Constraint Programming for the Control of Discrete Event Dynamic Systems

Gérard Verfaillie, Cédric Pralet
ONERA, Toulouse, France
{Gerard.Verfaillie,Cedric.Pralet}@onera.fr

ONERA
THE FRENCH AEROSPACE LAB

CP 2013 Tutorial, CP for the control of DEDS, Uppsala, September 20th, 2013

Objective of the tutorial

To present the different aspects of **Discrete Event Dynamic Systems** (DEDS) and of their **control**.

To show how **Constraint Programming** (CP) can or could be used for **modelling** and **solving control problems** or **subproblems** for DEDS.

To consider things from **theoretical** and **practical** points of view.

Tutorial outline

1. Basics of **DEDS**;
2. DEDS **control** problems;
3. **Planning** problems and four ways of using CP;
4. **Scheduling** problems and the use of CP;
5. **Real world** problems.

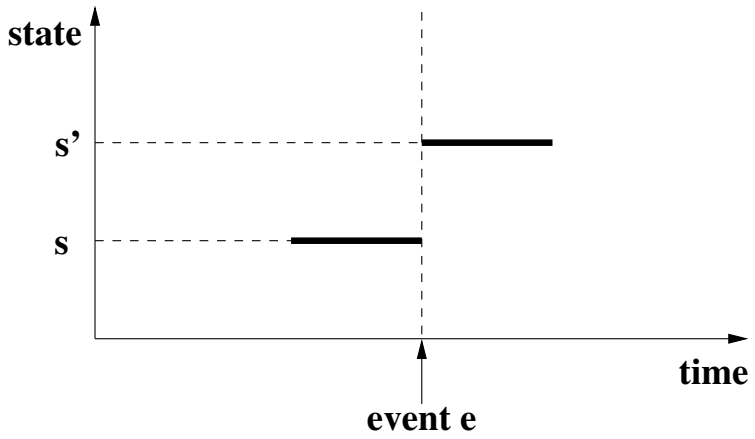
1. Basics of DEDS

What discrete event dynamic systems are

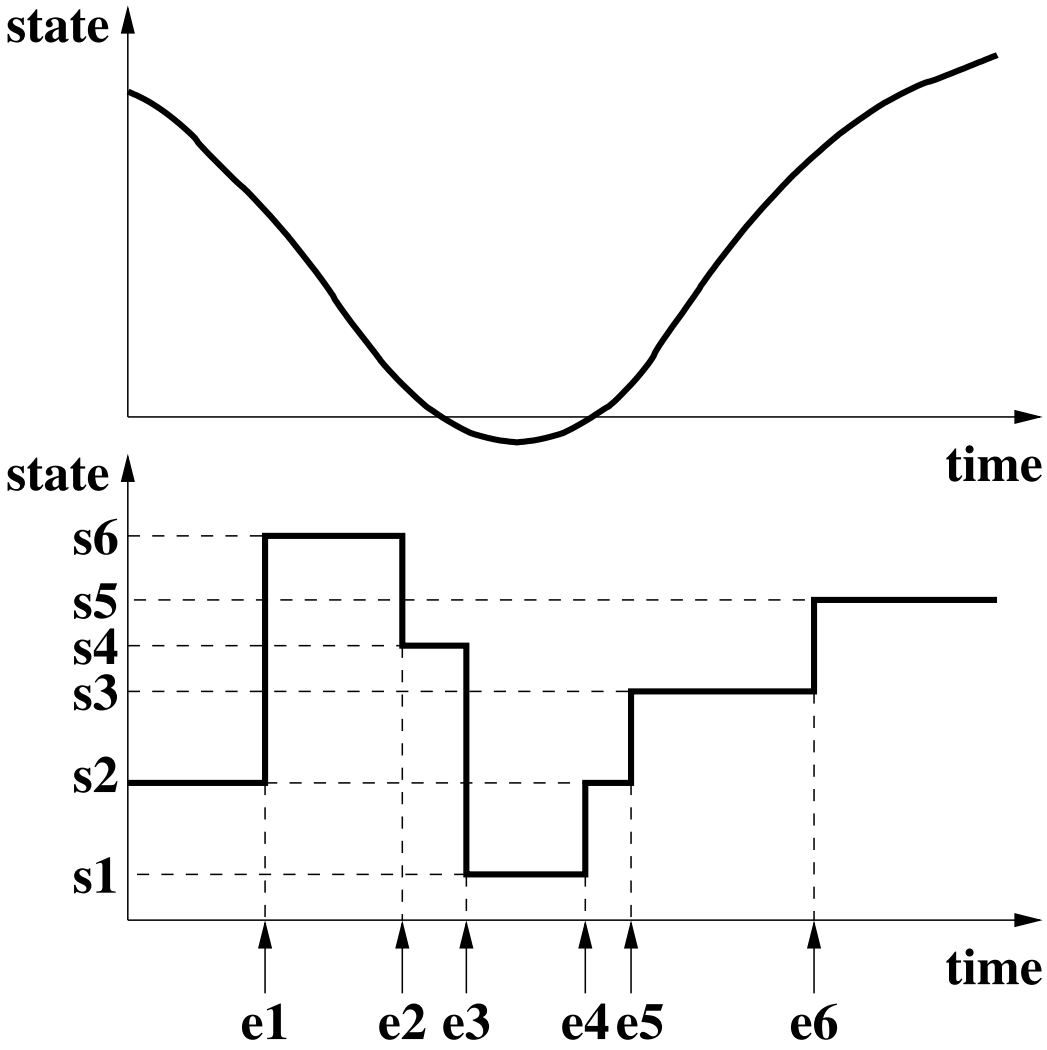
Dynamic systems: systems whose **state changes**.

Discrete event dynamic systems: systems whose state changes due to **instantaneous events**:

- events that occur at **discrete times** and produce **instantaneous changes** in the system state;
- state that remains **unchanged** between successive events.



Discrete event vs. Continuous time dynamic systems



Examples of DEDS

- a **computer** (jobs and processors);
- a **queueing** system (customers and servers);
- a **routing** system in a communication network (messages and switches);
- a **manufacturing** system (tasks and machines);
- a **database** management system (transactions);
- a **control** system in an aircraft (modes) ...

More and more present with the development of **numeric devices**.

Basic components of a DEDS

- a set of possible system **states**;
- a set of possible **events**;
- event **preconditions** and **effects**;
- a set of possible **initial** states;
- a set of **marked** (accepting, final) states.

Classical modeling frameworks for DEDS

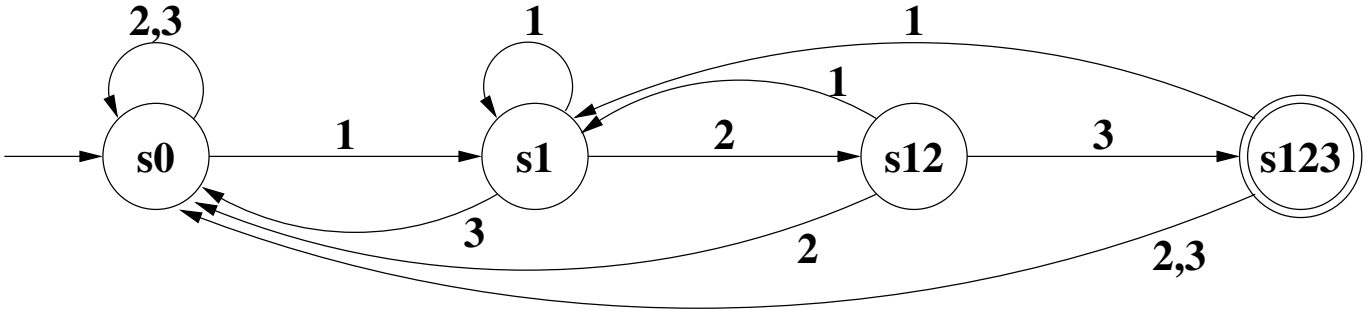
- **Automata**;
- **Petri nets**;
- **Markov chains**.

(Finite state deterministic) automata

$\mathbf{A} = \langle \mathbf{S}, \mathbf{E}, \mathbf{T}, \mathbf{I}, \mathbf{M} \rangle$

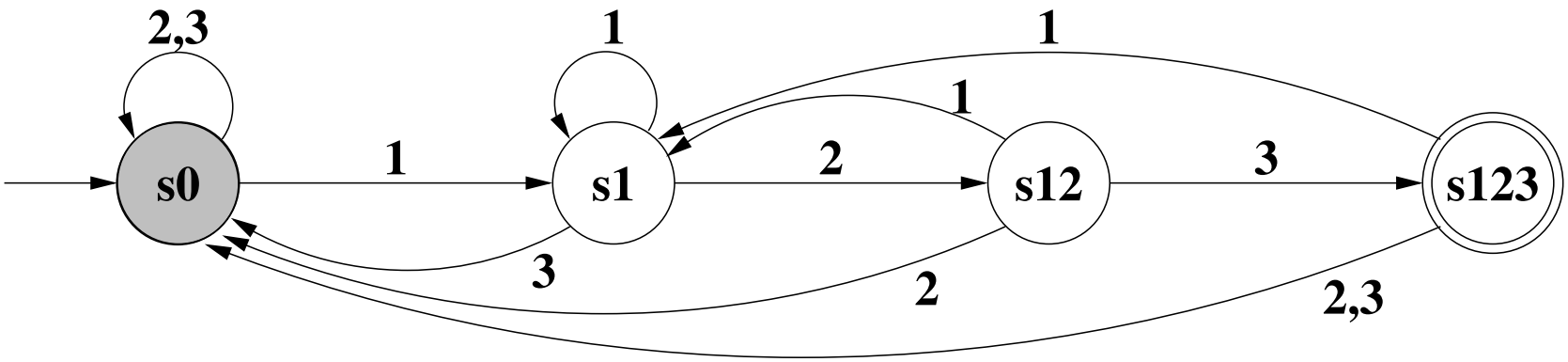
- finite set **S** of possible **states**;
- finite set **E** of possible **events**;
- **transition** (partial) function $\mathbf{T} : \mathbf{S} \times \mathbf{E} \rightarrow \mathbf{S}$
- **initial** state: $\mathbf{I} \in \mathbf{S}$
- set of **marked** states: $\mathbf{M} \subseteq \mathbf{S}$

Example: a **digicode** which recognizes the sequence 123.

