Discrete Systems

Karl-Erik Årzén

Material

- The material comes from
  - PhD course “Discrete Event Systems”, 1998
  - PhD course “Discrete & Hybrid Systems”, 2004
  - Lecture on Discrete Event Systems from the Market-Driven Systems course
  - Lecture on Discrete Event Systems from Real-Time Systems Course
- Disclaimer:
  - A lot of material comes from old pdf slides for which I do not have the source any more
  - Exported to PowerPoint slides
  - Some notation might have been lost on the way
    - And even more have been lost in my head

Outline

- Introduction
- Boolean Logic
- Automata
- Timed Automata
- Extended State Machine Formalisms
  - Statecharts
  - Grafcet
  - JGrafchart
- Petri Nets
- Industry
  - IEC 61131-3

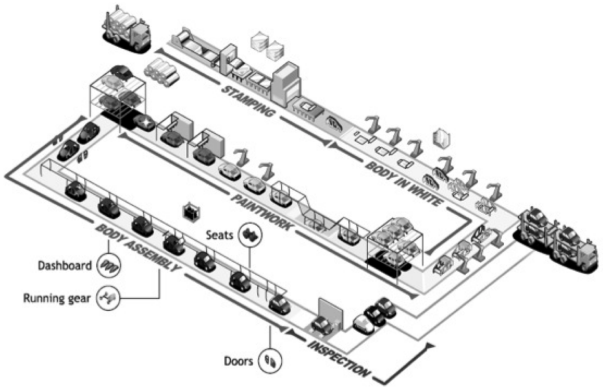
DES

Examples

Many of the systems that we deal with are discrete event systems or hybrid systems (combinations of discrete event systems and continuous systems):

- manufacturing processes
- communication networks
- computers
- transportation systems
- ...

Discrete Production Processes



DES

Academia

Large interest for DES and hybrid systems:

- Conferences, e.g., WODES - IFAC Workshop on Discrete Event Systems
- PhD projects at Swedish control departments over the years (e.g., Linköping and Chalmers)

Unfortunately, DES spread out in many different courses:

- digital systems, telecommunication systems, automatic control, industrial automation, mathematics, ...

Discrete Event Systems - Definitions

Definition:

A *Discrete Event System (DES)* is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.

Sometimes the name *Discrete Event Dynamic System (DEDS)* is used to emphasize the dynamic nature of DES.

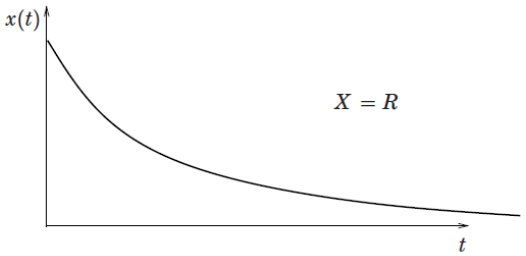
Discrete Event Systems

1. The state space is a discrete set
2. The state transition mechanism is event-driven
3. The events can be synchronized by a clock, but they do not have to be (i.e., the system can be asynchronous)

Continuous-Time Systems

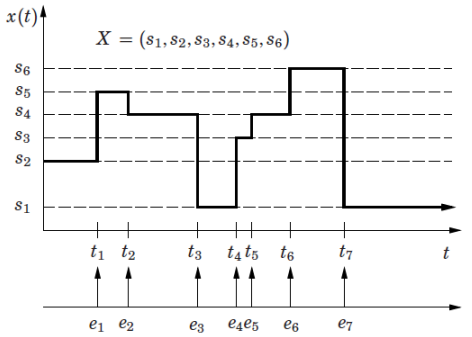
State trajectory is the solution of a differential equation

$$\dot{x}(t) = f(x(t), u(t), t)$$



Discrete-Event Systems

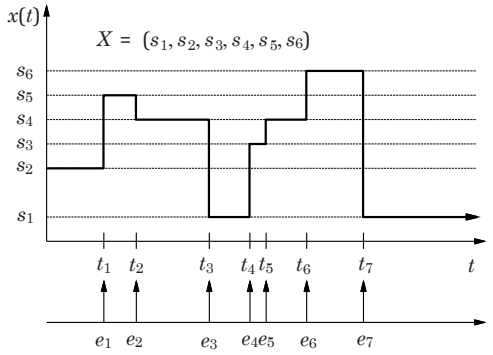
State trajectory (sample path) is piecewise constant function that jumps from one value to another when an event occurs.



DES

Untimed Models of DES

Only consider the order of the events (state changes) – untimed, or logical, languages

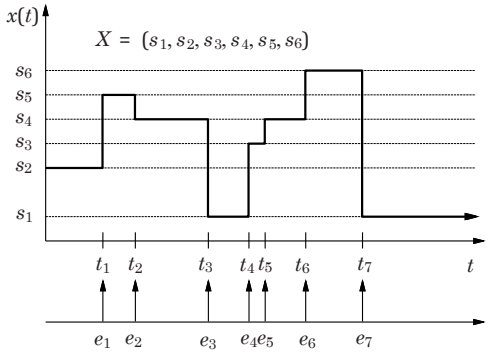


$(e_1, e_2, e_3, e_4, e_5, e_6, e_7)$ or $(s_5, s_4, s_1, s_3, s_4, s_6, s_1)$

DES

Timed Models of DES

Also consider when the events occur – timed languages



$(s_2, 0)(s_5, t_1)(s_4, t_2)(s_1, t_3)\cdots$ or $(e_1, t_1)(e_2, t_2)(e_3, t_3)(e_4, t_4)\cdots$

Examples: Timed automata, Timed Petri nets, ...

DES

Stochastic Timed Models of DES

A timed language (set of timed sequences of events) together with a probability distribution over this set.

Contain:

- event information (occurrences, orderings)
- timing information (exact times when the event occurs)
- statistical information (probabilities of different sample paths)

Examples: Stochastic timed Petri nets, Stochastic timed automata, Markov chains, Queuing theory, ...

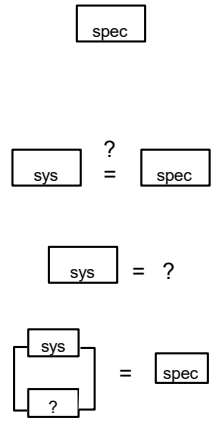
Uses of Formal Methods

Specification: Define a complete ans unambiguous syntax and semantics for describing the desired and actual system/program behaviour

Verification: Determine whether a given system/program satisfies given properties (specifications)

Analysis: Determine the behavioral characteristics (input/output state traj.) of a given dynamic system

Synthesis: Given a model of a sys-temand a specification of the desired controlled behaviour, synthesize a controller to achieve the specification



DES

Modeling Approaches

Several modeling approaches exist for DES:

Two of the more common approaches are:

- language-based modeling
- algebraic-based modeling

DES

Language-Based Modeling

A DES is modeled as a generator of a formal language.

The event set of the DES is thought of as the alphabet of the language.

Event sequences are thought of as words in the language.

An automaton generates a language by manipulating the alphabet (events) according to a specified set of rules.

State-transition based approach.

DES

Algebraic-Based Modeling

Modeling using polynomial over finite fields (ändliga kroppar).

Polynomial model = a set of polynomial equations:

$$p_1(x, u, x^+) = 0, \cdots, p_k(x, u, x^+) = 0$$

where p_i are polynomial in the state x, input u, and next state x^+.

Underlying field is $\{0, 1\}$ -> Boolean polynomials

Closely related to Boolean algebra

Boolean expression $x_1 \wedge x_2$:

$$1 - (1 - x_1)(1 - x_2) = x_1 + x_2 - x_1 x_2 \in F_2[x_1, x_2]$$

Linköping: (Germundsson, Gunnarsson)

DES

Generic Setting & Generic Problems

The events Σ are divided in two mutually exclusive groups

$$\Sigma = \Sigma_c \cup \Sigma_u$$

Σ_c : Controllable events. Can be prevented from occurring by the controller

Σ_u : Uncontrollable events. Cannot be prevented. Can, e.g. represent faults that occur.

Generic Problems:

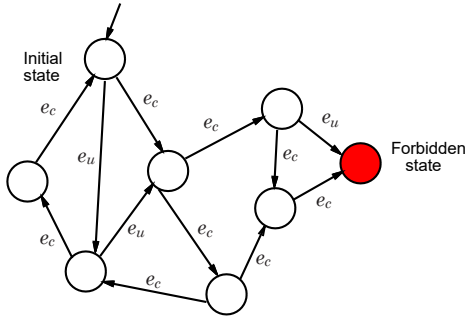
- Safety (Forbidden state) problem
- Liveness (Goal state) problem

DES

The Forbidden State Problem

The safety problem.

A certain set of dangerous states must be avoided.

