Discrete Event Systems

Introduction

Discrete Event System (DES): a system that exhibits discrete (non-continuous change)

The control engineer's view of language and automata theory

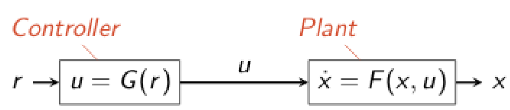
Application

- Embedded systems
- Robotics
- Verification

Outline

- Introduction to Control
- Discrete Event Systems Models
- DES Languages & Properties
- Supervisory Control

Introduction to Control



Symbol	Description
$x \in \mathcal{X}$	state
$u \in \mathcal{U}$	input
r	reference
$\mathcal{X}$	state space
F	Plant
G	Control Law

Figure 1: Controller Diagram

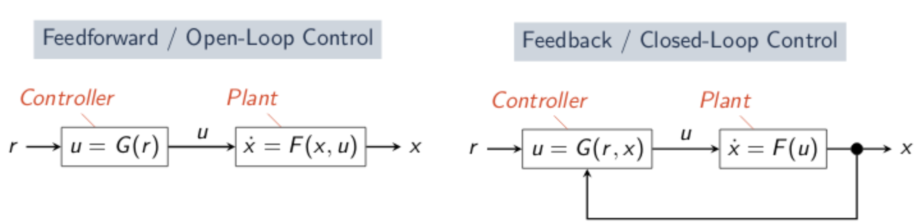


Figure 2: Feedfoward and Feedback Control

Discrete Event System Models

Discrete Event System: a discrete-state, event-drive system. That is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.

D = (X, E, f, Γ, x_0, X_m):

- X is the set of states
- E is the finite set of events
- f: X × E ↦ X is the transition function
- Γ: X ↦ P(E) is the active event function
- Γ(x) = {e ∈ E | f(x,e) is defined}
- x_0 is the initial state
- X_m is the set of marked states

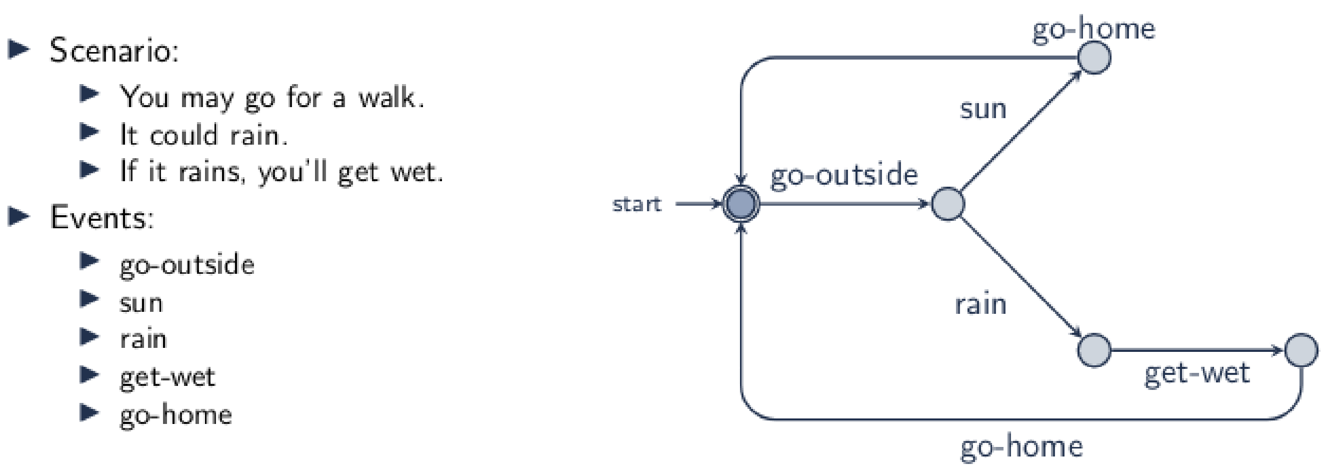


Figure 3: Example of a discrete event system

DES Languages & Properties

Extended Transition Function

Transition Function f: X × E ↦ X

Extended Transition Function $\hat{f}$: X × E* ↦ X

- Base: $\hat{f}(x, \epsilon) = x$
- Inductive: $\hat{f}(x, \alpha e) = f(\hat{f}(x, \alpha), e)$, where $\alpha \in E^*$ and $e \in E$

Define **transitions** recursively over strings

Language Marked

The language marked by D = (X, E, f, Γ, x_0, X_m) is the set of strings that take D to a final marked state:

$$L_m(D) = \{s \in E^* \mid \hat{f}(x_0, s) \in X_m\}$$

Different word for **acceptance**

Language Generated for DES

The language generated by $D = (X, E, f, \Gamma, x_0, X_m)$ is the set of string that have defined transitions in D:

$$L_g(D) = \{s \in E^* \mid \hat{f}(x_0, s) \text{ is defined}\}$$

Behaviour that is possible; but not necessarily "acceptable"

Prefix Closure

The prefix closure of language L is the set of all prefixes of strings in L:

$$\widetilde{L} = \{\alpha \in E^* \mid \beta \in E^*, \alpha\beta \in L\}$$

where E is the event set (alphabet) for L.

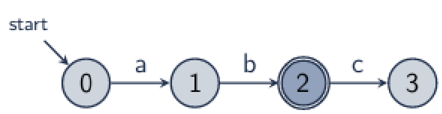
Prefix Closure Algorithm

Algorithm 1: prefix-closure

Input: $M = (Q, \Sigma, \delta, q_0, F)$
Output: $M' = (Q', \Sigma, \delta', q_0', F')$

$(Q', \delta', q_0', F') \leftarrow (Q, \delta, q_0, F)$;
function visit$(q,\ P)$ **is** // new-state $\times$ visited-states
  if $q \notin P$ **then**
    $P' \leftarrow p \cup \{q\}$;
    if $q \in F$ **then** $F' \leftarrow F' \cup P'$;
    forall $\sigma \in \Sigma$ **do** // Visit all neighbors of q
      visit$(\delta(q, \sigma), P')$;
visit$(q_0, \emptyset)$;

Example



- Marked Language:
  $$\mathcal{L}_m(D) = \{ab\}$$
- Generated Language:
  $$\mathcal{L}_g(D) = \{\varepsilon, a, ab, abc\}$$
- Prefix Closure of Marked Language:
  $$\widetilde{\mathcal{L}_m(D)} = \{\varepsilon, a, ab\}$$

Figure 4: Marked/Generated/Prefix Languages of DFA

DES Language and Prefix Relations

Every marked string is also a prefix of the marked language:

$$L_m(D) \subseteq \widetilde{L_m(D)}$$

Every marked string can be generated:

$$L_m(D) \subseteq L_g(D)$$

Every prefix of a marked string can be generated

$$\widetilde{L_m(D)} \subseteq L_g(D)$$

Deadlock

Automaton reaches a state from which no further execution is possible:

Live Lock

Automaton reaches a cycle from which a marked (accept) state is not reachable

Blocking

An automaton D is **blocking** if it can deadlock or livelock. An automaton D is **nonblocking** if neither deadlock nor livelock are possible

Blocking: we can generate a string that is not a prefix to a marked state

Nonblocking: every string we can generate is a prefix to a marked state

Supervisory Control

Supervisor Function

Supervisor Function: S: L_g (D) $\mapsto$ P(E)

- Dynamically enable/disable events in E
- Restricts the DES to desirable behavior
- For a model DES (FA) it basically gets rid of transitions (thus supervising the possible transitions)

Supervisor functions are languages

Supervised Generation

Supervised Generation: L_g (S/D)

- Basis: Contains the empty string:

  $\epsilon \in L_g(\text{S/D})$

- Inductive: Next event e in recursively-allowed string σ is allowed by the supervisor:

  $(\sigma\text{e} \in L_g(\text{S/D})) \leftrightarrow ((\sigma \in L_g(\text{S/D})) \wedge (\sigma\text{e} \in L_g(\text{D})) \wedge (\text{e} \in \text{S}(\sigma)))$ where e $\in$ E and $\sigma \in$ E*

Supervised Marking

Supervised Marked: L_m(S/D)

$L_m(\text{S/D}) = L_m(\text{D}) \cap L_g(\text{S/D})$

Supervised Language Relations

Unsupervised:

$L_m(\text{D}) \subseteq \widetilde{L_m(\text{D})} \subseteq L_g(\text{D})$

Supervised:

$\emptyset \subseteq L_m(\text{S/D}) \subseteq \widetilde{L_m(\text{S/D})} \subseteq L_g(\text{S/D}) \subseteq L_g(\text{D})$

Supervised Blocking

Blocking: $L_g(\text{S/D}) \neq \widetilde{L_m(\text{S/D})}$

- Generated strings **not** prefixes of marked strings
- Can generate unmarked strings

