Supervisory Control of Infinite State Systems under Partial Observation

B. Jeannet, G. Kalyon, T. Le Gall, H. Marchand, T. Massart

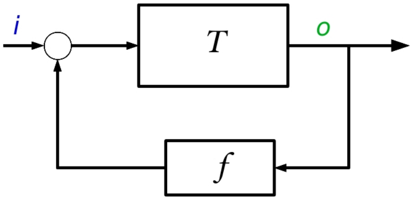
Séminaire CIRM - LMeASI

6 Décembre 2010

Outline

Outline

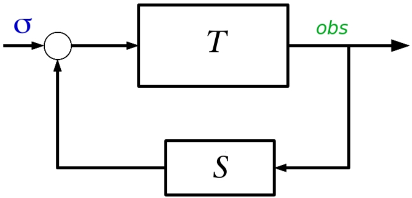
Once upon a Time in Supervisory Control



Automation and Control Theory

- Feedback loop to control an electronic device $\mathcal{T}$
- An input signal i, a control function f, a specification for the output o
- Question : what should be f so that the output signal matches the specification ?

Once upon a Time in Supervisory Control (2)



Computer science Theory [Ramadge & Wonham 86]

- Discrete event system $\mathcal{T}$
- The supervisor $\mathcal{S}$ observes $\mathcal{T}$ and may disable some events
- Question : what should be $\mathcal{S}$ so that $\mathcal{T}||\mathcal{S}$ satisfies the specification ?

Formal Definition of the Control Problem

Formalism : Regular languages

- System : $\mathcal{T} = \langle Q, \Sigma, q_0, Q_f, \rightarrow \rangle$, language M
- Partition : $\Sigma = \Sigma_c \cup \Sigma_{uc}$
- Specification : langage K

Formal Definition of the Control Problem

Formalism : Regular languages

- System : $\mathcal{T} = \langle Q, \Sigma, q_0, Q_f, \rightarrow \rangle$, language M
- Partition : $\Sigma = \Sigma_c \cup \Sigma_{uc}$
- Specification : langage K

Solution (for prefix-closed languages)

- L is controllable with respect to M if :

$$L.\Sigma_{uc} \cap M \subseteq L$$

- The is a unique supremal controllable language $\widehat{L} \subseteq K$
- Supervision : forbid every events not enable by $\widehat{L}$

Why does he talk about that ?

What about static analysis ? Abstract interpretation ? I want my money back !

Principles of Abstract Interpretation

Static Analysis and Abstract Interpretation

- Static Analysis : method to know what a program can do without running it
- Relies on a fixpoint computation in a lattice (generally $2^{\mathcal{D}_V}$)
- Abstract Interpretation : method to solve this fixpoint computation using approximation

Principles of Abstract Interpretation

Static Analysis and Abstract Interpretation

- Static Analysis : method to know what a program can do without running it
- Relies on a fixpoint computation in a lattice (generally $2^{\mathcal{D}_V}$)
- Abstract Interpretation : method to solve this fixpoint computation using approximation

Outline of the method

- Concrete lattice $2^{\mathcal{D}_V} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$ Abstract lattice
- We transpose the computation into the abstract lattice
- A widening operator ∇ ensures the convergence of this computation
- The obtained solution is an overapproximation of the least fixpoint

From Supervisory Control to Abstract interpretation

Supremal controllable language = fixpoint computation

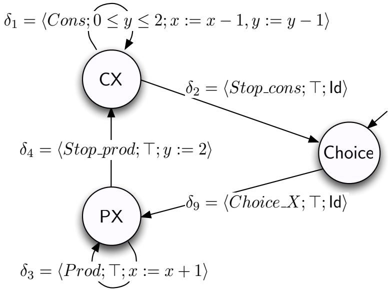
- Controllable = (post-) fixpoint
- State-based approach : if K is a set of "good" states, we want the greatest fixpoint of $X \rightarrow \mathrm{Post}_{uc}(X) \cap K$
- Supervision : disable all transitions leading out this set of states

What about Abstract Interpretation ?

- When reachability is undecidable, we may obtain a valid supervisor
- Over-approximation of a least fixpoint
- Safe control, but possible loss of permissivity

Outline

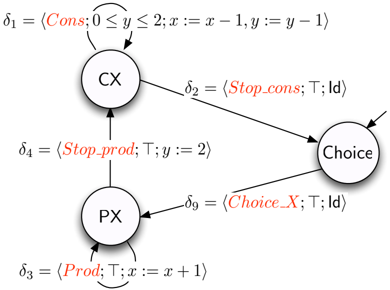
Model



Symbolic Transition System (STS)

- Numerical variables, domain $\mathcal{D}_V$
- Symbolic Transitions $\delta = \langle \sigma, G, A \rangle$
- Semantics : infinite Labelled Transition System (LTS)

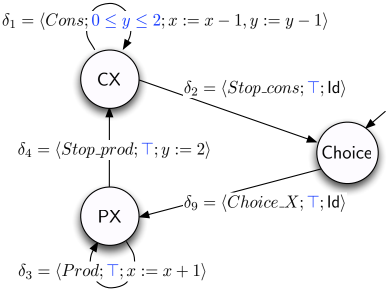
Model



Symbolic Transition System (STS)

- Numerical variables, domain $\mathcal{D}_V$
- Symbolic Transitions $\delta = \langle \sigma, G, A \rangle$
- Semantics : infinite Labelled Transition System (LTS)

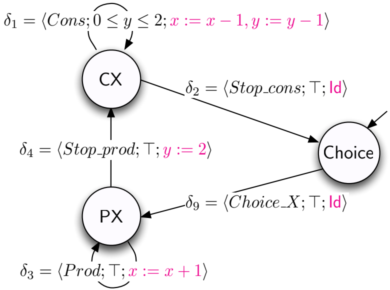
Model



Symbolic Transition System (STS)

- Numerical variables, domain $\mathcal{D}_V$
- Symbolic Transitions $\delta = \langle \sigma, G, A\rangle$
- Semantics : infinite Labelled Transition System (LTS)

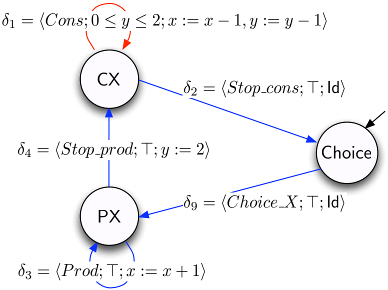
Model



Symbolic Transition System (STS)

- Numerical variables, domain $\mathcal{D}_V$
- Symbolic Transitions $\delta = \langle \sigma, G, A \rangle$
- Semantics : infinite Labelled Transition System (LTS)

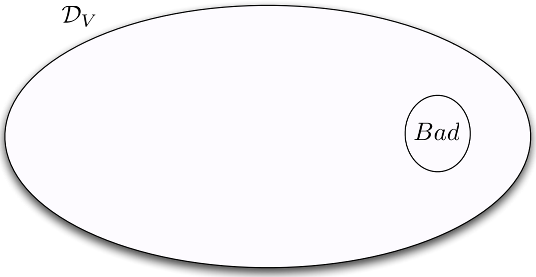
Model



Symbolic Transition System (STS)

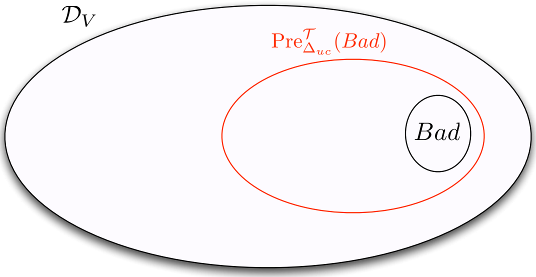
- Numerical variables, domain $\mathcal{D}_V$
- Symbolic Transitions $\delta = \langle \sigma, G, A \rangle$
- Semantics : infinite Labelled Transition System (LTS)
- Transitions $\Delta = \Delta_c \uplus \Delta_{uc}$
- Objective : avoid $Bad \subseteq \mathcal{D}_V$

