Software Design, Modelling and Analysis in UML

Lecture 17: Hierarchical State Machines II

2014-01-20

– 17 – 2014-01-20 – main –

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- State Machines and OCL
- Hierarchical State Machines Syntax
- Initial and Final State

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 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:**
  - Composite State Semantics
  - The Rest

– 17 – 2014-01-20 – Sprelim –

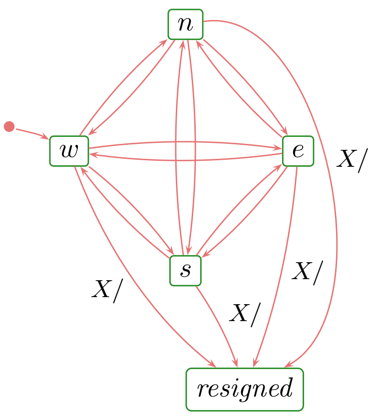
Composite States

(formalisation follows [Damm et al., 2003])

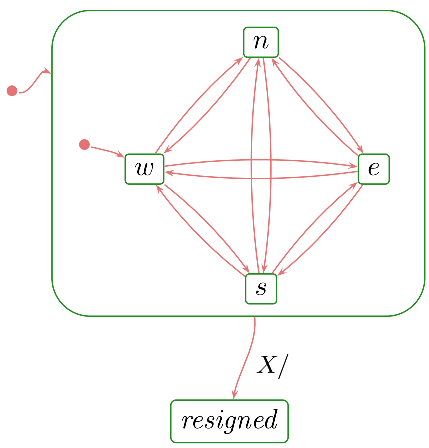
Composite States

- In a sense, composite states are about **abbreviation**, **structuring**, and **avoiding redundancy**.

- Idea: in Tron, for the Player's Statemachine, instead of

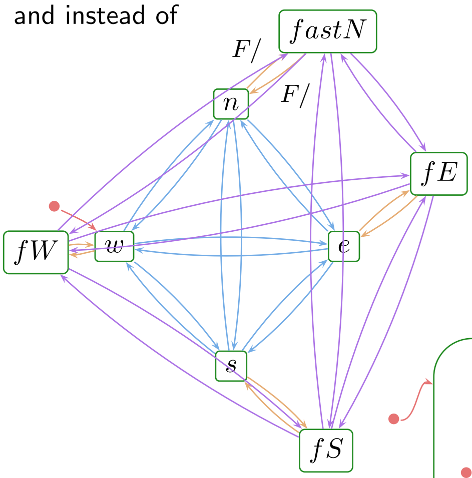


write

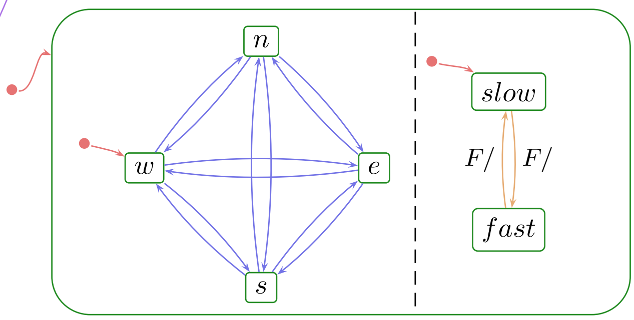


Composite States

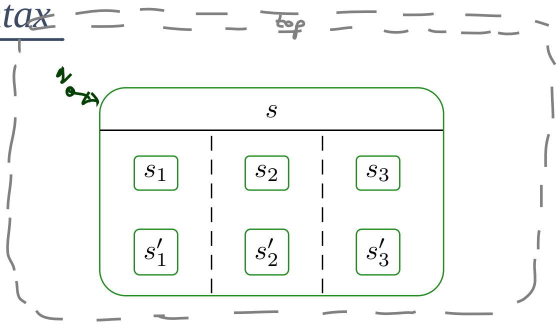
and instead of



write



Recall: Syntax



translates to

(q, init)

