# *Software Design, Modelling and Analysis in UML*

## *Lecture 15: Hierarchical State Machines I*

*2013-01-08*

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

Albert-Ludwigs-Universität Freiburg, Germany

# Contents & Goals

**Last Lecture:**

- RTC-Rules: Discard, Dispatch, Commence.
- Step, RTC, Divergence
- Putting It All Together – ODs for initial state
- Rhapsody Demo

$(\sigma, \varepsilon) \xrightarrow[v]{sys.cev} (\sigma', \varepsilon')$

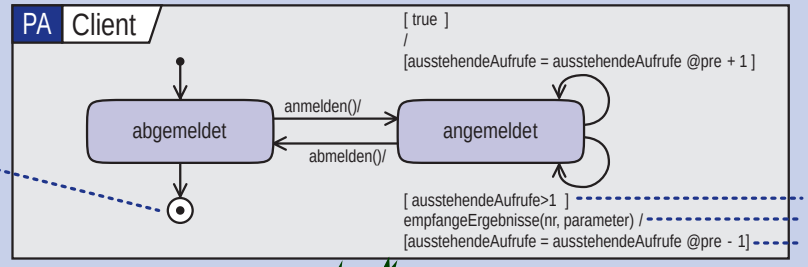
**This Lecture:**

- **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: initial state.
  - What does this **hierarchical** State Machine mean? What **may happen** if I inject this event?
  - What is: AND-State, OR-State, pseudo-state, entry/exit/do, final state, ...
- **Content:**
  - Hierarchical State Machines Syntax

# *Hierarchical State Machines*

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

[Störrle, 2005]



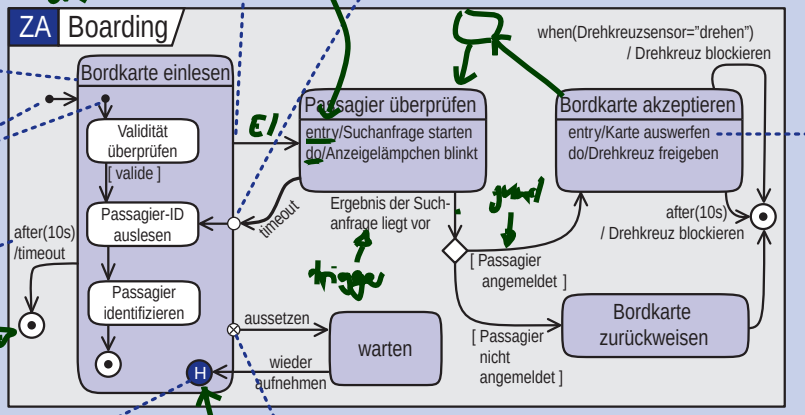

Wenn der **Endzustand** eines Zustandsautomaten erreicht wird, wird die Region beendet, in der der Endzustand liegt.

Die Zustandsübergänge von Protokoll-Zustandsautomaten verfügen über eine **Vorbedingung**, einen **Auslöser** und eine **Nachbedingung** (alle optional) – jedoch nicht über einen Effekt.

**Protokollzustandsautomaten** beschreiben das Verhalten von Softwaresystemen, Nutzfällen oder technischen Geräten.

Reguläre Beendigung löst ein **completion**-Ereignis aus.

Ein **Eintrittspunkt** definiert, dass ein komplexer Zustand an einer anderen Stelle betreten wird, als durch den Anfangszustand definiert ist.




Ein **komplexer Zustand** mit einer Region.

Der **Anfangszustand** markiert den voreingestellten Startpunkt von „Boarding“ bzw. „Bordkarte einlesen“.

Das **Zeitereignis** after(10s) löst einen Abbruch von „Bordkarte einlesen“ aus.

Ein Zustand löst von sich aus bestimmte Ereignisse aus:

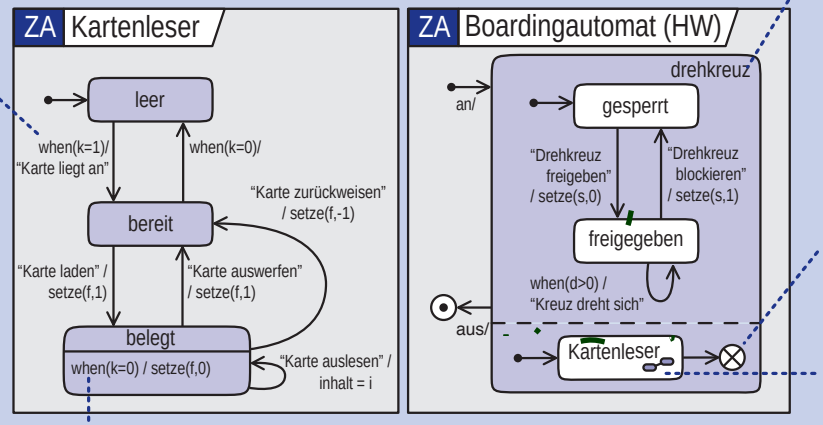
- **entry** beim Betreten;
- **do** während des Aufenthaltes;
- **completion** beim Erreichen des Endzustandes einer Unter-Zustandsmaschine
- **exit** beim Verlassen.

