Software Design, Modelling and Analysis in UML

Lecture 03: Object Constraint Language (OCL)

2011-11-02

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

$v \in \mathrm{dom}(\sigma)$ are called alive objects

$: \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow \mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}))$

Last Lecture:

- Basic Object System Signature $\mathscr{S}$ and Structure $\mathscr{D}$
- System State $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$

(Smells like they're related to class/object diagrams, officially we don't know yet. . .)

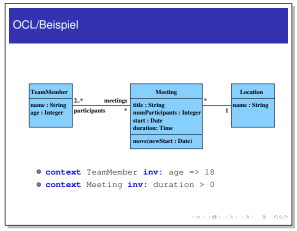
This Lecture:

- **Educational Objectives:** Capabilities for these tasks/questions:
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this system state?
  - Can you think of a system state satisfying this constraint?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are $\mathscr{D}(C)$ and τ_C related?
- **Content:**
  - OCL Syntax, OCL Semantics over system states

What is OCL? And What is It Good For?

What is OCL? How Does it Look Like?

- **OCL**: Object Constraint Logic.



OCL/Beispiel

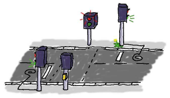
- `context TeamMember inv: age => 18`
- `context Meeting inv: duration > 0`

((C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/)

What's It Good For?

- **Most prominent:**
  write down **requirements** supposed to be satisfied by all system states.

  Often targeting all alive objects of a certain class.

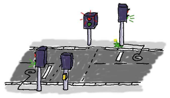


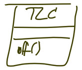
context TLC inv : not (red and green)

What's It Good For?

- **Most prominent:**
  write down **requirements** supposed to be satisfied by all system states.

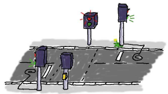
  Often targeting all alive objects of a certain class.
- **Not unknown:**
  write down **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over two system states.





context off
pre: (true)
post: (not red and not yellow
and not green)

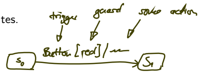
What's It Good For?



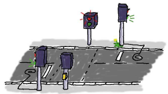
- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

  Often targeting all alive objects of a certain class.

- **Not unknown**:
  write down **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over **two** system states.
- **Common with State Machines**:
  guards in transitions.



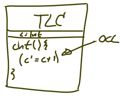
What's It Good For?



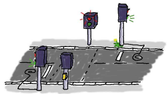
- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

  Often targeting all alive objects of a certain class.

- **Not unknown**:
  write down **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over **two** system states.
- **Common with State Machines**:
  guards in transitions.
- **Lesser known**:
  provide **operation bodies**.



What's It Good For?



- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

  Often targeting all alive objects of a certain class.

- **Not unknown**:
  write down **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over **two** system states.
- **Common with State Machines**:
  guards in transitions.
- **Lesser known**:
  provide **operation bodies**.

- **Metamodeling**: the UML standard is a MOF-Model of UML.
  OCL expressions define well-formedness of UML models (cf. Lecture ∼ 21).

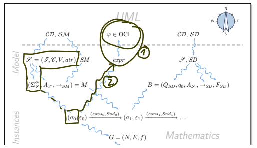
Plan.

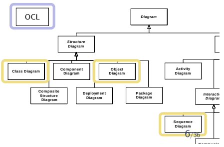
- **Today**:
  - ① The set $OCLExpressions(\mathscr{S})$ of OCL expressions over $\mathscr{S}$.
  - ② Given an OCL expression $expr$, a system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$, and a valuation of logical variables β, define

$$I[\![expr]\!](\sigma, \beta) \in \{\mathit{true}, \mathit{false}, \bot\}.$$

- **Later**: use I to define $\models\ \subseteq \Sigma_{\mathscr{S}}^{\mathscr{D}} \times OCLExpressions(\mathscr{S})$.

