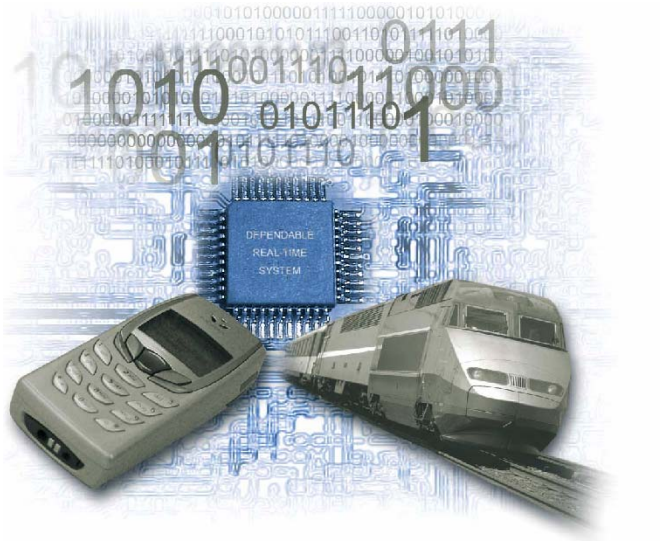
Embedded Systems



Embedded Systems

- Bernd Finkbeiner (finkbeiner@cs.uni-sb.de)
- Rüdiger Ehlers (ehlers@cs.uni-sb.de)
- Hans-Jörg Peter (peter@cs.uni-sb.de)
- Michael Gerke (micge@hotmail.com)

- Lectures:
  - Tuesday/Thursday 14:15 -15:45
- Webpage react.cs.uni-sb.de/courses/es

- Tutorial
  - time/place to be determined
  - vote for best time on doodle poll → webpage

Textbook

- Peter Marwedel.
  Embedded System Design.
  Springer, Berlin;
  2nd Print
  (1. November 2005)
  ISBN-10: 0387292373

Other Recommended Literature

- Giorgio C. Buttazzo
  Hard Real-Time Computing Systems

- Jürgen Teich,
  Digitale Hardware/Software Systeme

- Heinz Wörn,
  Uwe Brinkschulte,
  Echtzeitsysteme

Exam Policy

- Midterm/End-of-Term Exam/End-of-Semester Exam

Requirement for admission to end-of-term and end-of-semester exams:

- > 50% of points in homeworks and
- > 50% of points in midterm exam

Final grade:

- best grade in end-of-term or end-of-semester exam

Note: exam policy has been modified to ensure consistency with module description.

Embedded Systems

Embedded system =
system embedded into a large (technical) product which controls the larger system or provides information processing for it.

Estimates for number of embedded systems in current use: >10^{10}

[Rammig 2000, Motorola 2001]

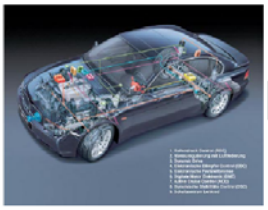
400 horses

100 microprocessors

Automotive SW Workshop
San Diego, USA
H.-G. Frischkorn
BMW Group
10 -12. Jan. 2004
Page 4

Automotive Software: An Emerging Domain
A Software Perspective

- Up to 40% of the vehicles' costs are determined by electronics and software
- 90% of all innovations are driven by electronics and software
- 50 – 70% of the development costs for an ECU are related to software
- Premium cars have up to 70 ECUs, connected by 5 system busses

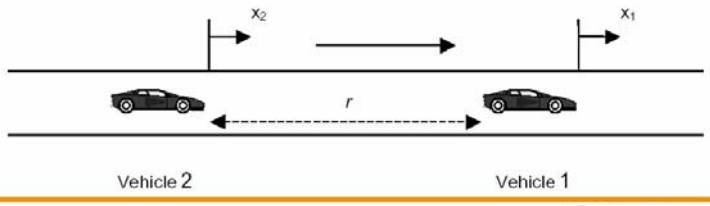


- **Growing system complexity**
- **More dependencies**
- **Costs play significant role**

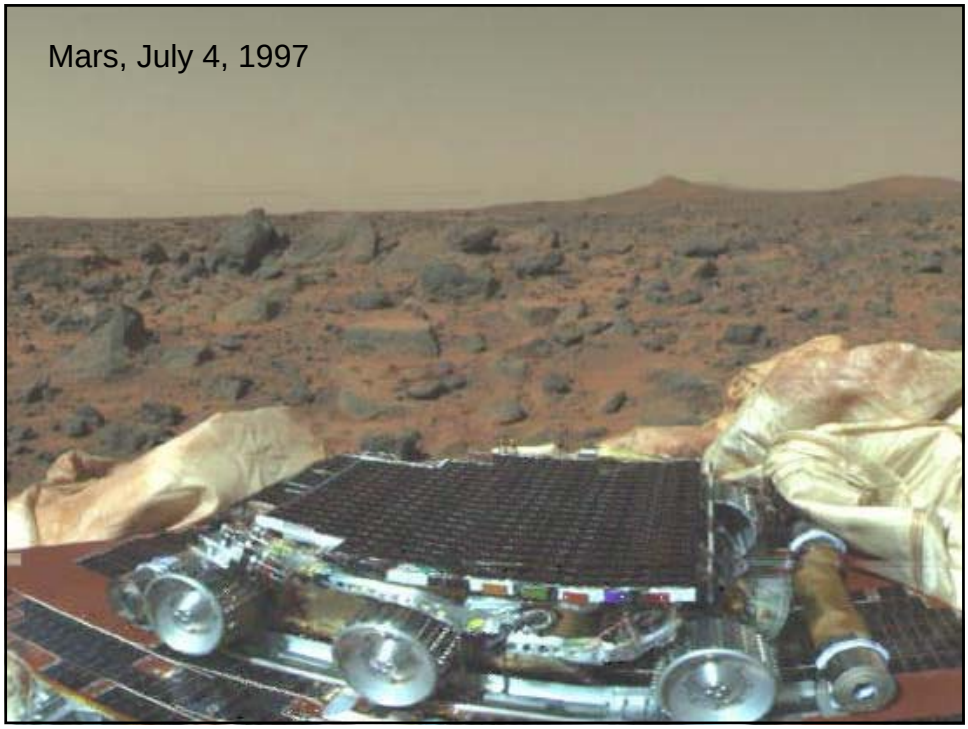
Intelligent Cruise Control

Cooperative Adaptive (Intelligent) Cruise Control (CACC):

Cruise at given speed when the road is clear (cruise control) , otherwise follow the car in front, using radar (adaptive) and/or communications (cooperative).



