Automating the Addition of Fault Tolerance with Discrete Controller Synthesis

Alain Girault and Eric Rutten

POP ART

INRIA RHÔNE-ALPES

22 January 2009

Outline

Outline

Problem and motivations

Motivation

Automate the addition of fault-tolerance in embedded systems with Discrete Controller Synthesis (DCS)

Provide a complete framework

- Based on a formal method (DCS) to provide a guaranteed fault-tolerant level
- Encompassing all kinds of hardware devices : processors, communication links, sensors, actuators
- Taking into account all kinds of failures : crashes, value failures, Byzantine
- Providing quantitative results thanks to optimal DCS

Discrete Controller Synthesis (DCS)

Aka supervisory control [Ramadge & Wonham, SIAM J. Cont. Opt., 1987]

Discrete controller synthesis problem

Given the plant $\mathcal{U}$ and the desired system $\mathcal{D}$, find the controller $\mathcal{C}$ s.t. :

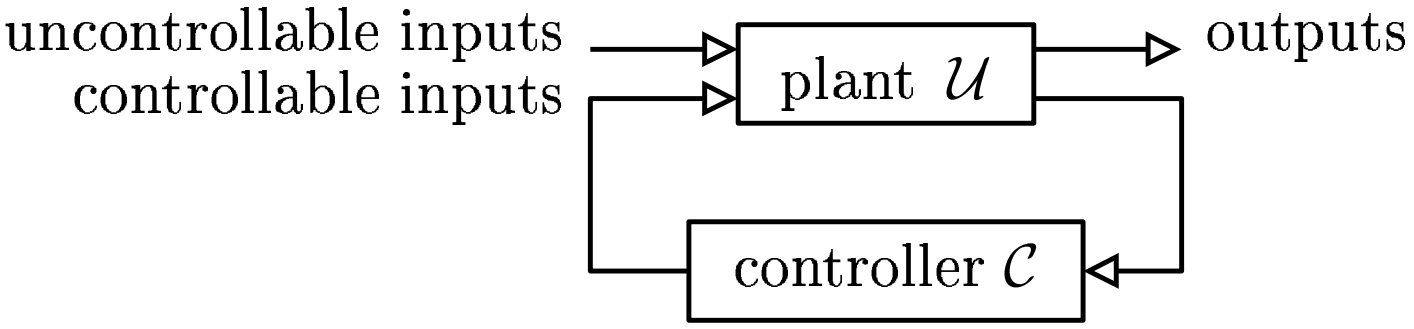
$$\mathcal{U} \cap \mathcal{C} \subseteq \mathcal{D}$$

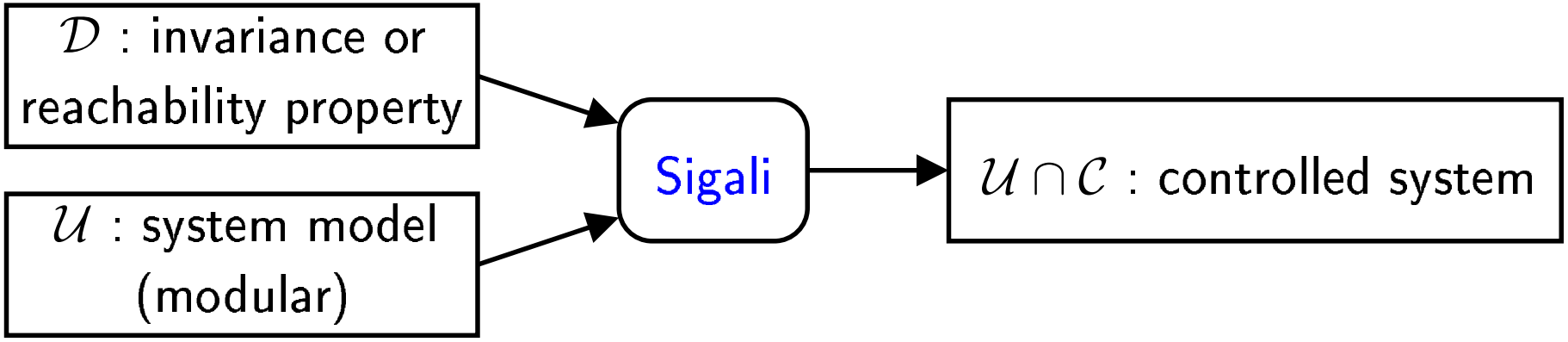
The language $\mathcal{U}$ is built over the set of events $\mathcal{I}$

$\mathcal{I}$ is partitioned into controllable $\mathcal{I}_c$ and uncontrollable $\mathcal{I}_u$

$\mathcal{C}$ can only act on the events in $\mathcal{I}_c$

Closed-loop controlled system and SIGALI





SIGALI : [http ://www.irisa.fr/vertecs/Logiciels/sigali.html]

DCS for Fault-tolerance : our contributions

Features of our method

- The possibility to try several fault hypotheses on the same specification.
- The possibility to evaluate several fault tolerance requirements.
- In the final program, the guarantee by construction of the fault tolerance level required by the user.

Features specific to DCS

- The failure recovery mechanism is dynamic (hence it does not induce too much redundancy overhead like static methods).
- With a static guarantee on the fault tolerance of the obtained system (unlike dynamic methods).

Formal models for DCS I

Labeled Transition Systems (LTS) $S = \langle \mathcal{Q}, q^0, \mathcal{I}, \mathcal{O}, \longrightarrow \rangle$

- $\mathcal{Q}$ is the set of states and $q^0 \in \mathcal{Q}$ is the initial state
- $\mathcal{I}$ is the set of input events, partitioned into $\mathcal{I}_c$ and $\mathcal{I}_u$
- $\mathcal{O}$ is the set of output events
- $\longrightarrow \subseteq \mathcal{Q} \times \mathcal{B}ool(\mathcal{I}) \times \mathcal{O}^* \times \mathcal{Q}$ is the transition relation

Synchronous product $\langle \mathcal{Q}_1, q_1^0, \mathcal{I}_1, \mathcal{O}_1, \longrightarrow_1 \rangle \,||\, \langle \mathcal{Q}_2, q_2^0, \mathcal{I}_2, \mathcal{O}_2, \longrightarrow_2 \rangle$

$$\langle \mathcal{Q}_1 \times \mathcal{Q}_2, (q_1^0, q_2^0), \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \longrightarrow \rangle$$

with $(q_1, q_2) \xrightarrow{g_1 \wedge g_2 / a_1 \wedge a_2} (q_1', q_2')$ iff $q_1 \xrightarrow{g_1/a_1}_1 q_1'$ and $q_2 \xrightarrow{g_2/a_2}_2 q_2'$

Formal models for DCS II

Path

A path in a LTS is a finite sequence of transitions
$q_1 \xrightarrow{g_1/a_1} q_2 \xrightarrow{g_2/a_2} q_3 \cdots q_n \xrightarrow{g_n/a_n} q_{n+1}$.

Reachability

A state q of $\mathcal{Q}$ is reachable iff there exists a path from the initial state q^0 to q.

Invariance

A set of states E is invariant iff any transition having as source a state of E has its destination state in E.

DCS functions and objectives

$S' = mk\_invar\,(S, E)$ for $E \subseteq S$

Returns a controlled system S' such that all controllable transitions leading to states $q_{i+1} \notin E$ are inhibited

$S' = kp\_reach\,(S, E)$ for $E \subseteq S$

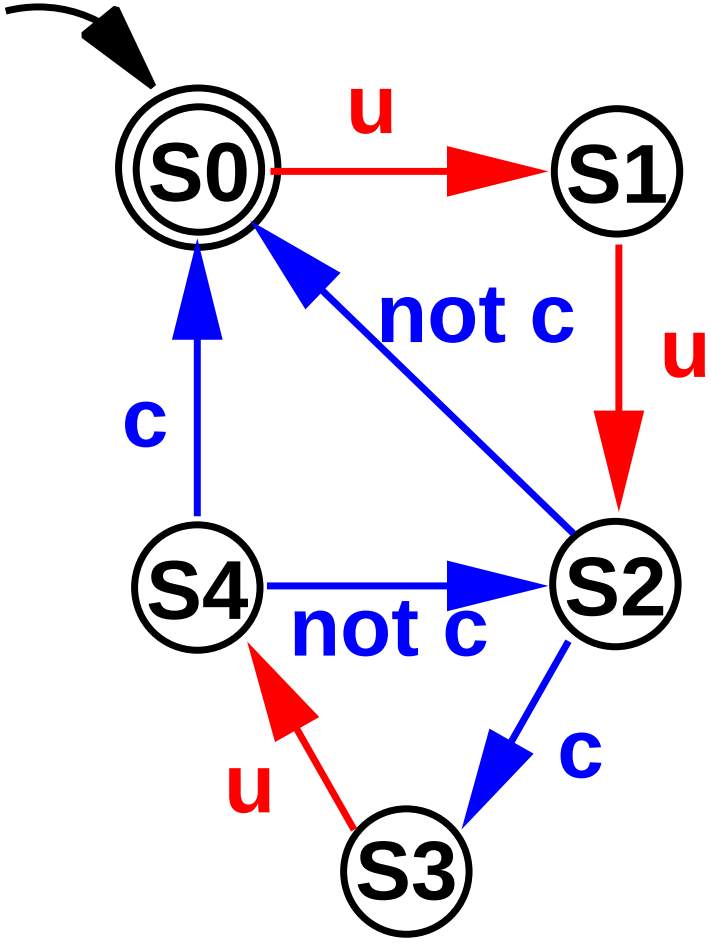
Returns a controlled system S' such that the controllable transitions entering subsets of states from where E is not reachable are disabled

$E' = reach\_under\_control\,(S, E)$ for $E \subseteq S$

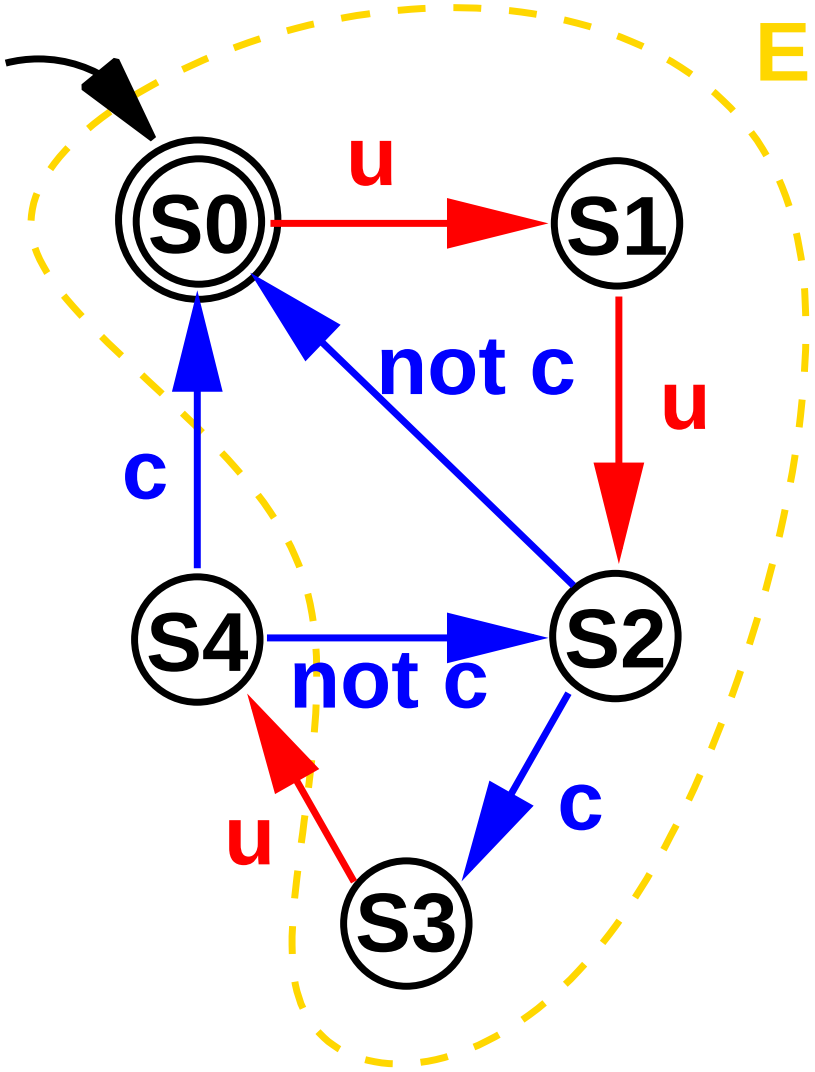
Returns a subset E' of the states of S such that states in E' are reachable by controllable transitions