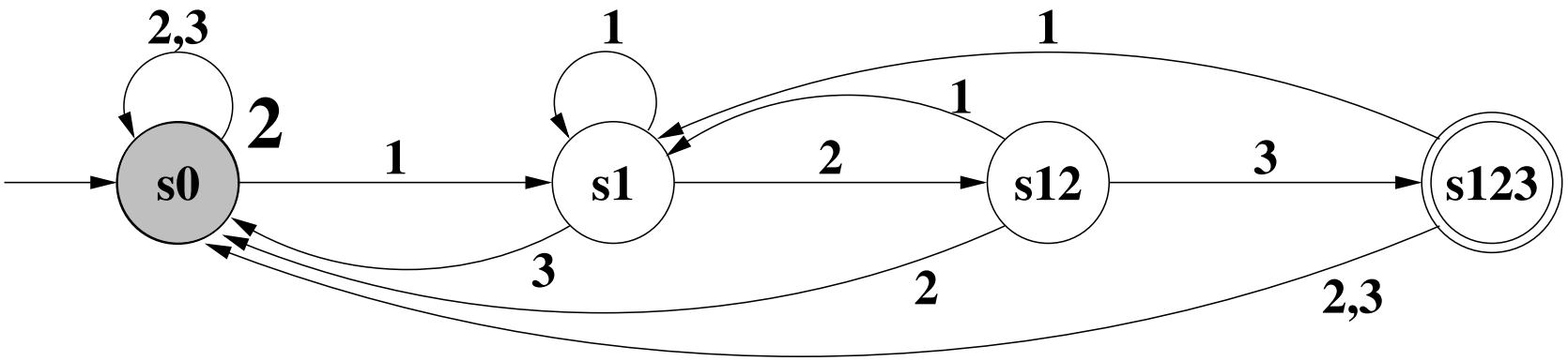


$\rightarrow$ **Directed labelled graph**.

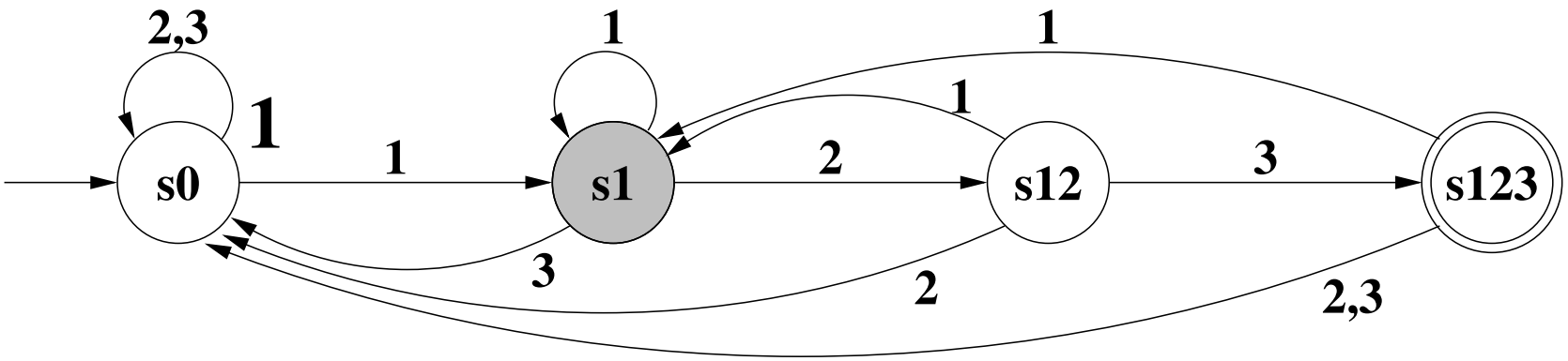
Example of execution (0)



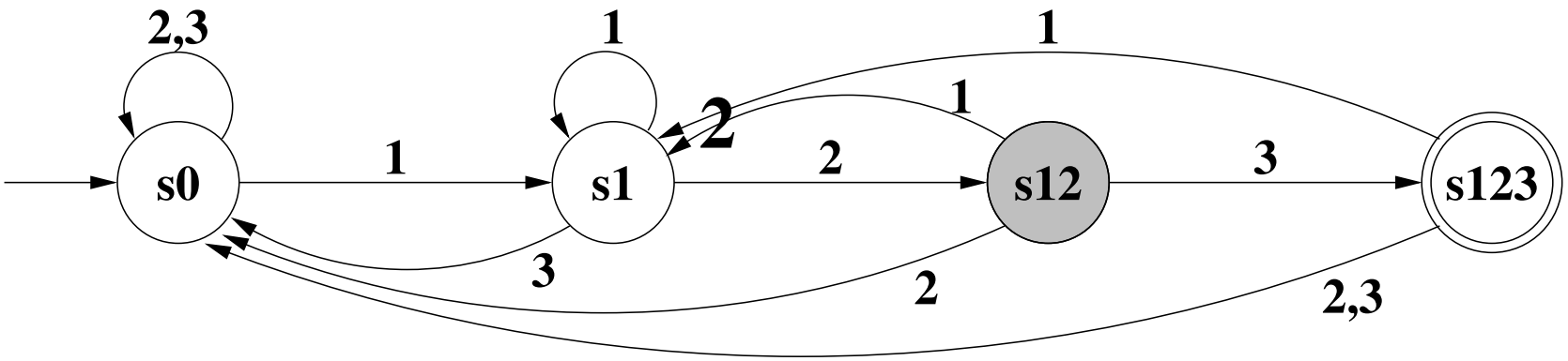
Example of execution (1)



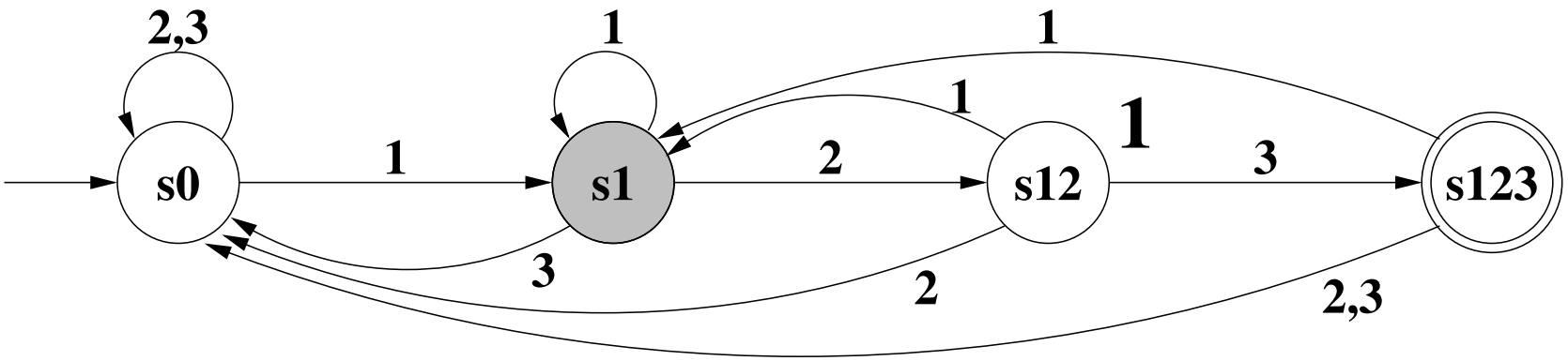
Example of execution (2)



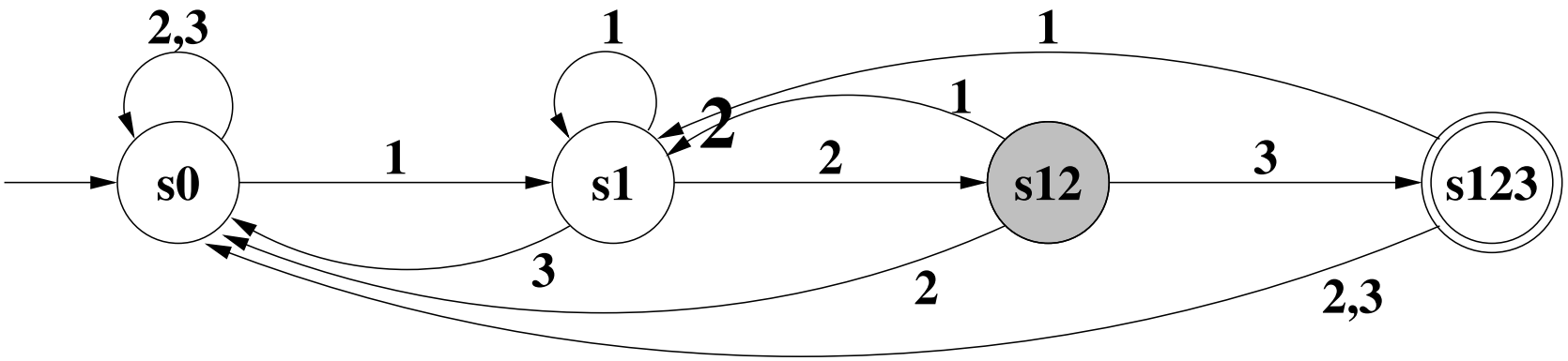
Example of execution (3)



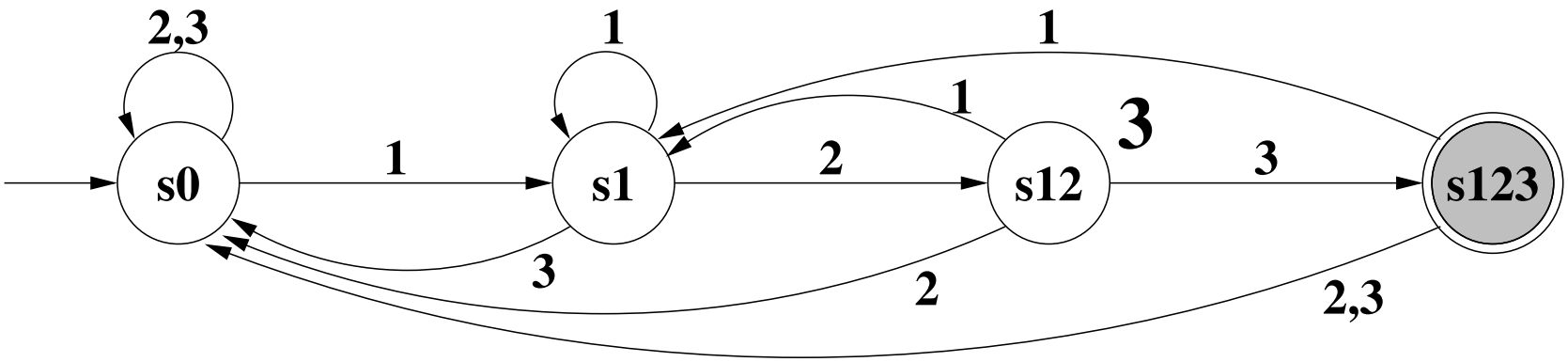
Example of execution (4)



