Lab3 LFTC Documentatie

Structura fisierului din care se citeste automatul

Automatul se citeste dintr-un fisier json cu urmatoarea structura:

```
{
  "stare_initiala": ,
  "stari": [],
  "alfabet": [],
  "tranzitii": [
    [],
    [],
  ],
  "stari_finale": []
}
```

unde stari, alfabet, stari finale = lista de stringuri

- tranzitii = lista de liste
- o tranzitie = ['stare', [lista legaturi], 'stare']
  Fisierul pentru identificatori

Fisierul pentru constante numerice

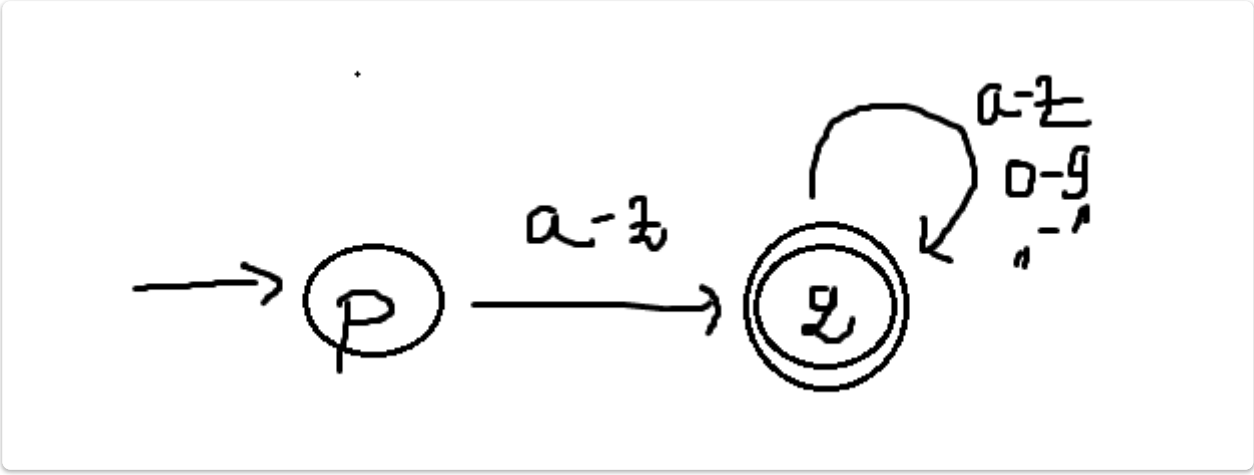
```
{
  "stare_initiala": "p",
  "stari": ["p", "q", "r", "t"],
  "alfabet": ["0","1","2","3","4","5","6", "7","8","9", "+", "-"],
  "tranzitii": [
    ["p",["+","-"],"q"],
    ["p", ["1","2","3","4","5","6", "7","8","9"],"r"],
    ["p",["0"],"t"],
    ["q",["1","2","3","4","5","6", "7","8","9"],"r"],
    ["r",["0","1","2","3","4","5","6", "7","8","9"],"r"]
  ],
  "stari_finale": ["r","t"]
}
```

Schema automatului

Automatul pentru identificatori

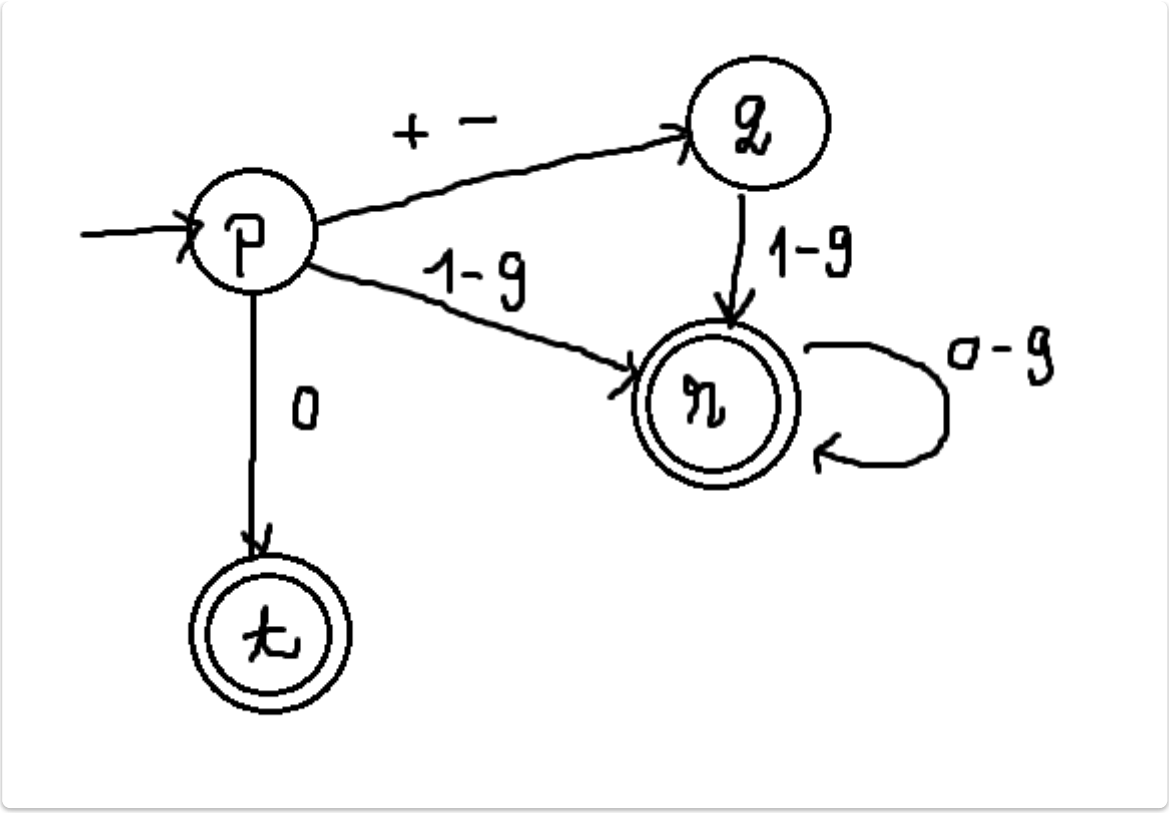
Un identificator = secventa de cel putin o litera, incepe cu litera urmata de litere, cifre si caracterul _

identificator::=litera{(litera|cifra|_)}



Automatul pentru constante numerice

Constanta numerica= numar intreg



Arhitectura automatului

```
class TADAutomat:

    def __init__(self, stare_initiala, stari: list, alfabet: list,
tranzitii: list, stari_finale: list):
        self.__stare_initiala = stare_initiala
        self.__stari = stari
        self.__alfabet = alfabet
        self.__tranzitii = tranzitii
        self.__stari_finale = stari_finale

    def get_stari(self):
        return self.__stari

    def get_stare_initiala(self):
        return self.__stare_initiala

    def get_alfabet(self):
        return self.__alfabet

    def get_tranzitii(self):
        return self.__tranzitii
```

```
    def get_stari_finale(self):
        return self.__stari_finale
```

Verificarea unei secvente

Se parcurge secventa caracter cu caracter, iar pentru fiecare caracter se verifica daca exista o tranzitie in lista de tranzitii. Se porneste din starea initiala si se apeleaza get_stare_pentru_simbol cu starea initiala si primul element din secventa. Apoi se actualizeaza starea initiala la noua stare, iar simbolul la urmatorul element din secventa

Daca get_stare_pentru_simbol nu gaseste o stare sau daca starea in care se ajunge cu ultimul element din secventa nu este finala se returneaza False, secventa nu este acceptata. Automatul nu accepta nici secventa vida.

```
def verifica_secventa(self, secventa):
    """
    Verifica daca o secventa data este acceptata de catre automat
    :param secventa: secventa primita
    :return: True daca este acceptata, False altfel
    """
    if len(secventa) == 0:
        return False
    first = secventa[0]
        stare_intermediara=self.get_stare_pentru_simbol(self.get_stare_
initiala(), first)
    if stare_intermediara is None:
        return False
    for elem in secventa[1:]:
        stare_intermediara =
self.get_stare_pentru_simbol(stare_intermediara, elem)
        if stare_intermediara is None:
            return False
    if stare_intermediara in self.get_stari_finale():
        return True
    return False
```

```
def get_stare_pentru_simbol(self, stare, simbol):
    """
    Pentru o anumita stare si un simbol de pe banda de intrare
returneaza starea in care se ajunge
    :param stare: starea care se da
    :param simbol:simbolul de pe banda
    :return:starea in care se ajunge sau None daca nu se ajunge
```

```
nicaieri
    """     for tranzitie in self.__tranzitii:
        if tranzitie[0] == stare and simbol in tranzitie[1]:
            return tranzitie[2]
    return None
```