Software Design, Modelling and Analysis in UML

Lecture 13: Core State Machines III

2016-12-15

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Recall: Transitions of UML State Machines

- discard event,
- dispatch event,
- continue RTC,
- environment interaction,
- error condition.

Example Revisited

Initial States

Rhapsody Demo III: Model Animation

Create and Destroy Transformers
Recall: Transition Relation
**Definition.** Let \( \mathcal{I}_0 = (\mathcal{T}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E}) \) be a signature with signals (all classes in \( \mathcal{C}_0 \) active), \( \mathcal{D}_0 \) a structure of \( \mathcal{I}_0 \), and \( (Eth, ready, \oplus, \ominus, [\cdot]) \) an ether over \( \mathcal{I}_0 \) and \( \mathcal{D}_0 \). Assume there is one core state machine \( M_C \) per class \( C \in \mathcal{C} \).

We say, the state machines **induce** the following labelled transition relation on states \( S := (\Sigma \mathcal{D} \times Eth) \cup \{\#\} \) with labels \( A := 2\mathcal{D}(\mathcal{E}) \times 2(\mathcal{D}(\mathcal{E}) \cup \{*,+\}) \times \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{C}) \):

- \((\sigma, \varepsilon) \xrightarrow{(cons,Snd)} u \) \((\sigma', \varepsilon')\) if and only if
  (i) an event with destination \( u \) is **discarded**,
  (ii) an event is **dispatched** to \( u \), i.e. stable object processes an event, or
  (iii) run-to-completion processing by \( u \) **continues**, i.e. object \( u \) is not stable and continues to process an event,
  (iv) the **environment** interacts with object \( u \),

- \( s \xrightarrow{(cons,\emptyset)} \# \) if and only if
  (v) an **error condition** occurs during consumption of \( cons \), or

\[ s = \# \text{ and } cons = \emptyset \]
(i) Discarding An Event

\[(\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})}_{u} (\sigma', \varepsilon')\]

if

- an \(E\)-event (instance of signal \(E\)) is ready in \(\varepsilon\) for object \(u\) of a class \(C\), i.e. if

\[u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \land \exists u_E \in \mathcal{D}(E) : u_E \in \text{ready}(\varepsilon, u)\]

- \(u\) is stable and in state machine state \(s\), i.e. \(\sigma(u)(\text{stable}) = 1\) and \(\sigma(u)(\text{st}) = s\),

- but there is no corresponding transition enabled (all transitions incident with current state of \(u\) either have other triggers or the guard is not satisfied)

\[\forall (s, F, expr, act, s') \in \rightarrow (SMC) : F \neq E \lor I[expr](\sigma, u) = 0\]

and

- in the system configuration, stability may change, \(u_E\) goes away, i.e.

\[\sigma' = \sigma[u.\text{stable} \mapsto b] \setminus \{u_E \mapsto \sigma(u_E)\}\]

where \(b = 0\) if and only if there is a transition with trigger ‘\_' enabled for \(u\) in \((\sigma', \varepsilon')\).

- the event \(u_E\) is removed from the ether, i.e.

\[\varepsilon' = \varepsilon \oplus u_E,\]

- consumption of \(u_E\) is observed, i.e.

\[\text{cons} = \{u_E\}, \quad \text{Snd} = \emptyset.\]
Example: Discard

$\text{SM}_C$:

\[ [x > 0]/x := x - 1; n! J \]

\[ G[x > 0]/x := y \]

\[ H/z := y/x \]

\[ \langle \langle \text{signal}, \text{env} \rangle \rangle \]

\[ H \]

\[ \langle \langle \text{signal} \rangle \rangle \]

\[ G, J \]

\[ n \]

\[ C \]

\[ 0, 1 \]

\[ x, z : \text{Int} \]

\[ y : \text{Int} \]

\[ \langle \langle \text{env} \rangle \rangle \]

\[ \sigma : \]

\[ \begin{array}{l}
  c : C \\
  x = 1, z = 0, y = 2 \\
  st = s_1 \\
  stable = 1
\end{array} \]

\[ \varepsilon : \]

\[ \langle \langle c, u_J : J \rangle \rangle, \langle \langle c, u_G : G \rangle \rangle \]

\[ : \sigma' \]

\[ \begin{array}{l}
  c : C \\
  x = , z = , y = \\
  st = \\
  stable =
\end{array} \]

\[ : \varepsilon' \]

