Table of Contents

[**Introducere** 2](#_Toc184740825)

[Ce este un AFD? 2](#_Toc184740826)

[Ce presupune minimizarea unui AFD? 2](#_Toc184740827)

[**Implementare** 3](#_Toc184740828)

[Algoritmul de minimizare și implementarea în cod 3](#_Toc184740829)

[Aplicația și Exemple 6](#_Toc184740830)

[Instalare și Rulare 10](#_Toc184740831)

# **Introducere**

În acest referat încercăm să abordăm noțiunile generale cum ar fi: ce este un AFD? Ce este minimizarea acestuia? Ce algoritm se folosește pentru a minimiza? Și cum se implementează în Python?

## Ce este un AFD?

Un **Automat Finit Determinist (AFD)** este un model matematic utilizat pentru recunoașterea limbajelor regulate. AFD-urile sunt esențiale în analiza lexicală, recunoașterea șirurilor și alte domenii legate de procesarea limbajului formal. Ele sunt un caz particular de automate finite care respectă reguli stricte pentru determinism.

Un AFD este definit ca un cvintet:

AFD=⟨S,A,δ,s0,F⟩

unde:

* **S**: Mulțimea de stări finite.
* **A**: Alfabetul de intrare (setul simbolurilor acceptate).
* **δ**: Funcția de tranziție, δ:S×A→S, care asociază o stare unică pentru fiecare pereche (s,a), unde s∈S și a∈A.
* **s0​**: Starea inițială, s0∈S.
* **F**: Mulțimea stărilor acceptoare, F⊆S

## Ce presupune minimizarea unui AFD?

Minimizarea unui AFD constă în reducerea numărului de stări, păstrând limbajul recunoscut neschimbat. Aceasta se realizează prin combinarea stărilor echivalente (care au același comportament) într-o singură stare.  
 Necesitatea minimizării:

* AFD-urile obținute din conversia automatelor finite nedeterministe (AFN) sau din expresii regulate pot avea multe stări redundante.
* Un AFD minimizat ocupă mai puțin spațiu în memorie și permite procesări mai rapide.

# **Implementare**

## Algoritmul de minimizare și implementarea în cod

Algoritmul de minimizare a unui automat finit determinist (AFD) reduce numărul de stări fără a schimba limbajul recunoscut. Procesul implică partiționarea stărilor automatului și se desfășoară astfel:

1. **Partiția inițială**:
   * Se creează două grupuri:
     + **F**: stările acceptoare.
     + **S - F**: stările neacceptoare.

Pentru a implementa acest pas în cod, a fost creată o variabilă care stochează grupurile date. Fig. 2.1.

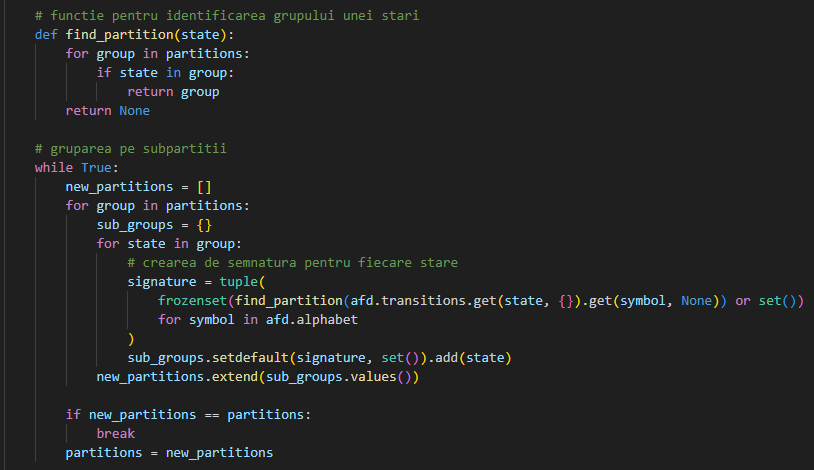
Fig. 2.1 Partiționarea

***afd.accept\_states*** reprezintă stările acceptoare.

***set(afd.states) - set(afd.accept\_states)*** reprezintă stările care nu sunt acceptoare.

1. **Rafinarea partiției**:
   * Se analizează fiecare grup și se separă stările în subgrupuri în funcție de tranzițiile lor pentru simbolurile din alfabet.
   * Stările sunt separate dacă ajung în grupuri diferite pentru același simbol de intrare.

