Software Design, Modelling and Analysis in UML

Lecture 4: OCL Semantics

2016-11-03

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

Albert-Ludwigs-Universität Freiburg, Germany

Content

- The **Object Constraint Language** (OCL):

  Semantics

  - Overview
  - OCL Types
  - Arithmetic / Logical Operators
  - OCL Expressions
  - Iterate

- **A Complete Example**

Recall

OCL Syntax 1/4: Expressions

$expr ::=$

w	$: \tau(w)$
$\mid expr_1 =_\tau expr_2$	$: \tau \times \tau \to Bool$ $\quad T_B$
$\mid$ oclIsUndefined$_\tau$($expr_1$)	$: \tau \to Bool$
$\mid \{expr_1, \dots, expr_n\}$	$: \tau \times \dots \times \tau \to Set(\tau)$
$\mid$ isEmpty($expr_1$)	$: Set(\tau) \to Bool$
$\mid$ size($expr_1$)	$: Set(\tau) \to Int$
$\mid$ allInstances$_C$	$: Set(\tau_C)$

$\mid v(expr_1)$	$: \tau_C \to \tau$	where	$v : \tau \in atr(C)$, $\tau \in \mathscr{T}$,
$\mid r_1(expr_1)$	$: \tau_C \to \tau_D$	where	$r_1 : D_{0,1} \in atr(C), C, D \in \mathscr{C}$,
$\mid r_2(expr_1)$	$: \tau_C \to Set(\tau_D)$	where	$r_2 : D_* \in atr(C), C, D \in \mathscr{C}$.

Where, given $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$,

- $w \in W \supseteq \{self_C : \tau_C \mid C \in \mathscr{C}\}$ is a set of typed logical variables, w has type $\tau(w)$
- τ is any type from $\mathscr{T} \cup T_B \cup T_\mathscr{C}$ $\cup \{Set(\tau_0) \mid \tau_0 \in \mathscr{T} \cup T_B \cup T_\mathscr{C}\}$
  - T_B is a set of (OCL) basic types, in the following we use $T_B = \{Bool, Int, String\}$
  - $T_\mathscr{C} = \{\tau_C \mid C \in \mathscr{C}\}$ is the set of object types,
  - $Set(\tau_0)$ denotes the set-of-τ_0 type for $\tau_0 \in T_B \cup T_\mathscr{C}$ (sufficient because of "flattening" (cf. standard)).

- 03 - 2016-10-27 - Sodsyn -

7/24

OCL Syntax 2/4: Constants & Arithmetics

For example:

$expr ::= \dots$

$\mid$ true $\mid$ false	$: Bool$
$\mid expr_1$ {and, or, implies} $expr_2$	$: Bool \times Bool \to Bool$
$\mid$ not $expr_1$	$: Bool \to Bool$
$\mid 0 \mid -1 \mid 1 \mid -2 \mid 2 \mid \dots$	$: Int$
$\mid expr_1 \{+, -, \dots\}\ expr_2$	$: Int \times Int \to Int$
$\mid expr_1 \{<, \leq, \dots\}\ expr_2$	$: Int \times Int \to Bool$
$\mid$ OclUndefined$_\tau$	$: \tau$

Generalised notation: (prefix normal form)

$$expr ::= \omega(expr_1, \dots, expr_n) \qquad : \tau_1 \times \dots \times \tau_n \to \tau$$

with $\omega \in \{+, -, \dots\}$

$$1 + 2 \rightsquigarrow \underbrace{+}_{\omega}(\underbrace{1}_{expr_1}, \underbrace{2}_{expr_2})$$

- 03 - 2016-10-27 - Sodsyn -

10/24

OCL Syntax 3/4: Iterate

$$expr ::= \dots \mid expr_1\text{->iterate}(w_1 : T_1 ; w_2 : T_2 = expr_2 \mid expr_3)$$

or, with a little renaming,

$$expr ::= \dots \mid expr_1\text{->iterate}(iter : T_1; result : T_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some τ_0),
- $iter \in W$ is called **iterator**, of the type denoted by T_1 (if T_1 is omitted, τ_0 is assumed as type of $iter$)
- $result \in W$ ($\neq iter$) is called **result variable**, gets type τ_2 denoted by T_2,
- $expr_2$ in an expression of type τ_2 giving the **initial value** for $result$, (OclUndefined$_{\tau_2}$, if omitted)
- $expr_3$ is an expression of type τ_2, in particular $iter$ and $result$ may appear in $expr_3$.

- 03 - 2016-10-27 - Sodsyn -

12/24

OCL Syntax 4/4: Context

Syntax: (Assuming signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.)

$$context ::= \text{context } w_1 : T_1, \ \dots, w_n : T_n \text{ inv} : expr$$

where $T_i \in \mathscr{C}$ and $w_i : \tau_{T_i} \in W$ for all $1 \leq i \leq n, n \geq 0$.

