Lab4 Documentatie

Structura fisierului din care se citeste gramatica

```
{
  "non_terminale": ["S","A","B","C"],
  "terminale": ["a","b","c"],
  "productii": {
    "S": ["aB","bA"],
    "A": ["cB"],
    "B": ["aS","c"],
    "C": ["b"]
  },
  "simbol_start": "S"
}
```

Arhitectura gramaticii

```
class Gramatica:

    def __init__(self, non_terminale:list, terminale:list,
productii:dict, simbol_start:str):
        self.__non_terminale = non_terminale
        self.__terminale = terminale
        self.__productii = productii
        self.__simbol_start = simbol_start

    def get_non_terminale(self):
        return self.__non_terminale

    def get_terminale(self):
        return self.__terminale

    def get_productii(self):
        return self.__productii

    def get_simbol_start(self):
        return self.__simbol_start
```

Afisarea elementelor gramaticii

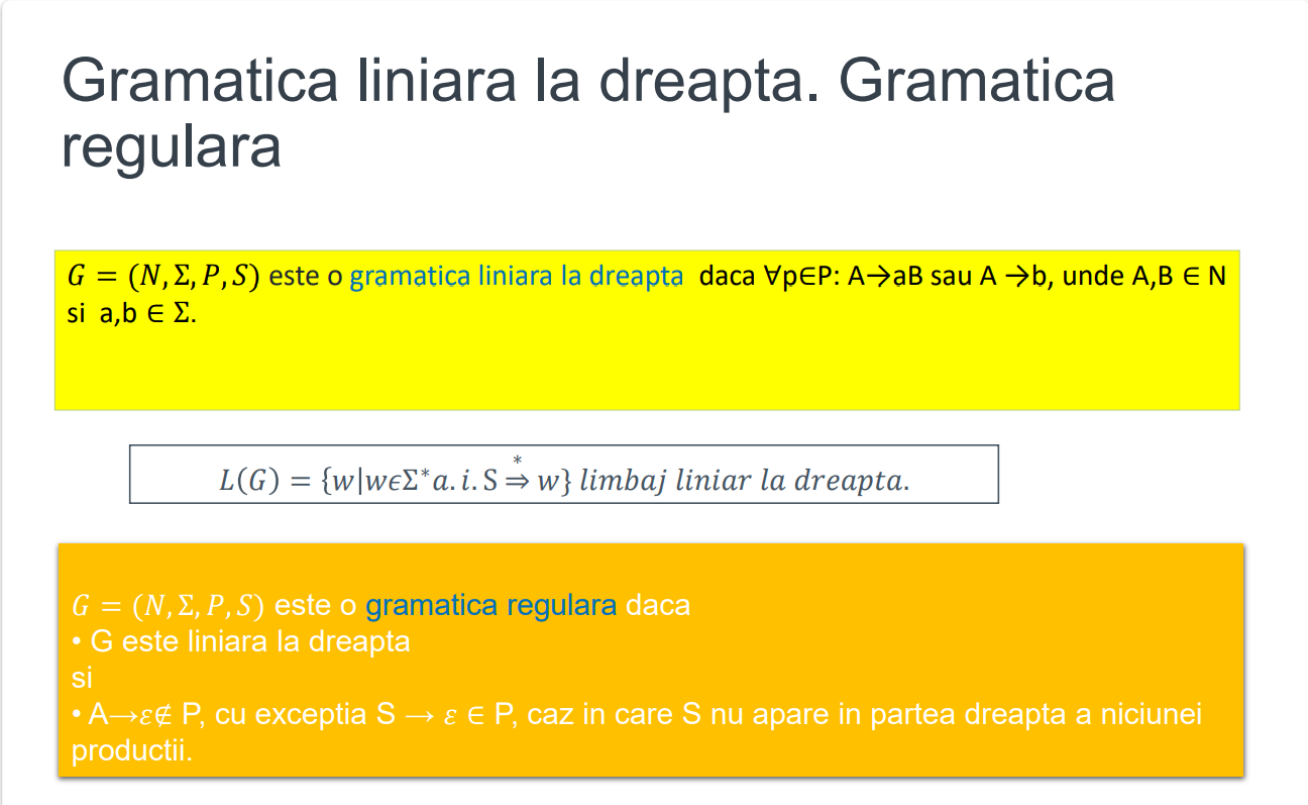
```
def afiseaza_gramatica(gramatica: Gramatica):
    s = ", "
    non_terminale = s.join(gramatica.get_non_terminale())
    terminale = s.join(gramatica.get_terminale())
    print("Non-terminale: {}".format(non_terminale))
    print("Terminale: {}".format(terminale))
    print("Productii: ")
    productii = gramatica.get_productii()
    for (key, values) in productii.items():
        print("{} -> {}".format(key, '|'.join(values)))
    print("Simbol de start: {}".format(gramatica.get_simbol_start()))
```

