

# Software Design, Modelling and Analysis in UML

## Lecture 17: Hierarchical State Machines II

2014-01-20

Prof. Dr. Andreas Podolski, Dr. Bernd Westphal  
Albert-Ludwigs-Universität Freiburg, Germany

### Contents & Goals

#### Last Lecture:

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

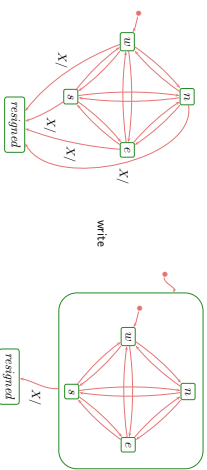
#### This Lecture:

- **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

Composite States  
*(modification follows Demme et al., 2003)*

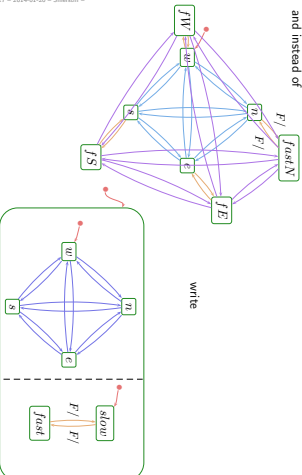
### Composite States

- In a sense, composite states are about **abstraction, structuring, and avoiding redundancy.**
- Idea: in Tron, for the Player's StateMachine, instead of

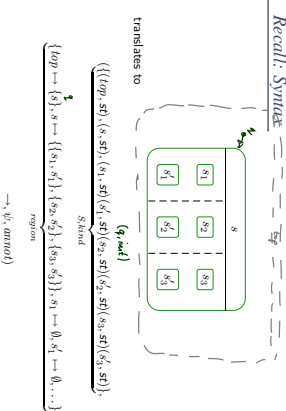


### Composite States

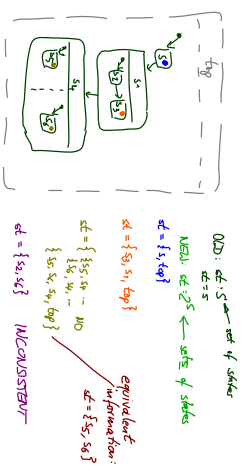
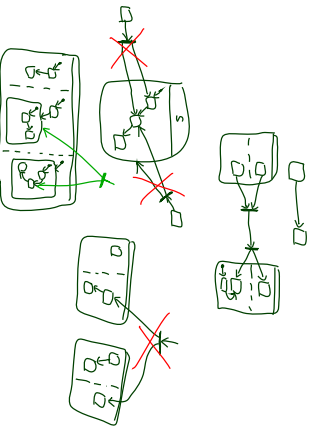
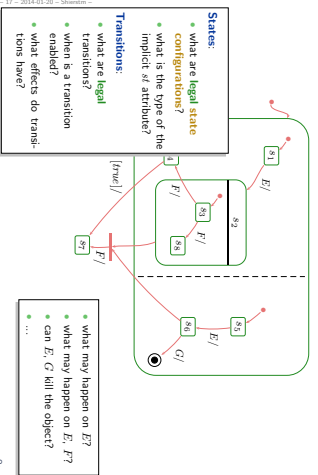
and instead of



### Recall: Syntax



### Composite States: Blessing or Curse?



### Syntax: ForkJoin

- For brevity, we always consider transitions with (possibly) multiple sources and targets, i.e. set of src states, set of target or desc.

