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
Definition. Let $\mathcal{S} = (\mathcal{R}_0, \mathcal{E}_0, V_0, atr_0, \mathcal{E})$ be a signature with signals, $\mathcal{R}_0$ a structure of $\mathcal{S}$, $(\mathcal{Eth}, \text{ready}, \oplus, \ominus, [\cdot])$ an ether over $\mathcal{S}$ and $\mathcal{R}_0$. Furthermore assume there is one core state machine $M_C$ per class $C \in \mathcal{C}$.

A system configuration over $\mathcal{S}$, $\mathcal{R}_0$, and $\mathcal{Eth}$ is a pair $(\sigma, \varepsilon) \in \Sigma^\mathcal{R}_0 \times \mathcal{Eth}$ where

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

- $\mathcal{E} = \mathcal{R}_0 \cup \{S_{MC} \mapsto S(M_C) \mid C \in \mathcal{C}\}$, and

- $\sigma(u)(r) \cap \mathcal{E}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_{0,0}$ (e.g. $r(C)$).
Where are we?

- **Wanted:** a labelled transition relation
  \[ (\sigma, \varepsilon) \xrightarrow{\text{cons}, \text{Std}} (\sigma', \varepsilon') \]
  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 \(\sigma(u_x)(st_C)\).

- **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 \( \Sigma^T \) the set of system configurations over some \( \mathcal{S}_0, \mathcal{D}_0, \mathcal{E}_0 \).

We call a relation \( t \subseteq \mathcal{D}(\mathcal{F}) \times (\Sigma^T \times \mathcal{E}_0) \times (\Sigma^T \times \mathcal{E}_0) \) 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

\[
\text{Obs}_t[u_x](\sigma, \varepsilon) \in 2^{\mathcal{D}(\mathcal{F}) \times \mathcal{E}_0 \cup \{*, +\}} \times \mathcal{E}_0 \times (\Sigma^T \times \mathcal{E}_0)
\]

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

**Special cases:** creation/destruction.

---

**Acty:** \[ \{ \] }

Initial \[
\text{update} \left( \text{exp}_i, \nu, \text{exp}_j \right) | \text{exp}_i, \text{exp}_j \in \text{oclExpr}, \nu \in \mathcal{V}_3
\]

Sort \[
\text{send} \left( \text{exp}_i, E, \text{exp}_j \right) | \text{exp}_i, \text{exp}_j \in \text{oclExpr}, E \in \mathcal{E}_3
\]

Create \[
\text{create} \left( C, \text{exp}_i \right) | \text{exp}_i \in \text{oclExpr}, C \in \mathcal{C}, \nu \in \mathcal{V}_3
\]

Destroy \[
\text{destroy} \left( \text{exp}_i \right) | \text{exp}_i \in \text{oclExpr}
\]

Empty: OCL expressions are empty.
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>

intuitive semantics

**do nothing**

well-typedness

\. /.

semantics

\[ t[u_{x}](\sigma, \epsilon) = \{(\sigma, \epsilon)\} \]

observables

\[ \text{Obs}_{\text{skip}}[u_{x}](\sigma, \epsilon) = \emptyset \]

(error) conditions

---

Transformer: Update

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{update}(expr_{1}, v, expr_{2})</td>
<td>expr_{1}, v \leftarrow expr_{2}</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 \( v : \tau \in \text{atr}(C) \); \( expr_{2} : \tau \);

\( expr_{1}, expr_{2} \) obey visibility and navigability

semantics

\[ t_{\text{update}}(expr_{1}, v, expr_{2})[u_{x}](\sigma, \epsilon) = \{(\sigma', \epsilon)\} \]

where \( \sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto I[expr_{2}](\sigma, \beta)]] \)

\( u = I[expr_{1}](\sigma, \beta), \beta = \{\text{this} \mapsto u_{x}\} \).

observables

\[ \text{Obs}_{\text{update}}[expr_{1}, v, expr_{2}][u_{x}] = \emptyset \]