$\sigma, \beta \models expr$





(Core) OCL Syntax [OMG, 2006]

OCL Syntax 1/4: Expressions

some expression

type of the expression

types of subexpr.

type of expr.

$$
\begin{array}{lll}
expr ::= & & \\
\quad w & & : \tau(w) \\
\quad \mid expr_1 =_\tau expr_2 & & : \tau \times \tau \rightarrow Bool \\
\quad \mid \mathsf{oclIsUndefined}_\tau(expr_1) & & : \tau \rightarrow Bool \\
\quad \mid \{expr_1, \ldots, expr_n\} & & : \tau \times \cdots \times \tau \rightarrow Set(\tau) \\
\quad \mid \mathsf{isEmpty}(expr_1) & & : Set(\tau) \rightarrow Bool \\
\quad \mid \mathsf{size}(expr_1) & & : Set(\tau) \rightarrow Int \\
\quad \mid \mathsf{allInstances}_C & & : Set(\tau_C) \\
\quad \mid v(expr_1) & & : \tau_C \rightarrow \tau(v) \\
\quad \mid r_1(expr_1) & & : \tau_C \rightarrow \tau_D \\
\quad \mid r_2(expr_1) & & : \tau_C \rightarrow Set(\tau_D)
\end{array}
$$

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

- $W \supseteq \{self_C \mid C \in \mathscr{C}\}$ is a set of typed logical variables, w has type $\tau(w)$
- τ is any type from $\mathscr{T} \cup T_B \cup T_{\mathscr{C}}$ $\cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup T_{\mathscr{C}}\}$
  - T_B is a set of 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))
- $v : \tau(v) \in atr(C)$, $\tau(v) \in \mathscr{T}$,
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,
- $C, D \in \mathscr{C}$.

OCL Syntax: Notational Conventions for Expressions

- Each expression

$$\omega(\underbrace{expr_1, expr_2, \dots, expr_n) : \tau_1}) \times \dots \times \tau_n \to \tau$$

may alternatively be written ("abbreviated as")

- $expr_1$. $\omega(expr_2, \dots, expr_n)$ if τ_1 is an **object type**, i.e. if $\tau_1 \in T_{\mathscr{C}}$.
- $expr_1$ -> $\omega(expr_2, \dots, expr_n)$ if τ_1 is a **collection type** (here: only sets), i.e. if $\tau_1 = Set(\tau_0)$ for some $\tau_0 \in T_B \cup T_{\mathscr{C}}$.

expr₁ ω log. variables attributes

- **Examples:** $(self_C : \tau_C \in W; \quad v, w : Int \in V; \quad r_1 : D_{0,1}, r_2 : D_* \in V)$
  - $self\ .\ v$ // assume $v \in atr(C)$
    - $\omega(self, v)$ · ~~self.ω(v)~~ · ~~self(v)~~ · $\boxed{v(self)}$
  - $self\ .\ r_1\ .\ w$ / assume $w \in atr(D)$, $r_1 \in atr(C)$
    - $\underbrace{w(\underbrace{r_1(self)}_{: \tau_D})}_{: Int}$
  - $self\ .\ r_2$ -> isEmpty
    - $isEmpty(r_2(self))$

∘ not OCL: self -> v

OCL Syntax 2/4: Constants, Arithmetical Operators

For example:

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

Generalised notation:

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

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

OCL Syntax 3/4: Iterate

$$expr ::= \dots \mid expr_1\text{->iterate}(w_1 : \tau_1 ;\ w_2 : \tau_2 = expr_2 \mid expr_3)$$

or, with a little renaming,

$$expr ::= \dots \mid expr_1\text{->iterate}(iter : \tau_1;\ result : \tau_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some τ_0),
- $iter \in W$ is called **iterator**, gets type τ_1 (if τ_1 is omitted, τ_0 is assumed as type of $iter$)
- $result \in W$ is called **result variable**, gets type τ_2,
- $expr_2$ in an expression of type τ_2 giving the **initial value** for $result$, ('OclUndefined' if omitted)
- $expr_3$ is an expression of type τ_2 in which in particular $iter$ and $result$ may appear.