For instance,  
 $\psi : (-) \mapsto (2^S \setminus \emptyset) \times (2^S \setminus \emptyset)$   
 assume: one source per "time"

translates to

$$(S \text{ kind, region, } \{t_i\}) \xrightarrow{\psi} (\{t_i \mapsto (\{s_1, s_2, s_3\}, \{s_4, s_5, s_6\})\}) \xrightarrow{\text{arrivals}} (\{t_i \mapsto (\{r_i \text{ obj. arr.}\})\})$$

• Naming convention:  $\psi^s(t) = (\text{source}(t), \text{target}(t))$

### State Configuration

- The type of  $s_i$  is from now on a set of states, i.e.  $s_i: 2^S$
- A set  $S_i \subseteq S$  is called (legal) state configurations if and only if
  - $\text{top} \in S_i$ , and
  - for each state  $s \in S_i$ , for each non-empty region  $\emptyset \neq R \in \text{region}(s)$ , exactly one (non pseudo-state) child of  $s$  (from  $R$ ) is in  $S_i$ , i.e.
 
$$|\{s_{in} \in R \mid \text{kind}(s_{in}) \in \{\text{st, fn}\}\} \cap S_i| = 1.$$

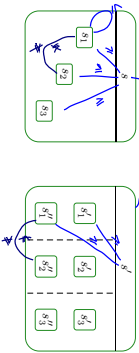
### A Partial Order on States

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

- $\text{top} \leq s$ , for all  $s \in S$ .
  - $s \leq s'$ , for all  $s' \in \text{child}(s)$ .
  - transitive, reflexive, antisymmetric.
  - $s' \leq s$  and  $s'' \leq s$  implies  $s' \leq s''$  or  $s'' \leq s'$ .
- $$\begin{array}{ccccccc}
 s' & & s'' & & & & s \\
 \swarrow & & \searrow & & & & \downarrow \\
 s_1 & & s_2 & & & & s_3 \\
 \downarrow & & \downarrow & & & & \downarrow \\
 s_1 & & s_2 & & & & s_3 \\
 \downarrow & & \downarrow & & & & \downarrow \\
 s_1 & & s_2 & & & & s_3
 \end{array}$$

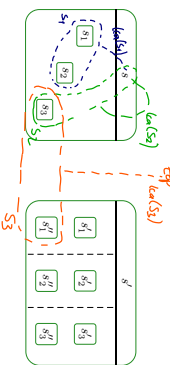
The substrate- (or child-) relation induces a **partial order on states**:

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

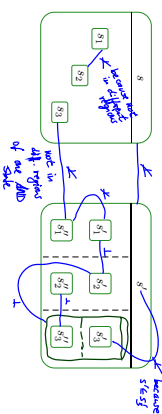


- The **least common ancestor** is the function  $lca: 2^S \setminus \{\emptyset\} \rightarrow S$  such that
- The states in  $S_l$  are (transitive) children of  $lca(S_l)$ , i.e.  $lca(S_l) \leq s_i$  for all  $s_i \in S_l \subseteq S$ .

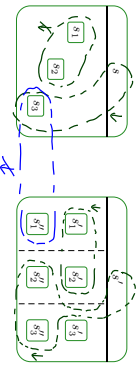
- $lca(S_l)$  is **unique**, i.e. if  $s \leq s'$  for all  $s_i \in S_l$ , then  $s \leq lca(S_l)$
- **Note**:  $lca(S_l)$  exists for all  $S_l \subseteq S$  (last candidate:  $top$ )



- Two states  $s_1, s_2 \in S$  are called **orthogonal**, denoted  $s_1 \perp s_2$ , if and only if
- they are **unrelated**, 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.  $S_{\text{AND}}(s) = \{S_1, \dots, S_n\} \exists 1 \leq i \neq j \leq n: s_1 \in child(S_i) \wedge s_2 \in child(S_j)$ .



- A set of states  $S_l \subseteq S$  is called **consistent**, denoted by  $\perp S_l$ , if and only if for each  $s_i, s_j \in S_l$ ,
- $s_i \leq s_j$ , or
- $s_j \leq s_i$ , or
- $s_i \perp s_j$ .



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

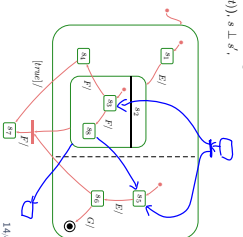
- (i) source and destination are consistent, i.e.  $\perp source(t)$  and  $\perp target(t)$ ,
  - (ii) source (and destination) states are pairwise orthogonal, i.e.
    - $\forall s_i, s_j \in source(t) (\in target(t)), s_i \perp s_j$ .
  - (iii) the top state is neither source nor destination, i.e.
    - $top \notin source(t) \cup target(t)$ .
- Recall: final states are not sources of transitions.



