December 19, 2007

Software Modeling and Verification
Lehrstuhl für Informatik 2
RWTH Aachen University
Prof. Dr. Ir. J.-P. Katoen
C. Kern

Foundations of the UML

Winter Term 07/08

– Lecture number 8 –

(Date (19th Dec 2007))

summarized by *Ratna Widyastuti(277010)* and *Teena Mary Mihan(284125)*

In the previous lecture, we have learned about MSCs. Now, in this lecture, we learn also about visual formalilsm, namely, statecharts. While MSCs are used to specify the requirement, statecharts visualizes the behaviour of discrete-event systems, the behavior of components in the system which run parallel, the hierarchy, the communication between the components. In other words:

Statecharts = automata + hierarchy + communication + time + concurrency

Statecharts are developed by David Harel, in 1987, a professor at Weizmann Institute, an institute in Israel, and he is also a co-founder of i-Logic Inc. which is a company that sells tools for statecharts. Statecharts are extensively used in embedded systems, automotive, and avionics industry, and has variant like stateflow which is used in MATLAB/Simulink. Simulink is popular tool for embedded system design.

1. What are Statecharts

According to David Harel, ”Statecharts constitute a visual formalism for:

- describing states and transitions in a modular way
- enabling clustering [of states]
- orthogonality
- and refinement, and
- encouraging “zoom” for moving easily back and forth between levels of abstraction”

If we break down the definition above one by one:

- Modular → several statecharts can be placed in 1 statechart
- Clustering → putting state in other state, so 1 state has substates, hierarchy/structure
- Orthogonality → different component/process which run parallel
- Refinement → a state may have substates
- Zoom → to make state more detail, zoom in can be used, vice versa zoom out to reach higher hierarchy of the state

Statecharts = Mealy machines + state hierarchy + broadcast communication +orthogonality

2. Mealy Machine

Mealy machine (or finite state transducer) is:
a finite state machine that produces output on a transition, based on the current input and the current state. In automata, output can be rejected or accepted, but in a Mealy machine, the acceptation/rejection of output is not of interest, its task is only generating output.

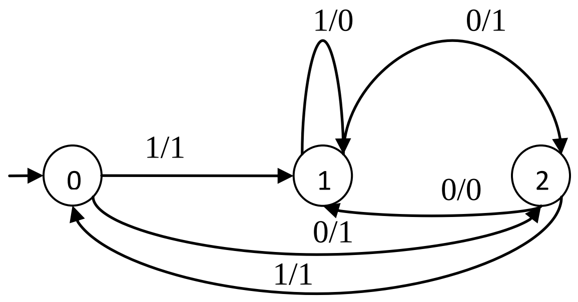
A mealy machine $A=(Q, q_o, \Sigma, \Gamma, \delta, \omega)$ where:

- Q is a finite set of states
- $q_o \in Q$ is the initial state
- Σ is the input alphabet
- Γ is the output alphabet (what possible output the machine can generate)
- δ: $Q \times \Sigma$ is the transition function such that $|\delta(q,a)| \leq 1$, $\forall q \epsilon Q, a \epsilon \Sigma$
- ω: $Q \times \Sigma \rightarrow \Gamma$ is output function
  the number of possible successors is deterministic

Characteristics of Mealy machine:

- No final states
- Transitions produce output, no acceptance of input word, but generating output from input is important
- No nondeterministic states

An example of Mealy machine:
Notation meaning:
input/output, ex: 0/1 means input=0 output=1
$\omega(2,0)=0$ means function which return output 0 in state 2 if the input is $0 \rightarrow \omega(\text{state},\text{input})=\text{output}$



In a Moore machine, output only depends on the current state, i.e. ω: $Q \rightarrow \Gamma$ is the output function.

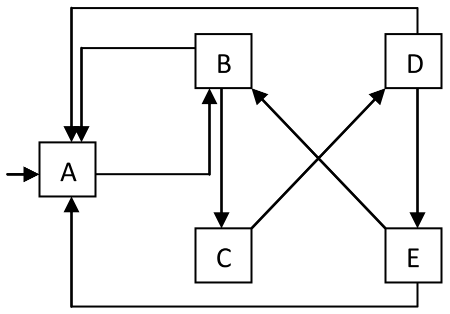
2.1. Limitations of Mealy Machines

- **No support for hierarchy**, since all states are arranged in “flat” fashion and no notion of substates
- **Scalability problem**, since for realistic systems, a complex transition structure and huge number of states are needed, and it yields unstructured state diagrams
- **No notion of concurrency**, which is needed for modeling independent component
- **No notion for communication**
- **No direct support for data**, in order to yield data, it has to encode data from input output

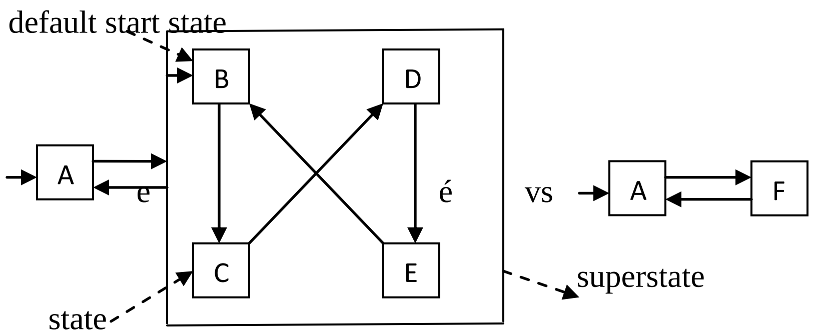
3. Characteristics of Statecharts

3.1. Scalability

<u>**Example of scalability problem**</u>:



Mealy machine



Statechart

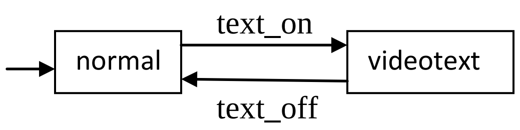
Statecharts yield a modular, hierarchical and more structured model compared to Mealy machine. In the example above, F is an ancestor of B, C, D, and E. B,C,D,E are descendants of F. Every state here can be interrupted, if we are in é, and e happens we can go back to A immediately. The second picture simplifies the states representation with hierarchy. In a Mealy machine, interruption in each state and then returning to A, can be obtained, but results in many transitions.

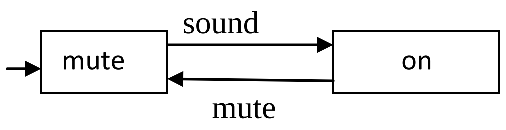
3.2. Orthogonality

Example of orthogonality:

2 different components:

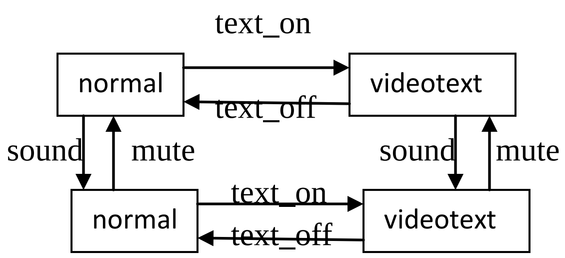
Image: Sound:





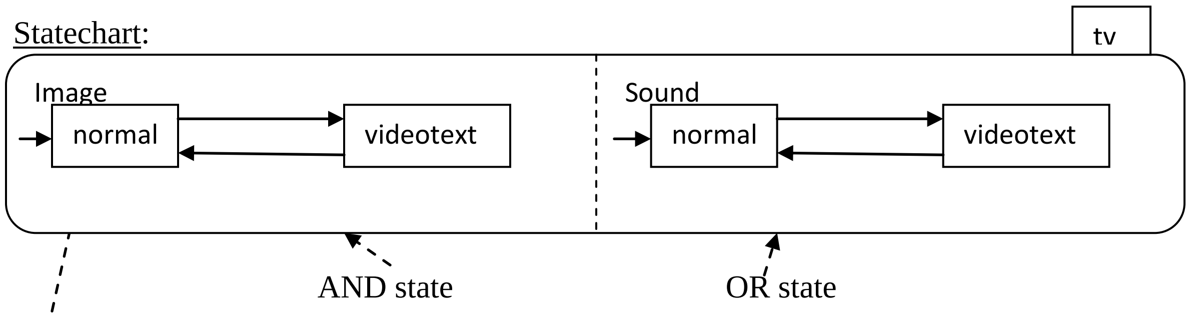
Mealy machine of the parallel composition of image and sound

:



number of states is exponential

Statechart:



Basic state: state which can't further decomposed

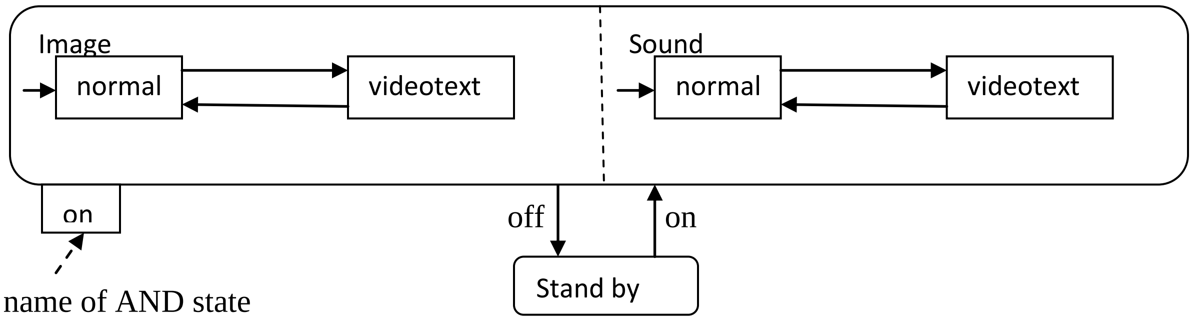
Dashed line in between: conjunction, run parallel

The difference between statechart with the previous Mealy machine is this system consists of 2 automata running concurrently.

3.3. Hierarchy

Example of hierarchy in statechart:

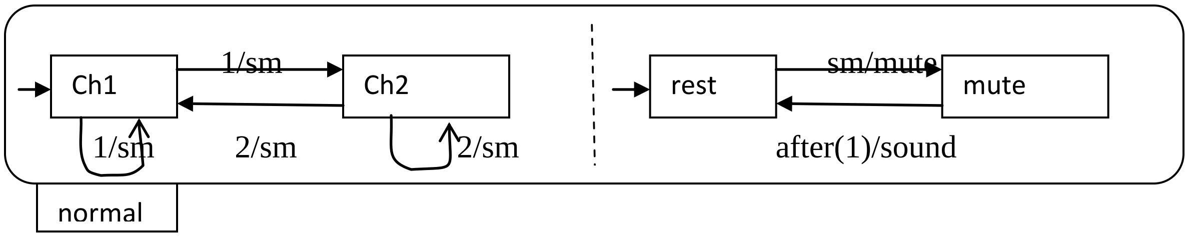
Switching on and switching off the television



3.4. Broadcast

Example of broadcast communication in statechart:

Turn off sound while changing a channel



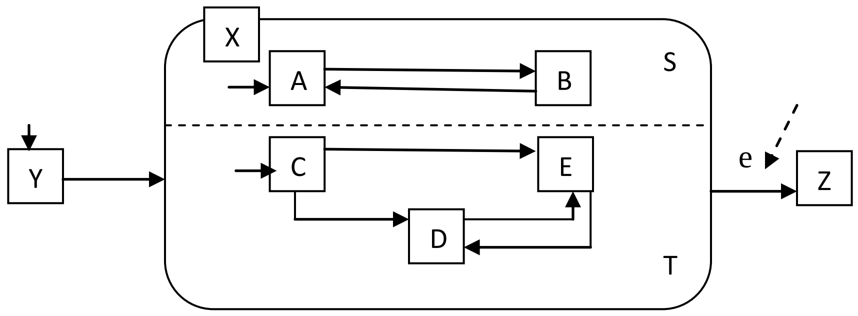
Output is broadcast that can be received by any other component.

In this statechart, if we press 1 or 2, the channel will be switch, and it produce signal sm. On the occurrence of sm, the right component can be executed, and the sound will be muted, after 1 second it backs into normal again.

3.5. Concurrency

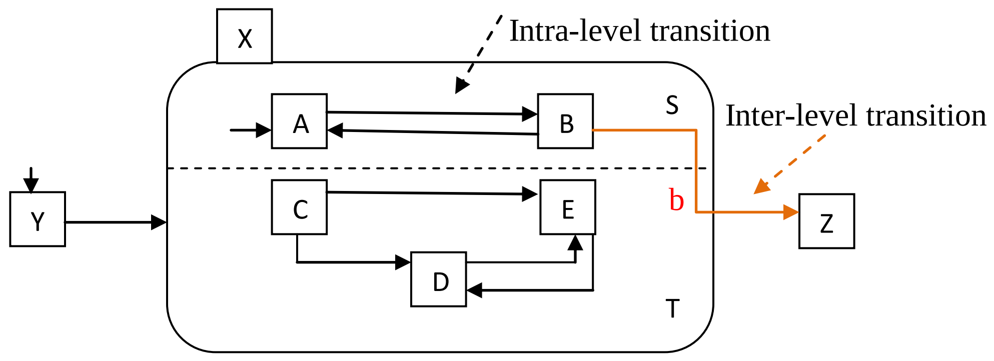
Example of concurrency in statechart:

Statechart 1:



State X is decomposed into 2 sub states: S and T, thus X is a super state of S and T.
As long as X is "active", S and T are "active".
S is "active" when either A or B is "active".
T is "active" if either C,D, or E (one of them) is "active".
When X exits, both S and T exit together.
When Y exits, X starts, S in A, T in C.
On the *occurrence of event e, X exits* (regardless of the current state in S or T).

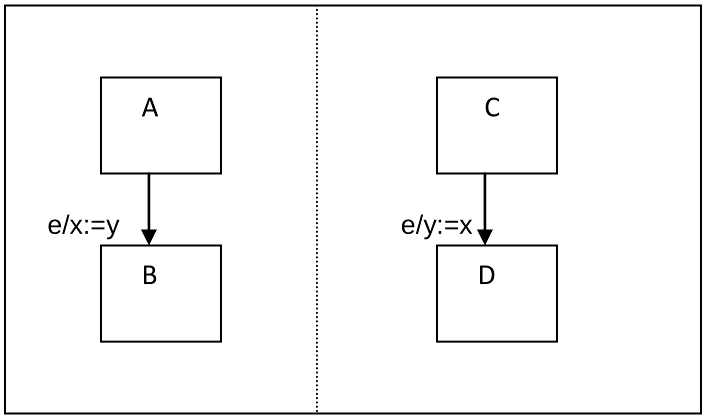
Statechart 2:



In the figure above, S may autonomously decide to move to Z on the *occurrence of event b*, while being in state B and T just continues.

Inter Level Transitions are possible in statecharts. The above example is 1 level deeper. Similarly there is 2 level and higher level transitions are possible.

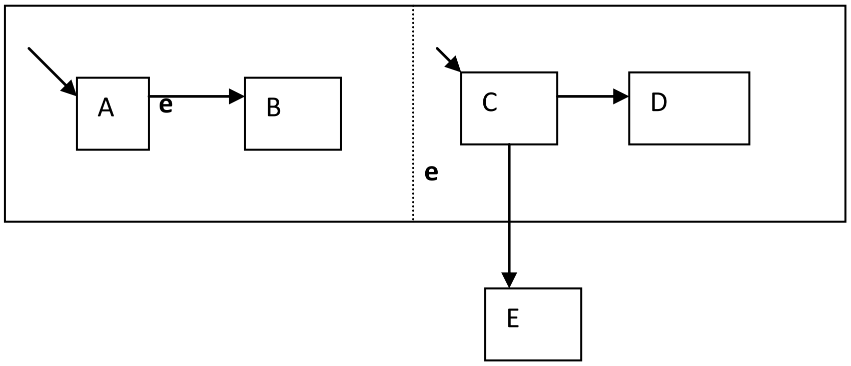
3.6. Swapping 2 variables



- x and y are global variables.
- If A and C are active, x=1 and y=2
- Event e occurs
- Next status: B and D are active ,x=2 and y=1
- Variables x and y are thus swapped

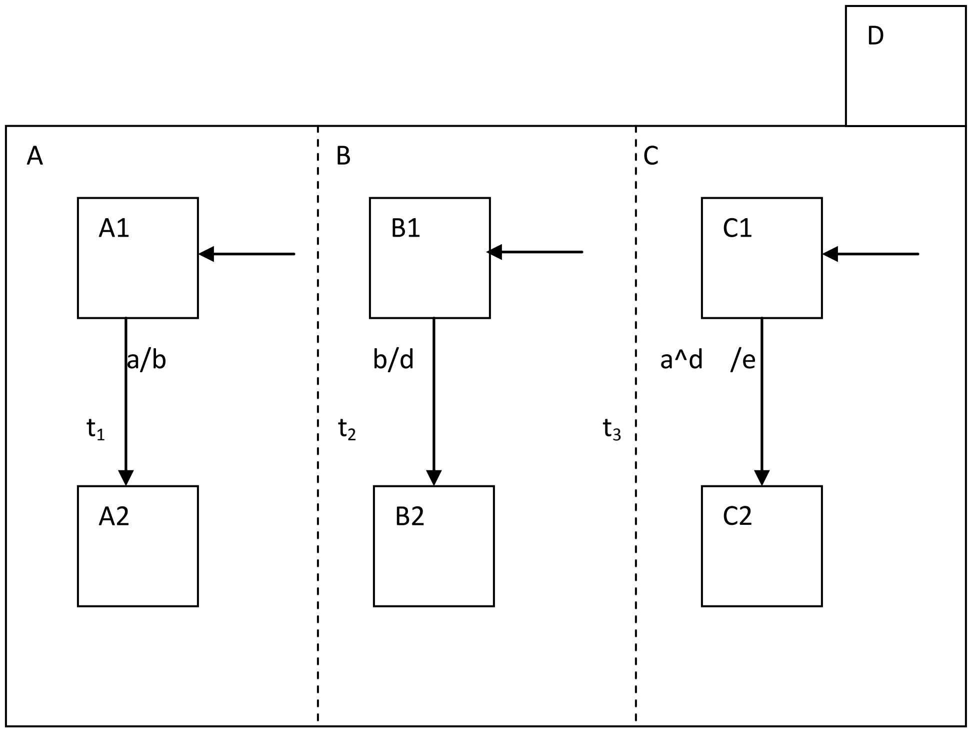
Memory is shared, i.e. concurrent processes see changes to variables immediately.

3.7. Priority



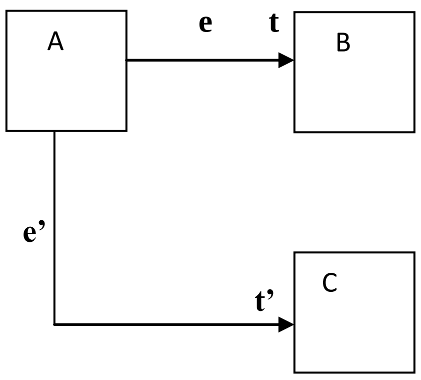
- **Event e** happens when State A and C are active
- Add Priority mechanism that decides whether inter level(C to E) or intra level transitions (A to B) transitions prevail.
- **Inter Level-Transitions** occurs between different levels
- **Intra Level-Transitions** occurs at same Level

3.8. Zero Response Time



- Here A1,B1 and C1 are active states.
- And when the event **a** occurs it generates output signal **b**
- And this **b** is used by B1 and it generates output signal **d.**
- As all this are happening in Zero Response Time ,**event a and event b** together acts on C1 generating signal **e**
- That is on occurrence of **event a** , a chain reaction occurs,t1 triggers t2, and t2 triggers t3
- But transitions t1,t2,t3 occurs at the **same time** as events do not take time

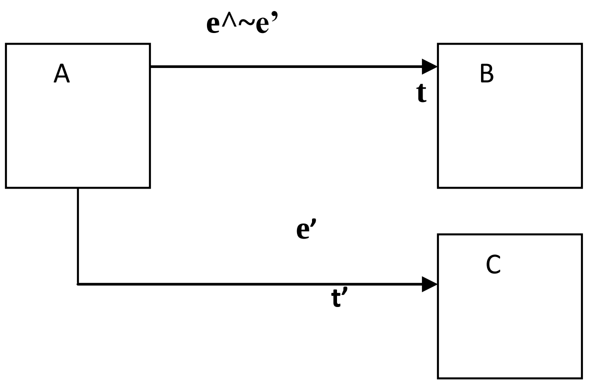
3.9. Nondeterminism



In the above figure

- when A is active and events e and e' happen.
- Then it can move to either State A or State B, but not both.
- This is called Non Determinism

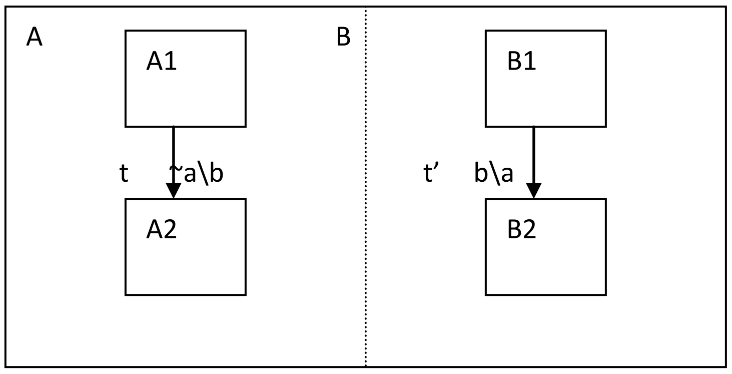
3.10. Negation of Events



Priority of t' over t can be specified by negated events, i.e. in the absence of events

- This may solve non determinism (but not always)
- According to the above diagram when **e happens and e' is not there** (t), it will move to state B.
- And when both **e and e'happens** (t') will move to state C.

3.11. Paradox



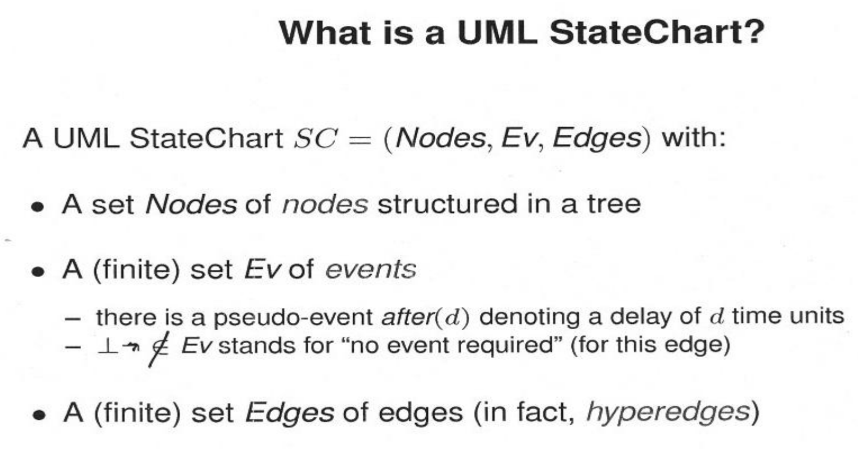
The Paradox is due to negated events and Zero Response Time.

- Suppose states A1 and B1 are active.
- Initially no events either a or b occurs.

- As **event a** is not happening , state A1 generates **b**(transition t)
- And **event b** is used by B1 generating **a**. (transition t')
- According to Zero Response time this may not happen. As **event a, is generated**, **transition t** should not have taken place.
- So **t'** cannot happen as **b** does not occur.
- And as **t'** is not occurred ,**event a** is not generated.
- And this again triggers **t.**
- And so forth.

4. UML Statecharts

4.1 UML Statecharts Definition

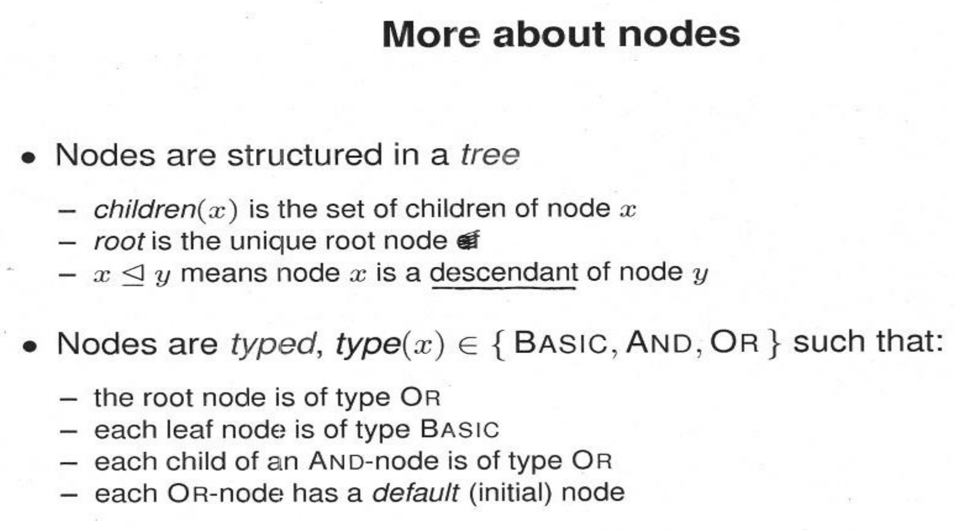


What is a UML StateChart?

A UML StateChart $SC = (\mathit{Nodes}, \mathit{Ev}, \mathit{Edges})$ with:

- A set *Nodes* of *nodes* structured in a tree
- A (finite) set *Ev* of *events*
  - there is a pseudo-event *after*(d) denoting a delay of d time units
  - $\bot \notin$ *Ev* stands for "no event required" (for this edge)
- A (finite) set *Edges* of edges (in fact, *hyperedges*)

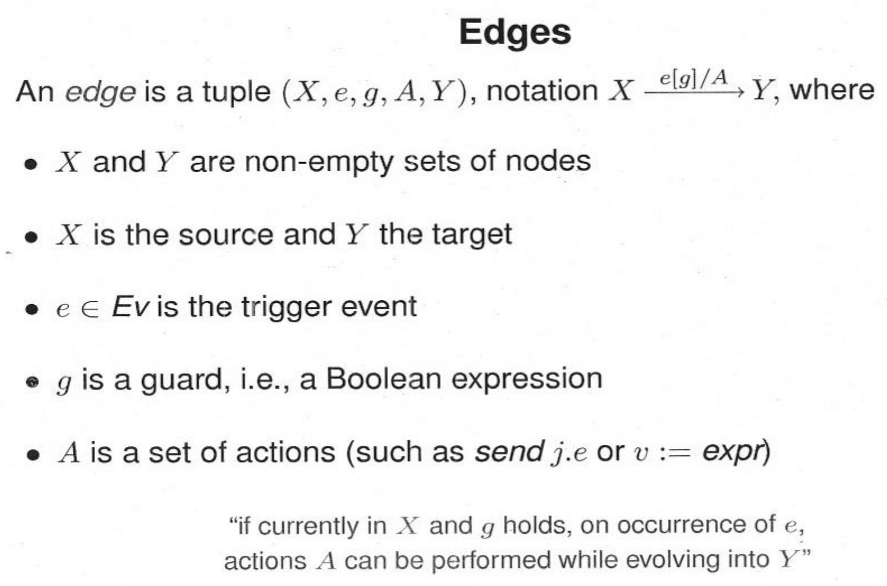
4.2 Node



More about nodes

- Nodes are structured in a *tree*
  - *children*(x) is the set of children of node x
  - *root* is the unique root node
  - $x \trianglelefteq y$ means node x is a descendant of node y
- Nodes are *typed*, *type*$(x) \in \{$ BASIC, AND, OR $\}$ such that:
  - the root node is of type OR
  - each leaf node is of type BASIC
  - each child of an AND-node is of type OR
  - each OR-node has a *default* (initial) node

4.3 Edges



Edges

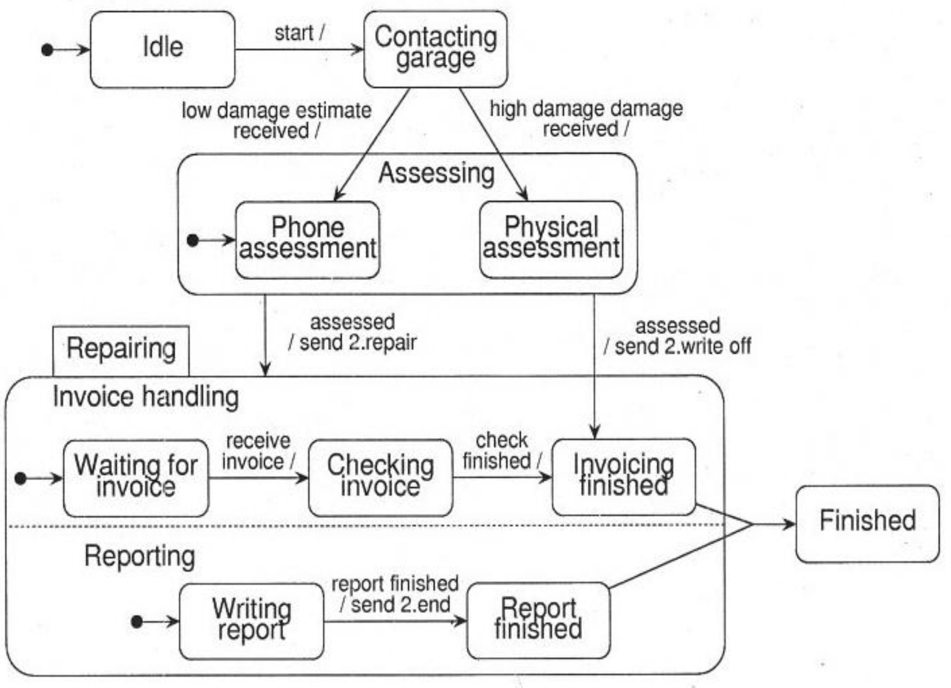
An *edge* is a tuple (X, e, g, A, Y), notation $X \xrightarrow{e[g]/A} Y$, where

- X and Y are non-empty sets of nodes
- X is the source and Y the target
- $e \in$ *Ev* is the trigger event
- g is a guard, i.e., a Boolean expression
- A is a set of actions (such as *send* $j.e$ or $v :=$ *expr*)

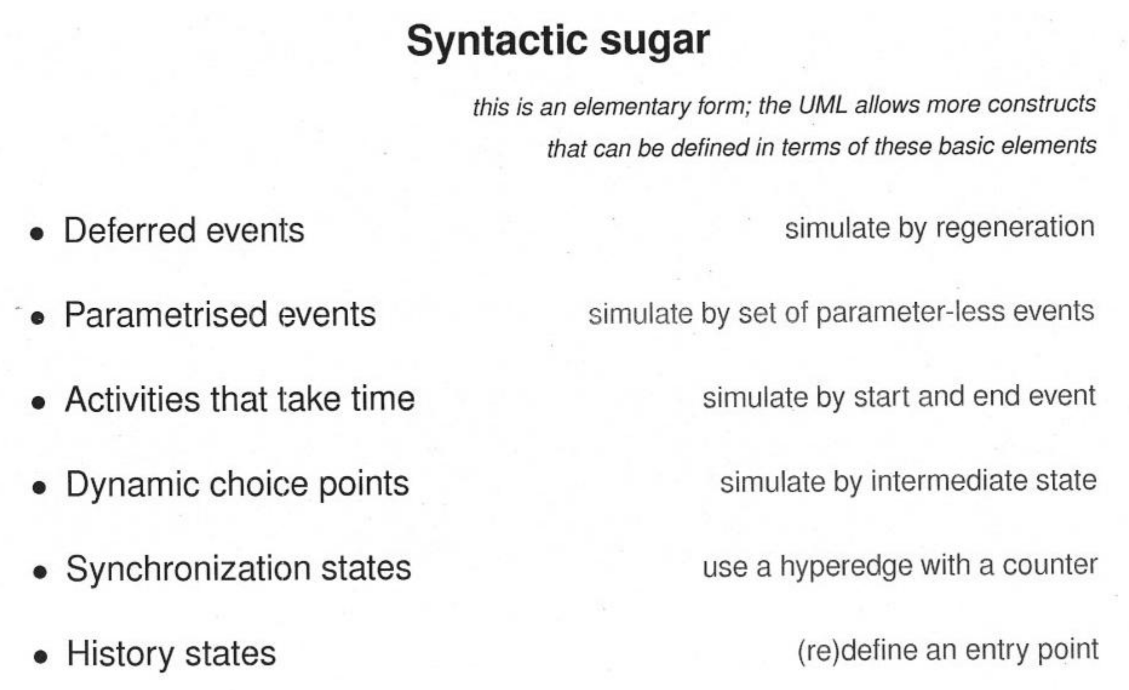
"if currently in X and g holds, on occurrence of e, actions A can be performed while evolving into Y"

4.4 Example of UML Statechart

A UML StateChart



State chart above explains that there are two different handling after *Assessing*, namely ***send to repair*** which will execute **the whole process in *Repairing*** and ***send to write off*** which only execute ***Invoicing finished*. *Repairing*** needs to finish *Invoicing* and *Report* in order to get all *finished*, but *Invoicing finished* doesn't need to wait approval from *Report finished* to further to *Finished* step.



Syntactic sugar

this is an elementary form; the UML allows more constructs that can be defined in terms of these basic elements

• Deferred events	simulate by regeneration
• Parametrised events	simulate by set of parameter-less events
• Activities that take time	simulate by start and end event
• Dynamic choice points	simulate by intermediate state
• Synchronization states	use a hyperedge with a counter
• History states	(re)define an entry point