Projet tp THL

Realiser par :

- Laredj Taha Yacine “Section 2 Groupe 7”
  Email : “laredjyacine955@gmail.com”

- Hamane Aziz “Section 2 Groupe 6”
  Email: “aziz.hamane@outlook.com”

2022/2023

Automate choisi :

A=(∑ , E,q0,F, Δ)

Nombre de symboles: 2

Nombre d'etats: 3

Nombre d'etats initial: 1

Nombre d'etats finaux: 1

Symboles: a b

Etats initial: q0

Etats: q0 q1 q2

Etats Finaux: q2

Transitions:

delta (q0 ,a) q1

delta (q0 ,b) q0

delta (q1 ,a) q1

delta (q1 ,b) q2

delta (q2 ,a) q2

delta (q2 ,b) q2

Représentation matricielles

	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_2	q_2

Compte rendu :

Le programme ci-dessus est un programme en langage C qui implémente un automate fini déterministe. Il est composé de plusieurs fonctions qui contribuent au fonctionnement global du programme.

La fonction stringEquals compare deux chaînes de caractères et retourne true si elles sont identiques. Cette fonction est utilisée pour la recherche d'états dans la fonction getState.

```c
bool stringEquals(char *str1, int size1, char *str2, int size2)
{
    return strcmp(str1, str2) == 0;
}
```

La fonction getState recherche et renvoie un pointeur vers un état spécifique dans l'automate en fonction de son nom. Elle parcourt la liste des états de l'automate et utilise la fonction stringEquals pour comparer les noms.

```c
Etat *getState(char name[256], Automaton *automaton)
{
    for (int i = 0; i < automaton->num_states; i++)
    {
        if (stringEquals(automaton->etats[i].nom, 256, name, 256))
        {
            return &automaton->etats[i];
        }
    }
    return NULL;
}
```

La fonction copyString copie une chaîne de caractères source dans une chaîne de caractères destination d'une taille donnée.

```c
void copyString(char *src, char *destination, int size)
{
    for (int i = 0; i < size; i++)
    {
        destination[i] = src[i];
    }
}
```

La fonction Create crée une nouvelle transition en allouant dynamiquement de la mémoire pour une structure Transition et en initialisant ses valeurs.

```c
Transition *Create(char symbol, char destination[256])
{

    Transition *transition = (Transition *)malloc(sizeof(Transition));


    if (transition == NULL)
    {
        printf("BIG ERROR\n");
        return NULL;
    }


    transition->symbol = symbol;
    copyString(destination, transition->destination, 256);
    transition->suiv = NULL;


    return transition;
}
```

La fonction addTransition ajoute une transition à la liste des transitions d'un état. Elle parcourt la liste des transitions jusqu'à trouver la dernière transition et y ajoute la nouvelle transition.

```c
void addTransition(Transition **head, Transition *toAdd)
{
    Transition *p = *head;
    if (p == NULL)
    {
        *head = toAdd;
        return;
    }

    while (p->suiv != NULL)
    {
        p = p->suiv;
    }
    p->suiv = toAdd;
}
```

La fonction isDeterministic vérifie si l'automate est déterministe. Elle parcourt tous les états de l'automate et vérifie s'il existe des symboles de transition en double. Si un symbole est déjà apparu pour un état donné, l'automate n'est pas déterministe.

```c
bool isDeterministic(Automaton *automaton)
{
    for (int i = 0; i < automaton->num_states; i++)
    {
        Etat *state = &automaton->etats[i];

        bool seenSymbols[MAX_SYMBOLS] = {false};

        Transition *transition = state->head;

        while (transition != NULL)
        {
            char symbol = transition->symbol;
            if (seenSymbols[symbol])
            {
                return false;
            }
            seenSymbols[symbol] = true;
            transition = transition->suiv;
        }
    }

    return true;
}
```

La fonction Affichage affiche les détails de l'automate, y compris le nombre de symboles, d'états, d'états finaux, les symboles, l'état initial, les états, les états finaux et les transitions. Elle utilise la fonction isDeterministic pour afficher si l'automate est déterministe ou non.

```
void Affichage(Automaton *automaton)
{
    printf("Nombre de symboles: %d\n", automaton->num_symbols);
    printf("Nombre d'etats: %d\n", automaton->num_states);
    printf("Nombre d'etats finaux: %d\n",   automaton->num_final_states);

    printf("Symboles:");
    for (int i = 0; i < automaton->num_symbols; i++)
    {
        printf(" %c", automaton->symbols[i]);
    }
    printf("\n");

    printf("Etats initial: %s\n", automaton->etatinitial);

    printf("Etats:");
    for (int i = 0; i < automaton->num_states; i++)
    {
        printf(" %s", automaton->etats[i].nom);
    }
    printf("\n");

    printf("Etats Finaux:");
     for (int i = 0; i < automaton->num_states; i++)
    {

        if (automaton->etats[i].isFinal == true)
        {
            printf(" %s", automaton->etats[i].nom);
        }
    }
    printf("\n");
```

```
    printf("Transitions:\n");
    for (int i = 0; i < automaton->num_states; i++)
    {
        Etat state = automaton->etats[i];
        Transition *transition = state.head;
        while (transition != NULL)
        {
            printf("delta (%s, %c) %s\n", state.nom, transition->symbol, transition->destination);
            transition = transition->suiv;
        }
    }
    bool isDet = isDeterministic(automaton);
if (isDet)
{
    printf("L'automate est deterministe.\n");
}
else
{
    printf("L'automate n'est pas deterministe.\n");
}

}
```

La fonction afficherletransition affiche une transition spécifique entre deux états avec un symbole donné.

```
void afficherletransition(Etat* etat ,char symbol, Etat* destination){
    printf("/  %s -> %c ->%s", etat->nom,symbol, destination->nom );
```

La fonction Searching recherche une transition spécifique à partir d'un état donné et avec un symbole donné. Elle parcourt les transitions de l'état et retourne l'état de destination correspondant si une transition correspondante est trouvée.

```
Etat* Searching(char word , Etat* etat , Automaton* automate){
Transition *transition = etat->head;
while (transition != NULL)
{
    if(transition->symbol== word){
        return getState(transition->destination, automate);
}

transition = transition->suiv;
}
return NULL;
```

La fonction Accepted vérifie si un mot est accepté par l'automate. Elle prend un mot en entrée, recherche chaque symbole du mot en utilisant la fonction Searching et affiche les transitions effectuées. Si le mot est accepté, elle renvoie true, sinon elle renvoie false.

```
bool Accepted( Automaton* automate , char* word){

Etat* etat = getState(automate->etatinitial, automate);
int wordlength = strlen(word);

for (int i = 0; i < wordlength; i++)
{
Etat* nextetat = Searching(word[i], etat, automate);
if (nextetat == NULL)
{
    return false;
}
afficherletransition(etat, word[i], nextetat);
        etat = nextetat;
}

return etat->isFinal;
```

La fonction getAutomaton lit un fichier contenant la description de l'automate et crée l'automate correspondant en utilisant les fonctions précédentes. Elle retourne un pointeur vers l'automate créé.

```
Automaton *getAutomaton(char *fichier)
{
    Automaton *automaton = (Automaton *)malloc(sizeof(Automaton));
    if (fichier == NULL)
    {
        printf("Error: could not open file %s\n", fichier);
        return NULL;
    }

    Etat *state = (Etat *)malloc(sizeof(Etat));

    FILE *file = fopen(fichier, "r");

    int nombreSymboles, nombreEtats, nombreEtatsFinaux;
    int nombreetatinitial;

    fscanf(file, "Nombre de symboles: %d\n", &nombreSymboles);
    fscanf(file, "Nombre d'etats: %d\n", &nombreEtats);

    fscanf(file, "Nombre d'etats initial: %d\n", &nombreetatinitial);
    fscanf(file, "Nombre d'etats finaux: %d\n", &nombreEtatsFinaux);
    automaton->num_symbols = nombreSymboles;
    automaton->num_states = nombreEtats;
    automaton->num_final_states = nombreEtatsFinaux;

    fscanf(file, "Symboles:");
    for (int i = 0; i < nombreSymboles; i++)
    {
        char symbole;
        fscanf(file, " %c", &symbole);
        automaton->symbols[i] = symbole;
    }
    fscanf(file, "\n");

    char etatinitial[256];
    fscanf(file, "Etats initial:%s", etatinitial);
    copyString(etatinitial, automaton->etatinitial, 256);

    fscanf(file, "\n");
```

```
char etatinitial[256];
fscanf(file, "Etats initial:%s", etatinitial);
copyString(etatinitial, automaton->etatinitial, 256);

fscanf(file, "\n");

fscanf(file, "Etats: ");
for (int i = 0; i < nombreEtats; i++)
{
    char etat[256];
    fscanf(file, " %s", etat);
    copyString(etat, automaton->etats[i].nom, 256);
}
fscanf(file, "\n");

fscanf(file, "Etats Finaux: ");
for (int i = 0; i < nombreEtatsFinaux ; i++)
{
    char etat_fineaux[256];
    fscanf(file, " %s", etat_fineaux);
    Etat *state = getState(etat_fineaux, automaton);
    state->isFinal = true;
}

char etat_initial[256], etat_destination[256];
char symbol;

fscanf(file, "\n");
while (EOF != fscanf(file, "delta (%s ,%c) %s\n", etat_initial, &symbol, etat_destination))
{
    Etat *state = getState(etat_initial, automaton);
    if (state != NULL)
    {
        Transition *toAdd = Create(symbol, etat_destination);

        addTransition(&(state->head), toAdd);
    }
    else
    {
        printf("Source State not found.\n");
    }
}
```

```
    fclose(file);
    return automaton;
}
```

La fonction main est la fonction principale du programme. Elle demande à l'utilisateur d'entrer le nom du fichier contenant la description de l'automate, crée l'automate en utilisant la fonction getAutomaton, affiche les détails de l'automate en utilisant la fonction Affichage, demande à l'utilisateur d'entrer un mot et vérifie si le mot est accepté par l'automate en utilisant la fonction Accepted. Enfin, elle libère la mémoire allouée pour l'automate.

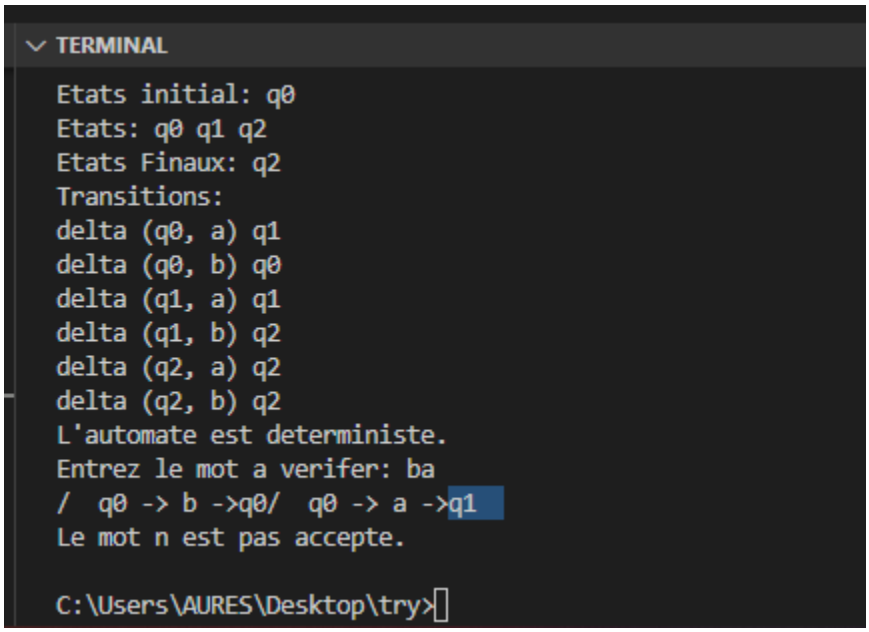
```
int main(int argc, char const *argv[])
{
    char file[MAX_FILENAME];
    printf("Entrez le nom du fichier: ");
    scanf("%s", file);
    Automaton *automaton = getAutomaton(file);
    if (automaton == NULL)
    {
        return 1;
    }
    Affichage(automaton);

    char word[MAX_FILENAME];
    printf("Entrez le mot a verifer: ");
    scanf("%s", word);
   bool isAccepted = Accepted(automaton, word);
if (isAccepted) {
    printf("\nLe mot est accepte.\n");
} else {
    printf("\nLe mot n est pas accepte.\n");
}


    free(automaton);
    return 0;
}
```

En résumé, ce programme implémente un automate fini déterministe et permet à l'utilisateur de vérifier si un mot donné est accepté par cet automate.

Exemple 1



Exemple 2