The Language OCL

BNF-converter

April 10, 2019

This document was automatically generated by the *BNF-Converter*. It was generated together with the lexer, the parser, and the abstract syntax module, which guarantees that the document matches with the implementation of the language (provided no hand-hacking has taken place).

The lexical structure of OCL

Identifiers

Identifiers ⟨*Ident*⟩ are unquoted strings beginning with a letter, followed by any combination of letters, digits, and the characters `_ '`, reserved words excluded.

Literals

Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in OCL are the following:

`Bag`	`Collection`	`Sequence`
`Set`	`and`	`context`
`def`	`else`	`endif`
`endpackage`	`false`	`if`
`implies`	`in`	`inv`
`let`	`not`	`null`
`or`	`package`	`post`
`pre`	`then`	`true`
`xor`		

The symbols used in OCL are the following:

```
:    ::   (
)    =    +
-    <    <=
>    >=   /
*    <>   ,
^    ?    [
]    @    |
;    {    }
..   .    ->
```

Comments

Single-line comments begin with −−.
Multiple-line comments are enclosed with `/*` and `*/`.

The syntactic structure of OCL

Non-terminals are enclosed between ⟨ and ⟩. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

```
⟨OCLfile⟩  ::=  ⟨ListOCLPackage⟩

⟨ListOCLPackage⟩  ::=  ⟨OCLPackage⟩
                   |   ⟨OCLPackage⟩ ⟨ListOCLPackage⟩

⟨OCLPackage⟩  ::=  package ⟨PackageName⟩ ⟨OCLExpressions⟩ endpackage

⟨PackageName⟩  ::=  ⟨PathName⟩

⟨OCLExpressions⟩  ::=  ⟨ListConstrnt⟩

⟨ListConstrnt⟩  ::=  ϵ
                 |   ⟨Constrnt⟩ ⟨ListConstrnt⟩

⟨Constrnt⟩  ::=  ⟨ContextDeclaration⟩ ⟨ListConstrBody⟩

⟨ListConstrBody⟩  ::=  ⟨ConstrBody⟩
                   |   ⟨ConstrBody⟩ ⟨ListConstrBody⟩
```

```
⟨ConstrBody⟩ ::= def ⟨Ident⟩ : ⟨ListLetExpression⟩
              |  def : ⟨ListLetExpression⟩
              |  ⟨Stereotype⟩ ⟨Ident⟩ : ⟨OCLExpression⟩
              |  ⟨Stereotype⟩ : ⟨OCLExpression⟩

⟨ContextDeclaration⟩ ::= context ⟨OperationContext⟩
                      |  context ⟨ClassifierContext⟩

⟨ClassifierContext⟩ ::= ⟨Ident⟩ : ⟨Ident⟩
                     |  ⟨Ident⟩

⟨OperationContext⟩ ::= ⟨Ident⟩ :: ⟨OperationName⟩ ( ⟨ListFormalParameter⟩ )
                    |  ⟨Ident⟩ :: ⟨OperationName⟩ ( ⟨ListFormalParameter⟩ ) : ⟨ReturnTy

⟨Stereotype⟩ ::= pre
              |  post
              |  inv

⟨OperationName⟩ ::= ⟨Ident⟩
                 |  =
                 |  +
                 |  −
                 |  <
                 |  <=
                 |  >
                 |  >=
                 |  /
                 |  *
                 |  <>
                 |  implies
                 |  not
                 |  or
                 |  xor
                 |  and

⟨ListFormalParameter⟩ ::= ϵ
                       |  ⟨FormalParameter⟩
                       |  ⟨FormalParameter⟩ , ⟨ListFormalParameter⟩

⟨FormalParameter⟩ ::= ⟨Ident⟩ : ⟨TypeSpecifier⟩

⟨TypeSpecifier⟩ ::= ⟨SimpleTypeSpecifier⟩
                 |  ⟨CollectionType⟩

⟨CollectionType⟩ ::= ⟨CollectionKind⟩ ( ⟨SimpleTypeSpecifier⟩ )

⟨ReturnType⟩ ::= ⟨TypeSpecifier⟩
```

```
⟨OCLExpression⟩      ::=  ⟨Expression⟩
                      |   ⟨ListLetExpression⟩ in ⟨Expression⟩

⟨LetExpression⟩      ::=  let ⟨Ident⟩ = ⟨Expression⟩
                      |   let ⟨Ident⟩ : ⟨TypeSpecifier⟩ = ⟨Expression⟩
                      |   let ⟨Ident⟩ ( ⟨ListFormalParameter⟩ ) = ⟨Expression⟩
                      |   let ⟨Ident⟩ ( ⟨ListFormalParameter⟩ ) : ⟨TypeSpecifier⟩ = ⟨Expression⟩

⟨ListLetExpression⟩  ::=  ⟨LetExpression⟩
                      |   ⟨LetExpression⟩ ⟨ListLetExpression⟩

⟨IfExpression⟩       ::=  if ⟨Expression⟩ then ⟨Expression⟩ else ⟨Expression⟩ endif

⟨Expression⟩         ::=  ⟨Expression⟩ implies ⟨Expression1⟩
                      |   ⟨Expression1⟩

⟨Expression1⟩        ::=  ⟨Expression1⟩ ⟨LogicalOperator⟩ ⟨Expression2⟩
                      |   ⟨Expression2⟩

⟨Expression2⟩        ::=  ⟨Expression2⟩ ⟨EqualityOperator⟩ ⟨Expression3⟩
                      |   ⟨Expression3⟩

⟨Expression3⟩        ::=  ⟨Expression3⟩ ⟨RelationalOperator⟩ ⟨Expression4⟩
                      |   ⟨Expression4⟩

⟨Expression4⟩        ::=  ⟨Expression4⟩ ⟨AddOperator⟩ ⟨Expression5⟩
                      |   ⟨Expression5⟩

⟨Expression5⟩        ::=  ⟨Expression5⟩ ⟨MultiplyOperator⟩ ⟨Expression6⟩
                      |   ⟨Expression6⟩

⟨Expression6⟩        ::=  ⟨UnaryOperator⟩ ⟨Expression7⟩
                      |   ⟨Expression7⟩

⟨Expression7⟩        ::=  ⟨Expression7⟩ ⟨PostfixOperator⟩ ⟨PropertyCall⟩
                      |   ⟨Expression7⟩ ^ ⟨PathName⟩ ( ⟨ListMessageArg⟩ )
                      |   ⟨Expression8⟩

⟨Expression8⟩        ::=  ⟨PropertyCall⟩
                      |   ⟨LiteralCollection⟩
                      |   ⟨OCLLiteral⟩
                      |   ⟨IfExpression⟩
                      |   null
                      |   ( ⟨Expression⟩ )

⟨MessageArg⟩         ::=  ⟨Expression⟩
                      |   ?
                      |   ? : ⟨TypeSpecifier⟩
```

```
⟨ListMessageArg⟩  ::=  ϵ
                    |   ⟨MessageArg⟩
                    |   ⟨MessageArg⟩ , ⟨ListMessageArg⟩

⟨PropertyCall⟩  ::=  ⟨PathName⟩ ⟨PossTimeExpression⟩ ⟨PossQualifiers⟩ ⟨PossPropCallParam⟩

⟨PathName⟩  ::=  ⟨ListPName⟩

⟨PName⟩  ::=  ⟨Ident⟩

⟨ListPName⟩  ::=  ⟨PName⟩
               |   ⟨PName⟩ :: ⟨ListPName⟩

⟨PossQualifiers⟩  ::=  ϵ
                    |   ⟨Qualifiers⟩

⟨Qualifiers⟩  ::=  [ ⟨ListExpression⟩ ]

⟨PossTimeExpression⟩  ::=  ϵ
                        |   @ pre

⟨PossPropCallParam⟩  ::=  ϵ
                       |   ⟨PropertyCallParameters⟩

⟨Declarator⟩  ::=  ⟨DeclaratorVarList⟩ |
                |   ⟨DeclaratorVarList⟩ ; ⟨Ident⟩ : ⟨TypeSpecifier⟩ = ⟨Expression⟩ |

⟨DeclaratorVarList⟩  ::=  ⟨ListIdent⟩
                       |   ⟨ListIdent⟩ : ⟨SimpleTypeSpecifier⟩

⟨ListIdent⟩  ::=  ⟨Ident⟩
               |   ⟨Ident⟩ , ⟨ListIdent⟩

⟨PropertyCallParameters⟩  ::=  ( )
                            |   ( ⟨Expression⟩ ⟨ListPCPHelper⟩ )

⟨ListExpression⟩  ::=  ϵ
                    |   ⟨Expression⟩
                    |   ⟨Expression⟩ , ⟨ListExpression⟩

⟨PCPHelper⟩  ::=  , ⟨Expression⟩
               |   : ⟨SimpleTypeSpecifier⟩
               |   ; ⟨Ident⟩ : ⟨TypeSpecifier⟩ = ⟨Expression⟩
               |   | ⟨Expression⟩

⟨ListPCPHelper⟩  ::=  ϵ
                   |   ⟨PCPHelper⟩ ⟨ListPCPHelper⟩
```

```
⟨OCLLiteral⟩  ::=  ⟨String⟩
               |   ⟨OCLNumber⟩
               |   true
               |   false

⟨SimpleTypeSpecifier⟩  ::=  ⟨PathName⟩

⟨LiteralCollection⟩  ::=  ⟨CollectionKind⟩ { ⟨ListCollectionItem⟩ }
                      |   ⟨CollectionKind⟩ { }

⟨ListCollectionItem⟩  ::=  ⟨CollectionItem⟩
                       |   ⟨CollectionItem⟩ , ⟨ListCollectionItem⟩

⟨CollectionItem⟩  ::=  ⟨Expression⟩
                   |   ⟨Expression⟩ .. ⟨Expression⟩

⟨OCLNumber⟩  ::=  ⟨Integer⟩
              |   ⟨Double⟩

⟨LogicalOperator⟩  ::=  and
                    |   or
                    |   xor

⟨CollectionKind⟩  ::=  Set
                   |   Bag
                   |   Sequence
                   |   Collection

⟨EqualityOperator⟩  ::=  =
                     |   <>

⟨RelationalOperator⟩  ::=  >
                       |   >=
                       |   <
                       |   <=

⟨AddOperator⟩  ::=  +
                |   −

⟨MultiplyOperator⟩  ::=  *
                     |   /

⟨UnaryOperator⟩  ::=  −
                  |   not

⟨PostfixOperator⟩  ::=  .
                    |   ->
```