

Tai de Mathématiques pour l'informatique



Sommaire

Introduction	3
Les structures adoptées	4
La transition	4
L'état en lui-même	5
Restaurer un af	6
La création de notre table vierge	6
Mettre les EI et EF	7
Remplir les transitions	7
Description textuelle & schématique	7
Determiniser	9
Vérifier s'il est déterministe	9
Plusieurs états initiaux	9
Une transition va vers deux états	9
Le déterminer	10
Créer le nouvel état initial	10
Compléter	13
Vérifier s'il est complet	13
Le compléter	13
Minimiser	15
La table de corrélation	15
La structure Groupe	15
La structure Tab_corr	15
Séparer les Etats terminaux et non-terminaux	16
La minimisation en elle-même	16
Lire des mots	19
Conclusion	20

Introduction

Après avoir passé le semestre à étudier les mathématiques pour l'informatique, nous n'avons jusque-là expérimenté notre savoir uniquement sur papier. Or, ce TAI aura pour objectif de faire un lien entre nos connaissances acquises sur les automates et celles acquises dans nos cours d'algorithmie et programmation C.

C'est pourquoi nous avons pour objectif d'implémenter en C un « générateur d'automate » qui fera toutes les opérations élémentaires vues en cours sur les automates. De plus, notre programme devra aussi utiliser ces automates pour reconnaître des mots.

Les structures adoptées

Pour commencer, il a fallu bien évidemment réfléchir aux différentes structures à adopter. Lorsque le projet a commencé, nous n'avions pas encore une base assez solide pour coder le tout en C++, alors nous avons fait le choix d'utiliser du code C avec des structures pour modèle.

La transition

Il a fallu tout d'abord concevoir ce qu'est une transition. Au début nous avons pensé à un pointeur qui pointerait vers l'état suivant tout simplement.

Toutefois, après réflexion nous aurions eu deux problèmes avec ce type de méthode :

-Lors de la création de l'automate, nous n'aurions pas pu initialiser nos pointeurs tant que les états cibles n'avaient pas été créés, ce qui nous aurait poussé à créer l'état, revenir en arrière pour initialiser les pointeurs, enfin vous l'aurez compris ça n'aurait pas été évident.

-Aussi, un seul pointeur ne peut pas aller vers deux adresses distinctes, donc deux états, alors la création d'automates non-déterministes n'aurait pas été possible.

Ainsi nous avons choisi la structure suivante :

Nom du caractere

Nombre d'états associés

Tableau contenant le
nom des états associés

Note : « Nombre d'états associés » permettra deux choses :

-Pour notre automate fini, ceci permettra de détecter s'il est déterministe, si le nombre d'états associés à une transition est supérieur à 1 alors notre automate n'est pas déterministe.

-Pour le reste des automates, ça stockera la taille utile du tableau contenant le nom des états associés (qui seront des assemblages de nos états de base dans l'AF)

L'état en lui-même

Le sujet suggérait que le nom de nos états irait en ordre croissant : 1,2,3,4 etc...

Toutefois son nom ne peut pas comporter qu'un seul numéro car sinon l'étape de détermination ne pourrait pas fonctionner. En effet, lorsqu'on détermine, on va scinder plusieurs états ainsi que leurs transitions pour n'en faire qu'un seul, c'est pourquoi le nom de ce nouvel état comportera le nom de tous les états qui le composent.

Ainsi le nom de l'automate sera dans un tableau dynamique.

Tableau contenant le nom de l'état
Le nombre de "noms" qui composent l'état
Indicateur pour dire qu'il est initial
Indicateur pour dire qu'il est terminal
Tableau de transitions

Vous l'aurez remarqué ! Notre structure contient la structure transition que nous venons d'aborder ! Ainsi voici un petit exemple :

Admettons que notre langage aille de 'a' à 'd' alors notre tableau de transition comportera 4 champs où Tab[0] sera le caractère 'a', Tab[1] sera le caractère 'b' et ainsi de suite...

Sauf que cette structure ne correspond qu'à un seul état. Donc il faudra bien évidemment créer un tableau d'états qui représentera la table de transition de notre automate.

Restaurer un af

Nous avons opté pour un fichier .txt sous cette forme :

```
X //Nombre d'états#
X //Nombre de transitions*
X,Y,Z... //Ensemble des états initiaux+
X,Y,Z... //Ensemble des états terminaux
//Liste des transitions-
XaY,
XbZ,
!
```

Note Ici, X,Y et Z ne sont que des exemples, normalement nous mettons des états.