În această etapă, partițiile existente sunt rafinate iterativ în funcție de tranzițiile pentru fiecare simbol din alfabet. Fig. 2.2

 Fig. 2.2 Rafinarea partiției

Fiecare stare dintr-un grup este analizată pentru a crea o semnătură pe baza tranzițiilor sale către alte grupuri (***signature***).

Stările sunt grupate în subgrupuri (***sub\_groups***) în funcție de semnături.

1. **Iterare**:
   * Rafinarea continuă până când partiția nu mai poate fi divizată.

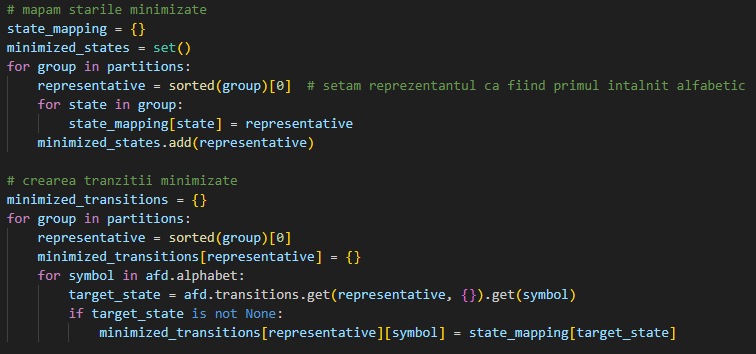
Procesul de rafinare se repetă până când partițiile nu se mai schimbă. Această parte este realizată de bucla ***while True***. Codul pentru verificarea opririi este prezentat în Fig. 2.3

Fig. 2.3 Ieșire din buclă

Aici algoritmul compară partiția nou creată (***new\_partitions***) cu cea precedentă (***partitions***). Dacă sunt identice, procesul se oprește.

1. **Construirea AFD minimizat**:
   * Fiecare grup din partiția finală devine o stare în noul AFD.
   * Tranzițiile sunt definite pe baza reprezentantului fiecărui grup.

Fiecare grup din partiția finală este transformat într-o stare a AFD-ului minimizat, iar tranzițiile sunt construite. Fig. 2.4.

 Fig. 2.4 Construirea AFD-ului Minimizat

***state\_mapping*** mapează fiecare stare originală la reprezentantul grupului său.

***minimized\_transitions*** conține tranzițiile noului AFD, construite pe baza reprezentanților.

1. **Curățare**:
   * Se elimină stările care nu pot fi atinse din starea inițială.
   * Se elimină starea moartă (dacă există).

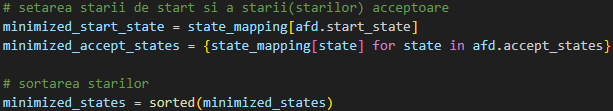
În final, se stabilesc starea inițială și stările acceptoare ale noului AFD. Fig. 2.5

Fig. 2.5 Curățare

***minimized\_start\_state*** este starea inițială din noul AFD, determinată prin maparea stării inițiale a AFD-ului original.

***minimized\_accept\_states*** sunt stările acceptoare, obținute din maparea stărilor acceptoare originale.

## Aplicația și Exemple

Aplicația reprezintă o modalitate de a minimiza numărul stărilor ale unui AFD. Aceasta este scrisă în **Python**, având librării de creare de GUI. În cazul nostru, am utilizat librăria **tkinter**.

