Rapport Projet de stage : Limitedness

Brebant Alexandre
Xue Juedong

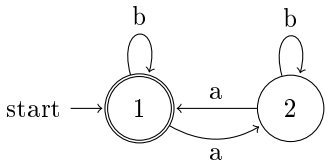
27 juillet 2014

Table des matières

Chapitre 1

Introduction au problème

Durant ce stage, nous avons voulu approfondir nos connaissances dans une branche de l'informatique théorique que nous avons commencé à étudier lors de ce dernier semestre de licence. Ce sont donc les automates et les interrogations qui gravitent autour de cette structure qui nous ont intéressés.
Un automate est une machine abstraite qui va accepter ou rejeter des mots dans un alphabet fini. Voici l'exemple d'un automate très basique qui accepte tous les mots sur l'alphabet $\{a, b\}$ qui comportent un nombre pair de a.



Pour représenter graphiquement un automate, on utilise donc un graphe orienté dans lequel les sommets du graphe sont les états de l'automate et où chaque arrête correspond à une transition de l'automate, c'est à dire à la lecture d'une lettre. Un mot est accepté par un automate $\mathcal{A}$, s'il existe un chemin dans celui-ci, partant d'un état initial et allant jusqu'à un état final en lisant les lettres une par une. Un tel chemin est appelé *chemin acceptant* et l'ensemble des mots acceptés par un automate est appelé *Langage*, on le note $L(\mathcal{A})$.

Formellement, un automate est défini, par 5 ensembles, de la façon suivante :

$$\mathcal{A} = \{A, Q, I, F, \delta\} \text{ avec,}$$

A : l'alphabet
Q : l'ensemble des états
I : l'ensemble des états initiaux
F : l'ensemble des états finaux
δ : l'ensemble des transitions

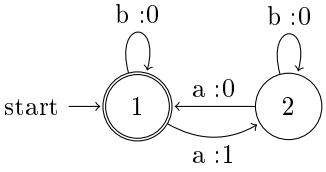
Nous pouvons donc définir l'automate ci-dessus de cette manière :

$$\mathcal{A} = \left(\begin{array}{l} A = \{a, b\},\ Q = \{1, 2\},\ I = \{1\},\ F = \{1\},\ \delta = \left\{ \begin{array}{lcl} (1, a) & \to & 2 \\ (2, a) & \to & 1 \\ (1, b) & \to & 1 \\ (2, b) & \to & 2 \end{array} \right. \end{array} \right)$$

La théorie des automates et des langages est un domaine très actif depuis le milieu du 20e siècle et il existe aujourd'hui plusieurs modèles d'automates nés pour répondre à certaines problématiques.

Ainsi les automates permettent de caractériser certains langages, par exemple, les automates finis qui sont le modèle de base des automates, définissent ce qu'on appelle les langages rationnels ([**?**]). Mais il existe des extensions à ce modèle, comme par exemple les automates à pile qui permettent de définir les langages algébriques ([**?**]), qui sont grandement utiliser en analyse syntaxique. Et c'est une autre extension de l'automate fini qui va nous intéresser dans ce rapport : les automates à distance (distance automata).
Ce sont des automates finis, non déterministes, dans lesquels les transitions sont pondérées. On associe donc un coût pour chaque mot accepté par l'automate. Le coût d'un mot correspond au minimum des coûts de tous ses chemins acceptants dans l'automate .

Reprenons l'automate représenté plus haut en y ajoutant des coûts sur les transitions :



Avant d'aller plus loin dans la description de cet automate, nous devons définir une notion élémentaire : le déterminisme. Un automate est déterministe si et seulement si il possède un unique état initial et que pour tout état, il existe au plus, une transition par lettre de l'alphabet.
Cependant, la déterminisation des automates à distance est encore une question ouverte aujourd'hui. On dit alors qu'un automate à distance est déterministe si l'automate sans les poids l'est.

Notre automate à distance est donc déterministe. Cela signifie qu'il n'existe qu'un seul chemin acceptant pour tout les mots reconnus par l'automate et donc qu'il n'existe également qu'un seul coût pour ce mot. On peut observer dans cet exemple que le coût correspondra au nombre de paires de a dans le mot.

Ce type d'automate a été présenté par Hashiguchi dans [?] pour la résolution d'une célèbre problématique, la question de hauteur d'étoile (star height problem) [?]. Mais les automates à distance posent également un autre problème, la limitation des automates à distance (Limitedness problem).
Le problème de limitedness est très simple à comprendre, il est simplement question de savoir si il existe une majoration des coûts pour tous les mots acceptés par un automate à distance.
La décidabilité de ce problème n'est pas évidente. En effet il existe des problème similaires dont on a prouvé leur indécidabilité, c'est à dire la preuve qu'il n'existe pas d'algorithme capable de résoudre le problème. Il est par exemple prouvé que la question suivante est indécidable :

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates à distance distincts. Est ce que le coût de w dans $\mathcal{A}_1$ est égale au coût de w dans $\mathcal{A}_2$, $\forall w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$?

Le travail d'Hashiguchi étant considéré comme difficile à comprendre, d'autre chercheurs, comme Simon [?] ou Leung [?], ont étudié le problème et apporté d'autres solutions sur le sujet.

Notre travail a donc consisté, dans un premier temps, à lire et comprendre les différents travaux de ces chercheurs afin d'être capable d'implémenter un algorithme permettant la détection de limite dans un automate à distance.

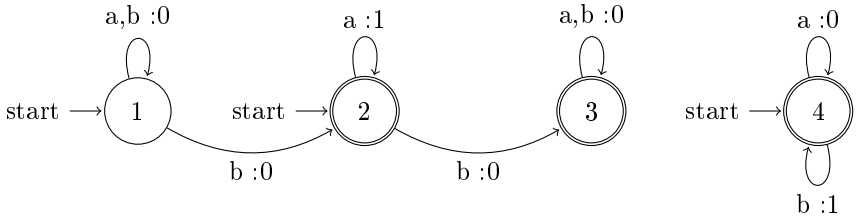
Première partie

Mise en place théorique de l'algorithme

Chapitre 2

De l'automate aux matrices

Le papier dont nous nous sommes le plus servi afin de comprendre et résoudre le problème de limitedness, est [**?**] mais il nous a d'abord fallu comprendre la complexité du problème.
Pour simplifier les calculs de coûts, il fallait utiliser des structures mathématiques pour représenter la lecture d'une lettre ou d'un mot. Les matrices permettent de créer des "tables de coût" pour la lecture d'une lettre sur les différents états. Pour la suite du rapport nous utiliseront l'automate suivant en tant que référence pour nos exemples :



Automate $\mathcal{A}_{ref}$

Cet automate est-il limité ? Plus formellement :

$$\exists ?B \in \mathbb{N} \text{ tel que } \forall w \in L(\mathcal{A}_{ref}),\ cost(w) \leq B$$

La réponse n'est évidemment pas triviale. C'est pourquoi nous allons utiliser des matrices pour représenter les différents coûts pour une lecture de lettres. La matrice $M(a)$ va par exemple contenir le coût de la lecture de la lettre a depuis tout les états de l'automate de la manière suivante :

$$M_{i,j} = \begin{cases} n\ , & s'il\ existe\ une\ transition\ (i, a : n) \to j \\ \infty, & sinon \end{cases}$$

Pour $\mathcal{A}_{ref}$, on a donc par exemple :

$$M(a) = \begin{array}{c} \left(\begin{array}{cccc} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{array}\right) \\ \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}$$

et

$$M(b) = \begin{array}{c} \left(\begin{array}{cccc} 0 & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 \end{array}\right) \\ \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}$$

Ces matrices sont donc définies sur $\{\mathbb{N} \cup \infty\}$, nous les appellerons matrices de coûts. Leur fonctionnement est très simple : la numérotation de chaque ligne et de chaque colonne correspond aux états de l'automate. Par exemple, dans la matrice de coûts pour la lettre b, on a $M(b)_{(2,3)} = 0$. Cela signifie que le coût pour aller de l'état 2 à l'état 3 en lisant un b est de 0, ce qu'on peut vérifier sur l'automate.

Avant d'aller plus loin il est nécessaire de définir la structure algébrique qui est au cœur de l'algorithme de résolution du problème de limitedness, les semi-anneaux.
Un semi-anneau est formé d'un ensemble E muni de deux lois associatives, $\oplus$ et $\otimes$, et d'un élément neutre pour chaque loi. De plus, la première loi est commutative et la seconde est distributive sur la première.
Exemple : Les entiers naturels, munis de l'addition et de la multiplication, forment un semi-anneau : $\{\mathbb{N},\ +,\ *\}$. Les éléments neutres sont 0 pour l'addition et 1 pour la multiplication. Ces deux lois sont bien associative, l'addition est commutative et la multiplication est distributive sur l'addition. On remarque que la multiplication est également commutative, on dit que $\mathbb{N}$ est un semi-anneau commutatif.

Cette structure va nous servir pour définir la multiplication de deux matrices de coûts. Le premier semi-anneau que nous allons définir est celui formé par l'ensemble $\{\mathbb{N} \cup \infty\}$ tel que :
- $a \oplus b = min(a,\ b)$, avec élément neutre ∞
- $a \otimes b = a + b$, avec élément neutre 0

Nous avons maintenant la possibilité de multiplier deux matrices de coûts entre elles afin d'obtenir la matrice correspondant à la concaténation des mots des matrices. On a donc :

$$M(ab) = M(a) \times M(b) = \begin{pmatrix} 0 & 0 & \infty & \infty \\ \infty & \infty & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 \end{pmatrix}$$

Il est ainsi possible d'obtenir les matrices de coûts pour n'importe quel mots composé de lettres de l'alphabet de l'automate. Et pour obtenir le coût du mot, il faut prendre le minimum des coûts situés dans l'intersection des états initiaux en ligne et des états finaux en colonne. Nous représentons les coûts compris dans cette intersection en rouge.
On peut donc dire que $cost(ab) = M(ab)_{(1,2)} = 0$ et il s'obtient en partant de l'état 1 vers l'état 2.

Cela ne nous permet cependant pas de déterminer si l'automate est limité ou pas. En effet deux facteurs rendent le nombre de matrices infini : l'ensemble de définition des matrices $\{\mathbb{N} \cup \infty\}$ et le nombre de mots acceptés par l'automate qui est lui aussi infini.

Chapitre 3

Matrices idempotentes

Dans ce chapitre, nous allons faire évoluer nos matrices afin de rendre leur nombre limité. Tout d'abord, il est nécessaire de modifier le semi-anneaux qui est défini sur un ensemble infini. Dans le problème de limite, la valeur du coût en elle-même n'a pas vraiment d'utilité. Ce qui nous intéresse, c'est de savoir si cette valeur est fixe ou si elle va croître avec la longueur du mot. Si elle croît, ce signifie qu'il existe une boucle dans l'automate dont le poids est strictement supérieur à 0.
Nous allons donc définir un ensemble de 4 valeurs qui suffiront à représenter l'intégralité des coûts possibles :

- 0 qui correspond au coût gratuit
- 1 qui correspond à un coût potentiellement grand mais fixe
- ω qui correspond à un coût instable, qui va croître
- ∞ qui a toujours la même signification, à savoir correspondre aux coûts impossibles à payer

Et on a donc $0 < 1 < \omega < \infty$, il reste maintenant à définir les deux lois associatives du nouveau semi-anneau. La première est identique car nous souhaitons toujours obtenir le minimum parmi tous les chemins : $a \oplus b = min(a,\ b)$. Mais l'addition de coûts ne nous intéresse plus, nous voulons maintenant connaître le plus gros coût parmi toutes les transitions du chemin de lecture d'un mot. On prend donc $a \otimes b = max(a,\ b)$.
Nous formons ainsi le semi-anneau qui est le sujet principal de l'article de Simon, le semi-anneau tropical que nous nommeront également $\mathcal{R}$.

On remarque tout de fois qu'à partir des matrices de base dont on dispose et de ces deux nouvelles lois, il est impossible d'obtenir la valeur ω par simple multiplication matricielle dans $\mathcal{R}$. Il nous est donc impossible de détecter les coûts instables, en effet certaines positions contenant la valeur 1 vont rester à inchangées après une multiplication alors que le coût réel va croître d'une certaine valeur pour chaque lecture du mot représenté par la matrice.

Reprenons par exemple notre matrice $M(a)$:

$$M(a) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

Calculons maintenant $M(aa)$ dans $\mathcal{R}$:

$$M(aa) = M(a) \times M(a) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

On remarque que $M(aa) = M(a)$. Ce qui implique que $M(aaaa...) = M(a)$ et donc que $M(aaaa...)_{(2,2)} = M(a)_{(2,2)} = 1$.
Or on peut observer sur l'automate que la lecture répétée de la lettre a en bouclant sur l'état 2 va causer un coût croissant dépendant du nombre de a lus.

Pour distinguer ce genre de coût de ceux qui ne causent pas ce problème de croissance infinie, il faut donc ajouter un traitement supplémentaire aux matrices dont la multiplication avec elles-même dans R donne la même matrice. On appelle ces matrices des matrices idempotentes.

Le dernier chapitre de cette partie théorique va donc consister à la détection des coûts instables dans les matrices idempotentes.

Chapitre 4

Matrices stables

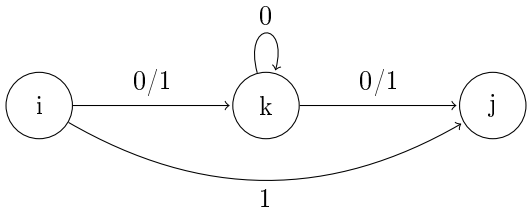
Nous avons donc vu que lorsqu'une matrice est idempotente, il y a un risque que certaines positions voient leur coût exploser à chaque lecture du mot de cette matrice. Il s'avère donc nécessaire de représenter l'itération d'un mot d'une matrice.
Au lieu d'écrire $M(aaaaa...)$ comme nous l'avons fait dans le chapitre précédent, nous allons introduire un nouveau symbole afin de simplifier la notation et écrire ce genre de matrices de la façon suivante : $M(a^{\#})$.

Soit w un mot, tel que $M(w)$ idempotente, la matrice $M(w^{\#})$ va donc contenir les coûts dans $\mathcal{R}$ d'un nombre quelconque de lectures successives du mot w.
Dans ces matrices que nous appellerons simplement matrices dièse, il reste donc à déterminer la valeur de chaque position. Dans son article, Simon défini le terme "ancrée" pour définir les positions dont le coût est stable ou va se stabiliser au bout d'un nombre fini d'itération.

Lemme : *Soit w un mot tel que $M(w)$ idempotente dans $\mathcal{R}$. Si $M(w)_{(i,j)} = 0$ alors la position (i, j) est ancrée. De plus, la position (i, j) est ancrée si il existe $k \in [1,\ n]$ qui respecte les conditions suivantes :*

- $M(w)_{(k,k)} = 0$
- $M(w)_{(i,j)} = max(M(w)_{(i,k)},\ M(w)_{(k,j)}) = 1$

Graphiquement, cela se schématise de la façon suivante :



- $M(w)_{(i,j)} = max(M(w)_{(i,k)},\ M(w)_{(k,j)}) = 1$ signifie qu'on peut atteindre l'état k en payant un coût fixe puis rejoindre en j avec un coût également fixe.
- $M(w)_{(k,k)} = 0$ correspond à une boucle de coût 0 sur l'état k. Elle va permettre d'absorber les lectures successives de w.

Cette configuration permet donc une lecture multiple d'un mot sans que le coût ne grandisse avec le nombre de lectures. D'un point de vue matricielle, il suffit d'appliquer le lemme pour chaque position de la matrice pour laquelle nous voulons obtenir sa "version dièse". Si une position n'est pas ancrée, on dit qu'elle est instable. On a donc :

$$M(w^{\#})_{(i,j)} = \begin{cases} \omega\ , & si\ M(w)_{(i,j)} = 1\ et\ (i,j)\ est\ instable \\ M(w)_{(i,j)}, & sinon \end{cases}$$

Revenons maintenant sur notre automate $\mathcal{A}_{ref}$. Nous avons vu que sa matrice $M(a)$ était idempotente et que par conséquent, il était nécessaire de déterminer sa stabilité.

$$M(a) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

On a $M(a)_{(2,2)} = 1$ et on remarque qu'on peut prendre $k = 1$ ou $k = 3$ ou $k = 4$ car on a $M(a)_{(1,1)} = M(a)_{(3,3)} = M(a)_{(4,4)} = 0$.
Cependant, la deuxième condition du lemme n'est vérifiée pour aucun de ces k :

$$\forall k \in \{1,\ 3,\ 4\}, \quad M(a)_{(i,j)} \neq max(M(a)_{(i,k)},\ M(a)_{(k,j)})$$

La position $(2,2)$ est donc instable, on peut alors déterminer la matrice $M(a^{\#})$:

$$M(a^{\#}) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & \omega & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

Nous sommes maintenant en possession de l'intégralité des éléments qui vont nous permettre de déterminer si un automate à distance est limité ou non. Comme nous l'avons vu, $\mathcal{R}$ est un semi-anneau défini sur un ensemble de 4 valeurs. Le principe de l'algorithme sera donc le calcul de l'intégralité des matrices de l'automate pour chacune de ses lettres, puis en cas d'impotence, il

faudra calculer la matrice dièse correspondante.
Une fois l'intégralité des matrices calculée, il ne reste qu'à regarder les valeurs situées dans l'intersection des états initiaux en ligne et des états finaux en colonne comme nous l'avons expliqué dans le chapitre 2. Si le minimum dans chacune de ces matrices est 0 ou 1, alors l'automate sera limité. Mais si l'une (ou plusieurs) d'entre elles a pour minimum ω, alors l'automate ne sera pas limité et le mot correspondant à cette matrice sera celui qui causera le coût non limité.

Le nombre de matrices calculé augmente très rapidement avec le nombre d'états puisque celui ci défini n, la taille des matrices. Le nombre de matrices possibles est donc égal à 4^{n^2}.
Pour $\mathcal{A}_{ref}$ par exemple, on a $n = 4$ et donc possiblement $4^{4^2} = 4\ 294\ 967\ 296$ matrices.

Une fois cette partie théorique assimilée, ils nous a donc fallu mettre en place des structures permettant le stockage important de matrices ainsi qu'un accès efficace à ces dernières. Nous allons donc dans cette deuxième partie décrire l'implémentation des différentes fonctions et structures que nous avons mis en place ainsi que les choix que nous avons fait pour, à la fois être efficace mais également conserver une certaine clarté dans le code pour que cette partie théorique soit visible au sein des fonctions.

Deuxième partie

Exécution détaillée du programme

Chapitre 5

Réutilisation du programme d'Adrien Boussicault

(présentation des différents modules et fonctions disponible + détails des ajouts pour la gestion des coût, de l'affichage et autres fonctions additionnelles que nous avons ajouté)

Chapitre 6

Création et manipulation des matrice

(creer_matrice_transition, multiplication, etc...)

Dans notre sujet, la matrice représente la matrice d'incidence. Les indices dans la matrice sont des étiquettes d'état. Du coup,dans notre cas,toutes les matrices sont de la forme carrée n x n , dont n est le nombre d'états.

D'après ce fait,on construit la matrice avec la fonction créer_matrice(int taille). La matrice contient la taille le mot et un tableau de deux dimensions du type int. alors ,on alloue le mémoire successivement pour la structure de la matrice et le tableau d'int de deux dimensions.

Également, pour la destruction, si la matrice est déjà NULL, on la laisse tranquille, sinon on des-alloue la structure de la matrice ainsi pour le tableau de deux dimensions.

Pour accéder dans la matrice, on a trois successeurs pour obtenir les informations sur le tableau, le mot et la taille de la matrice.

vous voyez bien que dans notre structure de la matrice il y a un "mot" qui est du type tree, ce mot est pour exprimer la matrice sous forme d'expression rationnel. le mot contient trois genres de choses :
l'alphabet (dans R a,b,c,etc.)
la dièse '#' (la répétition comme * dans l'expression rationnel)
le point "." ("."signifie la concaténation)
Car on construit le mot par la manipulation de concaténation en collant les sous mot, on include arbresytaxique.h pour bien résoudre le problem de construction du mot.

La fonction print_matrice_in_R permet d'afficher la matrice de façon mathématique. en précise, on définit INFINI est 3 et OMÉGA est 2 dans Matrice.h, ici on reconverti INFINI et OMÉGA comme ∞ et ω .

La fonction multiplication fais pas la multiplication normale de matrice. en fait on prend deux matrices de la même taille comme paramètres m1 et m2, en suite, on crée un nouveau matrice m3.on initialise tous les élément de m3 a -1.puis on parcours toutes les cases des deux tableaux de matrice m1 et m2. on compare les valeurs de la même position de ces deux matrices et redéfinie la valeur de m3 comme le max de m1 et m2,si sa valeur est -1.sinon on définit la valeur comme le min des deux matrices dans la même position.

Le principe de la fonction est_idempotent est facile, on fait la multiplication du matrice dans R et on teste si le produit généré reste pareille comme ce matrice.si c'est vrai alors c'est une matrice idempotent.

La fonction stable est produise par le théorème de

La fonction creer_mattice_diese calcul et renvoie la matrice dièse d'une matrice dans R.si la matrice n'est pas dans R,alors on peut pas le traiter,on retourne NULL et print "erreur".si c'est d'une matrice dans R,on crée une nouvelle et le remplie la matrice par la règle suivante : si tab[i][j] vaut 1 et a la position i j,la valeur est stable,alors on définit tab[i][j] a OMEGA.

Chapitre 7

Structure de stockage des matrices

(mautomate, description des fonctions de création) La structure de matrice automate inclut un automate,une table qui contient les états,une table qui contient les lettres.

Il y a des accesseurs pour lire le coût,la lettre,le origine de la clé.la fonction comparer_cle utilise ces accesseur pour comparer deux clés données.elle va retourner 0 si et seulement si les deux clés sont égaux.

Pour créer une automate,on tout d'abord alloue la mémoire nécessaire pour lui.en suite, on crée et initialise 6 choses pour cette automate : l'ensemble des états,l'ensemble des alphabets,l'ensemble des transitions,l'ensemble des états initiaux,l'ensemble des états finaux et l'ensemble des états vide.

Également,pour détruire une automate,on libère successivement les 6 choses dans l'automate et libère l'automate lui-même à la fin.

On a tous les outils pour accéder l'automate et le modifier.par exemple,on a get_etats pour lire tous les états dans l'automate.on a aussi ajouter_etats pour insérer une nouveau états dans l'automate.ainsi pour les autres choses dans l'automate.

La fonction copier_automate retourner une copie de l'automate donnée,on crée et alloue une automate qui s'appelle res,en suites ,on remplie la même contenu dans l'automate copie en parcourant successivement les 6 choses notées dessus.par exemple,pour copier les états dans la nouvelle automate,on parcourt les états un par un en lisant l'ensemble des états avec l'accesseur get_etats,et ajouter les états lus dans l'automate res avec la fonction ajouter_etat.on le boucle si le suivant de cet ensemble parcourir par nous n'est pas vide.on fais la

même choses pour les autres.à la fin,on retourne cet automate res.

la fonction creer_automate_etat_different renvoie l'automate destination,dont le nom de tous les etat est different du nom des états de l'automate source en utilisant la fonctions get_etat_libere,on parcourt tous les états dans l'automate,s'il y a des états non utilisée, on prend ce état qui est libre et non utilisée pour remplace le ancien état dans l'automate source en gardant tous ces ancien transitions lient avec les autres états.la fonction creer_automate_etat_0_n a la même fonctionnalité,elle renomme les états en entier de zero à n.

la fonction translater_etat,on parcourt l'ensemble des états de l'automate,pour chaque état on additionner l'entier n et ajouter l'état dans la nouvelle automate.on parcourt les translations d'automate,on copie successivement les clés.on lit chaque valeur des clés,ajouter la transition dans la nouvelle automate avec l'origine + n , fin + n , le coût et la lettre reste non changée.

la fonction print_mautomate affiche l'automate des matrices puis affiche les matrices de l'automate avec un nombre que vous voulez.

supprimer_etats
ajouter_transitions
est_complet
completer_automate
calcule_diese_mautomate
est_limite
//pour créer un mautomate à partir d'un automate a,on alloue un mémoire pour la structure de mautomate qu'il nécessaire .dans le mautomate on ajoute l'automate,le table des matrices des états ,le table des matrices des lettres.on créer un automate abis qui est de l'état 0 à n.
//?
//on complète l'automate des matrices de m,calcul le dièse de mautomate de m. Puis on libère l'automate abis.renvoie le mautomate m à la fin d'exécution de cette création.
la libération de mémoire pour le mautomate libère l'automate au début ,puis on parcourt le iterateur de matrices d'états et libère tous les matrices.en suite ,il libère le table des états et le table des lettres.à la fin,on libère l'automate a.

Chapitre 8

Résolution du problème de limitedness

Dans la fonction main du fichier test_automate.c,on crée une automate comme on veux en ajoutant des états initiaux,des états finaux et des transitions qui lient les différent états.en suites,on va tester cet automate ,regarde s'il est limite.

Tout d'abord,on va essayer d'afficher cet automate dans un graphe visuelle,on inclue un outils pour afficher l'automate qui s'appelle graphiz, ce outils va lire un fichier formelle qui comprendre les informations de l'automate.

on utilise la fonction creergraphe pour créer un fichier auto.gv qui porte tous les informations formelle dépend notre automate construit au début.en utilisant ce fichier auto.gv et la commande de graphiz "dot -Tpng auto.gv -o auto.png" on obtient le graphe auto.png.on le ouvrir avec le logiciel d'image visionneur par défaut .

en suite,la première étape pour test la limite de l'automate est la création du mautomate.on afficher la limite de l'automate avec la fonction est_limite.s'il n'est pas limite,on retourne le premier matrice qui provoque le coût infini de cet automate.on print le mautomate a la fin.