Software Design, Modelling and Analysis in UML

Lecture 4: OCL Semantics

2015-11-03

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

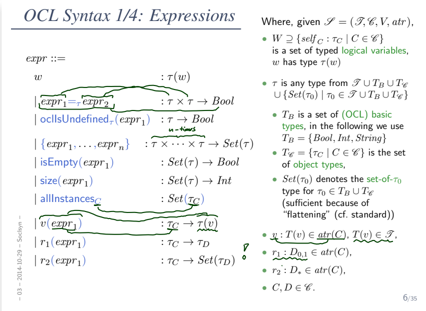
Last Lecture:

- OCL Syntax

This Lecture:

- **Educational Objectives:** Capabilities for these tasks/questions:
  - Please un-abbreviate all abbreviations in this OCL expression. ✓
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this system state?
  - Give a system state satisfying this constraint?
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are $\mathscr{D}(C)$ and T_C related?
- **Content:**
  - OCL Semantics
  - OCL Consistency and Satisfiability

Recall



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$_C$ $: Set(\tau_C)$

$\mid v(expr_1)$ $: \tau_C \to \tau(v)$

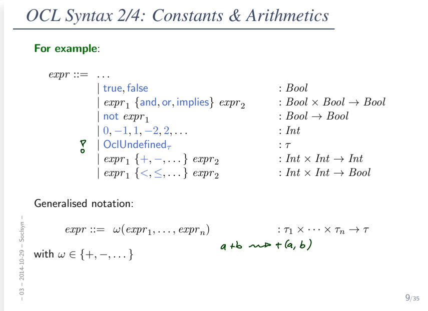
$\mid r_1(expr_1)$ $: \tau_C \to \tau_D$

$\mid r_2(expr_1)$ $: \tau_C \to Set(\tau_D)$

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

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

6/35



OCL Syntax 2/4: Constants & Arithmetics

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$_\tau$ $: \tau$

$\mid expr_1 \{+, -, \dots\}\ expr_2$ $: Int \times Int \to Int$

$\mid expr_1 \{<, \leq, \dots\}\ expr_2$ $: Int \times Int \to Bool$

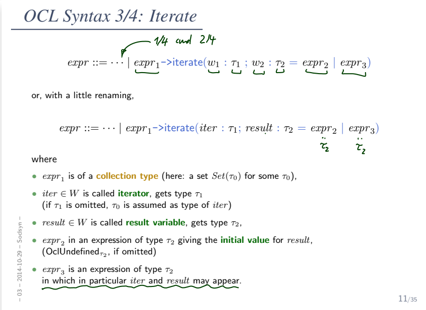
Generalised notation:

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

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

$a+b \rightsquigarrow +(a,b)$

9/35



OCL Syntax 3/4: Iterate

1/4 and 2/4

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

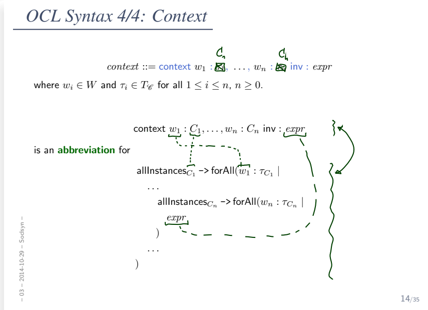
or, with a little renaming,

$expr ::= \dots \mid expr_1$->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),
- $iter \in W$ is called iterator, gets type τ_1 (if τ_1 is omitted, τ_0 is assumed as type of $iter$)
- $result \in W$ is called result variable, gets type τ_2,
- $expr_2$ in an expression of type τ_2 giving the initial value for $result$, (OclUndefined$_{\tau_2}$, if omitted)
- $expr_3$ is an expression of type τ_2 in which in particular $iter$ and $result$ may appear.

