LANGUAGES AND AUTOMATA

Pedro U. Lima

Instituto Superior Técnico (IST)
Instituto de Sistemas e Robótica (ISR)
Av.Rovisco Pais, 1
1049-001 Lisboa
PORTUGAL

September 2002

LANGUAGES

E finite event set (*alphabet*)

sequence of events (*word, string, trace*)

$E=\{a,b,c\}$ aaa, aabccc, cbabbaaa

ε string of no events (*empty string*)

$|s|$ length of a string $|\varepsilon| = 0$

LANGUAGES

A *Language* over a finite event set *E* is a set of finite length strings formed from events in *E*.

A string is obtained from events in E by *concatenation*.

s_1=rob s_2=otica $s_1 s_2$=robotica

ε is the *identity element* for the concatenation

$$\varepsilon s = s\varepsilon = s$$

LANGUAGES

E^* is the set of all finite strings of elements of E, including the empty string.

(Kleene closure or Kleene star operation)

$s=tuv$

t is a *prefix* of *s*.

u is a *substring* of *s*.

v is a *suffix* of *s*.

OPERATIONS ON LANGUAGES

In addition to the usual set theoretic operations like union, intersection, difference and complement with respect to E^* we define:

Concatenation: Let $L_a, L_b \subseteq E^*$, then

$$L_a L_b = \{s \in E^* : s = s_a s_b \ and \ s_a \in L_a \ and \ s_b \in L_b\}$$

Prefix-closure: Let $L \subseteq E^*$, then

$$\overline{L} = \{s \in E^* : \exists t \in E^* \ st \in L\}$$

L is said to be prefix-closed if $L = \overline{L}$

Kleene-closure: Let $L \subseteq E^*$, then

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

AUTOMATA

An *automaton* is a tuple $G=(X,E,f,\Gamma,x_0,X_M)$, where

X is a set of states *(Q)*

E is a set of labels *(event set or alphabet, I or Σ)*

$f : X \times E \rightarrow X$ *(transition or next-state function, possibly a partial function, δ)*

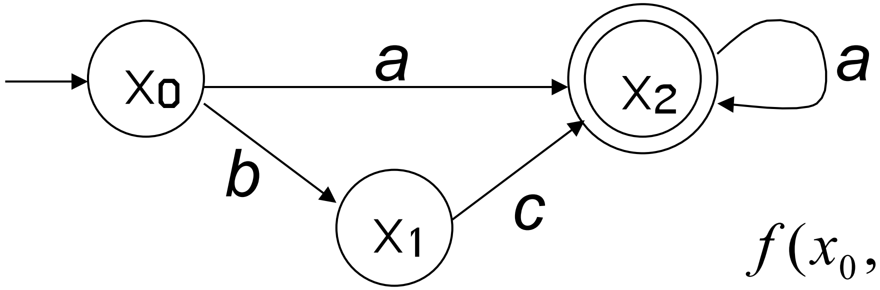
x_0 is the initial state

X_M is a set of marked states *(final states)*

$\Gamma : X \rightarrow 2^E$ active event function

AUTOMATA

Example:



$$f(x_0, a) = x_2$$
$$f(x_0, b) = x_1$$
$$f(x_1, c) = x_2$$
$$f(x_2, a) = x_2$$

$X = \{x_0, x_1, x_2\}$
$E = \{a, b, c\}$

x_0 is the initial state

$X_M = \{x_2\}$

$$\Gamma(x_0) = \{a, b\}$$
$$\Gamma(x_1) = \{c\}$$
$$\Gamma(x_2) = \{a\}$$

AUTOMATA

f can be uniquely extended from E to E* by:

$$f(x, \varepsilon) = x$$

$f(x, se) = f(f(x, s), e)$ for $s \in$ E* and $e \in$ E.

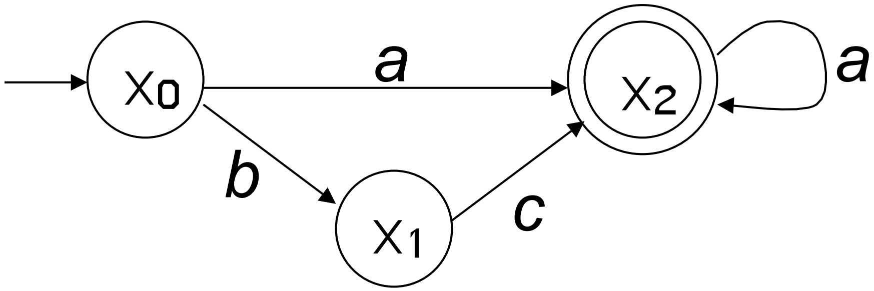
AUTOMATA

The language *generated* by $G = (X, E, f, \Gamma, x_0, X_M)$ is:

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

The language *marked* by $G = (X, E, f, \Gamma, x_0, X_M)$ is:

$$L_m(G) = \{s \in L(G) : f(x_0, s) \in X_m\}$$



$$L(G) = \{\varepsilon, a, b, bc, aa, bca, aaa, bcaa, aaaa, bcaaa, \ldots\}$$

$$L_m(G) = \{a, bc, aa, bca, aaa, bcaa, aaaa, bcaaa, \ldots\} \subseteq L(G)$$

AUTOMATA

Automata G_1 and G_2 are said to be *equivalent* if

$$L(G_1) = L(G_2) \text{ and } L_m(G_1) = L_m(G_2).$$

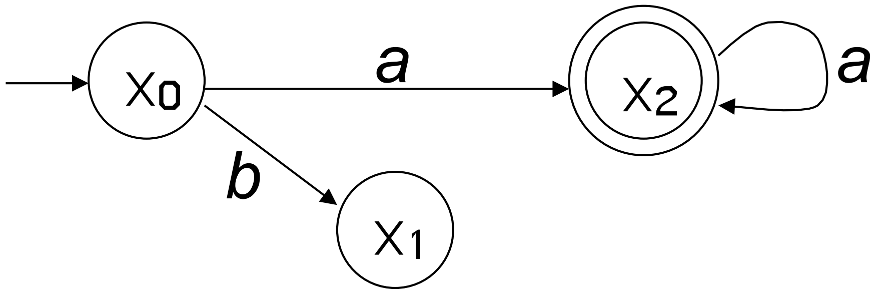
Automaton G is said to be *blocking* if

$$\overline{L_m(G)} \subset L(G)$$

and *nonblocking* when:

$$\overline{L_m(G)} = L(G)$$

Ex.:
Blocking



NON-DETERMINISTIC AUTOMATA

A *non-deterministic automaton* is a tuple
$G_{nd}=(X,E \cup \{\varepsilon\},f_{nd},\Gamma,x_0,X_M)$, where

X is a set of states

E is a set of labels

$f_{nd} : X \times E \bigcup \{\varepsilon\} \rightarrow 2^X$

$f_{nd}(x,\varepsilon) \subseteq X$

x_0 is the initial state

$x_0 \subseteq X$

X_M is a set of marked states

$\Gamma : X \rightarrow 2^E$

NON-DETERMINISTIC AUTOMATA

f_{nd} can be uniquely extended from E to E* by:

$$f_{nd}(x, se) = \{z : z \in f_{nd}(y, e) \ \text{for some state} \ y \in f_{nd}(x, s)\}$$

for $s \in E^*$ and $e \in E \cup \{\varepsilon\}$.

The language *generated* by $G_{nd} = (X, E \bigcup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_M)$

$$L(G_{nd}) = \{s \in E^* : \exists x \in x_0 \quad (f_{nd}(x, s) \ \text{is defined})\}$$

The language *marked* by $G_{nd} = (X, E \bigcup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_M)$

$$L_m(G_{nd}) = \{s \in L(G_{nd}) : \exists x \in x_0 \ (f_{nd}(x, s) \bigcap X_m \neq \varnothing)\}$$

UNARY OPERATIONS ON AUTOMATA

$Ac(G) = (X_{ac}, E, f_{ac}, x_0, X_{ac,M})$ is the **accessible** part of G where:

$$X_{ac} = \{x \in X : \exists s \in E^* \quad f(x_0, s) = x\}$$

$$X_{ac,M} = X_M \cap X_{ac}$$
$$f_{ac} = f|_{X_{ac} \times E \to X_{ac}}$$

Deletes from *G* all states not *accessible* or *reachable* from x_0 by some string in *L(G)*, without affecting *L(G)* and $L_m(G)$

UNARY OPERATIONS ON AUTOMATA

$CoAc(G) = (X_{coac}, E, f_{coac}, x_{0,coac}, X_M)$ is the **coaccessible** part of G where:

$$X_{coac} = \{x \in X : \exists s \in E^* \quad f(x,s) \in X_M\}$$

$$x_{0,coac} = \begin{cases} x_0 & \Leftarrow \quad x_0 \in X_{coac} \\ undefined & \Leftarrow \quad otherwise \end{cases}$$

$$f_{coac} = f|_{X_{coac} \times E \rightarrow X_{coac}}$$

Deletes from *G* all states not *coaccessible.*

A *state x* is *coaccessible* if there exists a string leading to X_m that goes through *x*

This operation may shrink $L(G)$ but it does not affect $L_m(G)$

UNARY OPERATIONS ON AUTOMATA

If *G* = *CoAc*(*G*), *G* is said to be *coaccessible* and

$$\overline{L_m(G)} = L(G)$$

i.e., *G* is non-blocking.
If *G* were non-blocking there would exist accessible states which are not coaccessible

Trim operation

Trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)]

Complement

$G=(X,E,f,\Gamma,x_0,X_M)$ is a trim automaton that marks

$L_m(G) = L \subseteq E^*$ (thus, G generates $\overline{L}$).

Let us build in two steps an automaton G^{comp} that will mark $E^* \setminus L$

$$L(G^{comp}) = E^*, \;\; L_m(G^{comp}) = E^* \setminus L_m(G) = E^* \setminus L$$

UNARY OPERATIONS ON AUTOMATA

Complement (cont'd)

1. add a new "dead" or "dump" state $x_d \notin Xm$ and complete f to make it total

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

$$f_{tot}(x_d, e) = x_d, \forall e \in E$$

so that $G_{tot} = (X \cup \{x_d\}, E, f_{tot}, x_0, X_m\}$ is such that

$$L(G_{tot}) = E^*,\ L_M(G_{tot}) = L$$

2. mark all unmarked states and unmark all marked states in G_{tot}

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

COMPOSITION OPERATIONS ON AUTOMATA

The **product** of G_1 and G_2 is the automaton:

$$G_1 \times G_2 = Ac(X_1 \times X_2, E_1 \cap E_2, f, (x_{01}, x_{02}), X_{M1} \times X_{M2})$$

where

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

$$L(G_1 \times G_2) = L(G_1) \cap L(G_2)$$
$$L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$$

COMPOSITION OPERATIONS ON AUTOMATA

The **parallel** *composition* of G_1 and G_2 is the automaton:

$$G_1 \parallel G_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f, (x_{01}, x_{02}), X_{M1} \times X_{M2})$$

where

$$f((x_1, x_2), e) = \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & \Leftarrow \; e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1, e), x_2) & \Leftarrow \; e \in \Gamma_1(x_1) \backslash E_2 \\ (x_1, f_2(x_2, e)) & \Leftarrow \; e \in \Gamma_2(x_2) \backslash E_1 \\ undefined & \Leftarrow \; otherwise \end{cases}$$

COMPOSITION OPERATIONS ON AUTOMATA

If $E_1 = E_2$, the parallel composition reduces to the product.

If $E_1 \cap E_2 = \{\}$, there are no synchronized transitions and $G_1 \| G_2$ is the concurrent behavior or shuffle of G_1 and G_2.

$$G_1 \| G_2 = G_2 \| G_1$$
$$G_1 \| (G_2 \| G_3) = (G_1 \| G_2) \| G_3$$

COMPOSITION OPERATIONS ON AUTOMATA

Projection

$$P_i : (E_1 \cup E_2)^* \rightarrow E_i^* \text{ for } i = 1,2$$

$$P_i(\varepsilon) = \varepsilon$$

$$P_i(e) = \begin{cases} e & \text{if } e \in E_i \\ \varepsilon & \text{if } e \notin E_i \end{cases}$$

$$P_i(se) = P_i(s)P_i(e) \text{ for } s \in (E_1 \cup E_2)^*, e \in (E_1 \cup E_2)$$

Inverse Projection

$$P_i^{-1} : E_i^* \rightarrow 2^{(E_1 \cup E_2)^*} \text{ for } i = 1,2$$

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

Given a string in the smaller event set E_i, the inverse projection returns the set of all strings in the larger event set $E_1 \cup E_2$ that project, with P_i , to the given string.

COMPOSITION OPERATIONS ON AUTOMATA

(Inverse) Projection - Extension to Languages

$$P_i(L) = \left\{ t \in E_i^* : \exists s \in L \ \ (P_i(s) = t) \right\}$$

and for $L_i \subseteq E_i^*$,

$$P_i^{-1}(L_i) = \left\{ s \in (E_1 \cup E_2)^* : \exists t \in L_i \ \ (P_i(s) = t) \right\}$$

$P_i[P_i^{-1}(L)] = L$ but in general $L \subseteq P_i^{-1}[P_i(L)]$

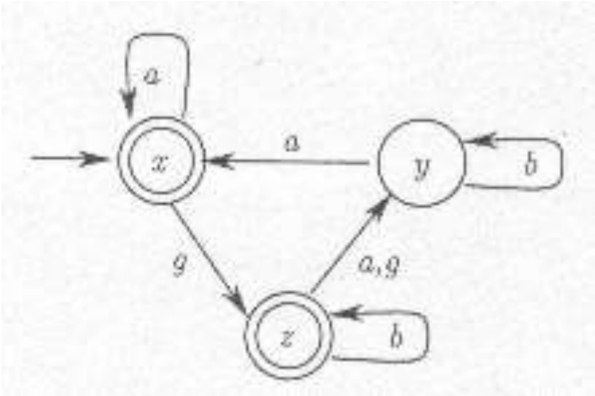
$$L(G_1 \| G_2) = P_1^{-1}[L(G_1)] \cap P_2^{-1}[L(G_2)]$$

$$L_m(G_1 \| G_2) = P_1^{-1}[L_m(G_1)] \cap P_2^{-1}[L_m(G_2)]$$

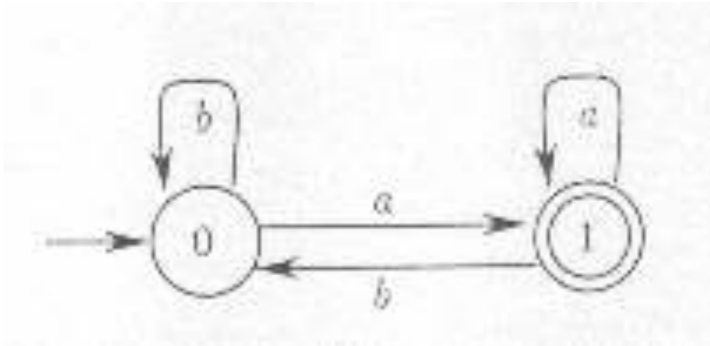
therefore $L_1 \| L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$

COMPOSITION OPERATIONS ON AUTOMATA

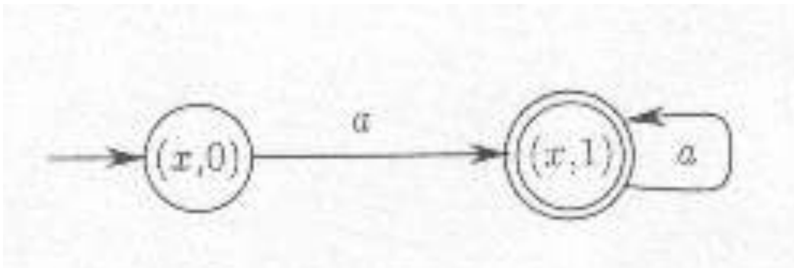
Examples (reprinted from [Cassandras, Lafortune]):



X

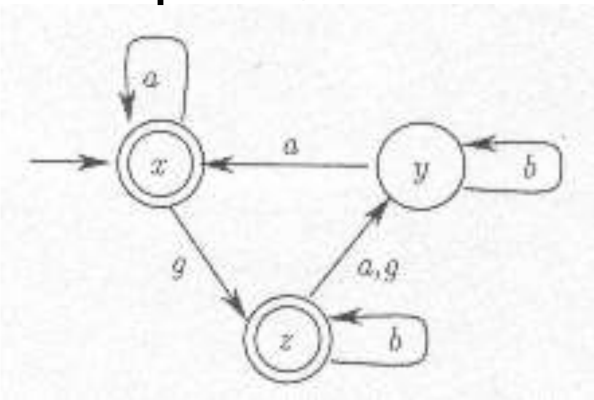


=

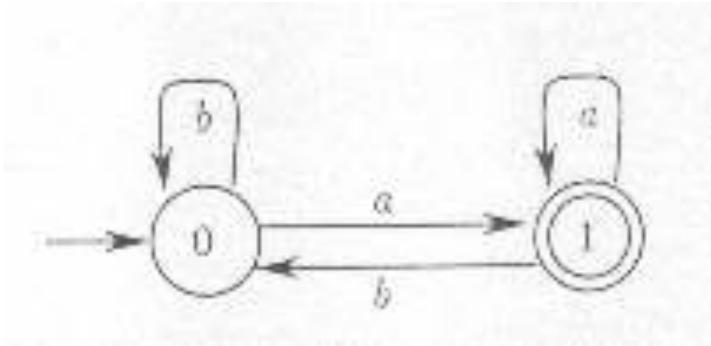


COMPOSITION OPERATIONS ON AUTOMATA

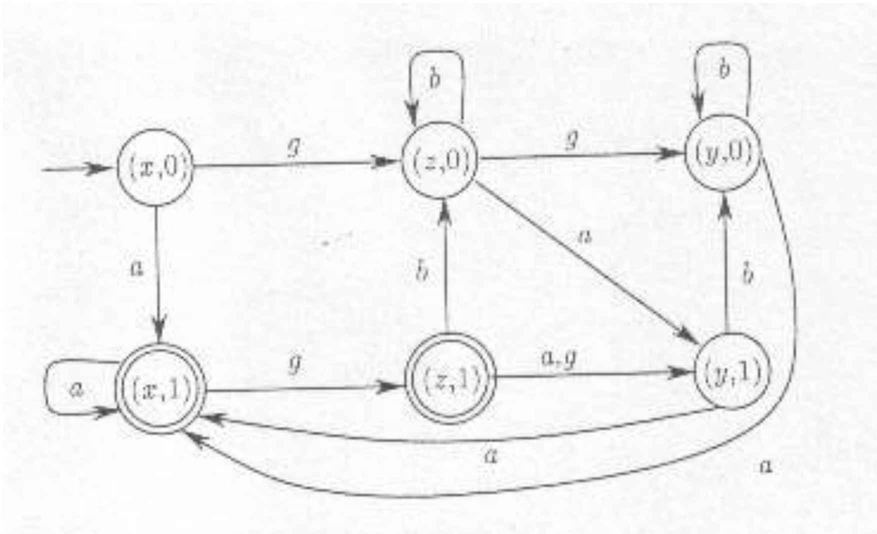
Examples (reprinted from [Cassandras, Lafortune]):



||

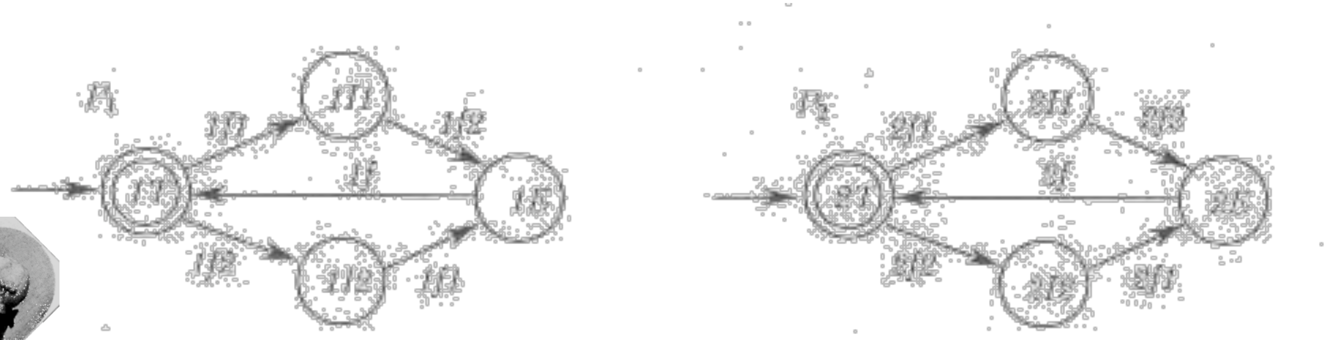


=

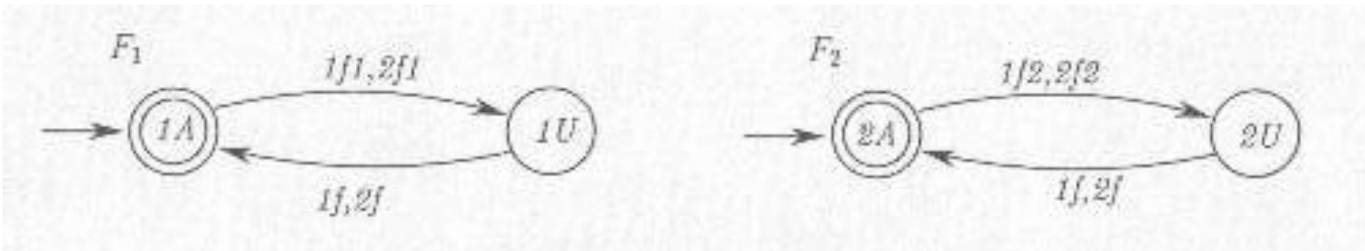


COMPOSITION OPERATIONS ON AUTOMATA

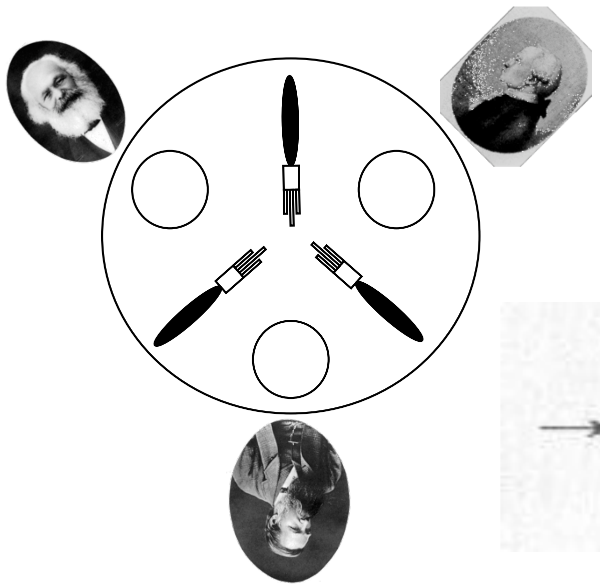
The Dining Philosophers example (reprinted from [Cassandras, Lafortune]):



Ex.: automata for 2 philosophers



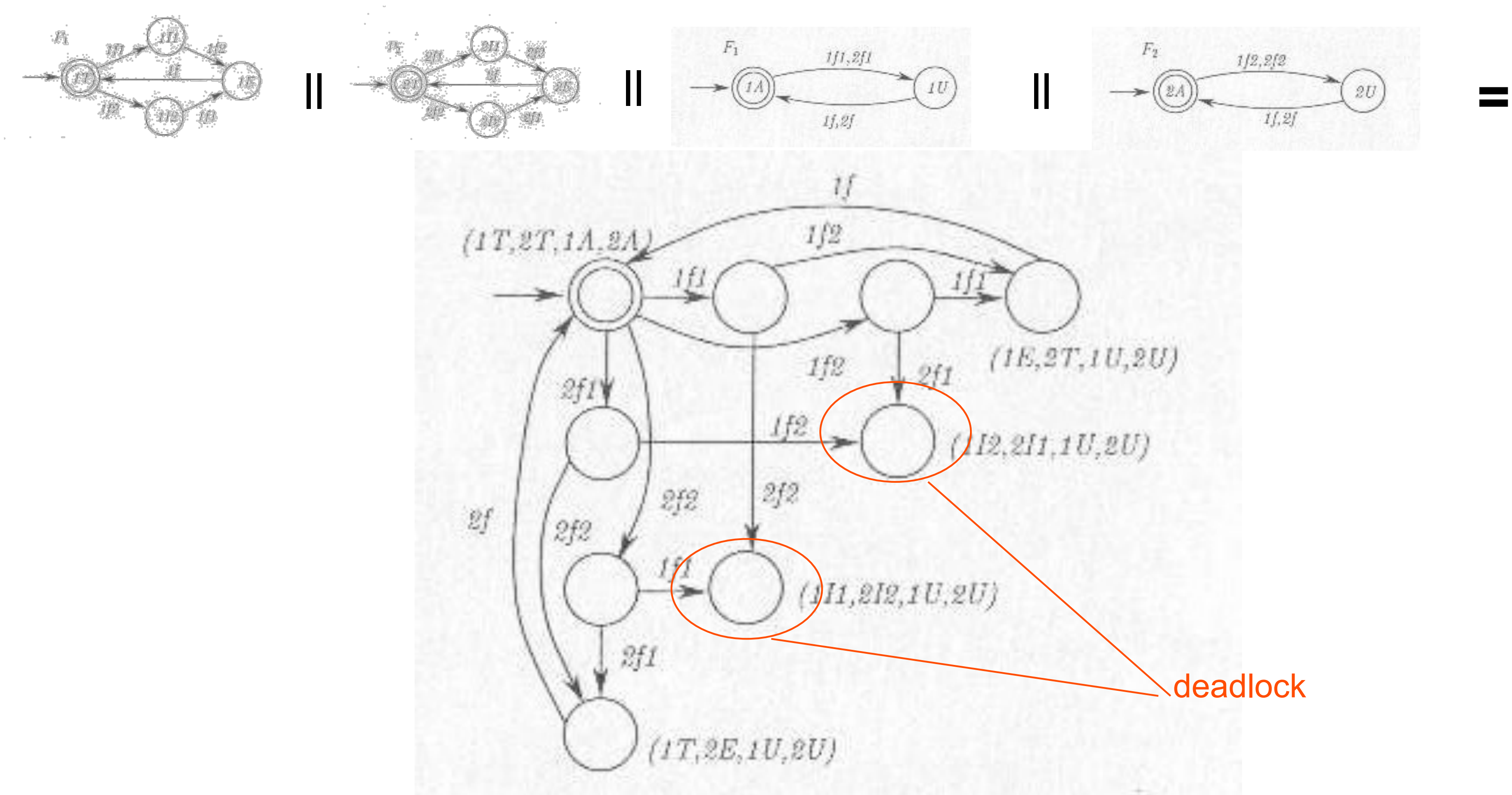
Ex.: automata representing resource constraints for 2 philosophers



Ex.: 3 philosophers

COMPOSITION OPERATIONS ON AUTOMATA

The Dining Philosophers example (reprinted from [Cassandras, Lafortune]):



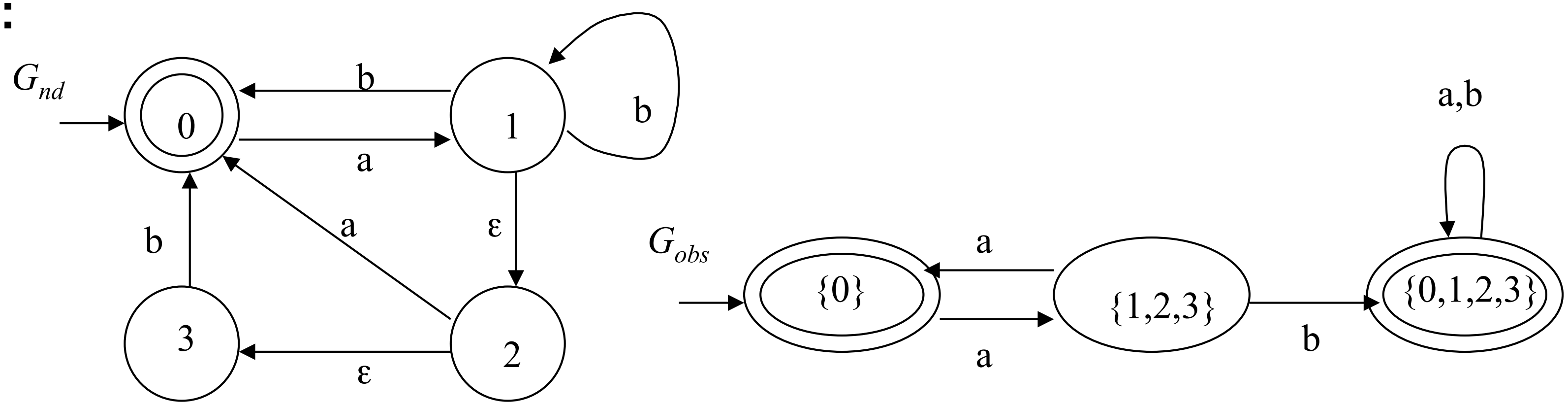
OBSERVER AUTOMATA

A non-deterministic automaton can always be transformed into an equivalent deterministic automaton.
The state space of the deterministic equivalent will be a subset of the power set of the state space of the non-deterministic automaton.

A non-deterministic **finite state** automaton has an equivalent deterministic **finite state** automaton.

The resulting equivalent deterministic automaton is called *observer* (G_{obs})

Example:



OBSERVER AUTOMATA

PROCEDURE TO BUILD OBSERVER G_{obs} OF NON-DETERMINISTIC AUTOMATON G_{nd}

$$G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, x_0, X_m) \qquad G_{obs} = (X_{obs}, E, f_{obs}, x_{0,obs}, X_{m,obs})$$

Step 1: $X_{obs} = 2^X \setminus \{\}$

Step 2: for each state $x \in X$ define the unobservable reach

$$UR(x) = f_{nd}(x, \varepsilon) \quad (\text{set extension}: UR(B) = \bigcup_{x \in B} UR(x))$$

Step 3: Define $x_{0,obs} = UR(x_0)$

Step 4: For each $S \subseteq X$ and $e \in E$, define

$$f_{obs}(S,e) = UR(\{x \in X : \exists x_e \in S\, [x \in f_{nd}(x_e, e)]\})$$

$$= \{x \in X : \exists x_e \in S\, [x \in f_{nd}(x_e, e)]\}$$

Set of states reachable from any state in *S* when *e* occurs

By definition of the extended version of f_{nd}

OBSERVER AUTOMATA

PROCEDURE TO BUILD OBSERVER G_{obs} OF NON-DETERMINISTIC AUTOMATON G_{nd} (cont'd)

Step 5 : $X_{m,obs} = \{S \subseteq X : S \cap X_m \neq \{\}\}$

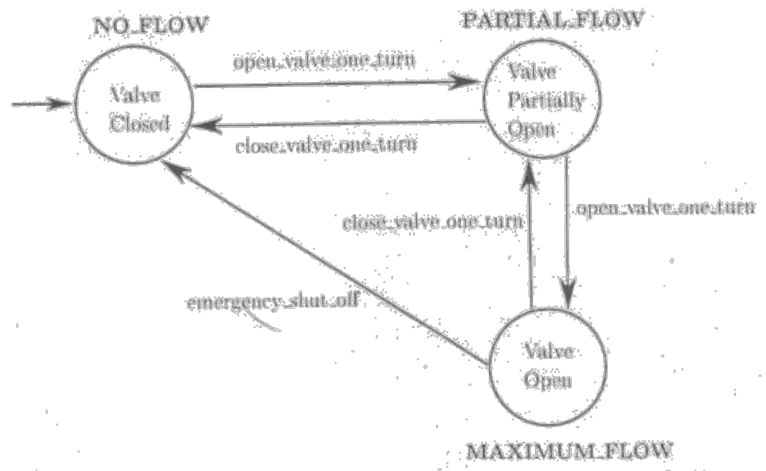
Step 6 : *D*o the above in a breadth - first manner so that only the accessible part of G_{obs} is constructed. The resulting state space $X_{obs} \subseteq 2^X$. The empty subset of X need not be considered, since it is never an accessible state of X_{obs}.

G_{obs} is a deterministic automaton

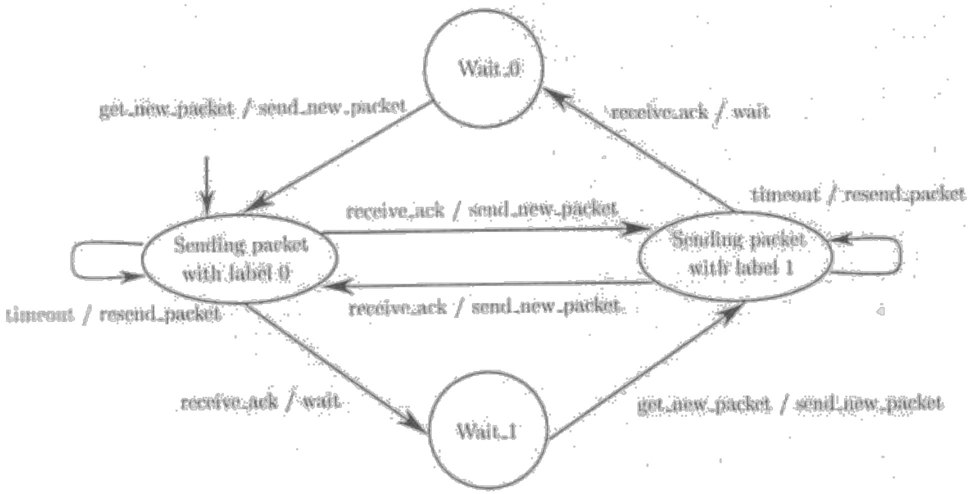
$$L(G_{obs}) = L(G_{nd})$$

$$L_m(G_{obs}) = L_m(G_{nd})$$

AUTOMATA WITH INPUTS AND OUTPUTS



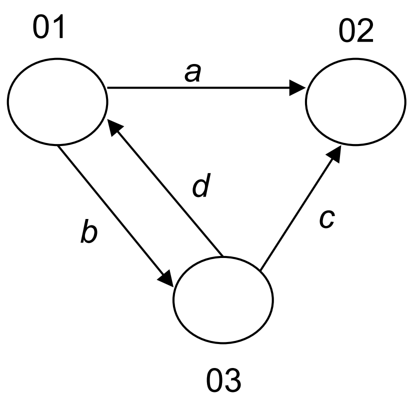
Moore automaton

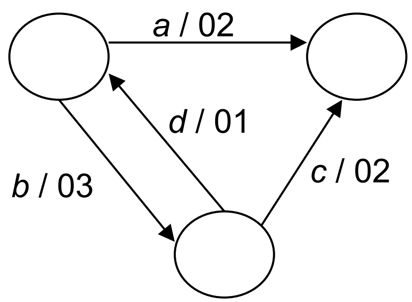


Mealy automaton

(reprinted from [Cassandras, Lafortune])

Conversion from Moore automaton to Mealy automaton





(adapted from [Cassandras, Lafortune])

FINITE STATE AUTOMATA (FSA)

A language is said to be *regular* if it can be marked by an FSA.

The class of languages representable by nondeterministic FSA is the same as the class of languages representable by deterministic FSA.

Let L_1 and L_2 be regular languages. Then

$$\overline{L_1}\,,\; L_1^*,\; L_1^c = E^* \backslash L_1,\; L_1 \cup L_2\,,\; L_1 L_2,\; L_1 \cap L_2$$

are also regular.

FINITE STATE AUTOMATA (FSA)

• The class of regular languages $\mathcal{R}$ delimits the languages that possess automaton representations that require finite memory when stored in a computer.

• Non-regular languages require infinite memory and can not be represented by FSA. However, another finite transition structure (Petri Nets) we will study can represent some of these non-regular languages (e.g.,

$$\{a^n b^n : n \geq 0\}).$$

Theorem – The class of languages representable by non-deterministic FSA is exactly the same as the class of languages representable by deterministic FSA: $\mathcal{R}$

FINITE STATE AUTOMATA (FSA)

Let E be any alphabet. A *regular expression* over E, $\mathcal{R}_E$ and the language it denotes are inductively defined by the following rules:

$\emptyset \in \mathcal{R}_E$ and denotes the empty set (language)

$\varepsilon \in \mathcal{R}_E$ and denotes the set (language) $\{\varepsilon\}$

$e \in \mathcal{R}_E, \quad \forall e \in E$ and denotes the set (language) $\{e\}$

$$a + b \in \mathcal{R}_E, \forall a, b \in \mathcal{R}_E \qquad a + b = \{a\} \cup \{b\}$$

$$ab \in \mathcal{R}_E, \quad \forall a, b \in \mathcal{R}_E \qquad ab = \{ab\}$$

$$a^* \in \mathcal{R}_E, \quad \forall a \in \mathcal{R}_E \qquad a^* = \{a\}^*$$

$$(a) \in \mathcal{R}_E, \quad \forall a \in \mathcal{R}_E \qquad (a) = \{(a)\}$$

nothing else is a regular expression

FINITE STATE AUTOMATA (FSA)

Kleene's Theorem (S. C. Kleene, 1950s) - A language can be denoted by a regular expression *iff* it is a regular language.

Examples for *E* = *{a,b,g}*:

$$(a+b)g^* \mapsto L = \{a, b, ag, bg, agg, bgg, aggg, bggg, \ldots\}$$

$$(ab)^* + g \mapsto L = \{\varepsilon, g, ab, abab, ababab, \ldots\}$$

ANALYSIS OF DES

- Most DES analysis problems imply navigating their state transition diagrams.
- For a deterministic automaton, the corresponding computational complexity is $O(n)$, where n is the number of states, unless iterations are necessary, in which case it will typically be $O(n^2)$.
- Usual assumption: $|E| \ll n$.
- This may work well for systems with up to a million states (or even for $n \sim 10^{29}$ with special symbolic techniques).
- Typically, the first step consists of building automaton models of the system components and then obtain the complete model by parallel composition.

