Contents & Goals

Last Lecture:

- Hierarchical State Machines: partial order, “lca”, orthogonality, ...

This Lecture:

- Educational Objectives: Capabilities for following tasks/questions.
  - 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:
  - Legal Transitions
  - Exit/Entry, internal transitions
  - History and others
  - Rhapsody Demo
Composite States

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

\(\) for all \(s, s' \in \text{source}(t) (\in \text{target}(t))\), \(s \perp s'\),

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

\(\) \(\text{top} \notin \text{source}(t) \cup \text{source}(t)\).

\(\) Recall: final states are not sources of transitions.

Example:

CLAIM: \(\text{(ii)} \implies \text{(i)}\)
The Depth of States

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

Example:

```
s_1 \to E/ s_2
\quad [\text{true}]/ \quad F/
\quad s_3 \quad F/
\quad s_4 \to F/
\quad s_5 \to E/
\quad s_6 \to G/
\quad s_7 \to F/
\quad s_8

\text{Rule 1}:
- \{$t_1, t_2$\} cons.
- \{$t_3, t_4$\} not cons.
- \{$t_5, t_6$\} not cons.

\text{Rule 2}:
- \{$t_7, t_8$\} cons.
```
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.

  $$prio(t) := \max\{\text{depth}(s) \mid s \in source(t)\}$$

- A set of transitions $T$ 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 $(\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})} (\sigma', \varepsilon')$ 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,
  - $\sigma', \varepsilon', \text{cons}$, and $\text{Snd}$ 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.

$\Rightarrow$ 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, \ldots, E_n \in \mathcal{D}, \text{‘entry’}, \text{‘do’}, \text{‘exit’ are reserved names!}

- Recall: each action’s supposed to have a transformer. Here: \( t_{\text{act}^\text{entry}}\ s_1, t_{\text{act}^\text{exit}}\ s_1, \ldots \)
- Taking the transition above then amounts to applying
  \[
  t_{\text{act}^\text{entry}}\ s_2 \circ t_{\text{act}} \circ t_{\text{act}^\text{exit}}\ s_1 (s) \sim \epsilon_{\text{entry}}(t_{\text{act}}(t_{\text{act}^\text{entry}}(s)))
  \]
  instead of only
  \[
  t_{\text{act}}
  \]
  \( \sim \) adjust (2.), (3.) accordingly.
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.

$\Rightarrow$ 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 abbreviation for ...

\[ tr_0[gd_0]/act_0 \]
Alternative View: Entry/Exit/Internal as Abbreviations

- ... 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 (see later).

- Abbreviation may avoid confusion in context of hierarchical states (see later).
**Do Actions**

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry/ $act^\text{entry}_1$</td>
<td>entry/ $act^\text{entry}_2$</td>
</tr>
<tr>
<td>do/ $act^\text{do}_1$</td>
<td>do/ $act^\text{do}_2$</td>
</tr>
<tr>
<td>exit/ $act^\text{exit}_1$</td>
<td>exit/ $act^\text{exit}_2$</td>
</tr>
<tr>
<td>$E_1/act_{E_1}$</td>
<td>$E_2/act_{E_2}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$E_n/act_{E_n}$</td>
<td>...</td>
</tr>
</tbody>
</table>

- **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...?
The Concept of History, and Other Pseudo-States
What happens on...

- $R_s$?
  - $s_0, s_2$

- $R_d$?
  - $s_0, s_2$

- $A, B, C, S, R_s$?
  - $s_0, s_1, s_2, s_3, s_4, s_5, \text{susp}, s_3$

- $A, B, C, S, R_d$?
  - $s_0, s_1, s_2, s_3, s_5, \text{susp}, s_5$

- $A, B, C, D, E, R_s$?
  - $s_0, s_1, s_2, s_3, s_4, s_5, \text{susp}, s_4, E, S$

- $A, B, C, D, R_d$?
  - $s_0, s_1, s_2, s_3, s_4, s_5, \text{susp}, s_5$, deep, shallow
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**

Note: not so sure about naming and symbols, e.g., I’d guessed it was just the other way round...
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, 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.
Contemporary UML Modelling Tools
References
References


