Contents & Goals

Last Lecture:

- System configuration
- Transformer

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:
  - Transformer cont’d
  - Examples for transformer
  - Run-to-completion Step
  - Putting It All Together
System Configuration, Ether, Transformer
System Configuration

**Definition.** Let $\mathcal{S}_0 = (\mathcal{T}_0, \mathcal{C}_0, V_0, \text{atr}_0, \mathcal{E})$ be a signature with signals, $\mathcal{D}_0$ a structure of $\mathcal{S}_0$, $(\text{Eth}, \text{ready}, \oplus, \ominus, [\cdot])$ an ether over $\mathcal{S}_0$ and $\mathcal{D}_0$. Furthermore assume there is one core state machine $M_C$ per class $C \in \mathcal{C}$.

A **system configuration** over $\mathcal{S}_0$, $\mathcal{D}_0$, and $\text{Eth}$ is a pair $(\sigma, \varepsilon) \in \Sigma_\mathcal{D} \times \text{Eth}$ where

- $\mathcal{I} = (\mathcal{T}_0 \cup \{S_{MC} \mid C \in \mathcal{C}\}, \mathcal{C}_0, V_0 \cup \{\langle \text{stable} : \text{Bool}, -, \text{true}, \emptyset \rangle\})$
- $\cup \{\langle \text{st}_C : S_{MC}, +, s_0, \emptyset \rangle \mid C \in \mathcal{C}\}$
- $\cup \{\langle \text{params}_E : E_{0,1}, +, \emptyset, \emptyset \rangle \mid E \in \mathcal{E}_0\}$,
- $\{C \mapsto \text{atr}_0(C) \\cup \{\text{stable, st}_C\} \cup \{\text{params}_E \mid E \in \mathcal{E}_0\} \mid C \in \mathcal{C}\}$, and

- $\mathcal{D} = \mathcal{D}_0 \cup \{S_{MC} \mapsto S(M_C) \mid C \in \mathcal{C}\}$, and
- $\sigma(u)(r) \cap \mathcal{D}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_{\text{atr}_0}$, (e.g. $r : C_n$).
\( \mathcal{S}_0 = (\{ a: \text{Int} \}, \{ \mathcal{C}, \mathcal{E}, \mathcal{I} \}, \{ x: \text{Int}, y: \text{Int} \}, \{ \mathcal{C}(\mathcal{C}(x, y)), \mathcal{E}(\mathcal{I}(\mathcal{C}(x, y))), \{ \mathcal{E}(\mathcal{I}) \} \}) \)

\( \mathcal{S} = (\{ \text{Int}, \mathcal{S}_{\mathcal{C}} \}, \{ \mathcal{C}, \mathcal{E}, \mathcal{I} \}, \{ x, y, a: \text{Int}, \text{stable}: \text{Bool}, \mathcal{S}_{\mathcal{C}2}, \text{process}_{\mathcal{E}}: \mathcal{E}, \text{process}_{\mathcal{C}}: \mathcal{C} \}, \{ (x, y, \text{stable}, \mathcal{S}_{\mathcal{C}2}, \text{process}_{\mathcal{E}}, \text{process}_{\mathcal{C}}), \mathcal{E}(\mathcal{I}(\mathcal{C}(x, y))), \{ \mathcal{E}(\mathcal{I}) \} \}) \)

\( D(\text{Int}) = D_0(\text{Int}) \)

\( D(\mathcal{S}_{\mathcal{C}2}) = \{ s_{10}, s_{11}, s_{21}, s_{22} \} \)

if \( \mathcal{E} \mathcal{I} \) is shared FIFO queue

\( \mathcal{E} = \mathcal{V} \mathcal{V}' \)

\( \text{V} \text{V}' \) (an instance of \( \mathcal{F} \) is ready for \( \mathcal{W} \)
Where are we?

• **Wanted**: a labelled transition relation

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

on system configuration, labelled with the **consumed** and **sent** events, 

\((\sigma', \varepsilon')\) being the result (or effect) of **one object** \(u_x\) taking a transition of **its** state machine from the current state mach. state \(\sigma(\text{st}_C)(u_x)\).