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

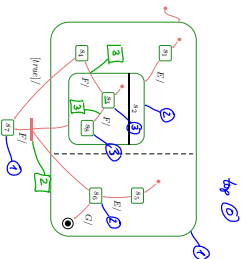
- (i) source and destination are consistent, i.e.  $\perp source(t)$  and  $\perp target(t)$ ,
  - (ii) source (and destination) states are pairwise orthogonal, i.e.
    - $\forall s_i, s_j \in source(t) (\in target(t)), s_i \perp s_j$ .
  - (iii) the top state is neither source nor destination, i.e.
    - $top \notin source(t) \cup target(t)$ .
- Recall: final states are not sources of transitions.

**Example:**



- $depth(top) = 0$ .
- $depth(s') = depth(s) + 1$ , 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.  $priority(t) := \max\{depth(s) \mid s \in source(t)\}$ .
- A set of transitions  $T \subseteq \rightarrow$  is **enabled** in an object  $u$  if and only if
  - $T$  is consistent,
  - $T$  is maximal w.r.t. 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.  $source(t) \subseteq \sigma(u)(sk) \subseteq S$ .

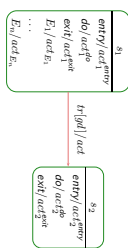
Transitions in Hierarchical State-Machines

- Let  $T$  be a set of transitions enabled in  $u$ .
- Then  $(\sigma, \rho) \xrightarrow{trans, S, \text{Std}} (\sigma', \rho')$  if
  - $\sigma'(u)(sk)$  consists of the target states of  $T$ , i.e. for simple states the simple states themselves, for composite states the initial states.
  - $\sigma', \rho', \text{cons}$ , and  $S, \text{Std}$  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.

Entry/Do/Exit Actions, Internal Transitions

Entry/Do/Exit Actions

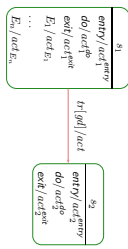
- In general, with each state  $s \in S$  there is associated an **entry**, a **do**, and an **exit** action (default: skip)
- a possibly empty set of trigger/action pairs called **internal transitions**, (default: empty).  $E_1, \dots, E_n \in \mathcal{E}$ , 'entry', 'do', 'exit' are reserved named

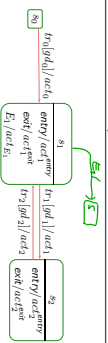


- Recall: each action's supposed to have a transformer. Here:  $f_{act1^entry}, f_{act1^do}, \dots$
- Taking the transition above then amounts to applying  $f_{act1^entry} \circ f_{act1^do} \circ f_{act1^exit}$  instead of only  $f_{act1}$
- adjust (2.), (3.) accordingly

Internal Transitions

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





... as abbreviation for ...

- That is: Entry/Internal/Exit don't add *expressive power* to Core State Machines if internal actions should have priority: s1 can be embedded into an Or-state (see later).
- Abbreviation may avoid confusion in context of hierarchical states (see later).

## References

[Cane and Dingel, 2007] Cane, 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.

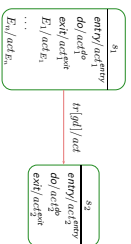
[Damm et al., 2003] Damm, W., Josko, B., Vainseva, A., and Pnueli, A. (2003). A formal semantics for a UML kernel language 1.2. IST/33522/WP 1.1/D1.1.2-Part1, Version 1.2.

[Fischer and Schönborn, 2007] Fischer, H. and Schönborn, J. (2007). UML 2.0 state machines: Complete formal semantics via core state machines. In Birn, L., Havelkort, 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., Gotte-Rivode, M., Reif, W., Schneider, E., and Weiskämper, 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.



- **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