

Software Design, Modelling and Analysis in UML

Lecture 14: Core State Machines IV

2013-12-18

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- System configuration
- Transformer
- Action language: skip, update, send

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:**
 - Transformers for Action Language
 - Run-to-completion Step
 - Putting It All Together

Transformer Cont'd

Transformer: Create

(*) SO NOT: $x := (\text{new } C).x + (\text{new } C).y$

if needed

$t_{exp_1} := \text{new } C;$
 $t_{exp_2} := \text{new } C;$
 $x := t_{exp_1}.x$
 $+ t_{exp_2}.y$
 $t_{exp_1} := \text{Null};$
 $t_{exp_2} := \text{Null};$

abstract syntax

$\text{create}(C, \text{expr}, v)$

concrete syntax

$\text{expr}.v := \text{new } C$

intuitive semantics

Create an object of class C and assign it to attribute v of the object denoted by expression expr .

well-typedness

$\text{expr} : \tau_D, v \in \text{atr}(D), \text{atr}(C) = \{\langle v_i : \tau_i, \text{expr}_i^0 \rangle \mid 1 \leq i \leq n\}$

semantics

...

observables

...

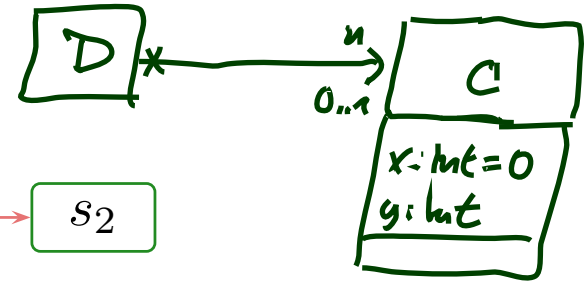
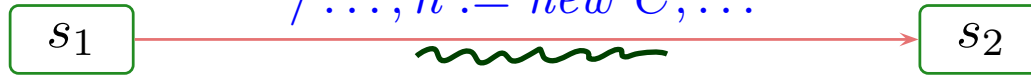
(error) conditions

$I[\text{expr}](\sigma, \beta)$ not defined.

- We use an “and assign”-action for simplicity — it doesn’t add or remove expressive power, but moving creation to the expression language raises all kinds of other problems such as order of evaluation (and thus creation). (*)
- Also for simplicity: no parameters to construction (\sim parameters of constructor). Adding them is straightforward (but somewhat tedious).

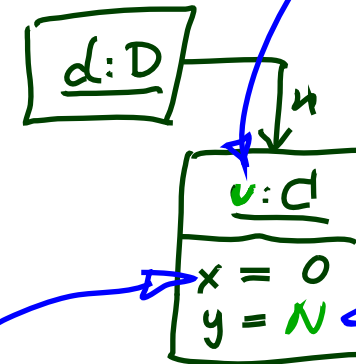
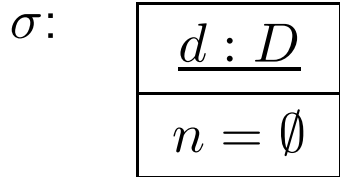
Create Transformer Example

SM_C :



$\text{create}(C, \text{expr}, v)$
 $t_{\text{create}(C, \text{expr}, v)}(\sigma, \varepsilon) = \dots$

$\in \mathcal{D}(e)$
 $\setminus \text{dom } \sigma$
 (non-det. choice)



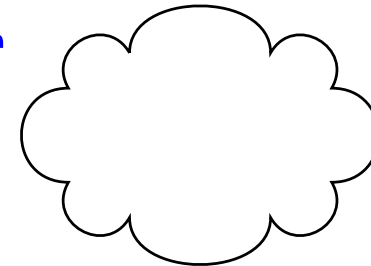
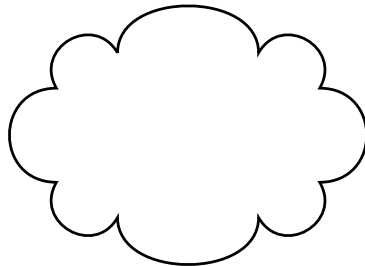
$:\sigma'$

by initial value expression

$\in \mathcal{D}(\text{int})$
 (non-det. choice)

$:\varepsilon'$

ε :



How To Choose New Identities?

- **Re-use**: choose any identity that is not alive **now**, i.e. not in $\text{dom}(\sigma)$. *our choice*
 - Doesn't depend on history.
 - May “undangle” dangling references – may happen on some platforms.
- **Fresh**: choose any identity that has not been alive **ever**, i.e. not in $\text{dom}(\sigma)$ and any predecessor in current run.
 - Depends on history.
 - Dangling references remain dangling – could mask “dirty” effects of platform.

