Introduction to automata

Discrete Event Systems and Supervisory Control

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

March 2021

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

Outline

- (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

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

- (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**

- (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

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

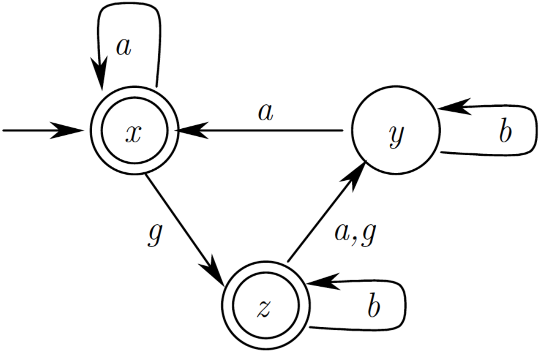
- **It would be better to have a formal tool to specify languages, in order to enable *quantitative* methods to solve analysis and synthesis problems**
- **Automata are one of these tools**

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)



a
a
x
y
b
g
a, g
z
b

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 – $\mathcal{L}(G)$

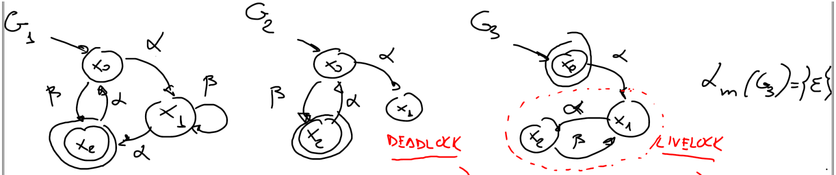
$$\mathcal{L}(G) := \{w \in E^* \;:\; f(x_0, w) \textit{ is defined}\}$$

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*



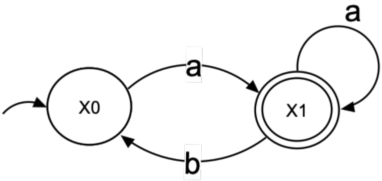
$$\mathcal{L}(G_1) = \overline{\mathcal{L}(G_1)} \qquad \mathcal{L}(G_2) = \overline{\mathcal{L}(G_2)} \qquad \mathcal{L}(G_3) = \overline{\mathcal{L}(G_3)}$$

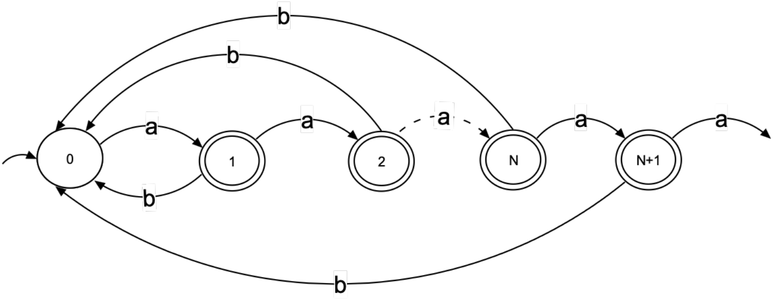
$$\overline{\mathcal{L}_m(G_1)} = \mathcal{L}(G_1) \qquad \overline{\mathcal{L}_m(G_2)} \subset \mathcal{L}(G_2) \qquad \overline{\mathcal{L}_m(G_3)} \subset \mathcal{L}(G_3)$$

Automata G_1 and G_2 are said to be equivalent if

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

Equivalence of automata - Example





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

- $X_{ac} = \{x \in X \mid \exists\, w \in E^* \text{ s.t. } f(x_0, w) = x\}$
- $X_{ac,m} = X_m \cap X_{ac}$
- $f_{ac} = f_{|X_{ac} \times E \mapsto X_{ac}}$

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*

Coaccessible part

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
- $f_{coac} = f_{|X_{coac} \times E \mapsto X_{coac}}$

Coaccessible part

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

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

- 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

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

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

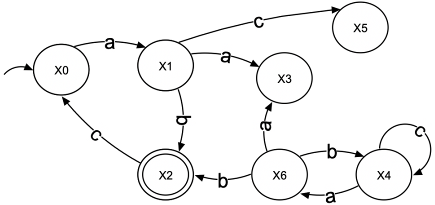
$$G^{comp} = (X \cup \{x_d\}\,, E\,, f_{tot}\,, x_0\,, X_m^{new})$$