Basic Problem



Fixpoint computation

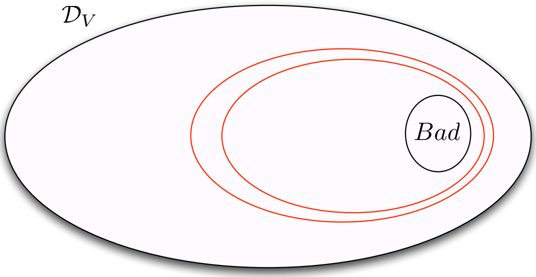
Basic Problem



Fixpoint computation

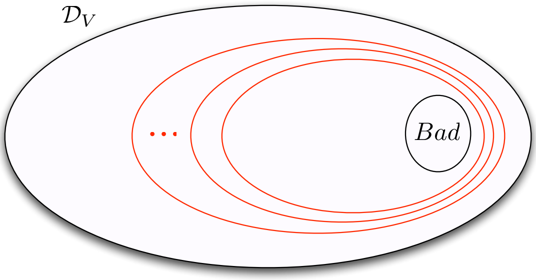
- $\mathrm{Pre}^{\mathcal{T}}_{\Delta_{uc}}(\mathit{Bad})$ = set of states leading to *Bad* through an uncontrollable transition

Basic Problem



Fixpoint computation

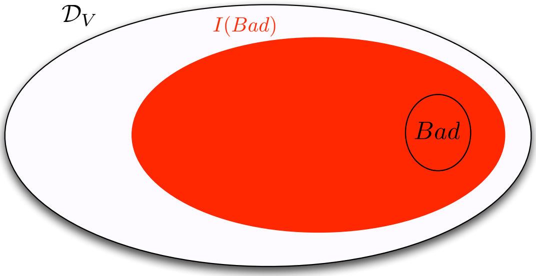
- $\text{Pre}^{\mathcal{T}}_{\Delta_{uc}}(\mathit{Bad})$ = set of states leading to *Bad* through an uncontrollable transition
- Iteration of $\text{Pre}^{\mathcal{T}}_{\Delta_{uc}}$



Fixpoint computation

- $\mathrm{Pre}^{\mathcal{T}}_{\Delta_{uc}}(Bad)$ = set of states leading to *Bad* through an uncontrollable transition
- Iterations of $\mathrm{Pre}^{\mathcal{T}}_{\Delta_{uc}}$

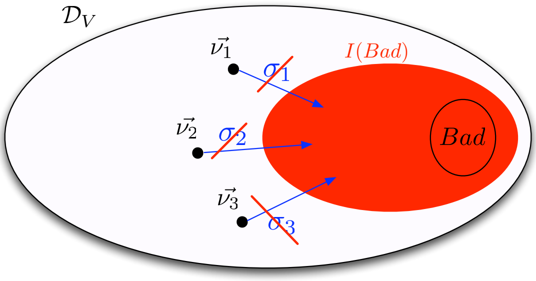
Basic Problem



Fixpoint computation

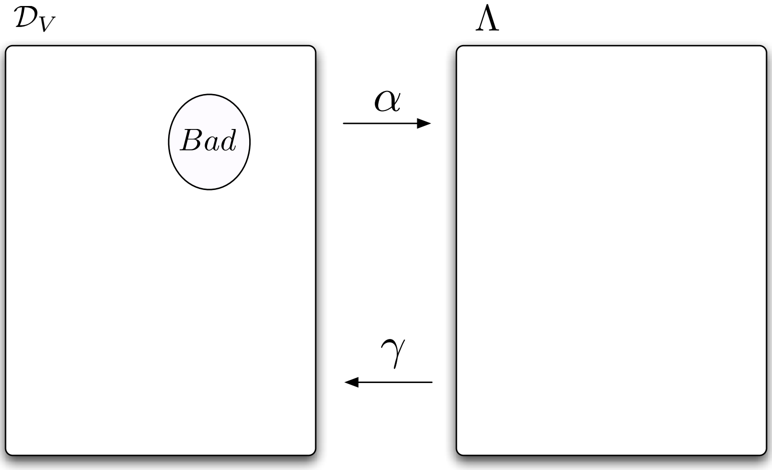
- $\text{Pre}^{\mathcal{T}}_{\Delta_{uc}}(Bad)$ = set of states leading to *Bad* through an uncontrollable transition
- Iterations of $\text{Pre}^{\mathcal{T}}_{\Delta_{uc}}$
- $I(Bad)$ = Set of states leading to *Bad* through several uncontrollable transitions

Basic Problem

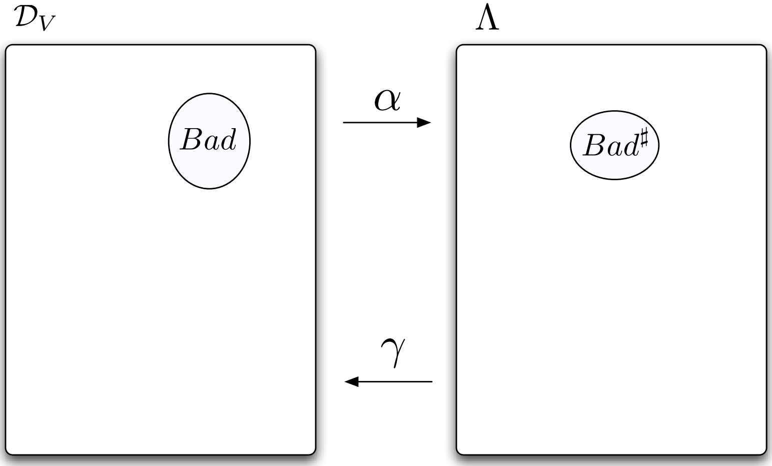


Fixpoint computation

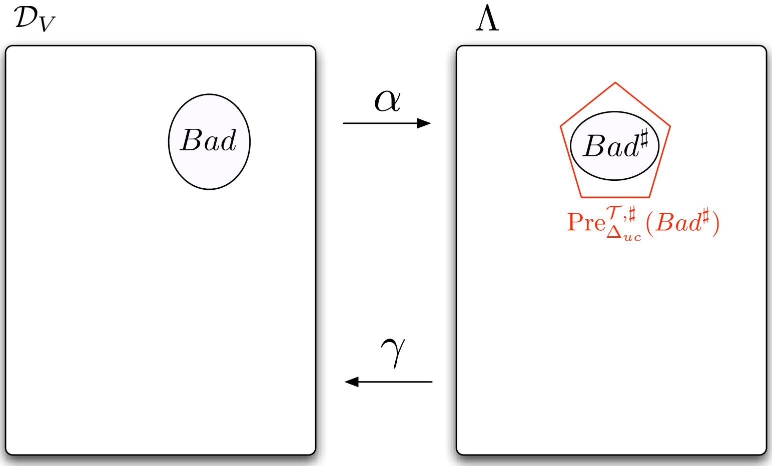
- $\text{Pre}^{\mathcal{T}}_{\Delta_{uc}}(Bad)$ = set of states leading to *Bad* through an uncontrollable transition
- Iterations of $\text{Pre}^{\mathcal{T}}_{\Delta_{uc}}$
- $I(Bad)$ = Set of states leading to *Bad* through several uncontrollable transitions
- Supervisor : forbid all transitions leading to $I(Bad)$

