Contents & Goals

Last Lecture:

• OCL Semantics

This Lecture:

• Educational Objectives: Capabilities for following tasks/questions.
  • What does it mean that an OCL expression is satisfiable?
  • When is a set of OCL constraints said to be consistent?
  • What is an object diagram? What are object diagrams good for?
  • When is an object diagram called partial? What are partial ones good for?
  • When is an object diagram an object diagram (wrt. what)?
  • How are system states and object diagrams related?
  • Can you think of an object diagram which violates this OCL constraint?

• Content:
  • OCL: consistency, satisfiability
  • Object Diagrams
  • Example: Object Diagrams for Documentation
OCL Satisfaction Relation
In the following, \( \mathcal{S} \) denotes a signature and \( \mathcal{D} \) a structure of \( \mathcal{S} \).

**Definition (Satisfaction Relation).**

Let \( \varphi \) be an OCL constraint over \( \mathcal{S} \) and \( \sigma \in \Sigma^\mathcal{D} \) a system state. We write

- \( \sigma \models \varphi \) if and only if \( I[\varphi](\sigma, \emptyset) = true \).
- \( \sigma \not\models \varphi \) if and only if \( I[\varphi](\sigma, \emptyset) = false \).

**Note:** In general we can't conclude from \( \neg(\sigma \models \varphi) \) to \( \sigma \not\models \varphi \) or vice versa.
Definition (Consistency). A set \( \text{Inv} = \{ \varphi_1, \ldots, \varphi_n \} \) of OCL constraints over \( \mathcal{I} \) is called consistent (or satisfiable) if and only if there exists a system state of \( \mathcal{I} \) wrt. \( \mathcal{D} \) which satisfies all of them, i.e. if

\[
\exists \sigma \in \Sigma_{\mathcal{D}} : \sigma \models \varphi_1 \land \ldots \land \sigma \models \varphi_n
\]

and inconsistent (or unsatisfiable) otherwise.
• context \( Location \) \( \text{inv}: name = 'Lobby' \) implies meeting \( \rightarrow \) isEmpty()

• context \( Meeting \) \( \text{inv}: title = 'Reception' \) implies location . name = 'Lobby'

• allInstances \( Meeting \rightarrow \exists (w: Meeting | w . title = 'Reception') \)
Deciding OCL Consistency

- Whether a set of OCL constraints is consistent or not is in general not as obvious as in the made-up example.

- **Wanted**: A procedure which decides the OCL satisfiability problem.

- **Unfortunately**: in general **undecidable**.

OCL is as expressive as first-order logic over integers.
Deciding OCL Consistency

- Whether a set of OCL constraints is consistent or not is in general not as obvious as in the made-up example.

- Wanted: A procedure which decides the OCL satisfiability problem.

- Unfortunately: in general undecidable.

  OCL is as expressive as first-order logic over integers.

- And now? Options:
  - Constrain OCL, use a less rich fragment of OCL.
  - Revert to finite domains — basic types vs. number of objects.

Cabot and Clarisó (2008)
OCL Critique
**OCL Critique**

- **Concrete Syntax / Features**
  
  “The syntax of OCL has been criticized – e.g., by the authors of Catalysis [...] – for being hard to read and write.

- OCL’s expressions are stacked in the style of Smalltalk, which makes it hard to see the scope of quantified variables.

- Navigations are applied to atoms and not sets of atoms, although there is a collect operation that maps a function over a set.

- Attributes, [...], are partial functions in OCL, and result in expressions with undefined value.”  
  
  Jackson (2002)
OCL Critique

- **Expressive Power:**
  “Pure OCL expressions only compute primitive recursive functions, but not recursive functions in general.” Cengarle and Knapp (2001)

- **Evolution over Time:** “finally self.x > 0”
  Proposals for fixes e.g. Flake and Müller (2003). (Or: sequence diagrams.)

- **Real-Time:** “Objects respond within 10s”
  Proposals for fixes e.g. Cengarle and Knapp (2002)

- **Reachability:** “After insert operation, node shall be reachable.”
  Fix: add transitive closure.
What Is OCL Good For?
What’s It Good For?

- **Most prominent:**
  
  Formalise **requirements** supposed to be satisfied by all system states.
  