Aussi, vous pourrez remarquer la présence de '#' ou bien '' par exemple, ceux-ci ont leur utilité puisqu'ils feront office de marqueurs pour se retrouver facilement dans le fichier.*

La création de notre table vierge

Pour commencer, on récupèrera le nombre d'états puis le nombre de transitions.

ATTENTION : Pour ceci, il ne suffit pas de lire qu'un seul caractère, puisqu'imaginons nous ayons 100 états, alors nombre d'états comportera 3 caractères. C'est pourquoi nous avons créé une fonction `recup_chiffre()` qui lira le fichier tant que le curseur sera confronté à un chiffre et convertira le texte en chiffre.

Exemple avec 100 : soit '/' le curseur.

Au lancement de la fonction on a le curseur au début : /100

On lit le premier caractère : 1/00, c'est un chiffre alors on continue et enregistre le chiffre.

On lit le deuxième : 10/0, pareille.

Le troisième : 100/ pareille.

Le quatrième : 100 / et là ce n'est pas un chiffre alors on n'enregistre pas le caractère et retourne le chiffre.

Revenons-en à nos états et nos transitions : On lit le premier chiffre du fichier qui correspond au nombre d'états. Ensuite, on va jusqu'à notre marqueur '#' et on lit le chiffre qui correspond à notre nombre de transitions.

Ainsi avec ces deux variables, on va créer notre table vierge selon les étapes suivantes :

- On fait une allocation dynamique de notre tableau d'états égale au nombre d'états (Prenons N par exemple).

- Pour chaque états, on initialise son nom qui ira de 1 à N. Mais on initialise aussi chacun des états comme étant ni initial, ni terminal.

- Aussi, pour chaque état, on initialise sa table de transitions où chaque transitions ne va vers aucun autre état.

Mettre les EI et EF

Après avoir fait une table vierge on va récupérer l'ensemble des états-initiaux et terminaux. Pour cela, on commence par aller à notre marqueur '*' et on lit des chiffres jusqu'à ce qu'on tombe sur le caractère '/' ce qui signifie que c'est la fin de l'ensemble de nos EI ! (En effet, nos états sont séparés par des ',' donc on attend d'atteindre le '/'). Pour chaque chiffre reçu, on va à l'état correspondant dans notre table et on initialise son champ qui définit s'il est initial ou non.

Puis on va au marqueur '+' et on fait la même chose pour les terminaux.

Remplir les transitions

Cette partie aura été la plus délicate car il faut gérer toutes les allocations dynamiques. Ainsi, pour bien comprendre nous allons faire une description textuelle et schématique du processus réalisé.

DESCRIPTION TEXTUELLE & SCHEMATIQUE

Voici les étapes pour créer nos transitions. Pour chaque état nous allons faire la même logique :

//Liste des transitions-

1a2,
1b2,
1c3,
2a3,
!

- On lit le premier caractère → s'il correspond au nom de notre état, alors on continue, sinon c'est qu'il n'a aucune transition associée et on passe à l'état suivant et refaisons cette étape.

//Liste des transitions-

1a2, 1
1b2, 2
1c3, 3
2a3,
!

-On va compter le nombre de lignes où le premier caractère correspond à notre état.

//Liste des transitions-

→1a2,
1b2,
1c3,
2a3,
!

-On retourne au début du listing des transitions correspondant à notre état

//Liste des transitions-

1a2,
1b2,
1c3,
2a3,
!

-Pour le nombre de transitions trouvées on inscrit les couples encadrés en vert dans un tableau temporaire.

Imaginons maintenant que nous n'ayons pas eu un automate déterministe, alors on aurait eu la séquence suivante :

//Liste des transitions-

1a2,
1a3,
1b2,
1c3,
2a3,
!

Alors là il faut faire l'allocation pour a de deux transitions alors on utilise le même procédé, on compte combien de transitions à le caractère 'a' dans notre table temporaire puis on retourne au début.

//Liste des transitions-

1a2,
1b2,
1c3,
→2a3,
!

-Ensuite on enregistre nos transitions temporaires dans notre table de vérité (AF). Et on va à l'état suivant, pour lequel on refera toutes les étapes depuis le début.

Imaginons maintenant que nous soyons directement sur le point d'exclamation, alors ceci veut dire que nous avons fini ou alors que notre état n'a pas de transitions associées.

Nous voici donc enfin avec notre AF, enregistrer dans notre programme. Pour ce qui est de son affichage, nous avons décidé d'utiliser un tableur excel, pour plus de confort afin de visionner la table.

