Softwaretechnik / Software-Engineering

Lecture 11: Structural Software Modelling II

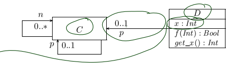
2019-06-24

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

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Architecture & Design: Content

From Abstract to Concrete Syntax



$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, F, mth)$

- $\mathscr{T} = \{Int, Bool\}$
- $\mathscr{C} = \{C, D\}$
- $V = \{x : Int, p : C_{0,1}, n : C_*\}$
- $atr = \{C \mapsto \{n, p\}, D \mapsto \{x, p\}\}$
- $F = \{f : Int \to Bool, \ldots$
- $mth = \{C \mapsto \emptyset, \ldots$

31/61

Basic Object System Structure Example

Wanted: a structure for signature

$$\mathscr{S}_0 = (\{Int, Bool\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$

A structure $\mathscr{D}$ maps
- $\tau \in \mathscr{T}$ to **some** $\mathscr{D}(\tau)$, $C \in \mathscr{C}$ to **some** identities $\mathscr{D}(C)$ (infinite, pairwise disjoint),
- C_* and $C_{0,1}$ for $C \in \mathscr{C}$ to $\mathscr{D}(C_{0,1}) = \mathscr{D}(C_*) = 2^{\mathscr{D}(C)}$.

$$\begin{aligned} \mathscr{D}(Int) &= \mathbb{Z} \\ \mathscr{D}(C) &= \mathbb{N} \times \{C\} = \{1_C, 2_C, 3_C, \ldots\} \\ \mathscr{D}(D) &= \mathbb{N} \times \{D\} = \{1_D, 2_D, 3_D, \ldots\} \\ \mathscr{D}(C_{0,1}) = \mathscr{D}(C_*) &= 2^{\mathscr{D}(C)} \\ \mathscr{D}(D_{0,1}) = \mathscr{D}(D_*) &= 2^{\mathscr{D}(D)} \end{aligned}$$

$\mathscr{D}'$: $\{3, 17, 25\}$, $\{\bullet, \blacktriangle, \ldots\}$, $\{\circ, \ldots\}$

40/61

System State Examples

$$\mathscr{S}_0 = (\{Int, Bool\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$
$$\mathscr{D}(Int) = \mathbb{Z}, \quad \mathscr{D}(C) = \{1_C, 2_C, 3_C, \ldots\}, \quad \mathscr{D}(D) = \{1_D, 2_D, 3_D, \ldots\}$$

A system state is a partial function $\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$ such that
- $\mathrm{dom}(\sigma(u)) = atr(C)$,
- $\sigma(u)(v) \in \mathscr{D}(\tau)$ if $v : \tau, \tau \in \mathscr{T}$,
- $\sigma(u)(v) \in \mathscr{D}(C_*)$ if $v : D_*$ or $v : D_{0,1}$ with $D \in \mathscr{C}$.

$\sigma_1 = \{2_C \mapsto \{p \mapsto \{1_C\}, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{2_C\}, x \mapsto 27\}\}$

$\sigma_2 = \emptyset$

$\sigma_3 = \{5_C \mapsto \{p \mapsto \{13_C\}, n \mapsto \emptyset\}\}$ ✓

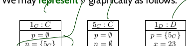
42/61

Object Diagrams

$$\mathscr{S}_0 = (\{Int, Bool\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

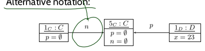
$$\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}$$

- We may **represent** σ graphically as follows:

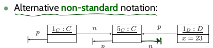


This is an **object diagram**.

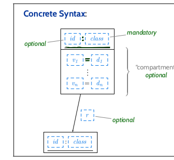
- Alternative notation:



- Alternative **non-standard** notation:



Concrete Syntax:



50/61

Content

- **Object Diagrams Cont'd**
  - dangling references
  - partial vs. complete
  - object diagrams at work
- **Proto-OCL**
  - syntax, semantics
  - Proto-OCL vs. OCL
  - Putting It All Together: Proto-OCL vs. Software

Object Diagrams Cont'd

Special Case: Dangling Reference

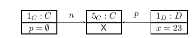
Definition.
Let $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$ be a system state and $u \in \text{dom}(\sigma)$ an alive object of class C in σ.