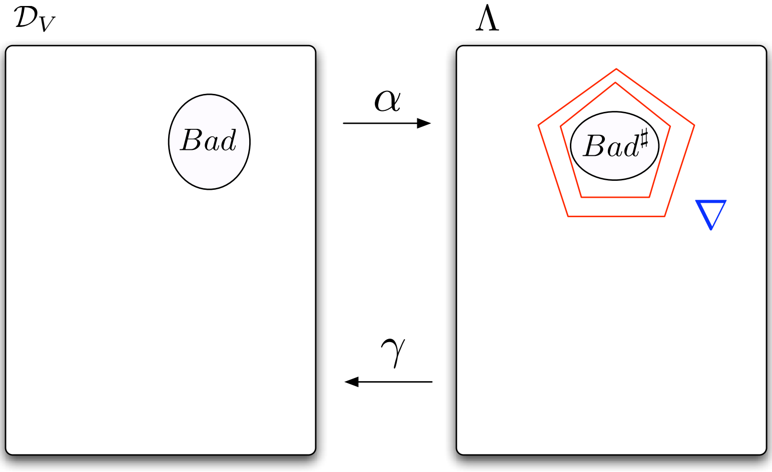

$\mathcal{D}_V$
Λ
Bad
α
γ

Effective computation

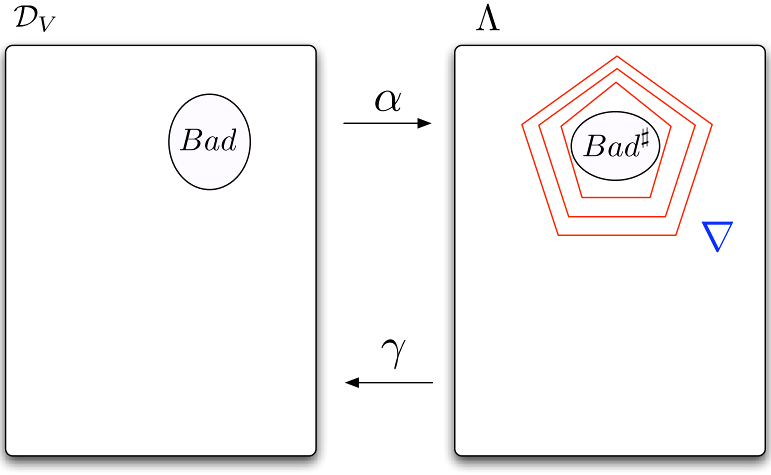


Effective computation

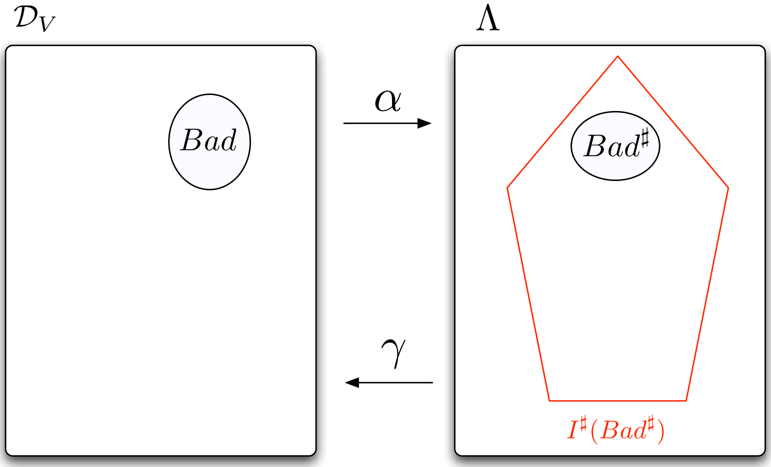


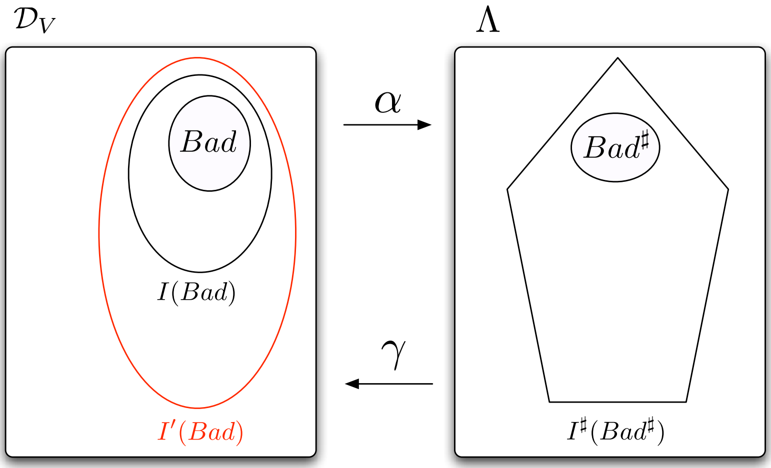

$\mathcal{D}_V$
Bad
α
γ
Λ
$Bad^\sharp$
∇

Effective computation

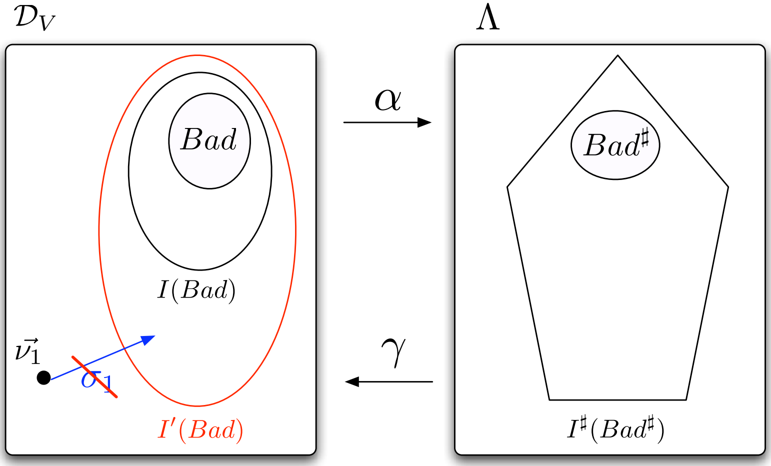


Effective computation




$\mathcal{D}_V$
Λ
Bad
$I(Bad)$
$I'(Bad)$
α
γ
$Bad^\sharp$
$I^\sharp(Bad^\sharp)$

Effective computation



Permissiveness and non-blocking control

Permissiveness

- Quality of control : how permissive the supervisor is
- More precise fixpoint computation $\Rightarrow$ better supervisor
- Choice of the abstract lattice, of the fixpoint computation strategy, ...

Permissiveness and non-blocking control

Permissiveness

- Quality of control : how permissive the supervisor is
- More precise fixpoint computation $\Rightarrow$ better supervisor
- Choice of the abstract lattice, of the fixpoint computation strategy, ...

Deadlock-free case

- Deadlock-free : at least one event is enable in every state
- Non-blocking : always able to reach a given objective

Permissiveness and non-blocking control

Permissiveness

- Quality of control : how permissive the supervisor is
- More precise fixpoint computation $\Rightarrow$ better supervisor
- Choice of the abstract lattice, of the fixpoint computation strategy, ...

Deadlock-free case

- Deadlock-free : at least one event is enable in every state
- Non-blocking : always able to reach a given objective
- We “solved” the deadlock-free case, underapproximations are needed so solve the non-blocking problem

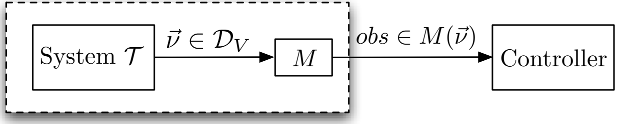
(Partial) Conclusion

Advantages of the approach

- For the "supervisory control" community :
  1. Rigorous fixpoint computation method
  2. Termination (enven when the problem is undecidable)
- For the "static analysis" community :
  1. A new playground !
  2. Some problems that do not occur in software verification (e.g. non-blocking)

Outline

Systems under Partial Observation



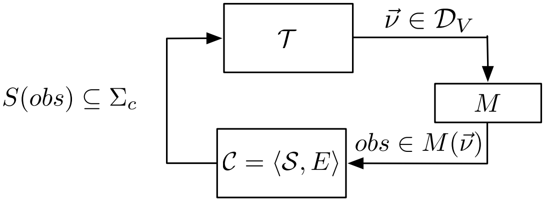
Partial observation

- Modeled by an observer $\langle \mathcal{D}_{Obs}, M \rangle$
- The observation space $\mathcal{D}_{Obs}$ can be infinite
- $M : \mathcal{D}_V \mapsto 2^{\mathcal{D}_{Obs}}$ is a mask

Example of observers

- Hidden variables : $M(\langle \mathsf{PX}, 10, 15 \rangle) = \langle \mathsf{PX}, 10 \rangle$
- Undistinguishable locations : $M(\langle \mathsf{PX}, 10, 15 \rangle) = M(\langle \mathsf{Choice}, 10, 15 \rangle)$
- ...