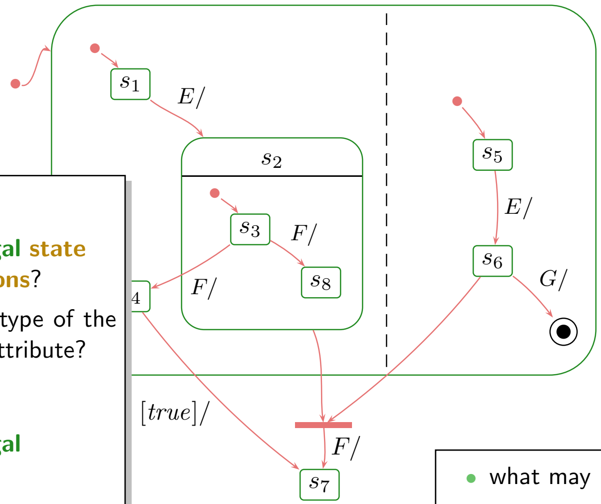
$$(\underbrace{\{(top, st), (s, st), (s_1, st)(s_1', st)(s_2, st)(s_2', st)(s_3, st)(s_3', st)\}}_{S,kind},$$

q

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

$$\to, \psi, annot)$$

Composite States: Blessing or Curse?



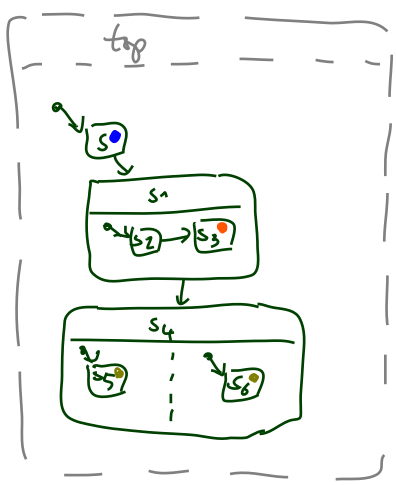
States:

- what are **legal state configurations**?
- what is the type of the implicit st attribute?

Transitions:

- what are **legal** transitions?
- when is a transition enabled?
- what effects do transitions have?

- what may happen on E?
- what may happen on E, F?
- can E, G kill the object?
- ...



OLD: $st : S$ ← set of states

$st = s$

NEW: $st : 2^S$ ← sets of states

$st = \{s, top\}$

$st = \{s_3, s_1, top\}$

$st = \{ \{s_5, s_4, \ldots$ NO

$\{s_6, s_4, \ldots$

$\{s_5, s_6, s_4, top\}$

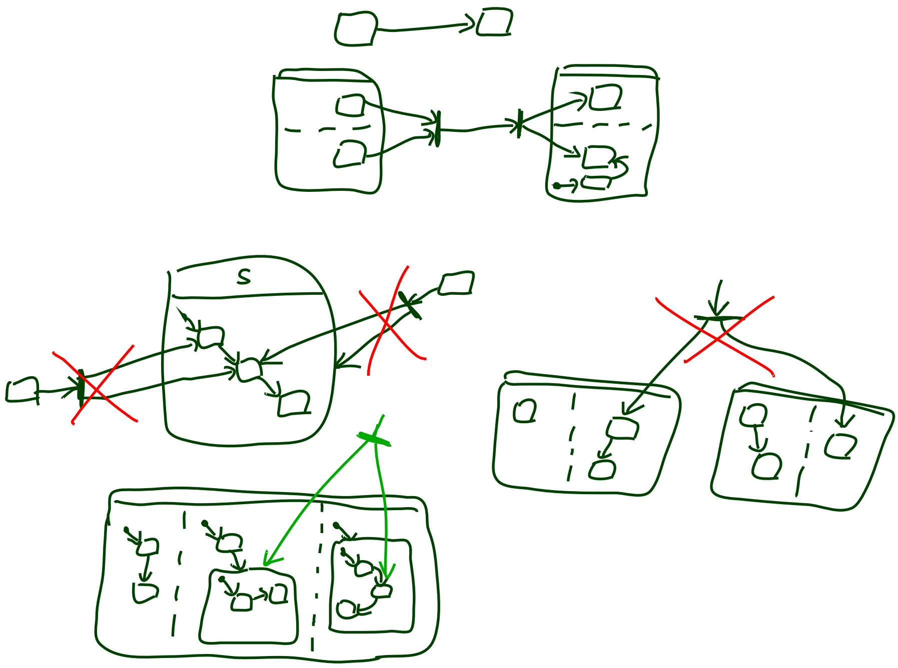
equivalent information: $st = \{s_5, s_6\}$

$st = \{s_2, s_6\}$ INCONSISTENT

State Configuration

- The type of st is from now on **a set of** states, i.e. $st : 2^S$
- A set $S_1 \subseteq S$ is called (**legal**) **state configurations** if and only if
  - $top \in S_1$, and
  - for each state $s \in S_1$, for each non-empty region $\emptyset \neq R \in region(s)$, exactly one (non pseudo-state) child of s (from R) is in S_1, i.e.

