Software Design, Modelling and Analysis in UML

Lecture 14: Hierarchical State Machines I

2016-12-22

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

Albert-Ludwigs-Universität Freiburg, Germany

Content

Putting It All Together

Initial States

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

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

Wanted: initial states S_0.

Proposal:
Require a (finite) set of **object diagrams** $\mathscr{OD}$ as part of a UML model

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

And set

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

Other Approach: (used by Rhapsody tool) multiplicity of classes (plus initialisation code).
We can read that as an abbreviation for an object diagram.

Semantics of UML Model (So Far)

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

$$\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$$

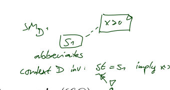
where

- some classes in $\mathscr{CD}$ 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,
- $\mathscr{OD}$ is a set of object diagrams over $\mathscr{CD}$,

is the **transition system** $(S, A, \to, S_0)$ constructed on the previous slide(s).

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

OCL Constraints and Behaviour

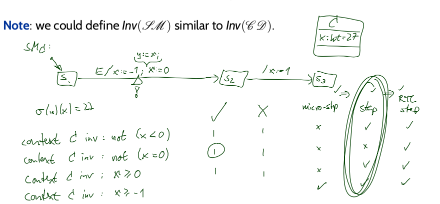


- Let $\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$ be a UML model.
- We call $\mathcal{M}$ **consistent** iff, for each OCL constraint $expr \in Inv(\mathscr{CD})$,

  $\sigma \models expr$ for each "reasonable point" (σ, ε) of computations of $\mathcal{M}$.

  (Cf. tutorial for discussion of "reasonable point".)

Note: we could define *Inv*$(\mathscr{SM})$ similar to *Inv*$(\mathscr{CD})$.



Last Missing Piece: Create and Destroy Transformer

Transformer: Create

abstract syntax — `create`$(C, expr, v)$ — **concrete syntax** — $expr.v := new\ C$

intuitive semantics

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

well-typedness

$$expr : T_D, v \in atr(D),$$
$$atr(C) = \{\langle v_1 : T_1, expr_i^0\rangle \mid 1 \le i \le n\}$$

semantics

...

observables

...

(error) conditions

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

x = (new C).y + (new D).z;
can be written as
tmp_1 := new C;
tmp_2 := new D;
x := tmp_1.y + tmp_2.z;

Transformer: Create

abstract syntax — `create`$(C, expr, v)$ — **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

$$expr : T_D, v \in atr(D),$$
$$atr(C) = \{\langle v_1 : T_1, expr_i^0\rangle \mid 1 \le i \le n\}$$

semantics

...

observables

...

(error) conditions

$I[\![expr]\!](\sigma, \beta)$ 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 since then expressions would need to modify the system state.
- Also for simplicity: no parameters to construction ($\sim$ parameters of constructor). Adding them is straightforward (but somewhat tedious).

How To Choose New Identities?

- **Re-use**: choose any identity that is not alive **now**, i.e. not in $\mathrm{dom}(\sigma)$.
  - 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 $\mathrm{dom}(\sigma)$ and any predecessor in current run.
  - Depends on history.
  - Dangling references remain dangling – could mask "dirty" effects of platform.

Transformer: Create

abstract syntax — `create`$(C, expr, v)$ — **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

$$expr : T_D, v \in atr(D),$$
$$atr(C) = \{\langle v_1 : T_1, expr_i^0\rangle \mid 1 \le i \le n\}$$

semantics

$$((\sigma, \varepsilon), (\sigma', \varepsilon')) \in t_{\texttt{create}(C, expr, v)}[u_x]$$

iff

$$\sigma' = \underbrace{\sigma[u_0 \mapsto \sigma(u_0)[v \mapsto u]]}_{\text{similar to update}} \cup \underbrace{\{u \mapsto \{v_i \mapsto d_i \mid 1 \le i \le n\}\}}_{\text{similar to send}},$$
$$\varepsilon' = [u](\varepsilon); \quad u \in \mathscr{D}(C) \text{ fresh, i.e. } u \notin \mathrm{dom}(\sigma);$$
$$u_0 = I[\![expr]\!](\sigma, u_x);\ d_i = I[\![expr_i^0]\!](\sigma, \emptyset) \text{ if } expr_i^0 \neq \text{ and}$$
arbitrary value from $\mathscr{D}(T_i)$ otherwise.

clean ethers

observables

$$Obs_{\texttt{create}}[u_x] = \{(*, u)\}$$

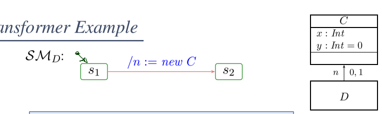
creation

(error) conditions

$expr_i^0$

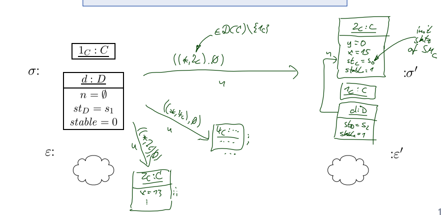
$I[\![expr]\!](\sigma, u_x)$ not defined.

Create Transformer Example



`create`$(C, expr, v)$

$t_{\texttt{create}(C, expr, v)}[u_x](\sigma, \varepsilon) = \ldots$



Transformer: Destroy

abstract syntax — **concrete syntax**

$\mathtt{destroy}(expr)$

intuitive semantics

Destroy the object denoted by expression $expr$.

well-typedness

$$expr : T_C, C \in \mathscr{C}$$

semantics

$$\ldots$$

observables

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

(error) conditions

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

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 $\sigma(u_1)(r)$?
- 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)
- 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 — **concrete syntax**

$\mathtt{destroy}(expr)$

intuitive semantics

Destroy the object denoted by expression $expr$.

well-typedness

$$expr : T_C, C \in \mathscr{C}$$

semantics

function restriction

$$t_{\mathtt{destroy}(expr)}[u_x](\sigma, \varepsilon) = \{(\sigma', \varepsilon')\} \qquad \varepsilon' = [u](\varepsilon)$$

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

observables

$$Obs_{\mathtt{destroy}(expr)}[u_x] = \{(+, u)\}$$

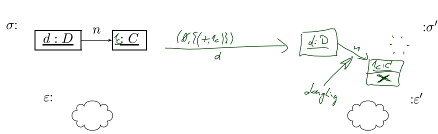
(error) conditions

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

Destroy Transformer Example

$\mathtt{destroy}(expr)$

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

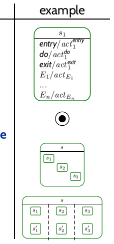


Hierarchical State-Machines

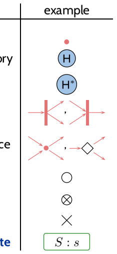
The Full Story

UML distinguishes the following **kinds of states**:

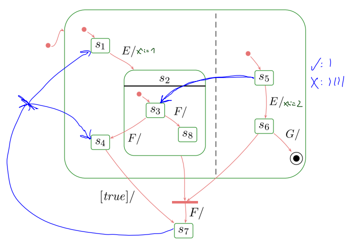
	example
simple state	
final state	
composite state	
OR	
AND	



	example
pseudo-state	
initial	
(shallow) history	
deep history	
fork/join	
junction, choice	
entry point	
exit point	
terminate	
submachine state	



Blessing or Curse. . . ?



Blessing or Curse. . . ?

Plan:

States / Syntax:

- What is the abstract syntax of a diagram?

States / Semantics:

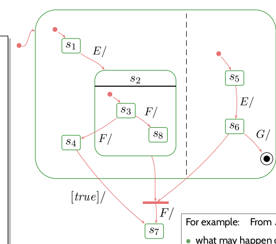
- what is the type of the implicit st attribute?
- what are **legal system configurations**?

Transitions / Syntax:

- what are **legal** / well-formed transitions?

Transitions / Semantics:

- when is a legal transition enabled?
- which effects do transitions have?



For example: From s_1, s_5,

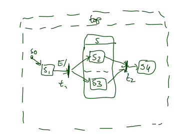
- what may happen on E?
- what may happen on $\underline{E}, F$?
- can $\underline{E}, G$ kill the object?
- ...

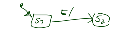
Representing All Kinds of States

- **So far**:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \quad \rightarrow \subseteq S \times (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}} \times S$$

set of states, init state, edges, source state, trigger, guard, action, dst. state





kind: init
kind: st
$\{s_1, s_2, \dots, s_4, top\}$,
$\{t_1, t_2\}$,
$\{t_1 \mapsto (\{s_1\}, \{s_2, s_3\}), \dots\}$
$\{t_1 \mapsto (E, true, skip)\}$
$s \mapsto \{\{s_2\}, \{s_3\}\}$,
$s_2 \mapsto \emptyset$
$top \mapsto \{\{s_0, s_1, s, s_4\}\}$

Representing All Kinds of States

- **So far**:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \quad \rightarrow \subseteq S \times (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}} \times S$$

- **From now on**: **(hierarchical) state machines**

$$(S, kind, region, \rightarrow, \psi, annot)$$

where

- $S \supseteq \{top\}$ is a finite set of states **(new: top)**,
- $kind : S \rightarrow \{$*st, init, fin, shist, dhist, fork, join, junc, choi, ent, exi, term*$\}$ is a function which labels states with their **kind**, **(new)**
- $region : S \rightarrow 2^{2^S}$ is a function which characterises the **regions** of a state, **(new)**
- $\rightarrow$ is a set of transitions, **(changed)**
- $\psi : (\rightarrow) \rightarrow 2^S \times 2^S$ is an **incidence function**, and **(new)**
- $annot : (\rightarrow) \rightarrow (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}}$ provides an annotation for each transition. **(new)**

(s_0 is then redundant – replaced by proper state (!) of kind '*init*'.)

Well-Formedness: Regions

	$\in S$	$kind$	$region \subseteq 2^S, S_i \subseteq S$	$child \subseteq S$
final state	s	*fin*	$\emptyset$	$\emptyset$
pseudo-state	s	*init*, ...	$\emptyset$	$\emptyset$
simple state	s	*st*	$\emptyset$	$\emptyset$
composite state	s	*st*	$\{S_1, \dots, S_n\}, n \geq 1$	$S_1 \cup \dots \cup S_n$
implicit top state	top	*st*	$\{S_1\}$	S_1

- Final and pseudo states **must not comprise** regions.
- States $s \in S$ with $kind(s) =$ *st* **may comprise** regions.
  Naming conventions can be defined based on regions:
  - No region: simple state.
  - One region: OR-state.
  - Two or more regions: AND-state.
- Each state (except for top) **must** lie in exactly one region.
- **Note**: The region function induces a **child** function.
- **Note**: Diagramming tools (like Rhapsody) can ensure well-formedness.

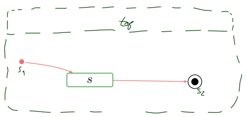
From UML to Hierarchical State Machine: By Example

$$(S, kind, region, \rightarrow, \psi, annot)$$

	example	$\in S$	$kind$	$region$
simple state	s	s	st	$\emptyset$
final state	q	q	fin	$\emptyset$
composite state				
OR		s	st	$\{\{s_1, s_2, s_3\}\}$
AND		s	st	$\{\{s_1, s_1'\}, \{s_2, s_2'\}, \{s_3, s_3'\}\}$
submachine state	(later)			
pseudo-state	q, ...	q	init, ...	

$(s, kind(s))$ for short

From UML to Hierarchical State Machine: By Example



... denotes $(S, kind, region, \to, \psi, annot)$ with

- $S = \{top, s_1, s, s_2\}$
- $kind = \{top \mapsto$ **st**$, s_1 \mapsto$ ***init***$, s \mapsto$ **st**$, s_2 \mapsto$ ***fin***$\}$
- or $(S, kind) = \{(top,$ **st**$), (s_1,$ ***init***$), (s,$ **st**$), (s_2,$ ***fin***$)\}$
- $region = \{top \mapsto \{\{s_1, s, s_2\}\}, s_1 \mapsto \emptyset \quad , s \mapsto \emptyset \quad , s_2 \mapsto \emptyset \quad \}$
- $\to, \psi, annot$: in a minute.

Recall

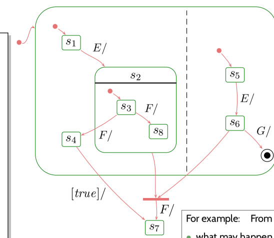
Plan:

States / Syntax:
- What is the abstract syntax of a diagram? ✓

States / Semantics:
- what is the type of the implicit st attribute?
- what are **legal system configurations?**

Transitions / Syntax:
- what are **legal** / well-formed transitions?

Transitions / Semantics:
- when is a legal transition enabled?
- which effects do transitions have?



For example: From s_1, s_5,
- what may happen on E?
- what may happen on $\underline{E}, F$?
- can $\underline{E}, G$ kill the object?
- ...

Semantics: State Configuration

- The type of (implicit attribute) st is from now on **a set of** states, i.e. $\mathscr{D}(S_{M_C}) = 2^S$
- A set $S_1 \subseteq S$ is called **(legal) state configuration** if and only if
  - $top \in S_1$, and
  - for each region R of a state in S_1, exactly one (non pseudo-state) element of R is in S_1, i.e.

$$\forall s \in S_1 \, \forall R \in region(s) \bullet |\{s \in R \mid kind(s) \in \{\textsf{st}, \textit{fin}\}\} \cap S_1| = 1.$$

- **Examples:**

$S_1 = \{s_0\}$ ✗
$S_2 = \{s_0, top\}$ ✓
$S_3 = \{s_1, top\}$ ✗
$S_4 = \{s_1, top, s_3, s_4\}$ ✗
$S_5 = \{s_1, top, s_4\}$ ✓
$S_6 = \{s_5, top, s_6, s_9\}$ ✗
$S_7 = \{s_5, top, s_6, s_9, s_{10}\}$ ✓
$S_8 = \{top, s_0, s_1, s_2\}$ ✗

Recall

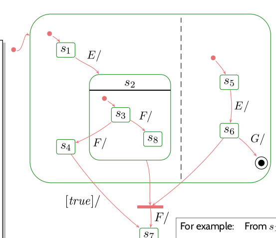
Plan:

States / Syntax:
- What is the abstract syntax of a diagram? ✓

States / Semantics:
- what is the type of the implicit st attribute?
- what are **legal system configurations?** ✓

Transitions / Syntax:
- what are **legal** / well-formed transitions?

Transitions / Semantics:
- when is a legal transition enabled?
- which effects do transitions have?



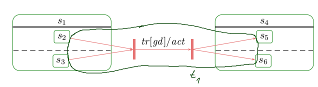
For example: From s_1, s_5,
- what may happen on E?
- what may happen on $\underline{E}, F$?
- can $\underline{E}, G$ kill the object?
- ...

Transitions Syntax: Fork/Join

- For simplicity, we consider transitions with (possibly) multiple sources and targets, i.e.

$$\psi : (\to) \to (2^S \setminus \emptyset) \times (2^S \setminus \emptyset)$$

- For instance,



translates to

$$(S, kind, region, \underbrace{\{t_1\}}_{\to}, \underbrace{\{t_1 \mapsto (\{s_2, s_3\}, \{s_5, s_6\})\}}_{\psi}, \underbrace{\{t_1 \mapsto (tr, gd, act)\}}_{annot})$$

- Naming convention: $\psi(t) = (source(t), target(t))$.

Orthogonal States

- Two states $s_1, s_2 \in S$ are called **orthogonal**, denoted $s_1 \perp s_2$, if and only if
  - they "live" in different regions of **one** AND-state, i.e.

$$\exists s, region(s) = \{S_1, \dots, S_n\}, 1 \leq i \neq j \leq n : s_1 \in child(S_i) \wedge s_2 \in child(S_j),$$

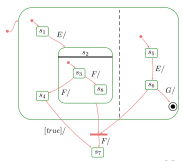
Legal Transitions

A hierarchical state-machine $(S, kind, region, \to, \psi, annot)$ is called **well-formed** if and only if for all transitions $t \in \to$,

- source (and destination) states are pairwise orthogonal, i.e.
  - $\forall\, s, s' \in source(t)\ (\in target(t)) \bullet s \perp s'$,
- the top state is neither source nor destination, i.e.
  - $top \notin source(t) \cup source(t)$.

Recall: final states are not sources of transitions.

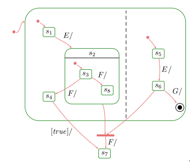
Example:



Plan

	example		example
simple state		**pseudo-state**	
		initial	
		(shallow) history	
		deep history	
		fork/join	
final state		junction, choice	
composite state		entry point	
OR		exit point	
AND		terminate	
		submachine state	$S : s$

- Transitions involving non-pseudo states.
- Initial pseudostate, final state.
- Entry/do/exit actions, internal transitions.
- History and other pseudostates, the rest.



Tell Them What You've Told Them. . .

- For the **Create Action**, we have two main choices:
  - **re-use** identities ("nasty semantics"),
  - use **fresh** identities ("clean semantics", depends on history).

  Similar for **Destroy**.
- **Hierarchical State Machines** introduce **Regions**.
  - Thereby, **states** can lie within **states** as **children**.
  - The implicit variable st becomes set-valued.
- **Transitions** may now have
  - **multiple** source states, **multiple** destination states,
  - but need to adhere to **well-formedness conditions**.
- **Enabledness** of a set (!) of transitions is **a bit tricky to define** ($\to$ scope, priority, maximality).
- **Steps** are a proper generalisation of core state machines.

References

References

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.