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 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 conceptually 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 s2),
  - (iii) remove event from the ether,
  - (iv) after having finished entry and do action of current state (here: s2) — 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: (s2, s3)).
  - 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 ⇝ adjust (2.
  - and 3.
  - in the semantics accordingly

One consequence:

• u never “survives” reaching a state (s, fin) with s ∈ child (top).
In a sense, composite states are about abbreviation, structuring, and avoiding redundancy.

Idea: in Tron, for the Player's Statemachine, instead of

\[ \text{we resigned} \]

write

\[ \text{we slow fast} \]

Recall: Syntax

\[
\begin{align*}
\{ & (\text{top, st}), (s, st), (s_1, st), (s_2, st), (s_3, st), (s_4, st), (s_5, st), (s_6, st), \\
& (s_7, st), (s_8, st), (s_9, st), (s_10, st), (s_11, st), (s_12, st), (s_13, st), (s_14, st), \\
& (s_15, st), (s_16, st), (s_17, st), (s_18, st), (s_19, st), (s_20, st), (s_21, st), (s_22, st), \\
& (s_23, st), (s_24, st), (s_25, st), (s_26, st), (s_27, st), (s_28, st), (s_29, st), (s_30, st), \\
& (s_31, st), (s_32, st), (s_33, st), (s_34, st), (s_35, st), (s_36, st), (s_37, st), (s_38, st), \\
& (s_39, st), (s_40, st), (s_41, st), (s_42, st), (s_43, st), (s_44, st), (s_45, st), (s_46, st), \\
& (s_47, st), (s_48, st), (s_49, st), (s_50, st), (s_51, st), (s_52, st), (s_53, st), (s_54, st), \\
& (s_55, st), (s_56, st), (s_57, st), (s_58, st), (s_59, st), (s_60, st), (s_61, st), (s_62, st), \\
& (s_63, st), (s_64, st), (s_65, st), (s_66, st), (s_67, st), (s_68, st), (s_69, st), (s_70, st), \\
& (s_71, st), (s_72, st), (s_73, st), (s_74, st), (s_75, st), (s_76, st), (s_77, st), (s_78, st), \\
& (s_79, st), (s_80, st), (s_81, st), (s_82, st), (s_83, st), (s_84, st), (s_85, st), (s_86, st), \\
& (s_87, st), (s_88, st), (s_89, st), (s_90, st), (s_91, st), (s_92, st), (s_93, st), (s_94, st), \\
& (s_95, st), (s_96, st), (s_97, st), (s_98, st), (s_99, st), (s_{100}, st)
\}
\]

\[
\begin{align*}
\text{Syntax Function: } & F \text{ or } \text{fork/join} \\
\quad \text{For brevity, we always consider transitions with (possibly) multiple sources and targets, i.e.} \\
\quad \psi : (\rightarrow) \rightarrow (2^S \times \emptyset) \times (2^S \times \emptyset) \\
\quad \text{For instance,} \\
\quad (s_1, s_2, s_3, s_4, s_5, s_6) \text{ translates to} \\
\quad (S, \text{kind}, \text{region}, \{t_1\}) \\
\quad \psi \mapsto (\text{source}(t_1), \text{target}(t_1)) \\
\end{align*}
\]
Leaves, Common Ancestor, and Tree

Least Common Ancestor and Tree

Leaves, Common Ancestor, and Tree

Least Common Ancestor and Tree

A Final Note on States

Composite States: Blessing or Curse?
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
- \( s \perp s' \).

A hierarchical state-machine \((S, \text{kind}, \text{region}, \to, \psi, \text{annot})\) is called **well-formed** if and only if for all transitions \( t \in \to \),

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.
   - \( \forall s, s' \in \text{source}(t) \) \( s \perp s' \),
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.

**Example:**

\[ \begin{array}{cccc}
s_1 & s_2 & s_3 & s_4 \\
E & F & E & G \\
\end{array} \]

\[ \begin{array}{cccc}
s_5 & s_6 & s_7 \\
[ \text{true} ] & F & F \\
\end{array} \]