$$|\{s_0 \in R \mid kind(s_0) \in \{\mathsf{st}, \mathsf{fin}\}\} \cap S_1| = 1.$$



Syntax: Fork/Join

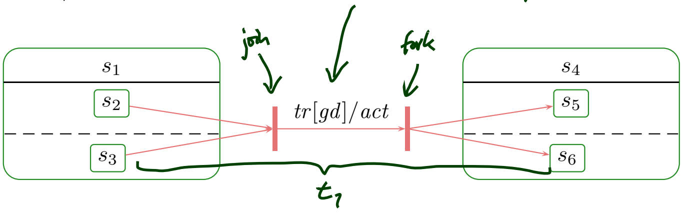
- For brevity, we always consider transitions with (possibly) multiple sources and targets, i.e.

set of src states   set of target or dest. states

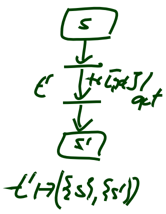
$$\psi : (\rightarrow) \rightarrow (2^S \setminus \emptyset) \times (2^S \setminus \emptyset)$$

- For instance,

assume: one annot. per "tree"



SPECIAL CASE:



translates to

$$(S, kind, region, \underbrace{\{t_1\}}_{\rightarrow}, \underbrace{\{t_1 \mapsto (\{s_2, s_3\}, \{s_5, s_6\})\}}_{\psi}, \underbrace{\{t_1 \mapsto (tr, gd, act)\}}_{annot})$$

- Naming convention: $\psi(t) = (source(t), target(t))$.

A Partial Order on States

The substate- (or **child-**) relation **induces** a **partial order on states**:

- $top \leq s$, for all $s \in S$,
- $s \leq s'$, for all $s' \in child(s)$,
- transitive, reflexive, antisymmetric,
- $s' \leq s$ and $s'' \leq s$ implies $s' \leq s''$ or $s'' \leq s'$.

$$\begin{array}{cc} s' & s'' \\ \wedge\!| & \nearrow\!\! \\ s & \end{array} \Rightarrow \begin{array}{c} s' = s'' \\ \wedge\!| \\ s \end{array} \text{ or } \begin{array}{c} s'' \\ \wedge\!| \\ s' \\ \wedge\!| \\ s \end{array} \text{ or } \begin{array}{c} s' \\ \wedge\!| \\ s'' \\ \wedge\!| \\ s \end{array}$$

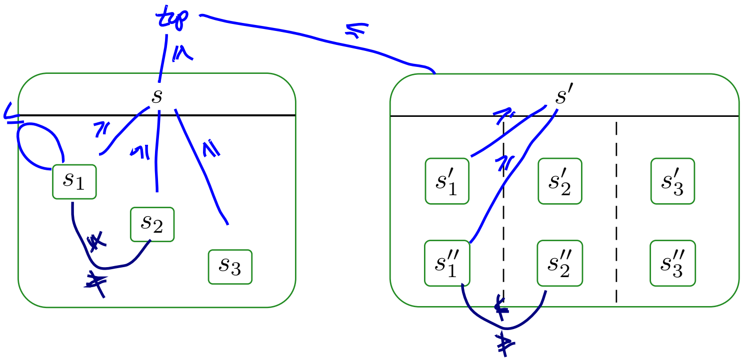
A Partial Order on States

The substate- (or **child**-) relation **induces** a **partial order on states**:

- $top \leq s$, for all $s \in S$,
- $s \leq s'$, for all $s' \in child(s)$,
- transitive, reflexive, antisymmetric,
- $s' \leq s$ and $s'' \leq s$ implies $s' \leq s''$ or $s'' \leq s'$.

$\forall s \bullet s \leq s$

write $s \geq s'$ iff $s' \leq s$

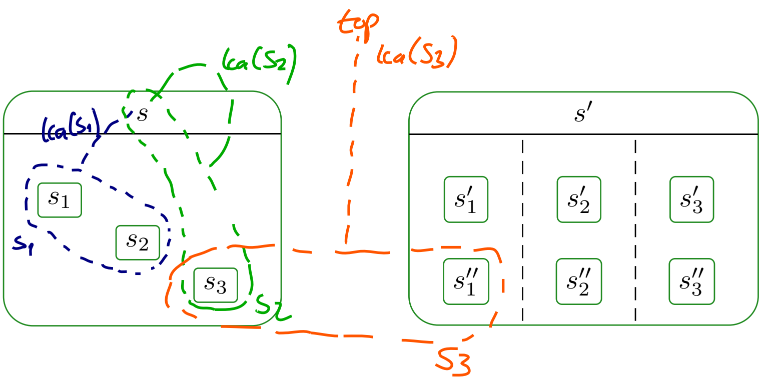


Least Common Ancestor and Ting

- The **least common ancestor** is the function $lca : 2^S \setminus \{\emptyset\} \to S$ such that
  - The states in S_1 are (transitive) children of $lca(S_1)$, i.e.

$$lca(S_1) \leq s, \text{ for all } s \in S_1 \subseteq S,$$

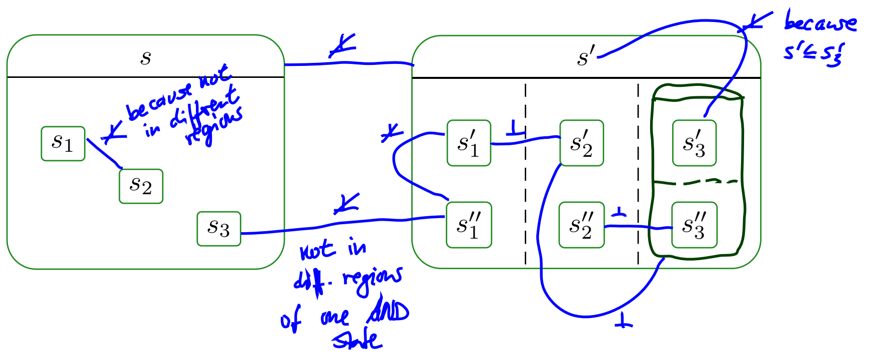
  - $lca(S_1)$ is minimal, i.e. if $\hat{s} \leq s$ for all $s \in S_1$, then $\hat{s} \leq lca(S_1)$
- **Note**: $lca(S_1)$ exists for all $S_1 \subseteq S$ (last candidate: top).



Least Common Ancestor and Ting

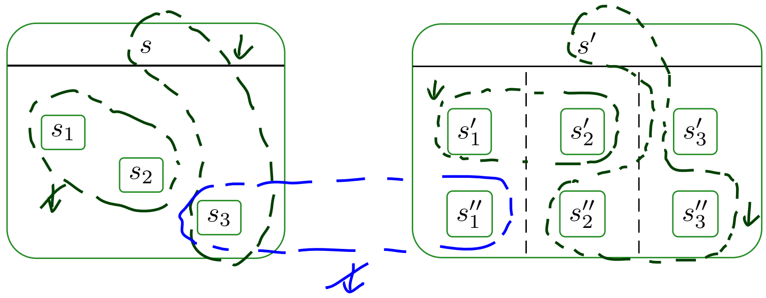
- Two states $s_1, s_2 \in S$ are called **orthogonal**, denoted $s_1 \perp s_2$, if and only if
  - they are unordered, i.e. $s_1 \not\leq s_2$ and $s_2 \not\leq s_1$, and
  - they "live" in different regions of an AND-state, i.e.

$$\exists\, s, \mathit{region}(s) = \{S_1, \dots, S_n\}\ \exists\, 1 \leq i \neq j \leq n : s_1 \in \mathit{child}^*(S_i) \wedge s_2 \in \mathit{child}^*(S_j),$$



Least Common Ancestor and Ting

- A set of states $S_1 \subseteq S$ is called **consistent**, denoted by $\downarrow S_1$, if and only if for each $s, s' \in S_1$,
  - $s \leq s'$, or
  - $s' \leq s$, or
  - $s \perp s'$.



Legal Transitions

A hiearchical state-machine $(S, kind, region, \rightarrow, \psi, annot)$ is called **well-formed** if and only if for all transitions $t \in \rightarrow$,

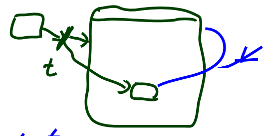
(i) source and destination are consistent, i.e. $\downarrow source(t)$ and $\downarrow target(t)$,

(ii) source (and destination) states are pairwise orthogonal, i.e.
- forall $s \neq s' \in source(t)$ $(\in target(t))$, $s \perp s'$,

(iii) the top state is neither source nor destination, i.e.
- $top \notin source(t) \cup \cancel{source}\ target(t)$.

- Recall: final states are not sources of transitions.

Example:



↳ t not legal

Legal Transitions

A hiearchical state-machine $(S, kind, region, \rightarrow, \psi, annot)$ is called **well-formed** if and only if for all transitions $t \in \rightarrow$,

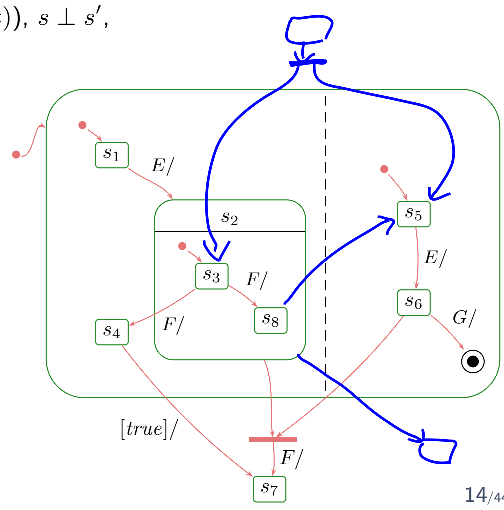
(i) source and destination are consistent, i.e. $\downarrow source(t)$ and $\downarrow target(t)$,

(ii) source (and destination) states are pairwise orthogonal, i.e.
- forall $s \neq s' \in source(t)$ $(\in target(t))$, $s \perp s'$,

(iii) the top state is neither source nor destination, i.e.
- $top \notin source(t) \cup \cancel{source}\ target(t)$.

- Recall: final states are not sources of transitions.

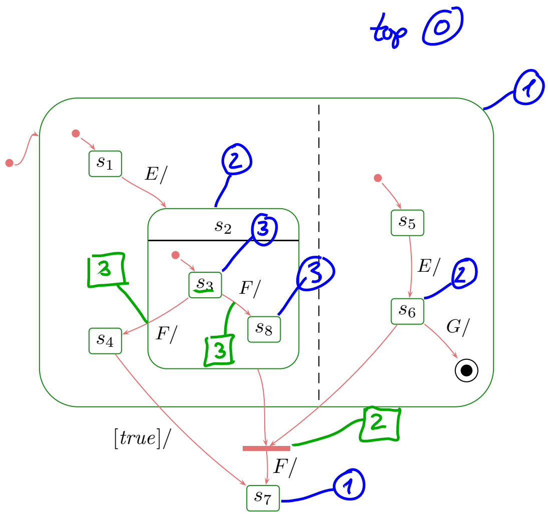
Example:



The Depth of States

- $depth(top) = 0$,
- $depth(s') = depth(s) + 1$, for all $s' \in child(s)$

Example:



Enabledness in Hierarchical State-Machines

- The **scope** ("set of possibly affected states") of a transition t is the **least common region** of
$$source(t) \cup target(t).$$
- Two transitions t_1, t_2 are called **consistent** if and only if their scopes are orthogonal (i.e. states in scopes pairwise orthogonal).
- The **priority** of transition t is the depth of its innermost source state, i.e.
$$prio(t) := \max\{depth(s) \mid s \in source(t)\}$$
- A set of transitions $T \subseteq \rightarrow$ is **enabled** in an object u if and only if
  - T is consistent,
  - T is maximal wrt. priority,
  - all transitions in T share the same trigger,
  - all guards are satisfied by $\sigma(u)$, and
  - for all $t \in T$, the source states are active, i.e.
$$source(t) \subseteq \sigma(u)(st) \ (\subseteq S).$$

Transitions in Hierarchical State-Machines

- Let T be a set of transitions enabled in u.
- Then $(\sigma, \varepsilon) \xrightarrow{(cons, Snd)} (\sigma', \varepsilon')$ if
  - $\sigma'(u)(st)$ consists of the target states of t,

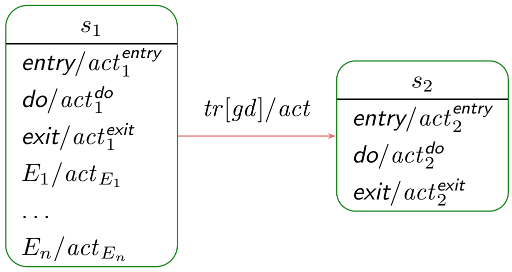
    i.e. for simple states the simple states themselves, for composite states the initial states,
  - σ', ε', $cons$, and Snd are the effect of firing each transition $t \in T$ **one by one**, **in any order**, i.e. for each $t \in T$,
    - the exit transformer of all affected states, highest depth first,
    - the transformer of t,
    - the entry transformer of all affected states, lowest depth first.

    later

$\leadsto$ adjust (2.), (3.), (5.) accordingly.

Entry/Do/Exit Actions, Internal Transitions

Entry/Do/Exit Actions



- In general, with each state $s \in S$ there is associated
  - an **entry**, a **do**, and an **exit** action (default: skip)
  - a possibly empty set of trigger/action pairs called **internal transitions**,

    (default: empty). $E_1, \dots, E_n \in \mathscr{E}$, 'entry', 'do', 'exit' are reserved names!

- Recall: each action's supposed to have a transformer. Here: $t_{act_1^{entry}}$, $t_{act_1^{exit}}$, ...
- Taking the transition above then amounts to applying

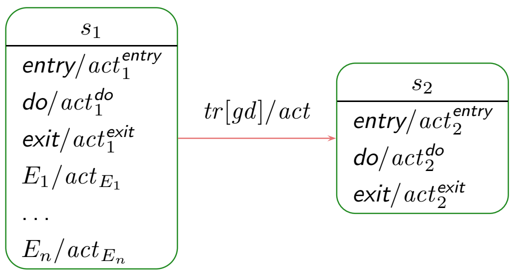
$$t_{act_{s_2}^{entry}} \circ t_{act} \circ t_{act_{s_1}^{exit}}$$

instead of only

$$t_{act}$$

$\rightsquigarrow$ adjust $(2.)$, $(3.)$ accordingly.

Internal Transitions



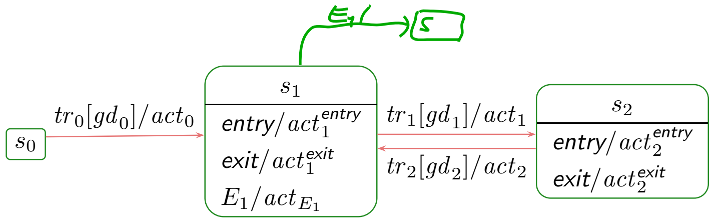
- For **internal transitions**, taking the one for E_1, for instance, still amounts to taking **only** $t_{act_{E_1}}$.
- Intuition: The state is neither left nor entered, so: no exit, no entry.

  $\rightsquigarrow$ adjust $(2.)$ accordingly.
- Note: internal transitions also start a run-to-completion step.

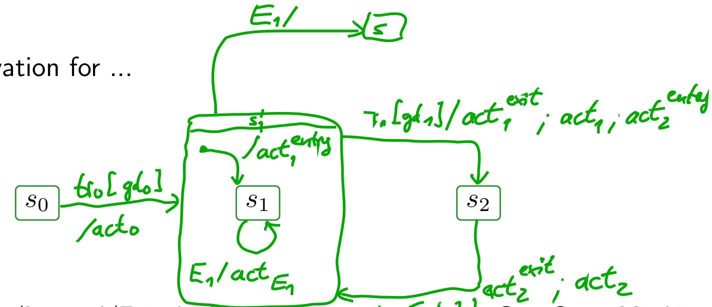
- Note: the standard seems not to clarify whether internal transitions have **priority** over regular transitions with the same trigger at the same state.

  Some code generators assume that internal transitions have priority!

Alternative View: Entry/Exit/Internal as Abbreviations

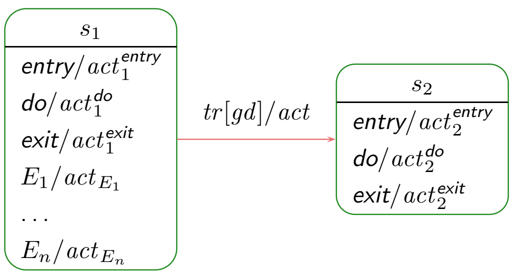


- … as abbrevation for …



- That is: Entry/Internal/Exit don't add expressive power to Core State Machines. If internal actions should have priority, s_1 can be embedded into an OR-state (see later).
- Abbreviation may avoid confusion in context of hierarchical states (see later).

Do Actions



- **Intuition**: after entering a state, start its do-action.
- If the do-action terminates,
  - then the state is considered **completed**,
- otherwise,
  - if the state is left before termination, the do-action is stopped.

- Recall the overall UML State Machine philosophy:

  "An object is either idle or doing a run-to-completion step."
- Now, what is it exactly while the do action is executing…?

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.