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

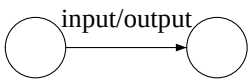
– Lecture number 9 –

(7th Jan 2008)
summarized by *Christian Dernehl* and *Martin Lang*

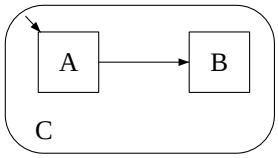
1 Statechart Ingredients

Statecharts consist of a collection of different ideas. Basically these are:

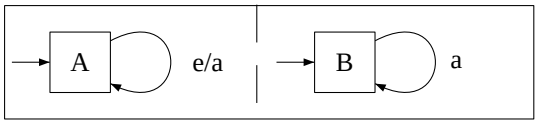
- Mealy machine



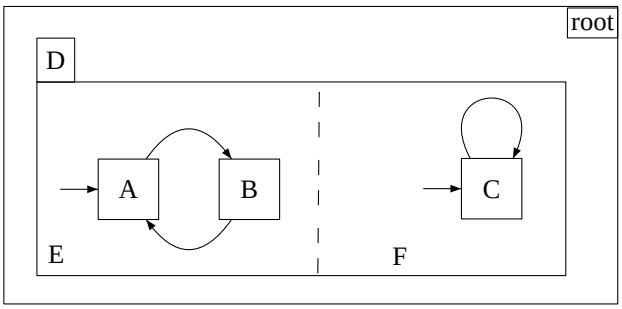
- Hierarchy



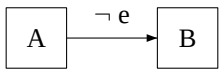
- Communication



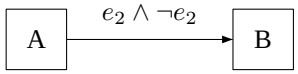
- Concurrency (orthogonality)



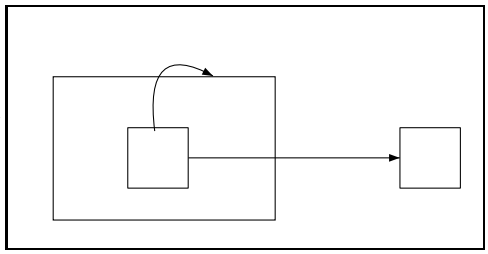
- Negated events



- Compound Events



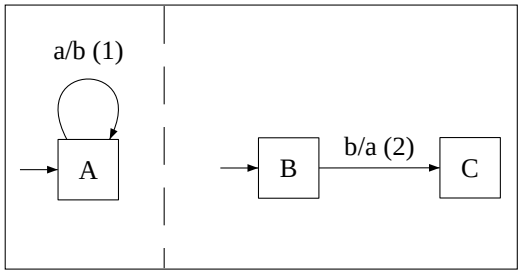
- Inter-level and intra-level edges



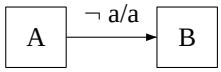
2 Some Problems

The rich possibilities of statecharts yield a number of problems and inconsistencies. In the following some of them are presented to deliver you an insight.

- **zero response time assumption:**
  input and corresponding output occur at the same time
- **self-triggering:**
  The zero response time induces a problem on 2 edges which depend on an event generated as output by the other edge. In the example edge 1 depends on the event “a”. By assumption of zero response time this is generated by edge 2 because edge 1 generated event “b” which triggers edge 2. So edge 2 "triggers" itself.

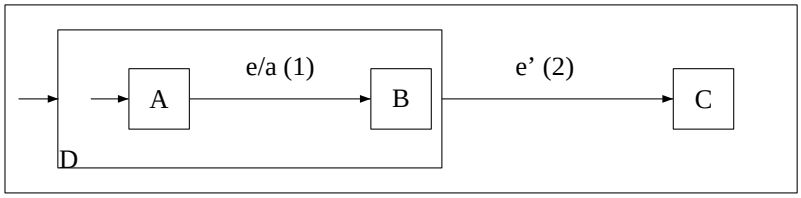


- **effect of edge may contradict its cause**: The possibility of negated events combined with zero response time leads to a contradiction in events that generate an output contradicting its cause. In the example the event a is generated when it is absent. Because of zero response time this edge could be taken if it could not be taken.



This is a very obvious example but it is not sufficient to eliminate only such edges to avoid the given problem because it can also occur over a chain of edges triggered at the same time. For instance there is no event a. An edge $\neg a/b$ is taken. This generates event b, so that a second edge b/a could be taken that results in the same problem shown above.

- **competing events:**



What happens when "A" is active and events e and e' occur?

a) Edge 2 prevents edge 1 from happening and event a is not generated.

b) Both edges occur. Event a is generated. Is the system for zero time in "B"? Would other edges possibly from "B" be taken?

c) Only edge 1 occurs. Event a is generated but the system does not change to state "C"

To avoid such problems in the interpretation of statecharts we like to impose certain restrictions on the model and formalize it. Though we get an definite interpretation of a statechart.

