

*Software Design, Modelling and Analysis in UML*

*Lecture 16: Hierarchical State Machines III*

*2017-01-12*

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

Albert-Ludwigs-Universität Freiburg, Germany

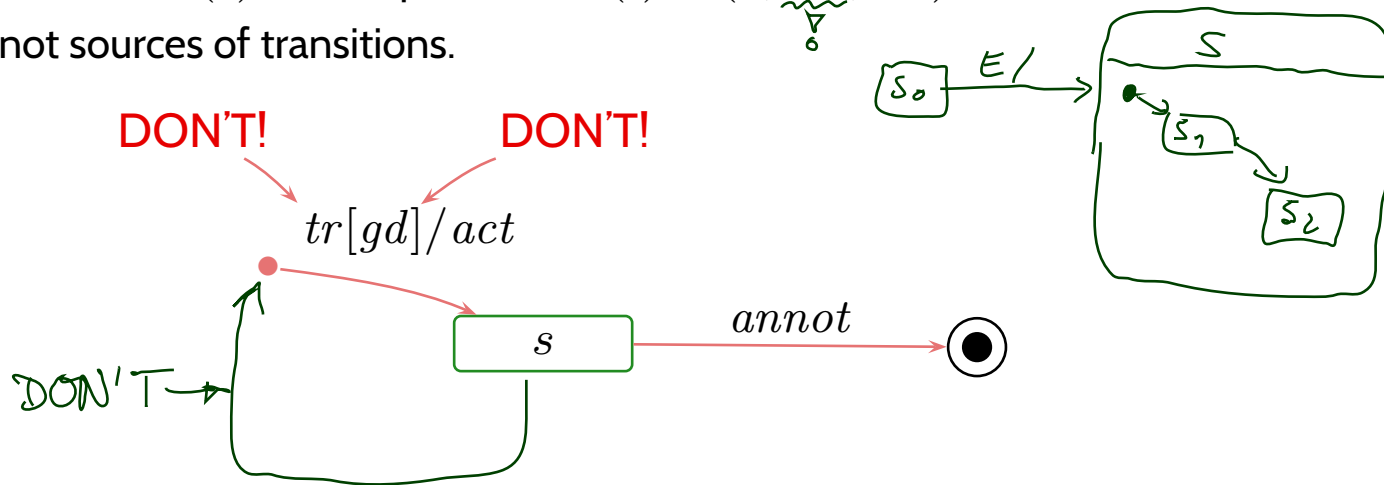
- **Hierarchical State Machines**
  - Additional **Well-Formedness** Constraints
  - An(other) **intuition** for hierarchical states
  - **Entry and Exit Actions**
  - **Initial and Final States**
- **Rhapsody Demo**: Automated Tests
- **Hierarchical State Machines**: The Rest
  - **History Connectors**
  - **Junction and Choice**
  - **Entry and Exit Points**
  - **Terminate**
- **Active vs. Passive Objects**

# Additional Well-Formedness Constraints

- Each non-empty region has **exactly one** initial pseudo-state and **at least one** transition from there to a state of the region, i.e.
  - for each  $s \in S$  with  $region(s) = \{S_1, \dots, S_n\}$ ,
  - for each  $1 \leq i \leq n$ , there exists exactly one initial pseudo-state  $(s_1^i, init) \in S_i$  and at least one transition  $t \in \rightarrow$  with  $s_1^i$  as source,
- Initial pseudo-states are not targets of transitions.