$$kp\_reach\,(S, E) = mk\_invar\,(S, reach\_under\_control\,(S, E))$$

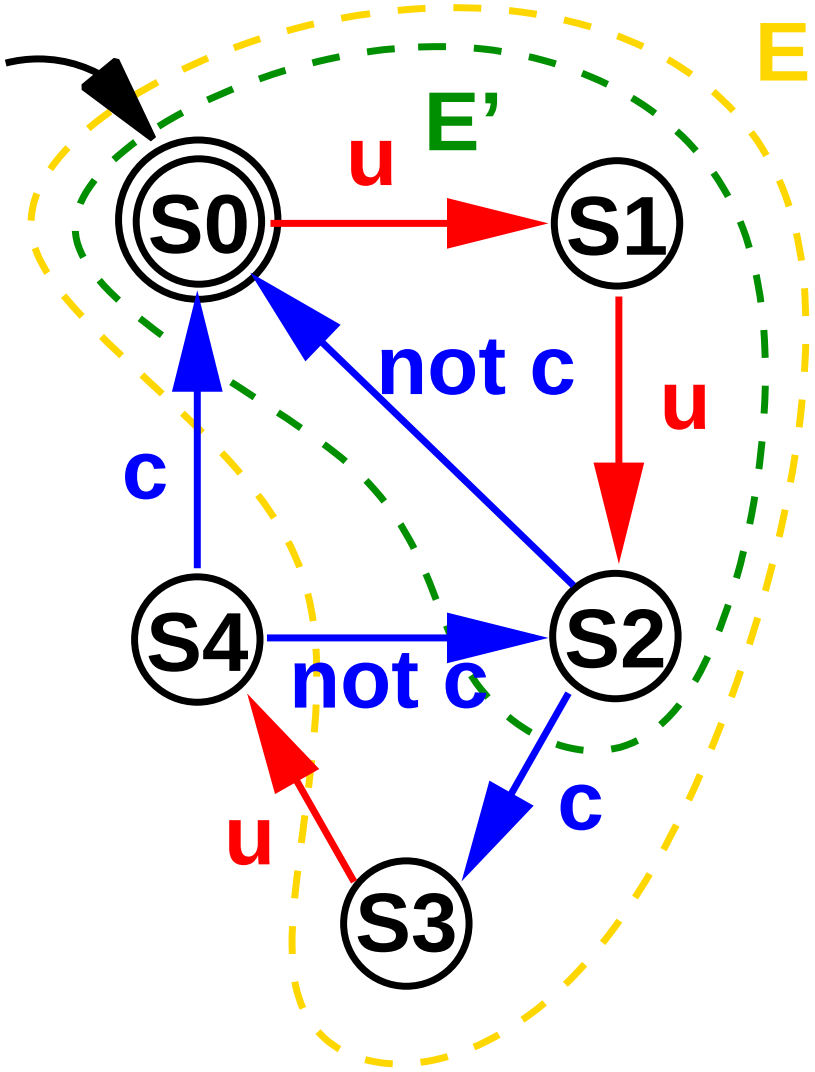
A short example of DCS



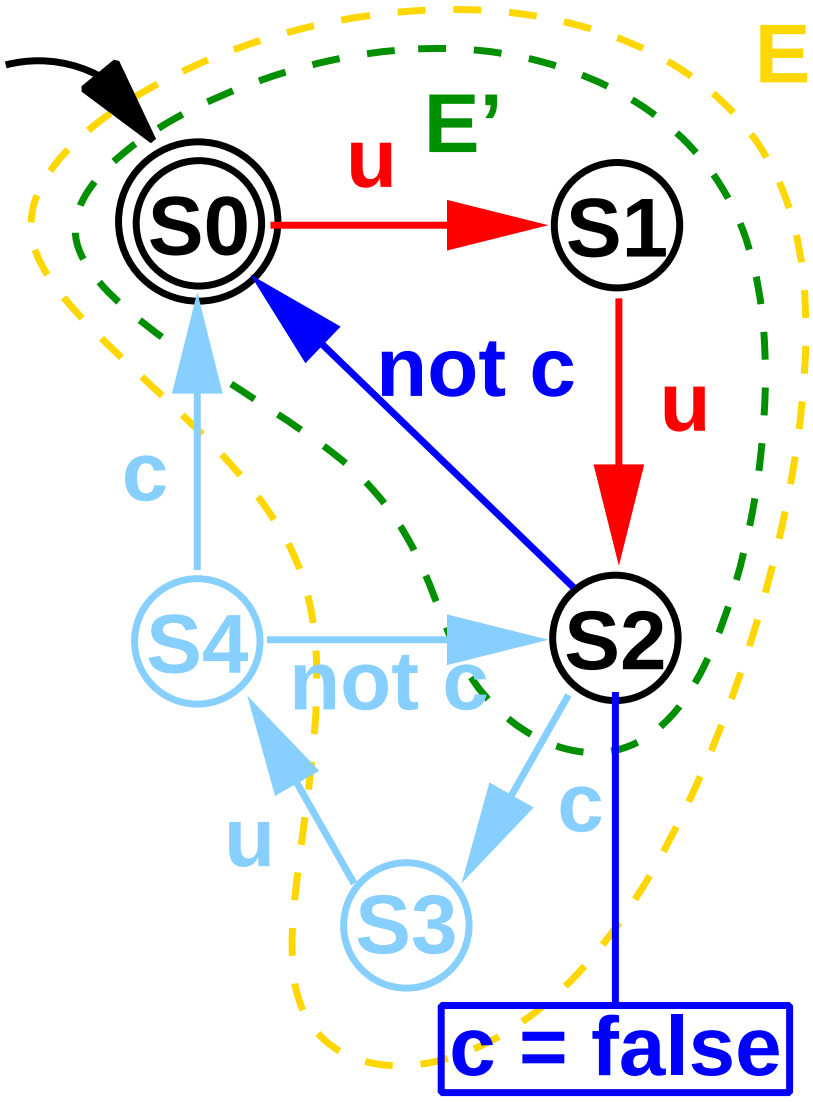
A short example of DCS



A short example of DCS



A short example of DCS

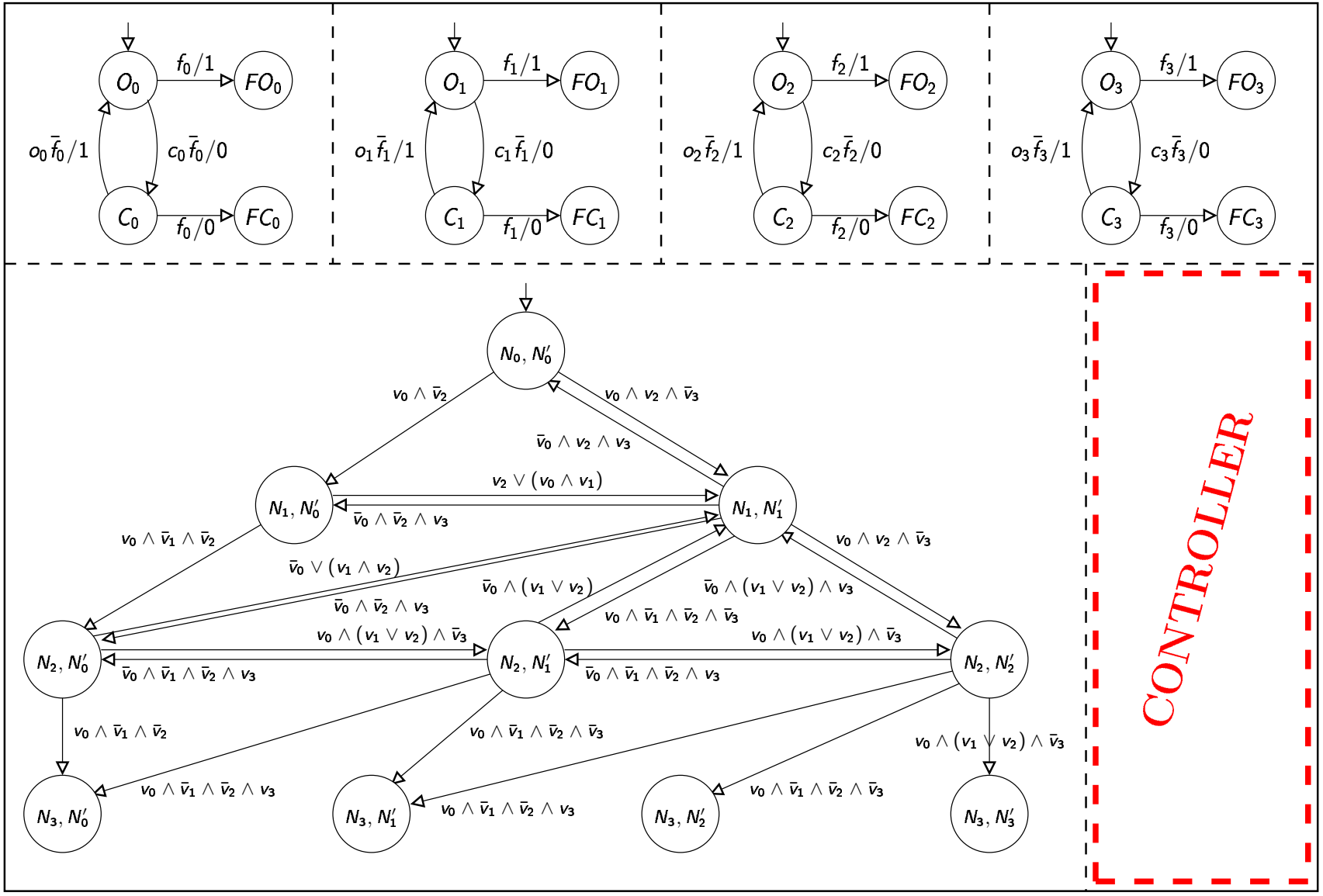


Outline

Framework for automating the addition of FT with DCS

- Only the hardware components can fail.
- The faults are uncontrollable events.
- The plant $\mathcal{U}$ is specified modularly as the synchronous product of several LTSs.
- The plant $\mathcal{U}$ represents all the possible behaviors, both the good ones and the bad ones.
- The fault model is specified as an LTS (composed synchronously with $\mathcal{U}$).
- The fault tolerance policy is specified as a reachability or invariance property (or more generally as a synchronous observer).

