Lingwistyka Matematyczna

Laboratorium

Rok akademicki 2023/2024

Zadanie 2b

Analizator składniowy LL(1)

Mikołaj Rajczyk 254403

Zadaniem 2 laboratorium jest napisanie programu komputerowego w dowolnym języku programowania będącego analizatorem składniowym LL(1) realizującym algorytm rozbioru generacyjnego zstępującego z wyprzedzeniem o jeden symbol LL(1) do następującej gramatyki

S ::= W ; Z
Z ::= W ; Z | ε
W ::= P | POW
P ::= R | (W)
R ::= L | L.L
L ::= C | CL
C ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
O ::= * | : | + | - | ^

Powyższa gramatyka umożliwia budowanie ciągów składających się z liczb rzeczywistych i operacji arytmetycznych (zdań arytmetycznych) zakończonych średnikiem, np.:
(1.2 ∗ 3) + 5 − (23.4 + 3) ^3; 8 : 3;

1. Sprawdzenie, czy gramatyka jest zgodna z LL(1):

FIRST(S) = FIRST(W) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}
FIRST(Z) = FIRST(W) ∪ { **ε** } = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (, ε }
FIRST(W) = FIRST(P) ∪ FIRST(P) = FIRST(P) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}
FIRST(P) = FIRST(R) ∪ { (} = FIRST(C) ∪ { (} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}
FIRST(R) = FIRST(L) ∪ FIRST(L) = FIRST(L) = FIRST(C) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
FIRST(L) = FIRST(C) ∪ FIRST(C) = FIRST(C) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
FIRST(C) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
FIRST(O) = {*, :, +, -, ^)

FOLLOW(Z) = FOLLOW(S) = {∅}

Sprawdzenie zgodności z I regułą:

S -> brak alternatywy, reguła spełniona
Z -> FIRST(W) ∩ {ε} = {∅} – reguła spełniona
W -> FIRST(P) ∩ FIRST(P) = FIRST(P) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (} ≠ {∅} – reguła niespełniona
P -> FIRST(R) ∩ { (} = {0, 1, 2 , 3, 4, 5, 6, 7, 8, 9) ∩ { (} = {∅} – reguła spełniona
R -> FIRST(L) ∩ FIRST(L) = FIRST(L) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ≠ {∅} – reguła niespełniona
L -> FIRST(C) ∩ FIRST(C) = FIRST(C) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ≠ {∅} – reguła niespełniona
C -> W każdej z alternatyw produkcji występuje **inny pojedynczy symbol terminalny**, więc iloczyn każdej z par różnych symboli będzie symbolem pustym. Oznacza to, że reguła jest spełniona.
O -> W każdej z alternatyw produkcji występuje **inny pojedynczy symbol terminalny**, więc iloczyn każdej z par różnych symboli będzie symbolem pustym. Oznacza to, że reguła jest spełniona.

Sprawdzenie zgodności z II regułą (Tylko produkcja Z zawiera symbol pusty):

FOLLOW(Z) ∩ FIRST(Z) = { ε } ∩ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (, ε } = { ε } występowanie symbolu pustego (ε) jest równoważne z brakiem symbolu. Reguła jest spełniona.

Z powodu niespełnienia I zasady przez wszystkie produkcje analizowana gramatyka nie spełnia założeń LL(1). Nie można jednoznacznie określić produkcji do rozwinięcia symboli nieterminalnych w drzewie składniowym, należy więc przepisać tę produkcję, a decyzję o wyborze odłożyć do chwili aż przeczytanych zostanie więcej znaków wejściowych - możliwa jest wielokrotna faktoryzacja.

Wzór na lewostronną faktoryzację jest następujący:

$A ::= \alpha\zeta_1 \mid \alpha\zeta_2 \mid \ldots \mid \alpha\zeta_n \quad \Rightarrow \quad A ::= \alpha A'$

$A' ::= \zeta_1 \mid \zeta_2 \mid \ldots \mid \zeta_n$

2. Poprawienie gramatyki:

a) Można uprościć produkcje S, Z:
S ::= W ; Z
Z ::= S | ε

b) W produkcji W sekwencja **P** występuje po obu stronach alternatywy; prawa strona alternatywy składa się dodatkowo z sekwencji **OW**. Pomimo różnych sekwencji, nie można jednoznacznie wybrać na podstawie pierwszego symbolu, które rozwinięcie wybrać. W celu poprawy produkcji należy zastosować lewostronną faktoryzację.
W ::= PW'
W' ::= OW | ε

