Theoretical Foundations of the UML

Lecture 17: Introduction to Statecharts

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

moves.rwth-aachen.de/teaching/ss-16/theoretical-foundations-of-the-uml/

11. Juli 2016

Outline

Overview

1. Background
2. Ingredients of Statecharts
   - Mealy Machines
   - State Hierarchy
   - Orthogonality
   - Broadcast Communication
   - Some Small Examples
   - Other Features: Priority, Nondeterminism and Negated Events
3. Semantics of Statecharts
4. Formal Definition of UML Statecharts

Statecharts

- MSCs are a visual modelling formalism for requirements

Statecharts

- MSCs are a visual modelling formalism for requirements
- Statecharts is a visual modelling formalism for describing the behaviour of discrete-event systems
  - automata + hierarchy + communication + concurrency

Statecharts

- MSCs are a visual modelling formalism for requirements
- Statecharts is a visual modelling formalism for describing the behaviour of discrete-event systems
  - automata + hierarchy + communication + concurrency
- Developed by David Harel in 1987
  - professor at Weizmann Institute (Israel); co-founder of I-Logix Inc.

Statecharts

- MSCs are a visual modelling formalism for requirements
- Statecharts is a visual modelling formalism for describing the behaviour of discrete-event systems
  - automata + hierarchy + communication + concurrency
- Developed by David Harel in 1987
  - professor at Weizmann Institute (Israel); co-founder of I-Logix Inc.
- Extensively used in embedded systems, automotive and avionics

Statecharts

- MSCs are a visual modelling formalism for requirements
- Statecharts is a visual modelling formalism for describing the behaviour of discrete-event systems
  - automata + hierarchy + communication + concurrency
- Developed by David Harel in 1987
  - professor at Weizmann Institute (Israel); co-founder of I-Logix Inc.
- Extensively used in embedded systems, automotive and avionics
- Variants: UML Statecharts, Stateflow, hierarchical state machines
  - supported by Statemate toolset, and Matlab/Simulink

Overview

1. Background
2. Ingredients of Statecharts
   - Mealy Machines
   - State Hierarchy
   - Orthogonality
   - Broadcast Communication
   - Some Small Examples
   - Other Features: Priority, Nondeterminism and Negated Events
3. Semantics of Statecharts
4. Formal Definition of UML Statecharts

What are Statecharts?

Statecharts constitute a visual formalism for: [Harel, 1987]

What are Statecharts?

Statecharts constitute a visual formalism for: [Harel, 1987]

- Describing states and transitions in a modular way

What are Statecharts?

Statecharts constitute a visual formalism for: [Harel, 1987]

- Describing states and transitions in a modular way
- Enabling clustering of states

What are Statecharts?

Statecharts constitute a visual formalism for: [Harel, 1987]

- Describing states and transitions in a modular way
- Enabling clustering of states
- Orthogonality, i.e., concurrency

What are Statecharts?

Statecharts constitute a visual formalism for: [Harel, 1987]

- Describing states and transitions in a modular way
- Enabling clustering of states
- Orthogonality, i.e., concurrency
- Refinement, and

What are Statecharts?

Statecharts constitute a visual formalism for: [Harel, 1987]

- Describing states and transitions in a modular way
- Enabling clustering of states
- Orthogonality, i.e., concurrency
- Refinement, and
- Encouraging "zoom" capabilities for moving easily back and forth between levels of abstraction

What are Statecharts?

$$\begin{aligned} \text{Statecharts} := & \text{ Mealy machines} \\ & + \text{State hierarchy} \\ & + \text{Broadcast communication} \\ & + \text{Orthogonality} \end{aligned}$$

Mealy machines [Mealy, 1953]

Definition (Mealy machine)

A Mealy machine $\mathcal{A} = (Q, q_0, \Sigma, \Gamma, \delta, \omega)$ with:

- Q is a finite set of states with initial state $q_0 \in Q$
- Σ is the input alphabet
- Γ is the output alphabet
- $\delta : Q \times \Sigma \to Q$ is the deterministic (input) transition function, and
- $\omega : Q \times \Sigma \to \Gamma$ is the output function

Mealy machines [Mealy, 1953]

Definition (Mealy machine)

A Mealy machine $\mathcal{A} = (Q, q_0, \Sigma, \Gamma, \delta, \omega)$ with:

- Q is a finite set of states with initial state $q_0 \in Q$
- Σ is the input alphabet
- Γ is the output alphabet
- $\delta : Q \times \Sigma \to Q$ is the deterministic (input) transition function, and
- $\omega : Q \times \Sigma \to \Gamma$ is the output function

Intuition

A Mealy machine (or: finite-state transducer) is a finite-state automaton that produces output on a transition, based on current input and state.