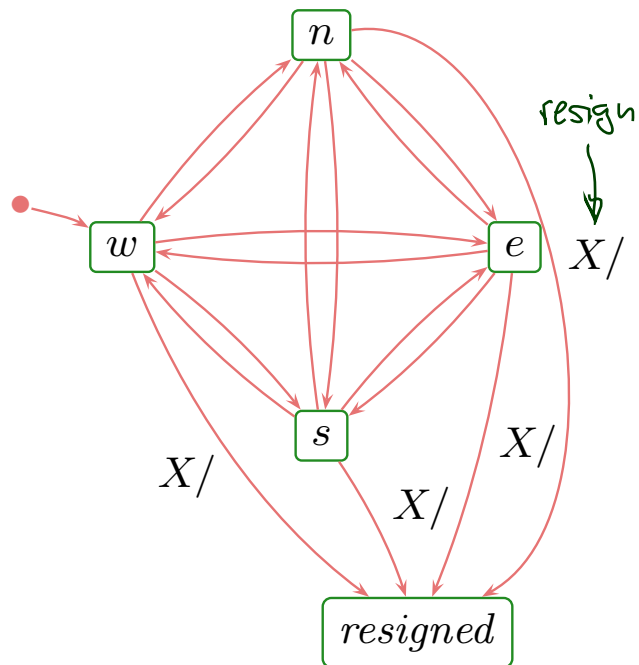
## For simplicity:

- The target of a transition with initial pseudo-state source in  $S_i$  is (also) in  $S_i$ .
- Transitions from initial pseudo-states have no trigger or guard, i.e.  $t \in \rightarrow$  from  $s$  with  $kind(s) = st$  implies  $annot(t) = (\_, \underbrace{true}_{\circlearrowleft}, act)$ .
- Final states are not sources of transitions.