Defining the fault-intolerant system



Defining the fault hypothesis

- What HW components can fail
- If several components can fail, a list of failure configurations
- A fault model for each component that can fail
  - ➪ An LTS that specifies :
    - the behavior of the component when it fails
    - the duration of the failure

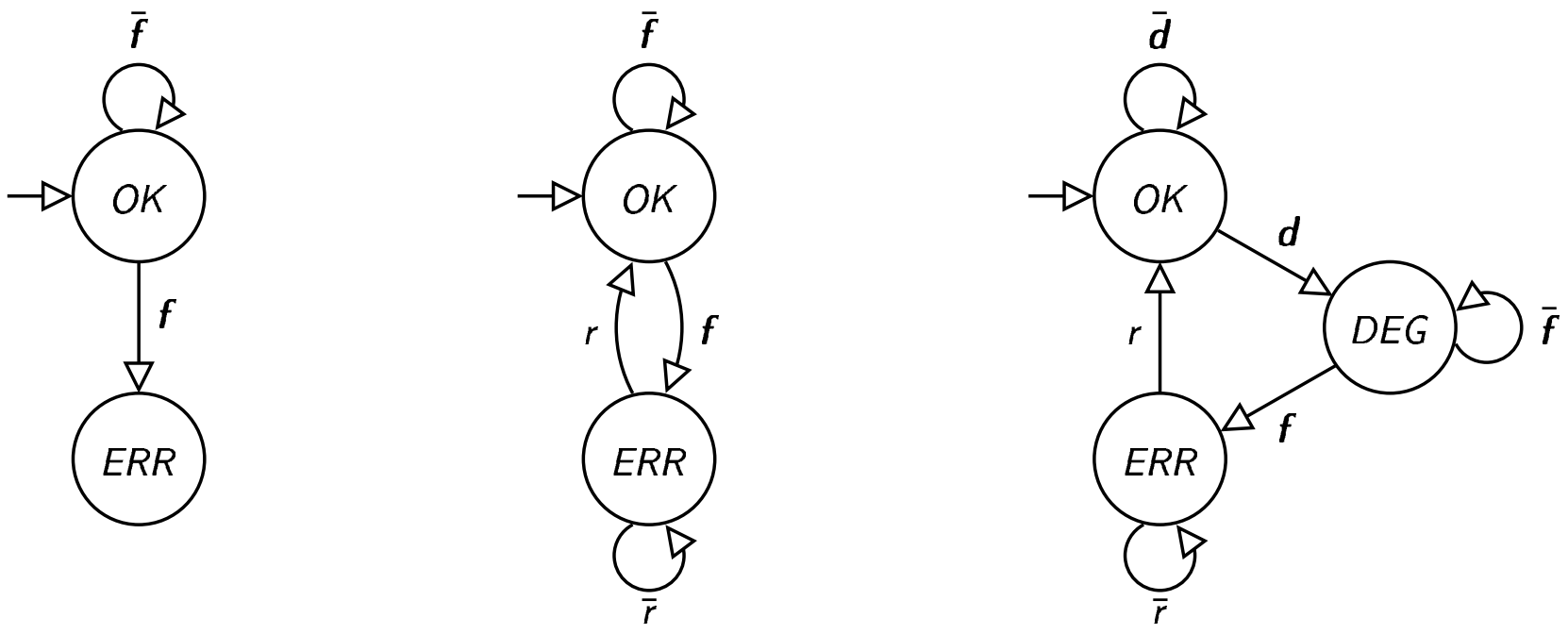
Defining a fault tolerance policy

Specified as a DCS control objective

- Invariance property, e.g. :
  - ➪ a subset of good behaviors should be invariant
  - ➪ a subset of bad behaviors should never be reached
- Reachability property, e.g. :
  - ➪ a subset of good behaviors should always be reachable
- More complex properties :
  - ➪ conditioned objectives
  - ➪ synchronous observers

Outline

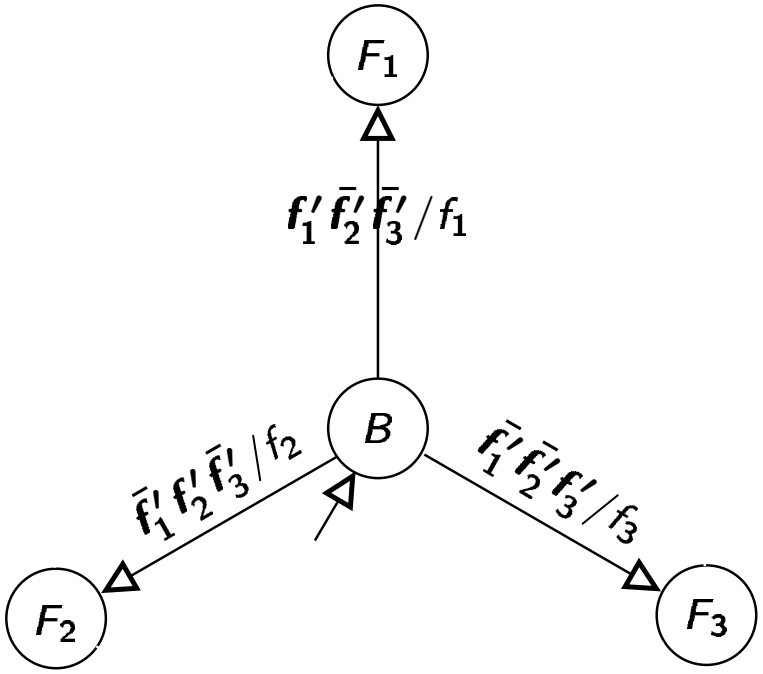
Processor failures I



Permanent failures / Transient failures / Degraded transient mode

The events f and d are uncontrollable / For r it depends

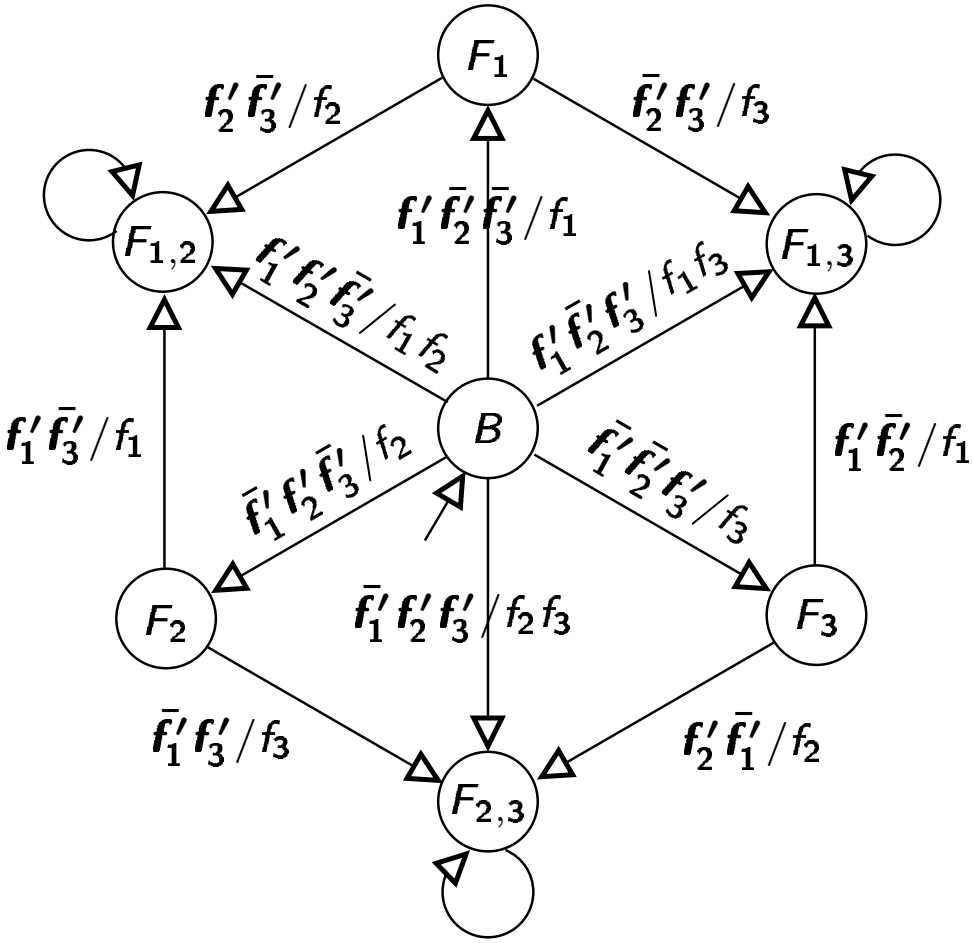
Processor failures II



One failure at a time

The events f_i are d_i are local / The events f_i' and d_i' are uncontrollable

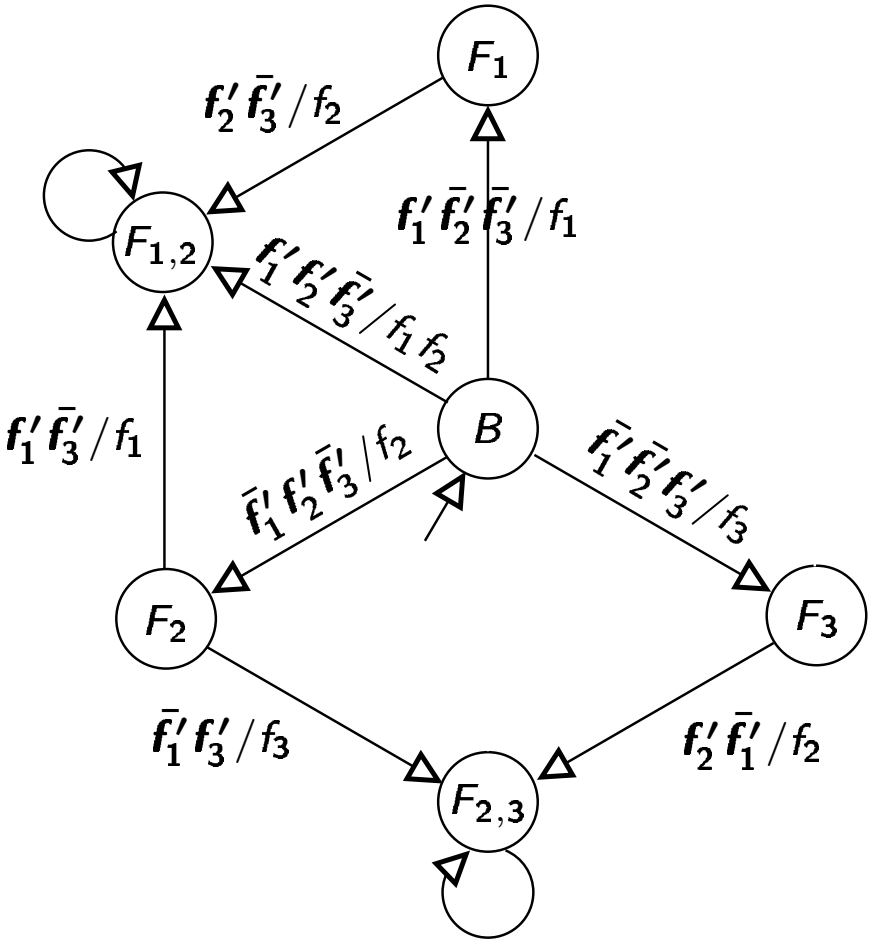
Processor failures II



At most two failures at a time (possibly simultaneous)

