Software Design, Modelling and Analysis in UML

Lecture 05: OCL Semantics Cont'd, Object Diagrams

2014-11-06

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

Albert-Ludwigs-Universität Freiburg, Germany

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]

(vi) Putting It All Together

OCL Syntax 1/4: Expressions

$expr ::=$

w	$: \tau(w)$
$\mid expr_1 =_\tau expr_2$	$: \tau \times \tau \to Bool$
$\mid$ oclIsUndefined$_\tau(expr_1)$	$: \tau \to Bool$
$\mid \{expr_1, \dots, expr_n\}$	$: \tau \times \dots \times \tau \to Set(\tau)$
$\mid$ isEmpty$(expr_1)$	$: Set(\tau) \to Bool$
$\mid$ size$(expr_1)$	$: Set(\tau) \to Int$
$\mid$ allInstances$_C$	$: Set(\tau_C)$
$\mid v(expr_1)$	$: \tau_C \to \tau(v)$
$\mid r_1(expr_1)$	$: \tau_C \to \tau_D$
$\mid r_2(expr_1)$	$: \tau_C \to Set(\tau_D)$

Where, given $\mathscr{S} = (\mathscr{T}, \mathscr{C},$

- $W \supseteq \{self\}$ is a set of logical variables, w has
- τ is any type from $\mathscr{T} \cup$ $\cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup$
  - T_B is a set of basic the following we use $T_B = \{Bool, Int, St$
  - $T_\mathscr{C} = \{\tau_C \mid C \in \mathscr{C}\}$ set of object types,
  - $Set(\tau_0)$ denotes the set-of-τ_0 type for $\tau_0 \in T_B \cup T_\mathscr{C}$ (sufficient because "flattening" (cf. sta
- $v : \tau(v) \in atr(C)$, $\tau(v)$
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,
- $C, D \in \mathscr{C}$.

– 03 – 2010-10-27 – Soclsyn –

OCL Syntax 2/4: Constants, Arithmetical Operators

For example:

$expr ::= \dots$

$\mid$ true, false	$: Bool$
$\mid expr_1$ {and, or, implies} $expr_2$	$: Bool \times Bool \to Bool$
$\mid$ not $expr_1$	$: Bool \to Bool$
$\mid 0, -1, 1, -2, 2, \dots$	$: Int$
$\mid$ OclUndefined	$: \tau$
$\mid expr_1 \{+, -, \dots\}\ expr_2$	$: Int \times Int \to Int$
$\mid expr_1 \{<, \leq, \dots\}\ expr_2$	$: Int \times Int \to Bool$

Generalised notation:

$$expr ::= \omega(expr_1, \dots, expr_n) \qquad : \tau_1 \times \dots \times \tau_n \to \tau$$

with $\omega \in \{+, -, \dots\}$

– 03 – 2010-10-27 – Soclsyn –

OCL Syntax /4: Iterate

$$expr ::= \dots \mid expr_1\text{->iterate}(w_1 : \tau_1 \,;\, w_2 : \tau_2 = expr_2 \mid expr_3)$$

or, with a little renaming,

$$expr ::= \dots \mid expr_1\text{->iterate}(iter : \tau_1 ;\, result : \tau_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a collection type (here: a set $Set(\tau_0)$ for some τ_0),

OCL Syntax 4/4: Context

$$context ::= \text{context } w_1 : \tau_1, \dots, w_n : \tau_n \text{ inv} : expr$$

where $w \in W$ and $\tau_i \in T_\mathscr{C}$, $1 \leq i \leq n$, $n \geq 0$.

(vi) Putting It All Together...

$$expr ::= w \mid \omega(expr_1, \dots, expr_n) \mid \text{allInstances}_C \mid v(expr_1) \mid r_1(expr_1)$$
$$\mid r_2(expr_1) \mid expr_1\text{->iterate}(v_1 : \tau_1 \,;\, v_2 : \tau_2 = expr_2 \mid expr_3)$$

$\beta : W \to \bigcup_\tau I(\tau)$

- $I[\![w]\!](\sigma, \beta) := \beta(w)$

$: \tau_1 \times \dots \times \tau_n \to \tau$ $\qquad I(\tau_1) \times \dots \times I(\tau_n) \to I(\tau)$

- $I[\![\omega(expr_1, \dots, expr_n)]\!](\sigma, \beta) := I(\omega)\Big(I[\![expr_1]\!](\sigma,\beta), \dots, I[\![expr_n]\!](\sigma,\beta)\Big)$
- $I[\![\text{allInstances}_C]\!](\sigma, \beta) := \underbrace{dom(\sigma)}_{\text{all alive objects in } \sigma} \cap \mathscr{D}(C)$

Note: in the OCL standard, $\text{dom}(\sigma)$ is assumed to be **finite**.
Again: doesn't scare us.

