Contents & Goals

Last Lecture:
- Ether
- System configuration

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
  - Examples for transformer
  - Run-to-completion Step
  - Putting It All Together
Where are we?

- **Wanted**: a labelled transition relation
  \[ (\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})}_{u_x} (\sigma', \varepsilon') \]
  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 machine state \(\sigma(u_x)(stC)\).

- **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 \( \Sigma_{BW} \) the set of system configurations over some \( \mathcal{S}_0, \mathcal{P}_0, Eth \).

We call a relation \( t \subseteq \mathcal{P}(\mathcal{S}) \times (\Sigma_{BW} \times Eth) \times (\Sigma_{BW} \times Eth) \) 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
  \[
  \text{Obs}_{t}[u_x](\sigma, \varepsilon) \in 2^{\mathcal{P}(\mathcal{S}) \times Eth \cup \{+, \} \times Eth}.
  \]
- An observation \( (u_{src}, u_{e}, (E, d), u_{dst}) \in \text{Obs}_{t}[u_x](\sigma, \varepsilon) \) represents the information that, as a "side effect" of \( u_x \) executing \( t \), an event (!) \( (E, d) \) has been sent from \( u_{src} \) to \( u_{dst} \).
  
  **Special cases**: creation/destruction.

Why Transformers?

- **Recall** the (simplified) syntax of transition annotations:
  \[
  \text{annot ::= [ } \langle \text{event} \rangle [ \text{'} \langle \text{guard} \rangle \text{'} ] [ \text{'} \langle \text{action} \rangle \text{'} ] \text{ ]}
  \]
- **Clear**: \( \langle \text{event} \rangle \) is from \( \mathcal{S} \) of the corresponding signature.
- **But**: What are \( \langle \text{guard} \rangle \) and \( \langle \text{action} \rangle \)?
  - UML can be viewed as being parameterized in expression language (providing \( \langle \text{guard} \rangle \)) and action language (providing \( \langle \text{action} \rangle \)).
  
  **Examples**:
  - **Expression Language**:
    - OCL
    - Java, C++, \ldots expressions
    - \ldots
  - **Action Language**:
    - UML Action Semantics, "Executable UML"
    - Java, C++, \ldots statements (plus some event send action)
    - \ldots
Transformers as Abstract Actions!

In the following, we assume that we’re given

- an expression language $Expr$ for guards, and
- an action language $Act$ for actions,

and that we’re given

- a semantics for boolean expressions in form of a partial function
  \[ I[e](\cdot, \cdot) : Expr \rightarrow (\Sigma_{BW} \times \mathcal{P}(\mathcal{C})) \rightarrow B \]
  which evaluates expressions in a given system configuration,
  
  Assuming $I$ to be partial is a way to treat “undefined” during runtime. If $I$ is not defined (for instance because of dangling-reference navigation or division-by-zero), we want to go to a designated “error” system configuration.

- a transformer for each action: for each $act \in Act$, we assume to have
  \[ t_{act} \subseteq \mathcal{P}(\mathcal{C}) \times (\Sigma_{BW} \times \mathcal{C}) \times (\Sigma_{BW} \times \mathcal{E}) \]

Expression/Action Language Examples

We can make the assumptions from the previous slide because instances exist:

- for OCL, we have the OCL semantics from Lecture 03. Simply remove the pre-images which map to “⊥”.
- for Java, the operational semantics of the SWT lecture uniquely defines transformers for sequences of Java statements.

We distinguish the following kinds of transformers:

- **skip**: do nothing — recall: this is the default action
- **send**: modifies $\varepsilon$ — interesting, because state machines are built around sending/consuming events
- **create/destroy**: modify domain of $\sigma$ — not specific to state machines, but let’s discuss them here as we’re at it
- **update**: modify own or other objects’ local state — boring
Action Language

In the following we consider

\[ \text{Act}_\mathcal{P} := \{ \text{step} \}
\]

- \{ update (exp_1, v, exp_2) | exp_1, exp_2 \in \text{OCLExp}, v \in V \}
- \{ send (exp_1, E, exp_2) | exp_1, exp_2 \in \text{OCLExp}, E \in E \}
- \{ create (c, exp, v) | c \in \mathcal{C}, exp \in \text{OCLExp}, v \in V \}
- \{ destroy (exp) | exp \in \text{OCLExp} \}

\[ \text{Exp}_\mathcal{P} : \text{OCL expressions over } \mathcal{P} \]

Transformer Examples: Presentation

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>op \text{update}(\sigma, v, \varepsilon)</td>
<td>e_v \varepsilon \sigma</td>
</tr>
<tr>
<td>intuitive semantics</td>
<td>...</td>
</tr>
<tr>
<td>well-typedness</td>
<td>...</td>
</tr>
<tr>
<td>semantics</td>
<td>((\sigma, \varepsilon), (\sigma', \varepsilon') \in \text{top}[u_x] \text{ iff } ... )</td>
</tr>
<tr>
<td></td>
<td>or ( \text{top}[u_x](\sigma, \varepsilon) = { (\sigma', \varepsilon') } \text{ where } ... )</td>
</tr>
<tr>
<td>observables</td>
<td>( \text{Obs}_{\text{top}}[u_x] = { ... }, \text{ not a relation, depends on choice } )</td>
</tr>
<tr>
<td>(error) conditions</td>
<td>Not defined if ...</td>
</tr>
</tbody>
</table>
**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**

\[ t[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\} \]

**observables**

\[ Obs_{\text{skip}}[u_x](\sigma, \varepsilon) = \emptyset \]

**error conditions**

Not defined if \( I[expr_1](\sigma, u) \) or \( I[expr_2](\sigma, u') \) not defined.