• **Have**: system configuration \((\sigma, \varepsilon)\) comprising current state machine state and stability flag for each object, and the ether.

• **Plan**:

  (i) Introduce **transformer** as the semantics of action annotations. **Intuitively**, \((\sigma', \varepsilon')\) is the effect of applying the transformer of the taken transition.

  (ii) Explain how to choose transitions depending on \(\varepsilon\) and when to stop taking transitions — the **run-to-completion “algorithm”**.
Transformer

Definition.
Let $\Sigma_0$ the set of system configurations over some $I_0$, $D_0$, Eth.

We call a relation $t \subseteq D(C) \times (\Sigma_0 \times Eth) \times (\Sigma_0 \times Eth)$ a (system configuration) transformer.

- In the following, we assume that each application of a transformer $t$ to some system configuration $(\sigma, \varepsilon)$ for object $u_x$ is associated with a set of observations

$$\text{Obs}_t[u_x](\sigma, \varepsilon) \in 2^{D(C) \times D(S) \times \text{Evs}(S \cup \{*, +\}, D) \times D(C)}.$$ 

- An observation $(u_{src}, u_{e}, (E, \bar{d}), u_{dst}) \in \text{Obs}_t[u_x](\sigma, \varepsilon)$ represents the information that, as a “side effect” of $u_x$ executing $t$, an event $(E, \bar{d})$ has been sent from $u_{src}$ to $u_{dst}$.

Special cases: creation/destruction.
In the following, we consider:

\[
\text{Act}_p ::= \{ \text{skip} \}
\]

\[
u \{ \text{update} (\text{expr}_1, v, \text{expr}_2) \mid \text{expr}_1, \text{expr}_2 \in \text{OCLExpr}, v \in V \}
\]

\[
u \{ \text{send} (\text{expr}_1, E, \text{expr}_2) \mid \text{expr}_1, \text{expr}_2 \in \text{OCLExpr}, E \in E \}
\]

\[
u \{ \text{create} (C, \text{expr}_1, v) \mid \text{expr}_1 \in \text{OCLExpr}, C \in C, v \in V \}
\]

\[
u \{ \text{destroy} (\text{expr}) \mid \text{expr} \in \text{OCLExpr} \}
\]

\[
\text{Expr}_p ::= \text{OCL expressions over } Y
\]
### Transformer: Skip

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>skip</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>intuitive semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>do nothing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>well-typedness</th>
</tr>
</thead>
<tbody>
<tr>
<td>././</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t[u_x](\sigma, \varepsilon) = {(\sigma, \varepsilon)}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>observables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Obs_{skip}[u_x](\sigma, \varepsilon) = \emptyset$</td>
</tr>
</tbody>
</table>

(error) conditions
### Transformer: Update

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>Concrete Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update(expr_1, v, expr_2)</code></td>
<td><code>expr_1, v := expr_2</code></td>
</tr>
</tbody>
</table>

**Intuitive Semantics**

Update attribute `v` in the object denoted by `expr_1` to the value denoted by `expr_2`.

**Well-typedness**

- `expr_1 : \tau_C` and `v : \tau \in atr(C)`; `expr_2 : \tau`;
- `expr_1, expr_2` obey visibility and navigability.

**Semantics**

\[
\begin{align*}
t_{update}(expr_1, v, expr_2)[u_x](\sigma, \varepsilon) &= \{ (\sigma', \varepsilon) \} \\
\text{where } \sigma' &= \sigma[u \mapsto \sigma(u)[v \mapsto I[expr_2](\sigma, \beta)]] \text{ with } \\
& \quad u = I[expr_1](\sigma, \beta), \quad \beta = \{ \text{this} \mapsto u_x \}.
\end{align*}
\]

**Observables**

\[
\begin{align*}
Obs_{update}(expr_1, v, expr_2)[u_x] &= \emptyset
\end{align*}
\]

**Error Conditions**

Not defined if `I[expr_1](\sigma, \beta)` or `I[expr_2](\sigma, \beta)` not defined.
Update Transformer Example