Semantics:

context $w_1 : C_1, \dots, w_n : C_n$ inv : $expr$

is (just) an **abbreviation** for

allInstances$_{C_1}$ -> forAll($w_1 : \tau_{C_1}$ |
  ...
    allInstances$_{C_n}$ -> forAll($w_n : \tau_{C_n}$ |
      $expr$
)
  ...
)

- 03 - 2016-10-27 - Sodsyn -

17/24

OCL Semantics: The Task

- **Given**
  - an OCL expression (over signature $\mathscr{S}$), e.g.

$$expr_1 = \mathsf{context}\ \ CP\ \ \mathsf{inv:}\ \ wen\ \ \mathsf{implies}\ \ dd\,.\,wis > 0$$

  - and a system state

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{wis \mapsto 13\},$$
$$3_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \mathit{true}\}, \quad 5_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \mathit{false}\}\} \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$$

  - and a valuation of the logical variables $\beta_1 : W \to I(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})$,

    $\cup \{self_C \mid C \in \mathscr{C}\}$

- **compute** the value $I[\![expr_1]\!](\sigma_1, \beta_1) \in \{\mathit{true}, \mathit{false}, \bot_{Bool}\}$ of $expr_1$ in σ_1 under β_1.

  ! three-valued logic

- More general: **Define** the **interpretation** $I[\![expr]\!](\sigma, \beta)$ of $expr$ **in** σ **under** β:

$$I[\![\,\cdot\,]\!](\,\cdot\,,\,\cdot\,) : OCLExpressions(\mathscr{S}) \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \to I(Bool)$$

OCL Semantics OMG (2006)

Basically business as usual...

(i) Equip each OCL (!) **type** with a reasonable **domain**, i.e. **define function**

$$I_{(i)} \text{ with } \mathrm{dom}(I_{(i)}) = \mathscr{T} \cup T_B \cup T_{\mathscr{C}}$$

(ii) Equip each **set type** $Set(\tau_0)$ with reasonable **domain**, i.e. **define function**

$$I_{(ii)} \text{ with } \mathrm{dom}(I_{(ii)}) = \{Set(\tau_0) \mid \tau_0 \in \mathscr{T} \cup T_B \cup T_{\mathscr{C}}\}$$

(iii) Equip each **arithmetical operation** with a reasonable **interpretation** (that is, with a **function** operating on the corresponding **domains**), i.e. **define function**

$$I \text{ with } \mathrm{dom}(I_{(iii)}) = \{+, -, \leq, \dots\}, \text{ e.g., } \underbrace{I(+)} \in I(Int) \times I(Int) \to I(Int)$$

(iv) Same game for **set operations**: **define function** $I_{(iv)}$ with $\mathrm{dom}(I) = \{\text{isEmpty}, \dots\}$

(v) Equip each **expression** with a reasonable **interpretation**, i.e. define function

$$I_{(v)} : Expr \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I_{(i)}(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \to I_{(i)}(Bool)$$

...except for OCL being a **three-valued logic**, and the "iterate" expression.

$$I := I_{(i)} \cup I_{(ii)} \cup I_{(iii)} \cup I_{(iv)} \cup I_{(v)}$$

(i) Domains of OCL and (!) Model Basic Types

Recall: **OCL basic types**

$$T_B = \{Bool, Int, String\}$$

We set:

three-valued

- $I(Bool) := \{true, false, \bot_{Bool}\}$
- $I(Int) := \mathbb{Z} \,\dot{\cup}\, \{\bot_{Int}\}$
- $I(String) := \ldots \,\dot{\cup}\, \{\bot_{String}\}$

disjoint union

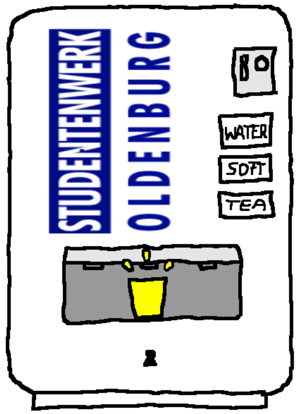
We may omit index τ of $\bot_\tau$ if it is clear from context.

Given signature $\mathscr{S}$ with **model basic types** $\mathscr{T}$ and domain $\mathscr{D}$, set

$$I(T) := \mathscr{D}(T) \,\dot{\cup}\, \{\bot_T\}$$

for each model basic type $T \in \mathscr{T}$.

