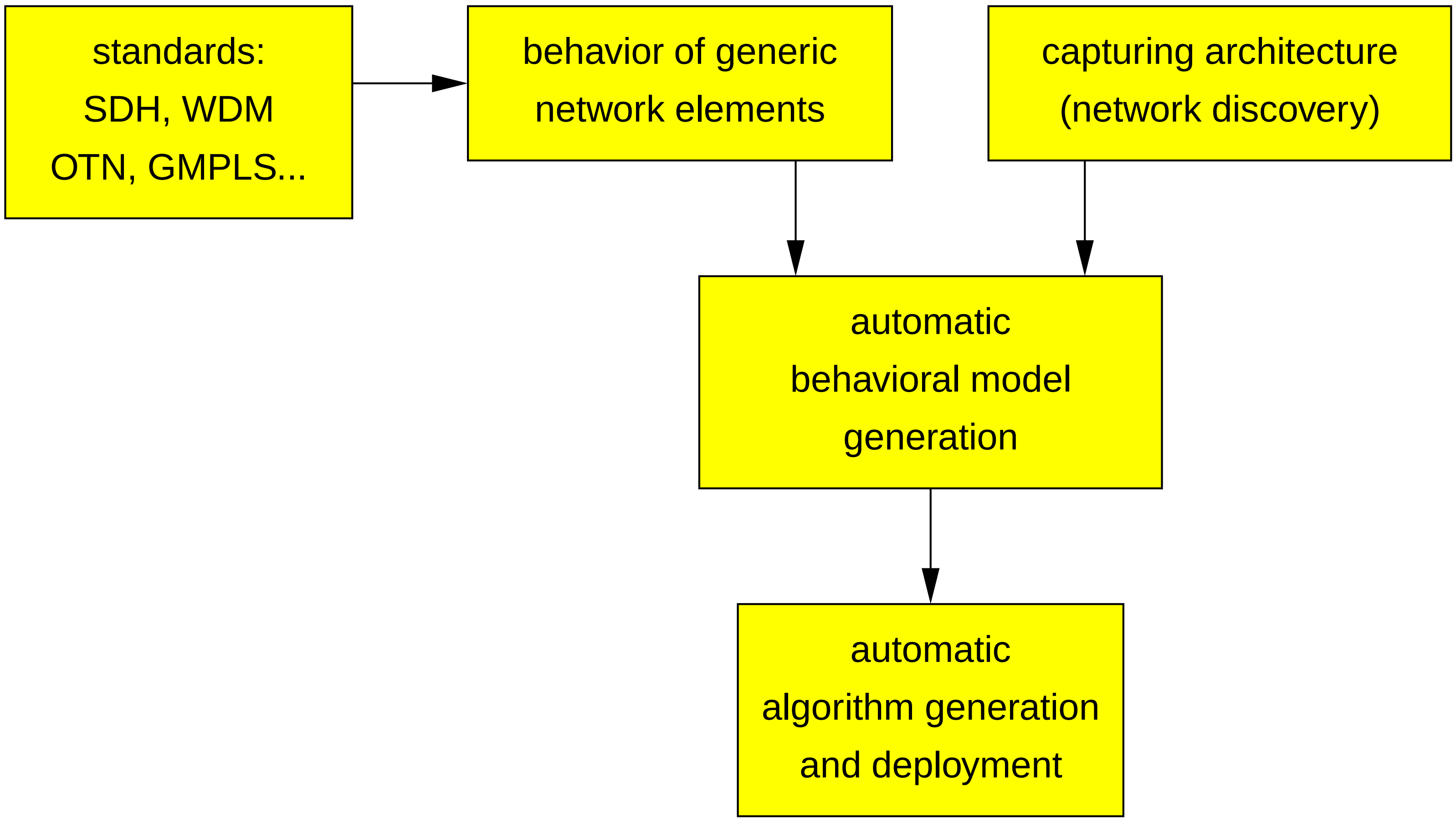


Figure A.5: returning alarm correlation information to the operator.

The need for self-modeling



Model based techniques require models! Models for such huge systems can't be built by hand.

Contents

1. Industrial motivation
2. Problem setting: (on-line) distributed monitoring of distributed systems
3. Using classical tools: automata and products
4. Problem of state explosion: more compact data structures:
   (a) execution trees
   (b) trellises
   (c) partial orders
5. Other issues

Distributed systems monitoring

We are given:

- A distributed system $\mathcal{A}$ with subsystems $\mathcal{A}_i, i \in I$;
- $\mathcal{O}_i, i \in I$, observation system attached to each subsystem;

Distributed systems monitoring

We are given:

- A distributed system $\mathcal{A}$ with subsystems $\mathcal{A}_i, i \in I$;
- $\mathcal{O}_i, i \in I$, observation system attached to each subsystem;

Perform the monitoring of $\mathcal{A}$ under the following constraints:

- A supervisor $\mathcal{S}_i$ is attached to each subsystem;
- $\mathcal{S}_i$ only knows the local system model $\mathcal{A}_i$ plus interface information relating $\mathcal{A}_i$ to its neighbours;
- $\mathcal{S}_i$ accesses observations made by $\mathcal{O}_i$; it can exchange messages with its neighbouring supervisors;
- No global clock is available and communications are asynchronous.

Approach for this talk

We shall first try to address this problem with most classical frameworks: automata and their products

We shall push this game to its very limits

However, at some point, the stringent need for moving to a partial order framework will appear

Here are my lawyers

be prepared to overnight

Monitoring a finite state machine

$$\mathcal{A} \;=\; (S, L, \rightarrow, s_0),\;\; S : \text{states},\;\; L : \text{labels}$$

$$L \;=\; L_o \cup L_u \;=\; \text{observed} \;\cup\; \text{unobserved}$$

$$\sigma \;:\; s_0 \xrightarrow{\ell_1} s_1 \xrightarrow{\ell_2} s_2 \xrightarrow{\ell_3} s_3 \ldots \;\text{a run}$$

$\Sigma_{\mathcal{A}}$: set of all runs of $\mathcal{A}$

$\mathbf{Proj}_o(\sigma)$: erasing states and unobs labels from σ

observation : $O \in \{\mathbf{Proj}_o(\sigma) \mid \sigma \in \Sigma_{\mathcal{A}}\}$

Monitoring a finite state machine

$$\mathcal{A} = (S, L, \rightarrow, s_0),\ S: \text{states},\ L: \text{labels}$$

$$L = L_o \cup L_u = \text{observed} \cup \text{unobserved}$$

$$\sigma : s_0 \xrightarrow{\ell_1} s_1 \xrightarrow{\ell_2} s_2 \xrightarrow{\ell_3} s_3 \ldots \text{ a run}$$

$\Sigma_{\mathcal{A}}$: set of all runs of $\mathcal{A}$

$\mathbf{Proj}_o(\sigma)$: erasing states and unobs labels from σ

observation : $O \in \{\mathbf{Proj}_o(\sigma) \mid \sigma \in \Sigma_{\mathcal{A}}\}$

monitor : algorithm that computes, for every observation O of $\mathcal{A}$, the set $\mathbf{Proj}_o^{-1}(O)$

On-line monitoring

This amounts to synchronizing on-line,

- observation O
- with a "simulation" of $\mathcal{A}$.

Since product captures synchronization, this amounts to constructing, on-line, the set of all runs of the product $\mathcal{A} \times O$.

We need to achieve this in our distributed setting:

$$\begin{aligned} \mathcal{A} &= \times_{i \in I} \mathcal{A}_i \\ O &= \|_{i \in I} O_i \end{aligned}$$

Su & Wonham approach [2004, 2006]

1. compute and store the local monitor $\mathcal{V}_i =_{\text{def}} \mathbf{Proj}_{o,i}^{-1}(O_i)$, seen as a language;
2. perform a *consistent* merge of local monitors:

$$\mathcal{V} =_{\text{def}} \|_{i \in I} \mathcal{V}_i = \mathbf{Proj}_o^{-1}(\|_{i \in I} O_i)$$

and compute $\mathbf{Proj}_i(\mathcal{V})$ without computing $\mathcal{V}$, by allowing exchanges of messages btw supervisors.

Su & Wonham approach [2004, 2006]

1. compute and store the local monitor $\mathcal{V}_i =_{\text{def}} \mathbf{Proj}_{o,i}^{-1}(O_i)$, seen as a language;
2. perform a *consistent* merge of local monitors:

$$\mathcal{V} =_{\text{def}} \|_{i \in I} \mathcal{V}_i = \mathbf{Proj}_o^{-1}(\|_{i \in I} O_i)$$

   and compute $\mathbf{Proj}_i(\mathcal{V})$ without computing $\mathcal{V}$, by allowing exchanges of messages btw supervisors.

Let $\widehat{\mathcal{V}}_i$ be the solutions found by the distributed algorithm. The authors distinguish *local consistency*: local solutions $\widehat{\mathcal{V}}_i$ agree on their interfaces, and *global consistency*: the algorithm succeeds in computing $\widehat{\mathcal{V}}_i = \mathbf{Proj}_i(\mathcal{V})$ for each i.

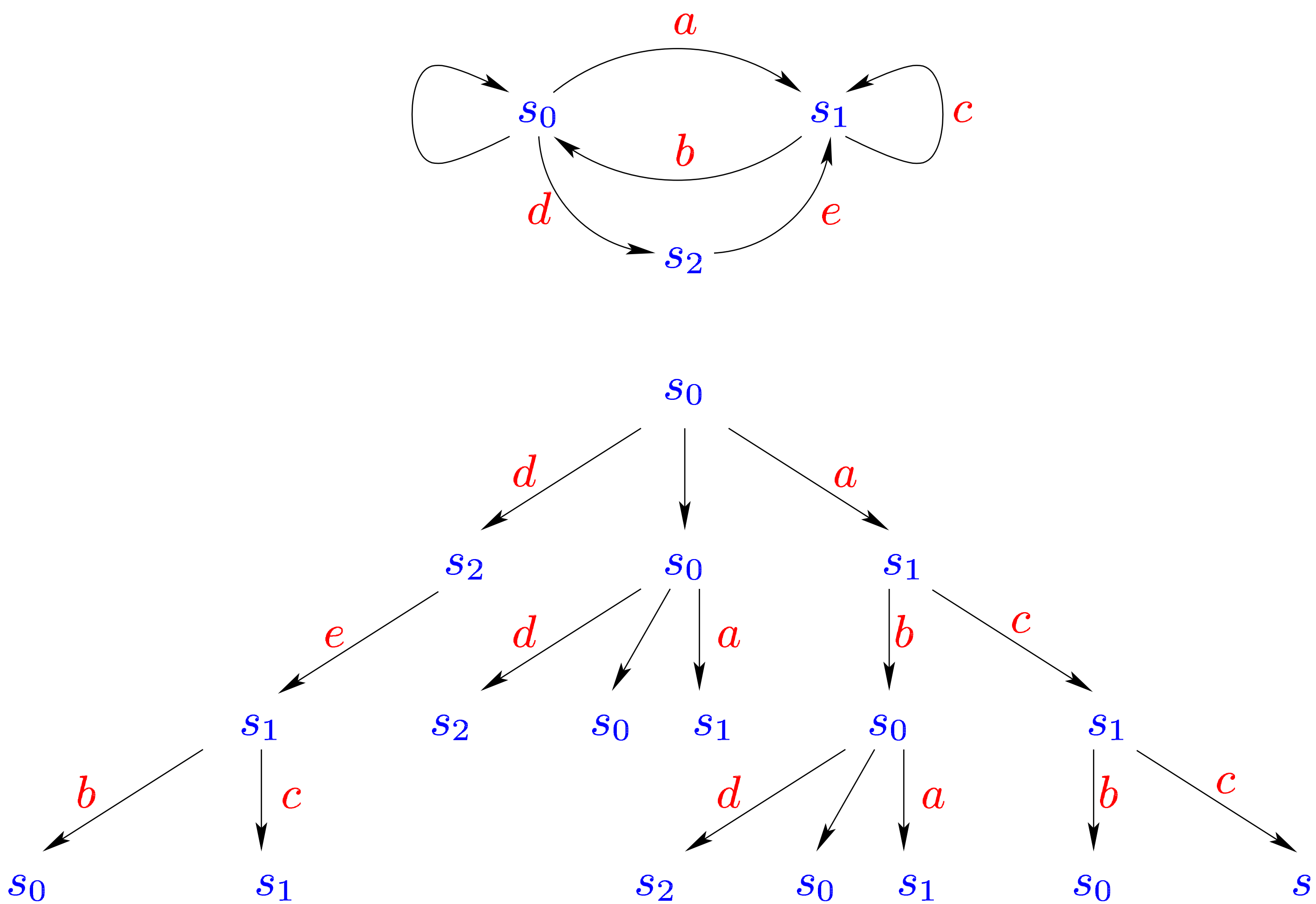
On-line monitoring (cont’d)

- Manipulating languages in the form of sets of runs is costly.
- Representing them by automata is not suitable for on-line processing.
- *Greatest attention must be paid to data structures and how to compute with them.*

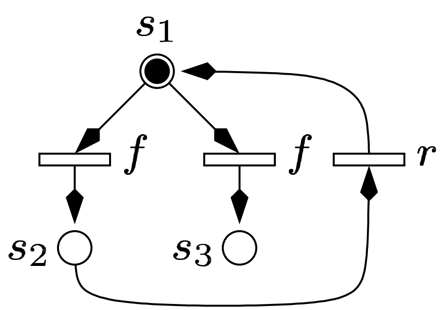
We need *efficient* data structures to construct the set of all runs of an automaton, incrementally, in a distributed way:

- automata unfoldings/trellises,
- partial order unfoldings/trellises.

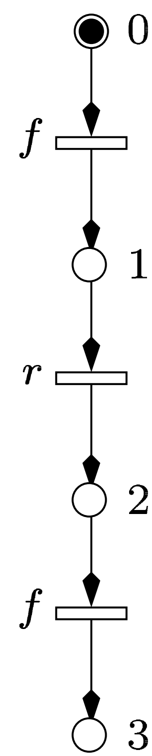
Automata unfoldings (execution trees)



