SAKARYA
ÜNİVERSİTESİ

Boşluk Geçişli NFA’dan DFA'ya Dönüşüm

1. Ödev

2024 - 2025

Younes Rahebi | B221210588 | 2.A

BİÇİMSEL DİLLER VE SOYUT MAKİNELER

Prof.Dr. NEJAT YUMUŞAK

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

1. Başla

Program başlıyor ve gerekli sabitler, değişkenler, ve NFA (Epsilon-NFA) geçiş tablosu alınıyor.

2. Geçiş Tablosunu Kontrol

Eğer NFA geçiş tablosu girilmişse, programa devam edilir.

Aksi durumda programdan çıkılır.

3. Epsilon Kapsama Hesaplama

Her NFA durumunu sırayla işle:

Epsilon geçişlerini takip ederek her durumun epsilon kapsamasını hesapla.

Hesaplanan kapsama durumlarını kaydet.

4. DFA Oluşturma

NFA'nın başlangıç durumunun epsilon kapsamasını alarak DFA'nın ilk durumunu başlat.

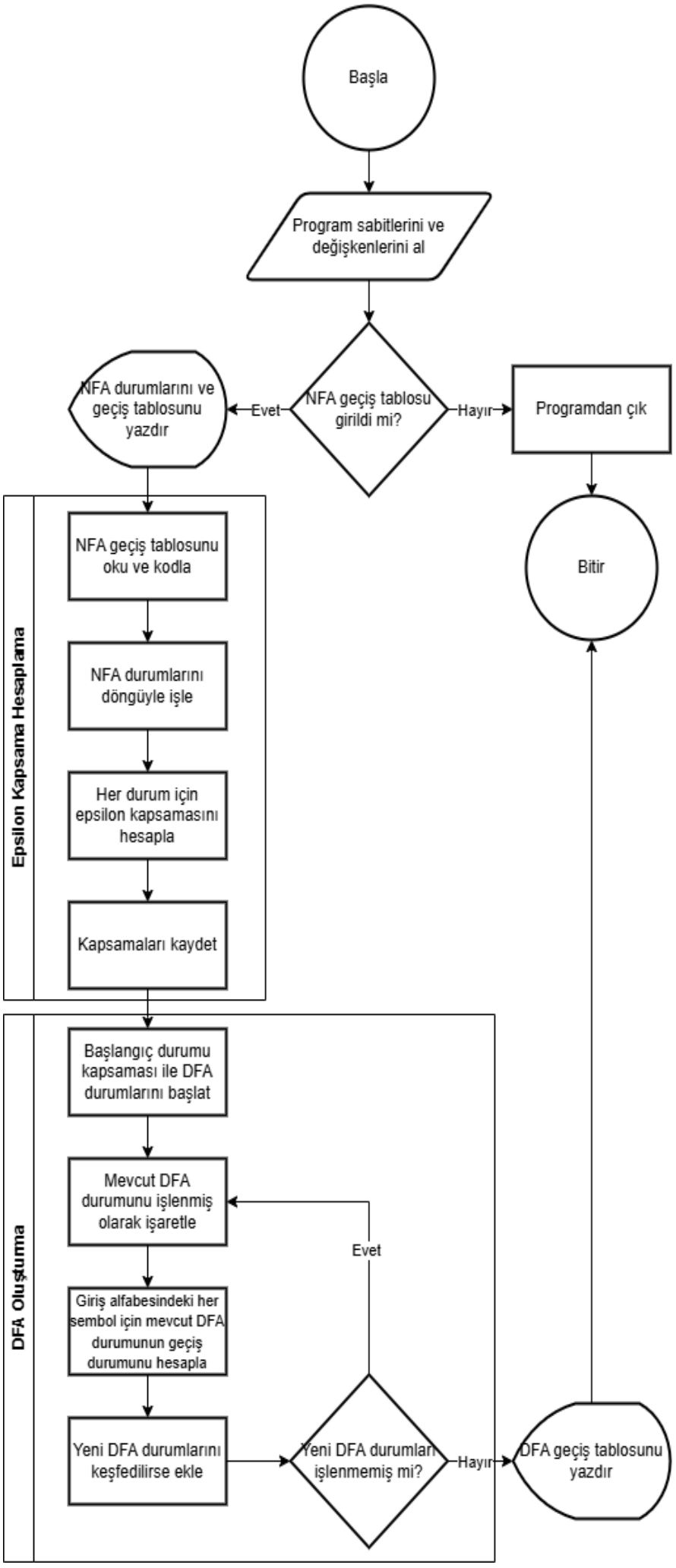
Giriş alfabesindeki her sembol için, mevcut durumun geçişlerini hesapla.