(vi) Putting It All Together...

$$expr ::= w \mid \omega(expr_1, \dots, expr_n) \mid \text{allInstances}_C \mid v(expr_1) \mid r_1(expr_1)$$
$$\mid r_2(expr_1) \mid expr_1\text{->iterate}(v_1 : \tau_1 \,;\, v_2 : \tau_2 = expr_2 \mid expr_3)$$

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

$\tau_C \to \tau(v)$

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

$\tau_C \to \tau_D,\ r_1 : D_{0,1}$

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

$\tau_C \to Set(\tau_D),\ r_2 : D_*$

- $I[\![r_2(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2) & \text{, if } u_1 \in dom(\sigma) \\ \bot_{Set(\tau_D)} & \text{, otherwise} \end{cases}$

(Recall: σ evaluates r_2 of type C_* to a set)

(vi) Putting It All Together...

the set denoted by $expr_1$ in σ under β

$$expr ::= w \mid \omega(expr_1, \dots, expr_n) \mid \text{allInstances}_C \mid v(expr_1) \mid r_1(expr_1)$$
$$\mid r_2(expr_1) \mid expr_1\text{->iterate}(v_1 : \tau_1 ; v_2 : \tau_2 = expr_2 \mid expr_3)$$

base set, for now: $Set(\tau_1)$ — iterator — result — initial value — iteration expression

- $I[\![\underbrace{expr_1}_{\text{base set}}\text{->iterate}(\underbrace{v_1}_{\text{iterator}} : \tau_1 ; \underbrace{v_2}_{\text{result}} : \tau_2 = \underbrace{expr_2}_{\text{initial value}} \mid \underbrace{expr_3}_{\text{iteration expression}})]\!](\sigma, \beta)$

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

modification of β at hlp and v_2

where $\beta' = \beta[hlp \mapsto \underbrace{I[\![expr_1]\!](\sigma, \beta)}, v_2 \mapsto \underbrace{I[\![expr_2]\!](\sigma, \beta)}_{\text{initial value as given by } expr_2}]$ and

- $iterate(hlp, v_1, v_2, expr_3, \sigma, \beta')$

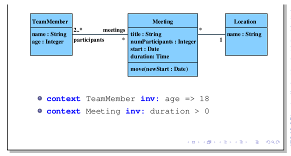
iterator — last element — hlp has exactly one element

$$:= \begin{cases} I[\![expr_3]\!](\sigma, \beta'[v_1 \mapsto x]) & \text{, if } \overbrace{\beta'(hlp) = \{x\}}^{\text{hlp has exactly one element}} \\ I[\![expr_3]\!](\sigma, \beta'') & \text{, if } \underbrace{\beta'(hlp) = X \,\dot\cup\, \{x\}}_{\text{hlp has more than one element left}} \text{ and } X \neq \emptyset \end{cases}$$

where $\beta'' = \beta'[v_1 \mapsto x, v_2 \mapsto \underbrace{iterate(hlp, v_1, v_2, expr_3, \sigma, \beta'[hlp \mapsto X])}_{\text{bind hlp to the rest } \beta'(hlp) \setminus \{x\}}]$

Quiz: Is (our) I a function? recursion

Example



σ:

$3_{TM} : TM$
name = "NN"
age = 27

$\beta : self_{TM} \mapsto \{3_{TM}\}$

blue — def. (2) — black, world

$I[\![self_{TM}.age \geq 18]\!](\sigma,\beta) = I[\![\geq(age(self_{TM}), 18)]\!](\sigma,\beta) \overset{(2)}{=} I(\geq)(27, 18) = true$

$I[\![age(self_{TM})]\!](\sigma,\beta) \overset{\text{def. (1)}}{=} \sigma(3_{TM})(age) = 27 \quad (2)$

$I[\![self_{TM}]\!](\sigma,\beta) = \beta(self_{TM}) = 3_{TM} \quad (1)$

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 $I[\![\varphi]\!](\sigma, \emptyset) = true$.
- $\sigma \not\models \varphi$ 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.