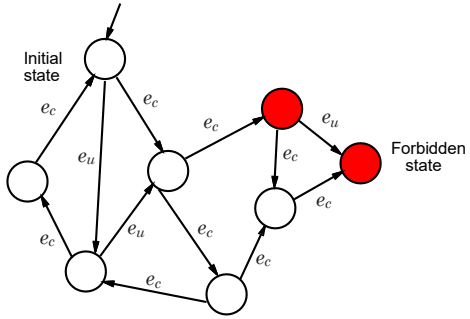


DES

The Forbidden State Problem

The safety problem.

A certain set of dangerous states must be avoided.



DES

The Goal State Problem

The liveness problem.

Find a controller that ensures that the system reaches a set of goal states (+ in minimum time) (+ for all possible initial states)

Idea:

- avoid going to states from which an uncontrollable event may take you into a state from which the goal state cannot be reached.

Boolean Logic

How do we control a machine?

Automation of the discrete operations (on-off) is largely a matter of a series of carefully timed on-off steps. The equipment performing the operations operates in an on-off manner.

Discrete signals

- Control parameters: true or false
- Actuators: on or off

Interlocks (”förreglingar”)

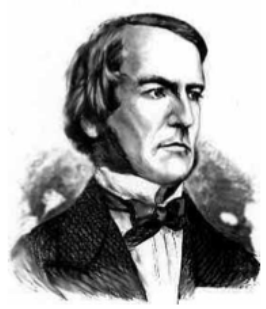
- Output = function(input)
- Boolean algebra

George Boole (1815-1864)

Boole approached logic in a new way reducing it to a simple algebra, incorporating logic into mathematics.

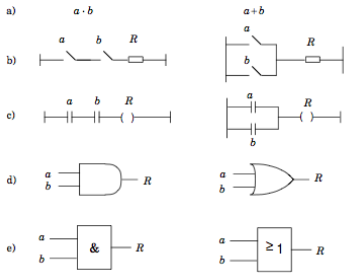
He also worked on differential equations, the calculus of finite differences and general methods in probability.

An investigation into the Laws of Thought, on Which are founded the Mathematical Theories of Logic and Probabilities (1854)



Logic:
Operations and symbols

Three types of operations			
AND	$a \cdot b$	a and b	$a \wedge b$
OR	$a + b$	a or b	$a \vee b$
NOT	$\bar{a}$ ($or\ a'$)	not a	!a



Logic: Rules

Boolean Algebra:

Example: $1 + a = 1$ and $0 + a = a$
Example: $a + \bar{a} = 1$ and $a \cdot \bar{a} = 0$
Example: $a + a = a$ and $a \cdot a = a$

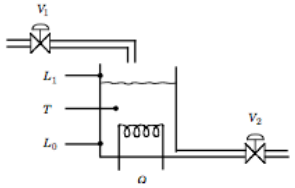
Logical Rules

Commutative	$a \cdot b = b \cdot a$
Associative	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Distributive	$a \cdot (b + c) = a \cdot b + a \cdot c$
De Morgan	$\overline{a + b} = \bar{a} \cdot \bar{b}, \quad \overline{a \cdot b} = \bar{a} + \bar{b}$

Logics: Example

Discrete logics can also be used for other types of applications, e.g., alarms.

Alarm for a batch reactor:
Raise an alarm if the temperature in the tank is too high, T, and the cooling is closed, not-Q, or if the temperature is high and the inlet valve is open, V1.



Truth table:

T	Q	V_1	y =alarm
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

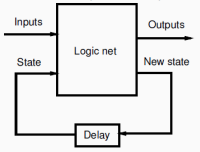
Logic (Disjunctive normal form):

$$y = T\bar{Q}\bar{V} + T\bar{Q}V + TQV$$
$$y = T\bar{Q}(\bar{V} + V) + TQV$$
$$y = T\bar{Q} + TQV$$
$$y = T(\bar{Q} + QV)$$

Logic nets

- Combinatorial nets
  - outputs = f(inputs)
  - interlocks, "förreglingar"
- Sequence nets
  - newstate = f(state,inputs)
  - outputs = g(state,inputs)
  - state machines
  - automata

Asynchronous nets or synchronous (clocked) nets



Automata

Automata Models

- Automata
- Regular expressions & languages
- Other language classes

Lots of definitions – not particularly difficult

Objectives

Assume that a discrete event system is described by an automaton.

Given an event sequence, we want to determine if each admissible sequence (trajectory) has some desired property.

Typical properties:

- stability (e.g., state convergence)
- correct use of resources (e.g., mutual exclusion)
- correct event ordering (e.g., database consistency)
- desirable dynamic behavior (e.g., no deadlocks)
- ...

In a control context, one asks if it is possible modify (by a control action) the set of admissible trajectories so that each trajectory has the desired property.

Definitions

- Σ : set of events, “alphabet”
- A *string (trace, word)* is a finite sequence of events from Σ.
- $|s|$: length of a string
- ε : empty string
- s_1 concatenated with s_2 => $s_1 s_2$

DES

- Σ^* : Kleene closure = the set of all finite strings of elements of Σ

  Example:

  $\Sigma = \{a, b\} => \Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \cdots\}$

- A language L = a subset of Σ^*

  Example: $0, \Sigma, \Sigma^*$

DES

- s' *is* a *prefix* of s if $s't = s$ with $s, s', t = \Sigma^*$ (Note: ε and s are prefixes of s)
- Languages are sets:
  - union
  - intersection
  - complement
  - difference
  - concatenation : Let $L_1, L_2 \subseteq \Sigma^*$, then

$$L_1 L_2 = \{s \in \Sigma^* : (s = s_1 s_2) \wedge (s_1 \in L_1) \wedge (s_2 \in L_2)\}$$

DES

- *Prefix closure*: The prefix-closure $\bar{L}$ of L is the language consisting of all the prefixes of all strings in L.

  Example: If $L = \{abc, cde\}$ then
  $\bar{L} = \{\varepsilon, a, ab, abc, c, cd, cde\}$

  In general, $L \subseteq \bar{L}$. L is *prefix-closed* if $L = \bar{L}$.

DES

Regular Expressions

A compact way of defining complex languages with a possibly infinite number of words.

1. $\emptyset$ is a regular expression denoting the empty set. ε is a regular expression denoting the set $\{\varepsilon\}$, and e is a regular expression denoting the set $\{e\}$ for all $e \in \Sigma$.
2. If r and s are regular expressions, then $rs, (r + s), r^*$, and s^* are regular expressions.
3. There are no other regular expressions than those constructed by applying rules 1 and 2 a finite number of times.

Example: Let $\Sigma = \{\alpha, \beta, \gamma\}$ be an alphabet. The regular expression $(\alpha + \beta)\gamma^*$ denotes the language

$$L = \{\alpha, \beta, \alpha\gamma, \beta\gamma, \alpha\gamma\gamma, \beta\gamma\gamma, \alpha\gamma\gamma\gamma, \beta\gamma\gamma\gamma, \cdots\}$$

Example: The regular expression $(\alpha\beta)^* + \gamma$ denotes the language

$$L = \{\varepsilon, \gamma, \alpha\beta, \alpha\beta\alpha\beta, \alpha\beta\alpha\beta\alpha\beta, \cdots\}$$

- Any language that can be denoted by a regular expression is a *regular language*.

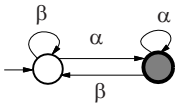
Automata and Regular Expressions

Kleene's Theorem:

Regular expressions and finite-state automata are *equivalent*, in the sense that there always exists a finite-state automaton that marks a given regular language, and, given a finite-state automaton, the language that it generates can be denoted by a regular expression.

Regular expression $(\alpha + \beta)^*\alpha$

$$L = \{\alpha, \alpha\alpha, \beta\alpha, \alpha\alpha\alpha, \alpha\beta\alpha, \beta\alpha\alpha, \beta\beta\alpha, \cdots\}$$



Automata

A *Deterministic Automaton*, denoted G is a 6-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states

Automata

A *Deterministic Automaton*, denoted G is a 6-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states
- Σ is the set of events associated with the transitions in G

DES

Automata

A *Deterministic Automaton*, denoted G is a 6-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states
- Σ is the set of events associated with the transitions in G
- $f : X \times \Sigma \rightarrow X$ is the partial transition function, $f(x, e) = x'$

DES

Automata

A *Deterministic Automaton*, denoted G is a 6-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states
- Σ is the set of events associated with the transitions in G
- $f : X \times \Sigma \rightarrow X$ is the partial transition function, $f(x, e) = x'$
- $\Sigma_G(x)$ is the active event function of G at x, i.e., the set of all events e for which $f(x, e)$ is defined

DES

Automata

A *Deterministic Automaton*, denoted G is a 6-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states
- Σ is the set of events associated with the transitions in G
- $f : X \times \Sigma \rightarrow X$ is the partial transition function, $f(x, e) = x'$
- $\Sigma_G(x)$ is the active event function of G at x, i.e., the set of all events e for which $f(x, e)$ is defined
- x_0 is the initial state

DES

Automata

A *Deterministic Automaton*, denoted G is a 6-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states
- Σ is the set of events associated with the transitions in G
- $f : X \times \Sigma \rightarrow X$ is the partial transition function, $f(x, e) = x'$
- $\Sigma_G(x)$ is the active event function of G at x, i.e., the set of all events e for which $f(x, e)$ is defined
- x_0 is the initial state
- $X_m \subseteq X$ is the set of marked states of X, completion states

DES

- Also called
  - Generator
  - State Machine
- X finite => deterministic finite automaton (DFA)

Generated and Marked Languages

Think of G as a directed graph and consider 1) all paths that can be followed from the initial state and 2) among these paths all the paths that end in a marked state.