We say $r \in atr(C)$ **is a dangling reference in** u if and only if $r : C_{0,1}$ or $r : C_*$ and u refers to a **non-alive** object via r, i.e.

$$\sigma(u)(r) \not\subseteq \text{dom}(\sigma).$$

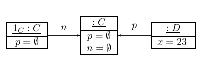
Example:

- $\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}$
- Object diagram representation:



Special Case: Anonymous Objects

If the object diagram



is considered as **complete**, then it denotes the set of all system states

$$\{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{c\}\}, c \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, d \mapsto \{p \mapsto \{c\}, x \mapsto 23\}\}$$

where $c \in \mathscr{D}(C)$, $d \in \mathscr{D}(D)$, $c \neq 1_C$.

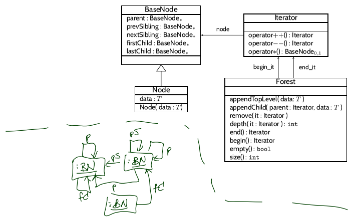
Intuition: different boxes represent different objects.

Content

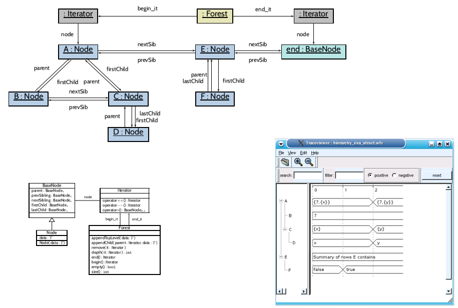
- **Object Diagrams Cont'd**
  - dangling references
  - partial vs. complete
  - object diagrams at work
- **Proto-OCL**
  - syntax, semantics
  - Proto-OCL vs. OCL
  - Putting It All Together: Proto-OCL vs. Software

Object Diagrams at Work

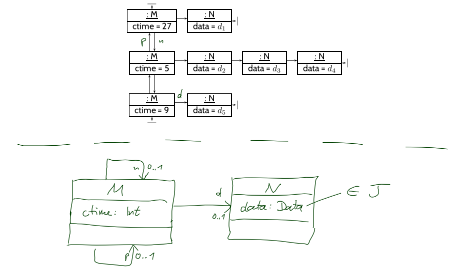
Example: Data Structure (Schumann et al., 2008)



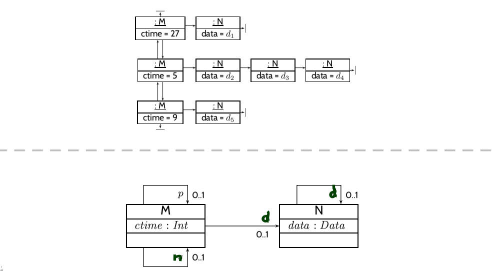
Example: Illustrative Object Diagram (Schumann et al., 2008)



Object Diagrams for Structural Analysis



Object Diagrams for Structural Analysis

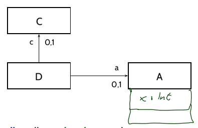


Content

- **Object Diagrams Cont'd**
  - dangling references
  - partial vs. complete
  - object diagrams at work
- **Proto-OCL**
  - syntax, semantics
  - Proto-OCL vs. OCL
  - Putting It All Together: Proto-OCL vs. Software

Towards Object Constraint Logic (OCL) — "Proto-OCL" —

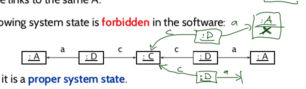
Motivation



- How do I **precisely, formally** tell **my developers** that

  All D-instances having a link to the same C object must have links to the same A.

- That is, the following system state is **forbidden** in the software:

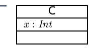


  Note: formally, it is a **proper system state**.

