Discrete Event Systems, Languages and Automata

From observability to privacy and security in discrete event systems

Prof. Gianmaria De Tommasi
Email: detommas@unina.it

December 2020

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Outline

Context

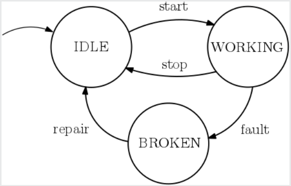
- Study of dynamic systems modelled as Discrete Event Systems (DES)
  - nonlinear. . .
  - . . .with discrete state space. . .
  - . . .whose dynamic is driven by the occurrence of *asynchronous* events over time
- *Uncertain* DES, where the main source of uncertainty is due to the occurrence of *unobservable events*
- This framework can be used to study fault-detection and secrecy problems when the system of interest can be modelled as a logical DES

Formal languages

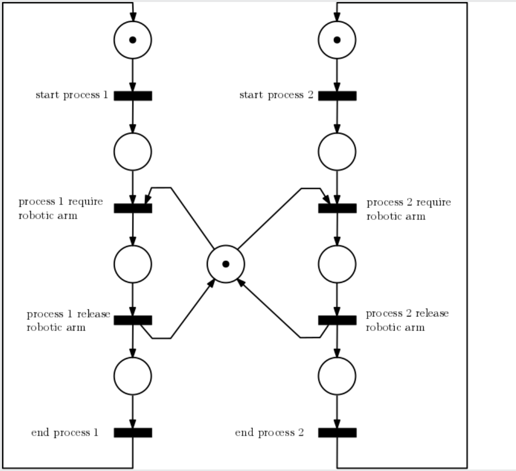
A logical DES can be seen as a formal language *generator*

- The events that drive the system dynamic can be regarded as *letters of an alphabet E*
- The system trajectories become *words* (*strings*, *sequences*)
- The system itself can be regarded as a generator of words $\rightarrow$ a *generator* (*recognizer*) of a *formal language*
- Different tools can be used to model DES at the logical level: queue systems, look-up-tables, automata, Petri nets
- Some of this tools can be also extended to study *timed* DES: timed automata and timed Petri nets, Markov chains, $(\max, +)$ algebra,...

Automata



Petri nets



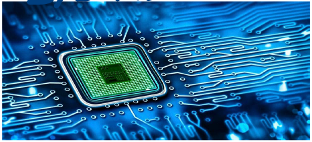
Examples

FLEXIBLE MANUFACTURING SYSTEMS (FMS)

TRAFFIC SYSTEMS

COMMUNICATION PROTOCOLS

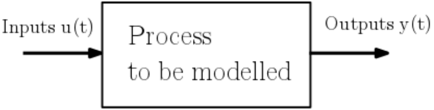
LARGE SCALE SYSTEMS



Different levels of abstraction when studying dynamical systems

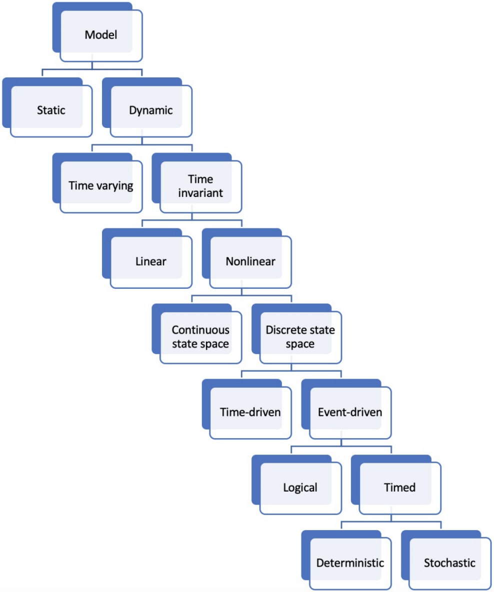
- There are analysis and synthesis tasks that cannot be practically performed when dealing with large scale/complex systems, if these are modelled using differential equations (ODEs)

$$\dot{x}(t) = f\left(x(t), u(t), t\right),$$
$$y(t) = g\left(x(t), u(t), t\right).$$

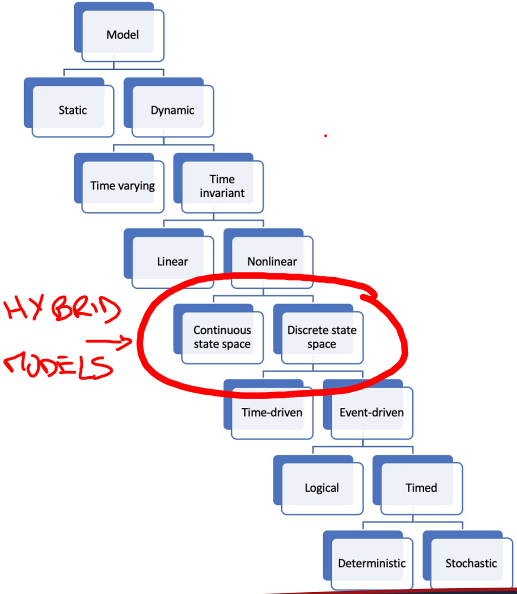


- The DES framework permits to move to a higher level of abstraction, where (some) physical details can be neglected
- When this is not possible some hybrid approaches are possible (both for modelling and control)