Thanks to PATH publication unit



The MARS Pathfinder problem

“But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".” …

The MARS Pathfinder problem

- System overview:
  - ***Information Bus (IB):***
    - Buffer for exchanging data between different tasks
    - Shared resource of two tasks M and B
  - Three tasks:
    - ***Meteorological data gathering task (M):***
      - collects meteorological data
      - reserves IB, writes data to IB, releases IB
      - infrequent task, low priority
    - ***Bus management (B):***
      - data transport from IB to destination
      - reserves IB, data transport, releases IB
      - frequent task, high priority

The MARS Pathfinder problem

- Three tasks:
  - ***...***
  - ***"Communication task" (C):***
    - medium priority, does not use IB
- Scheduling with fixed priorities.
- ***Watch dog timer (W):***
  - Execution of B as indicator of system hang-up
  - If B is not activated for certain amount of time: Reset the system

The MARS Pathfinder problem

(see http://research.microsoft.com/~mbj/Mars_Pathfinder/)

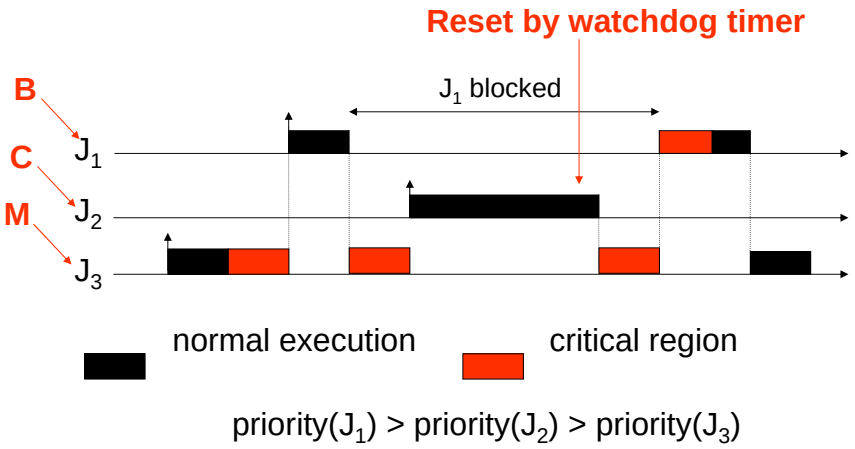
"Most of the time this combination worked fine.

However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.

After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.

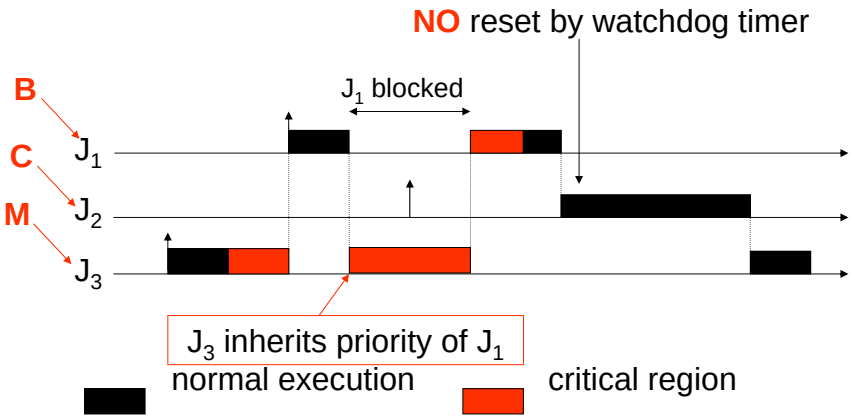
This scenario is a classic case of priority inversion."

Priority inversion



$priority(J_1) > priority(J_2) > priority(J_3)$

Classic solution: Priority inheritance



priority(J_1) > priority(J_2) > priority(J_3)

Priority inversion on Mars

- Priority inheritance also solved the Mars Pathfinder problem:
  - the VxWorks operating system used in the pathfinder implements a flag for the calls to mutual exclusion primitives.
  - This flag allows priority inheritance to be set to “on”.
  - When the software was shipped, it was set to “off”.

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to “on”, while the Pathfinder was already on the Mars [Jones, 1997].

Embedded Systems

Embedded system = **engineering artifact involving computation that is subject to physical constraints**

Constraint #1: Reaction to the physical environment

Reaction constraints: deadlines, throughput, jitter

Constraint #2: Execution on a physical platform

Execution constraints: Bounds on available processor speeds, power, hardware failure rates

Challenge: Gain control over the interplay of computation with reaction and execution constraints, so as to meet given requirements.

Characteristics of Embedded Systems

Must be **dependable**:

Reliability *R(t)* = probability of system working correctly provided that is was working at t=0

Maintainability *M(d)* = probability of system working correctly d time units after error occurred.

Availability *A*(*t*): probability of system working at time t

Safety: no harm to be caused

Security: confidential and authentic communication

Even perfectly designed systems can fail if the assumptions about the workload and possible errors turn out to be wrong.

