Software Design, Modelling and Analysis in UML

Lecture 03: Object Constraint Language

2015-10-29

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

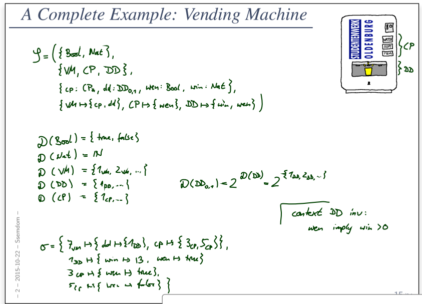
Last Lecture:

- Basic Object System Signature $\mathscr{S}$ and Structure $\mathscr{D}$, System State $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$

This Lecture:

- **Educational Objectives:** Capabilities for these tasks/questions:
  - 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?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are $\mathscr{D}(C)$ and T_C related?
- **Content:**
  - OCL Syntax
  - OCL Semantics (over system states)

Recall. . .



context DD inv : wen implies $win > 0$

(Core) OCL Syntax OMG (2006)

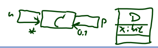
OCL Syntax 1/4: Expressions

$expr ::=$	
w	$: \tau(w)$
$\mid expr_1 =_\tau expr_2$	$: \tau \times \tau \to Bool$
$\mid$ oclIsUndefined$_\tau(expr_1)$	$: \tau \to Bool$
$\mid \{expr_1, \dots, expr_n\}$	$: \underbrace{\tau \times \dots \times \tau}_{n\text{-times}} \to Set(\tau)$
$\mid$ 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))
- $v : T(v) \in atr(C)$, $T(v) \in \mathscr{T}$,
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,
- $C, D \in \mathscr{C}$.

Expression Examples



$expr ::=$			
① w	$: \tau(w)$	| size$(expr_1)$	$: Set(\tau) \to Int$ ⑥
② | $expr_1 =_\tau expr_2$	$: \tau \times \tau \to Bool$	| allInstances$_C$	$: Set(\tau_C)$ ⑦
③ | oclIsUndefined$_\tau(expr_1)$	$: \tau \to Bool$	| $v(expr_1)$	$: \tau_C \to \tau(v)$ ⑧
④ | $\{expr_1, \dots, expr_n\}$	$: \tau \times \dots \times \tau \to Set(\tau)$	| $r_1(expr_1)$	$: \tau_C \to \tau_D$ ⑨
⑤ | isEmpty$(expr_1)$	$: Set(\tau) \to Bool$	| $r_2(expr_1)$	$: \tau_C \to Set(\tau_D)$ ⑩

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

- $self_C : \tau_C$ by ①
- allInstances$_{Int}$ NO ($Int \notin \mathscr{C}$)
- allInstances$_D$: Set(τ_D) ⑦
- size(allInstances$_D$) : Int ⑥
- allInstances$_C$ = allInstances$_D$ ↯ ② (same type on both sides)
- $x(self_C)$ ↯ ⑧ $x \notin atr(C)$
- $p(self_C) : \tau_C$ by ⑨, $p : C_{0,1}$
- $n(self_C)$: Set(τ_C) by ⑩
- $x(self_D)$: Int by ⑧
- $x(self_D)$ = size($n(self_C)$)

Expression Examples

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

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

~ win(·)?

context DD inv : wen implies $win > 0$

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 $\tau_0 \in T_B \cup T_\mathscr{C}$.

size(allInstances$_C$) ~ allInstances$_C$ -> size

$x(self_D)$ ⤳ $self_D . x$

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:** ($self : \tau_C \in W$; $v, w : Int \in V$; $r_1 : \underbrace{D_{0,1}}_{\mathscr{T}}, r_2 : \underbrace{D_*}_{\mathscr{T}} \in V$)
  - $self . v$ ⤳ $v(self)$
  - $self . r_1 . w$ ⤳ $w(r_1(\underbrace{self}_{\tau_C}))$, τ_D
  - $self . r_2$ -> isEmpty ⤳ isEmpty(r_2(self))

OCL Syntax 2/4: Constants & Arithmetics

For example:

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

Generalised notation:

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

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

a + b ⤳ + (a, b)

Constants & Arithmetics Examples

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

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

$self_D . x > self_D . x - 1$

$>(\underbrace{x(self_D)}_{:Int}, \underbrace{-(\underbrace{x(self_D)}_{:Int}, \underbrace{1}_{Int})}_{:Int})$

context DD inv : $\underbrace{wen}_{:Bool}$ implies $\underbrace{\overbrace{win}^{:Int} \geq 0}_{:Bool}$

OCL Syntax 3/4: Iterate

1/4 and 2/4

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

or, with a little renaming,

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

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.

Iterate: Intuitive Semantics (Formally: later)

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

$Set(\tau_0)$ $hlp = expr_1$;
τ_1 $iter$;
τ_2 $result = expr_2$;
while $(!hlp.empty())$ *do*
  $iter = hlp.pop()$;
  $result = expr_3$;
od

Note: In our (simplified) setting, we always have $expr_1 : Set(\tau_1)$ and $\tau_0 = \tau_1$.
In the type hierarchy of full OCL with inheritance and `oclAny`, they may be different and still type consistent.

Abbreviations on Top of Iterate

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

- $expr_1\text{->forAll}(w_1 : \tau_1 \mid expr_3)$ $w_1 \in W$, $result : Bool \in W$

$expr_1 \text{->iterate}(w_1 : \tau_1 ; result : Bool = true \mid result \text{ and } expr_3)$

$\underbrace{allInstances_D}_{expr_1}\text{->forAll}(\overset{w_1}{d} \mid \overset{expr_3}{d.x > 0})$

$allInstances_D \text{->iterate}(d : \tau_D ; result : Bool = true \mid result \text{ and } d.x > 0)$

Abbreviations on Top of Iterate

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

- $expr_1\text{->forAll}(w_1 : \tau_1 \mid expr_3)$
  is an abbreviation for
  $expr_1\text{->iterate}(w_1 : \tau_1 ; w_2 : Bool = \text{true} \mid w_2 \text{ and } expr_3)$.
- $\underbrace{expr_1}_{: Set(\tau_0)}\text{->Exists}(w : \tau_1 \mid expr_3)$
  is an abbreviation for
  $expr_1 \text{->iterate}(w_1 : \tau ; w_2 : Bool = false \mid w_2 \text{ or } expr_3)$.

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

OCL Syntax 4/4: Context

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

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

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

is an **abbreviation** for

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

Context: More Notational Conventions

- For
  context $self : \overset{C}{\cancel{\tau_C}}$ inv : $expr$
  we may alternatively write ("abbreviate as")
  context $\overset{C}{\cancel{\tau_C}}$ inv : $expr$
- **Within** the latter abbreviation, we may omit the "$self$" in $expr$, i.e. for
  $self.v$ and $self.r$
  we may alternatively write ("abbreviate as")
  v and r

Example

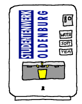
context *DD* inv : $\underbrace{wen \text{ implies } win > 0}_{expr}$

all Instances$_{DD}$ -> forAll ($\underline{self_{DD}}$: τ_{DD} |

expr { $\underline{self_{DD}}$. wen implies
$\underline{self_{DD}}$. win > 0)

Example

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



"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)
- ...

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.