c) W produkcji R sekwencja L występuje po obu stronach alternatywy; prawa strona alternatywy składa się dodatkowo z sekwencji .L. Pomimo różnych sekwencji, nie można jednoznacznie wybrać na podstawie pierwszego symbolu, które rozwinięcie wybrać. W celu poprawy produkcji należy zastosować lewostronną faktoryzację.
R ::= LR'
R' ::= L | ε

d) W produkcji R sekwencja C występuje po obu stronach alternatywy; prawa strona alternatywy składa się dodatkowo z sekwencji L. Pomimo różnych sekwencji, nie można jednoznacznie wybrać na podstawie pierwszego symbolu, które rozwinięcie wybrać. W celu poprawy produkcji należy zastosować lewostronną faktoryzację.
L ::= CL'
L' ::= L | ε

Poprawiona gramatyka:

S ::= W ; Z
Z ::= S | ε
W ::= PW'
W' ::= OW | ε
P ::= R | (W)
R ::= LR'
R' ::= .L | ε
L ::= CL'
L' ::= L | ε
C ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
O ::= * | : | + | - | ^

FIRST(S) = FIRST(W) ∪ { ε } = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (, ε}
FIRST(Z) = FIRST(W) ∪ { **ε** } = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (, ε }
FIRST(W) = FIRST(P) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}
FIRST(W') = FIRST(O) ∪ { ε } = {*, :, +, -, ^, ε }
FIRST(P) = FIRST(R) ∪ { (} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}
FIRST(R) = FIRST(L) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
FIRST(R') = {.} ∪ { ε } = {., ε}
FIRST(L) = FIRST(C) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
FIRST(L') = FIRST(L) ∪ { ε } = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ε}
FIRST(C) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
FIRST(O) = {*, :, +, -, ^)

FOLLOW(L') = FOLLOW(L) = FIRST(R') ∪ FOLLOW(R') = FIRST(R') ∪ FOLLOW(R) =
= FIRST(R') ∪ FOLLOW(P) = FIRST(R') ∪ FIRST(W') ∪ FOLLOW(W') =
= FIRST(R') ∪ FIRST(W') ∪ FOLLOW(W) = FIRST(R') ∪ FIRST(W') ∪ {;} ∪ {)} =

= {., ε } ∪ {*, :, +, -, ^, ε} ∪ {;} ∪ {)} = {., *, :, +, -, ^, ε, ;,) }

FOLLOW(R') = FOLLOW(R) = FOLLOW(P) = FIRST(W') ∪ FOLLOW(W') =
= FIRST(W') ∪ FOLLOW(W) = FIRST(W') ∪ {;} ∪ {)} = {*, :, +, -, ^, ε } ∪ {;} ∪ {)} =
= {*, :, +, -, ^, ε, ;,) }

FOLLOW(W') = FOLLOW(W) = {;} ∪ {)} = {;,) }

FOLLOW(Z) = FOLLOW(S) = {∅}

Sprawdzenie zgodności z I regułą:

S -> brak alternatywy, reguła spełniona
Z -> FIRST(S) ∩ {**ε**} = {∅}, reguła spełniona
W -> brak alternatywy, reguła spełniona
W' -> FIRST(W) ∩ {**ε**} = {∅}, reguła spełniona
P -> FIRST(R) ∩ {(} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∩ {(} = {∅}, reguła spełniona
R -> brak alternatywy, reguła spełniona
R' -> {.} ∩ {**ε**} = {∅}, reguła spełniona
L -> brak alternatywy, reguła spełniona
L' -> FIRST(L) ∩ {**ε**} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∩ {**ε**} = {∅}, reguła spełniona
C -> W każdej z alternatyw produkcji występuje **inny pojedynczy symbol terminalny**, więc iloczyn każdej z par różnych symboli będzie symbolem pustym. Oznacza to, że reguła jest spełniona.
O -> W każdej z alternatyw produkcji występuje **inny pojedynczy symbol terminalny**, więc iloczyn każdej z par różnych symboli będzie symbolem pustym. Oznacza to, że reguła jest spełniona.

Sprawdzenie zgodności z II regułą (Produkcje S, W', R', L' zawierają symbol pusty):

FOLLOW(Z) ∩ FIRST(Z) = { ε } ∩ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (, ε } = { ε } występowanie symbolu pustego (ε) jest równoważne z brakiem symbolu. Reguła jest spełniona.

