OCL Satisfaction Relation

Definition (Satisfaction Relation).
Let $\phi$ be an OCL constraint over $S$ and $\sigma \in \Sigma_D$ a system state.
We write
\[
\sigma \models \phi \quad \text{if and only if} \quad \llbracket \phi \rrbracket (\sigma, \emptyset) = \text{true}.
\]
\[
\sigma \not\models \phi \quad \text{if and only if} \quad \llbracket \phi \rrbracket (\sigma, \emptyset) = \text{false}.
\]

Note: In general we can't conclude from $\neg (\sigma \models \phi)$ to $\sigma \not\models \phi$ or vice versa.

OCL Consistency

Definition (Consistency).
A set $\text{Inv} = \{\phi_1, \ldots, \phi_n\}$ of OCL constraints over $S$ is called consistent (or satisfiable) if and only if there exists a system state of $S$ wrt. $D$ which satisfies all of them, i.e. if
\[
\exists \sigma \in \Sigma_D : \sigma \models \phi_1 \land \ldots \land \sigma \models \phi_n
\]
and inconsistent (or unsatisfiable) otherwise.

Example: OCL Consistent?

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

TeamMember
name : String
age : Integer

Location
participants 2..* meetings

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

move(newStart : Date)

1 *

location

meeting

• context Location inv : name = 'Lobby' implies meeting - > isEmpty()
• context Meeting inv : title = 'Reception' implies location . name = 'Lobby'
• allInstances Meeting - > exists (w : Meeting | w . title = 'Reception')
```
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:
  - Cabot and Clarisó (2008)
    - Constrain OCL, use a less rich fragment of OCL.
  - Revert to finite domains — basic types vs. number of objects.

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.

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.
You are Here.

\[ CD, SM = (T, C, V, atr) \]
\[ SM = (\Sigma DS, A S, \rightarrow SM) \]
\[ \phi \in OCL \]
\[ CD, SD, SD B = (Q SD, q_0, A S, \rightarrow SD, F SD) \]
\[ \pi = (\sigma_0, \epsilon_0) \]
\[ \rightarrow u_0 \]
\[ \cdots \]
\[ \pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}} \-

Recall: Graph

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 \( D \) be a structure of signature \( S = (T, C, V, atr) \) and \( \sigma \in \Sigma DS \) a system state. Then any node-labelled graph \( G = (N, E, f) \) where
- nodes are alive objects, i.e. \( N \subset D(C) \cap \text{dom}(\sigma) \),
- edges start are labelled with derived type attributes, i.e. \( E \subseteq N \times \{ v: T \in V | T \in \{ C_0, 1, C^* | C \in C \} \} \)
- edges correspond to "links" between objects, i.e. \( \forall u_1, u_2 \in D(C), r \in V_0, 1; \ast: (u_1, r, u_2) \in E = \Rightarrow u_2 \in \sigma(u_1)(r) \),
- nodes are labelled with an identity and attribute valuations, i.e. \( X = (V \dot{\cup} \{ id \} \rightharpoonup (D(T) \cup D(C^*))) \)

\( \forall u \in N: f(u) \subseteq \{ id \mapsto \{ u \} \cup \sigma(u) | V T \cup \{ r \mapsto R | r \in V_0, 1; \ast, R \subseteq \sigma(u)(r) \} \) where \( V T := \{ v: T \in V | T \in T \} \) (basic type attributes in \( S \)).
Complete vs. Partial: Examples

- \( N \subset D(\mathcal{C}) \cap \text{dom}(\sigma) \)

- \( E \subset N \times V_0, 1; \star \times N \)

- \( (u_1, r, u_2) \in E \Rightarrow u_2 \in \sigma(u_1)(r) \)

- \( f : N \to X \)

- \( X = (V_\dot{} \cup \{\text{id}\}) \not\to (D(\mathcal{T}) \cup D(\mathcal{C}_*)) \)

- \( f(u) \subseteq \{\text{id} \mapsto \{\text{id}\} \cup \sigma(u) \mid V_T \cup \{r \mapsto \mathbb{R} \mid \mathbb{R} \subseteq \sigma(u)(r)\} \}

- \( \mathcal{S} = (\{\text{Int}\}, \{\mathcal{C}\}, \{x : \text{Int}, y : \text{Int}, r : \mathcal{C}_*\}, \{\mathcal{C} \mapsto \{v_1, v_2, r\}\}) \)

- \( D(\text{Int}) = \mathbb{Z} \)

- \( \sigma = \{1 \mathcal{C} \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \emptyset\}, 2 \mathcal{C} \mapsto \{x \mapsto 13, y \mapsto 27, r \mapsto \emptyset\}\} \)

- \( \mathcal{C}_1 : \mathcal{C}_x = 1, \mathcal{C}_y = 2 \)

- \( \mathcal{C}_2 : \mathcal{C}_x = 13, \mathcal{C}_y = 27 \)

- \( \mathcal{C}_3 : \mathcal{C}_x = 1, \mathcal{C}_y = 2 \)

Complete/Partial is Relative

- Each object diagram-like graph \( G \) represents a set of system states, namely \( G^{-1} = \{\sigma \in \Sigma DS \mid G \text{ is an object diagram of } \sigma\} \).

- Each system state has exactly one complete object diagram.

- A system state can have many partial object diagrams.

- Observation: If somebody tells us for a given 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

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

- Instead of \( 1 \mathcal{C} : \mathcal{C}_p = \emptyset \) we want to write \( 1 \mathcal{C} : \mathcal{C}_p = \emptyset \) to explicitly indicate that attribute \( p : \mathcal{C}_* \) has value \( \emptyset \) (also for \( p : \mathcal{C}_0, 1 \)).

Discussion

- 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, \( \mathcal{C}_0, 1 \) and \( \mathcal{C}_* \)-typed attributes only have sets as values.

- UML also considers multisets, that is, they can have \( u_1 : \mathcal{C}_u_2 : \mathcal{C}_n \mid n \mid p \mid p \mid p \) to explicitly indicate that attribute \( p : \mathcal{C}_* \) has value \( \emptyset \) (also for \( p : \mathcal{C}_0, 1 \)).

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.
Example: Object Diagrams for Documentation

Example: Object Diagram wrt. Signature/Structure

Example: Data Structure

References
References


