Object Constraint Language 2

Martin Nečaský
Dept. of Software Engineering
Faculty of Mathematics and Physics
Charles University in Prague
Type System

- predefined types in OCL standard library
  - generic types
    - OclAny, OclInvalid
  - basic types
    - similar to those in other known languages
    - Boolean, Real, Integer, String
  - collection types
    - results of navigation through associations in class diagrams
    - Collection
    - Set, Bag, OrderedSet, Sequence

- user-defined types
  - defined by a user in UML diagrams
  - every instantiable model element in UML diagrams is automatically a type in OCL expressions
OclInvalid

- OclInvalid = \{invalid\}
- conforms to all other types
  - i.e. invalid can be an instance of any type in OCL
- any property call applied on invalid results in invalid
  - except for oclIsUndefined() and oclIsInvalid()
OclAny

- behaves as a supertype for all OCL types
- $a = b, a \neq b : Boolean$
  - equals, not equals
- $a.oclIsNew() : Boolean$
  - true if $a$ is created during performing an operation
  - can only be used in the operation postcondition
- $a.oclIsUndefined() : Boolean$
  - true if $a$ is equal to invalid or equal to null
- $a.oclIsInvalid() : Boolean$
  - true if $a$ is equal to invalid
- $a.oclIsTypeOf(t : Classifier) : Boolean$
  - true if $a$ is of the type $t$ but not a subtype of $t$
- $a.oclIsKindOf(t : Classifier) : Boolean$
  - true if $a$ is of the type $t$ or a subtype of $t$
- $a.oclType() : Classifier$
  - evaluates to type of $a$
Boolean

- represents common true/false values
- \( a \text{ or } b, a \text{ xor } b \) : Boolean
- \( a \text{ and } b \) : Boolean
- \( a \text{ implies } b \) : Boolean
- \( \text{not} \ (a) \) : Boolean
Real

- mathematical concept of real
- $a + b, a - b, -a : \text{Real}$
- $a \times b : \text{Real}$
- $a / b : \text{Real}$
- $a < b, a > b : \text{Boolean}$
- $a \leq b, a \geq b : \text{Boolean}$
- $a.\text{abs}() : \text{Real}$
- $a.\text{floor}(), a.\text{round}() : \text{Integer}$
- $a.\text{min}(b : \text{Real}) : \text{Real}$
- $a.\text{max}(b : \text{Real}) : \text{Real}$
**Integer**

- mathematical concept of integer
- subclass of **Real**
- \( a + b, a - b, -a : \text{Integer} \)
- \( a \times b : \text{Integer} \)
- \( a \div b : \text{Real} \)
- \( a.\text{abs}() : \text{Integer} \)
- \( a.\text{div}(b : \text{Integer}) : \text{Integer} \)
- \( a.\text{mod}(b : \text{Integer}) : \text{Integer} \)
- \( a.\text{min}(b : \text{Integer}) : \text{Integer} \)
- \( a.\text{max}(b : \text{Integer}) : \text{Integer} \)
String

- string is a sequence of characters in some suitable character set used to display information about the model
- \( a + b \) : String
- \( a.\text{size}() \) : Integer
- \( a.\text{concat}(b) \) : String
- \( a.\text{substring}(s, e: \text{Integer}) \) : String
  - substring starting and ending at positions between 1 and \( a.\text{size}() \)
- \( a.\text{toInteger}() \) : Integer, ...
- \( a.\text{toUpperCase}() \) : String
- \( a.\text{toLowerCase}() \) : String
- ...
Collections

- navigation via properties (association ends or attributes) results in a **Collection**
- **Collection** is an abstract type with four concrete sub-types:
  - Set
  - OrderedSet
  - Bag
  - Sequence
Navigation

- navigation via property $p$ from $a$
  
  $a.p$

- $a$ is `self` or a variable with an instance

- results to
  - single instance (object or value) or an empty `Set` when the max multiplicity of $p$ equals to 1
  - a `Set` when the max multiplicity of $p$ is greater than 1
  - an `OrderedSet` when the max multiplicity of $p$ is greater than 1 and $p$ is modified by `{ordered}`
Navigation

- navigation via a chain of properties $p_1 \ldots p_n$ from $a$

  $$a.p_1.\ldots.p_n$$

- results to Bag
Collection Constants

Set\{1,2,5,88\}
Sequence\{'apple', 'orange', 'pear'\}
Sequence\{1..(6+4)\}
Collection Iterator Operations

- different operations which iterate a collection and create a new collection from the existing one
- **select** and **reject** – specify a selection from a collection
- **collect**
- **forEach**
- **exists**
- **closure**
- **iterate**
- ... and more
Collection Iterator Operations