Iterate: Intuitive Semantics (Formally: later)

$$expr ::= expr_1\text{->iterate}(iter : \tau_1;\ result : \tau_2 = expr_2 \mid expr_3)$$

```
Set(τ0) hlp = ⟨expr1⟩;
τ2 result = ⟨expr2⟩;
while (!hlp.empty()) do
    τ1 iter = hlp.pop();
    result = ⟨expr3⟩;
od
```

not OCL, but some pseudocode

remove element from hlp

Note: In our (simplified) setting, we always have $expr_1 : Set(\tau_1)$ and $\tau_0 = \tau_1$.
In the type hierarchy of full OCL with inheritance and `oclAny`, they may be different and still type consistent.

Abbreviations on Top of Iterate

$$expr ::= expr_1\text{->iterate}(w_1 : \tau_1;\ w_2 : \tau_2 = expr_2 \mid expr_3)$$

- $expr_1$->forAll($w : \tau_1 \mid expr_3$) is an abbreviation for

  $expr_1$->iterate($w : \tau_1;\ w_1 : Bool = true \mid w_1 \wedge expr_3$).

  (To ensure confusion, we may again omit all kinds of things, cf. [OMG, 2006]).
- Similar: $expr_1$->Exists($w : \tau_1 \mid expr_3$)

OCL Syntax 4/4: Context

$$context ::= \text{context } w_1 : \tau_1, \dots, w_n : \tau_n \text{ inv : } expr$$

where $w \in W$ and $\tau_i \in T_{\mathscr{C}}$, $1 \leq i \leq n$, $n \geq 0$.

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

is an **abbreviation** for

```
allInstances_C1 -> forAll(w1 : C1 |
  ...
    allInstances_Cn -> forAll(wn : Cn |
      expr
    )
  ...
)
```

Context: More Notational Conventions

- For

  context $self : \tau_C$ inv : $expr$

  we may alternatively write ("abbreviate as")

  context τ_C inv : $expr$

- **Within** the latter abbreviation, we may omit the "$self$" in $expr$, i.e. for

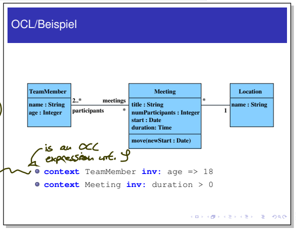
  $self.v$ and $self.r$

  we may alternatively write ("abbreviate as")

  v and r

Examples (from lecture "Softwaretechnik 2008")

𝒮 = ({String, Integer, ...},
{TeamMember, Meeting,
Location},
{age : Integer, ... },
{TeamMembers ↦ {age, name,
meetings}, ...})



OCL/Beispiel

is an OCL
expression wrt. 𝒮

un-abbreviation step

- **context** TeamMember **inv:** age => 18
- **context** Meeting **inv:** duration > 0

allInstances$_{TeamMember}$
-> forAll (self : τ_{TM} | age => 18)

allInstances$_{TM}$
-> iterate(self : τ_{TM}, res : Bool = true | age >= 18 and res)

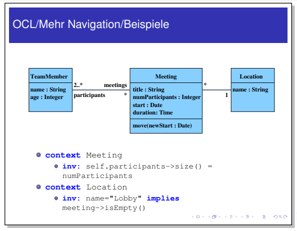
——"—— self . age ≥ 18 and res)

——"—— and (self . age ≥ 18 , res)

allInstances$_{TM}$ -> iterate (self$_{TM}$: τ_{TM} , res : Bool = true | and (≥ (age(self$_{TM}$), 18), res))

((C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/)

Examples (from lecture "Softwaretechnik 2008")

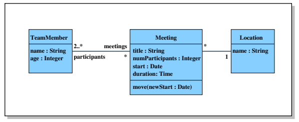


OCL/Mehr Navigation/Beispiele

- **context** Meeting
  - **inv:** self.participants->size() = numParticipants
- **context** Location
  - **inv:** name="Lobby" **implies** meeting->isEmpty()

((C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/)

Example (from lecture "Softwaretechnik 2008")



- context $Meeting$ inv :

$$\Big(\big(participants \text{ -> iterate}(i : TeamMember; n : Int = 0 \mid n + i\,.\,age)\big) \Big/ \big(participants \text{ -> size}()\big)\Big) > 25$$

- count team members in meeting
- sum up age of participants of a meeting
- "for each meeting, the average age of participants shall be greater 25"

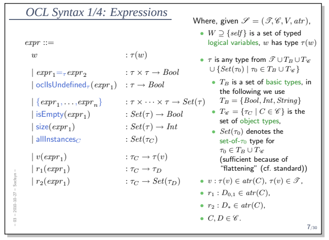
"Not Interesting"

Among others:

- Enumeration types
- Type hierarchy
- Complete list of arithmetical operators
- The two other collection types Bag and Sequence
- Casting
- Runtime type information
- Pre/post conditions
  (maybe later, when we officially know what an operation is)
- ...

OCL Semantics [OMG, 2006]

The Task



OCL Syntax 1/4: Expressions

$expr ::=$
$w \quad : \tau(w)$
$\mid expr_1 =_\tau expr_2 \quad : \tau \times \tau \to Bool$
$\mid$ oclIsUndefined$_\tau(expr_1) \quad : \tau \to Bool$
$\mid \{expr_1, \dots, expr_n\} \quad : \tau \times \dots \times \tau \to Set(\tau)$
$\mid$ isEmpty$(expr_1) \quad : Set(\tau) \to Bool$
$\mid$ size$(expr_1) \quad : Set(\tau) \to Int$
$\mid$ allInstances$_C \quad : Set(\tau_C)$
$\mid v(expr_1) \quad : \tau_C \to \tau(v)$
$\mid r_1(expr_1) \quad : \tau_C \to \tau_D$
$\mid r_2(expr_1) \quad : \tau_C \to Set(\tau_D)$

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

- $W \supseteq \{self\}$ is a set of typed logical variables, w has type $\tau(w)$
- τ is any type from $\mathscr{T} \cup T_B \cup T_{\mathscr{C}} \cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup T_{\mathscr{C}}\}$
  - T_B is a set of 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))
- $v : \tau(v) \in atr(C)$, $\tau(v) \in \mathscr{T}$,
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,
- $C, D \in \mathscr{C}$.

– 03 – 2010-10-27 – Soclsyn –

7/30

- Given an OCL expression $expr$, a system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$, and a valuation of logical variables β, define

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

such that

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

Basically business as usual...

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

$$T_B \subset \mathrm{dom}(I)$$

(ii) Equip each **object type** τ_C with a reasonable **domain**,
i.e. define function I **on**

$$\{\tau_C\} \subset \mathrm{dom}(I)$$

(most reasonable: $\mathscr{D}(C)$
as determined by structure $\mathscr{D}$ of $\mathscr{S}$).

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

$$\{Set(\tau_0) \mid \tau_0 \in T_B \cup T_{\mathscr{C}}\} \subset \mathrm{dom}(I)$$

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

$$\{+, -, \leq, \dots\} \subset \mathrm{dom}(I),$$

e.g., $I(+) \in I(Int) \times I(Int) \to I(Int)$

(v) **Set operations** similar:
Define function I **on**

$$\{\text{isEmpty}, \dots\} \subset \mathrm{dom}(I)$$

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

$$\cancel{I:}\ Expr \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \quad \subset \mathrm{dom}(I) \quad \cancel{\to I(Bool)}$$

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

(i) Domains of Basic Types

Recall:

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

We set:

"undefined"

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

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

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

- Now we need a structure $\mathscr{D}$ of our signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.
- **Recall:** $\mathscr{D}$ assigns an (infinite) domain $\mathscr{D}(C)$ to each class $C \in \mathscr{C}$.

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

disjoint union

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

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

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

Note: in the OCL standard, only **finite** subsets of $I(\tau)$.
But infinity doesn't scare **us**, so we simply allow it.

Basically business as usual...

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

$$T_B \subset \mathrm{dom}(I)$$

(ii) Equip each **object type** τ_C with a reasonable **domain**,
i.e. define function I **on**

$$\tau_C \subset \mathrm{dom}(I)$$

(most reasonable: $\mathscr{D}(C)$
as determined by structure $\mathscr{D}$ of $\mathscr{S}$).

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

$$\{Set(\tau_0) \mid \tau_0 \in T_B \cup T_{\mathscr{C}}\} \subset \mathrm{dom}(I)$$

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

$$\{+, -, \leq, \dots\} \subset \mathrm{dom}(I),$$

e.g., $I(+) \in I(Int) \times I(Int) \to I(Int)$

(v) **Set operations** similar:
Define function I **on**

$$\{\text{isEmpty}, \dots\} \subset \mathrm{dom}(I)$$

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

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

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

(iv) Interpretation of Arithmetic Operations

- **Literals** map to fixed values:

$$I(\text{true}) := true, \quad I(\text{false}) := false, \qquad I(0) := 0, \quad I(1) := 1, \dots$$
$$I(\text{OclUndefined}_\tau) := \bot_\tau$$

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

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

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

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

Note: There is a **common principle**.
Namely, the **interpretation** of an operation $\omega : \tau_1 \times \dots \tau_n \to \tau$
is a function $I(\omega) : I(\tau_1) \times \dots \times I(\tau_n) \to I(\tau)$ on corresponding semantical domain(s).

(iv) Interpretation of OclIsUndefined

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

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

(v) Interpretation of Set Operations

Basically the same principle as with arithmetic operations...

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

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

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

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

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

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

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

$$I(\text{size}^\tau)(x) := |x| \text{ if } x \neq \bot_{Set(\tau)} \text{ and } \bot_{Int} \text{ otherwise}$$

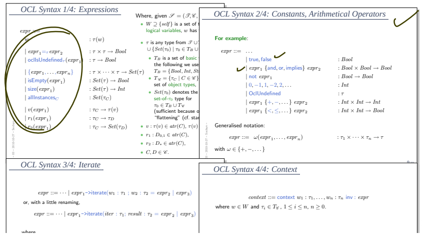
(vi) Putting It All Together: Semantics of Expressions

- **Task**: Given OCL expression $expr$, system state $\sigma \in \Sigma^{\mathscr{D}}_{\mathscr{S}}$, and valuation β, define

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

such that

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



OCL Syntax 1/4: Expressions

$expr ::=$
w : $\tau(w)$
$\mid expr_1 =_\tau expr_2$: $\tau \times \tau \to Bool$
$\mid \text{oclIsUndefined}_\tau(expr_1)$: $\tau \to Bool$
$\mid \{expr_1, \dots, expr_n\}$: $\tau \times \dots \times \tau \to Set(\tau)$
$\mid \text{isEmpty}(expr_1)$: $Set(\tau) \to Bool$
$\mid \text{size}(expr_1)$: $Set(\tau) \to Int$
$\mid \text{allInstances}_C$: $Set(\tau_C)$
$\mid v(expr_1)$: $\tau_C \to \tau(v)$
$\mid r_1(expr_1)$: $\tau_C \to \tau_D$
$\mid r_2(expr_1)$: $\tau_C \to Set(\tau_D)$

Where, given $\mathscr{S} = (\mathscr{T}, \mathscr{C},$
- $W \supseteq \{self\}$ is a set of logical variables, w has
- τ is any type from $\mathscr{T} \cup$ $\cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup$
  - T_B is a set of basic the following we use $T_B = \{Bool, Int, St$
  - $T_{\mathscr{C}} = \{\tau_C \mid C \in \mathscr{C}\}$ 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 “flattening” (cf. sta
- $v : \tau(v) \in atr(C)$, $\tau(v)$
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,
- $C, D \in \mathscr{C}$.

OCL Syntax 2/4: Constants, Arithmetical Operators

For example:

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

Generalised notation:

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

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

OCL Syntax 3/4: Iterate

$expr ::= \dots \mid expr_1\text{->iterate}(w_1 : \tau_1 \,;\, w_2 : \tau_2 = expr_2 \mid expr_3)$

or, with a little renaming,

$expr ::= \dots \mid expr_1\text{->iterate}(iter : \tau_1 ;\, result : \tau_2 = expr_2 \mid expr_3)$

where

OCL Syntax 4/4: Context

$context ::= \text{context } w_1 : \tau_1, \dots, w_n : \tau_n \text{ inv} : expr$

where $w \in W$ and $\tau_i \in T_{\mathscr{C}}$, $1 \leq i \leq n$, $n \geq 0$.

Preliminaries: 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 $w \in W$,

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

(vi) Putting It All Together...

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

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

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

(vi) Putting It All Together...

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

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

- $I[\![v(expr_1)]\!](\sigma, \beta) := \begin{cases} (\sigma(u_1))(v) & \text{, if } u_1 \in dom(\sigma) \\ \bot & \text{, otherwise} \end{cases}$
- $I[\![r_1(expr_1)]\!](\sigma, \beta) := \begin{cases} u & \text{, if } u_1 \in \text{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\ \bot & \text{, otherwise} \end{cases}$
- $I[\![r_2(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2) & \text{, if } u_1 \in \text{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases}$

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

(vi) Putting It All Together...

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

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

$$:= \begin{cases} I[\![expr_2]\!](\sigma, \beta) & \text{, if } I[\![expr_1]\!](\sigma, \beta) = \emptyset \\ iterate(\tilde{v}_1, v_1, v_2, expr_3, \sigma, \beta'') & \text{, otherwise} \end{cases}$$

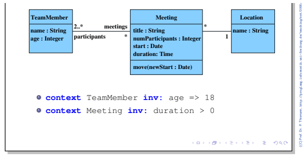
where $\beta'' = \beta[\tilde{v}_1 \mapsto I[\![expr_1]\!](\sigma, \beta) \setminus \{x\}, v_1 \mapsto x, v_2 \mapsto I[\![expr_2]\!](\sigma, \beta)]$ and

$$iterate(\tilde{v}_1, v_1, v_2, expr_3, \sigma, \beta) = \begin{cases} \beta(v_2) & \text{, if } \beta(\tilde{v}_1) = \emptyset \\ iterate(\tilde{v}_1, v_1, v_2, expr_3, \sigma, \beta') & \text{, otherwise} \end{cases}$$

where $\beta' = \beta[\tilde{v}_1 \mapsto \beta(v_1) \setminus \{x\}, v_1 \mapsto x, v_2 \mapsto I[\![expr_3]\!](\sigma, \beta)]$, $x \in \beta(\tilde{v}_1)$

Quiz: Is (our) I a function?
Not if the outcome depends on order of choice of the x!

Example



- `context TeamMember inv: age => 18`
- `context Meeting inv: duration > 0`

((C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/

Outlook on Type Theory

Well-Typedness...

- **Note**: in the definition of I, we have silently assumed that expressions are **well-typed**.
- Which is **somewhat clear** from the **typed** syntax. For instance,

  $$\text{context } C \text{ inv} : r \text{ -> size}() + 1$$

  is "**obviously**" well-typed, while

  $$\text{context } C \text{ inv} : r + 1$$

  is not (if $r : D_*$).

- **In Lecture 06**:

  A precise definition of well-typed expressions using **basic type theory**.

  Why so late? Consider **visibility** of attributes **in one go**.

References

References

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

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

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

[Warmer and Kleppe, 1999] Warmer, J. and Kleppe, A. (1999). *The Object Constraint Language*. Addison-Wesley.