Software Design, Modelling and Analysis in UML

Lecture 14: Core State Machines IV

2014-12-18

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

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

abstract syntax	**concrete syntax**
`skip`	*skip*
intuitive semantics	
do nothing	
well-typedness	
./.	
semantics	
$t_{skip}[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}$	
observables	
$Obs_{\mathtt{skip}}[u_x](\sigma, \varepsilon) = \emptyset$	
(error) conditions	

Transformer: Update

abstract syntax	concrete syntax
$\mathtt{update}(expr_1, v, expr_2)$	$expr_1.v := expr_2$

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 $v : \tau \in atr(C)$; $expr_2 : \tau$;
$expr_1, expr_2$ obey visibility and navigability !

semantics

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

where $\sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto I[\![expr_2]\!](\sigma, u_x)]]$
with $u = I[\![expr_1]\!](\sigma, u_x)$.

observables

$$Obs_{\mathtt{update}(expr_1,v,expr_2)}[u_x] = \emptyset$$

(error) conditions

Not defined if $I[\![expr_1]\!](\sigma, \cancel{\beta}^{u_x})$ or $I[\![expr_2]\!](\sigma, \cancel{\beta}^{u_x})$ not defined.

ε does not change

change value of v in $\sigma(u)$

change local state of object u

object denoted by $expr_1$ (relative to u_x)

value denoted by $expr_2$ (relative to u_x)

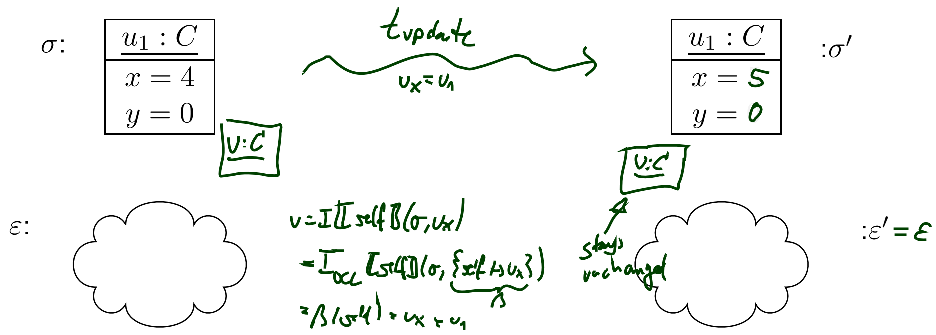
Update Transformer Example

$\mathcal{SM}_C$: $s_1 \xrightarrow{/x := x + 1} s_2$

$expr_1$ (implicitly self.) $\quad v \quad expr_2$

$$t_{\mathtt{update}(expr_1,v,expr_2)}[u_x](\sigma, \varepsilon) = (\sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto I[\![expr_2]\!](\sigma, u_x)]], \varepsilon),\ u = I[\![expr_1]\!](\sigma, u_x)$$

$I[\![x+1]\!](\sigma, u_x) = I[\![x+1]\!](\sigma, u_1) = 5$



$v = I[\![self]\!]\beta(\sigma, u_x)$
$= I_{OCL}[\![self]\!](\sigma, \underbrace{\{self \mapsto u_x\}}_{\beta})$
$= \beta(self) = u_x = u_1$

Transformer: Send

abstract syntax — send$(E(expr_1, ..., expr_n), expr_{dst})$

concrete syntax — $expr_{dst}\,!\,E(expr_1, \ldots, expr_n)$

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

$expr_{dst} : \tau_D$, $C, D \in \mathscr{C} \setminus \mathscr{E}$; $E \in \mathscr{E}$; $atr(E) = \{v_1 : \tau_1, \ldots, v_n : \tau_n\}$;
$expr_i : \tau_i$, $1 \le i \le n$;
all expressions obey visibility and navigability in C

semantics

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

disjoint union

new signal instance

iff $\sigma' = \sigma \,\dot\cup\, \{u \mapsto \{v_i \mapsto d_i \mid 1 \le i \le n\}\}$; $\varepsilon' = \varepsilon \oplus (u_{dst}, u)$;
if $u_{dst} = I[\![expr_{dst}]\!](\sigma, u_x) \in \mathrm{dom}(\sigma)$; $d_i = I[\![expr_i]\!](\sigma, u_x)$ for $1 \le i \le n$;
$u \in \mathscr{D}(E)$ a fresh identity, i.e. $u \notin \mathrm{dom}(\sigma)$,
and where $(\sigma', \varepsilon') = (\sigma, \varepsilon)$ if $u_{dst} \notin \mathrm{dom}(\sigma)$. ← do nothing if u_{dst} not alive in σ

our choice! we could also just send, or consider it an error condition

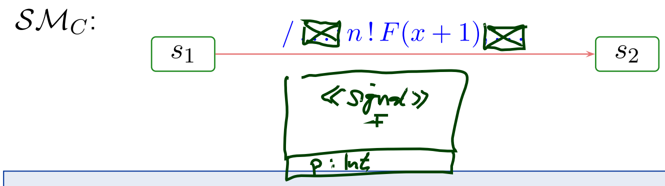
observables

$$Obs_{\texttt{send}}[u_x] = \{(u_x, u, (E, d_1, \ldots, d_n), u_{dst})\}$$

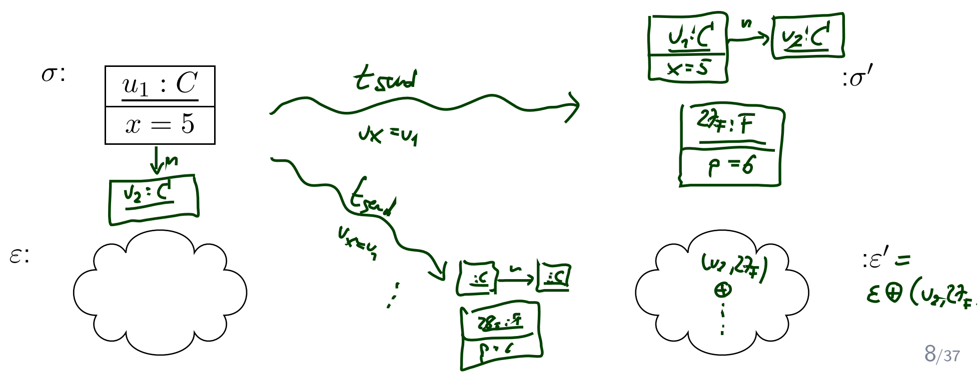
(error) conditions

$I[\![expr]\!](\sigma, u_x)$ not defined for any $expr \in \{expr_{dst}, expr_1, \ldots, expr_n\}$

Send Transformer Example



$t_{\texttt{send}(expr_{src},E(expr_1,\ldots,expr_n),expr_{dst})}[u_x](\sigma, \varepsilon) \ni (\sigma', \varepsilon')$ iff $\varepsilon' = \varepsilon \oplus (u_{dst}, u)$;
$\sigma' = \sigma \,\dot\cup\, \{u \mapsto \{v_i \mapsto d_i \mid 1 \le i \le n\}\}$; $u_{dst} = I[\![expr_{dst}]\!](\sigma, u_x) \in \mathrm{dom}(\sigma)$;
$d_i = I[\![expr_i]\!](\sigma, u_x)$, $1 \le i \le n$; $u \in \mathscr{D}(E)$ 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

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

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 the two intermediate "micro steps" is not defined.

Transformers And Denotational Semantics

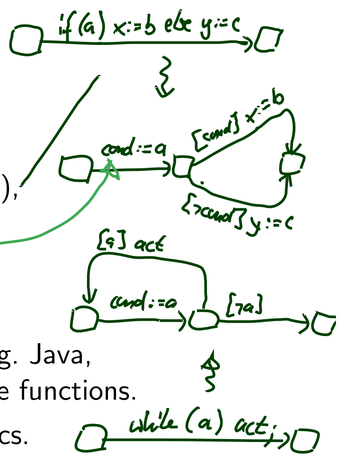
Observation: our transformers are in principle the **denotational semantics** of the actions/action sequences. The trivial case, to be precise.

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**.

add cond : Bool
to attr (C)
if this is SM_C



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

Transition Relation, Computation

Definition. Let A be a set of **actions** and S a (not necessarily finite) set of of **states**.

We call

$$\to \ \subseteq S \times A \times S$$

a (labelled) **transition relation**.

Let $S_0 \subseteq S$ be a set of **initial states**. A sequence

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

with $s_i \in S$, $a_i \in A$ is called **computation** of the **labelled transition system** $(S, \underbrace{\to}_{A}, S_0)$ if and only if

- **initiation**: $s_0 \in S_0$
- **consecution**: $(s_i, a_i, s_{i+1}) \in \to$ 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.

From Core State Machines to LTS

Definition. Let $\mathscr{S}_0 = (\mathscr{T}_0, \mathscr{C}_0, V_0, atr_0, \mathscr{E})$ be a signature with signals (all classes **active**), $\mathscr{D}_0$ a structure of $\mathscr{S}_0$, and $(Eth, ready, \oplus, \ominus, [\,\cdot\,])$ an ether over $\mathscr{S}_0$ and $\mathscr{D}_0$.
Assume there is one core state machine M_C per class $C \in \mathscr{C}$.

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

- $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$ 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,
- $s \xrightarrow{(cons, \emptyset)} \#$ if and only if
  - (v) $s = \#$ and $cons = \emptyset$, or an error condition occurs during consumption of $cons$.

(i) Discarding An Event

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

if

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

$$u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C) \wedge \exists\, u_E \in \mathscr{D}(\cancel{E}\,\mathscr{E}) : u_E \in ready(\varepsilon, u)$$

- u is stable and in state machine state s, i.e. $\sigma(u)(stable) = 1$ and $\sigma(u)(st) = s$,
- 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)

$$\forall\,(s, F, expr, act, s') \in \rightarrow (\mathcal{SM}_C) : F \neq E \vee I[\![expr]\!](\tilde{\sigma}, u) = 0$$

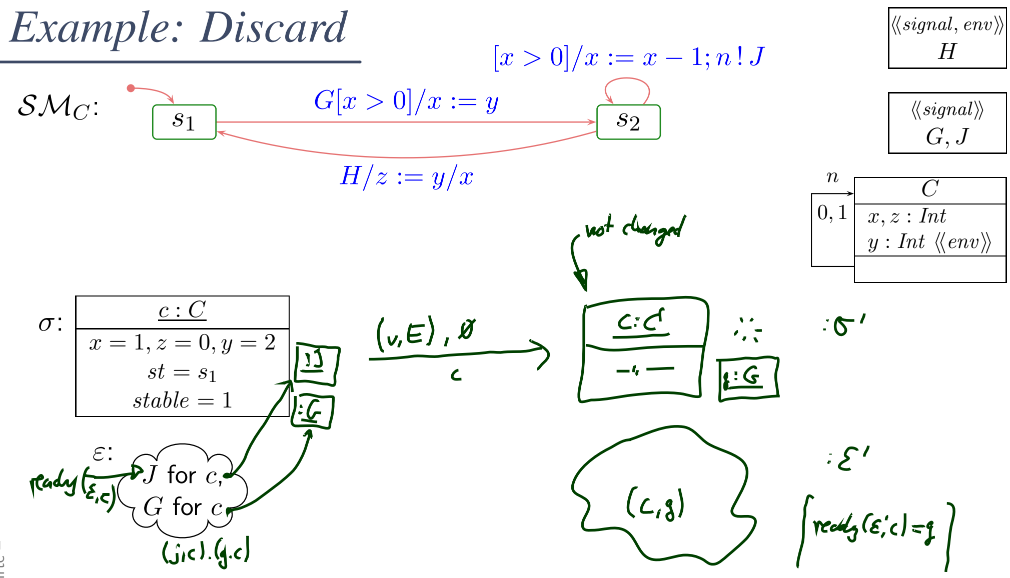
current state assumed above see (ii)

and

- the system configuration ~~doesn't~~ change~~s~~ 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



- $\exists\, u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C)$ ✓
  $\exists\, u_E \in \mathscr{D}(\mathscr{E}) : u_E \in ready(\varepsilon, u)$ ✓
- $\forall\,(s, F, expr, act, s') \in \rightarrow (\mathcal{SM}_C)$:
  $F \neq E \vee I[\![expr]\!](\sigma) = 0$ ✓
- $\sigma(u)(stable) = 1$ ✓ $\sigma(u)(st) = s$ ✓
- $\sigma' = \sigma$, $\varepsilon' = \varepsilon \ominus u_E$ \ u_E
- $cons = \{(u, (E, \sigma(u_E)))\}$, $Snd = \emptyset$

(ii) Dispatch

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

- $u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C) \wedge \exists\, u_E \in \mathscr{D}(E) : u_E \in ready(\varepsilon, u)$
- u is stable and in state machine state s, i.e. $\sigma(u)(stable) = 1$ and $\sigma(u)(st) = s$,
- a transition is enabled, i.e.

$$\exists\, (s, F, expr, act, s') \in \to (\mathcal{SM}_C) : F = E \wedge I[\![expr]\!](\tilde{\sigma}) = 1$$

  where $\tilde{\sigma} = \sigma[u.params_E \mapsto u_E]$.

and

- (σ', ε') results from applying t_{act} to (σ, ε) and removing u_E from the ether, i.e.

$$(\sigma'', \varepsilon') \in \overbrace{t_{act}}^{[u]}(\tilde{\sigma}, \varepsilon \ominus u_E),$$

$$\sigma' = (\sigma''[u.st \mapsto s', u.stable \mapsto b, u.params_E \mapsto \emptyset])|_{\mathscr{D}(\mathscr{C}) \setminus \{u_E\}}$$

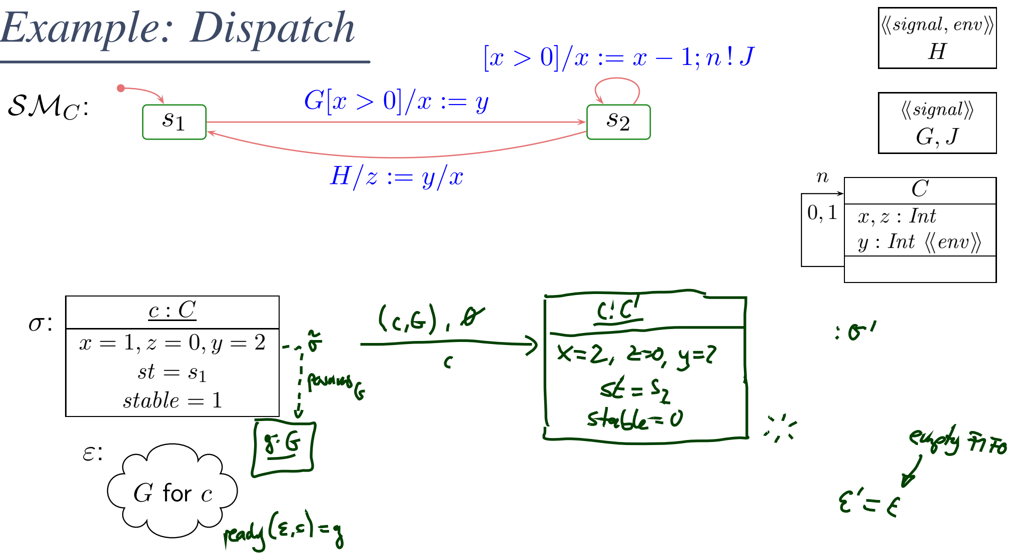
  where b **depends**:

  - 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 (σ', ε').
  - Otherwise $b = 0$.

- Consumption of u_E and the side effects of the action are observed, i.e.

$$cons = \{(u, (E, \sigma(u_E)))\}, Snd = Obs_{\overbrace{t_{act}}^{\{u\}}}(\tilde{\sigma}, \varepsilon \ominus u_E).$$

Example: Dispatch



- $\exists\, u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C)$ ✓
  $\exists\, u_E \in \mathscr{D}(E) : u_E \in ready(\varepsilon, u)$ ✓
- $\exists\, (s, F, expr, act, s') \in \to (\mathcal{SM}_C)$:
  $F = E \wedge I[\![expr]\!](\tilde{\sigma}) = 1$ ✓
- $\tilde{\sigma} = \sigma[u.params_E \mapsto u_E]$.
- $\sigma(u)(stable) = 1$ ✓ $\sigma(u)(st) = s$ ✓
- $(\sigma'', \varepsilon') = t_{act}(\tilde{\sigma}, \varepsilon \ominus u_E)$
- $\sigma' = (\sigma''[u.st \mapsto s', u.stable \mapsto b, u.params_E \mapsto \emptyset])|_{\mathscr{D}(\mathscr{C}) \setminus \{u_E\}}$
- $cons = \{(u, (E, \sigma(u_E)))\}$, $Snd = Obs_{t_{act}}(\tilde{\sigma}, \varepsilon \ominus u_E)$

(iii) Commence Run-to-Completion

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

if

- there is an unstable object u of a class $\mathscr{C}$, i.e.

$$u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C) \wedge \sigma(u)(stable) = 0$$

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

$$\exists\, (s, \_, expr, act, s') \in \to (\mathcal{SM}_C) : I[\![expr]\!](\sigma) = 1$$

not $\tilde{\sigma}$

and

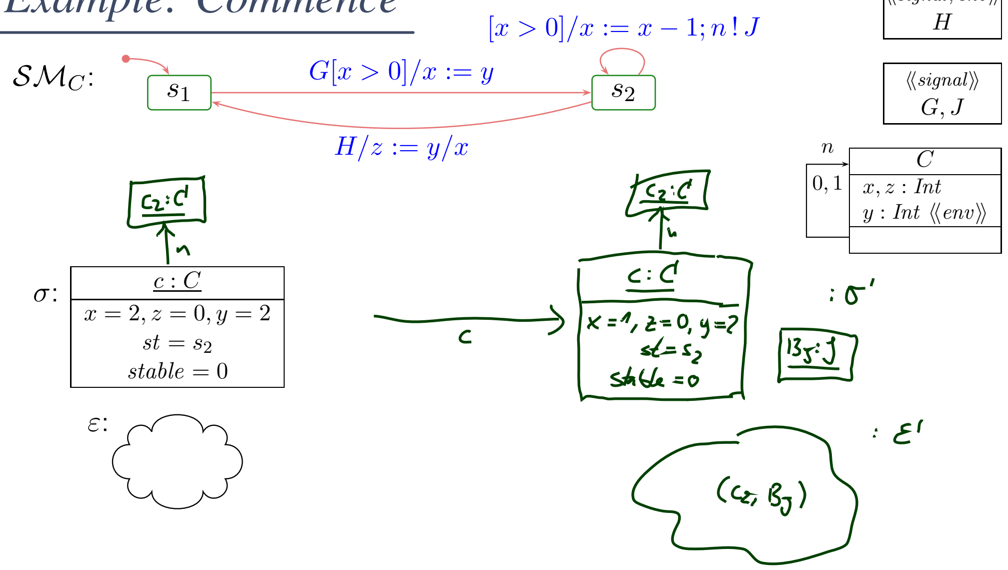
- (σ', ε') results from applying t_{act} to (σ, ε), i.e.

$$(\sigma'', \varepsilon') \in t_{act}[u](\sigma, \varepsilon), \quad \sigma' = \sigma''[u.st \mapsto s', u.stable \mapsto b]$$

  where b **depends** as before.
- Only the side effects of the action are observed, i.e.

$$cons = \emptyset, Snd = Obs_{t_{act}}(\sigma, \varepsilon).$$

Example: Commence



- $\exists\, u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C) : \sigma(u)(stable) = 0$ ✓
- $\exists\, (s, \_, expr, act, s') \in \to (\mathcal{SM}_C)$: $I[\![expr]\!](\sigma) = 1$ ✓
- $\cancel{\sigma(u)(stable) = 1}$ $\sigma(u)(st) = s$, ✓
- $(\sigma'', \varepsilon') = t_{act}(\sigma, \varepsilon)$, $\sigma' = \sigma''[u.st \mapsto s', u.stable \mapsto b]$
- $cons = \emptyset$, $Snd = Obs_{t_{act}}(\sigma, \varepsilon)$

(iv) Environment Interaction

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

Then

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

if

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

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

  where $u_E \notin \mathrm{dom}(\sigma)$ and $atr(E) = \{v_1, \dots, v_n\}$.
- Sending of the event is observed, i.e. $cons = \emptyset$, $Snd = \{(env, E(\vec{d}))\}$.

or

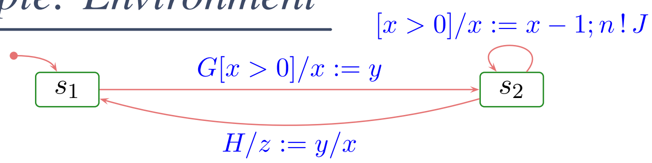
- Values of input attributes change freely in alive objects, i.e.

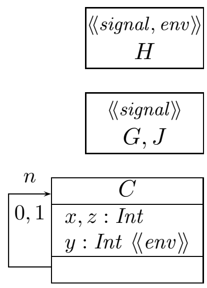
$$\forall\, v \in V \;\forall\, u \in \mathrm{dom}(\sigma) : \sigma'(u)(v) \neq \sigma(u)(v) \implies v \in V_{env}.$$

  and no objects appear or disappear, i.e. $\mathrm{dom}(\sigma') = \mathrm{dom}(\sigma)$.
- $\varepsilon' = \varepsilon$.

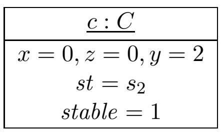
Example: Environment

$\mathcal{SM}_C$:

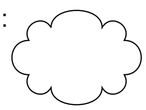




σ:



ε:



- $\sigma' = \sigma \,\dot\cup\, \{u_E \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}$
- $\varepsilon' = \varepsilon \oplus u_E$ where $u_E \notin \mathrm{dom}(\sigma)$ and $atr(E) = \{v_1, \dots, v_n\}$.
- $u \in \mathrm{dom}(\sigma)$
- $cons = \emptyset$, $Snd = \{(env, E(\vec{d}))\}$.

(v) Error Conditions

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

(i)

if, in (ii) or (iii),

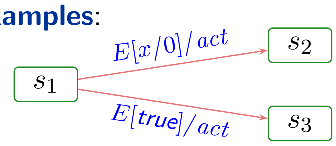
- $I[\![expr]\!]$ is not defined for σ, or
- t_{act} is not defined for (σ, ε),

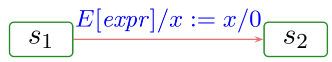
and

(i)

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

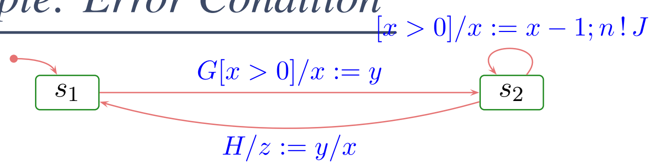
Examples:

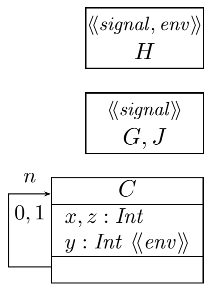
- 

- 

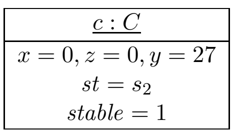
Example: Error Condition

$\mathcal{SM}_C$:

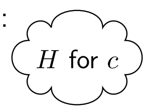




σ:



ε:



- $I[\![expr]\!]$ not defined for σ, or
- t_{act} is not defined for (σ, ε)
- consumption according to (ii) or (iii)
- $Snd = \emptyset$

Notions of Steps: The Step

Note: we call one evolution $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$ 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.

– 14 – 2014-12-18 – Sstmstep –

Notions of Steps: The Step

Note: we call one evolution $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$ 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.

– 14 – 2014-12-18 – Sstmstep –

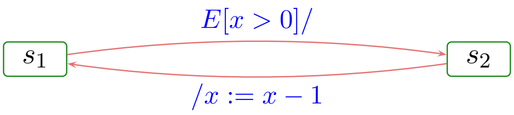
Notions of Steps: The Run-to-Completion Step

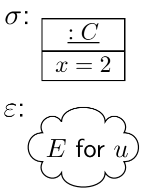
What is a **run-to-completion** step...?

- **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 syntacically definable — one transition may be taken multiple times during an RTC-step.

Example:





Notions of Steps: The RTC Step Cont'd

Proposal: Let

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

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

- object u is alive in σ_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.

$$cons_i \cap \{u\} \times Evs(\mathscr{E}, \mathscr{D}) = \emptyset, i > 1,$$

- $u_{n-1} = u$ and u is stable only in σ_0 and σ_n, i.e.

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

Notions of Steps: The RTC Step Cont'd

Proposal: Let

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

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

- object u is alive in σ_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.

$$cons_i \cap \{u\} \times Evs(\mathscr{E}, \mathscr{D}) = \emptyset, i > 1,$$

- $u_{n-1} = u$ and u is stable only in σ_0 and σ_n, i.e.

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

Let $0 = k_1 < k_2 < \dots < k_N = n$ be the maximal sequence of indices such that $u_{k_i} = u$ for $1 \le i \le N$.

Notions of Steps: The RTC Step Cont'd

Proposal: Let

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

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

- object u is alive in σ_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.

$$cons_i \cap \{u\} \times Evs(\mathscr{E}, \mathscr{D}) = \emptyset, i > 1,$$

- $u_{n-1} = u$ and u is stable only in σ_0 and σ_n, i.e.

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

Let $0 = k_1 < k_2 < \dots < k_N = n$ be the maximal sequence of indices such that $u_{k_i} = u$ for $1 \le i \le N$. Then we call the sequence

$$(\sigma_0(u) =) \quad \sigma_{k_1}(u), \sigma_{k_2}(u) \dots, \sigma_{k_N}(u) \quad (= \sigma_{n-1}(u))$$

a (!) **run-to-completion computation** of u (from (local) configuration $\sigma_0(u)$).