Unfolding based monitoring



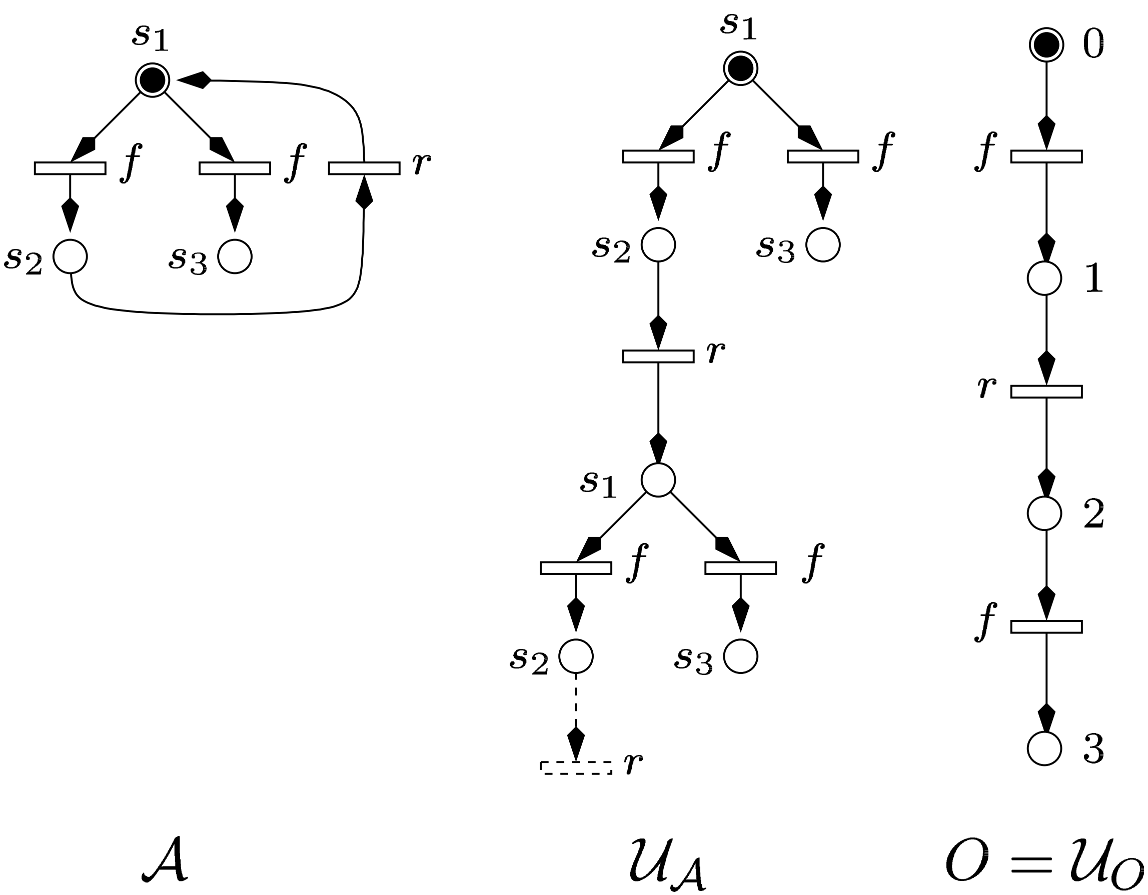
$\mathcal{A}$



O

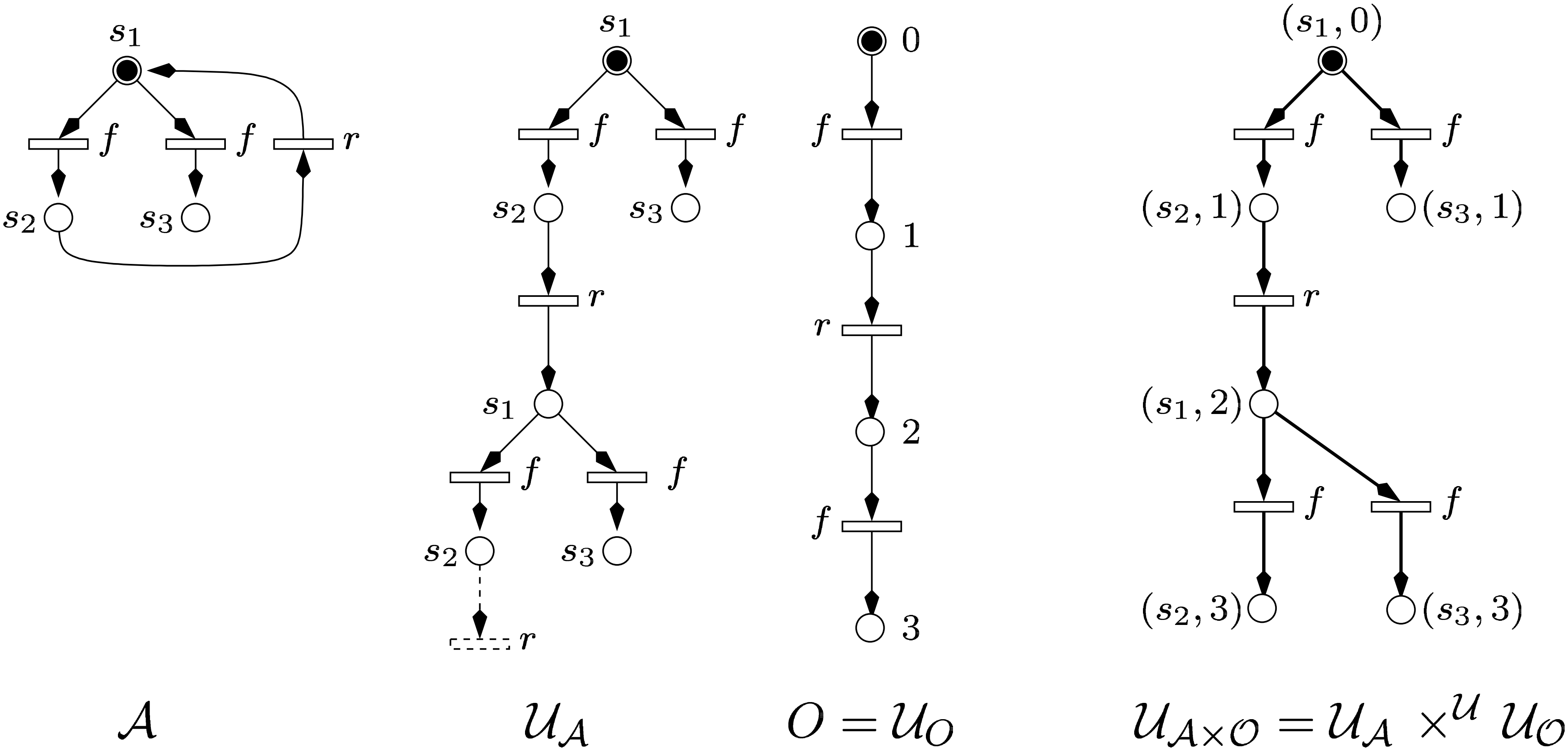
{automaton, observation}

Unfolding based monitoring



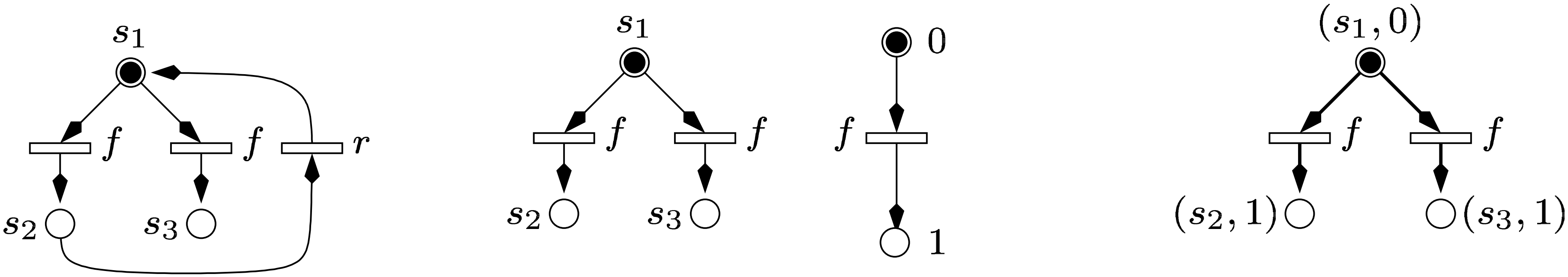
synchronizing their unfoldings

Unfolding based monitoring



the resulting product

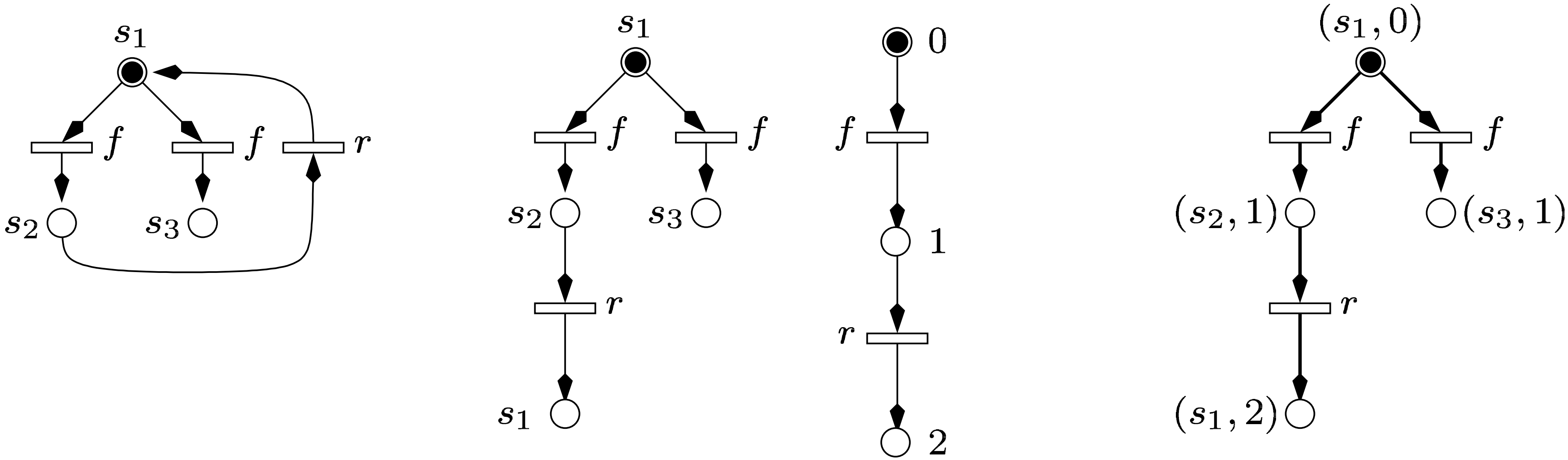
Unfolding based monitoring



$\mathcal{A}$ $\qquad \mathcal{U}_{\mathcal{A}}$ $\qquad O = \mathcal{U}_O$ $\qquad \mathcal{U}_{\mathcal{A}\times\mathcal{O}} = \mathcal{U}_{\mathcal{A}} \times^{\mathcal{U}} \mathcal{U}_{\mathcal{O}}$

on-line computation of monitoring

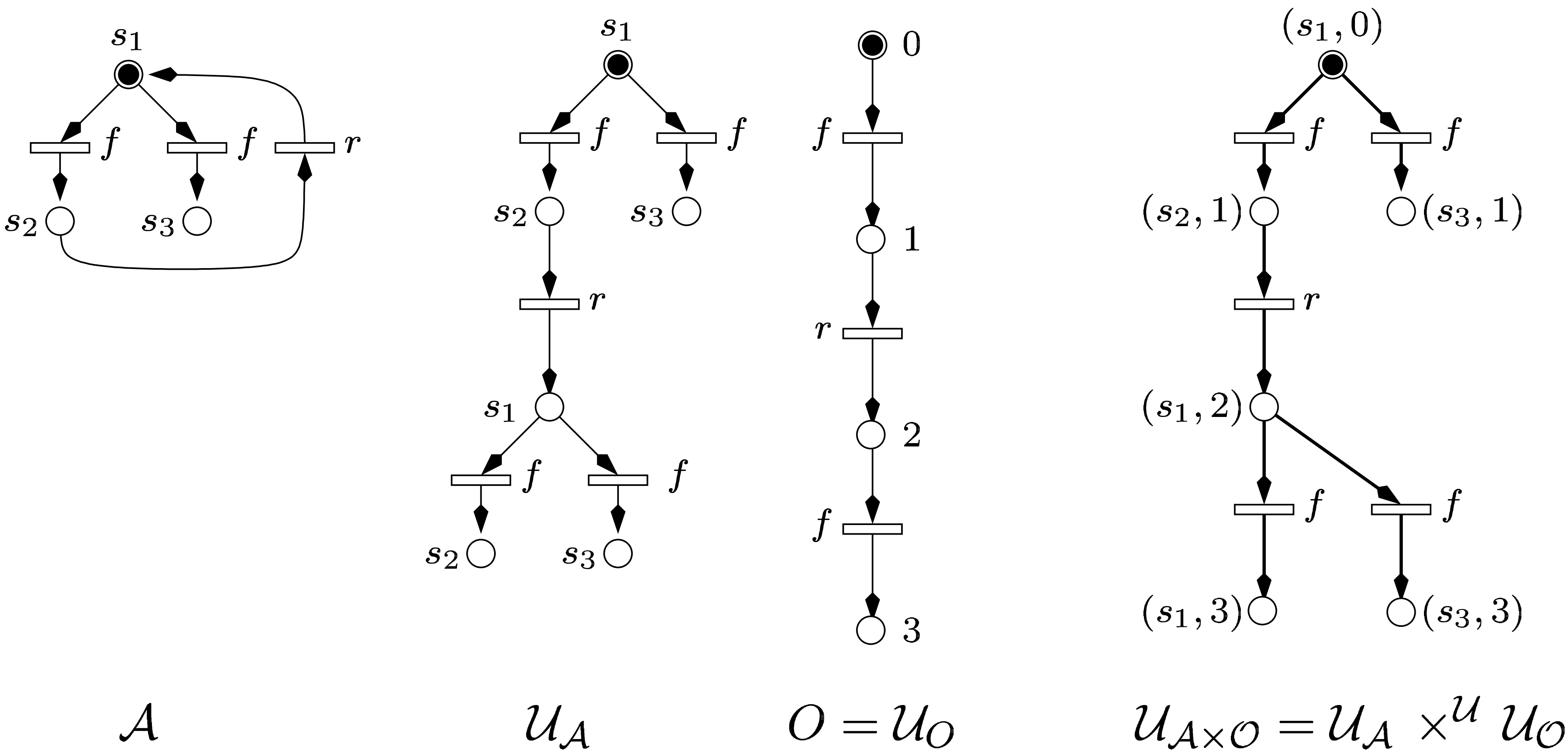
Unfolding based monitoring



$\mathcal{A}$ $\qquad$ $\mathcal{U}_{\mathcal{A}}$ $\qquad$ $\mathcal{O} = \mathcal{U}_{\mathcal{O}}$ $\qquad$ $\mathcal{U}_{\mathcal{A}\times\mathcal{O}} = \mathcal{U}_{\mathcal{A}} \times^{\mathcal{U}} \mathcal{U}_{\mathcal{O}}$

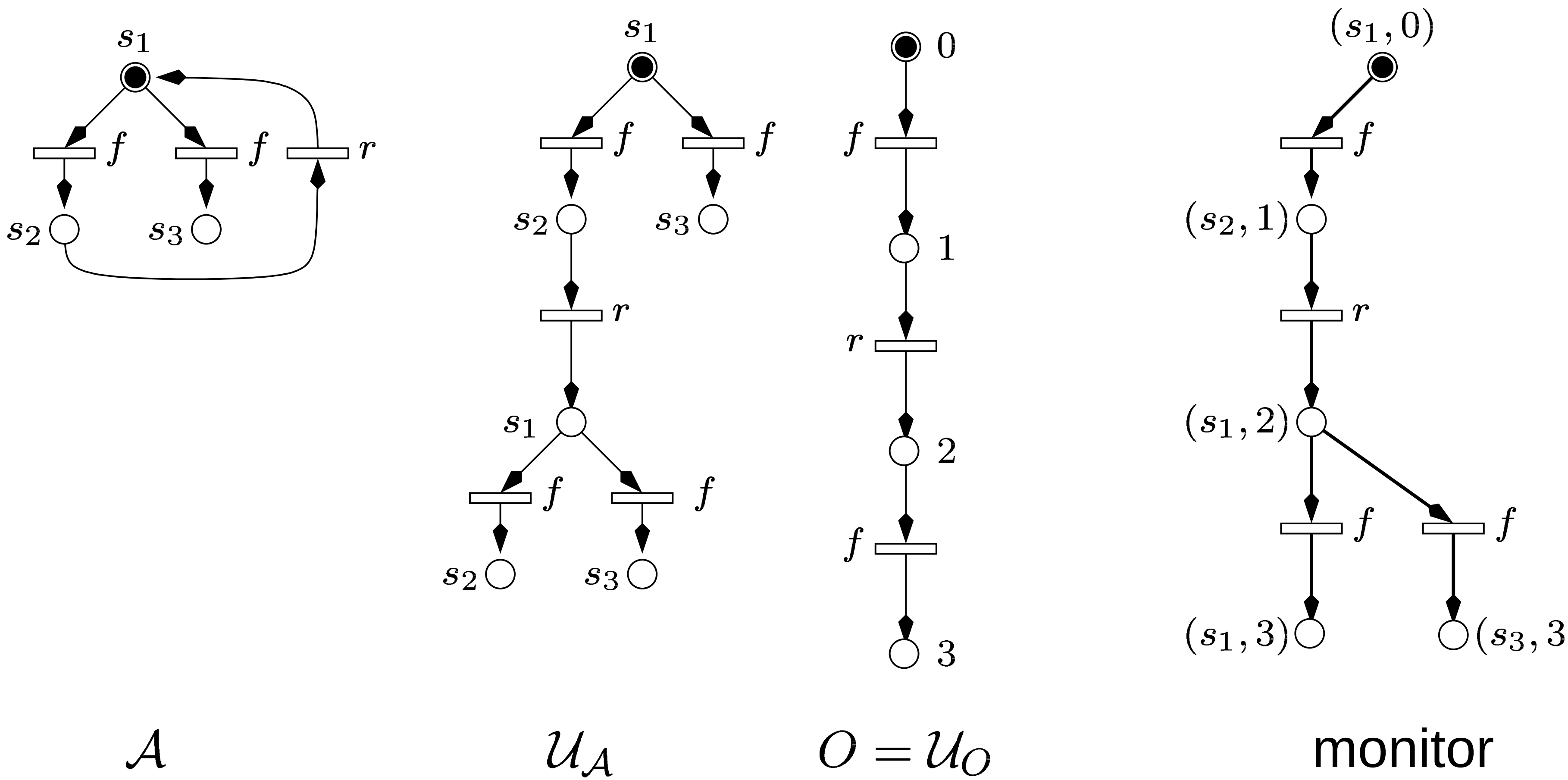
on-line computation of monitoring

Unfolding based monitoring



on-line computation of monitoring

Unfolding based monitoring



pruning blocked trajectories (with delay 1)

Basic tools to handle distributed systems

Assume $\mathcal{A} = \bigtimes_{i \in I} \mathcal{A}_i, \ O = \bigtimes_{i \in I} O_i$

Problem: computing the monitor $\mathcal{U}_{\mathcal{A} \times O}$ suffers from state explosion in $\mathcal{A}$, and thus in its unfolding.

