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: 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>

<table>
<thead>
<tr>
<th>intuitive semantics</th>
<th><code>do nothing</code></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>well-typedness</th>
<th><code>./.</code></th>
</tr>
</thead>
</table>

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

| observables | \( \text{Obs}_{\text{skip}}[u_x](\sigma, \varepsilon) = \emptyset \) |

| (error) conditions | |

**Transformer Cont’d**
abstract syntax
update(expr₁, v, expr₂)

concrete syntax
expr₁ = expr₂

intuitive semantics
Update attribute v in the object denoted by expr₁ to the value denoted by expr₂.

well-typedness
expr₁ : τ₁ and v : τ ∈ atr(C); expr₂ : τ;
expr₁, expr₂ obey visibility and navigability
semantics
$t_{update}(expr₁, v, expr₂)(σ, ε) = \{(σ', ε)\}$
where $σ' = σ[u ↦→ σ(u)[v ↦→ I[expr₂](σ, u)]$ with $u = I[expr₁](σ, u_x)$.

observables
$Obs_{update}(expr₁, v, expr₂)(u_x) = \emptyset$

(error) conditions
Not defined if $I[expr₁](σ, β)$ or $I[expr₂](σ, β)$ not defined.

Update Transformer Example

$SM_C$:

\[
 s_1 \quad \vdash x := x + 1 \quad s_2
\]

$\{\text{implicitly self}\}$

$t_{update}(expr₁, v, expr₂)(u_x)(σ, ε) = (σ' = σ[u ↦→ σ(u)[v ↦→ I[expr₂](σ, u)]], ε), u = I[expr₁](σ, u_x)$

$Ⅹ+Ⅰ(σ_0) = Ⅹ+Ⅰ(σ_0) = 5$

\[
σ: \quad \begin{array}{l}
  \vdash x : C \\
  x = 4 \\
  y = 0
\end{array}
\]

\[
ε: \quad \begin{array}{l}
  v = \{\text{modified}\} \\
  v = v'
\end{array}
\]

$\vdash x : C \\
  x = 5$

$ε' = ε$
abstract syntax
\[ \text{send}(E(expr_1, \ldots, expr_n), expr_{dst}) \]

concrete syntax
\[ \mathcal{E}_d = E(expr_1, \ldots, expr_n) \]

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

semantics
\[
\begin{align*}
\langle \sigma', \varepsilon' \rangle & \in \text{t}_{\text{send}}(E(expr_1, \ldots, expr_n), expr_{dst})[u_x](\sigma, \varepsilon) \\
& \text{iff } \sigma' = \sigma \cup \{ (v_i \mapsto d_i) \mid 1 \leq i \leq n \}; \; \varepsilon' = \varepsilon \oplus (u_{dst}, u); \; \\
& \text{if } u_{dst} = I[expr_{dst}](\sigma, u_x) \in \text{dom}(\sigma), \; d_i = I[expr_i](\sigma, u_x) \text{ for } 1 \leq i \leq n, \; \\
& u \in \mathcal{D}(E) \text{ a fresh identity, i.e. } u \notin \text{dom}(\sigma), \; \\
& \text{and where } \langle \sigma', \varepsilon' \rangle = (\sigma, \varepsilon) \text{ if } u_{dst} \notin \text{dom}(\sigma). 
\end{align*}
\]

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

\[ SM_C: \]

\[ \sigma: \]

\[ u_1 : C, \; x = 5 \]

\[ \varepsilon: \]

\[ \mathcal{E}_d = n!F(x + 1) \]

\[ t_{\text{send}}(expr_{dst}, E(expr_1, \ldots, expr_n), expr_{dst})[u_x](\sigma, \varepsilon) \text{ iff } \varepsilon' = \varepsilon \oplus (u_{dst}, u); \]

\[ \sigma' = \sigma \cup \{ (v_i \mapsto d_i) \mid 1 \leq i \leq n \}; \; u_{dst} = I[expr_{dst}](\sigma, u_x) \in \text{dom}(\sigma); \; \\
& d_i = I[expr_i](\sigma, u_x), \; 1 \leq i \leq n; \; u \in \mathcal{D}(E) \text{ a fresh identity}. \]
Sequential Composition of Transformers

- **Sequential composition** $t_1 \circ t_2$ of transformers $t_1$ and $t_2$ is canonically defined as

$$\left(t_2 \circ t_1\right)[u_x](\sigma, \varepsilon) = t_2[u_x]t_1[u_x](\sigma, \varepsilon)$$

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 of the two intermediate “micro steps” is not defined.

Transformers And Denotational Semantics

**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.
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

$$
\begin{align*}
    s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots
\end{align*}
$$

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

Transition Relation, Computation

Step and Run-to-completion Step
**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.