Making the system dependable must not be an after-thought, it must be considered from the very beginning.

Characteristics of Embedded Systems

Must be **efficient:**

- **Energy** efficient
- **Code-size** efficient (especially for systems on a chip)
- **Run-time** efficient
- **Weight** efficient
- **Cost** efficient

Dedicated towards a certain application

Knowledge about behavior at design time can be used to minimize resources and to maximize robustness

Dedicated user interface

(no mouse, keyboard and screen)

Characteristics of Embedded Systems

Many ES must meet real-time constraints

A real-time system must react to stimuli from the controlled object (or the operator) within the time interval dictated by the environment.

For real-time systems, right answers arriving too late are wrong.

„A real-time constraint is called **hard**, if not meeting that constraint could result in a catastrophe“ [Kopetz, 1997].

All other time-constraints are called **soft.**

Characteristics of Embedded Systems

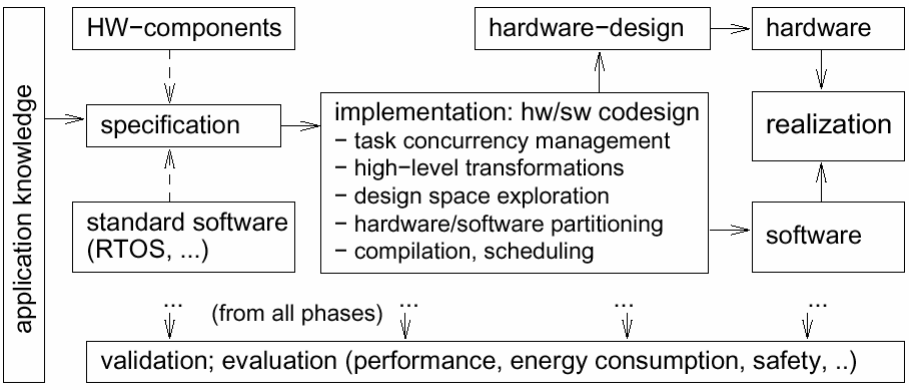
Frequently connected to physical environment through sensors and actuators.

Typically Embedded Systems are

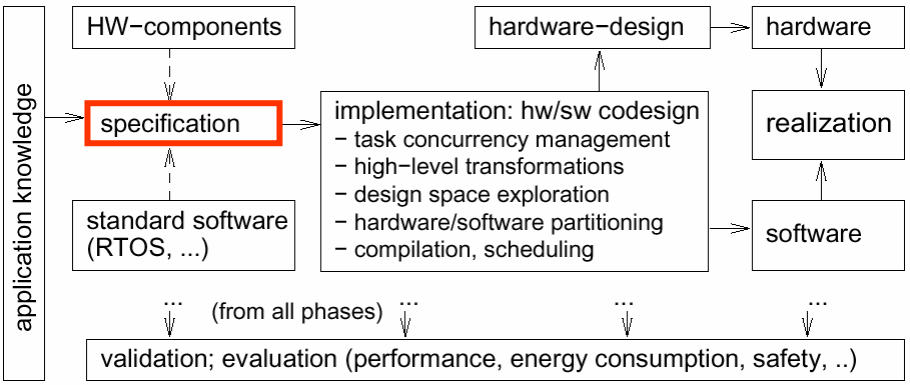
- **Hybrid systems** (analog + digital parts)
- **Reactive systems**

„A reactive system is one which is in continual interaction with is environment and executes at a pace determined by that environment“ [Bergé, 1995]
Behavior depends on input and current state.

Overview



Specifications



Specification of embedded systems: Requirements for specification techniques (1)

- **Hierarchy**
  Humans not capable to understand systems containing more than a few objects.
  Most actual systems require far more objects.
  ⇒ two kinds of hierarchy are used:
  - Behavioral hierarchy
    Examples: states, processes, procedures.
  - Structural hierarchy
    Examples: multipliers, FPUs, processors, printed circuit boards
- **Timing behavior**
- **State-oriented behavior**
  suitable for reactive systems

Requirements for specification techniques (2)

- **Event-handling** (external or internal events)
- **No obstacles for efficient implementation**
- **Support for the design of dependable systems**
  Unambiguous semantics, ...
- **Exception-oriented behavior**
  Not acceptable to describe exceptions for every state.

Requirements for specification techniques (3)

- **Concurrency**
  Real-life systems are concurrent
- **Synchronization and communication**
  Components have to communicate!
- **Presence of programming elements**
  For example, arithmetic operations, loops, and function calls should be available
- **Executability** (no algebraic specification)
- **Support for the design of large systems** (☞ OO)
- **Domain-specific support**

Requirements for specification techniques (4)

- **Readability**
- **Portability and flexibility**
- **Non-functional properties**
  fault-tolerance, availability, EMC-properties, weight, size, user friendliness, extendibility, expected life time, power consumption...
- **Adequate model of computation**

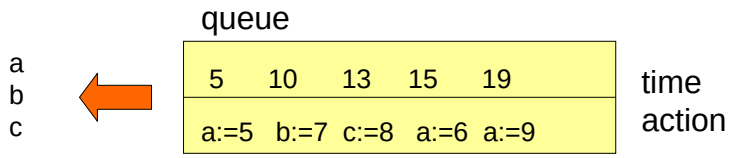
Models of computation

- **Models of computation define** [Lee, UCB, 1999]:
  - How computations of several components proceed.
  - What does it mean to be a **component**: Subroutine? Process? Thread?
  - The mechanisms by which components **interact**: Message passing? *Rendez-vous*?
  - **What components know** about each other (global variables? Implicit behavior of other components)

Models of computation
- Examples (1) -

- Communicating finite state machines (CFSMs):