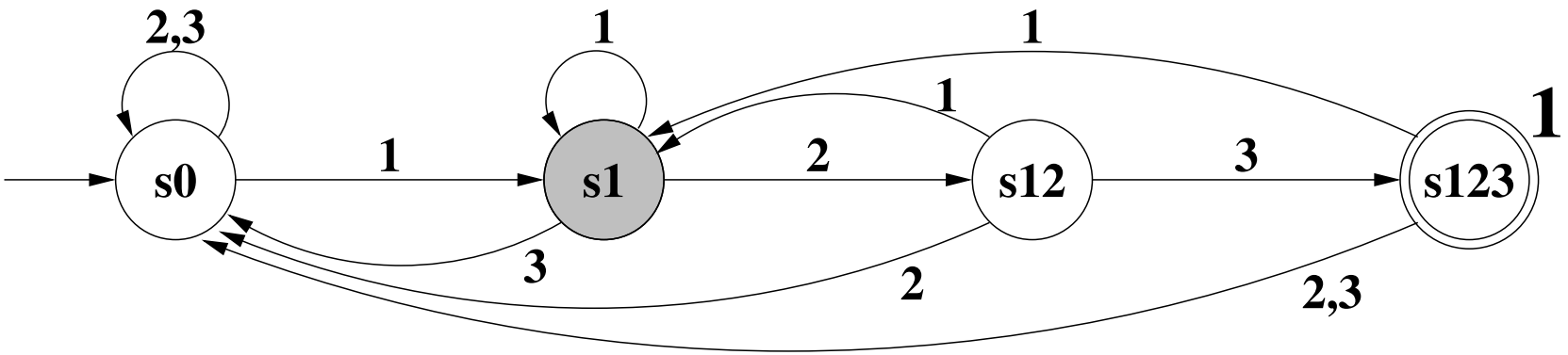
Example of execution (5)



Example of execution (6)



Example of execution (7)

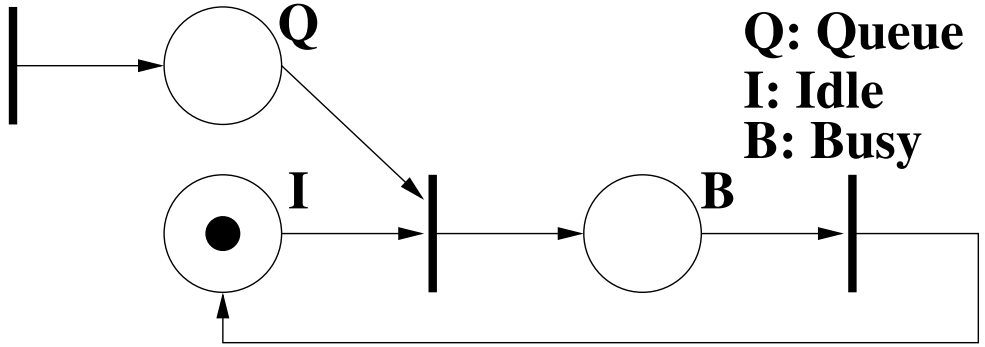


Petri nets

$\mathbf{N} = \langle \mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W}, \mathbf{I} \rangle$

- finite set $\mathbf{P}$ of **places**;
- finite set $\mathbf{T}$ of **transitions**;
- set of **arcs** from places to transitions and from transitions to places
  $\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T}) \cup (\mathbf{T} \times \mathbf{P})$
- arc **weighting** function: $\mathbf{W} : \mathbf{A} \to \mathbb{N}$
- **initial** marking: $\mathbf{I} : \mathbf{P} \to \mathbb{N}$

Example: a simple **queueing system**.



$\to$ **Directed bipartite weighted graph**.

Petri nets

Warning: a **state** is a **marking** *i*.e., a given number of tokens in each place
$\rightarrow$ **Unbounded** set of possible states.

Transition **preconditions**: enough tokens in all the input places.

$$\forall p \in \mathbf{P} : \ \mathbf{m}(p) \geq \mathbf{W}(p,t)$$

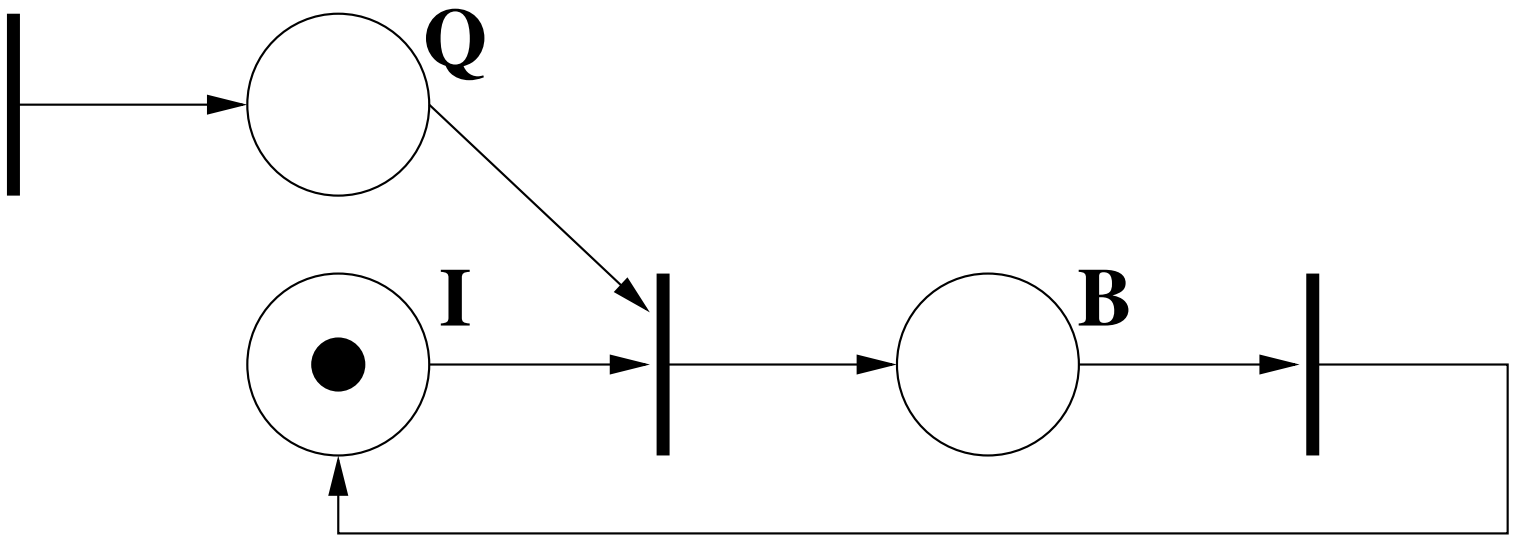
Transition **effects**: tokens removed from the input places and added to the output places.

$$\forall p \in \mathbf{P} : \ \mathbf{m'}(p) = \mathbf{m}(p) - \mathbf{W}(p,t) + \mathbf{W}(t,p)$$

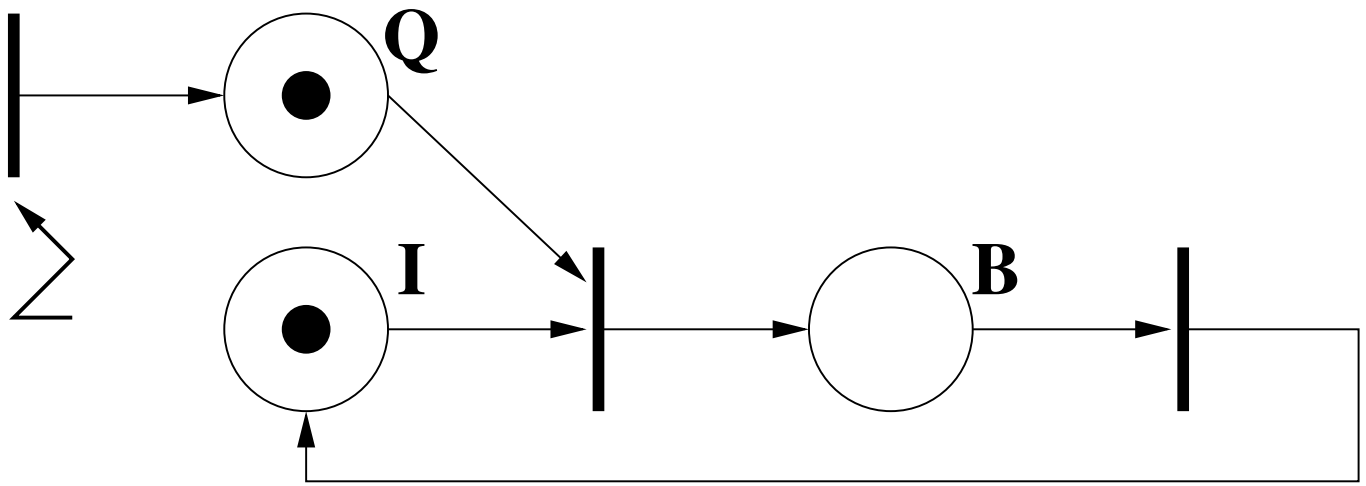
Almost the same expressing power as automata.

More "**structured**" than automata.

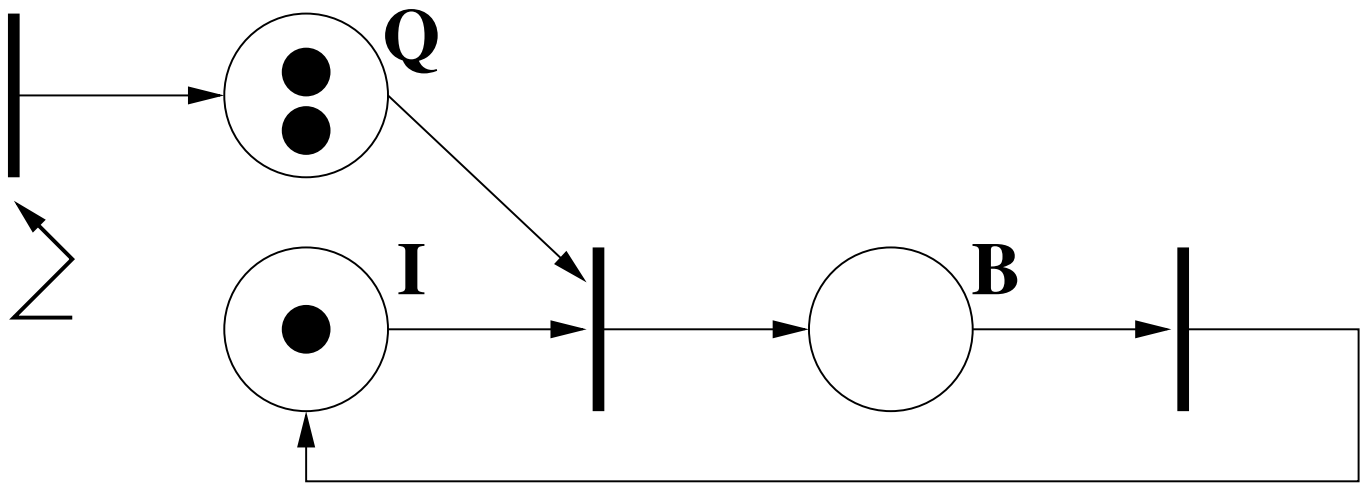
Example of execution (0)



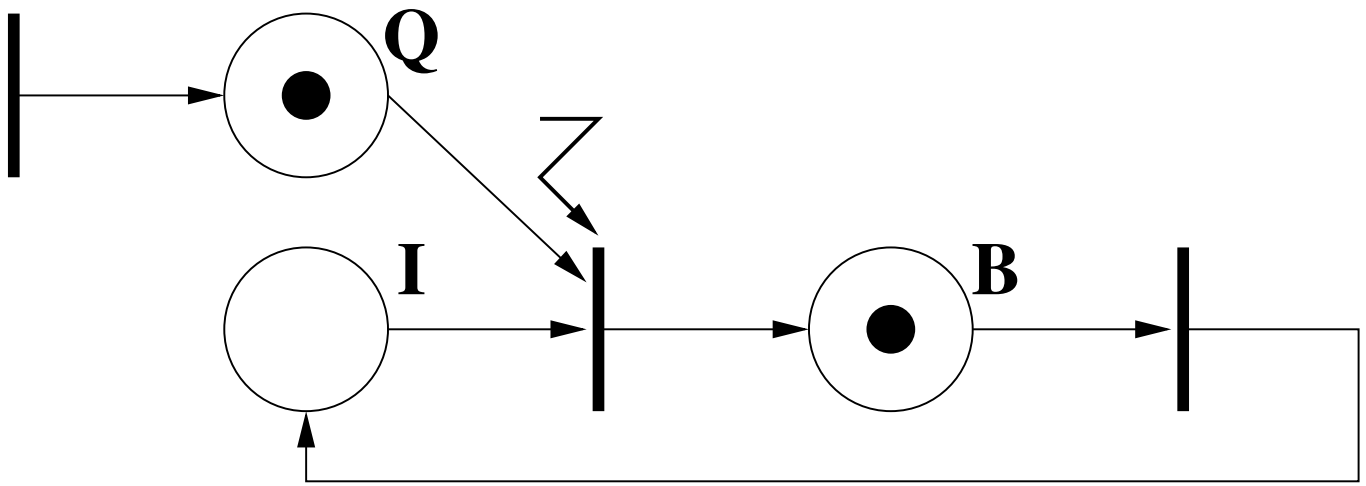
Example of execution (1)



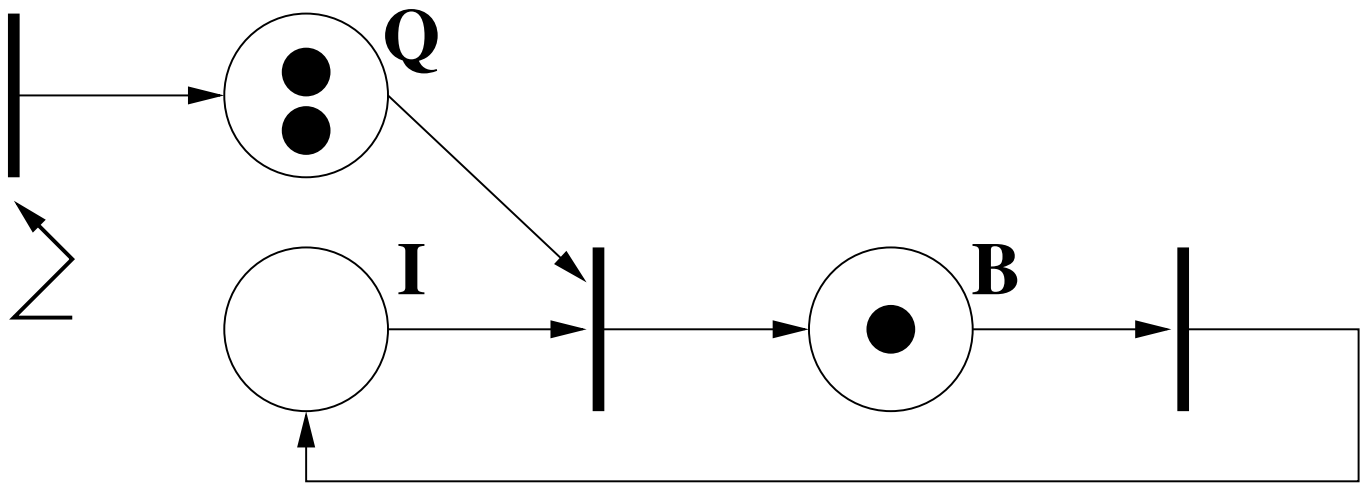
Example of execution (2)



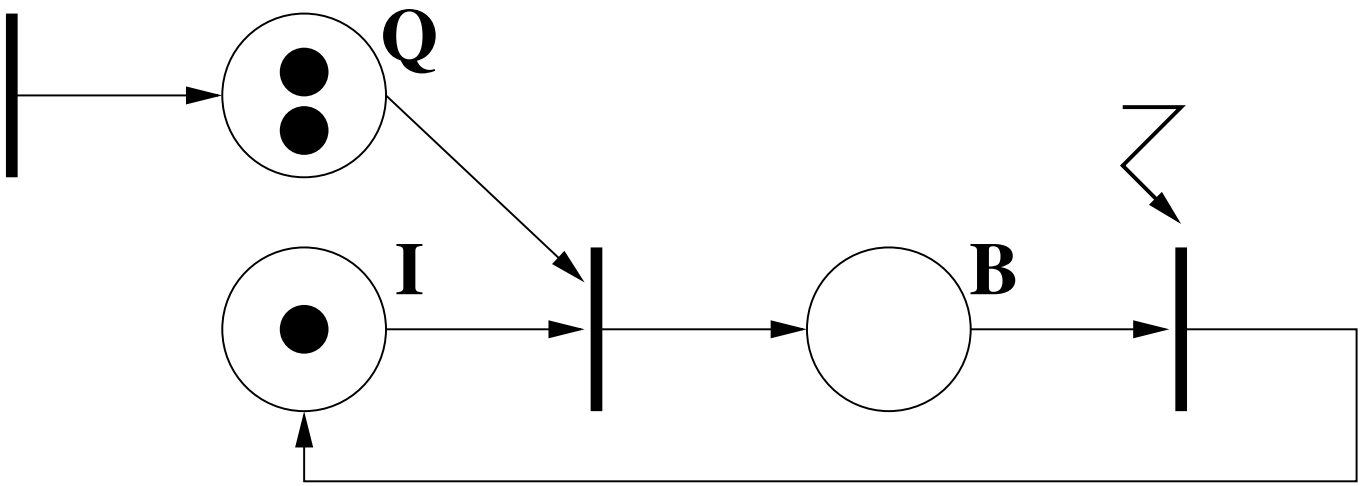
Example of execution (3)



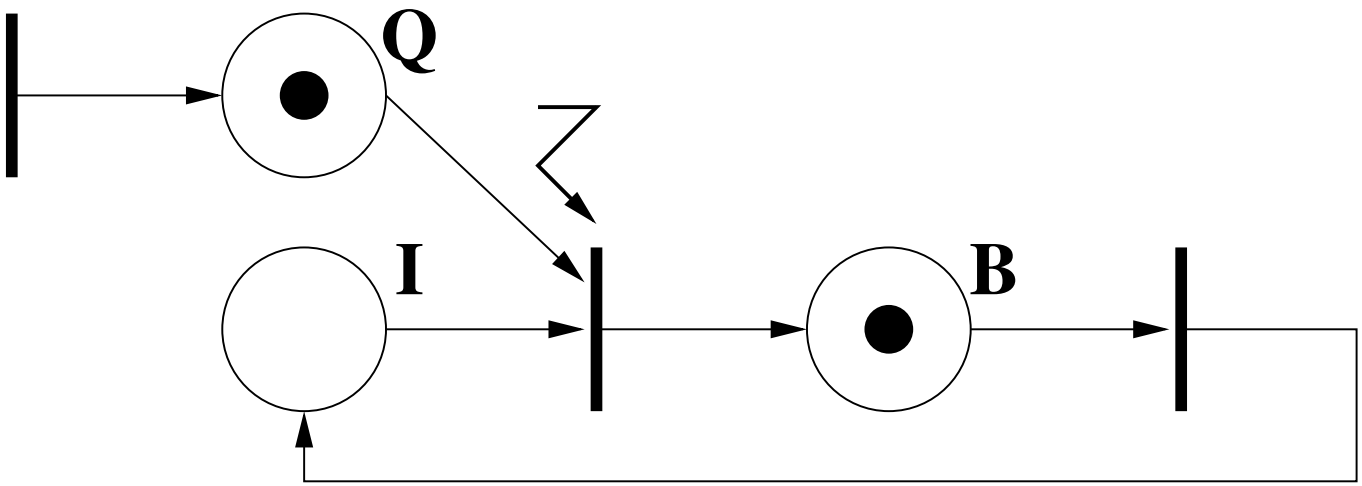
Example of execution (4)



Example of execution (5)



Example of execution (6)

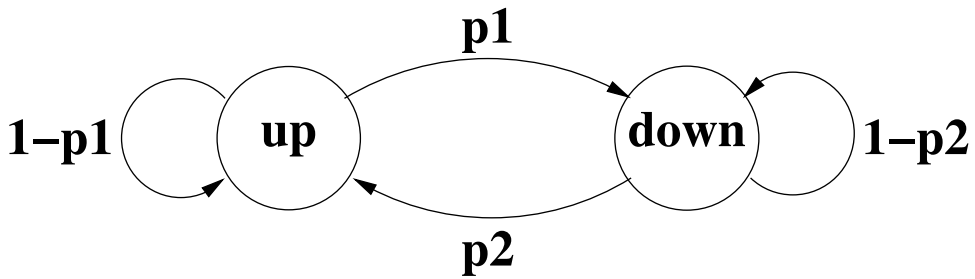


(Discrete space and time) Markov chains

$\mathbf{C} = \langle \mathbf{S}, \mathbf{T}, \mathbf{I} \rangle$

- finite set $\mathbf{S}$ of possible **states**;
- **transition** probability distribution $\mathbf{T} : \mathbf{S} \times \mathbf{S} \to [0; 1]$
- **initial state** probability distribution $\mathbf{I} : \mathbf{S} \to [0; 1]$

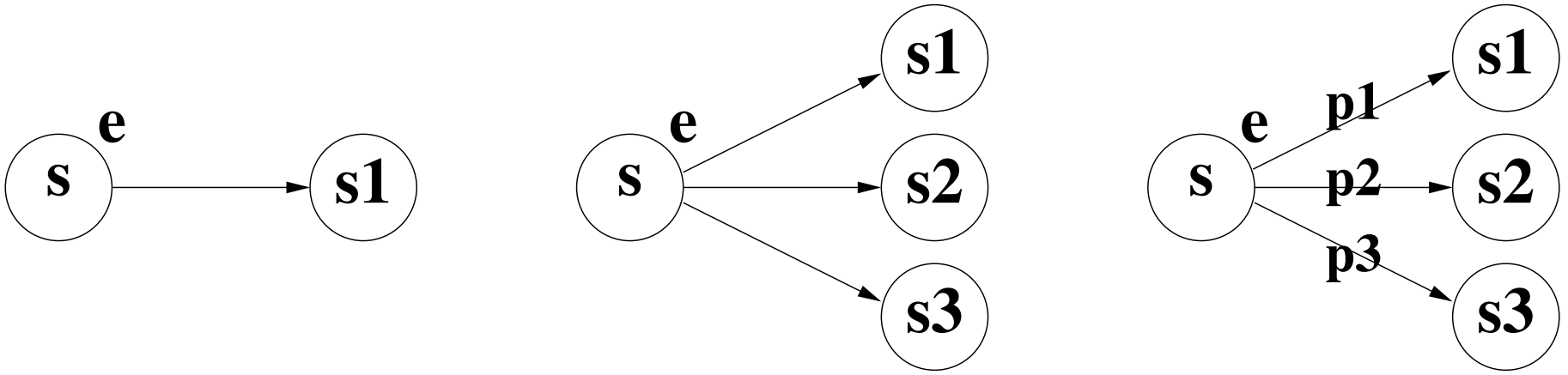
Example: possible **machine breakdown**.



$\to$ **Directed weighted graph**.

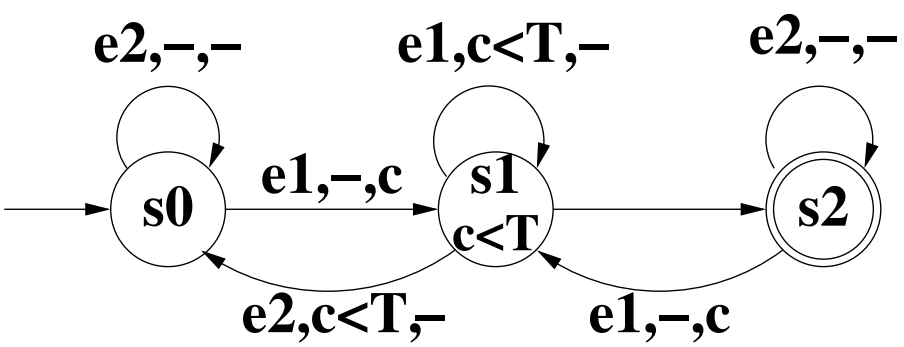
Variants of automata

- **deterministic** automata: $\mathbf{T} : \mathbf{S} \times \mathbf{E} \rightarrow \mathbf{S}$; $\mathbf{I} \in \mathbf{S}$
- **non deterministic** automata: $\mathbf{T} : \mathbf{S} \times \mathbf{E} \times \mathbf{S} \rightarrow \{0, 1\}$; $\mathbf{I} : \mathbf{S} \rightarrow \{0, 1\}$
- **stochastic** automata: $\mathbf{T} : \mathbf{S} \times \mathbf{E} \times \mathbf{S} \rightarrow [0; 1]$; $\mathbf{I} : \mathbf{S} \rightarrow [0; 1]$

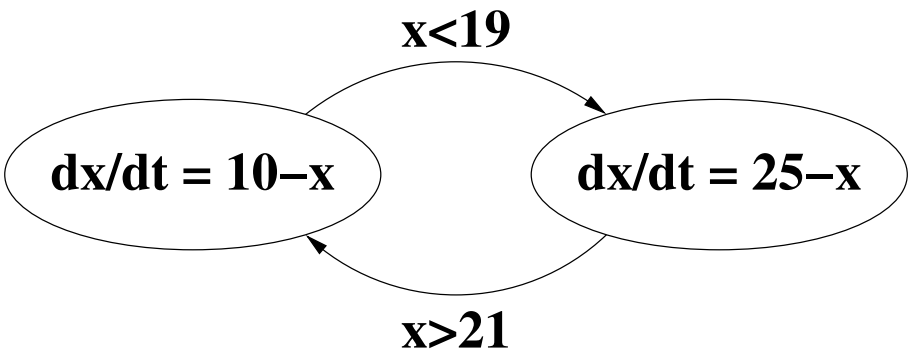


Variants and extensions

- **controllable** and **uncontrollable** events $\rightarrow$ **controlled automata**, **game automata**;
- **observable** and **unobservable** state variables $\rightarrow$ **partial observability**.
- **timed** events $\rightarrow$ **timed automata**;
  ex: **alarm** emitted if e_1 occurs and e_2 does not occur after T seconds;
- **continuous** state evolutions between events $\rightarrow$ **hybrid automata**;
  ex: **thermostat** which maintains the temperature between 19 and $21°$C.



timed automaton



hybrid automaton

Representation issues

Even for small DEDS, **no explicit representation** of the transition graph:
no drawing and no explicit representation in computer memory.

Factorized representation of states and events via **state** and **event variables**.

$\rightarrow$ **Constraints** can be used to represent:

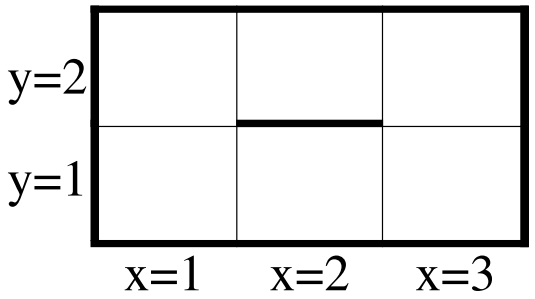
- **event** preconditions and effects;
- initial and marked **states**;
- reachability and safety **properties**.

Warning: the size of the state (event) space is an **exponential** function of the number of state (event) variables.

Example: a robot moving on a grid

Six **state variables**: x,y,w_N,w_S,w_E, and w_W with $\mathbf{d}(x) = [\![1;3]\!]$,$\mathbf{d}(y) = [\![1;2]\!]$, and $\mathbf{d}(w_\_) = \mathbb{B}$

One **event variable** m with $\mathbf{d}(m) = \{N, S, E, W\}$



Event preconditions and effects:

$$w_N \rightarrow (m \neq N),\ w_S \rightarrow (m \neq S),\ w_E \rightarrow (m \neq E),\ w_W \rightarrow (m \neq W)$$
$$x' = x + (m = E) - (m = W),\ y' = y + (m = N) - (m = S)$$
$$w'_N = (y' = 2 \vee (y' = 1 \wedge x' = 2)),\ w'_S = (y' = 1 \vee (y' = 2 \wedge x' = 2))$$
$$w'_E = (x' = 3),\ w'_W = (x' = 1)$$

Safety property: $\neg((x' = 3) \wedge (y' = 1))$

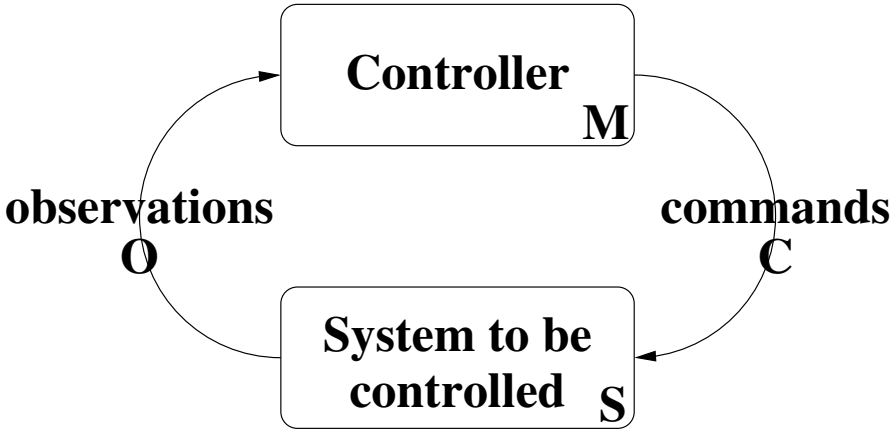
Reachability property: $x' = 1$

2. DEDS control problems

Control

These systems must behave **properly** and satisfy some **properties**.

$\rightarrow$ Need for **closed loop control** to be sure that these properties be satisfied.



At any step, the system is in some state **S**, the controller is in some state **M** (**controller memory**) and it **reactively**:

- performs **observations O**
- emits **commands C**, according to its policy π : **O** $\times$ **M** $\rightarrow$ **C**
- updates its own state, according to its updating function u : **O** $\times$ **M** $\times$ **C** $\rightarrow$ **M**

And the system moves to **another state S'**.

Properties

- **Reachability**: the system must **at some step** reach a state that satisfies some properties;
  Example: $x' = 1$

- **Safety**: **at each step**, the state must satisfy some properties;
  Example: $\neg((x' = 3) \wedge (y' = 1))$

- more generally any constraint (to be satisfied) or criterion (to be optimized) on state trajectories:
  - **Satisfaction**: **temporal logics**: propositional logics extended with **temporal operators** ($\bigcirc$ for "Next", $\Box$ for "Always", and $\Diamond$ for "Eventually")
    Example: any request will be eventually satisfied: $\Box(A \rightarrow \Diamond B)$
  - **Optimization**: **expected additive utility** (local utility associated with any transition).

Three ways of building a controller

1. to **build** it "**manually**", using any programming language, for example any language of the "**synchronous**" family, and **verify** *a posteriori* its properties via tests, simulations, or formal verification tools (for example *via* **model-checking**); to **repair** it in case of negative result;

2. to **synthesize** it **automatically** by guaranteeing *a priori* its properties: **controller synthesis** or **planning** in its widest sense;

3. to **learn** it iteratively from simulations: **machine learning**.

Focus on the second option.

3. Planning problems

Three problems considered

Focus on three planning problems:

1. **deterministic** automata + **reachability** properties:
   Classical **Artificial Intelligence (AI) Planning**
   or planning in deterministic domains (plan synthesis);

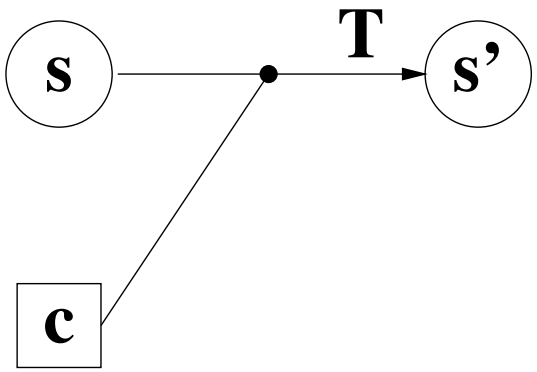
2. **non deterministic** automata + **reachability/safety** properties:

   (a) **Model-checking** when the policy is fixed (policy verification);
   (b) **Logical Markov Decision Processes (MDP)** when it is not fixed
   or planning in non deterministic domains (policy synthesis);

3. **stochastic** automata + **expected utility**:
   Classical **MDP**
   or planning in stochastic domains (policy synthesis).

Artificial Intelligence (AI) Planning

Deterministic controlled automata.



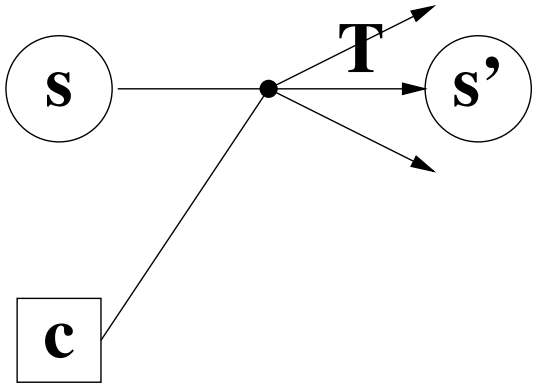
$\mathbf{P} = \langle \mathbf{S}, \mathbf{E}, \mathbf{T}, \mathbf{I}, \mathbf{G} \rangle$

- finite set **S** of possible **states**;
- finite set **C** of possible **commands**;
- **transition** function $\mathbf{T} : \mathbf{S} \times \mathbf{C} \rightarrow \mathbf{S}$
- **initial** state $\mathbf{I} \in \mathbf{S}$
- **goal** states $\mathbf{G} : \mathbf{S} \rightarrow \{0, 1\}$

Objective: to produce a **plan** *i.e*, a sequence of commands, that allows a **goal state** to be reached.

Logical Markov Decision Processes (MDP)

Non deterministic controlled automata.



$\mathbf{M} = \langle \mathbf{S}, \mathbf{E}, \mathbf{T}, \mathbf{I}, \mathbf{P} \rangle$

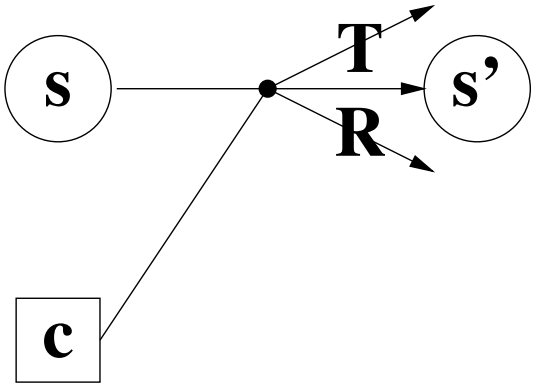
- finite set **S** of possible **states**;
- finite set **C** of possible **commands**;
- possible **transitions** $\mathbf{T} : \mathbf{S} \times \mathbf{C} \times \mathbf{S} \rightarrow \{0, 1\}$
- possible **initial** states $\mathbf{I} : \mathbf{S} \rightarrow \{0, 1\}$
- **property** $\mathbf{P} : \mathbf{S} \rightarrow \{0, 1\}$

Objective: to produce a **policy** *i.e*, a function from **S** to **C**, that guarantees that **property P** be satisfied at some step (**reachability**) or at each state (**safety**).

Markov Decision Processes (MDP)

Stochastic controlled automata.

$\mathbf{M} = \langle \mathbf{S}, \mathbf{E}, \mathbf{T}, \mathbf{I}, \mathbf{R}, \gamma \rangle$



- finite set **S** of possible **states**;
- finite set **C** of possible **commands**;
- **transition** probability distribution $\mathbf{T} : \mathbf{S} \times \mathbf{C} \times \mathbf{S} \rightarrow [0; 1]$
- **initial** state probability distribution $\mathbf{I} : \mathbf{S} \rightarrow [0; 1]$
- transition **reward** function $\mathbf{R} : \mathbf{S} \times \mathbf{C} \times \mathbf{S} \rightarrow \mathbb{R}$
- **discount** factor $\gamma : 0 < \gamma < 1$

Objective: to produce a **policy** π *i.e*, a function from **S** to **C**, that maximizes the **expected discounted total reward** over an **infinite** horizon:
$\mathbb{E}(\sum_{i=0}^{+\infty} \gamma^i \cdot \mathbf{R}(s_i, \pi(s_i), s_{i+1}))$

Four ways of using CP

Main **modeling difficulty**: the **number of steps** to be considered is **unknown**.

However, CP can be used.

1. to model and solve the **optimality/satisfiability equations**;
2. to model and solve **problems unfolded over a finite number of steps**;
3. to **check** or **propagate constraints**, to **compute bounds** at each node of a classical tree search;
4. to model and solve the **transition relation**.

1-Modeling/solving optimality/satisfiability equations

Classical MDP.

Optimality Bellman equations, with $V^*(s)$ being the maximum expected utility it is possible to get from state s and $0 < \gamma < 1$ (discount factor):

$$\forall s \in \mathbf{S}: \quad V^*(s) = \max_{c \in \mathbf{C}} [\sum_{s' \in \mathbf{S}} \mathbf{T}(s,c,s') \cdot [\mathbf{R}(s,c,s') + \gamma \cdot V^*(s')]]$$

$$\pi^*(s) = \arg\max_{c \in \mathbf{C}} [\sum_{s' \in \mathbf{S}} \mathbf{T}(s,c,s') \cdot [\mathbf{R}(s,c,s') + \gamma \cdot V^*(s')]]$$

Linear programming model, with $V^*(s)$ as **variables**:

Minimize $\sum_{s \in \mathbf{S}} V^*(s)$ **subject to**

$$\forall s \in \mathbf{S}, \forall c \in \mathbf{C}: V^*(s) \geq [\sum_{s' \in \mathbf{S}} \mathbf{T}(s,c,s') \cdot [\mathbf{R}(s,c,s') + \gamma \cdot V^*(s')]]$$

1-Modeling/solving optimality/satisfiability equations

Logical MDP with **safety** requirements **P** on transitions.

Satisfiability equations, with $\pi(s)$ being the policy and $r_\pi(s)$ the reachability associated with state s:

$$\begin{aligned}
\forall s \in \mathbf{S}, \quad & r_\pi(s) \rightarrow \pi(s) \neq \bot \\
& r_\pi(s) \rightarrow (\exists s' \in \mathbf{S},\ \mathbf{T}(s, \pi(s), s')) \\
& r_\pi(s) \rightarrow (\forall s' \in \mathbf{S},\ \mathbf{T}(s, \pi(s), s')) \rightarrow \mathbf{P}(s, \pi(s), s'))
\end{aligned}$$

Constraint programming model, with $\pi(s)$ and $wr_\pi(s)$ as **variables**:

$$\begin{aligned}
\forall s \in \mathbf{S}, \quad & \mathbf{I}(s) \rightarrow wr_\pi(s) \\
\forall s, s' \in \mathbf{S}, \quad & (wr_\pi(s) \wedge \mathbf{T}(s, \pi(s), s')) \rightarrow wr_\pi(s') \\
\forall s \in \mathbf{S}, \quad & wr_\pi(s) \rightarrow (\pi(s) \neq \bot) \\
& wr_\pi(s) \rightarrow (\exists s' \in \mathbf{S},\ \mathbf{T}(s, \pi(s), s')) \\
& wr_\pi(s) \rightarrow (\forall s' \in \mathbf{S},\ \mathbf{T}(s, \pi(s), s') \rightarrow \mathbf{P}(s, \pi(s), s'))
\end{aligned}$$

Constraint programming for Controller Synthesis, Verfaillie and Pralet, CP 2011

1-Modeling/solving optimality/satisfiability equations

Main **strength**: optimality/satisfiability equations are **directly** expressed and solved.

Main **weakness**: need for one or two LP/CP variables **per state**.

$\rightarrow$ Number of LP/CP variables **exponential** function of the number of state variables.

$\rightarrow$ **Extremely large** LP/CP models.

$\rightarrow$ Limited to systems with **only some state variables**.

2-Modeling/solving problems over a finite number of steps

Classical MDP $\rightarrow$ **Finite horizon MDP**.

Optimality Bellman equations, with $V_i^*(s)$ being the maximum expected utility it is possible to get from state s at step i:

$$\forall s \in \mathbf{S}, \forall i < \mathbf{k}: \quad V_i^*(s) = \max_{c \in \mathbf{C}} [\sum_{s' \in \mathbf{S}} \mathbf{T}(s,c,s') \cdot [\mathbf{R}(s,c,s') + V_{i+1}^*(s')]]$$

$$\forall s \in \mathbf{S}: \quad V_{\mathbf{k}}^*(s) = \mathbf{Vf}(s)$$

$\mathbf{Vf}(s)$ can be seen as an **estimation** of the maximum expected utility it is possible to get from state s beyond the horizon limit.

2-Modeling/solving problems over a finite number of steps

Model-checking $\rightarrow$ **Bounded Model-checking**.

Classical model-checking techniques may quickly reach their **limits**:

- too much **time** necessary with classical **forward tree search**;
- too much **memory** required with **symbolic model-checking** techniques.

Alternative:

- to **limit** reasoning to a finite number of steps;
- to model the verification problem over a finite horizon as a **SAT problem** with the **state** and **event** at each step as **variables**;
- to use generic **SAT solvers** and to **increment** the horizon step by step.

Incomplete approach, but may allow **reachability** properties to be verified and violation of **safety** properties to be detected.

Bounded Model Checking, Biere, Cimatti, Clarke, Strichman, and Zhu, Advances in Computers, 2003

CPBPV: a Constraint-Programming framework for Bounded Program Verification, Collavizza, Rueher, and Van Hentenryck, Constraints, 2010

2-Modeling/solving problems over a finite number of steps

AI planning $\rightarrow$ **SAT** or **CSP-based planning**.

Approach similar to **Bounded Model-checking**.

Basic model:

- an action implies its **preconditions** and **effects**:
  $\forall i < \mathbf{k} : (m^i = N) \rightarrow ((w^i_N = 0) \wedge (y^{i+1} = y^i + 1))$
- a state variable remains **unchanged** unless an action changes it:
  $\forall i < \mathbf{k} : (y^{i+1} \neq y^i) \rightarrow (m^i \in \{N, S\})$
- **initial** state at step 0 and **goal** conditions at step $\mathbf{k}$: $(x_0 = 3) \wedge (y_0 = 2)$ and $(x_{\mathbf{k}} = 1)$

Other more or less efficient **models**.

Question:

$$\exists c_1, s_1, c_2, s_2 : \mathbf{T}(s_0, c_1, s_1) \wedge \mathbf{T}(s_1, c_2, s_2) \wedge \mathbf{G}(s_2)$$

Planning as Satisfiability, Kautz and Selman, ECAI, 1992

CPlan: A Constraint Programming Approach to Planning, Van Beek and Chen, AAAI, 1999

2-Modeling/solving problems over a finite number of steps

AI planning $\rightarrow$ **SAT** or **CSP-based planning**.

Ability to use the so-called **planning graph**, which can be seen as the result of a planning problem compilation over a finite number of steps.

Planning as Constraint Satisfaction: Solving the Planning-Graph by Compiling it into CSP,
Do and Kambhampati, Artificial Intelligence Journal, 2001

Very close approach: to start from a **finite set** of **candidate events**.

Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming,
Vidal and Geffner, Artificial Intelligence Journal, 2006

A Timeline, Event, and Constraint-based Modeling Framework for Planning and Scheduling Problems,
Verfaillie and Pralet, KEPS ICAPS Workshop, 2013

2-Modeling/solving problems over a finite number of steps

Logical MDP $\rightarrow$ **QBF** or **QCSP-based planning**.

Approach similar to **Bounded Model-checking** and **SAT/CSP-based planning**.

With **Model-checking** and **AI Planning**, we have only **one actor** (the environment or the controller) $\rightarrow$ **SAT/CSP** models.

With **Logical MDP**, we have **two actors** (the environment and the controller) and we need to model commands as existentially quantified variables and states as universally quantified variables $\rightarrow$ **QBF/QCSP** models.

Question:

$$\forall s_0, \exists c_1, \forall s_1, \exists c_2, \forall s_2 : (\mathbf{I}(s_0) \wedge \mathbf{T}(s_0, c_1, s_1) \wedge \mathbf{T}(s_1, c_2, s_2)) \rightarrow \mathbf{G}(s_2)$$

2-Modeling/solving problems over a finite number of steps

Strengths:

- SAT/CSP/QBF/QCSP models solved using **generic solvers**;
- allows **non Markovian** constraints to be expressed.

Weaknesses:

- **incomplete** approach due to a limited horizon;
- need for one SAT/CSP variable per state/event variable at each step
  $\rightarrow$ may lead to **very large** SAT/CSP models, as the number of steps increases.

3-Reasoning at each node of a classical tree search

To use **tree search mechanisms**, classical in planning.

To use CP to check and propagate **constraints** and to compute **bounds** at each node of the search.

Examples:

- a **POCL** (Partial Order Causal Link) tree search which explores the space of partial plans and adds actions in a non chronological way + reasoning on action **preconditions** and **effects** and on **temporal** and **resource** constraints;

  IxTeT (LAAS-CNRS) and Europa (NASA Ames) planning tools, which combine planning and scheduling

- a **forward chronological** tree search + so-called "**observers**" which accumulate constraints and may detect inconsistent trajectories.

  Combining Static and Dynamic Models for Boosting Forward Planning,
  Pralet and Verfaillie, CPAIOR, 2012

4-Modeling/solving the transition relation

To use CP:

- to model the **transition relation** and the safety/reachability **properties**;
- to compute quickly the **set of actions** applicable to a given state or the **set of states** resulting from the application of a given action to a given state; to quickly detect inconsistency.

Minimal approach.

Constraint-based Controller Synthesis in Non-Deterministic and Partially Observable Domains, Pralet, Verfaillie, Lemaître, and Infantes, ECAI, 2010

4. Scheduling problems

Planning vs. Scheduling

Informally and roughly speaking:

1. **Planning**: **What to do**? Which actions?

2. **Scheduling**: **How to do** with time and resources?

In terms of models:

1. **Planning**: states, events, and transitions $\rightarrow$ **state trajectories**;

2. **Scheduling**: tasks, time, and resources $\rightarrow$ **task schedules**
   (**no** explicit notion of **state**);

3. **SAT/CP/LP**: variables, constraints, and criteria $\rightarrow$ **variable assignments**.

Constraint Programming for Scheduling

Constraint Programming **widely used** to model and solve scheduling problems.

Example: **Resource-constrained Project Scheduling Problem**.

Naive CP model:

Minimize $\max_{t \in \mathbf{T}} e(t)$ **subject to**

$$\begin{aligned}
\forall t \in \mathbf{T}: & \quad e(t) = s(t) + \mathbf{du}(t) \\
\forall t, t' \in \mathbf{T} | \mathbf{pr}(t, t') = 1: & \quad e(t) \leq s(t') \\
\forall t \in \mathbf{T}, \forall r \in \mathbf{R} | \mathbf{c}(t, r) > 0: & \quad [\sum_{t' \in \mathbf{T}} [(s(t') \leq s(t) < e(t')) \cdot \mathbf{c}(t', r)]] \leq \mathbf{cmax}(r)
\end{aligned}$$

More compact and **efficient** model using the **global constraint** **cumulative**$(\mathbf{T}, r)$.

Weakness: time **discretization**.

Simple Temporal Networks (STN)

A **very useful** framework:

- **real** variables;
- **distance** constraints of the form $x - y \in [a; b]$.

Strengths:

- deciding upon **consistency/inconsistency** of an STN and computing **minimum and maximum values** for each variable are polynomial tasks;
- no need for time **discretization**.

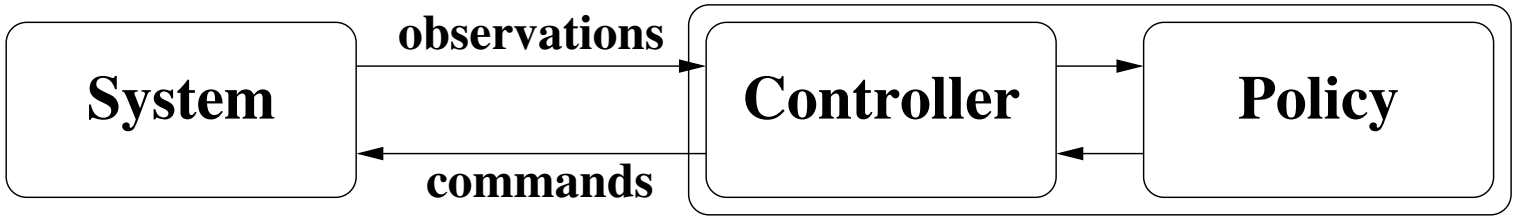
Temporal Constraint Networks,Dechter, Meiri, and Pearl, Artificial Intelligence Journal, 1991

Weakness: do not model **disjunctive** problems ($x - y \in [a; b] \cup [c; d]$) or require that all disjunctions be fixed
$\rightarrow$ **Disjunctive Temporal Networks** when disjunctions are present or remain.

5. Real world problems

Classical approach

Classical approach to **optimal control under uncertainty**.



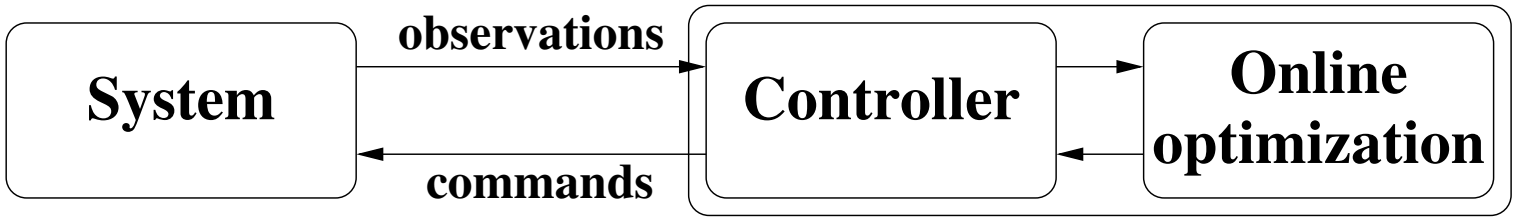
The control policy is **synthesized offline** and then **applied online**.

Difficulties:

- the model of uncertainty is **rarely available**; for example, model of request arrivals or of system failures;
- even if it is available, the synthesis of an optimal policy may be **computationally infeasible**, in terms of time and memory.

Model Predictive Control (MPC)

Alternative approach coming from **Automatic Control** (control of continuous time dynamic systems).



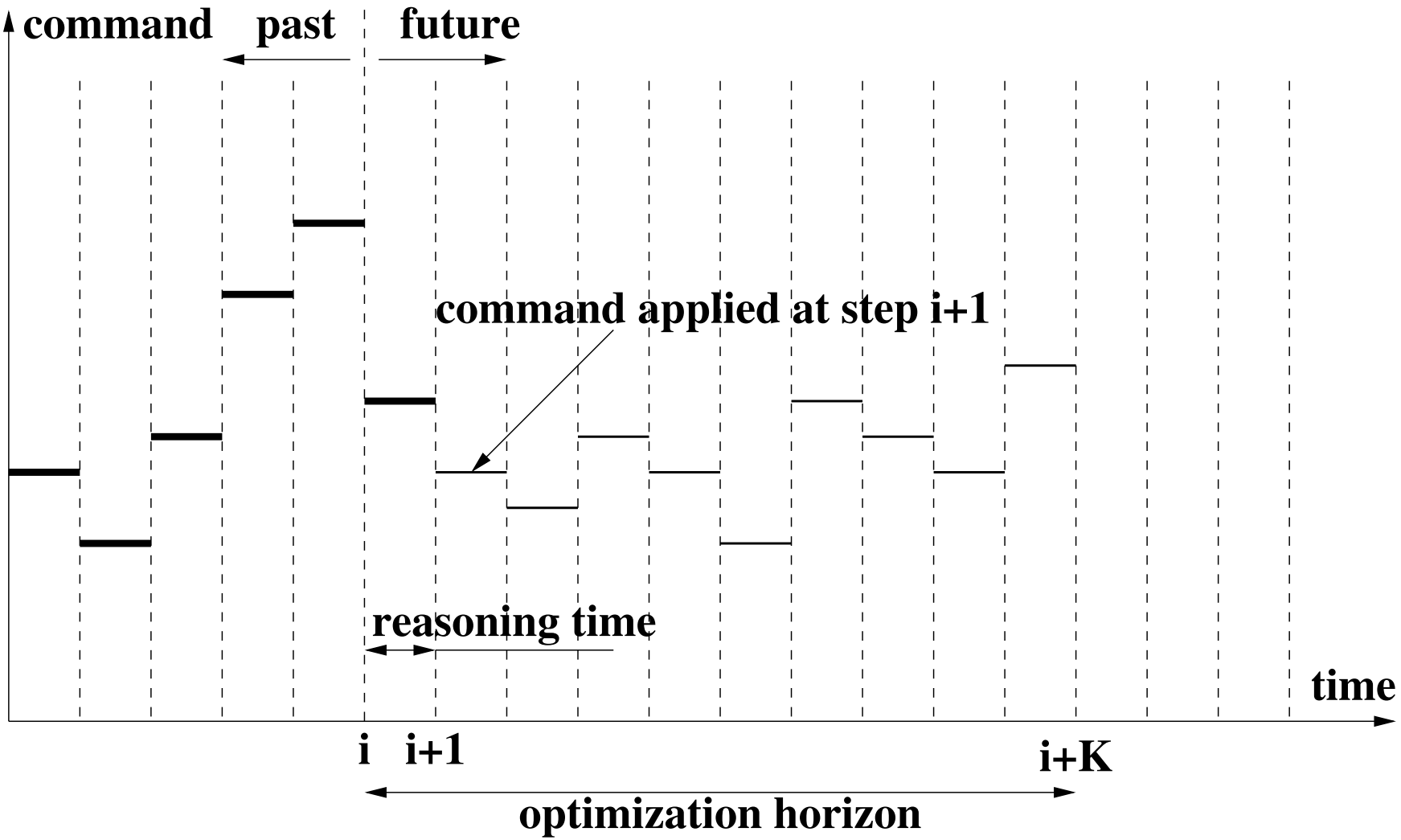
At each step, an **open-loop** optimal control problem over a **finite horizon** is solved **online**, using a **deterministic** model and the **current state** as an initial state.

Only the **first command** in the solution is **applied**.

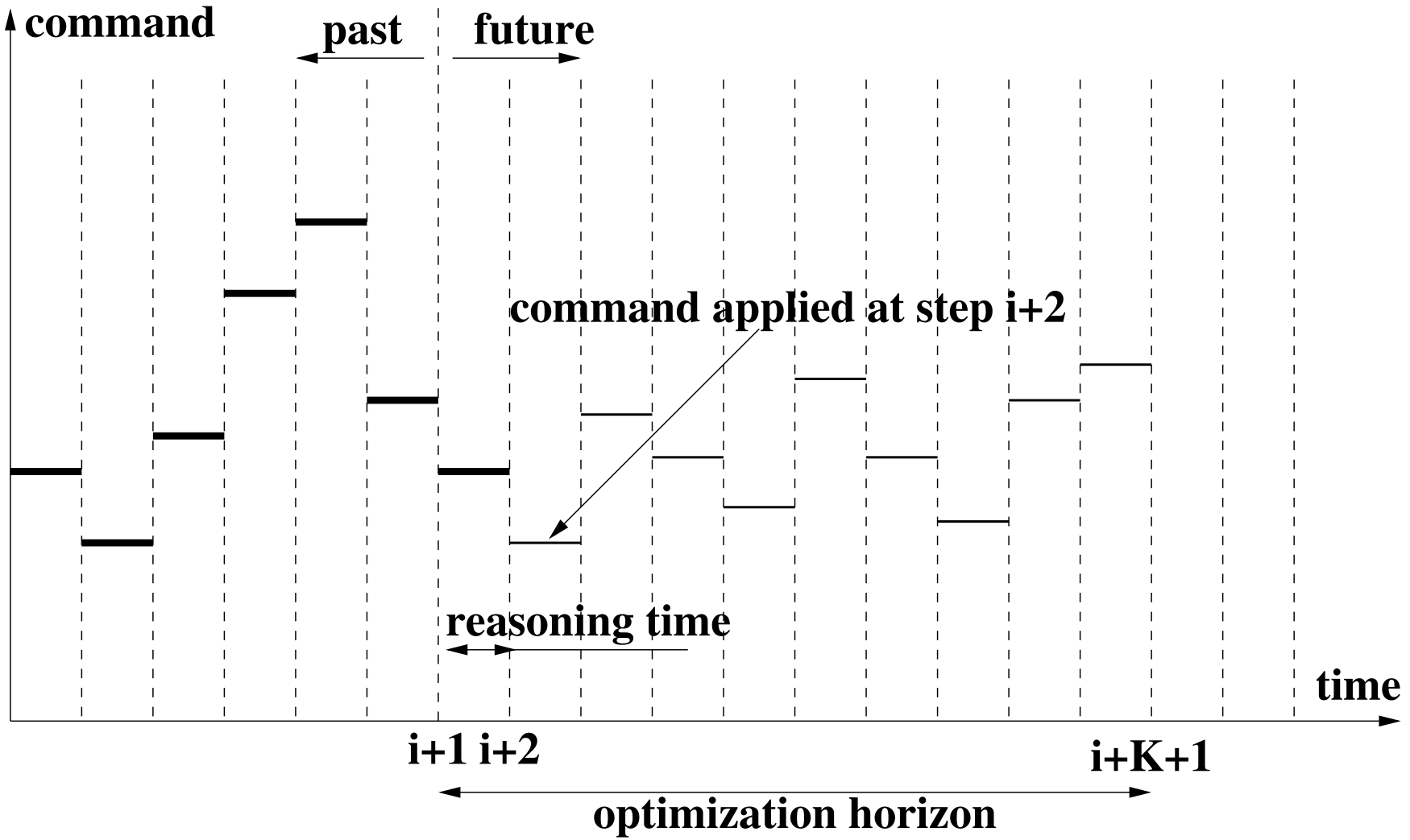
The operation is repeated at the next step over a **receding horizon**, starting from the state that has been actually reached (**feedback**).

Model Predictive Control: Theory and Practice–A Survey, Garcia, Prett, and Morari, Automatica, 1989

MPC in details (1)

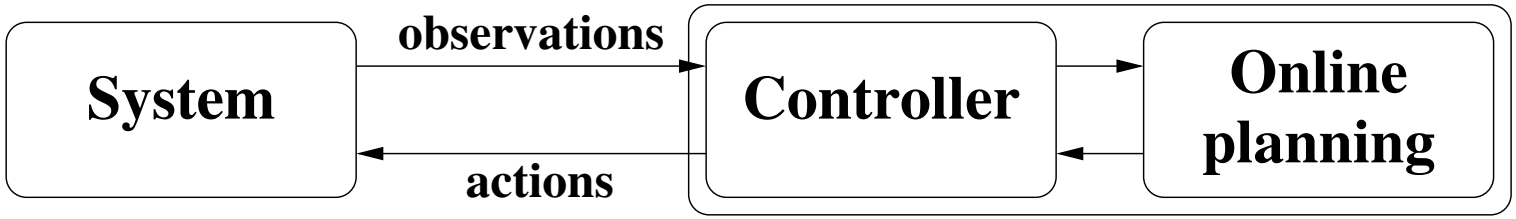


MPC in details (2)



Application of MPC to the control of DEDS

Can be easily extended to the control of DEDS.



Each time an event occurs, a **deterministic** planning problem over a **finite horizon** is solved **online**, using the **current state** as an initial state.

Only the **first action** in the plan is **applied**.

The operation is repeated at the next event occurrence over a **receding horizon**, starting from the state that has been actually reached (**feedback**).

Interaction between Reactive and Deliberative Tasks for On-line Decision-making, Lemaître and Verfaillie, ICAPS Workshops 2007

Key parameters when using MPC

- **length** of the planning horizon;
- **time** available for reasoning;
- available **computing resources**, in case of embedded control.

$\rightarrow$ **greedy** or **local search** algorithms.

$\rightarrow$ default **decision rules** in case no plan could be built.

$\rightarrow$ **Constraint-based Anytime Local Search** algorithms.

Strong connections with **Online Stochastic Optimization** which extends this approach by exploiting several **samples** of the future.

Online Stochastic Combinatorial Optimization, Van Hentenryck and Bent, MIT Press, 2006

Planning problems to be solved

From our experience in the aerospace domain, most of the planning problems to be solved using this MPC approach are in fact **complex scheduling-like problems** with:

- a finite set of **events** to be considered;
- various **temporal** and **resource** constraints;
- (sometimes) **state** constraints;
- a complex **criterion** to be optimized.

Neither classical **OR scheduling**, nor **AI planning** problems.

$\rightarrow$ Importance of the **flexibility** of CP.

Basic references

Basic references (1)

Discrete event dynamic systems:
Introduction to Discrete Event Systems, Cassandras and Lafortune, Second edition, Springer, 2008.

Model checking:
Model Checking, Clarke, Grumberg, and Peled, MIT Press, 1999.

Artificial intelligence planning:
Automated Planning: Theory and Practice, Ghallab, Nau and Traverso, Morgan Kaufmann, 2004.

Action and motion planning:
Planning Algorithms, LaValle, Cambridge University Press, 2006.

Operations research scheduling:
Scheduling: Theory, Algorithms, and Systems, Pinedo, Fourth edition, Springer, 2012.

Basic references (2)

Constraint-based scheduling:
Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems, Baptiste, Le Pape, and Nuijten, Kluwer Academic Publishers, 2001.

Markov decision processes:
Markov Decision Processes, Discrete Stochastic Dynamic Programming, Puterman, John Wiley & Sons, 1994.

Constraint programming:
Handbook of Constraint Programming, Rossi, Van Beek, and Walsh, Elsevier, 2006.

Local search:
Local Search in Combinatorial Optimization, Aarts and Lenstra, Princeton University Press, 2003.

Constraint-based local search:
Constraint-based Local Search, Van Hentenryck and Michel, MIT Press, 2005.