Yeni bir DFA durumu bulunursa, bunu ekle.

Eğer işlenmemiş DFA durumu kalmazsa, işlemi bitir.

5. Sonuçları Yazdır ve Bitir

Son olarak, oluşturulan DFA'nın geçiş tablosu ve durumlarını yazdır, ardından programdan çık.



C++ Dilinde Boşluk Geçişli NFA’dan DFA'ya Dönüşüm

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <iomanip>
#include <algorithm>
#include <cctype>

const int MAX_LEN = 100;

int last_index = 0;  //DFA tablosundaki son giriş indeksi
int symbols;         //Karakterlerin sayısı

//DFA yapısını tanımlıyoruz
struct DFA {
    std::string states;
    int count;
};

//Dizi sıfırlama fonksiyonu
void reset(std::vector<int>& ar) {
    std::fill(ar.begin(), ar.end(), 0);
}

//Belirli durumları kontrol et
void check(std::vector<int>& ar, const std::string& S) {
    for (char c : S) {
        int j = c - 'A';
        ar[j]++;
    }
}
```

```
//Yeni closure (kapanış) durumlarını oluştur
void state(const std::vector<int>& ar, std::string& S) {
    S.clear();
    for (int j = 0; j < ar.size(); j++) {
        if (ar[j] != 0)
            S += static_cast<char>('A' + j);
    }
}

//Closure işlemi için bir sonraki durumu bul
int closure(const std::vector<int>& ar) {
    auto it = std::find(ar.begin(), ar.end(), 1);
    return (it != ar.end()) ? std::distance(ar.begin(), it) : 100;
}

//Yeni DFA durumlarının tablosuna eklenip eklenemeyeceğini kontrol et
int new_states(std::vector<DFA>& dfa, const std::string& S) {
    auto it = std::find_if(dfa.begin(), dfa.begin() + last_index,
                           [&S](const DFA& d) { return d.states == S; });
    if (it != dfa.begin() + last_index)
        return 0;

    dfa[last_index++].states = S;
    dfa[last_index - 1].count = 0;
    return 1;
}

//NFA -> DFA geçiş işlemini uygula
void trans(const std::string& S, int M, const std::vector<std::string>& clsr_t, int st,
           const std::vector<std::vector<std::string>>& NFT, std::string& TB) {
    std::vector<int> arr(st, 0);
    std::string temp, temp2;
```

```
    for (char c : S) {
        int j = c - 'A';
        temp = NFT[j][M];

        if (temp != "-") {
            for (char g : temp) {
                int k = g - 'A';
                temp2 = clsr_t[k];
                check(arr, temp2);
            }
        }
    }

    state(arr, temp);
    TB = temp.empty() ? "-" : temp;
}

//Kullanıcı girdisinin doğru olup olmadığını kontrol eder
bool isValidInput(const std::string& input, int maxStates) {
    if (input == "-")
        return true;

    for (char c : input) {
        if (!std::isalpha(c) || c - 'A' >= maxStates) {
            return false;  //Sadece 'A'-'Z' ve geçerli durumlar izinli
        }
    }
    return true;
}

//Epsilon closure durumlarını göster
void Display_closure(int states, std::vector<int>& closure_ar,
                     std::vector<std::string>& closure_table,
                     const std::vector<std::vector<std::string>>& NFA_TABLE) {
```

```
    for (int i = 0; i < states; i++) {
        reset(closure_ar);
        closure_ar[i] = 2;

        if (NFA_TABLE[i][symbols] != "-") {
            std::string buffer = NFA_TABLE[i][symbols];
            check(closure_ar, buffer);
            int z = closure(closure_ar);

            while (z != 100) {
                if (NFA_TABLE[z][symbols] != "-") {
                    buffer = NFA_TABLE[z][symbols];
                    check(closure_ar, buffer);
                }
                closure_ar[z]++;
                z = closure(closure_ar);
            }
        }

        std::string buffer;
        state(closure_ar, buffer);
        closure_table[i] = buffer;
    }
}

//NFA durumlarını görüntüle
void Display_NFA(int states, const std::vector<std::vector<std::string>>& NFA_TABLE) {
    std::cout << "\nNFA DURUM TABLOSU\n\n";
    std::cout << std::setw(16) << "DURUMLAR";

    for (int i = 0; i < symbols; i++)
        std::cout << "|" << std::setw(16) << i;
    std::cout << "|" << std::setw(16) << "eps\n";
```

