Contents & Goals

Last Lecture:
- System configuration
- Transformer
- Action language: skip, update, send

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 is: Signal, Event, Ether, Transformer, Step, RTC.

- Content:
  - Transformers for Action Language
  - Run-to-completion Step
  - Putting It All Together
Transformer: Create

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create(C, expr, v)</code></td>
<td><code>expr \cdot v = \text{new } c</code></td>
</tr>
</tbody>
</table>

**intuitive semantics**

Create an object of class `C` and assign it to attribute `v` of the object denoted by expression `expr`.

**well-typedness**

\[
\text{expr} : \tau_D, \quad v \in \text{atr}(D), \quad \text{atr}(C) = \{ (v_i : \tau_i, \text{expr}^i) \mid 1 \leq i \leq n \}
\]

**semantics**

...  

**observables**

...  

**(error) conditions**

\[ I[\text{expr}](\sigma, \beta) \text{ not defined.} \]

- We use an "and assign"-action for simplicity — it doesn’t add or remove expressive power, but moving creation to the expression language raises all kinds of other problems such as order of evaluation (and thus creation).
- Also for simplicity: no parameters to construction (~ parameters of constructor). Adding them is straightforward (but somewhat tedious).
How To Choose New Identities?

- **Re-use**: choose any identity that is not alive **now**, i.e. not in \( \text{dom}(\sigma) \).
  - Doesn’t depend on history.
  - May “undangle” dangling references – may happen on some platforms.

- **Fresh**: choose any identity that has not been alive **ever**, i.e. not in \( \text{dom}(\sigma) \) and any predecessor in current run.
  - Depends on history.
  - Dangling references remain dangling – could mask “dirty” effects of platform.
**Transformer: Create**

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>create($C$, $expr$, $v$)</td>
<td>$\text{create}(\text{expr}, v)$</td>
</tr>
</tbody>
</table>

**intuitive semantics**

Create an object of class $C$ and assign it to attribute $v$ of the object denoted by expression $expr$.

**well-typedness**

$expr : \tau_D$, $v \in \text{atr}(D)$, $\text{atr}(C) = \{ \langle v_1 : \tau_1, expr_0 \rangle | 1 \leq i \leq n \}$

**semantics**

Id of newly created object:

$((\sigma, \epsilon), (\sigma', \epsilon')) \in t$  
initialization of new object

iff $\sigma' = \sigma[u_0 \mapsto \sigma(u_0)[v \mapsto \epsilon]] \cup \{ u \mapsto \{ v_i \mapsto d_i | 1 \leq i \leq n \} \}$,  
$\epsilon' = [u](\epsilon)$;  
$u \in C$ fresh, i.e. $u \not\in \text{dom}(\sigma)$;

$u_0 = I[expr](\sigma, \beta)$; $d_i = I[expr_i](\sigma, \omega)$ if $expr_i \neq \varepsilon$ and arbitrary value from $C$ otherwise;

**observables**

$\text{Obs}_{create}[u_x] = \{ (u_x, \bot, (+, \emptyset), u) \}$

**error conditions**

$I[expr](\sigma, \beta)$ not defined.

---

**Transformer: Destroy**

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>destroy($expr$)</td>
<td>$\text{destroy}(\text{expr})$</td>
</tr>
</tbody>
</table>

**intuitive semantics**

Destroy the object denoted by expression $expr$.

**well-typedness**

$expr : \tau_C$, $C \in \mathcal{C}$

**semantics**

...  

**observables**

$\text{Obs}_{destroy}[u_x] = \{ (u_x, \bot, (+, \emptyset), u) \}$

**error conditions**

$I[expr](\sigma, \beta)$ not defined.
**Destroy Transformer Example**

\[
\text{SM}_C: \\
\text{s}_1 \quad / \ldots; \text{delete n}; \ldots \quad \text{s}_2
\]

\[
\text{destroy}(expr) \\
\tau_{\text{destroy}(expr)}[u_2](\sigma, \varepsilon) = \ldots
\]

**What to Do With the Remaining Objects?**

Assume object \(u_0\) is destroyed...

- object \(u_1\) may still refer to it via association \(r\):
  - allow dangling references?
  - or remove \(u_0\) from \(\sigma(u_1)(r)\)?
- object \(u_0\) may have been the last one linking to object \(u_2\):
  - leave \(u_2\) alone?
  - or remove \(u_2\) also? (*garbage collection*)
  - Plus: (temporal extensions of) OCL may have dangling references.

**Our choice:** Dangling references and no garbage collection!

This is in line with "expect the worst", because there are target platforms which don’t provide garbage collection — and models shall (in general) be correct without assumptions on target platform.

**But:** the more "dirty" effects we see in the model, the more expensive it often is to analyse. Valid proposal for simple analysis: monotone frame semantics, no destruction at all.
### Transformer: Destroy

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>destroy(expr)</td>
<td></td>
</tr>
</tbody>
</table>

**intuitive semantics**

*Destroy the object denoted by expression expr.*

**well-typedness**

\[
expr : \tau_C, C \in \mathcal{C}
\]

**semantics**

\[
t[u_x](\sigma, \varepsilon) = (\sigma', \varepsilon) \quad \text{function restriction}
\]

where \(\sigma' = \sigma|_{\text{dom}(\sigma) \setminus \{u\}}\) with \(u = I[\text{expr}](\sigma, u)\).

**observables**

\[
\text{Obs}_{\text{destroy}}[u_x] = \{(u_x, \perp, (+, \emptyset), u)\}
\]

**(error) conditions**

\[
I[\text{expr}](\sigma, u) \text{ not defined.}
\]

---

**Sequential Composition of Transformers**

- **Sequential composition** \(t_1 \circ t_2\) of transformers \(t_1\) and \(t_2\) is canonically defined as

\[
(t_2 \circ t_1)[u_x](\sigma, \varepsilon) = t_2[u_x](t_1[u_x](\sigma, \varepsilon))
\]

with observation

\[
\text{Obs}_{(t_2 \circ t_1)[u_x]}(\sigma, \varepsilon) = \text{Obs}_{t_1}[u_x](\sigma, \varepsilon) \cup \text{Obs}_{t_2}[u_x](t_1(\sigma, \varepsilon)).
\]

- **Clear**: not defined if one the two intermediate “micro steps” is not defined.
**Transformers And Denotational Semantics**

**Observation:** our transformers are in principle the denotational semantics of the actions/action sequences. The trivial case, to be precise.

**Note:** with the previous examples, we can capture
- empty statements, skips,
- assignments,
- conditionals (by normalisation and auxiliary variables),
- create/destroy,

but not possibly diverging loops.

**Our (Simple) Approach:** if the action language is, e.g. Java, then (syntactically) forbid loops and calls of recursive functions.

**Other Approach:** use full blown denotational semantics.

No show-stopper, because loops in the action annotation can be converted into transition cycles in the state machine.

---

**Step and Run-to-completion Step**
**Transition Relation, Computation**

**Definition.** Let $A$ be a set of actions 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 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 $(S, \rightarrow, S_0)$ if and only if

- initiation: $s_0 \in S_0$
- consecution: $(s_i, a_i, s_{i+1}) \in \rightarrow$ for $i \in \mathbb{N}_0$.

**Note:** for simplicity, we only consider infinite runs.

---

**Active vs. Passive Classes/Objects**

- **Note:** From now on, assume that all classes are active for simplicity.

We’ll later briefly discuss the Rhapsody framework which proposes a way how to integrate non-active objects.

- **Note:** The following RTC “algorithm” follows [Harel and Gery, 1997] (i.e. the one realised by the Rhapsody code generation) where the standard is ambiguous or leaves choices.
From Core State Machines to LTS

Definition. Let $\mathcal{S}_0 = (\mathcal{S}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$ be a signature with signals (all classes active), $\mathcal{S}_0$ a structure of $\mathcal{S}_0$, and $(\mathcal{E}, \text{ready}, \oplus, \ominus, [\cdot])$ an ether over $\mathcal{S}_0$ and $\mathcal{S}_0$. 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

\[ S := (\sum \mathcal{C} \cup \{\#\} \times \mathcal{E}) \]

with actions $A := (2^{\mathcal{P}(\mathcal{C})} \times (\mathcal{P}(\mathcal{E}) \cup \{\bot\})) \mathcal{E} \times (\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{E})^2)$.

\[ \begin{align*}
(\sigma, \varepsilon) & \xrightarrow{(\text{cons}, \text{Snd})} (\sigma', \varepsilon') \\
\text{if and only if} & \\
(i) & \text{an event with destination } u \text{ is discarded,} \\
(ii) & \text{an event is dispatched to } u, \text{i.e. stable object processes an event, or} \\
(iii) & \text{run-to-completion processing by } u \text{ commences,} \\
& \text{i.e. object } u \text{ is not stable and continues to process an event,} \\
(iv) & \text{the environment interacts with object } u, \\
\text{if and only if} & \\
(v) & \text{cons} = \emptyset, \text{or an error condition occurs during consumption of cons.}
\end{align*} \]

(i) Discarding An Event

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

if

- an $E$-event (instance of signal $E$) is ready in $\varepsilon$ for object $u$ of a class $\mathcal{C}$, i.e. if
  \[ u \in \text{dom}(\sigma) \cap \mathcal{P}(\mathcal{C}) \land \exists u_E \in \mathcal{P}(\mathcal{E}) : u_E \in \text{ready}(\varepsilon, u) \]
- $u$ is stable and in state machine state $s$, i.e. $\sigma(u)(\text{stable}) = 1$ and $\sigma(u)(st) = s$,
- 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)

\[ \forall (s, F, expr, act, s') \in (\mathcal{S}M_C) : F \neq E \lor I[\text{expr}] (\sigma_0) = 0 \]

and

- the system configuration doesn't change, i.e. $\sigma' = \sigma$
- the event $u_E$ is removed from the ether, i.e.
  \[ \varepsilon' = \varepsilon \ominus u_E, \]
- consumption of $u_E$ is observed, i.e.
  \[ \text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset. \]
Example: Discard

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

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

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

\[ s_1 \]

\[ s_2 \]

\[ \sigma : C \]

\[ x = 1, z = 0, y = 2 \]

\[ st = s_1 \]

\[ stable = 1 \]

\[ \varepsilon : \text{nothing sent} \]

\[ \text{id of object which does } \)

\[ \sigma' = \sigma \]

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

\[ C' \]

\[ 0, 1 \]

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

\[ y : \text{Int} \sim \text{env} \]

\[ \sigma' \]

(ii) Dispatch

\((\sigma, \varepsilon) \xrightarrow{\text{cons}, \text{Snd}} u \sigma', \varepsilon'\) if

- \(u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \land \exists u_E \in \mathcal{D}(\varepsilon) : u_E \in \text{ready}(\varepsilon, u)\)
- \(u\) is stable and in state machine state \(s\), i.e. \(\sigma(u)(\text{stable}) = 1\) and \(\sigma(u)(st) = s\),
- a transition is enabled, i.e.
  \[ \exists (s, F, expr, act, s') \in \rightarrow \text{(SM}_C\text{)} : F \neq E \lor I[expr] (\sigma) = 0 \]

where \(\sigma = \sigma[u.\text{params}_E \mapsto u_E]\).

and

- \((\sigma', \varepsilon')\) results from applying \(t_{act}\) to \((\sigma, \varepsilon)\) and removing \(u_E\) from the ether, i.e.
  \[ \sigma''(u.st \mapsto s', u.\text{stable} \mapsto b, u.\text{params}_E \mapsto \emptyset)](\mathcal{D}(\varepsilon) \setminus \{u_E\}) \]

where \(b\) depends:

- If \(u\) becomes stable in \(s'\), then \(b = 1\). It does become stable if and only if there is no transition without trigger enabled for \(u\) in \((\sigma', \varepsilon')\).
- Otherwise \(b = 0\).
- Consumption of \(u_E\) and the side effects of the action are observed, i.e.
  \[ \text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{act}(\sigma, \varepsilon \oplus u_E)\].
Example: Dispatch

\[
\begin{align*}
&\text{SM}: \quad \text{if } x > 0, \text{ then } x := x - 1; n \text{! } J \\
&\text{H}: \quad y \\
&G, J : \quad \text{if } x > 0, \text{ then } x := x - 1
\end{align*}
\]

\[
\begin{align*}
G[z := y/x] : & \quad \text{if } x > 0, \text{ then } x := x - 1; \\
\sigma^1 : & \quad \sigma (\text{stable}) = 1, \sigma (\text{st}) = s, \\
\varepsilon^1 : & \quad \sigma' \text{ results from applying } t_{\text{act}} \text{ to } (\sigma, \varepsilon), \text{ i.e.} \\
& \quad \sigma'' , \varepsilon' \text{ depends as before.}
\end{align*}
\]

(iii) Commence Run-to-Completion

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

if

- there is an unstable object \( u \) of a class \( \mathcal{C} \), i.e.
  \[ u \in \text{dom}(\sigma) \cap \mathcal{P}(C) \]

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

and

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

where \( b \) depends as before.

- Only the side effects of the action are observed, i.e.
  \[ \text{cons} = \emptyset, \text{Snd} = \text{Obs}_{\text{act}}(\sigma, \varepsilon). \]
\section*{Example: Commence}

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

\begin{itemize}
\item \(x = 2, z = 0, y = 2\)
\item \(\text{stable} = 0\)
\end{itemize}

\begin{itemize}
\item \(\exists u \in \text{dom}(\sigma) \cap D(C) : \sigma(u)(\text{stable}) = 0\)
\item \(\exists (s, \text{expr}, \text{act}, s') \in \rightarrow \text{(SMC)} : I[\text{expr}](\sigma) = 1\)
\item \(\sigma(u)(\text{stable}) = 1, \sigma(u)(\text{st}) = s\)
\end{itemize}

\begin{itemize}
\item \((\sigma', \epsilon') = t_{act}(\sigma, \epsilon),\)
\item \(\sigma' = \sigma''[u, \text{st} \mapsto s', u, \text{stable} \mapsto b]\)
\item \(\text{cons} = \emptyset, \text{Snd} = \text{Obs}_{\text{env}}(\sigma, \epsilon)\)
\end{itemize}

\section*{(iv) Environment Interaction}

Assume that a set \(\mathcal{E}_{\text{env}} \subseteq \mathcal{E}\) is designated as \textit{environment events} and a set of attributes \(v_{\text{env}} \subseteq V\) is designated as \textit{input attributes}.

Then

\[
(\sigma, \epsilon) \xrightarrow{(\text{cons, Snd})_{\text{env}}} (\sigma', \epsilon')
\]

if

\begin{itemize}
\item an environment event \(E \in \mathcal{E}_{\text{env}}\) is spontaneously sent to an alive object \(u \in D(\sigma)\), i.e.
\[
\sigma' = \sigma \cup \{u_E \mapsto v_i \mapsto d_i \mid 1 \leq i \leq n\}, \quad \epsilon' = \epsilon \oplus u_E
\]
where \(u_E \notin \text{dom}(\sigma)\) and \(\text{at}(E) = \{v_1, \ldots, v_n\}\).
\item Sending of the event is observed, i.e. \(\text{cons} = \emptyset, \text{Snd} = \{(\text{env}, E(\vec{d}))\}\).
\end{itemize}

or

\begin{itemize}
\item Values of input attributes change freely in alive objects, i.e.
\[
\forall v \in V \forall u \in \text{dom}(\sigma) : \sigma'(u)(v) \neq \sigma(u)(v) \implies v \in V_{\text{env}}.
\]
and no objects appear or disappear, i.e. \(\text{dom}(\sigma') = \text{dom}(\sigma)\).
\item \(\epsilon' = \epsilon\).
\end{itemize}
Example: Environment

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

\[ SM C: \]

\[ s_1 \rightarrow G[x > 0] / x := y \]

\[ H / z := y ^ \langle \varepsilon \rangle \]

\[ s_2 \]

\[ \sigma: \]

\[ c : C \]

\[ \begin{align*}
  x &= 0, z = 0, y = 2 \\
  st &= s_2 \\
  stable &= 1
\end{align*} \]

\[ \varepsilon: \]

\[ \begin{align*}
  \sigma' &= \sigma \cup \{ u_E \mapsto v_i \mapsto d_i \mid 1 \leq i \leq n \} \\
  \varepsilon' &= \varepsilon \oplus u_E \text{ where } u_E \notin \text{dom}(\sigma) \\
  \text{and } \text{atr}(E) &= \{ v_1, \ldots, v_n \} \}
\]

\[ \text{(v) Error Conditions} \]

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

if, in (ii) or (iii),

- \( I\{\text{expr}\} \) is not defined for \( \sigma \), or
- \( t_{\text{act}} \) is not defined for \( (\sigma, \varepsilon) \),

and

- consumption is observed according to (ii) or (iii), but \( Snd = \emptyset \).

Examples:

- \[ s_1 \rightarrow E[x/0] / \text{act} \rightarrow s_2 \]

- \[ E[\text{true}] / \text{act} \rightarrow s_3 \]

- \[ s_1 \rightarrow E[\text{expr}] / x := x/0 \rightarrow s_2 \]
**Example: Error Condition**

$$ [x > 0] / x := y $$

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

$$ H/ z := y/ x $$

$$ C $$

$$ 0.1 $$

$$ x, z : \text{Int} $$

$$ y : \text{Int} \ (\lceil \text{env} \rceil) $$

$$ C $$

$$ \sigma $$

$$ x = 0, z = 0, y = 27 $$

$$ st = s_2 $$

$$ \text{stable} = 1 $$

$$ \varepsilon $$

$$ H \text{ for } c $$

- $I[\text{expr}]$ not defined for $\sigma$, or
- $t_{\text{act}}$ is not defined for $(\sigma, \varepsilon)$
- consumption according to (ii) or (iii)
- $\text{Snd} = \emptyset$

---

**Notions of Steps: The Step**

**Note:** we call one evolution $(\sigma, \varepsilon) \xrightarrow{\text{(cons, Snd)}} (\sigma', \varepsilon')$ a step.

Thus in our setting, a step directly corresponds to

-one object (namely $u$) takes a single transition between regular states.

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

**That is:** We’re going for an interleaving semantics without true parallelism.

**Remark:** With only methods (later), the notion of step is not so clear.

For example, consider
- $c_1$ calls $f()$ at $c_2$, which calls $g()$ at $c_1$ which in turn calls $h()$ for $c_2$.
- Is the completion of $h()$ a step?
- Or the completion of $f()$?
- Or doesn’t it play a role?

It does play a role, because constraints/invariants are typically (= by convention) assumed to be evaluated at step boundaries, and sometimes the convention is meant to admit (temporary) violation in between steps.
Notions of Steps: The Run-to-Completion Step

What is a run-to-completion step...?

- **Intuition**: a maximal sequence of steps, where the first step is a dispatch step and all later steps are commence steps.
- **Note**: 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:**

\[
\begin{array}{c}
\sigma:\ 
\begin{cases}
C \\
x = 2
\end{cases} \\
\varepsilon:\ 
E \text{ for } u
\end{array}
\]

\[
\begin{array}{c}
s_1 \quad E[x > 0]/ \\
/x := x - 1 \quad s_2
\end{array}
\]

Notions of Steps: The Run-to-Completion Step Cont’d

**Proposal**: Let

\[
(\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} u_0 \quad \ldots \quad \xrightarrow{(cons_{n-1}, Snd_{n-1})} u_{n-1} \rightarrow (\sigma_n, \varepsilon_n), \quad n > 0,
\]

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

- object \( u \) is alive in \( \sigma_0 \),
- \( u_0 = u \) and \( (cons_0, Snd_0) \) indicates dispatching to \( u \), i.e. \( cons = \{(u, \bar{v} \mapsto \bar{d})\} \),
- there are no receptions by \( u \) in between, i.e.
  \[
  cons_i \cap \{u\} \times \text{Evs}(\bar{e}, \mathcal{D}) = \emptyset, \quad i > 1,
  \]
- \( u_{n-1} = u \) and \( u \) is stable only in \( \sigma_0 \) and \( \sigma_n \), i.e.
  \[
  \sigma_0(u)(\text{stable}) = \sigma_n(u)(\text{stable}) = 1 \text{ and } \sigma_i(u)(\text{stable}) = 0 \text{ for } 0 < i < n,
  \]

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

\[
(\sigma_0(u) =) \quad \sigma_{k_1}(u), \sigma_{k_2}(u), \ldots, \sigma_{k_N}(u) \quad (= \sigma_{n-1}(u))
\]

a (!) run-to-completion computation of \( u \) (from (local) configuration \( \sigma_0(u) \)).
**Divergence**

We say, object $u$ can diverge on reception $cons$ from (local) configuration $\sigma_0(u)$ if and only if there is an infinite, consecutive sequence

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

such that $u$ doesn’t become stable again.

- **Note**: disappearance of object not considered in the definitions.
  By the current definitions, it’s neither divergence nor an RTC-step.

**Run-to-Completion Step: Discussion.**

What people may dislike on our definition of RTC-step is that it 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**.
The Missing Piece: Initial States

Recall: a labelled transition system is \((S, \rightarrow, S_0)\). 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 \(OD\) as part of a UML model \((\mathcal{OP}, \mathcal{MM}, \mathcal{OD})\).

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

Other Approach: (used by Rhapsody tool) multiplicity of classes. We can read that as an abbreviation for an object diagram.
The semantics of the UML model

\[ M = (\mathcal{C}, \mathcal{M}, \mathcal{D}) \]

where

- some classes in \( \mathcal{C} \) 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,
- \( \mathcal{D} \) is a set of object diagrams over \( \mathcal{C} \),

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

The computations of \( M \) are the computations of \((S, \rightarrow, S_0)\).

OCL Constraints and Behaviour

- Let \( M = (\mathcal{C}, \mathcal{M}, \mathcal{D}) \) be a UML model.
- We call \( M \) consistent iff, for each OCL constraint \( expr \in \text{Inv}(\mathcal{C}) \),
  \[ \sigma \models expr \] for each “reasonable point” \((\sigma, \varepsilon)\) of computations of \( M \).
  (Cf. exercises and tutorial for discussion of "reasonable point").

Note: we could define \( \text{Inv}(\mathcal{M}) \) similar to \( \text{Inv}(\mathcal{C}) \).

Pragmatics:

- In UML-as-blueprint mode, if \( \mathcal{M} \) doesn’t exist yet, then \( M = (\mathcal{C}, \emptyset, \mathcal{D}) \)
  is typically asking the developer to provide \( \mathcal{M} \) such that
  \( M' = (\mathcal{C}, \mathcal{M}, \mathcal{D}) \) is consistent.
  If the developer makes a mistake, then \( M' \) is inconsistent.
- Not common: if \( \mathcal{M} \) is given, then constraints are also considered when choosing transitions in the RTC-algorithm. In other words: even in presence of mistakes, the \( \mathcal{M} \) never move to inconsistent configurations.
References