The events f_i are d_i are local / The events f_i' and d_i' are uncontrollable

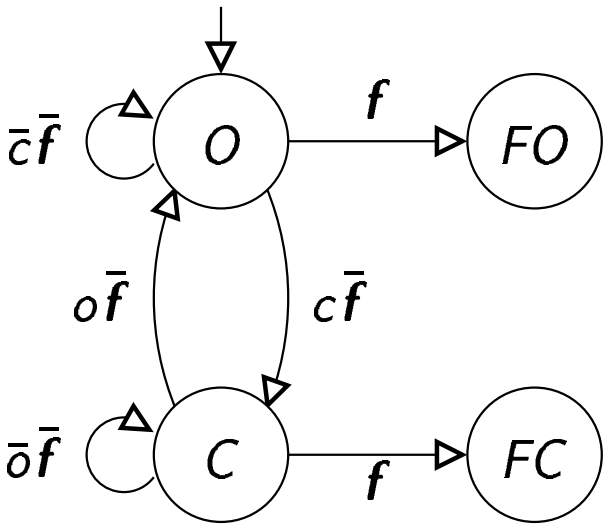
Processor failures II

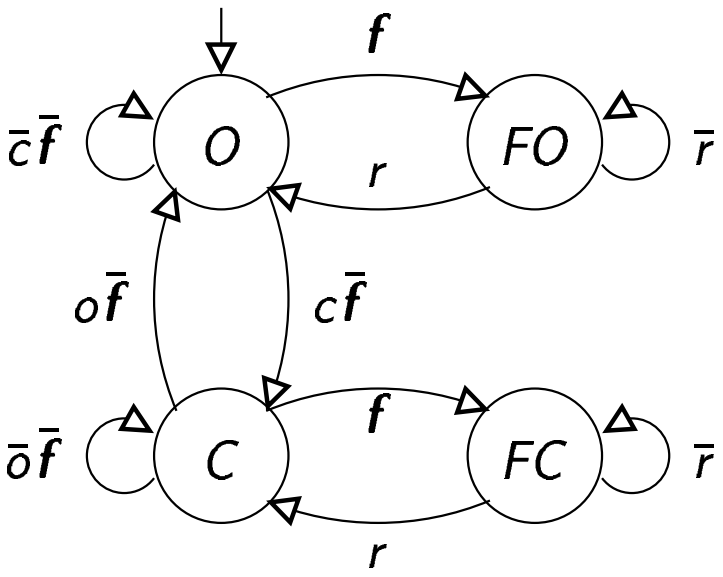


Arbitrary failure pattern

The events f_i are d_i are local / The events f_i' and d_i' are uncontrollable

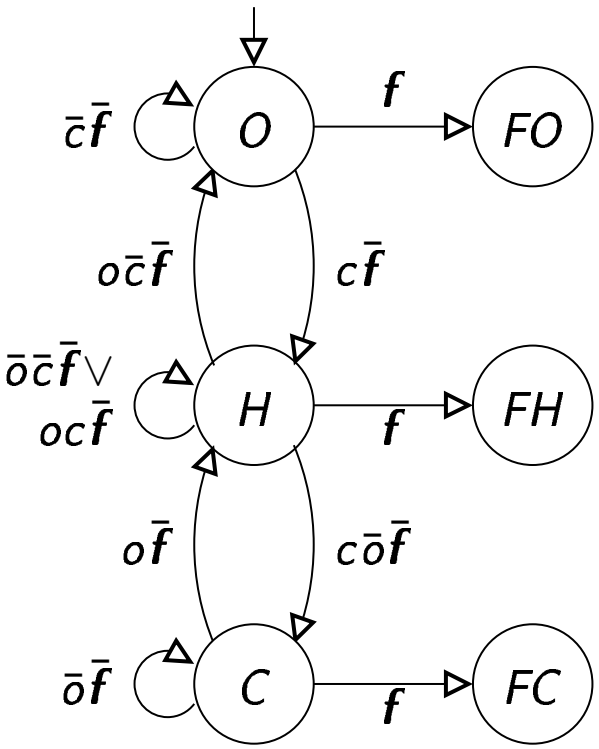
Actuator failures I

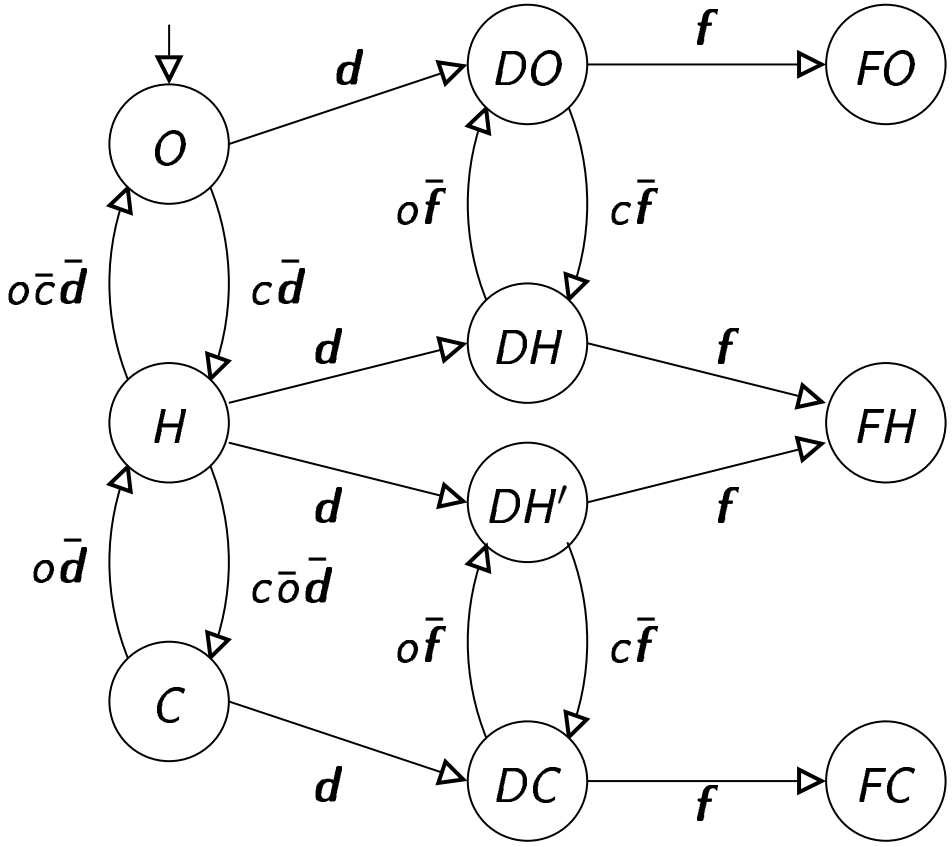




Valve with permanent failures / With transient failures

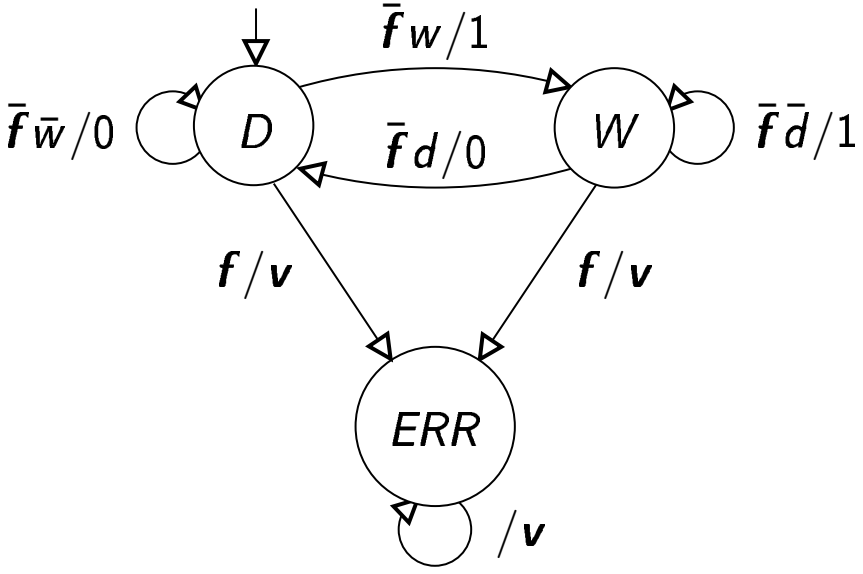
Actuator failures II

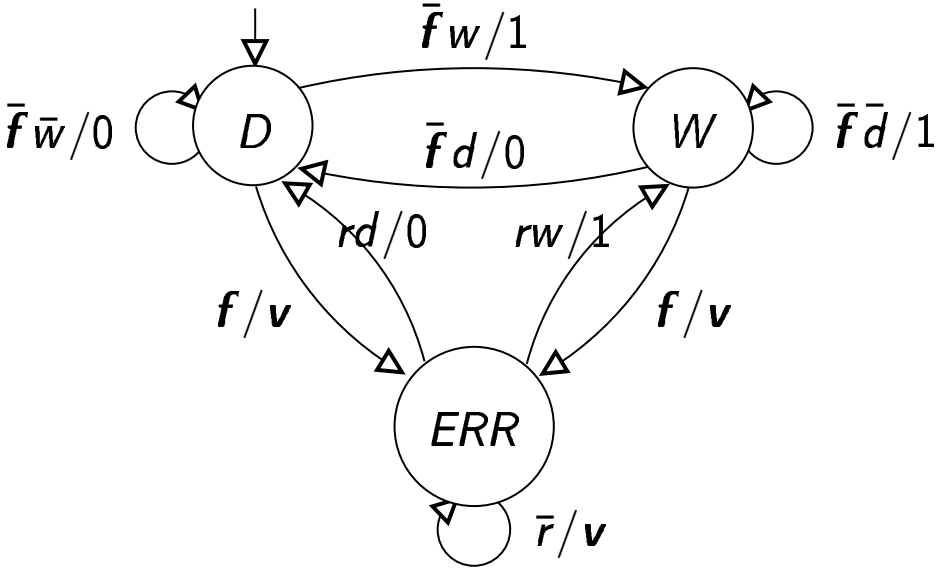




Three-state braking system with permanent failures / With degraded modes

Sensor failures





Liquid level sensor with permanent failures / With transient failures

Outline

Classification of failures

[Avizienis, Laprie & Randell, IFIP WCC'2004]

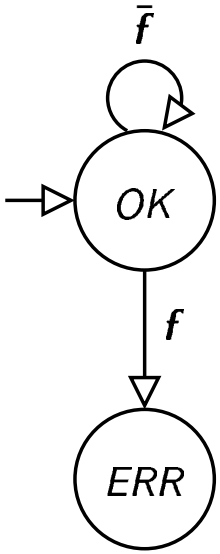
- Their domain : in value or temporal (in the latter case, their duration must also be specified)
- Their coherence w.r.t. all the users
- Their detectability by the user

➪ E.g., crash failures are permanent and coherent temporal failures

➪ E.g., Byzantine failures are incoherent value failures

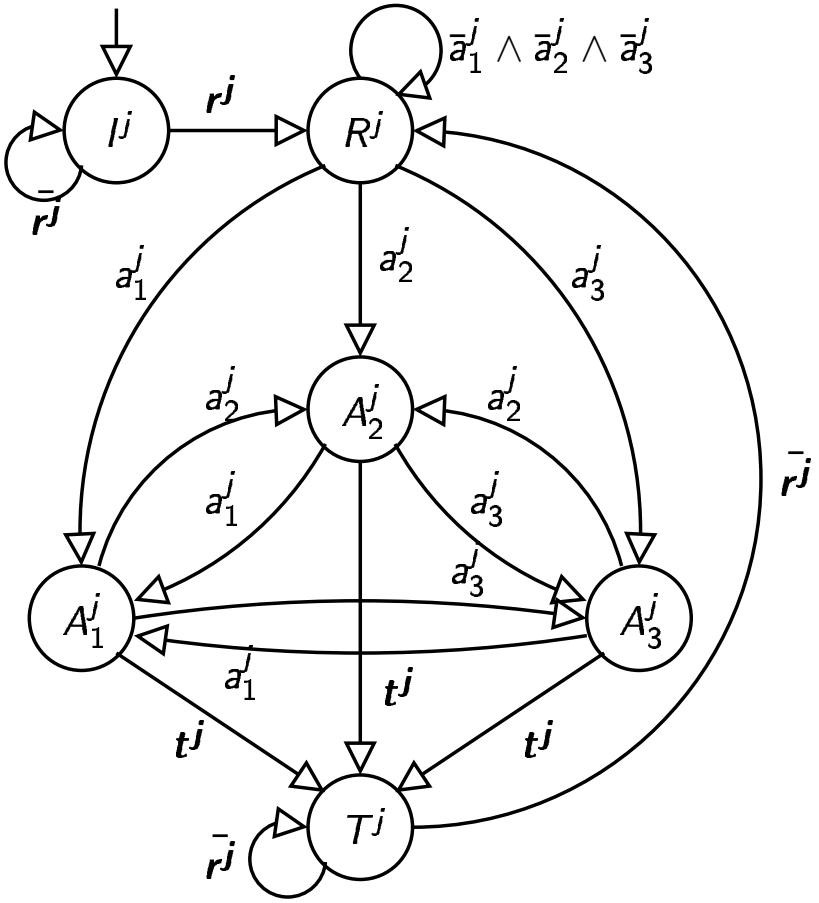
Crash failures

Processor model



||

Task model



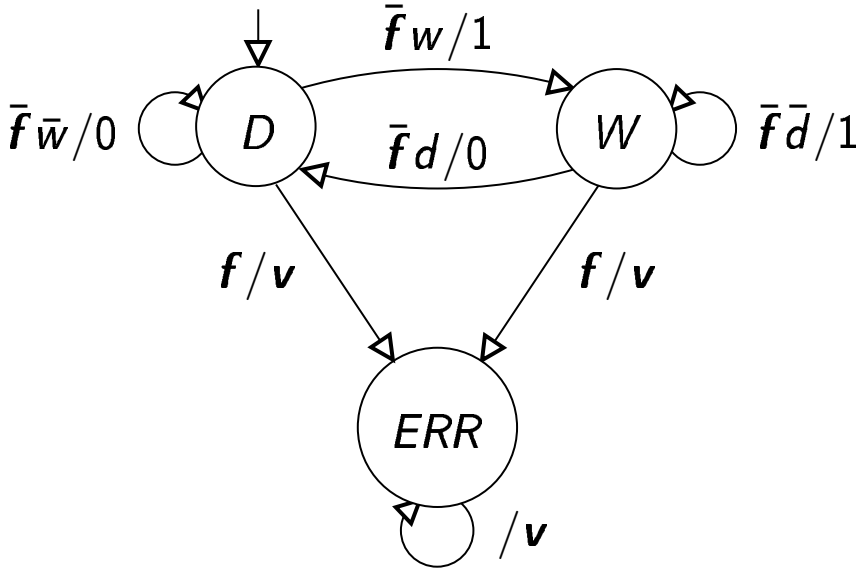
DCS objective

No task is active on a failed processor :

$$S' = mk\_invar\left(S, \neg \bigvee_{j=1}^{n} \bigvee_{i=1}^{p} (A_i^j \wedge ERR_i)\right)$$