  **Example:** “the choice panels of a VM should be consistent”

  \[
  \text{context VM inv : } \{\text{true}, \text{false}\} \rightarrow \text{exists}(b \mid cp \rightarrow \text{forAll}(c \mid c.\text{wen} = b))
  \]

- **Not unknown:**
  
  Formalise **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over **two** system states (before/after executing the method).

  **Example:** “the dispense water method should decrement \(\text{win}\)”

  \[
  \text{context } DD :: \text{dispense}_W \text{ pre : } \text{win} > 0 \\
  \text{ post : } \text{win} = \text{win}@\text{pre} − 1
  \]

- **Common with State Machines:** **Guards** in transitions.

  \[
  \text{DispenseW}[\text{win} > 0]/\text{dispense}_W
  \]

- **Lesser known:** Specify **operation bodies**.

- **Metamodelling:** the UML standard is a MOF-model of UML.
  OCL expressions define well-formedness of UML models (cf. Lecture \(\sim 21\)).
Where Are We?
\( \mathcal{I} = (\mathcal{S}, \mathcal{C}, V, \text{attr}) \)

\( M = (\Sigma^\mathcal{S}, A^\mathcal{C}, \rightarrow^\mathcal{SM}) \)

\( \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \varepsilon_1) \cdots \)

\( G = (N, E, f) \)

\( \mathcal{D} \)

\( \varphi \in \text{OCL} \)

\( \mathcal{CD}, \mathcal{SM} \)

\( \mathcal{CD}, \mathcal{SD} \)

\( \mathcal{OD} \)

\( \mathcal{UML} \)

\( \mathcal{UML} \)

\( \mathcal{UML} \)

\( \mathcal{UML} \)

\( \mathcal{UML} \)
Object Diagrams
**Definition.** A node-labelled graph is a triple

\[ G = (N, E, f) \]

consisting of

- vertexes \( N \),
- edges \( E \),
- node labeling \( f : N \rightarrow X \), where \( X \) is some label domain,
Definition. Let $\mathcal{D}$ be a structure of signature $\mathcal{I} = (\mathcal{T}, \mathcal{C}, V, atr)$ and $\sigma \in \Sigma_\mathcal{D}$ a system state.

Then any node-labelled graph $G = (N, E, f)$ where

- nodes are identities (not necessarily alive), i.e. $N \subset \mathcal{D}(\mathcal{C})$ finite,
- edges correspond to “links” of objects, i.e.

$$E \subseteq N \times \{v : T \in V | T \in \{C_{0,1}, C_* | C \in \mathcal{C}\}\} \times N,$$

$$\forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r),$$

- objects are labelled with attribute valuations, and non-alive identities with “$X$”, i.e.

$$X = \{X\} \cup (V \mapsto (\mathcal{D}(\mathcal{T}) \cup \mathcal{D}(\mathcal{C})))$$

$$\forall u \in N \cap \text{dom}(\sigma) : f(u) \subseteq \sigma(u)$$

$$\forall u \in N \setminus \text{dom}(\sigma) : f(u) = \{X\}$$

is called object diagram of $\sigma$. 
Object Diagram: Examples

- $N \subseteq D(C)$ finite
- $E \subseteq N \times V_{0,1;*} \times N$
- $\forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r)$,
- $f : N \rightarrow X$,$\quad X = \{X\} \cup (V \rightarrow (D(T) \cup D(C)))$
- $f(u) \subseteq \sigma(u)$,$\quad f(u) = \{X\}$ if $u \notin \text{dom}(\sigma)$

\[
\mathcal{I} = (\{\text{Int}\}, \{C\}, \{x : \text{Int}, y : \text{Int}, r : C*\}, \{C \mapsto \{x, y, r\}\}), \quad D(\text{Int}) = \mathbb{Z}
\]

$\sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{1_C, 3_C\}\}\}$

$G = (N, E, f)$ with

- nodes $N = \{1_C, 3_C\}$
- edges $E = \{(1_C, r, 1_C), (1_C, r, 3_C)\}$,
- node labelling $f = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2\}, 3_C \mapsto X\}$

is an object diagram of $\sigma$.

- Yes, and...? $G$ can equivalently (!) be represented graphically as follows:
Object Diagram: More Examples?

- $N \subset \mathcal{D}(\mathcal{C})$ finite
- $E \subset N \times V_{0,1;\ast} \times N$
- $\forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r)$,
- $f : N \rightarrow X$ • $X = \{X\} \cup (V \mapsto (\mathcal{D}(\mathcal{F}) \cup \mathcal{D}(\mathcal{C}_\ast)))$
- $f(u) \subseteq \sigma(u) / f(u) = \{X\}$ if $u \notin \text{dom}(\sigma)$

$$
\mathcal{I} = (\{\text{Int}\}, \{\mathcal{C}\}, \{x : \text{Int}, y : \text{Int}, r : \mathcal{C}_\ast\}, \{\mathcal{C} \mapsto \{x, y, r\}\}), \quad \mathcal{D}(\text{Int}) = \mathbb{Z}
$$

$$
\sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{2_C\}\}, \quad 2_C \mapsto \{x \mapsto 13, y \mapsto 27, r \mapsto \emptyset\}\}
$$
Definition. Let \( G = (N, E, f) \) be an object diagram of system state \( \sigma \in \Sigma_\mathcal{P} \). We call \( G \) complete wrt. \( \sigma \) if and only if

- \( G \) is object complete, i.e.
  - \( G \) consists of all alive and “linked” non-alive objects, i.e.
    \[
    N = \text{dom}(\sigma) \cup \{u \mid \exists u_1 \in \mathcal{D}(\mathcal{C}), r \in V_{0,1,*} \bullet u \in \sigma(u_1)(r)\}
    \]

- \( G \) is attribute complete, i.e.
  - \( G \) comprises all “links” between objects, i.e. if and only if \( u_2 \in \sigma(u_1)(r) \) for some \( u_1, u_2 \in \mathcal{D}(\mathcal{C}) \) and \( r \in V \), then \( (u_1, r, u_2) \in E \), and
  - each node is labelled with the values of all \( \mathcal{T} \)-typed attributes, i.e. for each \( u \in \text{dom}(\sigma) \),
    \[
    f(u) = \sigma(u)|_{V_{\mathcal{T}}}
    \]
    where \( V_{\mathcal{T}} := \{v : T \in V \mid T \in \mathcal{T}\} \).

Otherwise we call \( G \) partial.
Complete vs. Partial: Examples

- \( N \subset \mathcal{D}(C) \) finite
- \( E \subset N \times V_{0,1;*} \times N \)
- \( \forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r), \)
- \( f : N \to X \)
- \( X = \{X\} \cup (V \mapsto (\mathcal{D}(T) \rightarrow (C))) \)
- \( f(u) \subseteq \sigma(u) / f(u) = \{X\} \text{ if } u \notin \text{dom}(\sigma) \)

\[ \mathcal{J} = (\{\text{Int}\}, \{C\}, \{x : \text{Int}, y : \text{Int}, r : C_*\}, \{C \mapsto \{v_1, v_2, r\}\}), \quad \mathcal{D}(\text{Int}) = \mathbb{Z} \]

\( \sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{2_C, 3_C\}\}, \quad 2_C \mapsto \{x \mapsto 13, y \mapsto 27, r \mapsto \emptyset\}\}, \)

\begin{itemize}
  \item \( \begin{array}{c}
    3_C : C \\
    X
  \end{array} \quad r \quad \begin{array}{c}
    1_C : C \\
    x = 1 \\
    y = 2
  \end{array} \quad r \quad \begin{array}{c}
    2_C : C \\
    x = 13 \\
    y = 27
  \end{array} \)
  \end{itemize}

\begin{itemize}
  \item \( \begin{array}{c}
    3_C : C \\
    X
  \end{array} \quad r \quad \begin{array}{c}
    1_C : C \\
    x = 1 \\
    y = 2
  \end{array} \quad r \quad \begin{array}{c}
    2_C : C \\
    x = 13
  \end{array} \)
  \end{itemize}

\begin{itemize}
  \item \( \begin{array}{c}
    1_C : C \\
    x = 1 \\
    y = 2
  \end{array} \)
  \end{itemize}
Each (consistent) object diagram $G$ represents a set of system states, namely

$$G^{-1} := \{ \sigma \in \Sigma^{\mathcal{G}} \mid G \text{ is an object diagram of } \sigma \}$$

- How many?

- Each finite system state has exactly one complete object diagram.

- A finite system state can have many partial object diagrams.

**Observation:**

If somebody tells us for a given (consistent) object diagram $G$

- that it is meant to be complete, and

- if it is not inherently incomplete (e.g. missing attribute values),

then it uniquely denotes the corresponding system state, denoted by $\sigma(G)$.

Therefore we can use complete object diagrams exchangeably with system states.
Non-Standard Notation

- $\mathcal{S} = (\{\text{Int}\}, \{C\}, \{n, p : C_*\}, \{C \mapsto \{n, p\}\})$.

- Instead of

  \[
  1_{C : C} \rightarrow n \rightarrow 5_{C : C}
  \]

  we want to write

  \[
  1_{C : C} \quad \frac{p = \emptyset}{n} \rightarrow 5_{C : C} \quad \frac{p = \emptyset}{n = \emptyset}
  \]

  or

  \[
  1_{C : C} \quad \frac{p}{n} \rightarrow 5_{C : C} \quad \frac{p}{n}
  \]

  to explicitly indicate that attribute $p : C_*$ has value $\emptyset$ (also for $p : C_{0,1}$).
UML Object Diagrams
UML Notation for Object Diagrams

We assume: different "boxes" means different identities.

E.g. \[ \boxed{\text{C}} \rightarrow \boxed{\text{C}} \]
We slightly deviate from the standard (for reasons):

- We **allow** to show non-alive objects.
  - Allows us to represent “dangling references”, i.e. references to objects which are not alive in the current system state.

- We **introduce** a graphical representation of $\emptyset$ values.
  - Easier to distinguish partial and complete object diagrams.

- In the course, $C_{0,1}$ and $C_\ast$-typed attributes only have **sets** as values. UML also considers multisets, that is, they can have

  $$u_1 : C \xrightarrow{n} u_2 : C$$

  This is **not** an object diagram in the sense of **our definition** because of the requirement on the edges $E$. Extension is straightforward but tedious.
The Other Way Round
If we only have a diagram like

we typically assume that it is meant to be an object diagram wrt. some signature and structure.

In the example, we conclude that the author is referring to some signature \( S = (T, C, V, atr) \) with at least

- \( \{C, D\} \subseteq C \)
- \( T \in \mathcal{T} \)
- \( \{u : C^*, p : C_{01}, z : T\} \subseteq V \)
- \( \{z\} \subseteq atr(D) \)
- \( \{pin\} \subseteq atr(C) \)

and a structure \( D \) with

- \( \{L, L, L\} \subseteq D(C) \)
- \( 3D \subseteq D(D) \)
- \( 0 \in D(T) \)
Example: Object Diagrams for Documentation
Example: Data Structure (Schumann et al., 2008)

BaseNode
+ parent : BaseNode
+ prevSibling : BaseNode
+ nextSibling : BaseNode
+ firstChild : BaseNode
+ lastChild : BaseNode

Node
+ data : T
+ Node( data : T )

Iterator
+ operator++() : Iterator
+ operator--() : Iterator
+ operator*() : BaseNode

Forest
+ appendTopLevel( data : T )
+ appendChild( parent : Iterator, data : T )
+ remove( it : Iterator )
+ depth( it : Iterator ) : int
+ end() :.Iterator
+ begin() : Iterator
+ empty() : bool
+ size() : int
Example: Illustrative Object Diagram (Schumann et al., 2008)

```
Node
+ parent : BaseNode
+ prevSibling : BaseNode
+ nextSibling : BaseNode
+ firstChild : BaseNode
+ lastChild : BaseNode

+ node

BaseNode
+ parent : BaseNode
+ prevSibling : BaseNode
+ nextSibling : BaseNode
+ firstChild : BaseNode
+ lastChild : BaseNode

+ node

Iterator
+ operator ++() : Iterator
+ operator --() : Iterator
+ operator() : BaseNode

+ beginIt
+ endIt

Forest
+ appendTopLevel( data : T )
+ appendChild( parent : Iterator, data : T )
+ remove( it : Iterator )
+ depth( it : Iterator ) : int
+ end() : Iterator
+ begin() : Iterator
+ empty() : bool
+ size() : int

Node
+ data : T
+ Node( data : T )
```
References
References