---

**Transformer: Update**

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>update(\expr_1, v, \expr_2)</td>
<td>\expr_1, v \Rightarrow \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**

\( \expr_1 : \tau_0 \) and \( v : \tau \in \text{attr}(C) \); \( \expr_2 : \tau \); \( \expr_1, \expr_2 \) obey visibility and navigability.

**semantics**

\[ t_{\text{update}}(\expr_1, v, \expr_2)[u_x](\sigma, \varepsilon) = \{(\sigma', \varepsilon)\} \]

where \( \sigma' = \sigma[u \mapsto \sigma(u)v \mapsto I[expr_2](\sigma, u)] \) with \( \sigma'' = \{\text{expr}_1(\sigma, u)\} \),

**observables**

\[ Obs_{\text{update}}[u_x](\sigma, \varepsilon) = \emptyset \]

**error conditions**

Not defined if \( I[expr_1](\sigma, u) \) or \( I[expr_2](\sigma, u') \) not defined.

\[ i.e. t_{\text{update}}(\sigma, u, v)(\sigma, \varepsilon) = \emptyset \]

\[ t_{\text{update}}(\sigma, u, v)(\sigma, \varepsilon) = \emptyset \]

\[ t_{\text{update}}(\sigma, u, v)(\sigma, \varepsilon) = \emptyset \]
### Update Transformer Example

**SMC:**

\[
\begin{array}{c}
\text{s1} \\
\text{\(x := x + 1\)} \\
\text{s2}
\end{array}
\]

**update**\((expr_1, v, expr_2)\)

\[
\begin{align*}
\text{t}_{\text{update}(expr_1, v, expr_2)}(\sigma, \epsilon) &= (\sigma[u \mapsto \sigma(v) \mapsto I[expr_2](\sigma, \epsilon)], \epsilon), \\
\text{u} &= I[expr_1](\sigma, \epsilon)
\end{align*}
\]

\[
\begin{array}{c}
\text{\(\sigma\):} \\
v_1 : C \\
x = 4 \\
y = 0
\end{array}
\]

\[
\begin{array}{c}
\text{\(\sigma'\):} \\
v_1 : C \\
x = 5 \\
y = 0
\end{array}
\]

\[
\begin{array}{c}
\text{\(\epsilon\):} \\
u = I[\text{expr_1}](\sigma, \epsilon)
\end{array}
\]

\[
\begin{array}{c}
\text{\(\epsilon'\):} \\
u = I[\text{expr_1}](\sigma, \epsilon)
\end{array}
\]

### Transformer: Send

**abstract syntax**

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

**concrete syntax**

\[
\text{!E(expr_1, ..., expr_n)}
\]

**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**

\(\text{expr}_{\text{dst}}: \tau_D, D \in \mathcal{D} \setminus \mathcal{E}; E \in \mathcal{E};\)

\(\text{atr}(E) = \{v_1: \tau_1, ..., v_n: \tau_n\}; \text{expr}_i: \tau_i, 1 \leq i \leq n;\)

all expressions obey visibility and navigability in \(C\)

**semantics**

\[
\begin{align*}
\text{t}_{\text{send}(E(expr_1, ..., expr_n), expr_{dst})}(\sigma, \epsilon) &= (\sigma', \epsilon'), \\
\text{where} & \quad \sigma' = \sigma \cup \{u \mapsto \{v_i \mapsto d_i | 1 \leq i \leq n\}\}; \\
\text{and where} & \quad (\sigma', \epsilon') = (\sigma, \epsilon) \oplus (u_{\text{expr}_{\text{dst}}}, \epsilon), \\
\end{align*}
\]

\[
\begin{align*}
\text{u} & \in \mathcal{D}(E) \text{ a fresh identity, i.e. } u \notin \text{dom}(\sigma), \\
\text{and where} & \quad (\sigma', \epsilon') = (\sigma, \epsilon) \text{ if } u_{\text{expr}_{\text{dst}}} \notin \text{dom}(\sigma), \\
\text{all expressions obey visibility and navigability in } C
\end{align*}
\]

**disjoint union**

our choice we could also consider it to be our task

**observables**

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

**error conditions**

\[
\begin{align*}
I[\text{expr}](\sigma, \epsilon) \text{ not defined for any } \\
\text{expr} \in \{\text{expr}_{\text{dst}}, \text{expr}_1, ..., \text{expr}_n\}
\end{align*}
\]
Send Transformer Example

\[ SM_C: \]

\[
\begin{align*}
&\text{send}(E(expr_1, \ldots, expr_n), expr_{dst}) \\
&f_{\text{send}}(expr_{src}, E(expr_1, \ldots, expr_n), expr_{dst})(s_1, s_2) = ...
\end{align*}
\]

\[ \sigma: \]

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

\[ \varepsilon: \]

\[
\begin{align*}
&x = 23
\end{align*}
\]

\[ \sigma': \]

\[
\begin{align*}
&u_{23} = u_4 \\
&w_1 \equiv x + 1
\end{align*}
\]

\[ \varepsilon': \]

\[
\begin{align*}
&w, x \equiv 6
\end{align*}
\]

References
References

