Contents & Goals

Last Lecture:
- Basic Object System Signature $\mathcal{S}$ and Structure $\mathcal{D}$
- System State $\sigma \in \Sigma_\mathcal{D}$

(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 $\mathcal{D}(C)$ and $\tau_C$ 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
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.

```
TLC
red : Bool
green : Bool
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.
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.

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

- **Today:**
  - The set $\text{OCLExpressions}(\mathcal{I})$ of OCL expressions over $\mathcal{I}$.
  - Given an OCL expression $expr$, a system state $\sigma \in \Sigma_{\mathcal{Y}}$, and a valuation of logical variables $\beta$, define
    \[ I[expr](\sigma, \beta) \in \{\text{true}, \text{false}, \bot\}. \]
- **Later:** use $I$ to define $\models \subseteq \Sigma_{\mathcal{Y}} \times \text{OCLExpressions}(\mathcal{I})$. 

---

**Diagram:**

- **Model:**
  - $\mathcal{I} = (\mathcal{I}, C, V, \text{atn}), \text{SM}$
  - $M = (\Sigma_{\mathcal{Y}}, A_{\mathcal{Y}}, \rightarrow_{\text{SM}})$
  - $\pi = (\sigma_0, \varepsilon_0) \overset{(\text{cons}_{\gamma}, \text{Snd}_{\gamma})}{\rightarrow_{\omega}} (\sigma_1, \varepsilon_1) \cdots$
  - $w_\pi = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}}$
  - $G = (N, E, f)$

- **OCL:**
  - $I(\cdot, \sigma, \beta) \in \{\text{true}, \text{false}, \bot\}$
  - $I : \text{OCLExpressions}(\mathcal{I}) \times \Sigma_{\mathcal{Y}} \times \mathcal{C} \rightarrow \{\text{true}, \text{false}, \bot\}$

---

**Mathematics Diagram: UML**

- **Class Diagram**
- **Component Diagram**
- **Object Diagram**
- **Activity Diagram**
- **Use Case Diagram**
- **Deployment Diagram**
- **Package Diagram**
- **Interaction Diagram**
- **Sequence Diagram**

---

**Diagram:**

- **Diagram**
- **Structure Diagram**
- **Object Diagram**
- **Composite Structure Diagram**
- **Deployment Diagram**
- **Package Diagram**
- **Behavior Diagram**
- **Activity Diagram**
- **Use Case Diagram**
- **Interaction Diagram**
- **Sequence Diagram**
- **Interaction Overview Diagram**
(Core) OCL Syntax [OMG, 2006]
### OCL Syntax 1/4: Expressions

\[ \text{expr ::= w} \]

\[ \mid \text{expr}_1 =_\tau \text{expr}_2 \]

\[ \mid \text{oclIsUndefined}_\tau(\text{expr}_1) \]

\[ \mid \{ \text{expr}_1, \ldots, \text{expr}_n \} \]

\[ \mid \text{isEmpty}(\text{expr}_1) \]

\[ \mid \text{size}(\text{expr}_1) \]

\[ \mid \text{allInstances}_C \]

\[ \mid v(\text{expr}_1) \]

\[ \mid r_1(\text{expr}_1) \]

\[ \mid r_2(\text{expr}_1) \]

Where, given \( \mathcal{I} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \),

- \( W \supseteq \{ \text{self}_C \mid C \in \mathcal{C} \} \) is a set of typed logical variables, \( w \) has type \( \tau(w) \)

- \( \tau \) is any type from \( \mathcal{T} \cup T_B \cup T_C \)

- \( T_B \) is a set of basic types, in the following we use \( T_B = \{ \text{Bool, Int, String} \} \)

- \( T_C = \{ \tau_C \mid C \in \mathcal{C} \} \) is the set of object types,

- \( \text{Set}(\tau_0) \) denotes the set-of-\( \tau_0 \) type for \( \tau_0 \in T_B \cup T_C \) (sufficient because of “flattening” (cf. standard))

- \( v : \tau(v) \in \text{atr}(C), \tau(v) \in \mathcal{T}, \)

- \( r_1 : D_{0,1} \in \text{atr}(C), \)

- \( r_2 : D_\ast \in \text{atr}(C), \)

- \( C, D \in \mathcal{C}. \)
Each expression

$$\omega(expr_1, expr_2, \ldots, expr_n) : \tau_1 \times \cdots \times \tau_n \rightarrow \tau$$

may alternatively be written ("abbreviated as")

- $expr_1 \cdot \omega(expr_2, \ldots, expr_n)$ if $\tau_1$ is an object type, i.e. if $\tau_1 \in T_{\mathcal{C}}$.

- $expr_1 \twoheadrightarrow \omega(expr_2, \ldots, expr_n)$ if $\tau_1$ is a collection type
  (here: only sets), i.e. if $\tau_1 = \text{Set}(\tau_0)$ for some $\tau_0 \in T_B \cup T_{\mathcal{C}}.$
OCL Syntax: Notational Conventions for Expressions

- Each expression

\[ \omega(\text{expr}_1, \text{expr}_2, \ldots, \text{expr}_n) : \tau_1 \times \cdots \times \tau_n \rightarrow \tau \]

may alternatively be written ("abbreviated as")

- \text{expr}_1 \cdot \omega(\text{expr}_2, \ldots, \text{expr}_n) \text{ if } \tau_1 \text{ is an object type, i.e. if } \tau_1 \in T_\mathcal{O}.

- \text{expr}_1 \rightarrow \omega(\text{expr}_2, \ldots, \text{expr}_n) \text{ if } \tau_1 \text{ is a collection type (here: only sets), i.e. if } \tau_1 = \text{Set}(\tau_0) \text{ for some } \tau_0 \in T_B \cup T_\mathcal{O}.

Examples:

- \text{self} : \tau_C \in W; \quad \text{v, w} : \text{Int} \in V; \quad \text{r}_1 : D_{0,1}, \text{r}_2 : D_* \in V)

- if we have methods, e.g.

\[
\begin{array}{c}
\text{C}
\end{array}
\[
\begin{array}{c}
\text{f(1,2):Int}
\end{array}
\]

Then we will also allow in OCL...

\text{self.} \text{f(1,2)} \quad \text{NOT: } \quad \text{f(self, 1, 2)}

which normalizes to...

\text{f(self, 1, 2)}
**OCL Syntax 2/4: Constants, Arithmetical Operators**

For example:

\[ expr ::= \ldots \]

\[ | \text{true} \text{false} \]

\[ | \text{expr}_1 \{\text{and, or, implies}\} \text{expr}_2 \]

\[ | \text{not} \text{expr}_1 \]

\[ | 0| -1| 1| -2| 2| \ldots \]

\[ | \text{OclUndefined} \]

\[ | \text{expr}_1 \{+, -, \ldots\} \text{expr}_2 \]

\[ | \text{expr}_1 \{<, \leq, \ldots\} \text{expr}_2 \]

Generalised notation:

\[ expr ::= \omega(\text{expr}_1, \ldots, \text{expr}_n) \]

\[ : \tau_1 \times \cdots \times \tau_n \rightarrow \tau \]

with \( \omega \in \{+, -, \ldots\} \) \( \text{i.e. } + (\text{expr}_1, \text{expr}_2) \) instead of \( \text{expr}_1 + \text{expr}_2 \)
OCL Syntax 3/4: Iterate

\[ expr ::= \cdots | expr_1 \rightarrow \text{iterate}(w_1 : \tau_1; w_2 : \tau_2 = expr_2 | expr_3) \]

or, with a little renaming,

\[ expr ::= \cdots | expr_1 \rightarrow \text{iterate(iter : \tau_1; result : \tau_2 = expr_2 | expr_3) \]

where

- \( expr_1 \) is of a **collection type** (here: a set \( \text{Set}(\tau_0) \) for some \( \tau_0 \)),
- \( \text{iter} \in W \) is called **iterator**, gets type \( \tau_1 \)
  (if \( \tau_1 \) is omitted, \( \tau_0 \) is assumed as type of \( \text{iter} \))
- \( \text{result} \in W \) is called **result variable**, gets type \( \tau_2 \),
- \( expr_2 \) in an expression of type \( \tau_2 \) giving the **initial value** for \( \text{result} \),
  ('OclUndefined' if omitted)
- \( expr_3 \) is an expression of type \( \tau_2 \)
  in which in particular \( \text{iter} \) and \( \text{result} \) may appear.
Iterate: Intuitive Semantics (Formally: later)

\[
expr ::= \text{expr}_1 \rightarrow \text{iterate}(\text{iter} : \tau_1; \\
\text{result} : \tau_2 = \text{expr}_2 \mid \text{expr}_3)
\]

\[
\begin{align*}
\text{Set}(\tau_0) & \text{ hlp} = \langle \text{expr}_1 \rangle; \\
\tau_1 & \text{ iter}; \\
\tau_2 & \text{ result} = \langle \text{expr}_2 \rangle; \\
\text{while} \ (\text{!hlp}.\text{empty}()) \ \text{do} \\
& \quad \text{iter} = \text{hlp}.\text{pop}(); \\
& \quad \text{result} = \langle \text{expr}_3 \rangle; \\
\text{od}
\end{align*}
\]

**Note:** In our (simplified) setting, we always have \(\text{expr}_1 : \text{Set}(\tau_1)\) and \(\tau_0 = \tau_1\). In the type hierarchy of full OCL with inheritance and \texttt{oclAny}, they may be different and still type consistent.
Abbreviations on Top of Iterate

\[ expr ::= expr_1 \rightarrow \text{iterate}(w_1 : \tau_1; \]
\[ w_2 : \tau_2 = expr_2 | expr_3) \]

• \( expr_1 \rightarrow \text{forAll}(w : \tau_1 | expr_3) \)
  is an abbreviation for
  \( expr_1 \rightarrow \text{iterate}(w : \tau_1; w_1 : \text{Bool} = \text{true} | w_1 \not\in expr_3) \).

  (To ensure confusion, we may again omit all kinds of things, cf. [OMG, 2006].)

• Similar: \( expr_1 \rightarrow \text{Exists}(w : \tau_1 | expr_3) \)
context ::= context \( w_1 : \tau_1, \ldots, w_n : \tau_n \) inv : expr

where \( w \in W \) and \( \tau_i \in T_{\mathcal{C}}, 1 \leq i \leq n, n \geq 0 \).

context \( w_1 : C_1, \ldots, w_n : C_n \) inv : expr

is an abbreviation for

\[
\text{allInstances}_{C_1} \rightarrow \text{forAll}(w_1 : C_1 | \text{expr}) \ldots \\
\text{allInstances}_{C_n} \rightarrow \text{forAll}(w_n : C_n | \text{expr})
\]
Context: More Notational Conventions

- For

  \[
  \text{context } \textit{self} : \tau_C \ \text{inv} : \textit{expr}
  \]

  we may alternatively write ("abbreviate as")

  \[
  \text{context } \tau_C \ \text{inv} : \textit{expr}
  \]

- **Within** the latter abbreviation, we may omit the "**self**" in \textit{expr}, i.e. for

  \[
  \textit{self}.v \text{ and } \textit{self}.r
  \]

  we may alternatively write ("abbreviate as")

  \[
  v \text{ and } r
  \]
### Examples (from lecture “Softwaretechnik 2008”)

```
\[
S = (\{\text{String, Integer, Date, Time}\},
    \{\text{TeamMember, Meeting, Location}\},
    \{\text{age: Integer, ...}\},
    \{\text{TeamMember} \to \{\text{age, name}\}\})
\]
```

<table>
<thead>
<tr>
<th>TeamMember</th>
<th>Meeting</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : String</td>
<td>title : String</td>
<td>name : String</td>
</tr>
<tr>
<td>age : Integer</td>
<td>numParticipants : Integer</td>
<td></td>
</tr>
<tr>
<td>participants</td>
<td>start : Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>duration : Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move(newStart : Date)</td>
<td></td>
</tr>
</tbody>
</table>

- **context** TeamMember **inv:** age \(\rightarrow\) 18
- **context** Meeting **inv:** duration \(>\) 0

```latex
\text{is an OCL expression for: } S
```

```latex
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
\text{context TeamMember } \text{inv: age }\rightarrow\text{ 18} \\
```

(C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freibur...
**context** Meeting

- **inv**: self.participants->size() = numParticipants

**context** Location

- **inv**: name="Lobby" **implies**
  - meeting->isEmpty()
Example (from lecture “Softwaretechnik 2008”)

- context Meeting inv :
  \[
  \left(\text{self}.\ participants \rightarrow \text{iterate}(i: TeamMember; n: Int = 0 \mid n + i \cdot \text{age})\right) \text{ size()} > 25
  \]
“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]
Given an OCL expression \( expr \), a system state \( \sigma \in \Sigma_\mathcal{S} \), and a valuation of logical variables \( \beta \), define

\[
I[\cdot](\cdot, \cdot) : \text{OCLExpressions}(\mathcal{S}) \times \Sigma_\mathcal{S} \times (W \rightarrow I(\mathcal{S} \cup T_B \cup T_E)) \rightarrow I(\text{Bool})
\]

such that

\[
I[expr](\sigma, \beta) \in \{\text{true, false, } \bot_{\text{Bool}}\}.
\]
References
References


