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.
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.

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

(Core) OCL Syntax [OMG, 2006]
OCL Syntax 1/4: Expressions

\[ expr ::= w : \tau(w) \\
| expr_1 \Rightarrow expr_2 : \tau \rightarrow \tau \rightarrow \text{Bool} \\
| \text{oclIsUndefined} (\{ expr_1 \}) : \tau \rightarrow \text{Bool} \\
| \{ expr_1, \ldots, expr_n \} : \tau \times \cdots \times \tau \rightarrow \text{Set}(\tau) \\
| \text{isEmpty} (expr_1) : \text{Set}(\tau) \rightarrow \text{Bool} \\
| \text{size} (expr_1) : \text{Set}(\tau) \rightarrow \text{Int} \\
| \text{allInstances}_C (expr_1) : \tau_C \rightarrow \tau(v) \\
| r_1 (expr_1) : \tau_C \rightarrow \tau_D \\
| r_2 (expr_1) : \tau_C \rightarrow \text{Set}(\tau_D) \]

Where, given \( \mathcal{S} = (\mathcal{F}, \mathcal{C}, V, \text{atr}) \),

- \( W \supseteq \{ \text{self}_C | C \in \mathcal{C} \} \) is a set of typed logical variables, \( w \) has type \( \tau(w) \)
- \( \tau \) is any type from \( \mathcal{S} \cup \mathcal{T}_B \cup \mathcal{T}_e \)
  \( \cup \{ \text{Set}(\tau_0) | \tau_0 \in \mathcal{T}_B \cup \mathcal{T}_e \} \)
- \( \mathcal{T}_B \) is a set of basic types, in the following we use \( \mathcal{T}_B = \{ \text{Bool}, \text{Int}, \text{String} \} \)
- \( \mathcal{T}_e = \{ \tau_C | C \in \mathcal{C} \} \) is the set of object types,
- \( \text{Set}(\tau_0) \) denotes the set-of-\( \tau_0 \) type for \( \tau_0 \in \mathcal{T}_B \cup \mathcal{T}_e \cup \mathcal{S} \)
  (sufficient because of "flattening" (cf. standard))
- \( v : \tau(v) \in \text{atr}(C), \tau(v) \in \mathcal{S} \)
- \( r_1 : D_{0,1} \in \text{atr}(C) \)
- \( r_2 : D_* \in \text{atr}(C) \)
- \( C, D \in \mathcal{C} \).

OCL Syntax: Notational Conventions for Expressions

- Each expression

\[ \mathcal{S} \rightarrow \omega(expr_1, expr_2, \ldots, expr_n) : \tau_1 \times \cdots \times \tau_n \rightarrow \tau \]

may alternatively be written ("abbreviated as")

- \( expr_1 . \omega(expr_2, \ldots, expr_n) \) if \( \tau_1 \) is an object type, i.e. if \( \tau_1 \in \mathcal{T}_e \).
- \( expr_1 \Rightarrow \omega(expr_2, \ldots, expr_n) \) if \( \tau_1 \) is a collection type
  (here: only sets), i.e. if \( \tau_1 = \text{Set}(\tau_0) \) for some \( \tau_0 \in \mathcal{T}_B \cup \mathcal{T}_e \).
OCL Syntax: Notational Conventions for Expressions

- 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")

  - \[ expr_1 . \omega(expr_2, \ldots, expr_n) \] if \( \tau_1 \) is an **object type**, i.e. if \( \tau_1 \in T_{\text{O}} \).

  - \[ expr_1 \Rightarrow \omega(expr_2, \ldots, expr_n) \] if \( \tau_1 \) is a **collection type** (here: only sets), i.e. if \( \tau_1 = \text{Set}(\tau_0) \) for some \( \tau_0 \in T_{\text{B}} \cup T_{\text{O}} \).

- **Examples:**
  
  \[ \text{self} : \tau_{\text{C}} \in W; \quad v, w : \text{Int} \in V; \quad r_1 : D_{0,1}, r_2 : D_{1,2} \in V \]
  
  - \[ \text{self}.\, v() \]
  - \[ \text{self}.\, v('c') \]
  - \[ \text{not} \text{self}.\, v() \]
  - \[ \text{if \text{we have method}, } r_1 \text{, then \text{we will allow in OCL} } \]
  
  \[ \text{self}.\, f(r_1, r_2) \]
  
  which normalizes to
  
  \[ \text{f}(\text{self}, r_1, r_2) \]

OCL Syntax 2/4: Constants, Arithmetical Operators

For example:

\[ expr ::= \ldots \]

| true | false | : Bool  
| expr \{ and, or, implies \} expr | : Bool \times Bool \rightarrow Bool  
| not expr | : Bool \rightarrow Bool  
| 0| 1| 2| 3| 4| \ldots | : Int \in T_{\text{B}}  
| OclUndefined | : \tau  
| expr \{ +, - \} expr | : Int \times Int \rightarrow Int  
| expr \{ <, \leq \} expr | : Int \times Int \rightarrow \text{Bool}  

Generalised notation:

\[ expr ::= \omega(expr_1, \ldots, expr_n) : \tau_1 \times \cdots \times \tau_n \rightarrow \tau \]

with \( \omega \in \{ +, - \} \) i.e. \( expr_1 + \text{expr}_2 \) instead of \( expr_1 + expr_2 \)
expr ::= ... | expr₁->iterate\( (\text{iter} : \tau_1; \text{result} : \tau_2 = \text{expr}_2 \mid \text{expr}_3) \)

or, with a little renaming,

expr ::= ... | expr₁->iterate\( (\text{iter} \mid \text{result} : \tau_2 = \text{expr}_2 \mid \text{expr}_3) \)

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 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)**

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

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

**Note:** In our (simplified) setting, we always have \( expr_1 : \text{Set}(\tau_1) \) 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

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

- \( expr_1 \rightarrow \text{forall}(w : \tau_1 | expr_3) \)
  is an abbreviation for
  \( expr_1 \rightarrow \text{iterate}(w : \tau_1; w_1 : \text{Bool} = \text{true} | w_1 \land expr_3). \)

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

- Similar: \( expr_1 \rightarrow \text{exists}(w : \tau_1 | expr_3) \)

OCL Syntax 4/4: Context

\[ context ::= context \ w_1 : \tau_1, \ldots, w_n : \tau_n \ inv : expr \]

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

\[ context \ w_1 : C_1, \ldots, w_n : C_n \ inv : expr \]

is an abbreviation for
\[ \text{allInstances}_{C_1} \rightarrow \text{forall}(w_1 : C_1 | \ldots \text{allInstances}_{C_n} \rightarrow \text{forall}(w_n : C_n | expr) \ldots ) \]
Context: More Notational Conventions

- For

  context \( \text{self : } \tau_C \text{ inv : expr} \)

  we may alternatively write (“abbreviate as”)

  context \( \tau_C \text{ inv : expr} \)

- Within the latter abbreviation, we may omit the “\(\text{self}\)” in expr, i.e. for

  \( self.v \text{ and self.r } \)

  we may alternatively write (“abbreviate as”)

  \( v \text{ and } r \)

Examples (from lecture)

\[
\tau = \{ \text{String, Integer, Date, Time} \}, \quad \{ \text{TeamMember, Meeting, Location} \}, \quad \{ \text{age : Integer}, \ldots \}, \quad \{ \text{TeamMember#1 : Ego, name} \}\]

\[
\begin{align*}
\text{context TeamMember inv: age} & \geq 18 \\
\text{context Meeting inv: duration} & > 0
\end{align*}
\]
Examples (from lecture “Softwaretechnik 2008”)

OCL/Mehr Navigation/Beispiele

TeamMember
- name : String
- age : Integer

Meeting
- title : String
- numParticipants : Integer
- start : Date
- duration : Time

Location
- name : String

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

- context Location
  - inv : name="Lobby" implies meeting->isEmpty()
“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]
### OCL Syntax 1/4: Expressions

Where, given $\mathcal{P} = (\mathcal{F} \cup \mathcal{N} \cup \mathcal{C})$, 
- $W \supset \{ \text{self} \}$ is a set of typed logical variables, $w$ has type $\tau(w)$
- $\tau$ is any type from $\mathcal{F} \cup \mathcal{T}_B \cup \mathcal{T}_V$ 
  $\cup \{ \text{Set}(\tau_0) \mid \tau_0 \in \mathcal{T}_B \cup \mathcal{T}_V \}$
- $\mathcal{T}_B$ is a set of basic types, in the following we use $\mathcal{T}_B = \{ \text{Bool}, \text{Int}, \text{String} \}$
- $\mathcal{T}_V$ is the set of object types, $\text{Set}(\tau_0)$ denotes the set-of-$\tau_0$ type for $\tau_0 \in \mathcal{T}_B \cup \mathcal{T}_V$ (sufficient because of “flattening” (cf. standard))
- $v : \tau(v) \in \text{attr}(C), \tau(v) \in \mathcal{F}$,
- $r_1 : D_{1,1} \in \text{attr}(C)$,
- $r_2 : D_{2} \in \text{attr}(C)$,
- $C, D \in \mathcal{C}$.

**The Task**

Given an OCL expression $\text{expr}$, a system state $\sigma \in \Sigma_{\mathcal{P}}$, and a valuation of logical variables $\beta$, define

$\mathcal{I} : \text{OCLExpressions}(\mathcal{P}) \times \Sigma_{\mathcal{P}} \times (W \rightarrow \mathcal{I}(\mathcal{F} \cup \mathcal{T}_B \cup \mathcal{T}_V)) \rightarrow \mathcal{I}(\text{Bool})$

such that

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

References
References