- The language *generated* by G is

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

- The language *marked* by G is

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

DES

- Two automata are equivalent if they generate and mark the same languages.
- Blocking – the automaton reaches a state x where $\Sigma_G(x) = \emptyset$ but $x \notin X_m$. Commonly called a *deadlock*.
- Another cause of blocking is *livelock*. G enters a cycle of unmarked states with no transitions leading out.

DES

- Regular languages (denoted $\mathcal{R}$) are of practical interest
  - finite memory
  - Computer Science
  - Compilers/Parsers
- Not all languages are regular, e.g., $a^n b^n : n \geq 0$.

DES

Automata as String Acceptors

So far, an automaton has been viewed as a generator of events, i.e. the events can be seen as outputs that are generated by a state-transition.

Alternatively one can view an automaton as an acceptor (recognizer) of strings (words) from some language, i.e., the events can be viewed as inputs that trigger state transitions.

The latter view is often used in computer science.

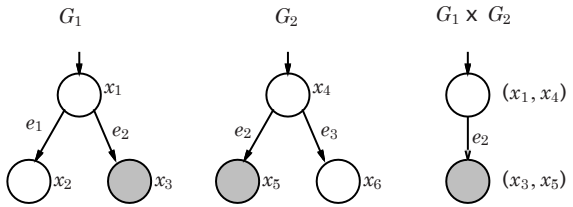
Automata Composition

Automata can be composed in different ways

- **Product composition**
- Parallel composition
- Shuffle composition

DES

Example: Product Composition

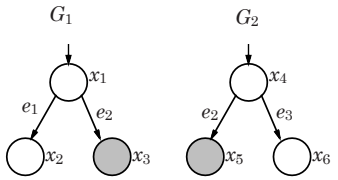


Automata Composition

Automata can be composed in different ways

- Product composition
- **Parallel composition**
- Shuffle composition

Example: Parallel Composition

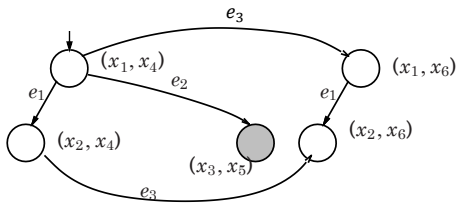


A common event can only be executed if the two automata both execute it simultaneously

The two automata are "synchronized" on the common events

The other events are not subject to such constraints and can be executed whenever possible.

$G_1 \parallel G_2$

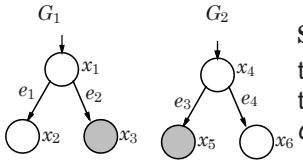


Automata Composition

Automata can be composed in different ways

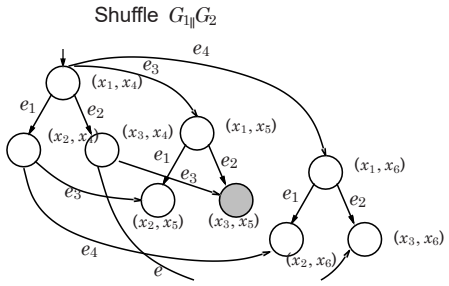
- Product composition
- Parallel composition
- **Shuffle composition**

Example: Shuffle Composition



Shuffle composition: If $\Sigma \cap \Sigma_2 = \emptyset$, then there are no synchronized transitions and $G_{1\|}G_2$ is the *concurrent* behavior of G_1 and G_2.

Shuffle $G_{1\|}G_2$



Applications of Finite Automata

- Modeling of DES, often in the context of Supervisory Control Theory
- Lexical analyzers
  - lexical-analyzer generators take as input regular expressions describing some tokens (e.g., C identifiers) and produces a finite automaton that recognizes any token
  - used as a module in a compiler
- Text editors
  - substitution of strings matching a regular expression

Other automata

- A Regular Language can be recognized by a Finite Automaton
- A Context-Free Language can be recognized by a Push-Down Automaton
- Push-down automaton: ≈
  - Finite automaton +
  - External memory consisting of a LIFO stack
- Context-free Languages and Grammars are of great practical importance
  - Defining programming languages (Backus-Naur form)
  - Formalizing the notion of parsing
  - …..
- Can describe
  - Arithmetic expressions with arbitrary nesting of balanced parenthesis
  - Block structures in programming languages

Other Classes of Languages

Chomsky's language hierarchy

Type	*Language*	*Grammar*	*Machine*
3	Regular Language	Regular Grammar	Finite Automaton
2	Context-Free Language	Context-Free Grammar	Push-Down Automaton
1	Context-Sensitive Language	Context-Sensitive Grammar	Linear-bounded Automaton
0	Recursively Enumerable Language	Unrestricted Grammar	Turing machine

DES

Litterature

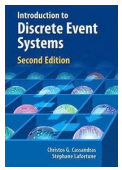
A large number of textbooks on automata theory are available.

Most of them are geared towards computer science applications of automata.

- Hopcroft, J.E. and J. D. Ullman (1979): Introduction to Automata Theory, Languages and Computation, Addison- Wesley

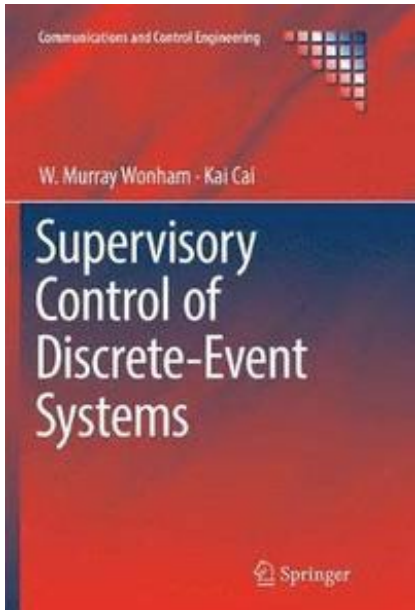
Geared towards control and Supervisory Control Theory

- Cassandras, G. C. and S. Lafortune (2010): Introduction to Discrete Event Systems, Springer-Verlag



Supervisory Control Theory (SCT)

- Ramadge–Wonham framework
  - Peter J. Ramadge & Murray Wonham, Univ of Toronto, 1982
- Method for automatically synthesizing supervisors that restrict the behavior of a plant such that as much as possible of the given specifications are fulfilled.
- The plant is assumed to spontaneously generate events.
- The events are either **controllable** or **uncontrollable**.
- The supervisor observes the string of events generated by the plant and might prevent the plant from generating a subset of the controllable events. However, the supervisor has no means of forcing the plant to generate an event.
- In its original formulation the SCT considered the plant and the specification to be modeled by formal languages, not necessarily regular languages generated by finite automata as was done in most subsequent work.



Supervisory Control Theory (SCT)

- In Sweden
  - Reglerteknik @ Chalmers
  - Bengt Lennartsson, Martin Fabian, Knut Åkesson
  - Supremica toolbox
- Wonham, M. and K. Cai: "Supervisory Control of Discrete-Event Systems", Springer Verlag

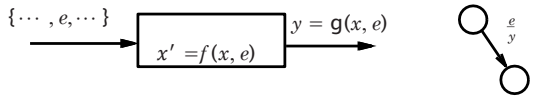
DES — State automata with output

State automata can also model discrete event systems with output.

The marked state set is usually omitted.

The definition of the automaton is expanded to also include

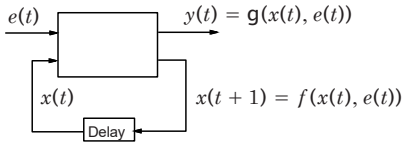
- Y – the set of output events (output set)
- the output function $g : X \times \Sigma \rightarrow Y$



DES — Mealy machines

A synchronized (clocked) network is known as a *Mealy-machine*.