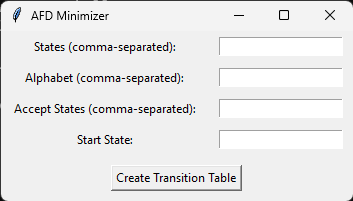
 La început, aplicația cere datele AFD-ului (**Stările – „States”, Alfabetul – „Alphabet”, Stările(starea) acceptoare – „Accept States”, Starea de start – „Start State”**) necesare pentru a crea tabelul de tranziție. (Fig. 2.6).

Fig. 2.6 Date de Intrare

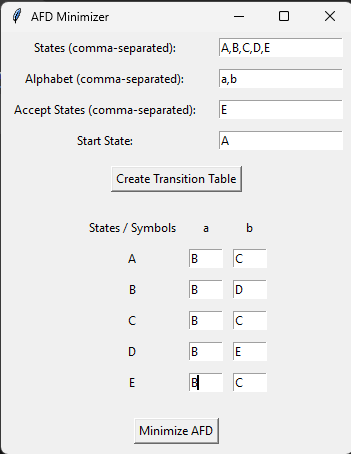
 După ce datele au fost introduse iar butonul „**Create Transition Table**” activat, se crează tabelul de tranziție care trebuie completat cu datele din AFD-ul dorit. Un exemplu este demonstrat mai jos în Fig. 2.7.

Fig. 2.7 Exemplu de AFD

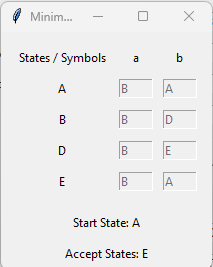
 La acționarea butonului „**Minimize AFD**” se afișează o pagină nouă unde este prezentat AFD-ul minimizat. Rezultatul din Fig. 2.7 poate fi observat și mai jos în Fig. 2.8.

Fig. 2.8. AFD Minimizat

Minimizarea în detaliu:

Stările: {A, B, C, D, E}

Alfabetul: {a, b}

Starea de start: A

Stări acceptoare: {E}

Tranzițiile:

Stare a b

A B C

B B D

C B C

D B E

E B C

Pasul 1: Partiția inițială

Se împarte mulțimea de stări în două grupuri:

Stări acceptoare: {E}

Stări neacceptoare: {A, B, C, D}

Π={{A,B,C,D},{E}}

Pasul 2: Refinarea partiției

Verificăm pentru fiecare stare dacă tranzițiile pe simbolurile 𝑎 și 𝑏 conduc la stări care aparțin unor grupuri diferite din Π. Dacă există diferențe, împărțim grupurile corespunzătoare.

Grup 1: {A,B,C,D}

Tranziții pentru 𝑎 și 𝑏:

Stare 𝑎 𝑏

A B C

B B D

C B C

D B E

Observăm:

𝐴 și 𝐶 rămân împreună deoarece ambele au 𝑎→𝐵 și 𝑏→𝐶, care sunt în același grup.

𝐵 și 𝐷 se separă: 𝐵 are 𝑏→𝐷 (din alt grup) iar 𝐷 are 𝑏→𝐸 (din grupul acceptor).

Noua partiție:

Π nou = {{A,C},{B},{D},{E}}

Pasul 3: Verificare stabilitate

Comparăm

Π cu Π𝑛𝑜𝑢 :

Π≠Π𝑛𝑜𝑢

Actualizăm Π:=Π𝑛𝑜𝑢​ și repetăm pasul 2.

Verificare:

Fiecare grup din Π𝑛𝑜𝑢 este acum stabil, deoarece tranzițiile fie duc în același grup, fie în grupuri diferite. Astfel, Π𝑓𝑖𝑛𝑎𝑙=Π𝑛𝑜𝑢.

Pasul 4: Construirea AFD minimizat 𝐷𝑚𝑖𝑛

Stări: Fiecare grup din Π𝑓𝑖𝑛𝑎𝑙 devine o stare în 𝐷𝑚𝑖𝑛 :

Stari: {[A,C],[B],[D],[E]}

Starea de start: Grupul care conține 𝐴:[𝐴,𝐶].

Stări acceptoare: Grupurile care conțin 𝐸:[𝐸].

Tranzițiile: Se calculează pe baza reprezentanților fiecărui grup:

Alte exemple:

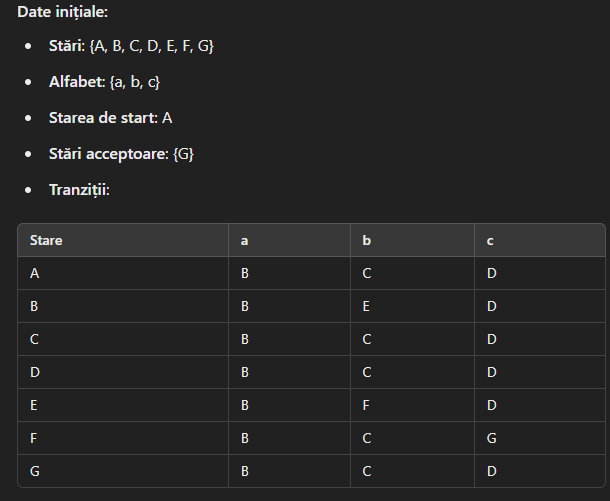
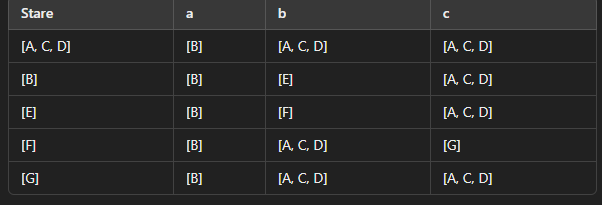
AFD Inițial AFD Minimizat

Fig. 2.10 AFD Minimizat

Fig. 2.9 AFD

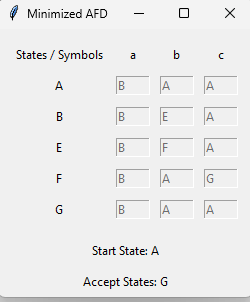
 Aplicația are același Output ca și cel așteptat. Fig. 2.11.

Fig. 2.11 Rezultat

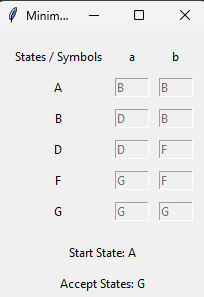
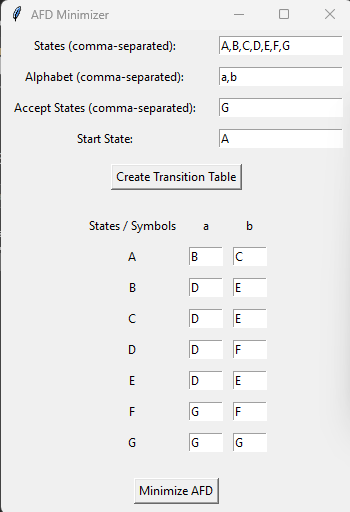


Fig. 2.12 AFD Exemplu 3 Fig. 2.13 AFD Minimizat

## Instalare și Rulare

Pentru a rula aplicația avem nevoie de Python v.3.x, care poate fi descărcat de [aici](https://www.python.org/downloads/).

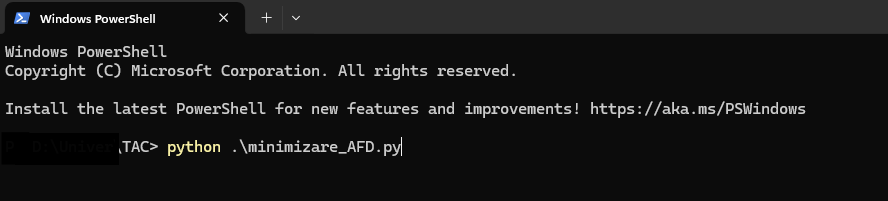
După ce Python a fost instalat cu succes, vom deschide un terminal și vom rula programul cu comanda „***python minimizare\_AFD.py***”. Fig. 2.14

Fig. 2.14 Rulare comandă în terminal