# CAS 745
# Supervisory Control of Discrete-Event Systems

## Slides 2: Linguistic Preliminaries

Dr. Ryan Leduc

Department of Computing and Software
McMaster University

*Material based on W. M. Wonham, Supervisory Control of Discrete-Event Systems, Department of Electrical and Computer Engineering, University of Toronto, July 2004. Lecture notes of Professor Wonham also used.*

## Strings - §2.1

- $\Sigma$ is a finite set of distinct symbols. ie. $\alpha, \beta, \gamma, \tau, \ldots$
- We refer to $\Sigma$ as an alphabet.
- $\Sigma^+$ is the set of all finite sequences $\sigma_1\sigma_2\ldots\sigma_k$, where $\sigma_i \in \Sigma$, $k \geq 1$.
- We denote the empty sequence $(k = 0)$ by symbol $\epsilon$ (called the empty string), with $\epsilon \notin \Sigma$.
- We denote the set of all *words* or *strings* over $\Sigma$ as $\Sigma^*$, defined as:

$$\Sigma^* := \{\epsilon\} \cup \Sigma^+$$

## Concatenation and String Length

- Let $s, t \in \Sigma^+$, and $s = \sigma_1\sigma_2\ldots\sigma_m$ and $t = \tau_1\tau_2\ldots\tau_n$.
- Concatenating strings $s$ and $t$ gives

$$st = \sigma_1\sigma_2\ldots\sigma_m\tau_1\tau_2\ldots\tau_n$$

- **Defn:** We define the operation catenation of strings, cat : $\Sigma^* \times \Sigma^* \to \Sigma^*$, as follows:

$$\begin{array}{rcll} \text{cat}(\epsilon, s) & = & \text{cat}(s, \epsilon) = s, & s \in \Sigma^* \\ \text{cat}(s, t) & = & st, & s, t \in \Sigma^+ \end{array}$$

- **Defn:** For string $s \in \Sigma^*$, we define the *length* of $s$, $|s|$, as:

$$|\epsilon| = 0, \quad |s| = k \text{ if } s = \sigma_1\sigma_2\ldots\sigma_k \in \Sigma^+$$

- For $s, t \in \Sigma^*$, we get:

$$|\text{cat}(s, t)| = |s| + |t|$$

## Languages

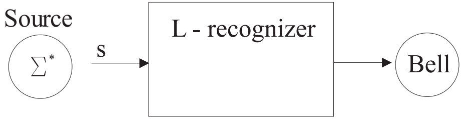
- **Defn:** We define a language $L$ over $\Sigma$ to be any subset of $\Sigma^*$. ie. $L \in \mathsf{Pwr}(\Sigma^*)$

- Examples:
  1. $L = \emptyset$ (empty language - contains no strings)
  2. $L = \{\epsilon\}$ (language is nonempty, but only contains the empty string)
  3. $L = \Sigma^*$ (all finite strings, including $\epsilon$)
  4. Let $\Sigma = \{\alpha, \beta\}$, could have:
  $$L = \{(\alpha\beta)^n |\, n = 0, 1, 2, \ldots\},$$
  where $(\alpha\beta)^0 = \epsilon, (\alpha\beta)^2 = \alpha\beta\alpha\beta$ etc.
  5. $L = \{s \in \Sigma^* |\, \#\alpha(s) = \#\beta(s)\} = \{\epsilon, \alpha\beta, \beta\alpha, \alpha\alpha\beta\beta, \alpha\beta\alpha\beta, \ldots\}$

     - ie. all finite strings that have equal number of $\alpha$ and $\beta$ symbols.

## Nerode Equivalence - §2.2

- Let $L \subseteq \Sigma^*$ be an arbitrary language.
- Want to be able to decide if a given string is a member of $L$
- Imagine a machine that beeps when the string fed in is a member of $L$.
- We start by constructing a partition on $\Sigma^*$ that is finer than $\{L, \Sigma^* - L\}$, but has a desired invariance property wrt $L$.