Si excel n'est pas installé, pas d'inquiétude on peut toujours le lire dans le bloc note où les séparateurs des tables sont des ';'.

Déterminiser

Maintenant que nous avons notre AF, il faut tout d'abord vérifier s'il est déterministe et le déterminer si nécessaire.

Vérifier s'il est déterministe

Rien de plus simple ! Un automate n'est pas déterministe pour une des deux raisons suivantes :

- Soit il a plusieurs états initiaux
- Soit il a une transition qui va vers deux états différents.

PLUSIEURS ETATS INITIAUX

Il nous suffit donc de compter le nombre d'états initiaux de notre automate et si c'est supérieur à 1 alors il n'est pas déterministe.

S'il n'y en a qu'un il va falloir faire l'étape suivante...

UNE TRANSITION VA VERS DEUX ETATS

Rappelons notre structure transition :

Nom du caractere

Nombre d'états associés

Tableau contenant le nom des états associés
--

Si notre champ : nombre d'états associés est supérieur à 1 alors il n'est pas déterministe.

Donc il nous suffira de parcourir chaque transition de chaque état jusqu'à en trouver une dont le nombre d'état est supérieur à 1.

Si on a atteint la fin de la table de vérité de l'AF sans trouver une transition partant vers plusieurs états, alors l'automate est déterministe.

Le déterminer

Avant de commencer, nous tenions à nous excuser d'avance sur la logique horrible que nous avons utilisée... En effet, lorsqu'on détermine un automate on ne sait pas combien d'états nous allons avoir au final, c'est pourquoi nous avons, tenez-vous bien, utilisé un tableau statique et stocké sa taille utile dans une autre variable.

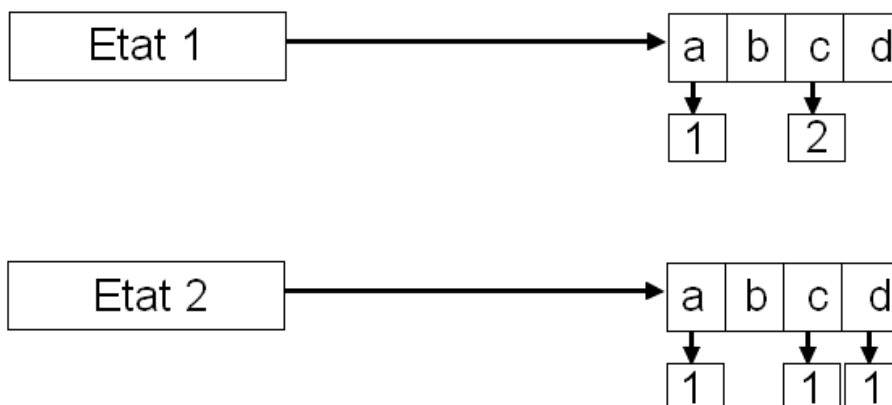
HORRIBLE ! Toutefois, lorsque le projet a été commencé, nous n'avions pas encore connaissance des miraculeux conteneurs en C++ comme il a été dit dans l'introduction et du coup nous avons été obligés de passer par là... Désolé !

Enfin bref, revenons-en à nos automates.

CREER LE NOUVEL ETAT INITIAL

En soit, le concept est assez simple, on prend tous les états initiaux existants puis on scinde les noms et les transitions dans notre nouvelle table.

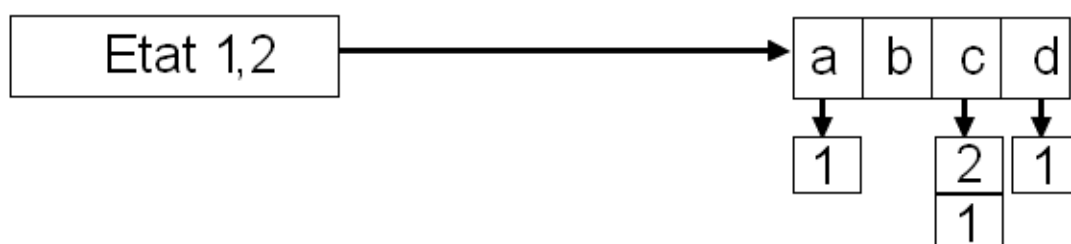
Mais imaginons le cas suivant pour bien comprendre la démarche, soit les états 1 et 2 initiaux :



On va tout d'abord créer notre nouvel état qui portera le nom '1,2' (c'est-à-dire que le tableau de nom de notre état aura 2 champs et la taille utile du tableau de nom vaudra 2).

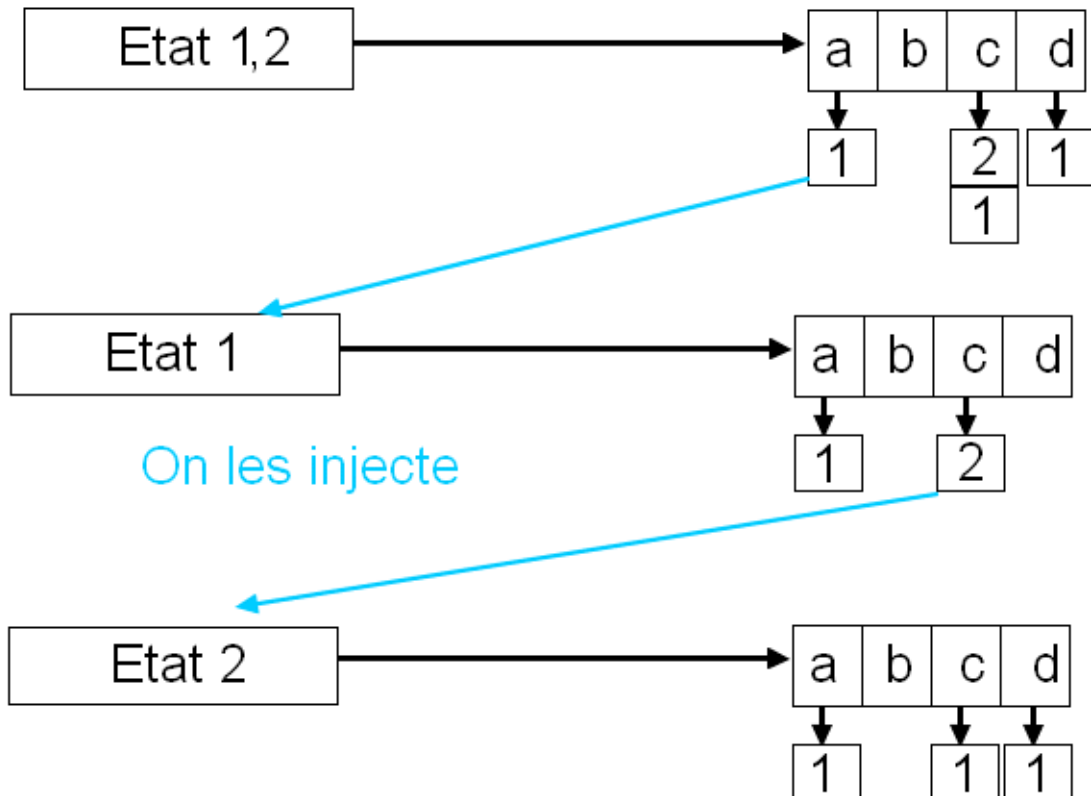
Et on scinde toutes les transitions. Mais regardez la transition 'a' pour les deux états, ils amènent au même état, alors il faudra bien faire attention à ne pas retrouver deux fois un même état cible dans notre transition, ainsi il faudra toujours comparer tous les états cibles d'une même transition. Si deux états cibles correspondent, on en élimine un.

Voici notre nouvel état initial.



Nous voilà donc avec notre nouvel état initial avec deux choses à noter :

-Même si l'état '1' de la transition 'a' existait déjà dans notre AF, dans notre nouvel AD, celui-ci n'existe pas. Pour le vérifier, on compare son nom à tous les états composant notre AD et si on n'a pas trouvé de correspondance alors on l'injecte (et si 'a' aurait mené vers un état composé de plusieurs états alors on aurait réalisé la même mécanique en scindant tous les états de l'AF, en un seul et unique dans notre AD).



-Et ceci nous amène donc à la transition 'c' de notre état '1,2', celle-ci mène vers l'état '2,1', qui est semblable à '1,2'. C'est pourquoi nous devons donc, pour vérifier si cet état-cible est déjà injecté dans notre AD, comparer tous les champs qui composent le nom de l'état-cible et des différents états déjà présent dans notre AD.

Rappelons leur structure :

TRANSITION

Nom du caractere

Nombre d'états associés

Tableau contenant le nom des états associés

ETAT

Tableau contenant le nom de l'état

Le nombre de "noms" qui composent l'état

Indicateur pour dire qu'il est initial

Indicateur pour dire qu'il est terminal

Tableau de transitions

Ici, on commencera par comparer Nombre d'états associés ET Le nombre de noms qui composent l'état. S'ils ne font pas la même taille alors les états ne peuvent pas être les mêmes. Mais s'ils font la même taille alors il faudra comparer chaque cellule du tableau contenant nom des états associés au tableau contenant le nom de l'état.