- general syntax of iterator operations is
  \[ \text{col->op(expression)} \]
  or
  \[ \text{col->op(v | expression-with-v)} \]
  or
  \[ \text{col->op(v: Type | expression-with-v)} \]

- sub-expressions of \text{expression} and \text{expression-with-v} implicitly start with the iteration variable when it is not present
  - contextual instance is referred by \textbf{self} reserved word but \textbf{self} is not implicit

context Person
inv: project
  ->op(startDate > self.startDate)

context Person
inv: self.project
  ->op(p|p.startDate > self.startDate)
Select and Reject Operations

- `select` specifies a subset of a collection containing all elements of the original collection for which a given expression evaluates to true.

  \[
  \text{collection} \rightarrow \text{select}(\text{boolean-expression})
  \]

- for each element of the original collection, evaluate the expression and put the result into the new collection.
Select and Reject Operations

- `reject` specifies a subset of a collection containing all elements of the original collection for which a given expression evaluates to false.

$$
collection \rightarrow \text{reject}(\text{boolean-expression})
\sim
$$

$$
collection \rightarrow \text{select}(\text{not(boolean-expression)})
$$
Select and Reject Operations

**context** Person

**inv:** self.authoredDoc

->reject(d | self.project.output->includes(d))

->size() = 0
Collect Operation

- **collect** specifies a collection that is computed from other collection
  - the new collection is not a sub-collection but contains elements computed/derived from the elements of the original collection
    
    `collection->collect(expression)`

- for each element of the original collection, evaluate the expression and put the result into the new collection
context Person

inv: self.authoredDoc->collect(d|d.project)
Collect Operation

collection->\texttt{collect}(\texttt{PropertyName})

\texttt{collection.PropertyName}
ForAll Operation

- **forAll** specifies a Boolean expression which must hold for all objects in a collection
  \[
  \text{collection} \rightarrow \text{forAll}(\text{expression})
  \]

- extended variant with more than one iterators of the same collection
  - iterator variables must be used in this case
    \[
    \text{collection} \rightarrow \text{forAll}(v_1,v_2 \mid \text{expression}) \\
    \sim \\
    \text{collection} \rightarrow \text{forAll}(v_1 \mid \text{expression}) \\
    \text{collection} \rightarrow \text{forAll}(v_2 \mid \text{expression})
    \]
context Person
inv: self.authoredDoc -> forall (d | self.project.output->includes(d))
context Project
inv: self.output ->forall (d1,d2 | d1<>d2 implies d1.serialNumber <> d2.serialNumber)
Exists Operation

- `exists` specifies a Boolean expression which must hold for at least one object in a collection

  \[\text{collection} \rightarrow \text{exists} (\text{expression})\]

- extended variant with more than one iterators is also possible
Closure Operation

- **closure** specifies a new collection created by a recursive application of an expression:
  \[ \text{collection} \to \text{closure}(\text{expression}) \]

- allows for expressing a transitive closure:
  - the expressive power of OCL exceeds the power of relationally complete languages
context Person
inv: self->asSet()->closure(boss)->size() <= 3
Iterate Operation

- **iterate** is a general loop operation

  \[
  \text{collection} \rightarrow \text{iterate}(
  \text{element} : \text{Type1};
  \text{result} : \text{Type2} = <\text{initial-value-expression}>
  \mid <\text{expression-with-element-and-result}>)
  \]

- **element** is iterator
- **result** accumulates the resulting value
  - it is also called accumulator
- for each element in **collection**, the **expression** is calculated using the previous value of **result**
Iterate Operation

source->forall(v | body ) = 
source->iterate(
    v; result : Boolean = true
    | result and body(v))

source->exists(v | body ) = 
source->iterate(
    v; result : Boolean = false
    | result or body(v))
Other Operations on Collections

- `collection->count(object) : Integer`
  - the number of occurrences of the object in the collection

- `collection->includes(object) : Boolean`
  - true if the collection contains the object

- `collection->isEmpty() : Boolean`
  - true if the collection is empty

- `collection->size() : Integer`
  - the number of elements in the collection

- ... and more
Other Operations on Collections

- difference to the iterator operations is that the default context variable is \texttt{self}, not the iteration variable

\texttt{context} \texttt{Person} \\
\texttt{inv: self.authoredDoc->} \\
excludesAll(\texttt{reviewedDoc})

\texttt{context} \texttt{Person} \\
\texttt{inv: self.authoredDoc->} \\
excludesAll(\texttt{self.reviewedDoc})