OCL and Model Types?! An Example.



$$\begin{aligned}\mathscr{S} = (&\{Bool_M, Nat\}, \{VM, CP, DD\},\\ &\{cp : CP_*, dd : DD_{0,1}, wen : Bool, wis : Nat\},\\ &\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen, dd\}, DD \mapsto \{wis\})\end{aligned}$$

Model Types:

$\mathscr{D}(Bool_M) = \{0, 1\}$

$\mathscr{D}(Nat) = \{0, \dots, 255\}$

$\mathscr{D}(VM) = \mathbb{N} \times \{VM\}$
$= \{1_{VM}, 2_{VM}, \dots\}$

OCL Types:

$$\left.\begin{aligned} I(Bool) &= \{true, false, \bot\} \\ I(Int) &= \mathbb{Z} \,\dot\cup\, \{\bot_{Int}\} \end{aligned}\right\} \text{fixed for OCL } T_B$$

$I(Bool_M) = \mathscr{D}(Bool_M) \,\dot\cup\, \{\bot_{Bool_M}\}$
$= \{0, 1, \bot_{Bool_M}\}$

$I(Nat) = \mathscr{D}(Nat) \,\dot\cup\, \{\bot_{Nat}\}$
$= \{0, \dots, 255\} \,\dot\cup\, \{\bot_{Nat}\}$

$I(\tau_{VM}) = \mathscr{D}(VM) \,\dot\cup\, \{\bot_{VM}\}$

(i) Domains of Object and (ii) Set Types

- Let τ_C be an (OCL) **object type** for a class $C \in \mathscr{C}$.
- We set

$$I(\tau_C) := \mathscr{D}(C) \mathbin{\dot{\cup}} \{\bot_{\tau_C}\}$$

- Let τ be a type from $\mathscr{T} \cup T_B \cup T_{\mathscr{C}}$.
- We set

$$I(Set(\tau)) := 2^{I(\tau)} \mathbin{\dot{\cup}} \{\bot_{Set(\tau)}\}$$

Note: in the OCL standard, only **finite** subsets of $I(\tau)$.

Infinity doesn't scare **us**, so we simply allow it.

(iii) Interpretation of Arithmetic Operations

- **Literals** map to fixed values:

$$\underset{\mathrm{OclExpr}(\varphi)}{I(\text{true})} := \overset{I(Bool)}{\textit{true}}, \quad I(\text{false}) := \overset{I(Bool)}{\textit{false}}, \quad I(0) := \overset{I(Int)}{0}, \quad I(1) := 1, \ldots$$

$$I(\text{OclUndefined}_\tau) := \bot_\tau$$

- **Boolean operations** (defined point-wise for $x_1, x_2 \in I(\tau)$):

$$I(=_\tau)(\overset{I(\tau)}{x_1}, \overset{I(\tau)}{x_2}) := \begin{cases} \mathit{true} & \text{, if } x_1 \neq \bot_\tau \neq x_2 \text{ and } x_1 = x_2 \\ \mathit{false} & \text{, if } x_1 \neq \bot_\tau \neq x_2 \text{ and } x_1 \neq x_2 \\ \bot_{Bool} & \text{, otherwise} \end{cases}$$

$$I(=_\tau) : I(\tau) \times I(\tau) \to I(Bool)$$

- **Logical connectives**, e.g. $I(\text{and})(\cdot,\cdot) : \{\mathit{true}, \mathit{false}, \bot\} \times \{\mathit{true}, \mathit{false}, \bot\} \to \{\mathit{true}, \mathit{false}, \bot\}$ is defined by the following truth table:

x_1	*true*	*true*	*true*	*false*	*false*	*false*	$\bot$	$\bot$	$\bot$
x_2	*true*	*false*	$\bot$	*true*	*false*	$\bot$	*true*	*false*	$\bot$
$I(\text{and})(x_1, x_2)$	*true*	*false*	$\bot$	*false*	*false*	*false*	$\bot$	*false*	$\bot$

We assume common logical connectives not, or, … with the canonical 3-valued interpretation.

(iii) Interpretation of OclIsUndefined

- The **is-undefined** predicate (defined point-wise for $x \in I(\tau)$):

$$I(\text{oclIsUndefined}_\tau)(\overset{I(\tau)}{x}) := \begin{cases} \mathit{true} & \text{, if } x = \bot_\tau \\ \mathit{false} & \text{, otherwise} \end{cases}$$

  Note: $I(\text{oclIsUndefined}_\tau)$ is **definite**, i.e., it never yields $\bot$.

- **Integer operations** (defined point-wise for $x_1, x_2 \in I(\mathit{Int})$):

$$I(+)(x_1, x_2) := \begin{cases} x_1 + x_2 & \text{, if } x_1 \neq \bot \neq x_2 \\ \bot & \text{, otherwise} \end{cases}$$

Note: There is a **common principle**.

The **interpretation** of an operation (symbol)

$$\omega : \tau_1 \times \dots \tau_n \to \tau, \quad n \geq 0$$

is a function

$$I(\omega) : I(\tau_1) \times \dots \times I(\tau_n) \to I(\tau)$$

on corresponding semantical domain(s) of OCL (!) types.

(iv) Interpretation of Set Operations

Basically the same principle as with arithmetic operations...

Let $\tau \in \mathscr{T} \cup T_B \cup T_{\mathscr{C}}$.

