Software Design, Modelling and Analysis in UML

Lecture 11: Core State Machines I

2013-12-04

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

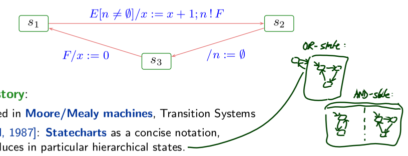
- Core State Machines
- UML State Machine syntax
- State machines belong to classes.

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: Signal, Event, Ether, Transformer, Step, RTC.
- **Content:**
  - UML Core State Machines (first half)
  - Ether, System Configuration, Transformer
  - Run-to-completion Step
  - Putting It All Together

UML State Machines

UML State Machines



Brief History:

- Rooted in **Moore/Mealy machines**, Transition Systems
- [Harel, 1987]: **Statecharts** as a concise notation, introduces in particular hierarchical states.
- Manifest in tool **Statemate** [Harel et al., 1990] (simulation, code-generation); nowadays also in **Matlab/Simulink**, etc.
- From UML 1.x on: **State Machines** (in State Chart Diagram) (not the official name, but understood: UML-Statecharts)
- Late 1990's: tool **Rhapsody** with code-generation for state machines.

Note: there is a common core, but each dialect interprets some constructs subtly different [Crane and Dingel, 2007]. *(Would be too easy otherwise. . .)*

Roadmap: Chronologically

(i) What do we (have to) cover?
UML State Machine Diagrams **Syntax**.

(ii) Def.: Signature with **signals**.

(iii) Def.: **Core state machine**.

(iv) Map UML State Machine Diagrams to core state machines.

Semantics:
The Basic Causality Model

(v) Def.: **Ether** (aka. event pool)

(vi) Def.: **System configuration**.

(vii) Def.: **Event**.

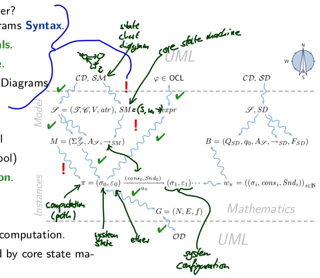
(viii) Def.: **Transformer**.

(ix) Def.: **Transition system**, computation.

(x) Transition relation induced by core state machine.

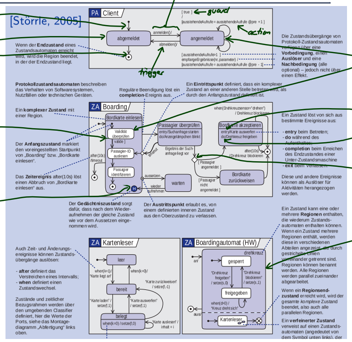
(xi) Def.: **step**, **run-to-completion step**.

(xii) Later: Hierarchical state machines.

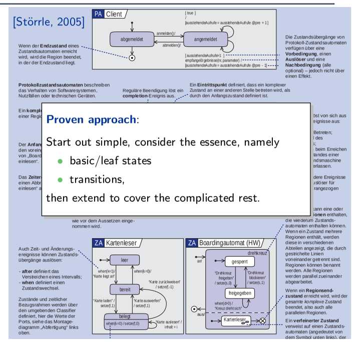


UML State Machines: Syntax

UML State-Machines: What do we have to cover?



UML State-Machines: What do we have to cover?



Proven approach:

Start out simple, consider the essence, namely

- basic/leaf states
- transitions,

then extend to cover the complicated rest.

Signature With Signals

Definition. A tuple

$$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E}),\ \mathscr{E} \subseteq \mathscr{C} \text{ a set of signals,}$$

is called signature (with signals) if and only if

$$(\mathscr{T}, \mathscr{C} \setminus \mathscr{E}, V, atr)$$

is a signature (as before).

Note: Thus conceptually, **a signal is a class** and can have attributes of plain type and associations.

Core State Machine

Definition.
A core state machine over signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E})$ is a tuple

$$\mathcal{SM} = (S, s_0, \rightarrow)$$

where

- S is a non-empty, finite set of **(basic) states**,
- $s_0 \in S$ is an **initial state**,
- and

$$\rightarrow \subseteq S \times \underbrace{(\mathscr{E} \mathbin{\dot\cup} \{\_\})}_{\text{trigger}} \times \underbrace{Expr_{\mathscr{S}}}_{\text{guard}} \times \underbrace{Act_{\mathscr{S}}}_{\text{action}} \times S$$

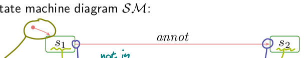
is a labelled transition relation.

We assume a set $Expr_{\mathscr{S}}$ of boolean expressions (may be OCL, may be something else) and a set $Act_{\mathscr{S}}$ of **actions** over $\mathscr{S}$.

source state; set of signals; destination state; "$\dot\cup$" disjoint union, $\_ \notin \mathscr{E}$

From UML to Core State Machines: By Example

UML state machine diagram $\mathcal{SM}$:



$annot ::= [\ \langle event\rangle[\text{','}\ \langle event\rangle]^*\ \ [\text{'['}\ \langle guard\rangle\ \text{']'}]\ \ [\text{'/'}\ \langle action\rangle]\ ]$

not in lecture; trigger; guard; action

with

- $event \in \mathscr{E}$,
- $guard \in Expr_{\mathscr{S}}$ (default: *true*, assumed to be in $Expr_{\mathscr{S}}$)
- $action \in Act_{\mathscr{S}}$ (default: `skip`, assumed to be in $Act_{\mathscr{S}}$)

maps to

$$\mathcal{SM}(\mathcal{SM}) = \big(\underbrace{\{s_1, s_2\}}_{S}, \underbrace{s_1}_{s_0}, \underbrace{(s_1, event, guard, action, s_2)}_{\rightarrow}\big)$$

Annotations and Defaults in the Standard

Reconsider the syntax of transition annotations:

$annot ::= [\ \ \langle event\rangle[\text{','}\ \langle event\rangle]^*\ \ [\text{'['}\ \langle guard\rangle\ \text{']'}]\ \ [\text{'/'}\ \langle action\rangle]\ \ ]$

and let's play a bit with the defaults:

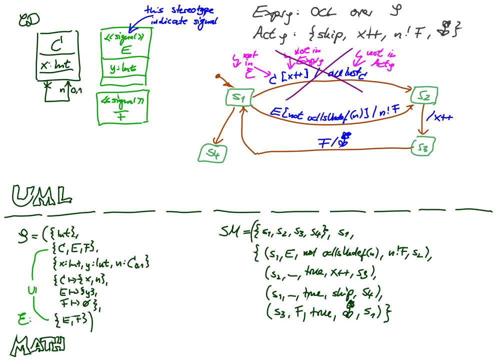
(the empty annotation)	$\leadsto$	$\_$, true, skip
$/$	$\leadsto$	—
$E\ /$	$\leadsto$	E, true, skip
$/\ act$	$\leadsto$	$\_$, true, act
$E\ /\ act$	$\leadsto$	E, true, act
[expr]	$\leadsto$	$\_$, expr, skip

trigger, guard, action; $E \in \mathscr{E}$, $act \in Act_{\mathscr{S}}$, $expr \in Expr_{\mathscr{S}}$

In the standard, the syntax is even more elaborate:

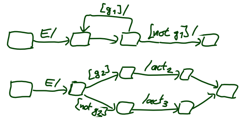
- $E(v)$ — when consuming E in object u, attribute v of u is assigned the corresponding attribute of E.
- $E(v : \tau)$ — similar, but v is a local variable, scope is the transition

in Rhapsody: $E(x)$ / ... $\leadsto E / x := params\text{->}x$ — special keyword to access event attributes



What is that useful for?

- **No Event:**



- **No annotation:**

see above

State-Machines belong to Classes

- In the following, we assume that a UML models consists of a set $\mathscr{CD}$ of class diagrams and a set $\mathscr{SM}$ of **state chart diagrams** (each comprising one **state machines** $\mathcal{SM}$).
- Furthermore, we assume that **each state machine** $\mathcal{SM} \in \mathscr{SM}$ is **associated** with **a class** $C_{\mathcal{SM}} \in \mathscr{C}(\mathscr{S})$.
- For simplicity, we even assume a bijection, i.e. we assume that each class $C \in \mathscr{C}(\mathscr{S})$ has a state machine $\mathcal{SM}_C$ and that its class $C_{\mathcal{SM}_C}$ is C.

  If not explicitly given, then this one:

  $$\mathcal{SM}_0 := (\{s_0\}, s_0, \emptyset).$$

  We'll see later that, semantically, this choice does no harm.

- **Intuition 1**: $\mathcal{SM}_C$ describes the behaviour of **the instances** of class C.

  Intuition 2: Each instance of C executes $\mathcal{SM}_C$ with own "program counter".

Note: we don't consider **multiple state machines** per class.
(Because later (when we have AND-states) we'll see that this case can be viewed as a single state machine with as many AND-states.)

References

References

[Crane and Dingel, 2007] Crane, M. L. and Dingel, J. (2007). UML vs. classical vs. rhapsody statecharts: not all models are created equal. *Software and Systems Modeling*, 6(4):415–435.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

[Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.

[Harel et al., 1990] Harel, D., Lachover, H., et al. (1990). Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

[Störrle, 2005] Störrle, H. (2005). *UML 2 für Studenten*. Pearson Studium.