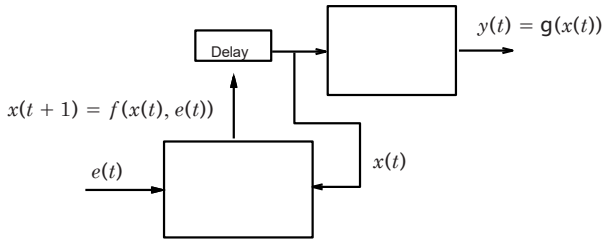
$$\begin{aligned} x(t+1) &= f(x(t), e(t)) \\ y(t) &= g(x(t), e(t)) \end{aligned}$$



DES — Moore machines

The output only depends on the current state (no direct term).

$$\begin{aligned} x(t+1) &= f(x(t), e(t)) \\ y(t) &= g(x(t)) \end{aligned}$$

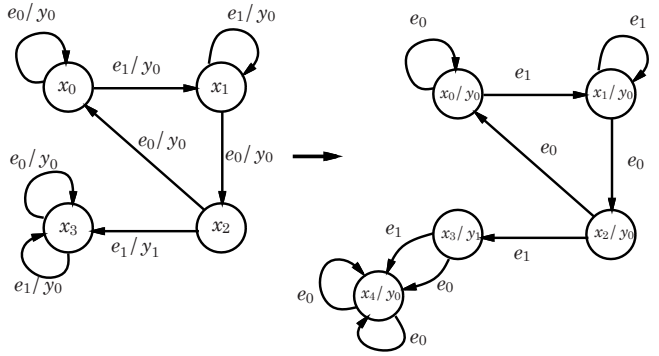


DES

- Every Mealy machine can be simulated by a Moore machine, but the Moore machine may require more states
- It is possible to convert a Moore machine to a Mealy machine
- It is possible to convert a Mealy machine to a Moore machine

DES

Mealy => Moore



Timed Automata

DES

Timed Automata

R. Alur and D.L. Dill, “A Theory of Timed Automata”, *Theoretical Computer Science*, **126**: 183–235, 1994

The framework is most often used in the computer science area for program verification, model checking, etc.

The basis for much of the discrete-based work on hybrid systems, e.g., linear hybrid automata, hybrid automata,

Timed Automata

Deterministic finite automaton augmented with a finite set of (real-valued) clocks.

The states of the automaton are called *vertices (locations)* and the transitions (arcs, edges) are called *switches*.

Switches are instantaneous.

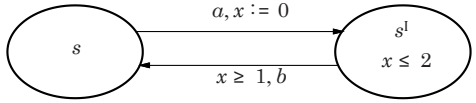
Time can elapse in locations.

A clock can be reset to zero simultaneously with any switch.

The reading of a clock equals the time elapsed since the last time it was reset (time is global).

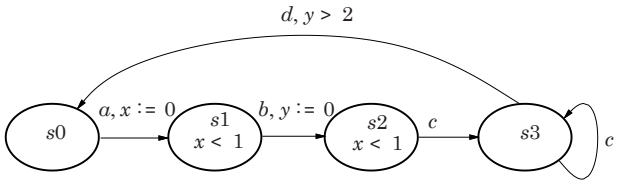
With each switch one may associate a clock constraint, and require that the switch may occur only if the current values of the clocks satisfy this constraint.

With each location we associate a clock constraint called its *invariant*, and require that time can elapse in a location only as long as its invariant stays true.



Multiple Clocks

Multiple clocks allow multiple concurrent delays.

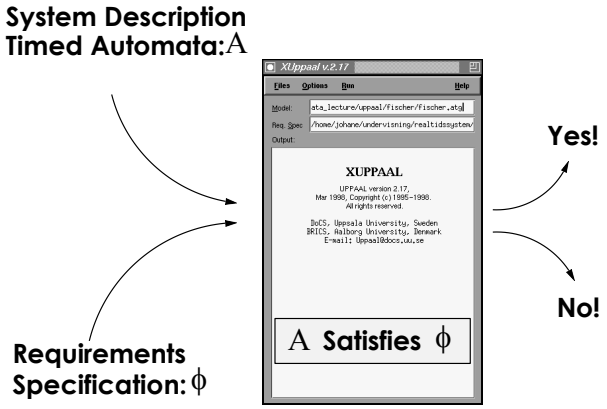


Definition

A *timed automaton* A is a tuple $\langle L, L^0, \Sigma, X, I, E \rangle$, where

- L is a finite set of locations
- $L^0 \subseteq L$ is a set of initial locations
- Σ is a finite set of labels (events)
- X is a finite set of clocks
- I is a mapping that labels each location $s \in L$ with some clock constraint in $\Phi(X)$
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is the set of switches. A switch $\langle s, a, \lambda, \phi, s^{\mathrm{I}} \rangle$ represents a transition from s to s^{I} on symbol a. ϕ is a clock constraint over X that specifies when the switch is enabled, and the set $\lambda \subseteq X$ gives the clocks to be reset with this switch.

Model Checking



- Requirements often specified using some variant of temporal logic
  - Linear Temporal Logic
  - Signal Temporal Logic
  - ……

Gautham Nayak Seetanadi's PhD thesis

State Machine Extensions

1- Statecharts

DES

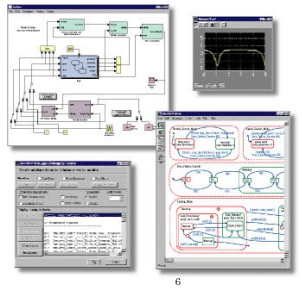
Extended State Machines

- Statecharts
- Grafcet
- JGrafchart

DES

Statecharts

Statecharts: A Visual Formalism for Complex Systems, David Harel, The Weitzmann Institute, Israel, Science of Computer Programming 8 (1987)



Currently very popular

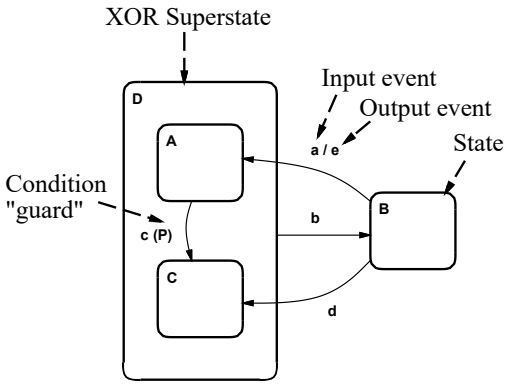
- Stateflow toolbox for Matlab
- UML tools (Rational Rose, Rational Rhapsody, MS Visio, ……)
- StateGraph in Modelica

Statecharts

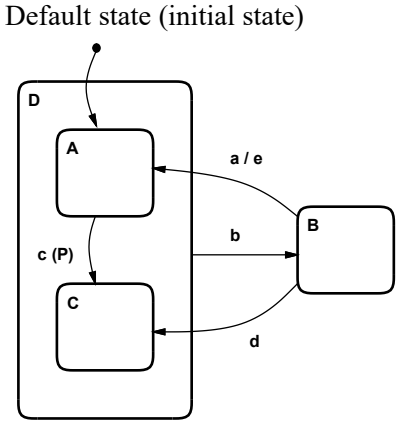
Statecharts =

- state-transition graphs +
- depth (hierarchical states) +
- orthogonality (concurrency, parallel states) +
- broadcast communication (communication between concurrent components)

Syntax

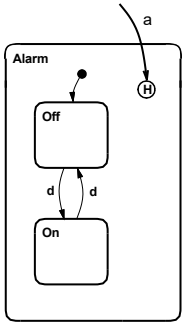


Syntax



Syntax

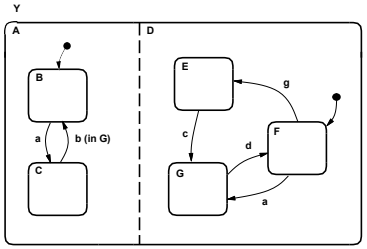
History arrows:



On event ‘a’ the last visited state within D becomes active.

Syntax

AND Superstates:



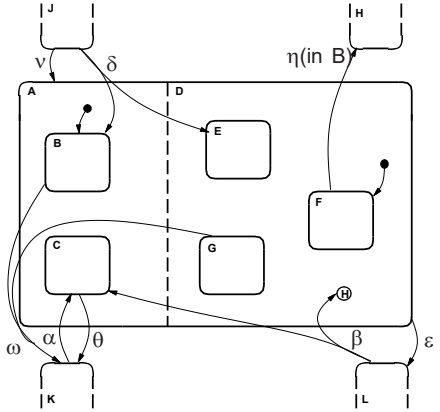
Y is the *orthogonal product* of A and D

When in state (B,F) and event **a** occurs, the system transfers *simultaneously* to (C,G).

Compare with automata composition.

