K Semantics for OCL

a Proposal for a Formal Definition for OCL

Vlad Rusu, Inria Lille, France
Dorel Lucanu, Al. I. Cuza University of Iaşi, Romania

10/08/2011, 2nd K Workshop, Cheile Grădiştei
Outline

1. Introduction & Motivation
2. OCL
3. K Semantics for OCL
4. Conclusion
Outline

1. Introduction & Motivation
2. OCL
3. K Semantics for OCL
4. Conclusion
OCL

😊 OCL is a formal language used to describe expressions like constraints or queries over objects in a UML model

- the constraints are used to give an exact description of the information contained in the models
- the queries are used to analyse these models and to validate them

😊 the evaluation of the OCL expressions does not have side effects

😊 OCL is not yet widely adopted in industry

- the lack of proper and integrated tool support
- experience has shown that the language definition is not precise enough
OMG Standard

- an intuitive description of the OCL
- abstract syntax
  - as a MOF (Meta Object Facility) compliant metamodel
- concrete syntax
  - as a full attribute grammar
  - a map from abstract syntax to concrete one is given
- semantics described using UML
  - both semantic domains and evaluation are described as UML packages
  - the evaluation can be seen as a mapping between semantic domain and abstract syntax
- OCL standard library
  - predefined types, their operations, and predefined expression templates

The Use of OCL Expressions in UML Models
OMG Standard is Not Quite Formal

- an **informative formal semantics** is given in Appendix A
- heavily based on mathematical notation
- without an execution engine is useless
- the relationship with the UML semantics not investigated
- each framework using OCL should have its own implementation
- no means to verify if an implementation conforms to the standard
Outline

1. Introduction & Motivation
2. OCL
3. K Semantics for OCL
4. Conclusion
Relation to the UML metamodel
Example of OCL Query

```ocl
Paper.allInstances()->select(
    authors->includes("John")
)
```

All instances of the class Paper having "John" as author or co-author
context Teacher inv:
  self.position == Professor implies
  Paper.allInstances() -> select(
    authors -> includes(self.name)
  ).publication.rank -> sum() >= 6.5

Any professor must have the sum of paper ranks at least 6.5, where the rank of a paper is given by the rank of the publication which it appears in.
Types in OCL

- basic types: Integer, Real, Boolean, String
- enumeration types
- collections: Set(T), OrderedSet(T), Bag(T), Sequence(T)
  Collection is the abstract supertype of all collection types
- structured values: Tuple(name:String, age:Integer)
- object types: each class \( c \) induces a type \( t_c \) having the same name as the class
  The domain of a class \( c \) is the set of objects that can be created by this class and all of its child classes
OCL Operations over Collections

collection->size() : Integer
collection->includes(object : T) : Boolean
collection->excludes(object : T) : Boolean
collection->count(object : T) : Integer
collection->includesAll(c2 : Collection(T)) : Boolean
collection->excludesAll(c2 : Collection(T)) : Boolean
collection->isEmpty() : Boolean
collection->notEmpty() : Boolean

collection->sum() : T

collection->product(c2 : Collection(T2)) : Set( Tuple( first: T, second: T2) )
OCL Collection Operations

collection->select( boolean-expression )
collection->reject( v : Type | boolean-expression-with-v )
collection->collect( v : Type | expression-with-v )
collection->forall( v : Type | boolean-expression-with-v )
collection->exists( v : Type | boolean-expression-with-v )
collection->iterate( elem : Type; acc : Type = <expression> | expression-with-elem-and-acc )
an object model includes:

- a set of classes \( \text{CLASS} \)
- a set of attributes for each class \( \text{ATT}_c \)
- a set of operations for each class \( \text{OP}_c \)
- a set of associations with role names and multiplicities \( \text{ASSOC}_c \)
- a generalization hierarchy over classes \( \prec \)
A system state for a model $M$ is a structure $\sigma(M) = (\sigma_{CLASS}, \sigma_{ATT}, \sigma_{ASSOC})$, where

- $\sigma_{CLASS}(c)$ contain all objects of a class $c \in CLASS$ existing in the system state
- $\sigma_{ATT}$ assigns attribute values to each object: $\sigma_{ATT}(a) : \sigma_{CLASS}(c) \rightarrow I(t)$ for each $a : t_c \rightarrow t \in ATT^*_c$
- $\sigma_{ASSOC}$ contain links connecting objects: $\sigma_{ASSOC}(as) \subset I_{ASSOC}(as)$ for each $as \in ASSOC$
A context for evaluation is given by an environment $\pi = (\sigma, \beta)$ consisting of a system state $\sigma$ and a variable assignment $\beta : Var_t \rightarrow I(t)$.

A system state $\sigma$ provides access to the set of currently existing objects, their attribute values, and association links between objects.

A variable assignment $\beta$ maps variable names to values.
Outline

1. Introduction & Motivation
2. OCL
3. K Semantics for OCL
4. Conclusion
- partial implemented
- we use the sort Val for OCL values
- OCL basic types Integer, Boolean, and String are described by the $K$ types Int, Bool, and String, respectively

```
syntax Val ::= val ( Bag )
```

- if expressions are correctly typed, we may work with Collection instead of Collection(T)

```
syntax K ::= typedElt ( Val , Id )
```

- we extend $K$ data structure Bag to implement OCL type Bag
- subtype relation Bag $\leq$ Val is an injection

```
syntax Val ::= val ( Bag )
```
Types (values.k) 2/2

– the relation \( \text{Bag}(T) \leq \text{Bag} \) for scalar types \( T \) is modelled by the injection \( \text{BagItem} \)
– a scalar value \( v \) has two representations: \( v \) and \( \text{val} (\text{BagItem}(v)) \)
– the operations should be defined both representations of the scalars

\[
\text{rule } \text{KL:KLabel} (\text{val} (\text{BagItem}(\text{Val1:Val}))) \Rightarrow \text{val} (\text{BagItem}(\text{Val2:Val}))
\]
\[
= \text{KL:KLabel} (\text{Val1}, \text{Val2})
\]
\[
\text{if } (\text{binaryOp}(\text{KL}))
\]
\[
\text{andBool isScalar} ((\text{Val1}))
\]
\[
\text{andBool isScalar} ((\text{Val2}))
\]

\[
\text{[structural]}
\]

\[
\_+\_ (\text{val} (\text{BagItem}(3))) \Rightarrow \_+\_ (\text{val} (\text{BagItem}(7))) \Rightarrow \_+\_ (3, 7)
\]
Model Syntax in \( \mathcal{K} \)

\[
kmod \text{ META-MODEL-INTERFACE is including } \mathcal{K} + \#\text{ID}
\]

\[
syntax \ \mathcal{K} ::= \text{Id}
\]

\[
syntax \ \text{Bag} ::= \text{children} \ ( \ \text{Id} \ )
\]

\[
syntax \ \text{Map} ::= \text{attributeDecl} \ ( \ \text{Id} \ )
\]

\[
syntax \ \text{Bool} ::= \text{isEnum} \ ( \ \text{Id} \ )
\]

\[
syntax \ \text{Bag} ::= \text{values} \ ( \ \text{Id} \ )
\]

\[\text{endkm}\]
Computations

– strictness attributes

```latex
syntax Exp ::= Exp ->forall( Id ' | Exp ) [strict (1) prec 1]
| Exp ->exists( Id ' | Exp ) [strict (1) prec 1]
| Exp ->select( Id ' | Exp ) [strict (1) prec 1]
| Exp ->collect( Id ' | Exp ) [strict (1) prec 1]
| let Id '=' Exp in Exp endlet [strict(2)]
| if Exp then Exp else Exp endif [strict(1)]
```

– the values (results)

```latex
syntax KResult ::= Val
syntax Exp ::= Val
syntax K ::= Exp
```
Configuration

instances

instance *

instName
oclUndefined

ofClass
oclUndefined

attributes
.

k
.

result
.

mem
.

nextLoc
0
Correspondece with the System State: $\sigma_{\text{CLASS}}(c)$
Correspondence with the System State: $\sigma_{ATT}$

\[ \sigma_{ATT}(name)(pers) = "John" \]
Correspondence with the System State: $\sigma_{ASSOC}$

$\text{associates}(as) = \{\text{Paper, Person}\}$
$\text{roles}(as) = \{\text{authors}\}$
$\sigma_{ASSOC}(\text{authors})(\text{pap}) = \{\text{pers}\}$
The base case is trivial:

\[
\text{val}(\cdot) \rightarrow \text{forAll}(_\text{List}\{K\} \mid _\text{List}\{K\}) \quad \text{true}
\]

The inductive step uses the substitution operator:

\[
\text{Exp} ::= \text{Exp} [ K / \text{Id} ] \quad [\text{ditto}]
\]

The K items "boxed" as bag items must be "unboxed":

\[
K ::= \text{open}(\text{BagItem})
\]

\[
\text{open}(\text{BagItem}(K)) \Rightarrow K
\]
Now, the rule for the inductive step is written as:

\[
\text{val}\left( \text{Bglt } Bg \right) \rightarrow \text{forAll}\left( \text{Var } | \text{BEXP} \right) \quad \text{if }\text{BEXP } \left[ \text{open}\left( \text{Bglt} \right) / \text{Var} \right] \text{ then } \text{val}\left( Bg \right) \rightarrow \text{forAll}\left( \text{Var } | \text{BEXP} \right) \text{ else false endif}
\]
NB. Correct syntax is ColExp.At but Maude parser ...

– returns the values of the attribute "At" for all objects obtained by the evaluation of "ColExp"

– the operator is strict in the first argument

– we need a way to store the results

... ihm, the evaluation of the argument "ColExp" could imply the evaluation other subexpressions __#__
Our solution:

- use a memory cell \( \text{mem} \) which stores pairs of maps
  \[
  \text{location} \mapsto \text{sharpExpression} \\
  \text{location} + 1 \mapsto \text{partialValue}
  \]

- an operator \( \ast \) such that the evaluation of a sharp expression is replaced with the evaluation of \( \ast \ \text{location} \)

- the intermediate steps update the value from \( \text{location} + 1 \)

- when this value (of the sharp (sub)expression from the \( \text{location} \)) is completely computed, it replaces \( \ast \ \text{location} \) in the \( k \) cell.
– additional syntax

\[ K ::= * \text{Nat} \]
\[ \mid [ K, K ] \]

– initial step

\[
\frac{\text{val}(\text{REMAINING}) \# \text{AT}}{\ast \tilde{N} \quad \text{mem}}
\]

\[
\frac{\text{mem}}{N \mapsto [\text{val}(\text{REMAINING}) \# \text{AT}, N + \text{Nat} 1]} \quad N + \text{Nat} 1 \mapsto \text{val}(\cdot)
\]

\[
\frac{\text{nextLoc}}{N \mapsto N + \text{Nat} 2}
\]
– loop step

\[
\begin{align*}
\text{mem}
N &\mapsto \begin{cases} 
\text{val}(\text{INAME} : \text{Cls \hspace{1mm} REMAINING}) \# \text{AT} , N' \end{cases} \\
N' &\mapsto \text{val}(\_\text{Bag} \cdot ) \\
\text{val}(\text{REMAINING}) \# \text{AT} &\mapsto \text{typedElt}(\text{val} (\text{Bg}) , \text{T}) 
\end{align*}
\]
– last step

\[ \begin{align*}
\text{mem} & \\
N & \mapsto [ \text{val}(\cdot) \# AT, N' ] \\
N' & \mapsto \text{val}(Bg)
\end{align*} \]
Outline

1. Introduction & Motivation
2. OCL
3. K Semantics for OCL
4. Conclusion
we presented a $\mathcal{K}$ definition for a kernel of OCL
the description in $\mathcal{K}$ not trivial
the use of substitution mechanism
the mechanism for _#_

it started as a component of the DSML project, but we think that OCL deserves to be completely developed as an independent project

it could be the first standard formal semantics for OCL (it is a definition, not a implementation!)
Future work

- complete description of the types
- operations
- contexts
- missing OCL expressions
- relationship with the UML semantics
- online tool for experiments and testing bench