Software Design, Modelling and Analysis in UML

Lecture 03: Object Constraint Language

2016-10-27

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

Albert-Ludwigs-Universität Freiburg, Germany

Content

- The **Object Constraint Language** (OCL):

  Syntax

  - Running Example
  - Overview
  - Expressions
    - Notational Conventions
      ("." (OCL-Dot) and "->" (OCL-Arrow))
  - Constants & Arithmetics
  - Iterate
  - Context
    - More Notational Conventions
  - The Running Example Revisited

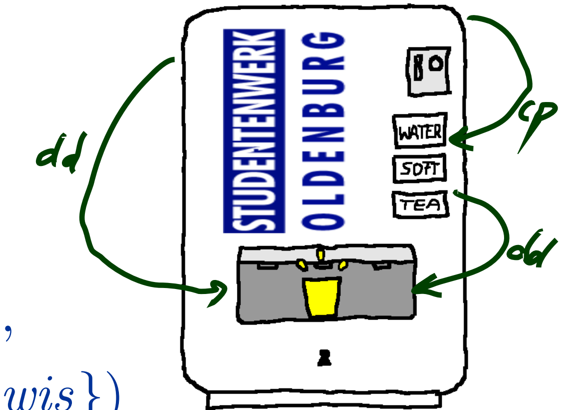
- **"Not Interesting"**

(Core) OCL Syntax OMG (2006)

Overview

$$
\begin{array}{rll}
expr ::= & w & : \tau(w) \\
& \mid expr_1 =_\tau expr_2 & : \tau \times \tau \to Bool \\
& \mid \mathsf{oclIsUndefined}_\tau(expr_1) & : \tau \to Bool \\
& \mid \{expr_1, \dots, expr_n\} & : \tau \times \dots \times \tau \to Set(\tau) \\
& \mid \mathsf{size}(expr_1) & : Set(\tau) \to Int \\
& \mid \mathsf{allInstances}_C & : Set(\tau_C) \\
& \mid v(expr_1) & : \tau_C \to \tau(v) \\
& \mid r_1(expr_1) & : \tau_C \to \tau_D \\
& \mid r_2(expr_1) & : \tau_C \to Set(\tau_D) \\
& \mid \mathsf{true}, \mathsf{false} & : Bool \\
& \mid \mathsf{not}\ expr_1 & : Bool \to Bool \\
& \mid expr_1\ \{\mathsf{and}, \mathsf{or}, \mathsf{implies}\}\ expr_2 & : Bool \times Bool \to Bool \\
& \mid \dots & \\
& \mid \mathsf{OclUndefined}_\tau & : \tau \\
& \mid expr_1 \mathsf{->iterate}(w_1 : T_1 ; w_2 : T_2 = expr_2 \mid expr_3) & : Set(\tau_0) \to \tau_{T_2} \\
context ::= & \mathsf{context}\ w_1 : T_1, \dots, w_n : T_n\ \mathsf{inv} : expr & : Bool
\end{array}
$$

Recall: Vending Machine Structure



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

Claim: this is a proper OCL constraint over $\mathscr{S}$:

$$\text{context} \quad CP \quad \text{inv}: \quad wen \quad \text{implies} \quad dd\,.\,wis > 0$$

String $\tau \to Int$ $\tau \times \tau \to Bool$

$T ::= \clubsuit \mid \Box(T) \mid \heartsuit(T_1, T_2)$

$\clubsuit$, ✓

$\Box(\clubsuit) : \overset{Int}{\cancel{Bool}}$ ✓

$\heartsuit(\clubsuit, \clubsuit) : Bool$ ✓

⋮

$\heartsuit(\underset{String}{\clubsuit}, \overset{Int}{\Box(\clubsuit)})$ ✗ (not well-typed)