\(SM_C:\)

\[
\begin{align*}
&\text{update}(expr_1, v, expr_2) \\
&t_{\text{update}}(expr_1, v, expr_2)[u_x](\sigma, \varepsilon) = (\sigma[u \mapsto \sigma(u)[v \mapsto I[expr_2](\sigma, \beta)]], \varepsilon), \\
&u = I[expr_1](\sigma, \beta)
\end{align*}
\]

\(\sigma:\)

\[
\begin{array}{c}
\hline
u_1 : C \\
x = 4 \\
y = 0 \\
\hline
\end{array}
\]

\(\varepsilon:\)

\[
\begin{array}{c}
\hline
u_1 : C \\
x = 5 \\
y = 0 \\
\hline
\end{array}
\]

\(\sigma':\)

\[
\begin{array}{c}
\hline
u_1 : C \\
x = 5 \\
y = 0 \\
\hline
\end{array}
\]

\(\varepsilon':\)
Transformer: Send

abstract syntax

\[
\text{send}(E(expr_1, \ldots, expr_n), expr_{dst})
\]

concrete syntax

\[
expr_{dst} ! E(expr_1, \ldots, expr_n)
\]

intuitive semantics

Object \( u_x : C \) sends event \( E \) to object \( expr_{dst} \), i.e. create a fresh signal instance, fill in its attributes, and place it in the ether.

well-typedness

\[
expr_{dst} : \tau_D, \ C, D \in \mathcal{E} \setminus \mathcal{E}; \ E \in \mathcal{E}; \\
\text{atr}(E) = \{v_1 : \tau_1, \ldots, v_n : \tau_n\}; \ expr_i : \tau_i, \ 1 \leq i \leq n; \\
\text{all expressions obey visibility and navigability in } C
\]

semantics

\[
t_{send}(E(expr_1, \ldots, expr_n), expr_{dst})[u_x](\sigma, \varepsilon) \triangleq (\sigma', \varepsilon')
\]

where \( \sigma' = \sigma \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}\} \); \( \varepsilon' = \varepsilon \oplus (u_{dst}, u) \);

if \( u_{dst} = I[expr_{dst}](\sigma, \beta) \in \text{dom}(\sigma); \ d_i = I[expr_i](\sigma, \beta) \) for \( 1 \leq i \leq n; \)

\( u \in \mathcal{D}(E) \) a fresh identity, i.e. \( u \notin \text{dom}(\sigma) \),

and where \( (\sigma', \varepsilon') = (\sigma, \varepsilon) \) if \( u_{dst} \notin \text{dom}(\sigma) \); \( \beta = \{\text{this} \mapsto u_x\} \).

observables

\[
\text{Obs}_{send}[u_x] = \{(u_x, u, (E, d_1, \ldots, d_n), u_{dst})\}
\]

(error) conditions

\[
I[expr](\sigma, \beta) \text{ not defined for any} \expr \in \{expr_{dst}, expr_1, \ldots, expr_n\}
\]
Send Transformer Example

$SM_C$:

\[ \ldots; n! F(x+1); \ldots \]

\[ \sigma: \]

\[
\begin{array}{c}
u_1: C \\
x = 5
\end{array}
\]

\[ \varepsilon: \]

send($E(expr_1, \ldots, expr_n), expr_{dst}$)

\[ t_{send}(E(expr_1, \ldots, expr_n), expr_{dst})[u_x](\sigma, \varepsilon) = \ldots \]
### 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.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**

\[
expr : \tau_D, \quad v \in atr(D), \quad atr(C) = \{ \langle v_i : \tau_i, expr^0_i \rangle | 1 \leq i \leq n \}\]

**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).
Create Transformer Example

$SM_C$:

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

create($C, expr, v$)

\[t_{\text{create}(C, expr, v)}(\sigma, \varepsilon) = \ldots\]
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

### Abstract Syntax

\[
\text{create}(C, \text{expr}, v)
\]

### Concrete Syntax

### Intuitive Semantics

Create an object of class \( C \) and assign it to attribute \( v \) of the object denoted by expression \( \text{expr} \).

### Well-Typedness

\[
\text{expr} : \tau_D, \quad v \in \text{atr}(D), \quad \text{atr}(C) = \{(v_1 : \tau_1, \text{expr}_0^0) | 1 \leq i \leq n\}
\]

### Semantics

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

- \( u \in \mathcal{D}(C) \) fresh, i.e. \( u \not\in \text{dom}(\sigma) \);
- \( u_0 = \text{I}[\text{expr}](\sigma, \beta) \); \( d_i = \text{I}[\text{expr}_i^0](\sigma, \beta) \) if \( \text{expr}_i^0 \neq '' \) and arbitrary value from \( \mathcal{D}(\tau_i) \) otherwise; \( \beta = \{\text{this} \mapsto u_x\} \).

### Observables

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

### (Error) Conditions

\( \text{I}[\text{expr}](\sigma) \) 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

... (contents not shown)

#### Observables

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

#### (Error) Conditions

\[ I[expr](\sigma, \beta) \text{ not defined.} \]
$\mathcal{SM}_C$:

\[ S_1 \xrightarrow{\ldots; \text{delete}\ n; \ldots} S_2 \]

\[
destroy(expr)
\]

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

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

**observables**

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

**(error) conditions**

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

\[
egin{align*}
\text{\( \chi := x + 1, \ y := 2 \cdot x \), } & \quad n \not\rightarrow F \\
\text{\( \mathcal{U} \mathcal{P} \mathcal{K} \) (\( \mathcal{T}_{\text{send}} [-](\sigma, \varepsilon) \))}
\end{align*}
\]
**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.
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 $\mathcal{I}_0 = (\mathcal{I}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$ be a signature with signals (all classes active), $\mathcal{D}_0$ a structure of $\mathcal{I}_0$, and $(Eth, \text{ready}, \oplus, \ominus, [\cdot])$ 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 $S := (\Sigma^{\mathcal{D}_0} \cup \{\#\} \times Eth)$ with actions $A := \left(2^{\mathcal{D}(\mathcal{C})} \times (\mathcal{D}(\mathcal{E}) \cup \{\bot\}) \times Evs(\mathcal{E}, \mathcal{D}) \times \mathcal{D}(\mathcal{C})\right)^2$:

1. $(\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})} (\sigma', \varepsilon')$ 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 $u$,

2. $s \xrightarrow{(\text{cons}, \emptyset)} \#$ if and only if
   (v) $s = \#$ and $\text{cons} = \emptyset$, or an error condition occurs during consumption of $\text{cons}$.
(i) Discarding An Event

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

if

- an \( E \)-event (instance of signal \( E \)) is ready in \( \varepsilon \) for object \( u \) of a class \( \mathcal{C} \), i.e. if
  \[
  u \in \text{dom}(\sigma) \cap \mathcal{D}(\mathcal{C}) \land \exists u_E \in \mathcal{D}(\mathcal{C}) : u_E \in \text{ready}(\varepsilon, u)
  \]
- \( u \) 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)
  \[
  \forall (s, F, expr, act, s') \in \rightarrow (\mathcal{SM}_C) : F \neq E \lor I[expr](\sigma) = 0
  \]

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

$$S\mathcal{M}_C:$$

$$\sigma:$$

- $$c: C$$
  - $$x = 1, z = 0, y = 2$$
  - $$st = s_1$$
  - $$stable = 1$$

$$\varepsilon:$$

- $$J$$ for $$c$$,
- $$G$$ for $$c$$

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

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

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

$$H$$

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

$$\langle\langle signal\rangle\rangle$$

$$G, J$$

$$n$$

$$C$$

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

$$y: \text{Int}$$

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

- $$\exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C)$$
  - $$\exists u_E \in \mathcal{D}(\varepsilon'): u_E \in \text{ready}(\varepsilon, u)$$
- $$\forall (s, F, expr, act, s') \in \rightarrow (S\mathcal{M}_C):$$
  - $$F \neq E \lor I[expr](\sigma) = 0$$
- $$\sigma(u)(\text{stable}) = 1, \sigma(u)(st) = s,$$
- $$\sigma' = \sigma, \varepsilon' = \varepsilon \oplus u_E$$
- $$\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset$$
(ii) Dispatch

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

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

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

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.

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

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

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

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

$SM_C$:

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

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

\[ H \]

\[ C \]

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

\[ G, J \]

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

\[ \langle \langle signal\rangle \rangle \]

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

\[ y : \text{Int} \]

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

\[ n \]

\[ 0, 1 \]

\[ \sigma : \]

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

\[ \varepsilon : \]

\[ G \text{ for } c \]

\[ c : C \]

\[ \langle c, \emptyset \rangle \]

\[ (c, \emptyset) \]

\[ c : C \]

\[ x = 2, \]

\[ t = 0, \]

\[ y = 2, \]

\[ \text{st} = s_2 \]

\[ \text{stable} = 0 \]

\[ : 0' \]

\[ \varepsilon = \varepsilon \cup u_E \]

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

\[ \exists u_E \in \mathcal{D}(\varepsilon') : u_E \in \text{ready}(\varepsilon, u) \]

\[ \exists (s, F, \text{expr}, \text{act}, s') \in \to (SM_C) : \]

\[ F = E \land I[\text{expr}](\tilde{\sigma}) = 1 \]

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

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

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

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

\[ \text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{\text{act}}(\tilde{\sigma}, \varepsilon \cup u_E) \]
(iii) Commence Run-to-Completion

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

if

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

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

and

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

  where \(b\) depends as before.

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

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

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

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

\[ SM_C: \]

\[ s_1 \rightarrow G \rightarrow s_2 \]

\[ \sigma: \]

\[ c : C \]
\[ x = 2, z = 0, y = 2 \]
\[ st = s_2 \]
\[ stable = 0 \]

\[ \varepsilon: \]

\[ d \in C \]

\[ \exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C) : \sigma(u)(stable) = 0 \]
\[ \exists (s, \_ , expr, act, s') \in \rightarrow (SM_C) : I[expr](\sigma) = 1 \]
\[ \sigma(u)(stable) = 1, \sigma(u)(st) = s, \]

\[ (\sigma'', \varepsilon') = t_{act}(\sigma, \varepsilon), \]
\[ \sigma' = \sigma''[u.st \mapsto s', u.stable \mapsto b] \]
\[ cons = \emptyset, Snd = \text{Obs}_{act}(\sigma, \varepsilon) \]
(iv) Environment Interaction

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

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

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

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

\[\text{Example: Environment}\]

\[\mathcal{SM}_C: \]

\[\sigma: \]

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

\[\varepsilon: \]

\[\begin{array}{c}
  (\phi, \cdot) \\
  (\omega, \cdot)
\end{array}
\]

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

\[\begin{array}{c}
  \varepsilon' = \varepsilon \oplus u_E \text{ where } u_E \notin \text{dom}(\sigma) \text{ and } \text{attr}(E) = \{v_1, \ldots, v_n\}
\end{array}
\]

\[\begin{array}{c}
  u \in \text{dom}(\sigma) \\
  \text{cons} = \emptyset, \text{Snd} = \{(\text{env}, E(\overline{d}))\}
\end{array}\]
(v) Error Conditions

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

if, in (ii) or (iii),

- \( I[expr] \) is not defined for \( \sigma \), or
- \( t_{\text{act}} \) is not defined for \((\sigma, \varepsilon)\), i.e. \( t_{\text{act}}(w)(\sigma, \varepsilon) = \emptyset \)

and

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

Examples:

- \( s_1 \xrightarrow{E[x/0]/\text{act}} s_2 \)
- \( s_1 \xrightarrow{E[\text{true}]/\text{act}} s_3 \)
- \( s_1 \rightarrow E[expr]/x := x/0 \rightarrow s_2 \)

---

If we add \((x), this is leading to it with F

---
**Example: Error Condition**

\[ x > 0 \] yields: \( x := x - 1; n \downarrow J \)

\( S M_C: \)

- \( S_1 \) to \( S_2 \) via \( G[x > 0] / x := y \)
- \( H / z := y / x \)

\( \sigma: \)

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

\( \varepsilon: \)

- \( H \) 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 \((\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})}{_u} (\sigma', \varepsilon')\) 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:**

\[
E[x > 0] / \\
\sigma:\ 
\begin{array}{|c|}
\hline
C \\
\hline
x = 2 \\
\hline
\end{array} \\
\varepsilon:\ E \text{ for } u
\]
References
References