Diese und andere Ereignisse können als Auslöser für Aktivitäten herangezogen werden.

Der **Gedächtniszustand** sorgt dafür, dass nach dem Wiederaufnehmen der gleiche Zustand wie vor dem Aussetzen eingenommen wird.

Der **Austrittspunkt** erlaubt es, von einem definierten inneren Zustand aus den Oberzustand zu verlassen.

Ein Zustand kann eine oder mehrere **Regionen** enthalten, die wiederum Zustandsautomaten enthalten können. Wenn ein Zustand mehrere Regionen enthält, werden diese in verschiedenen Abteilen angezeigt, die durch gestrichelte Linien voneinander getrennt sind. Regionen können benannt werden. Alle Regionen werden parallel zueinander abgearbeitet.




Auch Zeit- und Änderungsereignisse können Zustandsübergänge auslösen:

- **after** definiert das Verstreichen eines Intervalls;
- **when** definiert einen Zustandswechsel.

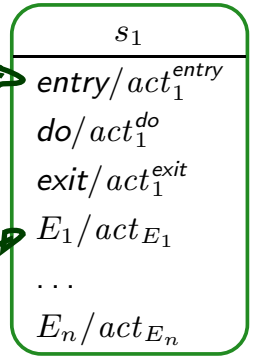
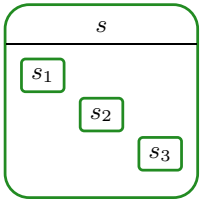
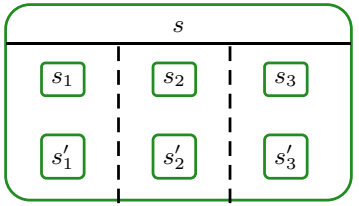
Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montagediagramm „Abfertigung“ links oben.

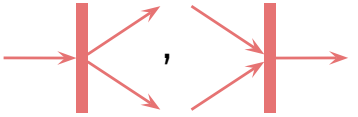
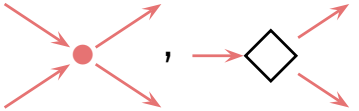
Wenn ein **Regionsendzustand** erreicht wird, wird der gesamte *komplexe* Zustand beendet, also auch alle parallelen Regionen.

Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der

# The Full Story

UML distinguishes the following **kinds of states**:

| | example |
|---|---|
| **simple state** |   |
| **final state** |  |
| **composite state** | |
| OR |   |
| AND |   |

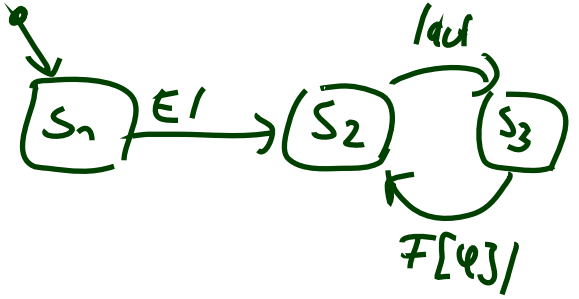
| | example |
|---|---|
| **pseudo-state** | |
| initial |  |
| (shallow) history |   |
| deep history |   |
| fork/join |  |
| junction, choice |  |
| entry point |  |
| exit point |  |
| terminate |  |
| **submachine state** |   |

# Representing All Kinds of States

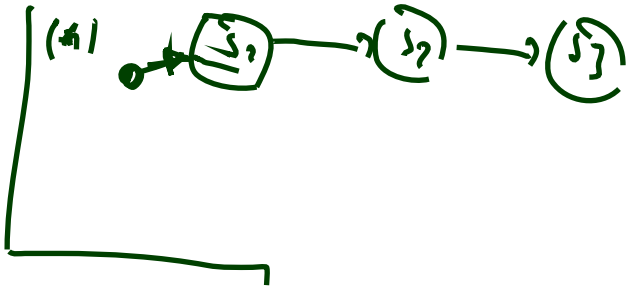
- **Until now**:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \rightarrow \subseteq S \times (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}} \times S$$

init. state · source · destination · set of states · transition · label



# Representing All Kinds of States

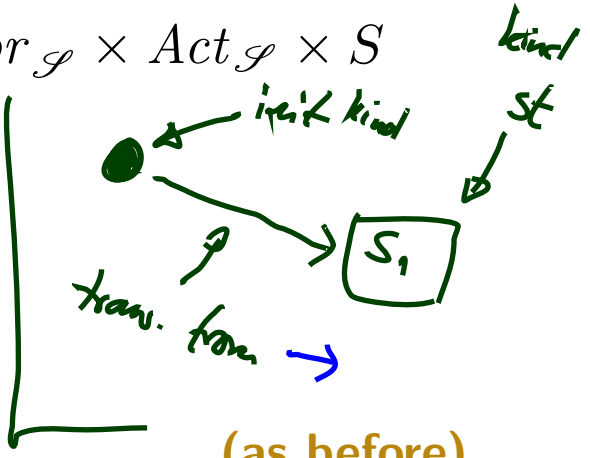


- **Until now**:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \rightarrow \subseteq S \times (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}} \times S$$