Bestiarium of dynamical systems



Bestiarium of dynamical systems



The DES research community

- Researchers in this field have different backgrounds: computer science, information theory, operations research, control & automation
- **Most of the concepts originated in the computer science community (some date back to Turing!)**
- These concepts have been brought in the control community in the 80's by Ramadge and Wonham (**Supervisory Control Theory, SCT**)
- Even earlier, in the mid 70's, Petri nets were used to derive the Grafcet programming language, which is used in PLCs (nowadays known as SFC)
- The *jargon* adopted in this course is the one usually adopted by the *automation-oriented* researchers, as well as most of the reported results have been published on control and automation journals

W. M. Wonham, K. Cai, K. Rudie
Supervisory control of discrete-event systems: A brief history
Annual Reviews in Control, 2018

Course syllabus

1. Discrete Event Systems (DES), Languages and Automata (**this lesson**)
2. Petri nets (PNs) and their twofold representation to model DES
3. MILP and ILP formulations: logical conditions, binary variables “do everything", and variable connecting (**prof. Claudio Sterle**)
4. Adding uncertainty: unobservable events and observers for finite state automata and PNs
5. Augmenting the observers: diagnosability of prefix-closed languages, diagnosers and the fault detection for finite state automata (**prof. Francesco Basile**)
6. Diagnosability and fault detection in PNs - Part I: graph-based approaches (**prof. Francesco Basile**)
7. Diagnosability and fault detection in PNs - Part II: algebraic approaches for bounded systems
8. Security issues in DES: non-interference and opacity
9. Non-interference and opacity enforcement
10. Open issues

C. G. Cassandras and S. Lafortune,
Introduction to Discrete Event Systems
Springer, 2008

C. Seatzu, M. Silva, J. H. van Schuppen (eds.),
Control of Discrete-Event Systems
Springer, 2013

Some papers :)

Now we can start!

1. **Discrete Event Systems (DES), Languages and Automata**
2. Petri nets (PNs) and their twofold representation to model DES
3. MILP and ILP formulations: logical conditions, binary variables “do everything", and variable connecting
4. Adding uncertainty: unobservable events and observers for finite state automata and PNs
5. Augmenting the observers: diagnosability of prefix-closed languages, diagnosers and the fault detection for finite state automata
6. Diagnosability and fault detection in PNs - Part I: graph-based approaches
7. Diagnosability and fault detection in PNs - Part II: algebraic approaches for bounded systems
8. Security issues in DES: non-interference and opacity
9. Non-interference and opacity enforcement
10. Open issues

Alphabet and words

- Given a DES, the events that may occur can be seen as elements (*symbols*) of an **alphabet** E set

$$E = \{a, b, c, \dots\} ,$$

  where the symbols $a, b, c, \dots$ are used to denotes events
- A **word** (**string, trace**) w is a sequence of event of **finite length**
  Example: $w = e^1 e^2 e^3 = aab$
- $|w|$ denotes the length of a word is denoted (some authors use $\|w\|$)
- ε denotes the empty word or silent event, i.e. $|\varepsilon| = 0$

- A language L defined over an alphabet E is a set of words defined on the symbols of E
- **The cardinality of a language L can be either finite or infinite**
- Being $E = \{a, b, c\}$
  - $L_1 = \{\varepsilon\,, a\,, abb\}$
  - $L_2 = \{\text{all words that starts with the event } a\}$
  - $L_3 = \{\varepsilon\,, b\,, b\,, bab\}$

- The key operation among words is the **concatenation**
- The concatenation of two words w_1 and w_2 is the new string w consisting of the events in w_1 immediately followed by the events in w_2, and it is denoted $w = w_1 w_2$
- In general, if $u = w_1 w_2$ and $v = w_2 w_1$ it does not necessarily follows that $u = v \rightarrow$ **concatenation is not commutative**
- The empty word ε is the identity element of concatenation, i.e. $w\varepsilon = \varepsilon w = w$
- Given a word $w = tuv$ the following terminology is adopted
  - t is called **prefix** of w
  - u is called **substring** of w
  - v is called **suffix** of w

- The **Kleene closure** of an alphabet E is the set of all the finite-length words defined on the elements of E and is denoted with E^*
- The empty word ε is always contained in E^*
- Example:
  - $E = \{\alpha, \beta\}$
  - $E^* = \{\varepsilon\,, \alpha\,, \beta\,, \alpha\alpha\,, \alpha\beta\,, \beta\alpha\,, \beta\beta\,, \alpha\alpha\alpha\,, \ldots\}$
- E^* contains every possible language L defined on the symbols of E

Set operations

- Being sets, all the set operations are defined also on languages:
  - **union** $L_1 \cup L_2$
  - **intersection** $L_1 \cap L_2$
  - **difference** $L_1 \setminus L_2$
  - **complement with respect to** E^* $E^* \setminus L$
- Language specific operations are
  - Concatenation (of languages)
  - Prefix-closure
  - Kleen-closure