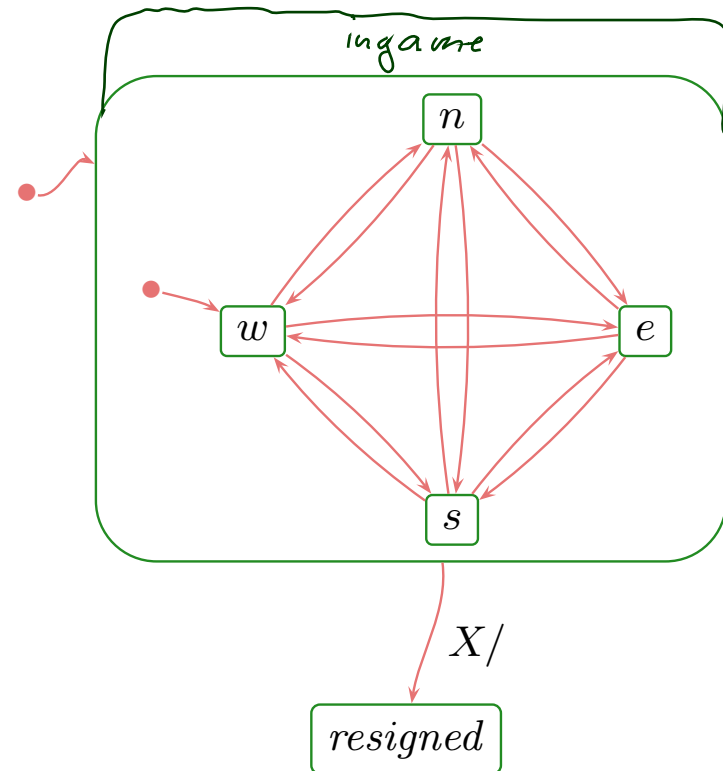


# An Intuition for “Or-States”

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

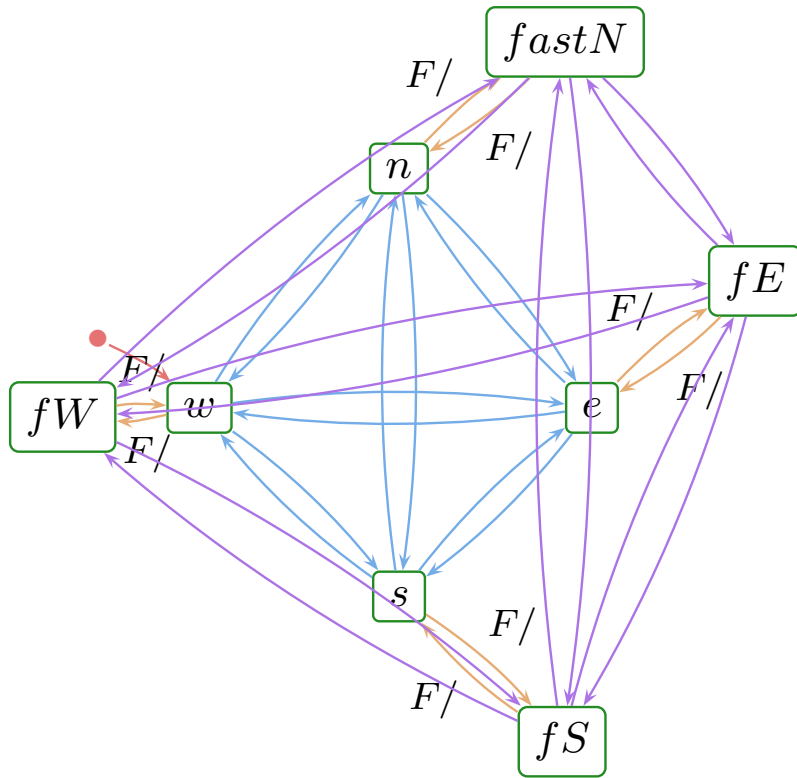


write

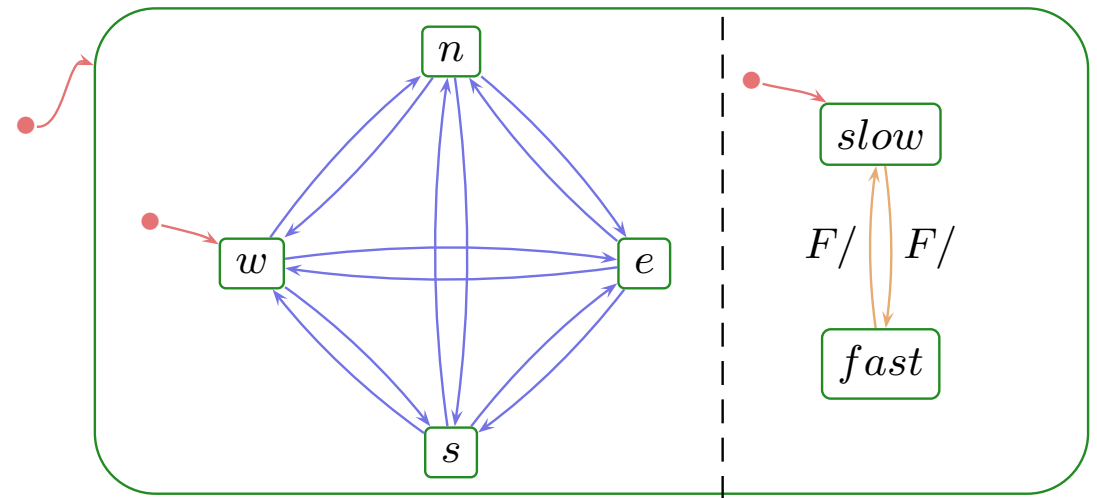


# An Intuition for “And-States”

and instead of



write

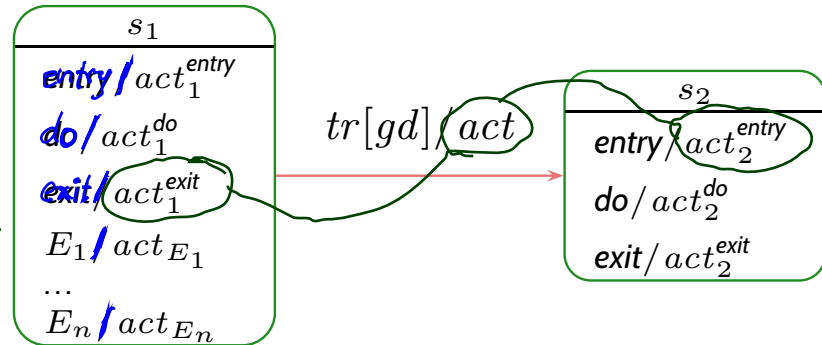


# *Entry and Exit Actions*

# 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).

internal transitions



**Note:** 'entry', 'do', 'exit' are reserved names;  $E_1, \dots, E_n \in \mathcal{E}$ .

- **Recall:** each action is supposed to have a transformer; assume  $t_{act_1^{entry}}, t_{act_1^{exit}}, \dots$
- Taking the transition above then amounts to applying