- Discrete event model



Models of computation
- Examples (2) -

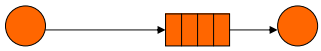
- Differential equations

$$\frac{\partial^2 x}{\partial t^2} = b$$

  - Needed for hybrid systems with analog part

Models of communication

- Asynchronous message passing



- Synchronous message passing

StateCharts

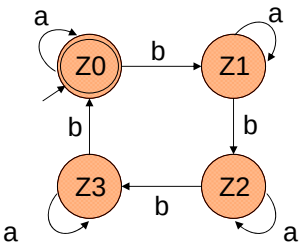
- StateCharts = the only unused combination of „flow“ or „state“ with „diagram“ or „charts“
- Based on classical automata (FSM):
  StateCharts = FSMs + Hierarchy + Orthogonality + Broadcast communication
- Industry standard for modelling automotive applications
- Appear in UML (Unified Modeling Language), Stateflow, Statemate, …
- *Warning: Syntax and Semantics may vary.*

- **Start with brief review on Finite State Machines.**

Acceptor (1)

- **Definition:**

  $M=(I, S, s_0, \delta, F)$ is a deterministic finite automaton iff
  - I is a finite, non-empty set (input symbols),
  - S is a finite, non-empty set (states),
  - $s_0 \in S$ (initial state),
  - $\delta : S \times I \rightarrow S$ (transition function),
  - $F \subseteq S$ (final states).
- Example for representation:



Acceptor (2)

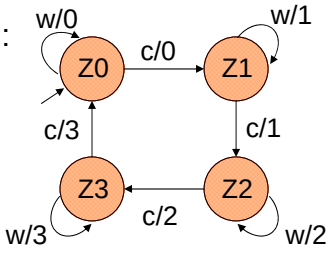
- Let
  - $w = w^0 \ldots w^{n-1}$ ($n \in \mathbb{N}$) be a finite sequence of input symbols
  - $s = s^0 \ldots s^n$ ($n \in \mathbb{N}$) be a finite sequence of states with $s^{i+1} = \delta(s^i, w^i)$ $(0 \leq i \leq n-1)$
- Then
  - $\delta^*(s^0, w) = s^n$
  - s^n is the state reached by input sequence w.
- M accepts $w = w^0 \ldots w^{n-1}$ ($n \in \mathbb{N}$) iff $\delta^*(s_0, w) \in F$.

Transducer – Mealy automaton (1)

- **Definition:**

  $M=(I, O, S, s_0, \delta, \lambda)$ is a Mealy automaton iff
  - I is a finite, non-empty set (input symbols),
  - O is a finite, non-empty set (output symbols),
  - S is a finite, non-empty set (states),
  - $s_0 \in S$ (initial state),
  - $\delta : S \times I \rightarrow S$ (transition function),
  - $\lambda : S \times I \rightarrow O$ (output function).

- Example for representation:



Transducer – Mealy automaton (2)

- Let
  - $w = w^0 w^1 \ldots$ be an infinite sequence of input symbols.
  - $s = s^0 s^1 \ldots$ be an infinite sequence of states with $s^{i+1} = \delta(s^i, w^i) \; \forall \; i \geq 0$
  - $u = u^0 u^1 \ldots$ be an infinite sequence of output symbols with $u^i = \lambda(s^i, w^i) \; \forall \; i \geq 0$
- Then
  - $\delta^*(s^0, w^0 \ldots w^{n-1}) = s^n$ for a finite prefix of w
  - s^n is the state reached by $w^0 \ldots w^{n-1}$.
  - $\lambda^*(s^0, w) = u$.
- M „transduces“ the input sequence w into the output sequence u iff $u = \lambda^*(s_0, w)$.

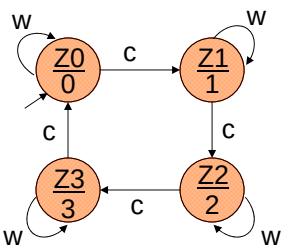
Mealy automaton (3)

In a Mealy automaton

- the output depends on the current state and the current input symbol,
- the next state depends on the current state and the current input symbol.

Transducer – Moore automaton (1)

- **Definition:**
  $M=(I, O, S, s_0, \delta, \lambda)$ is a Moore automaton iff
  - I is a finite, non-empty set (input symbols),
  - O is a finite, non-empty set (output symbols),
  - S is a finite, non-empty set (states),
  - $s_0 \in S$ (initial state),
  - $\delta : S \times I \rightarrow S$ (transition function),
  - $\lambda : S \rightarrow O$ (output function).
- Example for representation:



Transducer – Moore automaton (2)

- Let
  - $w = w^0 w^1 \ldots$ be an infinite sequence of input symbols.
  - $s = s^0 s^1 \ldots$ be an infinite sequence of states with $s^{i+1} = \delta(s^i, w^i) \; \forall \; i \geq 0$
  - $u = u^0 u^1 \ldots$ be an infinite sequence of output symbols with $u^i = \lambda(s^i) \; \forall \; i \geq 0$
- Then
  - $\delta^*(s^0, w^0 \ldots w^{n-1}) = s^n$ for a finite prefix of w
  - s^n is the state reached by $w^0 \ldots w^{n-1}$.
  - $\lambda^*(s^0, w) = u$.
- M „transduces“ the input sequence w into the output sequence u iff $u = \lambda^*(s_0, w)$.

Moore and Mealy automata

- A Moore automaton can be seen as a special Mealy automaton where the output function does not depend on the current input symbol.
- Moore and Mealy automata can be transformed into each other.

StateCharts

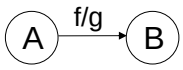
- Statecharts introduced in
  Harel: "StateCharts: A visual formalism for complex systems". Science of Computer Programming, 1987.
- More detailed in
  Drusinsky and Harel: "Using statecharts for hardware desription and synthesis", IEEE Trans. On Computer Design, 1989.
- Formal semantics in
  Harel, Naamad: "The statemate semantics of statecharts", ACM Trans. Soft. Eng. Methods, 1996.

