# 1. Входной язык.

## 1.1. Описание синтаксиса.

Целое число – это набор цифр $(57246, 7, 15\ldots)$, рациональное число – целое число или правильная дробь со знаком $(-3, 9/6, -451/7\ldots)$. Переменная – набор букв, цифр и нижних подчеркиваний, обязательно начинающийся с буквы или нижнего подчеркивания ($x$, _$3g$, $m$_3, _888). С помощью операций сложения, вычитания, деления, умножения и возведения в степень строятся многочлены:

$$4 + x/48$$

$$(x \wedge 6 - 81 * x \wedge 3)(7 - 8/6) + 6$$

$$65 * x \wedge 453/78\ldots$$

Возводить в степень можно только переменные, нельзя делить на переменные и выражения с переменными. Показатель степени должен быть натуральным. Элементарные формулы (неравенства) строятся с помощью знаков >, <, >=, <=, = и <>:

$$56 - 7 * x \wedge 2 >= 8 * x \wedge 78$$

$$x \wedge 45 * 67/(5 * 7 - 6) <> 7$$

Логические связки $and, or, -->$ и $not$ (вместо $not$ можно использовать '!') служат для конструирования сложных формул:

$$[5 > x]\ and\ [(78 + 4) * x < 5]$$

$$[-1 > 3] --> [[8 = -7]\ or\ [x \wedge 6 <= 7]]$$

В сложных формулах простые части заключаются в квадратные скобки. Кванторы 'существует x' (E x) и 'для любого x' (A x) ставятся перед формулой, а сама формула пишется в фигурных скобках:

$$E\ x\ \{56 + 7 * x \wedge 2 >= 8 * x \wedge 78\}$$

$$A\ x\ \{[5 > x]\ and\ [(78 + 4) * x < -5]\}$$

## 1.2. Грамматика входного языка:

Formula $\mapsto$ Quantor VAR '{' StatementSystem '}';

Quantor ↦ '∃' | '∀';
StatementSystem ↦ Statement | Statement Oper StatementSystem;
Oper ↦ 'and' | 'or' | '–>';
Statement ↦ '[' StatementSystem ']' | '!' Statement | Inequation;
Inequation ↦ Polynom IneqSign Polynom;
IneqSign ↦ '<' | '>' | '=' | '>=' | '<=' | '<>';
Polynom ↦ Mult | Mult PlusOp Polynom;
PlusOp ↦ '+' | '-';
Mult ↦ Factor | Factor MelOp Mult;
MelOp ↦ '*' | '/';
Factor ↦ NUMBER | '(' Polynom ')' | Power | PlusOp Factor;
Power ↦ VAR | VAR '∧' NUMBER;
VAR ↦ [ a - z A - Z _][a - z A - Z _ 0 - 9]*;
NUMBER ↦ [0 - 9]+;

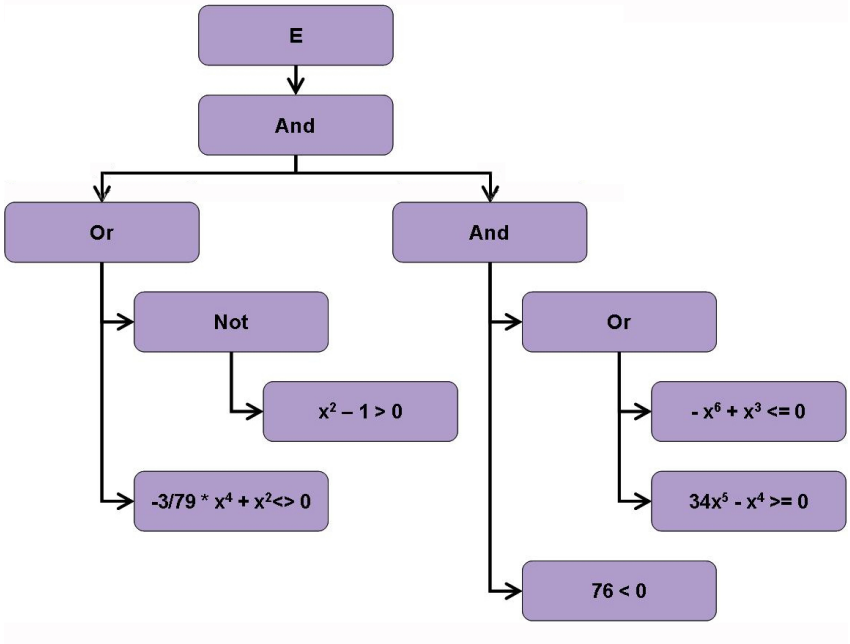
## 2. Парсер.

Парсер реализуется рекурсивно, с помощью лексера. Лексер – конечный автомат, который разбивает входную строку на токены. Парсер, состоит из двух частей, одна из них создает дерево, а другая полином. К парсеру можно подключить два модуля, один (StatementSystemBuilder) конструирует дерево, а другой (ParserTest) – выводит в консоль (формирует строку) формулу в виде скобочной последовательности, с помощью этого модуля тестировался парсер. Парсер преобразует любую строку вида '$A \to B$' (A и B – неравенства или сложные формулы) в строку '$[notA]or[B]$', а любое неравенство упрощает и переносит все в левую часть (получает выражение типа $a_i * x^i + a_{i-1} * x^{i-1} + \cdots + a_1 * x + a_0 > 0$, где вместо '>' может стоять любой другой знак).

### 2.1.Представление дерева.

Входную строку парсер преобразует в дерево, корень которого – это запись, в которой хранится квантор. Обычная вершина дерева – система утверждений. В ней лежит операция and (и), not (не) или or (или) и две ссылки на следующие вершины или листья (если операция not то одна из ссылок равна nil). Неравенство – лист дерева. Неравенство это запись, в которой хранится знак

неравенства и многочлен (если в неравенстве лежит знак '>' и многочлен A, то эта вершина соответствует неравенству $A > 0$).Многочлен – это пронумерованный с нуля массив указателей на рациональные числа, где i-ая ячейка – коэффициент при $x^i$. Рациональное число – запись, в которой хранится знак этого числа и два натуральных числа – числитель и знаменатель. Натуральное число – массив чисел типа Word. Пример дерева :


E
And
Or
And
Not
Or
$x^2 - 1 > 0$
$-x^6 + x^3 <= 0$
$-3/79 * x^4 + x^2 <> 0$
$34x^5 - x^4 >= 0$
$76 < 0$

### 2.2.Модули.

При обработке формулы парсер вызывает модуль, создающий дерево. Каждая функция парсера вызывает в соответствующем месте функцию из модуля, которая конструирует вершину дерева (неравенство, многочлен. . . ) и складывает её в специально созданный стек (есть два стека: для многочленов и для систем утверждений). Вот пример:

```
procedure ReadFormula(t : Integer);
begin
  initStek;
  currentTokenData := LexerNext(ResultType);
  CheckForError(gQuantor, ResultType, currentTokenData);
  onQuantor(t, currentTokenData);
  currentTokenData := LexerNext(ResultType);
  CheckForError(gVar, ResultType, currentTokenData);
  currentTokenData := LexerNext(ResultType);
  CheckForError(gBracketFigureOpen, ResultType, currentTokenData);
  currentTokenData := LexerNext(ResultType);
  ReadStaSyst(t + 1);
  onFormula(t);
  if errorFlag = true then
    Exit;
  CheckForError(gBracketFigureClose, ResultType, currentTokenData);
  currentTokenData := LexerNext(ResultType);
  CheckForError(gEnd, ResultType, currentTokenData);
end;
```

Процедура ReadFormula вызывает процедуры onQuantor и onFormula, которые конструируют корневую вершину дерева (первая определяет квантор, а вторая, после того как процедурой ReadStaSyst создано все остальное дерево, присваивает определенному полю вершины ссылку на верхнюю вершину дерева).

## 3. Конвертация в TEX.

Существует программа, которая конвертирует формулу, созданную парсером в TEXовский файл. Эта программа реализуется рекурсивно.

Несколько примеров того работы парсера и этой программы:

$$A\ x\ \{[5 > x]\ and\ [(78 + 4) * x < -5]\}$$

$$\forall x\ :\ \begin{cases} -x + 5\ >\ 0 \\ 82x + 5\ <\ 0 \end{cases}$$

$$E\ x\ \{[67 * x \wedge 678 <= 76 * (4 + 81 - 5 * x \wedge 7/8) - 1] - - > [1 = 1]\}$$

$$\exists x\ :\ \begin{bmatrix} \neg\ 67x^{678} + \frac{95}{2}x^7 - 6459\ \leq\ 0 \\ 0\ =\ 0 \end{bmatrix}$$

$$A\ x\ \{[6 = 5] - - > [[[8 * x - 1 >= 6]\ and\ [1 <> x]]\ or\ not\ [4 <= x/15]]\ and\ [\ not\ [-4 = 0]]\}$$

$$\forall x\ :\ \begin{bmatrix} \neg\ 1\ =\ 0 \\ \begin{cases} \begin{bmatrix} \begin{cases} 8x - 7\ \geq\ 0 \\ -x + 1\ \neq\ 0 \end{cases} \\ \neg\ -\frac{1}{15}x + 4\ \leq\ 0 \end{bmatrix} \\ \neg\ -4\ =\ 0 \end{cases} \end{bmatrix}$$