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

(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

```plaintext
write
```

```plaintext
X/
```

```plaintext
resigned
```

```plaintext
n
e
s
w
```

```plaintext
X/
```

```plaintext
X/
```

```plaintext
X/
```

```plaintext
X/
```

```plaintext
X/
```

```plaintext
X/
```

```plaintext
X/
```

```plaintext
w

```plaintext
n

```plaintext
s

```plaintext

```plaintext

```plaintext
```

```plaintext
```
Composite States

and instead of

\[ \text{write} \]

\[ F/ \]

\[ w \]

\[ e \]

\[ n \]

\[ fE \]

\[ fastN \]

\[ fS \]

\[ fW \]
Recall: Syntax

translates to

\[
(q, init) \\
\{(top, st), (s, st), (s_1, st)(s'_1, st)(s_2, st)(s'_2, st)(s_3, st)(s'_3, st)\}
\]

\[
S, kind \\
\{top \mapsto \{s\}, s \mapsto \{\{s_1, s'_1\}, \{s_2, s'_2\}, \{s_3, s'_3\}\}, s_1 \mapsto \emptyset, s'_1 \mapsto \emptyset, \ldots\}
\]

\[
\rightarrow, \psi, annot
\]
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?
- 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?
- ...

---

- \( s_1 \)
- \( s_2 \)
- \( s_3 \)
- \( s_4 \)
- \( s_5 \)
- \( s_6 \)
- \( s_7 \)
- \( s_8 \)

- \( E/ \), \( F/ \), \( G/ \), \([true]/\)
OLD: \( st : S \) — set of states

\( st = S \)

NEW: \( st : 2^S \) — sets of states

\( st = \{ s, \text{top}\} \)

\( st = \{ s_3, s_1, \text{top}\} \)

\( st = \{ \{ s_5, s_4 \} \quad \text{NO} \}
\)

\( = \{ s_6, s_4 \} \quad \}

\( = \{ s_5, s_6, s_4, \text{top}\} \)

\( st = \{ s_2, s_6\} \quad \text{INCONSISTENT} \)
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 $\emptyset \neq R \in \text{region}(s)$, exactly one (non pseudo-state) child of $s$ (from $R$) is in $S_1$, i.e.
    $$|\{s_0 \in R \mid \text{kind}(s_0) \in \{st, fin\}\} \cap S_1| = 1.$$
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,

  \[
  \begin{array}{c}
  s_1 \\
  \downarrow \\
  s_2 \\
  \downarrow \\
  s_3 \\
  \downarrow \\
  \hline
  \text{fork} \\
  \downarrow \\
  \text{join} \\
  \downarrow \\
  \text{tr[gd]/act} \\
  \downarrow \\
  \hline
  \end{array}
  \]

  translates to

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

- Naming convention: \( \psi(t) = (\text{source}(t), \text{target}(t)) \).
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 \( s'' \leq s' \).

\[ s \ \overset{\leq}{\rightarrow} s' = s'' \ \overset{\leq}{\rightarrow} s' \leq s'' \lor s'' \leq s' \]
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 \( lca : 2^S \setminus \{\emptyset\} \rightarrow S \) 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: \( \text{top} \)).
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 \(t \in \rightarrow\),

1. source and destination are consistent, i.e. \(\downarrow \text{source}(t)\) and \(\downarrow \text{target}(t)\),
2. source (and destination) states are pairwise orthogonal, i.e.
   - for all \(s, s' \in \text{source}(t)\) \((\in \text{target}(t))\), \(s \perp s'\),
3. 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.
A hierachical 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.
- \text{forall } 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:
The Depth of States

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

Example:

[Diagram showing the depth of states with nodes labeled from $s_1$ to $s_8$ and edges indicating the depth transitions.]
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 $T \subseteq \rightarrow$ 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 $(\sigma, \varepsilon) \xrightarrow{(\text{cons}, Snd)} (\sigma', \varepsilon')$ 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 $Snd$ 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.

$\leadsto$ 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). \( E_1, \ldots, E_n \in \mathcal{E} \), ‘entry’, ‘do’, ‘exit’ are reserved names!

- Recall: each action’s supposed to have a transformer. Here: \( t_{act_1^{entry}}, t_{act_1^{exit}}, \ldots \)

- Taking the transition above then amounts to applying

\[
 t_{act_{s_2}^{entry}} \circ t_{act} \circ t_{act_{s_1}^{exit}}
\]

instead of only

\[
 t_{act}
\]

\( \rightsquigarrow \) adjust (2.), (3.) accordingly.

<table>
<thead>
<tr>
<th>State ( s_1 )</th>
<th>( \text{entry}/act_1^{entry} )</th>
<th>( \text{do}/act_1^{do} )</th>
<th>( \text{exit}/act_1^{exit} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_2 )</td>
<td>( \text{entry}/act_2^{entry} )</td>
<td>( \text{do}/act_2^{do} )</td>
<td>( \text{exit}/act_2^{exit} )</td>
</tr>
</tbody>
</table>

\( \text{tr}[gd]/act \)
For **internal transitions**, taking the one for $E_1$, for instance, still amounts to taking only $t_{actE_1}$.

- Intuition: The state is neither left nor entered, so: no exit, no entry.

  $\leadsto$ 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 ...

- 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

- **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
References