Syntax

Interfaces for AND Superstates:



- δ exit from $J \Rightarrow (B, E)$
- α exit from $K \Rightarrow (C, F)$
- ν exit from $J \Rightarrow (B, F)$
- β exit from $L \Rightarrow (C$,most recently visited state in $D)$
- ω exit from $(B, G) \Rightarrow K$
- η exit from $(B, F) \Rightarrow H$
- θ exit from $(C, D) \Rightarrow K$
- ε exit from $(A, D) \Rightarrow L$

Statecharts Semantics

In Harel's first paper the formal semantics was not defined, i.e. Statecharts was an unofficial language.

In *"The STATEMATE Semantics of Statecharts", Harel and Naamad, ACM Trans. Soft. Eng. Method. 5:4, (1996)"* Harel defines "his" semantics.

By then around 20 competing semantics were around.

State Machine Extensions

2 - Grafcet

Grafcet

Defined in France 1977 by AFCET (Association Francaise pour la Cybernétique Economique et Technique) as a:

- formal specification and realization method for logical controllers

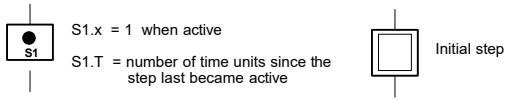
Standardized in IEC 848, IEC 61131-3 (Sequential Function Charts (SFC))

Has its roots in Petri nets and automata theory

Basic elements

Steps:

- active or inactive

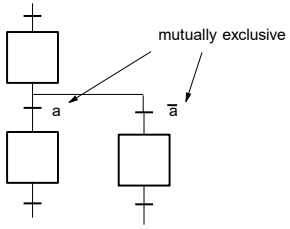


Transitions ("övergång"):

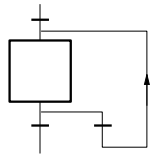
Control structures

Alternative paths:

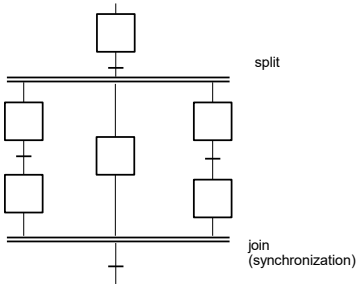
- branches



- repetition



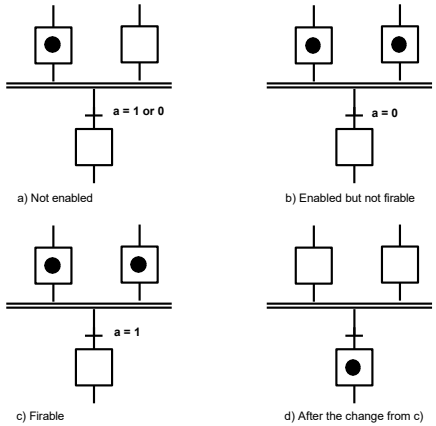
Parallel paths:



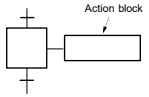
Simplified Evolution Rules

1. The initial step(s) is active when the function chart is initiated.
2. A transition is fireable if:
   - all steps preceding the the transition are active (enabled).
   - the receptivity (transition condition and/or event) of the transition is true

   A fireable transition must be fired.
3. All the steps preceding the transition are deactivated and all the steps following the transition are activated when a transition is fired

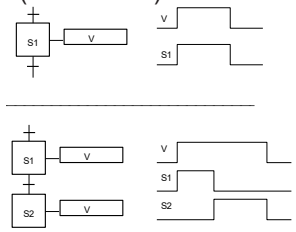


Actions



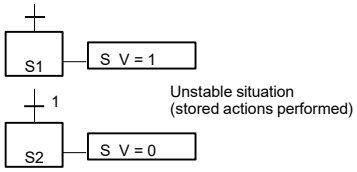
Action types:

- standard action (level action)

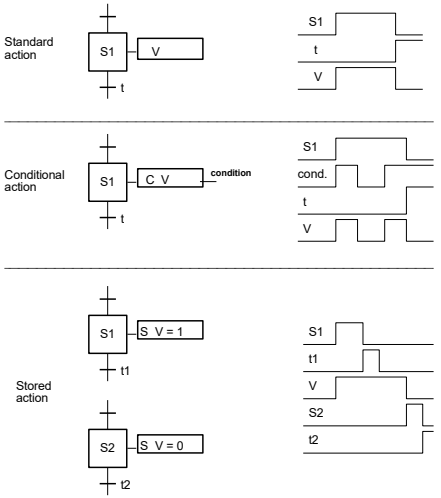


DES

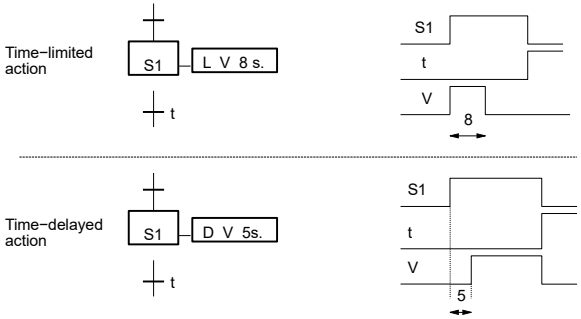
- stored action (impulse action)
  logical assignment



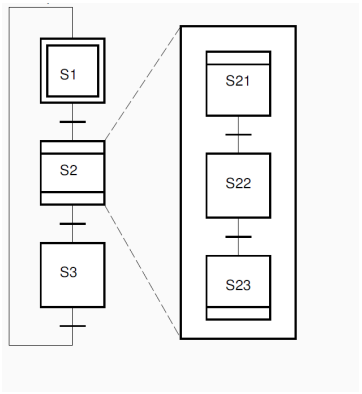
DES



DES

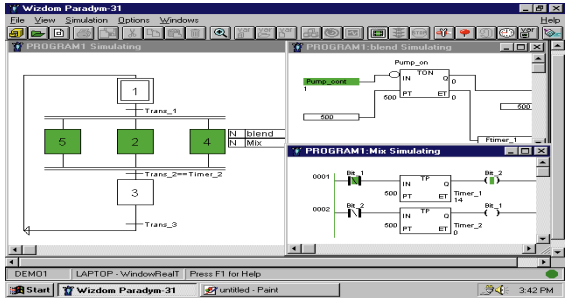


Hierarchy with Macro Steps



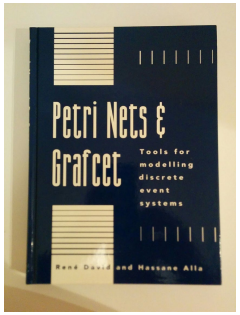
Grafcet editors

- Several Grafcet editors available
- Generate PLC code, C or even Java.
- In industry, Grafcet is known as Sequential Function Charts (SFC)
  - Slightly different semantics



Reference

- René David & Hassane Alla: “Petri Nets & Grafcet: Tools for modelling discrete event systems”, Addison Wesley
- GIPSA-Lab, Grenoble

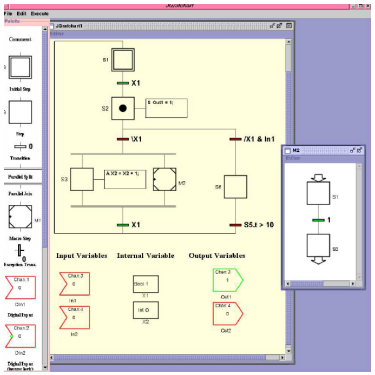


State Machine Extensions

3- JGrafchart

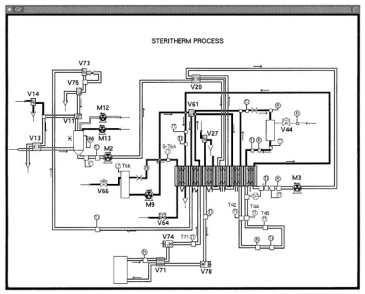
JGrafchart

- Graphical editor + runtime system (interprets the function charts) for extended Grafcet
- Combines concepts from ordinary programming languages with Grafcet
- Features
  - Exception handling
  - Procedures
  - Concurrent threads
  - …..
- Used in Lab 2 of Real-Time Systems course



Background

- IT4 (NUTEK) project 1988 – 1991: “Knowledge-Based Real-Time Control Systems”
  - ABB, Alfa-Laval Automation, Telelogic
- Digital twin technology
  - Multiple digital models and views of the plant
  - *E.g., R. Rengaswamy, Dinkar Mylaraswamy, K.-E. Årzén, V. Venkatasubramanian: (2001) A comparison of model-based and neural network-based diagnostic methods, Engineering Applications of Artificial Intelligence, Volume 14, Issue 6,*
- Used the Steritherm process from Alfa-Laval as test case
  - UHT treatment of dairy products
  - Sequence control important, i.e., Grafcet
- G2 from Gensym Corp
  - Real-time expert system framework
  - Ideally suited for implementing graphical languages
- Implemented Grafchart in G2



Background

- 1996-1998
  - Migrated Grafchart to Java
  - JGo Graphics library
- JGrafchart
- PhD Theses by Charlotta Johnsson, Rasmus Olsson and Alfred Theorin
- <Demo>

Petri Nets

DES

Petri Nets

C.A Petri, TU Darmstadt, 1962

A mathematical and graphical modeling method.

