Contents & Goals

Last Lecture:
- OCL Semantics (nearly complete)

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 Semantics Cont’d[OMG, 2006]
Putting It All Together

### OCL Syntax 1/4: Expressions

| expr ::= |
| w : τ(w) |
| expr₁ = expr₂ : τ × τ → Bool |
| oclIsUndefined(expr₁) : τ → Bool |
| {expr₁,…,exprₙ} : τ × … × τ → Set(τ) |
| isEmpty(expr₁) : Set(τ) → Bool |
| size(expr₁) : Set(τ) → Int |
| allInstancesC : Set(τ_C) |
| v(expr₁) : τ_C → τ(v) |
| r₁(expr₁) : τ_C → τ_D |
| r₂(expr₁) : τ_C → Set(τ_D) |

Where, given $\mathcal{I} = (\mathcal{T}, \mathcal{C}, \mathcal{F})$, we have:

- $W \supseteq \{\text{self}\}$ is a set of logical variables, $w$ has type $\tau(w)$.
- $\tau$ is any type from $\mathcal{I} \cup \mathcal{F} \cup \{\text{Set}(\tau_0) | \tau_0 \in T_B \cup \mathcal{C}\}$.
- $T_B$ is a set of basic types.
- $\tau_0$ denotes the set of all types.
- $\text{Set}(\tau_0)$ denotes the set of all types.

**For example:**

| expr ::= | ...
| true, false : Bool |
| expr₁ {and, or, implies} expr₂ : Bool × Bool → Bool |
| not expr₁ : Bool → Bool |
| 0, −1, 1, −2, 2, … : Int |
| OclUndefined : τ |
| expr₁ {+, −, …} expr₂ : Int × Int → Int |
| expr₁ {<, ≤, …} expr₂ : Int × Int → Bool |

Generalised notation:

| expr ::= | ω(expr₁,…,exprₙ) : τ₁ × … × τₙ → τ |

with $ω \in \{+, −, …\}$

### OCL Syntax 2/4: Constants, Arithmetical Operators

| expr ::= |
| w : τ(w) |
| expr₁ = expr₂ : τ × τ → Bool |
| oclIsUndefined(expr₁) : τ → Bool |
| {expr₁,…,exprₙ} : τ × … × τ → Set(τ) |
| isEmpty(expr₁) : Set(τ) → Bool |
| size(expr₁) : Set(τ) → Int |
| allInstancesC : Set(τ_C) |
| v(expr₁) : τ_C → τ(v) |
| r₁(expr₁) : τ_C → τ_D |
| r₂(expr₁) : τ_C → Set(τ_D) |

### OCL Syntax 3/4: Iterate

| expr ::= | … | expr₁->iterate(w₁ : τ₁ ; w₂ : τ₂ = expr₂ | expr₃) |

or, with a little renaming,

| expr ::= | … | expr₁->iterate(iter : τ₁; result : τ₂ = expr₂ | expr₃) |

where:

- $expr₁$ is of a collection type (here: a set $\text{Set}(\tau_0)$ for some $\tau_0$).

### OCL Syntax 4/4: Context

| context ::= context w₁ : τ₁,…,wₙ : τₙ inv : expr |

where $w \in W$ and $τ_i \in T_{Φ}, 1 \leq i \leq n, n \geq 0$. 
(vi) Putting It All Together...

\[
expr ::= w | \omega(expr_1, \ldots, expr_n) | \text{allInstances}_C | v(expr_1) | r_1(expr_1) \\
| r_2(expr_1) | expr_1 \rightarrow \text{iterate}(v_1 : \tau_1 ; v_2 : \tau_2 = expr_2 | expr_3)
\]

\[
\beta : \mathcal{W} \rightarrow \bigcup \mathcal{T}
\]

- \(I[w](\sigma, \beta) := \beta(w)\)
- \(I[\omega(expr_1, \ldots, expr_n)](\sigma, \beta) := I(\omega)(I[expr_1](\sigma, \beta), \ldots, I[expr_n](\sigma, \beta))\)
- \(I[\text{allInstances}_C](\sigma, \beta) := \text{dom}(\sigma) \cap \mathcal{D}(C)\)

**Note:** in the OCL standard, \(\text{dom}(\sigma)\) is assumed to be **finite**.

Again: doesn’t scare us.
(vi) Putting It All Together...

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

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

- \(I[v(expr_1)](\sigma, \beta) := \begin{cases} 
(\sigma(v_1))(v) & \text{if } u_1 \in \text{dom}(\sigma) \\
\bot_{\tau_C} & \text{otherwise}
\end{cases}\)

- \(I[r_1(expr_1)](\sigma, \beta) := \begin{cases} 
\sigma(u_1)(r_1) & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\}
\\
\bot_{\tau_2} & \text{otherwise}
\end{cases}\)

- \(I[r_2(expr_1)](\sigma, \beta) := \begin{cases} 
(\sigma(v_1))(v_2) & \text{if } u_1 \in \text{dom}(\sigma) \\
\bot_{\tau_2(\tau_2)} & \text{otherwise}
\end{cases}\)

(Recall: \(\sigma\) evaluates \(r_2\) of type \(C_*\) to a set)
(vi) Putting It All Together...

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

- Base set
- Iterator
- Rule
- Initial value
- Iteration expression

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

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

where \( \beta' = \beta[\text{hlp} \mapsto I[[expr_1]](\sigma, \beta), v_2 \mapsto I[[expr_2]](\sigma, \beta)] \) and

\[ \text{iterate}(\text{hlp}, v_1, v_2, expr_3, \sigma, \beta') \]
\[ := \begin{cases} & I[[expr_3]](\sigma, \beta'[v_1 \mapsto x]) \quad \text{, if } \beta'(\text{hlp}) = \{x\} \\ & I[[expr_3]](\sigma, \beta'') \quad \text{, if } \beta'(\text{hlp}) = X \cup \{x\} \text{ and } X \neq \emptyset \\ \end{cases} \]

where \( \beta'' = \beta'[v_1 \mapsto x, v_2 \mapsto \text{iterate}(\text{hlp}, v_1, v_2, expr_3, \sigma, \beta'[\text{hlp} \mapsto X])] \)

**Quiz:** Is (our) \( I \) a function?
Example

\[ \sigma : \begin{array}{c}
\begin{array}{c}
3_{TM} \vdash TM \\
\text{name} = \text{Name} \\
\text{age} = 27
\end{array}
\end{array} \]

\[ \beta : 5_{TM} \vdash \{ 3_{TM} \} \]

\[ \begin{align*}
\text{I} \text{C}_{\text{self}_{TM}} \text{. age} \geq 18 & \text{I} \text{C}_{\text{self}_{TM}} \text{. age} \geq 18 \\
& \text{I} \text{C}_{\text{age}(\text{self}_{TM}), 18} \\
& \text{I} \text{C}_{\text{duration}(\text{self}_{TM}), 27} \\
& \text{I} \text{C}_{\text{duration}(\text{self}_{TM})} \\
& \text{I} \text{C}_{\text{duration}(\text{self}_{TM}), 27} \\
\end{align*} \]

- **context** TeamMember **inv**: age => 18
- **context** Meeting **inv**: duration > 0
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 \( I[\varphi](\sigma, \emptyset) = \text{true} \).
- \( \sigma \not\models \varphi \) if and only if \( I[\varphi](\sigma, \emptyset) = \text{false} \).

**Note:** In general we can’t conclude from \( \neg (\sigma \models \varphi) \) to \( \sigma \not\models \varphi \) or vice versa.
Definition (Consistency). A set \( Inv = \{\varphi_1, \ldots, \varphi_n\} \) 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 unrealizable) otherwise.
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')
Deciding OCL Consistency

- Whether a set of OCL constraints is satisfiable 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.

  Otherwise we could, for instance, solve diophantine equations

\[
c_1 x_1^{n_1} + \cdots + c_m x_m^{n_m} = d.
\]

Encoding in OCL:

\[
\text{allInstances}_C \rightarrow \exists (w : C \mid c_1 \ast w.x_1^{n_1} + \cdots + c_m \ast w.x_m^{n_m} = d).
\]

- **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**

- **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.
OCL Critique

- **Concrete Syntax**
  
  “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]
Where Are We?
\[ \varphi \in \text{OCL} \]

\[ \mathcal{S} = (\mathcal{T}, \mathcal{C}, V, \text{atr}), \text{SM} \]

\[ M = (\Sigma_{\mathcal{S}}, A_{\mathcal{S}}, \rightarrow_{\text{SM}}) \]

\[ \pi = (\sigma_0, \varepsilon_0) \xrightarrow{\text{cons}_0, \text{Snd}_0} (\sigma_1, \varepsilon_1) \cdots \xrightarrow{u_0} w_\pi = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \]

\[ B = (Q_{SD}, q_0, A_{\mathcal{S}}, \rightarrow_{SD}, F_{SD}) \]

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

\[ \mathcal{O} \mathcal{D} \]
Object Diagrams
Definition. A node-labelled graph is a triple

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

consisting of

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

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

- nodes are identities (not necessarily alive), i.e. $N \subset D(C)$ finite,
- edges correspond to “links” of objects, i.e.
  $$E \subseteq N \times \{v : \tau \in V \mid \tau \in \{C_{0,1}, C_* \mid C \in C\}\} \times N,$$
  $$\forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r),$$
- objects are labelled with attribute valuations and non-alive identities with “X”, i.e.
  $$X = \{X\} \cup (V \rightarrow (D(T) \cup D(C_*))))$$
  $$\forall u \in N \cap \text{dom}(\sigma) : f(u) \subseteq \sigma(u)$$
  $$\forall u \in N \setminus \text{dom}(\sigma) : f(u) = \{X\}$$

is called object diagram of $\sigma$. 

Object Diagram: Example

\[ N \subset \mathcal{D}(\mathcal{C}) \text{ finite, } E \subset N \times V_{0,1,*} \times N, \quad X = \{X\} \cup (V \rightarrow (\mathcal{D}(\mathcal{T}) \cup \mathcal{D}(\mathcal{C}_*))) \]

\[ \forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r), \quad f(u) \subseteq \sigma(u) \text{ or } f(u) = \{X\} \]

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

\[ \sigma = \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2, r \mapsto \{u_2\}\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4, r \mapsto \emptyset\}\} \]

• Then \( G = (N, E, f) \) with

\[ N = \{(u_1, u_2)\}, \quad E = \{(u_1, r, u_2)\}, \quad f = \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2\}, \quad u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4\}\} \]
Object Diagram: Example

\[ N \subset \mathcal{D}(\mathcal{C}) \text{ finite, } \quad E \subset N \times V_{0,1,*} \times N, \quad X = \{X\} \cup (V \to (\mathcal{D}(\mathcal{I}) \cup \mathcal{D}(\mathcal{C}_*))) \]

\[ \forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r), \quad f(u) \subseteq \sigma(u) \text{ or } f(u) = \{X\} \]

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

\[ \sigma = \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2, r \mapsto \{u_2\}\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4, r \mapsto \emptyset\}\} \]

- Then \( G = (N, E, f) \) with

\[ = (\{u_1, u_2\}, \{(u_1, r, u_2)\}, \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4\}\}, \]

is an object diagram of \( \sigma \) wrt. \( \mathcal{I} \) and any structure \( \mathcal{D} \) with \( \mathcal{D}(\text{Int}) \supseteq \{1, 2, 3, 4\} \).
Object Diagram: Example

\[ N \subset \mathcal{D}(C) \text{ finite, } \quad E \subset N \times V_{0,1,*} \times N, \quad X = \{X\} \cup (V \rightarrow (\mathcal{D}(T) \cup \mathcal{D}(C_*))) \]

\[ \forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r), \quad f(u) \subseteq \sigma(u) \text{ or } f(u) = \{X\} \]

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

\[ \sigma = \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2, r \mapsto \{u_2\}\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4, r \mapsto \emptyset\}\} \]

- Then \( G = (N, E, f) \) with

\[ = (\{u_1, u_2\}, \{(u_1, r, u_2)\}, \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4\}\}, \]

is an object diagram of \( \sigma \) wrt. \( \mathcal{I} \) and any structure \( \mathcal{D} \) with \( \mathcal{D}(\text{Int}) \supseteq \{1, 2, 3, 4\} \).

- Node: we may equivalently (!) represent \( G \) graphically as follows:
Object Diagrams: More Examples?

N \subset \mathcal{D}(C) \text{ finite}, \quad E \subset N \times V_{0,1,*} \times N, \quad X = \{X\} \cup (V \mapsto (\mathcal{D}(\mathcal{T}) \cup \mathcal{D}(\mathcal{C}_*)) )

\forall (u_1, r, u_2) \in E : u_1 \in \text{dom}(\sigma) \land u_2 \in \sigma(u_1)(r), \quad f(u) \subseteq \sigma(u) \text{ or } f(u) = \{X\}

\mathcal{S} = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}, p : C_{0,1}, n : C_\ast\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}), \mathcal{D}(\text{Int}) = \mathbb{Z}

\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{x \mapsto 23\}\}

\checkmark \text{ obj. diagram for } \sigma

\times \text{ not an obj. diag. of } \sigma

\checkmark
**Definition.** Let \( G = (N, E, f) \) be an object diagram of system state \( \sigma \in \Sigma_D \).

We call \( G \) **complete** wrt. \( \sigma \) if and only if

- \( G \) is **object complete**, i.e. \( G \) consists of all alive objects, i.e. \( N \supseteq \text{dom}(\sigma) \),
- \( G \) is **attribute complete**, i.e. \( G \) comprises all “links” between alive objects, i.e. if \( u_2 \in \sigma(u_1)(r) \) for some \( u_1, u_2 \in \text{dom}(\sigma) \) and \( r \in V \), then \( (u_1, r, u_2) \in E \), and
- each node is labelled with the values of all \( T \)-typed attributes, i.e. for each \( u \in \text{dom}(\sigma) \),

\[
f(u) = \sigma(u)|_{V_T} \cup \{ r \mapsto (\sigma(u)(r) \setminus N) \mid r \in V : \sigma(u)(r) \setminus N \neq \emptyset \}
\]

where \( V_T := \{ v : \tau \in V \mid \tau \in \mathcal{T} \} \).

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

- \( N = \text{dom}(\sigma) \), if \( u_2 \in \sigma(u_1)(r) \), then \((u_1, r, u_2) \in E\),
- \( f(u) = \sigma(u)|_{V_T} \cup \{r \mapsto (\sigma(u)(r) \setminus N) \setminus \sigma(u)(r) \setminus N\} \)

Complete or partial?

\[
\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{x \mapsto 23\}\}
\]

- \( 1_C : C \)
  \[
  \begin{array}{l}
  p = \emptyset \\
  n = \{5_C\}
  \end{array}
  \]

- \( 5_C : C \)
  \[
  \begin{array}{l}
  n = \emptyset \\
  p = \emptyset
  \end{array}
  \]

- \( 1_D : D \)
  \[
  \begin{array}{l}
  x = 23
  \end{array}
  \]

- \( 1_C : C \)
  \[
  \begin{array}{l}
  p = \emptyset \\
  n = \emptyset
  \end{array}
  \]

- \( 5_C : C \)
  \[
  \begin{array}{l}
  n = \emptyset \\
  p = \emptyset
  \end{array}
  \]

- \( 1_D : D \)
  \[
  \begin{array}{l}
  x = 23
  \end{array}
  \]

Complete: \( \checkmark \)
Partial: \( \times \)
Special Notation

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

- Instead of

  \[
  1_C : C \quad 5_C : C \quad n \quad 1_C : C
  \]

  we want to write

  \[
  1_C : C \quad 5_C : C \quad n \quad 1_C : C
  \]
  
  \[
  p = \emptyset \quad n = \emptyset
  \]

  or

  \[
  1_C : C \quad 5_C : C \quad n \quad 1_C : C
  \]
  
  \[
  p \quad n
  \]

  to explicitly indicate that attribute \( p : C_* \) has value \( \emptyset \) (also for \( p : C_{0,1} \)).
Complete/Partial is Relative

- **Claim:**
  - Each finite system state has **exactly one complete** object diagram.
  - A finite system state can have **many partial** object diagrams.

- Each object diagram $G$ represents a set of system states, namely

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

- **Observation:**
  - If somebody **tells us**, that a given (consistent) object diagram $G$
    - is **meant to be complete**,  
    - and if it is not inherently incomplete (e.g. missing attribute values),
  then we can uniquely reconstruct the corresponding system state.
  In other words: $G^{-1}$ is then a singleton.
Find the 10 differences! (Both diagrams are meant to be complete.)

Definition. Let $\sigma$ be a system state. We say attribute $v \in V_{0,1,*}$ has a dangling reference in object $u \in \text{dom}(\sigma)$ if and only if the attribute’s value comprises an object which is not alive in $\sigma$, i.e. if

$$\sigma(u)(v) \not\subset \text{dom}(\sigma).$$

We call $\sigma$ closed if and only if no attribute has a dangling reference in any object alive in $\sigma$.

Observation: Let $G$ be the (!) complete object diagram of a closed system state $\sigma$. Then the nodes in $G$ are labelled with $\mathcal{T}$-typed attribute/value pairs only.
UML Object Diagrams
UML Notation for Object Diagrams

\[ id : class \]

\[ v_1 = d_1 \]
\[ \vdots \]
\[ v_n = d_n \]

We assume:
- different “boxes”
- different identities

We assume: different “boxes”
- different identities

optional

mandatory

“compartment”
optional

optional
We slightly deviate from the standard (for reasons):

- 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 $u_1 : C_{u_2} : C$

- We allow to give the valuation of $C_{0,1}$- or $C_\ast$-typed attributes in the **values compartment**.

- Allows us to indicate that a certain $r$ is not referring to another object.

- 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.
References