## Nerode Equivalence - II

- **Defn:** We define the Nerode equivalence relation on $\Sigma^*$ with respect to $L$ (or mod L) as follows:

$$(\forall s, t \in \Sigma^*)\, s \equiv_L t \text{ or } s \equiv t (\text{mod } L) \text{ iff}$$
$$(\forall u \in \Sigma^*)\, su \in L \text{ iff } tu \in L$$

- Means that if $s \equiv_L t$ then both can be continued (if at all) in same way to form a string in $L$.

- Note: if we take $u = \epsilon$, we get:

$$s\epsilon = s \in L \Leftrightarrow t\epsilon = t \in L$$

- Thus, the Nerode equivalence relation refines $\{L, \Sigma^* - L\}$.

- For the *index* (cardinality of the set of cosets) of relation $\equiv_L$, we write $||L||$.

- If $||L|| < \infty$, we say $L$ is *regular*.

## Right Congruence

- **Defn:** We say $R \in \mathcal{E}(\Sigma^*)$ is a right congruence on $\Sigma^*$ if

$$(\forall s, t, u \in \Sigma^*)\, sRt \Rightarrow (su)R(tu)$$

- Says relation $R$ is a right congruence iff the cells of $R$ are respected by *right concatenation.*

- **Theorem: 2.2.1**
  Let $L \subseteq \Sigma^*$. The Nerode equivalence (mod $L$) is the coarsest right congruence on $\Sigma^*$ that is finer than $\{L, \Sigma^* - L\} = \beta_L$.

  **Proof:** see text.

- This is equivalent to saying:

$$\equiv_L = \text{sup}\{\rho \in \mathcal{E}(\Sigma^*) |\, \rho \text{ is a right congruence and } \rho \le \beta_L\}$$

## Prefix Closure

- **Defn:** For $s \in \Sigma^*$, we say $t \in \Sigma^*$ is a prefix of $s$ and write $t \leq s$ if:
$$(\exists u \in \Sigma^*)\, s = tu$$
- **Defn:** For $L \subseteq \Sigma^*$, the prefix closure of $L$ is $\overline{L}$ defined as:
$$\overline{L} := \{t \in \Sigma^* |\, t \leq s \text{ for some } s \in L\}$$
- We have $L \subseteq \overline{L}$ as $s \leq s$. If $L \neq \emptyset$, then $\epsilon \in \overline{L}$ as $\epsilon \leq s$.
- **Defn:** A language $L$ is closed if $L = \overline{L}$.
- **Proposition 2.2.4:** Nerode equivalence $\equiv_L$ refines the partition $\{L, \overline{L} - L, \Sigma^* - \overline{L}\}$
- Cell $(\Sigma^* - \overline{L})$ (if it is nonempty) is called the *dump cell* and once entered can't be exited.

## Canonical Recognizers - §2.4

- We can use the Nerode equivalence relation to construct abstractly an automaton that can determine the current Nerode cell as a string evolves.
- We refer to the automaton as the canonical recognizer as $\equiv_L$ is the coarsest congruence that will allow us to do this.
- Fix $L \subseteq \Sigma^*$, and define our set of equivalence classes to be:

$$X := \Sigma^* / \equiv_L$$

- The canonical projection:

$$P_L : \Sigma^* \to X : s \mapsto [s]$$

- The *concatenation operator:*

$$\text{cat} : \Sigma^* \times \Sigma \to \Sigma^* : (s, \sigma) \mapsto s\sigma$$

## Canonical Recognizers - II

- The *identity operator* on $\Sigma$:

$$\mathsf{id}_\Sigma : \Sigma \to \Sigma : \sigma \mapsto \sigma$$

- Combined map:

$$P_L \times \mathsf{id}_\Sigma : \Sigma^* \times \Sigma \to X \times \Sigma : (s, \sigma) \mapsto ([s], \sigma)$$

- **Proposition 2.3.1:** There exists a unique map $\xi : X \times \Sigma \to X$ such that $\xi \circ (P_L \times \mathsf{id}_\Sigma) = P_L \circ \mathsf{cat}$, namely the following diagram commutes.

$$\begin{tikzcd}
\Sigma^* \times \Sigma \arrow[r, "\mathrm{cat}"] \arrow[d, "P_L \times id_\Sigma"'] & \Sigma^* \arrow[d, "P_L"] \\
X \times \Sigma \arrow[r, dashed, "\xi"] & X
\end{tikzcd}$$

## Canonical Recognizers Proof

**Proof:** By Proposition 1.4.2, to show the existence of $\xi$, is is sufficient to show that:

$$\mathsf{ker}\,(P_L \times \mathsf{id}_\Sigma) \leq \mathsf{ker}\,(P_L \circ \mathsf{cat})$$

Uniqueness of $\xi$ will then follow from fact that $P_L \times \mathsf{id}_\Sigma$ is surjective.

Let $((s,\sigma),(s',\sigma')) \in \mathsf{ker}\,(P_L \times \mathsf{id}_\Sigma)$ **(1)**

Must show implies: $((s,\sigma),(s',\sigma')) \in \mathsf{ker}\,(P_L \circ \mathsf{cat})$

We have: $(P_L \times \mathsf{id}_\Sigma)(s,\sigma) = (P_L \times \mathsf{id}_\Sigma)(s',\sigma')$ from **(1)** and definition of equivalence kernel.

$\Rightarrow ([s],\sigma) = ([s'],\sigma')$

$\Rightarrow s \equiv_L s'$ and $\sigma = \sigma'$

$\Rightarrow s\sigma \equiv_L s'\sigma'$ as $\equiv_L$ is a right congruence.

## Canonical Recognizers Proof - II

$\Rightarrow \mathsf{cat}(s, \sigma) \equiv_L \mathsf{cat}(s', \sigma')$

$\Rightarrow P_L(\mathsf{cat}(s, \sigma)) = P_L(\mathsf{cat}(s', \sigma'))$

$\Rightarrow (P_L \circ \mathsf{cat})(s, \sigma) = (P_L \circ \mathsf{cat})(s', \sigma')$

$\Rightarrow ((s, \sigma), (s', \sigma')) \in \mathsf{ker}\,(P_L \circ \mathsf{cat})$, as required.

**QED**

## Canonical Recognizer Definition

- For fixed $L \subseteq \Sigma^*$, define the *state set* to be:

$$X := \Sigma^* / \equiv_L$$

- Our *transition function* is the induced:

$$\xi : X \times \Sigma \to X$$

- Want string transition function so we recursively define $\hat{\xi} : X \times \Sigma^* \to X$ as follows:

$$\begin{array}{rcl} \hat{\xi}(x, \epsilon) & := & x, \quad x \in X \\ \hat{\xi}(x, \sigma) & := & \xi(x, \sigma), \quad x \in X, \sigma \in \Sigma \\ \hat{\xi}(x, s\sigma) & := & \xi(\hat{\xi}(x, s), \sigma), \quad x \in X, s \in \Sigma^*, \sigma \in \Sigma \end{array}$$

- As we are interested in strings, we will now omit the ˆ and use $\xi$ for $\hat{\xi}$.

## Canonical Recognizer Definition - II

- We define the *initial state* to be: $x_o := [\epsilon]$
- We define the set of *marker states* of $L$ to be:

$$X_m := \{[s] | \, s \in L\}$$

- Thus, if $s \in L$ then $\xi(x_o, s) = [\epsilon s] = [s] \in X_m$
- We define the *canonical recognizer* for $L$ to be the 5-tuple:

$$\mathbf{R} = (X, \Sigma, \xi, x_o, X_m)$$

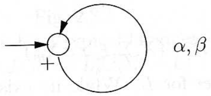
- We say a recognizer is canonical if its state set $X$ is in bijective correspondence with the equivalence classes of $\equiv_L$.

## Graph Representation

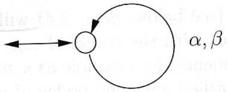
- Common representation of a recognizer is to equate its states $X$ with the nodes of a directed graph, $\mathcal{G}$.
- The edges of $\mathcal{G}$ are labelled by $\sigma \in \Sigma$.
- Thus $(x, \sigma, x')$ is an edge of $\mathcal{G}$ *iff* $\xi(x, \sigma) = x'$
- We say that $\mathcal{G}$ is a state transition graph for $\mathbf{R}$ (or $L$).

Let $\Sigma = \{\alpha, \beta\}$.

1. $L = \emptyset$

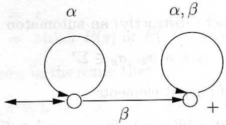


2. $L = \Sigma^*$

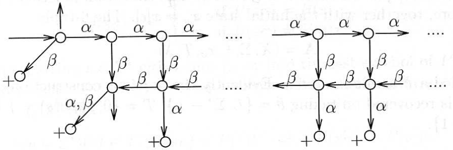


## Recognizer Examples

3. $L = \{\alpha^n | n = 0, 1, 2, ...\}$

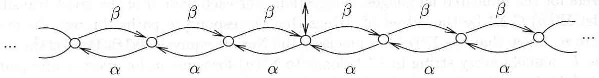


4. $L = \{\alpha^n \beta^n | n = 0, 1, 2, ...\}$.



In the above transition graph, the nodes labelled + should be merged to a single 'dump' node self-looped with $\{\alpha, \beta\}$.

5. $L = \{s | \#\alpha(s) = \#\beta(s)\}$, where $0 \le \#\sigma(s) =$ number of $\sigma$'s in the string $s$.



## Automata - §2.4

- Let: $\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m)$
- We say $\mathbf{A}$ is an automaton over the alphabet $\Sigma$ where $Y$ is a nonempty set, $y_o \in Y$, $Y_m \subseteq Y$ and $\eta : Y \times \Sigma \to Y$.
- We immediately extend $\eta$ as we did before to:

$$\eta : Y \times \Sigma^* \to Y$$

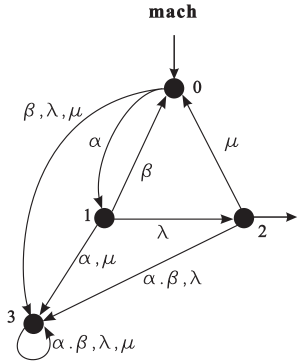
- The language $L$ recognized by $\mathbf{A}$ is:

$$L := \{s \in \Sigma^* \,|\, \eta(y_o, s) \in Y_m)\}$$

- We say that $\mathbf{A}$ is a recognizer for $L$.

## Automata Example

- For this example, $y_o = 0$, $Y_M = \{2\}$, $\Sigma = \{\alpha, \beta, \lambda, \mu\}$, and $Y = \{0, 1, 2, 3\}$.
- Eg. of $\eta$ are: $\eta(0, \alpha) = 1$, and $\eta(1, \lambda\mu) = 0$
- $L(\mathbf{A}) := \{s \in \Sigma^* | \, \eta(0, s) = 2\}$
- eg. $L(\mathbf{A}) = \{\alpha\lambda, \, \alpha\beta\alpha\lambda, \, \alpha\lambda\mu\alpha\lambda, \, \ldots\}$



**States**

0 = Idle
1 = Working
2 = Down
3 = Dump State

**Events**

$\alpha$ = machine starts job
$\beta$ = machine finishes job
$\lambda$ = machine breaks down
$\mu$ = machine is repaired

## Automata Example - II

**Claim:** $\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m) = \mathsf{Recog}(L(\mathbf{A}))$

**Sketch of Proof:** Must show that states $Y$ of $\mathbf{A}$ correspond to the cells of $\equiv_{L(\mathbf{A})}$.

Let $S_n := \{s \in \Sigma^* |\, \eta(0, s) = n\}$, for $n = 0, 1, 2, 3$

Show that $\{S_n |\, n = 0, 1, 2, 3\}$ is a partition of $\Sigma^*$.