3 Simplifications

- No global variables
- No compound events
- No negated events
- Communication is biliteral (no broadcast)
- Events generated in one step (i) can only be consumed in the following step (i+1) and decay otherwise.

4 Formal Definition

Formally a statechart is described as a triple $(N, E, Edges)$. The components are in detail:

- N is the set of *nodes*. It contains all types of nodes regardless the hierarchy. Types and ordering are defined later with special functions. According to the terms used with automata the nodes are sometimes also called states.
- E is the set of the *events* which can trigger the transitions in the statechart.
- $Edges$ is the set of *edges* or *transitions*. In the graphical representation of a statechart they are shown as arrows between the nodes. The elements contained by the *Edges*-set are described later in detail.

4.1 Nodes

The tree structure of the nodes is formally represented by a function that returns the immediate child-nodes of a given node. Therefore this function will be called *children*:

$$\text{children} : N \rightarrow 2^N$$

For example given 2 nodes A and B, A is the parent of B. So $B \in \text{children}(A)$.

This function induces a partial order (denoted $\leq$) on N representing the hierarchy between the nodes.

- $x \leq x$
- $x \leq y$ if $x \in \text{children}(y)$
- $x \leq y \wedge y \leq z \Rightarrow x \leq z$

Because of the transitivity of the order-relation it is possible to compare nodes over the levels of the tree. So $x \leq y (x \neq y)$ means that x is at a deeper level of the hierarchy than y. All nodes that are ancestrally related are comparable within this relation (either $x \leq y$ or $y \leq x$ $(y \neq x)$).

There is a *unique root* node which is not referenced by the children function. All nodes are descendants of this node ($\forall x \in N : x \leq root$)

Statecharts differentiate between *and*, *or*, and *basic* nodes to determine their meaning for concurrency. To identify the type of a node, a second function for the nodes is introduced matching the type to each node.

$$\text{type} : N \rightarrow \{and, or, basic\}$$

With the properties:

- $\text{type}(root) = or$
  The root node should not contain any concurrent nodes.
- $\text{type}(x) = basic \Leftrightarrow \text{children}(x) = \emptyset$
  Only the leaf nodes, which have no children, are basic-nodes.

- $\text{type}(x) = and \Rightarrow \forall y \in \text{children}(x) : \text{type}(y) = or$
  Nodes that model concurrency should only contain or-nodes specifying the concurrent behavior.

Like an automaton statecharts must have starting nodes. In comparison to a normal flat automaton, statecharts have starting nodes on the different levels of hierarchy. Therefore every or-node has one starting node which is identified by the partial function *default*. This starting node gets active when the given or-node is entered.

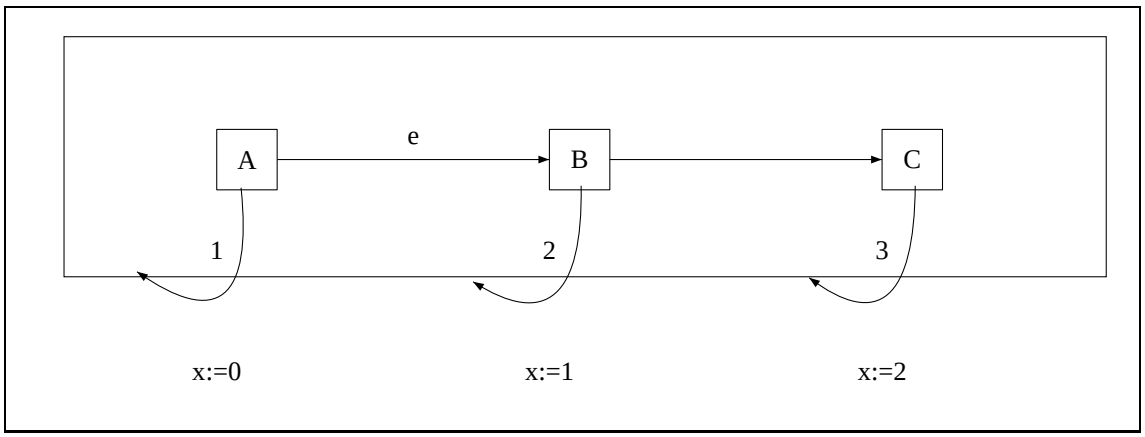
$$\text{default} : N \to N, \;\; \text{dom}(\text{default}) = \{x \in N | \, \text{type}(x) = or\}$$

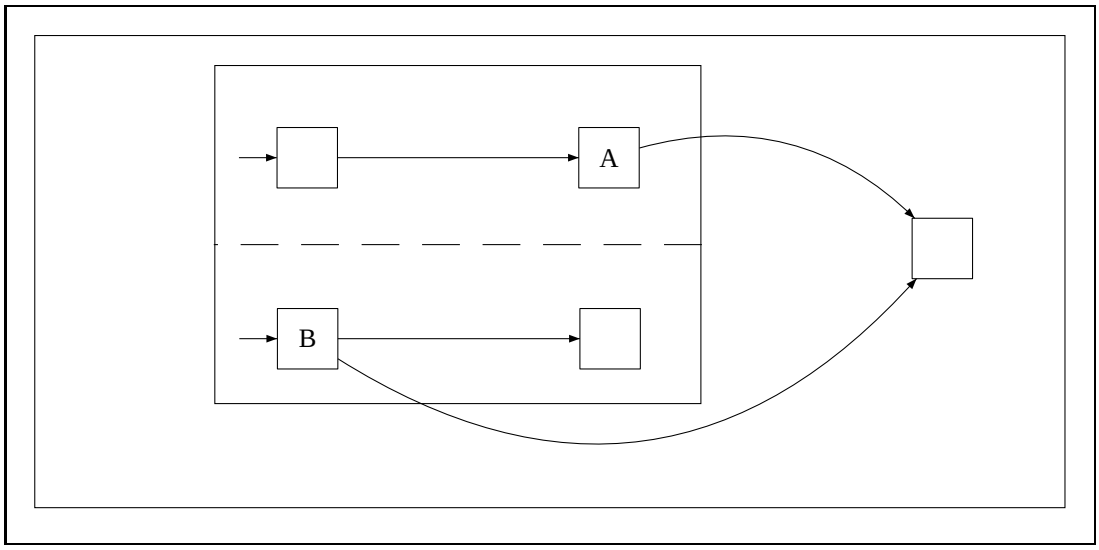
It is important, that the statement $\text{default}(x) = y \Rightarrow y \in \text{children}(x)$ is fulfilled, because the starting node should be directly contained by one given or-node to take the hierarchy into account.

4.2 Edges

Edges are defined by a quintuple (X, e, g, A, Y):

- $X \subseteq N, X \neq \emptyset$ is the set of *source nodes*. All these nodes must be active for the edge getting enabled.
- $e \in E \cup \{\perp\}$ is the *trigger event*. This event must be active for the edge being taken where $\perp$ means "no event" (comparable to ε-transitions in NFAs)
- $g \in \text{Cond}(Var)$ is a *guard expression*. This is a boolean expression over all variables in the given statechart. The edge can only be taken if this guard evaluates to true.
- $A \subseteq Act$ is a set of *actions* that occur when the edge is taken. For instance setting of a local variable or sending an event to another statechart.
- $Y \subseteq N, Y \neq \emptyset$ is a set of *target nodes*. All these nodes get active when the edge is taken. Therefore it should be possible for all elements of Y to be active at the same time.





It should be noticed that the sets X and Y may contain nodes at different levels of the hierarchy.

5 Sequential Step Semantics

Two notions of a step:

- **macro steps**: observable “time” steps
- **micro steps**: a set of edges

A macro step can be subdivided in an arbitrary but finite number of micro steps that cannot be prolonged anymore.

Events generated in macro step i are only available in the next macro step (i+1). If an event is not consumed, it decays and has no more influence on the following macro steps.

6 Assumptions

- *The input to a macro step is a set of events.*
  As a direct consequence of this the order of event generation is ignored and all edges that are triggered by those events are equal. Additionally it follows that the order of events that are generated in step i is not of relevance in step i+1.
- *A macro step reacts to all available events.*
  By executing a macro step all events are processed - even those which does not trigger any edges of the statechart. And therefore they can only be “used” in the immediately following step after their generation. All possible edges that could be taken without conflict are executed.

- *Instantaneous edges and actions*
  The execution of edges and action needs zero time.
- *Unlimited concurrency*
  No limitation on the events that are consumed by a macro step.
- *Perfect communication*
  No messages are lost in the system.

7 States and Configurations

A *configuration* of a statechart $SC = (N, E, Edges)$ is a set $C \subseteq N$ that contains the active nodes. Therefore it must fulfill the following conditions:

- $root \in C$
- $x \in C \wedge \text{type}(x) = or \Rightarrow |\text{children}(x) \cap C| = 1$
  In an or-node only one direct child can be active at a given point.
- $x \in C \wedge \text{type}(x) = and \Rightarrow \text{children}(x) \subseteq C$
  All direct children of an active and-node must be active, too.

In a system with more than one statechart $Conf_i$ is the *set of possible configurations* of the statechart SC_i.

Based on the configuration of a statechart, the *state* of a statechart is now defined as a triple (C, I, V) where the components are:

- $C \in Conf$ describes the current configuration
- $I \subseteq E$ identifies which events are currently active.
- V is a *valuation* of the statechart variables at this time.

7.1 Enabled edges in a state

To compute a step in a statechart the first question that should be answered is which edges could be taken. These edges are called enabled.

An edge $X \xrightarrow{e[g]/A} Y$ is enabled in a given state (C, I, V) if:

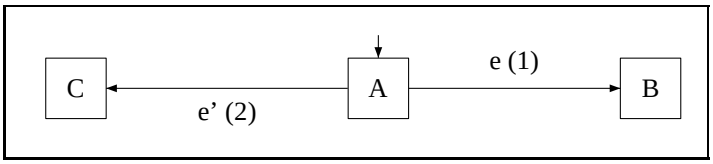
- $X \subseteq C$ - All source nodes of the edge are currently active.
- $e \in I$ - The event that triggers this edge is active.
- $(\underbrace{(C_1, \ldots, C_n)}_{\text{configurations}}, \underbrace{(V_1, \ldots, V_n)}_{\text{valuations}}) \models g$ - The guard g is satisfied by the variables and configurations.

The set of enabled edges in configuration (C, I, V) is denoted as $\text{En}(C, I, V)$.

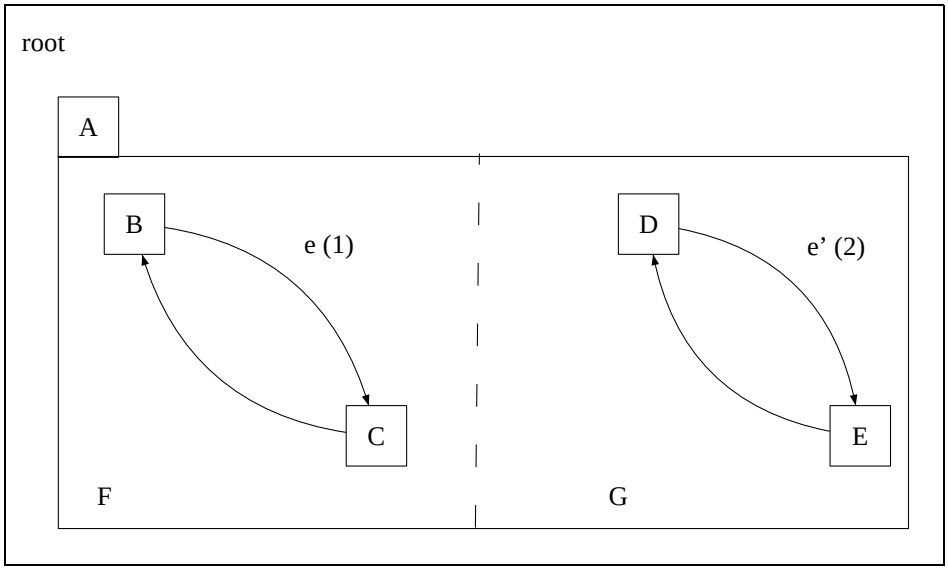
8 Consistency

The edges executed must lead to a valid new configuration. Therefore only consistent edges could be taken at the same time. If some edges are inconsistent the system has a nondeterministic choice between the different consistent sets of edges when there are no other mechanisms to decide which of them should be taken.

To get an impression of what kind such edges are take a look at the following 2 examples. Let the current state be $(\{root, A\}, \{e, e'\}, \{\})$.



In the next macro step only the edge triggered by e *or* e' could be taken because the nodes C and B can not both be active at the same time. So the edges labeled with e and e' are inconsistent.



Let this statechart be currently in the state $(\{root, G, F, A, A', B, D\}, \{e, e'\}, \{\})$. In this example the edges could both be taken in the next macro step because C and E may be active at the same time due to the arrangement in the and-node A.

To define what consistency exactly means we use orthogonality.

For $X \subseteq N$, the *least common ancestor*, shortly $\mathrm{lca}(X)$, is the node $y \in N$ that:

- $\forall x \in X : x \leq y$
- $\forall z \in N : (\forall x \in X : x \leq z) \Rightarrow y \leq z$

That means that y is an ancestor of all nodes in X and there is no node deeper in the hierarchy that also fulfills this property.

Nodes $x, y \in X$ are called *orthogonal*, denoted $x \perp y$, if they are not comparable in the hierarchy relation ($x \not\leq y \wedge y \not\leq x$) and $\text{type}(\text{lca}(\{x, y\})) = and$.

The *scope* of an edge $X \longrightarrow Y$ is the most nested or-node ancestor of X and Y. Note that a node is an ancestor of a set of nodes if it is an ancestor of all nodes in the set.

Now we can define *consistency* of two edges in the following way:

Two edges $ed, ed' \in Edges$ are *consistent* if:

$$ed = ed' \ \vee \ \text{scope}(ed) \perp \text{scope}(ed')$$

A set of edges T is consistent if all elements of T are pairwise consistent ($\forall ed, ed' \in T : (ed, ed')$ is consistent)