Value failures

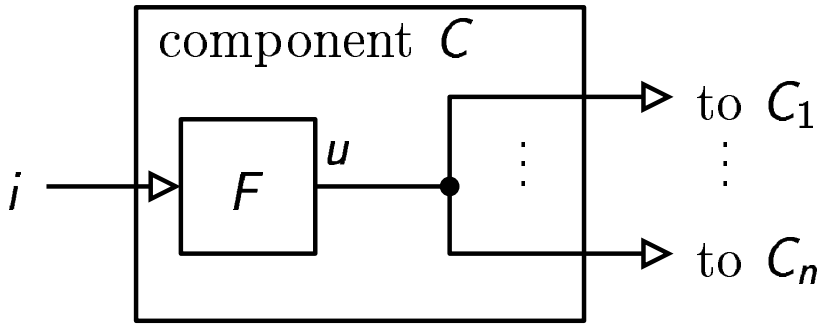
For a Boolean data : replace it by one uncontrollable input

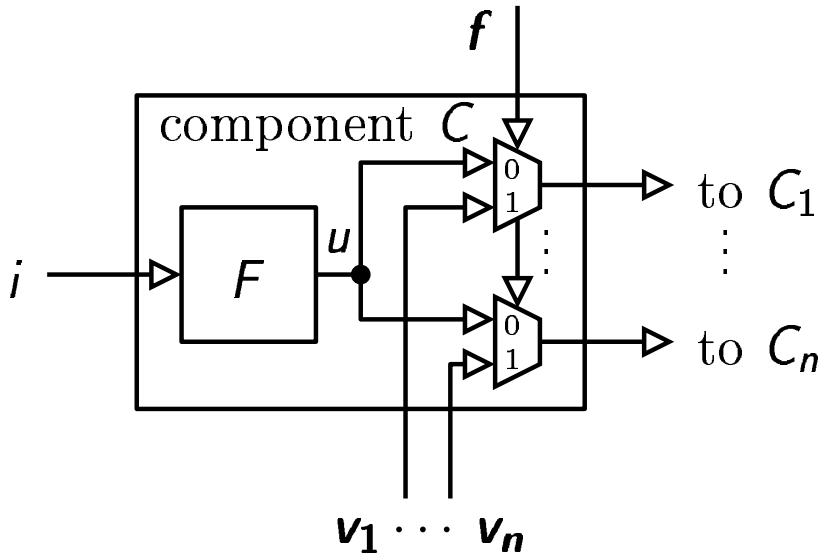


For a numerical data : discretize its domain and replace it by several uncontrollable inputs

➪ abstract interpretation

Byzantine failures





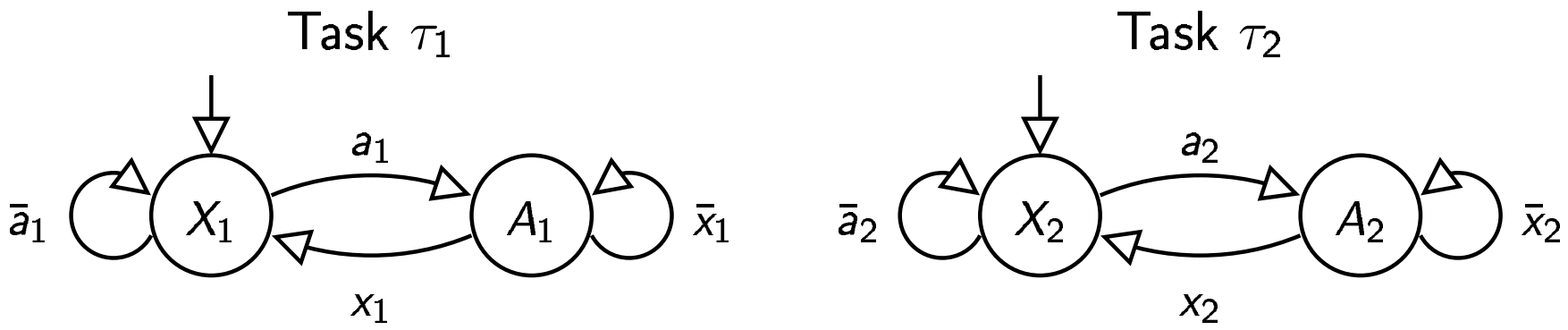
A basic non-faulty component / Same functionality with Byzantine failures

Byzantine failure behavior

When the component C is faulty, each one of its outputs is replaced by an uncontrollable input

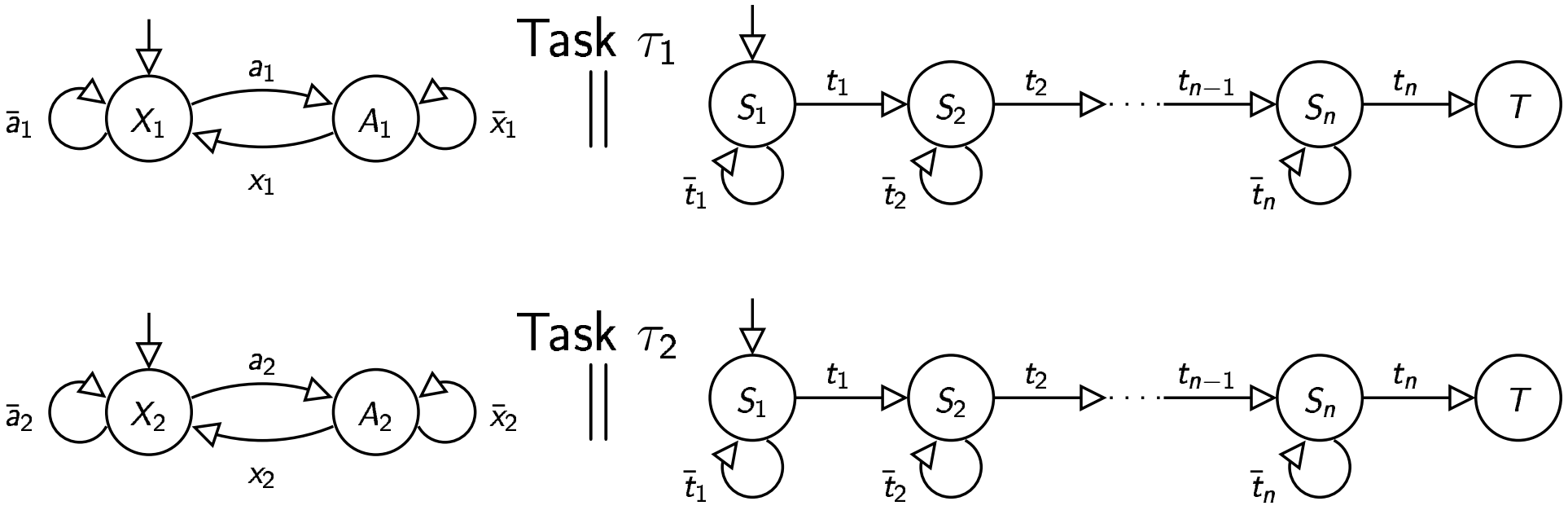
Outline

Shared resources



- Shared access : $true$
- Mutual exclusion : $S' = mk\_invar\,(S, \neg(A_1 \wedge A_2))$
- Continuous access : $S' = mk\_invar\,(S, A_1 \vee A_2)$

Critical section



State S_i of tasks τ_1 and τ_2 is a critical section

$$S' = mk\_invar\left(S, \neg\left(\left(A_1, S_{1,i}\right) \wedge \left(A_2, S_{2,i}\right)\right)\right)$$

Optimal discrete controller synthesis I

[Kumar & Garg, SIAM J. Cont. Opt., 1995] [Tronci, CDC'96]
[Sengupta & Lafortune, SIAM J. Cont. Opt., 1998]
[Marchand, Boivineau & Lafortune, J. Cont. Opt., 2000]

		power consumption C per processor			quality Q per processor		
		P_1	P_2	P_3	P_1	P_2	P_3
task	τ_1	4	4	2	3	5	3
	τ_2	2	2	3	2	2	5
	τ_3	2	3	4	2	2	5
	bound b_i	5	3	6			

Optimal discrete controller synthesis II

Basic fault tolerance DCS objective :

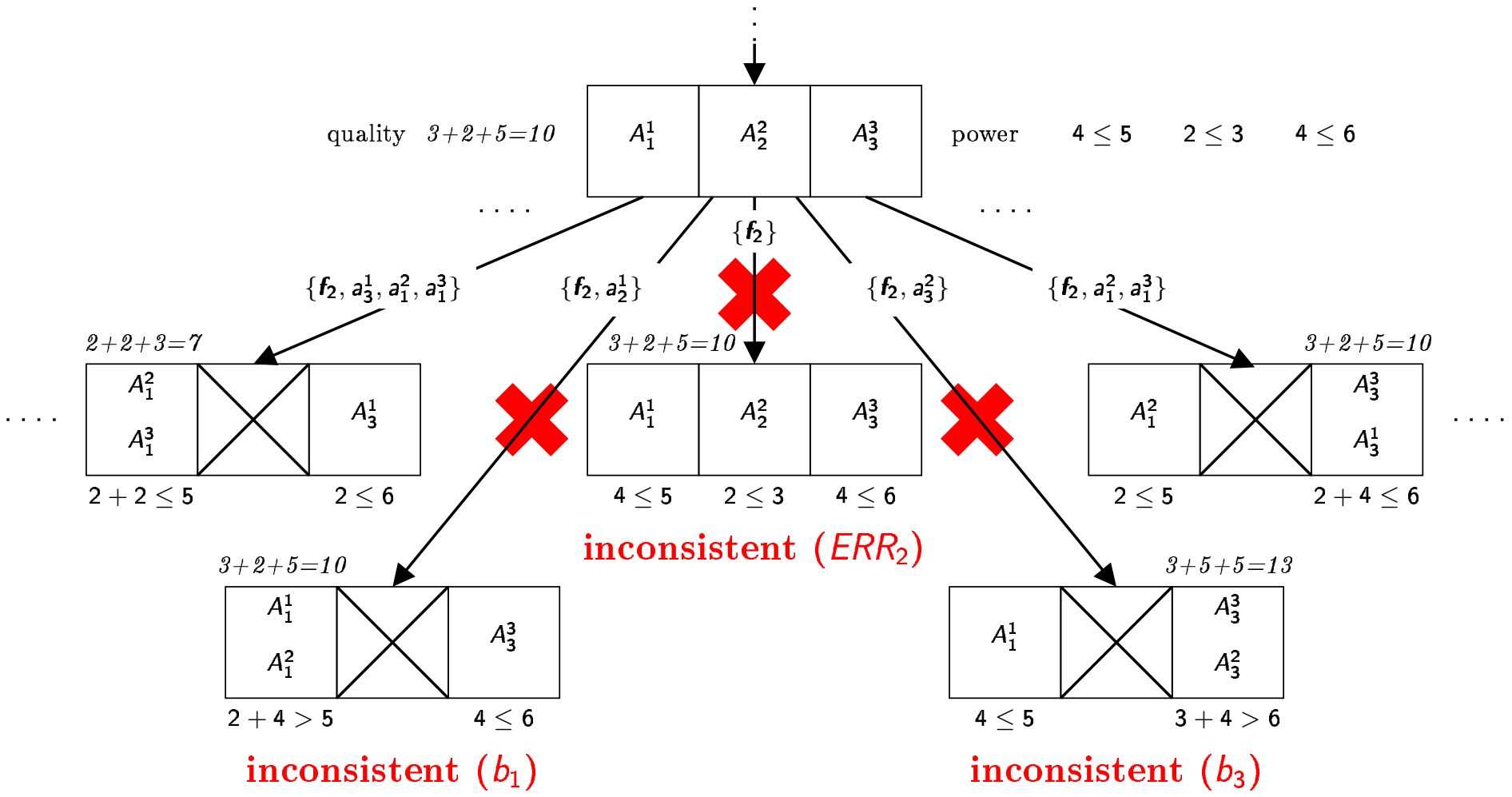
$$S' = mk\_invar\left(S, \neg \bigvee_{j=1}^{3} \bigvee_{i=1}^{3} (A_i^j \wedge ERR_i)\right)$$