Transformer: Create

abstract syntax

$\text{create}(C, \text{expr}, v)$

concrete syntax

intuitive semantics

Create an object of class C and assign it to attribute v of the object denoted by expression expr .

well-typedness

$\text{expr} : \tau_D, v \in \text{atr}(D), \text{atr}(C) = \{\langle v_1 : \tau_1, \text{expr}_i^0 \rangle \mid 1 \leq i \leq n\}$

semantics

similar to update

id of newly created object

$((\sigma, \varepsilon), (\sigma', \varepsilon')) \in t$ initialisation of new object

iff $\sigma' = \sigma[u_0 \mapsto \sigma(u_0)[v \mapsto u]] \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}\},$

$\varepsilon' = [u](\varepsilon); \quad u \in \mathcal{D}(C)$ fresh, i.e. $u \notin \text{dom}(\sigma);$ NONE

$u_0 = I[\text{expr}](\sigma, u_x); \quad d_i = I[\text{expr}_i^0](\sigma, u_x)$ if $\text{expr}_i^0 \neq \Box$ and arbitrary value from $\mathcal{D}(\tau_i)$ otherwise;

observables

$\text{Obs}_{\text{create}}[u_x] = \{(u_x, \perp, (*, \emptyset), u)\}$ creation... of this guy

(error) conditions

$I[\text{expr}](\sigma)$ not defined.

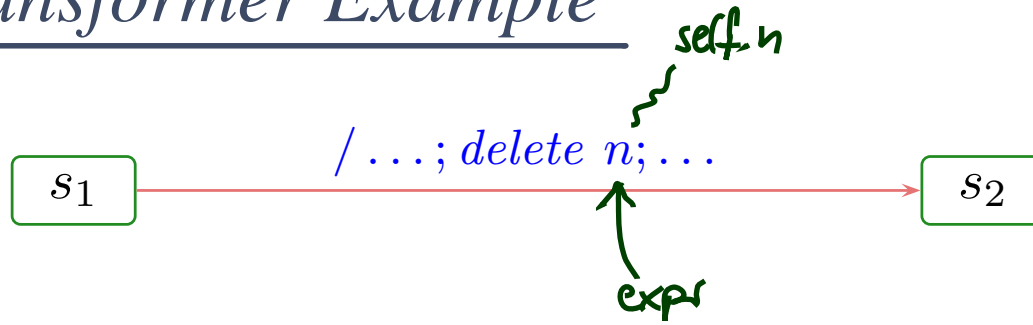
clean ether (here: no dirty/nasty effects)

Transformer: Destroy

abstract syntax $\text{destroy}(expr)$	concrete syntax <i>delete expr</i>
intuitive semantics <i>Destroy the object denoted by expression $expr$.</i>	
well-typedness $expr : \tau_C, C \in \mathcal{C}$	
semantics ...	
observables $Obs_{\text{destroy}}[u_x] = \{(u_x, \perp, (+, \emptyset), u)\}$	
(error) conditions $I \llbracket expr \rrbracket (\sigma, \beta)$ not defined.	

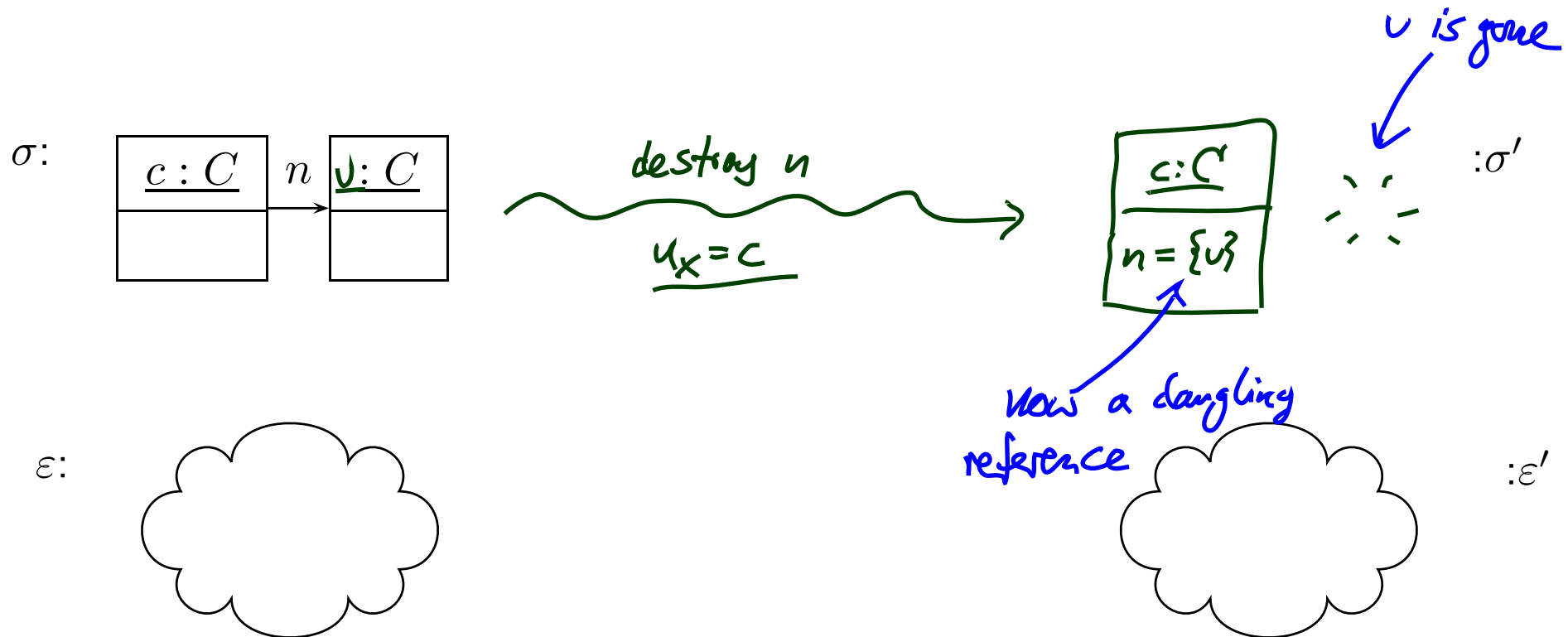
Destroy Transformer Example

SM_C :



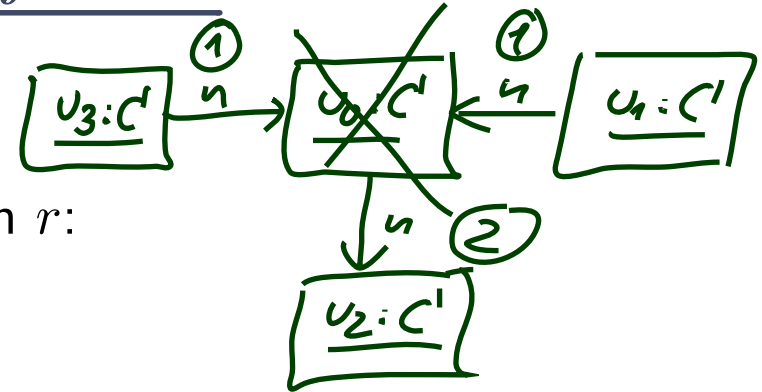
destroy($expr$)

$t_{\text{destroy}(expr)}[u_x](\sigma, \varepsilon) = \dots$



What to Do With the Remaining Objects?

Assume object u_0 is destroyed...



- object u_1 may still refer to it via association r :
 - allow dangling references?
 - or remove u_0 from $\sigma(u_1)(r)$?
- ①
- object u_0 may have been the last one linking to object u_2 :
 - leave u_2 alone?
 - or remove u_2 also? ("garbage collection")
- ②
- Plus: (temporal extensions of) OCL may have dangling references.

Our choice: Dangling references and no garbage collection!

This is in line with “expect the worst”, because there are target platforms which don’t provide garbage collection — and models shall (in general) be correct without assumptions on target platform.

But: the more “dirty” effects we see in the model, the more expensive it often is to analyse. Valid proposal for simple analysis: monotone frame semantics, no destruction at all.

Transformer: Destroy

abstract syntax

$\text{destroy}(expr)$

concrete syntax

intuitive semantics

Destroy the object denoted by expression $expr$.

well-typedness

$expr : \tau_C, C \in \mathcal{C}$

semantics

$t[u_x](\sigma, \varepsilon) = (\sigma', \varepsilon)$ *function restriction*

where $\sigma' = \sigma|_{\text{dom}(\sigma) \setminus \{u\}}$ with $u = I[expr](\sigma, u_x)$.

observables

$Obs_{\text{destroy}}[u_x] = \{(u_x, \perp, (+, \emptyset), u)\}$

(error) conditions

$I[expr](\sigma, u_x)$ not defined.

Sequential Composition of Transformers

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

read "t2 after t1" seq. composition of relations

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

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.

x := x + 1 ; delete m ; n ! F

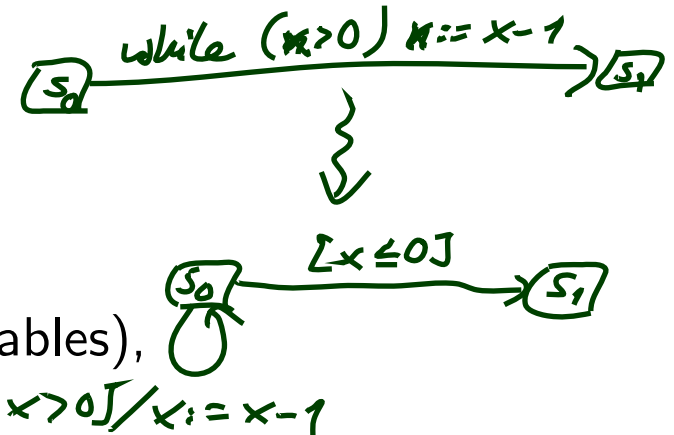
$$t_{\text{send}} (t_{\text{delete}} (t_{\text{update}} (\sigma, \varepsilon)))$$

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.

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 **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

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

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

Note: for simplicity, we only consider infinite runs.

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 [Harel and Gery, 1997] (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 $\mathcal{S}_0 = (\mathcal{I}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$ be a signature with signals (all classes **active**), \mathcal{D}_0 a structure of \mathcal{S}_0 , and $(Eth, ready, \oplus, \ominus, [\cdot])$ an ether over \mathcal{S}_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

$S := (\Sigma_{\mathcal{S}}^{\mathcal{D}} \dot{\cup} \{\#\}) \times Eth$ with actions $A := \left(2^{\mathcal{D}(\mathcal{C})} \times (\mathcal{D}(\mathcal{E}) \dot{\cup} \{\perp\}) Evs(\mathcal{E}, \mathcal{D}) \times \mathcal{D}(\mathcal{C}) \right)^2 \times \mathcal{D}(\mathcal{E})$:

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

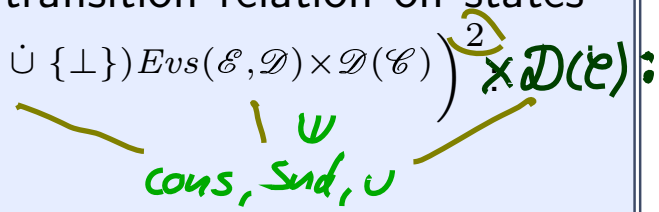
if and only if

- (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 \mathcal{C} , i.e. if

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

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

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

and

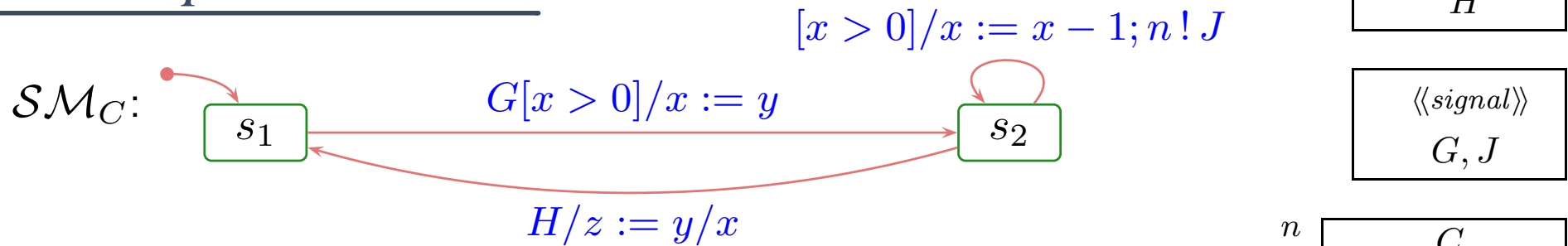
- the system configuration doesn't change, i.e. $\sigma' = \sigma$
- the event u_E is removed from the ether, i.e.

$$\varepsilon' = \varepsilon \ominus u_E,$$

- consumption of u_E is observed, i.e.

$$\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \emptyset.$$

Example: Discard



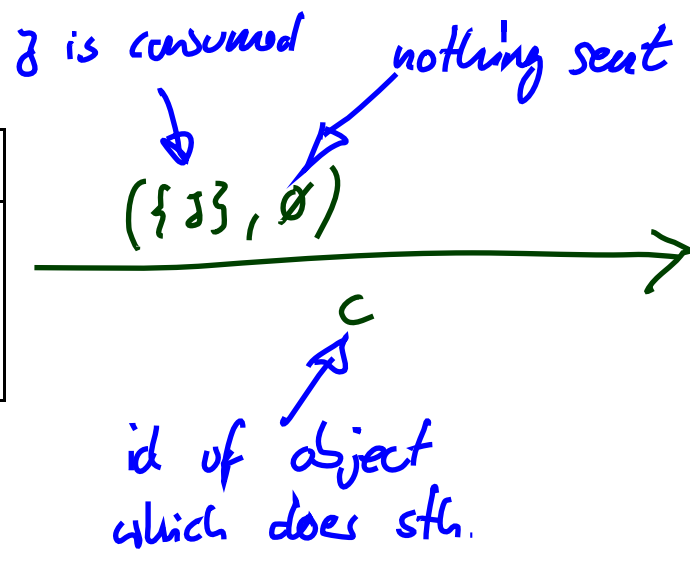
$\langle\langle signal, env \rangle\rangle$
H

$\langle\langle signal \rangle\rangle$
G, J

n	C
$0, 1$	$x, z : Int$
	$y : Int \langle\langle env \rangle\rangle$

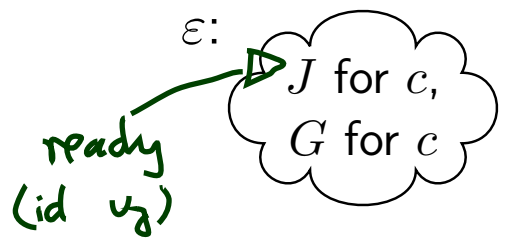
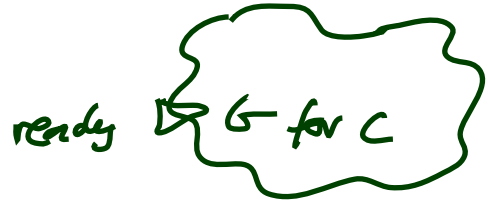
σ :

$c : C$
$x = 1, z = 0, y = 2$
$st = s_1$
$stable = 1$



$\sigma' = \sigma$

$\varepsilon' = \varepsilon \ominus j_f$



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

(ii) Dispatch

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

- $u \in \text{dom}(\sigma) \cap \mathcal{D}(C) \wedge \exists u_E \in \mathcal{D}(\mathcal{E}) : u_E \in \text{ready}(\varepsilon, u)$
- 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.

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


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.

$$(\sigma'', \varepsilon') \in t_{act}(\tilde{\sigma}, \varepsilon \ominus u_E),$$

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

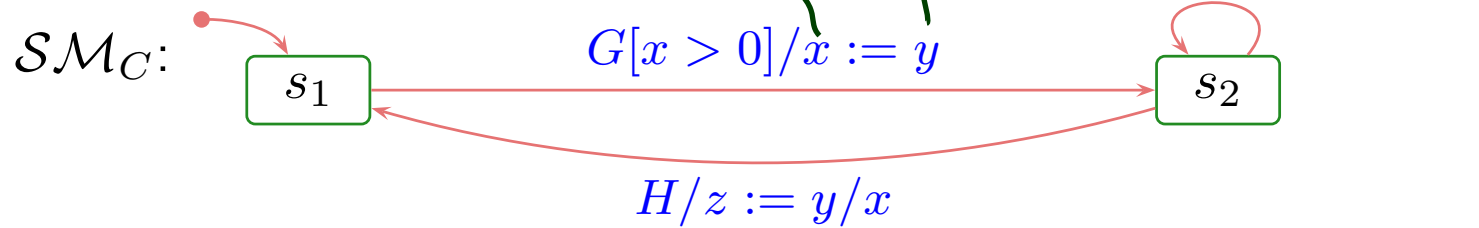
remove object u_E 

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.

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

Example: Dispatch



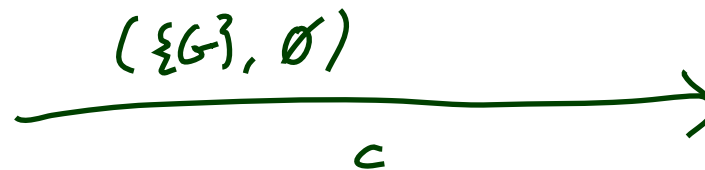
$\langle\langle signal, env \rangle\rangle$
H

$\langle\langle signal \rangle\rangle$
G, J

n	C
$0, 1$	$x, z : Int$
	$y : Int \langle\langle env \rangle\rangle$

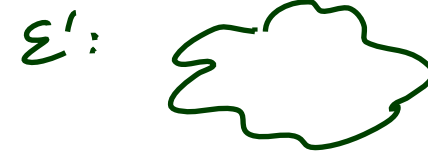
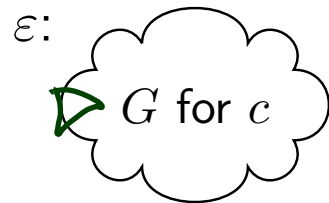
σ :

$c : C$
$x = 1, z = 0, y = 2$
$st = s_1$
$stable = 1$



σ'

$c : C'$
$x = 2, z = 0, y = 2$
$st = s_2$
$stable = 0$



- $\exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C)$
 $\exists u_E \in \mathcal{D}(\mathcal{E}) : u_E \in \text{ready}(\varepsilon, u)$
- $\exists (s, F, \text{expr}, \text{act}, s') \in \rightarrow (SM_C) :$
 $F = E \wedge I[\llbracket \text{expr} \rrbracket](\tilde{\sigma}) = 1$
- $\tilde{\sigma} = \sigma[u.\text{params}_E \mapsto u_E]$
- $\sigma(u)(\text{stable}) = 1, \sigma(u)(st) = s,$
- $(\sigma'', \varepsilon') = t_{act}(\tilde{\sigma}, \varepsilon \ominus u_E)$
- $\sigma' = (\sigma''[u.st \mapsto s', u.stable \mapsto b, u.params_E \mapsto \emptyset])|_{\mathcal{D}(\mathcal{E}) \setminus \{u_E\}}$
- $\text{cons} = \{(u, (E, \sigma(u_E)))\}, \text{Snd} = \text{Obs}_{t_{act}}(\tilde{\sigma}, \varepsilon \ominus u_E)$

(iii) Commence Run-to-Completion

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

if

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

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

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

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

and

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

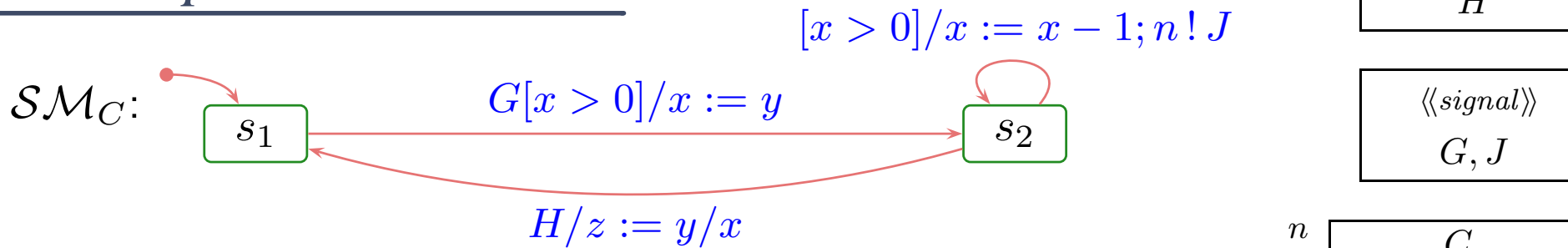
$$(\sigma'', \varepsilon') \in t_{act}[u](\sigma, \varepsilon), \quad \sigma' = \sigma''[u.st \mapsto s', u.stable \mapsto b]$$

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



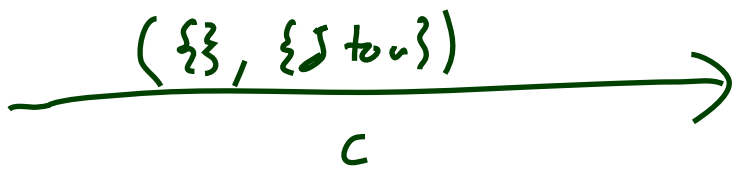
$\langle\langle signal, env \rangle\rangle$
H

$\langle\langle signal \rangle\rangle$
G, J

n	C
$0, 1$	$x, z : Int$ $y : Int \langle\langle env \rangle\rangle$

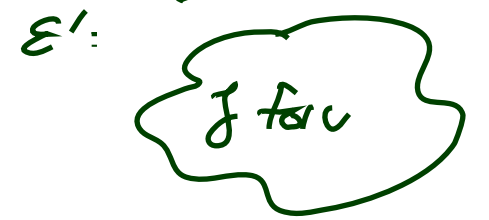
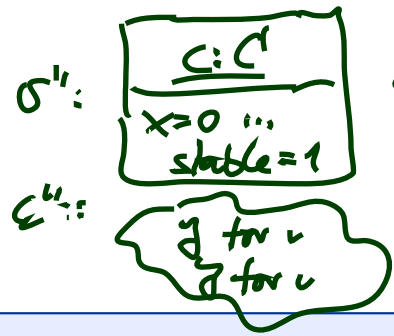
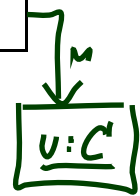
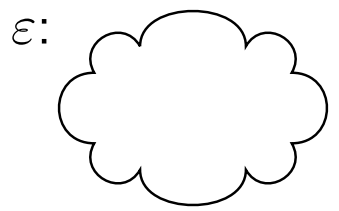
σ :

$c : C$
$x = 2, z = 0, y = 2$ $st = s_2$ $stable = 0$



σ' :

$c : C$
$x = 1, z = 0, y = 2$ $st = s_2$ $stable = 0$



- $\exists u \in \text{dom}(\sigma) \cap \mathcal{D}(C) : \sigma(u)(stable) = 0$
- $\exists (s, -, expr, act, s') \in \rightarrow (SM_C) : I[expr](\sigma) = 1$
- $\sigma(u)(stable) = 1, \sigma(u)(st) = s,$
- $(\sigma'', \varepsilon') = t_{act}(\sigma, \varepsilon),$
 $\sigma' = \sigma''[u.st \mapsto s', u.stable \mapsto b]$
- $cons = \emptyset, Snd = Obs_{t_{act}}(\sigma, \varepsilon)$

(iv) Environment Interaction

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

Then

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

if

- an environment event $E \in \mathcal{E}_{env}$ is spontaneously sent to an alive object $u \in \mathcal{D}(\sigma)$, 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 $u_E \notin \text{dom}(\sigma)$ and $\text{atr}(E) = \{v_1, \dots, v_n\}$.

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

or

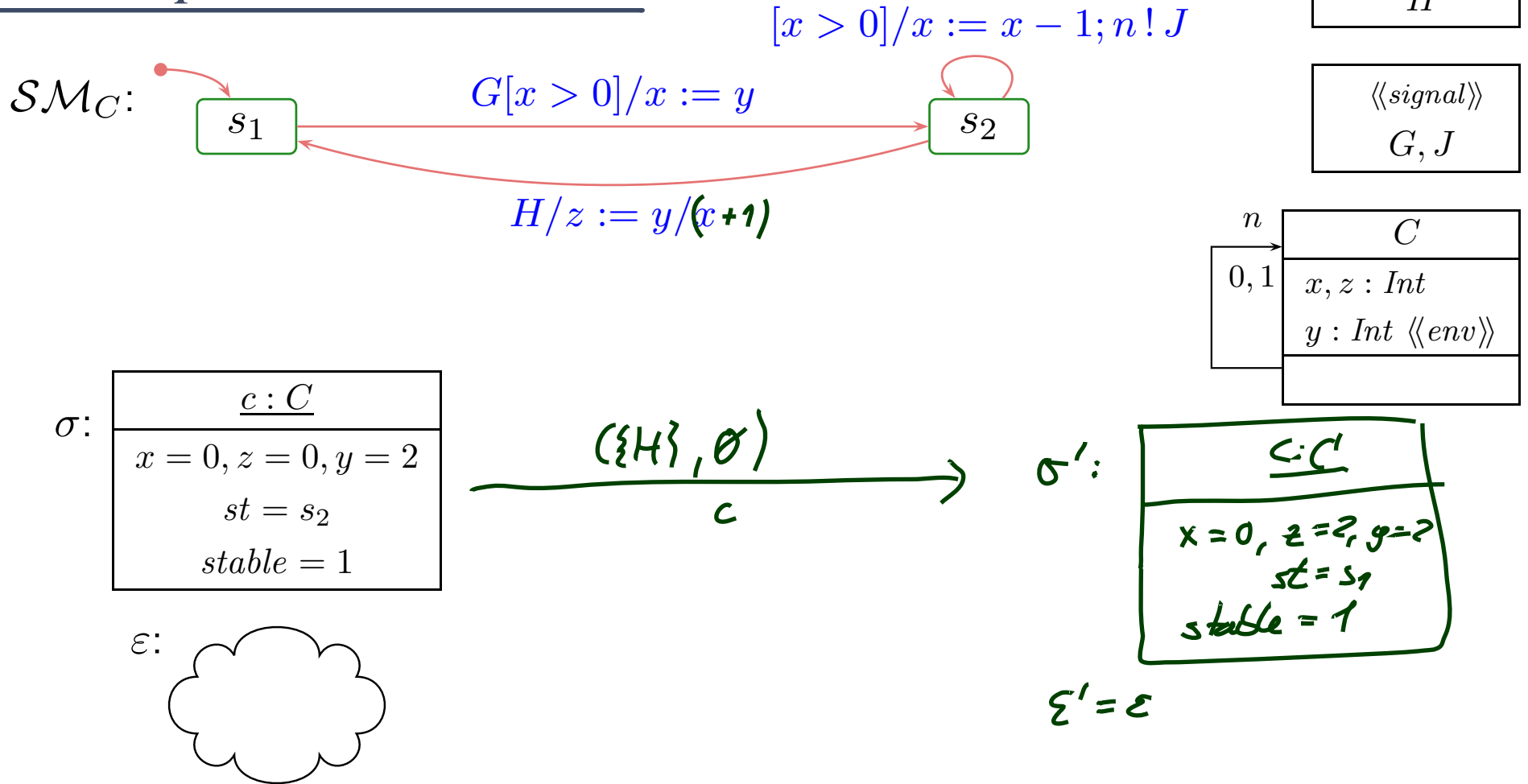
- Values of input attributes change freely in alive objects, i.e.

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

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

- $\varepsilon' = \varepsilon$.

Example: Environment



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

(v) Error Conditions

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

if, in (ii) or (iii),

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

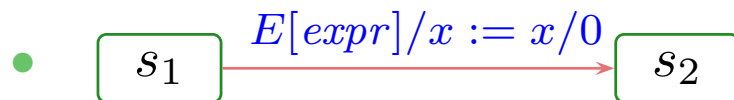
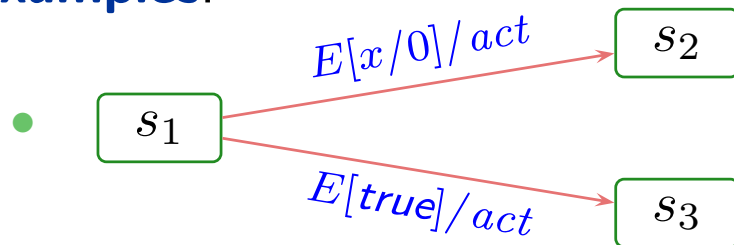
plus

$$\# \xrightarrow[\cup]{(\emptyset, \emptyset)} \#$$

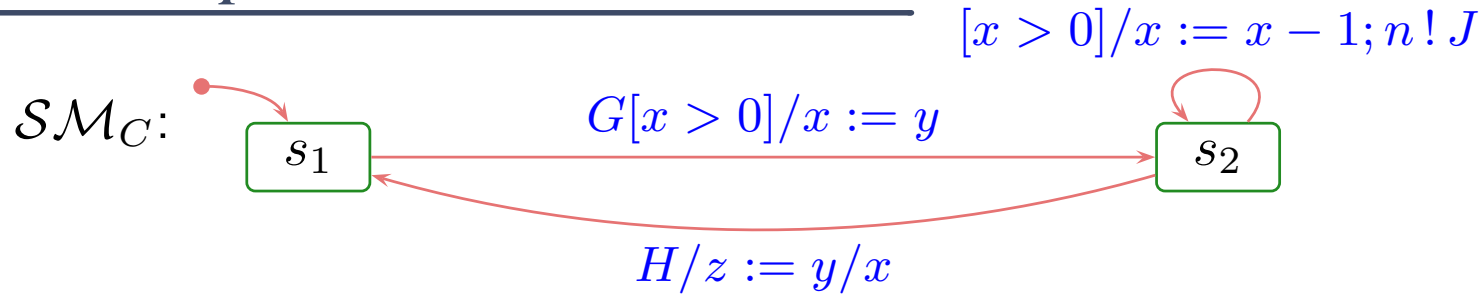
and

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

Examples:



Example: Error Condition



$\langle\langle \text{signal}, \text{env} \rangle\rangle$
H

$\langle\langle \text{signal} \rangle\rangle$
G, J

n	C
$0, 1$	$x, z : \text{Int}$ $y : \text{Int} \langle\langle \text{env} \rangle\rangle$

σ :

$c : C$
$x = 0, z = 0, y = 27$ $st = s_2$ $stable = 1$

ε :

- $I[\text{expr}]$ 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.

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.

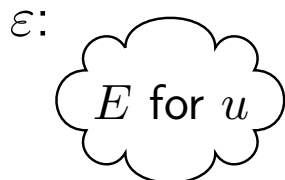
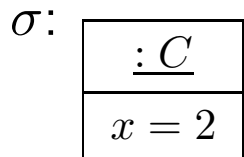
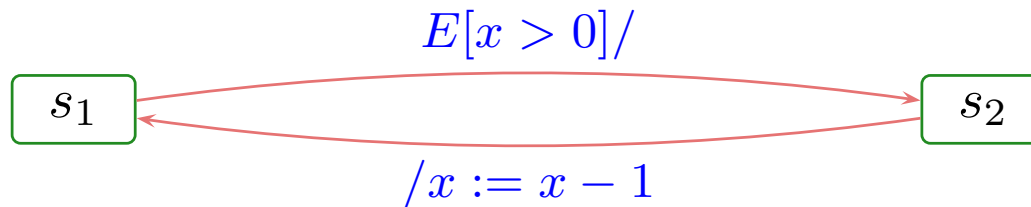
Notions of Steps: The Run-to-Completion Step

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:



Notions of Steps: The Run-to-Completion 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.

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

- $u_{n-1} = u$ 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 $1 \leq i \leq N$. 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 $\sigma_0(u)$).

Divergence

We say, object u **can diverge** on reception $cons$ from (local) configuration $\sigma_0(u)$ if and only if there is an infinite, consecutive sequence

$$(\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(cons_1, Snd_1)} \dots$$

such that u doesn't become stable again.

- **Note:** disappearance of object not considered in the definitions.
By the current definitions, it's neither divergence nor an RTC-step.

Run-to-Completion Step: Discussion.

What people may **dislike** on our definition of RTC-step is that it takes a **global** and **non-compositional** view. That is:

- In the projection onto a single object we still **see** the effect of interaction with other objects.
- Adding classes (or even objects) may change the divergence behaviour of existing ones.
- Compositional would be: the behaviour of a set of objects is determined by the behaviour of each object “in isolation”.

Our semantics and notion of RTC-step doesn't have this (often desired) property.

Can we give (syntactical) criteria such that any global run-to-completion step is an interleaving of local ones?

Maybe: Strict interfaces.

(Proof left as exercise...)

- **(A)**: Refer to private features only via “self” .
(Recall that other objects of the same class can modify private attributes.)
- **(B)**: Let objects only communicate by events, i.e.
don't let them modify each other's local state via links **at all**.

Putting It All Together

The Missing Piece: Initial States

Recall: a labelled transition system is (S, \rightarrow, S_0) . We **have**

- S : system configurations (σ, ε)
- \rightarrow : labelled transition relation $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$.

Wanted: initial states S_0 .

Proposal:

Require a (finite) set of **object diagrams** \mathcal{OD} as part of a UML model

$$(\mathcal{CD}, \mathcal{SM}, \mathcal{OD}).$$

And set

$$S_0 = \{(\sigma, \varepsilon) \mid \sigma \in G^{-1}(\mathcal{OD}), \mathcal{OD} \in \mathcal{OD}, \varepsilon \text{ empty}\}.$$

Other Approach: (used by Rhapsody tool) multiplicity of classes.

We can read that as an abbreviation for an object diagram.

Semantics of UML Model — So Far

The **semantics** of the **UML model**

$$\mathcal{M} = (\mathcal{C}\mathcal{D}, \mathcal{SM}, \mathcal{O}\mathcal{D})$$

where

- some classes in $\mathcal{C}\mathcal{D}$ are stereotyped as 'signal' (standard), some signals and attributes are stereotyped as 'external' (non-standard),
- there is a 1-to-1 relation between classes and state machines,
- $\mathcal{O}\mathcal{D}$ is a set of object diagrams over $\mathcal{C}\mathcal{D}$,

is the **transition system** (S, \rightarrow, S_0) constructed on the previous slide.

The **computations of** \mathcal{M} are the computations of (S, \rightarrow, S_0) .

OCL Constraints and Behaviour

- Let $\mathcal{M} = (\mathcal{CD}, \mathcal{IM}, \mathcal{OD})$ be a UML model.
- We call \mathcal{M} **consistent** iff, for each OCL constraint $expr \in Inv(\mathcal{CD})$,
 $\sigma \models expr$ for each “reasonable point” (σ, ε) of computations of \mathcal{M} .
(Cf. exercises and tutorial for discussion of “reasonable point”.)

Note: we could define $Inv(\mathcal{IM})$ similar to $Inv(\mathcal{CD})$.

Pragmatics:

- In **UML-as-blueprint mode**, if \mathcal{IM} doesn't exist yet, then $\mathcal{M} = (\mathcal{CD}, \emptyset, \mathcal{OD})$ is typically asking the developer to provide \mathcal{IM} such that $\mathcal{M}' = (\mathcal{CD}, \mathcal{IM}, \mathcal{OD})$ is consistent.

If the developer makes a mistake, then \mathcal{M}' is inconsistent.

- **Not common:** if \mathcal{IM} is given, then constraints are also considered when choosing transitions in the RTC-algorithm. In other words: even in presence of mistakes, the \mathcal{IM} never move to inconsistent configurations.

References

References

- [Crane and Dingel, 2007] Crane, M. L. and Dingel, J. (2007). UML vs. classical vs. rhapsody statecharts: not all models are created equal. *Software and Systems Modeling*, 6(4):415–435.
- [Damm et al., 2003] Damm, W., Josko, B., Votintseva, A., and Pnueli, A. (2003). A formal semantics for a UML kernel language 1.2. IST/33522/WP 1.1/D1.1.2-Part1, Version 1.2.
- [Fecher and Schönborn, 2007] Fecher, H. and Schönborn, J. (2007). UML 2.0 state machines: Complete formal semantics via core state machines. In Brim, L., Haverkort, B. R., Leucker, M., and van de Pol, J., editors, *FMICS/PDMC*, volume 4346 of *LNCS*, pages 244–260. Springer.
- [Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.
- [Harel and Kugler, 2004] Harel, D. and Kugler, H. (2004). The rhapsody semantics of statecharts. In Ehrig, H., Damm, W., Große-Rhode, M., Reif, W., Schnieder, E., and Westkämper, E., editors, *Integration of Software Specification Techniques for Applications in Engineering*, number 3147 in *LNCS*, pages 325–354. Springer-Verlag.
- [OMG, 2007] OMG (2007). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.