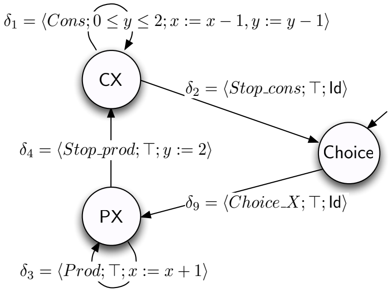
Memoryless Controllers



Formalization of the memoryless controller

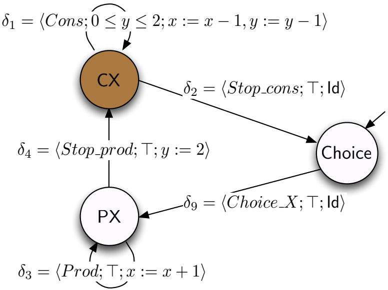
- Restricts the behavior of the system according to the observation
- The controller is a pair $\mathcal{C} = \langle \mathcal{S}, E \rangle$:
  1. The supervisory function $\mathcal{S} : \mathcal{D}_{Obs} \mapsto 2^{\Sigma_c}$ gives the sets $\mathcal{S}(obs)$ of controllable actions to forbid in obs $\Rightarrow$ memoryless controller
  2. $E \subseteq \mathcal{D}_V$ restricts the set of initial states

Example



System to be controlled and bad states

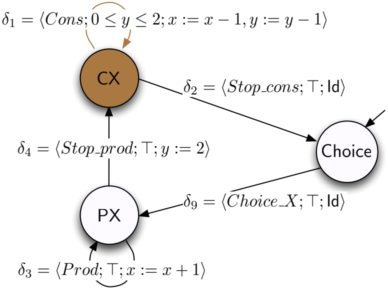
- A state is a tuple $\langle \ell, x, y \rangle \subseteq Loc \times \mathbb{N} \times \mathbb{N}$
- The mask M is defined as follows for each state $\vec{\nu} = \langle \ell, x, y \rangle$:
  - if $x \notin [10, 20]$, then $\vec{\nu}$ is perfectly observed
  - otherwise, $\vec{\nu}$ is undistinguishable from $\{\langle \ell, x_1, y \rangle \mid x_1 \in [10, 20]\}$
- $Bad = \{\langle \mathrm{CX}, x, y \rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\}$



System to be controlled and bad states

- A state is a tuple $\langle \ell, x, y \rangle \subseteq Loc \times \mathbb{N} \times \mathbb{N}$
- The mask M is defined as follows for each state $\vec{\nu} = \langle \ell, x, y \rangle$:
  - if $x \notin [10, 20]$, then $\vec{\nu}$ is perfectly observed
  - otherwise, $\vec{\nu}$ is undistinguishable from $\{\langle \ell, x_1, y \rangle \mid x_1 \in [10, 20]\}$
- $Bad = \{\langle \mathrm{CX}, x, y \rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\}$
- $I(Bad) = \{\langle \mathrm{CX}, x, y \rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\}$

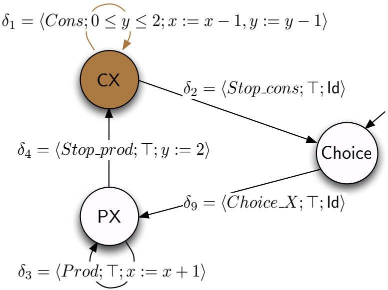
Example



System to be controlled and bad states

- A state is a tuple $\langle \ell, x, y\rangle \subseteq Loc \times \mathbb{N} \times \mathbb{N}$
- The mask M is defined as follows for each state $\vec{\nu} = \langle \ell, x, y\rangle$:
  - if $x \notin [10, 20]$, then $\vec{\nu}$ is perfectly observed
  - otherwise, $\vec{\nu}$ is undistinguishable from $\{\langle \ell, x_1, y\rangle \mid x_1 \in [10, 20]\}$
- $Bad = \{\langle \mathsf{CX}, x, y\rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\}$
- $I(Bad) = \{\langle \mathsf{CX}, x, y\rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\} \cup$
  $\{\langle \mathsf{CX}, x, y\rangle | (x \leq 11) \wedge (1 \leq y \leq 2)\}$

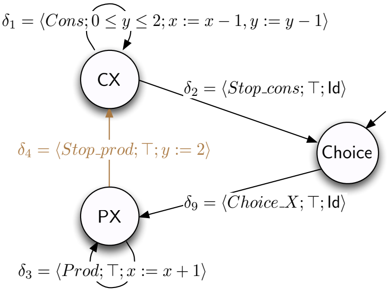
Example



System to be controlled and bad states

- A state is a tuple $\langle \ell, x, y \rangle \subseteq Loc \times \mathbb{N} \times \mathbb{N}$
- The mask M is defined as follows for each state $\vec{\nu} = \langle \ell, x, y \rangle$:
  - if $x \notin [10, 20]$, then $\vec{\nu}$ is perfectly observed
  - otherwise, $\vec{\nu}$ is undistinguishable from $\{\langle \ell, x_1, y \rangle \mid x_1 \in [10, 20]\}$
- $Bad = \{\langle \mathrm{CX}, x, y \rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\}$
- $\cup \{\langle \mathrm{CX}, x, y \rangle | (x \leq 11) \wedge (1 \leq y \leq 2)\} \cup \{\langle \mathrm{CX}, x, y \rangle | (x \leq 12) \wedge (y = 2)\}$

Example

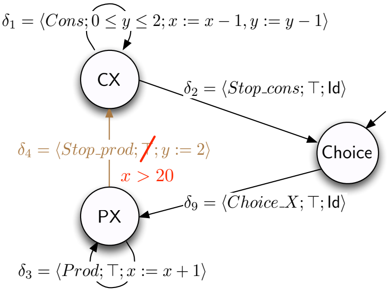


As a reminder : $I(Bad) = \{\langle \mathrm{CX}, x, y\rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\} \cup \{\langle \mathrm{CX}, x, y\rangle | (x \leq 11) \wedge (1 \leq y \leq 2)\} \cup \{\langle \mathrm{CX}, x, y\rangle | (x \leq 12) \wedge (y = 2)\}$.

Control function

Stop_prod forbidden in $M^{-1}(M(\{\langle \mathrm{PX}, x, y\rangle \,|\, x \leq 12\})) = \{\langle \mathrm{PX}, x, y\rangle \,|\, x \leq 20\}$

Example



As a reminder : $I(Bad) = \{\langle \mathsf{CX}, x, y \rangle | (x \leq 10) \wedge (0 \leq y \leq 2)\} \cup \{\langle \mathsf{CX}, x, y \rangle | (x \leq 11) \wedge (1 \leq y \leq 2)\} \cup \{\langle \mathsf{CX}, x, y \rangle | (x \leq 12) \wedge (y = 2)\}$.

Control function

Stop_prod forbidden in $M^{-1}(M(\{\langle \mathsf{PX}, x, y \rangle \,|\, x \leq 12\})) = \{\langle \mathsf{PX}, x, y \rangle \,|\, x \leq 20\}$

Beyond Memoryless Controller

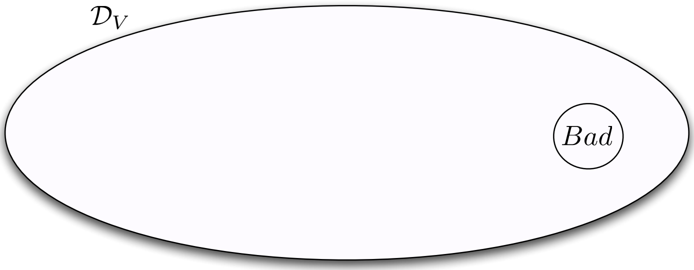
Memory improves the control decision

- Memoryless controller must take their decision on a single observation
- What about states that have the same observation, but can be distinguished by the past execution ?

