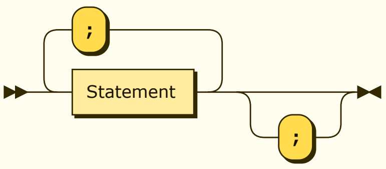
Statements:

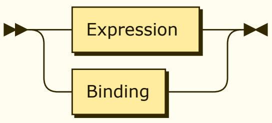


```
Statements
         ::= Statement ( ';' Statement )* ';'?
```

referenced by:

- Binding
- IfExpression
- IfNotExpression
- LambdaExpression
- LetExpression
- LoopExpression
- SwitchExpression

Statement:

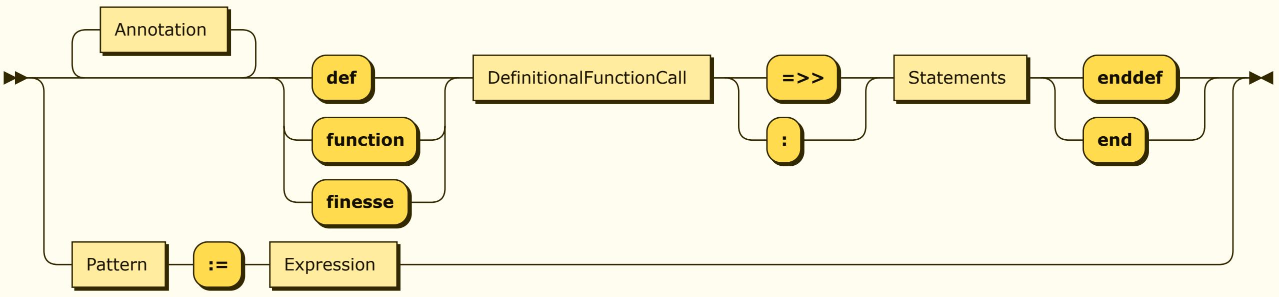


```
Statement
         ::= Expression
           | Binding
```

referenced by:

- Statements

Binding:

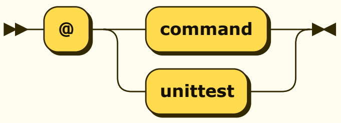


```
Binding  ::= Pattern ':=' Expression
           | Annotation* ( 'def' | 'function' | 'finesse' ) DefinitionalFunctionCall ( '=>>' | ':' ) Statements ( 'enddef' | 'end' )
```

referenced by:

- Query
- Statement

Annotation:

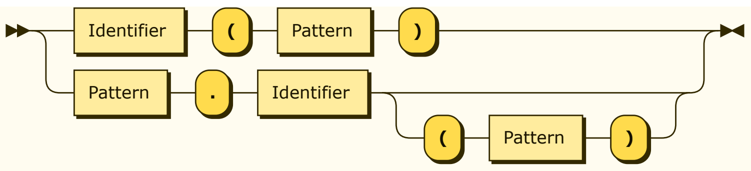


```
Annotation
         ::= '@' ( 'command' | 'unittest' )
```

referenced by:

- Binding

DefinitionalFunctionCall:

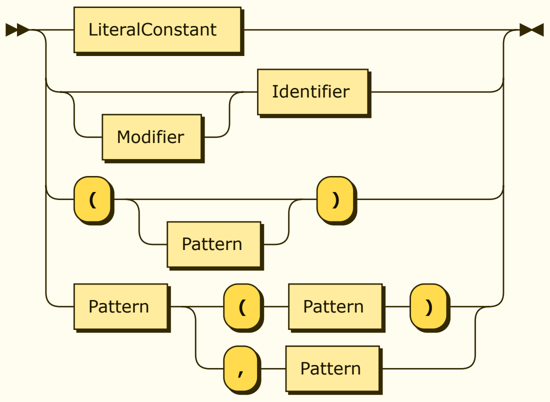


```
DefinitionalFunctionCall
         ::= Identifier '(' Pattern ')'
           | Pattern '.' Identifier ( '(' Pattern ')' )?
```

referenced by:

- Binding

Pattern:

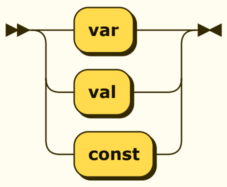


```
Pattern  ::= LiteralConstant
           | Modifier? Identifier
           | '(' Pattern? ')'
           | Pattern ( '(' Pattern ')' | ',' Pattern )
```

referenced by:

- Binding
- DefinitionalFunctionCall
- LambdaFunctionCall
- Pattern
- Query

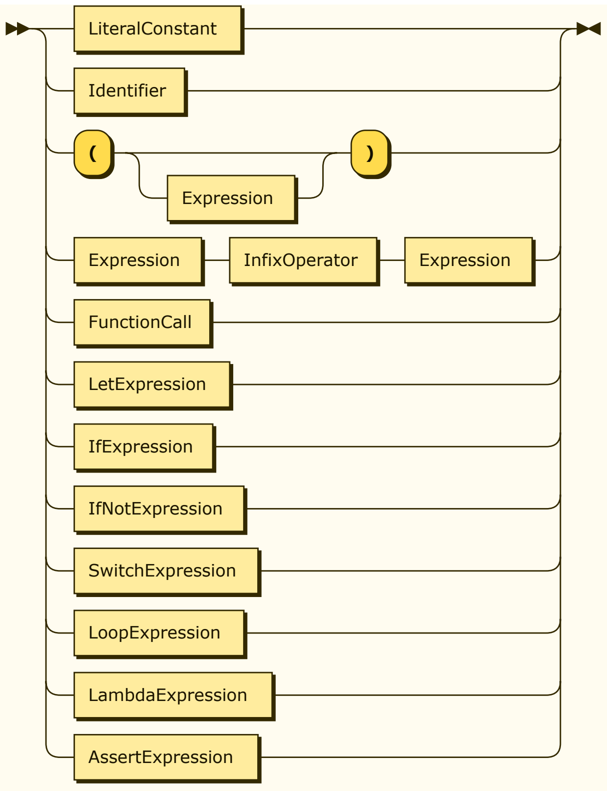
Modifer:



```
Modifer  ::= 'var'
           | 'val'
           | 'const'
```

no references

Expression:

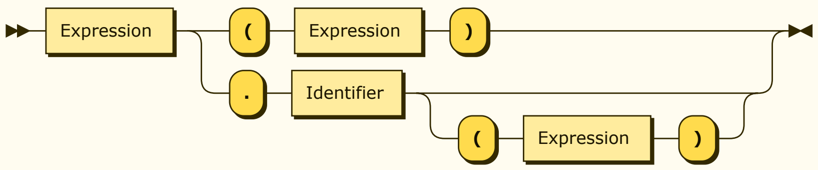


```
Expression
         ::= LiteralConstant
           | Identifier
           | '(' Expression? ')'
           | ExpressionInfixOperatorExpression
           | FunctionCall
           | LetExpression
           | IfExpression
           | IfNotExpression
           | SwitchExpression
           | LoopExpression
           | LambdaExpression
           | AssertExpression
```

referenced by:

- AssertExpression
- Binding
- Expression
- FunctionCall
- IfExpression
- IfNotExpression
- Query
- Statement
- SwitchExpression

FunctionCall:

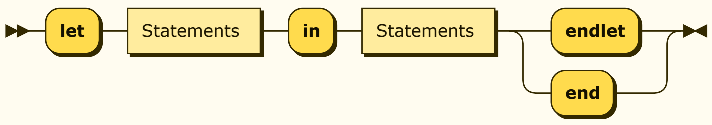


```
FunctionCall
         ::= Expression ( '(' Expression ')' | '.' Identifier ( '(' Expression ')' )? )
```

referenced by:

- Expression

LetExpression:

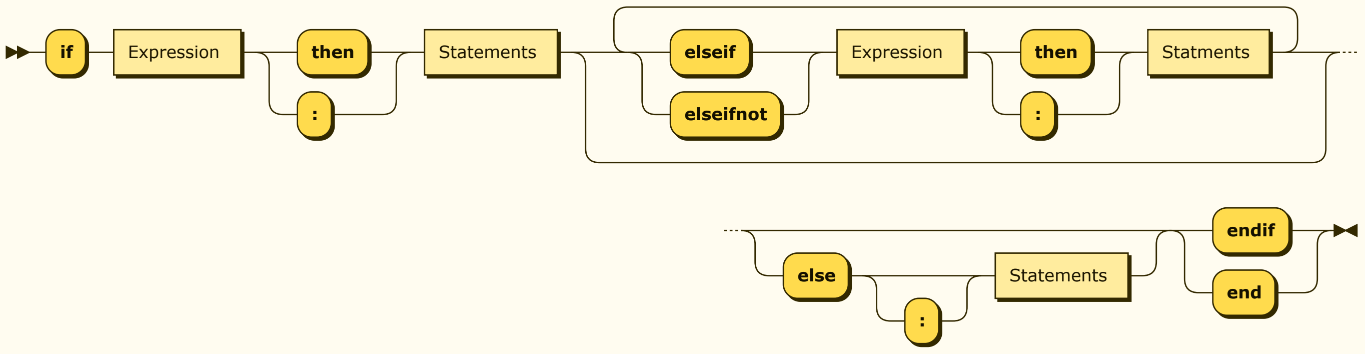


```
LetExpression
         ::= 'let' Statements 'in' Statements ( 'endlet' | 'end' )
```

referenced by:

- Expression

IfExpression:

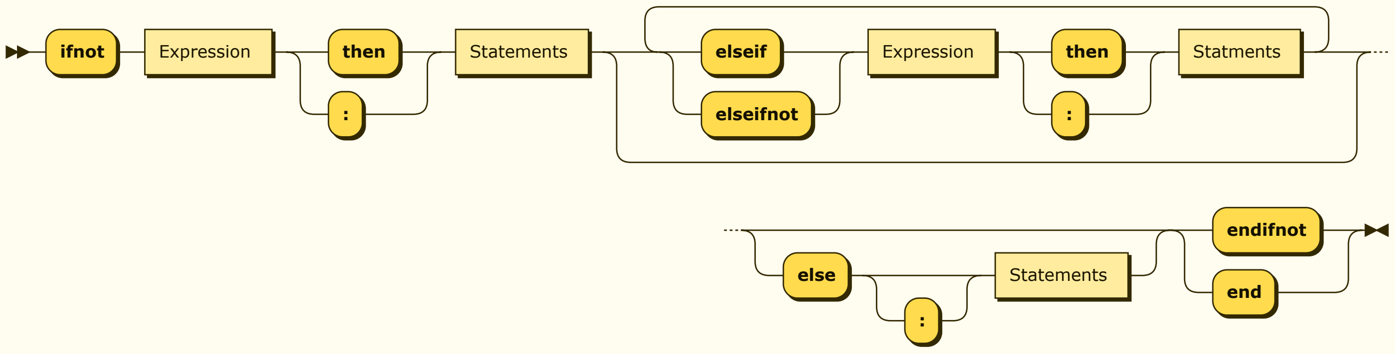


```
IfExpression
         ::= 'if' Expression ( 'then' | ':' ) Statements ( ( 'elseif' | 'elseifnot' ) Expression ( 'then' | ':' ) Statments )* ( 'else' ':'?
             Statements )? ( 'endif' | 'end' )
```

referenced by:

- Expression

IfNotExpression:

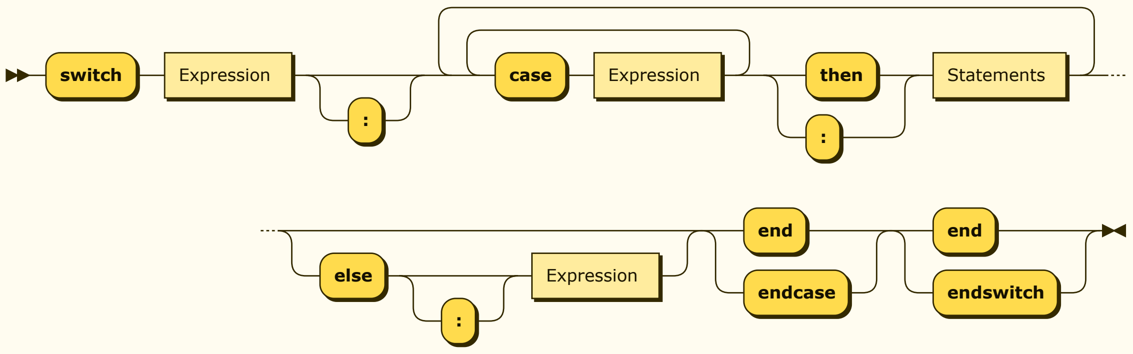


```
IfNotExpression
         ::= 'ifnot' Expression ( 'then' | ':' ) Statements ( ( 'elseif' | 'elseifnot' ) Expression ( 'then' | ':' ) Statments )* ( 'else' ':'?
             Statements )? ( 'endifnot' | 'end' )
```

referenced by:

- Expression

SwitchExpression:

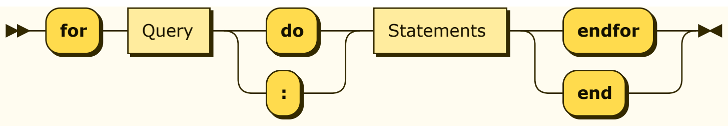


```
SwitchExpression
         ::= 'switch' Expression ':'? ( ( 'case' Expression )+ ( 'then' | ':' ) Statements )+ ( 'else' ':'? Expression )? ( 'end' | 'endcase' ) (
             'end' | 'endswitch' )
```

referenced by:

- Expression

LoopExpression:

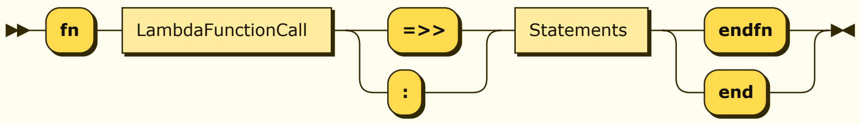


```
LoopExpression
         ::= 'for' Query ( 'do' | ':' ) Statements ( 'endfor' | 'end' )
```

referenced by:

- Expression

LambdaExpression:

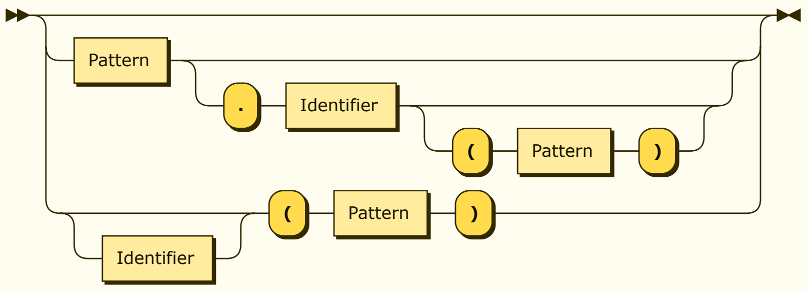


```
LambdaExpression
         ::= 'fn' LambdaFunctionCall ( '=>>' | ':' ) Statements ( 'endfn' | 'end' )
```

referenced by:

- Expression

LambdaFunctionCall:

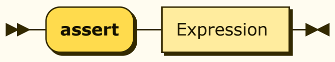


```
LambdaFunctionCall
         ::= ( Pattern ( '.' Identifier ( '(' Pattern ')' )? )? | Identifier? '(' Pattern ')' )?
```

referenced by:

- LambdaExpression

AssertExpression:

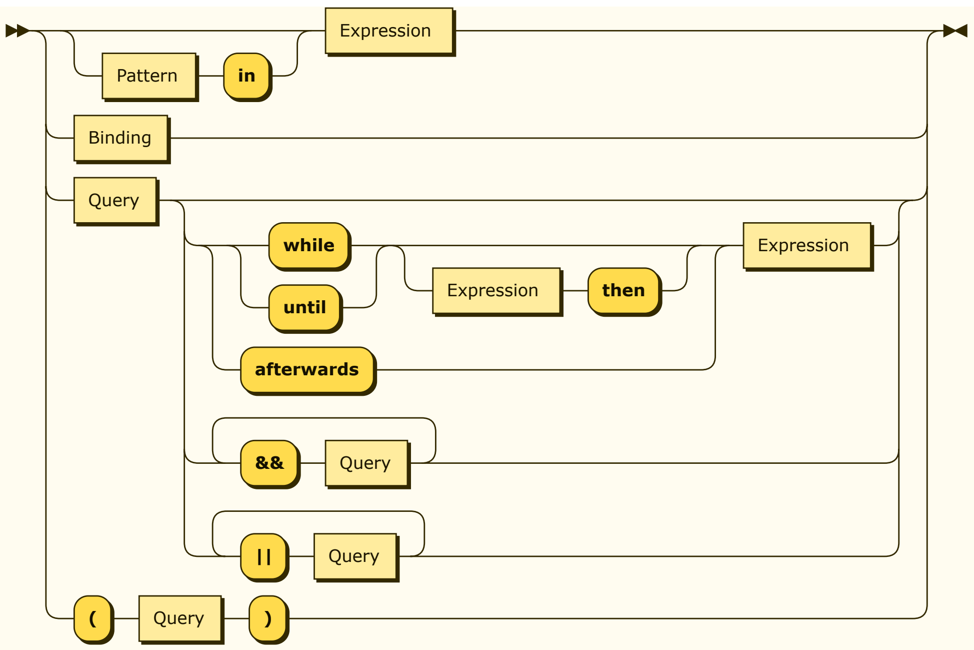


```
AssertExpression
         ::= 'assert' Expression
```

referenced by:

- Expression

Query:

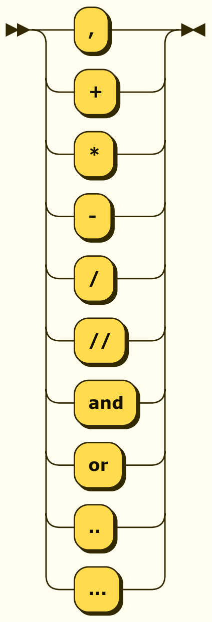


```
Query    ::= ( Pattern 'in' )? Expression
           | Binding
           | Query ( ( ( 'while' | 'until' ) ( Expression 'then' )? | 'afterwards' ) Expression | ( '&&' Query )* | ( '||' Query )+ )
           | '(' Query ')'
```

referenced by:

- LoopExpression
- Query

InfixOperator:

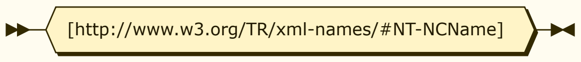


```
InfixOperator
         ::= ','
           | '+'
           | '*'
           | '-'
           | '/'
           | '//'
           | 'and'
           | 'or'
           | '..'
           | '...'
```

referenced by:

- Expression

Identifier:

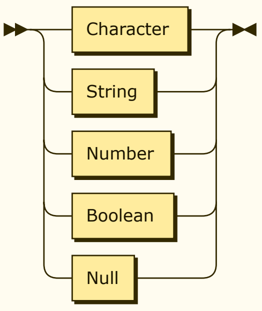


```
Identifier
         ::= [http://www.w3.org/TR/xml-names/#NT-NCName]
```

referenced by:

- DefinitionalFunctionCall
- Expression
- FunctionCall
- LambdaFunctionCall
- Pattern

LiteralConstant:



```
LiteralConstant
         ::= Character
           | String
           | Number
           | Boolean
           | Null
```

referenced by:

- Expression
- Pattern

... generated by RR - Railroad Diagram Generator