A first idea to avoid this is to apply, for unfoldings, the well known recommendation: never compute the product.

Since we have $\mathcal{A} \times O = \bigtimes_{i \in I} (\mathcal{A}_i \times O_i)$, this amounts to

computing $\mathcal{U}_{\times_{i \in I}(\mathcal{A}_i \times O_i)}$
without computing $\bigtimes_{i \in I} (\mathcal{A}_i \times O_i)$.

Basic tools to handle distributed systems

- The product of unfoldings is formally defined as follows:

$$\mathcal{V} \times^{\mathcal{U}} \mathcal{V}' \quad =_{\text{def}} \quad \mathcal{U}_{\mathcal{V} \times \mathcal{V}'}$$

- For $L' \subseteq L$ and $\pi : S \mapsto S'$, the projection

$$\mathbf{Proj}_{L',\pi}\left(\mathcal{U}_{\mathcal{A}}\right)$$

  is obtained by deleting transitions $\notin L'$, taking transitive closure, determinizing, and mapping s to $\pi(s)$.

- The intersection of sub-unfoldings of a same $\mathcal{U}_{\mathcal{A}}$:

$$\mathcal{V} \cap \mathcal{V}'$$

  possesses as runs the common runs of $\mathcal{V}$ and $\mathcal{V}'$.

Basic tools to handle distributed systems

Theorem **[Fabre & al 2003]** **(factorizing unfoldings)**

$$\mathcal{A} = \times_{j\in I}\, \mathcal{A}_j \Longrightarrow \mathcal{U}_{\mathcal{A}} \;=\; \times^{\mathcal{U}}_{i\in I}\; \mathcal{U}_{\mathcal{A}_i}$$

$$= \;\times^{\mathcal{U}}_{i\in I}\; \mathbf{Proj}_i\,(\mathcal{U}_{\mathcal{A}})$$

Definition (modular monitoring)

$$\mathcal{A}\times O \quad = \quad \times_{j\in I}\,(\mathcal{A}_i \times O_i)$$

$$\mathcal{M} \quad =_{\mathrm{def}} \quad \mathcal{U}_{\mathcal{A}\times O}$$

$$\mathcal{M}_{\mathrm{mod}} \;=_{\mathrm{def}}\; (\mathcal{M}_i)_{i\in I}\,,\, \mathcal{M}_i = \mathbf{Proj}_i\,(\mathcal{M})$$

Key Problems

Problem 1 *Compute* $\mathcal{M}_{\mathrm{mod}}$ *without computing* $\mathcal{M}$*.*

Problem 2 *Compute* $\mathcal{M}_{\mathrm{mod}}$ *by attaching a supervising peer to each site.*

Problem 3 *Compute* $\mathcal{M}_{\mathrm{mod}}$ *on-line and on the fly.*

Problem 4 *Address asynchronous distributed systems.*

Problem 5 *Avoid state explosion due to concurrency.*

Problem 6 *Address changes in the systems dynamics.*

A separation theorem

Theorem: $\mathcal{A}_i, i = 1, 2, 3$: automata.
Say that $\mathcal{A}_2$ separates $\mathcal{A}_1$ from $\mathcal{A}_3$ if $(L_1 \cap L_3) \subseteq L_2$. Then:

$$\mathbf{Proj}_2\left(\mathcal{U}_{\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3}\right) = \underbrace{\underbrace{\mathbf{Proj}_2\left(\mathcal{U}_{\mathcal{A}_1 \times \mathcal{A}_2}\right)}_{\text{local to (1,2)}} \bigcap \underbrace{\mathbf{Proj}_2\left(\mathcal{U}_{\mathcal{A}_2 \times \mathcal{A}_3}\right)}_{\text{local to (2,3)}}}_{\text{local to 2 (intersection)}}$$

$$\mathbf{Proj}_1\left(\mathcal{U}_{\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3}\right) = \mathbf{Proj}_1\left(\underbrace{\mathcal{U}_{\mathcal{A}_1} \times^{\mathcal{U}} \underbrace{\mathbf{Proj}_2\left(\mathcal{U}_{\mathcal{A}_2 \times \mathcal{A}_3}\right)}_{\text{local to (2,3)}}}_{\text{local to (1,2)}}\right)$$

A separation theorem

Define the following operators:

$$\mathbf{Msg}_{\mathcal{V}_i \to \mathcal{V}_j} =_{\text{def}} \mathbf{Proj}_j \left(\mathcal{V}_j \times^{\mathcal{U}} \mathcal{V}_i \right)$$

$$\mathbf{Fuse}\left(\mathcal{V}_i \, , \, \mathcal{V}_i' \right) =_{\text{def}} \mathcal{V}_i \cap \mathcal{V}_i'$$

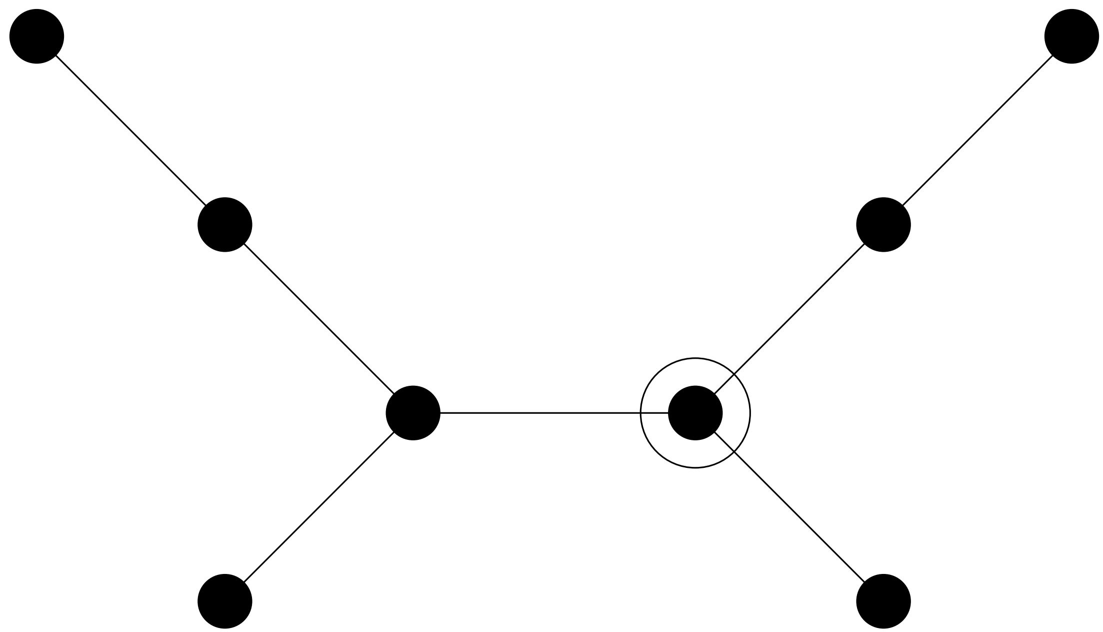
Using these operators, previous rules rewrite as

$$\mathbf{Proj}_2 \left(\mathcal{U}_{\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3} \right) = \mathbf{Fuse}\left(\mathbf{Msg}_{\mathcal{U}_{\mathcal{A}_1} \to \mathcal{U}_{\mathcal{A}_2}} \, , \, \mathbf{Msg}_{\mathcal{U}_{\mathcal{A}_3} \to \mathcal{U}_{\mathcal{A}_2}} \right)$$

$$\mathbf{Proj}_1 \left(\mathcal{U}_{\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3} \right) = \mathbf{Msg}_{\left(\mathbf{Msg}_{\mathcal{U}_{\mathcal{A}_3} \to \mathcal{U}_{\mathcal{A}_2}} \right) \to \mathcal{U}_{\mathcal{A}_1}}$$

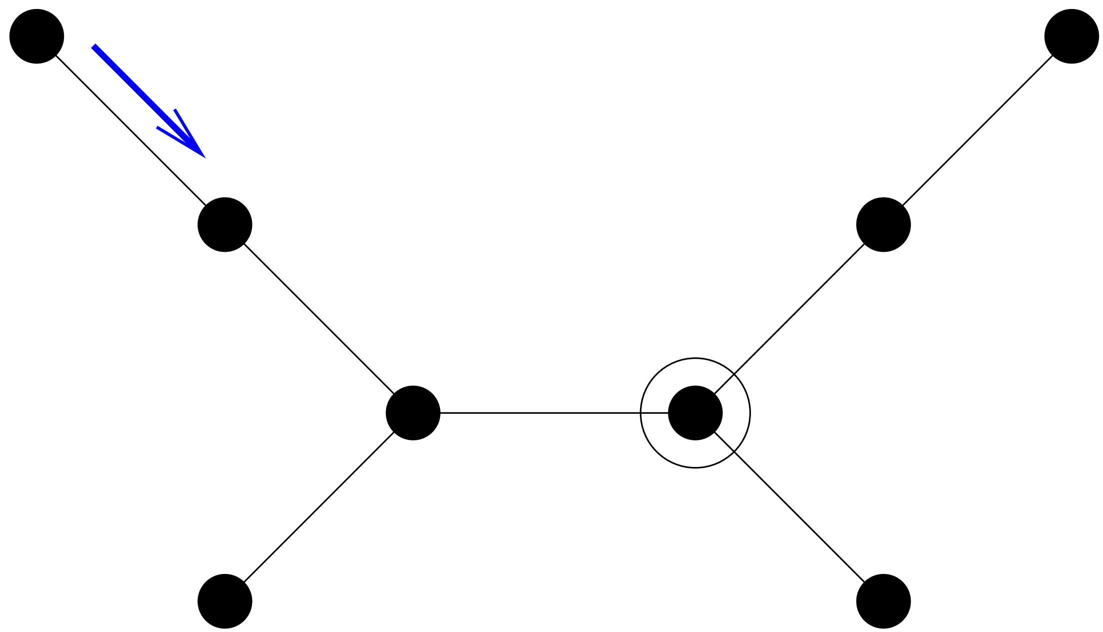
Lemma: The two operators $\mathbf{Msg}$ and $\mathbf{Fuse}$ are increasing w.r.t. their arguments.

Use for belief propagation algorithm



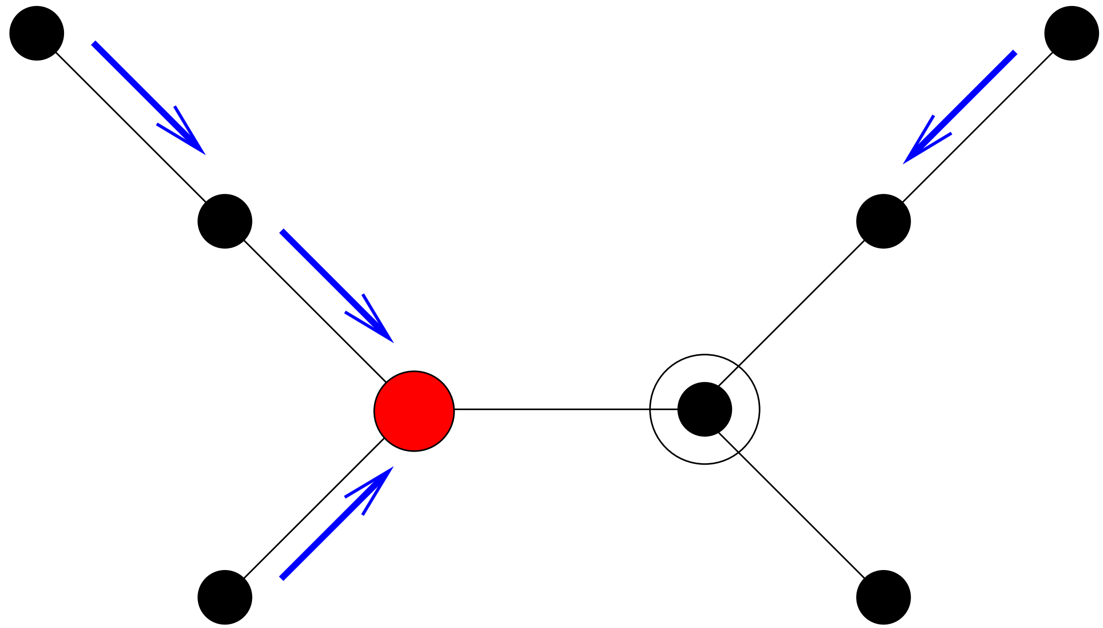
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

Use for belief propagation algorithm



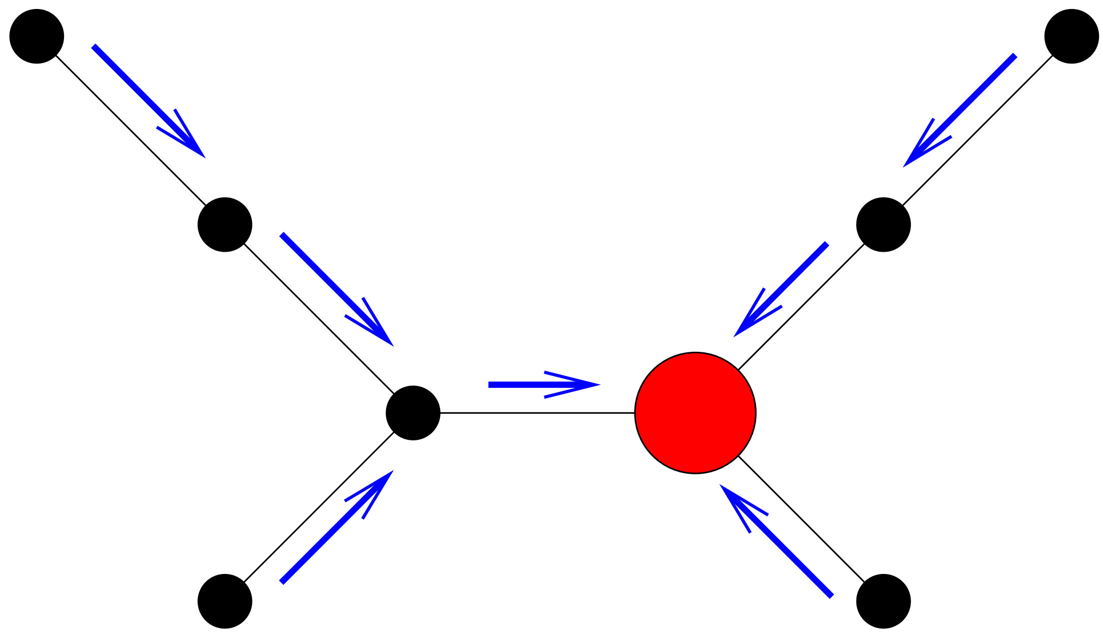
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i\in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

Use for belief propagation algorithm



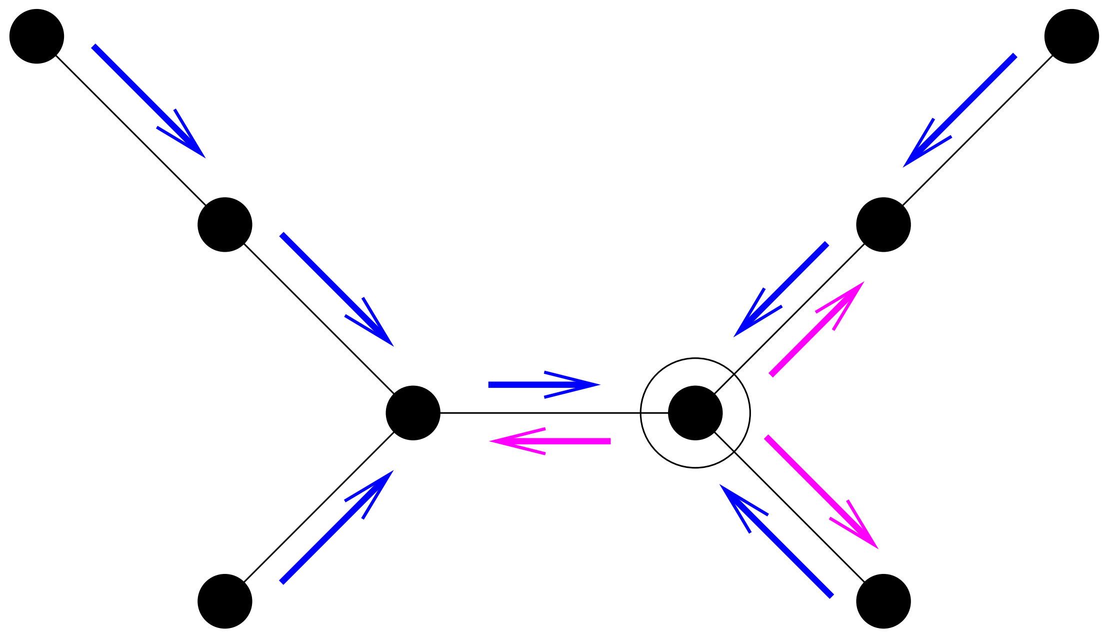
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

Use for belief propagation algorithm



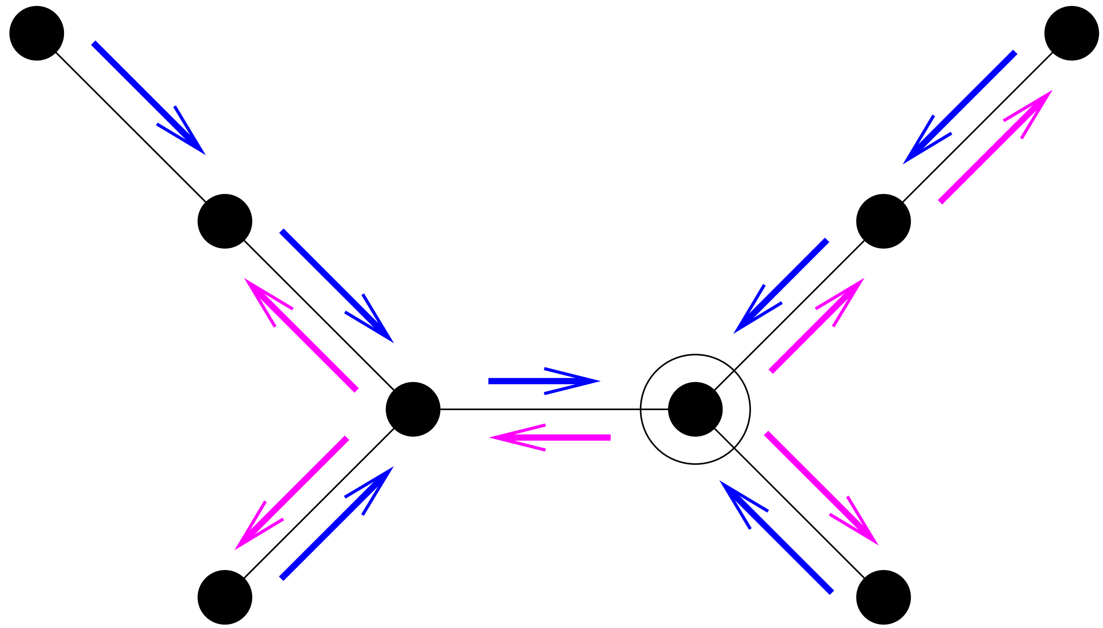
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

Use for belief propagation algorithm



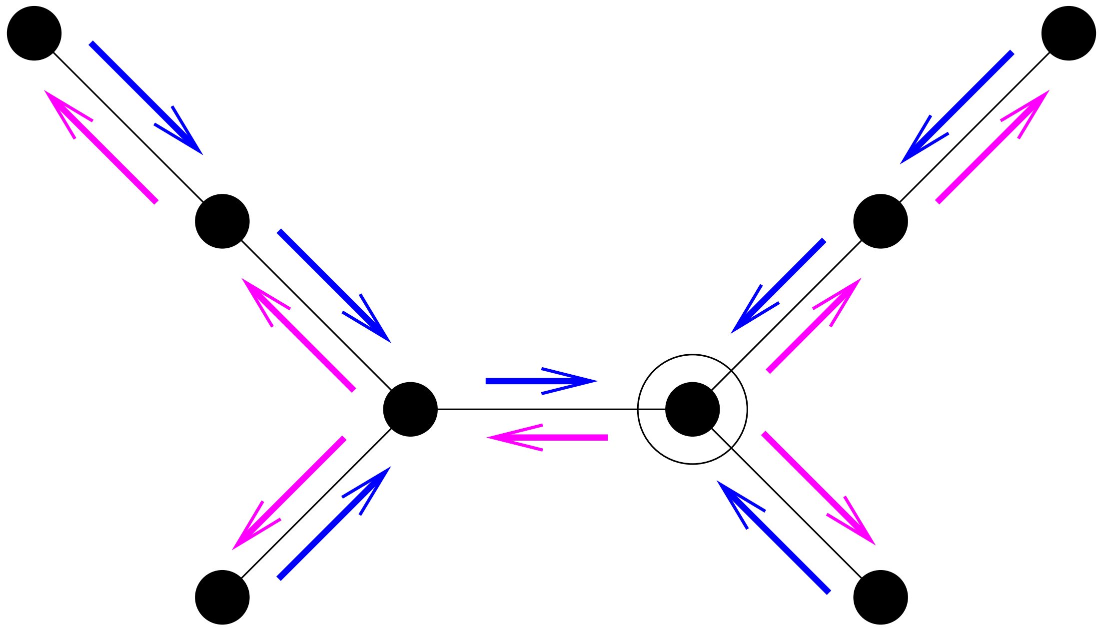
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

Use for belief propagation algorithm



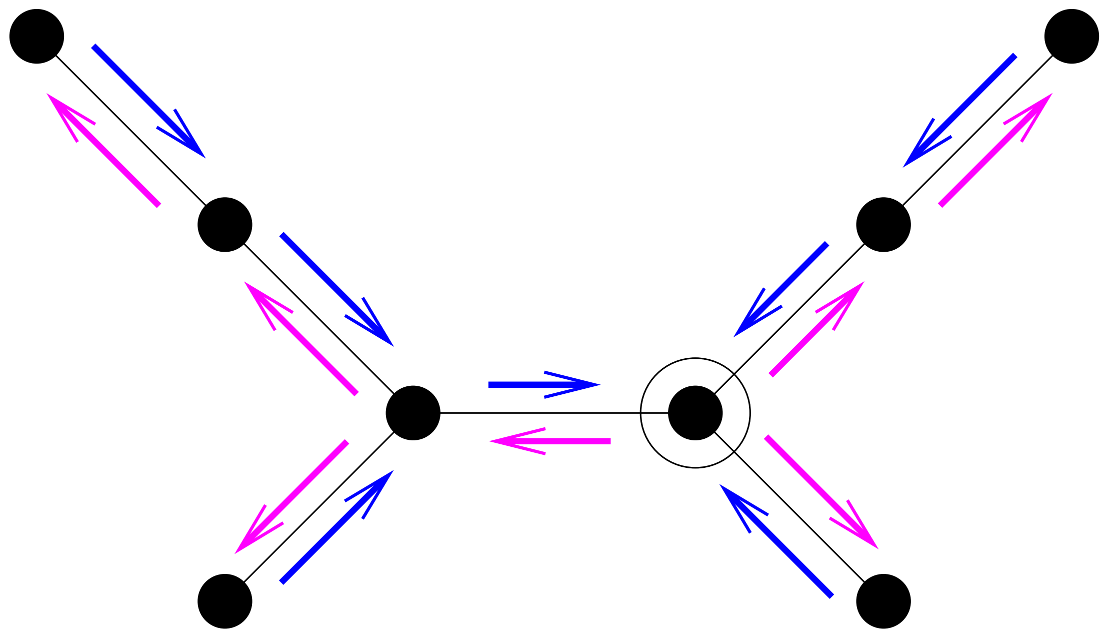
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

Use for belief propagation algorithm



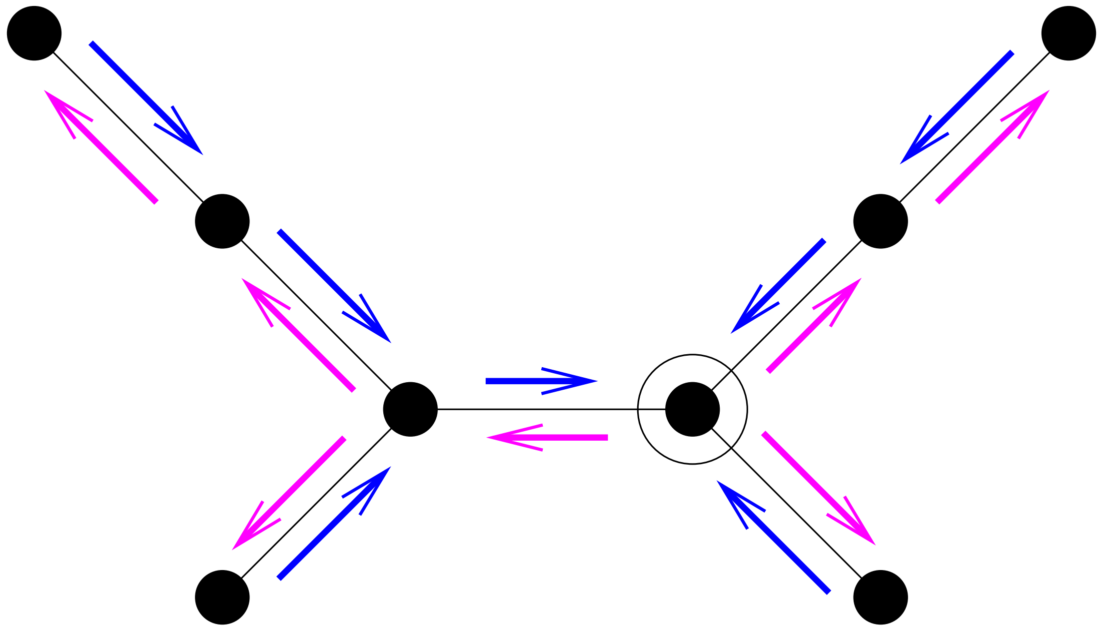
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i\in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.
Computes $\mathcal{M}_{\mathrm{mod}}$ without computing $\mathcal{M}$, by attaching a supervising peer to each site — with a rigid scheduling, however.

Use for belief propagation algorithm



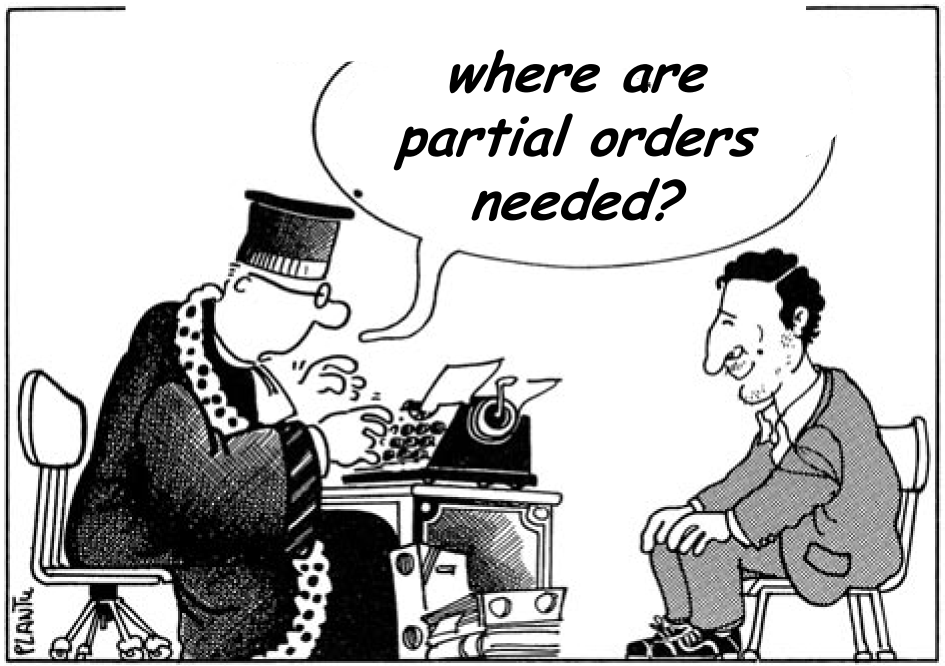
Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.
Since $\mathbf{Msg}$ and $\mathbf{Fuse}$ are increasing w.r.t. their arguments, chaotic asynchronous iterations can be used as well. These can be interleaved with getting new observations. Yields an on-line, distributed, and asynchronous algorithm.

Use for belief propagation algorithm



Belief propagation algorithm when the interaction graph of $(\mathcal{A}_i)_{i \in I}$ is a tree. For distributed monitoring: $\mathcal{A}_i \leftarrow \mathcal{A}_i \times \mathcal{O}_i$.

When cycles exist in the interaction graph, the same distributed chaotic algorithm yields *local consistency* but not global consistency.


where are partial orders needed?
PLANTU

From unfoldings to trellises

- We solved Problems 1, 2, and 3: on-the-fly modular and distributed supervision.
- Did we properly address state explosion? Asynchrony? Concurrency? Not quite so:

From unfoldings to trellises

- We solved Problems 1, 2, and 3: on-the-fly modular and distributed supervision.
- Did we properly address state explosion? Asynchrony? Concurrency? Not quite so:
  - Factorized unfoldings significantly reduces state explosion, but . . .
  - Automata unfoldings are trees that generally grow exponentially in width along with their depth.
  - This fact gets worse as components exhibit internal concurrency — something that follows from asynchrony.

From unfoldings to trellises

- The well known Viterbi algorithm for max likelihood estimation of hidden state in stochastic automata uses another data structure: trellises.

From unfoldings to trellises

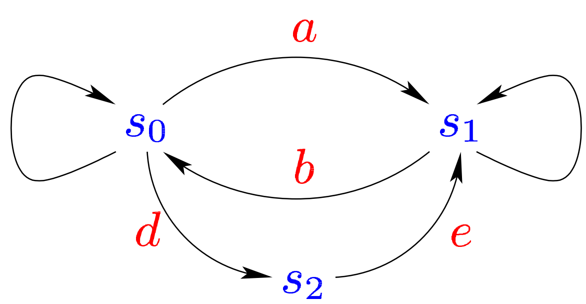
- The well known Viterbi algorithm for max likelihood estimation of hidden state in stochastic automata uses another data structure: trellises.
- Trellises are obtained from unfoldings by merging identical futures of different runs, according to various observation criteria. Examples are:
  - length of the path σ;
  - visible length of the path σ;
  - projection $\mathbf{Proj}_{L'}(\sigma)$, where $L' \subset L$.

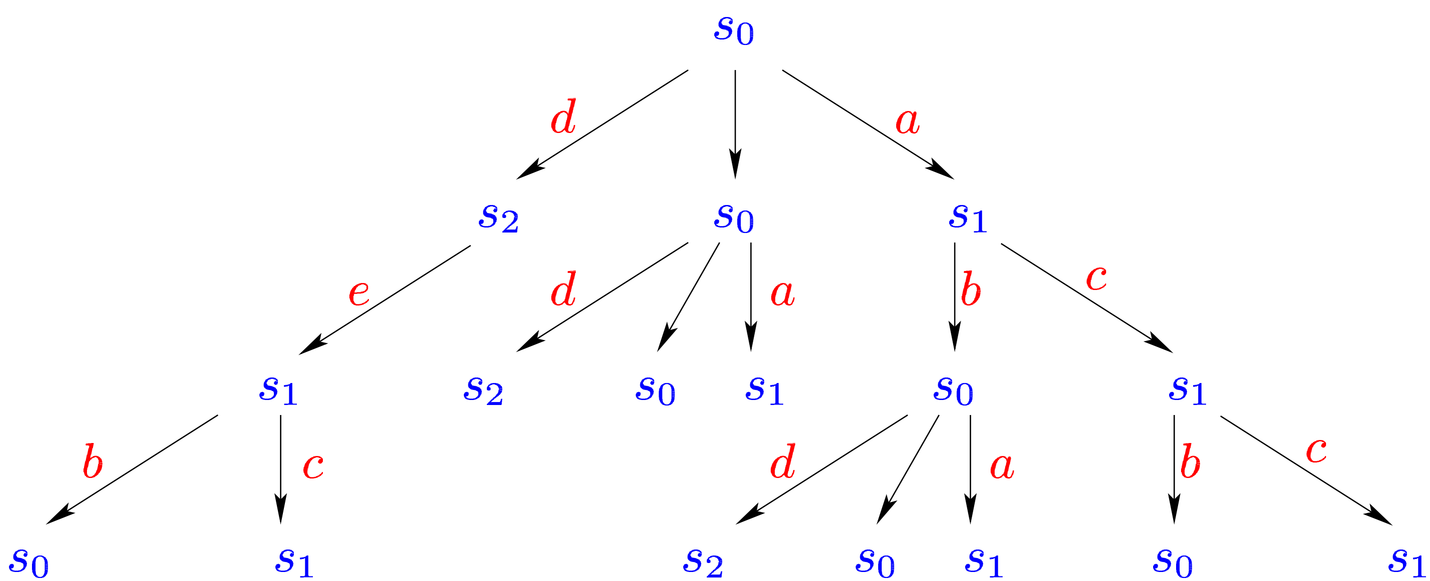
From unfoldings to trellises

- The well known Viterbi algorithm for max likelihood estimation of hidden state in stochastic automata uses another data structure: trellises.
- Trellises are obtained from unfoldings by merging identical futures of different runs, according to various observation criteria. Examples are:
  - length of the path σ;
  - visible length of the path σ;
  - projection $\mathbf{Proj}_{L'}(\sigma)$, where $L' \subset L$.

Formalization: $\theta : L \mapsto L_\theta$ observation criterion. Two paths σ and σ' of the unfolding are merged if they begin and end at identical states and produce identical words $\theta(\sigma) = \theta(\sigma')$.

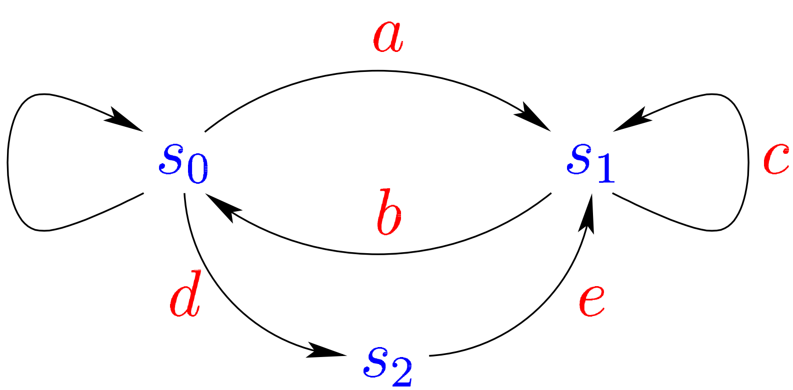
From unfoldings to trellises

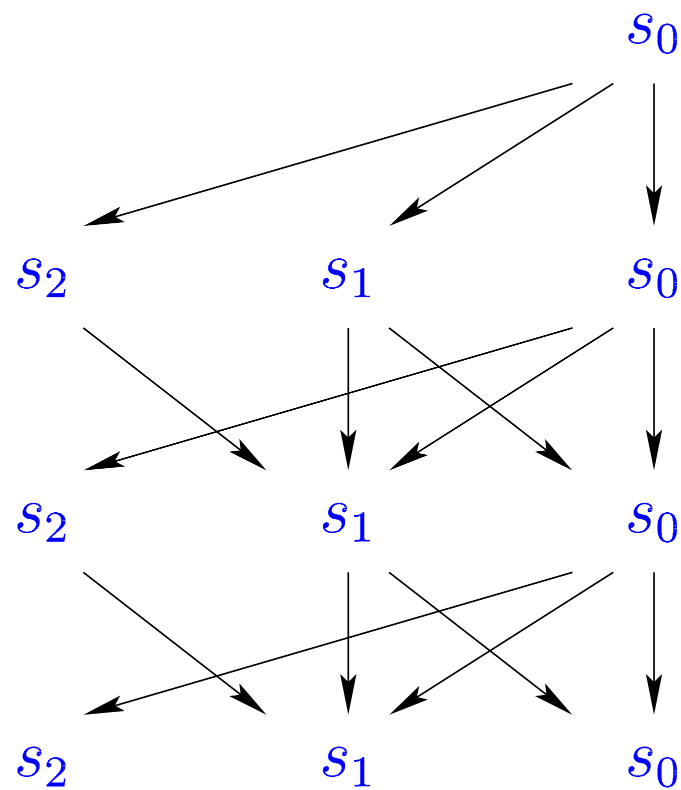




automaton $\mathcal{A}$ and its unfolding $\mathcal{U}_{\mathcal{A}}$

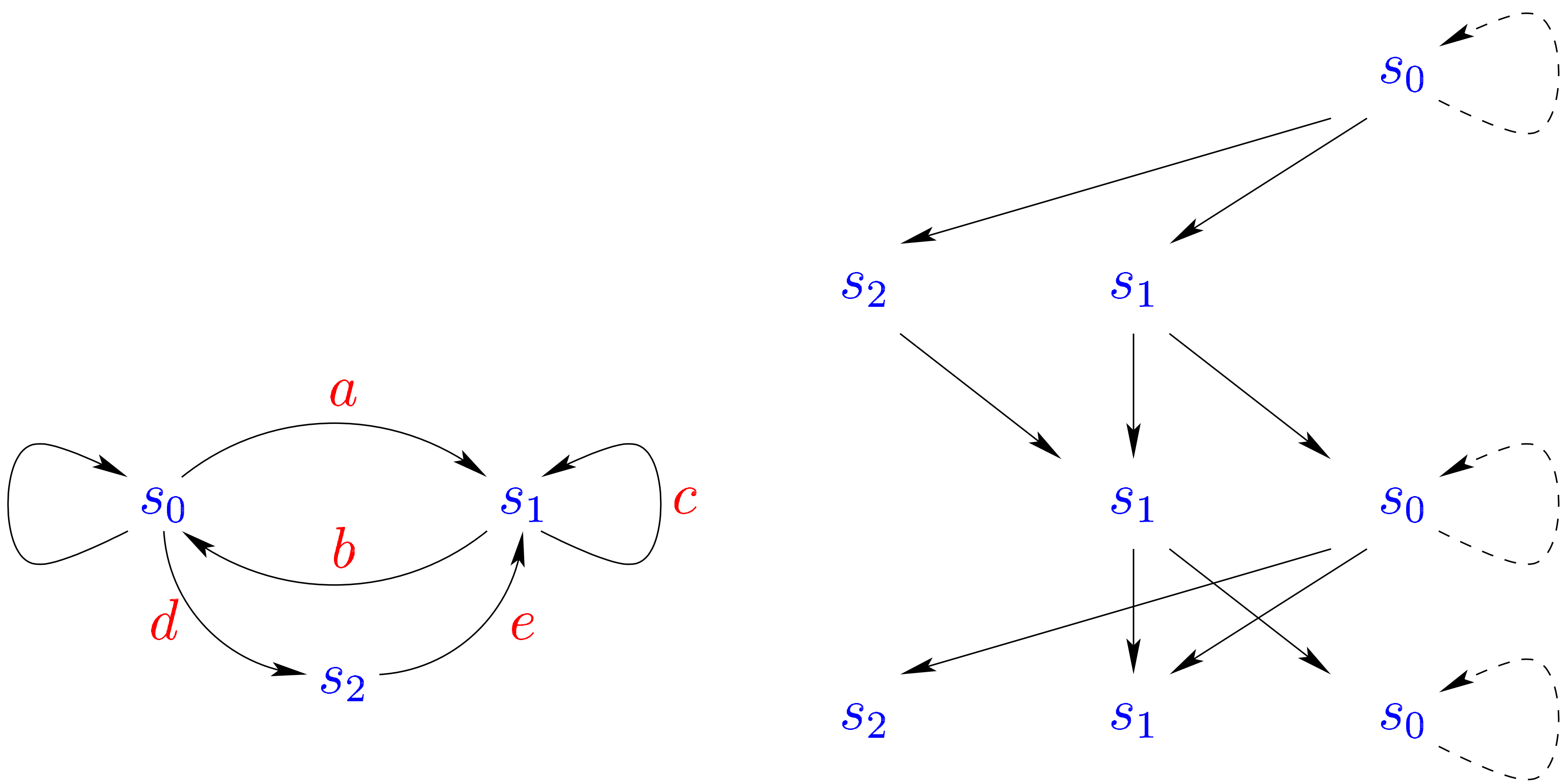
From unfoldings to trellises





automaton $\mathcal{A}$ and its trellis $\mathcal{T}_{\mathcal{A}}$
observation criterion = length of σ
$\theta : L \cup \{\star\} \mapsto \{1\}$

From unfoldings to trellises

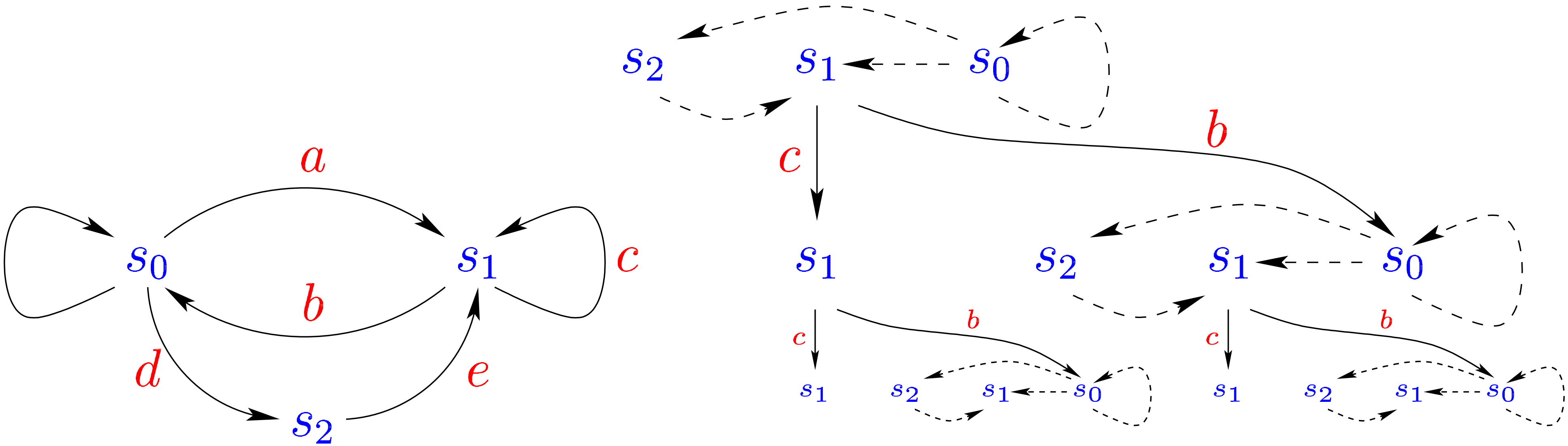


automaton $\mathcal{A}$ and its trellis $\mathcal{T}_{\mathcal{A}}$

observation criterion $=$ visible length of σ

$\theta : L \mapsto \{1\}$

From unfoldings to trellises



automaton $\mathcal{A}$ and its trellis $\mathcal{T}_{\mathcal{A}}$

observation criterion $= \mathbf{Proj}_{\{b,c\}}(\sigma)$

$$\theta = Id : \{b, c\} \mapsto \{b, c\}$$

Trellis-based monitoring: trial

Centralized monitoring: define

$$\mathcal{M} \;=_{\text{def}}\; \mathcal{T}^{\theta}_{\mathcal{A}\times O}$$

where θ is the length of $\mathbf{Proj}_{L_o}(\sigma)$, i.e., the length of O.

Trellis-based monitoring: trial

Centralized monitoring: define

$$\mathcal{M} =_{\mathrm{def}} \mathcal{T}^{\theta}_{\mathcal{A}\times O}$$

where θ is the length of $\mathbf{Proj}_{L_o}(\sigma)$, i.e., the length of O.

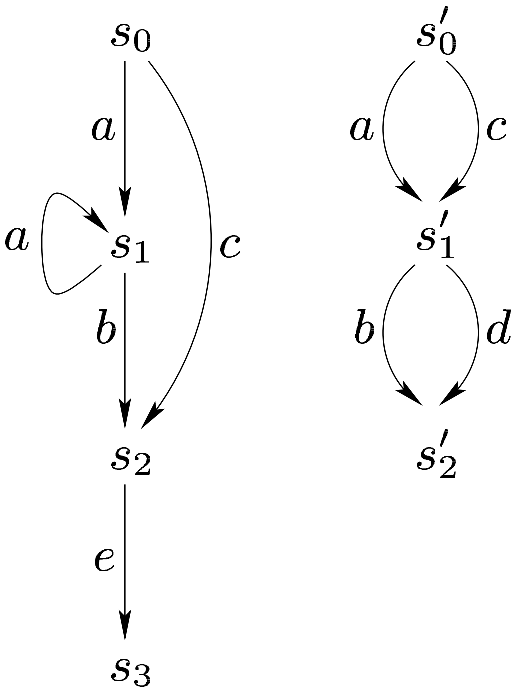
Trial: modular monitoring

Assume $\mathcal{A} = \bigtimes_{i\in I} \mathcal{A}_i, O = \bigtimes_{i\in I} O_i$
Attempt to define

$$\mathcal{M}_{\mathrm{mod}} =_{\mathrm{def}} \left(\mathbf{Proj}_i\left(\mathcal{T}^{\theta}_{\mathcal{A}\times O}\right)\right)_{i\in I}$$

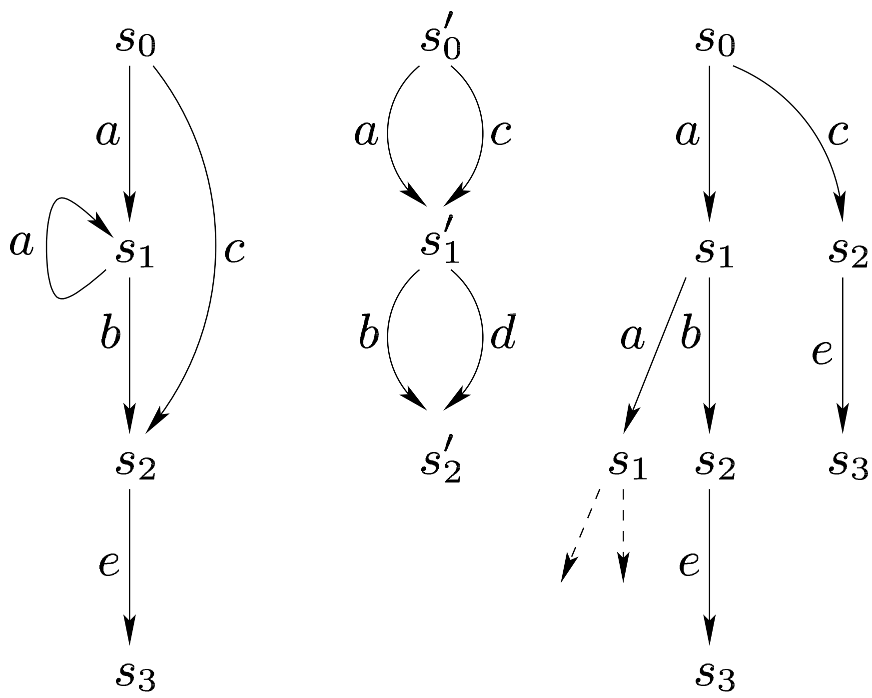
HHhhmmmm, there are problems . . .

Problems with trellises and projections



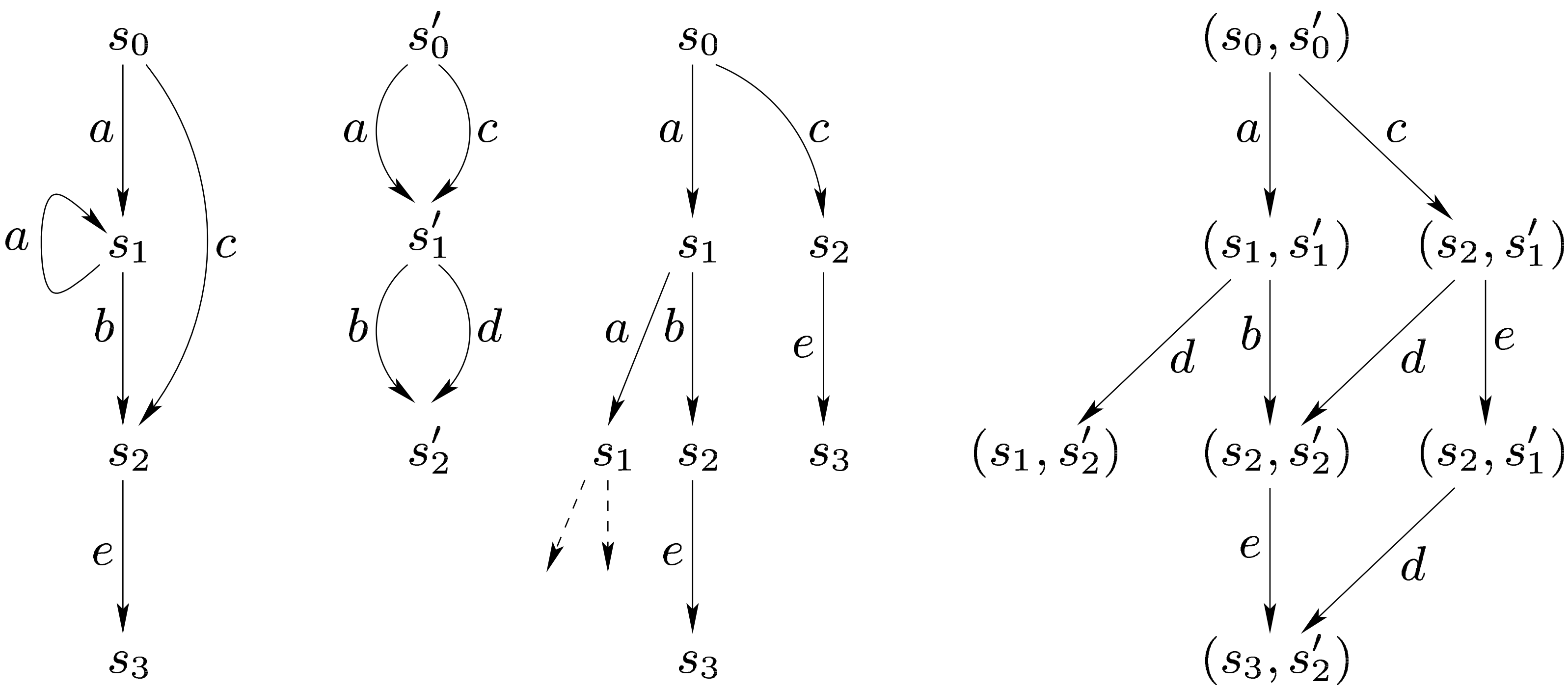
$\mathcal{A}$ $\qquad$ $\mathcal{A}' = \mathcal{T}_{\mathcal{A}',\theta'}$

Problems with trellises and projections



$\mathcal{A}$ $\quad$ $\mathcal{A}' = \mathcal{T}_{\mathcal{A}',\theta'}$ $\quad$ $\mathcal{T}_{\mathcal{A},\theta}$

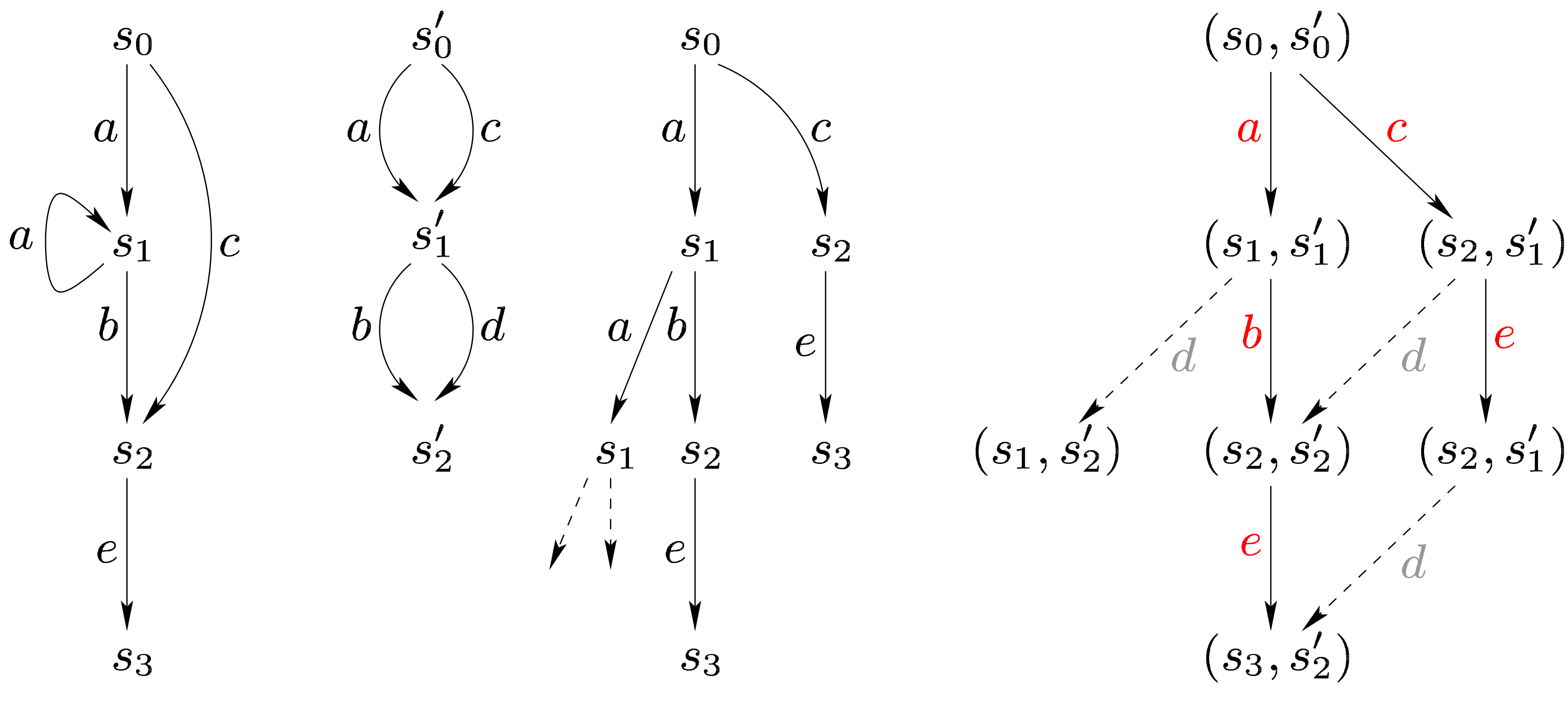
Problems with trellises and projections



$\mathcal{A}$ $\qquad$ $\mathcal{A}' = \mathcal{T}_{\mathcal{A}',\theta'}$ $\qquad$ $\mathcal{T}_{\mathcal{A},\theta}$ $\qquad$ $\mathcal{T}_{\mathcal{A}\times\mathcal{A}',\theta\sqcup\theta'}$

join of θ and θ'

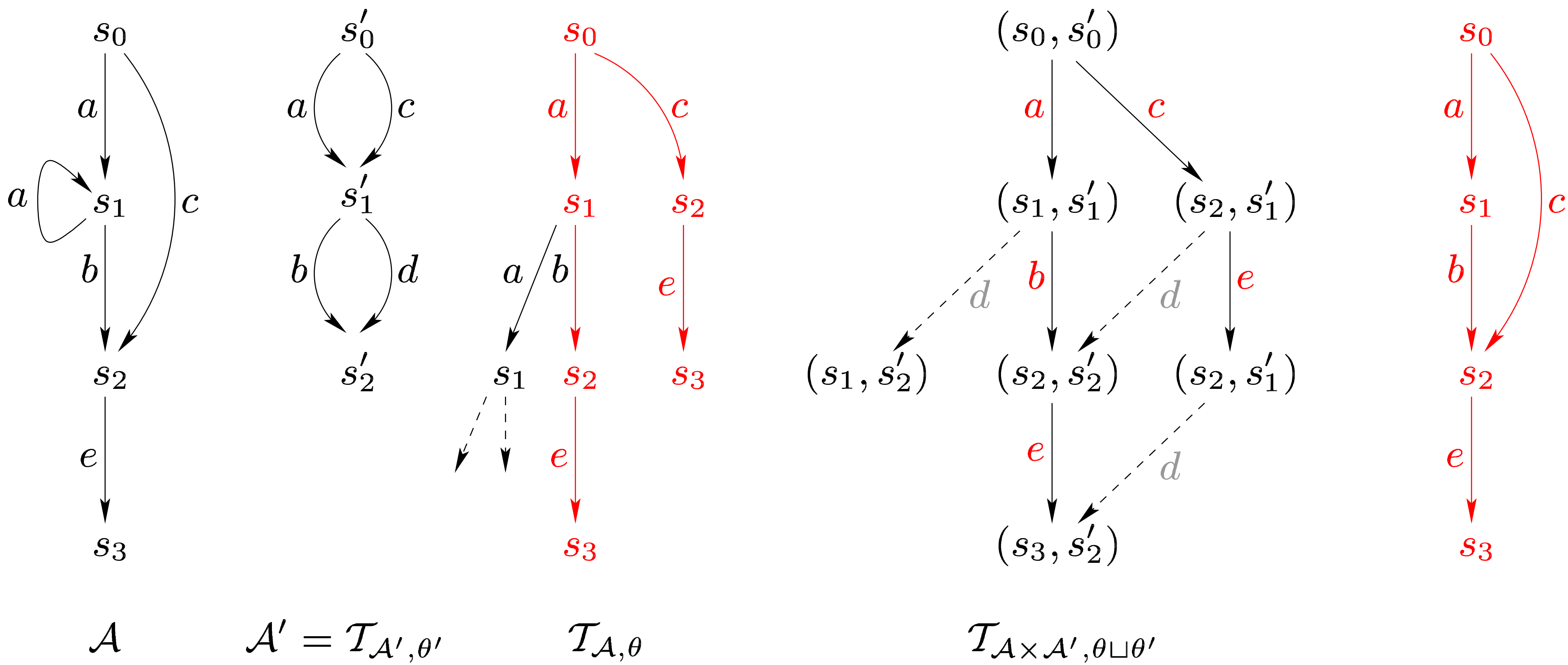
Problems with trellises and projections



$\mathcal{A}$ $\quad$ $\mathcal{A}' = \mathcal{T}_{\mathcal{A}',\theta'}$ $\quad$ $\mathcal{T}_{\mathcal{A},\theta}$ $\quad$ $\mathcal{T}_{\mathcal{A}\times\mathcal{A}',\theta\sqcup\theta'}$

$\mathbf{Proj}_{\{a,b,c,e\},\pi}\left(\mathcal{T}_{\mathcal{A}\times\mathcal{A}',\theta\sqcup\theta'}\right)$

Problems with trellises and projections

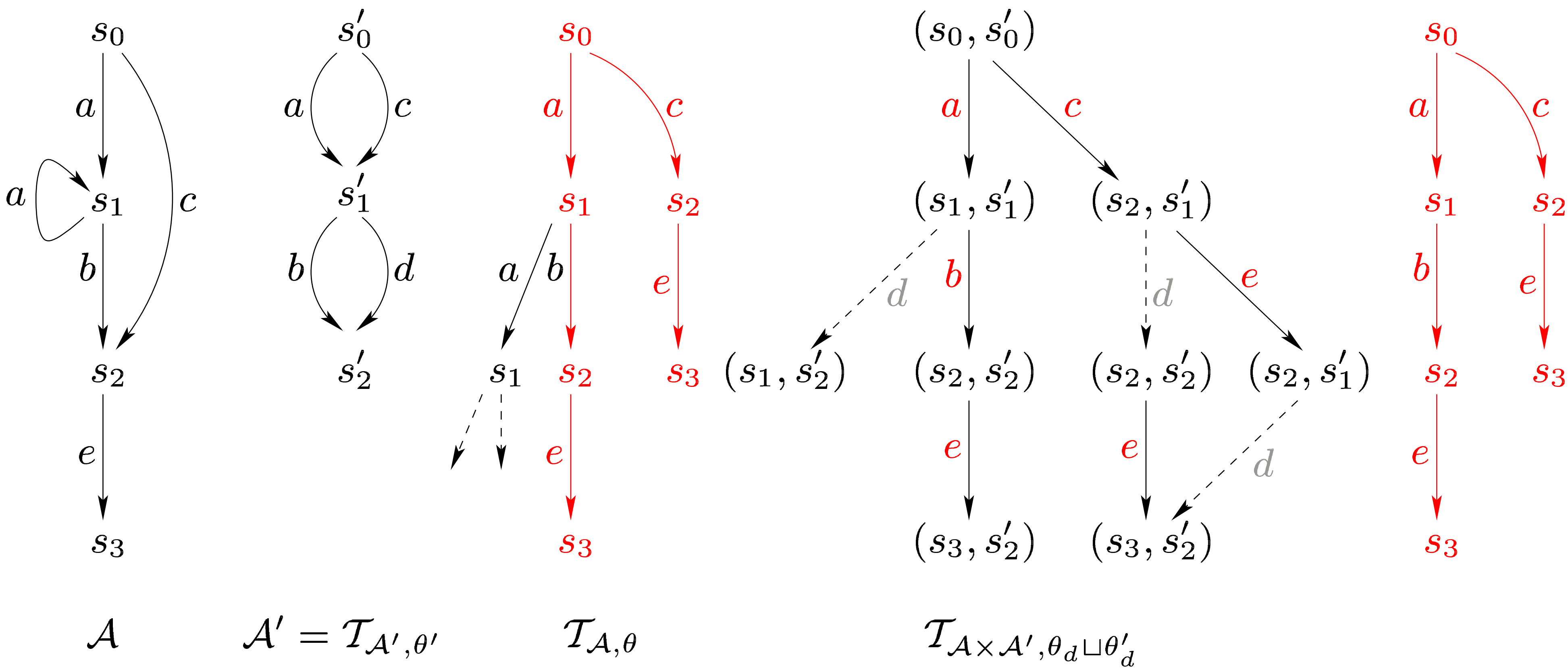


$\mathbf{Proj}_{\{a,b,c,e\},\pi}\,(\mathcal{T}_{\mathcal{A}\times\mathcal{A}',\theta\sqcup\theta'})$

the projection does not yield a valid trellis

the reason is that $\theta \sqcup \theta'$ counts the length of runs globally

Solution: distributable obs. criteria

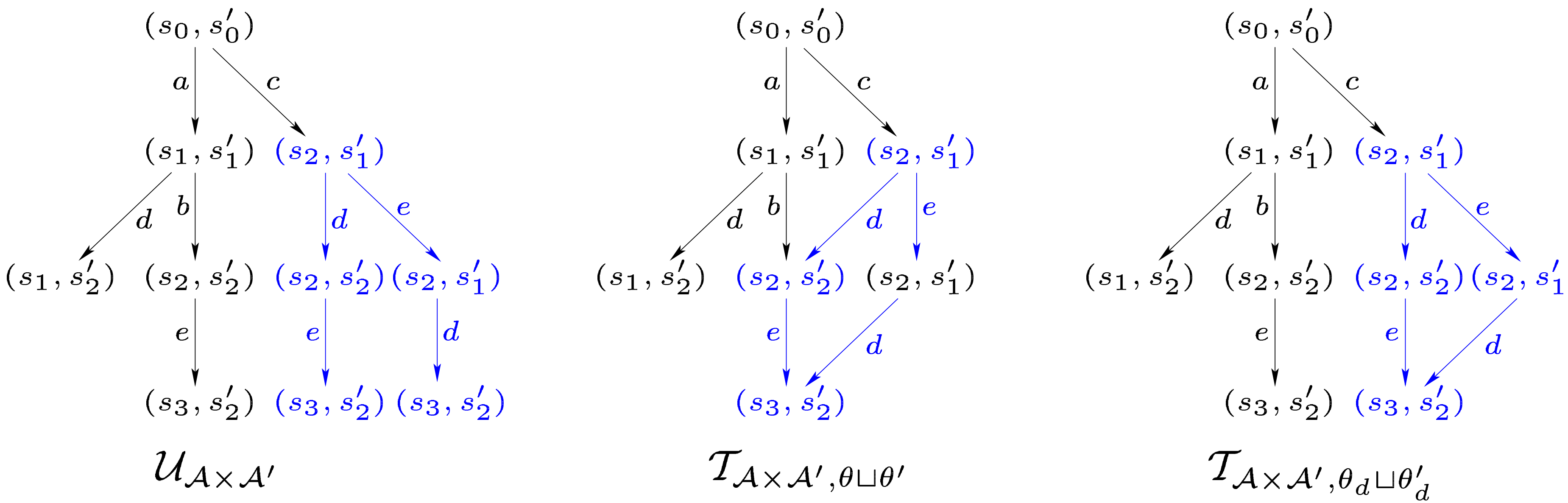


$\mathbf{Proj}_{\{a,b,c,e\},\pi}\left(\mathcal{T}_{\mathcal{A}\times\mathcal{A}',\theta_d\sqcup\theta'_d}\right)$

the projection yields a valid trellis

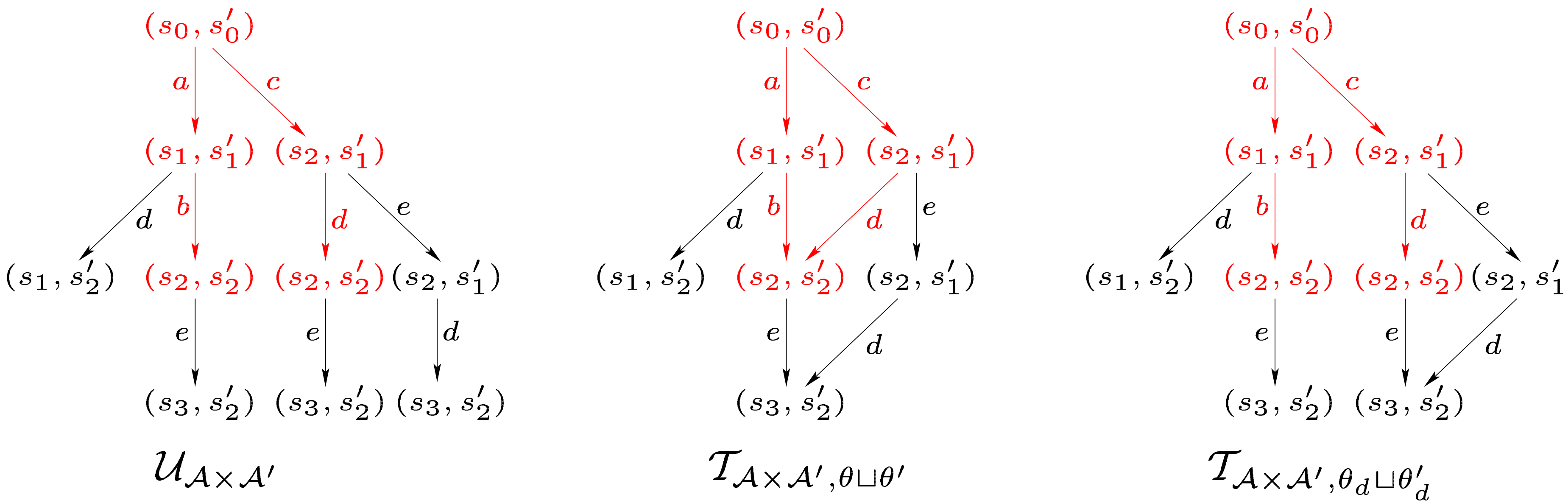
$\theta_d \sqcup \theta'_d$ counts the length of runs locally, in each component

Solution: distributable obs. criteria



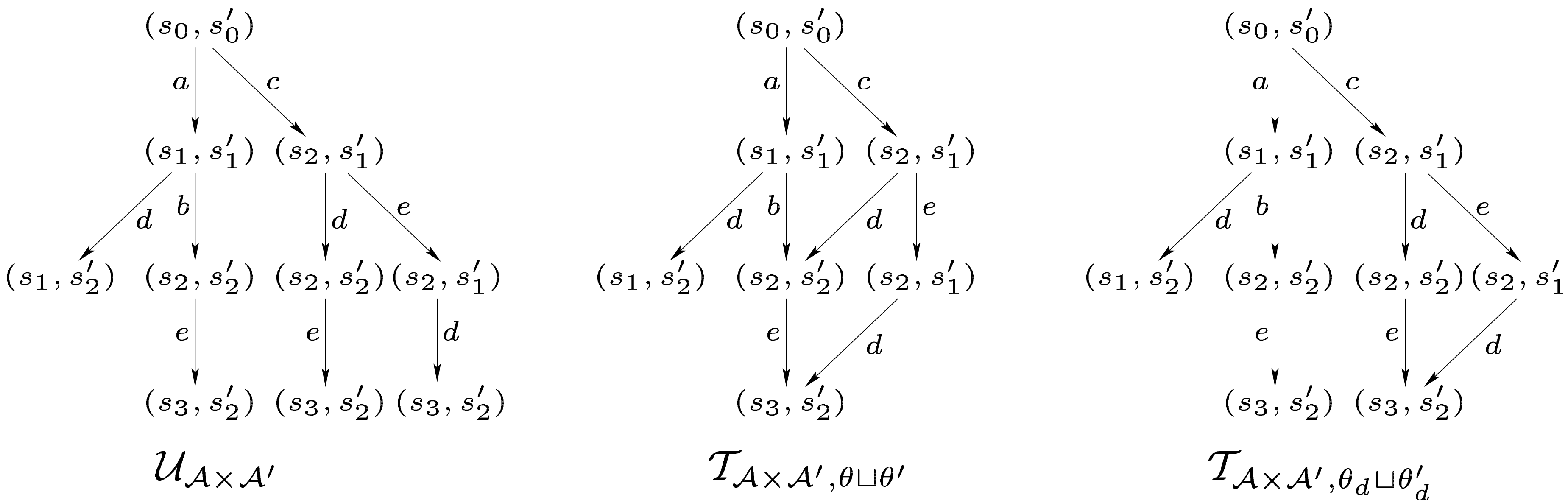
- $\theta \sqcup \theta'$ counts the length of runs, globally
- $\theta_d \sqcup \theta'_d$ is a vector counter that counts the length of runs in each component;
  $\theta_d \sqcup \theta'_d$ is distributable: compatible with projections
- With distributable criteria, trellises can be factorized and belief propagation works.

Solution: distributable obs. criteria



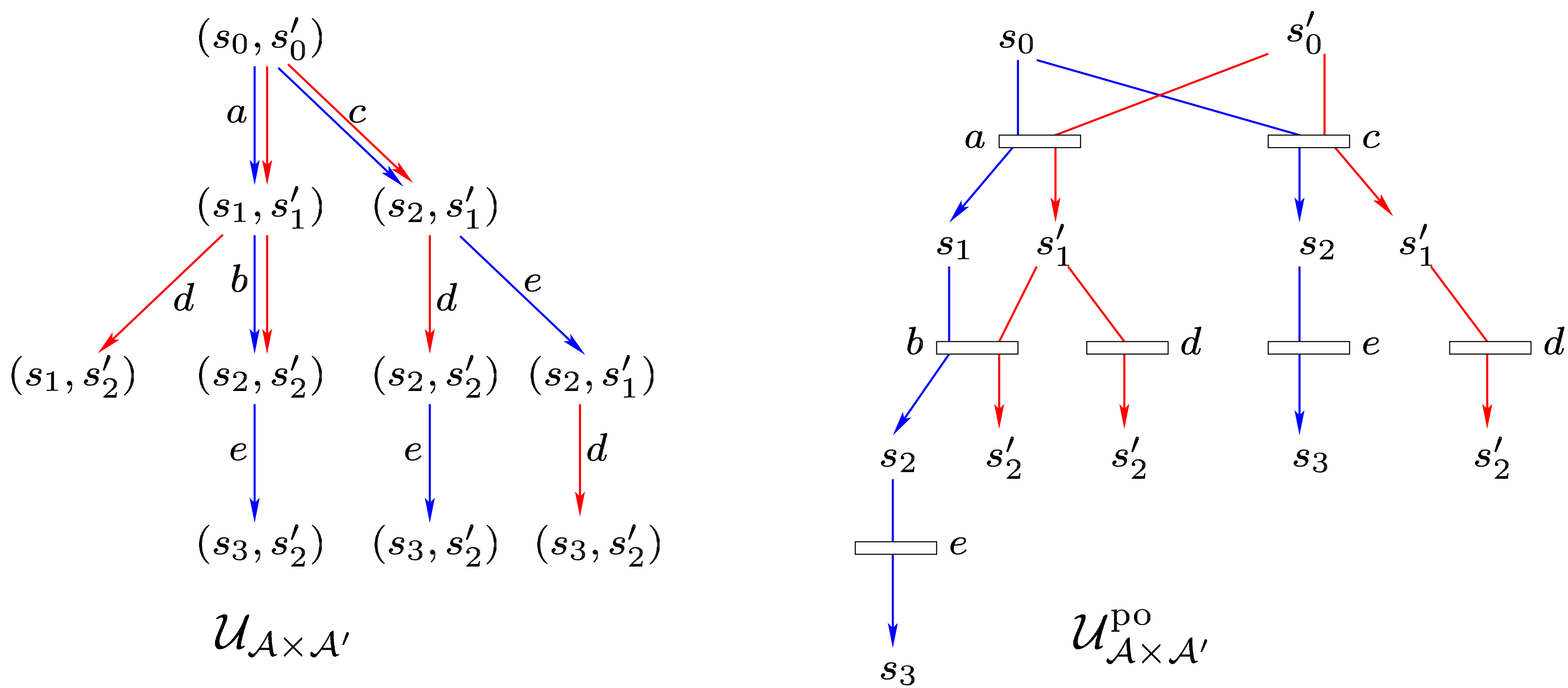
- $\theta \sqcup \theta'$ counts the length of runs, globally
- $\theta_d \sqcup \theta'_d$ is a vector counter that counts the length of runs in each component;
  $\theta_d \sqcup \theta'_d$ is distributable: compatible with projections
- With distributable criteria, trellises can be factorized and belief propagation works.

Solution: distributable obs. criteria



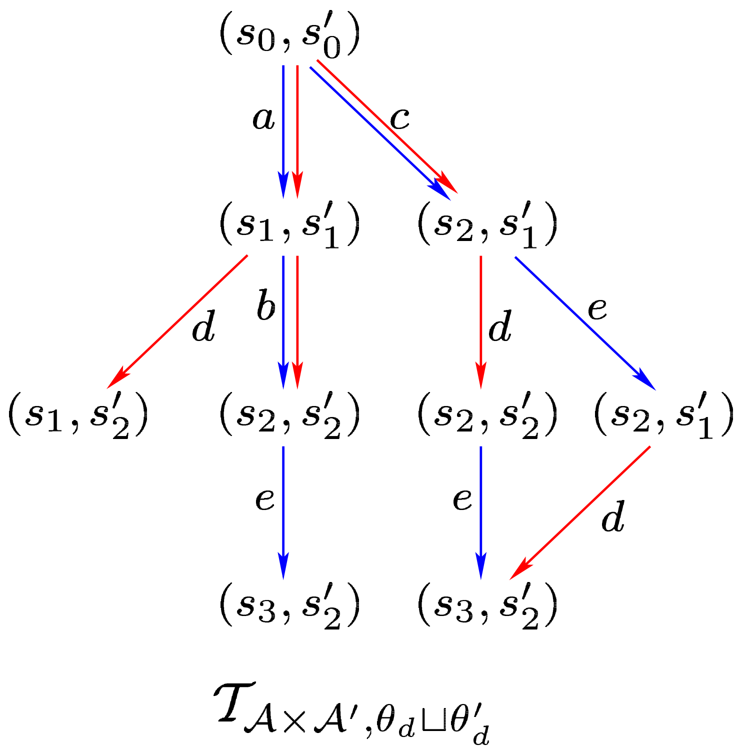
- $\theta_d \sqcup \theta'_d$ is a vector counter that counts the length of runs in each component;
- The idea of using *vector clocks* is not new. It was proposed by Fidge [1991] and Mattern [1989] for the distributed reconstruction of coherent states in distributed systems — e.g., for checkpointing.

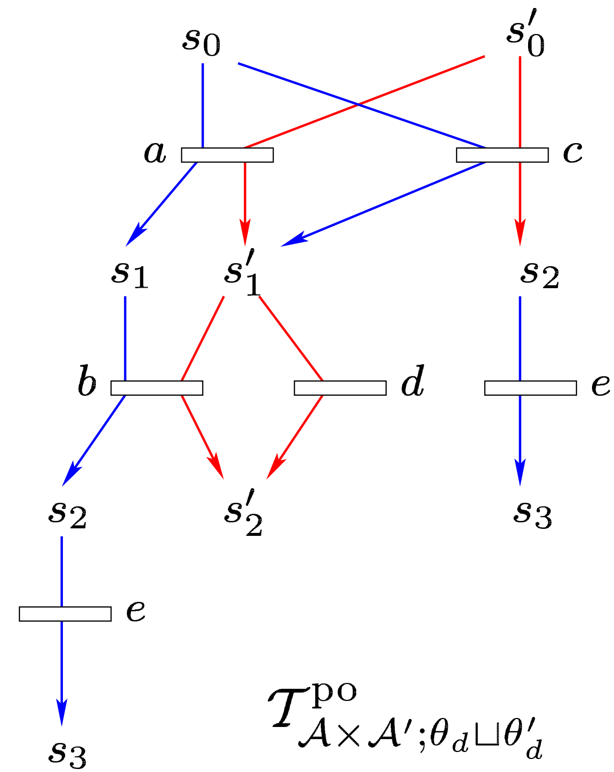
Moving to partial orders



With a distributable observation criterion, global runs are best seen as the synchronization of local runs, i.e., as *partial orders* — note the reduction in the number of events, due to concurrency.

Moving to partial orders





With a distributable observation criterion, global runs are best seen as the synchronization of local runs, i.e., as *partial orders* — note the reduction in the number of events, due to concurrency.

