- Recall: Basic Causality Model
- Event Pool
  - insert, remove, clear, ready.
- System Configuration
  - implicit attributes: stable, st, and friends.
  - system state plus event pool
- Actions
  - simple action language.
  - transformer: effects of actions.
Roadmap: Chronologically

**Syntax:**
(i) UML State Machine Diagrams.
(ii) Def.: Signature with **signals**.
(iii) Def.: Core state machine.
(iv) Map UML State Machine Diagrams to core state machines.

**Semantics:**
The Basic Causality Model
(v) Def.: Ether (aka. event pool)
(vi) Def.: System configuration.
(vii) Def.: Event.
(viii) Def.: Transformer.
(ix) Def.: Transition system, computation.
(x) Transition relation induced by core state machine.
(xi) Def.: step, run-to-completion step.
(xii) Later: Hierarchical state machines.
15.3.12 StateMachine ([OMG, 2011b, 574])

- Event occurrences are detected, dispatched, and then processed by the state machine, one at a time.
- The semantics of event occurrence processing is based on the **run-to-completion assumption**, interpreted as **run-to-completion processing**.
- **Run-to-completion processing** means that an event [...] can only be taken from the pool and dispatched if the processing of the previous [...] is fully completed.
- The processing of a single event occurrence by a state machine is known as a **run-to-completion step**.
- Before commencing on a **run-to-completion step**, a state machine is in a **stable state** configuration with all entry/exit/internal-activities (but not necessarily do-activities) completed.
- The same conditions apply after the **run-to-completion step** is completed.
- Thus, an event occurrence will never be processed [...] in some intermediate and inconsistent situation.
- [IOW.] The **run-to-completion step** is the passage between two state configurations of the state machine.
- The **run-to-completion assumption** simplifies the transition function of the StM, since concurrency conflicts are avoided during the processing of event, allowing the StM to safely complete its **run-to-completion step**.
- The order of dequeuing is **not defined**, leaving open the possibility of modeling different priority-based schemes.
- Run-to-completion may be implemented in **various ways**. [...]

---

**Event occurrences are detected, dispatched, and then processed by the state machine, one at a time.**

**The semantics of event occurrence processing is based on the run-to-completion assumption, interpreted as run-to-completion processing.**

**Run-to-completion processing** means that an event [...] can only be taken from the pool and dispatched if the processing of the previous [...] is fully completed.

The processing of a single event occurrence by a state machine is known as a **run-to-completion step**.

Before commencing on a **run-to-completion step**, a state machine is in a **stable state** configuration with all entry/exit/internal-activities (but not necessarily do-activities) completed.

The same conditions apply after the **run-to-completion step** is completed.

Thus, an event occurrence will never be processed [...] in some intermediate and inconsistent situation.

[IOW.] The **run-to-completion step** is the passage between two state configurations of the state machine.

The **run-to-completion assumption** simplifies the transition function of the StM, since concurrency conflicts are avoided during the processing of event, allowing the StM to safely complete its **run-to-completion step**.

The order of dequeuing is **not defined**, leaving open the possibility of modeling different priority-based schemes.

Run-to-completion may be implemented in **various ways**. [...]

---
Example

\[ C \]

\[
\begin{array}{c|c}
  x : \text{Int} & p \\
\end{array}
\]

\[ D \]

\[
\begin{array}{c|c}
  0..1 & n \\
\end{array}
\]

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

\[ E \]

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

\[ F \]

\[ SMC: \]

\[ s_1 \]

\[ E[n \neq \emptyset] / x := x + 1; n \! F \]

\[ s_2 \]

\[ F / x := 0 \]

\[ s_3 \]

\[ \langle \text{f} \rangle \! \text{to} \! \text{seq} \! \text{f} \]

\[ / n := \emptyset \]

\[ :SM_D \]

\[ s_1 \]

\[ F / \]

\[ s_2 \]

\[ / p \! F \]

\[ \text{run-to-completion step} \]

\[ \text{step} \]

\[ \text{what has been captured?} \]

\[ \text{what has been sent out?} \]

\[ (\{E\}, \{F\}) \]

\[ \sigma_1, \varepsilon_1 \]

\[ (\sigma_2, \varepsilon_2) \]

\[ u_1 \]

\[ (\sigma_3, \varepsilon_3) \]

\[ u_2 \]