- **Set comprehension** ($x_1, \dots, x_n \in I(\tau)$):

$$I(\{\}_n^\tau)(x_1, \dots, x_n) := \{x_1, \dots, x_n\}$$

  for all $n \in \mathbb{N}_0$

- **Empty-ness check** ($x \in I(Set(\tau))$):

$$I(\mathsf{isEmpty}^\tau)(x) := \begin{cases} true & \text{, if } x = \emptyset \\ \bot_{Bool} & \text{, if } x = \bot_{Set(\tau)} \\ false & \text{, otherwise} \end{cases}$$

- **Counting** ($x \in I(Set(\tau))$):

numbes of elements in x

$$I(\mathsf{size}^\tau)(x) := \begin{cases} |x| & \text{, if } x \neq \bot_{Set(\tau)} \text{ and } x \text{ finite} \\ \bot_{Int} & \text{, otherwise} \end{cases}$$

(v) Interpretation of OCL Expressions

OCL Syntax 1/4: Expressions

$expr ::=$

w	$: \tau(w)$	
$\mid expr_1 =_\tau expr_2$	$: \tau \times \tau \to Bool$	
$\mid$ oclIsUndefined$_\tau(expr_1)$	$: \tau \to Bool$	
$\mid \{expr_1, \dots, expr_n\}$	$: \tau \times \dots \times \tau \to Set(\tau)$	
$\mid$ isEmpty$(expr_1)$	$: Set(\tau) \to Bool$	
$\mid$ size$(expr_1)$	$: Set(\tau) \to Int$	
$\mid$ allInstances$_C$	$: Set(\tau_C)$	
$\mid v(expr_1)$	$: \tau_C \to \tau$	where $v : \tau \in atr(C)$, $\tau \in \mathscr{T}$,
$\mid r_1(expr_1)$	$: \tau_C \to \tau_D$	where $r_1 : D_{0,1} \in atr(C), C, D \in \mathscr{C}$,
$\mid r_2(expr_1)$	$: \tau_C \to Set(\tau_D)$	where $r_2 : D_* \in atr(C), C, D \in \mathscr{C}$.

T_B

Where, given $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$,

- $w \in W \supseteq \{self_C : \tau_C \mid C \in \mathscr{C}\}$ is a set of typed logical variables, w has type $\tau(w)$
- τ is any type from $\mathscr{T} \cup T_B \cup T_\mathscr{C} \cup \{Set(\tau_0) \mid \tau_0 \in \mathscr{T} \cup T_B \cup T_\mathscr{C}\}$
  - T_B is a set of (OCL) basic types, in the following we use $T_B = \{Bool, Int, String\}$
  - $T_\mathscr{C} = \{\tau_C \mid C \in \mathscr{C}\}$ is the set of object types,
  - $Set(\tau_0)$ denotes the set-of-τ_0 type for $\tau_0 \in T_B \cup T_\mathscr{C}$ (sufficient because of "flattening" (cf. standard)).

OCL Syntax 2/4: Constants & Arithmetics

For example:

$expr ::= \dots$	
$\mid$ true $\mid$ false	$: Bool$
$\mid expr_1$ {and, or, implies} $expr_2$	$: Bool \times Bool \to Bool$
$\mid$ not $expr_1$	$: Bool \to Bool$
$\mid 0 \mid -1 \mid 1 \mid -2 \mid 2 \mid \dots$	$: Int$
$\mid expr_1 \{+, -, \dots\}\ expr_2$	$: Int \times Int \to Int$
$\mid expr_1 \{<, \leq, \dots\}\ expr_2$	$: Int \times Int \to Bool$
$\mid$ OclUndefined$_\tau$	$: \tau$

Generalised notation: (prefix normal form)

$$expr ::= \omega(expr_1, \dots, expr_n) \qquad : \tau_1 \times \dots \times \tau_n \to \tau$$

with $\omega \in \{+, -, \dots\}$

$1 + 2 \rightsquigarrow \underbrace{+}_{\omega}(\underbrace{1}_{expr_1}, \underbrace{2}_{expr_2})$

OCL Syntax 3/4: Iterate

$$expr ::= \dots \mid expr_1\text{->iterate}(w_1 : T_1\ ;\ w_2 : T_2 = expr_2 \mid expr_3)$$

or, with a little renaming,

$$expr ::= \dots \mid expr_1\text{->iterate}(iter : T_1;\ result : T_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some τ_0),
- $iter \in W$ is called **iterator**, of the type denoted by T_1 (if T_1 is omitted, τ_0 is assumed as type of $iter$)
- $result \in W$ is called **result variable**, gets type τ_2 denoted by T_2, ($\neq iter$)
- $expr_2$ in an expression of type τ_2 giving the **initial value** for $result$, (OclUndefined$_{\tau_2}$, if omitted)
- $expr_3$ is an expression of type τ_2, in particular $iter$ and $result$ may appear in $expr_3$.

OCL Syntax 4/4: Context

Syntax: (Assuming signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.)

$$context ::= \text{context } w_1 : T_1, \dots, w_n : T_n \text{ inv} : expr$$

where $T_i \in \mathscr{C}$ and $w_i : \tau_{T_i} \in W$ for all $1 \leq i \leq n$, $n \geq 0$.

Semantics:

context $w_1 : C_1, \dots, w_n : C_n$ inv : $expr$

is (just) an **abbreviation** for

```
allInstances_C1 -> forAll(w1 : τ_C1 |
  ...
    allInstances_Cn -> forAll(wn : τ_Cn |
      expr
    )
  ...
)
```

Valuations of Logical Variables

- **Recall**: we have typed logical variables $(w \in)\ W$, $\tau(w)$ is the type of w.
- By β, we denote a valuation of the logical variables, i.e. for each $w \in W$,

$$\beta(w) \in I(\tau(w)).$$

- $self_{VM} \in W$
- $self_{VM} : \tau_{VM}$ is an OCL expression
- $I[\![self_{VM}]\!](\sigma, \beta) := \beta(self_{VM})$
- $\beta_1 = \{ self_{VM} \mapsto 1_{VM} \}$

  $\leadsto I[\![self_{VM}]\!](\sigma, \beta_1) = \beta_1(self_{VM}) = 1_{VM}$

- $\beta : W \longrightarrow I(T_B \cup T_C \cup \mathcal{J})$

(v) Interpretation of OCL Expressions

$$expr ::= w \mid \omega(expr_1, \dots, expr_n) \mid \mathsf{allInstances}_C \mid v(expr_1) \mid r_1(expr_1)$$
$$\mid r_2(expr_1) \mid expr_1\text{->}\mathsf{iterate}(v_1 : \tau_1\,;\, v_2 : \tau_2 = expr_2 \mid expr_3)$$

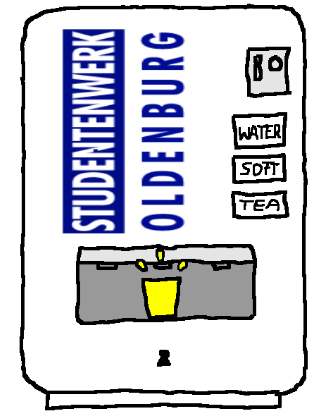
- $I[\![\overset{W}{w}]\!](\sigma, \beta) := \beta(w)$

- $I[\![\omega(expr_1, \dots, expr_n)]\!](\sigma, \beta) := I(\omega)\big(I[\![expr_1]\!](\sigma,\beta), \dots, I[\![expr_n]\!](\sigma,\beta)\big)$

- $I[\![\mathsf{allInstances}_C]\!](\sigma, \beta) := \underbrace{\mathrm{dom}(\sigma)}_{\text{all alive objects in } \sigma} \cap \underbrace{\mathscr{D}(C)}_{\text{objects of class } C}$

  Note: in the OCL standard, $\mathrm{dom}(\sigma)$ is assumed to be **finite**.
  Again: doesn't scare us.

Example



$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, wis : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen, dd\}, DD \mapsto \{wis\})$$

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{wis \mapsto 13\},$$
$$3_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textsf{true}\}, \quad 5_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textsf{false}\}\}$$

- $I[\![w]\!](\sigma, \beta) := \beta(w)$
- $I[\![\textsf{allInstances}_C]\!](\sigma, \beta) := \text{dom}(\sigma) \cap \mathscr{D}(C)$
- $I[\![\omega(expr_1, \ldots, expr_n)]\!](\sigma, \beta) := I(\omega)(I[\![expr_1]\!](\sigma, \beta), \ldots I[\![expr_n]\!](\sigma, \beta))$

- $I[\![\textsf{allInstances}_{CP}]\!](\sigma_1, \beta) = dom(\sigma_1) \cap \mathcal{D}(CP) = \{7_{VM}, 1_{DD}, 3_{CP}, 5_{CP}\} \cap \mathcal{D}(CP)$
  $= \{3_{CP}, 5_{CP}\}$

- $I[\![\textsf{allInstances}_{CP}\ \textsf{->size}]\!](\sigma_1, \beta) = I[\![size(allInstances_{CP})]\!](\sigma,\beta)$
  $= I(size)(I[\![allInstances_{CP}]\!](\sigma_1,\beta)) = I(size)(\{3_{CP}, 5_{CP}\}) = 2$

- $\beta_1 := \{\overset{self \mapsto}{} 3_{CP}\}, \quad I[\![self]\!](\sigma_1, \beta_1) = \beta_1(self) = 3_{CP}$

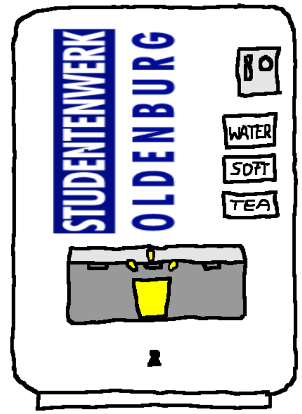
(v) Interpretation of OCL Expressions

$$expr ::= w \mid \omega(expr_1, \ldots, expr_n) \mid \mathsf{allInstances}_C \mid v(expr_1) \mid r_1(expr_1)$$
$$\mid r_2(expr_1) \mid expr_1 \texttt{->iterate}(v_1 : \tau_1 \; ; v_2 : \tau_2 = expr_2 \mid expr_3)$$

Assume $expr_1 : \tau_C$ for some $C \in \mathscr{C}$. Set $u_1 := I[\![expr_1]\!](\sigma, \beta) \in \cancel{\mathscr{D}(\tau_C)}$ $I(\tau_C)$

- $I[\![v(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(v) & \text{, if } u_1 \in dom(\sigma) \\ \perp & \text{, otherwise} \end{cases}$

Example



$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, wis : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen, dd\}, DD \mapsto \{wis\})$$

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{wis \mapsto 13\},$$
$$3_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{true}\}, \quad 5_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{false}\}\}$$

Assume $expr_1 : \tau_C$ for some $C \in \mathscr{C}$. Set $u_1 := I[\![expr_1]\!](\sigma, \beta) \in \mathscr{D}(\tau_C)$.

- $I[\![v(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(v) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases}$

- $\beta_1 := \{\text{self} \mapsto 3_{CP}\}$, $\quad I[\![wen(self)]\!](\sigma_1, \beta_1) = \sigma_1(u_1)(wen) = \sigma_1(3_{CP})(wen) = true$

  $u_1 = I[\![self]\!](\sigma_1, \beta_1) = 3_{CP}$

(v) Interpretation of OCL Expressions

$$\begin{aligned} expr ::= {} & w \mid \omega(expr_1, \ldots, expr_n) \mid \mathsf{allInstances}_C \mid v(expr_1) \mid r_1(expr_1) \\ & \mid r_2(expr_1) \mid expr_1\text{->iterate}(v_1 : \tau_1 ; v_2 : \tau_2 = expr_2 \mid expr_3) \end{aligned}$$

Assume $expr_1 : \tau_C$ for some $C \in \mathscr{C}$. Set $u_1 := I[\![expr_1]\!](\sigma, \beta) \in \cancel{\mathscr{D}(\tau_C)}$ $I(\tau_C)$.

- $I[\![v(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(v) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases}$

- $I[\![r_1(expr_1)]\!](\sigma, \beta) := \begin{cases} u & \text{, if } u_1 \in \mathrm{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\ \bot & \text{, otherwise} \end{cases}$

  $r_1 : C_{0,1}$

- $I[\![r_2(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases}$

  $r_2 : C_*$

Recall: σ evaluates r_2 of type C_* to a set.

Iterate: Intuitive Semantics

$$expr ::= expr_1\texttt{->iterate}(iter : T_1;\ result : T_2 = expr_2 \mid expr_3)$$

$$
\begin{array}{l}
Set(\tau_0)\ hlp := expr_1;\\
\tau_1\ iter;\\
\tau_2\ result := expr_2;\\
while\ (!hlp.empty())\ do\\
\quad iter := hlp.pop();\\
\quad result := expr_3;\\
od;\\
return\ result;
\end{array}
$$

pick and remove one element

may comprise iter and result

context CP inv : wen

iter

$\Downarrow$

all Inst$_{CP}$ $\rightarrow$ forAll(self | wen(self))

Iterate: Intuitive Semantics

$$expr ::= expr_1\texttt{->iterate}(iter : T_1;$$
$$result : T_2 = expr_2 \mid expr_3)$$

$$
\begin{array}{l}
Set(\tau_0)\ hlp := expr_1; \\
\tau_1\ iter; \\
\tau_2\ result := expr_2; \\
while\ (!hlp.empty())\ do \\
\qquad iter := hlp.pop(); \\
\qquad result := expr_3; \\
od; \\
return\ result;
\end{array}
$$

Recall: In our (simplified) setting, we always have $expr_1 : Set(\tau_0)$ and $\tau_1 = \tau_0$.

In the type hierarchy of full OCL with inheritance and `oclAny`, τ_0 and τ_1 may be different and still type consistent.

(v) Interpretation of OCL Expressions

$$\begin{aligned} expr ::= w \mid \omega(expr_1, \dots, expr_n) \mid \mathsf{allInstances}_C \mid v(expr_1) \mid r_1(expr_1) \\ \mid r_2(expr_1) \mid expr_1\text{->}\mathsf{iterate}(v_1 : \tau_1 \,;\, v_2 : \tau_2 = expr_2 \mid expr_3) \end{aligned}$$

- $I[\![expr_1\text{->}\mathsf{iterate}(v_1 : \tau_1 \,;\, v_2 : \tau_2 = expr_2 \mid expr_3)]\!](\sigma, \beta)$

$$:= \begin{cases} I[\![expr_2]\!](\sigma, \beta) & \text{, if } I[\![expr_1]\!](\sigma, \beta) = \emptyset \\ iterate(hlp, v_1, v_2, expr_3, \sigma, \beta') & \text{, otherwise} \end{cases}$$

where $\beta' = \beta[hlp \mapsto I[\![expr_1]\!](\sigma, \beta), \quad v_2 \mapsto I[\![expr_2]\!](\sigma, \beta)]$ and

modify β' at v_1, $\begin{cases} x & \text{, if } v_1 \text{ given} \\ \beta'(w), & \text{otherwise} \end{cases}$

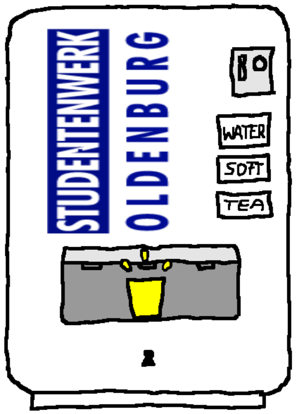
- $iterate(hlp, v_1, v_2, expr_3, \sigma, \beta')$

$$:= \begin{cases} I[\![expr_3]\!](\sigma, \beta'[v_1 \mapsto x]) & \text{, if } \beta'(hlp) = \{x\} \\ I[\![expr_3]\!](\sigma, \beta'') & \text{, if } \beta'(hlp) = X \mathbin{\dot\cup} \{x\} \text{ and } X \neq \emptyset \end{cases}$$

where $\beta'' = \beta'[v_1 \mapsto x, \quad v_2 \mapsto iterate(hlp, v_1, v_2, expr_3, \sigma, \beta'[hlp \mapsto X])]$

Quiz: Is (our) I a function?

Example



$$\begin{aligned}\mathscr{S} = (&\{Bool, Nat\}, \{VM, CP, DD\},\\ &\{cp : CP_*, dd : DD_{0,1}, wen : Bool, wis : Nat\},\\ &\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen, dd\}, DD \mapsto \{wis\})\end{aligned}$$

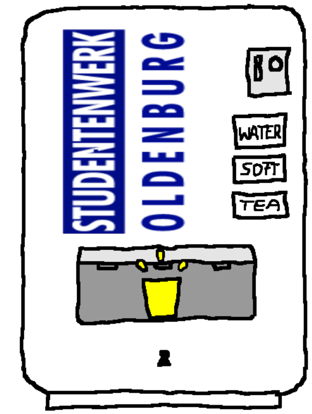
$$\begin{aligned}\sigma_1 = \{&7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{wis \mapsto 13\},\\ &3_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{true}\}, \quad 5_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{false}\}\}\end{aligned}$$

$$\text{context} \;\; CP \;\; \text{inv}: \;\; wen \;\; \text{implies} \;\; dd\,.\,wis > 0$$

} unabbr.

allInstances$_{CP}$ -> forAll (self | wen implies dd.wis > 0)

Example



$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, wis : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen, dd\}, DD \mapsto \{wis\})$$

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{wis \mapsto 13\},$$
$$3_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{true}\}, \quad 5_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{false}\}\}$$

$$\mathsf{context}\ \ CP\ \ \mathsf{inv:}\ \ wen\ \ \mathsf{implies}\ \ dd\,.\,wis > 0$$

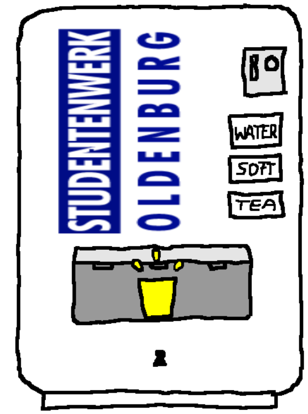
$$\mathsf{allInstances}_{CP}\ \texttt{->}\ \mathsf{forAll}\,(self \mid self\,.\,wen\ \ \mathsf{implies}\ \ self\,.\,dd\,.\,wis > 0)$$

$$\rightsquigarrow$$

$$\text{allInstances}_{CP} \rightarrow \text{iterate}\,(self\ ;\ r : Bool = true \mid r\ \text{and}\ \ldots\,)$$

Example

$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, wis : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen, dd\}, DD \mapsto \{wis\})$$



$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{wis \mapsto 13\},$$
$$3_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{true}\}, \quad 5_{CP} \mapsto \{dd \mapsto \{1_{DD}\}, wen \mapsto \textit{false}\}\}$$

$F :=$ context CP inv : wen implies $dd\,.\,wis > 0$

$$\mathrm{allInstances}_{CP} \text{ -> iterate } (self; r : Bool = \textit{true} \mid \underbrace{\mathrm{and}(r, \overbrace{\mathrm{implies}(wen(self), >(wis(dd(self)), 0))}^{expr_1})}_{=:\,expr})$$

$I⟦\mathrm{allInstances}_{CP}⟧(\sigma_1, \emptyset) = \{3_{CP}, 5_{CP}\}$

$I⟦expr⟧(\sigma_1, \overbrace{\{self \mapsto 3_{CP}, r \mapsto true\}}^{=:\,\beta_1}) = I(and)(I⟦r⟧(\sigma_1, \beta_1), I⟦expr_1⟧(\sigma_1, \beta_1)) \overset{(*)}{=} I(and)(true, true) = true$

$I⟦expr_1⟧(\sigma_1, \beta_1) = I(implies)(\underbrace{I⟦wen(self)⟧(\sigma_1, \beta_1)}_{true}, \underbrace{I(>)(\underbrace{I⟦wis(dd(self))⟧(\sigma_1, \beta_1)}_{13}, 0)}_{true})$ (**)

$\vdots$

$I⟦dd(self)⟧(\sigma_1, \beta_1) = 1_{DD}$

$I⟦wis(dd(self))⟧(\sigma_1, \beta_1) = 13$

$\vdots\; = I(implies)(true, true) = true$ (*)

$I⟦expr⟧(\sigma_1, \{self \mapsto 5_{CP}, r \mapsto true\}) \overset{(**)}{=} true$

$\underline{I⟦F⟧(\sigma_1, \beta_1) = true}$

Tell Them What You've Told Them...

- **Given**
  - an OCL expression $expr$,
  - and a system state σ,
  - and a valuation β of the logical variables

- we can **compute** the value

$$I[\![expr]\!](\sigma, \beta) \in \{\textit{\textbf{true}}, \textit{\textbf{false}}, \bot_{Bool}\}$$

  of $expr$ in σ under β

- using the **interpretation function**

$$I[\![\,\cdot\,]\!](\,\cdot\,,\,\cdot\,) : OCLExpressions(\mathscr{S}) \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \to I(Bool).$$

User's Guide

- **App**

  The

  It is

Example:

Task: Given a square with side length $a = 19.1$. What is the length of the longest straight line fully inside the square?

Submission A:

27

Submission B:

The length of the longest straight line fully inside the square with side length $a = 19.1$ is 27.01 (rounded).

The longest straight line inside the square is the diagonal. By Pythagoras, its length is $\sqrt{a^2 + a^2}$. Inserting $a = 19.1$ yields 27.01 (rounded).

- **Inte**

  Abs

- **Exercise submissions:**

  Each task is a **tiny little scientific work**:

  (i) Briefly rephrase the task in your own words.

  (ii) State your claimed solution.

  (iii) Convince your reader that your proposal is a solution (proofs are very convincing).

User's Guide

- **App**

  The

  It is

Example:

Task: Given a square with side length $a = 19.1$. What is the length of the longest straight line fully inside the square?

Submission A:

Submission B:

The length of the longest straight line fully inside the square with side l[illegible]gth $a = 19.1$ is 27.01 (rounded).

The longest s[illegible]ht[illegible]e inside the square is the diagona[illegible]ythagoras, its length is $\sqrt{a^2 + a^2}$. Ins[illegible]ng $a = 19.1$ yields 27.01 (rounded).

- **Inte**

  Abs

- **Exercise submissions:**

  Each task is a **tiny little scientific work**:

  (i) Briefly rephrase the task in your own words.

  (ii) State your claimed solution.

  (iii) Convince your reader that your proposal is a solution (proofs are very convincing).

Formalia: Exercises and Tutorials

- You should work in groups of **approx. 3**, clearly give **names** on submission.
- Please submit via ILIAS (cf. homepage); **paper submissions** are **tolerated**.
- **Schedule:**

Week N,	Thursday,	8–10	**Lecture A1**	(exercise sheet A **online**)
Week $N+1$,	Tuesday	8–10	**Lecture A2**	
	Thursday	8–10	**Lecture A3**	
Week $N+2$,	Monday,	12:00		(exercises A **early submission**)
	Tuesday,	8:00		(exercises A **late submission**)
		8–10	**Tutorial A**	
	Thursday	8–10	**Lecture B1**	(exercise sheet B **online**)
	...			

- **Rating system:** "most complicated rating system **ever**"
  - **Admission points** (good-will rating, upper bound)
    ("reasonable proposal given student's knowledge **before** tutorial")
  - **Exam-like points** (evil rating, lower bound)
    ("reasonable proposal given student's knowledge **after** tutorial")

  10% bonus for **early** submission.

- **Tutorial:** Plenary, **not recorded**.
  - Together develop **one good solution** based on selection of early submissions (anonymous) – there is no "Musterlösung" for modelling tasks.

- E.g.
  - give a syst. state as pos. example
  - system state
    $\sigma_1 = \{ \dots \}$
    satisfies the req. because ...
- 18 submissions
  - ~10 singleton groups

References

References

OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.

OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.