Formal Specification of Software

The Object Constraint Language by Example

Bernhard Beckert

Universität Koblenz-Landau
The Classifier Context

context (c :)? typeName
  inv expressionName?: OclExpression

context (c :)? typeName
  inv expressionName_1?: OclExpression_1
  ...
  ...
  inv expressionName_n?: OclExpression_n
The Operator Context

context ( c :)?

typeName ::opName(p1: type1; ...; pk: typek):rtype

{pre ,post } expressionName? : OclExpression
Constraints with Attributes

```
<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
</tr>
<tr>
<td>e-mail: String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>authors[*]: Person</td>
</tr>
<tr>
<td>number: Int</td>
</tr>
<tr>
<td>status: Status</td>
</tr>
<tr>
<td>totalnumber: Int</td>
</tr>
<tr>
<td>sumpages: Int</td>
</tr>
<tr>
<td>evaluate()</td>
</tr>
</tbody>
</table>

<<enumeration>> Status

submitted
accept
reject

context Paper

inv number ≥ 1

B. Beckert: Formal Specification of Software – p.4
context Paper

inv self.number $\geq 1$
<table>
<thead>
<tr>
<th>context</th>
<th>Paper</th>
<th>context</th>
<th>c:Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv</td>
<td>self.number ≥ 1</td>
<td>inv</td>
<td>c.number ≥ 1</td>
</tr>
</tbody>
</table>
Equivalent notational variations

context Paper
inv self.number ≥ 1

context c:Paper
inv c.number ≥ 1

context c:Paper
inv startCount : c.number ≥ 1
Equivalent notational variations

context Paper
  inv self.number \geq 1

context c:Paper
  inv c.number \geq 1

context c:Paper
  inv startCount : c.number \geq 1

context Paper
  inv startCount : number \geq 1
context \( \text{Paper::evaluate()} \)

pre \( \text{c.status} = \text{submitted} \)

post \( \text{c.status} = \text{accept or} \)

\( \text{c.status} = \text{reject} \)
Types

Model types

The classes form the context diagram of an OCL constraint
Types

Model types

The classes form the context diagram of an OCL constraint

Basic types

Integer, Real, Boolean and String
Types

Model types
The classes form the context diagram of an OCL constraint

Basic types
Integer, Real, Boolean and String

Enumeration types
The user defined enumeration types
Types

Model types
The classes form the context diagram of an OCL constraint

Basic types
Integer, Real, Boolean **and** String

Enumeration types
The user defined enumeration types

Collection types
Set, Bag, Sequence
Types

Model types
The classes form the context diagram of an OCL constraint

Basic types
*Integer, Real, Boolean and String*

Enumeration types
The user defined enumeration types

Collection types
*Set, Bag, Sequence*

Special types
*e.g. OclAny, OclType*
Subtyping

$T_1, T_2$ model types:

$T_1 < T_2$ holds exactly if $T_1$ is a subclass of $T_2$
Subtyping

\( T_1, T_2 \) model types:

\( T_1 < T_2 \) holds exactly if \( T_1 \) is a subclass of \( T_2 \)

\( \text{Integer} < \text{Real} \)
Subtyping

$T_1, T_2$ model types:

$T_1 < T_2$ holds exactly if $T_1$ is a subclass of $T_2$

$\text{Integer} < \text{Real}$

For all type expressions $T$, not denoting a collection type:

- $\text{Set}(T) < \text{Collection}(T)$
- $\text{Bag}(T) < \text{Collection}(T)$
- $\text{Sequence}(T) < \text{Collection}(T)$
Subtyping

- $T_1, T_2$ model types:
  - $T_1 < T_2$ holds exactly if $T_1$ is a subclass of $T_2$
  - $\text{Integer} < \text{Real}$

- For all type expressions $T$, not denoting a collection type:
  - $\text{Set}(T) < \text{Collection}(T)$
  - $\text{Bag}(T) < \text{Collection}(T)$
  - $\text{Sequence}(T) < \text{Collection}(T)$

- If $T$ is a model, basic, or enumeration type: $T < OCL\text{Any}$
Subtyping

- \( T_1, T_2 \) model types:
  \[ T_1 < T_2 \text{ holds exactly if } T_1 \text{ is a subclass of } T_2 \]

- \( \text{Integer} < \text{Real} \)

- For all type expressions \( T \), not denoting a collection type:
  - \( \text{Set}(T) < \text{Collection}(T) \)
  - \( \text{Bag}(T) < \text{Collection}(T) \)
  - \( \text{Sequence}(T) < \text{Collection}(T) \)

- If \( T \) is a model, basic, or enumeration type: \( T < \text{OCLAny} \)

- If \( T_1 < T_2 \) and \( C \) is any of the type constructors \( \text{Collection}, \text{Set}, \text{Bag}, \text{Sequence} \):
  \[ C(T_1) < C(T_2). \]
Typing Examples

**Person**

<table>
<thead>
<tr>
<th>attribute</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td>e-mail</td>
<td>String</td>
</tr>
</tbody>
</table>

**Paper**

<table>
<thead>
<tr>
<th>attribute</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>authors[*]</td>
<td>Person</td>
</tr>
<tr>
<td>number</td>
<td>Int</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
</tr>
<tr>
<td>totalnumber</td>
<td>Int</td>
</tr>
<tr>
<td>sumpages</td>
<td>Int</td>
</tr>
<tr>
<td>evaluate()</td>
<td></td>
</tr>
</tbody>
</table>

**<<enumeration>>**

**Status**

<table>
<thead>
<tr>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>submitted</td>
</tr>
<tr>
<td>accept</td>
</tr>
<tr>
<td>reject</td>
</tr>
</tbody>
</table>

**constraint**

**p:** Person

- `p.name`, `p.e-mail` have type `String`.

**c:** Paper

- `c.number` has type `Integer`,
- `c.status` has type `Status`,
- `c.authors` has type `Set(Person)`
Constraints with Associations

context c:Paper

inv c.author <> c.referee
context c:Paper

inv c.authors -> intersection(c.referee) -> isEmpty
context c:Paper

inv not(c.authors \rightarrow includes(c.session.chair))

context p:Person

inv p.reviewed_papers.session.chair \rightarrow includes(p)
**allInstances**

**context** Person

**inv** Person.allInstances $\rightarrow$ forAll(p $|$ p.e-mail.size $\geq$ 3)

**context** Paper

**inv** Paper.allInstances $\rightarrow$ forAll(p1, p2 $|$ p1 $\iff$ p2 implies p1.number $\iff$ p2.number)
Avoiding `allInstances`

context `Person`

inv `Person.allInstances \rightarrow \forall p \mid p.e-mail.size \geq 3`

Can be equivalently replaced by:

context `p:Person`

inv `p.e-mail.size \geq 3`
Avoiding `allInstances`

context Paper

inv Paper.allInstances → forall(p1, p2 | p1 <> p2 implies p1.number <> p2.number)

Can be equivalently replaced by:

context p1,p2:Papers

inv p1 <> p2 implies p1.number <> p2.number)
Avoiding *allInstances*

```
context Conference

inv self.submitted_papers -> forall(p1, p2 | p1 <> p2 implies p1.number <> p2.number)
```
Introducing the *iterate* Operation

```
context p:Papers

inv Papers.allInstances ->
iterate(x:Paper ; y:Int = 0 | y+x.pages)
= Papers.sumpages
```
Syntax of the \textit{iterate} construct

\begin{verbatim}
t -\rightarrow \text{iterate}(x : S; y : T = t_0 \mid u)
\end{verbatim}

- \textbf{iterator variable}
- \textbf{term of sort }T, \textbf{initial term}
- \textbf{range formula}
- \textbf{accumulator variable}
- \textbf{term of sort }T, \textbf{step term}
iterate: Example 1

Adding a new operation *occurences* to the built-in OCL type *String*

```plaintext
string.occurences(string2:String):Set(Integer)  The set of positions in
  string where an occurrence of string2 as a substring starts. Strings
  start with position 0.
```
iterate: Example 1

Adding a new operation occurrences to the built-in OCL type String

string.occurrences(string2:String):Set(Integer)  The set of positions in
  string where an occurrence of string2 as a substring starts. Strings
  start with position 0.

pre  : string2.size =< string.size

post : result = \{ 0 .. (string.size - string2.size) \} –>
          iterate(x; y:Set(Integer)=Set{} | if string.substring(x,x+string2.size) = string2
          then y –> including(x) else y)
iterate: Example 2

Adding a new operation `substringOcc` to the built-in OCL type `String`

`string.substringOcc(string2:String):Boolean` True if `string2` occurs at least once as a substring in `string`.
**iterate: Example 2**

Adding a new operation `substringOcc` to the built-in OCL type `String`:

```plaintext
string.substringOcc(string2:String):Boolean

True if string2 occurs at least once as a substring in string.
```

**post**

```plaintext
result = (string2.size < string.size) and
not (string.occurences(string2) -> isEmpty)
```
Quantifiers

\[ t \rightarrow \text{iterate}(x; y : \text{Boolean} = \text{true} \mid y \text{ and } a) \]

- \( t \) is an expression of type \( \text{Set}(T) \)
- \( x \) is a variable of type \( T \)
- \( a \) is an expression of type \( \text{Boolean} \)
Quantifiers

\[ t \rightarrow iterate(x; y : Boolean = true \mid y \text{ and } a) \]

- \( t \) is an expression of type \( Set(T) \)
- \( x \) is a variable of type \( T \)
- \( a \) is an expression of type \( Boolean \)

Can be equivalently expressed by

\[ t \rightarrow \text{forall}(x \mid a) \]
Quantifiers

\[ t \rightarrow \text{iterate}(x; y : \text{Boolean} = \text{true} \mid y \text{ and } a) \]

- \( t \) is an expression of type \( \text{Set}(T) \)
- \( x \) is a variable of type \( T \)
- \( a \) is an expression of type \( \text{Boolean} \)

Can be equivalently expressed by

\[ t \rightarrow \text{forall}(x \mid a) \]

Likewise

\[ t \rightarrow \text{iterate}(x; y : \text{Boolean} = \text{false} \mid y \text{ or } a) \]
Quantifiers

\[ t \rightarrow iterate(x; y : Boolean = true \mid y \text{ and } a) \]

- \( t \) is an expression of type \( Set(T) \)
- \( x \) is a variable of type \( T \)
- \( a \) is an expression of type \( Boolean \)

Can be equivalently expressed by

\[ t \rightarrow \text{forall}(x \mid a) \]

Likewise

\[ t \rightarrow iterate(x; y : Boolean = false \mid y \text{ or } a) \]

Can be expressed by

\[ t \rightarrow \text{exists}(x \mid a) \]
context \( c : \text{Conference} :: \text{isAuthor} (\text{name} : \text{String}) \)

pre true

post result =
\[ c.\text{sp} \rightarrow \text{collect}(p \mid p.\text{author.name}) \rightarrow \text{includes}(\text{name}) \]
Reducing \textit{collect} to \textit{iterate}

\[
\text{set} \rightarrow \text{collect}(x \mid \text{expr}) : \text{Bag}(T)
\]
\[
= \\
\text{set} \rightarrow \text{iterate}(x; \text{acc} : \text{Bag}(T) = \text{Bag}\{\} \mid \text{acc} \rightarrow \text{including}(\text{expr}) )
\]

Evaluation of

\[
c.\text{sp} \rightarrow \text{collect}(p \mid p.\text{authors}.\text{name})
\]

involves implicit flattening.
Selecting Elements

<table>
<thead>
<tr>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>authors[*]:Person</td>
</tr>
<tr>
<td>number:Int</td>
</tr>
<tr>
<td>pages:Int</td>
</tr>
<tr>
<td>countShortPapers():Integer</td>
</tr>
</tbody>
</table>

**context**  
`Paper::countShortPapers():Integer`

**pre**  
`true`

**post**  
`result = Paper.allInstances → select(p | p.pages < 10) → size`
Reducing \textit{select} to \textit{iterate}

\[ s \rightarrow \text{select}(x \mid \text{expr}) : \text{Set}(T) = \]

\[ s \rightarrow \text{iterate}(x; \text{acc} : \text{Set}(T) = \text{Set}\{\} \mid \]
\hspace{1cm} \text{if expr then acc} \rightarrow \text{including}(x) \]
\hspace{1cm} \text{else acc} \]

where

\begin{itemize}
  \item \( s \) is of type \( \text{Set}(T) \)
  \item \( \text{expr} \) is an OCL expression of type \( \text{Boolean} \)
\end{itemize}
Refering to Previous Values

c:Conference::addPaper()

\[\text{pre} \quad \text{true}\]

\[\text{post} \quad \text{totalnumber} = \text{totalnumber}@\text{pre} + 1\]
Multiple Occurences of @pre

(c.pa.phone) the new phone number of the current p.a.

(c.pa@pre.phone) the new phone number of the previous p.a.

(c.pa.phone@pre) the old phone number of the current p.a.

(c.pa@pre.phone@pre) the old phone number of the previous p.a.