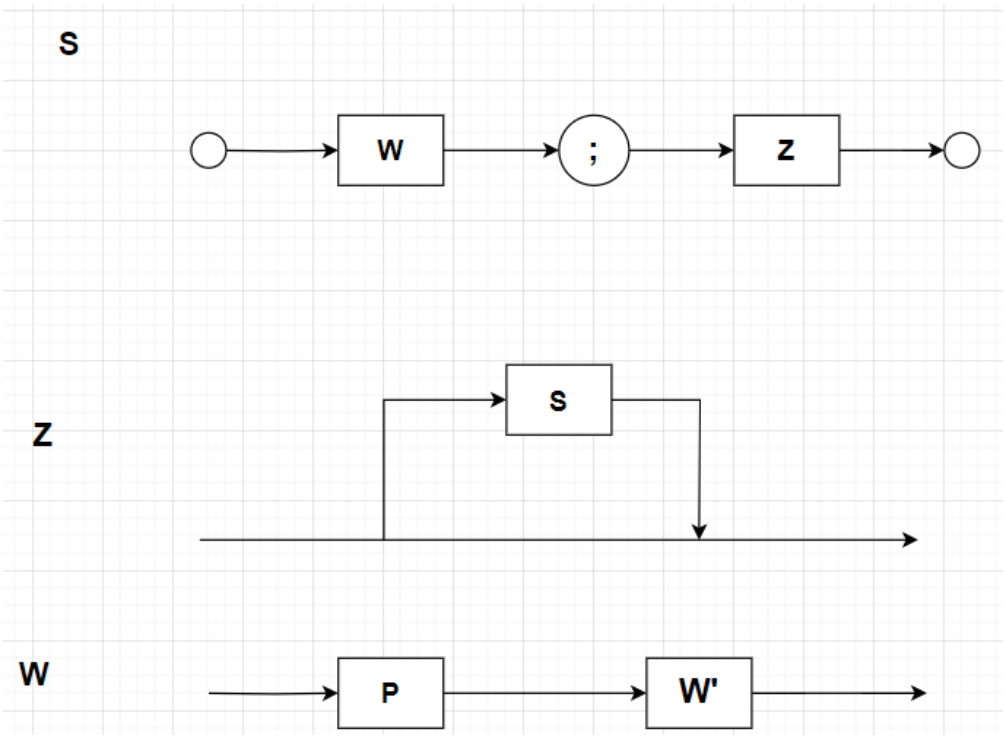
FOLLOW(W') ∩ FIRST(W') = {;,) } ∩ {*, :, +, -, ^, ε } = {∅} – reguła spełniona

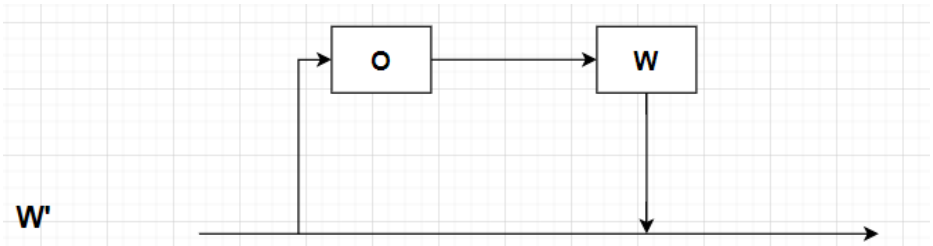
FOLLOW(R') ∩ FIRST(R') = {*, :, +, -, ^, ε, ;,) } ∩ {., ε} = { **ε** } - występowanie symbolu pustego (ε) jest równoważne z brakiem symbolu. Reguła jest spełniona.

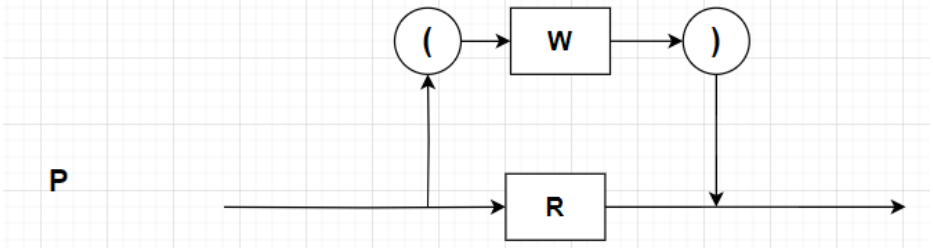
FOLLOW(L') ∩ FIRST(L') = {., *, :, +, -, ^, ε, ;,) } ∩ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ε} = { **ε** } - występowanie symbolu pustego (ε) jest równoważne z brakiem symbolu. Reguła jest spełniona.

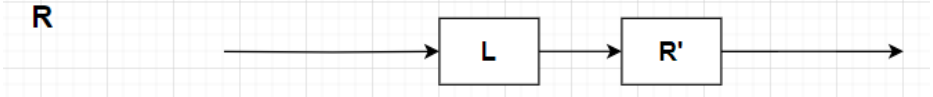
<u>Uproszczona gramatyka spełnia zasady LL(1).</u>

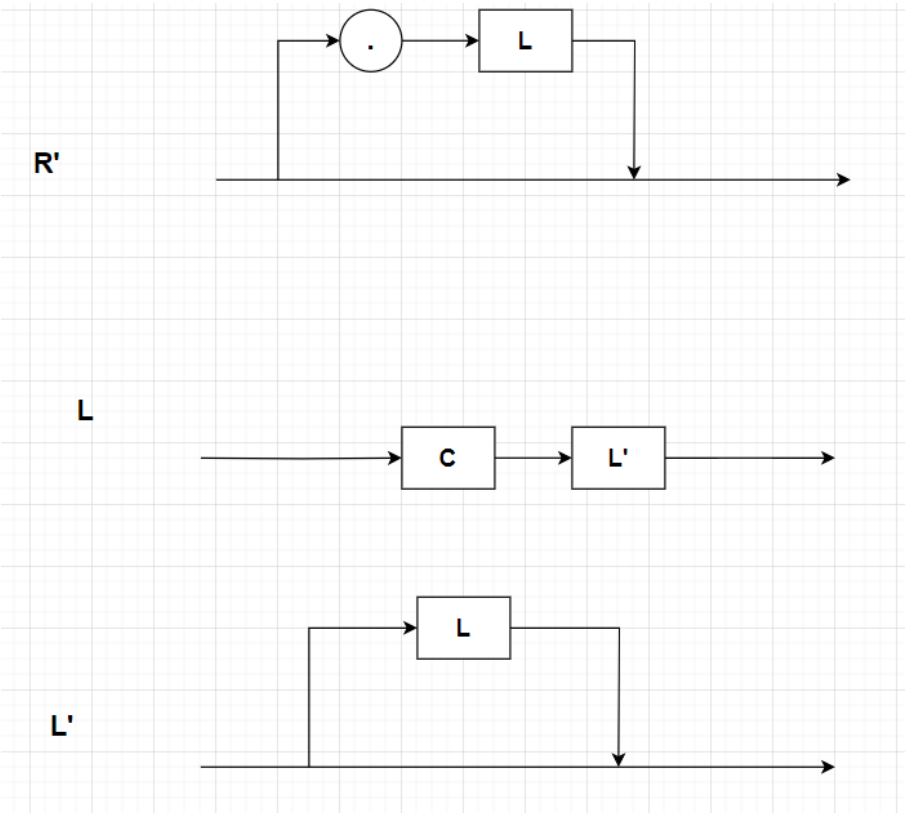
3. Diagramy składni

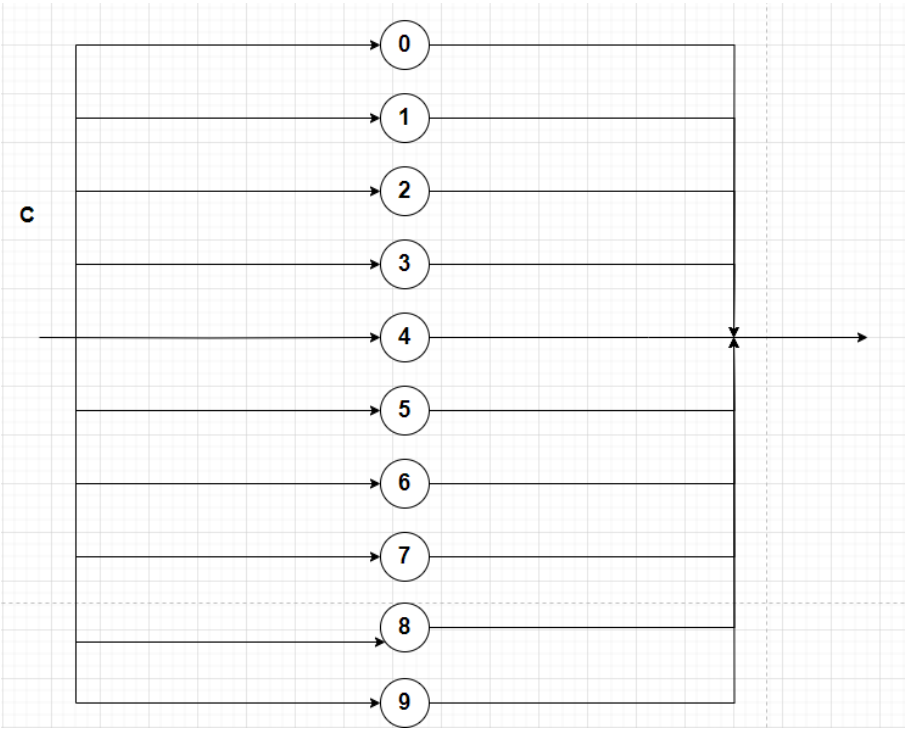


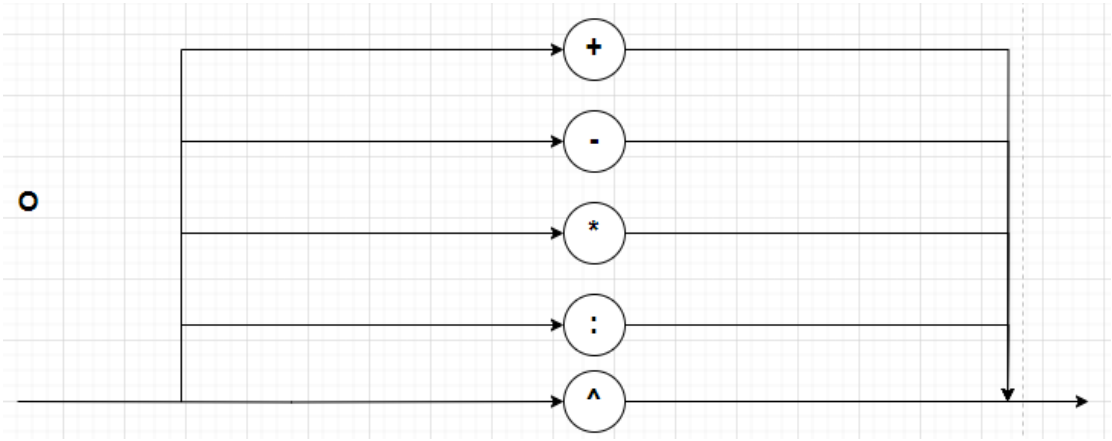






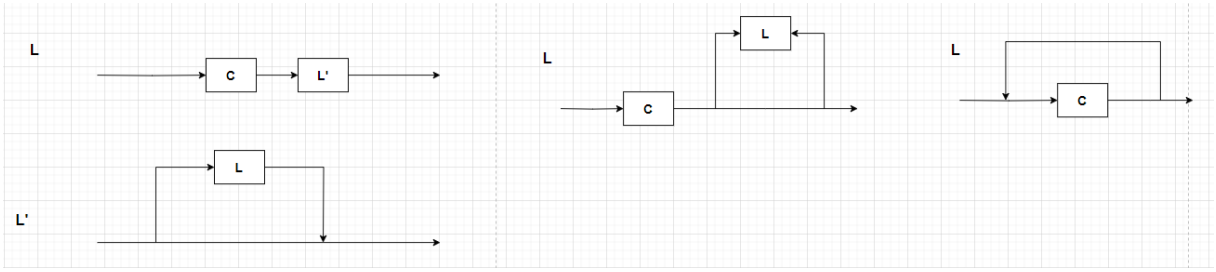

R'
.
L
L
C
L'
L'
L

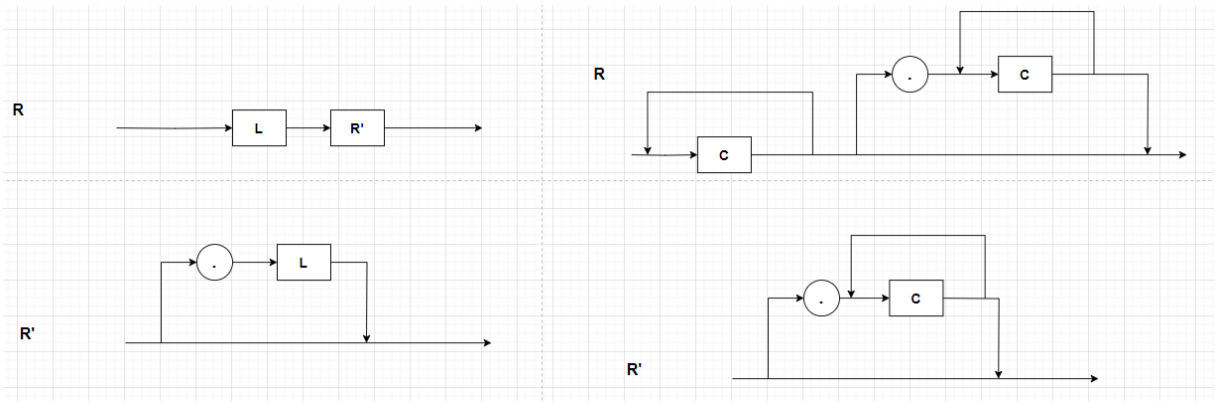

C
0
1
2
3
4
5
6
7
8
9

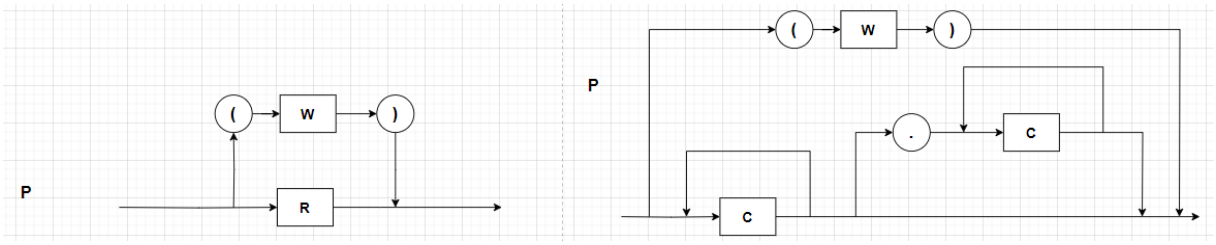


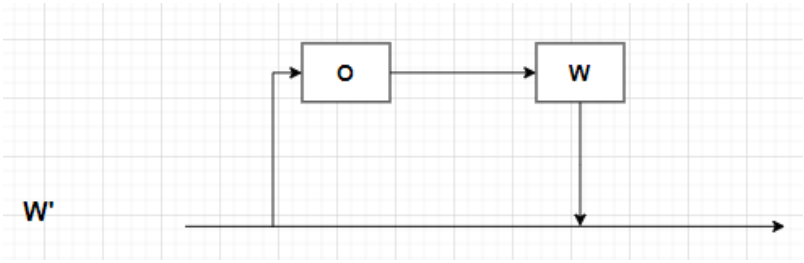
Upraszczanie diagramów

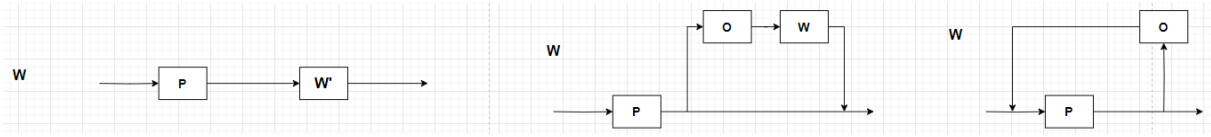
(Dla czytelności nie podstawiam diagramów produkcji C oraz O, które dają w rezultacie pojedyncze symbole terminalne.)

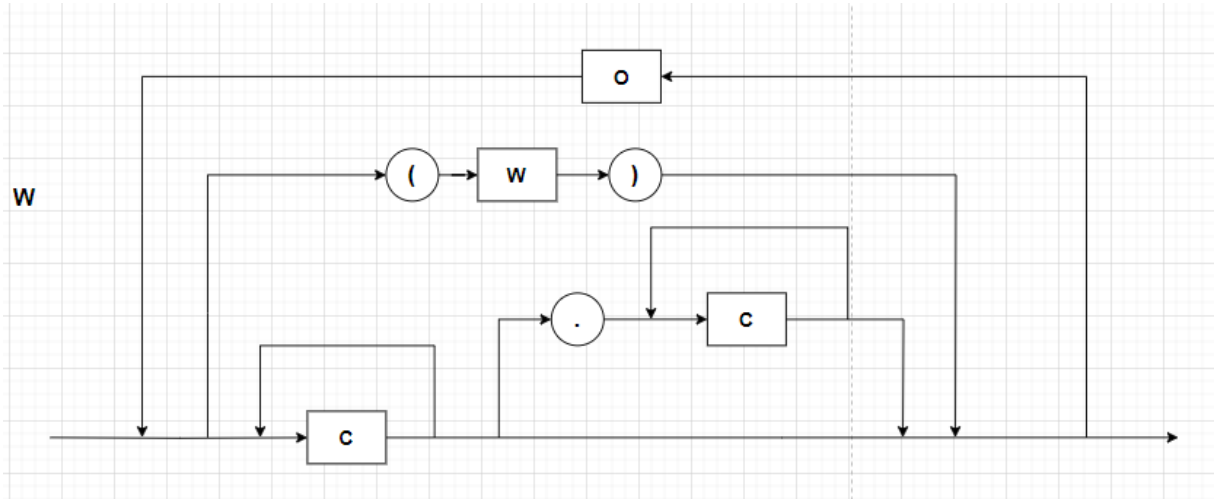


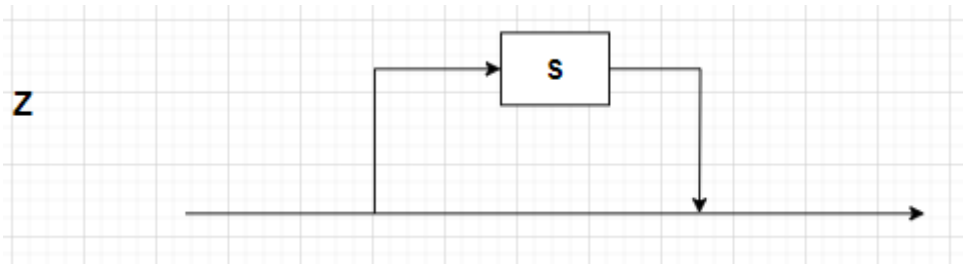


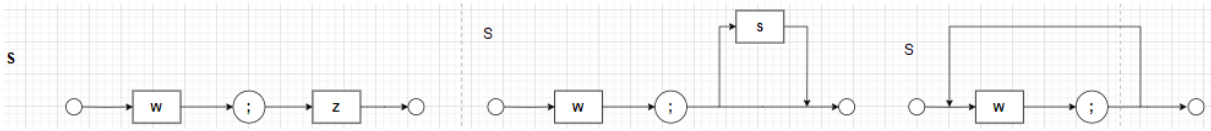



W'
O
W

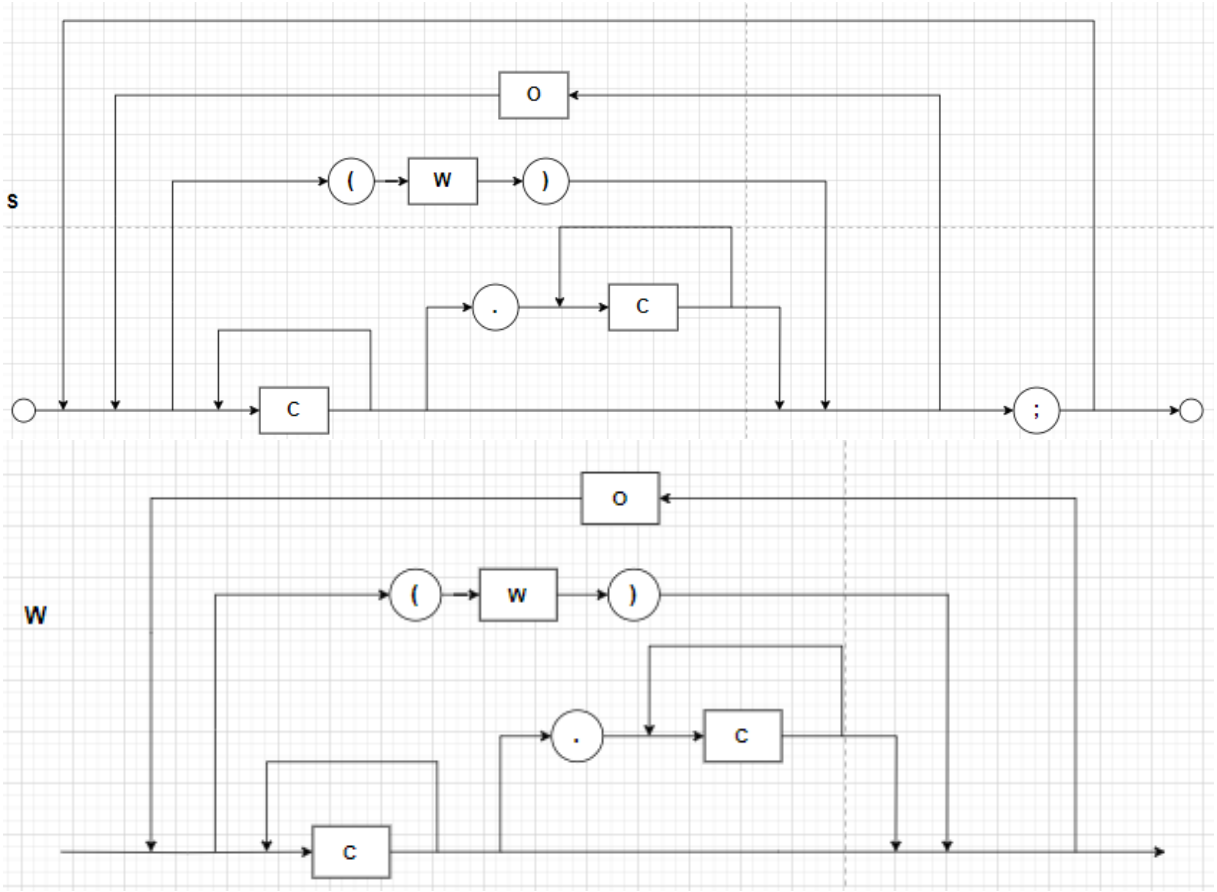

W
P
W'
W
O
W
P
W
O
P


W
O
(
W
)
.
C
C


Z
S


S
W
;
Z
S
W
;
S
S
W
;

Po finalnym podstawieniu ostateczne diagramy prezentują się następująco:
(Dla czytelności nie podstawiam diagramów produkcji C oraz O, które dają w rezultacie pojedyncze symbole terminalne.)



4. Sprawdzenie poprawności działania analizatora składniowego

Słowo wejściowe: 2+2;

Wynik działania programu:

```
Enter symbols string: 2+2;
Entered string belongs to grammar
```

Słowo wejściowe: (2+2);

Wynik działania programu:

```
Enter symbols string: (2+2);
Entered string belongs to grammar
```

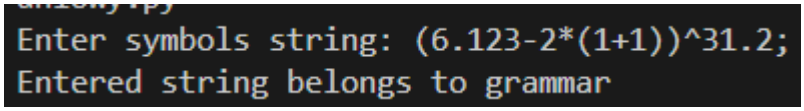
Słowo wejściowe: (9+11.2)*2;

Wynik działania programu:

```
Enter symbols string: (9+11.2)*2;
Entered string belongs to grammar
```

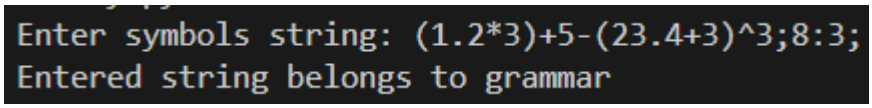
Słowo wejściowe: (6.123-2*(1+1))^31.2;

Wynik działania programu:



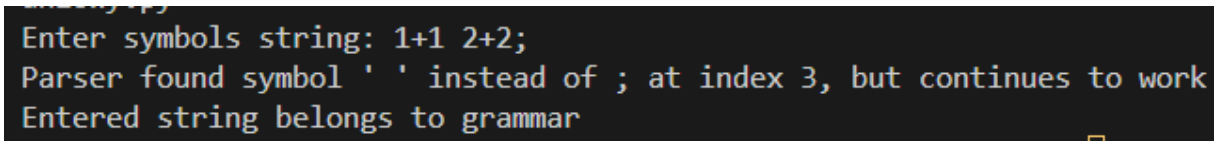
Słowo wejściowe: (1.2*3)+5-(23.4+3)^3;8:3;

Wynik działania programu:



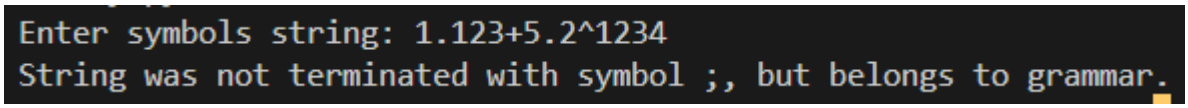
Słowo wejściowe: 1+1 2+2;

Wynik działania programu:



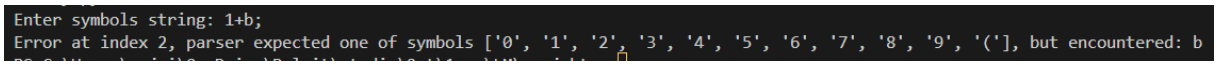
Słowo wejściowe: 1.123+5.2^1234

Wynik działania programu:



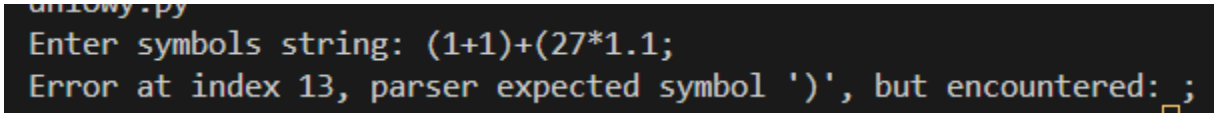
Słowo wejściowe: 1+b;

Wynik działania programu:



Słowo wejściowe: (1+1)+(27*1.1;

Wynik działania programu:



Słowo wejściowe: 32.1

Wynik działania programu: