Software Design, Modelling and Analysis in UML

Lecture 12: Core State Machines II

2015-12-15

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Basic causality model
- Ether/event pool
- System configuration

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:**
  - System configuration cont'd
  - Transformers
  - Step, Run-to-Completion Step

System Configuration

System Configuration

Definition. Let $\mathscr{S}_0 = (\mathscr{T}_0, \mathscr{C}_0, V_0, atr_0, \mathscr{E})$ be a signature with signals, $\mathscr{D}_0$ a structure of $\mathscr{S}_0$, $(Eth, ready, \oplus, \ominus, [\,\cdot\,])$ an ether over $\mathscr{S}_0$ and $\mathscr{D}_0$.

Furthermore assume there is one core state machine M_C per class $C \in \mathscr{C}$.

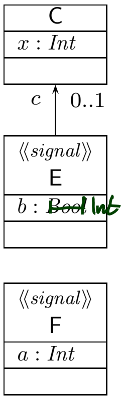
A system configuration over $\mathscr{S}_0$, $\mathscr{D}_0$, and Eth is a pair

$$(\sigma, \varepsilon) \in \Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth$$

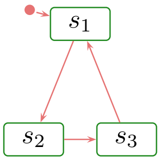
where

- $$\begin{aligned} \mathscr{S} = (\mathscr{T}_0 \,\dot\cup\, \{S_{M_C} \mid C \in \mathscr{C}_0\}, \quad & \mathscr{C}_0, \\ & V_0 \,\dot\cup\, \{\langle stable : Bool, -, \textit{true}, \emptyset\rangle\} \\ & \quad \dot\cup\, \{\langle st_C : S_{M_C}, +, s_0, \emptyset\rangle \mid C \in \mathscr{C}\} \\ & \quad \dot\cup\, \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset\rangle \mid E \in \mathscr{E}_0\}, \\ & \{C \mapsto atr_0(C) \\ & \quad \cup \{stable, st_C\} \cup \{params_E \mid E \in \mathscr{E}_0\} \mid C \in \mathscr{C}\}, \quad \mathscr{E}_0) \end{aligned}$$
- $\mathscr{D} = \mathscr{D}_0 \,\dot\cup\, \{S_{M_C} \mapsto S(M_C) \mid C \in \mathscr{C}\}$, and
- $\sigma(u)(r) \cap \mathscr{D}(\mathscr{E}_0) = \emptyset$ for each $u \in \mathrm{dom}(\sigma)$ and $r \in V_0$.

System Configuration: Example



$\mathcal{SM}_C$:



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

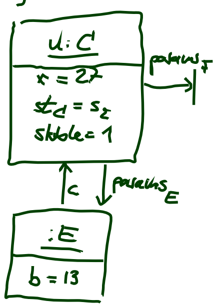
- $\mathscr{S} = (\mathscr{T}_0 \dot\cup \{S_{M_C} \mid C \in \mathscr{C}\}, \quad \mathscr{C}_0,$
  $V_0 \dot\cup \{\langle stable : Bool, -, \mathsf{true}, \emptyset\rangle\} \dot\cup \{\langle st_C : S_{M_C}, +, s_0, \emptyset\rangle \mid C \in \mathscr{C}\}$
  $\dot\cup \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset\rangle \mid E \in \mathscr{E}_0\},$
  $\{C \mapsto atr_0(C) \cup \{stable, st_C\} \cup \{params_E \mid E \in \mathscr{E}_0\} \mid C \in \mathscr{C}\}, \quad \mathscr{E}_0)$
- $\mathscr{D} = \mathscr{D}_0 \dot\cup \{S_{M_C} \mapsto S(M_C) \mid C \in \mathscr{C}\}$, and
- $\sigma(u)(r) \cap \mathscr{D}(\mathscr{E}_0) = \emptyset$ for each $u \in \mathrm{dom}(\sigma)$ and $r \in V_0$.

$\mathscr{S}_0 = (\{Int\},$
$\{C\},$
$\{x : Int, b : Int,$
$a : Int, c : C_{0,1}\},$
$\{C \mapsto \{x\},$
$E \mapsto \{b, c\},$
$F \mapsto \{a\}\},$
$\{E, F\})$

$\mathscr{S} = (\{Int, Bool, S_{M_C}\},$
$\{C\},$
$\{x : Int, b : Int,$
$a : Int, c : C_{0,1},$
$stable : Bool,$
$st_C : S_{M_C},$
$params_E : E_{0,1},$
$params_F : F_{0,1}\},$
$\{C \mapsto \{x, st_C,$
$stable, params_E,$
$params_F\},$
$E \mapsto \{b, c\},$
$F \mapsto \{a\}\},$
$\{E, F\})$

$\mathscr{D}(S_{M_C}) = \{s_1, s_2, s_3\}$

$\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$:



System Configuration Step-by-Step

- We start with some signature with signals $\mathscr{S}_0 = (\mathscr{T}_0, \mathscr{C}_0, V_0, atr_0, \mathscr{E})$.
- A **system configuration** is a pair (σ, ε) which comprises a system state σ wrt. $\mathscr{S}$ (not wrt. $\mathscr{S}_0$).
- Such a **system state** σ wrt. $\mathscr{S}$ provides, for each object $u \in \mathrm{dom}(\sigma)$,
  - values for the **explicit attributes** in V_0,
  - values for a number of **implicit attributes**, namely
    - a **stability flag**, i.e. $\sigma(u)(stable)$ is a boolean value,
    - a **current (state machine) state**, i.e. $\sigma(u)(st)$ denotes one of the states of core state machine M_C,
    - a temporary association to access **event parameters** for each class, i.e. $\sigma(u)(params_E)$ is defined for each $E \in \mathscr{E}$.
- For convenience require: there is **no link to an event** except for $params_E$.

Stability

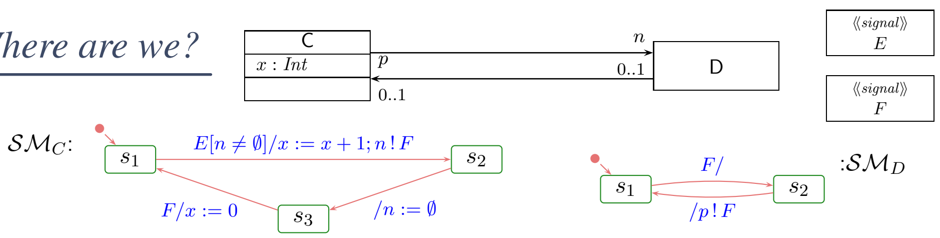
Definition.
Let (σ, ε) be a system configuration over some $\mathscr{S}_0$, $\mathscr{D}_0$, Eth.

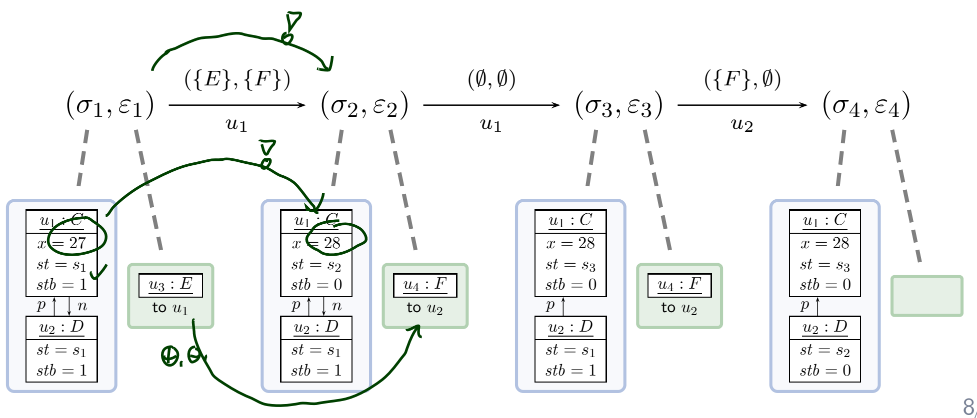
We call an object $u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(\mathscr{C}_0)$ stable in σ if and only if

$$\sigma(u)(stable) = \mathit{true}.$$

And unstable otherwise.

Where are we?





Transformer

Recall

- The (simplified) syntax of transition annotations:

$$annot ::= \big[\quad \langle event \rangle \quad [\text{ '['} \langle guard \rangle \text{ ']' }] \quad [\text{ '/' } \langle action \rangle] \quad \big]$$

- **Clear**: $\langle event \rangle$ is from $\mathscr{E}$ of the corresponding signature.
- **But:** What are $\langle guard \rangle$ and $\langle action \rangle$?
  - UML can be viewed as being **parameterized** in **expression language** (providing $\langle guard \rangle$) and **action language** (providing $\langle action \rangle$).
  - **Examples**:
    - **Expression Language**:
      - OCL
      - Java, C++, . . . expressions
      - . . .
    - **Action Language**:
      - UML Action Semantics, "Executable UML"
      - Java, C++, . . . statements (plus some event send action)
      - . . .
      - $\longrightarrow$

Needed: Semantics

$$I[\![expr]\!](\sigma, u) := \begin{cases} 1, & \text{if } I_{OCL}[\![expr]\!](\sigma, \{self \mapsto u\}) = 1 \\ 0, & \text{if } I_{OCL}[\![expr]\!](\sigma, \{self \mapsto u\}) = 0 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

In the following, we assume that we're **given**

- an **expression language** $Expr$ for guards, and
- an **action language** Act for actions,

and that we're **given**

- a **semantics** for boolean expressions in form of a partial function

$$I[\![\,\cdot\,]\!](\,\cdot\,,\,\cdot\,) : Expr \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times \mathscr{D}(\mathscr{C}) \rightharpoonup \mathbb{B}$$

  which evaluates expressions in a given system configuration,

  Assuming I to be partial is a way to treat "undefined" during runtime. If I is not defined (for instance because of dangling-reference navigation or division-by-zero), we want to go to a designated "error" system configuration.

- a **transformer** for each action: for each $act \in Act$, we assume to have

$$t_{act} \subseteq \mathscr{D}(\mathscr{C}) \times (\Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth) \times (\Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth)$$

Transformer

Definition.
Let $\Sigma_{\mathscr{S}}^{\mathscr{D}}$ the set of system configurations over some $\mathscr{S}_0$, $\mathscr{D}_0$, Eth.

We call a relation

$$t \subseteq \mathscr{D}(\mathscr{C}) \times (\Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth) \times (\Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth)$$

a (system configuration) **transformer**.

Example:

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

Observations

- In the following, we assume that
  - each application of a transformer t
  - to some system configuration (σ, ε)
  - for object u_x

  is associated with a set of **observations**

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

- An observation

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

  represents the information that,
  as a "side effect" of object u_x executing t in system configuration (σ, ε),
  the event u_e has been sent to u_{dst}.

  Special cases: creation ('$*$') / destruction ('$+$').

A Simple Action Language

In the following we use

$$\begin{aligned}
Act_{\mathscr{S}} = \{&\texttt{skip}\} \\
&\cup \{\texttt{update}(expr_1, v, expr_2) \mid expr_1, expr_2 \in Expr_{\mathscr{S}}, v \in atr\} \\
&\cup \{\texttt{send}(E(expr_1, ..., expr_n), expr_{dst}) \mid expr_i, expr_{dst} \in Expr_{\mathscr{S}}, E \in \mathscr{E}\} \\
&\cup \{\texttt{create}(C, expr, v) \mid C \in \mathscr{C}, expr \in Expr_{\mathscr{S}}, v \in V\} \\
&\cup \{\texttt{destroy}(expr) \mid expr \in Expr_{\mathscr{S}}\}
\end{aligned}$$

and OCL expressions over $\mathscr{S}$ (with partial interpretation) as $Expr_{\mathscr{S}}$.

Transformer Examples: Presentation

abstract syntax **concrete syntax**

`op`

intuitive semantics

$\ldots$

well-typedness

$\ldots$

semantics

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

or

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

observables

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

(error) conditions

Not defined if $\ldots$

Transformer: Skip

abstract syntax **concrete syntax**

`skip` skip

intuitive semantics

do nothing

well-typedness

./.

semantics

$t_{\mathtt{skip}}[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}$

observables

$Obs_{\mathtt{skip}}[u_x](\sigma, \varepsilon) = \emptyset$

(error) conditions

Transformer: Update

example:
$x := x + 1$
$(\mathit{self}.x := \mathit{self}.x + 1)$

abstract syntax
$\texttt{update}(expr_1, v, expr_2)$

concrete syntax
$expr_1 . v := expr_2$

intuitive semantics
Update attribute v in the object denoted by $expr_1$ to the value denoted by $expr_2$.

well-typedness
$expr_1 : T_C$ and $v : T \in atr(C)$; $\quad expr_2 : T$;
$expr_1, expr_2$ obey visibility and navigability

semantics

either does not change

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

change local state of object u

new value

where $\sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto \underbrace{I[\![expr_2]\!](\sigma, u_x)}]]$ with

$$u = \underbrace{I[\![expr_1]\!](\sigma, u_x)}.$$

object denoted by $expr_1$ (relative u_x as self)

observables

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

(error) conditions
Not defined if $I[\![expr_1]\!](\sigma, u_x)$ or $I[\![expr_2]\!](\sigma, u_x)$ not defined.

Update Transformer Example

$\mathcal{SM}_C$:

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

$\underbrace{\mathit{self}.x}_{expr_1}$ $\underbrace{:=}_{v}$ $\underbrace{\mathit{self}.x + 1}_{expr_2}$

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

exactly attribute v changes

σ:

$u_1 : C$
$x = 4$
$y = 0$
$st = s_1$
$stable = 0$

$\xrightarrow{\quad t_{update}[u_1] \quad}$

$u_1 : C$
$x = 5$
$y = 0$
$st = s_1$
$stable = 0$

$:\sigma'$

ε:

- $u = I[\![\mathit{self}]\!](\sigma, u_1)$
  $= I_{\alpha_L}[\![\mathit{self}]\!](\sigma, \underbrace{\{\mathit{self} \mapsto u_1\}}_{=:\beta})$
  $= u_1$
- $I[\![\mathit{self}.x + 1]\!](\sigma, \beta)$
  $= I_{\alpha}[\![+(x(\mathit{self}), 1)]\!](\sigma, \beta) = I(+)(I[\![x(\mathit{self})]\!](\sigma, \beta), I(1))$
  $= I(+)(4, I(1)) = 4 + 1 = 5$

$:\varepsilon' = \varepsilon$

Transformer: Send

abstract syntax $\quad$ **concrete syntax**

$\texttt{send}(E(expr_1, ..., expr_n), expr_{dst})$ $\quad$ $expr_{dst}\ !\ E(expr_1, ..., expr_n)$

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

$E \in \mathscr{E}$; $atr(E) = \{v_1 : T_1, \dots, v_n : T_n\}$; $expr_i : T_i$, $1 \le i \le n$;
$expr_{dst} : T_D$, $C, D \in \mathscr{C} \setminus \mathscr{E}$;
all expressions obey visibility and navigability in C

semantics

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

new signal instance

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

if $u_{dst} = I[\![expr_{dst}]\!](\sigma, u_x) \in \mathrm{dom}(\sigma)$; $\quad d_i = I[\![expr_i]\!](\sigma, u_x)$ for $1 \le i \le n$;

$u \in \mathscr{D}(E)$ a fresh identity, i.e. $u \notin \mathrm{dom}(\sigma)$,

2 and where $(\sigma', \varepsilon') = (\sigma, \varepsilon)$ if $u_{dst} \notin \mathrm{dom}(\sigma)$. $\}$ sending to a non-alive object: do nothing

observables

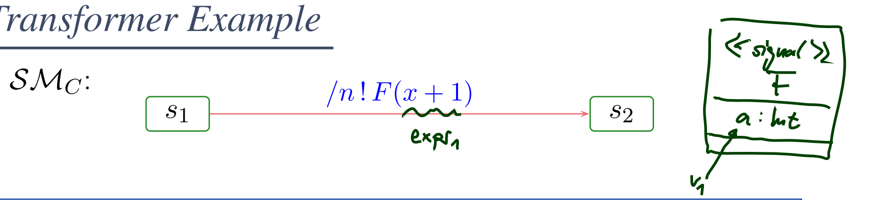
$Obs_{\texttt{send}}[u_x] = \{(u_x, u_{dst})\}$

(error) conditions

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

Send Transformer Example

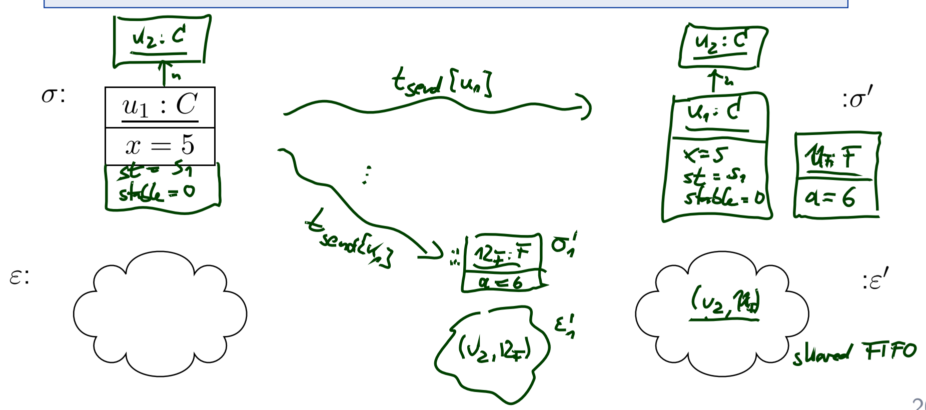
$\mathcal{SM}_C$:



$t_{\texttt{send}(expr_{src},E(expr_1,\dots,expr_n),expr_{dst})}[u_x](\sigma, \varepsilon) \ni (\sigma', \varepsilon')$ iff $\varepsilon' = \varepsilon \oplus (u_{dst}, u)$;

$\sigma' = \sigma \mathbin{\dot\cup} \{u \mapsto \{v_i \mapsto d_i \mid 1 \le i \le n\}\}$; $u_{dst} = I[\![expr_{dst}]\!](\sigma, u_x) \in \mathrm{dom}(\sigma)$;

$d_i = I[\![expr_i]\!](\sigma, u_x)$, $1 \le i \le n$; $u \in \mathscr{D}(E)$ a fresh identity;



Sequential Composition of Transformers

$act_1 ; act_2$

$t_{act_2}(t_{act_1}(\cdot))$

- **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.

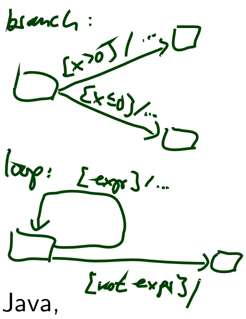
Transformers And Denotational Semantics

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 (later),

but not **possibly diverging loops**.



Our (Simple) Approach: if the action language is, e.g. Java,
then (**syntactically**) forbid loops and calls of recursive functions.

Other Approach: use full blown denotational semantics.

No show-stopper, because loops in the action annotation can be converted into transition cycles in the state machine.

References

References

Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.