Given two languages $L_a\,,L_b \subseteq E^*$, the concatenation L_aL_b is

$$L_aL_b := \{w \in E^* \;:\; w = w_aw_b \text{ with } w_a \in L_a\,, w_b \in L_b\}$$

Prefix-closure

- Let $L \subseteq E^*$, its prefix-closure is

$$\bar{L} := \{w \in E^* \ : \ \exists\, t \in E^* \text{ such that } wt \in L\}$$

- It is always $L \subseteq \bar{L}$
- L is said to be **prefix-closed** if $L = \bar{L}$

Kleene-closure

- Let $L \subseteq E^*$, its Kleene-closure is

$$L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cup \dots$$

- The Kleene-closure is idempotent, i.e. $(L^*)^* = L^*$

Operator precedence

- Closures comes first. . .
- . . .then concatenations. . .
- . . .finally set operators
- **unless there are brackets**

Special cases

- Always remember that $\varepsilon \neq \emptyset$
- If $L = \emptyset \Rightarrow \bar{L} = \emptyset$
- If $L \neq \emptyset \Rightarrow \varepsilon \in \bar{L}$
- $\emptyset^* = \{\varepsilon\}$
- $\{\varepsilon\}^* = \{\varepsilon\}$

- (Some) Languages with finite cardinality may be specified by enumerating their words
- (Some) Languages with infinite cardinality may be specified in terms of *word features*, example

$$L = \{\textit{all the words that start with } \alpha\}$$

- **It would be better to have a formal tool to specify languages, in order to enable *quantitative* methods to solve analysis and synthesis problems**
- **Automata are one of these tools**

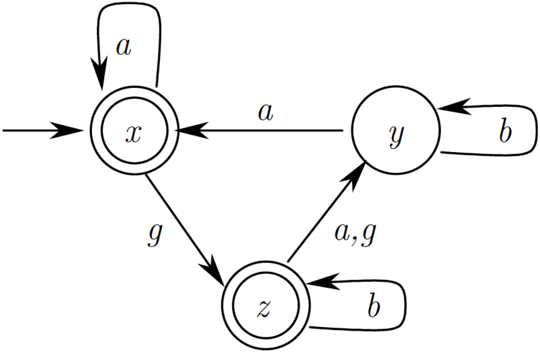
A (logic deterministic) automaton G is the 6-tuple

$$G = (X\,, E\,, f\,, \Gamma\,, x_0\, X_m)$$

where

- X is the *discrete* state space. If the cardinality of X is finite, then G is also referred to as *finite state machine (FSM)* or *finite state automaton*
- E is the set of events associated with the transitions in G
- $f(\cdot\,,\cdot) : X \times E \mapsto X$ is the *transition function*
- $\Gamma(\cdot) : X \mapsto 2^E$ is the active event function. $\Gamma(\cdot)$ is implicitly defined by $f(\cdot\,,\cdot)$
- x_0 is the initial state
- $X_m \subseteq X$ is the the set of *marked* or *final* states (used in the SCT context to deal with non-blocking requirements)

Graphical representation



Recursive extension of the transition function f

It is common to recursively extend the transition function $f(\cdot\,,\cdot)$ from the $X \times E$ domain to the $X \times E^*$ one as follows

- $f(x\,,\varepsilon) := x$ for all $x \in X$
- $f(x\,, we) := f\left(f(x\,, w)\,, e\right)$ with $w \in E^*$ and $e \in E$

Let $G = (X, E, f, \Gamma, x_0, X_m)$

Language generated by G – $\mathcal{L}(G)$

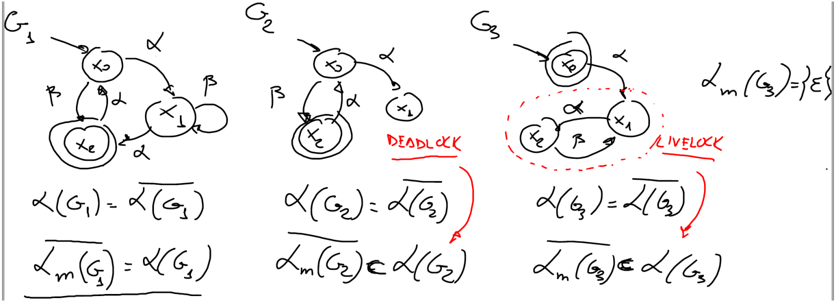
$$\mathcal{L}(G) := \{w \in E^* \ : \ f(x_0, w) \text{ is defined}\}$$

Language marked by G – $\mathcal{L}_m(G)$

$$\mathcal{L}_m(G) := \{w \in \mathcal{L}(G) \ : \ f(x_0, w) \in X_m\}$$

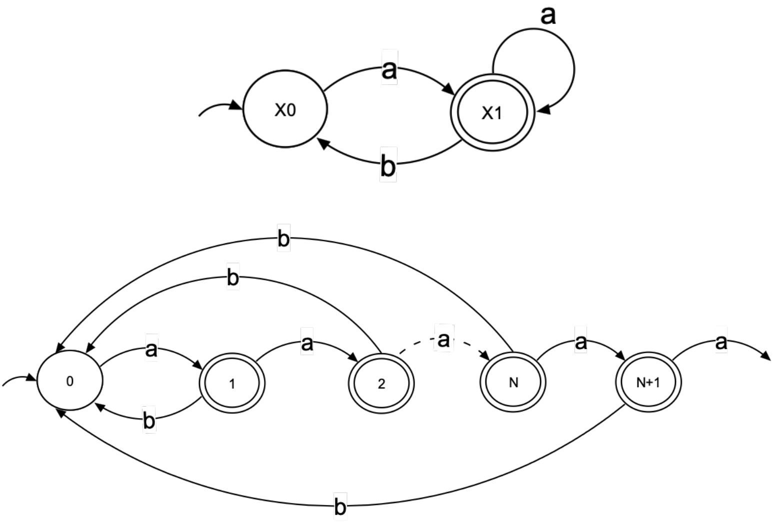
By definition

- $\mathcal{L}(G)$ is always prefix-closed, i.e. $\overline{\mathcal{L}(G)} = \mathcal{L}(G)$
- $\mathcal{L}_m(G) \subseteq \overline{\mathcal{L}_m(G)} \subseteq \mathcal{L}(G)$
- If $\overline{\mathcal{L}_m(G)} \subset \mathcal{L}(G)$, then there are *deadlock* and/or *livelock* in G
- If $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$, then G is said to be *non-blocking*


G1
G2
G3
DEADLOCK
LIVELOCK
L(G1) = L̄(G1)
L̄m(G1) = L(G1)
L(G2) = L̄(G2)
L̄m(G2) ⊂ L(G2)
Lm(G3) = {ε}
L(G3) = L̄(G3)
L̄m(G3) ⊂ L(G3)

Automata G_1 and G_2 are said to be equivalent if

- $\mathcal{L}(G_1) = \mathcal{L}(G_2)$
- $\mathcal{L}_m(G_1) = \mathcal{L}_m(G_2)$


a
a
X0
X1
b
b
b
a
a
a
a
a
0
1
2
N
N+1
b
b

Accessible part of G

Removes all the states that are unreachable from x_0 *(and the related transitions)* Given $G = (X, E, f, x_0, X_m)$ the **accessible part of G** $Ac(G)$ is

$$Ac(G) := (X_{ac}, E, f_{ac}, x_0, X_{ac,m})$$

where

- $X_{ac} = \{x \in X \mid \exists\, w \in E^* \text{ s.t. } f(x_0, w) = x\}$
- $X_{ac,m} = X_m \cap X_{ac}$
- $f_{ac} = f_{|X_{ac} \times E \mapsto X_{ac}}$
- The accessible part does not affect nor $\mathcal{L}(G)$ neither $\mathcal{L}_m(G)$
- If $G = Ac(G)$, then G is said to be *accessible*

Removes all the states that do not lead to a marked state (and the related transitions) Given $G = (X, E, f, x_0, X_m)$ the **coaccessible part of G** $CoAc(G)$ is

$$CoAc(G) := (X_{coac}, E, f_{coac}, x_{0_{coac}}, X_m)$$

where

- $X_{coac} = \{x \in X \mid \exists\, w \in E^* \text{ s.t, } f(x, w) \in X_m\}$
- $x_{0_{coac}} = x_0$ if $x_0 \in X_{coac}$, otherwise x_0 is left undefined
- $f_{coac} = f_{|X_{coac} \times E \mapsto X_{coac}}$
- By definition $CoAC(G)$ is always non-blocking, i.e. the generated language is modified in such a way that
  - $\mathcal{L}(CoAc(G)) = \overline{\mathcal{L}_m(CoAc(G))} = \overline{\mathcal{L}_m(G)}$
- If $G = CoAc(G)$, then G is said to be *coaccessible*

Trim operation

- $Trim(G) := CoAc\,(Ac(G)) = Ac\,(CoAc(G))$
- If $G = Trim(G)$, then G is said to be *trimmed*
- A trimmed automaton is both accessible and coaccessible

Complement wrt to E^*

- Let G be a trimmed automaton with
  - $\mathcal{L}_m(G) = L$
  - $\mathcal{L}(G) = \bar{L}$
- The **complement automaton** G^{comp} is such that

$$\mathcal{L}(G^{comp}) = E^* \setminus L$$

1. Complete the transition function $f(\cdot\,,\cdot)$ as follows

$$f_{tot}(x\,,e) := \begin{cases} f(x\,,e) & \text{if } e \in \Gamma(x) \\ x_d & \text{otherwise} \end{cases}$$

   with $x_d \notin X_m$ and $f_{tot}(x_d\,,e) = x_d \;\forall\; e \in E$

2. Let

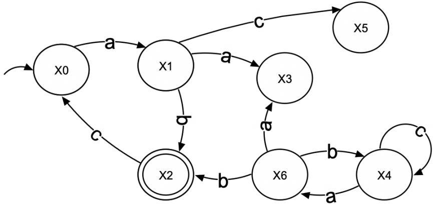
$$G^{comp} = (X \cup \{x_d\}\,, E\,, f_{tot}\,, x_0\,, X_m^{new})$$

   with $X_m^{new} = (X \cup \{x_d\}) \setminus X_m$

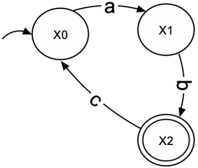
   Clearly it is

- $\mathcal{L}\,(G^{comp}) = E^*$
- $\mathcal{L}_m\,(G^{comp}) = E^* \setminus \mathcal{L}_m\,(G)$

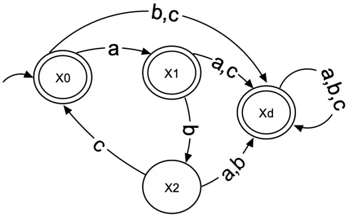
Example of complement automaton



Automaton G



Trimmed automaton



Complement automaton

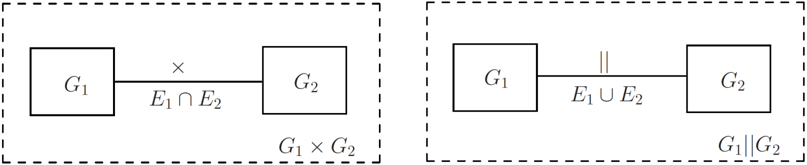


Figure: Cross product $G_1 \times G_2$ and parallel composition (or concurrent product) $G_1 \| G_2$

Given G_1 and G_2 the **product $G_1 \times G_2$** automaton is

$$G_1 \times G_2 := Ac\left(X_1 \times X_2\,, E_1 \cap E_2\,, f\,, \Gamma_{1\times 2}\,, (x_{0_1}\,, x_{0_2})\,, X_{m_1} \times X_{m_2}\right)$$

with

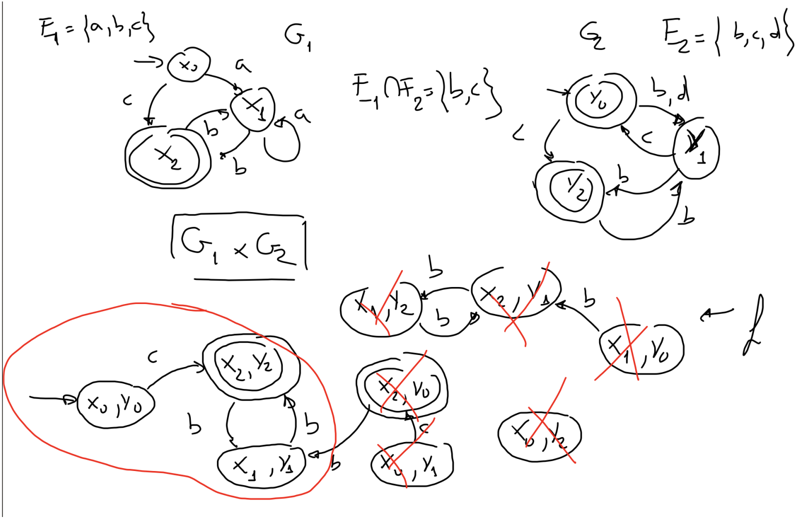
$$f\left((x_1\,, x_2)\,, e\right) := \begin{cases} (f_1(x_1\,, e)\,, f_2(x_2\,, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

and $\Gamma_{1\times 2}(x_1\,, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$

NOTE: an event occurs in $G_1 \times G_2$ if and only if it occurs in both automata G_1 and G_2. It follows that

- $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$
- $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$

Example of cross product



Given G_1 and G_2 the **parallel composition** $G_1 \| G_2$ automaton is

$$G_1 \| G_2 := Ac\left(X_1 \times X_2\,, E_1 \cup E_2\,, f\,, \Gamma_{1\|2}\,, (x_{0_1}\,, x_{0_2})\,, X_{m_1} \times X_{m_2}\right)$$

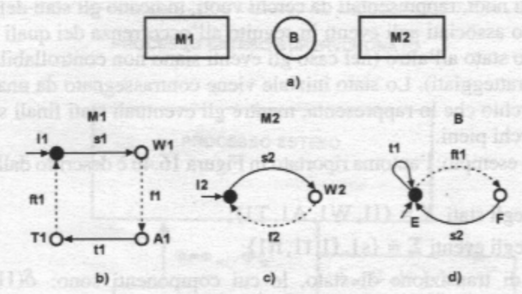
with

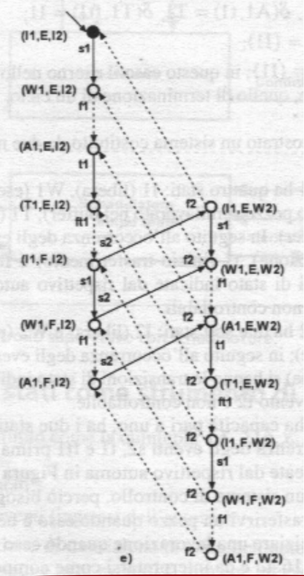
$$f\left((x_1\,, x_2)\,, e\right) := \begin{cases} (f_1(x_1\,, e)\,, f_2(x_2\,, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1\,, e)\,, x_2) & \text{if } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1\,, f_2(x_2\,, e)\,, x_2) & \text{if } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

and

$$\Gamma_{1\|2}(x_1\,, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E_2] \cup [\Gamma_2(x_2) \setminus E_1]$$