11/35



OCL Syntax 4/4: Context

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

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

context $w_1 : C_1, \dots, w_n : C_n$ inv : $expr$

is an abbreviation for

allInstances$_{C_1}$ -> forAll($w_1 : \tau_{C_1}$ |

...

allInstances$_{C_n}$ -> forAll($w_n : \tau_{C_n}$ |

$expr$

)

...

)

14/35

OCL Semantics: The Task

- Given an OCL expression $expr$ (over signature $\mathscr{S}$), e.g.

$$expr_1 = \text{context } DD \text{ inv} : wen \text{ implies } win > 0$$

- and a system state $\sigma \in \Sigma_\mathscr{S}^\mathscr{D}$, e.g.

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{win \mapsto 13, wen \mapsto \mathit{true}\},$$
$$3_{CP} \mapsto \{wen \mapsto \mathit{true}\}, \quad 5_{CP} \mapsto \{wen \mapsto \mathit{false}\}\}$$

- and a valuation of logical variables $\beta : W \to I(\mathscr{T} \cup T_B \cup T_\mathscr{C})$,

- **define** the **interpretation of** $expr$ **in** σ **under** β

$$I[\![\cdot]\!](\cdot, \cdot) : OCLExpressions(\mathscr{S}) \times \Sigma_\mathscr{S}^\mathscr{D} \times (W \to I(\mathscr{T} \cup T_B \cup T_\mathscr{C})) \to I(Bool)$$

i.e.

$$I[\![expr]\!](\sigma, \beta) \in \{\mathit{true}, \mathit{false}, \bot_{Bool}\}.$$

OCL Semantics OMG (2006)

Basically business as usual...

(i) Equip each OCL (!) **type** with a reasonable **domain**, i.e. define function

$$I_{(i)} \text{ with } \mathrm{dom}(I) = \mathscr{T} \cup T_B \cup T_{\mathscr{C}}$$

(ii) Equip each **set type** $Set(\tau_0)$ with reasonable **domain**, i.e. define function

$$I_{(ii)} \text{ with } \mathrm{dom}(I) = \{Set(\tau_0) \mid \tau_0 \in \mathscr{T} \cup T_B \cup T_{\mathscr{C}}\}$$

(iii) Equip each **arithmetical operation** with a reasonable **interpretation** (that is, with a **function** operating on the corresponding **domains**).

$$I_{(iii)} \text{ with } \mathrm{dom}(I) = \{+, -, \leq, \dots\}, \text{ e.g., } I_{(iii)}(+) \in I(Int) \times I(Int) \to I(Int)$$

(iv) **Set operations** similar: $I_{(iv)}$ with $\mathrm{dom}(I) = \{\text{isEmpty}, \dots\}$

(v) Equip each **expression** with a reasonable **interpretation**, i.e. define function

$$I_{(v)} : Expr \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I_{(i)}(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \to I_{(i)}(Bool)$$

Basically business as usual...

(i) Equip each OCL (!) **type** with a reasonable **domain**, i.e. define function

$$I \text{ with } \mathrm{dom}(I) = \mathscr{T} \cup T_B \cup T_{\mathscr{C}}$$

(ii) Equip each **set type** $Set(\tau_0)$ with reasonable **domain**, i.e. define function

$$I \text{ with } \mathrm{dom}(I) = \{Set(\tau_0) \mid \tau_0 \in \mathscr{T} \cup T_B \cup T_{\mathscr{C}}\}$$

(iii) Equip each **arithmetical operation** with a reasonable **interpretation** (that is, with a **function** operating on the corresponding **domains**).

$$I \text{ with } \mathrm{dom}(I) = \{+, -, \leq, \dots\}, \text{ e.g., } I(+) \in I(Int) \times I(Int) \to I(Int)$$

(iv) **Set operations** similar: I with $\mathrm{dom}(I) = \{\text{isEmpty}, \dots\}$

(v) Equip each **expression** with a reasonable **interpretation**, i.e. define function

$$I : Expr \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \to I(Bool)$$

...except for OCL being a **three-valued logic**, and the "iterate" expression.

(i) Domains of OCL and (!) Model Basic Types

Recall: **OCL basic types**

$$T_B = \{Bool, Int, String\}$$

We set:

- $I_{(i)}(Bool) := \{true, false, \bot_{Bool}\}$ three-valued
- $I_{(i)}(Int) := \mathbb{Z} \,\dot\cup\, \{\bot_{Int}\}$
- $I_{(i)}(String) := \dots \dot\cup \{\bot_{String}\}$ disjoint union

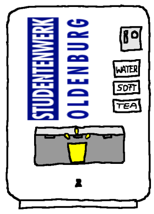
We may omit index τ of $\bot_\tau$ if it is clear from context.

Given signature $\mathscr{S}$ with **model basic types** $\mathscr{T}$ and domain $\mathscr{D}$, set

$$I(T) := \mathscr{D}(T) \,\dot\cup\, \{\bot_T\}$$

for each model basic type $T \in \mathscr{T}$.

OCL and Model Types?! An Example.



$$\begin{aligned}\mathscr{S} = (&\{Bool, Nat\}, \{VM, CP, DD\},\\ &\{cp : CP_*, dd : DD_{0,1}, wen : Bool, win : Nat\},\\ &\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen\}, DD \mapsto \{win, wen\})\end{aligned}$$

Model Types:

$\mathscr{D}_1(Bool_M) = \{0,1\}$

$\mathscr{D}_1(Nat) = \{0,..,255\}$

$\mathscr{D}_1(VM) = \{1_{VM}, 2_{VM}, ..\}$

OCL Types:

$I(Bool_{OCL}) = \{true, false, \bot_{Bool}\}$ } fixed

$I(Int) = \mathbb{Z} \,\dot\cup\, \{\bot_{Int}\}$ } for T_B

$I(Nat) = \{0,..,255\} \,\dot\cup\, \{\bot_{Nat}\}$

$I(\tau_{VM}) = \{1_{VM}, 2_{VM}, ..\} \,\dot\cup\, \{\bot_{\tau_{VM}}\}$

$I(Bool_M) = \{0,1\} \,\dot\cup\, \{\bot_{Bool_M}\}$

(i) Domains of Object and (ii) Set Types

- Let τ_C be an (OCL) **object type** for a class $C \in \mathscr{C}$.
- We set

$$I(\tau_C) := \mathscr{D}(C) \,\dot\cup\, \{\bot_{\tau_C}\}$$

- Let τ be a type from $\mathscr{T} \cup T_B \cup T_{\mathscr{C}}$.
- We set

$$I(Set(\tau)) := 2^{I(\tau)} \,\dot\cup\, \{\bot_{Set(\tau)}\}$$

Note: in the OCL standard, only **finite** subsets of $I(\tau)$.
But infinity doesn't scare **us**, so we simply allow it.

(iii) Interpretation of Arithmetic Operations

- **Literals** map to fixed values:

$$I(\underset{\in\,\mathit{OCLExpr}}{\mathsf{true}}) := \underset{\in\,I(Bool)}{\mathit{true}}, \quad I(\mathsf{false}) := \mathit{false}, \quad I(\overset{\mathit{OCLExpr}\,\ni}{0}) := \overset{I(Int)\,\ni}{0}, \quad I(1) := 1, \ldots$$

$$I(\underset{\in\,\mathit{OCLExpr}}{\mathsf{OclUndefined}_\tau}) := \underset{\in\,I(\tau)}{\bot_\tau}$$