StateCharts – Additional features compared to classical deterministic automata

- Non-determinism
- Hierarchy
- Variables with complex data types
- Concurrency
- Transitions with conditions
- Different I/O: transitions can
  - Be active depending on the presence of **events**,
  - "produce events",
  - change variables
- Timers used to produce "timeout events".

Non-deterministic transitions

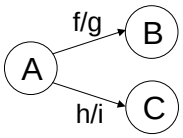
Edge label (simple version):



- Transition from A to B iff event f is present.
- Effect of transition from A to B: Event g is produced.
- Events may be
  - Present or
  - Not present
- Events may be
  - External events (provided by the *environment*)
  - Internal events (produced by internal transitions)
- Produced events exist only for one step.

Non-deterministic transitions

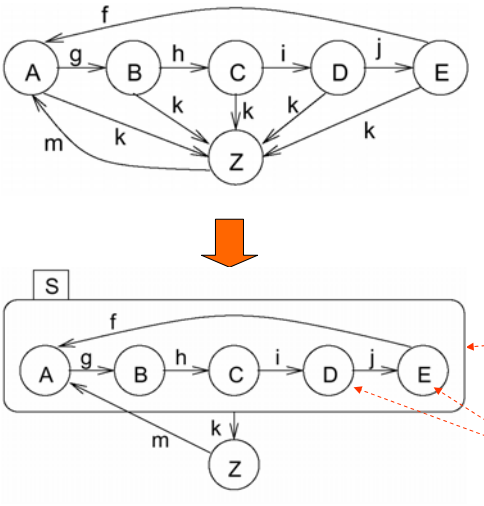
- Non-determinism:



- Events f and h may be present **at the same time**.

⇒ Non-deterministic transitions,
different behaviours are possible

Introducing hierarchy



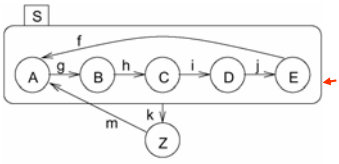
FSM will be **in** exactly one of the substates of S if S is **active** (either in A or in B or ..)

superstate

substates

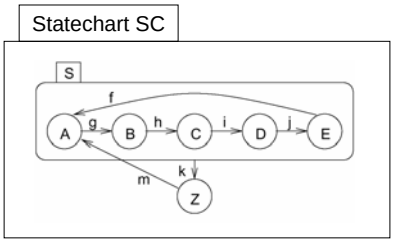
Definitions

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states.**
- States containing other states are called **super-states**.
- For each basic state s, the super-states containing s are called **ancestor states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.

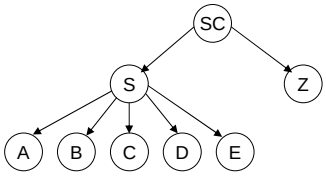


ancestor state of E

Hierarchy



- Hierachy information may be represented by a hierarchy tree with basic states as leaves.

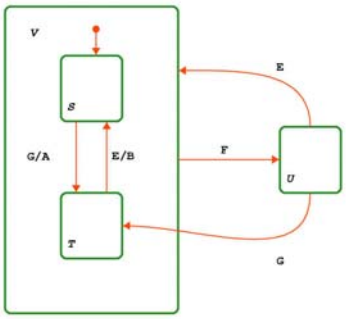


• Transitions between all levels of hierarchy possible!

• When a basic state is active, then all its ancestor states are active, too.

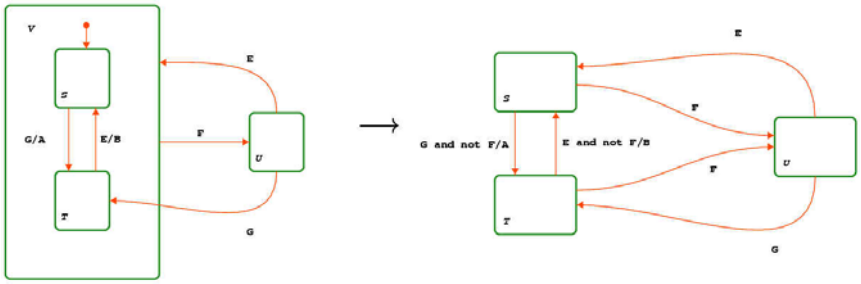
Hierarchy - priority rules for transitions (1)

- What happens, if
  - state S is active and
  - events G and F are present?

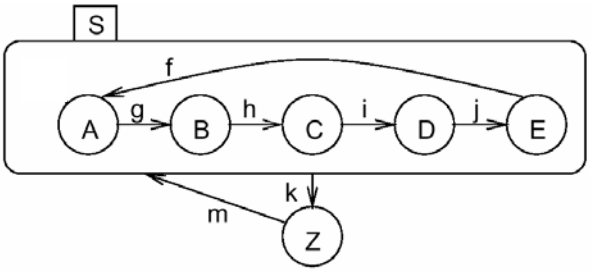


Hierarchy - priority rules for transitions (2)

- Priority of „higher level“ transitions over „lower level“ transitions!



Hierarchy - transitions to super-states



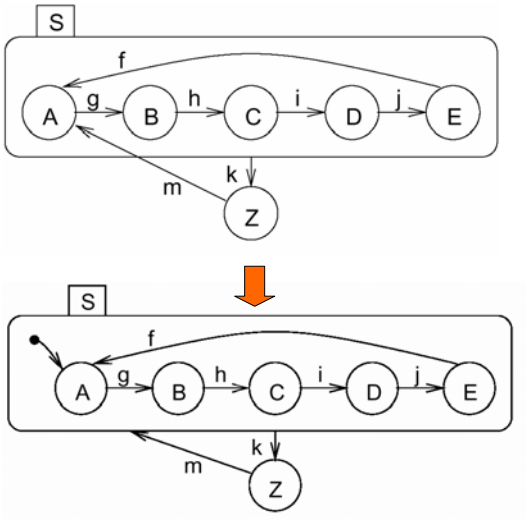
- What is the meaning of transitions to superstates,
  i.e., what basic state is entered when a superstate is entered?