(error) conditions

Not defined if \( I[expr_{1}](\sigma, \beta) \) or \( I[expr_{2}](\sigma, \beta) \) not defined.
**Update Transformer Example**

\[ s_1 \xrightarrow{\text{update}(\text{expr}_1, v, \text{expr}_2)} s_2 \]

\[
\text{update}(\text{expr}_1, v, \text{expr}_2)
\]

\[
\text{t}\_\text{update}(\text{expr}_1, v, \text{expr}_2)[u_2][\sigma, \varepsilon] = (\sigma[u \mapsto \sigma(u) [v \mapsto I[\text{expr}_2](\sigma, \beta)], \varepsilon),
\]

\[
u = I[\text{expr}_1](\sigma, \beta)
\]

- **Abstract Syntax**
  \[
  \text{abstract syntax}
  \]

- **Concrete Syntax**
  \[
  \text{concrete syntax}
  \]

- **Intuitive Semantics**
  \[
  \text{intuitive semantics}
  \]

- **Well-typedness**
  \[
  \text{well-typedness}
  \]

- **Semantics**
  \[
  \text{semantics}
  \]

- **Observables**
  \[
  \text{observables}
  \]

- **Error Conditions**
  \[
  \text{error conditions}
  \]

---

**Transformer: Send**

- **Abstract Syntax**
  \[
  \text{abstract syntax}
  \]

- **Concrete Syntax**
  \[
  \text{concrete syntax}
  \]

- **Intuitive Semantics**
  \[
  \text{intuitive semantics}
  \]

- **Well-typedness**
  \[
  \text{well-typedness}
  \]

- **Semantics**
  \[
  \text{semantics}
  \]

- **Observables**
  \[
  \text{observables}
  \]

- **Error Conditions**
  \[
  \text{error conditions}
  \]
Send Transformer Example

\[ \text{SM}_C: \]

\[
\begin{align*}
\text{send}(E(expr_1, \ldots, expr_n, expr_{dst})) \\
\text{create}(E(expr_1, \ldots, expr_n), expr_{dst})(\sigma, \epsilon) = \ldots
\end{align*}
\]

\[
\begin{align*}
\sigma':
\quad & u_1 : C \\
& x = 5
\end{align*}
\]

\[
\begin{align*}
\epsilon':
\quad & ε_1 \\
& ε_2
\end{align*}
\]

\[
\begin{align*}
\text{create}(E(expr_1, \ldots, expr_n), expr_{dst})(\sigma, \epsilon) = \ldots
\end{align*}
\]

\[
\begin{align*}
\text{create}(E(expr_1, \ldots, expr_n), expr_{dst})(\sigma, \epsilon) = \ldots
\end{align*}
\]

Transformer: Create

**abstract syntax**

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

**intuitive semantics**

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

**well-typedness**

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

**semantics**

\[
\begin{align*}
\text{(error) conditions}
\quad & I[expr](\sigma, \beta) \text{ not defined.}
\end{align*}
\]

- 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

\[ SMC: \]

\[
\begin{array}{c}
\text{create}(C, \text{expr}, v) \\
\end{array}
\]

Create Transformer Example

\[ \sigma: \]

\[
\frac{d: D}{n = 0}
\]

\[ \epsilon: \]

\[
\boxed{\begin{array}{c}
\epsilon: \\
\end{array}}
\]

\[
\begin{array}{c}
\text{create}(C, \text{expr}, v) \\
\end{array}
\]

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><code>create(C, expr, v)</code></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**
\[ expr : \tau_D, v \in atr(D), \text{at}r(C) = \{\langle v_1 : \tau_1, expr_1^0 \rangle | 1 \leq i \leq n\} \]

**semantics**
\[
\begin{align*}
\sigma' &= \sigma \uplus \bigcup \{ u \mapsto \sigma(u) \mapsto \{ v \mapsto u \} \mid 1 \leq i \leq n \}, \\
\varepsilon' &= [u](\varepsilon); \quad u \in \mathcal{T}(C)\text{ fresh, i.e. } u \notin \text{dom}(\sigma); \\
u_0 &= I[expr]\langle \sigma, \beta \rangle; \quad d_i = I[expr_0^i]\langle \sigma, \beta \rangle \text{ if } expr_0^i \neq \" \text{ and arbitrary value from } \mathcal{T}(\tau_i) \text{ otherwise; } \beta = \{\text{this } \mapsto u_x\}.
\end{align*}
\]

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

**(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></td>
</tr>
</tbody>
</table>

**intuitive semantics**
Destroy the object denoted by expression \( expr \).

**well-typedness**
\[ expr : \tau_C, C \in \mathcal{E} \]

**semantics**
\[ \ldots \]

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

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

\[ \text{SM} \]

\[ \sigma : \tilde{C} \quad \varepsilon : \tilde{C} \]

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

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

\[ (\sigma', \varepsilon') \]

\[ \text{What to Do With the Remaining Objects?} \]

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><code>expr</code></td>
</tr>
</tbody>
</table>

Intuitive semantics

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

Well-typedness

*expr* : `τ_C`, `C` ∈ ℒ

Semantics

`t[u_x](σ, ε) = \{(σ', ε)\}

where σ' = σ|_{dom(σ)\{u\}} with u = I[expr](σ, β).

Observables

`Obs_{destroy[u_x]} = \{(u_x, ⊥, (+, ∅), u)\}

(error) conditions

`I[expr](σ, β)` 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](σ, ε) = t_2[u_x](t_1[u_x](σ, ε))`

  with observation

  `Obs_{t_2\circ t_1}[u_x](σ, ε) = Obs_{t_1}[u_x](σ, ε) \cup Obs_{t_2}[u_x](t_1(σ, ε)).`

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

```
x := x+1; n, y := 2)

\text{let } \text{and } \text{in} \ F
```

\text{let } \text{and } \text{in} \ F
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 $\mathcal{S}_0 = (\mathcal{S}_0, \mathcal{C}_0, V_0, \text{attr}_0, \mathcal{E})$ be a signature with signals (all classes active), $\mathcal{S}_0$ a structure of $\mathcal{S}_0$, and $(\mathcal{Eth}, \text{ready}, \oplus, \ominus, [\cdot])$ 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 $S := (\Sigma \mathcal{C}_0 \cup \{\#\} \times \mathcal{Eth})$ with actions $A := \left(2^{\mathcal{C}(\mathcal{S})} \times (\mathcal{Eth} \cup \{\bot\}) \times \mathcal{E}_{\mathcal{S}_0}(\mathcal{Eth} \times \mathcal{C}(\mathcal{S}))\right)^2$:

- $(\sigma, \varepsilon)$ \xrightarrow{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 $u$ commences, i.e. object $u$ is not stable and continues to process an event,
  - (iv) the environment interacts with object $u$,

- $s$ \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

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

if

- an $E$-event (instance of signal $E$) is ready in $\varepsilon$ for object $u$ of a class $\mathcal{C}$, i.e. if
  $$u \in \text{dom}(\sigma) \cap \mathcal{E}(\mathcal{C}) \land \exists u_E \in \mathcal{E}(\mathcal{E}) : u_E \in \text{ready}(\varepsilon, u)$$
- $u$ is stable and in state machine state $s$, i.e. $\sigma(u)(\text{stable}) = 1$ and $\sigma(u)(\text{st}) = s$,
- 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)
  $$\forall (s, F, \text{expr}, \text{act}, s') \in \rightarrow (SM_C) : F \neq E \vee I[\text{expr}] (\mathcal{S}) = 0$$

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

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

SM\(C\):

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

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

\[ H/n \]

\[ C' \]

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

\[ y : \text{Int} \]

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

\[ H' \]

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

\[ G, J \]

\[ C \]

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

\[ y : \text{Int} \]

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

\[ \sigma \]

\[ \epsilon \]

\[ \sigma' \]

\[ \epsilon' \]

\[ \Theta \]

\[ \gamma \]

\[ g \text{ for } c \]

\[ r \text{ for } c \]


(ii) Dispatch

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

- \( u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \) \( \land \) \( \exists u_E \in \mathcal{D}(\mathcal{E}) : u_E \in \text{ready}(\epsilon, u) \)
- \( u \) 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.