- Use **(Proto-)OCL**: "Dear developers, please only use system states which satisfy:"

  $\forall\, d_1 \in allInstances_D \bullet \forall\, d_2 \in allInstances_D \bullet c(d_1) = c(d_2) \implies a(d_1) = a(d_2)$

Constraints on System States: Proto-OCL Syntax



- **Example**: for all C-instances, x should never have the value 27.

$$\forall\, c \in allInstances_C \bullet x(c) \neq 27$$

> **Definition. Proto-OCL Formulae** wrt. signature $(\mathscr{T}, \mathscr{C}, V, atr, F, mth)$ (c is a **logical variable**, $C \in \mathscr{C}$):
>
> $$\begin{array}{rlll} F ::= & c & : \tau_C & \\ \mid & allInstances_C & : 2^{\tau_C} & \\ \mid & v(F) & : \tau_C \to \tau_\perp, & \text{if } v : \tau \in atr(C), \tau \in \mathscr{T} \\ \mid & v(F) & : \tau_C \to \tau_D, & \text{if } v : D_{0,1} \in atr(C) \\ \mid & v(F) & : \tau_C \to 2^{\tau_D}, & \text{if } v : D_* \in atr(C) \\ \mid & f(F_1, \dots, F_n) & : \tau_1 \times \dots \times \tau_n \to \tau, & \text{if } f : \tau_1 \times \dots \times \tau_n \to \tau \\ \mid & \forall\, c \in F_1 \bullet F_2 & : \tau_C \times 2^{\tau_C} \times \mathbb{B}_\perp \to \mathbb{B}_\perp & \end{array}$$

- The formula above in **prefix normal form**: $\forall\, c \in allInstances_C \bullet \neq(\ x(c),\ 27\ )$

Semantics

- **Proto-OCL Types:**
  - $\mathcal{I}[\![\tau_C]\!] = \mathscr{D}(C) \,\dot\cup\, \{\perp\}$, $\mathcal{I}[\![\tau_\perp]\!] = \mathscr{D}(\tau) \,\dot\cup\, \{\perp\}$, $\mathcal{I}[\![2^{\tau_C}]\!] = \mathscr{D}(C_*) \,\dot\cup\, \{\perp\}$
  - $\mathcal{I}[\![\mathbb{B}_\perp]\!] = \{$*true*, *false*$\} \,\dot\cup\, \{\perp\}$, $\mathcal{I}[\![\mathbb{Z}_\perp]\!] = \mathbb{Z} \,\dot\cup\, \{\perp\}$
- **Functions:**
  - We assume $f_\mathcal{I}$ given for each function symbol f ($\to$ in a minute).
- **Proto-OCL Semantics** (interpretation function):

$$\mathcal{I}[\![\cdot]\!](\cdot,\cdot) : \text{Proto-OCL-Formulae} \times \Sigma_\mathscr{S}^\mathscr{D} \times B \to \{\textit{true}, \textit{false}, \perp\}$$

  - $\mathcal{I}[\![c]\!](\sigma, \beta) = \beta(c)$ (assuming β is a type-consistent valuation of the logical variables),
  - $\mathcal{I}[\![allInstances_C]\!](\sigma, \beta) = \mathrm{dom}(\sigma) \cap \mathscr{D}(C)$,
  - $\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \begin{cases} \sigma(\mathcal{I}[\![F]\!](\sigma,\beta))(v) & \text{, if } \mathcal{I}[\![F]\!](\sigma,\beta) \in \mathrm{dom}(\sigma) \\ \perp & \text{, otherwise} \end{cases}$ (if not $v : C_{0,1}$)
  - $\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \begin{cases} u' & \text{, if } \mathcal{I}[\![F]\!](\sigma,\beta) \in \mathrm{dom}(\sigma) \text{ and } \sigma(\mathcal{I}[\![F]\!](\sigma,\beta))(v) = \{u'\} \\ \perp & \text{, otherwise} \end{cases}$ (if $v : C_{0,1}$)
  - $\mathcal{I}[\![f(F_1, \dots, F_n)]\!](\sigma, \beta) = f_\mathcal{I}(\mathcal{I}[\![F_1]\!](\sigma,\beta), \dots, \mathcal{I}[\![F_n]\!](\sigma,\beta))$,
  - $\mathcal{I}[\![\forall\, c \in F_1 \bullet F_2]\!](\sigma, \beta) = \begin{cases} \textit{true} & \text{, if } \mathcal{I}[\![F_2]\!](\sigma, \beta[c := u]) = \textit{true} \text{ for all } u \in \mathcal{I}[\![F_1]\!](\sigma,\beta) \\ \textit{false} & \text{, if } \mathcal{I}[\![F_2]\!](\sigma, \beta[c := u]) = \textit{false} \text{ for some } u \in \mathcal{I}[\![F_1]\!](\sigma,\beta) \\ \perp & \text{, otherwise} \end{cases}$