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

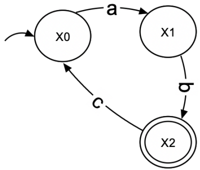
   Clearly it is

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

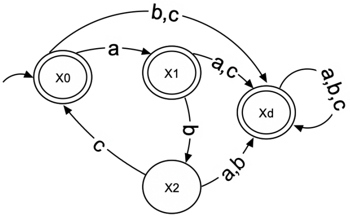
Example of complement automaton



Automaton G



Trimmed automaton



Complement automaton

Composition operations

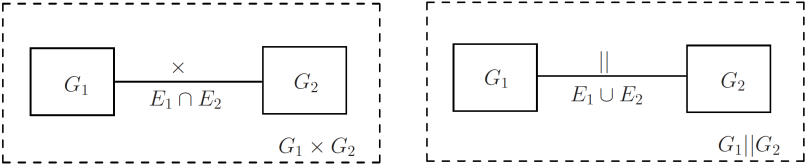


Figure: Cross product $G_1 \times G_2$ and parallel composition (or concurrent product) $G_1 \| G_2$

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

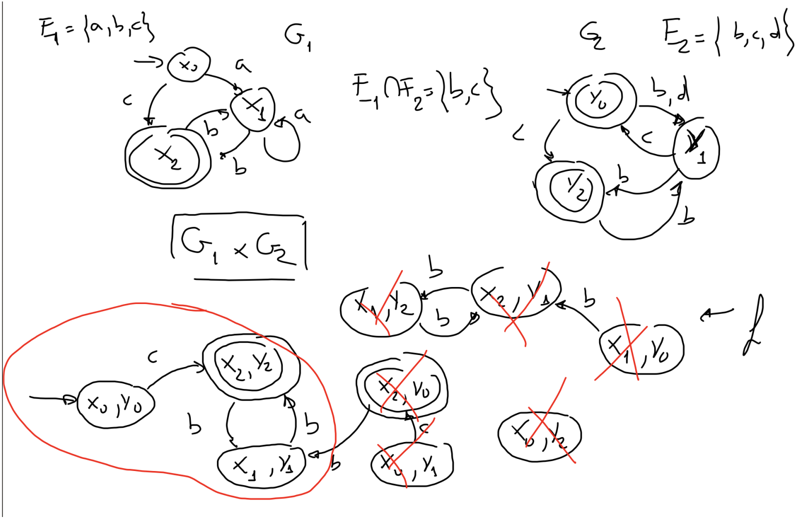
$$f\left((x_1\,, x_2)\,, e\right) := \begin{cases} (f_1(x_1\,, e)\,, f_2(x_2\,, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

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

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

Example of cross product



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

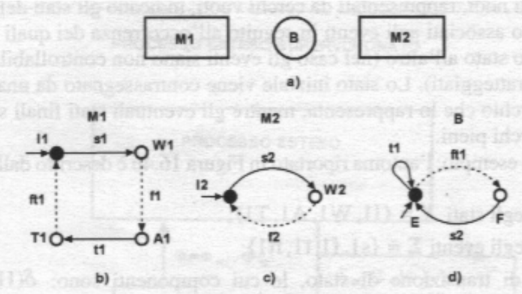
$$G_1 \| G_2 := Ac\left(X_1 \times X_2\,, E_1 \cup E_2\,, f\,, \Gamma_{1\|2}\,, (x_{0_1}\,, x_{0_2})\,, X_{m_1} \times X_{m_2}\right)$$

with

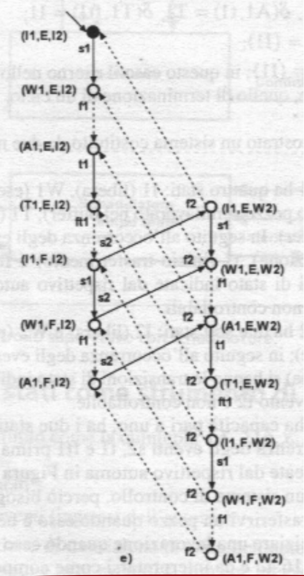
$$f\left((x_1\,, x_2)\,, e\right) := \begin{cases} (f_1(x_1\,, e)\,, f_2(x_2\,, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1\,, e)\,, x_2) & \text{if } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1\,, f_2(x_2\,, e)\,, x_2) & \text{if } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{undefined}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

and

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



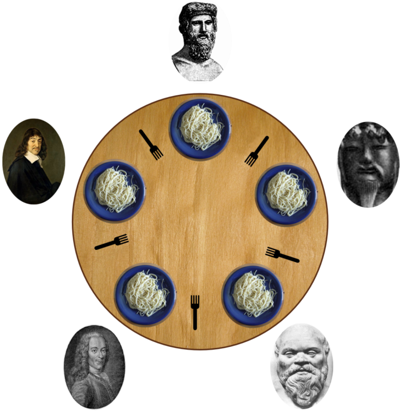
M1
B
M2
a)
M1
I1
s1
W1
ft1
f1
T1
t1
A1
b)
M2
s2
I2
W2
f2
c)
B
t1
ft1
E
F
s2
d)

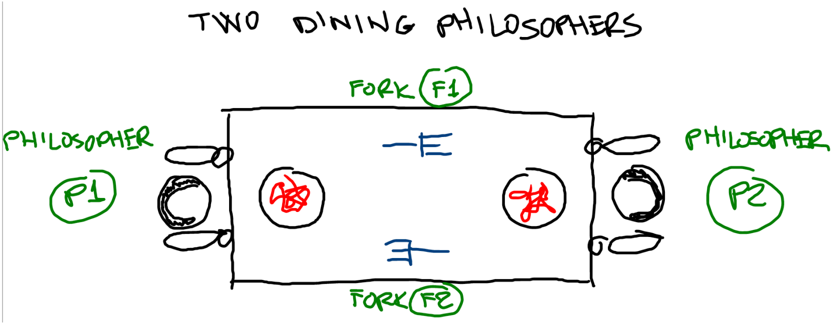


(I1,E,I2)
s1
(W1,E,I2)
f1
(A1,E,I2)
t1
(T1,E,I2)
ft1
(I1,F,I2)
s1
(W1,F,I2)
f1
(A1,F,I2)
(I1,E,W2)
(W1,E,W2)
(A1,E,W2)
(T1,E,W2)
(I1,F,W2)
(W1,F,W2)
(A1,F,W2)
s2
f2
e)

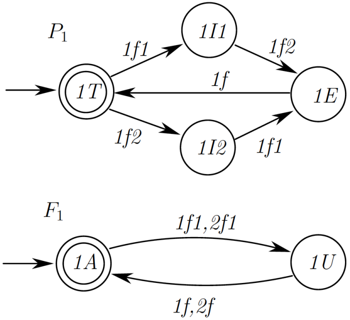
Dijkstra's dining philosophers problem (1965)

Deadlock due to shared resources (the forks)

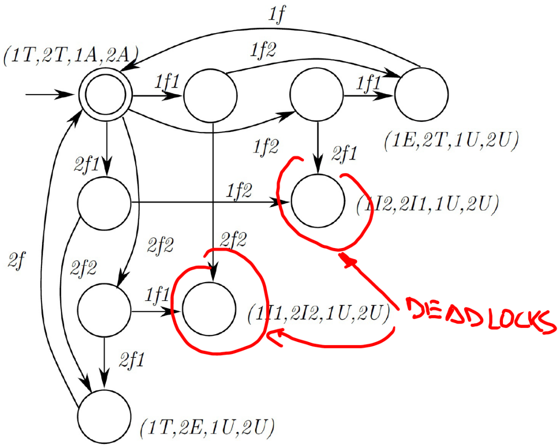



TWO DINING PHILOSOPHERS
FORK F1
PHILOSOPHER
P1
PHILOSOPHER
P2
FORK F2

Modelling philosophers and forks



F1 || F2 || P1 || P2 → 9 STATES



The curse of dimensionality

- 2 philosophers 2 forks $\rightarrow$ overall model with 9 states

- 2 philosophers 2 forks $\rightarrow$ overall model with 9 states
- 3 philosophers 3 forks $\rightarrow$ overall model with 504 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

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

- 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

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

- 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 $G_1 \| G_2$

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

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

Projection function

- 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

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

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

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

- 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 $G_1 \| G_2$

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

Language marked by $G_1 \| G_2$

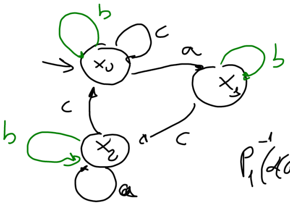
$$\mathcal{L}_m(G_1 \| G_2) = P_1^{-1}\left(\mathcal{L}_m(G_1)\right) \cap P_2^{-1}\left(\mathcal{L}_m(G_2)\right)$$

Why?

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

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

G



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

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

Introduction to automata

Discrete Event Systems and Supervisory Control

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

March 2021

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