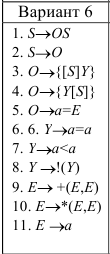
Пункт 1



Преобразуем данную грамматику G1 в G2, которая будет LL(1) грамматикой, проведем левую факторизацию.

G2

1. S -> OZ

2. Z -> S

3. Z -> e

4. O -> {P

5. P -> [S]Y}

6. P -> Y[S]}

7. O -> a=E

8. Y -> aW

9. W -> =a

10.W -> <a

11.Y -> !(Y)

12.E -> +(E,E)

13.E -> \*(E,E)

14.E -> a

Пункт 2

Найдем множество первых

FIRST(S) = { a = FIRST(O)

FIRST(Z) = { a e

FIRST(O) = { a

FIRST(P) = [ a !

FIRST(Y) = a !

FIRST(W) = < =

FIRST(E) = + \* a

Пункт 3

Найдем множество следующих

FOLLOW(S) = -| ]

FOLLOW(Z) = -| ]

FOLLOW(O) = { a e

FOLLOW(P) = { a e

FOLLOW(Y) = ) [ }

FOLLOW(W) = ) [ }

FOLLOW(E) = { a e , )

Пункт 4

Определим множество выбора

1. SELECT(S -> OZ) = { a

2. SELECT(Z -> S) = { a

3. SELECT(Z -> e) = -| ]

4. SELECT(O -> {P ) = {

5. SELECT(P -> [S]Y}) = [

6. SELECT(P -> Y[S]}) = a !

7. SELECT(O -> a=E) = a

8. SELECT(Y -> aW ) = a

9. SELECT(W -> =a) = =

10.SELECT(W -> <a) = <

11.SELECT(Y -> !(Y)) = !

12.SELECT(E -> +(E,E) = +

13.SELECT(E -> \*(E,E)) = \*

14.SELECT(E -> a) = a

Множества выбора у правил с одинаковой левой частью не пересекаются это значит что грамматика и правда LL(1)Пункт 5

Реализация программы-распознавателя методом рекурсивного спуска

G\_NOT\_TERMINALS = **'SZOPYWE'**

G\_TERMINALS = **'{[a=<!+\*]}(,)'**

**class** Recurse:

**def** \_\_init\_\_(self, chain: str):

self.chain = chain

**def** S(self):

FIRST = {

**'{'**: **'OZ'**,

**'a'**: **'OZ'**,

}

**try**:

inSym = self.chain[0]

print(**f'S -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** Z(self):

FIRST = {

**'{'**: **'S'**,

**'a'**: **'S'**

}

**try**:

inSym = self.chain[0]

print(**f'Z -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**if** self.chain[0] **in ']$'**:

print(**f'Z -> e'**)

**else**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** O(self):

FIRST = {

**'{'**: **'{P'**,

**'a'**: **'a=E'**

}

**try**:

inSym = self.chain[0]

print(**f'O -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** P(self):

FIRST = {

**'['**: **'[S]Y}'**,

**'a'**: **'Y[S]}'**,

**'!'**: **'Y[S]}'**

}

**try**:

inSym = self.chain[0]

print(**f'P -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** Y(self):

FIRST = {

**'a'**: **'aW'**,

**'!'**: **'!(Y)'**

}

**try**:

inSym = self.chain[0]

print(**f'Y -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** W(self):

FIRST = {

**'='**: **'=a'**,

**'<'**: **'<a'**

}

**try**:

inSym = self.chain[0]

print(**f'Y -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** E(self):

FIRST = {

**'a'**: **'a'**,

**'+'**: **'+(E,E)'**,

**'\*'**: **'\*(E,E)'**

}

**try**:

inSym = self.chain[0]

print(**f'E -> {**FIRST[inSym]**}'**)

self.parse(FIRST[inSym])

**except**:

**raise** MP.ParseError() *# В данном случае из нетерминала не выводится пустая цепочка*

**def** parse(self, u:str):

v = u

**while** v:

X = v[0]

z = v[1::]

**if** X **in** G\_TERMINALS:

**if** X != self.chain[0]:

**raise** MP.ParseError()

**else**:

self.chain = self.chain[1::]

**else**:

{

**'S'**: self.S,

**'Z'**: self.Z,

**'O'**: self.O,

**'P'**: self.P,

**'Y'**: self.Y,

**'W'**: self.W,

**'E'**: self.E,

}[X]()

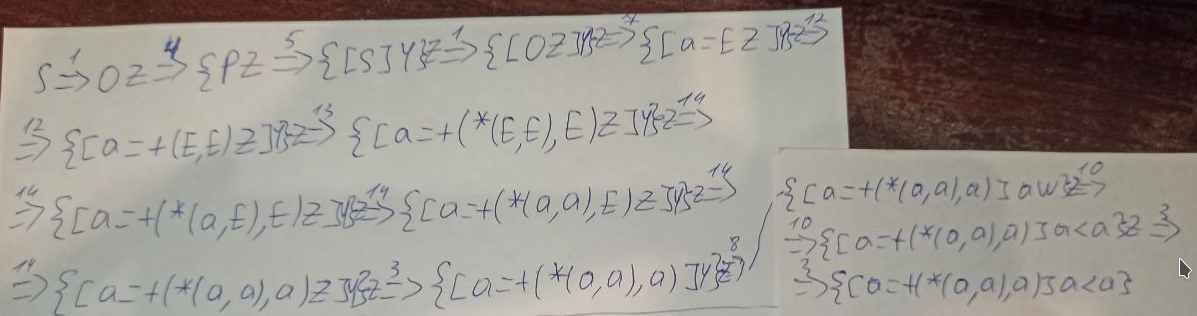
v = z

Пункт 6

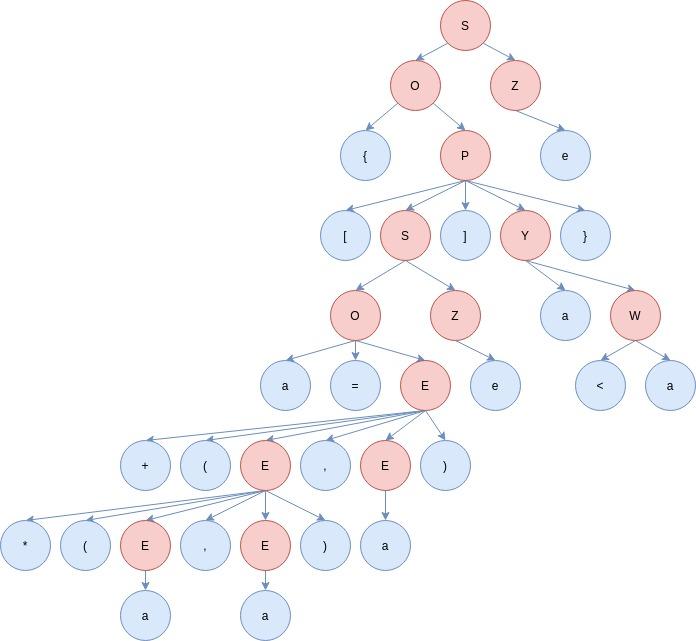
Принадлежащие языку

{[a=+(\*(a,a),a)]a<a}

Левый вывод

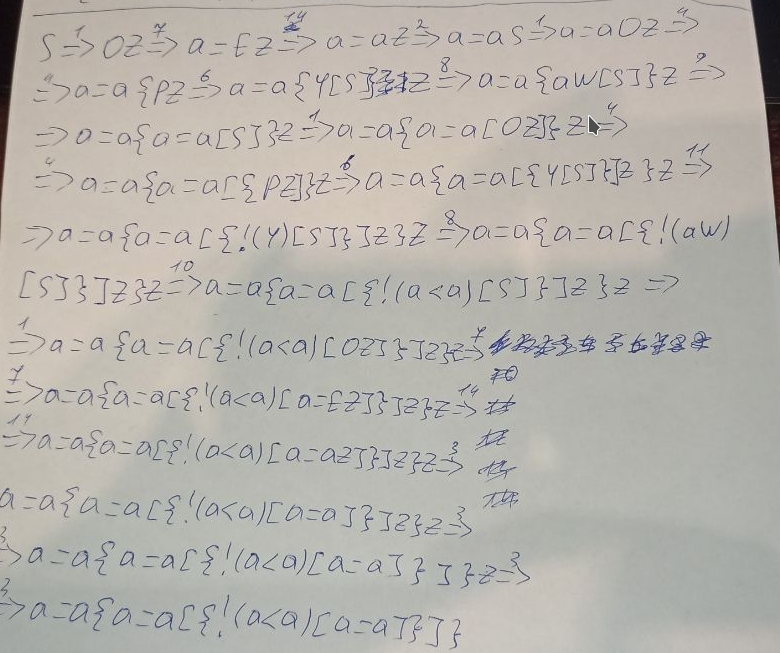


Дерево вывода



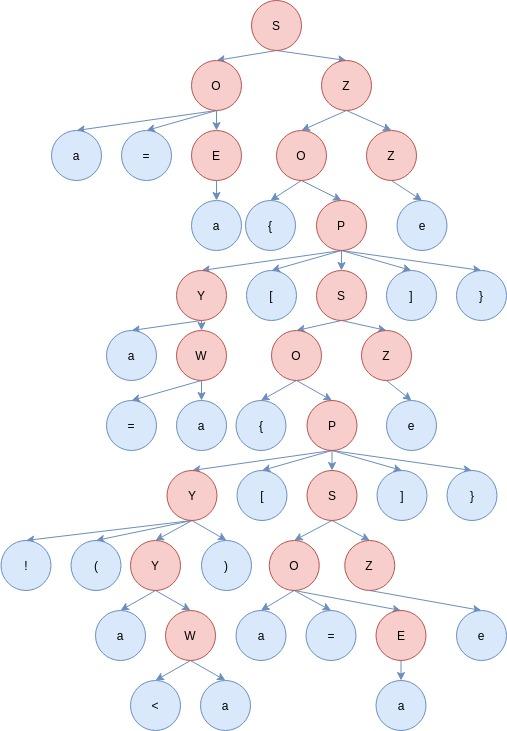
a=a{a=a[{!(a<a)[a=a]}]}

Левый вывод



Дерево вывода ниже

Дерево вывода



Не принадлежащие языку

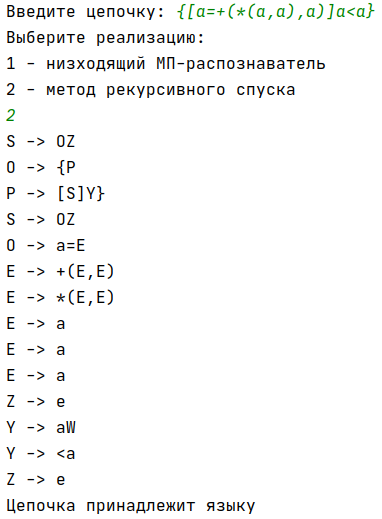
a

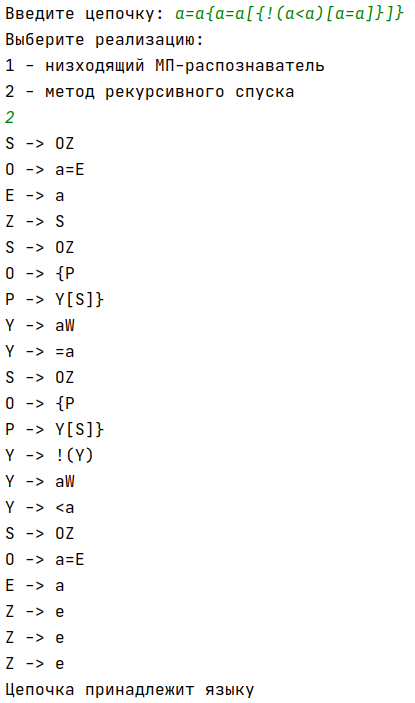
{a=+(\*(a,a),a)]}

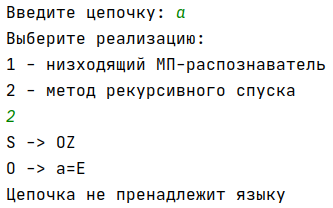
{[}

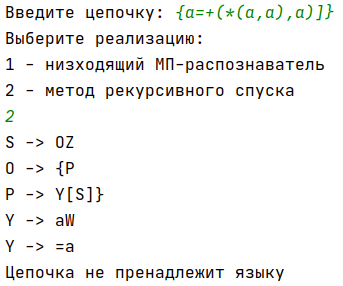
Пункт 7

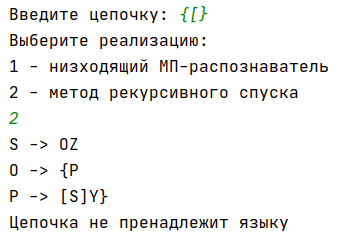
Проверим рекурсивный спуск











Пункт 8

Составим таблицу

|  | { | ] | [ | a | = | < | ! | + | \* | -| |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | S->OZ |  |  | S -> OZ |  |  |  |  |  |  |
| Z | Z->S | Z->e |  | Z -> S |  |  |  |  |  | Z->e |
| O | O->{P |  |  | O -> a=E |  |  |  |  |  |  |
| P |  |  | P->[S]Y} | P->Y[S]} |  |  | P->Y[S]} |  |  |  |
| Y |  |  |  | Y -> aW |  |  | Y->!(Y) |  |  |  |
| W |  |  |  |  | W->=a | W-><a |  |  |  |  |
| E |  |  |  | E -> a |  |  |  | E->+(E,E) | E->\*(E,E) |  |
| { | Выброс |  |  |  |  |  |  |  |  |  |
| ] |  | Выброс |  |  |  |  |  |  |  |  |
| [ |  |  | Выброс |  |  |  |  |  |  |  |
| a |  |  |  | Выброс |  |  |  |  |  |  |
| = |  |  |  |  | Выброс |  |  |  |  |  |
| < |  |  |  |  |  | Выброс |  |  |  |  |
| ! |  |  |  |  |  |  | Выброс |  |  |  |
| + |  |  |  |  |  |  |  | Выброс |  |  |
| \* |  |  |  |  |  |  |  |  | Выброс |  |
| <> |  |  |  |  |  |  |  |  |  | Допустить |

Пункт 9

Реализуем

G\_NOT\_TERMINALS = **'SZOPYWE'**

G\_TERMINALS = **'{[a=<!+\*]}(,)'**

RULES = [(**'S'**, **'OZ'**),

(**'Z'**, **'S'**),

(**'Z'**, **''**),

(**'O'**, **'{P'**),

(**'P'**, **'[S]Y}'**),

(**'P'**, **'Y[S]}'**),

(**'O'**, **'a=E'**),

(**'Y'**, **'aW'**),

(**'W'**, **'=a'**),

(**'W'**, **'<a'**),

(**'Y'**, **'!(Y)'**),

(**'E'**, **'+(E,E)'**),

(**'E'**, **'\*(E,E)'**),

(**'E'**, **'a'**)]

**class** MP:

**class** ParseError(Exception):

**def** \_\_init\_\_(self):

super().\_\_init\_\_(**'Цепочка не пренадлежит языку'**)

TABLE = {

**'S'**: {

**'{'**: RULES[0],

**'a'**: RULES[0]

},

**'Z'**: {

**'{'**: RULES[1],

**'a'**: RULES[1],

**'$'**: RULES[2],

**']'**: RULES[2]

},

**'O'**: {

**'{'**: RULES[3],

**'a'**: RULES[6]

},

**'P'**: {

**'['**: RULES[4],

**'a'**: RULES[5],

**'!'**: RULES[5]

},

**'Y'**: {

**'a'**: RULES[7],

**'!'**: RULES[10]

},

**'W'**: {

**'='**: RULES[8],

**'<'**: RULES[9]

},

**'E'**: {

**'a'**: RULES[13],

**'+'**: RULES[11],

**'\*'**: RULES[12]

}

}

**def** \_print\_rule(self, rule: (str, str)):

rule\_num = RULES.index(rule) + 1

print(**f'{" " if** rule\_num < 10 **else ""}{**rule\_num**}. {**rule[0]**} -> {**rule[1] **if** rule[1] **else "e"}'**)

**def** check(self, chain: str):

stack = [**'$'**, **'S'**]

chain += **'$'**

indexInSym = 0

run\_flag = **True**

print(**'Вывод'**)

**while** run\_flag:

X = stack[-1]

**if** (X **in** G\_TERMINALS) **or** (X == **'$'**):

**if** X == chain[indexInSym]:

stack.pop()

indexInSym += 1

**else**:

**raise** MP.ParseError()

**else**:

**try**:

rule: (str, str) = MP.TABLE[X][chain[indexInSym]] *# тут вылетит ошибка если ячейка таблицы пуста*

stack.pop()

stack += rule[1][::-1]

self.\_print\_rule(rule)

**except**:

**raise** MP.ParseError()

run\_flag = (X != **'$'**)

Пункт 10

Протестируем его