⇒ default state mechanism

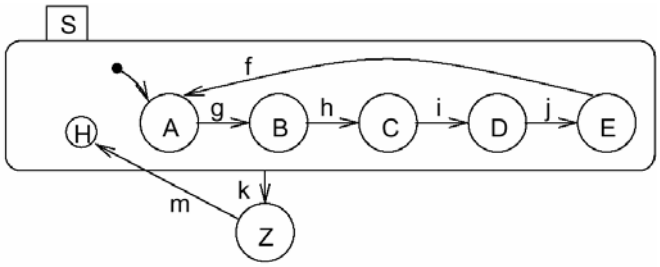
⇒ history mechanism

Default state mechanism

- Filled circle indicates sub-state entered whenever super-state is entered.
- Not a state by itself!
- Allows internal structure to be hidden for outside world

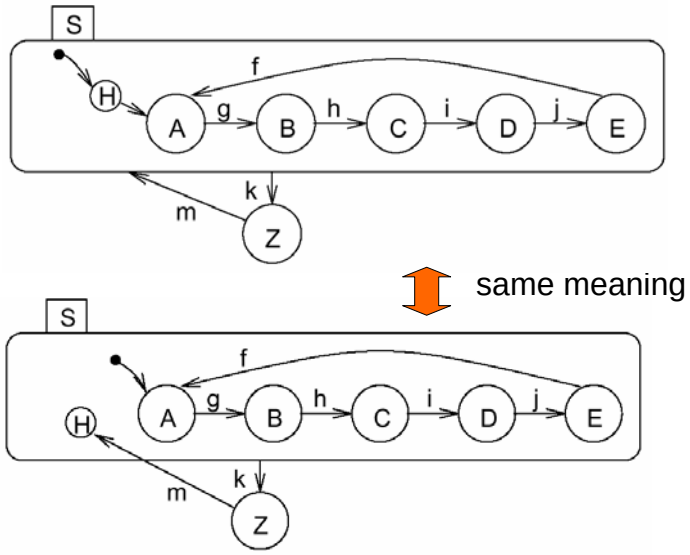


History mechanism



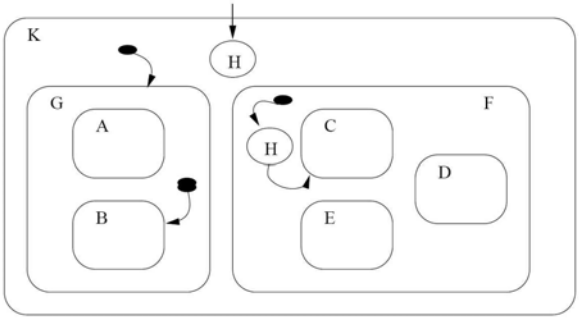
- For event m, S enters the state it was in before S was left (can be A, B, C, D, or E). If S is entered for the very first time, the default mechanism applies.

Combining history and default state mechanism

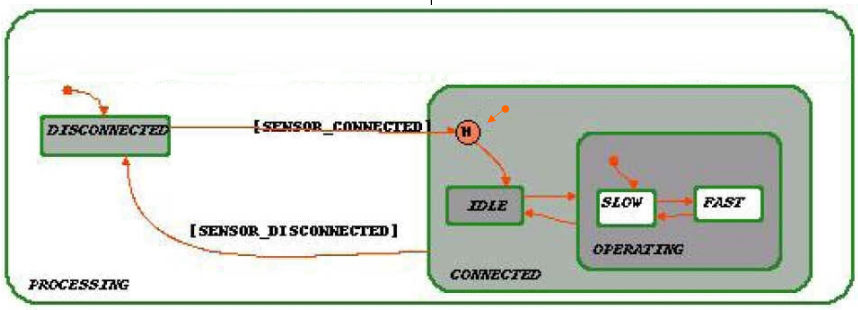


History and default state mechanism

- History and default mechanisms may be used at different levels of hierarchy.

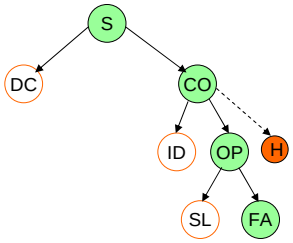


History and deep history



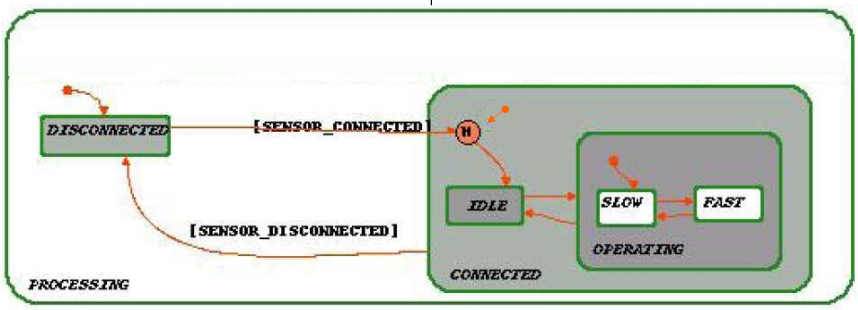
Default states

Active states



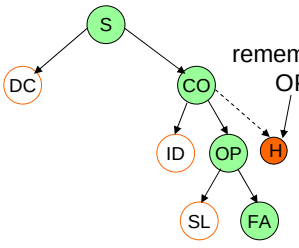
History connectors remember states **at the same level** as the history connector!

History and deep history



Default states

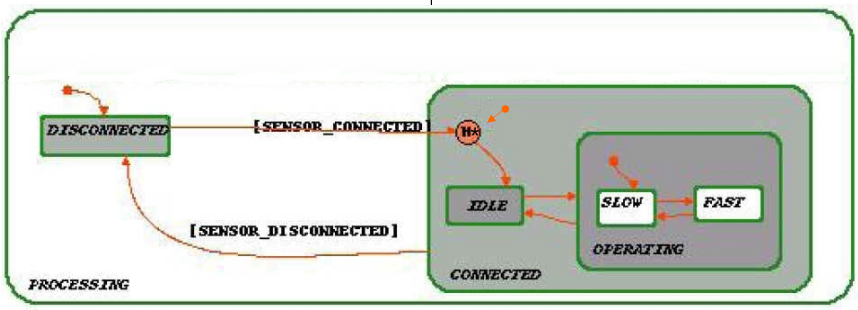
Active states



remembers OP

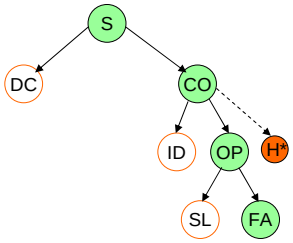
What state is entered after sequence SENSOR_DISCONNECTED, SENSOR_CONNECTED?

History and deep history



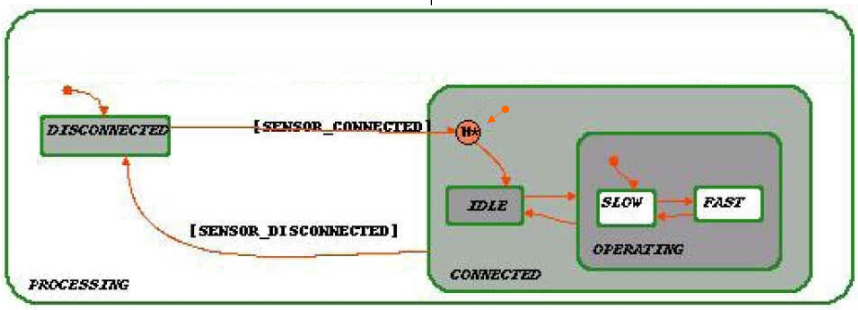
Default states

Active states



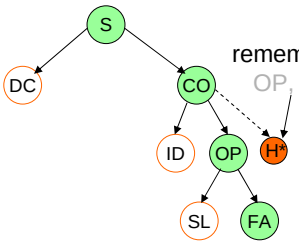
Deep history connectors H* remember **basic states**!

History and deep history



Default states

Active states



remembers OP, FA

What state is entered after sequence SENSOR_DISCONNECTED, SENSOR_CONNECTED?

Variables with complex data types

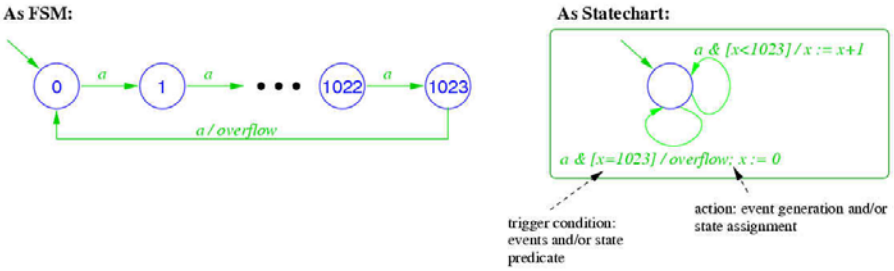
Problem of classical automata:

- Both control and data have to be represented as graphical states

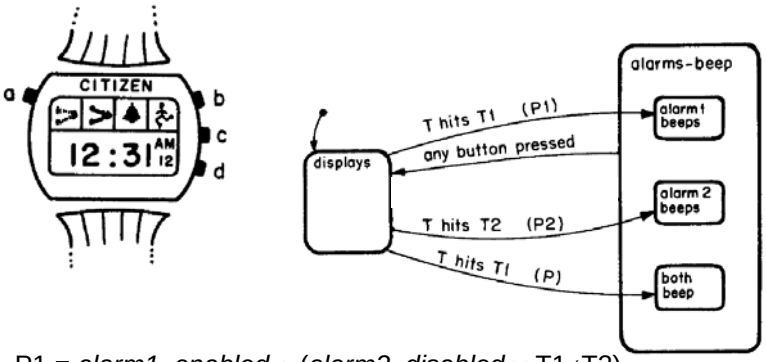
Here:

- Include typed variables (e.g. integers, reals, strings, records) to represent data
- Both „graphical states“ and variables contribute to the state of the statechart.
- Notation:
  - „graphical states“ = states
  - „graphical states“ + variables = **status**

A 10-Bit counter, counting on event *a* and issuing *overflow* after 1024 occurrences:



Example: Alarm Watch



P1 = *alarm1_enabled* ∧ (*alarm2_disabled* ∨ T1≠T2)
P2 = *alarm2_enabled* ∧ (*alarm1_disabled* ∨ T1≠T2)
P2 = *alarm1_enabled* ∧ *alarm2_enabled* ∧ T1=T2

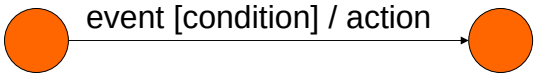
Example from *Harel: "StateCharts: A visual formalism for complex systems". Science of Computer Programming, 1987.*

Events and variables

Events:
- Exist only until the next evaluation of the model
- Can be either internally or externally generated

Variables:
- Values of variables keep their value until **they are reassigned.**

General form of edge labels



Meaning:
- Transition may be taken, if event occurred in last step and condition is true
- If transition is taken, then reaction is carried out.