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

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

and

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

\[ (\sigma'', \epsilon') = t_{\text{act}}(\hat{\sigma}, \epsilon \oplus u_E), \]

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

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 \( (\sigma', \epsilon') \).
- Otherwise \( b = 0 \).
- Consumption of \( u_E \) and the side effects of the action are observed, i.e.

\[ \text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{\text{act}}(\hat{\sigma}, \epsilon \oplus u_E) \]
Example: Dispatch

\[ \text{SMC:} \quad G[x > 0]/x := y \]

\[ H/\bar{z} := y/\bar{x} \]

\( \sigma: \]

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

\( \varepsilon: \]

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

(iii) Commence Run-to-Completion

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

if

- there is an unstable object \( u \) of a class \( \mathcal{C} \), i.e.

  \[ u \in \text{dom}(\sigma) \cap \mathcal{P}(C) \land u \in \text{ready}(\varepsilon, u) \]

- there is a transition without trigger enabled from the current state \( s = \sigma(u)(\text{st}) \), i.e.

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

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.

  \[ \text{cons} = \emptyset, \text{Snd} = \text{Obs}_{\text{act}}(\sigma, \varepsilon). \]
Example: Commence

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

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

(iv) Environment Interaction

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

Then

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

if

- an environment event \( E \in \mathcal{E}_{\text{env}} \) 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. \( \text{cons} = \emptyset, \text{Snd} = \{(env, E(d))\} \).
- 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. \( \text{dom}(\sigma') = \text{dom}(\sigma) \).
- \( \varepsilon' = \varepsilon \).
Example: Environment

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

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

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

\[ G, J \]

\[ C \]

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

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

\[ \sigma : C \]

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

\[ st = s_2 \]

\[ \text{stable} = 1 \]

\[ \varepsilon : \sigma \]

\[ H[s_2, \varepsilon \circ (s_1, w)] \]

\[ \langle x, y \rangle \]

\[ \langle s_1, s_2 \rangle \]

\[ s_1 \rightarrow \{ v_i \mapsto d_i \mid 1 \leq i \leq n \} \]

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

\[ \varepsilon' = \varepsilon \oplus u \]

\[ \text{where } u \notin \text{dom}(\sigma) \]

\[ \text{and } \text{atr}(E) = \{ v_1, \ldots, v_n \}. \]

\[ \sigma' = \sigma \cup \{ u_E \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n \} \mid u \in \text{dom}(\sigma) \} \]

\[ \varepsilon'' = \varepsilon' \oplus u \text{ where } u \notin \text{dom}(\sigma) \]

\[ \text{and } \text{atr}(E) = \{ v_1, \ldots, v_n \}. \]

\[ \sigma' = \sigma \cup \{ u_E \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n \} \mid u \in \text{dom}(\sigma) \} \]

\[ \varepsilon'' = \varepsilon' \oplus u \text{ where } u \notin \text{dom}(\sigma) \]

\[ \text{and } \text{atr}(E) = \{ v_1, \ldots, v_n \}. \]

(v) Error Conditions

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

if, in (ii) or (iii),

\[ I[\text{expr}] \text{ is not defined for } \sigma, \text{ or} \]

\[ t_{\text{act}} \text{ is not defined for } (\sigma, \varepsilon), \text{ i.e. } t_{\text{act}}[\omega](\varepsilon, \sigma) = \emptyset \]

and

\[ \text{consumption is observed according to } (ii) \text{ or } (iii), \text{ but } \text{Snd} = \emptyset. \]

Examples:

\[ E[x + 0]/\text{act} \]

\[ E[true]/\text{act} \]

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

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

**SMC:**

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

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

\[ \sigma: \]

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

**\( \varepsilon: \)**

\( H \) for \( c \)

\[
\langle \langle \text{signal}, \text{env} \rangle \rangle \xrightarrow{H} \#
\]

\[
\begin{align*}
\text{• } I[\text{expr}] &\text{ not defined for } \sigma, \text{ or} \\
\text{• } t_{\text{new}} &\text{ is not defined for } (\sigma, \varepsilon) \\
\text{• } \text{consumption according to (ii) or (iii)} \\
\text{• } \text{Snd} = \emptyset
\end{align*}
\]

**Notions of Steps: The Step**

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

\[ c_1 \text{ calls } f() \text{ at } c_2, \text{ which calls } g() \text{ at } c_1 \text{ which in turn calls } h() \text{ for } c_2. \]

\[ \text{Is the completion of } h() \text{ a step?} \]

\[ \text{Or the completion of } f()? \]

\[ \text{Or doesn’t it play a role?} \]

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:
References