---

**From Core State Machines to LTS**

**Definition.** Let \( \mathcal{H}_0 = (\mathcal{R}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E}) \) be a signature with signals (all classes active), \( \mathcal{D}_0 \) a structure of \( \mathcal{H}_0 \), and \( (\mathcal{E}_0, \text{ready}, \ominus, \oplus, [\cdot]) \) an ether over \( \mathcal{H}_0 \) 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 \( S := (\Sigma \cup \{\#\} \times \mathcal{E}_0) \) with actions \( A := (2^{\Sigma \cup \{\#\} \times \mathcal{E}_0} \cup \{\perp\} \times \mathcal{E}_0) \times (\Sigma \cup \{\#\} \times \mathcal{E}_0)^* \).

- \( (\sigma, \varepsilon) \xrightarrow{\text{cons, \text{Stud}}} (\sigma', \varepsilon') \) if and only if
  1. an event with destination \( u \) is discarded,
  2. an event is dispatched to \( u \), i.e. stable object processes an event, or
  3. run-to-completion processing by \( u \) commences, i.e. object \( u \) is not stable and continues to process an event,
  4. the environment interacts with object \( u \).
- \( s \xrightarrow{\text{cons, \emptyset}} \# \) if and only if
  1. \( s = \# \) and \( \text{cons} = \emptyset \), or an error condition occurs during consumption of \( \text{cons} \).
(i) Discarding An Event

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

if

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

\[
\begin{align*}
& u \in \text{dom}(\sigma) \cap S(C) \land \exists u_E \in S(E) : u_E \in \text{ready}(\varepsilon, u) \\
& \quad \land \exists u \in \text{dom}(\sigma) \cap D(C) : u \in \text{ready}(\varepsilon, u) \\
& \quad \land \exists u \in \text{dom}(\sigma) \cap D(C) : u \in \text{ready}(\varepsilon, u)
\end{align*}
\]

• \(u\) is stable and in state machine state \(s\), i.e.

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

• 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)

\[
\forall (s, F, \text{expr}, \text{act}, s') \in \rightarrow (SM_C) : F \neq E \lor \left[\text{expr}(\sigma) = 0\right] \land \exists u \in \text{dom}(\sigma) \cap D(C) : u \in \text{ready}(\varepsilon, u)
\]

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

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

\[
\begin{align*}
S/M_C: & \xymatrix{ & 81 \ar[dl] & 82 \ar[dl] \\
& G[x > 0]/x := y & H/z := y/x \\
& H & C, J \ar[u] & x, z : \text{Int} & y : \text{Int} & \text{expr} \ar[l]}
\end{align*}
\]

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

- \(\exists u \in \text{dom}(\sigma) \cap S(C)\)
- \(u_E \in S(E) : u_E \in \text{ready}(\varepsilon, u))\)
- \(\forall (s, F, \text{expr}, \text{act}, s') \in \rightarrow (SM_C) : F \neq E \lor \left[\text{expr}(\sigma) = 0\right] \land \exists u \in \text{dom}(\sigma) \cap D(C) : u \in \text{ready}(\varepsilon, u)
\]
- \(\exists u \in \text{dom}(\sigma) \cap S(C)\)
- \(u_E \in S(E) : u_E \in \text{ready}(\varepsilon, u))\)
- \(\exists u \in \text{dom}(\sigma) \cap D(C) : u \in \text{ready}(\varepsilon, u)\)
- \(\sigma(u)(\text{stable}) = 1 \land \sigma(u)(\text{st}) = s_1\)
- \(\varepsilon' = \varepsilon \ominus u_E\)
- \(\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset\)
(ii) Dispatch

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

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

\[\exists (s, F, expr, act, s') \in (\mathcal{S}, \mathcal{M}_C) : F = E \land I[expr](\bar{\sigma}) = 1\]

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

and

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

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

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

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

\[\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{\text{act}}(\bar{\sigma}, \varepsilon \ominus u_E).\]

Example: Dispatch

\[\begin{align*}
\text{SM}_C: & \quad [x > 0]/x := x - 1; n! J \\
G_{x > 0}/x := y & \xrightarrow{\text{c}} G_{x := y/x} \\
H/z := y/x & \xrightarrow{\text{c}} G_{z := y/x}
\end{align*}\]
(iii) Commence Run-to-Completion

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

if

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

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

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

Example: Commence

\[\text{SM}_C:\]

\[
\begin{align*}
&G[x > 0]/x := y \\
&H/z := y/x
\end{align*}
\]

\[
\begin{align*}
\sigma: &\quad c: C, \quad x = 2, z = 0, y = 2, \quad st = s_2, \text{stable} = 0 \\
\varepsilon: &\quad \sigma(u)(\text{stable}) = 0
\end{align*}
\]

\[
\begin{align*}
\exists u \in \text{dom}(\sigma) \cap D(C) &\land \sigma(u)(\text{stable}) = 0 \\
\exists (s,_{\text{expr}}, act, s') &\in (SM_C) : I[\text{expr}] (\sigma) = 1 \\
\exists (\sigma'', \varepsilon') &\in t_{\text{act}}[u](\sigma, \varepsilon), \quad \sigma' = \sigma''[u.\text{st} \mapsto s', u.\text{stable} \mapsto b] \\
\exists u \in \text{dom}(\sigma) &\land \sigma(u)(\text{stable}) = s
\end{align*}
\]

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

Assume that a set $E_{env} \subseteq \mathcal{E}$ is designated as environment events and a set of attributes $v_{env} \subseteq V$ is designated as input attributes.

Then

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

if

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

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

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

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

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

<table>
<thead>
<tr>
<th>$SM_C$: $[x &gt; 0]/x := x - 1; n! J$</th>
<th>$G[x &gt; 0]/x := y$</th>
<th>$H/z := y/x$</th>
<th>$\langle \langle \text{signal}, \text{env} \rangle \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$: $x = 0, z = 0, y = 2$</td>
<td>$st = s_2$</td>
<td>$\text{stable} = 1$</td>
<td>$C, J$</td>
</tr>
<tr>
<td>$\varepsilon$:</td>
<td></td>
<td></td>
<td>$\langle \langle \text{signal} \rangle \rangle$</td>
</tr>
</tbody>
</table>

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

$\varepsilon' = \varepsilon \oplus u_E$ where $u_E \notin \text{dom}(\sigma)$

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

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

if, in (ii) or (iii),

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

and

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

Examples:

- $s_1$ $E[x_0]/act$ $s_2$  
  $E[true]/act$ $s_3$

- $s_1$ $E[\text{expr}] / x := x_0$ $s_2$

Example: Error Condition

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

$\sigma$:  
\[
\begin{array}{l}
x = 0, z = 0, y = 27 \\
st = s_2 \\
stable = 1
\end{array}
\]

$\varepsilon$:  
\[
H \text{ for } e
\]

- $I[\text{expr}]$ not defined for $\sigma$, or
- $t_{\text{act}}$ is not defined for $(\sigma, \varepsilon)$
- consumption according to (ii) or (iii)
- $Snd = \emptyset$
Notions of Steps: The Step

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

\[ E[x > 0]/ \]
\[ /x := x - 1 \]

\[ \sigma: \]
\[ \begin{array}{c}
\vdots \\
| \\
\hline
x = 2
\end{array} \]

\[ \varepsilon: \]
\[ \begin{array}{c}
\vdots \\
\hline
E \text{ for } u
\end{array} \]

Notions of Steps: The RTC Step Cont’d

**Proposal**: Let

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

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

- object \( u \) is alive in \( \sigma_0 \),
- \( u_0 = u \) and \((cons_0, Snd_0)\) indicates dispatching to \( u \), i.e. \( cons = \{(u, \vec{v} \mapsto \vec{d})\}\),
- there are no receptions by \( u \) in between, i.e.

\[ cons_i \cap \{u\} \times Evs(\mathcal{E}, \mathcal{D}) = \emptyset, i > 1, \]

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

\[ \sigma_0(u)(stable) = \sigma_n(u)(stable) = 1 \quad \text{and} \quad \sigma_i(u)(stable) = 0 \quad \text{for} \quad 0 < i < n, \]
Proposal: Let
\[(\sigma_0, \varepsilon_0) \xrightarrow{\text{cons}_0, \text{Snd}_0} \ldots \xrightarrow{\text{cons}_{n-1}, \text{Snd}_{n-1}} (\sigma_n, \varepsilon_n), \quad n > 0,\]
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.
  \[\text{cons}_i \cap \{u\} \times \text{Evs}(E', D) = \emptyset, i > 1,\]
- \(u_{n-1} = u\) and \(u\) is stable only in \(\sigma_0\) and \(\sigma_n\), i.e.
  \[\sigma_0(u)(\text{stable}) = \sigma_n(u)(\text{stable}) = 1 \quad \text{and} \quad \sigma_i(u)(\text{stable}) = 0 \quad \text{for} \quad 0 < i < n,\]
Let \(0 = k_1 < k_2 < \ldots < k_N = n\) be the maximal sequence of indices such that \(u_{k_i} = u\) for \(1 \leq i \leq N\).

Let 0 = \(k_1 < k_2 < \ldots < k_N = n\) be the maximal sequence of indices such that \(u_{k_i} = u\) for \(1 \leq i \leq N\).

Then we call the sequence
\[\sigma_0(u) = \sigma_{k_1}(u), \sigma_{k_2}(u), \ldots, \sigma_{k_N}(u) \quad (= \sigma_{n-1}(u))\]
a (!) run-to-completion computation of \(u\) (from (local) configuration \(\sigma_0(u))\).