- $u \in \text{dom}(\sigma) \cap \mathcal{D}(C)$
- $u_E \in \mathcal{D}(E), u_E \in \text{ready}(\varepsilon, u)$
- $\forall (s, F, \text{expr}, \text{act}, s') \in \rightarrow (\text{SM}_C) : F \neq E \vee I[\text{expr}](\sigma, u) = 0$
- $\sigma(u)(\text{stable}) = 1, \sigma(u)(st) = s,
- \sigma' = \sigma[u.\text{stable} \mapsto b] \setminus \{u_E \mapsto \sigma(u_E)\}$
- $\varepsilon' = \varepsilon \ominus u_E$
- $\text{cons} = \{u_E\}, \text{Snd} = \emptyset$
(ii) Dispatch

\[(\sigma, \varepsilon) \xrightarrow{\text{cons, Snd}} (\sigma', \varepsilon')\]

if

- \(u \in \text{dom}(\sigma) \cap D(C) \land \exists u_E \in D(E) : u_E \in \text{ready}(\varepsilon, u)\)
- \(u\) is stable and in state machine state \(s\), i.e. \(\sigma(u)(\text{stable}) = 1\) and \(\sigma(u)(\text{st}) = s\),
- a transition is enabled, i.e.

\[\exists (s, F, \text{expr}, \text{act}, s') \in \rightarrow (SM_C) : F = E \land I[expr](\tilde{\sigma}, u) = 1\]

where \(\tilde{\sigma} = \sigma[u.params_E \mapsto u_E]\).

and

- \((\sigma', \varepsilon')\) results from applying \(t_{\text{act}}\) to \((\sigma, \varepsilon)\) and removing \(u_E\) from the ether, i.e.

\[(\sigma'', \varepsilon') \in t_{\text{act}}[u](\tilde{\sigma}, \varepsilon \oplus u_E),\]

\[\sigma' = (\sigma''[u.st \mapsto s', u.stable \mapsto b, u.params_E \mapsto \emptyset])|D(C)\{u_E\}\]

where \(b\) depends (see (i))

- Consumption of \(u_E\) and the side effects of the action are observed, i.e.

\[\text{cons} = \{u_E\}, \quad \text{Snd} = \text{Obs}_{t_{\text{act}}}[u](\tilde{\sigma}, \varepsilon \oplus u_E)\].
Example: Dispatch

\[ SMC: \]

\[
\begin{array}{c}
\sigma:
\begin{array}{c}
c : C \\
x = 1, z = 0, y = 2 \\
st = s_1 \\
stable = 1
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\varepsilon:
\begin{array}{c}
(c, u_G : G)
\end{array}
\end{array}
\]

\[
\begin{array}{c}
s_1 \xrightarrow{G[x > 0] / x := y} s_2 \\
H / z := y / x
\end{array}
\]

\[
\begin{array}{c}
\langle \langle signal, env \rangle \rangle \\
H
\end{array}
\]

\[
\begin{array}{c}
G, J
\end{array}
\]

\[
\begin{array}{c}
C \\
0, 1 \\
x, z : \text{Int} \\
y : \text{Int} \langle \langle env \rangle \rangle
\end{array}
\]

\[
\begin{array}{c}
\sigma':
\begin{array}{c}
c : C \\
x = , z = , y = \\
st = \\
stable =
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\varepsilon':
\end{array}
\]

- \( u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \)
- \( u_E \in \mathcal{D}(E), u_E \in \text{ready}(\varepsilon, u) \)
- \( \exists (s, F, expr, act, s') \in \rightarrow (SM_C) : F = E \land I[expr](\tilde{\sigma}, u) = 1 \)
- \( \tilde{\sigma} = \sigma[u.\text{params}_E \mapsto u_E]. \)
- \( \sigma(u)(\text{stable}) = 1, \sigma(u)(st) = s, \)
- \( (\sigma'', \varepsilon') = t_{\text{act}}(\tilde{\sigma}, \varepsilon \ominus u_E) \)
- \( \sigma' = (\sigma''[u.st \mapsto s', u.\text{stable} \mapsto b, u.\text{params}_E \mapsto \emptyset])|\mathcal{D}(\varepsilon)\setminus\{u_E\} \)
- \( \text{cons} = \{u_E\}, \quad \text{Snd} = \text{Obs}_{\text{act}}[u](\tilde{\sigma}, \varepsilon \ominus u_E) \)
(iii) Continue Run-to-Completion

\[(\sigma, \varepsilon) \xrightarrow{_{u}^{(cons, Snd)}} (\sigma', \varepsilon')\]

if

- there is an unstable object \(u\) of a class \(C\), i.e.
  \[u \in \text{dom}(\sigma) \cap D(C) \land \sigma(u)(\text{stable}) = 0\]

and

- there is a transition without trigger enabled from the current state \(s = \sigma(u)(st)\), i.e.
  \[\exists (s, \_, \text{expr}, \text{act}, s') \in \rightarrow (SM_C) : I[\text{expr}](\sigma, u) = 1\]

and

- \((\sigma', \varepsilon')\) results from applying \(t_{act}\) to \((\sigma, \varepsilon)\), i.e.
  \[(\sigma'', \varepsilon') \in t_{act}[u](\sigma, \varepsilon), \quad \sigma' = \sigma''[u.st \mapsto s', u.\text{stable} \mapsto b]\]
  where \(b\) depends as before.

- Only the side effects of the action are observed, i.e. \(cons = \emptyset, \quad Snd = Obst_{act}[u](\sigma, \varepsilon)\).

- \(\sigma' = \sigma[u.\text{stable} \mapsto 1], \varepsilon' = \varepsilon, cons = \emptyset, Snd = \emptyset, \text{otherwise}\).
Example: Continue

\[ \text{SM}_C: \]

\[ s_1 \rightarrow \text{G}[x > 0]/x := y \rightarrow s_2 \]

\[ H/z := y/x \]

\[ n \]

\[ \begin{array}{c}
\langle \langle \text{signal}, \text{env} \rangle \rangle \\
H
\end{array} \]

\[ \begin{array}{c}
\langle \langle \text{signal} \rangle \rangle \\
G, J
\end{array} \]

\[ \begin{array}{c}
\langle \langle \text{env} \rangle \rangle \\
x, z : \text{Int} \\
y : \text{Int} \\
0, 1
\end{array} \]

\[ \sigma: \]

\[ c : C \]

\[ x = 2, z = 0, y = 2 \]

\[ \text{st} = s_2 \]

\[ \text{stable} = 0 \]

\[ \varepsilon: \]

\[ \sigma': \]

\[ c : C \]

\[ x = , z = , y = \]

\[ \text{st} = \]

\[ \text{stable} = \]

- \( u \in \text{dom}(\sigma) \cap \mathcal{D}(C), \sigma(u)(\text{stable}) = 0 \)
- \( \exists (s, _, \text{expr}, \text{act}, s') \in \rightarrow (\text{SM}_C) : \)
  \[ I[\text{expr}](\sigma, u) = 1 \]
- \( \sigma(u)(\text{st}) = s, \)
- \( (\sigma'', \varepsilon') = t_{\text{act}}(\sigma, \varepsilon), \)
- \( \sigma' = \sigma''[u.\text{st} \mapsto s', u.\text{stable} \mapsto b] \)
- \( \text{cons} = \emptyset, \quad \text{Snd} = \text{Obst}_{\text{act}}(\sigma, \varepsilon) \)
Assume that a set $\mathcal{E}_{env} \subseteq \mathcal{E}$ is designated as environment events and a set of attributes $V_{env} \subseteq V$ is designated as input attributes.

Then

$$(\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})} \mathcal{E}_{env} (\sigma', \varepsilon')$$

if either (!)

- an environment event $E \in \mathcal{E}_{env}$ is spontaneously sent to an alive object $u \in \text{dom}(\sigma)$, i.e.

  $$\sigma' = \sigma \cup \{ u_E \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n \}, \varepsilon' = \varepsilon \oplus (u, u_E) \}$$

  where $u_E \notin \text{dom}(\sigma)$ and $\text{atr}(E) = \{ v_1, \ldots, v_n \}$.