\[ (\sigma_4, \varepsilon_4) \]

\[ \langle \sigma_i, \varepsilon_i \rangle \]

\[ u_1 : C \]

\[ x = 27 \]

\[ st = s_1 \]

\[ stb = 1 \]

\[ p \]

\[ u_2 : D \]

\[ st = s_1 \]

\[ stb = 1 \]

\[ n \]

\[ u_3 : E \]

\[ x = 28 \]

\[ st = s_2 \]

\[ stb = 0 \]

\[ p \]

\[ n \]

\[ u_4 : F \]

\[ x = 28 \]

\[ st = s_3 \]

\[ stb = 0 \]

\[ p \]

\[ n \]
Ether
Recall: 15.3.12 StateMachine (OMG, 2011b, 563)

- The order of dequeuing is **not defined**, leaving open the possibility of modeling different (priority-based) schemes.
Ether and OMG (2011b)

The standard distinguishes (among others)

- **SignalEvent** (OMG, 2011b, 450) and **Reception** (OMG, 2011b, 447).

On **SignalEvents**, it says

> A signal event represents the receipt of an asynchronous signal instance. A signal event may, for example, cause a state machine to trigger a transition. (OMG, 2011b, 449) [...]

**Semantic Variation Points**

The means by which requests are transported to their target depend on the type of requesting action, the target, the properties of the communication medium, and numerous other factors.

In some cases, this is instantaneous and completely reliable while in others it may involve transmission delays of variable duration, loss of requests, reordering, or duplication.

(See also the discussion on page 421.) (OMG, 2011b, 450)

Our *ether* (→ in a minute) is a general representation of many possible choices.

**Often seen minimal requirement**: order of sending by one object is preserved.
Definition. Let $\mathcal{I} = (\mathcal{I}, \mathcal{C}, V, atr, \mathcal{E})$ be a signature with signals and $\mathcal{D}$ a structure.

We call a tuple $(Eth, ready, \oplus, \ominus, [\cdot])$ an ether over $\mathcal{I}$ and $\mathcal{D}$ if and only if it provides

- a ready operation which yields a set of events (i.e., signal instances) that are ready for a given object, i.e.
  \[
  \text{ready} : Eth \times \mathcal{D}(\mathcal{C}) \rightarrow 2^{\mathcal{D}(\mathcal{E})}
  \]

- a operation to insert an event for a given object, i.e.
  \[
  \oplus : Eth \times \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}) \rightarrow Eth
  \]

- a operation to remove an event, i.e.
  \[
  \ominus : Eth \times \mathcal{D}(\mathcal{E}) \rightarrow Eth
  \]

- an operation to clear the ether for a given object, i.e.
  \[
  [\cdot] : Eth \times \mathcal{D}(\mathcal{C}) \rightarrow Eth.
  \]
Example: FIFO Queue

A (single, global, shared, reliable) FIFO queue is an ether:

- \( Eth = \left( \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}) \right)^* \)
  
  the set of finite sequences of pairs \((u, e) \in \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E})\)

- \( \text{ready} : Eth \times \mathcal{D}(\mathcal{C}) \to 2^{\mathcal{D}(\mathcal{E})} \)

  \[
  (\varepsilon, u_2) \mapsto \begin{cases} 
  \{(u_2, e)\} \quad &\text{if } \varepsilon = (u_2, e).\varepsilon' \\
  \emptyset \quad &\text{otherwise}
  \end{cases}
  \]

- \( \oplus : Eth \times \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}) \to Eth \)

  \[
  (\varepsilon, u, e) \mapsto \varepsilon. (u, e)
  \]

- \( \ominus : Eth \times \mathcal{D}(\mathcal{E}) \to Eth \)

  \[
  (\varepsilon, e) \mapsto \begin{cases} 
  \varepsilon' \quad &\text{if } \varepsilon = (u, e), \varepsilon', u \in \mathcal{D}(\mathcal{C}) \\
  \varepsilon. (u, e) \quad &\text{otherwise}
  \end{cases}
  \]

- \( [\cdot] : Eth \times \mathcal{D}(\mathcal{C}) \to Eth \)

  \[ [\cdot] (\varepsilon, u) : \text{remove all } (u, e) \text{ elements from the given } \varepsilon, e \in \mathcal{D}(\mathcal{E}) \]
Other Examples

- One FIFO queue per active object is an ether.
  \[
  \mathcal{E} = D(c) \to (D(c) \times D(c))^* 
  \]

- One-place buffer.
  \[
  \mathcal{E} = \epsilon \cup (D(c) \times D(c)) 
  \]

- Priority queue.
  ...

- Multi-queues (one per sender).
  ,

- Trivial example: sink, “black hole”.
  ...

- Lossy queue (⊕ needs to become a relation then).

  ...

System Configuration
Definition. Let $\mathcal{S}_0 = (T_0, C_0, V_0, \text{atr}_0, \mathcal{E}_0)$ 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 $C \in C$.

A system configuration over $\mathcal{S}_0$, $\mathcal{D}_0$, and Eth is a pair

$$\sigma, \varepsilon \in \Sigma_{\mathcal{S}} \times \text{Eth}$$

where

- $\mathcal{S} = (T_0 \cup \{S_{MC} \mid C \in C_0\}, \quad C_0,$
  $$V_0 \cup \{(\text{stable} : \text{Bool}, -, \text{true}, \emptyset)\} \cup \{(st_{C} : S_{MC}, +, s_0, \emptyset) \mid C \in C\} \cup \{(\text{params}_E : E_{0,1}, +, \emptyset, \emptyset) \mid E \in \mathcal{E}_0\}, \quad \{C \mapsto \text{atr}_0(C) \cup \{\text{stable}, st_{C}\} \cup \{\text{params}_E \mid E \in \mathcal{E}_0\} \mid C \in C\}, \quad \mathcal{E}_0\)$

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

- $\sigma(u)(r) \cap \mathcal{D}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_0$. 

\[\text{if } \text{Bool} \notin T_0 \text{ then add it and use } \mathcal{D} \text{ for } \text{Bool} = \{0,1\}\]
System Configuration: Example

$\mathcal{S}_0 = (\mathcal{T}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E}_0), \mathcal{D}_0; \quad (\sigma, \varepsilon) \in \Sigma^\mathcal{D}_0 \times Eth$ where

- $\mathcal{I} = (\mathcal{T}_0 \cup \{S_{MC} \mid C \in \mathcal{C}\}, \mathcal{C}_0,$
  
  $V_0 \cup \{\langle stable : Bool, -, true, \emptyset \rangle \} \cup \{\langle st_C : S_{MC}, +, s_0, \emptyset \rangle \mid C \in \mathcal{C} \}$

  $\cup \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset \rangle \mid E \in \mathcal{E}_0 \},$

  $\{C \mapsto atr_0(C) \cup \{stable, st_C\} \cup \{params_E \mid E \in \mathcal{E}_0 \} \mid C \in \mathcal{C} \}, \mathcal{E}_0 \}$

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

- $\sigma(u)(r) \cap \mathcal{D}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_0.$

$\mathcal{S}_{MC}:$

$\mathcal{I}_0 = (\{\text{int}, \{C\}, \{x:\text{int}, b:\text{int}, a:\text{int}, c:C_{0,1} \}, \} \mapsto V_0$

$\{c \mapsto \{\text{stable}, st_C, \{params_E \mid E \in \mathcal{E}_{0,1} \} \}$

$\{E \mapsto \{b,c\}, \{a,3\}\} \mapsto \{\text{int}, \{\text{stable}, st_C, \}}$

$\mathcal{D}(S_{MC}) = \{\text{idle}, s_2, s_3 \}$

\[
\begin{array}{c}
\sigma \in \Sigma^\mathcal{D}_0:
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\mathcal{I} = (\{\text{int}, \text{Bool}, S_{MC}\}, \{C\},
\{x, \text{stable}, st_C, \{params_E \mid E \in \mathcal{E}_{0,1} \}\}$

$\{E \mapsto \{b,c\}, \{a,3\}\} \mapsto \{\text{int}, \{\text{stable}, st_C, \}}$

$\mathcal{D}(S_{MC}) = \{\text{idle}, s_2, s_3 \}$

\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\mathcal{I} = (\{\text{int}, \text{Bool}, S_{MC}\}, \{C\},
\{x, \text{stable}, st_C, \{params_E \mid E \in \mathcal{E}_{0,1} \}\}$

$\{E \mapsto \{b,c\}, \{a,3\}\} \mapsto \{\text{int}, \{\text{stable}, st_C, \}}$

$\mathcal{D}(S_{MC}) = \{\text{idle}, s_2, s_3 \}$

\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\mathcal{I} = (\{\text{int}, \text{Bool}, S_{MC}\}, \{C\},
\{x, \text{stable}, st_C, \{params_E \mid E \in \mathcal{E}_{0,1} \}\}$

$\{E \mapsto \{b,c\}, \{a,3\}\} \mapsto \{\text{int}, \{\text{stable}, st_C, \}}$

$\mathcal{D}(S_{MC}) = \{\text{idle}, s_2, s_3 \}$

\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\mathcal{I} = (\{\text{int}, \text{Bool}, S_{MC}\}, \{C\},
\{x, \text{stable}, st_C, \{params_E \mid E \in \mathcal{E}_{0,1} \}\}$

$\{E \mapsto \{b,c\}, \{a,3\}\} \mapsto \{\text{int}, \{\text{stable}, st_C, \}}$

$\mathcal{D}(S_{MC}) = \{\text{idle}, s_2, s_3 \}$

\end{array}
\end{array}
\]
System Configuration Step-by-Step

- We start with some signature with signals $I_0 = (T_0, C_0, V_0, atr_0, E)$.

- A system configuration is a pair $(\sigma, \varepsilon)$ which comprises a system state $\sigma$ wrt. $I$ (not wrt. $I_0$).

- Such a system state $\sigma$ wrt. $I$ provides, for each object $u \in \text{dom}(\sigma)$,
  - values for the explicit attributes in $V_0$,
  - values for a number of implicit attributes, namely
    - a stability flag, i.e. $\sigma(u)(stable)$ is a boolean value,
    - a current (state machine) state, i.e. $\sigma(u)(st)$ denotes one of the states of core state machine $M_C$,
    - a temporary association to access event parameters for each class, i.e. $\sigma(u)(params_E)$ is defined for each $E \in \mathcal{E}$.

- For convenience require: there is no link to an event except for $params_E$. 
Definition. Let \((\sigma, \varepsilon)\) be a system configuration over some \(\mathcal{S}_0, \mathcal{D}_0, Eth\).

We call an object \(u \in \text{dom}(\sigma) \cap \mathcal{D}(C_0)\) stable in \(\sigma\) if and only if

\[
\sigma(u)(\text{stable}) = \text{true}.
\]

And unstable otherwise.
Where are we?

\[
\begin{array}{c}
\text{C} \\
x : \text{Int} \\
p \\
0..1
\end{array}
\rightarrow
\begin{array}{c}
\text{D} \\
n \\
0..1
\end{array}
\]

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

\[\text{E} \]

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

\[\text{F} \]

\[\text{SM}_C: \]

\[s_1 \xrightarrow{E[n \neq \emptyset]/x := x + 1; n! F} s_2\]

\[s_1 \xrightarrow{F/x := 0} s_3 \xrightarrow{n := \emptyset} s_2\]

\[s_1 \xrightarrow{F/} s_2 \xrightarrow{\text{SM}_D} \]

\[(\sigma_1, \varepsilon_1) \xrightarrow{u_1} (\sigma_2, \varepsilon_2) \xrightarrow{u_1} (\sigma_3, \varepsilon_3) \xrightarrow{u_2} (\sigma_4, \varepsilon_4)\]

\[u_1: C \]

\(x = 27\)

\(st = s_1\)

\(stb = 1\)

\(p \uparrow n\)

\[u_2: D \]

\(st = s_1\)

\(stb = 1\)

\(p \uparrow n\)

\[u_3: E \]

\(x = 28\)

\(st = s_2\)

\(stb = 0\)

\(p \uparrow n\)

\[u_4: F \]

\(x = 28\)

\(st = s_3\)

\(stb = 1\)

\(p \uparrow n\)
Transformer
Recall

- The (simplified) syntax of transition annotations:

\[
\text{annot ::= [ \langle \text{event} \rangle \ [\{\langle \text{guard} \rangle \} \] \ [\{\langle \text{action} \rangle \} \ ]}
\]

- **Clear**: \(\langle \text{event} \rangle\) is from \(E\) 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++, ... expressions
    - ...

  - **Action Language**:
    - UML Action Semantics, “Executable UML”
    - Java, C++, ... statements (plus some event send action)
    - ...
    - ↗
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

$$\llbracket \cdot \rrbracket : Expr \times \Sigma \times D(C) \rightarrow \mathbb{B}$$

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

$$t_{act} \subseteq D(C) \times (\Sigma \times Eth) \times (\Sigma \times Eth)$$
**Definition.**
Let $\Sigma^D$ the set of system configurations over some $\mathcal{I}_0$, $\mathcal{D}_0$, $Eth$. We call a relation
\[ t \subseteq (\mathcal{D}(\mathcal{C}) \times (\Sigma^D \times Eth) \times (\Sigma^D \times Eth) \]
a (system configuration) **transformer**.

**Example:**
- $t[u_x](\sigma, \varepsilon) \subseteq \Sigma^D \times Eth$ is
  - the set (!) of the **system configurations**
  - which **may** result from **object** $u_x$
  - **executing** transformer $t$.
- $t_{\text{skip}}[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}$
- $t_{\text{create}}[u_x](\sigma, \varepsilon) : \text{add a previously non-alive object to } \sigma$ (id. non-det, closer)
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**

$$Obs_t[u_x](\sigma, \varepsilon) \in 2^{(\mathcal{D}(\mathcal{E}) \cup \{\ast, +\}) \times \mathcal{D}(\mathcal{C})}.$$  

An observation

$$(u_e, u_{dst}) \in Obs_t[u_x](\sigma, \varepsilon)$$

represents the information that, as a “side effect” of object $u_x$ executing $t$ in system configuration $(\sigma, \varepsilon)$, the event $u_e$ has been sent to $u_{dst}$.

**Special cases**: creation (’$\ast$’) / destruction (’$+$’).
In the following we use

\[
\text{Act}\mathcal{S} = \{\text{skip}\} \\
\cup \{\text{update}(\text{expr}_1, v, \text{expr}_2) \mid \text{expr}_1, \text{expr}_2 \in \text{Expr}\mathcal{S}, v \in \text{atr}\} \\
\cup \{\text{send}(E(\text{expr}_1, \ldots, \text{expr}_n), \text{expr}_{\text{dst}}) \mid \text{expr}_i, \text{expr}_{\text{dst}} \in \text{Expr}\mathcal{S}, E \in \mathcal{E}\} \\
\cup \{\text{create}(C, \text{expr}, v) \mid C \in \mathcal{C}, \text{expr} \in \text{Expr}\mathcal{S}, v \in V\} \\
\cup \{\text{destroy}(\text{expr}) \mid \text{expr} \in \text{Expr}\mathcal{S}\}
\]

and OCL expressions over \(\mathcal{S}\) (with partial interpretation) as \(\text{Expr}\mathcal{S}\).
### Transformer Examples: Presentation

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>op</td>
</tr>
</tbody>
</table>
|intuitive semantics|...
|well-typedness|...

#### semantics

\[
((\sigma, \varepsilon), (\sigma', \varepsilon')) \in t_{op}[u_x] \text{ iff } \ldots
\]

or

\[
t_{op}[u_x](\sigma, \varepsilon) = \{(\sigma', \varepsilon') \mid \text{where } \ldots \}
\]

#### observables

\[
Obs_{op}[u_x] = \{\ldots\}
\]

#### (error) conditions

Not defined if \ldots
### Transformer: Skip

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>Concrete Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>skip</code></td>
<td><code>skip</code></td>
</tr>
</tbody>
</table>

| Intuitive Semantics | `do nothing` |

| Well-Typedness | `./*` |

<table>
<thead>
<tr>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{skip}}[u_x](\sigma, \varepsilon) = {(\sigma, \varepsilon)} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Obs}_{\text{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>update(expr₁, v, expr₂)</td>
<td>expr₁.v := expr₂</td>
</tr>
</tbody>
</table>

**Intuitive Semantics**

*Update attribute* `v` *in the object denoted by* `expr₁` *to the value denoted by* `expr₂`.

**Well-Typedness**

`expr₁ : T_C` *and* `v : T ∈ atr(C)`; `expr₂ : T`;

`expr₁, expr₂` *obey visibility and navigability*

**Semantics**

\[
\begin{align*}
t_{update}(expr₁, v, expr₂)[u_x](σ, ε) &= \{(σ', ε)\} \\
\text{where } σ' &= σ[u → σ(u)[v → I[expr₂](σ, u_x)]]
\end{align*}
\]

\[
u = I[expr₁](σ, u_x).
\]

**Observables**

\[
\text{Obs}_{update}(expr₁, v, expr₂)[u_x] = ∅
\]

**Error Conditions**

Not defined if `I[expr₁](σ, u_x)` *or* `I[expr₂](σ, u_x)` *not defined.*
Update Transformer Example

\[ SM_C: \]

\[
\begin{align*}
S_1: & \quad /x := x + 1 \\
S_2: & \quad \text{self}.x := \text{self}.x + 1 \\
\text{expr}_1: & \quad v := \text{int} \\
\text{expr}_2: & \quad \text{int}
\end{align*}
\]

\[
t_{\text{update}}(\text{expr}_1, v, \text{expr}_2)[u_x](\sigma, \varepsilon) = (\sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto I[\text{expr}_2](\sigma, u_x)]], \varepsilon), u = I[\text{expr}_1](\sigma, u_x)
\]

\[ \sigma: \]

\[
\begin{align*}
u_1 &: C \\
x &= 4 \\
y &= 0 \\
\text{table} &= 0 \\
\text{stk} &= \varepsilon
\end{align*}
\]

\[ \varepsilon: \]

\[
\begin{align*}
\text{expr}_1 & : C \\
\text{expr}_2 & : \text{int}
\end{align*}
\]

\[ \sigma': \]

\[
\begin{align*}
u_1 &: C \\
x &= 5 \\
y &= 0 \\
\text{table} &= 0 \\
\text{stk} &= \varepsilon
\end{align*}
\]

\[
u = I[\text{expr}_1](\sigma, u_1)
\]

\[
= I[\text{self}.x + 1](\sigma, \{\text{self}.x + 1\})
\]

\[
= u_1
\]

\[
u = I[\text{self}.x + 1](\sigma, \{\text{self}.x + 1\})
\]

\[
= 5
\]

\[
\varepsilon' = \varepsilon
\]
### Transformer: Send

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{send}(E(expr_1, \ldots, expr_n), expr_{dst}) )</td>
<td></td>
</tr>
</tbody>
</table>

### 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

\( E \in \mathcal{E} \); \( atr(E) = \{v_1 : T_1, \ldots, v_n : T_n\} \); \( expr_i : T_i, 1 \leq i \leq n \);

\( expr_{dst} : T_D, C, D \in \mathcal{C} \setminus \mathcal{E} \);

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

### Semantics

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

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

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

\( u \in \mathcal{D}(E) \) a fresh identity, i.e. \( u \not\in \text{dom}(\sigma) \),

and where \( (\sigma', \varepsilon') = (\sigma, \varepsilon) \) if \( u_{dst} \not\in \text{dom}(\sigma) \).

### Observables

\( \text{Obs}_{\text{send}}[u_x] = \{(u_e, u_{dst})\} \)

### (Error) Conditions

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

\[ \mathbf{SM}_C: \]

\[ /n! F(x + 1) \]

\[ \vdash \]

\[ s_1 \rightarrow s_2 \]

\[ t_{\text{send}}(\text{expr}_{\text{src}}, E(\text{expr}_1, \ldots, \text{expr}_n), \text{expr}_{\text{dst}})[u_x](\sigma, \varepsilon) \ni (\sigma', \varepsilon') \text{ iff } \varepsilon' = \varepsilon \oplus (u_{\text{dst}}, u); \]

\[ \sigma' = \sigma \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}\}; u_{\text{dst}} = I[\text{expr}_{\text{dst}}](\sigma, u_x) \in \text{dom}(\sigma); \]

\[ d_i = I[\text{expr}_i](\sigma, u_x), 1 \leq i \leq n; u \in \mathcal{D}(E) \text{ a fresh identity}; \]

\(\sigma:\)

\(\frac{u_1 : C}{x = 5}\)

: \(\sigma'\)

\(\varepsilon:\)

: \(\varepsilon'\)
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 of 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 (later),

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.
A **ether** is an abstract representation of different possible “event pools” like
- FIFO queues (shared, or per sender),
- One-place buffers,
- ...

A **system configuration** consists of
- an **event pool** (pending messages),
- a **system state** over a signature with **implicit attributes** for
  - current state,
  - stability,
  - etc.

Transitions are labelled with **actions**, the effect of actions is explained by **transformers**, transformers may modify **system state** and **ether**.
References
References
