2.2 State Machines
Model-driven Development with State Machines

- Statecharts / State Machines
  - UML 1.3: State Machines (2001)
  - UML 2.1: State Machines (2007)
  - Other definitions (e.g. Rhapsody, also by Harel) with slightly different syntax and semantics

- Why State Machines (in this lecture)?
  - Can describe behavior of a (part of a) system
  - Description is model-based
  - Modeling of states and concurrency (Alternatives: Petri nets, SDL, Z, Esterel)
Agenda

- Repetition: Automata and State Machines
- Advanced concepts for State Machines
- Conflicting transitions
- Comparison of different semantics
  - Stateflow
  - UML
  - Sequential Statecharts
Recapitulation: Automata

- Formal definition
  - 5-tuple $(S, \Sigma, \delta, q_0, F)$
    - $S$: set of states
    - $\Sigma$: input alphabet
    - $\delta$: transition function
    - $q_0$: start state
    - $F$: set of accept states

- Automaton models states of a system
- System component has finite set of internal states (called “configuration”)
- Example: vending machine from lecture “Modellierung”
- Moore-Automaton: output/actions in states
- Mealey-Automaton: output/actions at transitions
Recapitulation: State Machines

- Similar to automata
- Combination of Moore- and Mealy-Automata
- Describe the life-cycle of an object
- Notion of “Signals” \(\Rightarrow\) Communication between State Machines

UML Elements

- StateMachine
- SimpleState
- FinalState
- CompositeState
- Regions, Submachine,
- Pseudostates:
  - Junction, Choice, Fork, Join, ShallowHistory, Initial
- ConnectionPointReference
- ChangeEvent, TimeEvent, Signal
Transitions: Events

- Events trigger transitions, i.e. they cause a state change

- An event is anything that can \textit{happen} in a system
  - Signal trigger: Asynchronous triggering of a transition
  - Call trigger: Invocation of an operation
  - Time trigger: Deadline by which a transition is triggered
  - Change trigger: Transition is triggered when an attribute changes

- State Machines can communicate via signals, i.e. a signal event by one State Machine can trigger a transition in another State Machine
Transitions: Parameters and conditions

- Signal Trigger Events can have parameters
- Transition may only fire if condition is fulfilled
- All elements are optional (⇒ ε transitions for automata)

```
MoneyReceived(newAmount)
[currentAmount + newAmount = price]
/ currentAmount += newAmount
```

**Signal Trigger Event**

**Condition**

**Parameter**

**Action**

- Show price
- Show „OK“
Transitions: Compound Transitions

- Model alternatives in a well arranged way

Normal transitions

- CoffeeButtonPressed()
  - cupInPlace
  - Make Coffee

- EspressoButtonPressed()
  - cupInPlace
  - Make Espresso

- ChocolateButtonPressed()
  - cupInPlace
  - Make Hot Chocolate

Compound transitions with dynamic choice

- CoffeeButtonPressed()
- EspressoButtonPressed()
- ChocolateButtonPressed()

- Show „OK“
- [cupInPlace]
  - Make Coffee
  - Make Espresso
  - Make Hot Chocolate

- Show „OK“
- [cupInPlace]
Transitions: Compound Transition

- Junction: Chain Transitions, No Semantics
- Choice: Dynamic Conditional Branch

**Junction**

- Transition with conditions:
  - \( x = 1 \) and \( x = 3 \)
  - Transition to `Show „OK“`
  - Transition to `Make Coffee`
  - Transition to `Make Espresso`
  - Transition to `Make Hot Chocolate`

**Choice**

- Transition with conditions:
  - \( x = 1 \) and \( x = 3 \)
  - Transition to `Show „OK“`
  - Transition to `Make Coffee`
  - Transition to `Make Espresso`
  - Transition to `Make Hot Chocolate`
States: Actions and activities in states

- Actions (in states)
  - Entry and exit actions
  - Can change attributes (and thereby redefine the state!)
  - May not be interrupted (always run to completion)

- Activities
  - Marked with “Do”
  - Run while State Machine is in corresponding state
  - May be interrupted, hence cannot change the state of an object

```
Make Coffee
Entry / BoilWater()
Do / ShowWait()
Exit / RefillWater()
```
Hierarchical States: Composite States

- Sometimes events must be responded to equally, regardless of the current state
Hierarchical States: Simple composite state

- Reduce visual complexity by using a hierarchy of states
- Only one simple state is active at a given time (hence Composite)

![State Machine Diagram]

- System running
- Composite state
- Basic states
Hierarchical States: Simple composite state

- Reduce visual complexity by using a hierarchy of states
- Only one simple state is active at a given time (hence Composite)

![Diagram showing hierarchical states with components System running, Maintenance, Pause, and opened machine.]
Hierarchical States: Orthogonal Composite State

- Model concurrency with “regions” (= parallel states)
- One basic state in each region is active

![State Machine Diagram]

- System running
- Show price
- Show "OK"
- Make Coffee
- Make Espresso
- Make Hot Chocolate
- Refresh Display
Transitions: Fork and Join

- **fork** vertices serve to split an incoming transition into two or more transitions terminating on orthogonal target vertices.
- **join** vertices serve to merge several transitions emanating from source vertices in different orthogonal regions.

Diagram:

- Setup
- B1
- A1
- A2
- B2
- Cleanup
- Process
Hierarchical States: Shallow History

- Composite state with “memory”
- If a composite state is re-entered, the last active basic state directly below the composite state is activated
- Composite state can still be entered “normally” (bypassing history)
Hierarchical States: Deep History

- If an Composite state is re-entered, the last active basic state *at every nested level* is activated.
Entry and Exit Points

- **An entry point pseudostate**
  - Entry point of a state machine or composite state
  - At most a single transition to a vertex within the same region.

- **An exit point pseudostate**
  - Exit point of a state machine or composite state
  - Implies the exit of this composite state or submachine state
  - Triggering of the transition that has this exit point as source
Submachine State

- Reuse of state machines
- Embed a state machine into another
- Inner state machine can terminate normally, abort or be interrupted
- ExitPoints, EntryPoints
Conflicting transitions in Orthogonal Composite State

- Different semantics for execution of orthogonal states
  - Microstep Semantics (e.g. in Stateflow)
  - Superstep Semantics (e.g. in UML 2.0)
  - Sequential Semantics (proposed by A. Zündorf)
Both transitions are taken synchronously and consume \( e_1 \).

No order for evaluation of actions defined!

Result not uniquely determined (race condition)
Microstep

Both transitions are taken synchronously and consume $e_1$
No order for evaluation of events defined! Queue, Stack, Priorized
Result not uniquely determined (race condition)

Alternative 2

$$y := 2*x$$
before
$$x := y + 1$$

<table>
<thead>
<tr>
<th>Active states</th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B, G</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Each parallel state is executed on his own as far as possible
Result depends on how local variables are merged
Non-determinism introduced by guards

\[
\begin{align*}
E & \text{D} \text{C} \\
B & \text{F} \\
\text{G} & \text{H} \text{I} \\
& \text{e1} / \text{e2} \\
x & := y + 1; \\
& \text{e1} / \text{e3} \\
y & := 2 \times x; \\
& \text{e5} [x \leq 0] \\
& \text{e5} [x \geq 0]
\end{align*}
\]
Sequential Semantics

- Explicit use of priorities to remove non-determinism
- In this example: higher value = higher priority
- Define if high-level transitions have higher or lower priority
Problems with Semantics

- Non-determinism
  - If there are several possibilities for the execution of a step, the result depends on the implementation.

- Lifetime of events
  - According to Harel\(^1\) Events live for one step of the system, i.e. each State Machine has a chance to use an event that has been produced in the previous step. After that the events are discarded.

- Publishing of events in Superstep semantics
  - In Superstep semantics it depends on the implementation if a State Machine published all self-produced events after one superstep or if only the events of the last microstep are published.

---

1 Harel, Naamad: The STATEMATE Semantics of Statecharts, ACM Transactions of Software Engineering, 1996
Overview: Semantics

- Microstep semantics:
  - Changes become immediately visible (i.e. in the same step)
  - Can lead to non-determinism regarding assignments and state changes
  - Concrete semantics depend on implementation

- Superstep semantics:
  - Run-to-completion principle
  - Determinism regarding state changes, but still problems with assignments
  - Concrete semantics depend on implementation

- Sequential semantics:
  - Use explicit priorities to remove non-determinism
  - Useful for code generation
Summary

- State Machines are better suited to describe complex behavior than Deterministic Finite Automaton (DFA)
- State Machines are not more powerful in a computational sense (= can be mapped to DFA)
- Advanced notation elements
  - Transitions: events, conditions, actions, compound transitions
  - States: basic states, Composite states, Orthogonal Composite states, history
- Problems
  - Conflicting transitions
  - Execution semantics