Improvement of the controller

- *k*-memory controllers
- Online controllers

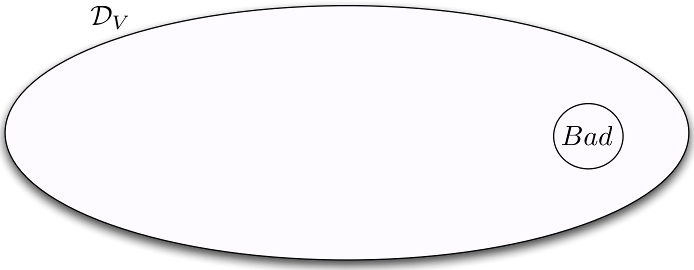
Online Controllers



Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy

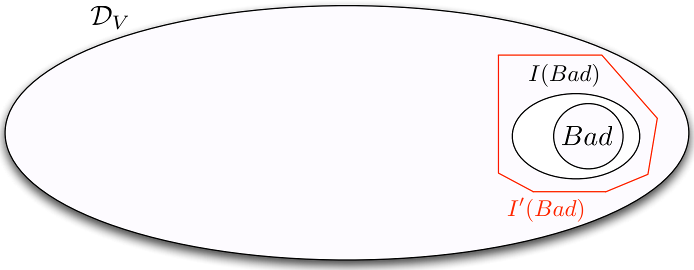
Online Controllers



Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy
- The algorithm is composed of two parts :

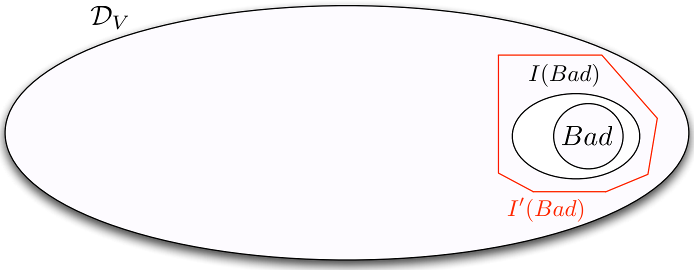
Online Controllers



Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy
- The algorithm is composed of two parts :
  1. offline part : an overapproximation $I'(Bad)$ of $I(Bad)$ is computed

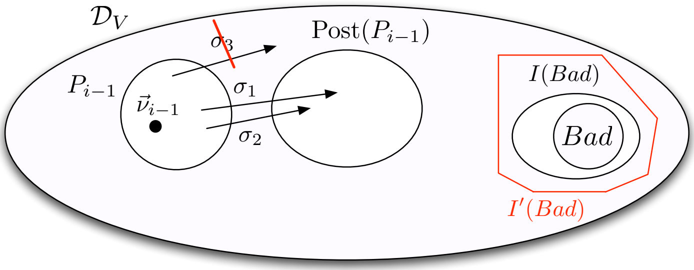
Online Controllers



Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy
- The algorithm is composed of two parts :
  1. offline part : an overapproximation $I'(Bad)$ of $I(Bad)$ is computed
  2. online part : for each observation obs_i received from $\mathcal{T}$, the controller computes an estimate $P_i = f(P_{i-1}, obs_i)$ of the current state $\vec{\nu}_i$ of $\mathcal{T}$

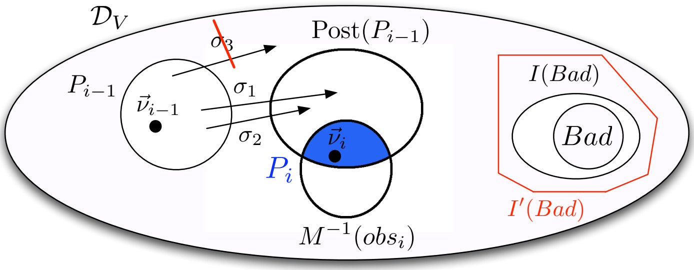
Online Controllers



Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy
- The algorithm is composed of two parts :
  1. offline part : an overapproximation $I'(Bad)$ of $I(Bad)$ is computed
  2. online part : for each observation obs_i received from $\mathcal{T}$, the controller computes an estimate $P_i = f(P_{i-1}, obs_i)$ of the current state $\vec{\nu}_i$ of $\mathcal{T}$

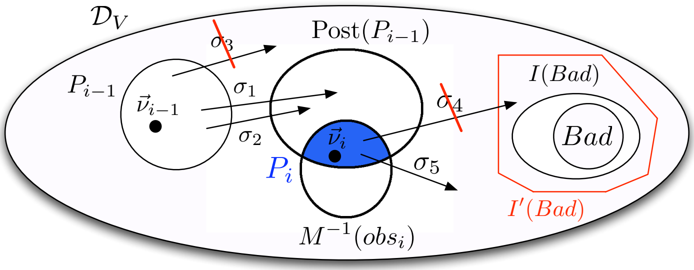
Online Controllers



Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy
- The algorithm is composed of two parts :
  1. offline part : an overapproximation $I'(Bad)$ of $I(Bad)$ is computed
  2. online part : for each observation obs_i received from $\mathcal{T}$, the controller computes an estimate $P_i = f(P_{i-1}, obs_i)$ of the current state $\vec{\nu}_i$ of $\mathcal{T}$

Online Controllers

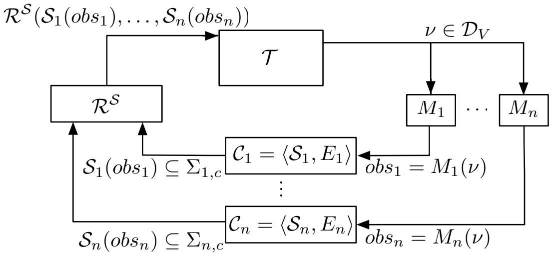


Online controllers

- The controller maintains an estimate of the current state of $\mathcal{T}$ to define its control policy
- The algorithm is composed of two parts :
  1. offline part : an overapproximation $I'(Bad)$ of $I(Bad)$ is computed
  2. online part : for each observation obs_i received from $\mathcal{T}$, the controller computes an estimate $P_i = f(P_{i-1}, obs_i)$ of the current state $\vec{\nu}_i$ of $\mathcal{T}$ and forbids the actions that lead to $I(Bad)$ from P_i

Outline

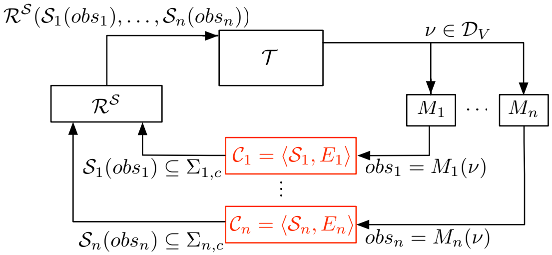
Decentralized Controller



Decentralized Approach

- The decentralized approach is more suitable for the control of distributed systems with synchronous communications

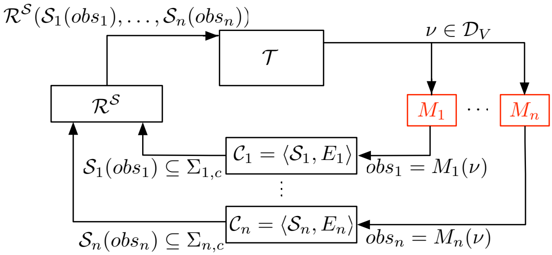
Decentralized Controller



Decentralized Approach

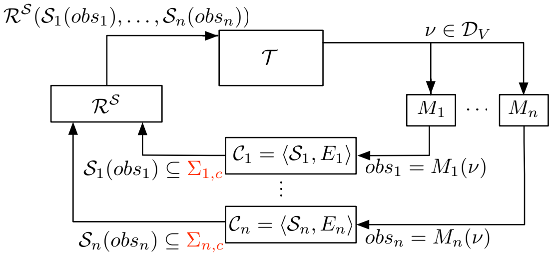
- The decentralized approach is more suitable for the control of distributed systems with synchronous communications
- In this approach, the system is controlled by n controllers $\mathcal{C}_i$

Decentralized Controller



Decentralized Approach

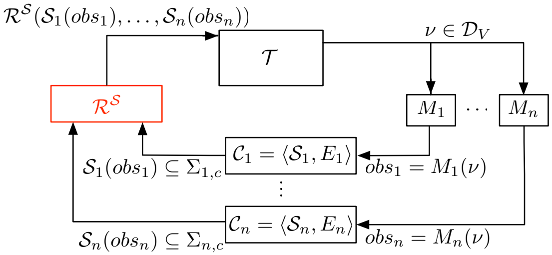
- The decentralized approach is more suitable for the control of distributed systems with synchronous communications
- In this approach, the system is controlled by n controllers $\mathcal{C}_i$
- Each controller $\mathcal{C}_i$ has a partial observation of the system modeled by the observer $\langle Obs_i, M_i \rangle$



Decentralized Approach

- The decentralized approach is more suitable for the control of distributed systems with synchronous communications
- In this approach, the system is controlled by n controllers $\mathcal{C}_i$
- Each controller $\mathcal{C}_i$ has a partial observation of the system modeled by the observer $\langle Obs_i, M_i \rangle$ and can control the set $\Sigma_{i,c}$ of actions