Plan

1/4

$$
\begin{array}{rll}
expr ::= & w & : \tau(w) \\
 & \mid expr_1 =_\tau expr_2 & : \tau \times \tau \to Bool \\
 & \mid \mathsf{oclIsUndefined}_\tau(expr_1) & : \tau \to Bool \\
 & \mid \{expr_1, \dots, expr_n\} & : \tau \times \dots \times \tau \to Set(\tau) \\
 & \mid \mathsf{size}(expr_1) & : Set(\tau) \to Int \\
 & \mid \mathsf{allInstances}_C & : Set(\tau_C) \\
 & \mid v(expr_1) & : \tau_C \to \tau(v) \\
 & \mid r_1(expr_1) & : \tau_C \to \tau_D \\
 & \mid r_2(expr_1) & : \tau_C \to Set(\tau_D)
\end{array}
$$

2/4

$$
\begin{array}{ll}
\mid \mathsf{true}, \mathsf{false} & : Bool \\
\mid \mathsf{not}\ expr_1 & : Bool \to Bool \\
\mid expr_1 \ \{\mathsf{and}, \mathsf{or}, \mathsf{implies}\}\ expr_2 & : Bool \times Bool \to Bool \\
\mid \dots & \\
\mid \mathsf{OclUndefined}_\tau & : \tau
\end{array}
$$

3/4

$$
\mid expr_1 \mathsf{->iterate}(w_1 : T_1 ; w_2 : T_2 = expr_2 \mid expr_3) \qquad : Set(\tau_0) \to \tau_{T_2}
$$

4/4

$$
context ::= \mathsf{context}\ w_1 : T_1, \dots, w_n : T_n\ \mathsf{inv} : expr \qquad : Bool
$$

$expr ::=$

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$_C$	$: Set(\tau_C)$		
$\mid v(expr_1)$	$: \tau_C \to \tau$	where	$v : \tau \in atr(C),\ \tau \in \mathscr{T}$,
$\mid r_1(expr_1)$	$: \tau_C \to \tau_D$	where	$r_1 : D_{0,1} \in atr(C), C, D \in \mathscr{C}$,
$\mid r_2(expr_1)$	$: \tau_C \to Set(\tau_D)$	where	$r_2 : D_* \in atr(C), C, D \in \mathscr{C}$.

T_B

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

- $w \in W \supseteq \{self_C : \tau_C \mid C \in \mathscr{C}\}$ 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)).

Expression Examples

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathit{expr} ::=} \\
\quad w & : \tau(w) & \mid \mathsf{size}(\mathit{expr}_1) & : Set(\tau) \to Int \\
\quad \mid \mathit{expr}_1 =_\tau \mathit{expr}_2 & : \tau \times \tau \to Bool & \mid \mathsf{allInstances}_C & : Set(\tau_C) \\
\quad \mid \mathsf{oclIsUndefined}_\tau(\mathit{expr}_1) & : \tau \to Bool & \mid v(\mathit{expr}_1) & : \tau_C \to \tau(v) \\
\quad \mid \{\mathit{expr}_1, \dots, \mathit{expr}_n\} & : \tau \times \dots \times \tau \to Set(\tau) & \mid r_1(\mathit{expr}_1) & : \tau_C \to \tau_D \\
\quad \mid \mathsf{isEmpty}(\mathit{expr}_1) & : Set(\tau) \to Bool & \mid r_2(\mathit{expr}_1) & : \tau_C \to Set(\tau_D)
\end{array}
$$

$$\mathscr{S}_0 = (\{Int_0\}, \{\bar{C}, \bar{D}\}, \{x : Int_0, p : \bar{C}_{0,1}, n : \bar{C}_*\}, \{\bar{C} \mapsto \{p, n\}, \bar{D} \mapsto \{x\}\})$$

- $self_{\bar{D}} : \tau_{\bar{D}}$ ✓
- $x(self_{\bar{D}}) : Int_0$ ✓
- $p(self_{\bar{D}})$ ✗ $p \notin atr(\bar{D})$
- $p(self_{\bar{C}}) : \tau_{\bar{C}} \to \tau_{\bar{C}}$ ✓
- $n(self_{\bar{C}}) : \tau_{\bar{C}} \to Set(\tau_{\bar{C}})$ ✓
- $n(\underbrace{p(self_{\bar{C}})}_{:\tau_{\bar{C}}}) : \tau_{\bar{C}} \to Set(\tau_{\bar{C}})$ ✓ $[self_{\bar{C}}.p.n]$

- $p(\underbrace{n(self_{\bar{C}})}_{\tau_{\bar{C}} \to Set(\tau_{\bar{C}})})$ ↯ $[self_{\bar{C}}.n \overset{↯}{.} p]$
- $size(n(self_{\bar{C}})) : Set(\tau_{\bar{C}}) \to Int$

Expression Examples

$$
\begin{array}{llll}
\multicolumn{2}{l}{expr ::=} & & \\
\quad w & : \tau(w) & \mid \mathsf{size}(expr_1) & : Set(\tau) \to Int \\
\quad \mid expr_1 =_\tau expr_2 & : \tau \times \tau \to Bool & \mid \mathsf{allInstances}_C & : Set(\tau_C) \\
\quad \mid \mathsf{oclIsUndefined}_\tau(expr_1) & : \tau \to Bool & \mid v(expr_1) & : \tau_C \to \tau(v) \\
\quad \mid \{expr_1, \dots, expr_n\} & : \tau \times \cdots \times \tau \to Set(\tau) & \mid r_1(expr_1) & : \tau_C \to \tau_D \\
\quad \mid \mathsf{isEmpty}(expr_1) & : Set(\tau) \to Bool & \mid r_2(expr_1) & : \tau_C \to Set(\tau_D)
\end{array}
$$

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\})$$

$\mathsf{context}\ \ CP\ \ \mathsf{inv} :\ \ wen\ \ \mathsf{implies}\ \ dd\,.\,wis > 0$

Notational Conventions for Expressions

- **Each expression**

$$\omega(expr_1, expr_2, \dots, expr_n) : \tau_1 \times \dots \times \tau_n \to \tau$$

!

may **alternatively** be written ("**abbreviated as**")

- $expr_1\ .\ \omega(expr_2, \dots, expr_n)$ if τ_1 is an **object type**, i.e. if $\tau_1 \in T_{\mathscr{C}}$.
- $expr_1$ -> $\omega(expr_2, \dots, expr_n)$ if τ_1 is a **collection type** (here: only sets), i.e. if $\tau_1 = Set(\tau_0)$ for some $\tau_0 \in T_B \cup T_{\mathscr{C}}$.

- **Examples:** $(\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\})$
  - $\overbrace{self_C}^{expr_1} . \overbrace{p}^{\omega}$ $\leftrightsquigarrow$ $p(self_C)$
  - $self_C . p . n$ $\leftrightsquigarrow$ $n(p(self_C))$
  - $\underbrace{self_C . p . n}_{expr_1}$ -> $\underbrace{\text{isEmpty}}_{\omega}$ $\leftrightsquigarrow$ $\text{isEmpty}(n(p(self_C)))$
  - context CP inv : wen implies $dd . wis > 0$ $\quad (atr(CP) = \{wen : Bool, dd : DD_{0,1}\})$

    $wis(dd)$

OCL Syntax 2/4: Constants & Arithmetics

For example:

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

Generalised notation: (prefix normal form)

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

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

$$1 + 2 \rightsquigarrow \underbrace{+}_{\omega}(\underbrace{1}_{expr_1}, \underbrace{2}_{expr_2})$$

Constants & Arithmetics Examples

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

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\})$$

implies(wen, >(wis(dd), 0))

$>(\underbrace{wis(dd)}_{expr_1}, \underbrace{0}_{expr_2})$

wis(dd)

context CP inv : wen implies $dd\ .\ wis > 0$

OCL Syntax 3/4: Iterate

$$expr ::= \cdots \mid expr_1\texttt{->iterate}(w_1 : T_1 \,;\, w_2 : T_2 = expr_2 \mid expr_3)$$

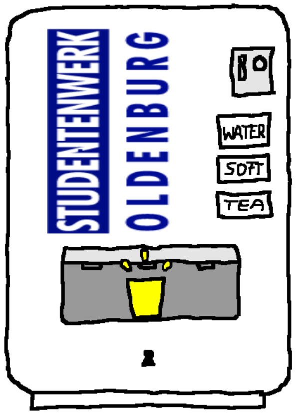
or, with a little renaming,

$$expr ::= \cdots \mid expr_1\texttt{->iterate}(iter : T_1; result : T_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some τ_0),
- $iter \in W$ is called **iterator**, of the type denoted by T_1
  (if T_1 is omitted, τ_0 is assumed as type of $iter$)
  $\neq$ iter
- $result \in W$ is called **result variable**, gets type τ_2 denoted by T_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 particular $iter$ and $result$ may appear in $expr_3$.