Afisarea productiilor pentru un non-terminal dat

```
def afiseaza_productii_non_terminal(gramatica:Gramatica, non_terminal):
    if non_terminal in gramatica.get_non_terminale():
        productii = gramatica.get_productii().get(non_terminal)
        print("Productiile pentru {} sunt: ".format(non_terminal))
        print("{} -> {}".format(non_terminal, "|".join(productii)))
    else:
        print("Non-terminalul dat nu se afla in gramatica")
```

Verificarea daca o gramatica este regulara

O gramatica regulara trebuie sa verifice conditiile din poza de mai jos



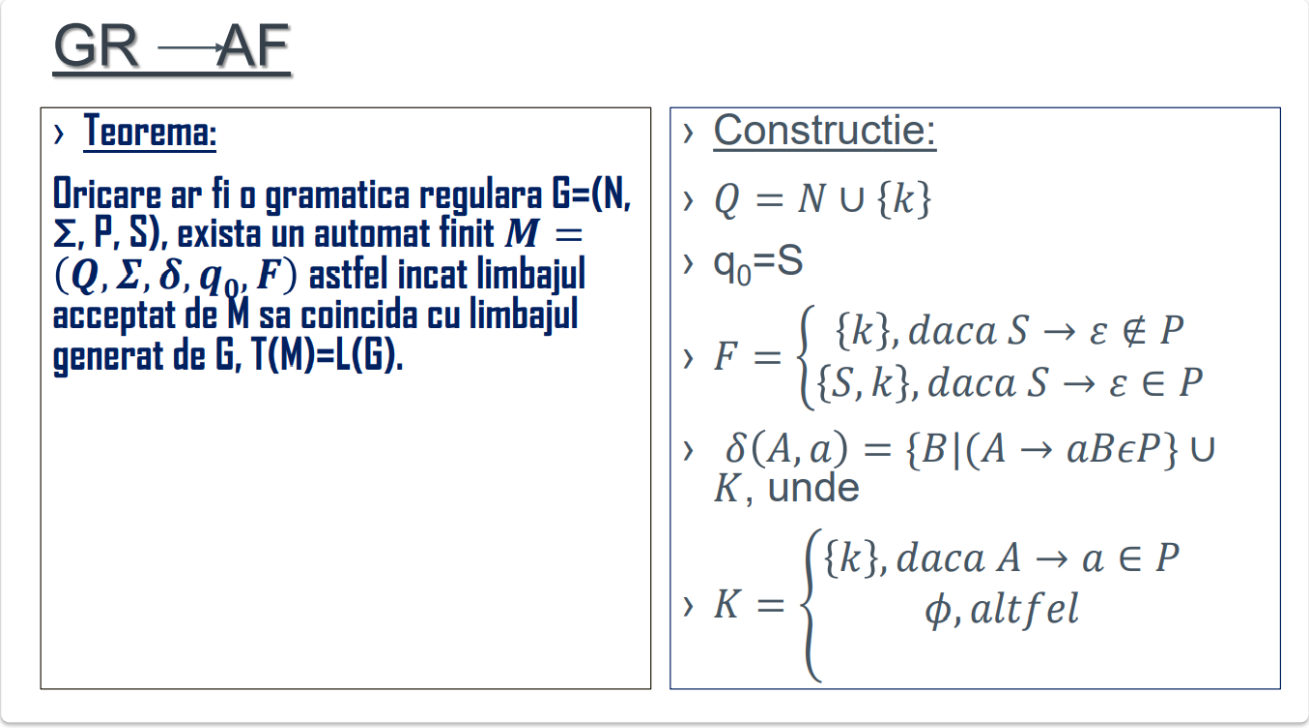
Implementare

```
def verifica_daca_este_regulara(self):
    """
    Verifica daca gramatica este regulara. O gramatica este regulara
daca este liniara la dreapta si singurul non-terminal care poate merge
in epsilon este simbolul de start. In acest caz simbolul de start nu
mai apare in partea dreapta a niciunei productii
    :return: True daca gramatica este regulara, False altfel
    """
    exista_epsilon_start = False
    exista_simbol_start_in_dreapta = False
    este_liniara_la_dreapta = True

    for (key, values) in self.__productii.items():
        #verificare liniaritate la dreapta
        for elem in values:
            if len(elem) > 2:
                este_liniara_la_dreapta = False
            if len(elem) == 1:
                if elem not in self.__terminale:
                    este_liniara_la_dreapta = False
            if len(elem) == 2:
                if elem[0] not in self.__terminale and elem[1] not in
self.__non_terminale:
                    este_liniara_la_dreapta = False
        #------------

            #verificare productii cu epsilon
            if self.__simbol_start in elem:
                exista_simbol_start_in_dreapta = True

        if EPSILON in values:
            if key != self.__simbol_start:
                return False
            else:
                exista_epsilon_start = True
            #-------------

        if exista_epsilon_start and exista_simbol_start_in_dreapta:
            return False
        if este_liniara_la_dreapta is False:
            return False
```

```
        return True
```

Conversie gramatica regulara -> automat finit

Teorie curs:



Constructie:

- Multimea de stari: non terminale + stare k(finala) ex: (S,A,K)
- Stare initiala: S
  Cand avem productii cu un singur terminal mergem in K (S->b devine din S in K prin b)
- Alfabetul = multimea de terminale
- Tranzitiile se obtin din productii
- Multimea starilor finale = {K U {S}} daca S merge in epsilon

Exemplu:

Exemplu

› G=({S,A,B, C}, {a,b,c},P, S)

› P:

$S \rightarrow aB|bA$

$A \rightarrow cB$

$C \rightarrow aS|c$

$B \rightarrow b$

› Automatul finit echivalent:

› M=({S,A,B,C, K}, {a,b,c},δ,S,{K})

› δ(S,a)={B}

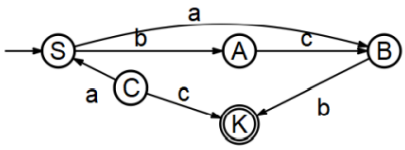
› δ(S,b)={A}

› δ (A,c)={B}

› δ (C,a)={S}

› δ (C,c)={K}

› δ (B,b)={K}



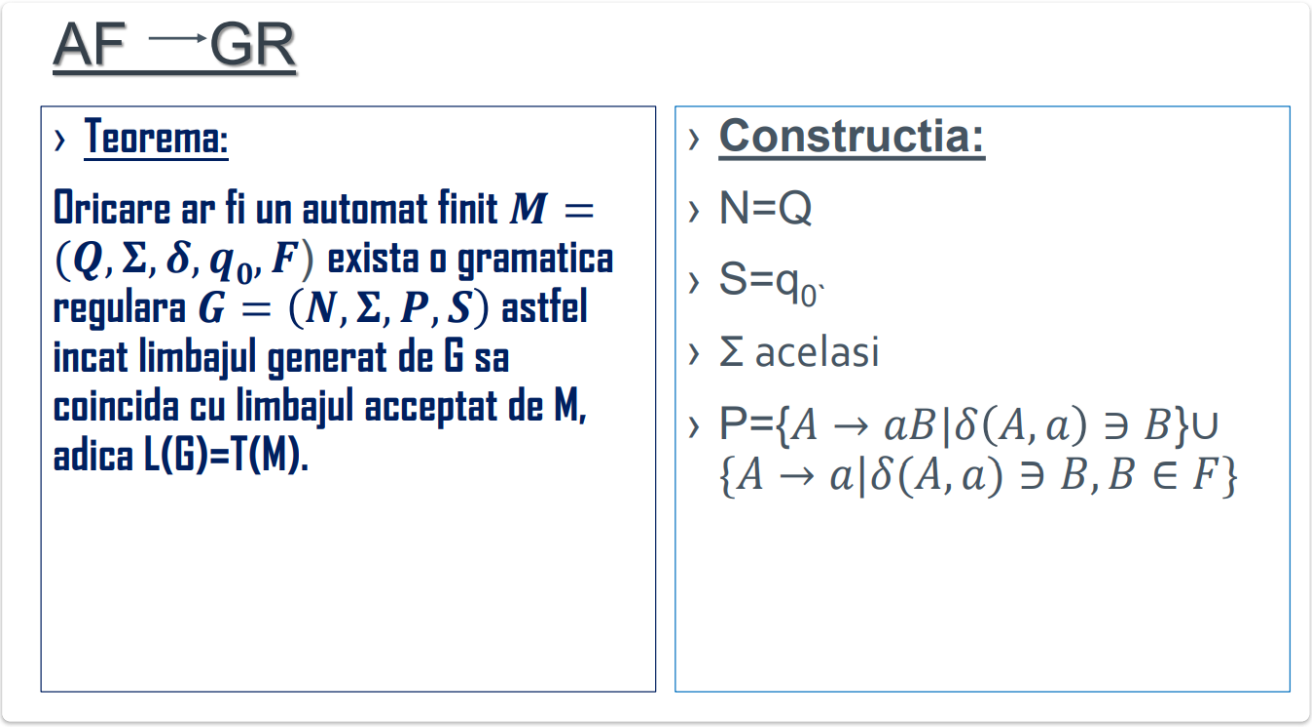
Implementare:

```
def convert_GR_AF(gramatica: Gramatica):
    """
    Transforma o gramatica regulara in automat finit
    :param gramatica: gramatica regulara
    :return: automatul finit corespunzator
    """
    if gramatica.verifica_daca_este_regulara() is False:
        return None

    multime_stari = gramatica.get_non_terminale()
    multime_stari.append("k")
    stare_initiala = gramatica.get_simbol_start()
    alfabet = gramatica.get_terminale()

    stari_finale = ["k"]
    productii_gramatica = gramatica.get_productii()
    if EPSILON in productii_gramatica.get(stare_initiala):
        stari_finale.append(stare_initiala)

    tranzitii = []
    for (key, values) in productii_gramatica.items():
        for val in values:
            if len(val) == 1:
                tranzitie = [key, [val], "k"]
```

```
            else:
                tranzitie = [key, [val[0]], val[1]]
            tranzitii.append(tranzitie)

    TAD = TADAutomat(stare_initiala, multime_stari, alfabet, tranzitii,
stari_finale)
    return TAD
```

Conversie automat finit -> gramatica regulara

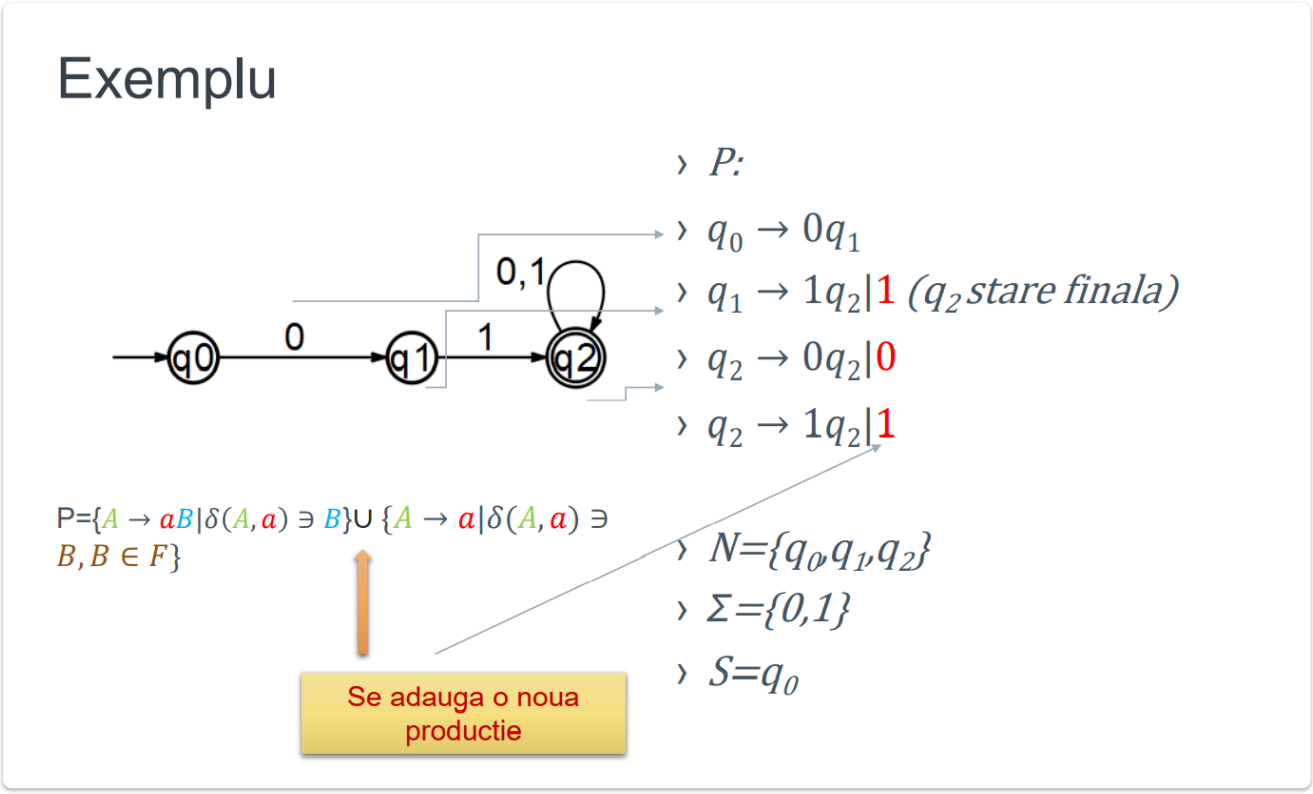
Teorie curs:



Constructie:

- S = starea initiala de la automat
- Non terminale = starile automatului
- Terminalele = alfabetul
- Productiile = din tranzitii (o sa il avem si pe K)

Exemplu:



Implementare:

```
def convert_AF_GR(TAD: TADAutomat):
    """
    Transforma un automat finit in gramatica regulara
    :param TAD: automatul finit
    :return: gramatica regulara
    """
    simbol_start = TAD.get_stare_initiala().upper()
    non_terminale = [x.upper() for x in TAD.get_stari()]
    terminale = TAD.get_alfabet()
    tranzitii = TAD.get_tranzitii()
    stari_finale = TAD.get_stari_finale()
    productii = {}
    for nt in non_terminale:
        productii[nt] = []
    for tranzitie in tranzitii:
        stare_stanga = tranzitie[0].upper()
        stare_dreapta = tranzitie[2].upper()
        terminale_stare = tranzitie[1]
        for ter in terminale_stare:
            productii[stare_stanga].append(ter+stare_dreapta)
            if stare_dreapta.lower() in stari_finale:
                productii[stare_stanga].append(ter)
```

```
    stari_finale_fara_tranzitii = []
    for (key, value) in productii.items():
        if len(value) == 0:
            stari_finale_fara_tranzitii.append(key)
    for key in stari_finale_fara_tranzitii:
        del productii[key]

    gramatica = Gramatica(non_terminale, terminale, productii,
simbol_start)
    return gramatica
```