Semantics Cont'd

- Proto-OCL is a **three-valued** logic: a formula evaluates to *true*, *false*, or $\perp$.
- **Example**: $\wedge_\mathcal{I}(\cdot,\cdot) : \{$*true*, *false*, $\perp\} \times \{$*true*, *false*, $\perp\} \to \{$*true*, *false*, $\perp\}$ is defined as follows:

x_1	*true*	*true*	*true*	*false*	*false*	*false*	$\perp$	$\perp$	$\perp$
x_2	*true*	*false*	$\perp$	*true*	*false*	$\perp$	*true*	*false*	$\perp$
$\wedge_\mathcal{I}(x_1, x_2)$	*true*	*false*	$\perp$	*false*	*false*	*false*	$\perp$	*false*	$\perp$

  We assume common logical connectives $\neg, \wedge, \vee, \dots$ with canonical 3-valued interpretation.

- **Example**: $+_\mathcal{I}(\cdot,\cdot) : (\mathbb{Z} \,\dot\cup\, \{\perp\}) \times (\mathbb{Z} \,\dot\cup\, \{\perp\}) \to \mathbb{Z} \,\dot\cup\, \{\perp\}$

$$+_\mathcal{I}(x_1, x_2) = \begin{cases} x_1 + x_2 & \text{, if } x_1 \neq \perp \text{ and } x_2 \neq \perp \\ \perp & \text{, otherwise} \end{cases}$$

  We assume common arithmetic operations $-, /, *, \dots$ and relation symbols $>, <, \leq, \dots$ with **monotone** 3-valued interpretation.

- And we assume the special unary function symbol $isUndefined$:

$$isUndefined_\mathcal{I}(x) = \begin{cases} \textit{true} & \text{, if } x = \perp, \\ \textit{false} & \text{, otherwise} \end{cases}$$

  $isUndefined_\mathcal{I}$ is **definite**: it never yields $\perp$.

Example: Evaluate Formula for System State

$$\forall\, c \in allInstances_C \bullet x(c) \neq 27$$

- Recall **prefix notation**: $\forall\, c \in allInstances_C \bullet \neq(x(c), 27)$

  Note: $\neq$ is a binary function symbol, 27 is a 0-ary function symbol.

- **Example**:

  $\mathcal{I}[\![\forall\, c \in allInstances_C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) =$ *true*, because…

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta)$, $\quad \beta := \emptyset[c := 1_C] = \{c \mapsto 1_C\}$

  $=$

Example: Evaluate Formula for System State

$$\forall\, c \in allInstances_C \bullet x(c) \neq 27$$

- Recall **prefix notation**: $\forall\, c \in allInstances_C \bullet \neq(x(c), 27)$

  Note: $\neq$ is a binary function symbol, 27 is a 0-ary function symbol.

- **Example**:

  $\mathcal{I}[\![\forall\, c \in allInstances_C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) =$ *true*, because…

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta)$, $\quad \beta := \emptyset[c := 1_C] = \{c \mapsto 1_C\}$

  $= \neq_\mathcal{I}(\ \mathcal{I}[\![x(c)]\!](\sigma, \beta),\ \mathcal{I}[\![27]\!](\sigma, \beta)\ )$

  $= \neq_\mathcal{I}(\ \sigma(\ \mathcal{I}[\![c]\!](\sigma, \beta)\ )(x),\ 27_\mathcal{I}\ )$

  $= \neq_\mathcal{I}(\ \sigma(\ \beta(c)\ )(x),\ 27_\mathcal{I}\ )$

  $=$

Example: Evaluate Formula for System State

$\forall\, c \in allInstances_C \bullet x(c) \neq 27$

- Recall **prefix notation**: $\forall\, c \in allInstances_C \bullet \neq(x(c), 27)$
  Note: $\neq$ is a binary function symbol, 27 is a 0-ary function symbol.
- **Example**:
  $\mathcal{I}[\![\forall\, c \in allInstances_C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) =$ *true*, because…

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta), \quad \beta := \emptyset[c := 1_C] = \{c \mapsto 1_C\}$