Iterate Example



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

$$\boxed{expr ::= expr_1 \texttt{->iterate}(w_1 : T_1; w_2 : T_2 = expr_2 \mid expr_3)}$$

- $\underbrace{cp(self_{vm})}_{:Set(\tau_{cp})}$ ->iterate($\underbrace{iter}_{:\tau_{cp}}$; $res : Int = 0 \mid res + \underbrace{iter.dd.wis}_{:Nat \in T_{\mathscr{S}}}$)
- $cp(self_{vm})$ -> iterate(iter ; $res : Bool = true \mid res$ and $iter.wen$)

Abbreviations on Top of Iterate

$$expr ::= expr_1\text{->iterate}(w_1 : T_1;\ w_2 : T_2 = expr_2 \mid expr_3)$$

- $expr_1\text{->forAll}(w_1 : T_1 \mid expr_3)$ ("$\forall\, w_1 \in expr_1 \bullet expr_3$")

  is an abbreviation for

  $$expr_1\text{->iterate}(w_1 : T_1;\ w_2 : Bool = \text{true} \mid w_2 \ \text{and}\ expr_3).$$

- $expr_1\text{->Exists}(w : T_1 \mid expr_3)$

  is an abbreviation for

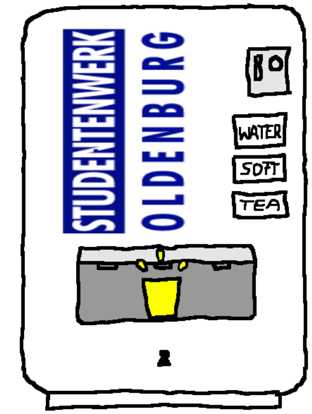
  $$expr_1 \text{->} \text{iterate}(w : T_1 ;\ w_2 : Bool = \text{false} \mid w_2 \ \text{or}\ expr_3)$$

To ensure confusion, we may again omit all kinds of things, cf. OMG (2006).

Recall: Overview

$$
\begin{array}{rlll}
expr ::= & w & & : \tau(w) \\
& | \; expr_1 =_\tau expr_2 & & : \tau \times \tau \to Bool \\
& | \; \mathsf{oclIsUndefined}_\tau(expr_1) & & : \tau \to Bool \\
& | \; \{expr_1, \dots, expr_n\} & & : \tau \times \dots \times \tau \to Set(\tau) \\
& | \; \mathsf{size}(expr_1) & & : Set(\tau) \to Int \\
& | \; \mathsf{allInstances}_C & & : Set(\tau_C) \\
& | \; v(expr_1) & & : \tau_C \to \tau(v) \\
& | \; r_1(expr_1) & & : \tau_C \to \tau_D \\
& | \; r_2(expr_1) & & : \tau_C \to Set(\tau_D) \\
& | \; \mathsf{true}, \mathsf{false} & & : Bool \\
& | \; \mathsf{not}\ expr_1 & & : Bool \to Bool \\
& | \; expr_1\ \{\mathsf{and}, \mathsf{or}, \mathsf{implies}\}\ expr_2 & & : Bool \times Bool \to Bool \\
& | \; \dots & & \\
& | \; \mathsf{OclUndefined}_\tau & & : \tau \\
& | \; expr_1 \mathsf{\text{->}iterate}(w_1 : T_1 \,;\, w_2 : T_2 = expr_2 \mid expr_3) & & : Set(\tau_0) \to \tau_{T_2} \\
context ::= & \mathsf{context}\ w_1 : T_1, \dots, w_n : T_n\ \mathsf{inv} : expr & & : Bool
\end{array}
$$

More Iterate Examples



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

$$expr ::= expr_1 \texttt{->iterate}(w_1 : T_1; w_2 : T_2 = expr_2 \mid expr_3)$$