Nonblocking: $L_g(\text{S/D}) = \widetilde{L_m(\text{S/D})}$

- **All** generated strings are prefixes of marked strings

Supervise to restrict generation to marked prefixes

Supervision Example

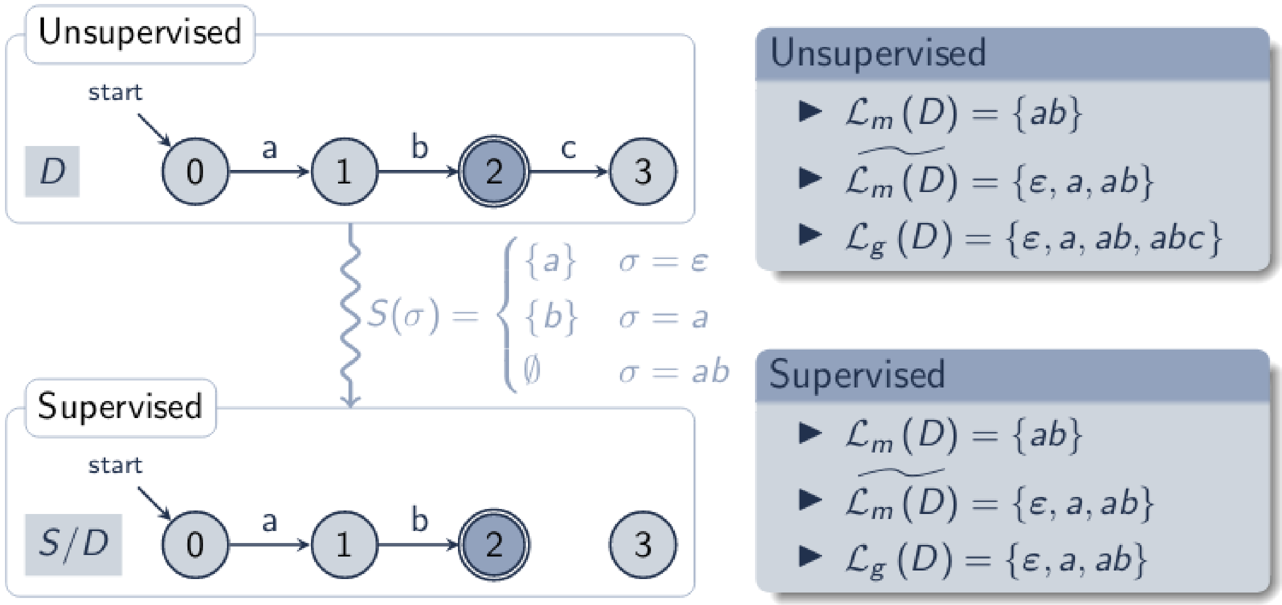


Figure 5: Example of Supervision

Specifications

Express desired behaviour using specifications

Represent (specify) language of acceptable/admissible behavior

System should only generate admissible behavior

Example:

Given E = {x, y, z}

A specification of “DONT DO X” would produce the following regex:

(y|z)*

Controllable and Uncontrollable events

Controllable Events: Events we can prevent

Uncontrollable events: Events we cannot prevent

Fully Controllable Case

All events are controllable

Supervision is a direct intersection operate:

$L_g(\mathrm{S/D}) = L_g(D) \cap \mathrm{L}^*(\mathrm{S})$

Use product automaton to find the intersection

Partially Controllable Case

Some events are uncontrollable

A simple synthesis algorithm:

1. Find "bad" states:
   - base: disallowed and blocking states in S
   - recursive: states with uncontrollable transitions to "bad" states
2. Avoid "bad" states

Fixed Point

Fixed point of a function is a value where the function's input and output are equal.

For f: X ↦ X, the fixpoint is some value x ∈ X where f(x) = x

Bad State Algorithm

Algorithm 2: partition-bad-states(D,S)

Input: $D = (X_D, E, f_D, \Gamma_D, x_{0,D}, X_{m,D})$; `// DES`

Input: $S = (X_S, E, f_S, \Gamma_S, x_{0,S}, X_{m,S})$; `// Specification`

Output: X_{bad}; `// Bad states`

$B \leftarrow$ blocking states in S;

$X' \leftarrow X_D \times X_S$;

$X_{\text{bad}} \leftarrow$

$$\left\{ (x_D, x_S) \in X' \;\middle|\; \overbrace{\exists e \in E_{uc},}^{\text{uncontrollable}} \underbrace{e \in \Gamma_D(x_D)}_{\text{allowed in } D} \wedge \left(\underbrace{f_S(x_S, e) = \emptyset}_{\text{not in } S} \vee \underbrace{f_S(x_S, e) \in B}_{\text{blocking in } S} \right) \right\};$$

$X_{\text{OK}} \leftarrow X' \setminus X_{\text{bad}}$;

$f'((x_D, x_S), e) \triangleq (f_D(x_D, e), f_S(x_S, e))$;

$X_{\text{bad}} \leftarrow$ partition-fixpoint$(X_{\text{OK}}, X_{\text{bad}}, f')$;

Partition Fixpoint Algorithm

Function partition-fixpoint($X_{\mathrm{OK}}, X_{\mathrm{bad}}, E, f$)

```
/* Uncontrollable transitions to bad states */
X' = {x ∈ X_OK | ∃e ∈ E_uc, f(x, e) ∈ X_bad};
if X' = ∅ then // base: fixpoint
    return X_bad;
else // Recurse
    return partition-fixpoint(X_OK \ X', X_bad ∪ X', f);
```