Optimal DCS objectives :

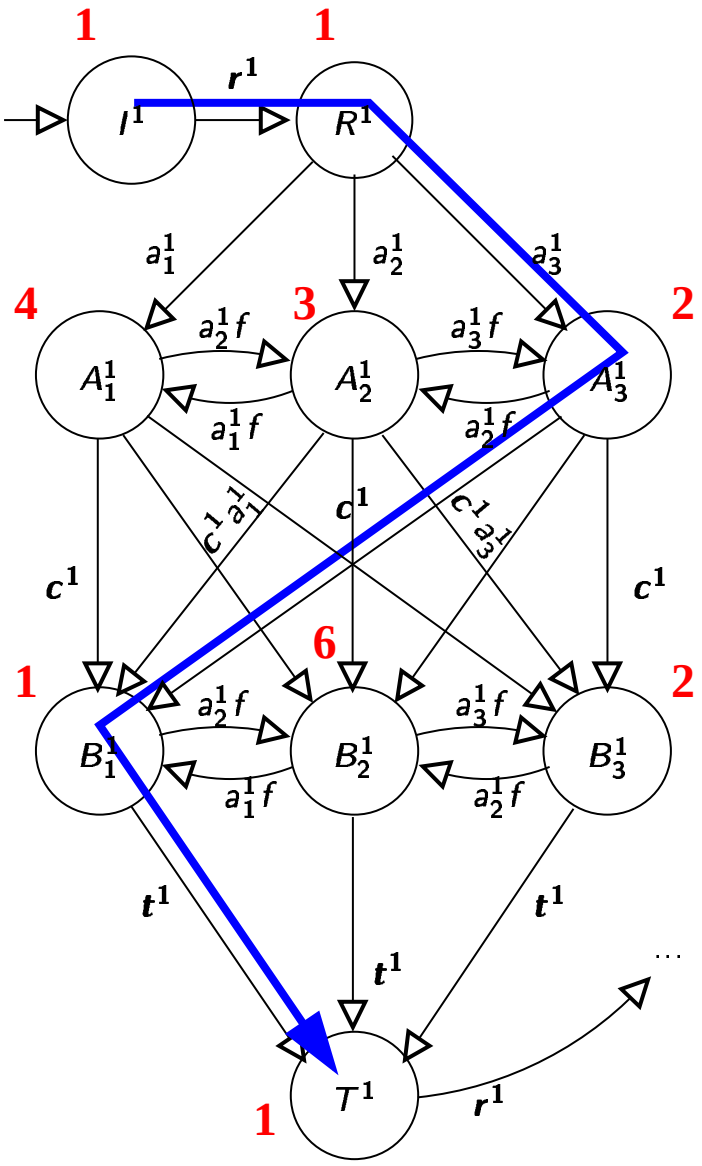
$$S'' = mk\_invar\left(S', \forall 1 \leq i \leq 3, \sum_{j=1}^{3} C_i^j \leq b_i\right)$$

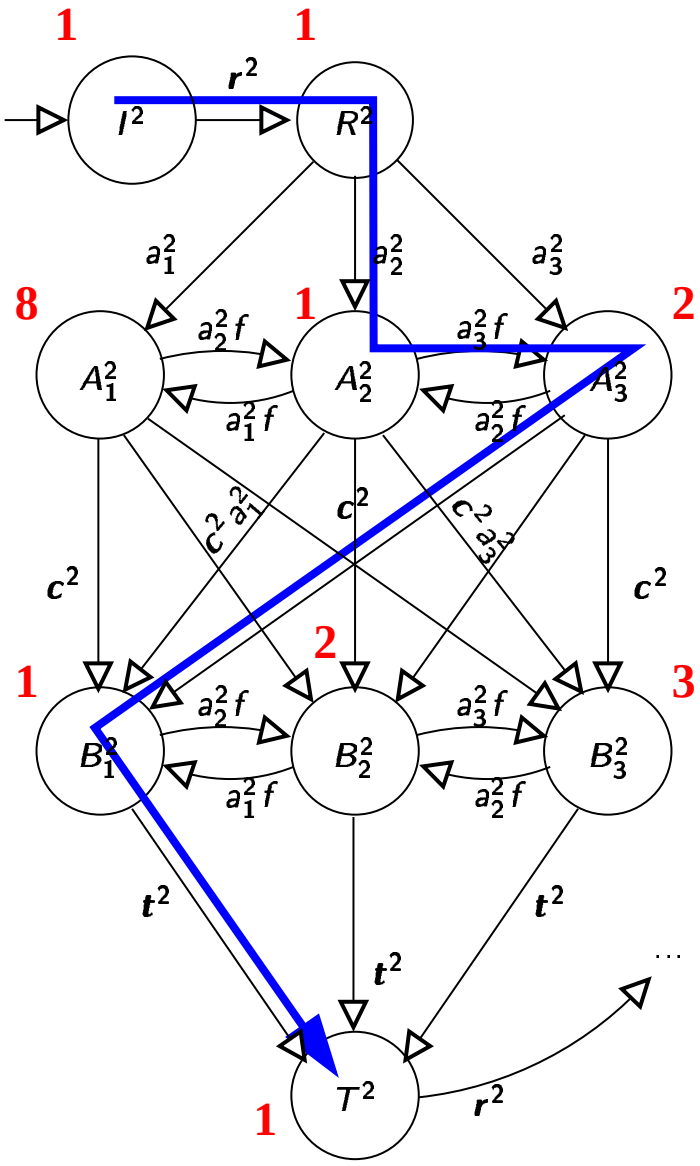
$$S''' = maximize\_step\left(S'', Q = \sum_{i=1}^{3} \sum_{j=1}^{3} Q_i^j\right)$$

Optimal discrete controller synthesis III



Optimal DCS on paths





Conditioned discrete controller objectives

In order to model degraded modes :

- Mode i : Condition $cond_i$ / DCS objective dcs_obj_i

Conditioned objective

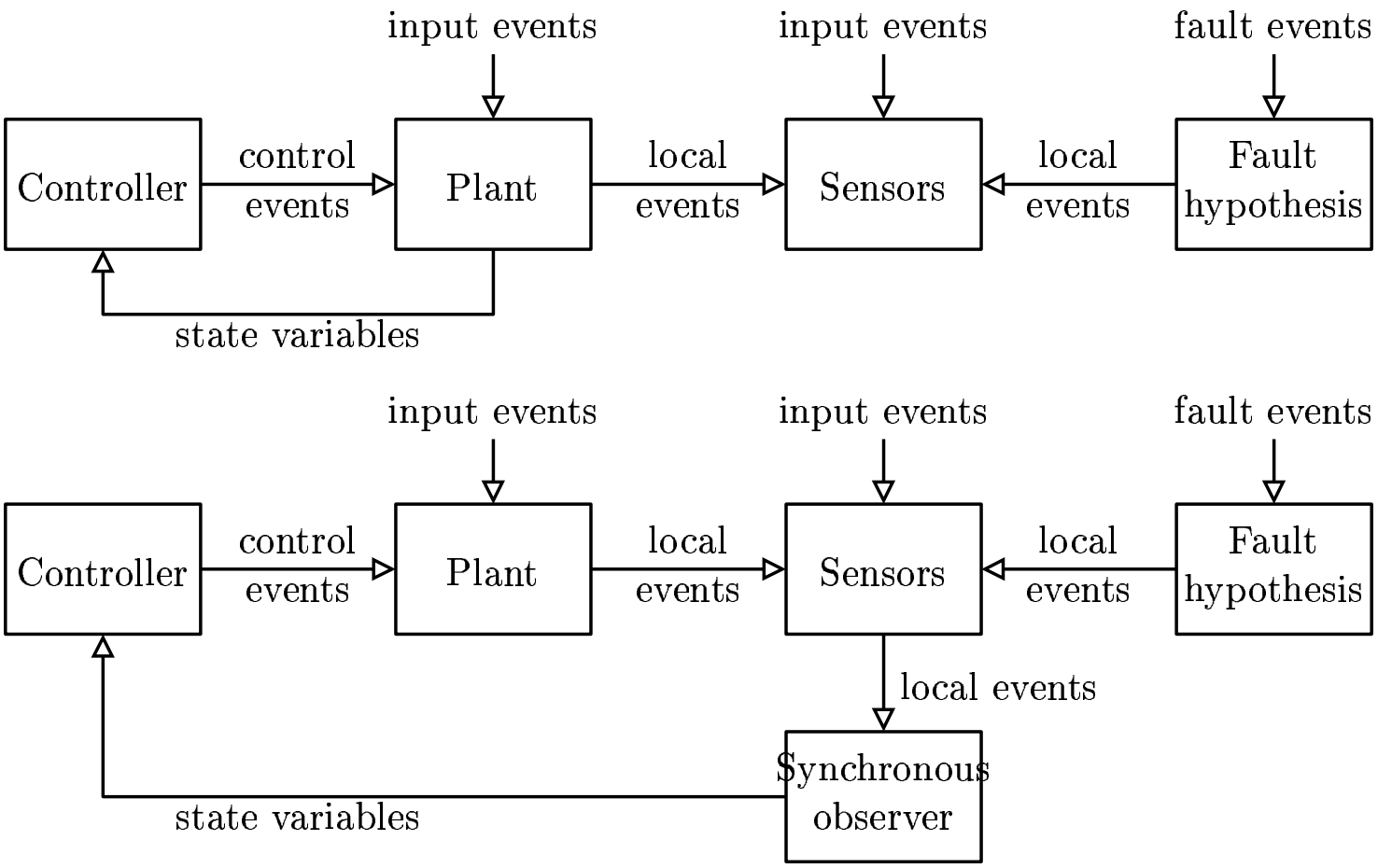
$$\bigwedge_{i=1}^{n} (cond_i \Rightarrow dcs\_obj_i)$$

Example

$$\begin{aligned} & (no\ failure \Rightarrow nominal\_dcs\_obj) \\ \wedge\ & (one\ failure \Rightarrow dcs\_obj_1) \\ \wedge\ & (two\ failures \Rightarrow dcs\_obj_2) \end{aligned}$$

Synchronous observers

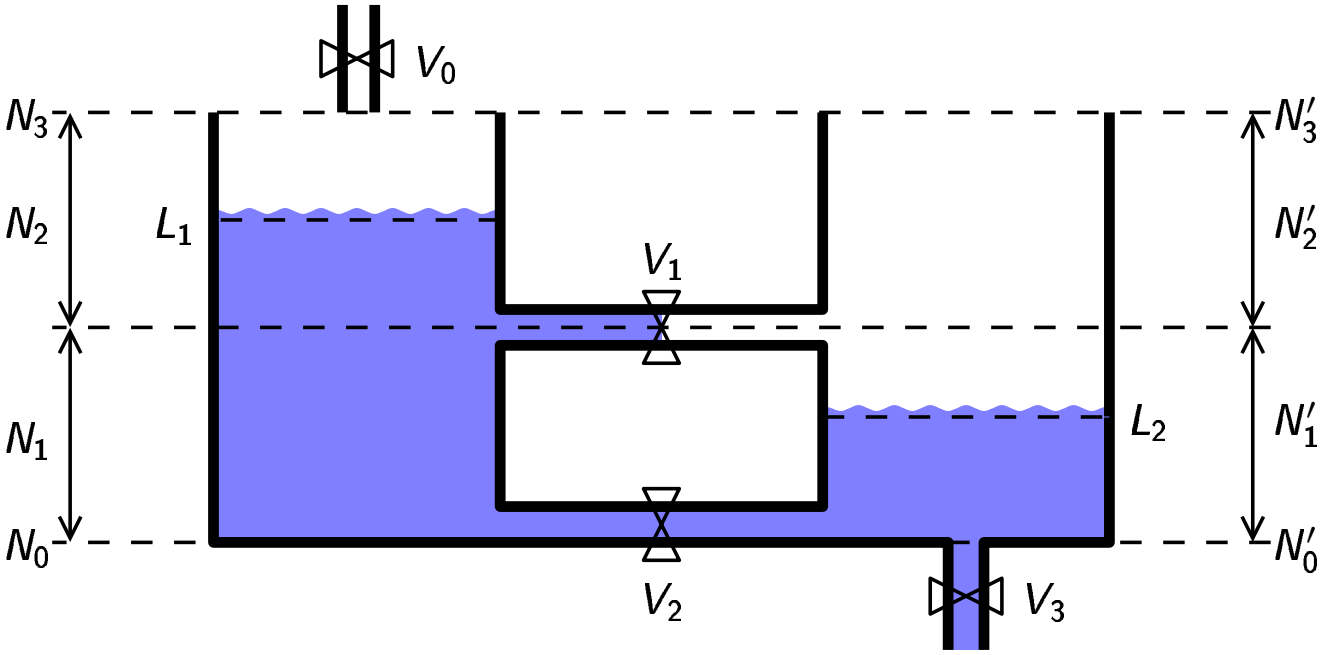
Observability in DCS : [Lin & Wonham, Inf. Sci., 1988]
[Cieslak, Desclaux, Fawaz & Varaiya, IEEE TAC, 1988]



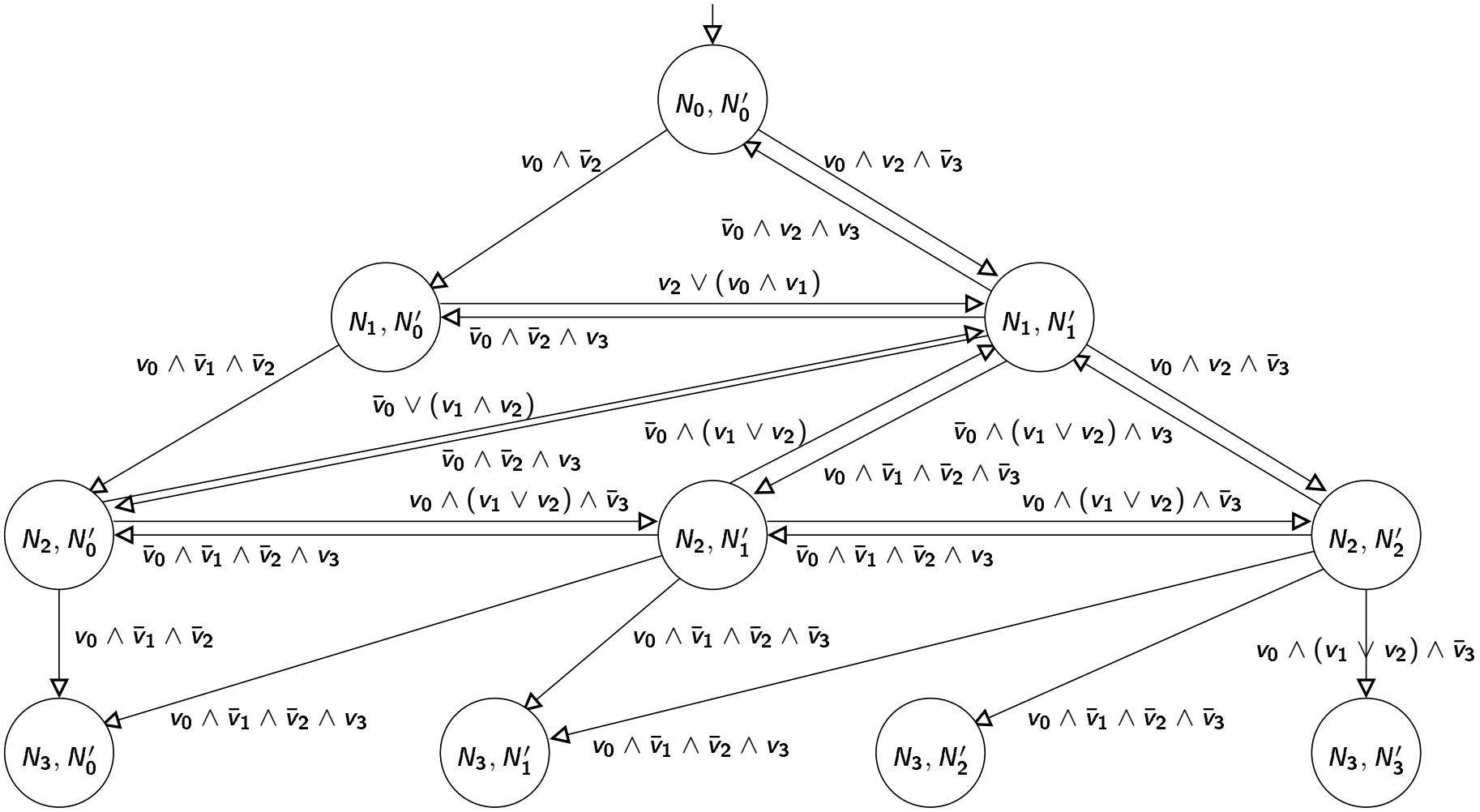
Outline

Case study I : two tanks

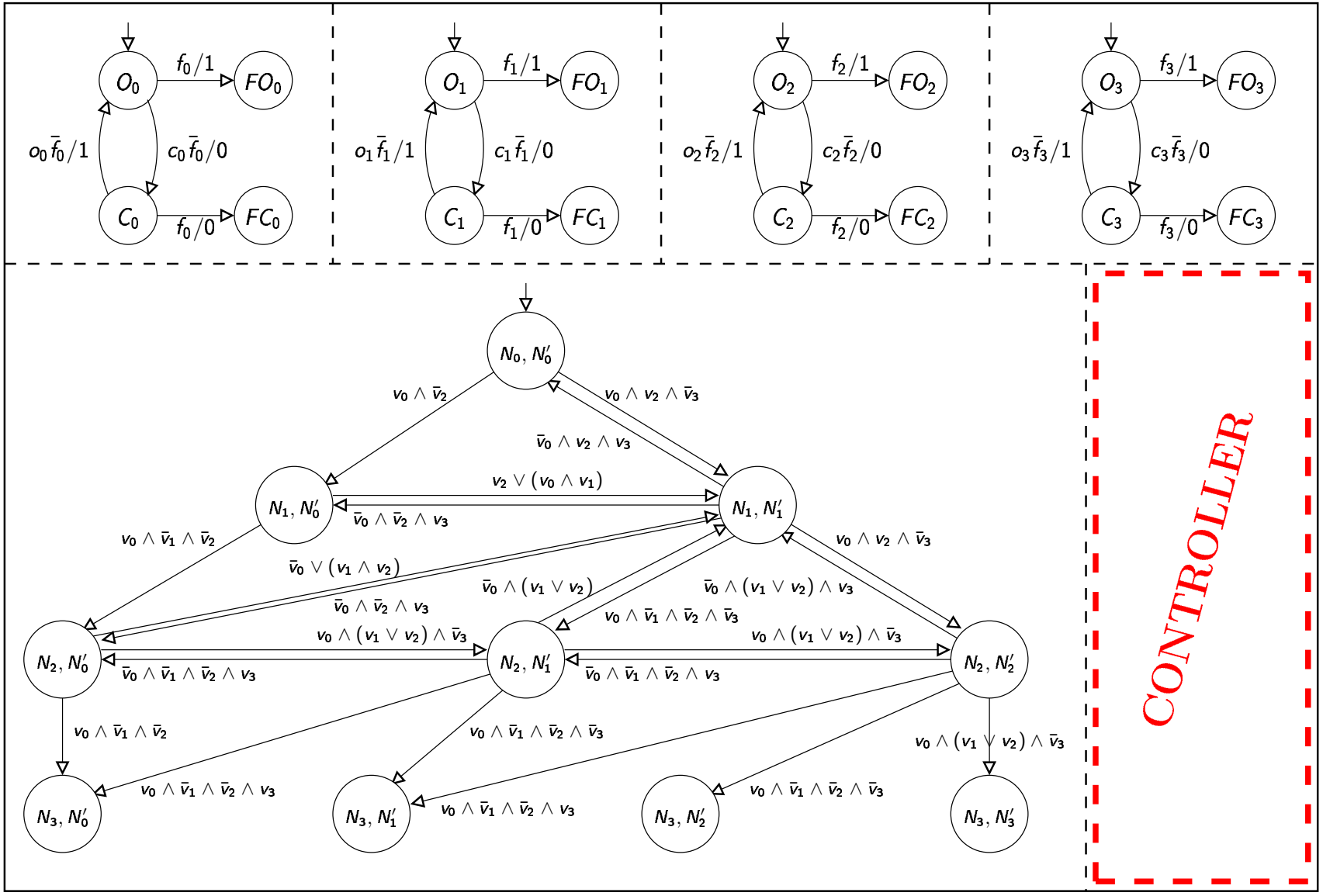
Two tanks, two pipes, and four valves
(four actuators : V_0, V_1, V_2, and V_3) :



LTS of the system



Complete system specification



Conditioned DCS objectives

1. As long as the valve V_0 is not stuck in the faulty and open state, neither of the two tanks should ever over-flood :

$$S' = mk\_invar\left(\neg FO_0 \Longrightarrow \neg \bigvee_{i=0}^{3}(N_3, N'_i), S\right)$$

2. If the four valves work fine, then the level in the left and right tanks should be regulated respectively at N_2 and N'_1 :

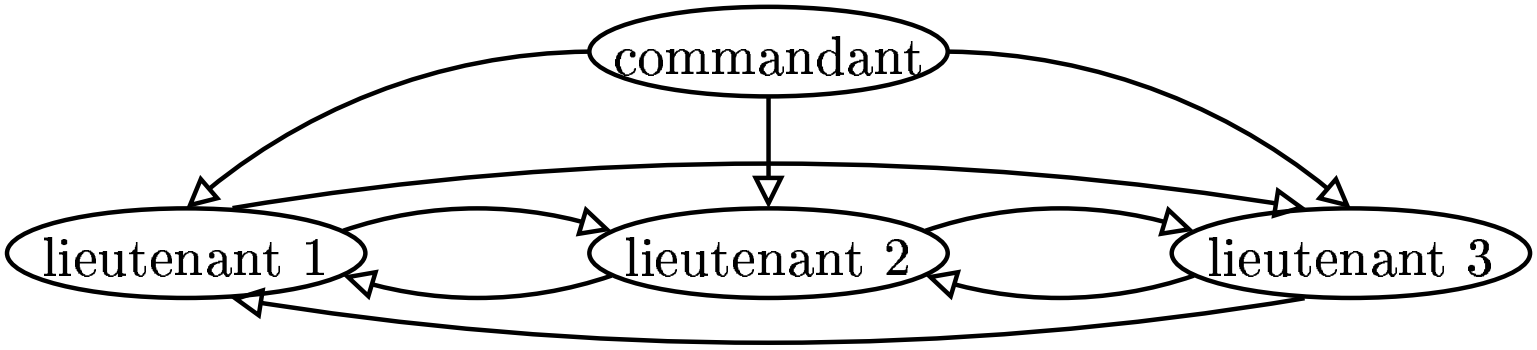
$$S' = \bigwedge_{i=0}^{3}(O_i \vee C_i) \Longrightarrow reach\_under\_control\left(\{(N_2, N'_1)\}, S\right)$$

