Contents & Goals

Last Lecture:
- System configuration
- Transformer
- Action language: skip, update

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:**
  - Action Language: send (create/destroy later)
  - Run-to-completion Step
  - Putting It All Together
Transformer Cont’d
### 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

. . .

#### Semantics

\[
\ell_{\text{tx}}(\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}
\]

#### Observables

\[
\text{Obs}_{\text{tx}}(\sigma, \varepsilon) = \emptyset
\]

#### (Error) Conditions
Transformer: Update

**abstract syntax**

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

**concrete syntax**

\[ \text{expr}_1 := \text{expr}_2 \]

**intuitive semantics**

*Update attribute* \( v \) *in the object denoted by* \( \text{expr}_1 \) *to the value denoted by* \( \text{expr}_2 \).

**well-typedness**

\( \text{expr}_1 : \tau_C \) and \( v : \tau \in \text{atr}(C) \); \( \text{expr}_2 : \tau \);
\( \text{expr}_1, \text{expr}_2 \) obey visibility and navigability

**semantics**

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

where \( \sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto I[\text{expr}_2](\sigma, u_x)]] \)

with \( u = I[\text{expr}_1](\sigma, u_x) \).

**observables**

\[ \text{Obs}_{\text{update}}(\text{expr}_1, v, \text{expr}_2)[u_x] = \emptyset \]

**(error) conditions**

Not defined if \( I[\text{expr}_1](\sigma, u_x) \) or \( I[\text{expr}_2](\sigma, u_x) \) not defined.
Update Transformer Example

\[ S_M C: \]

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

\[ \llbracket x + 1 \rrbracket (\sigma, u_x) = \llbracket x + 1 \rrbracket (\sigma, u_x) = 5 \]

\[
\begin{align*}
\sigma: & \quad u_1 : C \\
& \quad x = 4 \\
& \quad y = 0 \\
\end{align*}
\]

\[
\begin{align*}
\text{\( t_{\text{update}} \)} & \quad u_x = u_1 \\
\end{align*}
\]

\[
\begin{align*}
\varepsilon : & \quad v = I[\llbracket x + 1 \rrbracket](\sigma,u_x) \\
& = I_{\varepsilon_{\llbracket x \rrbracket}}[\llbracket x + 1 \rrbracket](\sigma,\{x\mapsto u_1\},u_x) \\
& = \beta(\varepsilon_{\llbracket x \rrbracket})[u_x = u_1] \\
\end{align*}
\]

\[
\begin{align*}
\varepsilon' & = \varepsilon \\
\end{align*}
\]
Transformer: Send

abstract syntax

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

concrete syntax

\[
\text{expr}_{dst} \downarrow E(expr_1, \ldots, expr_n)
\]

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

\( expr_{dst} : \tau_D, \quad C, D \in \mathcal{C} \setminus \mathcal{E}; \quad E \in \mathcal{E}; \quad \text{atr}(E) = \{v_1 : \tau_1, \ldots, v_n : \tau_n\}; \quad expr_i : \tau_i, \quad 1 \leq i \leq n; \)

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

semantics

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

iff \( \sigma' = \sigma \uplus \{ u \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n\} \}; \quad \varepsilon' = \varepsilon \uplus (u_{dst}, u) \);

if \( u_{dst} = I[expr_{dst}][\sigma, u_x] \in \text{dom}(\sigma) \);

\[ d_i = I[expr_i][\sigma, u_x] \text{ for } 1 \leq i \leq n; \]

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

and where \( (\sigma', \varepsilon') = (\sigma, \varepsilon) \) if \( u_{dst} \notin \text{dom}(\sigma) \).

observables

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

(error) conditions

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

$\mathcal{SM}_C$:

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

$\sigma' = \sigma \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}\}$;

$u_{\text{dst}} = I[\text{expr}_\text{dst}](\sigma,u_x) \in \text{dom}(\sigma)$;

$d_i = I[\text{expr}_i](\sigma,u_x), 1 \leq i \leq n$;

$u \in \mathcal{D}(E)$ a fresh identity;

$\sigma$: 

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

$\varepsilon$: 

\[
\begin{array}{c}
\vdash
\end{array}
\]
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

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

- **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 $i \in \mathbb{N}_0$. 

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 [?] (i.e. the one realised by the Rhapsody code generation) where the standard is ambiguous or leaves choices.
Definition. Let $I_0 = (T_0, C_0, V_0, atr_0, \mathcal{E})$ be a signature with signals (all classes active), $D_0$ a structure of $I_0$, and $(Eth, ready, \ominus, \Theta, [\cdot])$ an ether over $I_0$ and $D_0$.

Assume there is one core state machine $M_C$ per class $C \in C$.

We say, the state machines induce the following labelled transition relation on states $S := (\Sigma_{T_0} \cup \{\#\} \times Eth)$ with actions $A := \left(2^{\mathcal{C}} \times (\mathcal{E} \cup \{\perp\})Evs(\mathcal{E}, D) \times \mathcal{C}\right)^2$:

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

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

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

if

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

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

- \(u\) 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)

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

and

- the system configuration doesn't change, i.e. \(\sigma' = \sigma \setminus \{u_E \mapsto \sigma(u_E)\}\)

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

\[\varepsilon' = \varepsilon \ominus u_E,\]
Example: Discard

$SM_C$: $\states 1 \rightarrow \states 2$

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

$H/z := y/x$

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

\[ N \]

\[ C \]

\[ y : \text{Int} \]

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

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

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

\[ G, J \]

\[ H \]

\[ \sigma : c : C \]

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

\[ \text{st} = s_1 \]

\[ \text{stable} = 1 \]

\[ \varepsilon : J \text{ for } c, \]

\[ G \text{ for } c, \]

\[ \langle (\langle c \rangle, \langle c \rangle) \rangle \]

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

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

\[ \forall (s, F, expr, act, s') \in \rightarrow (SM_C) : F \neq E \lor I[expr](\sigma) = 0 \checkmark \]

\[ \sigma(u)(\text{stable}) = 1 \checkmark \sigma(u)(\text{st}) = s_1 \]

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

\[ \text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset \]
(ii) Dispatch

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

- \(u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \land \exists u_E \in \mathcal{D}(E) : u_E \in \text{ready}(\varepsilon, u)\)
- \(u\) 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.
  \[\exists (s, F, \text{expr}, \text{act}, s') \in \rightarrow (\mathcal{S}\mathcal{M}_C) : F = E \land I[\text{expr}](\tilde{\sigma}) = 1\]
  where \(\tilde{\sigma} = \sigma[u.\text{params}_E \mapsto u_E]\).

and

- \((\sigma', \varepsilon')\) results from applying \(t_{act}\) to \((\sigma, \varepsilon)\) and removing \(u_E\) from the ether, i.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 \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.
    \[\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{\text{act}}(\tilde{\sigma}, \varepsilon \ominus u_E).\]
Example: Dispatch

\[ \frac{x > 0}{x := x - 1; n ! J} \]

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

\[ SM_C: \]

\begin{align*}
S_1 & \quad \xrightarrow{G[x > 0] / x := y} \quad S_2 \\
\end{align*}

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

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

\[ H \]

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

\[ G, J \]

\[ n \]

\[ C \]

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

\[ y : \text{Int} \]

\[ \langle \langle env \rangle \rangle \]

\[ n \]

\[ C \]

\[ \sigma : \]

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

\[ \varepsilon : \]

\[ G \quad \text{for} \quad c \]

\[ \text{ready}(\varepsilon, c) = y \]

\[ \sigma : \]

\[ \sigma(\text{stable}) = 1 \]

\[ \sigma(\text{st}) = s_1 \]

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

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

\[ \text{cons} = \{ (u, (E, \sigma(u_E))) \} \]

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

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

if

- there is an unstable object \(u\) of a class \(C\), 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.
  \[\exists (s,ס, expr, act, s') \in \rightarrow (SM_C) : I[\llbracket expr \rrbracket](\sigma) = 1\]

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

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

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

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

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

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

\[ G, J \]

\[ C \]

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

\[ y : \text{Int} \]

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

\[ n \]

\[ 0, 1 \]

\[ \sigma : \]

\[ \begin{array}{l}
  x = 2, z = 0, y = 2 \\
  st = s_2 \\
  stable = 0
\end{array} \]

\[ \varepsilon : \]

\[ \begin{array}{l}
  \exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C) : \sigma(u)(\text{stable}) = 0 \\
  \exists (s, \_, \text{expr}, \text{act}, s') \in \rightarrow (\mathcal{SM}_C) : \\
  I[\text{expr}](\sigma) = 1 \\
  \mathbf{b}(u)(\text{stable}) = 1 \Rightarrow \sigma(u)(\text{st}) = s, \checkmark \\
  (\sigma'', \varepsilon') = t_{\text{act}}(\sigma, \varepsilon), \\
  \sigma' = \sigma''[u.st \mapsto s', u.\text{stable} \mapsto b] \\
  \text{cons} = \emptyset, \text{Snd} = \text{Obst}_{\text{act}}(\sigma, \varepsilon) \\
\end{array} \]
(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

- environment event $E \in \mathcal{E}_{env}$ is spontaneously sent to an alive object $u \in \mathcal{D}(\sigma)$, i.e.

  $$\sigma' = \sigma \cup \{u_E \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}, \varepsilon' = \varepsilon \oplus u_E$$

  where $u_E \notin \text{dom}(\sigma)$ and $\text{atr}(E) = \{v_1, \ldots, v_n\}$.