  $= \neq_{\mathcal{I}}(\, \mathcal{I}[\![x(c)]\!](\sigma, \beta),\ \mathcal{I}[\![27]\!](\sigma, \beta)\,)$

  $= \neq_{\mathcal{I}}(\, \sigma(\, \mathcal{I}[\![c]\!](\sigma, \beta)\,)(x),\ 27_{\mathcal{I}}\,)$

  $= \neq_{\mathcal{I}}(\, \sigma(\, \beta(c)\,)(x),\ 27_{\mathcal{I}}\,)$

  $= \neq_{\mathcal{I}}(\, \sigma(\, 1_C\,)(x),\ 27_{\mathcal{I}}\,)$

  $= \neq_{\mathcal{I}}(\, 13,\ 27\,) =$ *true* …and 1_C is the only C-object in σ: $\mathcal{I}[\![allInstances_C]\!](\sigma, \emptyset) = \{1_C\}$.

More Interesting Example

$\sigma = \{1_C \mapsto \{x \mapsto 13,\ n \mapsto \emptyset\}\}$

$\forall\, c \in allInstances_C \bullet x(n(c)) \neq 27$

- Similar to the previous slide, we need the value of
  $$\mathcal{I}[\![x(n(c))]\!](\sigma, \beta), \beta = \{c \mapsto 1_C\}$$
  - $\mathcal{I}[\![c]\!](\sigma, \beta) = \beta(c) = 1_C$
  - $\mathcal{I}[\![n(c)]\!](\sigma, \beta) = \bot$ since $\sigma(\, \mathcal{I}[\![c]\!](\sigma, \beta)\,)(n) = \emptyset \neq \{u'\}$ by rule

    $$\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \begin{cases} u' & \text{, if } \mathcal{I}[\![F]\!](\sigma, \beta) \in \mathrm{dom}(\sigma) \text{ and } \sigma(\mathcal{I}[\![F]\!](\sigma, \beta))(v) = \{u'\} \\ \bot & \text{, otherwise} \end{cases} \quad (\text{if } v : C_{0,1})$$

  - $\mathcal{I}[\![x(n(c))]\!](\sigma, \beta) = \bot$ since $\mathcal{I}[\![n(c)]\!](\sigma, \beta) = \bot$ by rule

    $$\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \begin{cases} \sigma\,(\mathcal{I}[\![F]\!](\sigma, \beta))\,(v) & \text{, if } \mathcal{I}[\![F]\!](\sigma, \beta) \in \mathrm{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases} \quad (\text{if not } v : C_{0,1})$$

More Interesting Example

$\forall\, c \in allInstances_C \bullet x(n(c)) \neq 27$

- Similar to the previous slide, we need the value of
  $$\mathcal{I}[\![x(n(c))]\!](\sigma, \beta), \beta = \{c \mapsto 1_C\}$$
  - $\mathcal{I}[\![c]\!](\sigma, \beta) = \beta(c) = 1_C$
  - $\mathcal{I}[\![n(c)]\!](\sigma, \beta) = \bot$ since $\sigma(\, \mathcal{I}[\![c]\!](\sigma, \beta)\,)(n) = \emptyset \neq \{u'\}$ by rule

    $$\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \begin{cases} u' & \text{, if } \mathcal{I}[\![F]\!](\sigma, \beta) \in \mathrm{dom}(\sigma) \text{ and } \sigma(\mathcal{I}[\![F]\!](\sigma, \beta))(v) = \{u'\} \\ \bot & \text{, otherwise} \end{cases} \quad (\text{if } v : C_{0,1})$$

  - $\mathcal{I}[\![x(n(c))]\!](\sigma, \beta) = \bot$ since $\mathcal{I}[\![n(c)]\!](\sigma, \beta) = \bot$ by rule

    $$\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \begin{cases} \sigma\,(\mathcal{I}[\![F]\!](\sigma, \beta))\,(v) & \text{, if } \mathcal{I}[\![F]\!](\sigma, \beta) \in \mathrm{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases} \quad (\text{if not } v : C_{0,1})$$

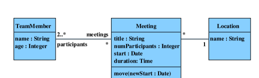
– 11 – 2019-06-24 – Socl –
24/36

Object Constraint Language (OCL)

OCL is the same – just with less readable (?) syntax.

Literature: (OMG, 2006; Warmer and Kleppe, 1999).

Examples *(from lecture "Softwaretechnik 2008")*

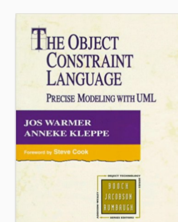


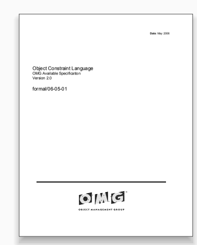

```
• context Meeting
    • inv: self.participants->size() =
           numParticipants
• context Location
    • inv: name="Lobby" implies
           meeting->isEmpty()
```

$\forall\, self \in allInstances_{Meeting} \bullet size(participants(self)) = numParticipants(self)$

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

Literature





Where To Put OCL Constraints?

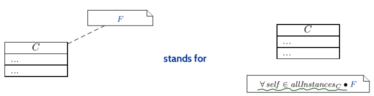
- **Notes**: A UML **note** is a diagram element of the form

  [*text*]

  text can principially be **everything**, in particular **comments** and **constraints**.

  Sometimes, content is **explicitly classified** for clarity: OCL: F

- Conventions:



Content

Putting It All Together

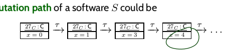
Modelling Structure with Class Diagrams

> Definition. **Software** is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) **computation paths** of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$ where
> - $\sigma_i \in \Sigma$, $i \in \mathbb{N}_0$, is called **state** (or **configuration**), and
> - $\alpha_i \in A$, $i \in \mathbb{N}_0$, is called **action** (or **event**).
>
> The (possibly partial) function $[\![\cdot]\!] : S \mapsto [\![S]\!]$ is called **interpretation** of S.

- The set of **states** Σ could be the set of **system states** as defined by a class diagram, e.g.

  $\Sigma := \Sigma_{\mathscr{S}}^{\mathscr{D}}$ $\qquad \mathscr{S}$: C | x : Int

- A corresponding **computation path** of a software S could be



- If a requirement is formalised by the Proto-OCL constraint

  $$F = \forall\, c \in allInstances_C \bullet x(c) < 4$$

  then S **does not** satisfy the requirement.

More General: Software vs. Proto-OCL

- Let $\mathscr{S}$ be an **object system signature** and $\mathscr{D}$ a **structure**.
- Let S be a **software** with
  - states $\Sigma \subseteq \Sigma_{\mathscr{S}}^{\mathscr{D}}$, and
  - **computation paths** $[\![S]\!]$.
- Let F be a Proto-OCL constraint over $\mathscr{S}$.
- We say $[\![S]\!]$ **satisfies** F, denoted by $[\![S]\!] \models F$, if and only if for all

  $$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \in [\![S]\!]$$

  and all $i \in \mathbb{N}_0$,

  $$\mathcal{I}[\![F]\!](\sigma_i, \emptyset) = \mathit{true}.$$

- We say $[\![S]\!]$ **does-not-satisfy** F, denoted by $[\![S]\!] \not\models F$, if and only if there exists $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \in [\![S]\!]$ and $i \in \mathbb{N}_0$, such that $\mathcal{I}[\![F]\!](\sigma_i, \emptyset) = \mathit{false}$.
- **Note**: $\neg([\![S]\!] \not\models F)$ does not imply $[\![S]\!] \models F$.

Topic Area Architecture & Design: Content

Tell Them What You've Told Them…

- **Class Diagrams** can be used to **graphically**
  - visualise code,
  - define an **object system structure** $\mathscr{S}$.
- An **Object System Structure** $\mathscr{S}$ (together with a structure $\mathscr{D}$)
  - defines a set of **system states** $\Sigma_{\mathscr{S}}^{\mathscr{D}}$.
- A **System State** $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$
  - can be **visualised** by an **object diagram**.
- **Proto-OCL** constraints can be evaluated on **system states**.
- A **software** over $\Sigma_{\mathscr{S}}^{\mathscr{D}}$ satisfies a Proto-OCL constraint F if and only if F evaluates to *true* in all system states of all the software's computation paths.

References

References

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

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

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.

Warmer, J. and Kleppe, A. (1999). *The Object Constraint Language*. Addison-Wesley.