Mealy machines [Mealy, 1953]

Definition (Mealy machine)

A Mealy machine $\mathcal{A} = (Q, q_0, \Sigma, \Gamma, \delta, \omega)$ with:

- Q is a finite set of states with initial state $q_0 \in Q$
- Σ is the input alphabet
- Γ is the output alphabet
- $\delta : Q \times \Sigma \to Q$ is the deterministic (input) transition function, and
- $\omega : Q \times \Sigma \to \Gamma$ is the output function

Intuition

A Mealy machine (or: finite-state transducer) is a finite-state automaton that produces output on a transition, based on current input and state.

Moore machines

In a Moore machine $\omega : Q \to \Gamma$, output is purely state-based.

Mealy machines

Mealy machines

- No final (accepting) states
- Transitions produce output
- Deterministic input transition function

$\Rightarrow$ Acceptance of input words is not important, but the generation of output words from input words is important

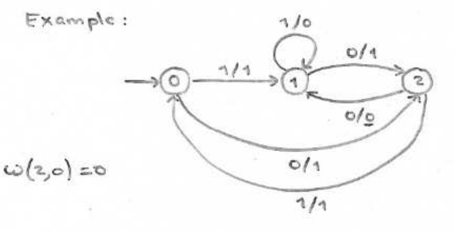
Mealy machines

Mealy machines

- No final (accepting) states
- Transitions produce output
- Deterministic input transition function

⇒ Acceptance of input words is not important, but the generation of output words from input words is important

Example



Limitations of Mealy machines

- No support for hierarchy
  - all states are arranged in a flat fashion
  - no notion of substates

Limitations of Mealy machines

- No support for hierarchy
  - all states are arranged in a flat fashion
  - no notion of substates
- Realistic systems require complex transition structure and huge number of states
  - scalability problems yields unstructured state diagrams

Limitations of Mealy machines

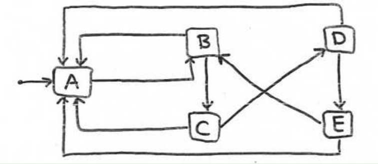
- No support for hierarchy
  - all states are arranged in a flat fashion
  - no notion of substates
- Realistic systems require complex transition structure and huge number of states
  - scalability problems yields unstructured state diagrams
- No notion of concurrency
  - need for modeling independent components

Limitations of Mealy machines

- No support for hierarchy
  - all states are arranged in a flat fashion
  - no notion of substates
- Realistic systems require complex transition structure and huge number of states
  - scalability problems yields unstructured state diagrams
- No notion of concurrency
  - need for modeling independent components
- No notion of communication between automata.

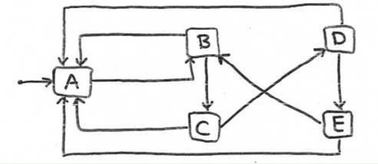
Scalability

A bit unstructured Mealy machine

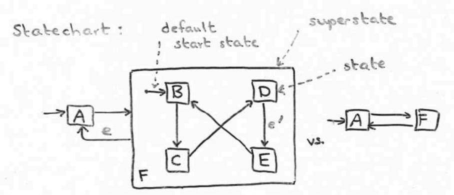


Scalability

A bit unstructured Mealy machine

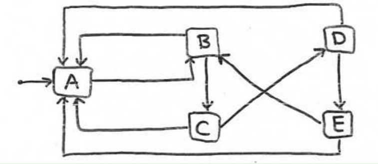


An equivalent statechart

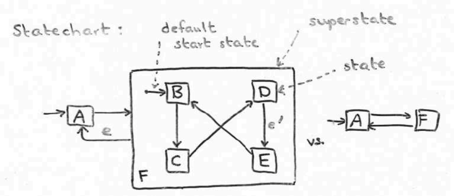


Scalability

A bit unstructured Mealy machine



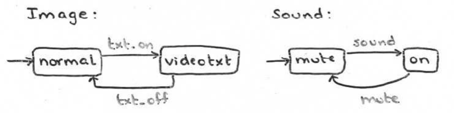
An equivalent statechart



State hierarchy yields modular, hierarchical and structured models.

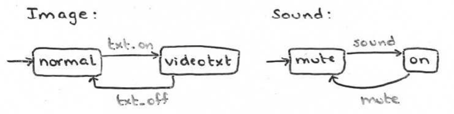
Orthogonality

Two independent components

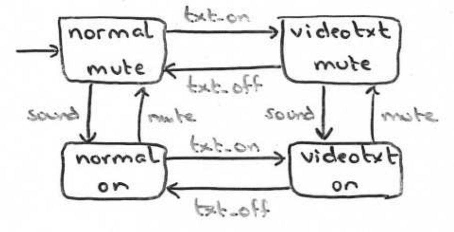


Orthogonality

Two independent components



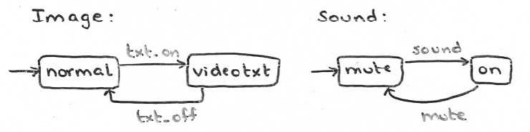
Mealy machine for $Image \parallel Sound$



Number of states is exponential in size of concurrent components

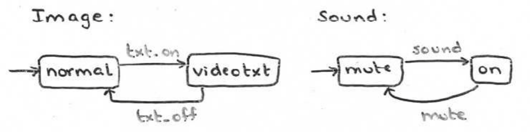
Orthogonality

Two independent components

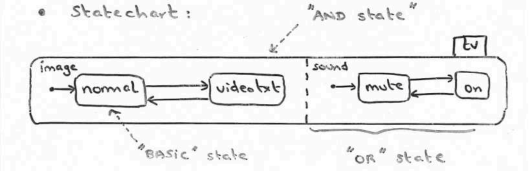


Orthogonality

Two independent components



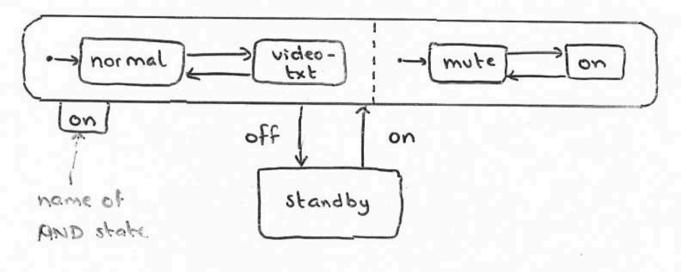
Statechart for *Image* ‖ *Sound*



Concurrency modeled by independence

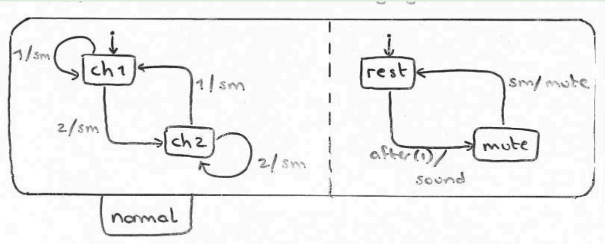
Combined with state hierarchy

Switching on and off the television



Broadcast

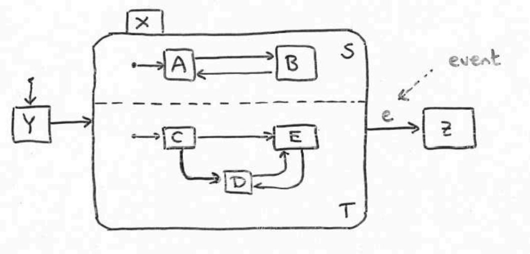
Turn off sound on switching a tv channe



- Output is broadcast that can be received by any other component
- When pushing button 1, channel switches to its state channel 1, while generating signal *sm* on which component *SM* switches off the sound.

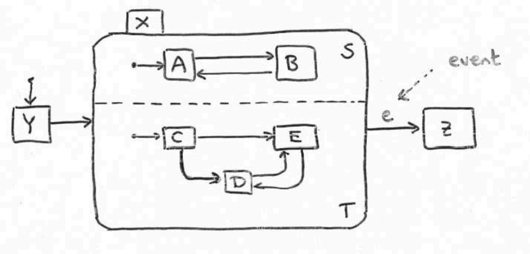
Concurrency

Example concurrency in statecharts



Concurrency

Example concurrency in statecharts

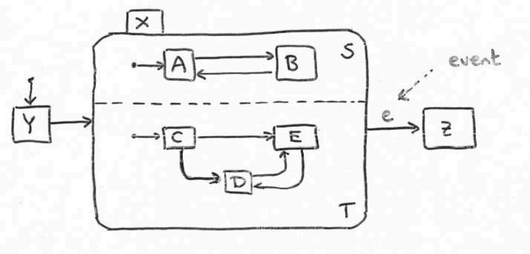


Active

- As long as node X is active, nodes S and T are active

Concurrency

Example concurrency in statecharts

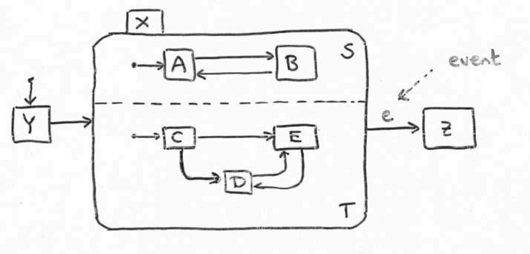


Active

- As long as node X is active, nodes S and T are active
- Node S is active when either node A or B is active

Concurrency

Example concurrency in statecharts

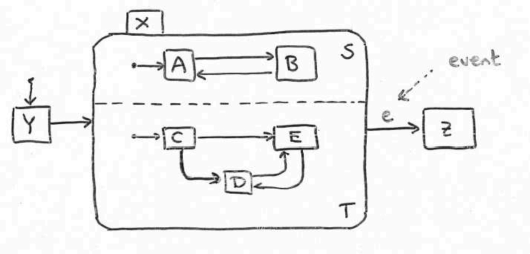


Active

- As long as node X is active, nodes S and T are active
- Node S is active when either node A or B is active
- Node T is active if one of C, D or E is active

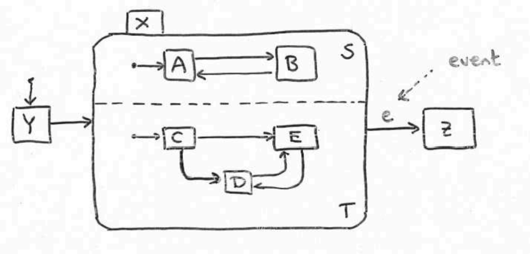
Concurrency

Example concurrency in statecharts



Concurrency

Example concurrency in statecharts

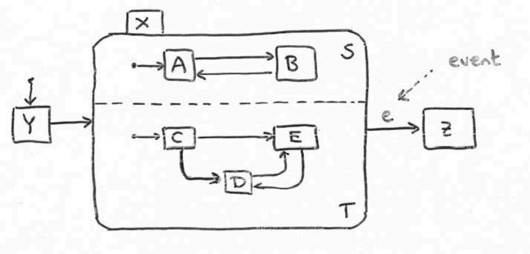


Exit behaviour

- When node X exits, both nodes S and T exit

Concurrency

Example concurrency in statecharts

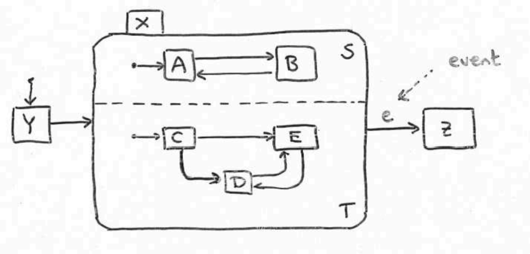


Exit behaviour

- When node X exits, both nodes S and T exit
- When Y exits, X starts, S starts in A, and T starts in C

Concurrency

Example concurrency in statecharts

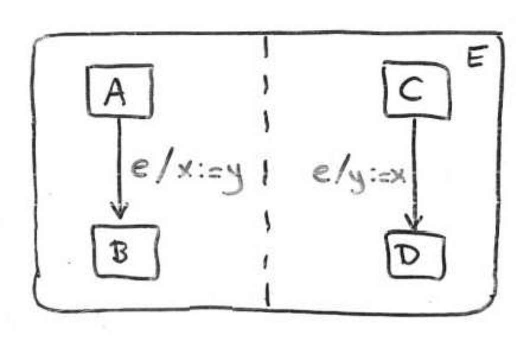


Exit behaviour

- When node X exits, both nodes S and T exit
- When Y exits, X starts, S starts in A, and T starts in C
- On the occurrence of event e, node X exits (regardless of current state in S or T)

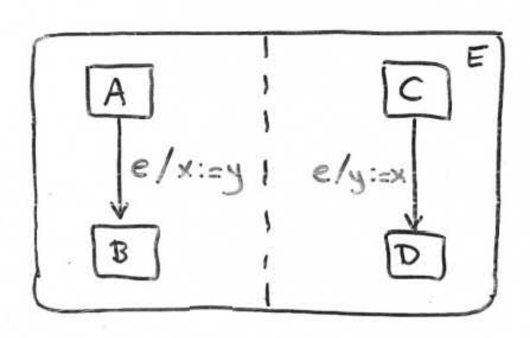
Swapping two variables

Swapping the value of variables x and y



Swapping two variables

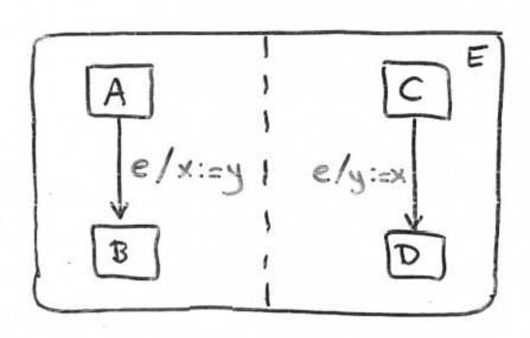
Swapping the value of variables x and y



- If nodes A and C are active, assume $x = 1$, $y = 2$

Swapping two variables

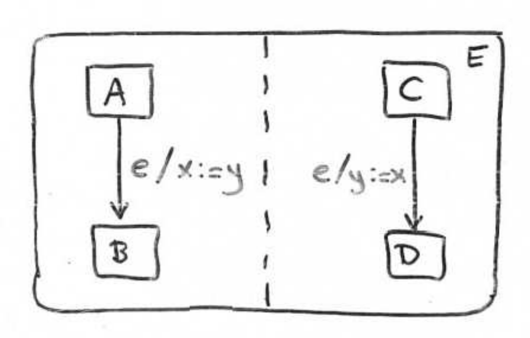
Swapping the value of variables x and y



- If nodes A and C are active, assume $x = 1$, $y = 2$
- On occurrence of event e, B and D are active, and $x = 2$, $y = 1$

Swapping two variables

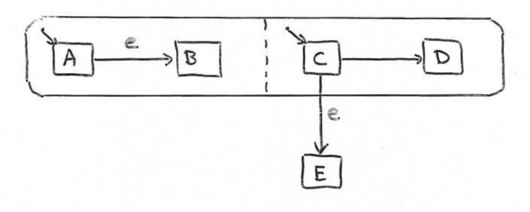
Swapping the value of variables x and y



- If nodes A and C are active, assume $x = 1$, $y = 2$
- On occurrence of event e, B and D are active, and $x = 2$, $y = 1$

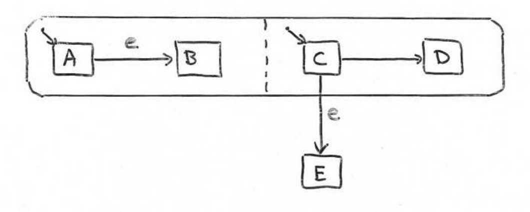
⇒ In Harel's statecharts, memory is shared, i.e., concurrent components have access to shared variables.

What if event e occurs when A and C are active?



Priority

What if event e occurs when A and C are active?



Solution:

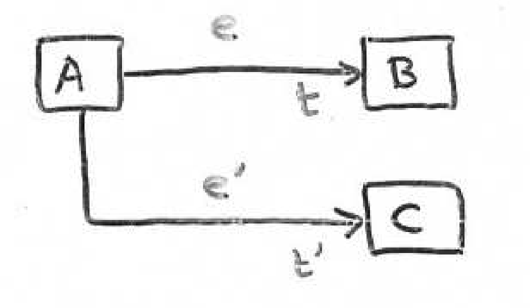
Add a priority mechanism that decides whether:

- inter-level transitions (such as $C \rightarrow E$), or
- intra-level transitions (such as $A \rightarrow B$)

prevail in case both are enabled.

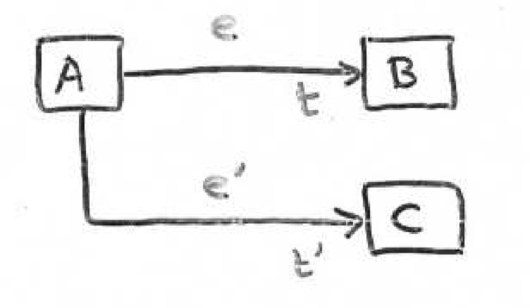
Nondeterminism

What if event e and e' occur in A?



Nondeterminism

What if event e and e' occur in A?

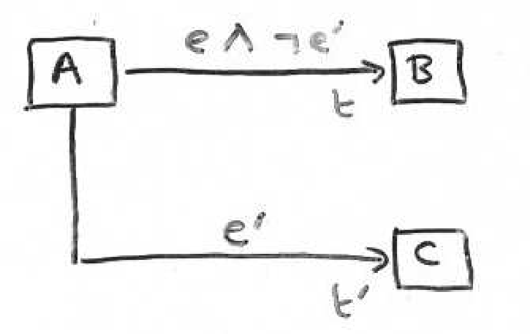


Solution:

Choice is resolved nondeterministically, i.e., the next state is either B or C, but not both.

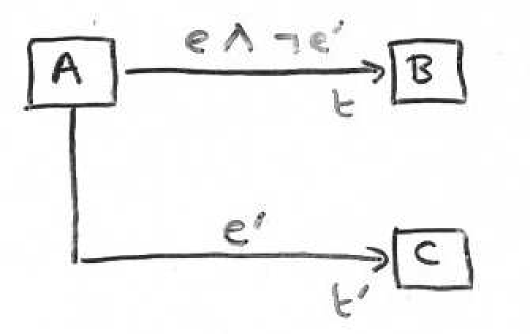
Negation of events

Priority of events by negated events



Negation of events

Priority of events by negated events



Note:

In UML statecharts, negated events do not occur

Overview

Semantic problems with Statecharts

- Synchrony hypothesis (or: zero response time)
- Self-triggering
- Negated trigger events
- Transition effect is contradicting its cause
- Interrupts

Semantic problems with Statecharts

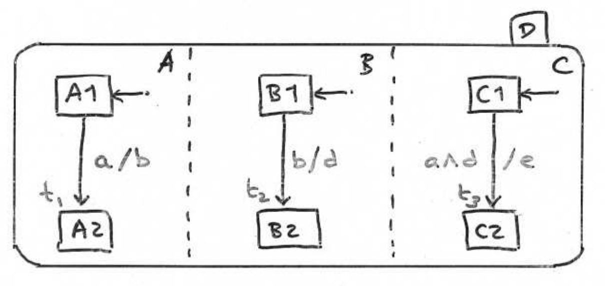
- Synchrony hypothesis (or: zero response time)
- Self-triggering
- Negated trigger events
- Transition effect is contradicting its cause
- Interrupts

Note: [von der Beeck, 1994]

Due to all these problems, hundred(s) (!) of different semantics for Statecharts have been defined in the literature.

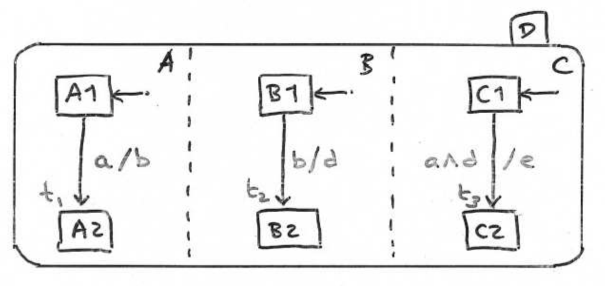
Synchrony hypothesis

Event may yield chain of reactions



Synchrony hypothesis

Event may yield chain of reactions

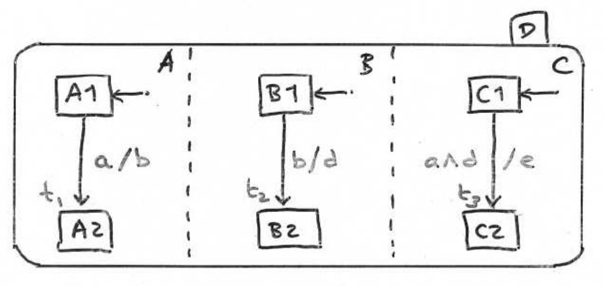


Note:

- If $A1$, $B1$ and $C1$ are active and event a occurs, a chain of reactions occurs: transition t_1 triggers t_2, and t_2 triggers t_3

Synchrony hypothesis

Event may yield chain of reactions

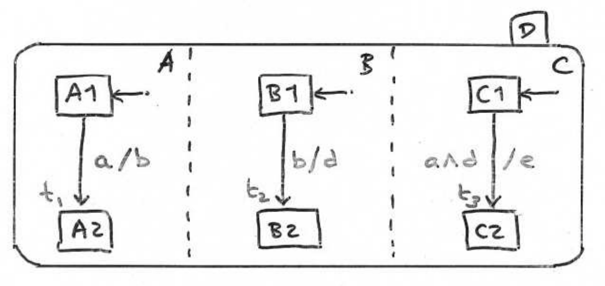


Note:

- If $A1$, $B1$ and $C1$ are active and event a occurs, a chain of reactions occurs: transition t_1 triggers t_2, and t_2 triggers t_3
- But transitions t_1, t_2, t_3 occur at the same time as events do not take time (except for $after(d)$ events with real d)

Synchrony hypothesis

Event may yield chain of reactions

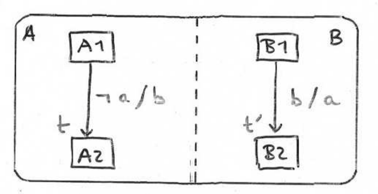


Note:

- If $A1$, $B1$ and $C1$ are active and event a occurs, a chain of reactions occurs: transition t_1 triggers t_2, and t_2 triggers t_3
- But transitions t_1, t_2, t_3 occur at the same time as events do not take time (except for *after*(d) events with real d)

Paradox

Negated events and synchrony may yield paradox

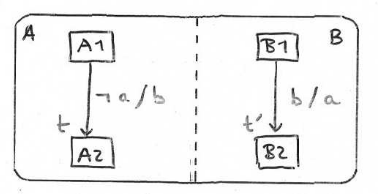


The paradox:

- Assume events a and b are not alive

Paradox

Negated events and synchrony may yield paradox

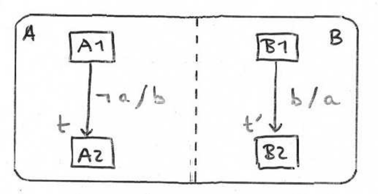


The paradox:

- Assume events a and b are not alive
- Transition t can be taken, generating event b

Paradox

Negated events and synchrony may yield paradox

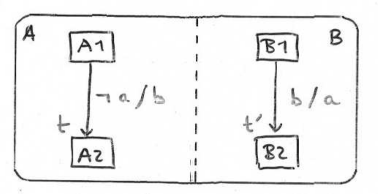


The paradox:

- Assume events a and b are not alive
- Transition t can be taken, generating event b
- Transition t' can be taken, generating event a

Paradox

Negated events and synchrony may yield paradox

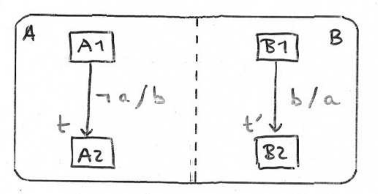


The paradox:

- Assume events a and b are not alive
- Transition t can be taken, generating event b
- Transition t' can be taken, generating event a
- But then t should not have taken place as it is not enabled

Paradox

Negated events and synchrony may yield paradox

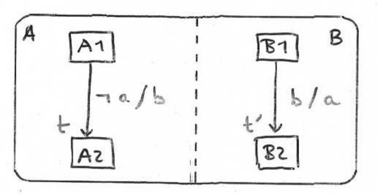


The paradox:

- Assume events a and b are not alive
- Transition t can be taken, generating event b
- Transition t' can be taken, generating event a
- But then t should not have taken place as it is not enabled
- But then t' cannot be taken since b does not occur

Paradox

Negated events and synchrony may yield paradox



The paradox:

- Assume events a and b are not alive
- Transition t can be taken, generating event b
- Transition t' can be taken, generating event a
- But then t should not have taken place as it is not enabled
- But then t' cannot be taken since b does not occur
- Hence, a does not occur and t cannot be taken

Simplifications in UML statecharts

1. No shared variables
2. No negated and no compound events (like $e \wedge e'$)
3. Two-party communication rather than broadcast
4. No synchrony hypothesis:
   - events generated in step i can only be consumed in step i+1,
   - and die otherwise, i.e., when they are not consumed in step i+1, events disappear

Overview

Definition (Statecharts)

A statechart SC is a triple $(N, E, Edges)$ with:

1. N is a set of nodes (or: states) structured in a tree
2. E is a set of events
    - pseudo-event $after(d)$ denotes a delay of $d \in \mathbb{R}_{\geqslant 0}$ time units
    - $\perp \notin E$ stands for "no event available"
3. $Edges$ is a set of (hyper-) edges, defined later on.

Statecharts

Definition (Statecharts)

A statechart SC is a triple (N, E, Edges) with:

1. N is a set of nodes (or: states) structured in a tree
2. E is a set of events
   - pseudo-event $\mathit{after}(d)$ denotes a delay of $d \in \mathbb{R}_{\geqslant 0}$ time units
   - $\perp \notin E$ stands for "no event available"
3. Edges is a set of (hyper-) edges, defined later on.

Definition (System)

A system is described by a finite collection of statecharts $(SC_1, \ldots, SC_k)$.

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

Tree structure

Function *children*

Nodes obey a tree structure defined by function $children : N \to 2^N$ where $x \in children(y)$ means that x is a child of y, or equivalently, y is the parent of x.

Partial order $\trianglelefteq$

The partial order $\trianglelefteq \subseteq N \times N$ is defined by:

- $\forall x \in N.\, x \trianglelefteq x$
- $\forall x, y \in N.\, x \trianglelefteq y$ if $x \in children(y)$
- $\forall x, y, z \in N.\, x \trianglelefteq y \wedge y \trianglelefteq z \Rightarrow x \trianglelefteq z$

$x \trianglelefteq y$ means that x is a descendant of y, or equivalently, y is an ancestor of x. If $x \trianglelefteq y$ or $y \trianglelefteq x$, nodes x and y are ancestrally related.

Root node

There is a unique root with no ancestors, and $\forall x \in N.\, x \trianglelefteq \text{root}$.

Functions on nodes

The type of nodes

Nodes are typed, $type(x) \in \{ \text{BASIC}, \text{AND}, \text{OR} \}$ such that for $x \in N$:

- $type(\text{root}) = \text{OR}$
- $type(x) = \text{BASIC}$ iff $children(x) = \varnothing$, i.e., x is a leaf
- $type(x) = \text{AND}$ implies $(\forall y \in children(x).\ type(y) = \text{OR})$

Default nodes

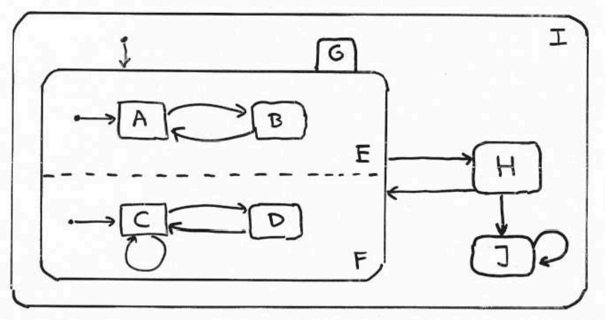
$default : N \to N$ is a partial function on domain $\{ x \in N \mid type(x) = \text{OR} \}$ such that

$$default(x) = y \quad \text{implies} \quad y \in children(x).$$

The function *default* assigns to each OR-node x one of its children as default node that becomes active once x becomes active.

Example

Example statechart



Edges

Definition (Edges)

An edge is a quintuple (X, e, g, A, Y), denoted $X \xrightarrow{e[g]/A} Y$ with:

- $X \subseteq N$ is a set of source nodes with $X \neq \varnothing$
- $e \in E \cup \{ \perp \}$ is the trigger event
- $A \subseteq Act$ is a set of actions
  - such as $v :=$ expr or local variable v and expression expr
  - or *send* $j.e$, i.e., send event e to statechart SC_j
- Guard g is a Boolean expression over all variables in $(SC_1, \ldots, SC_k)$
- $Y \subseteq N$ is a set of target nodes with $Y \neq \varnothing$

Edges

Definition (Edges)

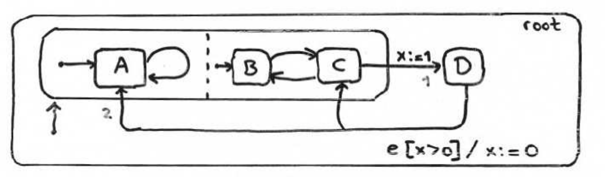
An edge is a quintuple (X, e, g, A, Y), denoted $X \xrightarrow{e[g]/A} Y$ with:

- $X \subseteq N$ is a set of source nodes with $X \neq \varnothing$
- $e \in E \cup \{ \perp \}$ is the trigger event
- $A \subseteq Act$ is a set of actions
  - such as $v :=$ expr or local variable v and expression expr
  - or *send* $j.e$, i.e., send event e to statechart SC_j
- Guard g is a Boolean expression over all variables in $(SC_1, \ldots, SC_k)$
- $Y \subseteq N$ is a set of target nodes with $Y \neq \varnothing$

The sets X and Y may contain nodes at different depth in the node tree.

Example (1)

Example statechart

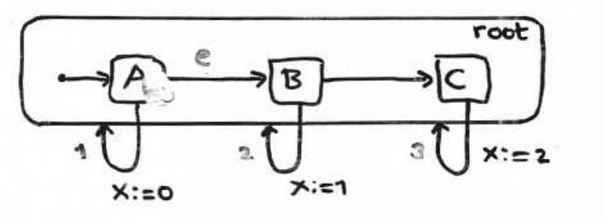


$$\text{edge 1: } \{ C \} \xrightarrow{\bot[true]/\{ x:=1 \}} \{ D \}$$

$$\text{edge 2: } \{ D \} \xrightarrow{e[x>0]/\{ x:=0 \}} \{ A, C \}$$

Example (2)

Example statechart

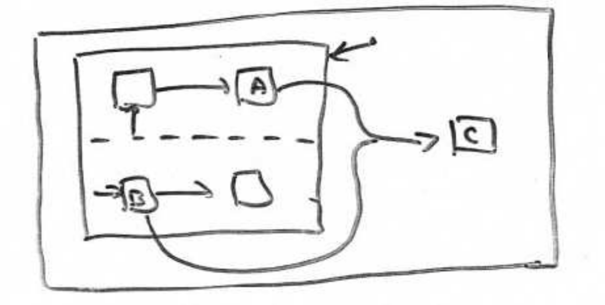


$$\text{edge 1: } \{ A \} \xrightarrow{e[true]/\varnothing} \{ B \}$$

$$\text{edge 2: } \{ B \} \xrightarrow{\bot[true]/\{ x{:=}1 \}} \{ \text{root} \}$$

Example (3)

Example statechart



$$\text{edge} : \{\, A, B \,\} \overset{\cdots}{\longrightarrow} \{\, C \,\}$$