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 Cont’d
## 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>exp1.v := 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

\[
expr : \tau_D, \ v \in atr(D), \ atr(C) = \{(v_1^i : \tau_1^i, expr_0^i) | 1 \leq i \leq n\}
\]

### Semantics

...  

### Observables

...  

### (Error) Conditions

\(I[expr](\sigma, \beta)\) 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).

\((\star)\) SO NOT: \(x := (new\ C).x + (new\ C).y\)

\(\text{if needed}\)

\[
\begin{align*}
\text{tup}_1 := \text{new} C; \\
\text{tup}_2 := \text{new} C; \\
x := +\text{tup}_1 x \\
+\text{tup}_2 y \\
\text{tup}_3 := \text{new} C; \\
\text{tup}_4 := \text{new} C;
\end{align*}
\]
Create Transformer Example

$SM_C$:

\[ s_1 \xrightarrow{\ldots; n := \text{new } C; \ldots} s_2 \]

\[
\begin{align*}
\text{create}(C, expr, v) \\
t_{\text{create}(C, expr, v)}(\sigma, \epsilon) &= \ldots
\end{align*}
\]

$\sigma$:

\[
\begin{array}{c|c}
\text{d} & D \\
\hline
n = \emptyset
\end{array}
\]

$\epsilon$:

\[
\begin{array}{c}
\\text{(non-det. choice)} \\
\text{by initial value expression}
\end{array}
\]

$\mathcal{D}(\epsilon)$

\[
\begin{array}{c}
\text{dom } \sigma \\
\text{(non-det. choice)}
\end{array}
\]

$\mathcal{D}(\text{init})$

\[
\begin{array}{c}
\text{(non-det. choice)} \\
\text{: } \sigma'
\end{array}
\]

$\mathcal{D}(\text{init})$

\[
\begin{array}{c}
\text{(non-det. choice)} \\
\text{: } \epsilon'
\end{array}
\]
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></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

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

#### semantics

\[ ((\sigma, \varepsilon), (\sigma', \varepsilon')) \in t \text{ iff } \sigma' = \sigma[u_0 \mapsto \sigma(u_0)[v \mapsto \hat{u}]] \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\} \}, \]

\[ \varepsilon' = [u](\varepsilon) \text{; } u \in D(C) \text{ fresh, i.e. } u \notin \text{dom}(\sigma); \]

\[ u_0 = I[expr](\sigma, \varnothing); d_i = I[expr_0^i](\sigma, \varnothing) \text{ if } expr_0^i \neq \varnothing \text{ and arbitrary value from } D(\tau_i) \text{ otherwise}; \]

#### observables

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

#### (error) conditions

\[ I[expr](\sigma) \text{ not defined.} \]
### Transformer: Destroy

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>Concrete Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>destroy(expr)</code></td>
<td><code>delete expr</code></td>
</tr>
</tbody>
</table>

#### Intuitive Semantics

*Destroy the object denoted by expression* $expr$.

#### Well-Typedness

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

#### Semantics

\[ \ldots \]

#### Observables

\[ Obs_{destroy}[u_x] = \{ (u_x, \perp, (+, \emptyset), u) \} \]

#### (Error) Conditions

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

$S_{MC}$:

\[ s_1 \rightarrow / \ldots; \text{delete } n; \ldots \rightarrow s_2 \]

\[ \text{destroy}(expr) \]

\[ t_{\text{destroy}(expr)[u_x]}(\sigma, \varepsilon) = \ldots \]

$\sigma$:

\[
\begin{array}{c}
c : C \\
\end{array} \quad n \quad \begin{array}{c}
u : C \\
\end{array}
\]

\[ \text{destroy } n \]

\[ u_x = c \]

\[ \frac{c : C}{n = \{u\}} \]

$\varepsilon$:

\[ \varepsilon' \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]

\[ \text{V is gone} \]
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><code>destroy(expr)</code></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

\[
\begin{align*}
  t[u_x](\sigma, \varepsilon) &= (\sigma', \varepsilon) \\
  \text{where } \sigma' &= \sigma|_{\text{dom}(\sigma) \setminus \{u\}} \text{ with } u = I[expr](\sigma, \underline{u}).
\end{align*}
\]

### Observables

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

### (Error) Conditions

\( I[expr](\sigma, \underline{u}) \) 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

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

- **Clear**: not defined if one the two intermediate “micro steps” is not defined.
**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
**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 $\mathcal{I}_0 = (\mathcal{I}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$ be a signature with signals (all classes 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 $S := (\Sigma \mathcal{D} \cup \{\#\} \times Eth)$ with actions $A := \left(2^{\mathcal{D}(\mathcal{E}) \times (\mathcal{D}(\mathcal{E}) \cup \{\bot\})} \times Evs(\mathcal{E}, \mathcal{D}) \times \mathcal{D}(\mathcal{E}) \right)^2$:

- $(\sigma, \varepsilon) \xrightarrow{(cons, \text{Snd})} (\sigma', \varepsilon')$ 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 $u$ commences, 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) $s = \#$ and $cons = \emptyset$, or an error condition occurs during consumption of $cons$. 
(i) Discarding An Event

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

if

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

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

- \( u \) 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)

\[ \forall (s, F, \text{expr}, \text{act}, s') \in \xrightarrow{} (SM_C) : F \neq E \lor I[\text{expr}](\sigma, u) = 0 \]

and

- the system configuration doesn’t change, i.e. \( \sigma' = \sigma \)

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

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

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

\[ cons = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset. \]
Example: Discard

$SM_C$:  

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

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

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

$\langle \langle \text{env} \rangle \rangle$

\[
\begin{align*}
C & : x, z : \text{Int} \\
y : \text{Int} \\
\langle \langle \text{env} \rangle \rangle
\end{align*}
\]

\[
\begin{align*}
\sigma: & \quad c : C \\
x = 1, z = 0, y = 2 \\
st = s_1 \\
stable = 1 \\
\varepsilon: & \quad J \text{ for } c, \\
\text{ready (id } u_d) \\
\end{align*}
\]

$\varepsilon'$ is consumed, nothing sent

\[
(\{s_3, \emptyset\})
\]

\[
\sigma' = \sigma
\]

$\varepsilon' = \varepsilon \oplus u_E$

ready $G$ for $C$

\[
\begin{align*}
\exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \\
\exists u_E \in \mathcal{D}(\varepsilon) : u_E \in \text{ready}(\varepsilon, u) \\
\forall (s, F, expr, act, s') \in (SM_C): \\
F \neq E \lor I[expr](\sigma) = 0 \\
\sigma(u)(\text{stable}) = 1, \sigma(u)(st) = s, \\
\sigma' = \sigma, \varepsilon' = \varepsilon \oplus u_E \\
\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset
\end{align*}
\]
\((ii)\) Dispatch

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

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

\[ \exists (s, F, expr, act, s') \in (\mathcal{S} \mathcal{M}_C) : F = E \land I[expr](\tilde{\sigma}_u) = 1 \]

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

and

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

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

\[ \sigma' = (\sigma''[u.\text{st} \mapsto s', u.\text{stable} \mapsto b, u.\text{params}_E \mapsto \emptyset])|_{\mathcal{D}(E)\setminus\{u_E\}} \]

where \(b\) depends:

- If \(u\) becomes stable in \(s'\), then \(b = 1\). It \textit{does} become stable if and only if there is no transition \textit{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.

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

\[
\begin{align*}
\forall s : \text{C} : \{s_1\} & \implies G[x > 0]/x := y \\
& \implies s_2 \\
H/z := y/x
\end{align*}
\]

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

\[
\begin{align*}
\sigma & : \{c : \text{C} \mid x = 1, z = 0, y = 2, \text{st} = s_1, \text{stable} = 1 \}
\end{align*}
\]

\[
\begin{align*}
\varepsilon & : \text{G for c}
\end{align*}
\]

\[
\begin{align*}
\exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \\
& \exists u_E \in \mathcal{D}(\varepsilon') : u_E \in \text{ready}(\varepsilon, u)
\end{align*}
\]

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

\[
\tilde{\sigma} = \sigma[u.\text{params}_E \mapsto u_E].
\]

\[
\begin{align*}
\sigma(u)(\text{stable}) & = 1, \sigma(u)(\text{st}) = s,
\end{align*}
\]

\[
\begin{align*}
(\sigma'', \varepsilon') & = t_{\text{act}}(\tilde{\sigma}, \varepsilon \ominus u_E)
\end{align*}
\]

\[
\begin{align*}
\sigma' & = (\sigma''[u.\text{st} \mapsto s', u.\text{stable} \mapsto b, u.\text{params}_E \mapsto \emptyset])|_{\mathcal{D}(\varepsilon') \setminus \{u_E\}}
\end{align*}
\]

\[
\begin{align*}
\text{cons} & = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{t_{\text{act}}}(\tilde{\sigma}, \varepsilon \ominus u_E)
\end{align*}
\]
(iii) Commence Run-to-Completion

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

if

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

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

and

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

where \( b \) depends as before.

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

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

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

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

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

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

\[ G, J \]

\[ C \]

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

\[ y : \text{Int} \]

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

\[ \sigma : \]

\[ c : C \]

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

\[ st = s_2 \]

\[ \text{stable} = 0 \]

\[ \varepsilon : \]

\[ c : C' \]

\[ \sigma' : \]

\[ c : C \]

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

\[ st = s_2 \]

\[ \text{stable} = 0 \]

\[ \varepsilon : \]

\[ t_{act} (\sigma, \varepsilon), \]

\[ (\sigma'', \varepsilon') = t_{act} (\sigma, \varepsilon), \]

\[ \sigma' = \sigma''[u.st \mapsto s', u.\text{stable} \mapsto b] \]

\[ \text{cons} = \emptyset, \text{Snd} = \text{Obs}_{t_{act}} (\sigma, \varepsilon) \]

\[ \exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C) : \sigma(u)(\text{stable}) = 0 \]

\[ \exists (s, \eta, \text{expr}, \text{act}, s') \in (SM_C) : I[\text{expr}](\sigma) = 1 \]

\[ \sigma(u)(\text{stable}) = 1, \sigma(u)(st) = s, \]

\[ (\sigma'', \varepsilon') = t_{act} (\sigma, \varepsilon), \]

\[ \sigma' = \sigma''[u.st \mapsto s', u.\text{stable} \mapsto b] \]

\[ \text{cons} = \emptyset, \text{Snd} = \text{Obs}_{t_{act}} (\sigma, \varepsilon) \]
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

$$\begin{align*}
(\sigma, \varepsilon) \xrightarrow{(cons, Snd)_{env}} (\sigma', \varepsilon')
\end{align*}$$

if

- an environment event $E \in \mathcal{E}_{env}$ is spontaneously sent to an alive object $u \in \mathcal{D}(\sigma)$, i.e.
  $$\sigma' = \sigma \cup \{u_E \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}, \quad \varepsilon' = \varepsilon \oplus u_E$$
  where $u_E \notin \text{dom}(\sigma)$ and $\text{atr}(E) = \{v_1, \ldots, v_n\}$.

- Sending of the event is observed, i.e. $cons = \emptyset$, $Snd = \{(env, E(\vec{d}))\}$.

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. $\text{dom}(\sigma') = \text{dom}(\sigma)$.

- $\varepsilon' = \varepsilon$. 
\[ [x > 0]/x := x - 1; n! J \]

\[ H/z := y/(x+1) \]

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

**Example: Environment**

\[
SM_C: \quad s_1 \xrightarrow{G[x > 0]/x := y} s_2 \xrightarrow{H/z := y/(x+1)}
\]

\[ x = 0, z = 0, y = 2 \]
\[ st = s_2 \]
\[ stable = 1 \]

\[ c : C \]

\[ \sigma = \{ (signal, env) \} \]

\[ \varepsilon' = \{ (signal) \} \]

\[ \varepsilon' = \varepsilon \]

\[ H \]

\[ G, J \]

\[ C \]

\[ x, z : \text{Int} \]
\[ y : \text{Int} \]
\[ \langle \langle \text{env} \rangle \rangle \]

\[ n \]

\[ 0, 1 \]

\[ \sigma' = \sigma \cup \{ u_E \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n \} \} \]

\[ u \in \text{dom}(\sigma) \]

\[ \text{cons} = \emptyset, \ Snd = \{ (\text{env}, E(\overrightarrow{d})) \} \]

\[ \text{atr}(E) = \{ v_1, \ldots, v_n \} \]
(v) Error Conditions

\[
S \xrightarrow{(\text{cons}, \text{Snd})_u} \# \quad \text{if, in (ii) or (iii),}
\]

- \(I[expr]\) 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:

- \(E[x/0]/\text{act}\) from \(S_1\) to \(S_2\)
- \(E[\text{true}]/\text{act}\) from \(S_1\) to \(S_3\)
- \(E[expr]/x := x/0\) from \(S_1\) to \(S_2\)
Example: Error Condition

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

\[ \sigma: \]

\[
\begin{array}{l}
c : C \\
x = 0, z = 0, y = 27 \\
st = s_2 \\
stable = 1
\end{array}
\]

\[ \varepsilon: \]

\[ H \text{ for } c \]

- \( I[expr] \) not defined for \( \sigma \), or
- \( t_{act} \) is not defined for \( (\sigma, \varepsilon) \)
- consumption according to (ii) or (iii)
- \( Snd = \emptyset \)
**Notions of Steps: The Step**

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

\[ E[x > 0]/ \]

\[ /x := x - 1 \]

\[ \sigma: \]

\[
\begin{array}{c}
:: C \\
\hline
x = 2
\end{array}
\]

\[ \varepsilon: \]

\[ E \text{ for } u \]
**Proposal:** Let

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

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

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.**

(Proof left as exercise...)

- **(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.
Putting It All Together
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 \((\sigma, \varepsilon) \xrightarrow{(\text{cons, Snd})} (\sigma', \varepsilon')\).

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

**Proposal:**

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

\((CD, SM, OD)\).

And set

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

**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**

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

where

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

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

The **computations of** \( \mathcal{M} \) are the computations of \((S, \rightarrow, S_0)\).
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. exercises and tutorial for discussion of “reasonable point”.)

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

**Pragmatics:**

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

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

- **Not common:** if $\mathcal{S}M$ is given, then constraints are also considered when choosing transitions in the RTC-algorithm. In other words: even in presence of mistakes, the $\mathcal{S}M$ never move to inconsistent configurations.
References
References


