Contents & Goals

Last Lecture:
- 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])

- In a sense, composite states are about abbreviation, structuring, and avoiding redundancy.

- Idea: in Tron, for the Player’s Statemachine, instead of
**Composite States**

and instead of

\[
\text{fastN}
\]

\[
\text{fE}
\]

\[
\text{fW} \rightarrow \text{w} \rightarrow \text{e}
\]

\[
\text{fS}
\]

\[
\text{write}
\]

Recall: Syntax

translates to

\[
\{(\text{top}, \text{st}), (s, \text{st}), (s_1, \text{st}), (s_1', \text{st}), (s_2, \text{st}), (s_2', \text{st}), (s_3, \text{st}), (s_3', \text{st})\},
\]

\[
S, \text{kind}
\]

\[
\{\text{top} \mapsto \{s\}, s \mapsto \{\{s_1, s_1'\}, \{s_2, s_2'\}, \{s_3, s_3'\}\}, s_1 \mapsto \emptyset, s_1' \mapsto \emptyset, \ldots\},
\]

\[
\text{region}
\]

\[
\rightarrow, \psi, \text{annot}
\]
**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,

  \[
  s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_6 \rightarrow s_7
  \]

  translates to

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

- Naming convention: \(\psi(t) = (\text{source}(t), \text{target}(t))\).

---

**Composite States: Blessing or Curse?**

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

---

**SPECIAL CASE:**

- \(E\)
- map to \(t'\) (33, 35), \(s'\)
**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?
- ...

\[ \text{States:} \]
- \( s_1 \) \[ \text{E/} \]
- \( s_2 \) \[ \text{F/} \]
- \( s_3 \) \[ \text{F/} \]
- \( s_4 \) \[ \text{F/} \]
- \( s_5 \) \[ \text{E/} \]
- \( s_6 \) \[ \text{G/} \]

\[ \text{Transitions:} \]
- \( \text{d} = \{ s_4 \} \]
- \( \text{d} = \{ s_3, s_4 \} \)
- \( \text{d} = \{ s_1, s_2, s_3 \} \)
- \( \text{d} = \{ s_1, s_2, \ldots, s_4 \} \)
- \( \text{d} = \{ s_4, s_5 \} \) \( \text{NO!} \)
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.
    $$|\{s_0 \in R | \text{kind}(s_0) \in \{st, fin\} \} \cap S_1| = 1.$$ 

Examples:

<table>
<thead>
<tr>
<th>$s$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$s_1$, $s_2$, $s_3$</td>
<td>NOT LEGAL, $top$ missing</td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td>$s_1$, $s_1$, $s_3$</td>
<td>NOT LEGAL, missing child of $s$</td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td>$s_1$, $s_1$, $s_2$</td>
<td>NOT LEGAL, too many children of $s$</td>
<td></td>
</tr>
<tr>
<td>$S_4$</td>
<td>$s_1$, $s_1$, $s_3$</td>
<td>LEGAL</td>
<td></td>
</tr>
</tbody>
</table>

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

Examples:

<table>
<thead>
<tr>
<th>$s$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$s_1$, $s_2$, $s_3$</td>
<td>NOT LEGAL, child of $s$ is missing</td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td>$s_1$, $s_1$, $s_3$</td>
<td>NOT LEGAL, child of $s$ for $R_3$ missing</td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td>$s_1$, $s_1$, $s_2$</td>
<td>NOT LEGAL, child of $s$ for $R_3$ missing</td>
<td></td>
</tr>
<tr>
<td>$S_4$</td>
<td>$s_1$, $s_1$, $s_3$</td>
<td>LEGAL</td>
<td></td>
</tr>
</tbody>
</table>
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).

\[
\begin{array}{c}
s_1 \quad s_2 \quad s_3 \\
\end{array}
\]

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.
    \[
    \exists s, \text{region}(s) = \{S_1, \ldots, S_n\} \ \exists 1 \leq i \neq j \leq n : s_i \in \text{child}^*(S_i) \wedge s_j \in \text{child}^*(S_j),
    \]

\[
\begin{array}{c}
s_1 \quad s_2 \quad s_3 \\
\end{array}
\]
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 $t \in \rightarrow$,

\[
\begin{align*}
\text{(i) } & \text{source and destination are consistent, i.e. } \downarrow \text{source}(t) \text{ and } \downarrow \text{target}(t), \\
\text{(ii) } & \text{source (and destination) states are pairwise orthogonal, i.e.} \\
& \hspace{1em} \text{for all } s, s' \in \text{source}(t) (\in \text{target}(t)), s \perp s', \\
\text{(iii) } & \text{the top state is neither source nor destination, i.e.} \\
& \hspace{1em} \text{top} \notin \text{source}(t) \cup \text{source}(t).
\end{align*}
\]

- Recall: final states are not sources of transitions.
**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.

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

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

\[ \text{top} \not\in \text{source}(t) \cup \text{source}(t). \]

Recall: final states are not sources of transitions.

**Example:**

![Diagram of legal transitions]

---

**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 of the depth of states]
Enabledness in Hierarchical State-Machines

- The scope ("set of possibly affected states") of a transition $t$ is the least common region of
  $$\text{source}(t) \cup \text{target}(t).$$
- 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.
  $$\text{prio}(t) := \max \{ \text{depth}(s) \mid s \in \text{source}(t) \}$$
- 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.
    $$\text{source}(t) \subseteq \sigma(u)(\text{st}) (\subseteq S).$$

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
  - $\sigma'(u)(\text{st})$ 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.

$\Rightarrow$ adjust (2.), (3.), (5.) accordingly.
Entry/Do/Exit Actions, Internal Transitions

- 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: $t_{act_{entry}}$, $t_{act_{exit}}$, \ldots
- Taking the transition above then amounts to applying
  $$t_{act_{entry}} \circ t_{act} \circ t_{act_{exit}}$$
  instead of only $t_{act}$
  $\rightsquigarrow$ adjust (2.), (3.) accordingly.
Internal Transitions

- For **internal transitions**, taking the one for \( E_1 \), for instance, still amounts to taking only \( t_{\text{act}_{E_1}} \).
- Intuition: The state is neither left nor entered, so: no exit, no entry.
  \( \sim \) 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

- \( s_1 \)
  - entry/\( \text{act}_{E_1}^{\text{entry}} \)
  - do/\( \text{act}_{E_1}^{\text{do}} \)
  - exit/\( \text{act}_{E_1}^{\text{exit}} \)
  - \( E_1/\text{act}_{E_1} \)
  - \( \ldots \)
  - \( E_n/\text{act}_{E_n} \)

- \( s_2 \)
  - entry/\( \text{act}_{E_2}^{\text{entry}} \)
  - do/\( \text{act}_{E_2}^{\text{do}} \)
  - exit/\( \text{act}_{E_2}^{\text{exit}} \)

- \( \text{tr}[gd]/\text{act} \)

- \( s_0 \)
  - \( \text{tr}[gd_0]/\text{act}_0 \)
**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**

- **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

History and Deep History: By Example

What happens on...

- $R_s$?

- $R_d$?

- $A, B, C, S, R_s$?

- $A, B, S, R_d$?

- $A, B, C, D, E, R_s$?

- $A, B, C, D, R_d$?
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...

Entry and Exit Point, Submachine State, Terminate

- 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.
References