- Sending of the event is observed, i.e. $\text{cons} = \emptyset, \text{Snd} = \{ u_E, \} \}$. 

or

- Values of input attributes change freely in alive objects, i.e.

  $$\forall v \in V \forall u \in \text{dom}(\sigma) : \sigma'(u)(v) \neq \sigma(u)(v) \implies v \in V_{env}.$$ 

  and no objects appear or disappear, i.e. $\text{dom}(\sigma') = \text{dom}(\sigma)$.

- $\varepsilon' = \varepsilon$. 
Example: Environment

\( \mathcal{SM}_C: \)

\[ \begin{array}{cccc}
\text{SM}_C: & S_1 & G[x > 0]/x := y & S_2 \\
\text{SM}_C: & H/z := y/x \\
\end{array} \]

\[ [x > 0]/x := x - 1; n! J \]

\( \text{SM}_C: \)

\[ \begin{array}{cccc}
\text{σ:} & c : C \\
\text{σ:} & x = 0, z = 0, y = 2 \\
\text{σ:} & st = s_2 \\
\text{σ:} & stable = 1 \\
\end{array} \]

\[ \text{σ':} \]

\[ \begin{array}{cccc}
\text{σ':} & c : C \\
\text{σ':} & x = , z = , y = \\
\text{σ':} & st = \\
\text{σ':} & stable = \\
\end{array} \]

\[ \begin{array}{cccc}
\text{ε:} & \text{ε} \\
\text{ε:} & \text{ε} \\
\text{ε:} & \text{ε} \\
\text{ε:} & \text{ε} \\
\end{array} \]

- \( \sigma' = \sigma \cup \{u_E \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\} \) 
- \( u \in \text{dom}(\sigma) \)
- \( \epsilon' = \epsilon \oplus u_E \) where \( u_E \notin \text{dom}(\sigma) \) and \( \text{atr}(E) = \{v_1, \ldots, v_n\} \).
- \( \epsilon' = \epsilon \oplus u_E \) where \( u_E \notin \text{dom}(\sigma) \) and \( \text{atr}(E) = \{v_1, \ldots, v_n\} \).
- \( u \in \text{dom}(\sigma) \)
- \( \text{cons} = \emptyset, \text{Snd} = \{((env, E(d))\} \).
(v) Error Conditions

\[ s \xrightarrow{(cons,Snd)} u \rightarrow \# \]

**if**, in (i), (ii), or (iii),
- \( I[[\text{expr}]] \) is not defined for \( \sigma \) and \( u \), or
- \( t_{act}[u] \) is not defined for \((\sigma, \varepsilon)\),

**and**
- \( cons = \emptyset \), and \( Snd = \emptyset \).

**Examples:**

- \( E[x/0]/act \) from \( s_1 \) to \( s_2 \)
- \( E[true]/act \) from \( s_1 \) to \( s_3 \)
- \( E[expr]/x := x/0 \) from \( s_1 \) to \( s_2 \)
Example: Error Condition

\[ \begin{align*}
S_{MC}: \quad & [x > 0]/x := x - 1; n ! J \\
& G[x > 0]/x := y \\
& H/z := y/x \\
\end{align*} \]

\[ \begin{align*}
\sigma: \quad & \begin{array}{c}
\begin{array}{c}
c : C \\
x = 0, z = 0, y = 27 \\
st = s_2 \\
stable = 1
\end{array}
\end{array} \\
\end{align*} \]

\[ \begin{align*}
\varepsilon: \quad & \begin{array}{c}
(c, u_H : H)
\end{array}
\end{align*} \]

- \( I[expr] \) not defined for \( \sigma \) and \( u \), or
- \( t_{act}[u] \) is not defined for \( (\sigma, \varepsilon) \)
- \( cons = \emptyset \),
- \( Snd = \emptyset \)
### Example Revisited

#### $SM_C$:

- $E[n \neq \emptyset] / x := x + 1 ; n ! F$
- $F / x := 0$
- $/ n := \emptyset$

#### $SM_D$:

- $\langle \langle \text{signal} \rangle \rangle$
- $\langle \langle \text{signal} \rangle \rangle$

#### Table:

<table>
<thead>
<tr>
<th>Nr.</th>
<th>$x$</th>
<th>$n$</th>
<th>$st$</th>
<th>stable</th>
<th>$p$</th>
<th>$st$</th>
<th>stable</th>
<th>$\varepsilon$</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>27</td>
<td>$5_D$</td>
<td>$s_1$</td>
<td>1</td>
<td>$1_C$</td>
<td>$s_1$</td>
<td>1</td>
<td>$(3_F, 1_C). (2_E, 1_C)$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>27</td>
<td>$5_D$</td>
<td>$s_1$</td>
<td>1</td>
<td>$1_C$</td>
<td>$s_1$</td>
<td>1</td>
<td>$(2_E, 1_C)$</td>
<td>$(i)$</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>$5_D$</td>
<td>$s_2$</td>
<td>0</td>
<td>$1_C$</td>
<td>$s_1$</td>
<td>1</td>
<td>$(4_F, 5_D)$</td>
<td>$(ii)$</td>
</tr>
<tr>
<td>3a</td>
<td>28</td>
<td>$\emptyset$</td>
<td>$s_3$</td>
<td>1</td>
<td>$1_C$</td>
<td>$s_1$</td>
<td>1</td>
<td>$(4_F, 5_D)$</td>
<td>$(iii)$</td>
</tr>
<tr>
<td>3b</td>
<td>28</td>
<td>$5_D$</td>
<td>$s_2$</td>
<td>0</td>
<td>$1_C$</td>
<td>$s_2$</td>
<td>0</td>
<td>$\varepsilon$</td>
<td>$(i)$</td>
</tr>
<tr>
<td>4b1</td>
<td>28</td>
<td>$\emptyset$</td>
<td>$s_3$</td>
<td>1</td>
<td>$1_C$</td>
<td>$s_2$</td>
<td>0</td>
<td>$\varepsilon$</td>
<td>$(ii)$</td>
</tr>
<tr>
<td>4b2</td>
<td>28</td>
<td>$5_D$</td>
<td>$s_2$</td>
<td>0</td>
<td>$1_C$</td>
<td>$s_1$</td>
<td>1</td>
<td>$(3_F, 1_C)$</td>
<td>$(iii)$</td>
</tr>
</tbody>
</table>
**Definition.** Let $A$ be a set of **labels** and $S$ a (not necessarily finite) set of **states**. We call

$$\rightarrow \subseteq S \times A \times S$$

a (labelled) **transition relation**.

Let $S_0 \subseteq S$ be a set of **initial states**. A (finite or infinite) sequence

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$$

with $s_i \in S$, $a_i \in A$ is called **computation** of the **labelled transition system** $(S, A, \rightarrow, S_0)$ if and only if

- **initiation**: $s_0 \in S_0$
- **consecution**: $(s_i, a_i, s_{i+1}) \in \rightarrow$ for $i \in \mathbb{N}_0$. 
Step and Run-to-Completion
Notions of Steps: The Step

**Note:** we call one evolution

\[(\sigma, \varepsilon) \xrightarrow{\text{cons}, \text{Snd}}_u (\sigma', \varepsilon')\]

a step.

Thus in our setting, a step often\(^1\) directly corresponds to one object (namely \(u\)) taking a single transition between regular states.

(We will extend the concept of “single transition” for hierarchical state machines.)

\(^1\): In case of dispatch and continue with enabled transition.

**That is:** We’re going for an interleaving semantics without true parallelism.
What is a run-to-completion step...?

- **Intuition**: a maximal sequence of steps of one object, where the first step is a dispatch step, all later steps are continue steps, and the last step establishes stability (or object disappears).

**Note**: while one step corresponds to one transition in the state machine, a run-to-completion step is in general not syntactically definable:

one transition may be taken multiple times during an RTC-step.

**Example:**

\[
\begin{align*}
E[x > 0]/ & \quad [x > 0]/x := x - 1 \\
S_1 & \rightarrow S_2 \\
\sigma: & \\
\begin{array}{c}
\Downarrow \text{table} = 1 \\
\Downarrow \text{st} = s_1 \\
\end{array} \\
\begin{array}{l}
C \\
x = 2
\end{array} \\
\varepsilon: & \\
\text{E for } u
\end{align*}
\]
Proposal: Let

\[(\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, S\text{nd}_0)} \ldots \xrightarrow{(\text{cons}_{n-1}, S\text{nd}_{n-1})} (\sigma_n, \varepsilon_n), \quad n > 0,\]

be a finite (!), non-empty, maximal, consecutive sequence such that

- \((\text{cons}_0, S\text{nd}_0)\) indicates dispatching to \(u := u_0\) (by Rule (ii)),
  i.e. \(\text{cons} = \{u_E\}, u_E \in \text{dom}(\sigma_0) \cap \mathcal{D}(\mathcal{E})\),

- if \(u\) becomes stable or disappears, then in the last step, i.e.

  \[
  \forall i > 0 \bullet (\sigma_i(u)(\text{stable}) = 1 \lor u \notin \text{dom}(\sigma_i)) \implies i = n
  \]

Let \(0 = k_1 < k_2 < \cdots < k_N < n\) be the maximal sequence of indices such that \(u_{k_i} = u\) for \(1 \leq i \leq N\). Then we call the sequence

\[
(\sigma_0(u) =) \quad \sigma_{k_1}(u), \sigma_{k_2}(u) \ldots, \sigma_{k_N}(u), \sigma_n(u)
\]

a (!) run-to-completion step of \(u\) (from (local) configuration \(\sigma_0(u)\) to \(\sigma_n(u)\)).
We say, object $u$ can diverge on reception $cons_0$ from (local) configuration $\sigma_0(u)$ if and only if there is an infinite, consecutive sequence

$$(\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(cons_1, Snd_1)} \ldots$$

where $u_i = u$ for infinitely many $i \in \mathbb{N}_0$ and $\sigma_i(u)(stable) = 0$, $i > 0$, i.e. $u$ does not become stable again.
Run-to-Completion Step: Discussion.

Our definition of RTC-step takes a **global** and **non-compositional** view, that is:

- In the projection onto a single object we still **see** the effect of interaction with other objects.
- Adding classes (or even objects) may change the divergence behaviour of existing ones.
- Compositional would be:
  
  the behaviour of a set of objects is determined by the behaviour of each object “in isolation”.
  
  Our semantics and notion of RTC-step doesn’t have this (often desired) property.
Our definition of RTC-step takes a global and non-compositional view, that is:

- In the projection onto a single object we still see the effect of interaction with other objects.
- Adding classes (or even objects) may change the divergence behaviour of existing ones.
- Compositional would be:
  the behaviour of a set of objects is determined by the behaviour of each object “in isolation”.
  Our semantics and notion of RTC-step doesn’t have this (often desired) property.

Can we give (syntactical) criteria such that any (global) run-to-completion step is an interleaving of local ones?
Our definition of RTC-step takes a **global** and **non-compositional** view, that is:

- In the projection onto a single object we still **see** the effect of interaction with other objects.
- Adding classes (or even objects) may change the divergence behaviour of existing ones.
- Compositional would be:
  
  the behaviour of a set of objects is determined by the behaviour of each object “in isolation”.
  
  Our semantics and notion of RTC-step doesn’t have this (often desired) property.

Can we give (syntactical) criteria such that any (global) run-to-completion step is an interleaving of local ones?

**Maybe: Strict interfaces.**

- **(A):** Refer to private features only via “self”.
  
  (Recall that other objects of the same class can modify private attributes.)

- **(B):** Let objects only communicate by events, i.e.
  
  don’t let them modify each other’s local state via links **at all.**

**Proof left as exercise...**
Putting It All Together
**Initial States**

**Recall:** a labelled transition system is \((S, A, \rightarrow, S_0)\).

**We have**

- \(S\): system configurations \((\sigma, \varepsilon)\)
- \(\rightarrow\): labelled transition relation \((\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})} (\sigma', \varepsilon')\).

**Wanted:** initial states \(S_0\).
Recall: a labelled transition system is \((S, A, \rightarrow, S_0)\).

We have

- \(S\): system configurations \((\sigma, \varepsilon)\)
- \(\rightarrow\): labelled transition relation \((\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})} (\sigma', \varepsilon')\).

Wanted: initial states \(S_0\).

Proposal:

Require a (finite) set of object diagrams \(\mathcal{OD}\) as part of a UML model

\[(\mathcal{CD}, \mathcal{SM}, \mathcal{OD})\].

And set

\[S_0 = \{ (\sigma, \varepsilon) \mid \sigma \in G^{-1}(\mathcal{OD}), \quad \mathcal{OD} \in \mathcal{OD}, \quad \varepsilon \text{ empty} \} \].
**Initial States**

**Recall**: a labelled transition system is \((S, A, \rightarrow, S_0)\).

We have

- \(S\): system configurations \((\sigma, \varepsilon)\)
- \(\rightarrow\): labelled transition relation \( (\sigma, \varepsilon) \xrightarrow{\text{cons}, \text{Snd}} (\sigma', \varepsilon') \).

**Wanted**: initial states \(S_0\).

**Proposal**:

Require a (finite) set of **object diagrams** \(\mathcal{OD}\) as part of a UML model

\[ (\mathcal{CD}, \mathcal{IM}, \mathcal{OD}). \]

And set

\[ S_0 = \{ (\sigma, \varepsilon) \mid \sigma \in G^{-1}(\mathcal{OD}), \quad \mathcal{OD} \in \mathcal{OD}, \quad \varepsilon \text{ empty} \} . \]

**Other Approach**: (used by Rhapsody tool) multiplicity of classes (plus initialisation code).

We can read that as an abbreviation for an object diagram.
The semantics of the UML model

\[ M = (C_D, SM, OD) \]

where

- some classes in \( C_D \) are stereotyped as ‘signal’ (standard), some signals and attributes are stereotyped as ‘external’ (non-standard),
- there is a 1-to-1 relation between classes and state machines,
- \( OD \) is a set of object diagrams over \( C_D \),

is the transition system \((S, A, \rightarrow, S_0)\) constructed on the previous slide(s).

The computations of \( M \) are the computations of \((S, A, \rightarrow, S_0)\).
OCL Constraints and Behaviour

- Let $\mathcal{M} = (\mathcal{C}D, \mathcal{S}M, \mathcal{O}D)$ be a UML model.

- We call $\mathcal{M}$ consistent iff, for each OCL constraint $expr \in Inv(\mathcal{C}D)$,

  $$\sigma \models expr$$

  for each “reasonable point” $(\sigma, \varepsilon)$ of computations of $\mathcal{M}$.

  (Cf. tutorial for discussion of “reasonable point”.)

**Note:** we could define $Inv(\mathcal{S}M)$ similar to $Inv(\mathcal{C}D)$. 
Let $\mathcal{M} = (C D, S M, O D)$ be a UML model.

- We call $\mathcal{M}$ consistent iff, for each OCL constraint $expr \in Inv(C D)$,
  $$\sigma \models expr$$ for each “reasonable point” $(\sigma, \varepsilon)$ of computations of $\mathcal{M}$.

  (Cf. tutorial for discussion of “reasonable point”.)

**Note:** we could define $Inv(S M)$ similar to $Inv(C D)$.

**Pragmatics:**

- In **UML-as-blueprint mode**, if $S M$ doesn’t exist yet, then providing $\mathcal{M} = (C D, \emptyset, O D)$ is typically asking the developer to provide state machines $S M$ such that $\mathcal{M}' = (C D, S M, O D)$ is consistent.

  If the developer makes a mistake, then $\mathcal{M}'$ is inconsistent.

- **Not so common** (but existing):

  If $S M$ is given, then constraints are also considered when choosing transitions in the RTC-algorithm.

  In other words: even in presence of “mistakes”, the state machines in $S M$ never move to inconsistent configurations.
State Machines induce a labelled transition system.

There are five kinds of transitions in the LTS:
- discard, dispatch, continue, environment, error.

For now, we assume that all classes are active, thus steps of objects may interleave.

We distinguish steps and run-to-completion step.

Initial states can be characterised using object diagrams.

Missing transformers:
- Create: re-use identities vs. use fresh ones.
- Destroy: allow dangling references vs. clean up.
References
References