Describe systems that are:

- concurrent
- asynchronous
- distributed
- nondeterministic

Petri Nets

Can be used at all stages of system development:

- modeling
- analysis
- simulation/visualization (“playing the token game”)
- synthesis (Petri net versions of SCT)
- implementation (Grafcet)

Application areas

- communication protocols
- distributed systems
- distributed database systems
- flexible manufacturing systems
- logical controller design
- multiprocessor memory systems
- data-flow computing systems
- fault tolerant systems
- ...

Introduction

A Petri net is a directed bipartite graph consisting of places P and transitions T.

Places are represented by circles.

Transitions are represented by bars (or rectangles)

Places and transitions are connected by arcs.

In a marked Petri net each place contains a cardinal (zero or positive integer) number of tokens of marks.

Firing rules

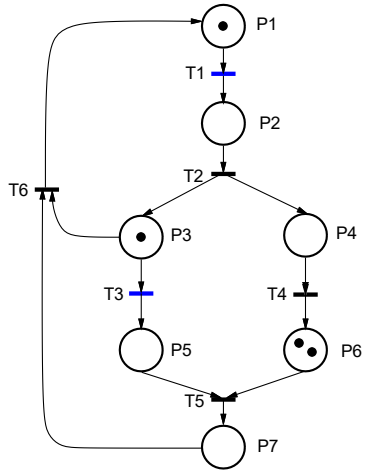
1. A transition t is enabled if each input place contains at least one token.
2. An enabled transition may or may not fire.
3. Firing an enabled transition t means removing one token from each input place of t and adding one token to each output place of t.

The firing of a transition has zero duration.

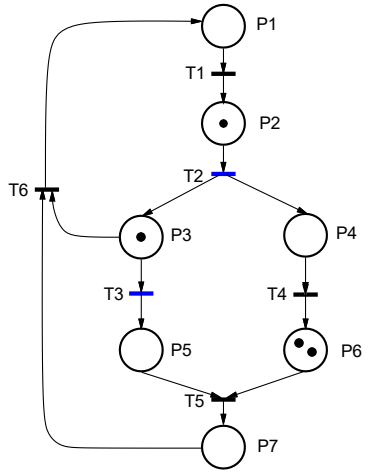
The firing of a sink transition (only input places) only consumes tokens.

The firing of a source transition (only output places) only produces tokens.

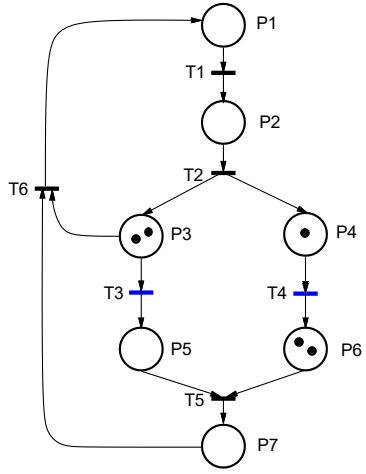
DES



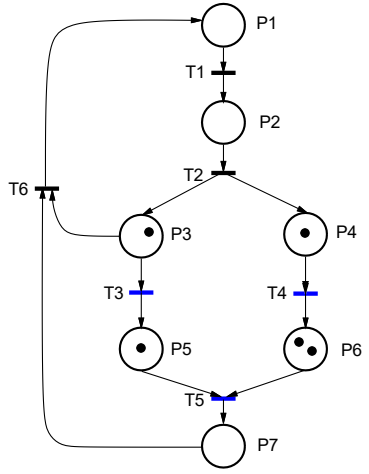
DES



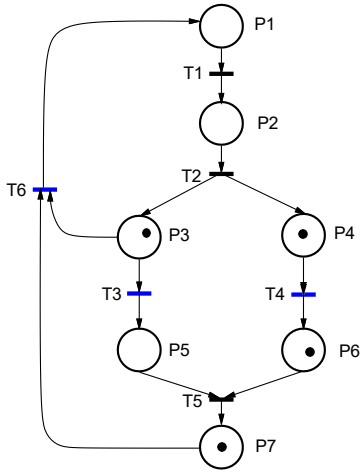
DES



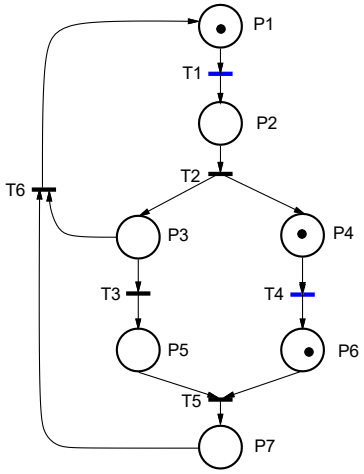
DES



DES



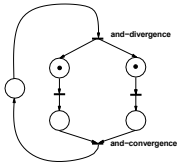
DES



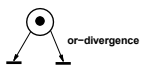
DES

Characteristics

Concurrency



Conflict (decision, choice)



DES

State

The state of a marked Petri net is its marking

$$M = [m(p_1), m(p_2), \cdots, m(p_n)].$$

The marking can be finite or infinite.

A Petri net can model an infinite system with a finite number of places.

DES

Modeling with Petri Nets

Tradeoff between

Modeling power ←→ Formal Analysis Power

Petri net specializations:

- restrictions on the allowed net structures
- more powerful analytical results and/or simpler algorithms

Petri net abbreviations:

- classes of Petri nets which always can be transformed to ordinary Petri nets
- PN properties are maintained
- “syntactic sugar”

Petri net extensions (generalizations):

- classes of Petri nets with additional transition firing rules
- cannot be transformed back to ordinary PNs
- PN properties are not always maintained
- often Turing machine equivalent

Non-autonomous Petri nets:

- transition firing synchronized and/or timed
  - Synchronized Petri Nets
    - events associated with transitions
  - Timed Petri Nets
    - time delays associated with places or transitions
  - Interpreted Petri Nets
    - Synchronized and timed PN + actions
    - Grafcet
  - Controlled Petri Nets
    - transitions can be enabled and disabled by special control places
    - Supervisory Control Theory for PN

DES

PN Specializations

State Graphs (state machines):

- an unmarked PN is a state graph iff every transition has one input place and one output place
- a marked state graph is equivalent to a automaton state machine iff it only contains one token

DES

PN Specializations

Other specializations:

- event graphs
- conflict-free PNs, free-choice PNs, simple PNs,
- pure PNs (have no self-loops)

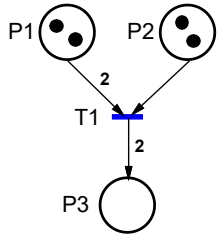
DES

PN abbreviations

- Generalized Petri nets
- Finite Capacity Petri nets
- High-Level Petri nets

DES

Generalized Petri Nets

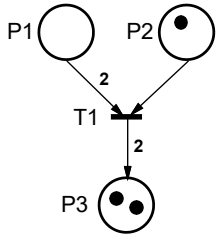


Firing rules:

1. A transition t is enabled if each input place p of t contains at least w(p,t) tokens
2. Firing a transition t means removing w(p,t) tokens from each input place p and adding w(t,q) tokens to each output place q.

DES

Generalized Petri Nets



Firing rules:

1. A transition t is enabled if each input place p of t contains at least w(p,t) tokens
2. Firing a transition t means removing w(p,t) tokens from each input place p and adding w(t,q) tokens to each output place q.

Finite-Capacity PN

Capacities (strictly positive integers) associated with places.

Transition firing rule:

- For a transition t to be enabled it is additionally required that the number of tokens in each output place p of t will not exceed its capacity $K(p)$ after firing t.

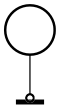
High-Level (Coloured) Petri Nets

Abstract data types (or objects) + Petri nets

A token has a type

Petri Net extensions

- FIFO nets
  - each place represents a FIFO queue where the tokens are queued
- Inhibitor arc Petri nets (zero-test PNs)
  - an inhibitor arc connects a place to a transition



  - the inhibitor arc disables the transition when the place contains tokens and enables the transition when the place is empty
- Priority Petri nets
  - PN + a partial order relation on the transitions

Petri net properties

What can we do with the nets?

What properties and problems can be analyzed?