Conditions:
- Refer to values of variables

Actions:
- Can either be assignments for variables or creation of events

Example:
- a & [x = 1023] / overflow; x:=0

Events, conditions, actions

- Possible events (incomplete list):
  - Atomic events
    - Basic events: A, B, BUTTON_PRESSED
    - Entering, exiting a state: en(S), ex(S)
    - Values of conditions: tr(cond), fs(cond)
    - Change of conditions: [cond], e.g. [X>5]
    - Change of values: ch(X)
    - Access to variables: rd(X), wr(X)
    - Timeout events (see later)
  - Compound events: logical connectives and, or, not

Events, conditions, actions

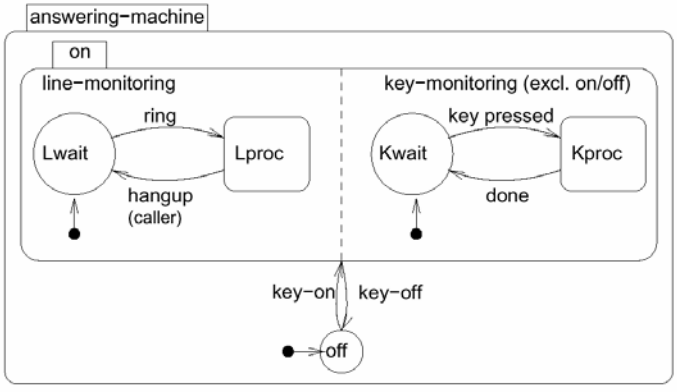
- Possible conditions (incomplete list):
  - Atomic conditions
    - Constants: true, false
    - Condition variables (i.e. variables of type boolean)
    - Relations between values: X > 1023, X $\leq$ Y
    - Residing in a state: in(S)
  - Compound events: logical connectives and, or, not

Events, conditions, actions

- Possible actions (incomplete list):
  - Atomic actions
    - Emitting events: E (E is event variable)
    - Assignments: X := expression
    - Scheduled actions: sc!(A, N) (means perform action after N time units)
  - Compound actions
    - List of actions: A1; A2; A3
    - Conditional action: if cond then A1 else A2

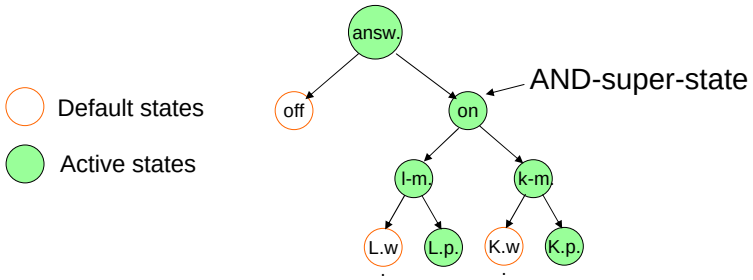
Concurrency

- Convenient ways of describing concurrency are required.
- **AND-super-states**: FSM is in **all** (immediate) sub-states of a AND-super-state; Example:



Concurrency

- Example for active states:

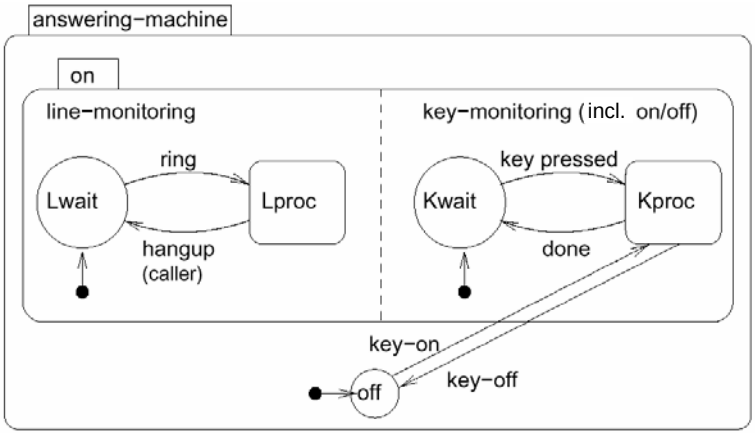


- Classical automata have to compute product automata to express concurrency

⇒ structural information is lost

⇒ increase in size

Entering and leaving AND-super-states



- Line-monitoring and key-monitoring are entered and left, when key-on and key-off events occur.

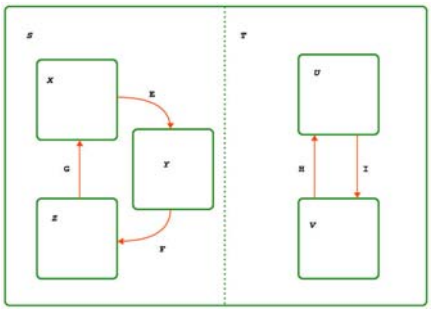
Types of states

In StateCharts, states are either

- **basic states,** or
- **AND-super-states,** or
- **OR-super-states.**

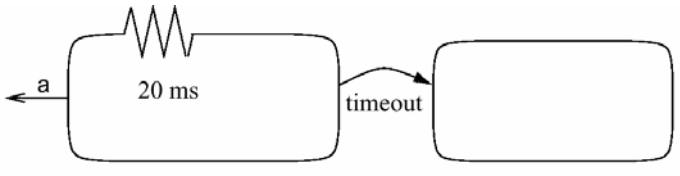
Concurrency

How to represent this statechart with OR-states only?



Timers

- Since time needs to be modeled in embedded systems, timers need to be modeled.
- In StateCharts, special edges can be used for timeouts.



If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.

Using timers in answering machine