(iii) Interpretation of Arithmetic Operations

- **Literals** map to fixed values:

$$I(\underset{\in\,\mathit{OCLExpr}}{\mathsf{true}}) := \underset{\in\,I(Bool)}{\mathit{true}}, \quad I(\mathsf{false}) := \mathit{false}, \quad I(\overset{\mathit{OCLExpr}\,\ni}{0}) := \overset{I(Int)\,\ni}{0}, \quad I(1) := 1, \ldots$$

$$I(\underset{\in\,\mathit{OCLExpr}}{\mathsf{OclUndefined}_\tau}) := \underset{\in\,I(\tau)}{\bot_\tau}$$

- **Boolean operations** (defined point-wise for $x_1, x_2 \in I(\tau)$):

$$I(=_\tau)(\overset{I(\tau)\,\ni}{x_1, x_2}) := \begin{cases} \overset{I(Bool)\,\ni}{\mathit{true}} & \text{, if } x_1 \neq \bot_\tau \neq x_2 \text{ and } x_1 = x_2 \\ \mathit{false} & \text{, if } x_1 \neq \bot_\tau \neq x_2 \text{ and } x_1 \neq x_2 \\ \bot_{Bool} & \text{, otherwise} \end{cases}$$

- **Integer operations** (defined point-wise for $x_1, x_2 \in I(\mathit{Int})$):

$$I(+)(x_1, x_2) := \begin{cases} x_1 + x_2 & \text{, if } x_1 \neq \bot \neq x_2 \\ \bot & \text{, otherwise} \end{cases}$$

but:
$I(\vee)(\mathit{false}, \bot) = \bot$
$I(\wedge)(\mathit{false}, \bot) = \mathit{false}$
and?

Note: There is a **common principle**.

The **interpretation** of an operation (symbol) $\omega : \tau_1 \times \ldots \tau_n \to \tau$
is a function $I(\omega) : I(\tau_1) \times \cdots \times I(\tau_n) \to I(\tau)$ on corresponding semantical domain(s).

(iii) Interpretation of OclIsUndefined

- The **is-undefined** predicate (defined point-wise for $x \in I(\tau)$):

$$I(\text{oclIsUndefined}_\tau)(x) := \begin{cases} \mathit{true} & \text{, if } x = \bot_\tau \\ \mathit{false} & \text{, otherwise} \end{cases}$$

Note: $I(\text{oclIsUndefined}_\tau)$ is **definite**, i.e., it never yields $\bot$.

(iv) Interpretation of Set Operations

Basically the same principle as with arithmetic operations...

Let $\tau \in \mathscr{T} \cup T_B \cup T_{\mathscr{C}}$.

- **Set comprehension** $(x_1, \dots, x_n \in I(\tau))$:

$$I(\{\}_n^\tau)(\underbrace{x_1}_{\in I(\tau)}, \dots, \underbrace{x_n}_{\in I(\tau)}) := \underbrace{\{x_1, \dots, x_n\}}_{\in I(Set(\tau))}$$

  for all $n \in \mathbb{N}_0$

- **Empty-ness check** $(x \in I(Set(\tau)))$:

$$I(\text{isEmpty}^\tau)(x) := \begin{cases} \mathit{true} & \text{, if } x = \emptyset \\ \bot_{Bool} & \text{, if } x = \bot_{Set(\tau)} \\ \mathit{false} & \text{, otherwise} \end{cases}$$

- **Counting** $(x \in I(Set(\tau)))$:

$$I(\text{size}^\tau)(x) := \begin{cases} \overbrace{|x|}^{\text{number of elements in } x} & \text{, if } x \neq \bot_{Set(\tau)} \text{ and } x \text{ finite} \\ \bot_{Int} & \text{, otherwise} \end{cases}$$

