Robotics and Embedded Systems

Technische Universität München

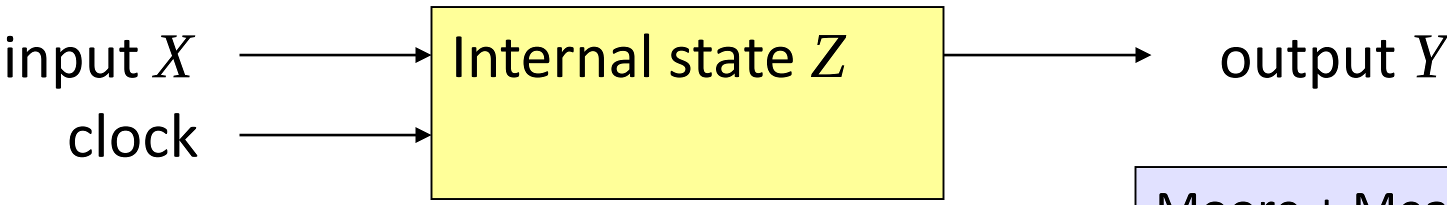
Specification and Modeling (2)
StateChart

Kai Huang

Outline

- Model of Computation (MoC)
- Data-Flow Models
- StateCharts

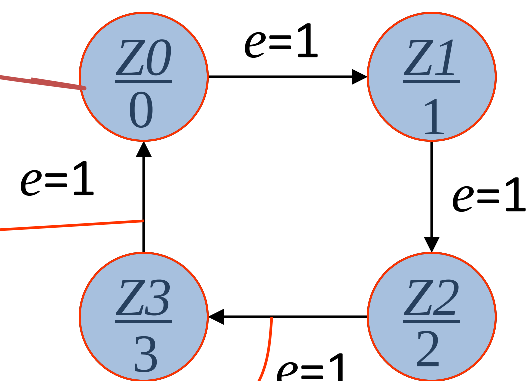
Classical Automata



Next state Z^+ computed by function δ
Output computed by function λ

Moore + Mealy automata=finite state machines (FSMs)

- Moore-automata:
  $Y = \lambda(Z);\quad Z^+ = \delta(X, Z)$
- Mealy-automata
  $Y = \lambda(X, Z);\quad Z^+ = \delta(X, Z)$

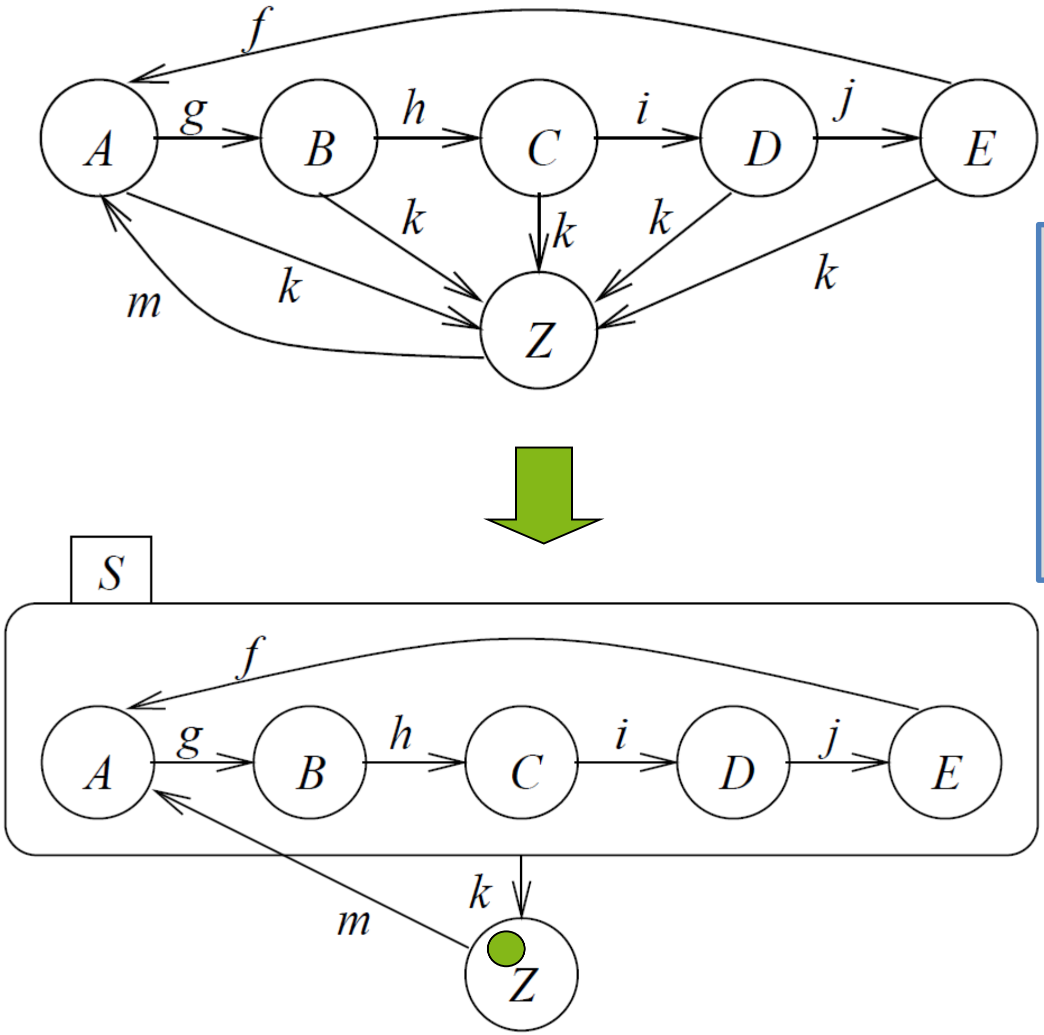


StateCharts

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

→ Introduction of hierarchy→StateCharts [Harel, 1987]

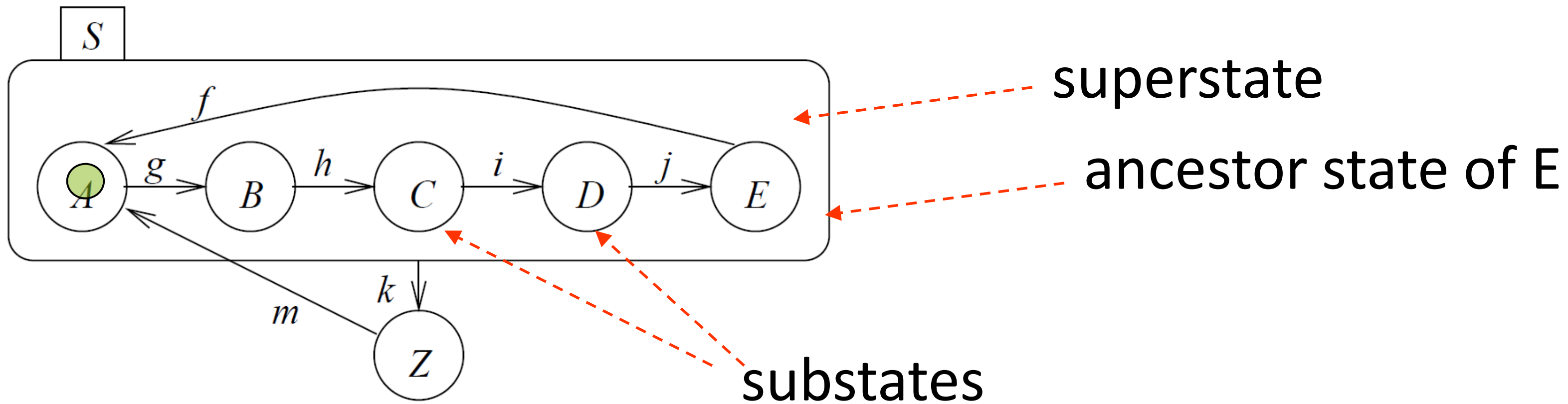
Introducing Hierarchy



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

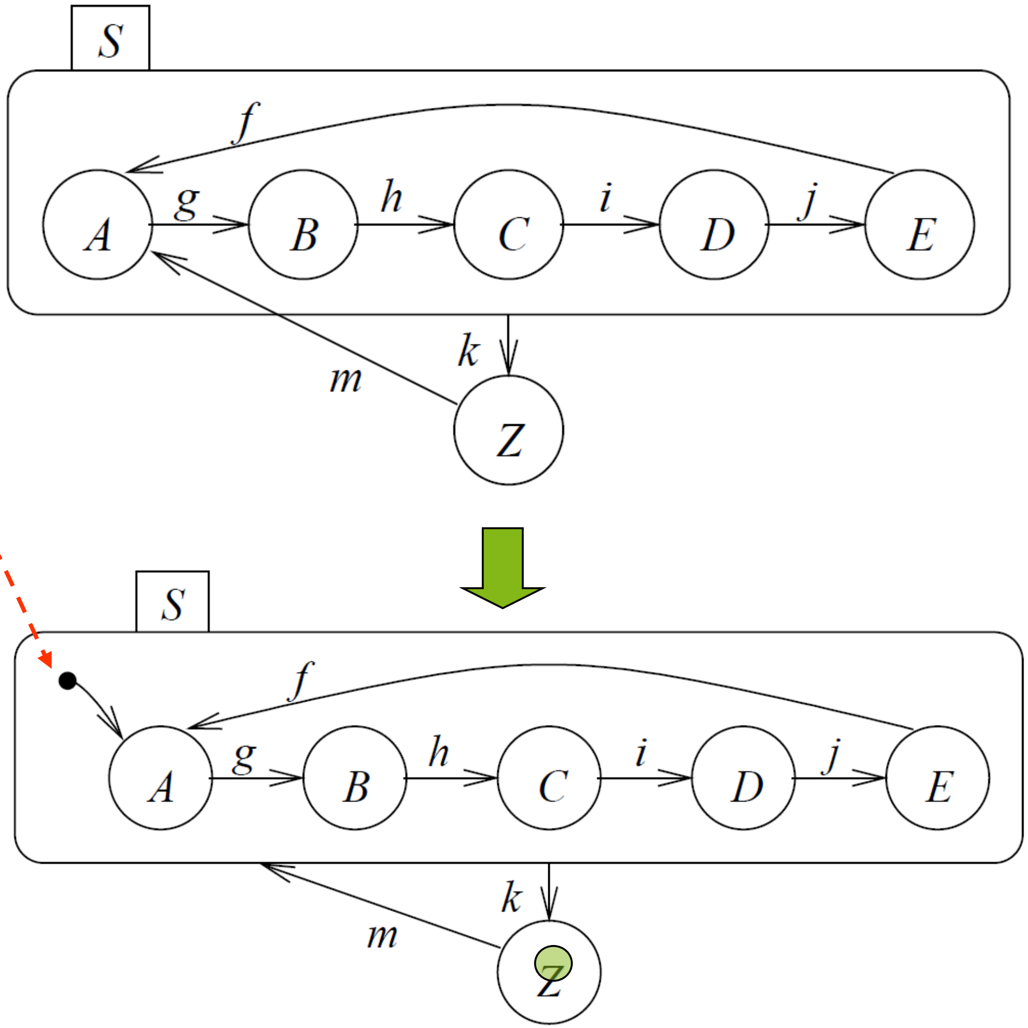
Definitions

- 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**.
- 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.

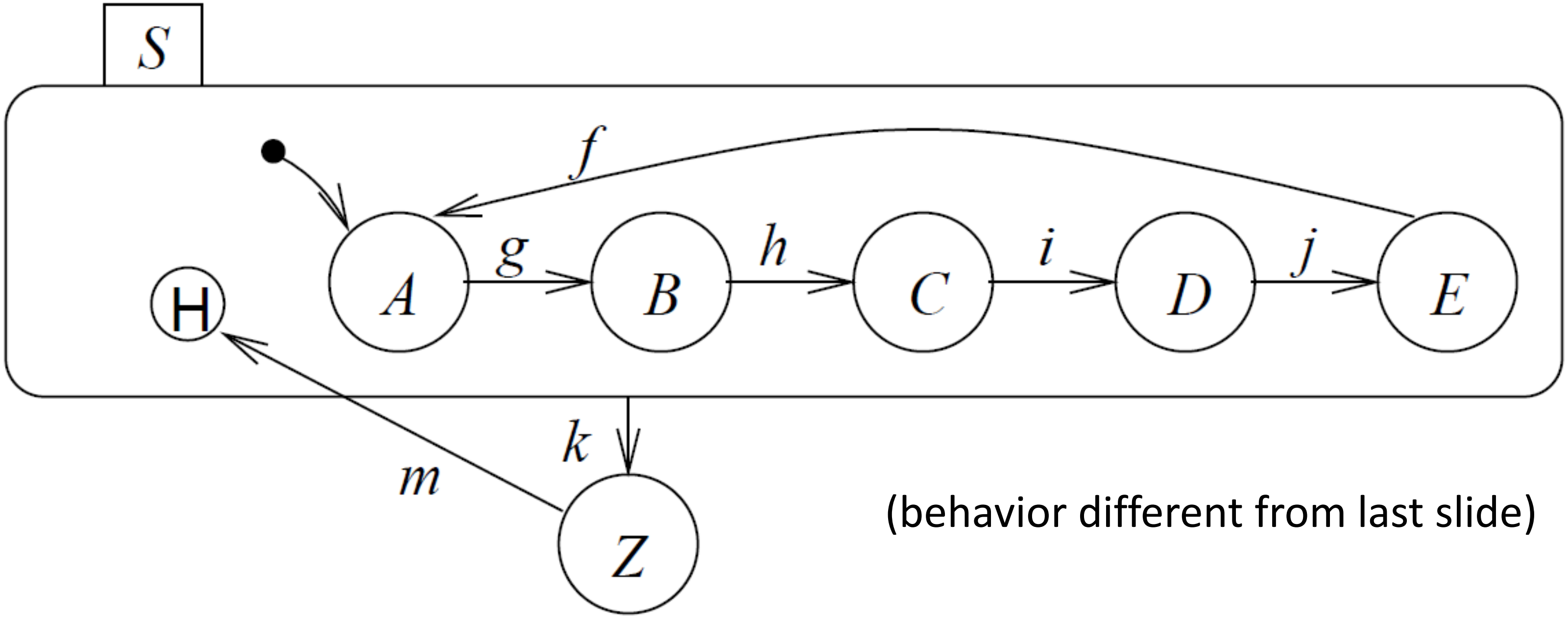


Default State Mechanism

- 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!



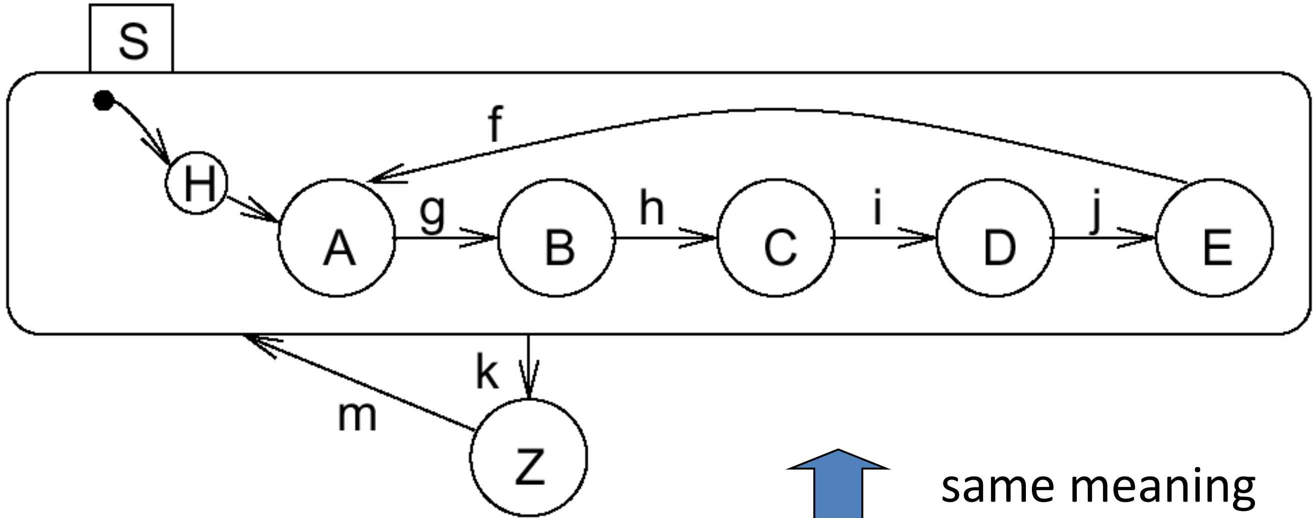
History Mechanism



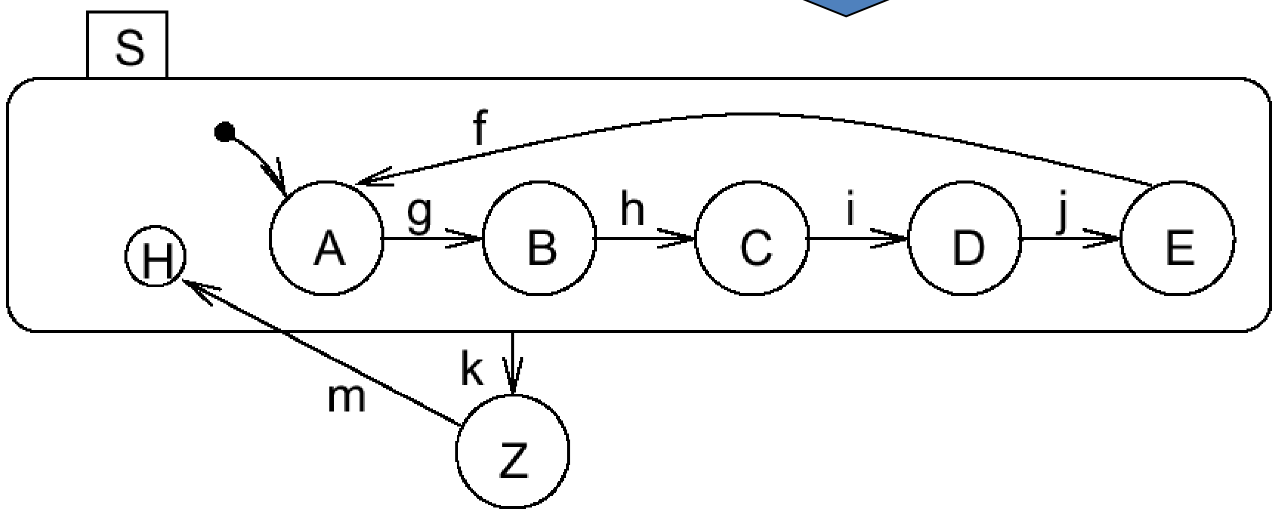
(behavior different from last slide)

- 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 very first time, the default mechanism applies.
- History and default mechanisms can be used hierarchically.

Combining History and Default State

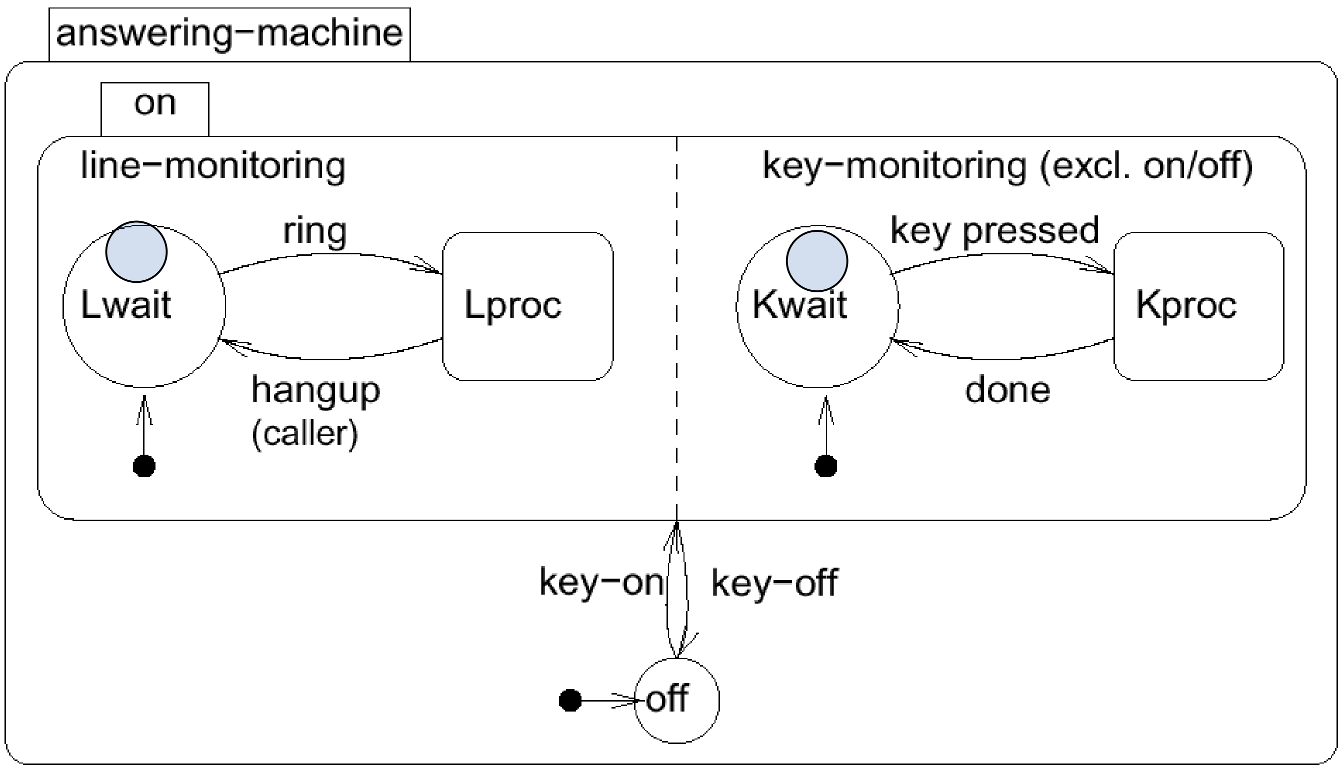


same meaning

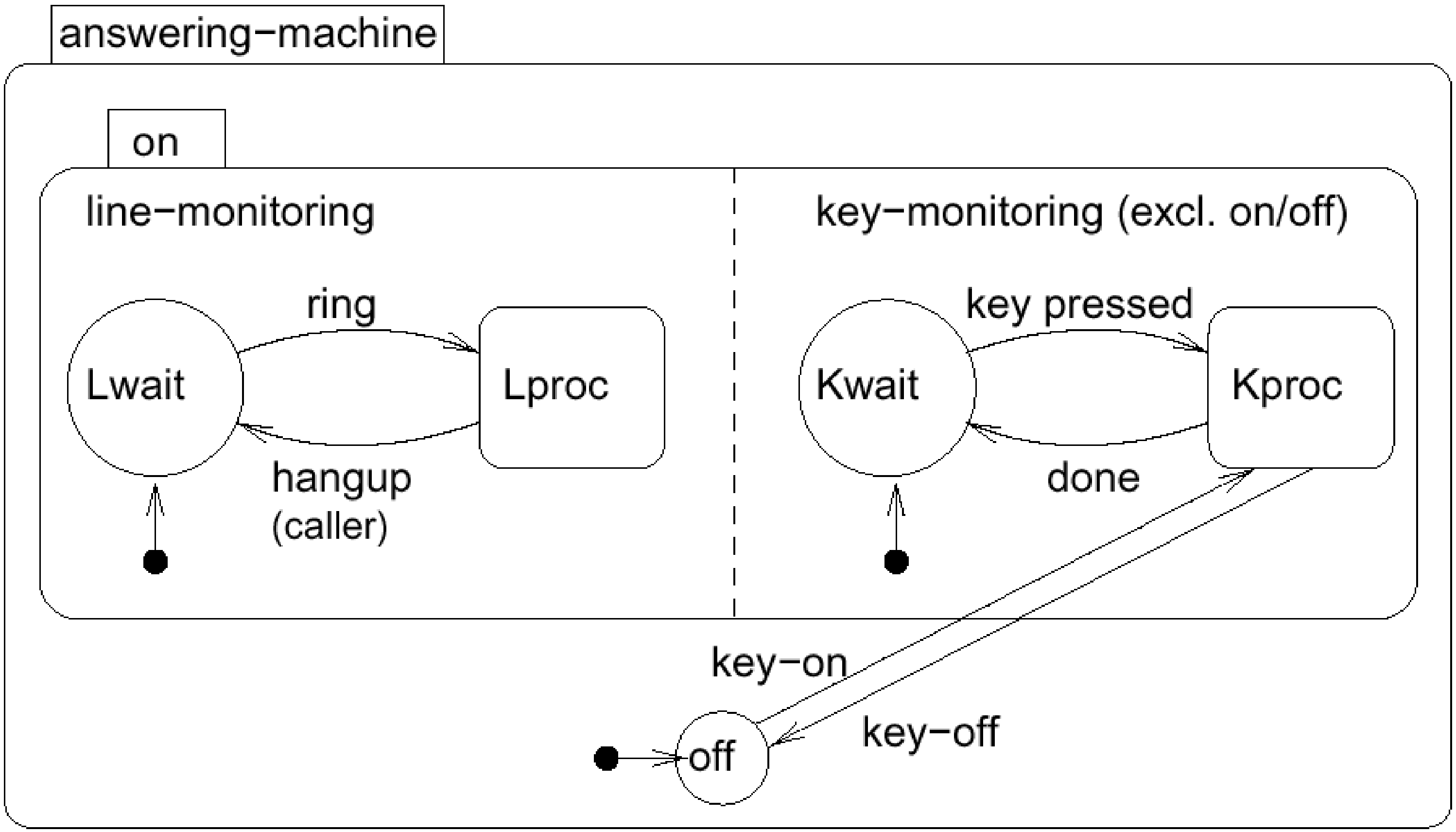


Concurrency

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

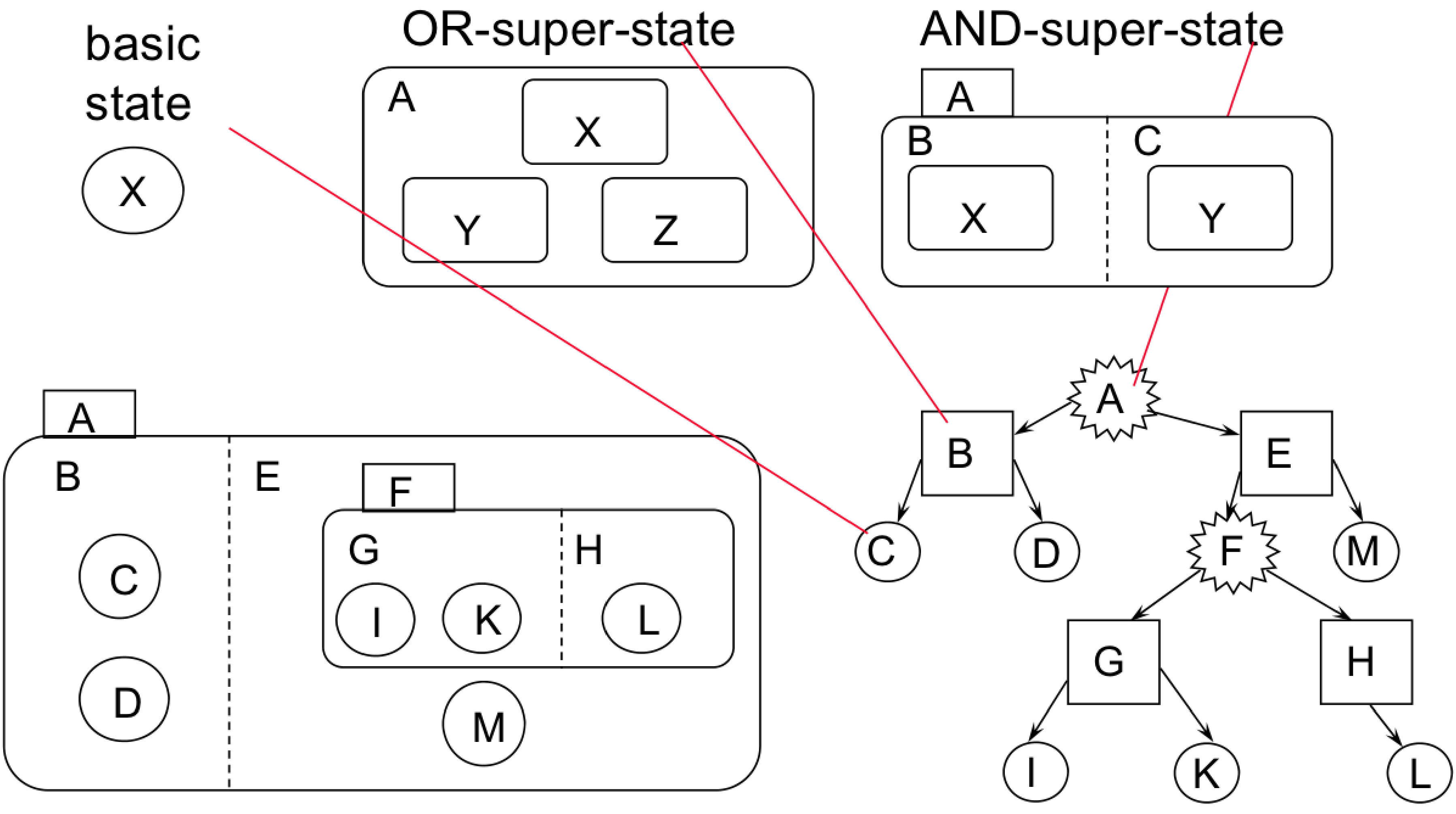


Entering and Leaving AND-super-states



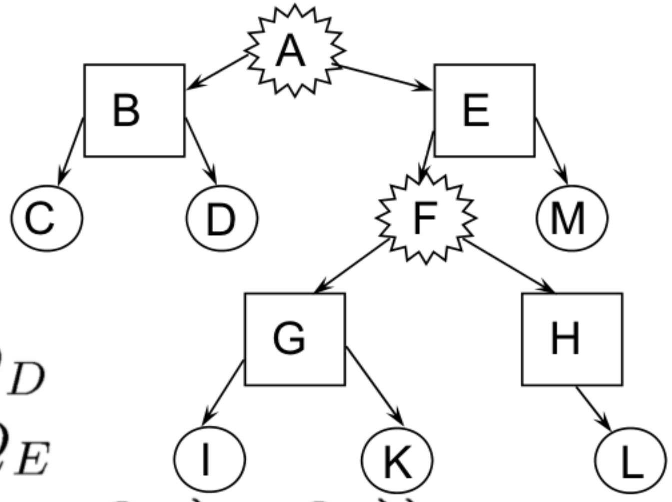
- Line-monitoring and key-monitoring are entered and left, when service switch is operated.

Tree Representation of State Sets



Computation of State Sets

- Computation of state sets by traversing the tree from leaves to root:
  - basic states: state set = state
  - OR-super-states: state set = union of children
  - AND-super-states: state set = Cartesian product of children



$$Q_H = Q_L,\ Q_G = Q_I \cup Q_K$$
$$Q_F = Q_G \times Q_H,\ Q_B = Q_C \cup Q_D$$
$$Q_E = Q_F \cup Q_M,\ Q_A = Q_B \times Q_E$$
$$Q_A = (Q_C \cup Q_D) \times (Q_M \cup ((Q_I \cup Q_K) \times Q_L))$$

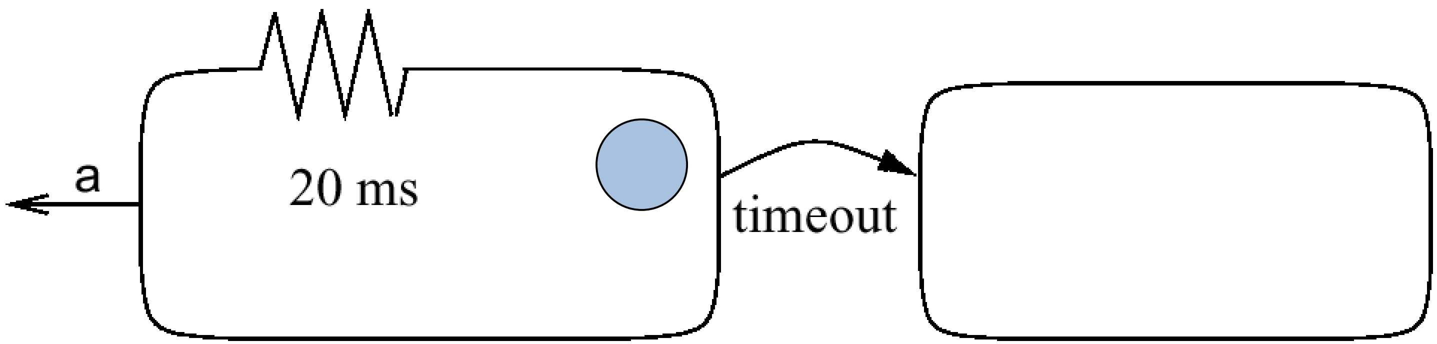
Types of States

- In StateCharts, states are either
  - Basic states, or
  - AND-super-states, or
  - OR-super-states.

- Stable state: there are no generated events and no enabled compound transition or static action

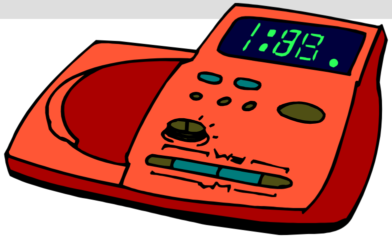
Timers

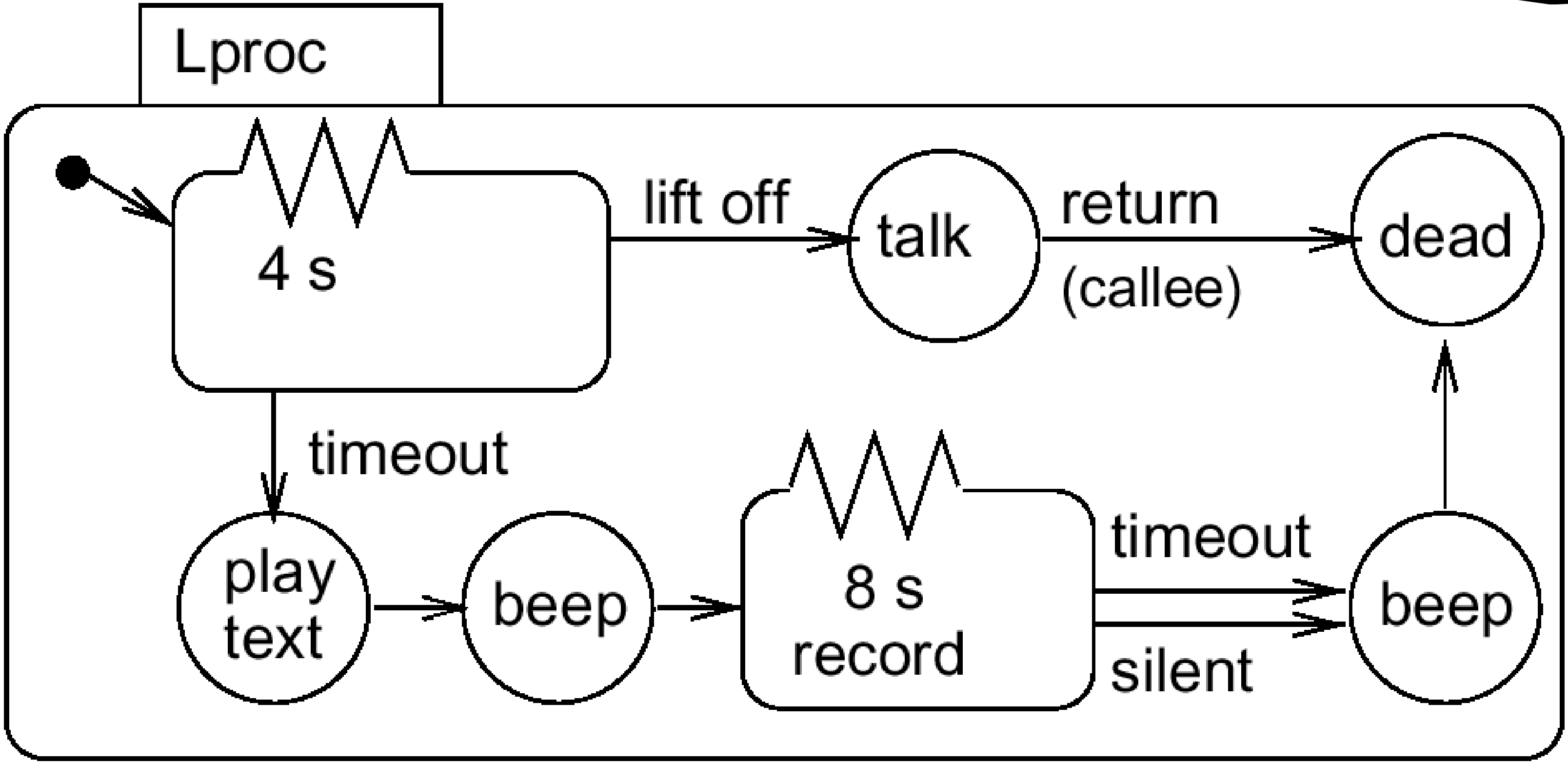
- Since time needs to be modeled in embedded systems, timers need to be modeled.
- In StateCharts, special edges can be used for timeouts.



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

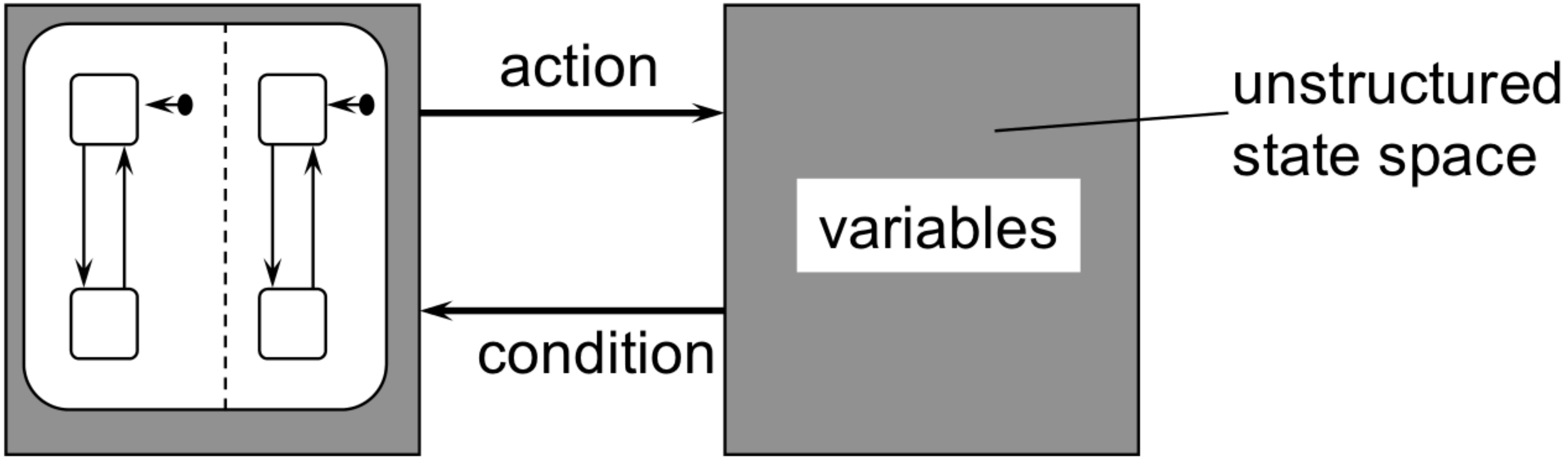
Using Timers in an Answering Machine



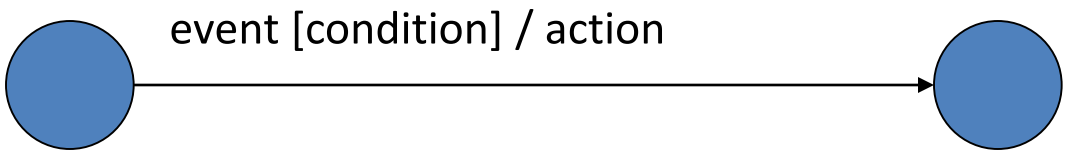


Representation of Computations

- Besides states, arbitrary many other variables can be defined. This way, not all states of the system are modeled explicitly.
- These variables can be changed as a result of a state transition ("action"). State transitions can be dependent on these variables ("condition").



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

- Example:
  - service-off [a <= 7] / service:=0

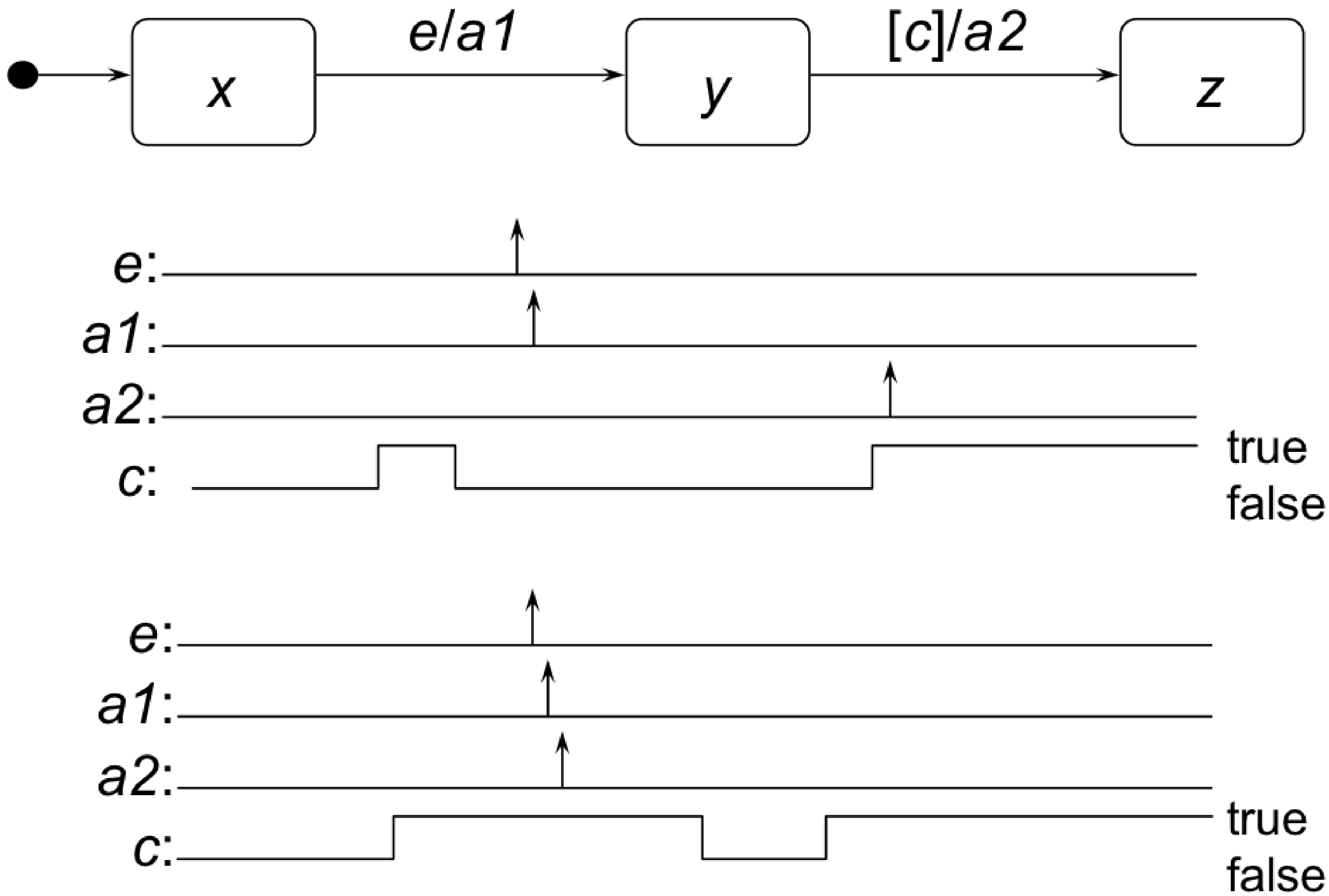
Events and Actions

- "event" can be composed of several events:
  - (e1 and e2) : event that corresponds to the simultaneous occurrence of e1 and e2.
  - (e1 or e2) : event that corresponds to the occurrence of either e1 or e2, or both.
  - (not e) : event that corresponds to the absence of event e.

- "action" can also be composed:
  - (a1; a2) : actions a1 and a2 are executed in parallel.

- All events, states and actions are globally visible.

Example



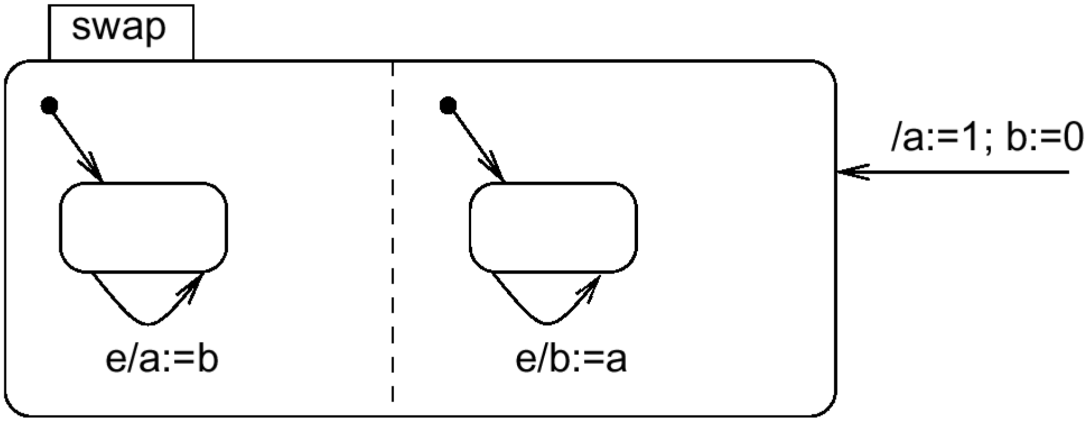
The StateCharts Simulation Phases

- How are edge labels evaluated?

- Three phases:
  1. Evaluate effect of external changes on events and conditions
  2. Compute set of transitions to be made in the current step and right hand sides of assignments
  3. Activate transitions, assign new values to variables

Separation into phases 2 and 3 guarantees deterministic and reproducible behavior

Example



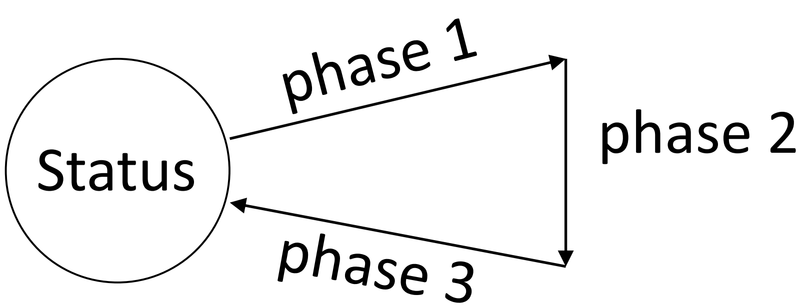
- In phase 2, variables a and b are assigned to temporary variables. In phase 3, these are assigned to a and b. As a result, variables a and b are swapped.
- In a single phase environment, executing the left state first would assign the old value of b (=0) to a and b. Executing the right state first would assign the old value of a (=1) to a and b. The execution would be non-deterministic.

Steps

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

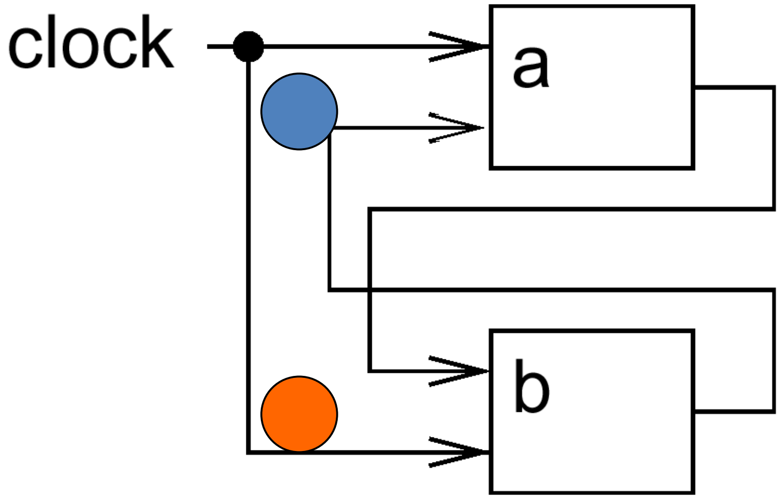
Status= values of all variables + set of events + current time

Step = execution of the three phases (StateMate semantics)



Other implementations of StateCharts do not have these 3 phases (and hence are nondeterministic)!

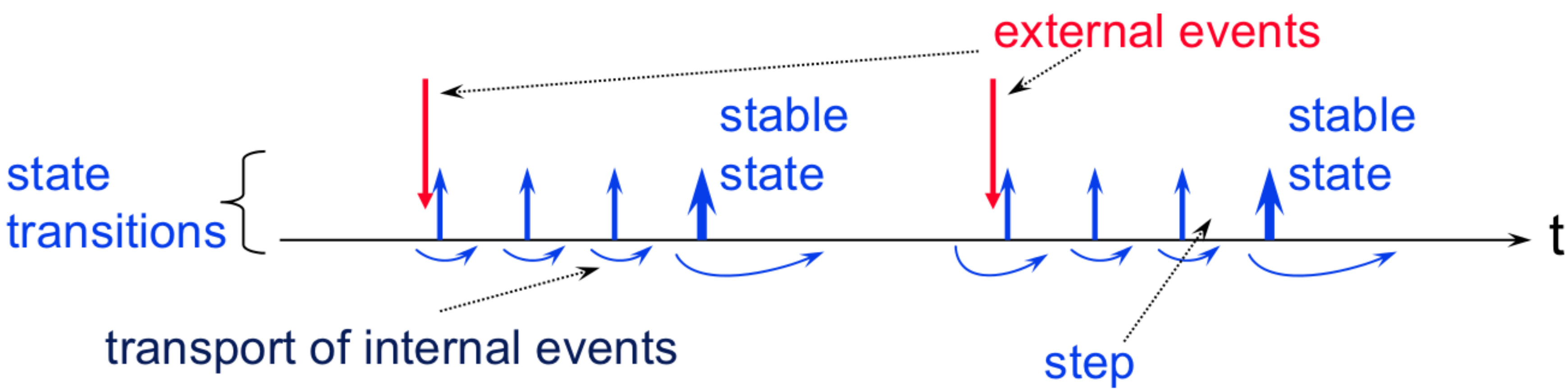
Reflects Model of Clocked Hardware



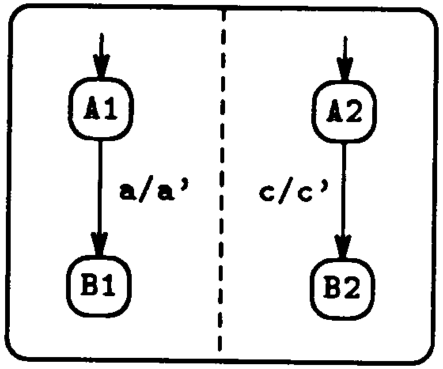
- In an actual clocked (synchronous) hardware system, both registers would be swapped as well.
- Same separation into phases found in other languages as well, especially those that are intended to model hardware.

More on Semantics of StateCharts

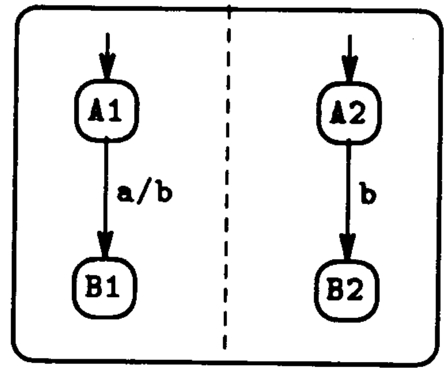
- Unfortunately, there are several time-semantics of StateCharts in use. This is another possibility:
  - A step is executed in arbitrarily small time.
  - Internal (generated) events exist only within the next step.
  - Difference: External events can only be detected after a stable state has been reached.



Examples



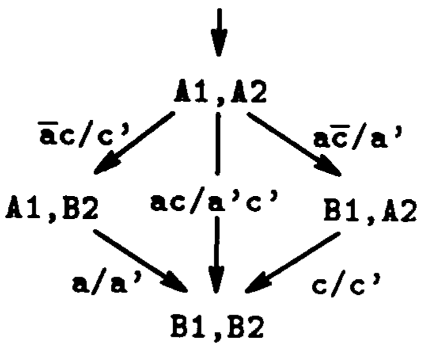
(a)



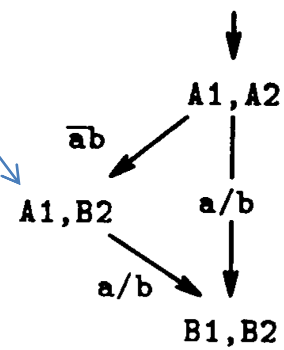
(b)

state diagram:

stable states

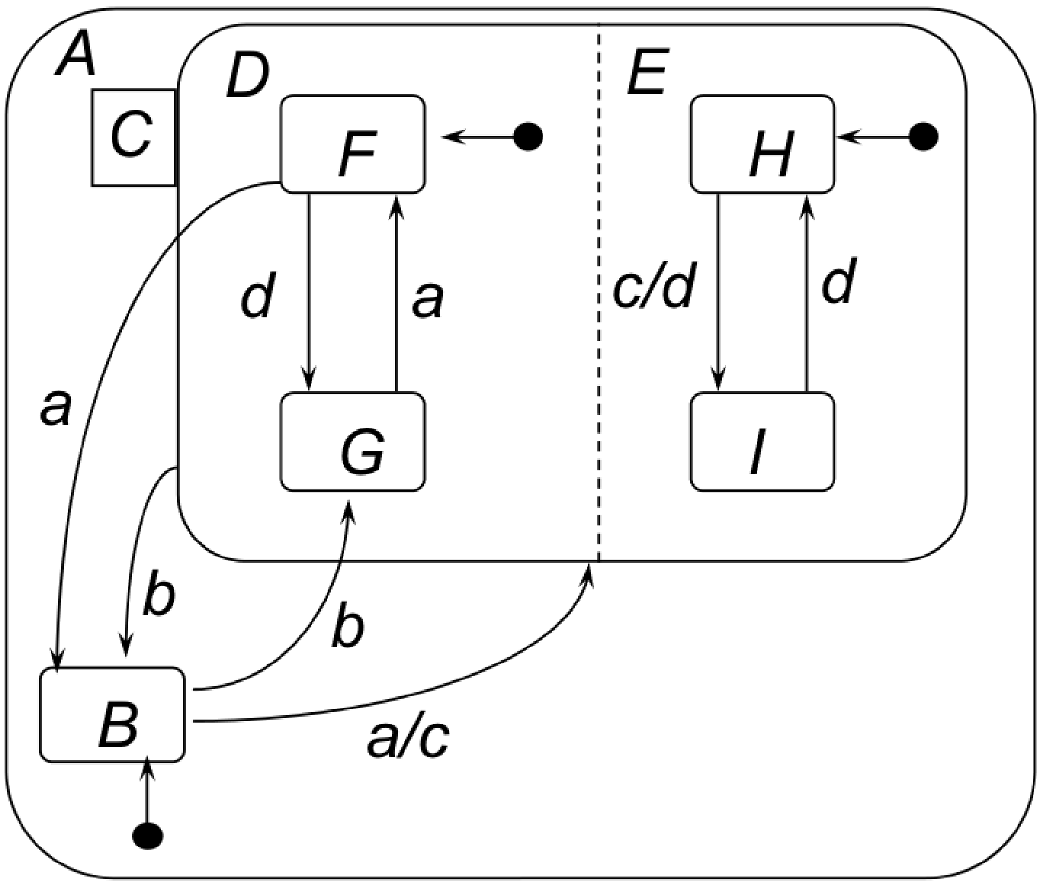


(a)

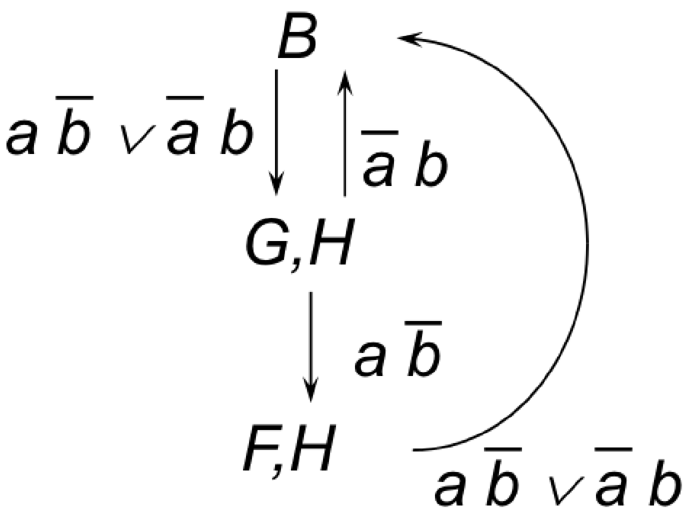


(b)

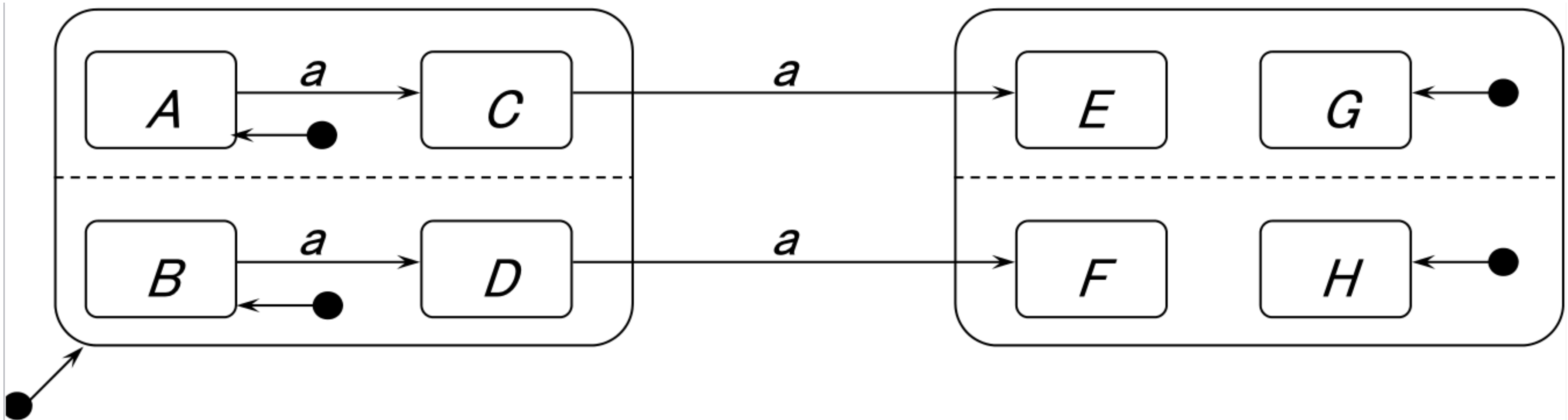
Example



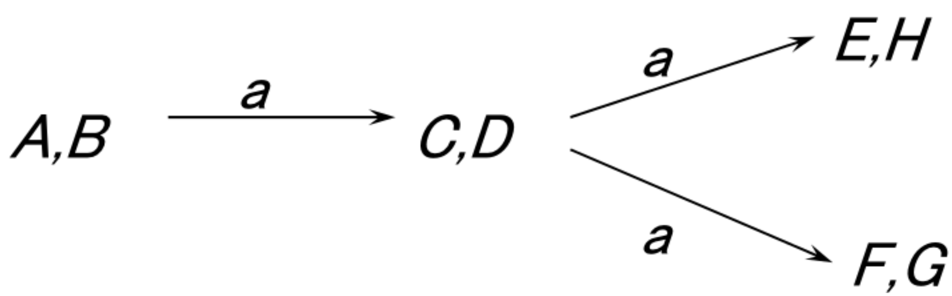
state diagram (only stable states are represented, only a and b are external):



Example



state diagram:



Nondeterministic !

Evaluation of StateCharts (1)

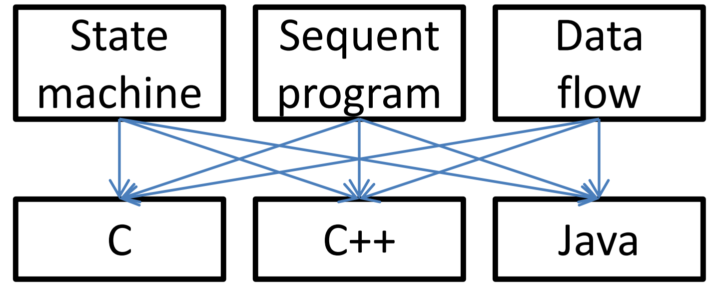
- Pros:
  - Hierarchy allows arbitrary nesting of AND- and OR-super states.
  - Semantics defined in a follow-up paper to original paper.
  - Large number of commercial simulation tools available (StateMate, StateFlow, BetterState, ...)
  - Available “back-ends“ translate StateCharts into C or VHDL,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

Evaluation of StateCharts (2)

- Cons:
  - Generated C programs frequently inefficient,
  - Not useful for distributed applications,
  - No description of non-functional behavior,
  - No object-orientation,
  - No description of structural hierarchy.

Specification Languages

Do not confuse Specification Languages with Models of Computation!!!



- Models of Computation describe system behavior
  - Conceptual notion, e.g., sequential execution, dataflow, FSM
- Specification Languages capture Models of Computation
  - Concrete syntax (textual or graphical) form, e.g., C, C++, Java
- Variety of languages can capture one model
  - E.g., C, C++, Java → sequential execution model
- One language can capture variety of models
  - E.g., C++ → sequential execution model, dataflow model, state machine model
- Certain languages better at capturing certain model of computation
  - E.g., VHDL captures best the Discrete Event (DE) model