Sprawozdanie		Data wykonania: 9 listopada 2025
Tytuł Mini-Projektu	Wykonał	Sprawdził
Automat skończony	Monika Musielik	dr. inż. Konrad Markowski

Spis treści

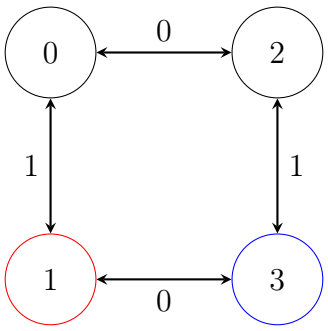
1 Cel projektu

Celem projektu było stworzenie symulacji automatu skończonego $M = (Q, \Sigma, \delta, q_0, F)$ o podanych symbolach wejściowych Σ, funkcji przejścia δ, stanem początkowym q_0 oraz końowym F.

W specyfikacji programu należało narysować diagram z pokazanymi przejściami z zaznaczonym aktualnym stanem automatu.

2 Rozwiązanie problemu

Pierwszym krokiem aby stworzyć symulacje jest wizualizacja automatu skończonego w formie grafu, który ma na celu pomóc mi zrozumieć jak działa dany automat. Stan początkowy oznaczam czerwonym okręgiem, a stan końcowy oznaczam niebieskim okręgiem.



Rysunek 1: Graf automatu skończonego

Wizualizacja maszyny (Rysunek 1) ma formę przypominającą kwadrat, gdzie każde przejście jest obustronne, dlatego pozwoliłam sobie na użycie strzałek obustronnych w celu uproszczenia rysunku. Inaczej mówiąc, jeżeli do tego automatu wpisze się 2 tą samą instrukcję, to stan automatu pozostanie taki sam.

3 Szczegóły implementacyjne

Program, zgodnie z poleceniem prowadzącego, należało zaimplementować w C/C++. Wybrałam do tego język C ze względu na niską złożoność problemu i brak wymogu używania programowania objektowego.

3.1 Algorytm automatu skończonego

Pierwszym krokiem w implementacji maszyny jest napisanie algorytmu funkcji przejścia δ, która przyjmuje 2 argumenty: symbol $a \in \Sigma$ oraz instrukcję $b \in Q$.

```
int process_instruction(int instruction, int pivot) {
    switch (pivot) {
                                    // if 1 / 0
        case 0: return instruction ? 1 : 2;
        case 1: return instruction ? 0 : 3;
        case 2: return instruction ? 3 : 0;
        case 3: return instruction ? 2 : 1;
        default: return -1;
    }
}
```

w tym przypadku `instruction` odpowiada instrukcji b, a `pivot` odpowiada symbolowi a. Funkcja `process_instruction` przechodzi do odpowiedniego symbolu $b \in Q$ używając instrukcji wielokrotnego wyboru i następnie zwraca odpowiednią wartość $a \in \Sigma$ zgodnie z operatorem warunkowym. W celu zabezpieczenia funkcji przed symbolami z poza zbioru Q, w przypdku domyślnym zwraca zawsze -1.

3.2 Metoda wprowadzania danych

Następnie należy wybrać metodę wprowadzania danych przez użytkownika i napisać funkcje która je przetwarza do formatu przyjaznego przetwarzaniu. Postanowiłam dać użytkownikowi wybór między dodawaniem ciągu znaków $c \in \{0; 1\}$ jako argument w konsoli, a wprowadzaniem ich na bierząco w przypadku braku podania ich jako argument.

```
void render_plane(int instruction, int pivot);

int main(int argc, char* argv[]) {

    int pivot = START;
    render_plane(-1, pivot);

    if (argc < 2) {
        char c;
        ...
```

```
    } else {
        char* input = argv[1];
        ...
```

Na początku z predefiniowanych zmiennych na etapie preprocesora ustawia się początkowy stan automatu, następnie sprawdza ilość podanych argumentów w konsoli i wybiera odpowiadającą formę wprowadzania danych. Jeżeli użytkownik nie podał instrukcji jako argument, program będzie o nie prosił w konsoli.

Powstała dodatkowo funkcja `render_plane` odpowiadająca za rysowanie diagramu automatu, którego kodu pozwalam sobie nie umieszczać w tym sprawozdaniu ze względu na jej kształt przypominający *spaghetti*.

3.2.1 Wprowadzanie danych w konsoli

Wprowadzanie danych za pośrednictwem konsoli jest zaimplementowane poprzez pętle `while` oraz funkcje `scanf`. Bez żadnych modyfikacji, pętla działałaby bez końca, dlatego został dodany warunek przerwania pętli poprzez wprowadzenie znaku `e`.

```
char c;
while( scanf("%c", &c) && c != 'e' ) {

    if (c != '0' && c != '1') {
        if (c != '\n')
            printf("Wykryto niewlasciwa instrukcje! (%c)\n", c);
        continue;
    }

    int instruction = c == '0' ? 0 : 1;
    pivot = process_instruction(instruction, pivot);

    render_plane(instruction, pivot);
    ...
}
```

Ten fragment kodu jest wykonywny w wypadku braku podania instrukcji jako argument. Program wchodzi w pętlę proszenia użytkownika o znak, którą można przerwać podając klawisz `e`. Można również użyć kombinacji `^C`, ale wtedy program nie pokaże wyników końcowych jak informacji o zaakceptowaniu ciągu.

Dalej program przypisuje wartość typu `int` dla danej instrukcji, przez wcześniejsze sprawdzanie zgodności formatu i specyfike zadania, która zakłada tylko 2 rodzaje instrukcji, można użyć prostej instrukcji warunkowej która zamieni `'0'` na 0, i `'1'` na 1 w bardzizej efektywny sposób niż np. funkcja `atoi`.

Na końcu program przechodzi do następnej instrukcji używając `process_instruction` i rysuje zaktualizowaną planszę (graf).

Dodatkowo można zauważyć ochronę programu przed *faux pas* użytkownika na znaki niezawierające się w zbiorze $c \in \{0, 1, e\}$. Jeśli użytkownik poda więcej niż jedną instruckje w linii, program wykona je po kolei.

3.2.2 Wprowadzanie danych jako argument

Wprowadzanie danych jako argument dostępny w `argv[1]` jest nieco bardziej skomplikowane, ale działa na podobnej zasadzie.Program zakłada tablicę znaków, preferowalnie należących do zbioru $c \in \{0, 1\}$, które będą służyć jako instrukcje.

Program został wzbogacony o funkcje `scanf("%c")`, która zwiększa doświadczenie użytkownika i zatrzymuje symulacje automatu co każdy krok dla wygodniejszej analizy przez użytkownika.

```
char* input = argv[1];
int n = strlen(input);

for (int i = 0; i < n ; i++) {
    scanf("%c");
    if (input[i] != '0' && input[i] != '1') {
        ...
        continue;
    }

    int instruction = input[i] == '0' ? 0 : 1;
    pivot = process_instruction(instruction, pivot);
    ...
}
```

Analogicznie do wycinka kodu w (3.2.1), program iteruje przez każdą instrukcje maszyny, ale tym razem w pętli `for`, iterując po indeksie tablicy znaków `input` przez interator. Następnie sprawdza poprawność znaku, czy należy do c. W przeciwnym wypadku zwraca stosowny komunikat i ignoruje tą iterację przez instrukcje `continue`.

Następnie identycznie do poprzedniego przypdaku program przypisuje liczbę całkowitą do `instruction`, przechodzi do kolejnej instrukcji i rysuje zaktualizowaną planszę (graf).

3.3 Pokazywanie wyników końcowych

Specyfikacja nakazuje podanie komunikatu o zaakceptowaniu ciągu, natomiast przez stukturę kodu w poprzednich etapach, wyświetlanie wyniku końcowego można zaimplementować prostą instrukcją warunkową.

```
if (pivot == END) {
    printf("Gratulujemy! Ciag zostal zaakceptowany.\n");
}

return 0;
```

Na poziomie preprocesora podobnie jak w przypadku zmiennej `START`, zmienna `END` jest zdefiniowana przez makro `#define`.

W obecnej budowie programu na koniec powinna zostać wyświetlona użytkownikowi plansza końcowa oraz komunikat o akceptacji ciągu jeśli zostały spełnione potrzebne warunki.

4 Sposób wywołania programu

Program jest napisany zgodnie od `ISO C99`, więc zwykła kompilacja używając dowolnego kompilatora C powinna wystarczyć. Program został napisany tak, aby nie potrzebował żadnych dodatkowych plików, a używa tylko bibliotek `string` oraz `stdio`. Można go skompilować w następujący sposób używając popularnych kompilatorów:

```
cc 343358.c
gcc 343358.c
g++ 343358.c
```

Następnie można go uruchomić na 2 sposoby, podając argumenty przy wywoływaniu programu lub podając je później w programie.

```
./a.out 0011001
./a.out
```

Program został uodporniony na działania użytkownika wbrew założeniom twórcy, jeśli użytkownik poda nieznany znak, to program go zignoruje, wyświetalając odpowiedni komunikat, jeśli jest stosowny.

5 Wnioski i spostrzeżenia

Pierwotnie program miał być okienkowy za pomocą biblioteki `raylib`, jednak ze względu na znaczące utrudnienie kompilowania programu na innych komputerach, w szczególności na systemie operacyjnym Windows z tego pomysłu szybko zrezygnowałam.

Sam sczkic programu, działający *aby działał*, powstał jeszcze podczas zajęć. Natomiast najwięcej czasu spędziłam na implementacji graficznej programu. Ze względu na jej monotoniczność i brak *nagrody* z napisania dedykowanego systemu rysowania grafów, napisałam brzydką funkcję która jest bardzo trudna do rozszerzenia, lecz spełnia wymogi tego konkretnego przypadku zadania.