

Communicating finite state machines

Peter Marwedel
TU Dortmund
Informatik 12



© Springer, 2010

2012年 10 月 24 日

Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, ...	
Discrete event (DE) model	VHDL, Verilog, SystemC, ...	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA	

StateCharts

Classical automata not useful for complex systems
(complex graphs cannot be understood by humans).

☞ Introduction of hierarchy ☞ StateCharts [Harel, 1987]

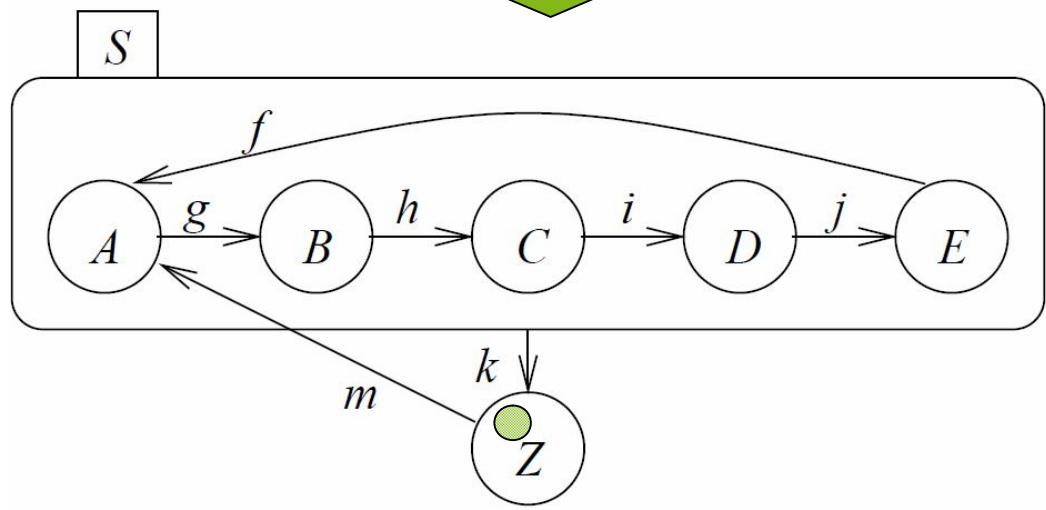
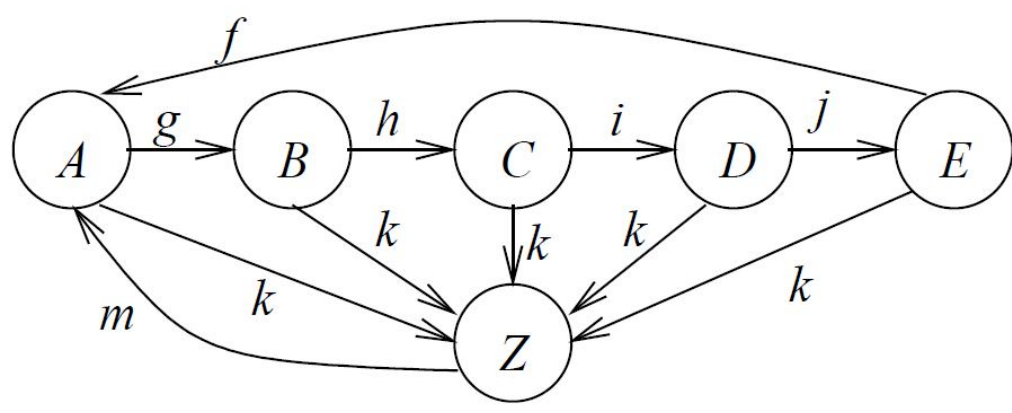
StateChart = *the only unused combination of
„flow“ or „state“ with „diagram“ or „chart“*

Used here as a (prominent) example of a
model of computation based on shared
memory communication.

☞ appropriate only for local
(non-distributed) systems



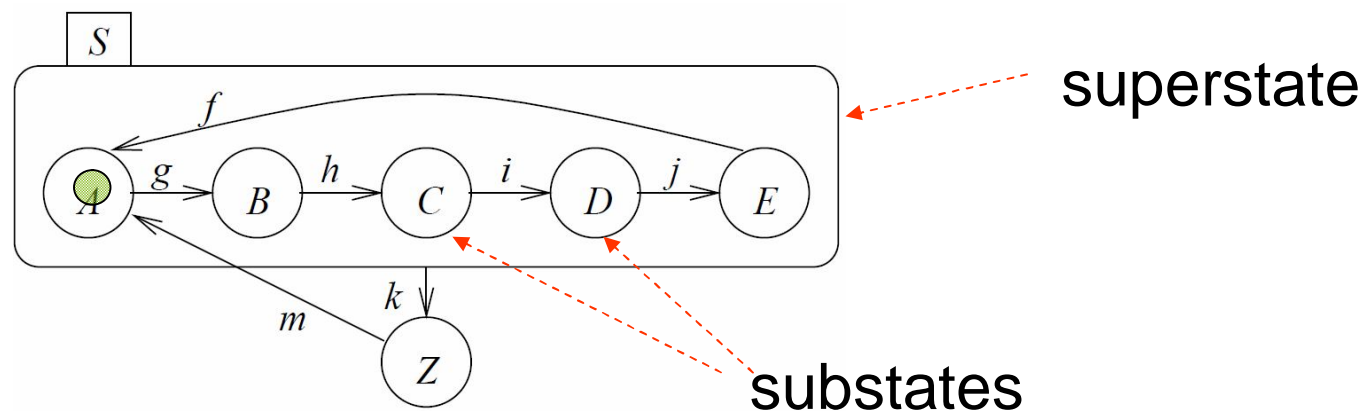
Introducing hierarchy



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

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**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.



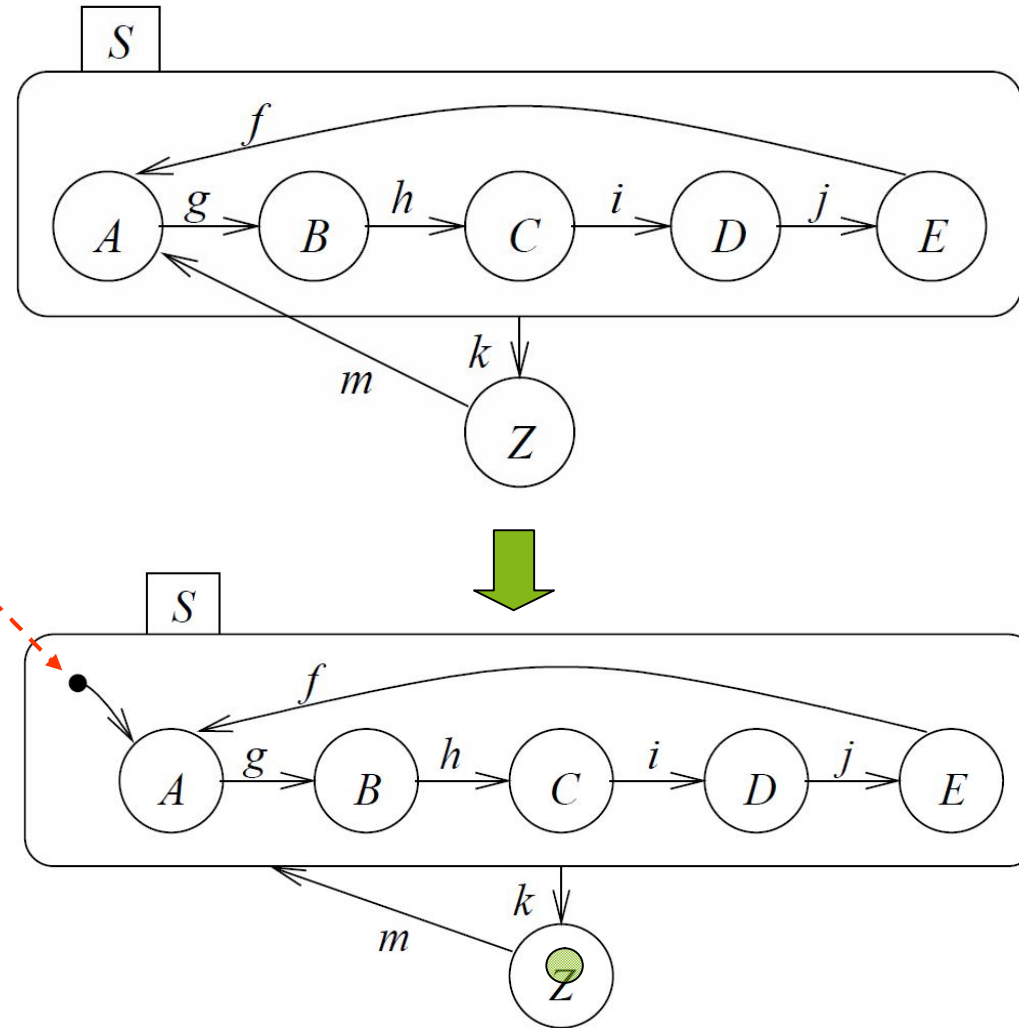
Default state mechanism

Try to hide internal structure from outside world!

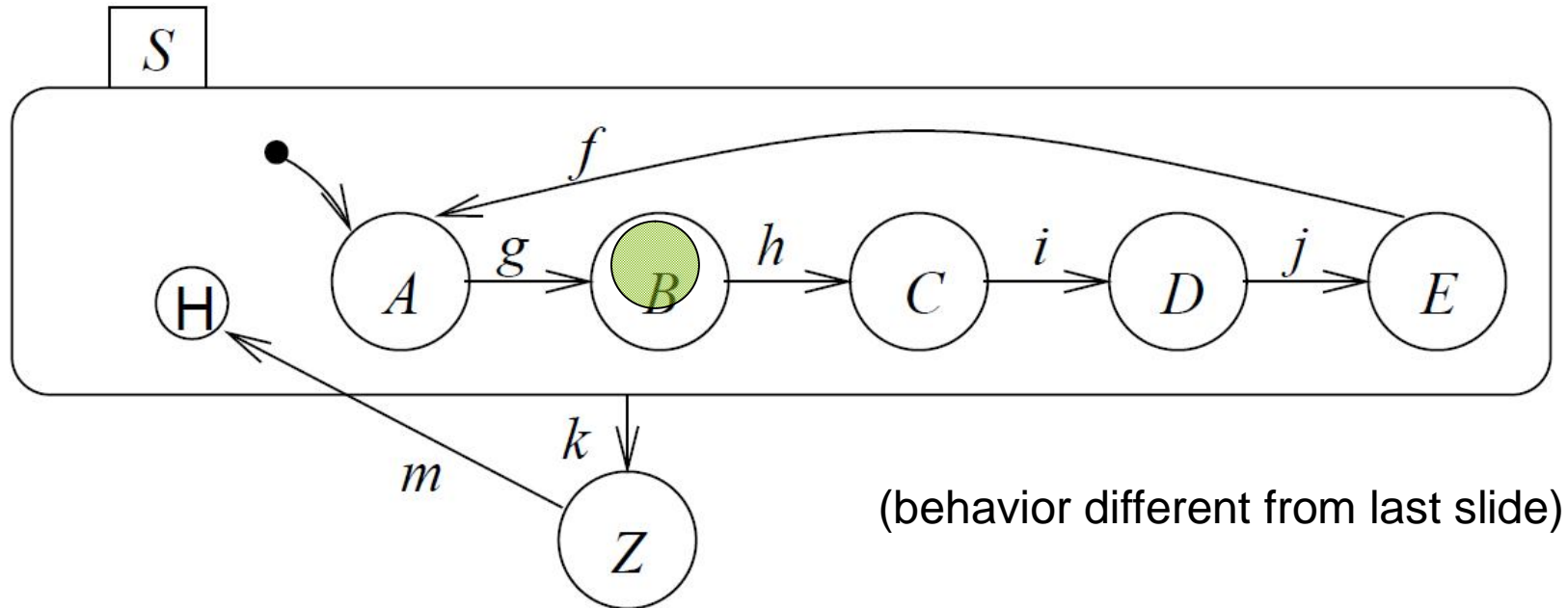
☞ Default state

Filled circle indicates sub-state entered whenever super-state is entered.

Not a state by itself!



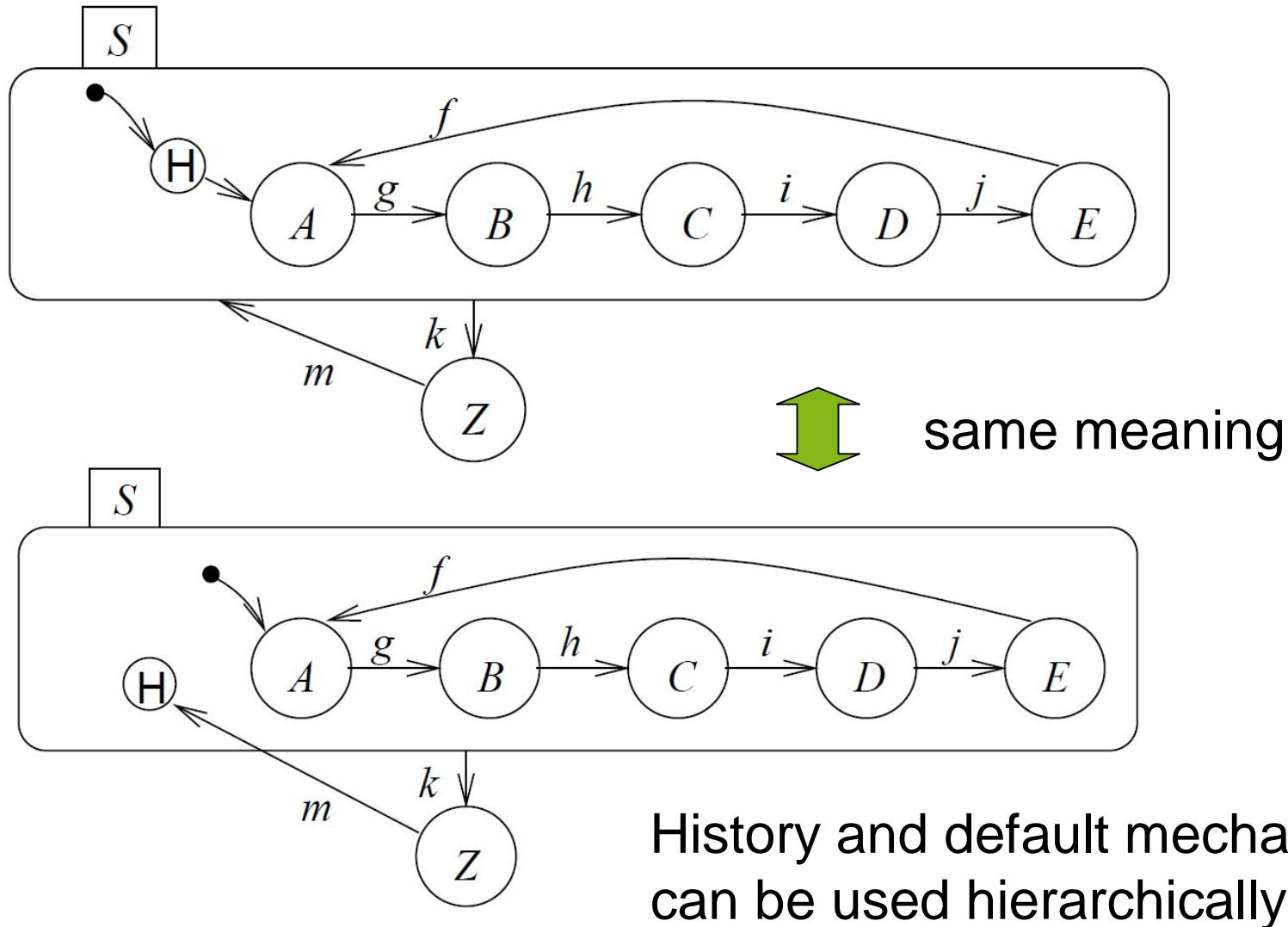
History mechanism



For input 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 first time, the default mechanism applies.

Combining history and default state mechanism

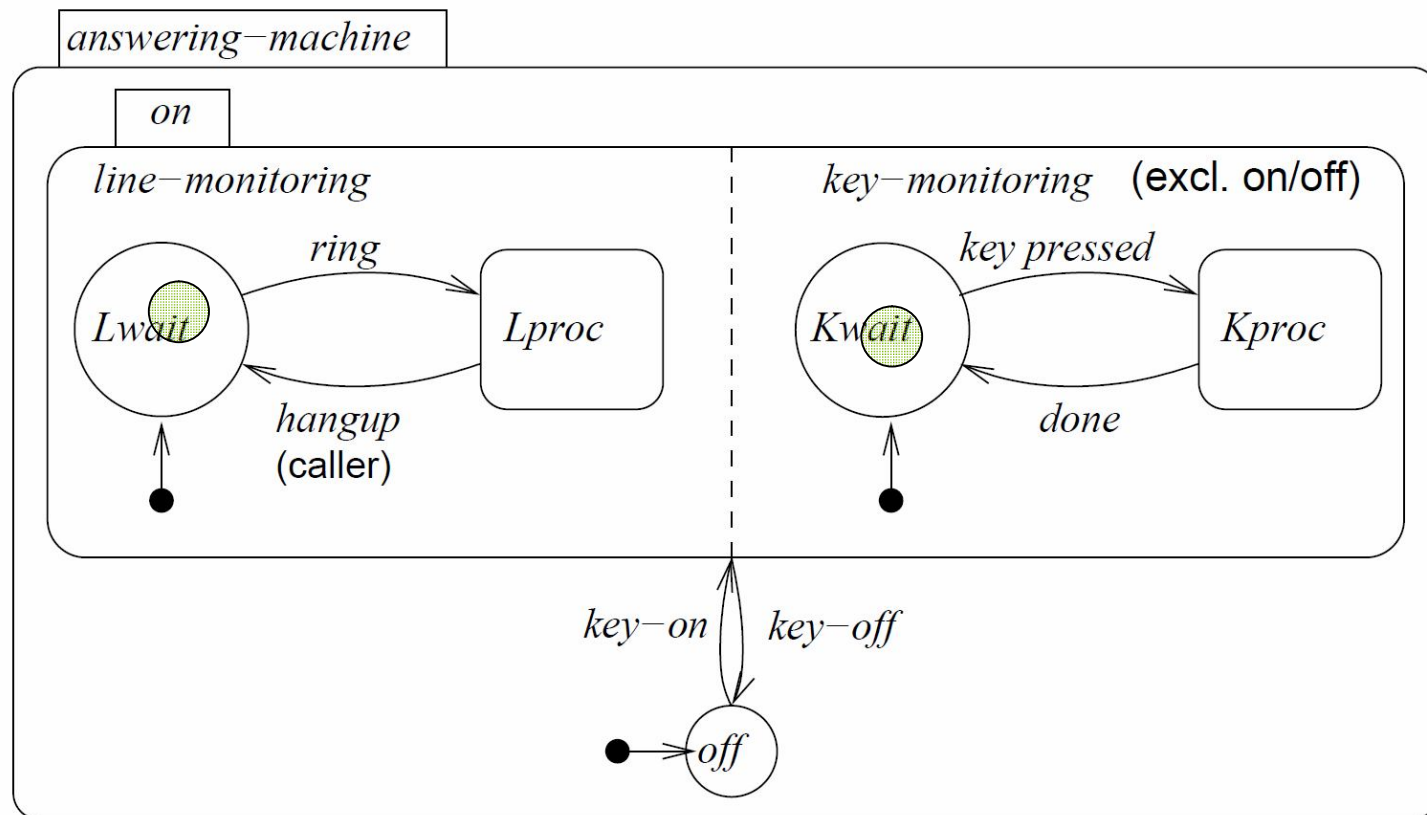


Concurrency

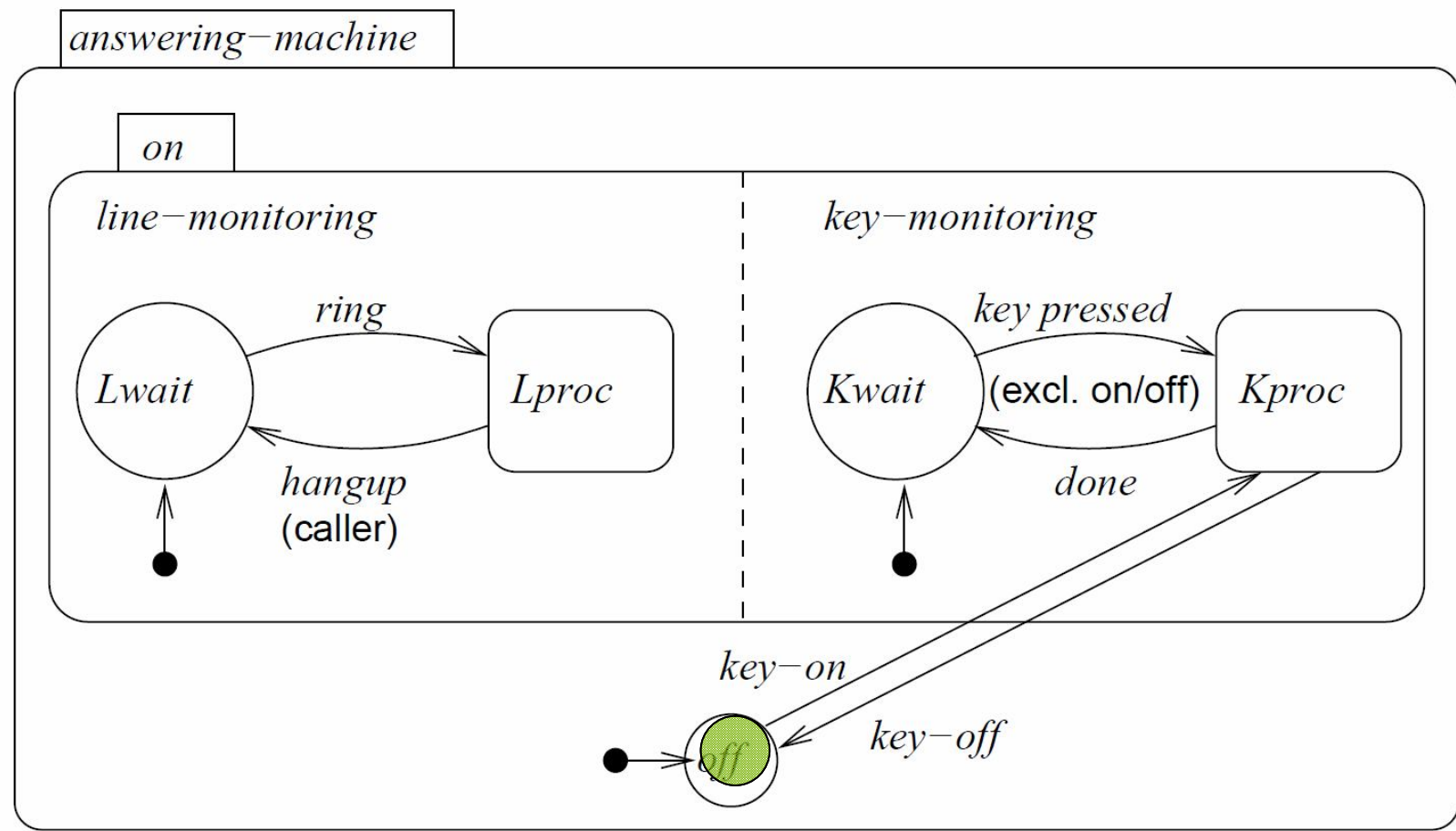
Convenient ways of describing concurrency req.

AND-super-states: FSM is in **all** (immediate) sub-states of a super-state;

Example:



Entering and leaving AND-super-states



Line-monitoring and key-monitoring are entered and left, when service switch is operated.

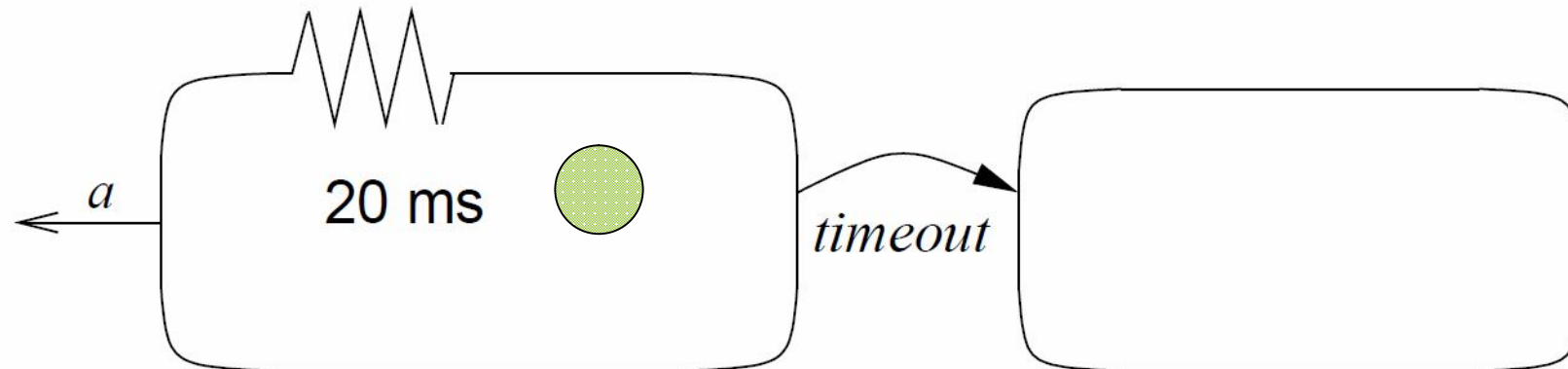
Types of states

In StateCharts, states are either

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

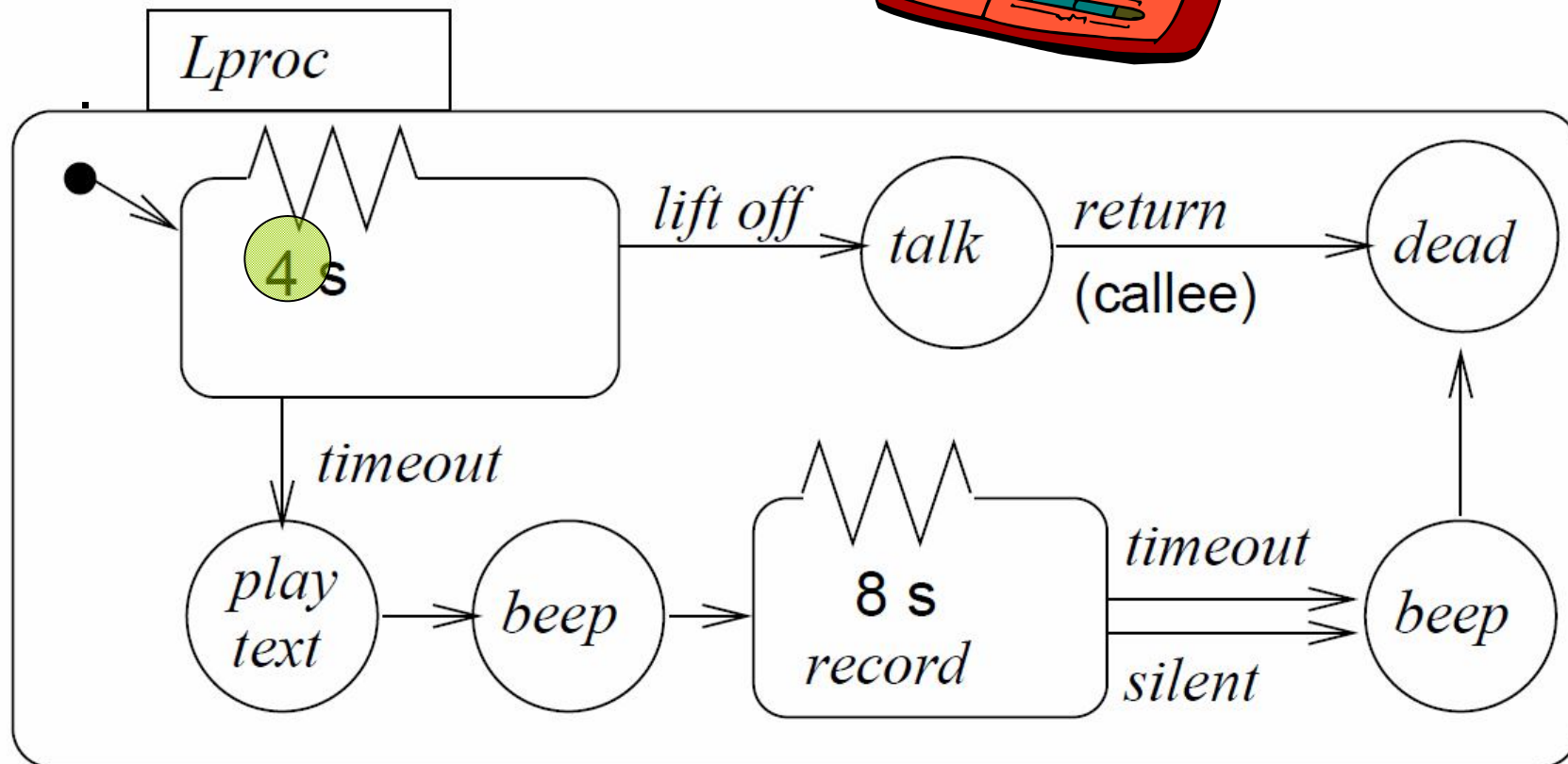
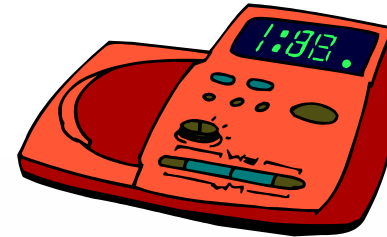
Timers

Since time needs to be modeled in embedded & cyber-physical 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 an answering machine



General form of edge labels



Events:

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

Conditions:

- Refer to values of variables that keep their value until **they are reassigned**

Reactions:

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

Example:

- *service-off* [not in *Lproc*] / *service:=0*

The StateCharts simulation phases (StateMate Semantics)

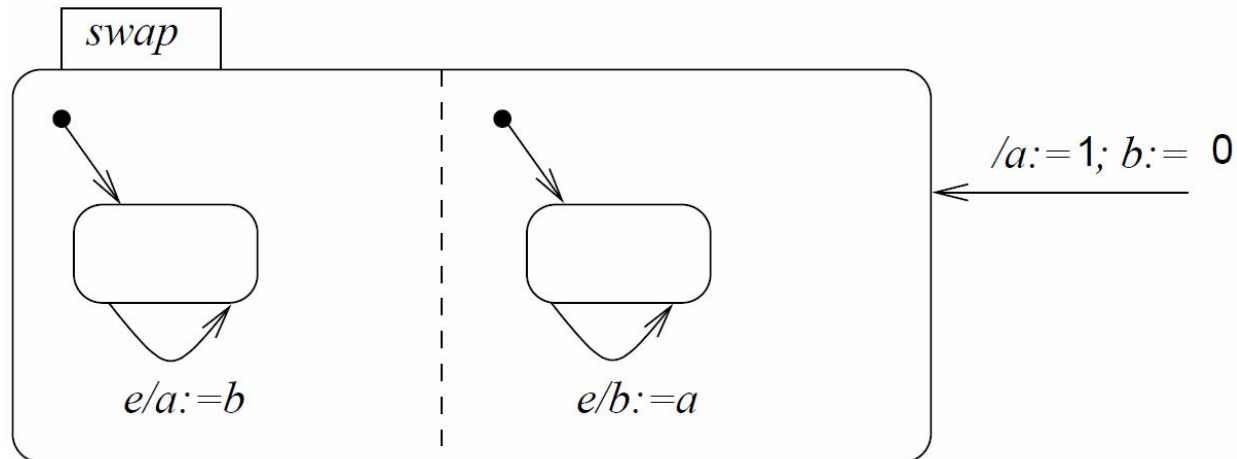
How are edge labels evaluated?

Three phases:

1. Effect of external changes on events and conditions is evaluated,
2. The set of transitions to be made in the current step and right hand sides of assignments are computed,
3. Transitions become effective, variables obtain new values.

Separation into phases 2 and 3 enables a resulting unique (“determinate”) behavior.

Example



In phase 2, variables a and b are assigned to temporary variables:

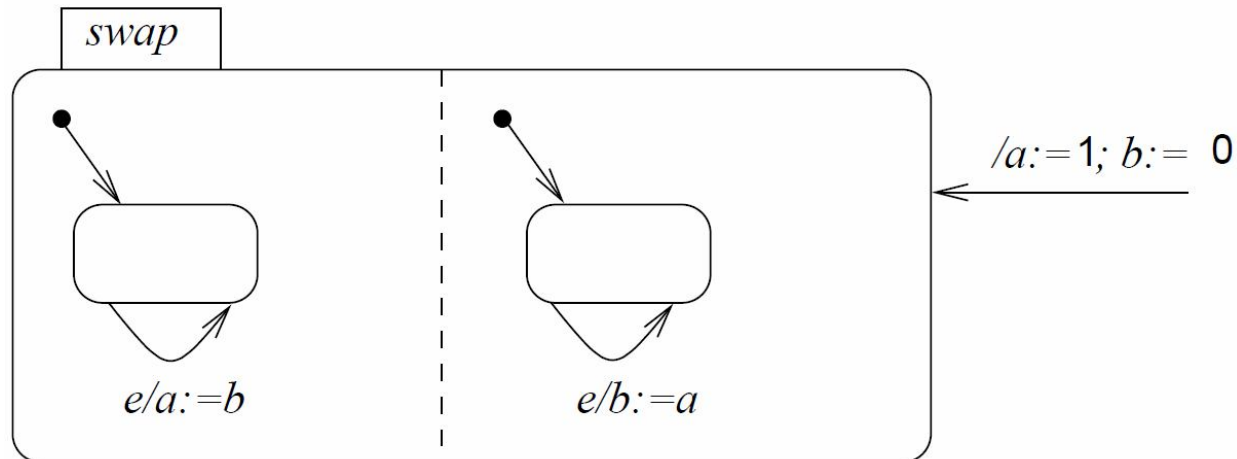
$$a' := b, b' := a;$$

In phase 3, these are assigned to a and b .

$$a := a', b := b';$$

As a result, variables a and b are swapped.

Example (2)



In a single phase environment, executing the left state first would assign the old value of b ($=0$) to a and b :

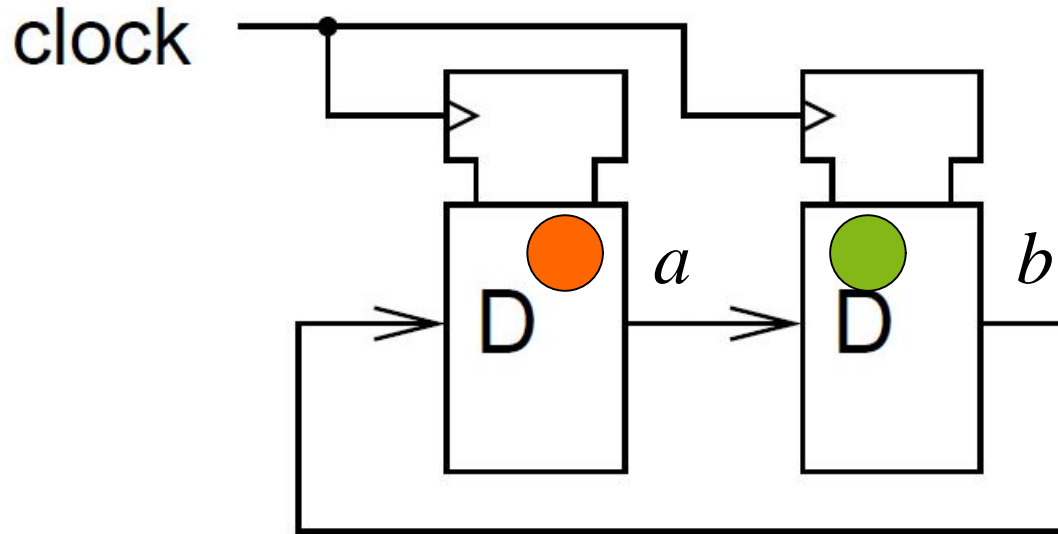
$a := 0, b := 0;$

Executing the right state first would assign the old value of a ($=1$) to a and b .

$b := 1, a := 1;$

The result would depend on the execution order.

Reflects model of clocked hardware

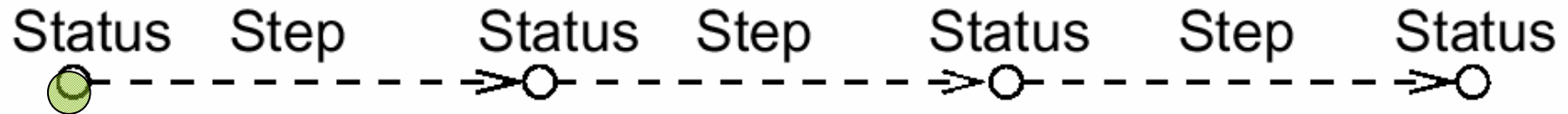


In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.

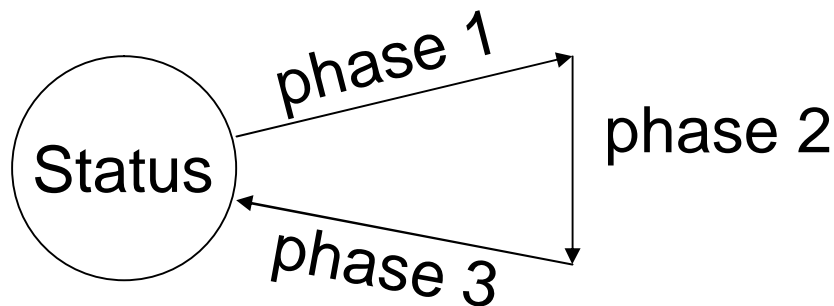
Steps

Execution of a StateMate model consists of a sequence of (status, step) pairs



Status= values of all variables + set of events + current time

Step = execution of the three phases (**StateMate** semantics)



Other implementations of StateCharts do not have these 3 phases (and hence could lead to different results)!

Other semantics

Several other specification languages for hierarchical state machines (UML, dave, ...) do not include the three simulation phases.

These correspond more to a SW point of view with no synchronous clocks.

Some systems allow turning the multi-phased simulation on and off.



Broadcast mechanism



Values of variables are visible to all parts of the StateChart model

New values become effective in phase 3 of the current step and are obtained by all parts of the model in the following step.

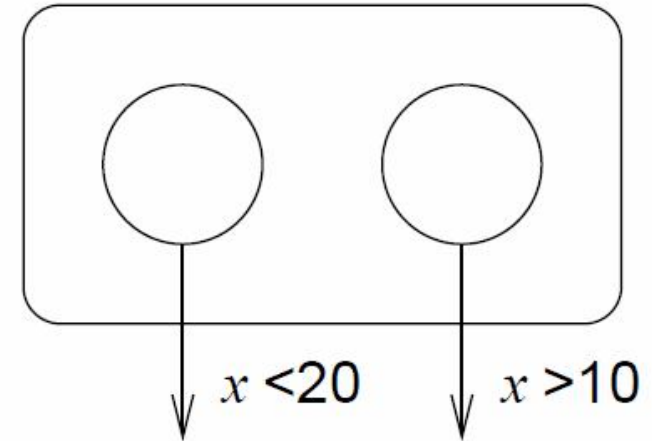
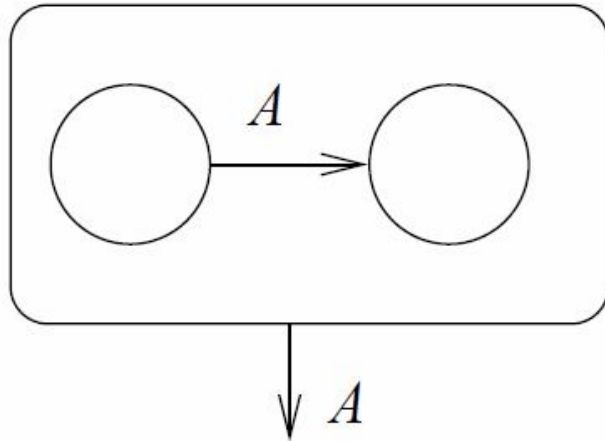
!

- ☞ StateCharts implicitly assumes a **broadcast** mechanism for variables
(→ implicit *shared memory communication*
–other implementations would be very inefficient –).
- ☞ StateCharts is appropriate for local control systems (☺), but not for distributed applications for which updating variables might take some time (☹).

Lifetime of events

Events live until the step following the one in which they are generated (“one shot-events”).

Conflicts



Techniques for resolving these conflicts wanted

Determinate vs. deterministic

- Kahn (1974) calls a system **determinate** if we will always obtain the same result for a fixed set (and timing) of inputs
- Others call this property **deterministic**
However, this term has several meanings:
 - Non-deterministic finite state machines
 - Non-deterministic operators
(e.g. + with non-deterministic result in low order bits)
 - Behavior not known before run-time
(unknown input results in non-determinism)
 - In the sense of determinate as used by Kahn

In order to avoid confusion, we use the term “determinate” in this course.

StateCharts determinate or not?

Must all simulators return the same result for a given input?

- Separation into 2 phases a required condition
- Semantics \neq StateMate semantics may be non-determinate

Potential other sources of non-determinate behavior:

- Choice between conflicting transitions resolved arbitrarily:
Tools typically issue a warning if such a situation could exist

→ Determinate behavior for StateMate semantics if transition conflicts are resolved and no other sources of undefined behavior exist

Evaluation of StateCharts (1)

Pros (👍):

- Hierarchy allows arbitrary nesting of AND- and OR-super states.
- (StateMate-) Semantics defined in a follow-up paper to original paper.
- Large number of commercial simulation tools available (StateMate, StateFlow, BetterState, ...)
- Available “back-ends“ translate StateCharts into SW or HW languages, thus enabling software or hardware implementations.

Evaluation of StateCharts (2)

Cons (👎):

- Not useful for distributed applications,
- no program constructs,
- no description of non-functional behavior,
- no object-orientation,
- no description of structural hierarchy,
- generated programs may be inefficient.

Extensions:

- Module charts for description of structural hierarchy.

Synchronous vs. asynchronous languages (1)

Description of several processes in many languages non-determinate: The order in which executable threads are executed is not specified (may affect result).

Synchronous languages: based on automata models.

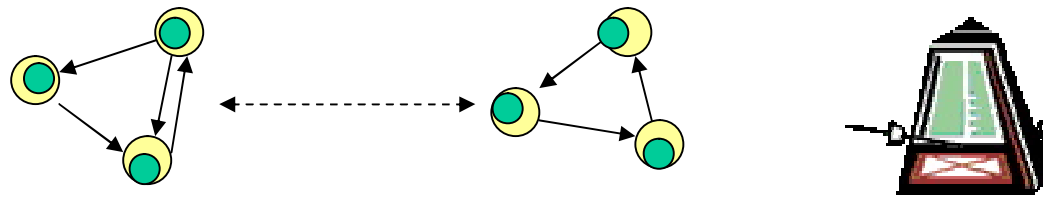
“Synchronous languages aim at providing high level, modular constructs, to make the design of such an automaton easier [Nicolas Halbwachs].



© P. Marwedel, 2008

Synchronous languages describe concurrently operating automata. “.. *when automata are composed in parallel, a transition of the product is made of the "simultaneous" transitions of all of them*“.

Synchronous vs. asynchronous languages (2)



Synchronous languages implicitly assume the presence of a (global) clock.

Each clock tick, all inputs are considered, new outputs and states are calculated and then the transitions are made.

Abstraction of delays



Let

- $f(x)$: some function computed from input x ,
- $\Delta(f(x))$: the delay for this computation
- δ : some abstraction of the real delay

Consider compositionality: $f(x)=g(h(x))$

Then, the sum of the delays of g and h would be a safe upper bound on the delay of f .

Two solutions:

1. $\delta = 0$, always  synchrony
2. $\delta = ?$ (hopefully bounded)  asynchrony

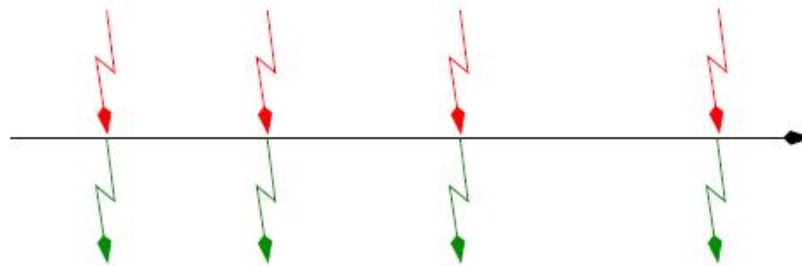
Asynchronous languages don't work [Halbwachs]

(Examples based on missing link to real time, e.g. what exactly does a **wait**(10 ns) in a programming language do?)

Compositionality

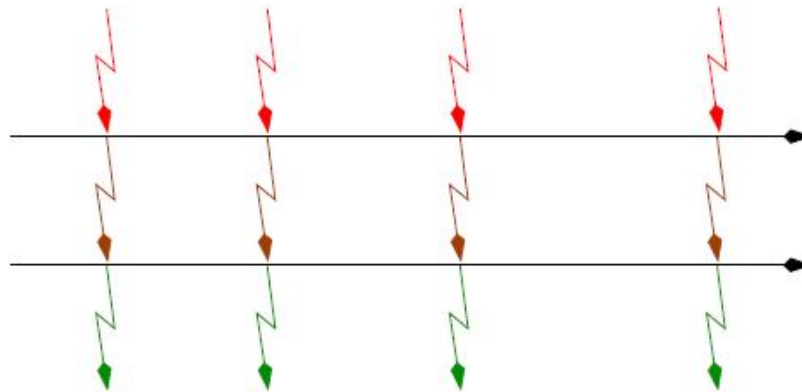
Abstract synchronous behavior

sequence of reactions to input events, to which all processes take part:



At the abstract level, a single FSM reacts ***immediately***

Composition of behaviors:



At the abstract level, reaction of connected other automata is ***immediate***


Based on slide 16 of N. Halbwachs: Synchronous Programming of Reactive Systems, ARTIST2 Summer School on Embedded Systems, Florianopolis, 2008

Concrete Behavior

The abstraction of synchronous languages is valid, as long as real delays are always shorter than the clock period.

Reference: slide 17 of N. Halbwachs: Synchronous Programming of Reactive Systems, *ARTIST2 Summer School on Embedded Systems*, Florianopolis, 2008

Synchronous languages

- Require a broadcast mechanism for all parts of the model.
- Idealistic view of concurrency.
- Have the advantage of guaranteeing determinate behavior.
-  *StateCharts* (using StateMate semantics) is an “almost” synchronous language [Halbwachs]. Immediate communication is the lacking feature which would make StateCharts a fully synchronous language.

Implementation and specification model

For synchronous languages, the **implementation** model is that of finite state machines (FSMs).

The specification may use different notational styles

- “Imperative”: Esterel (textual)
- SyncCharts: graphical version of Esterel
- “Data-flow”: Lustre (textual)
- SCADE (graphical) is a mix containing elements from multiple styles

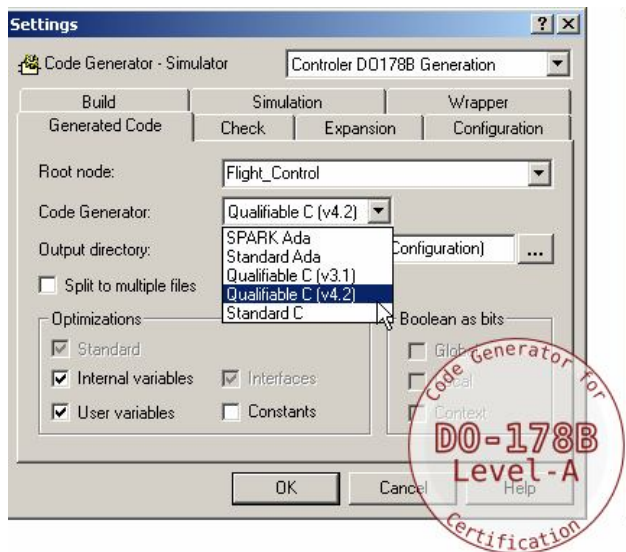
Nevertheless, specifications always include a close link to the generated FSMs (i.e., “imperative” does not have semantics close to von-Neumann languages)

Applications



SCADE Suite, including the SCADE KCG Qualified Code Generator, is used by AIRBUS and many of its main suppliers for the development of most of the A380 and A400M critical on board software, and for the A340-500/600 Secondary Flying Command System, aircraft in operational use since August 2002.

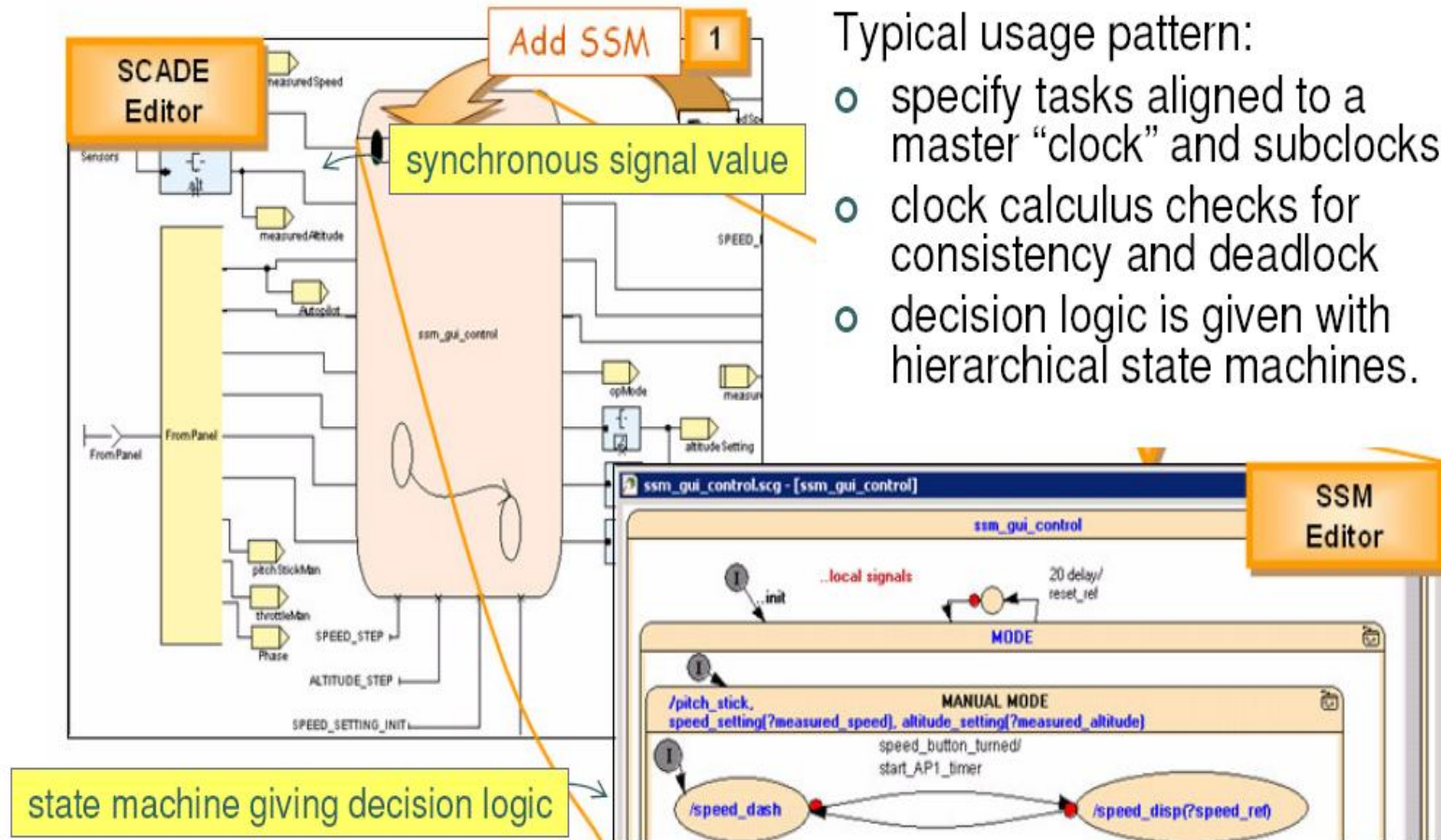
François Pilarski, Systems Engineering Framework - Senior Manager Engineering, Systems & Integration Tests; Airbus France.



Instance of “model-based design”

Source: <http://www.esterel-technologies.com/products/scade-suite/>


Threads are Not the Only Possibility: 4th example: Synchronous Languages



Summary

Communicating finite state machines

■ StateCharts

- Hierarchical states
 - OR-States
 - AND-States
- Timers
- Broadcasting of updates of variables  shared memory
- Determinate vs. deterministic

■ Synchronous languages

- Based on clocked finite state machine view
- Based on 0-delay (real delays must be small)