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

  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

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

\( SM_C: \)

\( s_1 \rightarrow \)

\( G[x > 0]/x := y \rightarrow s_2 \)

\( H/z := y/x \)

\[ s_1 \]

\[ s_2 \]

\( \langle \langle \text{signal, env} \rangle \rangle \)

\( H \)

\( \langle \langle \text{signal} \rangle \rangle \)

\( G, J \)

\( n \rightarrow C \)

\( x, z : \text{Int} \)

\( y : \text{Int} \langle \langle \text{env} \rangle \rangle \)

\( \sigma: \)

\[ c : C \]

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

\[ st = s_2 \]

\[ stable = 1 \]

\( \varepsilon: \)

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

\( \varepsilon' = \varepsilon \oplus u_E \) where \( u_E \notin \text{dom}(\sigma) \) and \( \text{atr}(E) = \{ v_1, \ldots, v_n \} \)

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

\( \text{cons} = \emptyset, \)

\( \text{Snd} = \{ (\text{env}, E(d)) \} \).
(v) Error Conditions

\[ S \xrightarrow{(cons, Snd)} u \xrightarrow{\#} \]

if, in (ii) or (iii),

- \( I[expr] \) is not defined for \( \sigma \), or
- \( t_{act} \) is not defined for \( (\sigma, \varepsilon) \),

and

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

Examples:

\[ E[x/0]/act \xrightarrow{} s_2 \]

\[ E[true]/act \xrightarrow{} s_3 \]

\[ E[expr]/x := x/0 \xrightarrow{} s_2 \]
Example: Error Condition

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

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

\[ H \]

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

\[ G, J \]

\[ n \]

\[ C \]

\[ 0, 1 \]

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

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

\[ \sigma : \]

\[ c : C \]

\[ x = 0, z = 0, y = 27 \]

\[ st = s_2 \]

\[ stable = 1 \]

\[ \varepsilon : \]

\[ H \text{ for } c \]

- \( I[expr] \) 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 \((\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})}{_u} (\sigma', \varepsilon')\) 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.
Note: we call one evolution \((\sigma, \varepsilon) \xrightarrow{u} (\sigma', \varepsilon')\) 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:**

\[ E[x > 0]/ \]

\[ /x := x - 1 \]

\[ \sigma: \]

\[
| : C \\
| \hline
| x = 2 |
\]

\[ \varepsilon: \]

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

$$(\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} u_0 \rightarrow \ldots \rightarrow (\text{cons}_{n-1}, \text{Snd}_{n-1}) u_{n-1} \rightarrow (\sigma_n, \varepsilon_n), \quad n > 0,$$

be a finite (!), non-empty, maximal, consecutive sequence such that

- object $u$ is alive in $\sigma_0$,
- $u_0 = u$ and $\text{(cons}_0, \text{Snd}_0)$ indicates dispatching to $u$, i.e. $\text{cons} = \{(u, \vec{v} \mapsto \vec{d})\}$,
- there are no receptions by $u$ in between, i.e.

$$\text{cons}_i \cap \{u\} \times \text{Evs}(\mathcal{E}, \mathcal{D}) = \emptyset, i > 1,$$

- $u_{n-1} = u$ and $u$ is stable only in $\sigma_0$ and $\sigma_n$, i.e.

$$\sigma_0(u)(\text{stable}) = \sigma_n(u)(\text{stable}) = 1 \quad \text{and} \quad \sigma_i(u)(\text{stable}) = 0 \quad \text{for} \quad 0 < i < n,$$
**Proposal:** Let

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

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 \(1 \leq i \leq N\).
**Proposal:** Let

\[
(\sigma_0, \varepsilon_0) \xrightarrow{\ (\text{cons}_0, \text{Snd}_0) \ } u_0 \xrightarrow{\ . \ . \ .} \xrightarrow{\ (\text{cons}_{n-1}, \text{Snd}_{n-1}) \ } (\sigma_n, \varepsilon_n), \quad n > 0,
\]

be a finite (!), non-empty, maximal, consecutive sequence such that

- object \(u\) is alive in \(\sigma_0\),
- \(u_0 = u\) and \((\text{cons}_0, \text{Snd}_0)\) indicates dispatching to \(u\), i.e. \(\text{cons} = \{(u, \vec{v} \mapsto \vec{d})\}\),
- there are no receptions by \(u\) in between, i.e.

  \[
  \text{cons}_i \cap \{u\} \times \text{Evs}(\mathcal{E}, \mathcal{D}) = \emptyset, \quad i > 1,
  \]
- \(u_{n-1} = u\) and \(u\) is stable only in \(\sigma_0\) and \(\sigma_n\), i.e.

  \[
  \sigma_0(u)(\text{stable}) = \sigma_n(u)(\text{stable}) = 1 \quad \text{and} \quad \sigma_i(u)(\text{stable}) = 0 \quad \text{for} \quad 0 < i < n,
  \]

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 \(\sigma_0(u)\)).