$allInstances_{CP}$ -> iterate ($self_{CP}$: CP ; res : Bool = true | res and ($self_{CP}$. wen implies $self_{CP}$. dd . wis > 0))

or:

$allInstances_{CP}$ -> forAll ($self_{CP}$ | $self_{CP}$. wen implies $self_{CP}$. dd . wis > 0)

context CP inv : wen implies dd . wis > 0

OCL Syntax 4/4: Context

Syntax: (Assuming signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.)

$$context ::= \textsf{context}\ w_1 : T_1,\ \dots,\ w_n : T_n\ \textsf{inv} : expr$$

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

Semantics:

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

is (just) an **abbreviation** for

$$
\begin{array}{l}
\textsf{allInstances}_{C_1} \textsf{ -> forAll}(w_1 : \tau_{C_1} \mid \\
\quad \dots \\
\qquad \textsf{allInstances}_{C_n} \textsf{ -> forAll}(w_n : \tau_{C_n} \mid \\
\qquad\quad expr \\
\qquad) \\
\quad \dots \\
)
\end{array}
$$

Context: More Notational Conventions

- For

$$\text{context } \mathit{self} : T \text{ inv} : \mathit{expr}$$

we may **alternatively** write ("**abbreviate as**")

$$\text{context } T \text{ inv} : \mathit{expr}$$

- **Within** the latter abbreviation, we may omit the "self" in expression expr, i.e. for

$$\text{context } T \text{ inv} : \mathit{self}.v$$

(which is an abbreviation for $\quad \text{context } T \text{ inv} : v(\mathit{self}) \quad$)

we may alternatively write ("**abbreviate as**")

$$\text{context } T \text{ inv} : v$$

The Running Example

context CP inv : wen implies $dd \,.\, wis > 0$

$\rightsquigarrow$

context self : CP inv : wen implies dd.wis > 0

$\rightsquigarrow$

context self : CP inv : self.wen implies self.dd.wis > 0

$\rightsquigarrow$

allInstances_{CP} -> forAll(self | self.wen implies self.dd.wis > 0)

$\rightsquigarrow$

– .. — iterate(....)

Recall: Overview

$$
\begin{array}{rll}
expr ::= & w & : \tau(w) \\
& \mid expr_1 =_\tau expr_2 & : \tau \times \tau \to Bool \\
& \mid \mathsf{oclIsUndefined}_\tau(expr_1) & : \tau \to Bool \\
& \mid \{expr_1, \dots, expr_n\} & : \tau \times \dots \times \tau \to Set(\tau) \\
& \mid \mathsf{size}(expr_1) & : Set(\tau) \to Int \\
& \mid \mathsf{allInstances}_C & : Set(\tau_C) \\
& \mid v(expr_1) & : \tau_C \to \tau(v) \\
& \mid r_1(expr_1) & : \tau_C \to \tau_D \\
& \mid r_2(expr_1) & : \tau_C \to Set(\tau_D) \\
& \mid \mathsf{true}, \mathsf{false} & : Bool \\
& \mid \mathsf{not}\ expr_1 & : Bool \to Bool \\
& \mid expr_1\ \{\mathsf{and}, \mathsf{or}, \mathsf{implies}\}\ expr_2 & : Bool \times Bool \to Bool \\
& \mid \dots & \\
& \mid \mathsf{OclUndefined}_\tau & : \tau \\
& \mid expr_1 \texttt{->}\mathsf{iterate}(w_1 : T_1 ; w_2 : T_2 = expr_2 \mid expr_3) & : Set(\tau_0) \to \tau_{T_2} \\
context ::= & \mathsf{context}\ w_1 : T_1, \dots, w_n : T_n\ \mathsf{inv} : expr & : Bool
\end{array}
$$

“Not Interesting”

Among others:

- Enumeration types
- Type hierarchy
- Complete list of arithmetical operators
- The two other collection types Bag and Sequence
- Casting
- Runtime type information
- Pre/post conditions
  (maybe later, when we officially know what an operation is)
- … context f pre: $expr_1$,
  post: $expr_2$

References

References

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

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.

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