Table of Contents

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:

$AFD=\langle S,A,\delta,s0,F\rangle$

unde:

- **S**: Mulțimea de stări finite.
- **A**: Alfabetul de intrare (setul simbolurilor acceptate).
- **δ**: Funcția de tranziție, $\delta:S\times A\rightarrow S$, care asociază o stare unică pentru fiecare pereche (s,a), unde $s\in S$ și $a\in A$.
- **s0**: Starea inițială, $s0\in S$.
- **F**: Mulțimea stărilor acceptoare, $F\subseteq S$

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.

```
# partitionare initiala
partitions = [set(afd.accept_states), set(afd.states) - set(afd.accept_states)]
```

Fig. 2.1 Partiționarea

afd.accept_states reprezintă stările acceptoare.

set(afd.states) - set(afd.accept_states) reprezintă stările care nu sunt acceptoare.

2. **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

```python
# functie pentru identificarea grupului unei stari
def find_partition(state):
    for group in partitions:
        if state in group:
            return group
    return None

# gruparea pe subpartitii
while True:
    new_partitions = []
    for group in partitions:
        sub_groups = {}
        for state in group:
            # crearea de semnatura pentru fiecare stare
            signature = tuple(
                frozenset(find_partition(afd.transitions.get(state, {}).get(symbol, None)) or set())
                for symbol in afd.alphabet
            )
            sub_groups.setdefault(signature, set()).add(state)
        new_partitions.extend(sub_groups.values())

    if new_partitions == partitions:
        break
    partitions = new_partitions
```

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.

3. **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

```python
if new_partitions == partitions:
    break
```

Fig. 2.3 Ieșire din buclă

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

4. **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.

```python
# mapam starile minimizate
state_mapping = {}
minimized_states = set()
for group in partitions:
    representative = sorted(group)[0]  # setam reprezentantul ca fiind primul intalnit alfabetic
    for state in group:
        state_mapping[state] = representative
    minimized_states.add(representative)

# crearea tranzitii minimizate
minimized_transitions = {}
for group in partitions:
    representative = sorted(group)[0]
    minimized_transitions[representative] = {}
    for symbol in afd.alphabet:
        target_state = afd.transitions.get(representative, {}).get(symbol)
        if target_state is not None:
            minimized_transitions[representative][symbol] = state_mapping[target_state]
```

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.

5. **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

```python
# setarea starii de start si a starii(starilor) acceptoare
minimized_start_state = state_mapping[afd.start_state]
minimized_accept_states = {state_mapping[state] for state in afd.accept_states}

# sortarea starilor
minimized_states = sorted(minimized_states)
```

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).

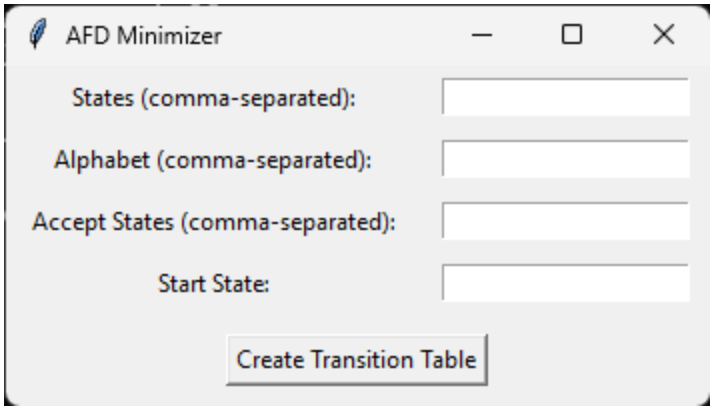

Fig. 2.6 Date de Intrare

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.

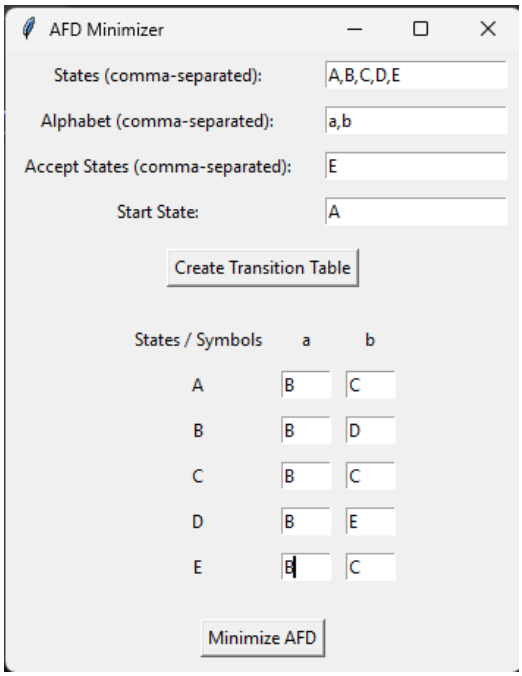

Fig. 2.7 Exemplu de AFD

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.

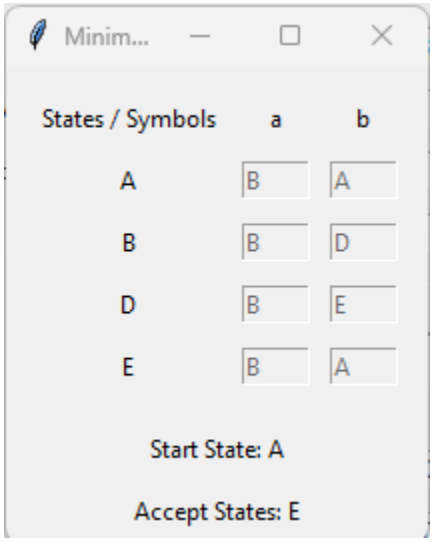

Fig. 2.8. AFD Minimizat

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 $a \rightarrow B$ și $b \rightarrow C$, care sunt în același grup.