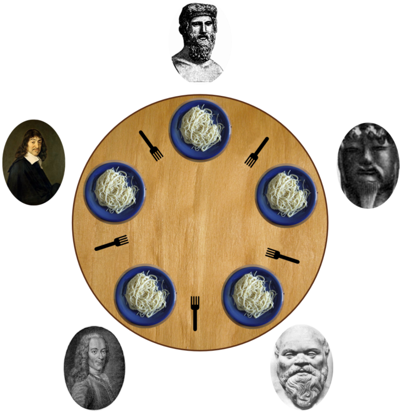
Example - Simple FMS



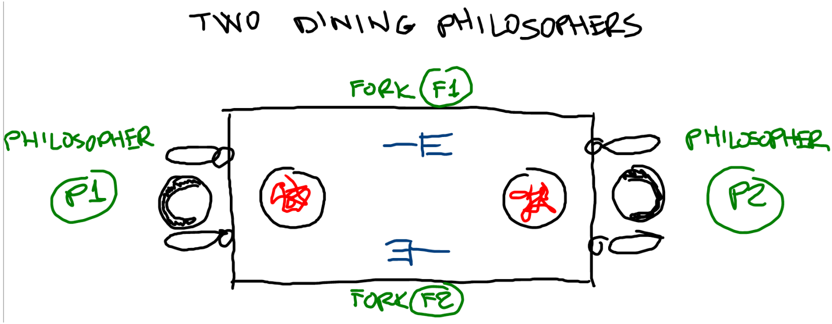


Dijkstra's dining philosophers problem (1965)

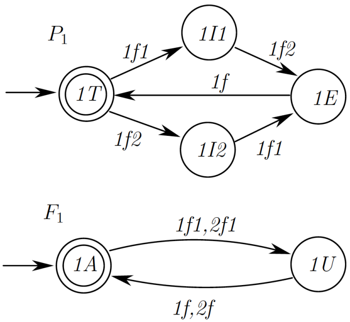
Deadlock due to shared resources (the forks)



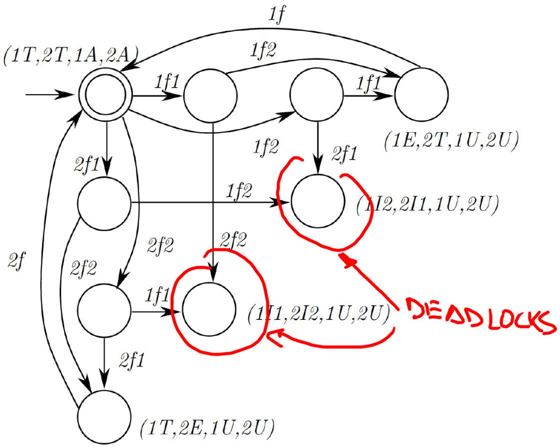
Two dining philopophers



Modelling philosophers and forks



F1 || F2 || P1 || P2 → 9 STATES



The curse of dimensionality

- 2 philosophers 2 forks $\rightarrow$ overall model with 9 states
- 3 philosophers 3 forks $\rightarrow$ overall model with 504 states
- 4 philosophers 4 forks $\rightarrow$ overall model with 4.080 states
- 5 philosophers 5 forks $\rightarrow$ overall model with 32.736 states
- 6 philosophers 6 forks $\rightarrow$ overall model with 262.080 states $\rightarrow$ **about 1 minute** to compute the parallel composition on this laptop
- 7 philosophers 7 forks $\rightarrow$ overall model with 2.097.024 states $\rightarrow$ **more than 1 hour** to compute the parallel composition on this laptop
- 8 philosophers 8 forks $\rightarrow$ **GOD KNOWS :)**

- Given the two sets of events E_1 and E_2, we need to introduce the **projection functions** $P_i(\cdot)$ and their inverse in order to derive a compact expression for both the generated and marked languages of $G_1 \| G_2$

$$P_i : (E_1 \cup E_2)^* \mapsto E_i^*$$

$$\begin{cases} P_i(\varepsilon) := \varepsilon & \\ P_i(e) := e & \text{if } e \in E_i \\ P_i(e) := \varepsilon & \text{if } e \notin E_i \\ P_i(we) := P_i(w)P_i(e) & w \in (E_1 \cup E_2)^* \,, e \in (E_1 \cup E_2) \end{cases}$$

- The projection function will be used also when uncertainty in terms of presence of unobservable events will be considered