- **From now on**: (**hierarchical**) **state machines**

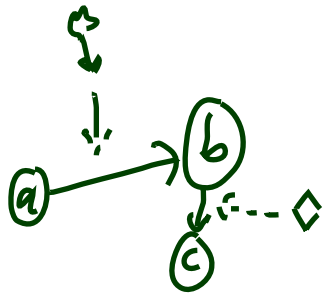
$$(S, kind, region, \rightarrow, \psi, annot)$$



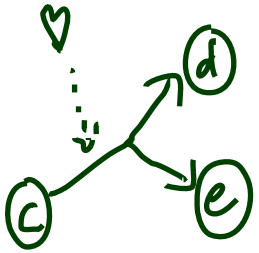
where

- $S \supseteq \{top\}$ is a finite set of states **(as before)**,
- $kind : S \rightarrow \{st, init, fin, shist, dhist, fork, join, junc, choi, ent, exi, term\}$ is a function which labels states with their **kind**, **(new)**
- $region : S \rightarrow 2^{2^S}$ is a function which characterises the **regions** of a state, **(new)**
- $\rightarrow$ is a set of transitions, **(changed)**
- $\psi : (\rightarrow) \rightarrow 2^S \times 2^S$ is an **incidence function**, and **(new)**
- $annot : (\rightarrow) \rightarrow (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}}$ provides an annotation for each transition. **(new)**

(*) ($s_0$ is then redundant — replaced by proper state (!) of kind '*init*'.)




- $(\{a,b,c\}, \{(a,b),(b,c)\})$
- $(\{a,b,c\}, \{\clubsuit, \diamondsuit\}, \{\clubsuit \mapsto (a,b), \diamondsuit \mapsto (b,c)\})$




- $(\{c,d,e\}, \{\heartsuit\}, \heartsuit \mapsto (\{c\},\{d,e\}))$


represent

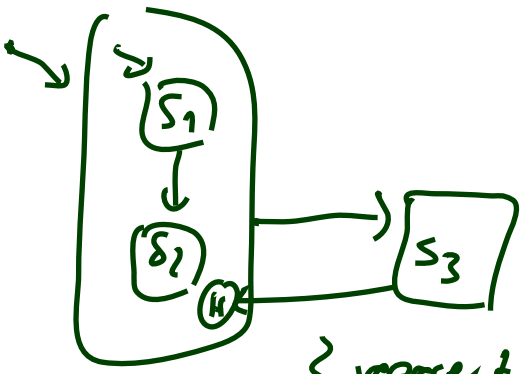
$(S, s_0, \to)$

indices

transition system on $\Sigma_g^{\mathbb{D}} \times \mathcal{E}th$,

with

$(\sigma, \varepsilon) \xrightarrow[v]{\text{send, conn}} (\sigma', \varepsilon')$




represent

$(S, \text{kind}, \text{region}, \to, \Psi, \text{annot})$

indices

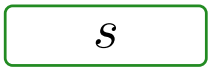
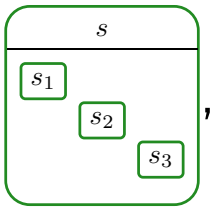
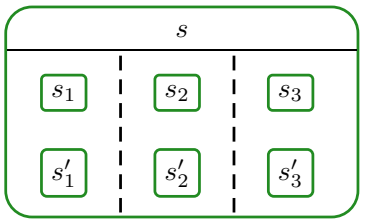
transition system on $\Sigma_g^{\mathbb{D}} \times \mathcal{E}th$

now

later

# *From UML to Hierarchical State Machines: By Example*

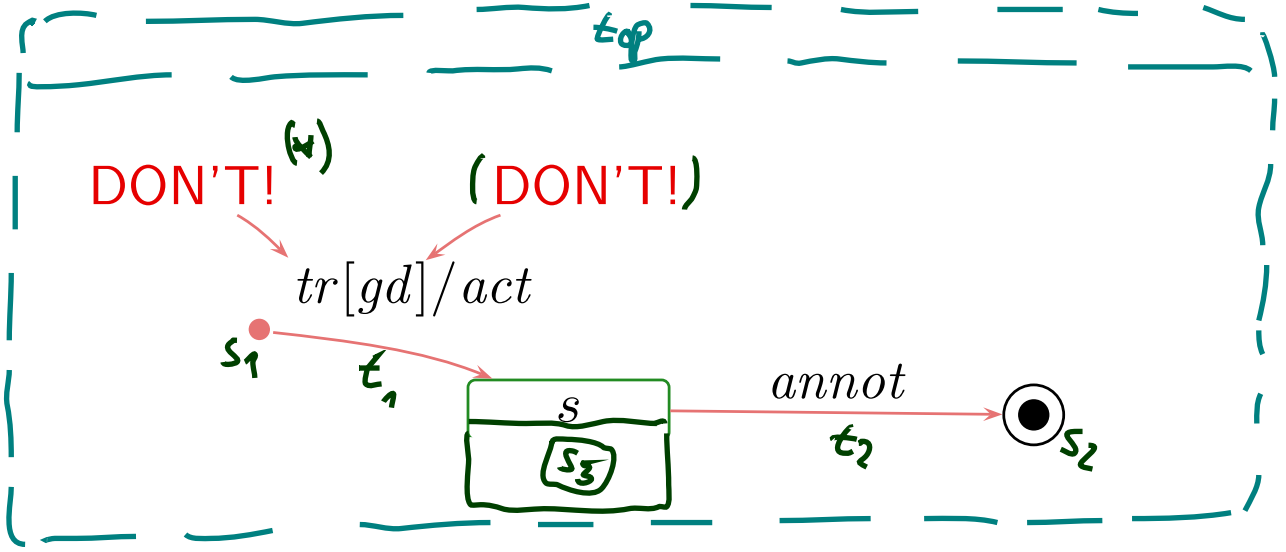
$$(S, kind, region, \rightarrow, \psi, annot)$$

| | example | $\in S$ | $kind$ | $region$ |
|---|---|---|---|---|
| **simple state** (nothing nested) |   | $s$ | st | $\emptyset$ |
| **final state** |  fresh name | $q$ | fin | $\emptyset$ |
| **composite state** | | | | |
| OR |   | $s$ | st | $\{\{s_1, s_2, s_3\}\}$ region |
| AND |   | $s$ | st | $\{\{s_1, s'_1\}, \{s_2, s'_2\}, \{s_3, s'_3\}\}$ |
| **submachine state** | (later) | - | - | - |
| **pseudo-state** |  | $q$ | init, shist, .. | $\emptyset$ |

$(s, kind(s))$ for short

e.g. $(q, fin), (s, st)$

# From UML to Hierarchical State Machines: By Example




... translates to $(S, kind, region, \rightarrow, \psi, annot) =$

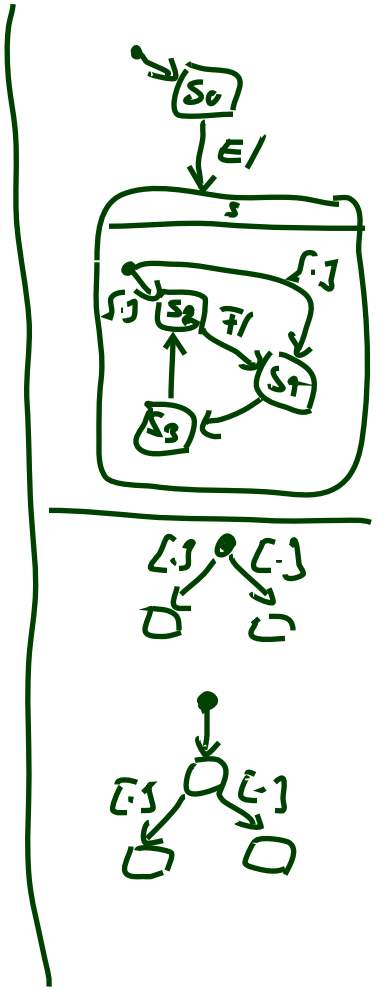
$(\underbrace{\{(top, st), (s, st), (s_1, init), (s_2, fin)\}}_{S,kind}, (s_3, st),$

$\underbrace{\{top \mapsto \{\{s, s_1\ s_2\}\}, s_1 \mapsto \emptyset, s_2 \mapsto \emptyset, s \mapsto \{\{s_3\}\}, s_3 \mapsto \emptyset\}}_{region}$

$\underbrace{\{t_1, t_2\}}_{\rightarrow}, \underbrace{\{t_1 \mapsto (\{s_1\}, \{s\}), t_2 \mapsto (\{s\}, \{s_2\})\}}_{\psi},$

$\underbrace{\{t_1 \mapsto (tr, gd, act), t_2 \mapsto annot\}}_{annot})$

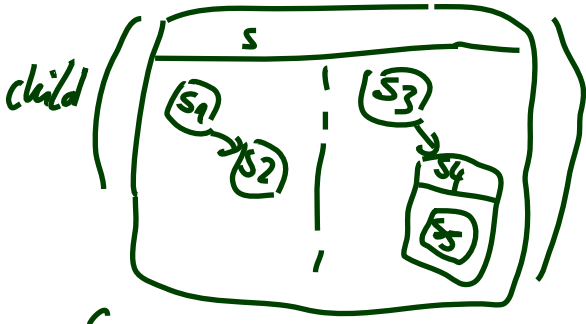
(*) because



# *Well-Formedness: Regions (follows from diagram)*

| | $\in S$ | $kind$ | $region \subseteq 2^S, S_i \subseteq S$ | $child(s) \subseteq S$ |
|---|---|---|---|---|
| **simple state** | $s$ | *st* | $\emptyset$ | $\emptyset$ |
| **final state** | $s$ | *fin* | $\emptyset$ | $\emptyset$ |
| **composite state** | $s$ | *st* | $\{S_1, \ldots, S_n\}, n \geq 1$ | $S_1 \cup \cdots \cup S_n$ |
| **pseudo-state** | $s$ | *init*, ... | $\emptyset$ | $\emptyset$ |
| **implicit top state** | $top$ | *st* | $\{S_1\}$ | $S_1$ |

Def.



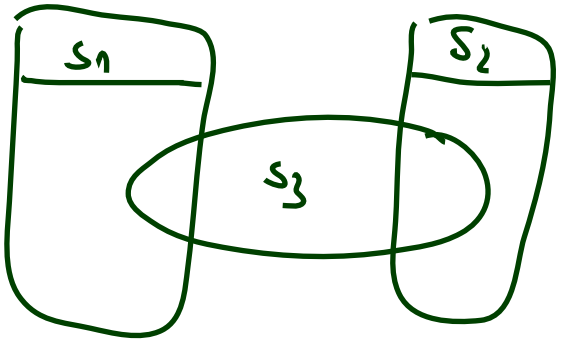

$= \{s_1, s_2, s_3, s_4\}$

- Each state (except for $top$) lies in exactly one region,
- States $s \in S$ with $kind(s) = st$ **may comprise** regions.
  - No region: simple state.
  - One region: OR-state.
  - Two or more regions: AND-state.
- Final and pseudo states **don't comprise** regions.
- The region function induces a **child** function.

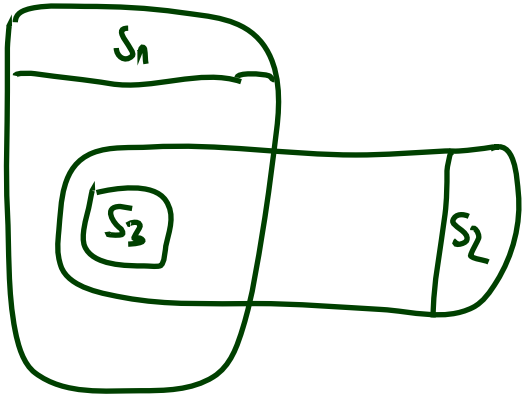
$region(s) = \{\{s_1, s_2\}, \{s_3, s_4\}\}$

Each state (except for top) lies in exactly one region.

Follows from diagrams because we may not draw:

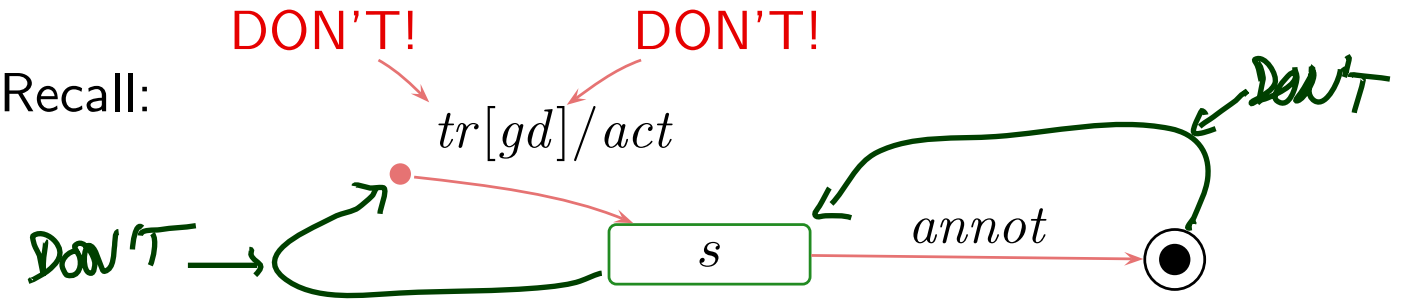



or

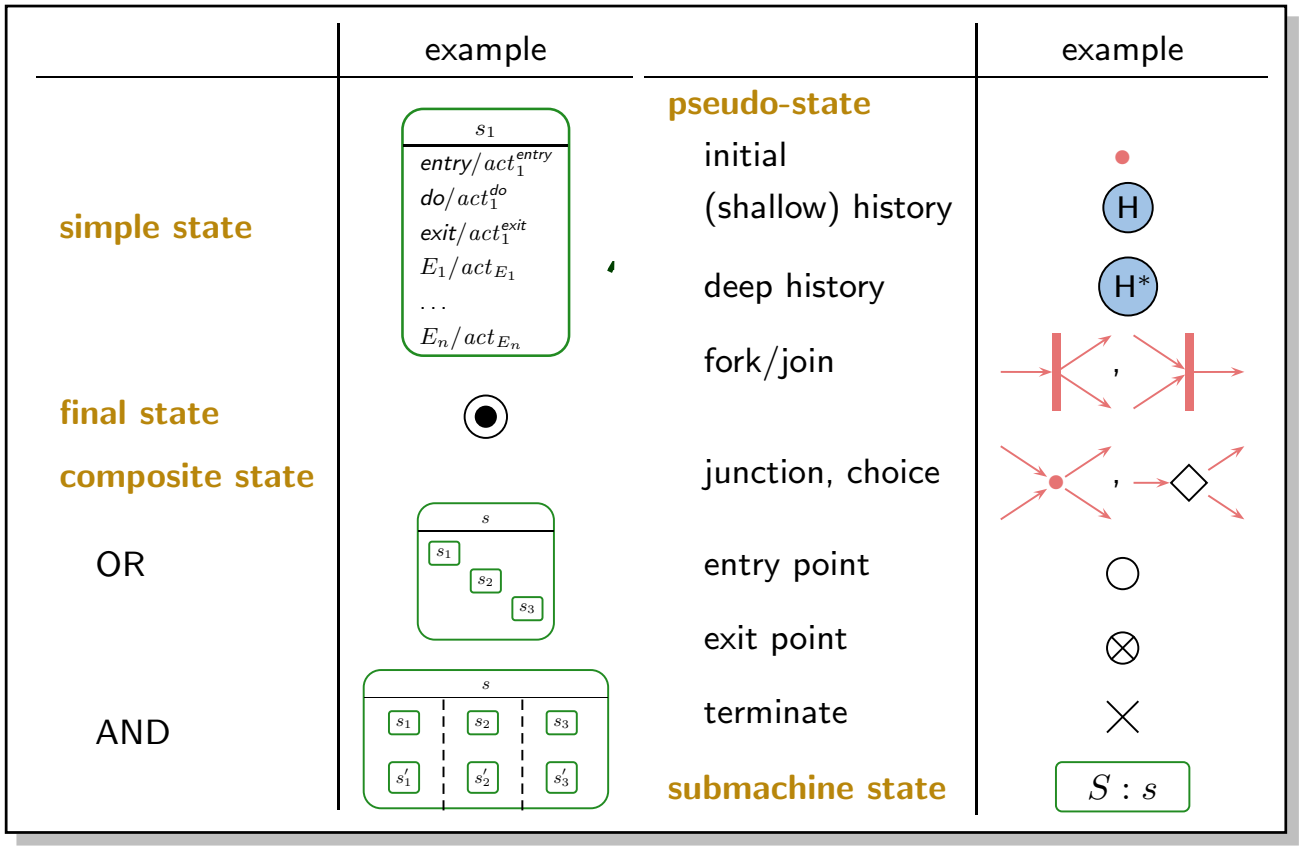


# Well-Formedness: Initial State (requirement on diagram)

- Each non-empty region has a (reasonable) initial state and at least one transition from there, i.e.
  - for each $s \in S$ with $region(s) = \{S_1, \ldots, S_n\}$, $n \geq 1$, for each $1 \leq i \leq n$,
  - there exists exactly one initial pseudo-state $(s_1^i, \mathit{init}) \in S_i$ and at least one transition $t \in \longrightarrow$ with $s_1^i$ as source,
  - and such transition's target $s_2^i$ is in $S_i$, and (**for simplicity!**) $kind(s_2^i) = \mathit{st}$, and $annot(t) = (\_, \mathit{true}, act)$.
- No ingoing transitions to initial states.
- No outgoing transitions from final states.
- Recall:



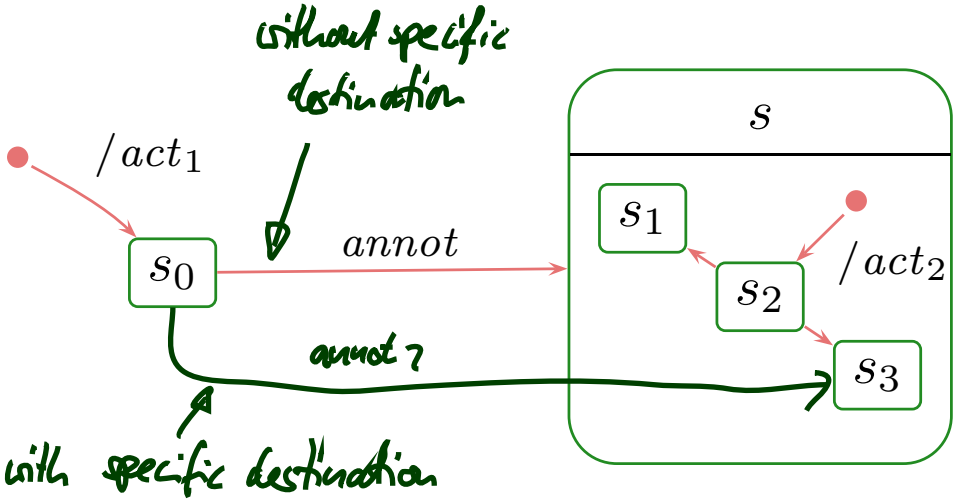
# Plan




- Initial pseudostate, final state.
- Composite states.
- Entry/do/exit actions, internal transitions.
- History and other pseudostates, the rest.