B și D se separă: B are $b \rightarrow D$ (din alt grup) iar D are $b \rightarrow E$ (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, Π$final$=Πnou.

Pasul 4: Construirea AFD minimizat $Dmin$

Stări: Fiecare grup din Π$final$ 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

Date inițiale:

- **Stări:** {A, B, C, D, E, F, G}
- **Alfabet:** {a, b, c}
- **Starea de start:** A
- **Stări acceptoare:** {G}
- **Tranziții:**

Stare	a	b	c
A	B	C	D
B	B	E	D
C	B	C	D
D	B	C	D
E	B	F	D
F	B	C	G
G	B	C	D

Fig. 2.9 AFD

AFD Minimizat

Stare	a	b	c
[A, C, D]	[B]	[A, C, D]	[A, C, D]
[B]	[B]	[E]	[A, C, D]
[E]	[B]	[F]	[A, C, D]
[F]	[B]	[A, C, D]	[G]
[G]	[B]	[A, C, D]	[A, C, D]

Fig. 2.10 AFD Minimizat

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

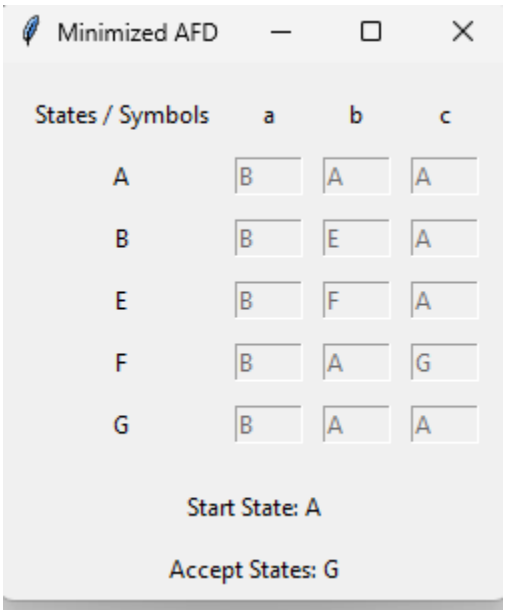

Fig. 2.11 Rezultat

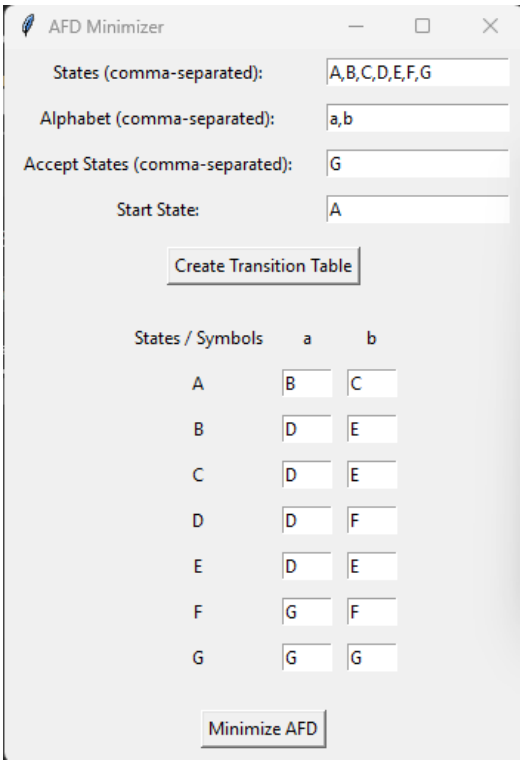

Fig. 2.12 AFD Exemplu 3

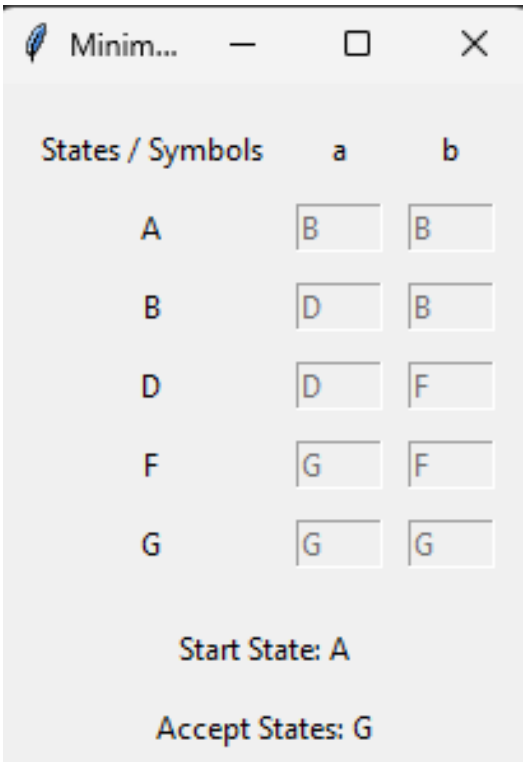

Fig. 2.13 AFD Minimizat

Instalare și Rulare

Pentru a rula aplicația avem nevoie de Python v.3.x, care poate fi descărcat de aici.

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