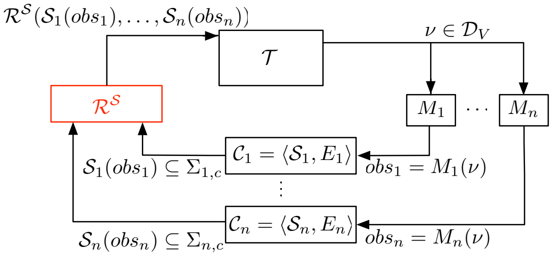
Decentralized Controller



Decentralized Approach

- The decentralized approach is more suitable for the control of distributed systems with synchronous communications
- In this approach, the system is controlled by n controllers $\mathcal{C}_i$
- Each controller $\mathcal{C}_i$ has a partial observation of the system modeled by the observer $\langle Obs_i, M_i \rangle$ and can control the set $\Sigma_{i,c}$ of actions
- A synchronization mechanism, called fusion rule, defines the global control to be applied to the system from the control decisions of the controllers $\mathcal{C}_i$:

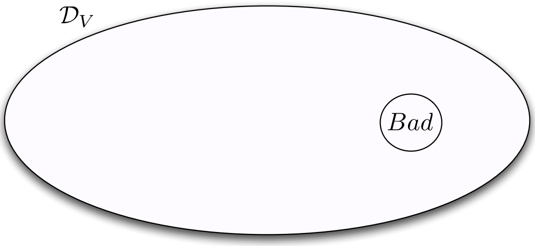
Decentralized Controller



Decentralized Approach

- The decentralized approach is more suitable for the control of distributed systems with synchronous communications
- In this approach, the system is controlled by n controllers $\mathcal{C}_i$
- Each controller $\mathcal{C}_i$ has a partial observation of the system modeled by the observer $\langle Obs_i, M_i \rangle$ and can control the set $\Sigma_{i,c}$ of actions
- A synchronization mechanism, called fusion rule, defines the global control to be applied to the system from the control decisions of the controllers $\mathcal{C}_i$: an action is forbidden if each controller controlling this action proposes to forbid it

Decentralized Controller(2)

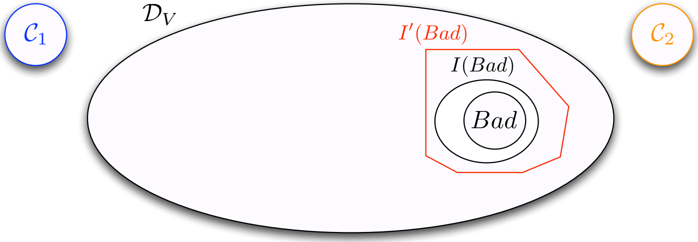


Basic decentralized Problem

To synthesize n valid and non-trivial controllers i.e., n controllers that

- prevent from reaching *Bad*
- do not reduce the behavior of the controlled system to the empty set

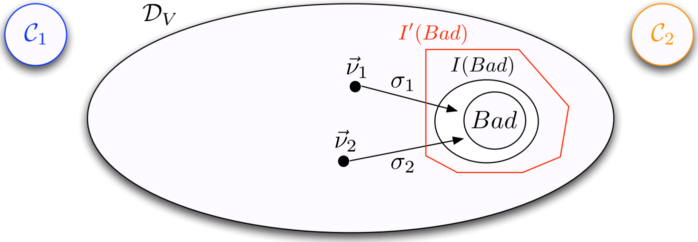
Decentralized Controller(2)



Control Policy

- Each controller $\mathcal{C}_i$ forbids the actions that it controls and that lead to $I'(Bad)$

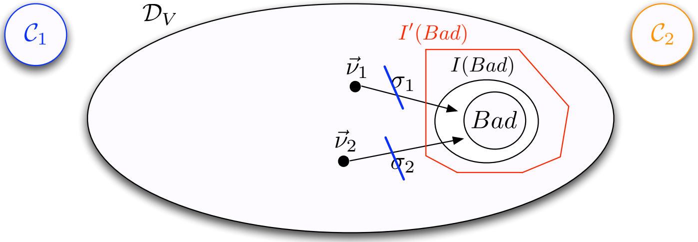
Decentralized Controller(2)



Control Policy

- Each controller $\mathcal{C}_i$ forbids the actions that it controls and that lead to $I'(Bad)$

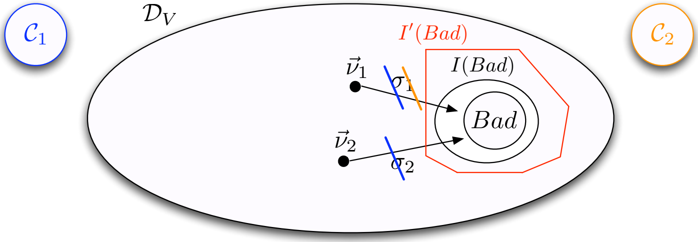
Decentralized Controller(2)



Control Policy

- Each controller $\mathcal{C}_i$ forbids the actions that it controls and that lead to $I'(Bad)$
  $\mathcal{C}_1$ forbids σ_1 in $\vec{\nu}_1$
  $\mathcal{C}_1$ forbids σ_2 in $\vec{\nu}_2$

Decentralized Controller(2)

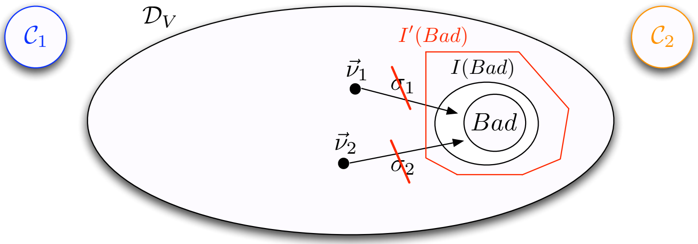


Control Policy

- Each controller $\mathcal{C}_i$ forbids the actions that it controls and that lead to $I'(Bad)$

$\mathcal{C}_1$ forbids σ_1 in $\vec{\nu}_1$	$\mathcal{C}_2$ forbids σ_1 in $\vec{\nu}_1$
$\mathcal{C}_1$ forbids σ_2 in $\vec{\nu}_2$	$\mathcal{C}_2$ allows σ_2 in $\vec{\nu}_2$ (uncontrollable action)

Decentralized Controller(2)



Control Policy

- Each controller $\mathcal{C}_i$ forbids the actions that it controls and that lead to $I'(Bad)$

$\mathcal{C}_1$ forbids σ_1 in $\vec{\nu}_1$	$\mathcal{C}_2$ forbids σ_1 in $\vec{\nu}_1$
$\mathcal{C}_1$ forbids σ_2 in $\vec{\nu}_2$	$\mathcal{C}_2$ allows σ_2 in $\vec{\nu}_2$ (uncontrollable action)

$\mathcal{R}^{\mathcal{S}}$ forbids σ_1 in $\vec{\nu}_1$
$\mathcal{R}^{\mathcal{S}}$ forbids σ_2 in $\vec{\nu}_2$ (because $\mathcal{C}_2$ does not control σ_2)

Outline

Distributed Systems with Asynchronous Communications

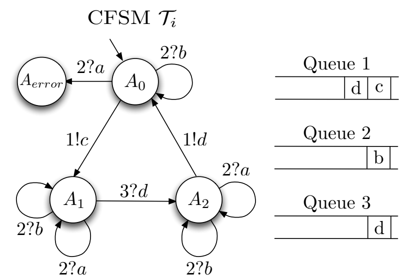
Motivation

- The decentralized and modular approaches cannot be used for the control of distributed systems with asynchronous communications
- The communications between the subsystems are not instantaneous

Distributed Approach

- We propose a distributed approach that takes into account the asynchronous nature of communications
- We consider distributed systems $\mathcal{T}$ composed of several subsystems $\mathcal{T}_i$ communicating through reliable unbounded FIFO channels
- $\mathcal{T}_i$ is modeled by a communicating finite state machine (CFSM)

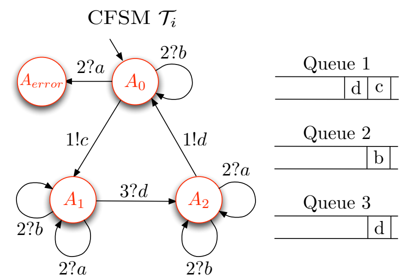
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

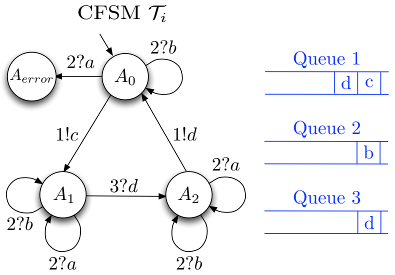
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

