Software Design, Modelling and Analysis in UML

Lecture 18: Hierarchical State Machines II

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 \textbf{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 \textbf{between} exit and entry actions.

Special case: the region of \textit{top}.
- If class \textit{C} has a state-machine, then “create-\textit{C} transformer” is the concatenation of
  - the transformer of the “constructor” of \textit{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:
  1. fetch event (here: \(E\)) from the ether,
  2. take an enabled transition (here: to \(s_2\)),
  3. remove event from the ether,
  4. after having finished entry and do action of current state (here: \(s_2\)) — the state is then called **completed** —,
  5. raise a **completion event** — with strict priority over events from ether!
  6. 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 \(u\)
  \( \rightsquigarrow \text{adjust (2.) and (3.) in the semantics accordingly} \)

- **One consequence:**
  \(u\) never "survives" reaching a state \((s, fin)\) with \(s \in \text{child}(\text{top})\).
Composite States
(formalisation follows [Damm et al., 2003])

- 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

\[
\text{write}
\]

Recall: Syntax

translates to

\[
\{(\{\text{top}, \text{st}\}, \{(s, \text{st}), (s_1, \text{st})(s'_1, \text{st})(s_2, \text{st})(s'_2, \text{st})(s_3, \text{st})(s'_3, \text{st})\}),
\quad S, \text{kind} \}
\]

\[
\{(\text{top} \mapsto \text{\{\text{s}\}}, \text{s} \mapsto \{\{s_1, s'_1\}, \{s_2, s'_2\}, \{s_3, s'_3\}\}, s_1 \mapsto \emptyset, s'_1 \mapsto \emptyset, \ldots\},
\quad \text{region} \}
\]

\[
\rightarrow, \psi, \text{annot}
\]
**Syntax: Fork/Join**

- For brevity, we always consider transitions with (possibly) multiple sources and targets, i.e.
  \[
  \psi : (\rightarrow) \rightarrow (2^S \setminus \emptyset) \times (2^S \setminus \emptyset)
  \]

- For instance,

  ![Diagram of Fork/Join transition]

  translates to

  \[
  (S, \text{kind}, \text{region}, \{t_1\}, \{t_1 \mapsto (\{s_2, s_3\}, \{s_5, s_6\})\}, \{t_1 \mapsto (\text{tr, gd, act})\})
  \]

  \[
  \xrightarrow{\psi} \xrightarrow{\text{annot}} (\{s_1, s_4\})
  \]

- Naming convention: \( \psi(t) = (\text{source}(t), \text{target}(t)) \).
**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? edges?
- when is a transition enabled?
- what effects do transitions have?

---

**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:**
  - \( S = \{s_2, s\} \times \text{ (top missing) } \)
  - \( S = \{s_2, top\} \times \text{ (no child of top region) } \)
  - \( S = \{top, s, s_2\} \checkmark \)
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
  - \( \text{top} \in S_1 \), and
  - for each state \( s \in S_1 \), for each non-empty region \( \emptyset \neq R \in \text{region}(s) \), exactly one (non pseudo-state) child of \( s \) (from \( R \)) is in \( S_1 \), i.e.
    \[
    |\{s_0 \in R \mid \text{kind}(s_0) \in \{\text{st, fin}\}\} \cap S_1| = 1.
    \]

- Examples:

  \[
  S = \{\text{top}, s_1, s_2, s_3\}
  \]

  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:

- \( \text{top} \leq s \), 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 \( s'' \leq s' \).
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).

*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.

$$\exists s, \text{region}(s) = \{S_1, \ldots, S_n\} \exists 1 \leq i \neq j \leq n : s_1 \in \text{child}^*(S_i) \land s_2 \in \text{child}^*(S_j).$$
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 hierarchical state-machine $(S, \text{kind}, \text{region}, \rightarrow, \psi, \text{annot})$ is called well-formed if and only if for all transitions $t \in \rightarrow$,

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

(ii) source (and destination) states are pairwise orthogonal, i.e.
- for all $s \neq s' \in \text{source}(t) \cup \text{target}(t)$, $s \perp s'$.

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

Recall: final states are not sources of transitions.

Example:
References