```
    std::cout << "-----------------";
    for (int i = 0; i <= symbols; i++)
        std::cout << "-----------------";
    std::cout << "\n";

    for (int i = 0; i < states; i++) {
        std::cout << std::setw(16) << static_cast<char>('A' + i);

        for (int j = 0; j <= symbols; j++) {
            std::cout << "|" << std::setw(16) << NFA_TABLE[i][j];
        }
        std::cout << "\n";
    }
}

//DFA durumlarını görüntüle
void Display_DFA(int last_index, const std::vector<DFA>& dfa_states,
                 const std::vector<std::vector<std::string>>& DFA_TABLE) {
    std::cout << "\n\n\n************************************************************\n\n\n";
    std::cout << "DFA DURUM TABLOSU\n\n";

    std::cout << "DFA DURUMLARI: ";
    for (int i = 1; i < last_index; i++)
        std::cout << dfa_states[i].states << ", ";
    std::cout << "\n";

    std::cout << "DFA ICIN VERILEN KARAKTERLER: ";
    for (int i = 0; i < symbols; i++)
        std::cout << i << ", ";
    std::cout << "\n\n";

    std::cout << std::setw(16) << "DURUMLAR";
    for (int i = 0; i < symbols; i++)
        std::cout << "|" << std::setw(16) << i;
```

```
    std::cout << "\n";

    std::cout << "-----------------";
    for (int i = 0; i < symbols; i++)
        std::cout << "-----------------";
    std::cout << "\n";

    for (int i = 0; i < last_index - 1; i++) {
        std::cout << std::setw(16) << dfa_states[i + 1].states;

        for (int j = 0; j < symbols; j++) {
            std::cout << "|" << std::setw(16) << DFA_TABLE[i][j];
        }
        std::cout << "\n";
    }
}

int main() {
    char choice;
    do {
        int states;
        std::cout << "NFA'daki durum sayisini girin: ";
        std::cin >> states;

        std::cout << "NFA'daki karakter sayisini girin (epsilon haricinde): ";
        std::cin >> symbols;

        std::vector<std::vector<std::string>> NFA_TABLE(states, std::vector<std::string>(symbols + 1));
        std::vector<std::vector<std::string>> DFA_TABLE(MAX_LEN, std::vector<std::string>(symbols));

        std::cout << "\nNFA gecis tablosunu girin (Durumlar A'dan baslayarak belirtilmeli, '-' gecis yok
anlaminda):\n";
        for (int i = 0; i < states; i++) {
            std::cout << static_cast<char>('A' + i) << " durumunun gecisleri:\n";
```

```
    for (int j = 0; j <= symbols; j++) {
        std::string input;
        if (j < symbols)
            std::cout << "Karakter " << j << ": ";
        else
            std::cout << "Epsilon: ";

        std::cin >> input;
        while (!isValidInput(input, states)) {
            std::cout << "Hatali giris! Lutfen gecerli bir durum ya da '-' girin: ";
            std::cin >> input;
        }
        NFA_TABLE[i][j] = input;
    }
}

Display_NFA(states, NFA_TABLE);

std::vector<int> closure_ar(states);
std::vector<std::string> closure_table(states);

Display_closure(states, closure_ar, closure_table, NFA_TABLE);

std::vector<DFA> dfa_states(MAX_LEN);
last_index = 0;
dfa_states[last_index++].states = "-";
dfa_states[last_index - 1].count = 1;

std::string buffer = closure_table[0];
dfa_states[last_index++].states = buffer;

int zz = 0;
int Sm = 1, ind = 1;
int start_index = 1;
```

```
        while (ind != -1) {
            dfa_states[start_index].count = 1;
            Sm = 0;
            for (int i = 0; i < symbols; i++) {
                std::string T_buf;
                trans(buffer, i, closure_table, states, NFA_TABLE, T_buf);

                DFA_TABLE[zz][i] = T_buf;

                Sm += new_states(dfa_states, T_buf);
            }
            ind = -1;
            for (int i = 1; i < last_index; i++) {
                if (dfa_states[i].count == 0) {
                    buffer = dfa_states[i].states;
                    start_index = i;
                    ind = 1;
                    break;
                }
            }
            zz++;
        }

        Display_DFA(last_index, dfa_states, DFA_TABLE);

        std::cout << "\nTekrar denemek ister misiniz? (Y/N): ";
        std::cin >> choice;
    } while (choice == 'Y' || choice == 'y');

    std::cout << "Program sonlandiriliyor. Tesekkurler!" << std::endl;
    return 0;
}
```