```
str:
    |[a-zA-Z][a-zA-Z_0-9]

keyword:
    | "#" | "all" | "binary" | "by" | "check" | "coeff"
    | "complements" | "contains" | "cover" | "Current"
    | "default" | "dimen" | "div" | "else" | "environ"
    | "exists" | "forall" | "if" | "IN" | "in" | "Infinity"
    | "Initial" | "INOUT" | "integer" | "less" | "LOCAL"
    | "logical" | "minimize" | "maximize" | "option" | "OUT"
    | "setof" | "shell_exitcode" | "solve_exitcode" | "solve_message"
    | "solve_result" | "solve_result_num" | "suffix" | "symbolic"
    | "table" | "then" | "union" | "until" |"while" | "within"

attributeKeyword:
    | "binary" | "integer" | "symbolic" | "in"     | "dimen"
    | "within" | "default" | "coeff" | "cover" | "obj"

nonKeyword:
    | str // if str != keyword

nonAttributekeyword:
    | str // if str != attributeKeyword

expressionReductionKeyword:
    | "max"
    | "min"
    | "prod"
    | "sum"

refName:
    |str // if str != expressionReductionKeyword

member:
    | singleMember
    | "(" singleMember ("," singleMember)* ")"

singleMember:
    | ([^"\p{Cntrl}\\]|\\[\\'"bfnrt]|\\u[a-fA-F0-9]{4})*
    | expression


**********************************************************************
declaration:
    | data
    | objective
    | constraint
    | assertion

**********************************************************************
```

DataDeclaration

data:
    | datatype nonKeyword [nonAttributeKeyword] [indexing] [attribute ("," attribute)*] ";"

datatype:
    | "set"
    | "param"
    | "var"

attribute:
    | "binary"
    | "integer"
    | "symbolic"
    | "in" setExpression
    | "dimen" (-?\d+)
    | "within" setExpression
    | (":=" | "=") (setExpression | expression)
    | "default" (setExpression | expression)
    | "<>" expression
    | ("<=" | "<" | "==" | "!=" | ">=" | ">") expression
    | "coeff" [indexing] reference expression
    | "cover" [indexing] reference
    | "obj" [indexing] reference expression

> "set oranges apples { 1 + 3 .. 10 by 4 };"
> "param x integer, in {1, 2, 3};"
> "var x obj {A} y 1;"

****

ObjectiveDeclaration

objective:
    | objectiveType nonKeyword [nonKeyword] [indexing] ":" [expression] ";"

objectiveType:
    | "maximize"
    | "minimize"

> "maximize Net_Profit;"
> "maximize x {i in A} : i ;"

****

ConstraintDeclaration

constraint:
    | ["subject to"] nonKeyword [nonKeyword] [indexing] ":" constraintExpression ";"

constraintExpression:
    | complementary
    | bounded

bounded:
    | dualBounda
    | singleBound

complementary:
    | dualBounds "complements" expression
    | expression "complements" dualBounds
    | singleBound "complements" singleBound

singleBounds:
    | expression ("<=" | "==" | ">=") expression

dualBound:
    | expression "<=" expression "<=" expression
    | expression ">=" expression ">=" expression

> "subject to Fill {i in WIDTHS}:
>   sum {j in PATTERNS} nbr[i,j] * Cut[j] >= orders[i];""
> "subject to Pri_Compl {i in PROD}:
>   Price[i] >= 0 complements
>   sum {j in ACT} io[i,j] * Level[j] >= demand[i];""

****

AssertionDeclaration

assertion:
    | "check" [indexing] [":"] [logicalExpression] ";"

> "check {i in A} : i == 1; "

****

Expression

```
expression:
    | arithmetic
    | exprFreeTokens

arithmetic:
    | prod1 (("+" | "-" | "less") prod1)*

prod1:
    | prod2 (("*" | "/" | "div" | "mod") prod2)*

prod2:
    | "+" prod3 | "-" "-" prod3 | "-" prod3

prod3:
    | prod4 (("^" | "**") prod4)*

prod4:
    | exprFreeTokens | "(" expression ")"

exprFreeTokens:
    | ifExpression
    | expressionReduction
    | functionCall
    | number
    | reference

ifExpression:
    | "if" logicalExpression "then" expression ["else" expression]

expressionReduction:
    | expressionReductionKeyword indexing expression

functionCall:
    | nonKeyword "(" [expression ("," expression)*] ")"

number:
    | -?(\d+(\.\d+)?|\d*\.\d+)([eE][+-]?\d+)?[fFdD]?

> "max(1, 2, 3)"
> "if 1 == 1 then if 1 == 1 then 2 else 3"
>" sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
>    prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]) - sum {t in 1..T}""

**********
```

LogicalExpression

```
logicalExpression:
      | or
      | lexprFreeTokens

or:
      | and (("or" | "||") and)*

and:
      | lexprFreeTokens (("and" | "&&") lexprFreeTokens)*

lexprFreeTokens:
      | reduction
      | comparition
      | inclusion
      | exclusion
      | not
      | reference
      | expression // C-like check if != 0

reduction:
      | ("forall" | "exists") indexing logicalExpression

comparision:
      | expression ("<" | "<=" | ">=" | ">" | "==" | "=" | "!=" | "<>")  expression

inclusion:
      | member "in" setExpression
      | setExpression "within" setExpression

exclusion:
      | member "not" "in" setExpression
      | setExpression "not" "within" setExpression

not:
      | ("!" | "not") lexprFreeTokens

> "1 .. 5 not within {1, 2, 3}"
> "1 in 1 .. 5 and not {1, 2, 3} within 1 .. 4 or {1, 2} not within 1 .. 2 and  x > 10 + 5"
> "1 + 3 >= 7 * z - 5"

**************************************************************************
```

SetExpression

```
setExpression:
      | setOp
      | sexprFreeTokens

setOp:
      | intersection (("union" | "diff" | "symdiff") intersection)*

intersection:
      | cross ("inter" cross)*

cross:
      | sexprFreeTokens ("cross" sexprFreeTokens)*

sexprFreeTokens:
      | setOf
      | ifSetExpr
      | reference
      | explicitSet
      | comprehensionSet
      | indexing
    | parenthesized

setOf:
      | "setof" indexing member

ifSetExpr:
      | "if" logicalExpression "then" setExpression "else" setExpression

explicitSet:
      | "{" [member ("," member)*] "}"

comprehensionSet:
      | member ".." member ["by" number]

indexing:
      | "{" sexprList [":" logicalExpression] "}"

sexprList:
      | (indexedSet | setExpression) ("," (indexedSet | setExpression))*

indexedSet:
      | nonKeyword "in" setExpression
      | "(" nonKeyword ("," nonKeyword)* ")" "in" setExpression

> "{1, 2} symdiff  {1, 2} inter {1, 2}"
> "if 1 == 1 then if 1 == 1 then {1} else {2} else {3}"
> "{i in A, (j, k) in B : i == j and i + j > k}"

**********************************************************************
```

Reference

reference:
    | indexedReference
    | simpleReference

indexedReference:
    | simplereference "[" expression ("," expression)* "]"

simpleReference:
    | refName