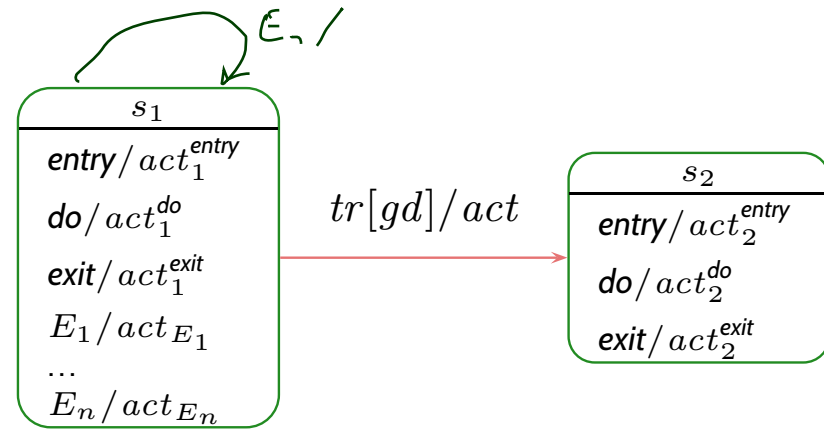
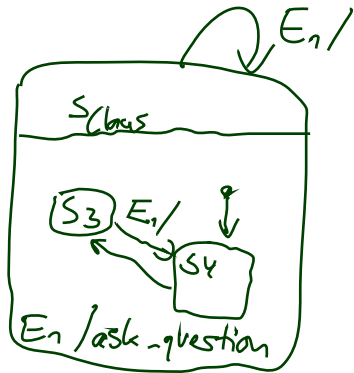
$$t_{act_2^{entry}} \circ t_{act} \circ t_{act_1^{exit}}$$

instead of just

$$t_{act}$$

$\rightsquigarrow$  adjust Rules (ii), (iii), and (v) accordingly.

# Internal Transitions



- Taking an **internal transition**, e.g. on  $E_1$ , only executes  $t_{act_{E_1}}$ .
- **Intuition**: The state is neither left nor entered, so: no exit, no entry action.
- **Note**: internal transitions also start a run-to-completion step.

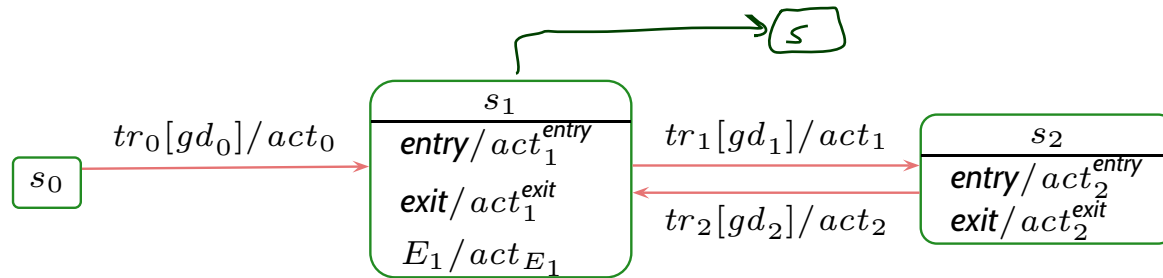
$\rightsquigarrow$  adjust Rules (i), (ii), and (v) accordingly.

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

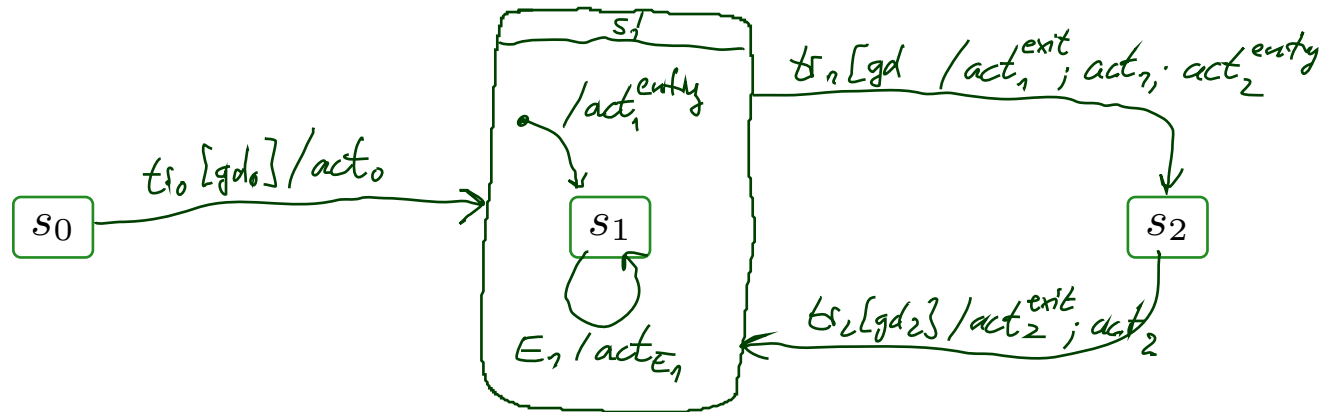
Some code generators assume that internal transitions have priority!



# Alternative View: Entry / Exit / Internal as Abbreviations

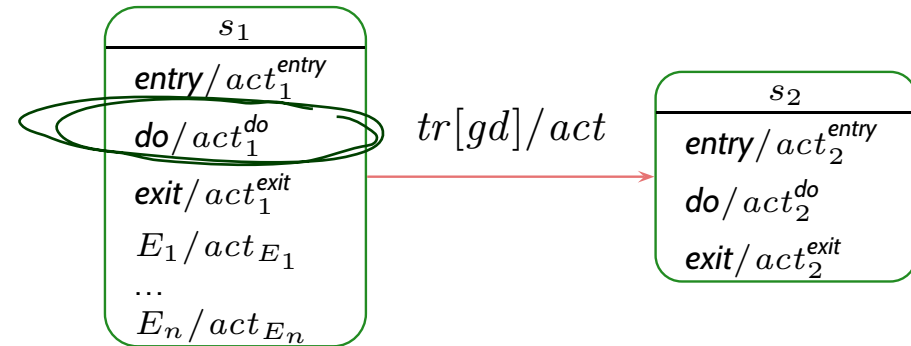


Can be viewed as abbreviation for ...



- **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.
- The "abbreviation view" may avoid confusion in the context of hierarchical states.

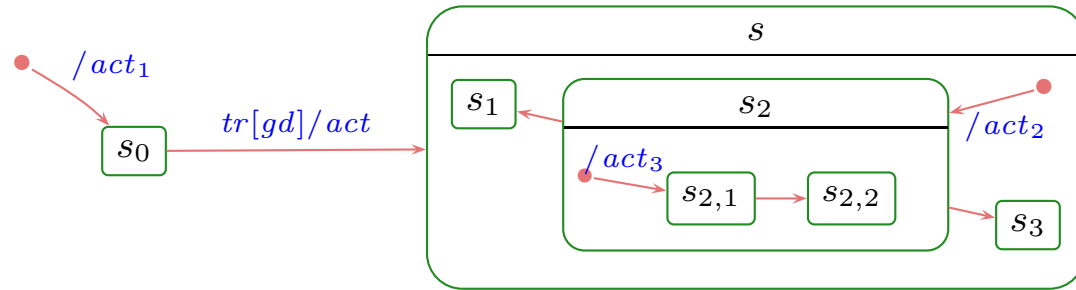
# Do Actions



- **Intuition:** after entering a state, start its do-action.
- If the do-action terminates,
  - then the state is considered **completed** (like reaching a **final state** child ( $\rightarrow$  in a minute)),
    - then rule (iii) (continue) may apply
  - 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...?

## *Initial and Final States*

# Initial Pseudostate



## Principle:

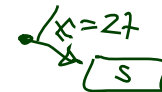
- when entering a non-simple state,
- then go to the destination state of a transition with initial pseudo-state source,
- execute the action of the chosen initiation transition(s) **between** exit and entry actions.

**Recall:** For simplicity, we assume exactly one initiation transition per non-empty region. Could also be: "at least one" and choosing one non-deterministically.

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

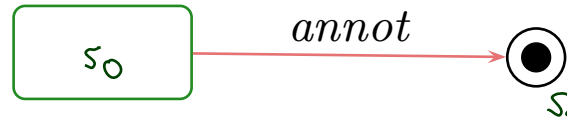
- If class  $C$  has a state-machine, then "create- $C$  transformer" is the concatenation of
  - the transformer of the "constructor" of  $C$  (here not introduced explicitly) and
  - a transformer corresponding to one initiation transition of the top region.

$SM_C$ :



$\{x=13\}$

# Final States

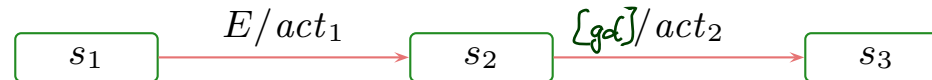


- If  $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$   
and all **simple states**  $s$  in  $\sigma'(u)(st)$  are **final**, i.e.  $kind(s) = fin$ , then
  - stay **unstable** if there is a common parent of the simple states in  $\sigma(u)(st)$  which is source of a transition without trigger and satisfied guard,
  - otherwise **kill** (destroy) object  $u$ .

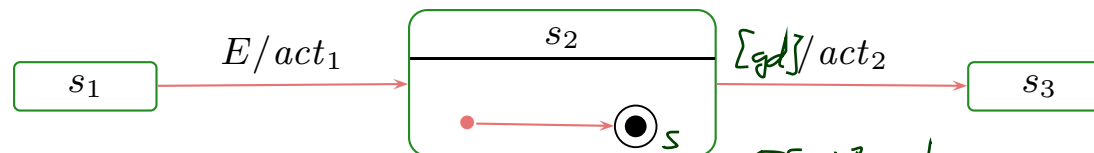
↪ adjust Rules (i), (ii), (iii), and (v) accordingly.

**Observation:**  $u$  never “survives” reaching a state  $(s, fin)$  with  $s \in child(top)$ .

**Observation:**



vs.



$\llbracket [gd] \rrbracket$ : true  $\mapsto$  move to  $s_3$   
false  $\mapsto$  kill object

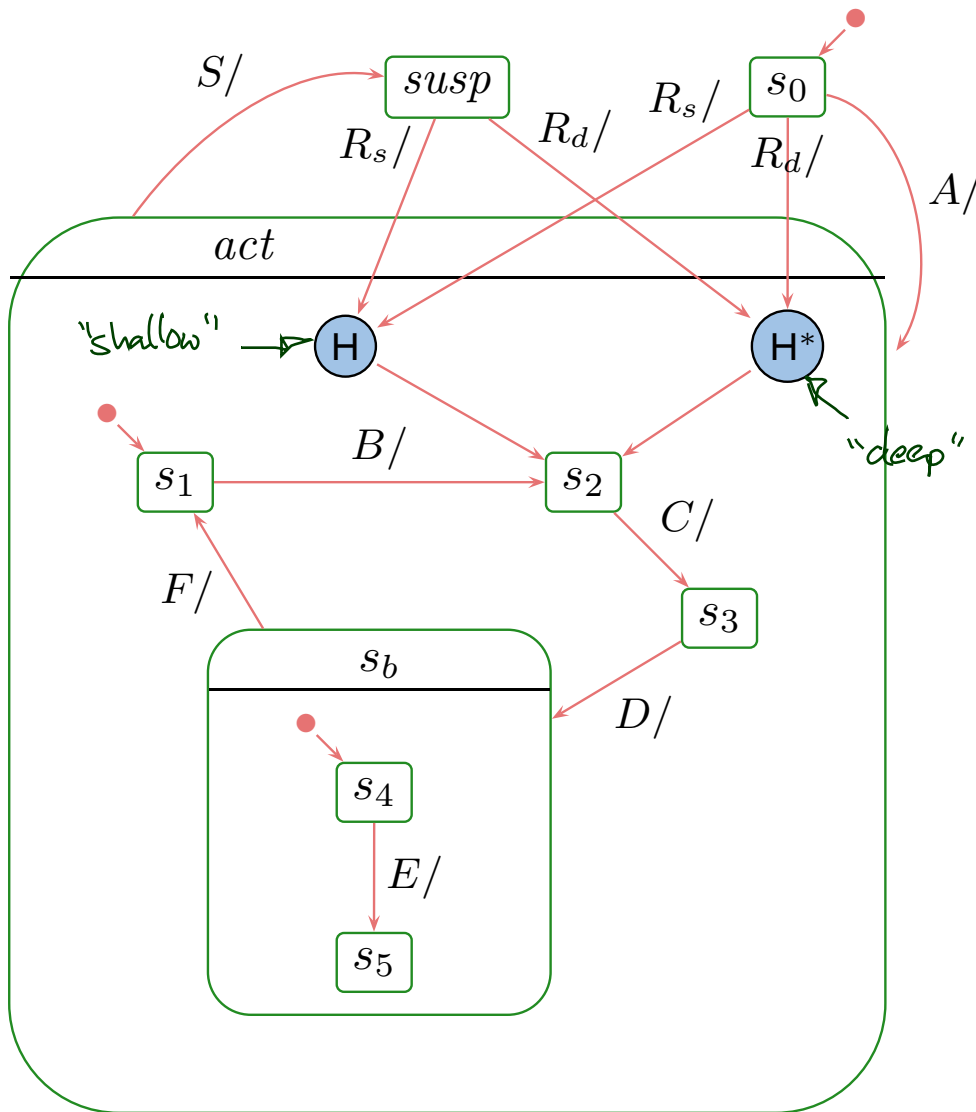
# *Rhapsody Demo: Automated Testing*



# *The Concept of History, and Other Pseudo-States*



# History and Deep History: By Example

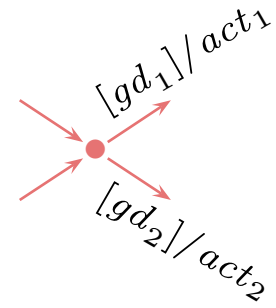
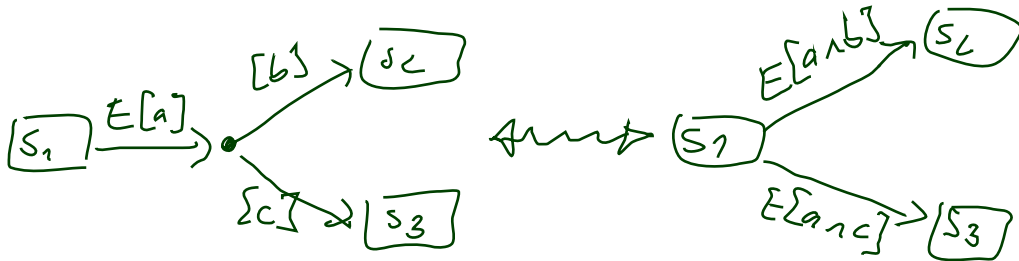


What happens on...

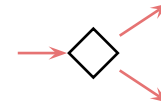
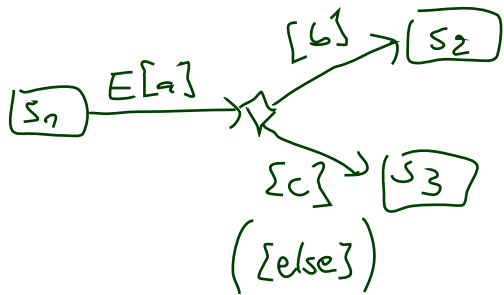
- *R<sub>s</sub>*?  
*s0, s2*
- *R<sub>d</sub>*?  
*s0, s2*
- *A, B, C, S, R<sub>s</sub>*?  
*s0, s1, s2, s3, susp, s3*
- *A, B, C, S, R<sub>d</sub>*?  
*s0, s1, s2, s3, s3*
- *A, B, C, D, E, S, R<sub>s</sub>*?  
*s0, s1, s2, s3, s4, s5, susp, s4, s3*
- *A, B, C, D, E, S, R<sub>d</sub>*?  
*s0, s1, s2, s3, s4, s5, susp, s5*

# Junction and Choice

- Junction (“static conditional branch”):



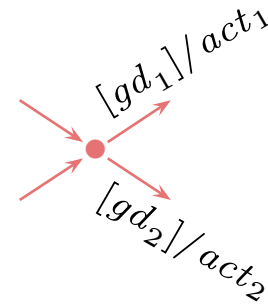
- Choice: (“dynamic conditional branch”)



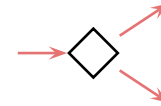
# Junction and Choice

---

- Junction (“**static conditional branch**”):
  - **good**: abbreviation
  - unfolds to so many similar transitions with different guards, the unfolded transitions are then checked for enabledness
  - at best, start with trigger, branch into conditions, then apply actions



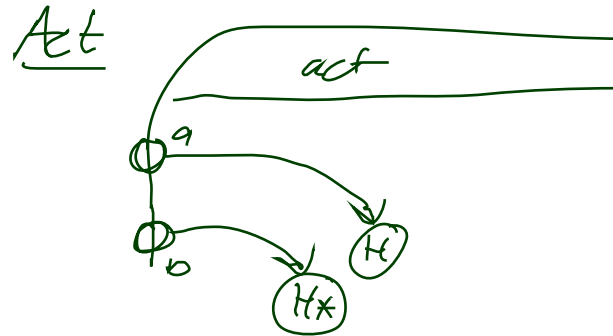
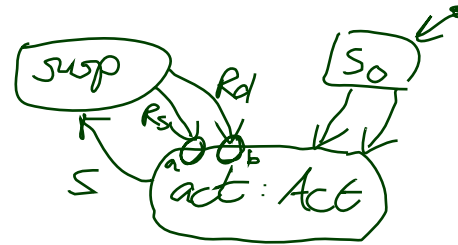
- Choice: (“**dynamic conditional branch**”)
  - **evil**: may get stuck
  - enters the transition **without knowing** whether there’s an enabled path
  - at best, use “else” and convince yourself that it cannot get stuck
  - maybe even better: **avoid**



# Entry and Exit Point, Submachine State, Terminate

- Hierarchical states can be “**folded**” for readability.  
(but: this can also hinder readability.)
- Can even be taken from a different state-machine for re-use.

$S : s$



# Entry and Exit Point, Submachine State, Terminate

- Hierarchical states can be “**folded**” for readability.  
(but: this can also hinder readability.)
- Can even be taken from a different state-machine for re-use.
- **Entry/exit points**
  - Provide connection points for finer integration into the current level, finer than just via initial state.
  - Semantically a bit tricky:
    - **First** the exit action of the exiting state,
    - **then** the actions of the transition,
    - **then** the entry actions of the entered state,
    - **then** action of the transition from the entry point to an internal state,
    - and **then** that internal state’s entry action.
- **Terminate Pseudo-State**
  - When a terminate pseudo-state is reached, the object taking the transition is immediately killed.

$S : s$



Rhapsody : 

# *Tell Them What You've Told Them...*

---

- OR- and AND-states could also be explained as an “unfolding” into core state machines.
  - They add **conciseness**, not **expressive power**.
  - The remaining pseudo-states (history, junction, choice, etc.) are not so difficult.
  - Modelling guideline: Avoid **choice**.
- ( ● Rhapsody also supports **non-active objects** – their instances share an event pool with an **active object**. )

# *References*

# References

---

- Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.
- OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.
- OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.