OCL 2.0 QUICK REFERENCE

The University of Texas at El Paso

OCL Constructs

context
Specifies the context for OCL expressions.

```
context Account
```

inv
States a condition that must always be met by all instances of a context type.

```
context Account
  inv: balance >= 0
```

pre
States a condition that must be true at the moment when an operation starts its execution.

```
context Account::deposit(amt: Integer): void
  pre: amt > 0
```

post
States a condition that must be true at the moment when an operation ends its execution.

```
context Account::deposit(amt: Integer): void
  post: balance = balance@pre + amt
```

init
Specifies the initial value of an attribute or association role.

```
context Account::balance: Integer
  init: 0
```

derive
Specifies the value of a derived attribute or association role.

```
context Account::interest: Real
derive: balance * .03
```

body
Defines the result of a query operation.

```
context Account::getBalance(): Integer
  body: balance
```

def
Introduces a new attribute or query operation.

```
context Account
  def: getBalance(): Integer = balance
```

package
Specifies explicitly the package in which OCL expressions belong.

```
package BankSystem::Accounting
context Account
  inv: balance >= 0
endpackage
```

OCL Expressions

self
Denotes the contextual instance.

```
context Account
  inv: self.balance >= 0
```

result
In a postcondition, denotes the result of an operation.

```
context Account::getBalance(): Integer
  post: result = balance
```

@pre
In a postcondition, denotes the value of a property at the start of an operation.

```
context Account::deposit(amt: Integer): void
  post: balance = balance@pre + amt
```

Navigation
Navigation through attributes, association ends, association classes, and qualified associations.

```
context Account
   inv: self.balance >= 0 -- dot notation
   -- collection operator (->)
   inv: owners->size() > 0
   -- association class, TransInfo
   inv: transactions.TransInfo->forall(amount > 0)
   -- qualified association, owners
   inv: not owners['primary'].isOclUndefined()
```

if-then-else expression
Conditional expression with a condition and two expressions.

```
context Account::interestRate: Real
derive: if balance > 5000 then .03 else .02 endif
```

let-in expression
Expression with local variables.

```
context Account::canWithdraw(amt: Integer): boolean
  def: let newBalance: Integer = balance - amt
       in newBalance > minimumBalance
```

Messaging (^)
Indicates that communication has taken place.

```
context Account::deposit(s: Sequence(Integer)): void
  pre: s->forAll(amt: Integer | amt > 0)
  post: balance = balance@pre + s->sum()
  post: s->forAll(amt: Integer | self^deposit(amt))
```

OCL Standard Library

Basic Types

Type	Values	Operations
Boolean	false, true	or, and, xor, not, =, <>, implies
Integer	-10, 0, 10, …	=, <>, <, >, <=, >=, +, -, *, /, mod(), div(), abs(), max(), min(), round(), floor()
Real	-1.5, 3.14, …	
String	'Carmen'	=, <>, concat(), size(), toLower(), toUpper(), substring()

OclAny
Supertype of all UML and OCL types

Operation	Description
=	True if *self* and the argument are the same
<>	True if *self* and the argument are not the same
oclIsNew()	True if *sel* was created during the operation
oclIsUndefined()	True if *self* is undefined
oclAsType(type)	*self* as of the given type, *type*
oclIsTypeOf(type)	True if *self* is an instance of the given type, *type*
oclIsKindOf(type)	True if *self* conforms to the given type, *type*
oclisInState(state)	True if *self* is in the given state, *state*
T::allInstance()	Set of all instances of the type *T*

OclVoid

Type with one single instance (*undefined*) that conforms to all others types

Operation	Description
oclIsUndefined()	Always true

OclMessage

Messages that can be sent to and received by objects

Operation	Description
hasReturned()	Is the operation (*self*) called and returned?
result()	Result of the operation (*self*) or undefined
isSignalSent()	Is *self* a sending of a UML signal?
isOperationCall()	Is *self* a UML operation call?

Tuple

A tuple consists of named parts each of which can have a distinct type.

```
-- Tuple(name: String, age: Integer)
Tuple {name: String = 'John, age: Integer = 20}
```

Collection Types

Four collection types (Set, OrderedSet, Bag, and Sequence) with Collection as the abstract supertype.

Collection constants

Set {1, 2, 3} -- Set(Integer)
OrderedSet {‘apple, ‘pear’, ‘orange’} -- OrderedSet(String)
Bag {1, 1, 2, 2} -- Bag(Integer)
Sequence {1..(4 + 6), 15} – Sequence(Integer)

Standard operations

Operation	Description
count(o)	Number of occurrences of *o* in the collection (*self*)
execludes(o)	Is *o* not an element of the collection?
excludesAll(c)	Are all the elements of *c* not present in the collection?
includes(o)	Is *o* an element of the collection?
includesAll(c)	Are all the elements of *c* contained in the collection?
isEmpty()	Does the collection contain no element?
notEmpty()	Does the collection contain one or more elements?
size()	Number of elements in the collection
sum()	Addition of all elements in the collection

Collection operations

Operation	Set	OrderedSet	Bag	Sequence
=	O	O	O	O
<>	O	O	O	O
-	O	O		
append(o)		O		O
asBag()	O	O	O	O
asOrderedSet()	O	O	O	O
asSequence()	O	O	O	O
asSet()	O	O	O	O
at(i)*		O		O
excluding(o)	O	O	O	O
first()		O		O
flatten()	O	O	O	O
including(o)	O	O	O	O
indexOf(o)		O		O
insertAt(i, o)		O		O
intersection(c)	O		O	
last()		O		O
prepend(o)		O		O
subOrderedSet(l, u)		O		
subsequence(l, u)				O
symmetricDifference(c)	O			
union(c)	O	O	O	O

*OCL uses 1-based index for ordered sets and sequences.

including(o): new collection as *self* but with *o* added
excluding(o): new collection as *self* but with *o* removed

Iteration operations

Operation	Description
any(expr)	Returns any element for which *expr* is true
collect(expr)	Returns a collection that results from evaluating *expr* for each element of *self*
collectNested(expr)	Returns a collection of collections that result from evaluating *expr* for each element of *self*
exists(expr)	Has at least one element for which *expr* is true?
forAll(expr)	Is *expr* true for all elements?
isUnique(expr)	Does *expr* has unique value for all elements?
iterate(x: S; y: T| expr)	Iterates over all elements
one(expr)	Has only one element for which *expr* is true?
reject(expr)	Returns a collection containing all elements for which *expr* is false
select(expr)	Returns a collection containing all elements for which *expr* is true
sortedBy(expr)	Returns a collection containing all elements ordered by *expr*

accounts->any(a: Account | a.balance > 1000)
accounts->collect(name) -- all the names
accounts->collectNested(owners)
accounts->exists(balance > 5000)
accounts->forAll(balance >= 0)
accounts->isUnique(name)
accounts->iterate(a: Account; sum: Integer = 0 | sum + a.balance)
accounts->one(name = “Carmen”)
accounts->reject(balance > 1000)
accounts->select(balance <= 1000)
accounts->sortedBy(balance)