Inverse projection

- The **inverse projection** $P_i^{-1}(\cdot)$ is defined as

$$P_i^{-1} : E_i^* \mapsto 2^{(E_1 \cup E_2)^*}$$

$$P_i^{-1}(t) := \{w \in (E_1 \cup E_2)^* \ : \ P_i(w) = t\}$$

- While the projection of a word is a (possibly empty) word, **the inverse projection of a word is a language**

Extend projection to languages

- Given a language L defined over $E_1 \cup E_2$, the extensions of the projection functions to L are

$$P_i\,(L) := \{t \in E_i^* \; : \; \exists\, w \in L\,,\, P_i(w) = t\}$$

- Given a language $L_i \subseteq E_i^*$ defined over $E_i(i = 1\,,2)$, the extension of the inverse projection to L_i is

$$P_i^{-1}\,(L_i) := \{w \in (E_1 \cup E_2)^* \; : \; \exists\, t \in L_i,\,,\, P_i(w) = t\}$$

- Note that

$$P_i\left(P_i^{-1}\,(L)\right) = L$$

  and that

$$L \subseteq P_i^{-1}\,(P_i\,(L))$$

Let $E_1 = \{a, b\}$ and $E_2 = \{b, c\}$ and

$$L = \{c, ccb, abc, cacb, cabcbbca\}$$

Then

- $P_1(L) = \{\varepsilon, b, ab, abbba\}$
- $P_2(L) = \{c, ccb, bc, cbcbbc\}$
- $P_1^{-1}(\{\varepsilon\}) = \{c\}^*$
- $P_1^{-1}(\{ab\}) = \{c\}^* \{a\} \{c\}^* \{b\} \{c\}^*$
- $P_1^{-1}(P_1(\{abc\})) = P_1^{-1}(\{ab\})$

Language generated by $G_1 \| G_2$

$$\mathcal{L}(G_1 \| G_2) = P_1^{-1}(\mathcal{L}(G_1)) \cap P_2^{-1}(\mathcal{L}(G_2))$$

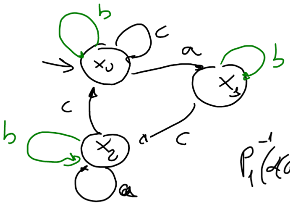
Language marked by $G_1 \| G_2$

$$\mathcal{L}_m(G_1 \| G_2) = P_1^{-1}(\mathcal{L}_m(G_1)) \cap P_2^{-1}(\mathcal{L}_m(G_2))$$

$$E_1 = \{a, c\} \subseteq E = \{a, b, c\}$$

$$\boxed{L \subseteq P_i^{-1}(L)}$$

G



$$\mathcal{L}(G) \subseteq E_1^*$$

$$P_1^{-1}(\mathcal{L}(G)) = \{ t \in E^* \mid P_1(t) = s \text{ with } s \in \mathcal{L}(G) \}$$

Regular expressions over an alphabet E – $RE(E)$

Given the alphabet E, the regular expressions defined over E are defined as follows

1. $\emptyset$ and e, for all $e \in E$ are regular expressions
2. if $a, b \in RE(E)$ then the following are regular expressions
   - $a \cup b$ (union)
   - ab and ba (concatenation)
   - a^* and b^* (Kleene closure)
3. There are no other regular expressions than the ones defined by rules 1 and 2

Regular languages – $Reg(E)$

In words, the class of **regular languages** defined over E – $Reg(E)$ – is the class of languages that can be *built* by using regular expressions

Let us denote with $Rec(E)$ the class of recognizable languages, i.e. the languages that can be marked by finite state automata.

Kleene Theorem (1936)

$$Rec(E) = Reg(E)$$

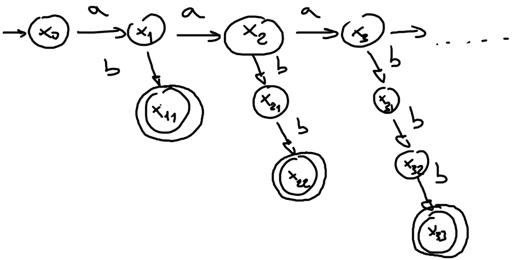
S. Haar, T. Masopust
Languages, decidability, and complexity
in *Control of Discrete-Event Systems*
Springer, 2013

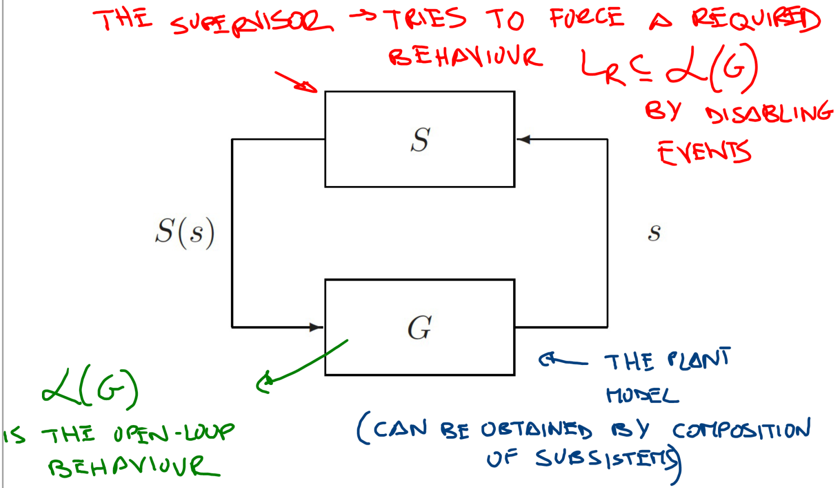
Clearly, the answer is **NO!**

$$L = \{ w \in E^* \text{ s.t. } w = a^n b^n, n \geq 0 \} \qquad E = \{a, b\}$$

$$a^n = \underbrace{aa \dots a}_{n}$$

THIS IS NOT A REGULAR LANGUAGE




THE SUPERVISOR → TRIES TO FORCE A REQUIRED BEHAVIOUR $L_R \subseteq \mathcal{L}(G)$
BY DISABLING EVENTS
S
S(s)
s
G
THE PLANT MODEL
(CAN BE OBTAINED BY COMPOSITION OF SUBSYSTEMS)
$\mathcal{L}(G)$
IS THE OPEN-LOOP BEHAVIOUR

A nice tutorial lecture can be found here
https://www.control.utoronto.ca/~wonham/Research.html

Proposed (1982): Supervisory Control Theory

(*Peter Ramadge & WMW*)

- Automaton representation
  - ***internal*** state descriptions for concrete modeling and computation
- Language representation
  - ***external*** i/o descriptions for implementation-independent concept formulation
- Simple control 'technology'

Community Response

Anonymous Referees (1983-87)

- ***[Leading control journal]***
  "Automata have no place in control engineering."
  Reject!
- ***[Leading computer journal]***
  "Finite automata and regular languages are nothing new at best and trivial at worst."
  Reject!
- ***SIAM J. Control & Optimization***
  "So this is optimal control? Well..."
  Accept

Software tools

- Many tools – Most of them developed by academic groups
  - TCT (Wonham's group @ University of Toronto)
    - `https://www.control.utoronto.ca/~wonham/Research.html`
  - **UMDES-DESUMA** (Lafortune's group @ University of Michigan)
    - `https://wiki.eecs.umich.edu/desuma/index.php/DESUMA`
  - Supremica (Chalmers University)
    - `https://supremica.org/`
  - ...
- Some tools includes automatic code generation for industrial devices (IEC-61131 compliant)
- For those who are interested a nice (and not too old) overview can be found here

L. Preischadt Pinheiro et al.
Nadzoru: A Software Tool for Supervisory Control of Discrete Event Systems
5th IFAC Int. Workshop on Dependable Control of Discrete Systems, 2015

UMDES allows to specify automata as .fsm text files

Example

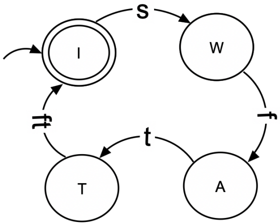
4 ← *number of states*

I 1 1 *(state name, marked state flag, number of output transitions)*
s W c o *(output event, new state, controllability flag, observability flag)*

W 0 1
f A uc o

A 0 1
t T c o

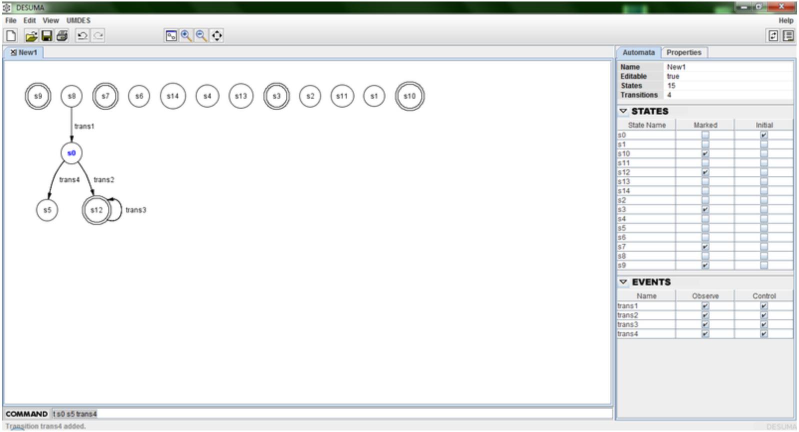
T 0 1
ft I uc o



Some UMDES commands

- `acc` → accessible part $Ac(G)$
- `co_acc` → coaccessible part $CoAc(G)$
- `trim` → trim automaton $Trim(G)$
- `comp_fsm` → complement automaton G^{comp}
- `product` → cross product $G_1 \times G_2$
- `par_comp` → parallel composition $G_1 \| G_2$

For those of you who like GUI, there is DESUMA (or Supremica or . . .)



References

Chapter 2 (up to section 2.3.2) in

C. G. Cassandras and S. Lafortune
Introduction to Discrete Event Systems
Springer, 2008

Discrete Event Systems, Languages and Automata

From observability to privacy and security in discrete event systems

Prof. Gianmaria De Tommasi
Email: detommas@unina.it

December 2020

DIETI.
UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE