RAPPORT DU TP1 THP

Implémentation d'un autiomate simple déterministe

BEN RAMDANE Mustapha Kamal
YADDADEN ALI
1CS SB G07

2016/2017

TABLE DES MATIÈRES

INTRODUCTION

Un automate simple déterministe A est un quintuplé $\langle X, S, S_0, \mathbb{F}, \mathbb{I} \rangle$ telle que :
— Xest l'alphabet ;
— S est l'ensemble des états ;
— S_0 appartient à l'ensemble des états S est l'état initial ;
— F est un sous-ensemble de S est l'ensemble des états finaux ;
— I est une application : $S \times X \rightarrow S$ qui peut être vue comme un triplet $(S_i\,;\,x_i\,;\,S_j)$ telle que :
  — S_i est l'état de départ ;
  — S_j est l'état d'arrivé ;
  — x_i est la lettre lue par l'automate qui permettera de passer de S_i vers S_j.

Sa fonction principale est la reconnaissence de mots appartenants à un langage régulier. Sa transition d'un état à un autre ne se fait que par lecture d'une seule lettre à la fois du mot en entrée. De plus, cette transition se fait d'un état vers un seul autre état.

Dans ce TP, on se propose d'en implémenter un à qui on donnera le quintuplé mentionné ci-dessus en entrée. Il se chargera de reconnaître si un mot appartient au langage décrit par ce quintuplé. Puis, on essaiera d'implémenter l'algorithme qui tire l'automate réduit à partir de l'automate donné.

Rappelons qu'un automate réduit A' est un quintuplé $\langle X', S', S'_0, \mathbb{F}', \mathbb{I}' \rangle$ obtenu à partir de A en ne laissant que ses états accessibles et coaccessibles.

CHAPITRE

1

ANALYSE DU PROBLÈME

1.1 Construction de l'automate

L'automate A est complétement défini par son quintuplé $\langle X, S, S_0, \mathbb{F}, \mathbb{I} \rangle$ donc pour le construire, on renseigne son quintuplé.

X est un ensemble de symboles qui seront choisis préalablement par l'utilisateur.

S est l'ensemble des états, l'utilisateur se contentera juste de donner leur nombre et un numéro sera affecté à chaqu'un d'entre eux à partir de zéro jusqu'au nombre mentionné moins un. Par défaut, on choit que l'état de départ soit toujours S_0 le premier état déclaré.

L'ensemble des états finaux $\mathbb{F}$ sera choisit à partir de l'ensemble des états. C'est un sous-ensemble dont le cardinal est inférieur ou égal à celui de S.

L'ensemble des instructions $\mathbb{I}$ sera donné sous forme de triplet $(S_i\,;\,x_i\,;\,S_j)$. Pour X,S et $\mathbb{F}$, on peut convenir de les implémenter dans des tableaux à une dimension qu'on se contentera de parcourir au besoin. Pour le choix de l'implémentation de $\mathbb{I}$, il éxiste plusieurs représentations dont celle en forme de tableau à deux dimensions comme représenté dans l'exemple ci-dessous :

	a	b
S_0	S_0	S_1
S_1	S_2	S_0
S_2	S_2	$-$

Cette représentation traduit clairement le fait que I est une application.

$$I : (S \times X) \to S$$

Néanmoins, pour des raisons de facilité d'implémentation dans la machine, on lui préferera la représentation qui traduit le triplet $(S_i\,;\,x_i\,;\,S_j)$ comme dans l'exemple ci-dessous :

(S_0,a,S_0)	(S_0,b,S_1)	(S_1,a,S_2)	(S_1,b,S_0)	(S_2,a,S_2)

1.2 Réduction de l'automate

La réduction de l'automate se fera en trois temps suivant la définition :
— D'abord, on cherche les états accessibles ;
— Ensuite, on cherche les états coaccessibles ;
— Enfin, on fait l'intersection des deux ensembles, celui des états accessibles et celui des états coaccessibles.

Ainsi, on obtient l'ensemble des états réduits S'.

Un état S_i est dit accessible, s'il existe un mot w appartenant à X^* telle que à partir de S_0 (l'état de départ) et en suivant le chemin induit par les lettres du mot w dans l'ordre de leur écriture, on aboutit à S_i.

Un état S_i est dit coaccessible, s'il existe un mot w appartenant à X^* et état final S_k elle que en suivant le chemin induit par les lettres du mot w dans l'ordre de leur écriture à partir de S_i on aboutit à S_k.

Pour obtenir $\mathbb{F}'$, l'ensemble des états finaux de l'automate réduit, on fait l'intersection entre $\mathbb{F}$, l'ensemble des états finaux de l'automate de départ, et S'.

Pour obtenir $\mathbb{I}'$, l'ensemble des instruction de l'automate réduit, on cherche dans l'ensemble des instructions de l'automate de départ toutes les instructions $(S_i\ ;\ x_i\ ;\ S_j)$ dont Si appartient à S' et S_j appartient à S'.

Pour obtenir X', l'alphabet de l'automate réduit, on cherche à partir de X, l'alphabet de l'automate de départ, tous les x_i telle que x_i apparaît dans une instruction de $\mathbb{I}'$.

CHAPITRE

2

ALGORITHMES ET STRUCTURES DE DONNÉES UTILISÉES

Dans cette section, nous ne présenterons que les algorithmes pertinents du TP.

2.1 Structures de données et machine abstraite

Comme présenté dans la section analyse, on a besoin d'implémenter qu'une seule structure de données : c'est celle de l'instruction. Quant à l'état, on est suffisant de l'identifier par son numéro. Ci-dessous le détail de la structure de donnée et sa machine abstraite en langage C.

```
 typedef struct Instruction
{
    int Etat1; //état Si
    char lettre; //lettre de l'alphabet xi
    int Etat2; //état Sj
}Instruction;

/*alloue de l'espace mémoire pour Instruction*/
void allouer(Instruction** nouveau);

/*libère l'espace mémoire alloué pour Instruction*/
void liberer(Instruction** ancien);

/*affecte la valeur q dans le champs Etat1 de
l'instruction pointée par p*/
void aff_Etat1(Instruction* p,int q);

/*affecte la valeur q dans le champs Etat2 de
l'instruction pointée par p*/
```

```
void aff_Etat2(Instruction* p,int q);

/*affecte la valeur l dans le champs lettre de
l'instruction pointée par p*/
void aff_Lettre(Instruction* p, char l);

/*retourne la valeur du champs état1 pointé par p*/
int valEtat1(Instruction* p);

/*retourne la valeur du champs état2 pointé par p*/
int valEtat2(Instruction* p);

/*retourne la valeur du champs état1 pointé par p*/
char valLettre(Instruction* p);
```

2.2 Algorithme de vérification d'un mot

Fonction verifMot (mot : chaine ; $\mathbb{I}$: tableau d'instruction ; S : tableau des états ; $\mathbb{F}$: tableau des états finaux) : booléen
Variables :
    Arrêt : booléen ;
    EtatActuel : état [1] ;
    EtatIntermédiaire : état ;
    i : entier ;
DEBUT
    Arrêt← faux ;
    /*Booléen qui servira à arrêter la boucle si
    on trouve une instruction n'appartenant à $\mathbb{I}$*/
    EtatActuel ← S_0 ;// On commence toujours par l'état initial S_0
    Tant que ((i < longueur (mot)) et (Arrêt = faux)) faire
        DTQ
        EtatIntermédiaire← EtatActuel ;
        /*Sauvegarder l'état actuel dans un
        état intermédiaire pour l'impression*/
        Pour chaque (i ∈ $\mathbb{I}$) faire
            DPOUR
            Si((EtatActuel = valEtat1(i)) et (mot[i] = valLettre(i))) alors
            EtatActuel ← valEtat2(i) ;
            /*si le couple existe dans l'ensemble
            des instrcutions alors on affecte à l'état actuel l'état d'arrivé du couple*/
            FPOUR
        Si (EtatActuel = -1) alors
        /*Si après affectation l'état actuel
        est vide (valeure -1) donc (S_i, x_i, S_j) n'existe pas dans $\mathbb{I}$*/
            DSI
            Ecrire (« Instruction incorrecte») ;
            Arrêt← vrai ;
            FSI
        Sinon
            DSIN

1. En réalité, il s'agit du type entier mais pour des raisons de compréhension le type état a été mis.

```
            Ecrire(EtatIntermédiaire, mot[i], EtatActuel) ;
            /*On imprime à l'écran le triplet*/
             FSIN
        i ← i + 1 ;//on passe à la prochaine lettre dans le mot
         FTQ
    Si ((Arrêt = faux) et (EtatActuel ∈ 𝔽)) alors Ecrire(« le mot appartient au langage ») ;
    /*Si toutes les lettres mènent à un état et
    si le dernier état visité est un état final alors le mot est reconnu par l'automate.*/
FIN
```

2.3 Algorithme de réduction

2.3.1 Accessible

```
Procédure Accessible (𝕀 : tableau des instructions ; X : l'alphabet ; Var Acc : tableau d'états)
Variables :
    s : état ;
    x_i : caractère ;
    i : Instruction ;
DEBUT
    Acc← S_0 ;//l'état S_0 est accessible car par défaut c'est l'état initial.
     Pour chaque ((s ∈ Acc) et (x_i ∈ X) et (i ∈ 𝕀)) faire
        DPOUR
            Si ((valEtat1(i)= s) et (valLettre(i)=x_i)) alors Acc← Acc ∪ {valEtat2(i)} ;
            /*Si dans l'ensemble des instructions 𝕀, il existe
            une instruction i qui a comme état de départ s et
            comme lettre de transition x_i alors on ajoute dans
            le tableau d'état Acc la valeur de l'état d'arrivé valEtat2(i)*/
         FPOUR
FIN
```

2.3.2 Coaccessible

```
Procédure Coaccessible (𝕀 : tableau des instructions ; 𝔽 : tableau des états finaux ; Var Coacc :
tableau d'états)
Variables :
    i : Instruction ;
    s, s' : état ;
DEBUT
    /*On teste pour chaque état final s s'il existe un état dans les instructions qui arrive vers lui*/
     Pour chaque(s ∈ 𝔽) faire
        DPOUR
        Pour chaque (i ∈ 𝕀) faire
            DPOUR
            Si (valEtat2(i)=s) alors
                DSI
                Coacc ← Coacc ∪ {valEtat1(i)} ;
                FSI
```

```
            FPOUR
            /*On teste pour chaque état s' coaccessible s'il existe
            un état dans les instructions qui arrive vers lui*/
            Pour chaque (s' ∈ Coacc)
                DPOUR
                Si (s' ∉ 𝔽)
                /*Inutile de tester les états finaux car on boucle dessus dans la boucle principale*/
                    DSI
                    Pour chaque (i ∈ 𝕀) faire
                        DPOUR
                        Si (valEtat2(i)=s') alors
                            DSI
                            Coacc ← Coacc ∪ {valEtat1(i)} ;
                            FSI
                        FPOUR
                    FSI
                FPOUR
        FPOUR
FIN
```

2.3.3 Intersection

```
Procédure Intersection (Tab1 : tableau d'états ; Tab2 : tableau d'état ; var Tab3 : tableau d'état)
Variables :
    s, s' : état ;
DEBUT
    Pour chaque (s ∈ Tab1) faire
        DPOUR
        Pour chaque (s' ∈ tab2) faire
            DPOUR
            Si (s = s') alors
                DSI
                Tab3 ← Tab3 ∪ s ;
                FSI
            FPOUR
        FPOUR
FIN
```

2.3.4 L'algorithme général pour la réduction

```
Procédure réduction (A [2] : automate ; var A' [3] : automate)
Variables :
    Acc, Coacc : tableau d'états ;
    i : Instruction ;
    s : état ;
```

2. Par souci de lisibilité de l'entête de la procédure et d'allègement de l'écriture, le type automate a été employé pour référer au quintuplé $\langle X, S, S_0, \mathbb{F}, \mathbb{I} \rangle$ sans réellement avoir été implémenté.

3. A' = $\langle X, S, S_0, \mathbb{F}, \mathbb{I} \rangle$

```
DEBUT
    Accessible (𝕀, X, Acc) ;
    Coaccessible (𝕀, 𝔽, Coacc) ;
    Intersection (Acc, Coacc, S') ; //S' est l'ensemble des états de A'
     /* Pour Obtenir 𝕀' et X', respectivement, l'ensemble
    des instructions et l'alphabet de l'automate A' */
     Pour chaque (i ∈ 𝕀)
        DPOUR
        Si (ValEtat1(i) ∈ S') et (valEtat2(i) ∈ S') alors
            DSI
            𝕀' ← 𝕀' ∪ i ;
            X' ← X' ∪ {valLettre(i)} ;
            FSI
        FPOUR
    /* Obtenir 𝔽' l'ensemble des états finaux de l'automate A' */
     Pour chaque ( s ∈ 𝔽)
        DPOUR
        Si (s ∈ S') alors 𝔽' ← 𝔽' ∪ {s}
        FPOUR
FIN
```

CHAPITRE

3

TESTES ET RÉSULTATS DES TESTES

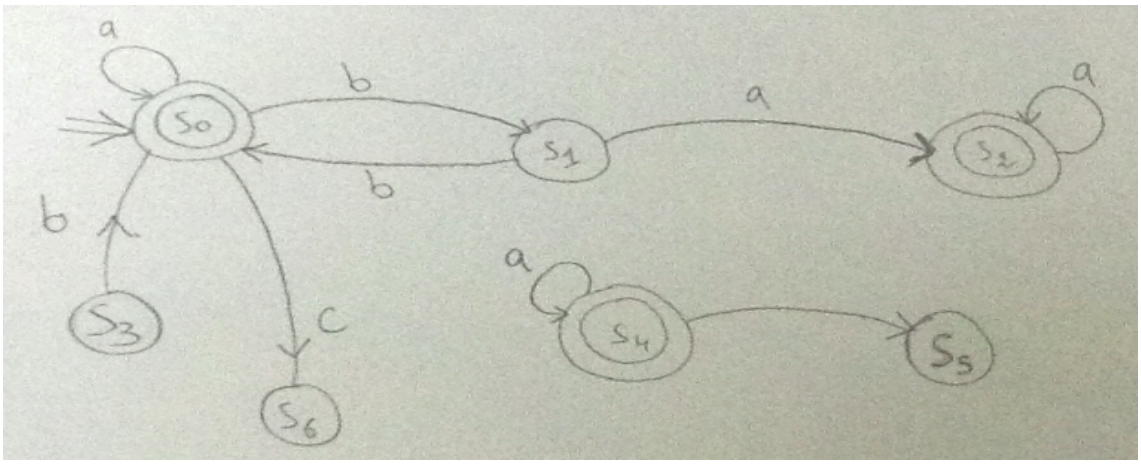
Quand on exécute le programme, le menu suivant apparaît sur l'écran :

```
Le menu:

        1) Construire l'automate
        2) Voir l'automate
        3) Tester un mot
        4) Reduire l'automate
        5) Voir l'automate reduit
        0) Sortir

        Votre choix:
```

Nous allons implémenter l'automate simple déterministe suivant :



Pour cela on choisit l'option 1 dans le menu. Voici le rendu en image :

```
Creation de l'automate:
Donner le nombre de symboles de l'alphabet:
3
Donner l'alphabet:
a
b
c
X={a; b; c}
Donner le nombre d'etats:
7
S={S0; S1; S2; S3; S4; S5; S6}
Est-ce que l'etat S0 est final? (o/n)o
Est-ce que l'etat S1 est final? (o/n)n
Est-ce que l'etat S2 est final? (o/n)o
Est-ce que l'etat S3 est final? (o/n)n
Est-ce que l'etat S4 est final? (o/n)o
Est-ce que l'etat S5 est final? (o/n)n
Est-ce que l'etat S6 est final? (o/n)n
F={S0; S2; S4}
Donnez le nombre d'instructions:
9
Nouvelle instruction:
Etat de depart: S0
Lettre: a
Etat d'arrivee: S0
Nouvelle instruction:
Etat de depart: S0
Lettre: b
Etat d'arrivee: S1
Nouvelle instruction:
Etat de depart: S0
Lettre: c
Etat d'arrivee: S6
Nouvelle instruction:
Etat de depart: S1
Lettre: b
Etat d'arrivee: S0
Nouvelle instruction:
Etat de depart: S1
Lettre: a
Etat d'arrivee: S2
Nouvelle instruction:
Etat de depart: S2
Lettre: a
Etat d'arrivee: S2
Nouvelle instruction:
Etat de depart: S3
Lettre: b
Etat d'arrivee: S0
Nouvelle instruction:
Etat de depart: S4
Lettre: a
Etat d'arrivee: S4
Nouvelle instruction:
Etat de depart: S4
Lettre: b
Etat d'arrivee: S5
II={(S0, a, S0); (S0, b, S1); (S0, c, S6); (S1, b, S0); (S1, a, S2); (S2, a, S2)
; (S3, b, S0); (S4, a, S4); (S4, b, S5)}
```

Pour être sûr que l'automate a bien été créé, on choisit l'option 2 qui nous permet de voir les détails de l'automate $\langle X, S, S_0, \mathbb{F}, \mathbb{I} \rangle$. Voici le résultat pour l'automate saisi en entrée :

```
Voir l'automate
L'alphabet: X={a; b; c}
L'ensemble des etats: S={S0; S1; S2; S3; S4; S5; S6}
L'ensemble des etats finaux: F={S0; S2; S4}
L'ensemble des instructions: II={(S0, a, S0); (S0, b, S1); (S0, c, S6); (S1, b,
S0); (S1, a, S2); (S2, a, S2); (S3, b, S0); (S4, a, S4); (S4, b, S5)}
```

L'automate étant prêt, on peut tester des mots avec. Voici trois tests réalisés :

```
Tester un mot:
Donner un mot: aab
( S0 ; a ; S0 )
( S0 ; a ; S0 )
( S0 ; b ; S1 )
Le mot n'appartient pas au langage
```

```
Tester un mot:
Donner un mot: abaa
( S0 ; a ; S0 )
( S0 ; b ; S1 )
( S1 ; a ; S2 )
( S2 ; a ; S2 )
Le mot appartient au langage
```

```
Tester un mot:
Donner un mot: acaa
( S0 ; a ; S0 )
( S0 ; c ; S6 )
( S6 ; a ; ?): Instruction incorrecte!

Le mot n'appartient pas au langage
```

Nous allons maintenant réduire l'automate. Pour cela on saisit l'option 4 dans le menu. Puis avec l'option 5, on consulte les détails de l'automate réduit. Voici le résultat :

```
Reduction de l'automate:
S'={S0; S1; S2}
II'={(S0, a, S0); (S0, b, S1); (S1, b, S0); (S1, a, S2); (S2, a, S2)}
X'={a; b}
F'={S0; S2}
```

On constate que l'automate a bien été réduit.

```
Voir l'automate reduit
L'alphabet reduit: X'={a; b}
L'ensemble des etats reduit: S'={S0; S1; S2}
L'ensemble des etats finaux reduit: F'={S0; S2}
L'ensemble des instructions reduit: II'={(S0, a, S0); (S0, b, S1); (S1, b, S0);
(S1, a, S2); (S2, a, S2)}
```

CONCLUSION

Ce TP nous a permi de comprendre le fonctionnement d'un automate simple déterministe et de son efficacité pour l'identification de mots appartenants à un langage régulier. De plus, ce TP nous a amenés à réfléchir sur l'algorithme de réduction d'un automate simple et déterministe. La structure de donnée choisie dès le départ nous a beaucoup aidé pour implémenter cet algorithme de manière simple en suivant les étapes vues en cours.

La principale difficultée observée sont les contraites du langage C. Ce langage ne possède pas de structures de données implémentées au départ. Donc, il fallait tout implémenter manuellement. Toutefois, cela a été un point positif du fait que cela nous a aidé à mieux comprendre le fonctionnement de l'automate en mettant au points des structures et des algorithmes divers qui allaient dans ce sens. De plus, il faut avoir une bonne maîtrise des pointeurs pour ne pas tomber dans des erreurs de ségmentation.