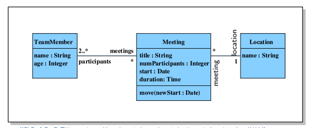
OCL Consistency

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

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

and inconsistent (or unrealizable) otherwise.

OCL Inconsistency Example



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

- 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')

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**

  $$\underbrace{c_1}_{\text{constant}}\underbrace{x_1}_{\text{logical variables}}^{\overbrace{n_1}^{\text{constant exponent}}} + \cdots + c_m x_m^{n_m} = \underbrace{d}_{\text{constant}}.$$

  Encoding in OCL:

  allInstances$_C$ -> exists$(w : C \mid c_1 * w.x_1^{n_1} + \cdots + c_m * w.x_m^{n_m} = d)$.

- **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

- **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

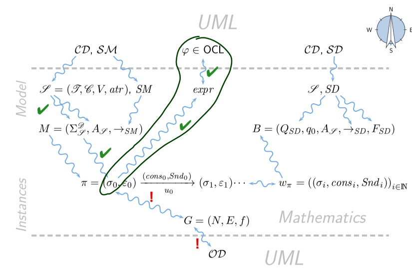
- **Concrete Syntax**

  "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?

You Are Here.



Object Diagrams

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 identities (not necessarily alive), i.e. $N \subset \mathscr{D}(\mathscr{C})$ finite,
- edges correspond to "links" of objects, i.e. $=: V_{0,1,*}$

source attribute dest. object

$$E \subseteq N \times \{v : \tau \in V \mid \tau \in \{C_{0,1}, C_* \mid C \in \mathscr{C}\}\} \times N,$$

source object is alive nodes/

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

source refers to dest. via r

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

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

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

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

is called object diagram of σ.

Object Diagram: Example

$N \subset \mathscr{D}(\mathscr{C})$ finite, $E \subset N \times V_{0,1,*} \times N$, $X = \{\mathsf{X}\} \mathbin{\dot\cup} (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$
$\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$, $f(u) \subseteq \sigma(u)$ or $f(u) = \{\mathsf{X}\}$

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

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

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

$N = \{u_1, u_2\}$
$E = \{(u_1, r, u_2)\}$
$f = \{u_1 \mapsto \{v_1 \mapsto 1, v_2 \mapsto 2\}, u_2 \mapsto \{v_1 \mapsto 3, v_2 \mapsto 4\}\}$

Object Diagram: Example

$N \subset \mathscr{D}(\mathscr{C})$ finite, $E \subset N \times V_{0,1,*} \times N$, $X = \{\mathsf{X}\} \mathbin{\dot\cup} (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$
$\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$, $f(u) \subseteq \sigma(u)$ or $f(u) = \{\mathsf{X}\}$

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

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

- 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 σ wrt. $\mathscr{S}$ and any structure $\mathscr{D}$ with $\mathscr{D}(Int) \supseteq \{1, 2, 3, 4\}$.

Object Diagram: Example

$N \subset \mathscr{D}(\mathscr{C})$ finite, $E \subset N \times V_{0,1,*} \times N$, $X = \{\mathsf{X}\} \mathbin{\dot\cup} (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$
$\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$, $f(u) \subseteq \sigma(u)$ or $f(u) = \{\mathsf{X}\}$

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

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

$\{u_3\}$

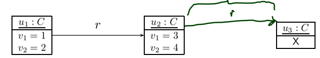
- Then $G = (N, E, f)$ with

u_3

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

is an object diagram of σ wrt. $\mathscr{S}$ and any structure $\mathscr{D}$ with $\mathscr{D}(Int) \supseteq \{1, 2, 3, 4\}$.

- Node: we may equivalently (!) **represent** G graphically as follows:

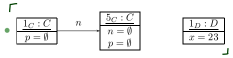
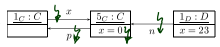
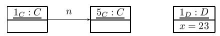
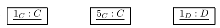


Object Diagrams: More Examples?

$N \subset \mathscr{D}(\mathscr{C})$ finite, $E \subset N \times V_{0,1,*} \times N$, $X = \{\mathsf{X}\} \dot\cup (V \rightharpoonup (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$
$\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$, $f(u) \subseteq \sigma(u)$ or $f(u) = \{\mathsf{X}\}$

$\mathscr{S} = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}), \mathscr{D}(Int) = \mathbb{Z}$

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

-  ✓ obj. diagram for σ
-  ✗ not an obj. diag. of σ
-  ✓
- 
-

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~~ comprises of all alive objects, i.e. $N \supseteq \mathrm{dom}(\sigma)$,
- 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 $u_1, u_2 \in \mathrm{dom}(\sigma)$ and $r \in V$, then $(u_1, r, u_2) \in E$, and
  - each node is labelled with the values of all $\mathscr{T}$-typed attributes, i.e. for each $u \in \mathrm{dom}(\sigma)$,

$$f(u) = \sigma(u)|_{V_{\mathscr{T}}} \cup \{r \mapsto (\sigma(u)(r) \setminus N) \mid r \in V : \sigma(u)(r) \setminus N \neq \emptyset\}$$

where $V_{\mathscr{T}} := \{v : \tau \in V \mid \tau \in \mathscr{T}\}$.

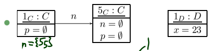
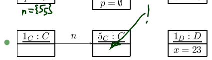
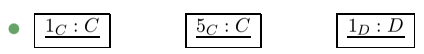
Otherwise we call G partial.

Complete vs. Partial Examples

- $N = \mathrm{dom}(\sigma)$, if $u_2 \in \sigma(u_1)(r)$, then $(u_1, r, u_2) \in E$,
- $f(u) = \sigma(u)|_{V_{\mathscr{T}}} \cup \{r \mapsto (\sigma(u)(r) \setminus N) \mid \sigma(u)(r) \setminus N\}$

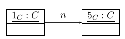
Complete or partial?

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

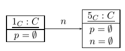
-  ✓ complete
-  ✗ partial
-  partial

Special Notation

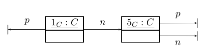
- $\mathscr{S} = (\{Int\}, \{C\}, \{n, p : C_*\}, \{C \mapsto \{n, p\}\})$.
- Instead of



we want to write



or



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

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

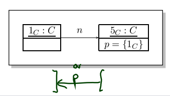
$$G^{-1} := \{\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}} \mid G \text{ is an object diagram of } \sigma\}$$

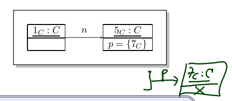
- **Observation**:
  If somebody **tells us**, that a given (consistent) object diagram G
  - is **meant to be complete**,
  - and if it is not inherently incomplete (e.g. missing attribute values),

  then we can uniquely reconstruct the corresponding system state.
  In other words: G^{-1} is then a singleton.

Closed Object Diagrams vs. Dangling References

Find the 10 differences! (Both diagrams are meant to be complete.)





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

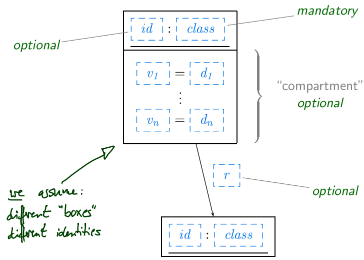
$$\sigma(u)(v) \not\subseteq \mathrm{dom}(\sigma).$$

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

Observation: Let G be the (!) complete object diagram of a **closed** system state σ. Then the nodes in G are labelled with $\mathscr{T}$-typed attribute/value pairs only.

UML Object Diagrams

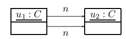
UML Notation for Object Diagrams



Discussion

We slightly deviate from the standard (for reasons):

- 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.)

- We **allow** to give the valuation of $C_{0,1}$- or C_*-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

[Cabot and Clarisó, 2008] 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 and Knapp, 2001] 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 and Knapp, 2002] 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 and Müller, 2003] 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, 2002] Jackson, D. (2002). Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290.

[OMG, 2006] OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.