Software Design, Modelling and Analysis in UML

Lecture 18: Hierarchical State Machines II

2015-01-22

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

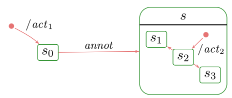
- Hierarchical State Machine Syntax
- Entry/Exit Actions

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:**
  - Initial and Final State
  - Composite State Semantics
  - The Rest

Initial Pseudostates and Final States

Initial Pseudostate



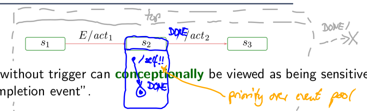
Principle:

- 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.

Special case: the region of top.

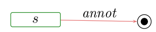
- If class C has a state-machine, then "create-C transformer" is the concatenation of
  - the transformer of the "constructor" of C (here not introduced explicitly) and
  - a transformer corresponding to one initiation transition of the top region.

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, 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

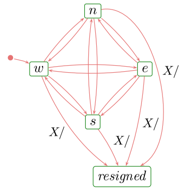
  $\leadsto$ adjust (2.) and (3.) in the semantics accordingly
- **One consequence**:
  u never "survives" reaching a state (s, fin) with $s \in child(top)$.

Composite States

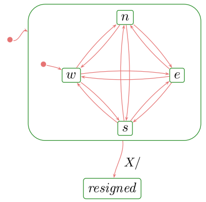
(formalisation follows [Damm et al., 2003])

Composite States

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

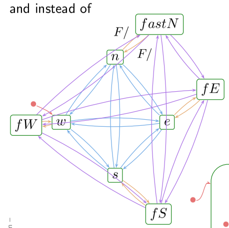


write

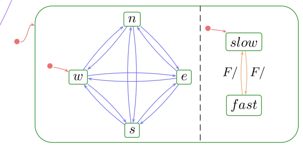


Composite States

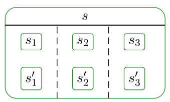
and instead of



write



Recall: Syntax



translates to

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

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

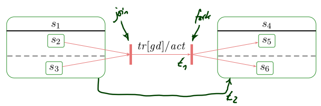
$$\to, \psi, annot)$$

Syntax: Fork/Join

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

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

- For instance,



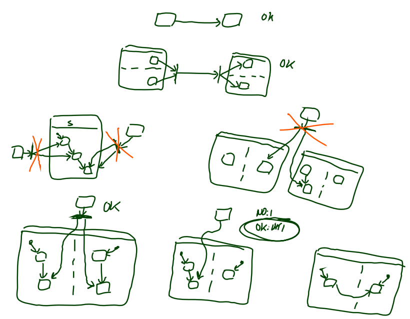
translates to

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

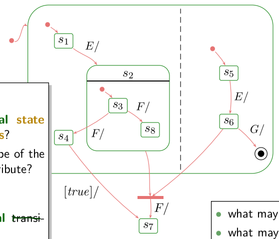
t_2

$t_2 \mapsto (\{s_1\}, \{s_4\})$

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



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?~~ effects?
- when is a transition enabled?
- what effects do transitions have?

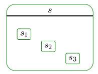
- what may happen on E?
- what may happen on E, F?
- can E, G kill the object?
- ...

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.

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

- **Examples**:



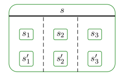
$S = \{s_2\}$ ✗ (top missing)
$S = \{s_2, top\}$ ✗ (no child of top's region)
$S = \{top, s, s_2\}$ ✓

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.

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

- **Examples**:



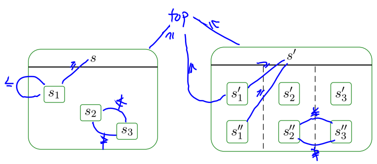
$S = \{top, s, s_1, s_2, s_3\}$ ✓

NOTE: S can be abbreviated as $\{s_1, s_2, s_3\}$

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 $s'' \leq s'$.

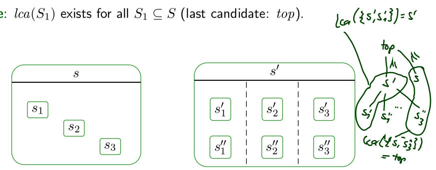


Least Common Ancestor and Ting

- 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).

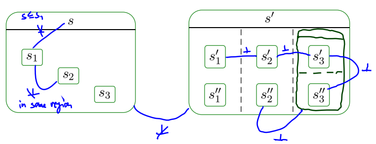


Least Common Ancestor and Ting

- 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.

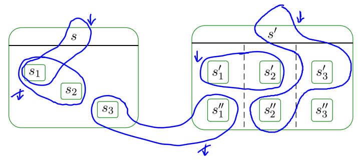
transitive child

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



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 (tags)

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.

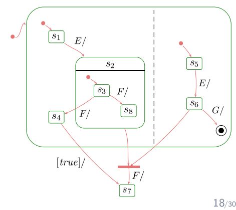
- forall $s \neq s' \in source(t)$ $(\in target(t))$, $s \perp s'$,

(iii) the top state is neither source nor destination, i.e.

- $top \notin source(t) \cup source(t)$.

- Recall: final states are not sources of transitions.

Example:



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 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.