Properties can be divided into

- structural properties – marking independent
- behavioral properties – marking dependent

Reachability

A marking M is reachable from a marking M_0 if there exists a sequence of firings that transforms M_0 to M. Denoted $M_0[T_1T_2\cdots T_n > M$

Properties

Boundedness:

- A place p_i is bounded for an initial marking M_0 if for all markings reachable from M_0, $m(p_i) \leq k$ (positive integer)

Properties

Liveness:

- A transition t_i is live for an initial marking M_0 if for every reachable marking $M_i \in^* M_0$, a firing sequence S from M_i exists, which contains transition t_i.
- A PN is live for M_0 if all its transitions are live for M_0.
- Interpretation: No matter what marking has been reached from M_0, it is possible to ultimately fire any transition of the net.

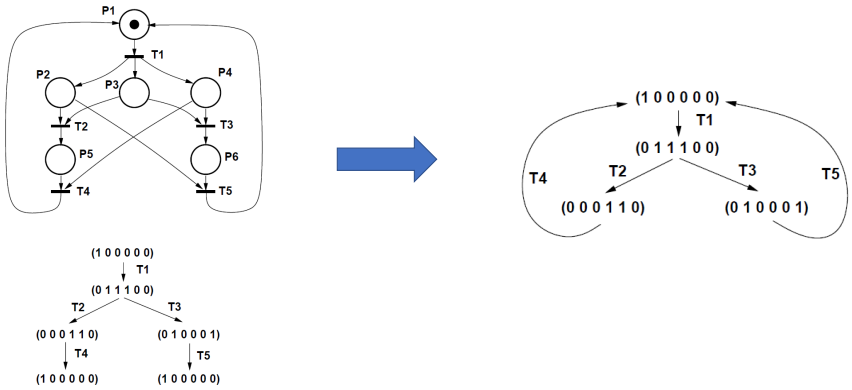
Analysis Methods

How determine if a PN has a certain property?

Three types of methods:

- **reachability (coverability) methods**
  - exhaustive enumeration of all possible markings.
- linear algebra methods
  - describe the dynamic behaviour as matrix equations
- reduction methods
  - transformation rules that reduce the net to a simpler net while preserving the properties of interest

Reachability Methods



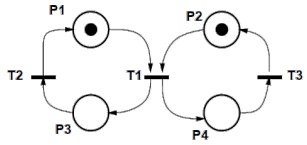
DES Analysis Methods

How determine if a PN has a certain property?

Three types of methods:

- reachability (coverability) methods
  - exhaustive enumeration of all possible markings.
- **linear algebra methods**
  - describe the dynamic behaviour as matrix equations
- reduction methods
  - transformation rules that reduce the net to a simpler net while preserving the properties of interest

Linear Algebra Methods



- Methods exist for calculating different invariants of the net
  - P-invariants
  - T-invariants

Incidence matrix:

$$W = \begin{array}{c} \begin{matrix} T_1 & T_2 & T_3 \end{matrix} \\ \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \end{array} \begin{matrix} P_1 \\ P_2 \\ P_3 \\ p_4 \end{matrix}$$

Firing sequence: $S1 = T_1T_2T_3T_1T_3$

Characteristic vector: $\underline{S1} = [2\,1\,2]^T$

Fundamental equation:

$$M_i = M_0 + W\underline{S1}$$

DES Analysis Methods

How determine if a PN has a certain property?

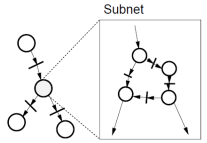
Three types of methods:

- reachability (coverability) methods
  - exhaustive enumeration of all possible markings.
- linear algebra methods
  - describe the dynamic behaviour as matrix equations
- **reduction methods**
  - transformation rules that reduce the net to a simpler net while preserving the properties of interest

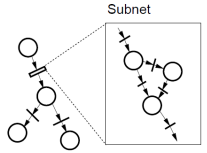
Reduction Methods

- Semi-automatic methods

Substitution places



Substitution transitions



DES

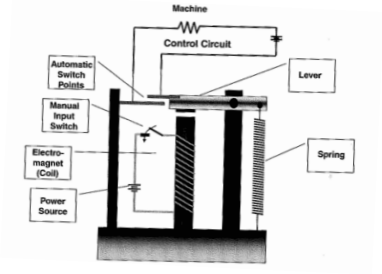
Nonautonomous PNs

- Synchronized Petri Nets
  - events associated with transitions
- Timed Petri Nets
  - time delays associated with places or transitions
- Interpreted Petri Nets
  - Synchronized and timed PN +
  - data processing part for computation of variables (actions) and transition conditions
  - very similar to Grafcet
- Controlled Petri Nets
  - transitions can be enabled and disabled by special control places
  - Supervisory Control Theory for PN

Industry

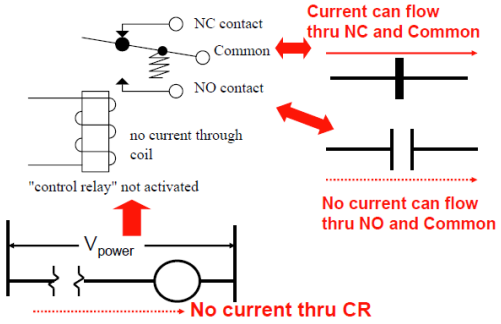
Electro-Mechanical Relays

The basic device invented for control of discrete production processes is the automatic switch and interconnected sequences of automatic switches.

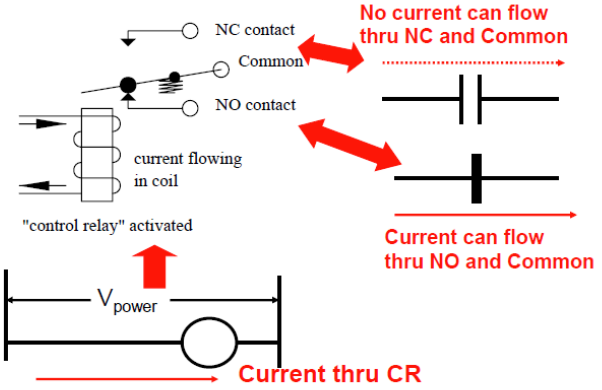


Today, the automatic switches are replaced by computer programs. This technological innovation took place in the 1960s, since then discrete systems have become more automated.

Control Relay – Not Activated



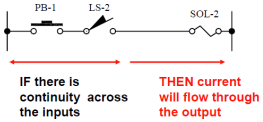
Control Relay - Activated



Logic Control

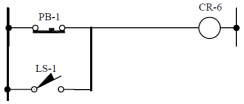
AND

IF (PB-1 is pressed) AND (LS-2 is activated)
THEN (SOL-2 will be turned on)



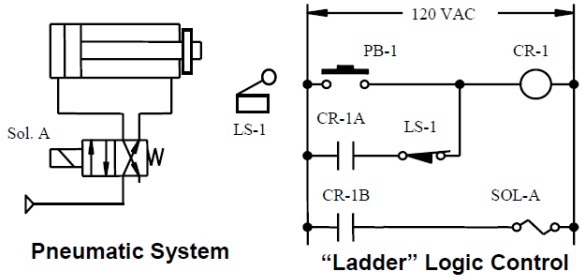
OR

IF (PB-1 is NOT pressed) OR (LS-1 is activated)
THEN (CR-6 will be turned on)



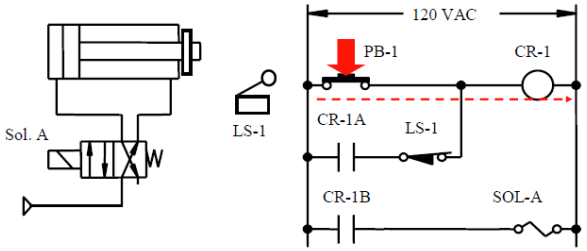
Single Cylinder Stroke #1

- Pressing the pushbutton PB-1 will cause the cylinder to extend and retract one time



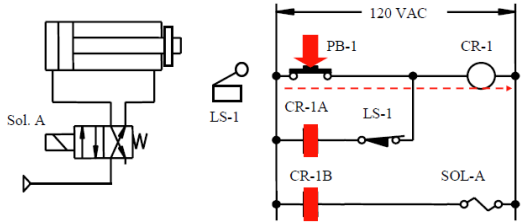
Single Cylinder Stroke #2

- Pressing the momentary contact pushbutton PB-1 energizes the control relay CR-1



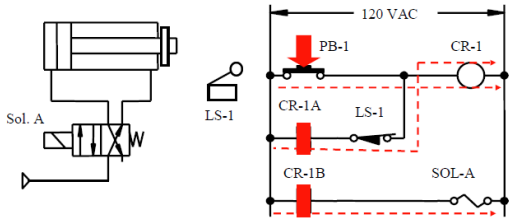
Single Cylinder Stroke #3

- After control relay CR-1 energizes, normally open contacts CR-1A and CR-1B activate