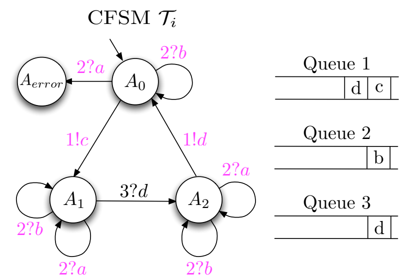
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

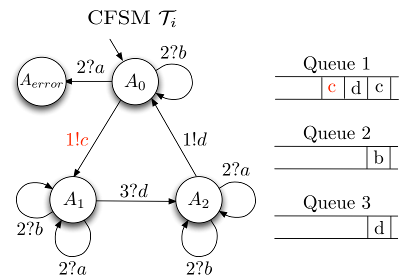
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

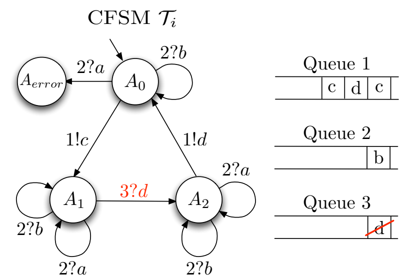
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell'_i \rangle$
- Semantics = infinite Labeled Transition System (LTS)

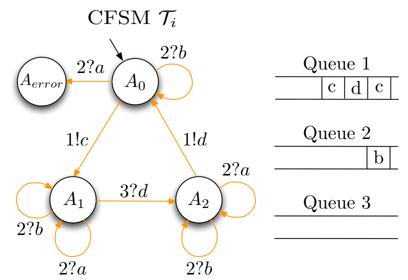
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

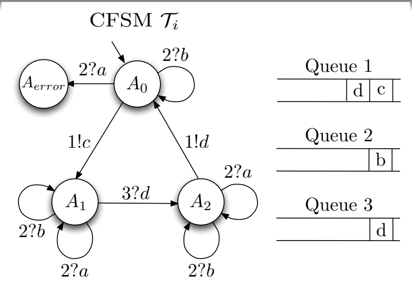
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

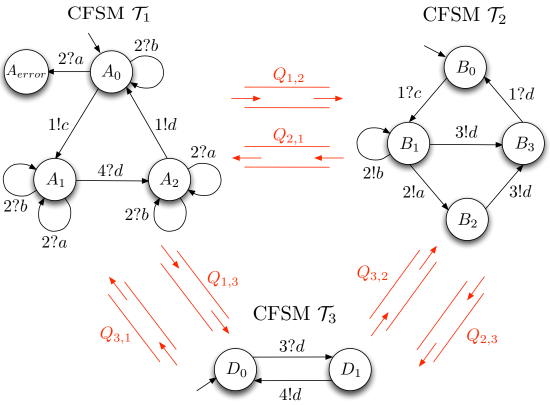
System to be Controlled



Communicating Finite State Machine $\mathcal{T}_i$

- Finite set L_i of locations
- Reliable and unbounded FIFO channels
- Finite set Σ_i of actions. An action σ is :
  - either an output $Q!m$
  - or an input $Q?m$
- Finite set of transitions $\langle \ell_i, \sigma, \ell_i' \rangle$
- Semantics = infinite Labeled Transition System (LTS)

System to be Controlled

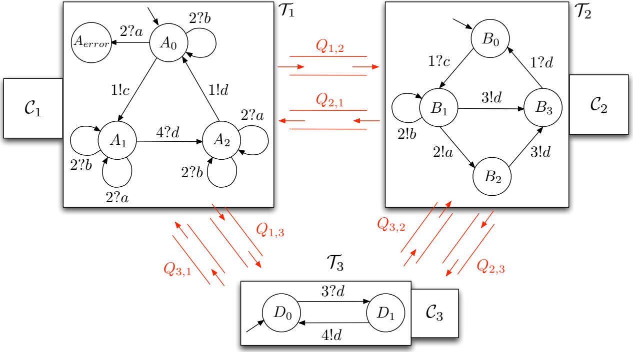


Communication Architecture

There two queues between each pair of subsystems $\mathcal{T}_i$ and $\mathcal{T}_j$:

- $Q_{i,j}$: $\mathcal{T}_i$ writes on this queue and $\mathcal{T}_j$ reads the sent messages
- $Q_{j,i}$: $\mathcal{T}_i$ reads on this queue and $\mathcal{T}_j$ writes on it

Means of Observation and Control



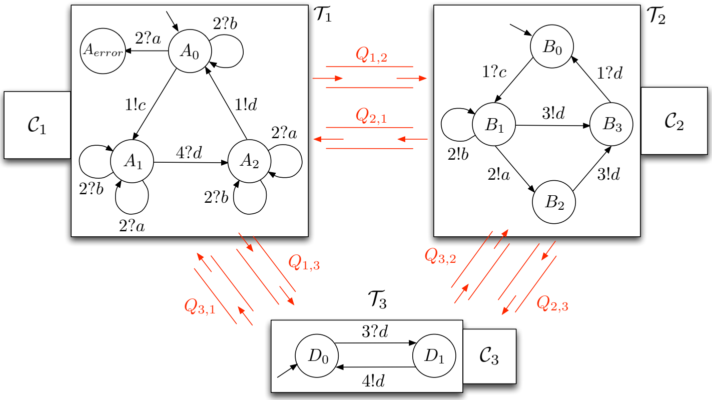
Observation

- Each subsystem $\mathcal{T}_i$ is controlled by a controller $\mathcal{C}_i$
- $\mathcal{C}_i$ only observes the current state of $\mathcal{T}_i$

Control Mechanism

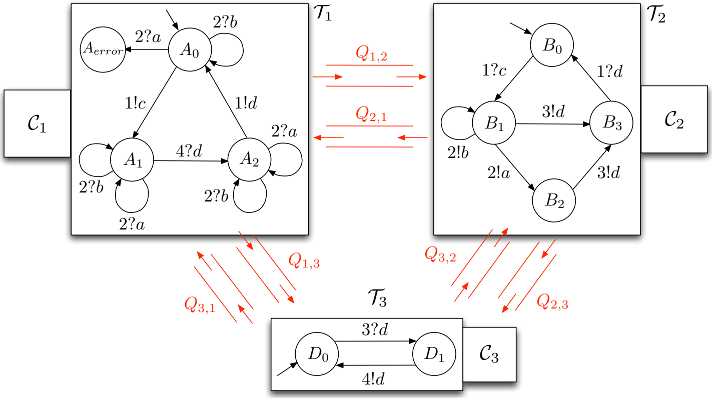
- For each $\mathcal{T}_i$, the set $\Sigma_i = \Sigma_{i,c} \uplus \Sigma_{i,uc}$

Formalization of the Controllers



Communication

- The controllers $\mathcal{C}_i$ communicate with each other to have a better knowledge of the system $\mathcal{T}$
- They communicate by adding information to the messages normally exchanged by the subsystems
- Each controller $\mathcal{C}_i$ uses the exchanged information and the information received from $\mathcal{T}_i$ to compute an estimate of the current state of the distributed system $\mathcal{T}$



Formalization of the controllers $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$

1. $\mathcal{S}_i$ defines, for each set P of states of $\mathcal{T}$, a set $\mathcal{S}_i(P)$ of controllable actions that $\mathcal{T}_i$ cannot execute when P is the estimate of the current state of $\mathcal{T}$ computed by $\mathcal{C}_i$.
2. E_i restricts the set of initial states of $\mathcal{T}$.

Problem

Distributed Problem

To synthesize valid and non-trivial controllers i.e., controllers that

- prevent from reaching *Bad*
- do not reduce the behavior of the controlled system to the empty set

Problem

Distributed Problem

To synthesize valid and non-trivial controllers i.e., controllers that

- prevent from reaching *Bad*
- do not reduce the behavior of the controlled system to the empty set

Property

The distributed problem is undecidable

Problem

Distributed Problem

To synthesize valid and non-trivial controllers i.e., controllers that

- prevent from reaching *Bad*
- do not reduce the behavior of the controlled system to the empty set

Property

The distributed problem is undecidable

Our approach

- Online controllers
- States estimates computed using abstract interpretation
- Communication (without explicit synchronisation) between controllers

Algorithm for the Distributed Problem

Outline

Synthesize n controllers that compute, during the execution of the distributed system $\mathcal{T}$, estimates of the current state of $\mathcal{T}$ and define their control policy from their state estimates. Since the system is asynchronous, a state estimates for controller $\mathcal{C}_i$ must "guess" the possible future behaviour of the other subsystems $\mathcal{T}_j$, $j \neq i$.