(v) Putting It All Together

OCL Syntax 1/4: Expressions

$$
\begin{array}{ll}
expr ::= & \\
\quad w & : \tau(w) \\
\quad \mid expr_1 =_\tau expr_2 & : \tau \times \tau \to Bool \\
\quad \mid \text{oclIsUndefined}_\tau(expr_1) & : \tau \to Bool \\
\quad \mid \{expr_1, \dots, expr_n\} & : \overbrace{\tau \times \dots \times \tau}^{n\text{-times}} \to Set(\tau) \\
\quad \mid \text{isEmpty}(expr_1) & : Set(\tau) \to Bool \\
\quad \mid \text{size}(expr_1) & : Set(\tau) \to Int \\
\quad \mid \text{allInstances}_C & : Set(\tau_C) \\
\quad \mid v(expr_1) & : \tau_C \to \tau(v) \\
\quad \mid r_1(expr_1) & : \tau_C \to \tau_D \\
\quad \mid r_2(expr_1) & : \tau_C \to Set(\tau_D)
\end{array}
$$

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

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

– 03 – 2014-10-29 – Soclsyn – 6/35

OCL Syntax 2/4: Constants & Arithmetics

For example:

$$
\begin{array}{ll}
expr ::= \dots & \\
\quad \mid \text{true}, \text{false} & : Bool \\
\quad \mid expr_1 \ \{\text{and}, \text{or}, \text{implies}\} \ expr_2 & : Bool \times Bool \to Bool \\
\quad \mid \text{not} \ expr_1 & : Bool \to Bool \\
\quad \mid 0, -1, 1, -2, 2, \dots & : Int \\
\quad \mid \text{OclUndefined}_\tau & : \tau \\
\quad \mid expr_1 \ \{+, -, \dots\} \ expr_2 & : Int \times Int \to Int \\
\quad \mid expr_1 \ \{<, \leq, \dots\} \ expr_2 & : Int \times Int \to Bool
\end{array}
$$

Generalised notation:

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

with $\omega \in \{+, -, \dots\}$ $\qquad a + b \rightsquigarrow +(a, b)$

– 03 – 2014-10-29 – Soclsyn – 9/35

OCL Syntax 3/4: Iterate

1/4 and 2/4

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

or, with a little renaming,

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

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some τ_0),
- $iter \in W$ is called **iterator**, gets type τ_1 (if τ_1 is omitted, τ_0 is assumed as type of $iter$)
- $result \in W$ is called **result variable**, gets type τ_2,
- $expr_2$ in an expression of type τ_2 giving the **initial value** for $result$, ($\text{OclUndefined}_{\tau_2}$, if omitted)
- $expr_3$ is an expression of type τ_2 in which in particular $iter$ and $result$ may appear.

– 03 – 2014-10-29 – Soclsyn – 11/35

OCL Syntax 4/4: Context

$$context ::= \text{context} \ w_1 : \overset{C_1}{\tau_1}, \dots, w_n : \overset{C_n}{\tau_n} \ \text{inv} : expr$$

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

context $w_1 : C_1, \dots, w_n : C_n$ inv : $expr$

is an **abbreviation** for

allInstances$_{C_1}$ -> forAll($w_1 : \tau_{C_1}$ |
  ...
    allInstances$_{C_n}$ -> forAll($w_n : \tau_{C_n}$ |
      $expr$
)
  ...
)

– 03 – 2014-10-29 – Soclsyn – 14/35

Valuations of Logical Variables

$\{self_C \mid C \in \mathscr{C}\}$

- **Recall**: we have typed logical variables $(w \in)\ W$, $\tau(w)$ is the type of w.
- By β, we denote a valuation of the logical variables, i.e. for each $w \in W$,

$$\beta(w) \in I(\tau(w)).$$

$$\beta : W \to I(T_B \cup T_\mathscr{C} \cup \mathscr{T})$$

$$I(\tau(result)) = I(Bool)$$

$$\beta = \{ result \mapsto true, \quad self_{VM} \mapsto 27_{VM} \}$$

$$I(\tau(self_{VM})) = I(\tau_{VM})$$

(v) Putting It All Together...