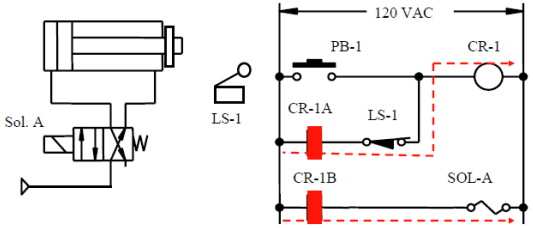
Single Cylinder Stroke #4

- Control relay CR-1 is now energized by a 2nd path, solenoid SOL-A also activates



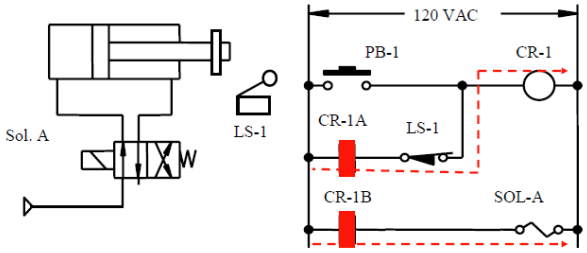
Single Cylinder Stroke #5

- PB-1 is released, but control relay CR-1 is still energized by the 2nd path ("hold" circuit)



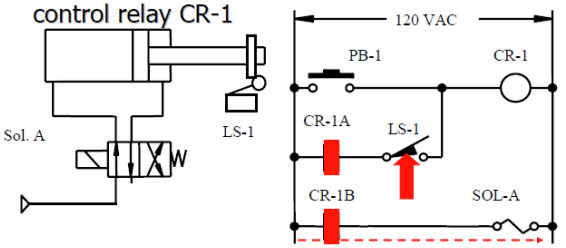
Single Cylinder Stroke #6

- Solenoid A shifts the valve spool to the right, and the cylinder begins to extend



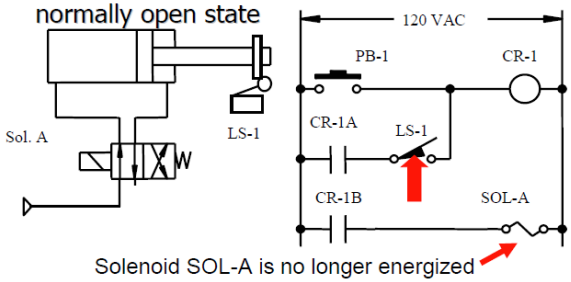
Single Cylinder Stroke #7

- Cylinder activates the normally closed limit switch LS-1, which "kills" the hold circuit for control relay CR-1



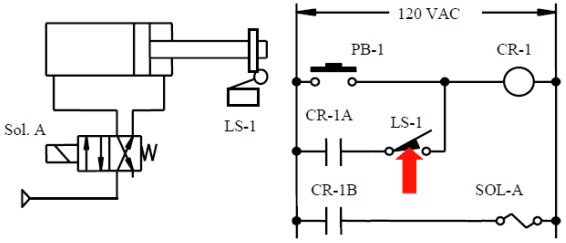
Single Cylinder Stroke #8

- With control relay CR-1 de-activated, the contacts CR-1A and CR-1B return to their normally open state



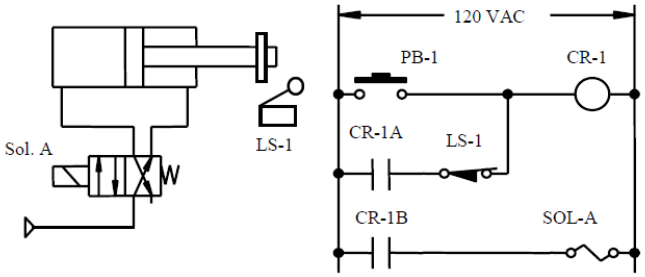
Single Cylinder Stroke #9

- CR-1B is now open, SOL-A is de-activated, spring returns valve to default state



Single Cylinder Stroke #10

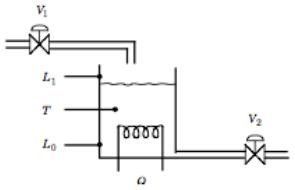
- Cylinder begins to retract, and "rolls off" of LS-1, which returns to its N.C. state

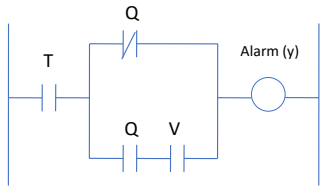


Batch Reactor Example revisited

Using ladder diagrams for simple interlocks

Alarm for a batchreactor:
Give an alarm if the temperature in the tank is too high, T, and the cooling is closed, not-Q, or if the temperature is too high, T, and the inlet valve is open,

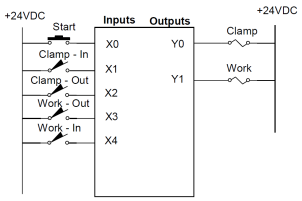


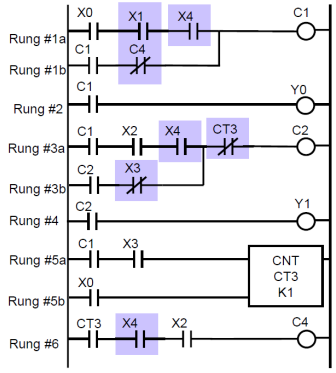


Logic:

$$y = T\bar{Q}\bar{V} + T\bar{Q}V + TQV$$
$$y = T\bar{Q}(\bar{V} + V) + TQV$$
$$y = T\bar{Q} + TQV$$
$$y = T(\bar{Q} + QV)$$

Example: Clamp/Work Layout





Relay Diagram (Ladder diagram) with counter

Each relay represents a state variable

Very user unfriendly way of programming logic

Still very common in discrete production processes
E.g. Tetra Pak

IEC 61131

- IEC standard for programmable controllers (PLCs)
- Several parts, e.g.
  - 61131-3 Programming languages
  - 61131-5 Communications
  - 61131-6 Functional safety
- Adopted by essentially all PLC vendors

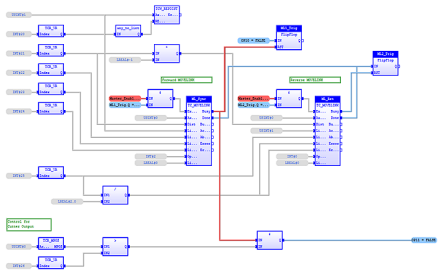
IEC 61131-3

- Defines 5 PLC programming languages
  - Function block diagrams (FBD)
  - Ladder Diagrams (LD)
  - Structrured text (ST)
  - Instruction list (IL)
  - Sequential function chart (SFC), i.e. Grafcet

+ how they may interact

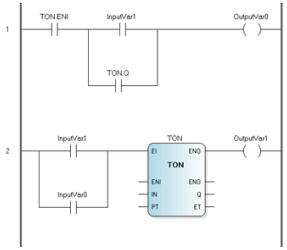
Function block diagrams (FBD)

- Graphical data-flow language
- Interconnects function blocks
- Cp. Simulink



Ladder Diagrams

- Ladder logic extended with function blocks



Structured Text

- Block-structured high-level programming language inspired by Pascal or C
- Iteration loops (WHILE, REPEAT)
- Conditional branches (IF, CASE)
- Functions

```
    END_IF;
Setpoint_IN_STAGE_1_FAILED:
    (* During 'STAGE_1_FAILED': '<S1>:119' *)
    IF (stage3_sensor <= 0) OR (stage2_sensor <= 0) THEN
        (* Transition: '(<S1>:150' *)
        (* Transition: '(<S1>:152' *)
        IF stage2_sensor > 0 THEN
            (* Transition: '<S1>:155' *)
            is_c2_Setpoint := Setpoint_IN_STAGES_1_3_FAILED;
            (* Entry 'STAGES_1_3_FAILED': '<S1>:120' *)
            rtb_stage1_setpoint := L0;
            rtb_stage2_setpoint := L0 - overall_target;
            distributed_target := rtb_stage2_setpoint;
        ELSE
            (* Transition: '<S1>:154' *)
            IF stage3_sensor > 0 THEN
                (* Transition: '<S1>:159' *)
                is_c2_Setpoint := Setpoint_IN_STAGES_1_2_FAILED;
                (* Entry 'STAGES_1_2_FAILED': '<S1>:121' *)
                rtb_stage1_setpoint := L0;
                rtb_stage2_setpoint := L0;
                distributed_target := L0 - overall_target;
            ELSE
                guard_0 := TRUE;
            END_IF;
        END_IF;
    ELSE
        guard_0 := TRUE;
    END_IF;
```

Instruction List

- Low-level textual assembly-like language
- Stack-machine oriented

```
          LD      Speed
          GT      1000
          JMPCN   VOLTS_OK
          LD      Volts
VOLTS_OK  LD      1
          ST      %Q75
```


Sequential Function Charts

- Grafcet