Software Design, Modelling and Analysis in UML

Lecture 5: Object Diagrams

2016-11-10

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Content

- **Object Constraint Language** completed:
  - Satisfaction Relation, Consistency
  - Decidability
  - OCL Critique
- **Object Diagrams**
  - Definition
  - Graphical Representation
  - Partial vs. Complete Object Diagrams
- **The Other Way Round**
- Object Diagrams for **Documentation**

OCL Satisfaction Relation

OCL Satisfaction Relation

In the following, $\mathscr{S}$ denotes a signature and $\mathscr{D}$ a structure of $\mathscr{S}$.

Definition (Satisfaction Relation).
Let φ be an OCL constraint over $\mathscr{S}$ and $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$ a system state.
We write

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

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

OCL Consistency

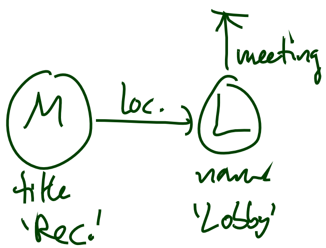
Definition (Consistency). A set *Inv* $= \{\varphi_1, \dots, \varphi_n\}$ of OCL constraints over $\mathscr{S}$ is called consistent (or satisfiable) if and only if there exists a system state of $\mathscr{S}$ wrt. $\mathscr{D}$ which satisfies all of them, i.e. if

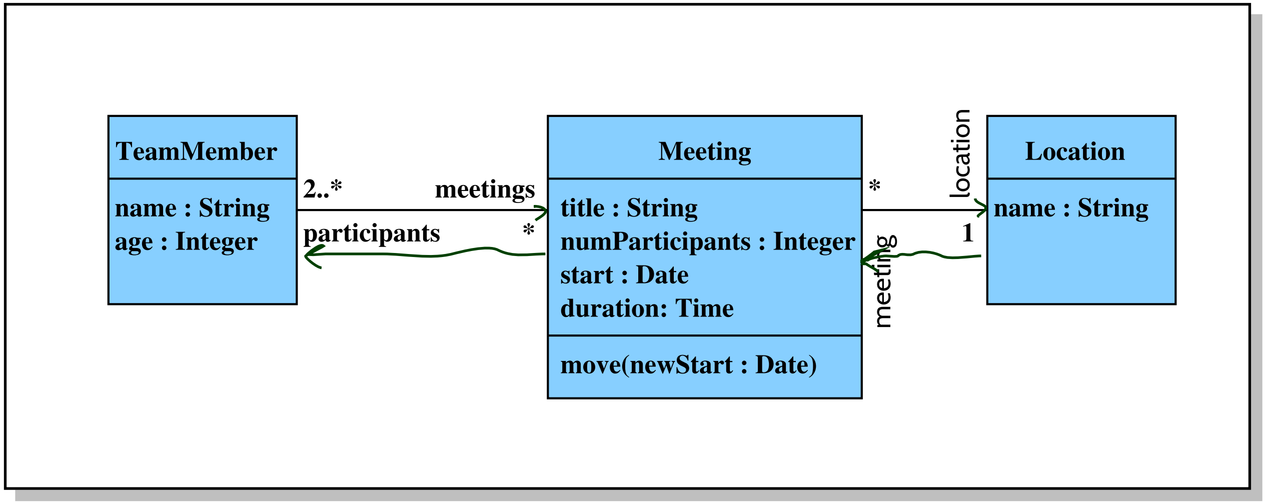
$$\exists\, \sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}} : \sigma \models \varphi_1 \quad \wedge \ldots \wedge \quad \sigma \models \varphi_n$$

and inconsistent (or unsatisfiable) otherwise.

Example: OCL Consistent?

$\mathcal{S} = (\{Integer, String\},$
$\{TeamMember, \ldots\},$
$\{name : String, \ldots\}, \{TeamMember \mapsto \{name, \ldots\}\})$





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

consistent

consistent

- context $Location$ inv : $name =$ 'Lobby' implies $meeting$ -> $isEmpty()$
- context $Meeting$ inv : $title =$ 'Reception' implies $location$. $name =$ 'Lobby'
- allInstances$_{Meeting}$ -> exists($w : Meeting \mid w$. $title =$ 'Reception')
- context Meeting inv: location.meeting -> Exists(i | i = self)

consistent

"self.loc ⇒ self ∈ self.loc.meeting"

not consistent

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.

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.

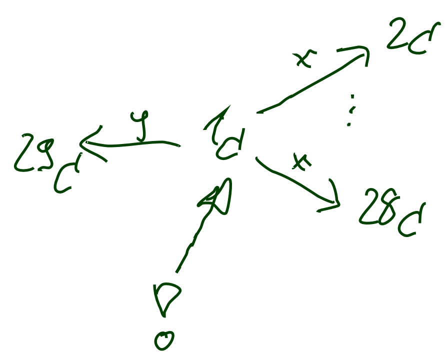
- **Unfortunately**: in general **undecidable**.

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

$$\exists\, x, y \bullet x + y > 27 \qquad \begin{array}{l} x = 27 \\ y = 1 \end{array}$$

$$\mathcal{S} = (\emptyset, \{C\}, \{x : C_*, y : C_*\}, \{C \mapsto \{x, y\}\})$$

$$allInstances_C \rightarrow exists(c \mid c.x \rightarrow size() + c.y \rightarrow size() > 27)$$



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**.

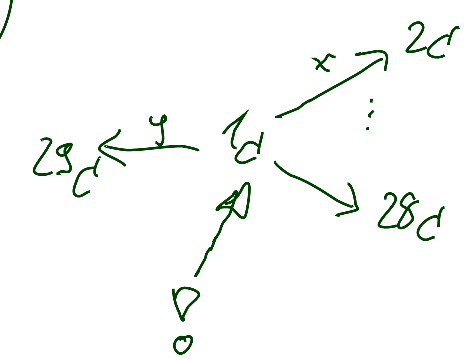
  OCL is as expressive as first-order logic over integers.

$$x = 27 \quad y = 1$$

$$\exists\, x, y \bullet x + y > 27$$

$$\mathcal{S} = (\emptyset, \{C\}, \{x : C_*, y : C_*\}, \{C \mapsto \{x, y\}\})$$

$$allInstances_C \to exists(c \mid c.x \to size() + c.y \to size() > 27)$$



- **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.

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"

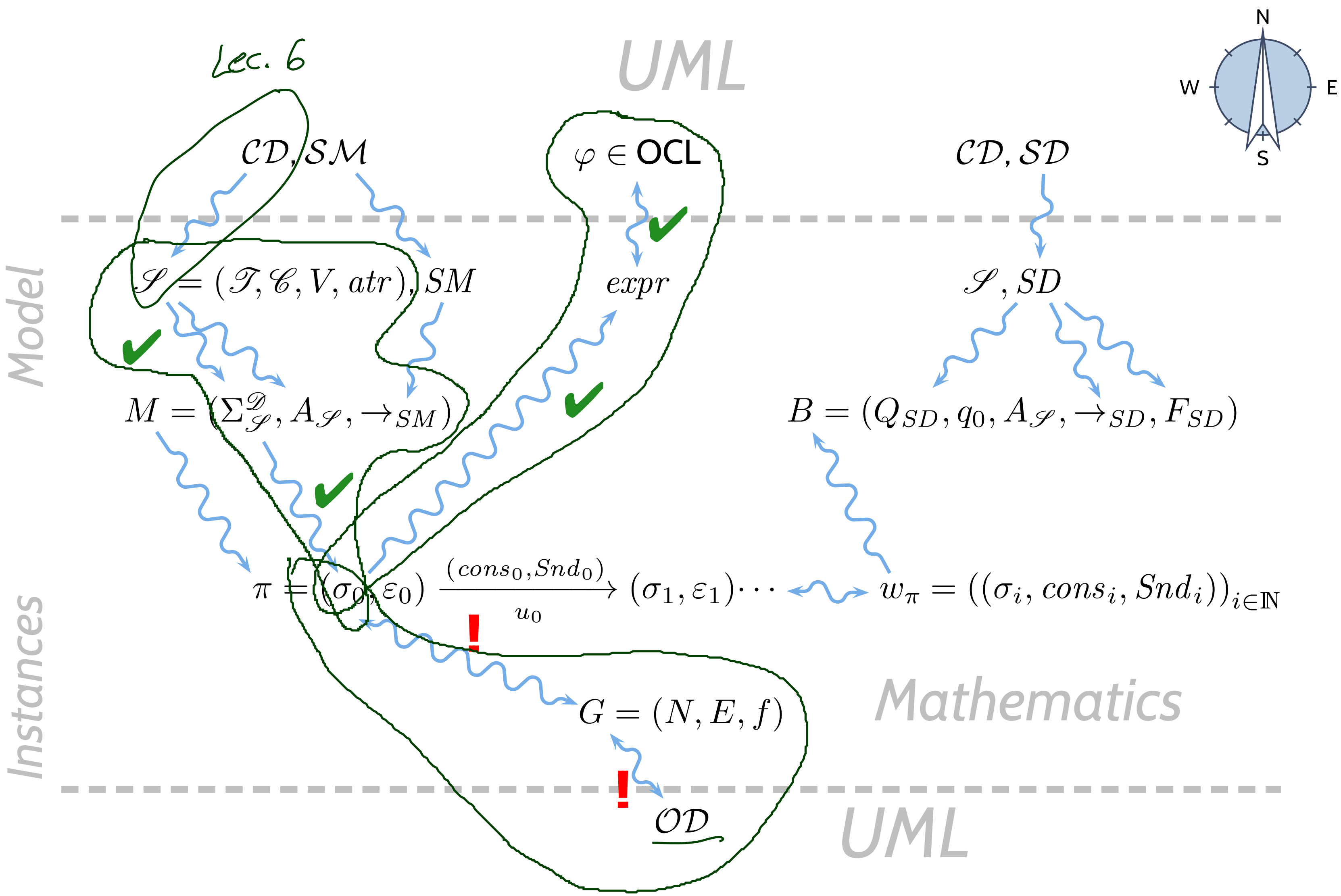
  Proposals for fixes e.g. Cengarle and Knapp (2002)

- **Reachability**: "After insert operation, node shall be reachable."

  Fix: add transitive closure.

Where Are We?

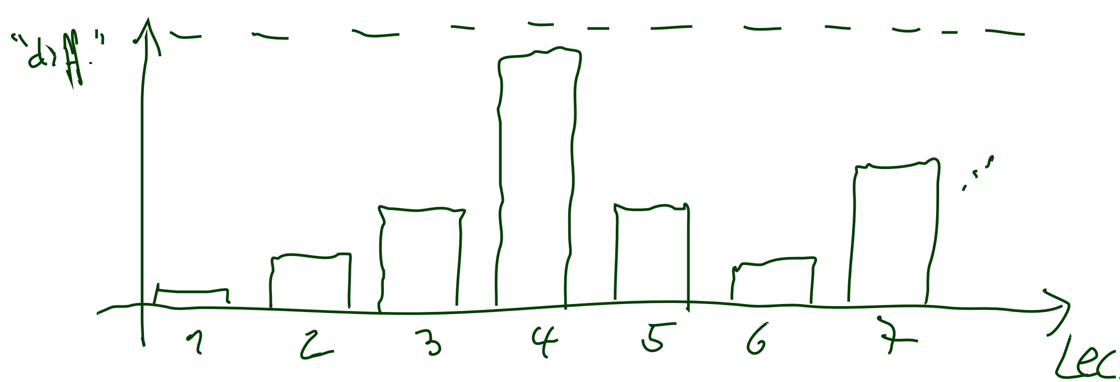
You Are Here.



Lec. 6
UML
$\mathcal{CD}, \mathcal{SM}$
$\varphi \in$ OCL
$\mathcal{CD}, \mathcal{SD}$
N
W
E
S
Model
$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr), SM$
$expr$
$\mathscr{S}, SD$
$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \rightarrow_{SM})$
$B = (Q_{SD}, q_0, A_{\mathscr{S}}, \rightarrow_{SD}, F_{SD})$
$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$
$w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$
Instances
Mathematics
$G = (N, E, f)$
$\mathcal{OD}$
UML

Content

- **Object Constraint Language** completed:
  - Satisfaction Relation, Consistency
  - Decidability
  - OCL Critique
- **Object Diagrams**
  - Definition
  - Graphical Representation
  - Partial vs. Complete Object Diagrams
- **The Other Way Round**
- Object Diagrams for **Documentation**



Object Diagrams

Recall: Graph

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,

Object Diagrams

Definition. Let $\mathscr{D}$ be a structure of signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$ and $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$ a system state.

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

- nodes are alive objects, i.e. $N \subset \mathscr{D}(\mathscr{C}) \cap \mathrm{dom}(\sigma)$,
- edges start are labelled with derived type attributes, i.e.

$$E \subseteq N \times \underbrace{\{v : T \in V \mid T \in \{C_{0,1}, C_* \mid C \in \mathscr{C}\}\}}_{=: V_{0,1;*} \text{ (derived type attributes in } \mathscr{S})} \times N,$$

- edges correspond to "links" between objects, i.e.

$$\forall\, u_1, u_2 \in \mathscr{D}(\mathscr{C}), r \in V_{0,1;*} : (u_1, r, u_2) \in E \implies u_2 \in \sigma(u_1)(r),$$

- nodes are labelled with an identity and attribute valuations, i.e.

$$X = (V \,\dot{\cup}\, \{id\} \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$$

$$\forall\, u \in N : f(u) \subseteq \{id \mapsto \{u\}\} \cup \sigma(u)|_{V_{\mathscr{T}}} \cup \{r \mapsto R \mid r \in V_{0,1;*}, R \subseteq \sigma(u)(r)\}$$

  where $V_{\mathscr{T}} := \{v : T \in V \mid T \in \mathscr{T}\}$ (basic type attributes in $\mathscr{S}$).

is called object diagram of σ.

Object Diagram: Examples

- $N \subset \mathscr{D}(\mathscr{C}) \cap \mathrm{dom}(\sigma)$
- $E \subset N \times V_{0,1;*} \times N$
- $(u_1, r, u_2) \in E \implies u_2 \in \sigma(u_1)(r)$
- $f : N \to X$
- $X = (V \,\dot\cup\, \{id\}) \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*))$
- $f(u) \subseteq \{id \mapsto \{u\}\} \cup \sigma(u)|_{V_{\mathscr{T}}} \cup \{r \mapsto R \mid R \subseteq \sigma(u)(r)\}$

$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{x, y, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

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

- $G = (N, E, f)$ with
  - nodes $N = \{1_C\}$
  - edges $E = \{(1_C, r, 1_C)\}$,
  - node labelling $f = \{1_C \mapsto \{id \mapsto \{1_C\}, x \mapsto 1, r \mapsto \{3_C\}\}\}$

  is an object diagram of σ.

Object Diagram: Examples

- $N \subset \mathscr{D}(\mathscr{C}) \cap \mathrm{dom}(\sigma)$ • $E \subset N \times V_{0,1;*} \times N$ • $(u_1, r, u_2) \in E \implies u_2 \in \sigma(u_1)(r)$ • $f : N \to X$
- $X = (V \mathbin{\dot\cup} \{id\}) \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)$ • $f(u) \subseteq \{id \mapsto \{u\}\} \cup \sigma(u)|_{V_\mathscr{T}} \cup \{r \mapsto R \mid R \subseteq \sigma(u)(r)\}$

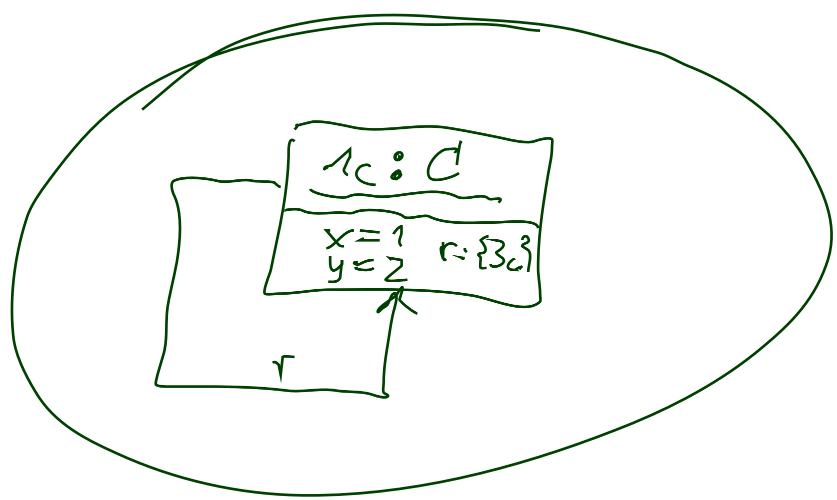
$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{x, y, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

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

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

  is an object diagram of σ.

- Yes, and...?



Object Diagram: Examples

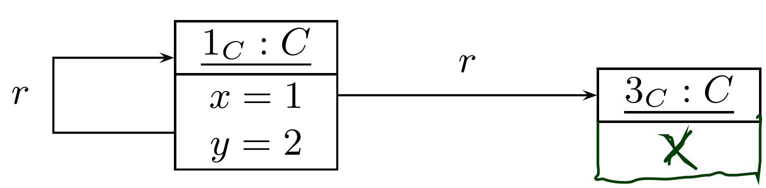
- $N \subset \mathscr{D}(\mathscr{C}) \cap \mathrm{dom}(\sigma)$ • $E \subset N \times V_{0,1;*} \times N$ • $(u_1, r, u_2) \in E \implies u_2 \in \sigma(u_1)(r)$ • $f : N \to X$
- $X = (V \dot\cup \{id\}) \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*))$ • $f(u) \subseteq \{id \mapsto \{u\}\} \cup \sigma(u)|_{V_\mathscr{T}} \cup \{r \mapsto R \mid R \subseteq \sigma(u)(r)\}$

$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{x, y, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

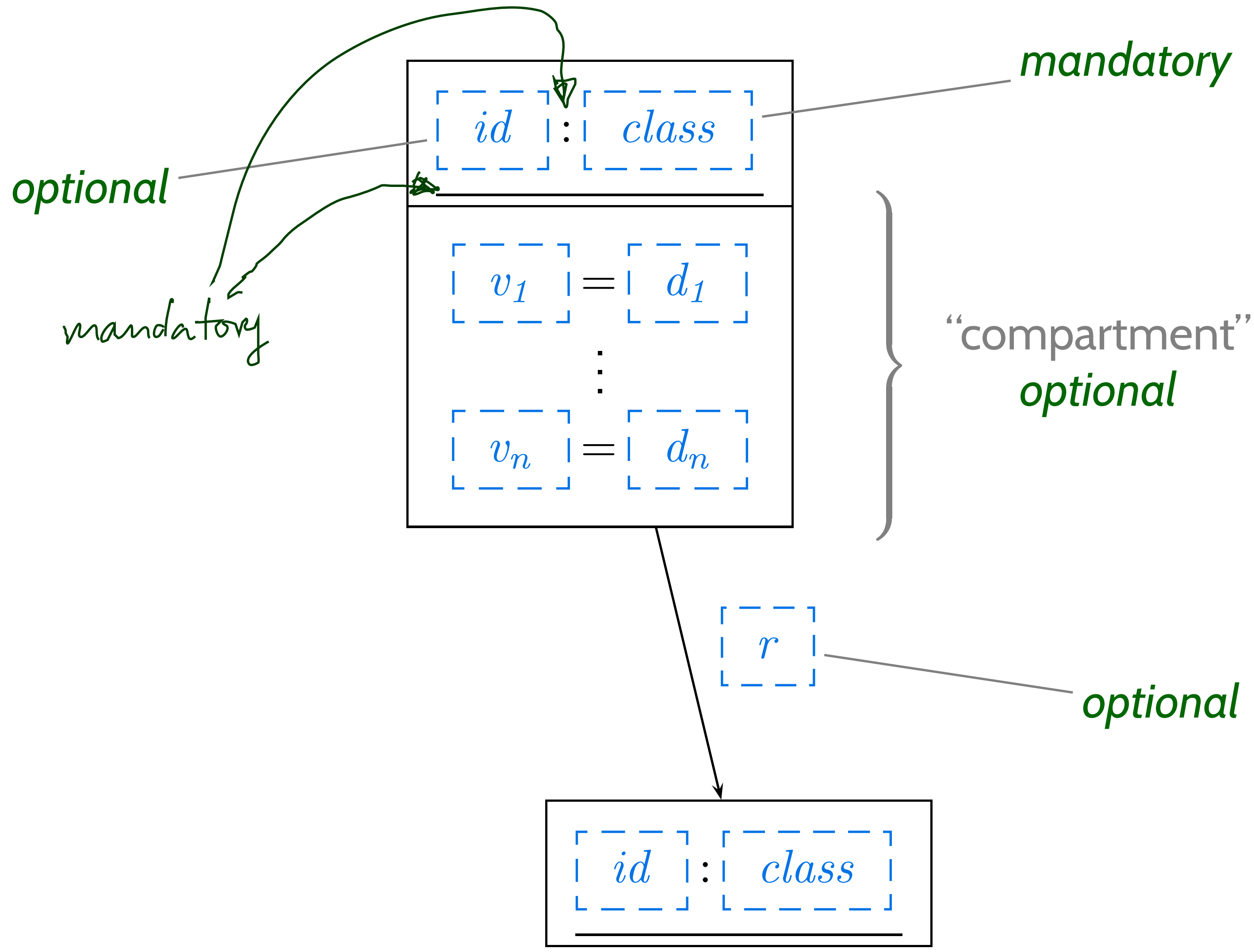
$$\sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{1_C, 3_C\}\}\}$$

- $G = (N, E, f)$ with
  - nodes $N = \{1_C, \quad\}$
  - edges $E = \{(1_C, r, 1_C) \qquad\qquad\qquad \}$,
  - node labelling $f = \{1_C \mapsto \{id \mapsto \{1_C\}, x \mapsto 1, y \mapsto 2\} \qquad\qquad\qquad\qquad \}$

  is an object diagram of σ.
- Yes, and...? G can equivalently (!) be represented graphically:



UML Notation for Object Diagrams

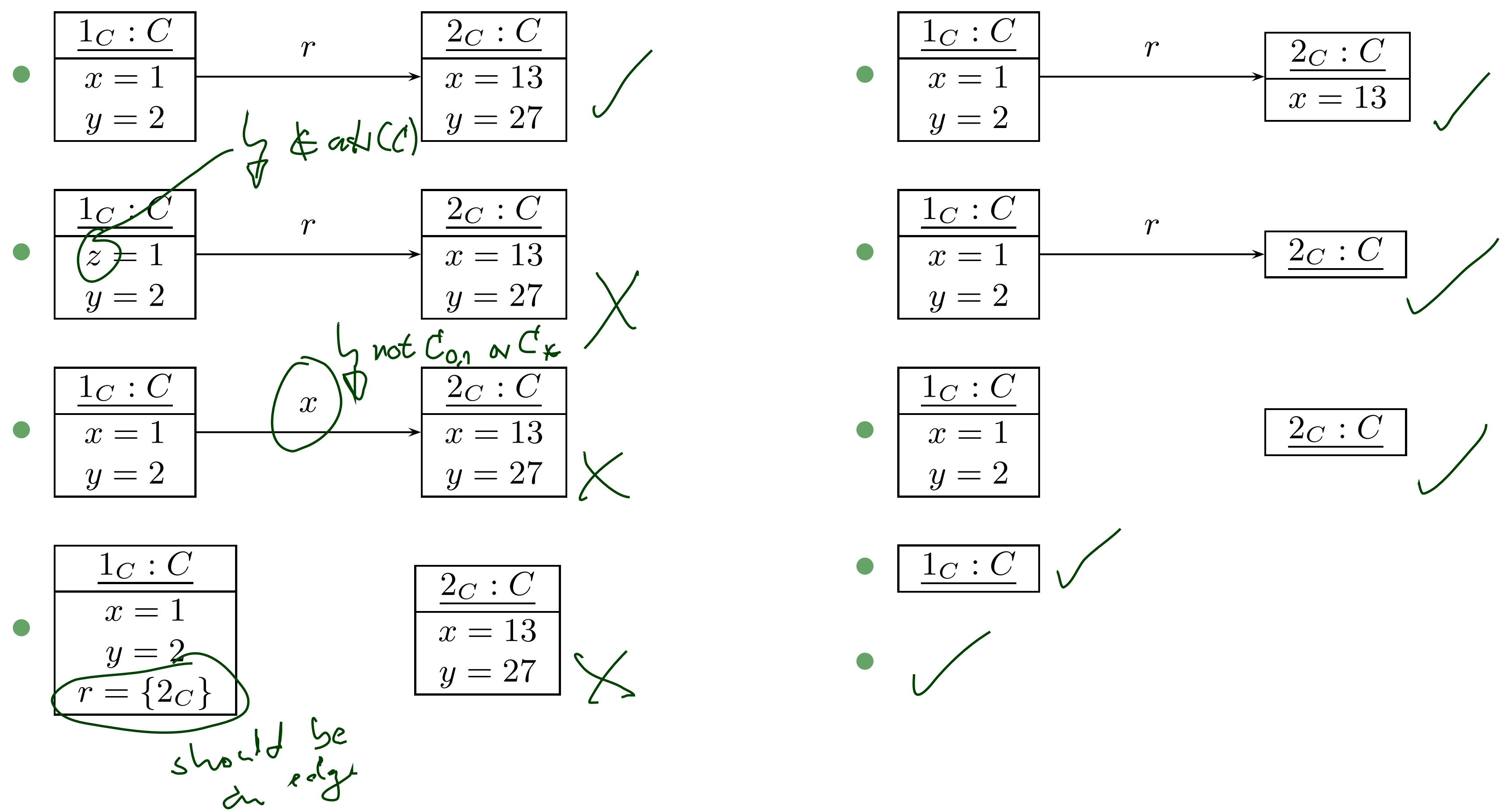


Object Diagram: More Examples?

- $N \subset \mathscr{D}(\mathscr{C}) \cap \mathrm{dom}(\sigma)$
- $E \subset N \times V_{0,1;*} \times N$
- $(u_1, r, u_2) \in E \implies u_2 \in \sigma(u_1)(r)$
- $f : N \to X$
- $X = (V \,\dot\cup\, \{id\}) \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*))$
- $f(u) \subseteq \{id \mapsto \{u\}\} \cup \sigma(u)|_{V_\mathscr{T}} \cup \{r \mapsto R \mid R \subseteq \sigma(u)(r)\}$

$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{v_1, v_2, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

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



Complete vs. Partial Object Diagram

Definition. Let $G = (N, E, f)$ be an object diagram of system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$.
We call G complete wrt. σ if and only if

- G is object complete, i.e.
  - G consists of all alive and "linked" non-alive objects, i.e.

    $$N = \mathrm{dom}(\sigma)$$

- G is attribute complete, i.e.
  - G comprises all "links" between objects, i.e.

    $$\forall\, u_1, u_2 \in N, r \in V_{0,1;*} : (u_1, r, u_2) \in E \iff u_2 \in \sigma(u_1)(r),$$

  - each node is labelled with the values of all $\mathscr{T}$-typed attributes and the dangling references, i.e.

    function restriction

    $$\forall\, u \in \mathrm{dom}(\sigma) \bullet f(u) = \{id \mapsto u\} \cup \sigma(u)|_{V_{\mathscr{T}}} \cup \{r \mapsto \sigma(u)(r) \setminus \mathrm{dom}(\sigma) \mid \sigma(u)(r) \not\subseteq \mathrm{dom}(\sigma)\}.$$

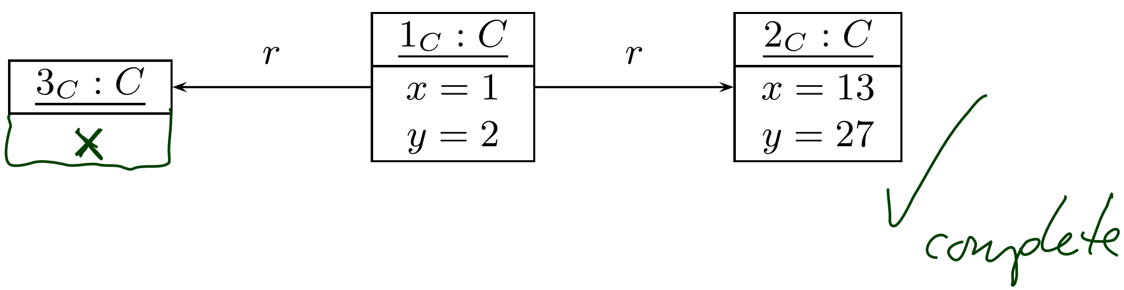
Otherwise we call G partial.

Complete vs. Partial: Examples

- $N \subset \mathscr{D}(\mathscr{C}) \cap \mathrm{dom}(\sigma)$
- $E \subset N \times V_{0,1;*} \times N$
- $(u_1, r, u_2) \in E \implies u_2 \in \sigma(u_1)(r)$
- $f : N \to X$
- $X = (V \,\dot\cup\, \{id\}) \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*))$
- $f(u) \subseteq \{id \mapsto \{u\}\} \cup \sigma(u)|_{V_{\mathscr{T}}} \cup \{r \mapsto R \mid R \subseteq \sigma(u)(r)\}$

$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{v_1, v_2, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

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

- 

Complete/Partial is Relative

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

$$G^{-1} := \{\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}} \mid G \text{ is an object diagram of } \sigma\}$$

- How many?

- 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),

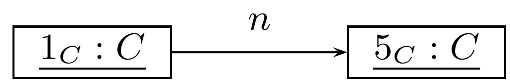
  then it uniquely denotes **the** corresponding system state, denoted by $\sigma(G)$.

  Therefore we can use complete object diagrams **exchangeably** with system states.

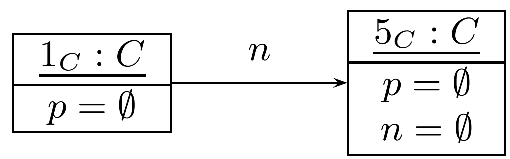
Non-Standard Notation

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

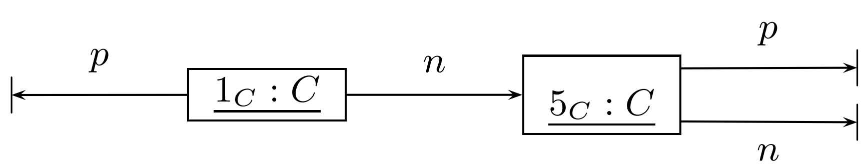
- Instead of



we want to write



or



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

UML Object Diagrams

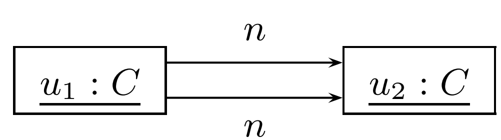
Discussion

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



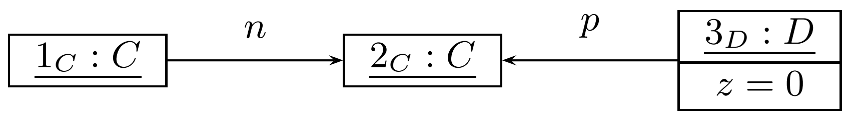
This is **not** an object diagram in the sense of **our definition** because of the requirement on the edges E.

Extension is straightforward but tedious.

The Other Way Round

From Object Diagram to Signature / Structure

- If we **only** have a diagram like



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

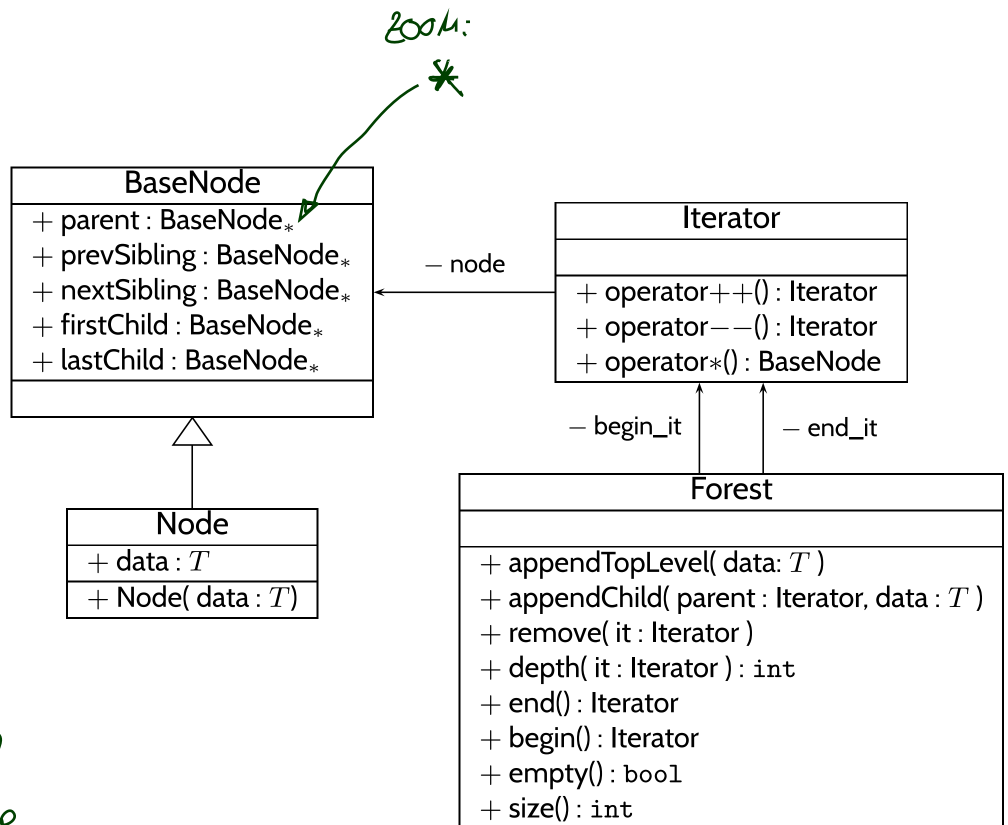
- In the example, we conclude that the author is referring to **some** signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$ with at least
  - $\{C, D\} \subseteq \mathscr{C}$
  - $T \in \mathscr{T}$
  - $\{z : T, n : C_{0,1}, p : C_{0,1}\} \subseteq V$
  - $atr(C) \supseteq \{n\}$
  - $atr(D) \supseteq \{p, z\}$

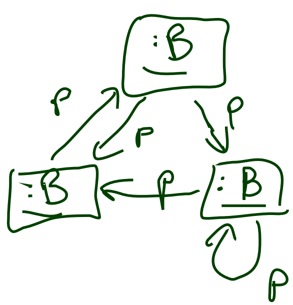
  and a structure $\mathscr{D}$ with

  - $\{1_C, 2_C\} \subseteq \mathscr{D}(C)$
  - $3_D \in \mathscr{D}(D)$
  - $0 \in \mathscr{D}(T)$

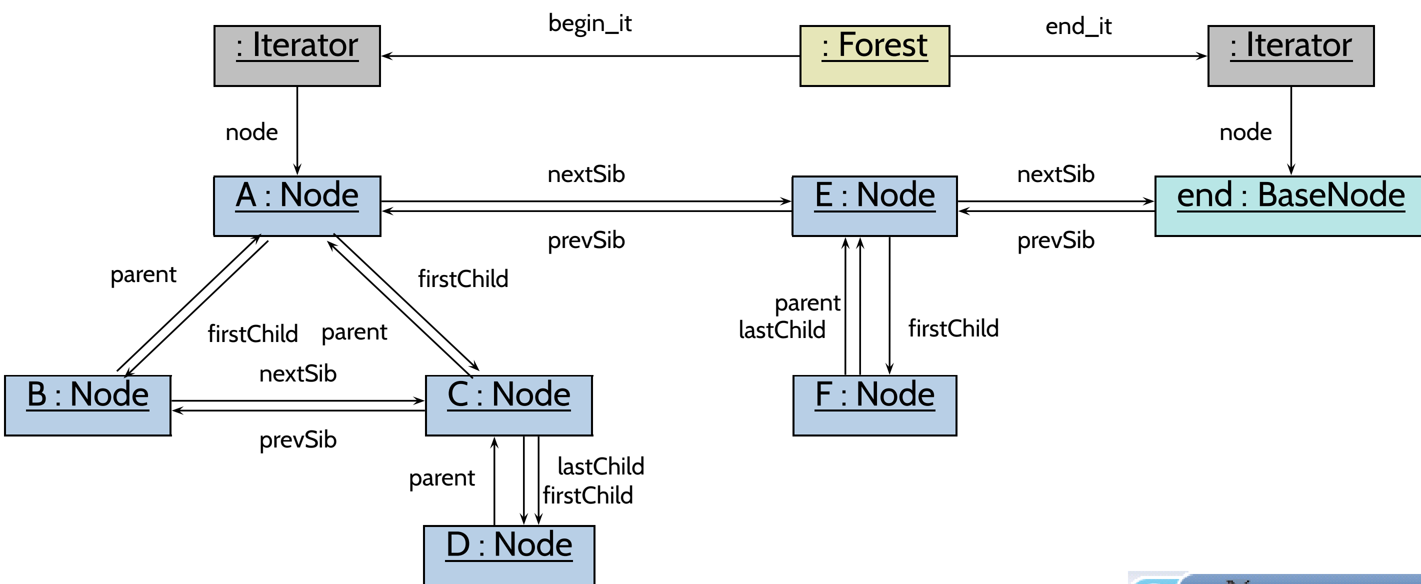
Example: Object Diagrams for Documentation

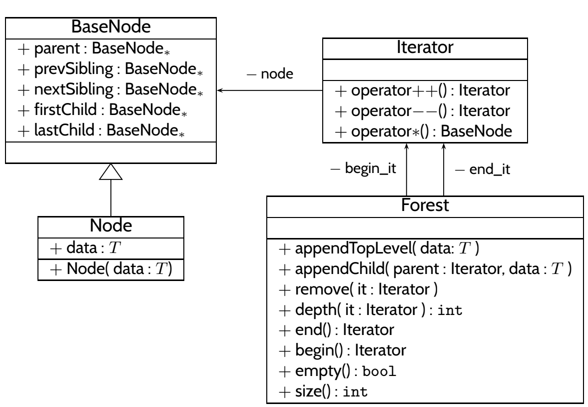
Example: Data Structure (Schumann et al., 2008)

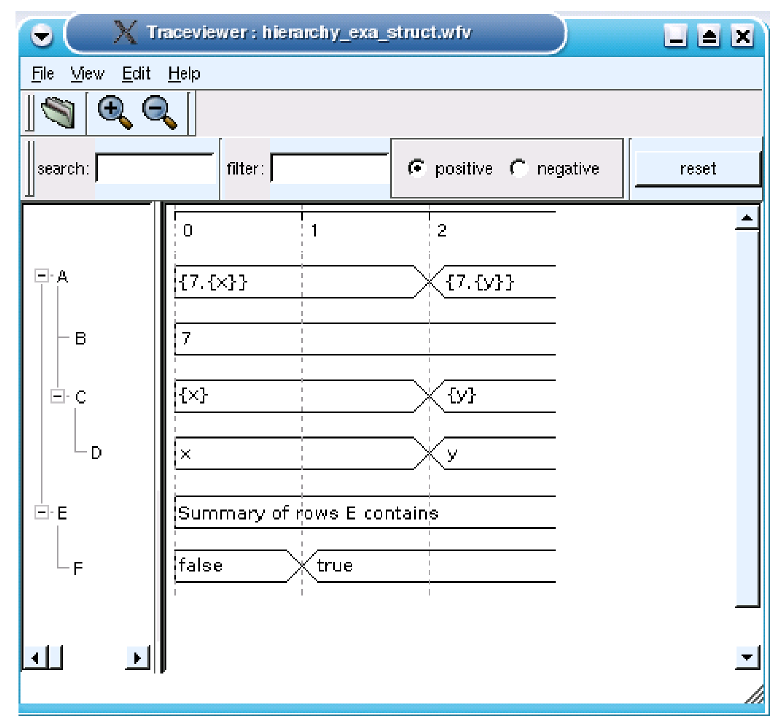




Example: Illustrative Object Diagram (Schumann et al., 2008)







Tell Them What You've Told Them...

- When using an OCL constraint F to formalise **requirements**, we typically ask to ensure $\sigma \models F$.

- **System states** can graphically be represented using **Object Diagrams**.

- Our notation is slightly **non-standard** (for reasons) – mind the syntax (to not **confuse** Object and Class Diagrams)!

- Object diagrams can be **partial** or **complete**, the author's got to tell us.

- An **Object Diagram** for a typical system state can be used as a starting point to **design a signature**.

- **Object Diagrams** can be used to **illustrate**/document how a **structure** is supposed to be used.

References

References

Cabot, J. and Clarisó, R. (2008). UML-OCL verification in practice. In Chaudron, M. R. V., editor, *MoDELS Workshops*, volume 5421 of *Lecture Notes in Computer Science*. Springer.

Cengarle, M. V. and Knapp, A. (2001). On the expressive power of pure OCL. Technical Report 0101, Institut für Informatik, Ludwig-Maximilians-Universität München.

Cengarle, M. V. and Knapp, A. (2002). Towards OCL/RT. In Eriksson, L.-H. and Lindsay, P. A., editors, *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 390–409. Springer-Verlag.

Flake, S. and Müller, W. (2003). Formal semantics of static and temporal state-oriented OCL constraints. *Software and Systems Modeling*, 2(3):164–186.

Jackson, D. (2002). Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.

Schumann, M., Steinke, J., Deck, A., and Westphal, B. (2008). Traceviewer technical documentation, version 1.0. Technical report, Carl von Ossietzky Universität Oldenburg und OFFIS.