1) Let $s, s' \in S_n$ for some $n \in \{0, 1, 2, 3\}$

Show $s \equiv_{L(\mathbf{A})} s'$

2) Show:

$$(\forall n, m \in \{0, 1, 2, 3\}), n \neq m \Rightarrow \\ (\forall s \in S_m)(\forall s' \in S_n) s \not\equiv_{L(\mathbf{A})} s'$$

## Reachable Automata

- **Defn:** For $\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m)$, a state $y \in Y$ is reachable if $(\exists s \in \Sigma^*)\, \eta(y_o, s) = y$.

- $\mathbf{A}$ is reachable if all of its states are reachable.

- Let $Y_{\mathsf{rch}} \subseteq Y$ be the subset of reachable states.

- Define the reachable subautomaton $\mathbf{A}_{\mathsf{rch}}$ of $\mathbf{A}$ to be:

$$\mathbf{A}_{\mathsf{rch}} := (Y_{\mathsf{rch}}, \Sigma, \eta_{\mathsf{rch}}, y_o, Y_{m,\mathsf{rch}})$$

  where: $\eta_{\mathsf{rch}} = \eta_{|Y_{\mathsf{rch}} \times \Sigma}$, $Y_{m,\mathsf{rch}} = Y_m \cap Y_{\mathsf{rch}}$

- Clear that $L(\mathbf{A}_{\mathsf{rch}}) = L(\mathbf{A})$.

## $\lambda$-equivalent

- Fix $\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m)$ and let $L = L(\mathbf{A})$
- **Defn:** We define equivalence relation $\lambda$ on $Y$ as:

  $$(\forall y, y' \in Y)\, y \equiv y' (\text{mod } \lambda) \text{ iff}$$
  $$(\forall s \in \Sigma^*)\, \eta(y, s) \in Y_m \Leftrightarrow \eta(y', s) \in Y_m$$

  ie. two states are equivalent if they have the same marked future.
- **Proposition 2.4.1**

  *(i)* $(\forall t, t' \in \Sigma^*)\, \eta(y_o, t) \equiv \eta(y_o, t') (\text{mod } \lambda) \Leftrightarrow t \equiv t' (\text{mod } L)$

  *(ii)*
  $(\forall y, y' \in Y)\, y \equiv y' (\text{mod } \lambda) \Leftrightarrow (\forall s \in \Sigma^*) \eta(y, s) \equiv \eta(y', s) (\text{mod } \lambda)$

  *(iii)*$(\forall y, y' \in Y)\, y \in Y_m \,\&\, y \equiv y' (\text{mod } \lambda) \Rightarrow y' \in Y_m$
- As the cosets of $\lambda$ corresponds to the cells of $\equiv_L$, we can use $\lambda$ to construct a minimal state recognizer.

## Complementary and Product Automata

- **Defn:** For $\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m)$, we define the complementary automaton as:

$$\mathbf{A}_{\mathsf{co}} = (Y, \Sigma, \eta, y_o, Y - Y_m)$$

- Easy to see that $\mathbf{A}$ recognizes $L$ iff $\mathbf{A}_{\mathsf{co}}$ recognizes the *complementary language* $L_{\mathsf{co}} = \Sigma^* - L$.

- **Defn:** If $\mathbf{A}_1$, $\mathbf{A}_2$ are automata over $\Sigma$, then the product automaton is defined to be:

$$\mathbf{A}_1 \times \mathbf{A}_2 := (Y_1 \times Y_2, \Sigma, \eta_1 \times \eta_2, (y_{1o}, y_{2o}), Y_1 m \times Y_{2m})$$

  where $\eta_1 \times \eta_2 : Y_1 \times Y_2 \times \Sigma : Y_1 \times Y_2$ is defined for $y_1 \in Y_1$, $y_2 \in Y_2$, and $\sigma \in \Sigma$:

$$(\eta_1 \times \eta_2)((y_1, y_2), \sigma) = (\eta_1(y_1, \sigma), \eta_2(y_2, \sigma))$$

- If $A_i$ recognizes $L_i$ $(i = 1, 2)$, then $\mathbf{A}_1 \times \mathbf{A}_2$ recognizes $L_1 \cap L_2$

## Generators - §2.5

- Generators are more flexible and economical way to represent a language than recognizers.

- A generator is a transition structure in which normally only a proper subset of $\Sigma$ can occur at a given stage.

- We define the generator $\mathbf{G}$ as follows:

$$\mathbf{G} = (Y, \Sigma, \eta, y_o, Y_m)$$

- Here, the transition function $\eta$ is defined at each state only for a subset of $\Sigma$, thus $\eta$ is a partial function (pfn). We write:

$$\eta : Y \times \Sigma \to Y \text{ (pfn)}$$

- For $y \in Y$ and $\sigma \in \Sigma$, we use notation $\eta(y, \sigma)!$ to mean that $\eta(y, \sigma)$ is defined.

## Generators - II

- We extend $\eta$ to $\eta : Y \times \Sigma^* \rightarrow Y$ (pfn) as follows:

$$\begin{aligned} \eta(y, \epsilon) &= y \\ \eta(y, s\sigma) &= \eta(\eta(y, s), \sigma), \ s \in \Sigma^* \end{aligned}$$

  as long as $y' := \eta(y, s)!$ and $\eta(y', \sigma)!$.

- **Defn:** We define the closed behavior of $\mathbf{G}$ to be:

$$L(\mathbf{G}) := \{s \in \Sigma^* | \, \eta(y_o, s)!\}$$

- We define the marked behavior of $\mathbf{G}$ to be:

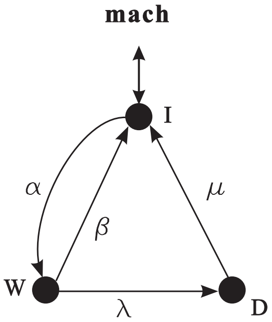
$$L_m(\mathbf{G}) := \{s \in \Sigma^* | \, \eta(y_o, s)! \ \& \ \eta(y_o, s) \in Y_m\}$$

- Clearly, $L(\mathbf{G})$ is closed, and $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$

## Generator Interpretation

- Can think of $\mathbf{G}$ as something that "generates" strings by starting at initial state, and doing only transitions that are defined by $\eta$.
- Thus $\mathbf{G}$ can be used to model the behavior of a dynamic system by only producing strings that are consistent with the systems behavior.
- $L(\mathbf{G})$ represents all possible sequences of events that could occur in the system.
- Transitions possible at a given state represent possible events at the current state of the dynamic system.
- If more than one event possible at a given state, then event selection is determined by some unmodelled system behaviour or perhaps random.

## Generator Example



**Events**

$\alpha$ = machine starts job

$\beta$ = machine finishes job

$\lambda$ = machine breaks down

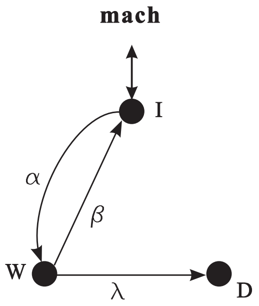
$\mu$ = machine is repaired

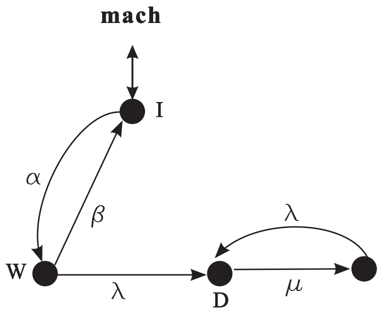
- $\Sigma = \{\alpha, \beta, \lambda, \mu\}$
- ie. $\neg\eta(I, \beta)!$ and $\neg\eta(W, \alpha)!$

## Generator Definitions

- For generator $\mathbf{G} = (Y, \Sigma, \eta, y_o, Y_m)$:
- **Defn:** A state $y \in Y$ is reachable if there is a string $s \in \Sigma^*$ such that $\eta(y_o, s)!$ and $\eta(y_o, s) = y$.
- $\mathbf{G}$ is reachable if all of its states are reachable.
- **Defn:** We say a state $y \in Y$ is coreachable if there is a string $s \in \Sigma^*$ such that $\eta(y, s)!$ and $\eta(y, s) \in Y_m$.
- $\mathbf{G}$ is *coreachable* if all of its states are coreachable.
- **Defn:** We say $\mathbf{G}$ is nonblocking if every reachable state is coreachable.
- Equivalent to saying: $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$
- **Defn:** We say $\mathbf{G}$ is *trim* if it is both coreachable and reachable.

## Generator Definitions

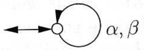




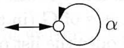
## Generator Examples

1. $L_m = \emptyset$ **EMPTY** (having empty state set)

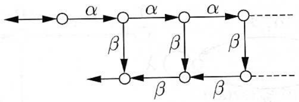
2. $L_m = \Sigma^*$



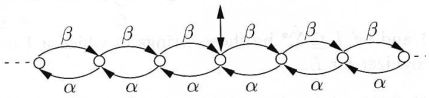
3. $L_m = \{\alpha^n | n = 0, 1, 2, ...\}$



4. $L_m = \{\alpha^n \beta^n | n = 0, 1, 2, ...\}$



5. $L_m = \{s | \#\alpha(s) = \#\beta(s)\}$



## $\lambda$-equivalent For Generators

- For generator $\mathbf{G} = (Y, \Sigma, \eta, y_o, Y_m)$:
- As for recognizers, can define equivalence relation on state set, $Y$.
- Will allow us to reduce a reachable generator to a minimal state version that represents the same closed and marked behavior.
- However, we now need Nerode equivalence relations on $\Sigma^*$ for both $L(\mathbf{G})$ (label $\equiv_c$) and $L_m(\mathbf{G})$ (label $\equiv_m$).
- **Defn:** We define equivalence relation $\lambda$ on $Y$ as:
  $(\forall y, y' \in Y)\, y \equiv y' (\text{mod } \lambda)$ iff

  i) $(\forall s \in \Sigma^*)\, \eta(y, s)! \Leftrightarrow \eta(y', s)!$

  ii) $(\forall s \in \Sigma^*)\eta(y, s)! \& \eta(y, s) \in Y_m \Leftrightarrow \eta(y', s)! \& \eta(y', s) \in Y_m$

## $\lambda$-equivalent For Generators - II

- **Proposition 2.5.1**

(i) $(\forall s, s' \in \Sigma^*)\, \eta(y_o, s) \equiv \eta(y_o, s')(\text{mod } \lambda) \Leftrightarrow s \equiv_c s' \,\&\, s \equiv_m s'$

(ii) $(\forall y, y' \in Y)\, y \equiv y'(\text{mod } \lambda) \Leftrightarrow (\forall s \in \Sigma^*)\eta(y, s) \equiv \eta(y', s)(\text{mod } \lambda)$

(iii) $(\forall y, y' \in Y)\, y \in Y_m \,\&\, y \equiv y'(\text{mod } \lambda) \Rightarrow y' \in Y_m$

- The minimal version of $\mathbf{G}$ can be constructed using the cells of $\lambda$ as states, and the induced transition function (ie. projection (mod $\lambda$)).

- The TCT procedure **minstate** does this.

## Nondeterministic Generators

- **Defn:** A nondeterministic generator is a generator where more than one transition at a given state may carry the same event label.

- We define it as the 5-tuple:

$$\mathbf{T} = (Y, \Sigma, \tau, y_o, Y_m)$$

- *Difference:* transition function now maps $(y, \sigma)$ into subsets of $Y$:

$$\tau : Y \times \Sigma \to \mathsf{Pwr}(Y)$$

- We extend $\tau$ to operate on strings by:

$$\begin{array}{rcl} \tau(y, \epsilon) & = & \{y\} \\ \tau(y, s\sigma) & = & \cup\{\tau(y', \sigma) \,|\, y' \in \tau(y, s)\}, \quad s \in \Sigma^*, \sigma \in \Sigma \end{array}$$

## Nondeterministic Generators - II

- **Defn:** The *closed behavior* is:

$$L(\mathbf{T}) := \{s \in \Sigma^* \mid \tau(y_o, s) \neq \emptyset\}$$

- The *marked behavior* is:

$$L_m(\mathbf{T}) := \{s \in \Sigma^* \mid \tau(y_o, s) \cap Y_m \neq \emptyset\}$$

## Subset Construction

- Given a nondeterministic generator $\mathbf{T} = (Y, \Sigma, \tau, y_o, Y_m)$, we can construct a deterministic generator $\mathbf{T_{det}}$ that generates the same closed and marked behavior.

- To construct $\mathbf{T_{det}}$, we take as its states the nonempty subsets of $Y$:
$$Y_{\mathbf{det}} = \mathsf{Pwr}(Y) - \{\emptyset\}$$

- The process of constructing $\mathbf{T_{det}}$ is called subset construction.

- Let $\mathbf{T_{det}} = (X, \Sigma, \xi, x_o, X_m)$ where:
$$\begin{aligned} X &:= \mathsf{Pwr}(Y) - \{\emptyset\}, & \xi(x,\sigma) &= \cup\{\tau(y,\sigma) \mid y \in x\} \\ x_o &:= \{y_o\}, & X_m &:= \{x \mid x \cap Y_m \neq \emptyset\} \end{aligned}$$

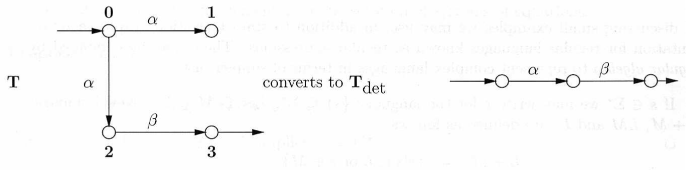
- Transition $\xi(x,\sigma)!$ iff $\cup\{\tau(y,\sigma) \mid y \in x\} \neq \emptyset$.

## Subset Construction - II

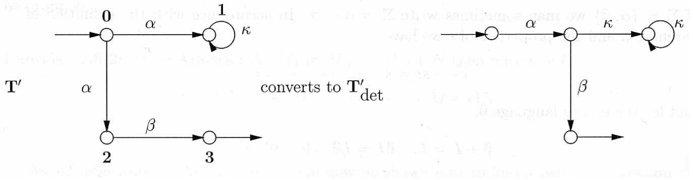
- **NOTE:** Subset construction is used in the TCT procedure **Project**, as the initial result may be nondeterministic.
- The **Project** operator constructs a new DES by removing specified events from the DES' event set.
- See defn in course notes. See also defn of *natural projection* in notes.
- TCT operations are defined in course notes as well.

# Subset Construction Problem

- **Warning:** subset construction may hide blocking situations as in the example below:



- A solution: extend $\mathbf{T}$ so that all non-coreachable are selflooped by new event label $\kappa \notin \Sigma$



## Regular Expressions - §2.6

- We will not cover this, but you should know about *regular expressions*.
- Read Section 2.6