ANALYSIS OF AUTOMATA

SAFETY

• ***reachability from x of an undesired or unsafe state y:*** take the Ac operation, with *x* declared as the initial state and look for state *y* in the result → $O(n)$

• ***presence of certain undesirable strings or substrings in the generated language:*** try to "execute" the substring from all the accessible states in the automaton (easy with the state transition diagram represented as a linked list) → $O(n)$

• ***inclusion of the generated language A in a "legal" or "admissible" language B:*** testing $A \subseteq B$ is equivalent to testing $A \cap B^c = \emptyset$. The *complement* of *B* is computable in $O(n_B)$. The intersection is obtained by taking the *product* of the corresponding automata → $O(n_A n_B)$

ANALYSIS OF AUTOMATA

BLOCKING

• ***blocking*** **(**$\overline{L_m(G)} \subset L(G)$**)** ***or not*** **(**$\overline{L_m(G)} = L(G)$**)** ***:*** take the CoAc operation of a given accessible automaton *G.* If any state is deleted, then *G* is blocking, otherwise is non-blocking. → $O(n)$

• ***if blocking identify deadlock and livelock states:*** start by finding all non-coaccessible states of *G.* Then:

  • deadlock states are found by examining the active event sets of the non-coaccessible states;

  • livelock cycles are found by finding the strongly connected components of the part of *G* consisting of the non-coaccessible states and their associated transitions among themselves → $O(n)$

STATE ESTIMATION

- ε-transitions in a non-deterministic automaton represent events that occur in the system modeled by the automaton (e.g., faults, absence of a sensor, event occurs at a remote location but is not communicated to the site being modeled) but which are not observed by an external *observer* of the system behavior
- instead of using ε-transitions and a non-deterministic automaton we will now use "genuine" (but non-observable) events and a deterministic automaton G with E partitioned in E_o and E_{uo}
- Projection $P: E^* \rightarrow E_0^*$

STATE ESTIMATION (cont'd)

- Projection $P: E^* \rightarrow E_0^*$

$$P(\varepsilon) = \varepsilon$$

$$P(e) = \begin{cases} e & \text{if } e \in E_0 \\ \varepsilon & \text{if } e \notin E_0 \end{cases}$$

$$P(se) = P(s)P(e) \text{ for } s \in E^*, e \in E$$

- by construction of the observer G_{obs}:

$$L(G_{obs}) = P[L(G)]$$

$$L_m(G_{obs}) = P[L_m(G)]$$

ANALYSIS OF AUTOMATA

STATE ESTIMATION (cont'd)

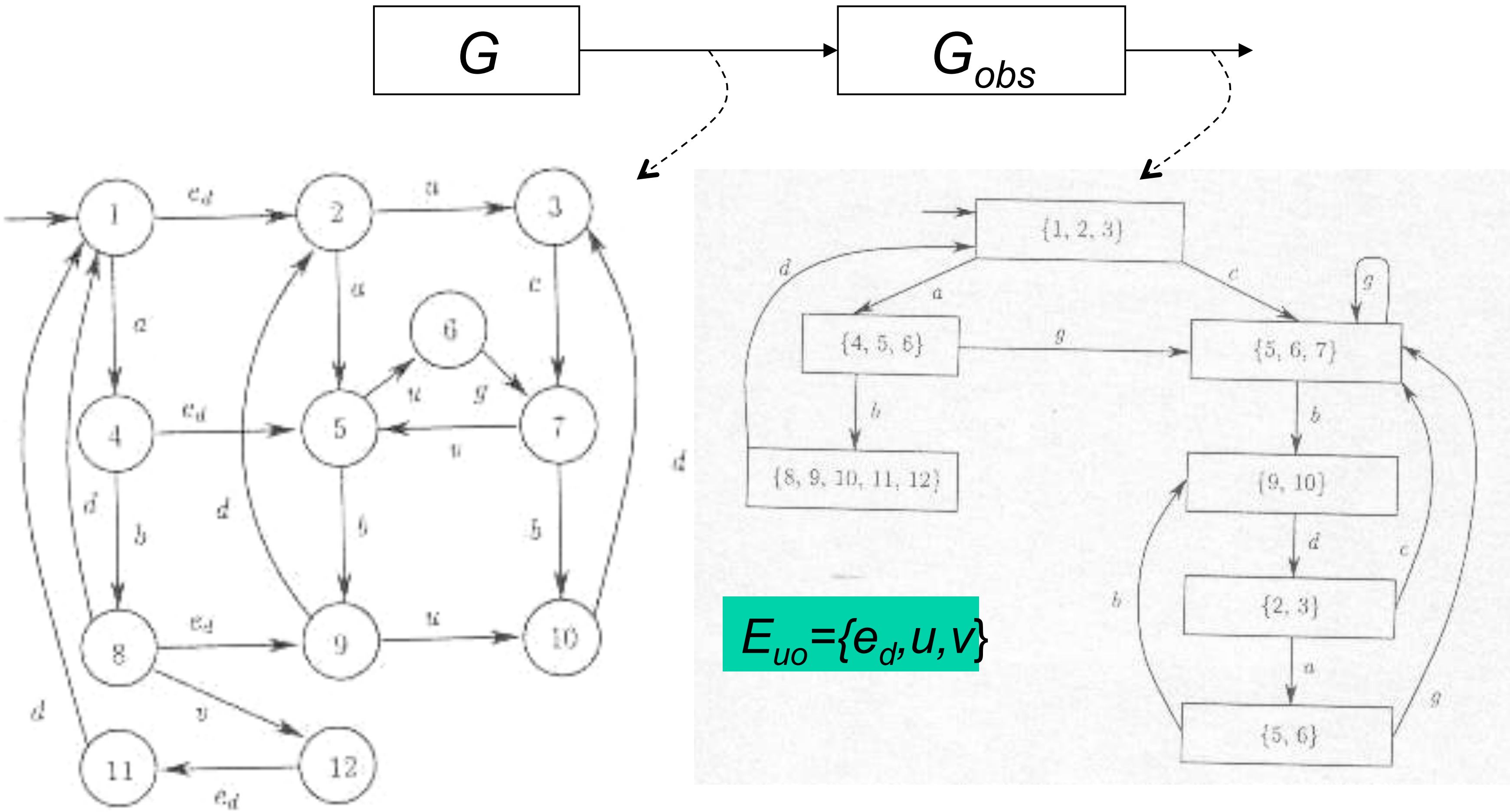
•the state of G_{obs} reached after string $t \in P[L(G)]$ will contain all states of G that can be reached after any of the strings in

$$P^{-1}(t) \cap L(G)$$

In this sense, the state of G_{obs} is an *estimate* of the current state of G

ANALYSIS OF AUTOMATA

STATE ESTIMATION (example - reprinted from [Cassandras, Lafortune]):



$E_{uo}=\{e_d,u,v\}$

ANALYSIS OF AUTOMATA

DIAGNOSTICS

•when the system model contains unobservable events, we may be interested to determine if some of those *could have ocurred* or *have occurred with certainty*.

• As we continue observing the system behavior, our uncertainty is reduced, but the diagnostic may not be conclusive in some cases.

• We build a modified *observer* and call it *diagnoser* G_{diag}.

• We consider, for simplicity, only one event $e_d \in E_{uo}$ and attach labels to the states of G_{diag} stating whether e_d has occurred so far (label `Y`) or not (label `N`)

DIAGNOSTICS (cont'd)

• key modifications of the construction of G_{obs} for the purpose of of building G_{diag} :

M1: when building $UR(x_0)$,

(a) attach label N to all states reachable from x_0 by unobservable strings in $[E_{uo} \setminus \{e_d\}]^*$;

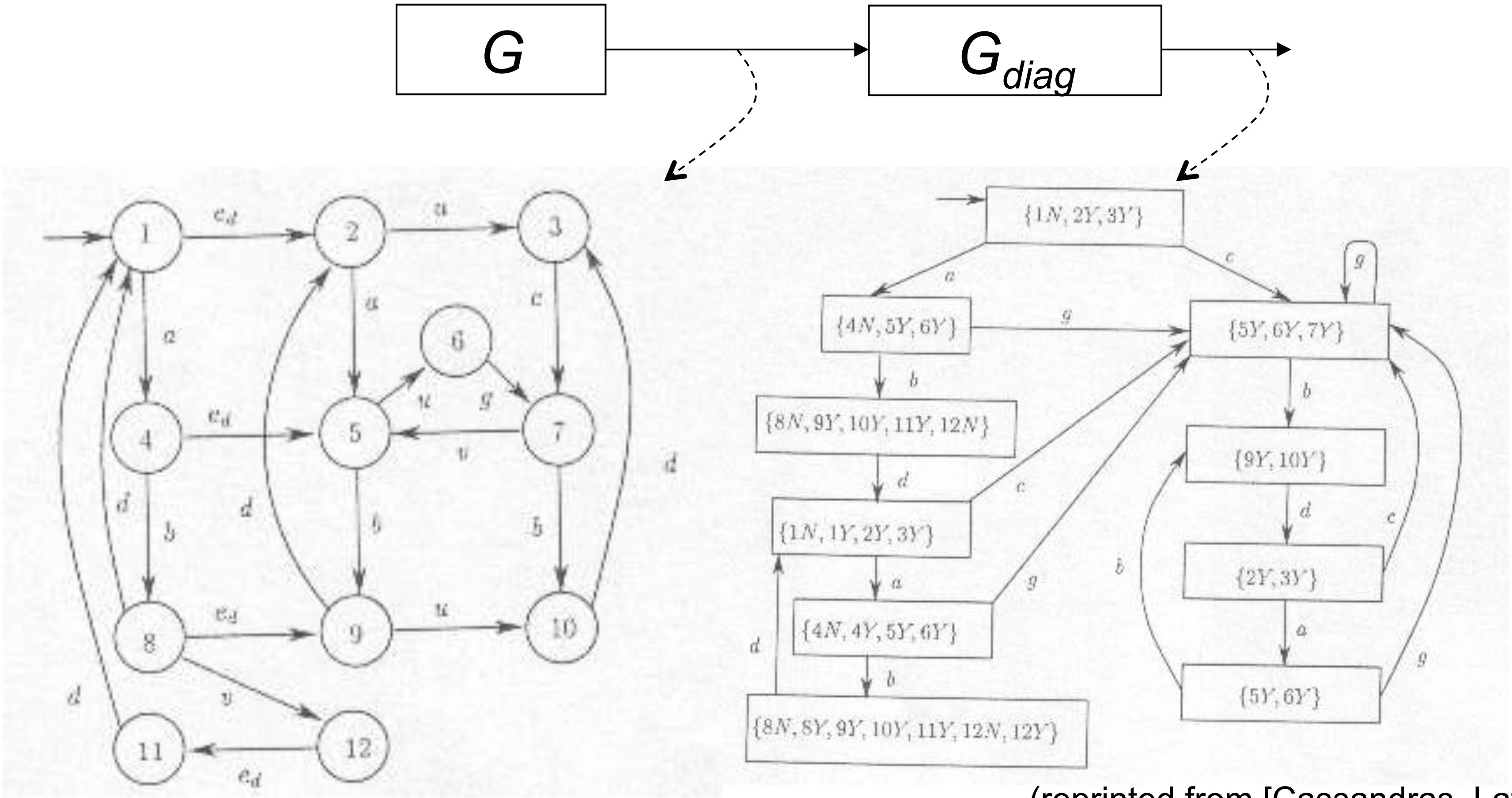
(b) attach label Y to states reachable from x_0 by unobservable strings that contain at least one occurrence of e_d;

(c) if state z can be reached both with and without executing e_d, then create two entries in the initial state of G_{diag} : zN and zY.

M2: build subsequent states of G_{diag} by following the rules for G_{obs} (with the above modified way to build unobservable reaches) and by propagating label Y

DIAGNOSTICS (example)

unobservable event to be diagnosed: e_d



(reprinted from [Cassandras, Lafortune])

LANGUAGES AND AUTOMATA

Further reading

- state space refinement
- state space aggregation (with loss of less relevant information)
- state space minimization (with no loss of information)
- model building for estimation and diagnosis

Other references

- *An Introduction to Automata, Languages and Computation,* J. Hopcroft, R. Motwani, and J. Ullman. Addison Wesley, 1979 (DEEC Library)

Acknowledgments to Paulo Tabuada, who helped preparing some of the slides in this chapter, for some sessions of an ISR/ IST Reading Group on DES and of ISR/IST Control Theory Group.