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

double-u

Σ_φ^Ω

$\in$

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

  e.g. $self_{VM}$ omega

- $I[\![\omega(expr_1, \ldots, expr_n)]\!](\sigma, \beta) := I(\omega)(I[\![expr_1]\!](\sigma,\beta), \ldots, I[\![expr_n]\!](\sigma,\beta))$

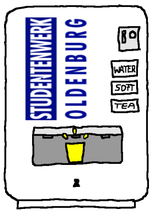
  e.g. $+(13, result)$ e.g. $I(+)(I[\![13]\!](\sigma,\beta), I[\![result]\!](\sigma,\beta))$

- $I[\![\mathsf{allInstances}_C]\!](\sigma, \beta) := \underbrace{dom(\sigma)}_{\text{all alive objects in } \sigma} \cap \underbrace{\mathscr{D}(C)}_{\text{objects of class } C}$

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

Example

$\beta = \{ self_{VM} \mapsto 7_{VM}, x \mapsto 27 \}$



$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, win : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen\}, DD \mapsto \{win, wen\})$$

$$\sigma_1 = \{\underline{7_{VM}} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad \underline{1_{DD}} \mapsto \{win \mapsto 13, wen \mapsto \mathit{true}\},$$
$$\underline{3_{CP}} \mapsto \{wen \mapsto \mathit{true}\}, \quad \underline{5_{CP}} \mapsto \{wen \mapsto \mathit{false}\}\}$$

- $I[\![w]\!](\sigma, \beta) := \beta(w)$ (1)
- $I[\![\mathsf{allInstances}_C]\!](\sigma, \beta) := \mathrm{dom}(\sigma) \cap \mathscr{D}(C)$ (2)
- $I[\![\omega(expr_1, \ldots, expr_n)]\!](\sigma, \beta) := I(\omega)(I[\![expr_1]\!](\sigma, \beta), \ldots I[\![expr_n]\!](\sigma, \beta))$ (3)

- $I[\![self_{VM}]\!](\sigma_1, \beta) \overset{(1)}{=} \beta(self_{VM}) = 7_{VM}$
- $I[\![x]\!](\sigma_1, \beta) \overset{(1)}{=} \beta(x) = 27$ (*)
- $I[\![allInstances_{CP}]\!](\sigma_1, \beta) \overset{(2)}{=} dom(\sigma_1) \cap \mathscr{D}(CP)$

  $= \{7_{VM}, 1_{DD}, 3_{CP}, 5_{CP}\} \cap \{1_{CP}, 2_{CP}, \ldots\} = \underbrace{\{3_{CP}, 5_{CP}\}}_{(**)}$

- $I[\![x + allInstances_{CP} \to size]\!](\sigma_1, \beta) = 29$

  (3) $(I[\![+(x, size(allInstances_{CP}))]\!](\sigma_1, \beta))$

  $I(+)(I[\![x]\!](\sigma_1, \beta),$
  $I[\![allInstances_{CP} \to size]\!](\sigma_1, \beta))$

  $\overset{(*)}{=} I(+)(27, I(size)(I[\![allInstances_{CP}]\!](\sigma_1, \beta)))$

  $\overset{(**)}{=} I(+)(27, I(size)(\{3_{CP}, 5_{CP}\}))$

  $= I(+)(27, 2) = 29$ ✓

(v) Putting It All Together...

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

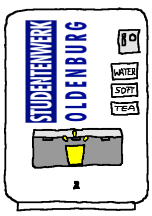
Assume $expr_1 : \tau_C$ for some $C \in \mathscr{C}$. Set $u_1 := I[\![expr_1]\!](\sigma, \beta) \in \mathscr{D}(\tau_C)$.

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

  $\mathcal{E}_{att}$
  $v : T,$
  $T \in \mathscr{T}$

Example

$\beta = \{self_{VM} \mapsto 7_{VM}, self_{CP} \mapsto 3_{CP}, y \mapsto 1_{CP}\}$



$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$
$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, win : Nat\},$
$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen\}, DD \mapsto \{win, wen\})$

$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{win \mapsto 13, wen \mapsto \cancel{true}\ 1\},$
$3_{CP} \mapsto \{wen \mapsto \cancel{true}\ 1\}, \quad 5_{CP} \mapsto \{wen \mapsto \cancel{false}\ 0\}\}$

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

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

- $I[\![wen(\overbrace{self_{CP}}^{expr_1})]\!](\sigma_1, \beta) = (\sigma_1(u_1))(wen) = 1$

  $u_1 = I[\![self_{CP}]\!](\sigma_1, \beta) = 3_{CP}$ alive!

- $I[\![wen(y)]\!](\sigma_1, \beta) = \perp$

  $I[\![y]\!](\sigma_1, \beta) = 1_{CP}$ not alive!

(v) Putting It All Together...

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

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

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

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

  $r_1 : D_{0,1}$

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

  $r_2 : D_*$

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

(v) Putting It All Together...

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

- $I[\![expr_1 \texttt{->iterate}(v_1 : \tau_1 \,;\, v_2 : \tau_2 = expr_2 \mid expr_3)]\!](\sigma,\beta)$

  $$:= \begin{cases} \bot_{\tau_2} & \text{, if } I[\![expr_1]\!](\sigma,\beta) = \bot_{\tau_1} \\ I[\![expr_2]\!](\sigma,\beta) & \text{, if } I[\![expr_1]\!](\sigma,\beta) = \emptyset \\ iterate(hlp, v_1, v_2, expr_3, \sigma, \beta') & \text{, otherwise} \end{cases}$$

  where $\beta' = \beta[hlp \mapsto I[\![expr_1]\!](\sigma,\beta), v_2 \mapsto I[\![expr_2]\!](\sigma,\beta)]$ and

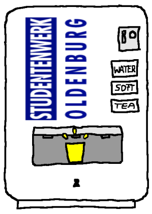
- $iterate(hlp, v_1, v_2, expr_3, \sigma, \beta')$

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

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

Quiz: Is (our) I a function?

Example



$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, win : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen\}, DD \mapsto \{win, wen\})$$

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{win \mapsto 13, wen \mapsto \underline{\textsf{true}}\},$$
$$3_{CP} \mapsto \{wen \mapsto \underline{\textsf{true}}\}, \quad 5_{CP} \mapsto \{wen \mapsto \underline{\textsf{false}}\}\}$$

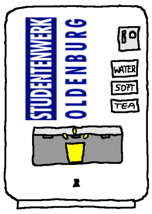
expr = context DD inv : wen implies $win > 0$

$I[\![$ allInstances$_{DD}$ -> forAll (self$_{DD}$: τ_{DD} | implies (wen(self$_{DD}$), >(win(self$_{DD}$), 0)) $]\!](\sigma, \emptyset)$

iterate (… ; result = true | result and …)

= true

Another Example



$$\mathscr{S} = (\{Bool, Nat\}, \{VM, CP, DD\},$$
$$\{cp : CP_*, dd : DD_{0,1}, wen : Bool, win : Nat\},$$
$$\{VM \mapsto \{cp, dd\}, CP \mapsto \{wen\}, DD \mapsto \{win, wen\})$$

$$\sigma_1 = \{7_{VM} \mapsto \{dd \mapsto \{1_{DD}\}, cp \mapsto \{3_{DD}, 5_{DD}\}\}, \quad 1_{DD} \mapsto \{win \mapsto 13, wen \mapsto \textsf{true}\},$$
$$3_{CP} \mapsto \{wen \mapsto \textsf{true}\}, \quad 5_{CP} \mapsto \{wen \mapsto \textsf{false}\}\}$$

$I[\![$ context VM inv : cp -> forAll (c | wen(c) = true)
or cp -> forAll (c | wen(c) = false) $]\!](\sigma_1, \emptyset)$ = false

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

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

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