Algorithm for the Distributed Problem

Outline

Synthesize n controllers that compute, during the execution of the distributed system $\mathcal{T}$, estimates of the current state of $\mathcal{T}$ and define their control policy from their state estimates. Since the system is asynchronous, a state estimates for controller $\mathcal{C}_i$ must "guess" the possible future behaviour of the other subsystems $\mathcal{T}_j$, $j \neq i$.

Algorithm computing the state estimates

- $\mathcal{C}_i$ maintains a estimate CS_i of the current state of $\mathcal{T}$
- The controllers exchange their state estimate by adding these information to the messages normally exchanged by the subsystems
- For each transition fired by $\mathcal{T}_i$, the controller $\mathcal{C}_i$ updates CS_i from the information received from $\mathcal{T}_i$ and the other controllers

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell'_i \rangle$

When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\text{Post}_\delta^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_\delta^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ $(\forall k \neq i)$ continues its execution

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell'_i \rangle$

When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\mathrm{Post}_\delta^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\mathrm{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\mathrm{Post}_\delta^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ $(\forall k \neq i)$ continues its execution

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell_i' \rangle$

When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\text{Post}_{\delta}^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta}^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ $(\forall k \neq i)$ continues its execution

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell'_i \rangle$

When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\mathrm{Post}_\delta^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\mathrm{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\mathrm{Post}_\delta^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ $(\forall k \neq i)$ continues its execution

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell_i' \rangle$

When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\text{Post}_\delta^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_\delta^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ ($\forall k \neq i$) continues its execution

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell_i' \rangle$

When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\text{Post}_\delta^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_\delta^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ $(\forall k \neq i)$ continues its execution

Update of CS_i after the input transition $\delta = \langle \ell_i, Q_{j,i}?m, \ell_i' \rangle$

- $\mathcal{C}_i$ tries to refine its knowledge of the system using the state estimate CS_j
- Roughly an intersection, followed by Post_δ : $CS_i := \text{Post}_\delta(CS_i \cap CS_j)$

Computation of the state estimates

Update of CS_i after the output transition $\delta = \langle \ell_i, Q_{i,j}!m, \ell_i' \rangle$

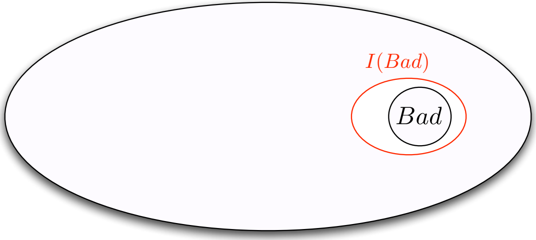
When $\mathcal{T}_i$ wants to send a message m to $\mathcal{T}_j$ on the queue $Q_{i,j}$:

- CS_i is sent to $\mathcal{C}_j$ with m
- CS_i is updated :
  - $\text{Post}_\delta^{\mathcal{T}}(CS_i)$: takes into account the execution of δ
  - $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_\delta^{\mathcal{T}}(CS_i))$:
    takes into account the fact that $\mathcal{T}_k$ $(\forall k \neq i)$ continues its execution

Update of CS_i after the input transition $\delta = \langle \ell_i, Q_{j,i}?m, \ell_i' \rangle$

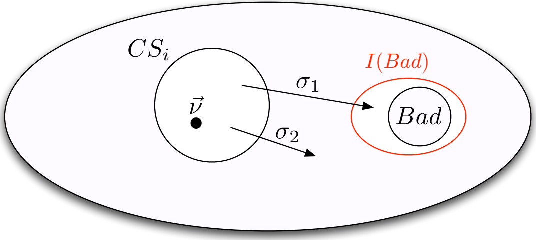
- $\mathcal{C}_i$ tries to refine its knowledge of the system using the state estimate CS_j
- Roughly an intersection, followed by Post_δ : $CS_i := \text{Post}_\delta(CS_i \cap CS_j)$
- CS_j may be out-of-date, we may have to update it first, or ignore it. We use vector clocks to detect out-of-date messages

Control algorithm



Semi-algorithm

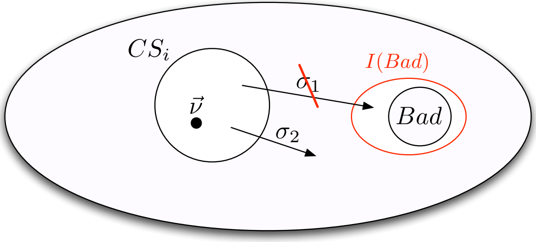
- Offline part : Computation of the set $I(Bad)$ of states of $\mathcal{T}$ leading uncontrollably to Bad



Semi-algorithm

- Offline part : Computation of the set $I(Bad)$ of states of $\mathcal{T}$ leading uncontrollably to Bad
- Online part : For each action executed by $\mathcal{T}_i$, the controller $\mathcal{C}_i$ defines its control policy as follows :
  - Updates its estimate CS_i of the current state $\vec{\nu}$ of $\mathcal{T}$

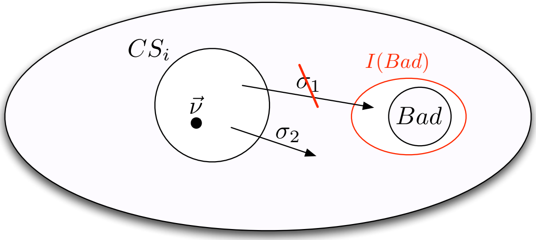
Control algorithm



Semi-algorithm

- Offline part : Computation of the set $I(Bad)$ of states of $\mathcal{T}$ leading uncontrollably to Bad
- Online part : For each action executed by $\mathcal{T}_i$, the controller $\mathcal{C}_i$ defines its control policy as follows :
  - Updates its estimate CS_i of the current state $\vec{\nu}$ of $\mathcal{T}$
  - Forbids the actions leading to $I(Bad)$ from CS_i

Control algorithm



Semi-algorithm

- Offline part : Computation of the set $I(Bad)$ of states of $\mathcal{T}$ leading uncontrollably to Bad
- Online part : For each action executed by $\mathcal{T}_i$, the controller $\mathcal{C}_i$ defines its control policy as follows :
  - Updates its estimate CS_i of the current state $\vec{\nu}$ of $\mathcal{T}$
  - Forbids the actions leading to $I(Bad)$ from CS_i

Effective Algorithm by Means of Abstract Interpretation

- Abstract lattice : regular languages is used to abstract the content of the FIFO channels
- Widening operator : works on regular languages

Abstract Lattices for FIFO Systems

Regular languages as an abstract lattice

- Concrete lattice : 2^{Σ^*}, abstract lattice $\mathrm{Reg}(\Sigma)$
- Canonical representation : minimal deterministic finite automomata
- When there are several FIFO channels : QDD representation
- Representation framework (no Galois connection)

Abstract Lattices for FIFO Systems

Regular languages as an abstract lattice

- Concrete lattice : 2^{Σ^*}, abstract lattice $\mathrm{Reg}(\Sigma)$
- Canonical representation : minimal deterministic finite automomata
- When there are several FIFO channels : QDD representation
- Representation framework (no Galois connection)

Widening operator

- Idea : quotient the automaton by an equivalence relation
- (colored) k-bounded bisimulation
- Convergence : number of equivalence classes is bounded (depends on k and $|\Sigma|$)

Abstract Lattices for FIFO Systems

Regular languages as an abstract lattice

- Concrete lattice : 2^{Σ^*}, abstract lattice $\mathrm{Reg}(\Sigma)$
- Canonical representation : minimal deterministic finite automomata
- When there are several FIFO channels : QDD representation
- Representation framework (no Galois connection)

Widening operator

- Idea : quotient the automaton by an equivalence relation
- (colored) k-bounded bisimulation
- Convergence : number of equivalence classes is bounded (depends on k and $|\Sigma|$)

Remark : we may work with an infinite alphabet of messages

Outline

Conclusion

Contributions

- Abstract interpretation is an efficient tool for controller synthesis
- The “real” problems : partial observation, interaction between controllers

Open questions

- Non-blocking properties : is it possible to use under-approximations ?
- Is it interesting for LMeASI ?