Case study II : the Byzantine generals

[Lamport, Shostak & Pease, ACM TOPLAS, 1982]

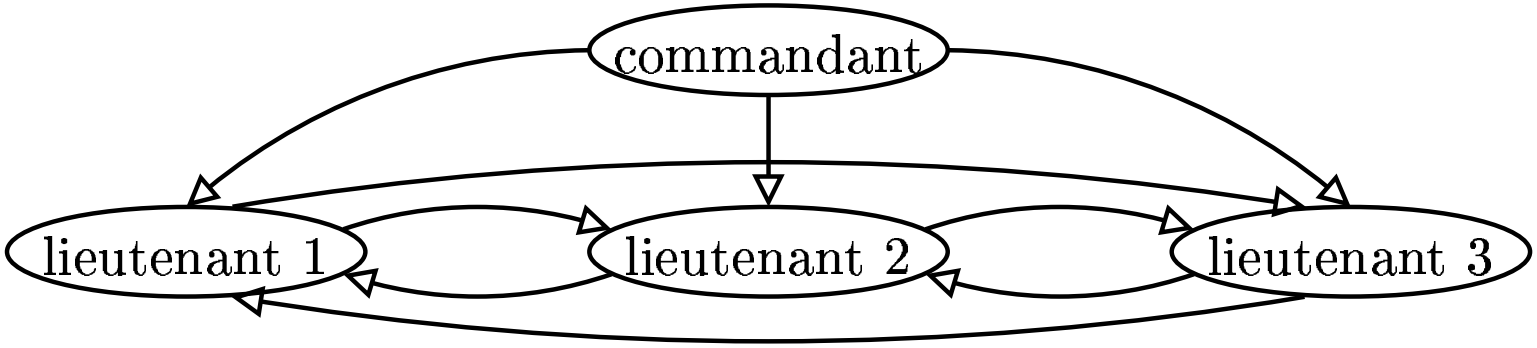
- n divisions of the Byzantine army, each commanded by its own general, are camped outside an enemy city
- One of the generals is the commander of the army, while the $n-1$ remaining ones are his lieutenants
- The generals must decide on a common plan of action, either attack or retreat
- Only by communicating with one another only by oral messages
- The problem is that some generals are traitors

The Byzantine generals



- The commander first sends an order (attack or retreat) to his $n-1$ lieutenants :
  - If he is loyal, then he must send the same order to all his lieutenants
  - But if he is a traitor, then he can send different orders to his lieutenants, that is, incoherent orders
- Each lieutenant transmits the received order to all the other lieutenants (similar behavior as the commander)
- The loyal generals will reach a consensus for their plan of action

Conditions that must be satisfied



- IC1 : All the loyal lieutenants obey the same order
- IC2 : If the commander is loyal, then each loyal lieutenant obeys the order sent to him

Solution of Lamport et al

Algorithm 1 Byzantine Commander (m, v)

```
Send my order v to the n − 1 lieutenants ;
```

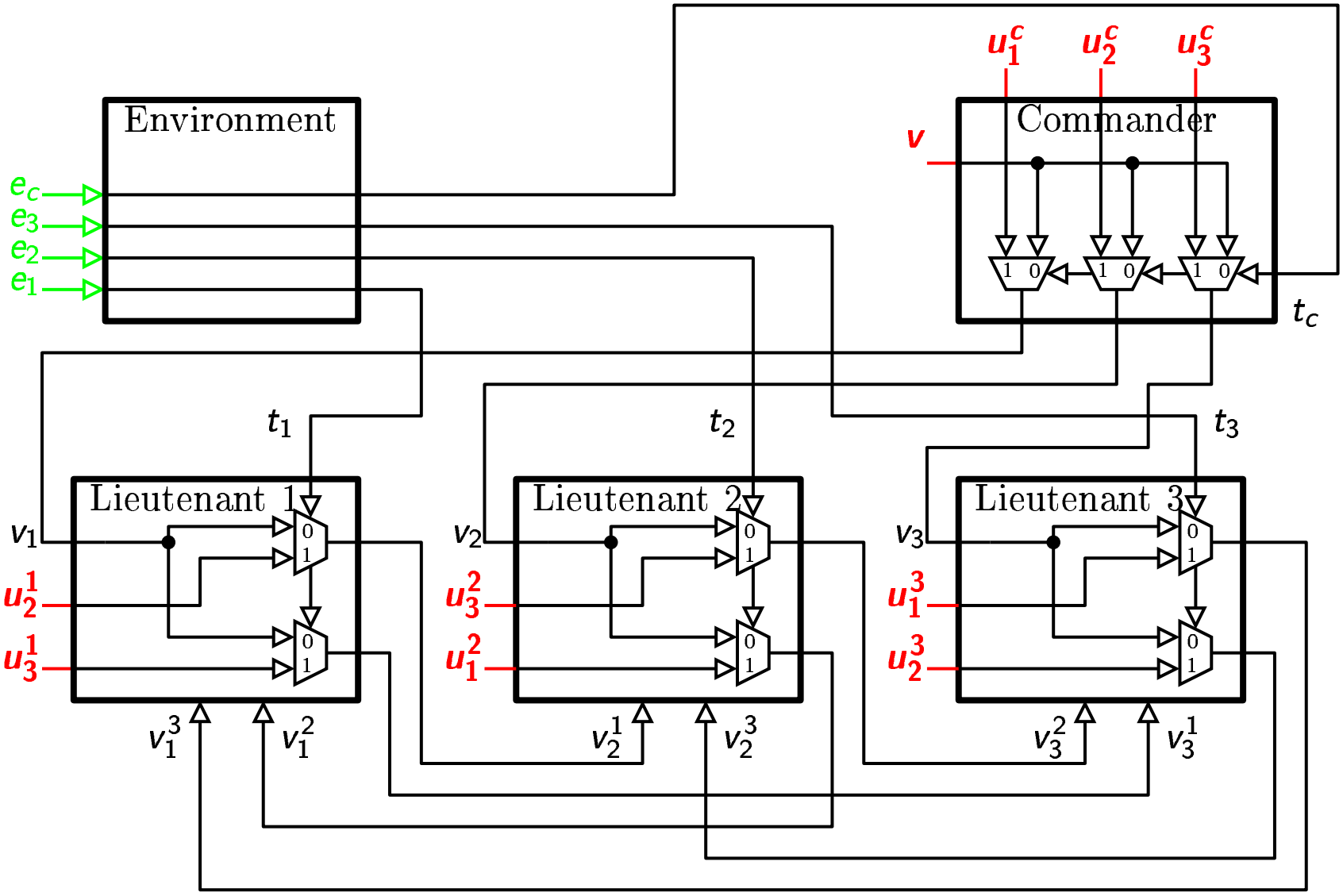
Algorithm 2 Byzantine Lieutenant (m, i)

```
v_i := value received from the commander ;
if m = 0 then
    Use as order the value v_i ;
else
    Send v_i to the n − 2 other lieutenants ;
    forall j ≠ i do
        v_j := value received from the lieutenant j ;
    end do
    Use as order the majority maj (v_1, v_2, ..., v_{n−1}) ;
end if
```

Lamport et al theorem

If there are m traitors, then there must be at least $3m + 1$ generals

Fault tolerant Byzantine generals



DCS objectives

- Property IC1 = unreachability of the states such that the predicate $\forall i \neq j, Loy_i \wedge Loy_j \wedge \big((Att_i \wedge Retr_j) \vee (Retr_i \wedge Att_j)\big)$ is true :

$$S' = mk\_invar\ \big(S, \forall i \neq j, Loy_i \wedge Loy_j \wedge \big((Att_i \wedge Retr_j) \vee (Retr_i \wedge Att_j)\big) = false\big)$$

- Property IC2 = unreachability of the states such that the predicate $\forall i, Loy_c \wedge Loy_i \wedge \big((Att_c \wedge Retr_i) \vee (Retr_c \wedge Att_i)\big)$ is true :

$$S' = mk\_invar\ \big(S, \forall i, Loy_c \wedge Loy_i \wedge \big((Att_c \wedge Retr_i) \vee (Retr_c \wedge Att_i)\big) = false\big)$$

Outcome of DCS

Most permissive environment model :

$$\left\langle Loy_c \xrightarrow{e_c/t_c} Tra_c \right\rangle || \left\langle Loy_1 \xrightarrow{e_1/t_1} Tra_1 \right\rangle || \left\langle Loy_2 \xrightarrow{e_2/t_2} Tra_2 \right\rangle || \left\langle Loy_3 \xrightarrow{e_3/t_3} Tra_3 \right\rangle$$

DCS restricts the environment model by allowing only one general be a traitor

It proves the theorem of Lamport et al in the particular case of $m = 1$

Proving the theorem in the general case would require parametric DCS

Outline

Related work

Fault-tolerant supervisors : [Cho & Lim, IEEE TRA, 1998]

Power transformer station controller : [Marchand & Samaan, IEEE TSE, 2000]

Automating the addition of fault-tolerance : [Kulkarni & Arora, FTRTFT'00]

Synthesis of fault-tolerant concurrent programs : [Attie, Arora & Emerson, ACM TOPLAS, 2004]

Supervisor for operating mode systems : [Kamach, Pietrac & Niel, IFAC WG'04]

Synthesis of quasi-static schedulings : [Cortadella et al, IEEE TCAD, 2005]

Fault-tolerant planning : [Jensen, Veloso & Bryant, WPUII'03]

Future work : distributed synthesis

Several complex issues

- Decentralized supervisory control [Lin & Wonham, IS'88] [Cieslak, Desclaux, Fawaz & Varaiya, IEEE TAC'88]
- How to make the controller tolerant to the failures of its processor ?
- Undecidable without communication between the local controllers [Tsitsiklis, MCSS'89] [Tripakis, IEEE TAC'04]

Future work : distributed synthesis

Several complex issues

- Decentralized supervisory control [Lin & Wonham, IS'88] [Cieslak, Desclaux, Fawaz & Varaiya, IEEE TAC'88]
- How to make the controller tolerant to the failures of its processor ?
- Undecidable without communication between the local controllers [Tsitsiklis, MCSS'89] [Tripakis, IEEE TAC'04]

Our proposal

- To distribute the controller afterwards
- Automatic distribution of LTSs [Caspi, Girault & Pilaud, IEEE TSE'99]
- Classical FT techniques to tolerate the failures of the processors and the communication links
- Lots of future work !

Conclusions : our contributions

- Framework for automating the addition of fault tolerance based on DCS
- All kinds of HW components : processors, communication links, actuators, and sensors
- All kinds of failures : crash, value, Byzantine
- Model of the plant : synchronous product of LTSs
- DCS objective : reachability and invariance properties
- Advanced DCS features : optimal DCS, conditioned objectives, synchronous observers

Conclusions : advantages of our framework

- Our method is fully automatic
- Separation of concerns
- It is flexible : it is easy to try several fault tolerant objectives
- The controlled system is fault tolerant by construction
- The failure recovery mechanism is dynamic (hence it does not induce too much redundancy overhead like static methods)
- With a static guarantee on the fault tolerance of the obtained system (unlike dynamic methods).