Moving to partial orders

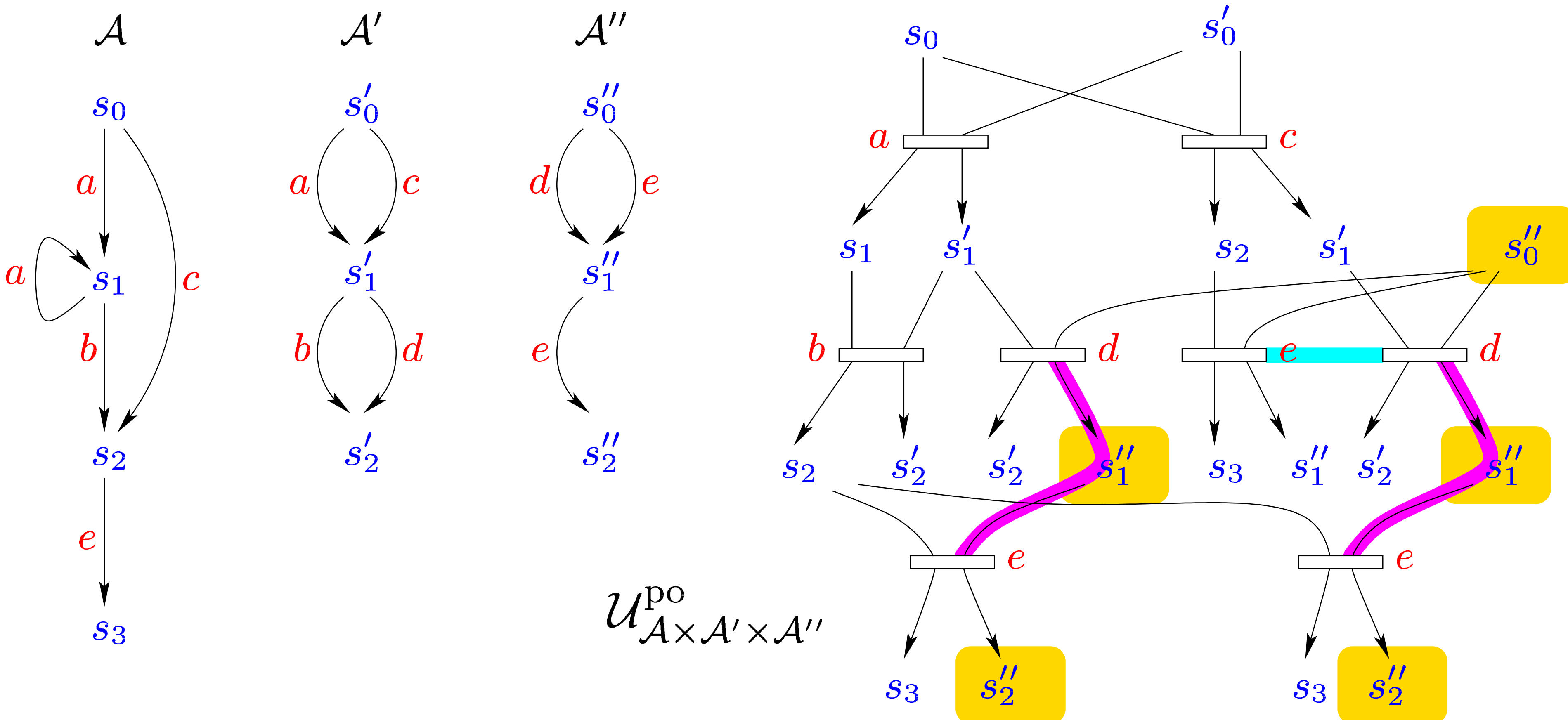
For large distributed systems, internal concurrency and asynchrony may exist also within each local subsystem.

Thus, with the above data structures, state explosion may occur, locally to each subsystem.

Therefore it is advisable to use partial order data structures also to represent the sets of local runs.

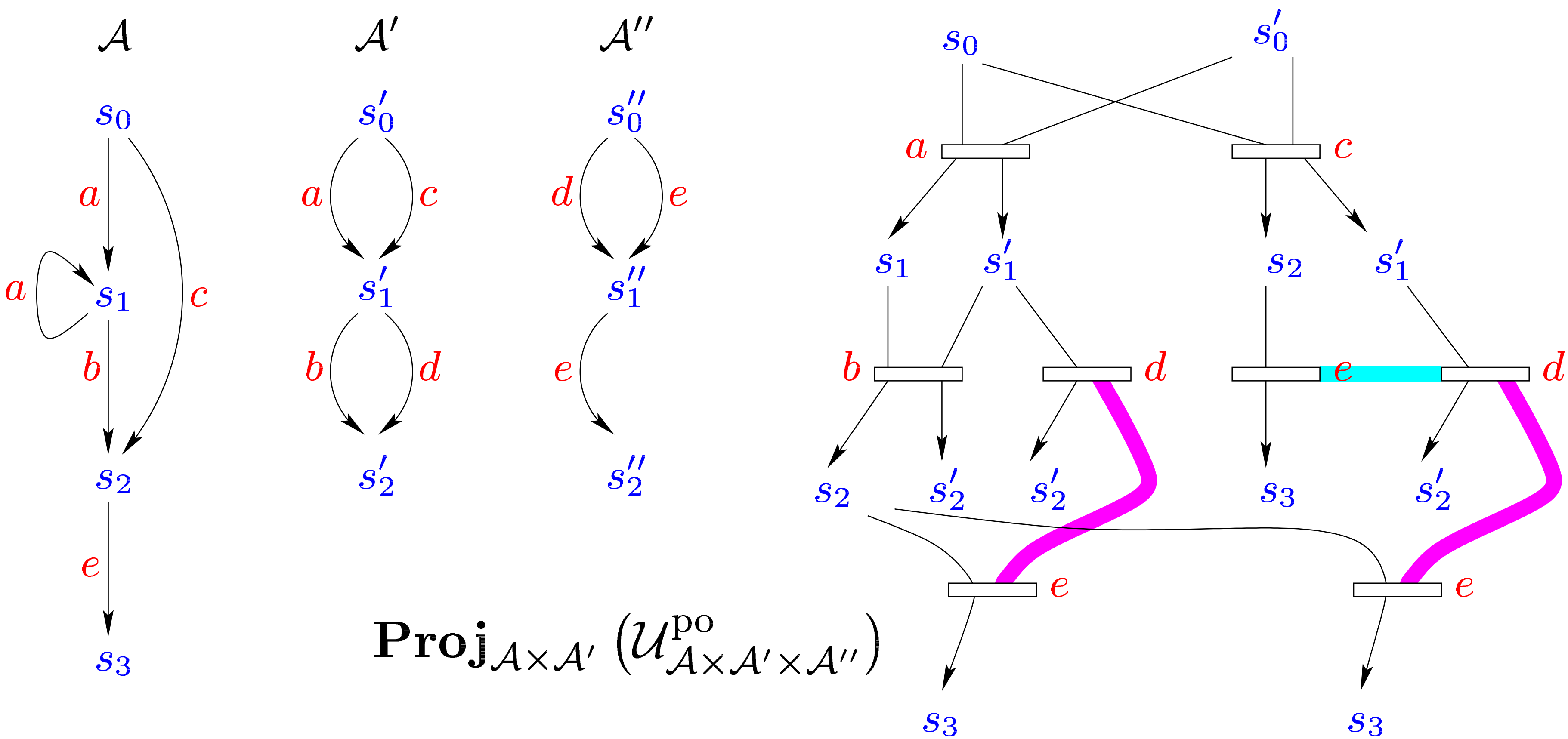
While efficiency increases by doing so, new problems appear . . .

Moving to partial orders



some causalities and conflicts are caused by $\mathcal{A}''$

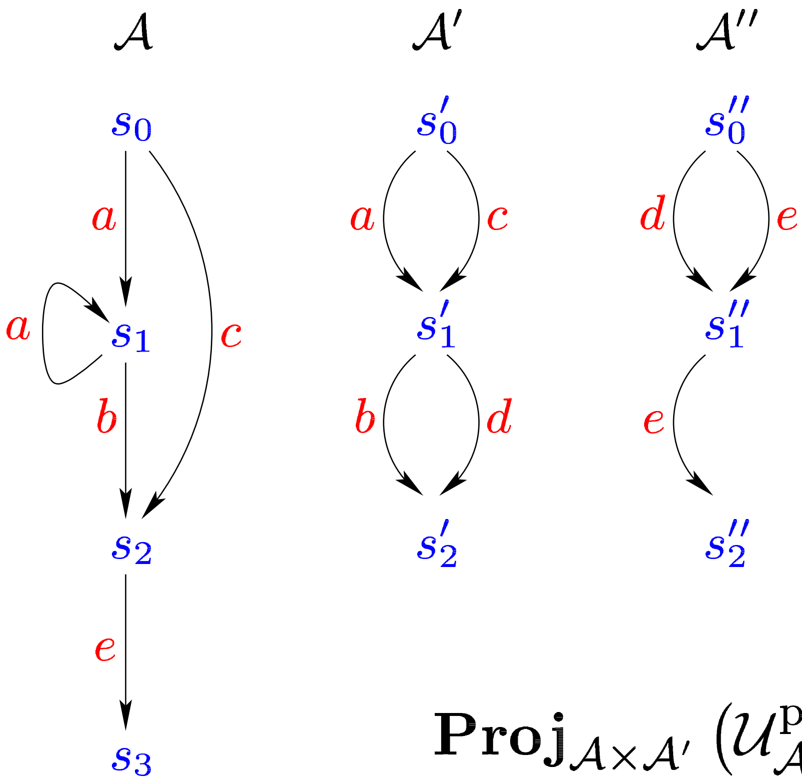
Moving to partial orders



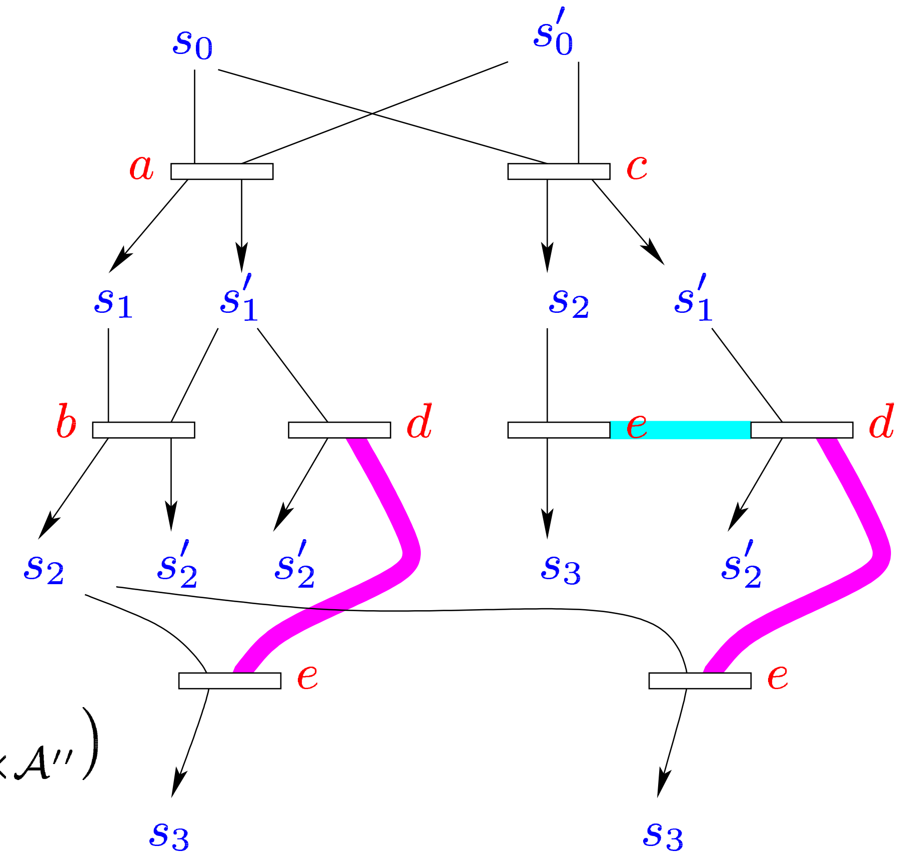
$$\mathbf{Proj}_{\mathcal{A}\times\mathcal{A}'}\left(\mathcal{U}^{\mathrm{po}}_{\mathcal{A}\times\mathcal{A}'\times\mathcal{A}''}\right)$$

some causalities and conflicts are caused by $\mathcal{A}''$
projecting $\mathcal{A}''$ away make them dangling

Moving to partial orders



$$\mathbf{Proj}_{\mathcal{A}\times\mathcal{A}'}\left(\mathcal{U}^{\mathrm{po}}_{\mathcal{A}\times\mathcal{A}'\times\mathcal{A}''}\right)$$



solutions exist, e.g., moving to *event structures* where only event are involved and causality/conflict is encoded explicitly

Other issues

Problem *Address changes in the systems dynamics.*

Solution: unfolding dynamic Petri nets, or Graph Grammars (in progress)

Other issues

Problem: *Address changes in the systems dynamics.*

Solution: unfolding dynamic Petri nets, or Graph Grammars (in progress)

Problem: *Address incomplete models*

Solution???

Other issues

Problem: *Address changes in the systems dynamics.*

Solution: unfolding dynamic Petri nets, or Graph Grammars (in progress)

Problem: *Address incomplete models*

Solution???

Problem: *Address nondeterminism in monitors via concurrent probabilistic models*

Solution: true concurrency probabilistic models, by Samy Abbes [2004, 2005]

Other issues

Problem: *Address changes in the systems dynamics.*

Solution: unfolding dynamic Petri nets, or Graph Grammars (in progress)

Problem: *Address incomplete models*

Solution???

Problem: *Address nondeterminism in monitors via concurrent probabilistic models*

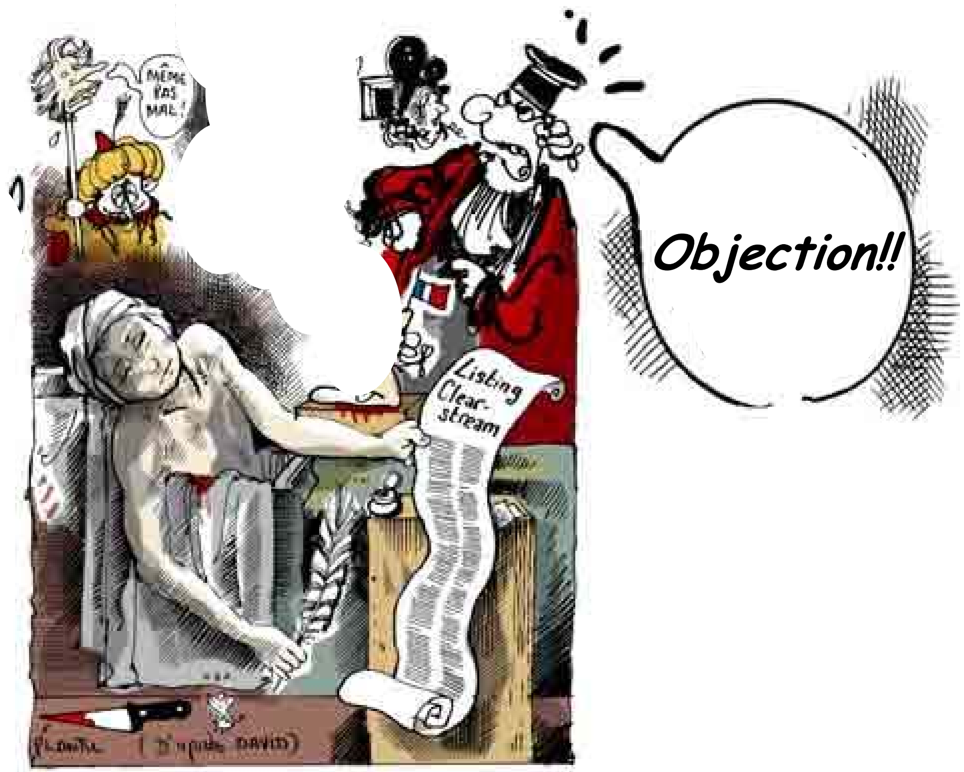
Solution: true concurrency probabilistic models, by Samy Abbes [2004, 2005]

Problem: *Address timing aspects in monitors*

Solution: unfolding timed Petri nets, Chatain [2005]

Partial orders needed…

Nancy, help!!



Coming tomorrow...

Partial orders needed...

Nancy, help!!