# *Initial Pseudostates and Final States*

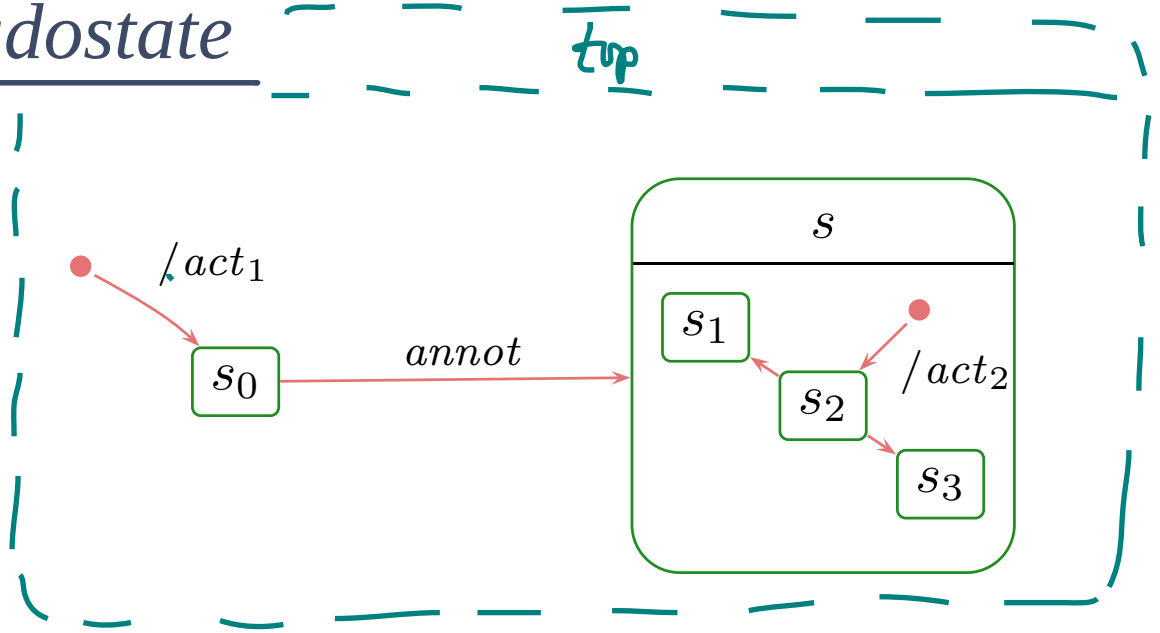
# Initial Pseudostate




**Principle**:

- when entering a region **without** a specific destination state,
- then go to a state which is destination of an initiation transition,
- execute the action of the chosen initiation transitions **between** exit and entry actions. of source and destination (later).

# Initial Pseudostate



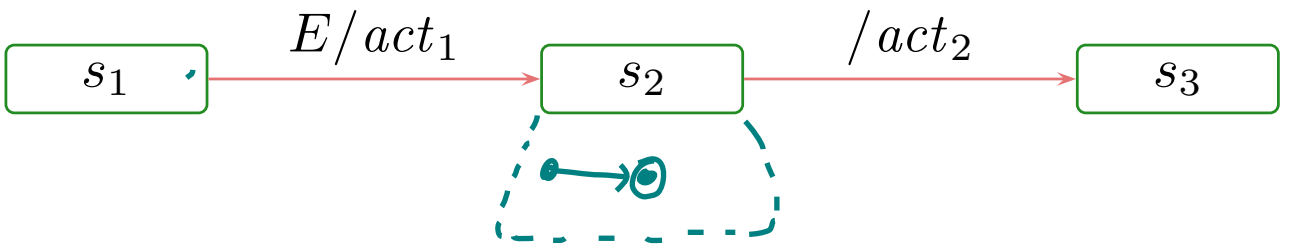

**Principle**:

- when entering a region **without** a specific destination state,
- then go to a state which is destination of an initiation transition,
- execute the action of the chosen initiation transitions **between** exit and entry actions.

**Special case**: the region of $top$.

- If class $C$ has a state-machine, then "create-$C$ transformer" is the concatenation of
  - the transformer of the "constructor" of $C$ (here not introduced explicitly) and
  - a transformer corresponding to one initiation transition of the top region.

# *Towards Final States: Completion of States*




- Transitions without trigger can **conceptionally** be viewed as being sensitive for the "completion event".

- Dispatching (here: $E$) **can then alternatively** be **viewed** as

  (i) fetch event (here: $E$) from the ether,

  (ii) take an enabled transition (here: to $s_2$),

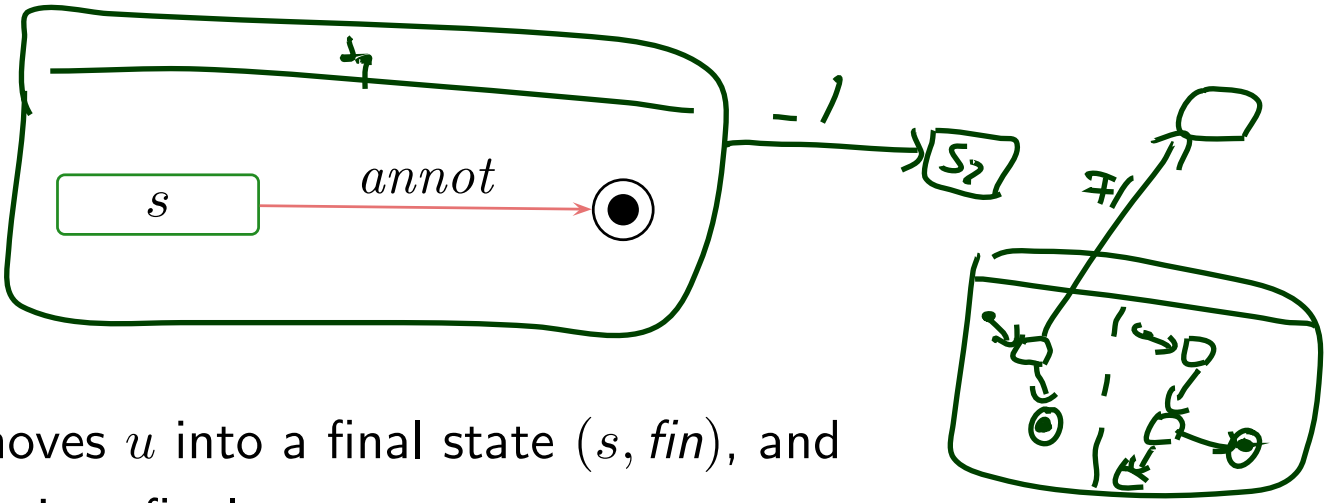
  (iii) remove event from the ether,

  (iv) after having finished entry and do action of current state (here: $s_2$) — the state is then called **completed** —,

  (v) raise a **completion event** — with strict priority over events from ether!

  (vi) if there is a transition enabled which is sensitive for the completion event,
  - then take it (here: $(s_2, s_3)$).
  - otherwise become stable.

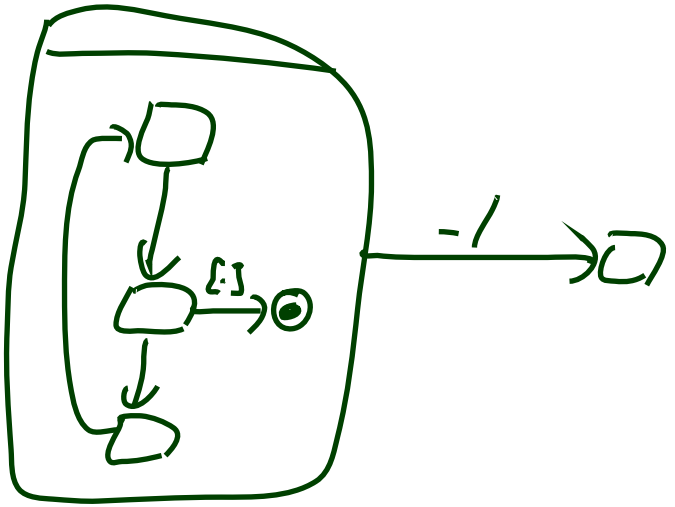
# Final States



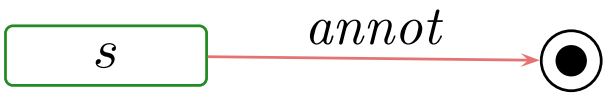

- If
  - a step of object $u$ moves $u$ into a final state $(s, \mathit{fin})$, and
  - all sibling regions are in a final state,

  then (conceptionally) a completion event for the current composite state $s$ is raised.
- If there is a transition of a **parent state** (i.e., inverse of $child$) of $s$ enabled which is sensitive for the completion event,
  - then take that transition,
  - otherwise kill $u$

  $\rightsquigarrow$ adjust $(2.)$ and $(3.)$ in the semantics accordingly



# *Final States*




- If
  - a step of object $u$ moves $u$ into a final state $(s, \mathit{fin})$, and
  - all sibling regions are in a final state,

  then (conceptionally) a completion event for the current composite state $s$ is raised.
- If there is a transition of a **parent state** (i.e., inverse of $child$) of $s$ enabled which is sensitive for the completion event,
  - then take that transition,
  - otherwise kill $u$

  $\rightsquigarrow$ adjust $(2.)$ and $(3.)$ in the semantics accordingly
- **One consequence**: $u$ never survives reaching a state $(s, \mathit{fin})$ with $s \in child(top)$.
- **Now:** in Core State Machines, there is no parent state.
- **Later:** in Hierarchical ones, there may be one.

# *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.

[Damm et al., 2003] Damm, W., Josko, B., Votintseva, A., and Pnueli, A. (2003). A formal semantics for a UML kernel language 1.2. IST/33522/WP 1.1/D1.1.2-Part1, Version 1.2.

[Fecher and Schönborn, 2007] Fecher, H. and Schönborn, J. (2007). UML 2.0 state machines: Complete formal semantics via core state machines. In Brim, L., Haverkort, B. R., Leucker, M., and van de Pol, J., editors, *FMICS/PDMC*, volume 4346 of *LNCS*, pages 244–260. Springer.

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

[Harel and Kugler, 2004] Harel, D. and Kugler, H. (2004). The rhapsody semantics of statecharts. In Ehrig, H., Damm, W., Große-Rhode, M., Reif, W., Schnieder, E., and Westkämper, E., editors, *Integration of Software Specification Techniques for Applications in Engineering*, number 3147 in LNCS, pages 325–354. Springer-Verlag.

[OMG, 2007] OMG (2007). 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.