Si ça correspond on ne l'injecte pas et passe à la transition suivante pour insérer l'état-cible.

Compléter

Maintenant que notre automate est déterministe, nous allons le compléter si nécessaire.

Vérifier s'il est complet

Pour commencer il faut vérifier s'il est complet, pour cela la mécanique est simple : on parcourt toutes les transitions de notre AD et si un des champs « nombre d'états associés » dans notre structure Transition est nul, alors c'est que la transition ne mène vers rien et qu'il faut rajouter une poubelle.

Le compléter

Ici, l'utilisation d'un tableau statique avec une taille utile devient intéressant au final, puisque nous devons rajouter une poubelle.

L'utilisation d'un conteneur c++ aurait été certes plus adéquate mais comme il a été dit plus haut, nous n'étions pas encore au courant de telles bibliothèques.

Ainsi, une allocation dynamique aurait été compliquée car il aurait fallu supprimer la table créée et faire une nouvelle allocation contenant un espace mémoire supplémentaire afin de stocker la poubelle.

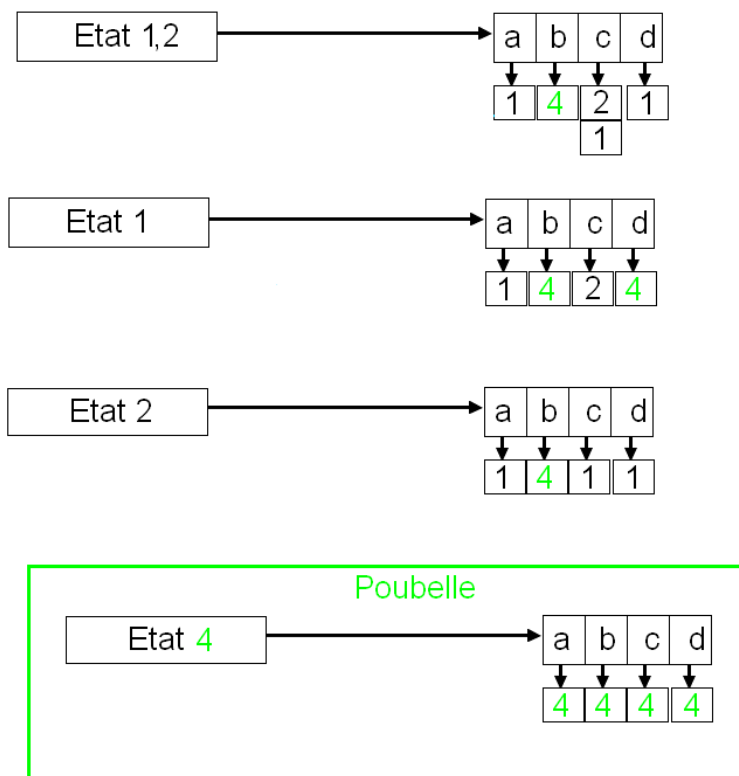
Alors voici ce que nous faisons, on commence par rajouter l'état poubelle qui porte le numéro égale à la `taille_utile + 1`.

Aussi, on complète toutes ses transitions pour que l'état poubelle aille vers lui-même.

Par exemple si nous avons 4 états dans notre AD, alors la poubelle portera le numéro 5.

Et ensuite, on re-parcourt toutes les transitions de chaque état et complétons les trous en mettant le champ 'nombre d'état cible' = 1 et en rajoutant le nom de la poubelle dans le tableau qui contient le nom de l'état cible.

Si on reprend l'exemple d'AD plus haut, voici ce que ça donnerai :



Minimiser

La minimisation a été l'étape la plus difficile à mettre en œuvre car nous avons du mal à trouver l'algorithme à adopter. En effet, sur papier, ceci est très visuel et instinctif tandis qu'en langage machine il aura fallu poser une réelle méthode afin de réussir cette fonction.

Pour effectuer celle-ci, nous allons avoir besoin des structures suivantes.

La table de corrélation

Pour commencer rappelons ce qu'est une table de corrélation. Il s'agit d'une table qui va contenir la séparation en « groupes » de nos états.

LA STRUCTURE GROUPE

Nom du groupe
Nombre d'états qui composent le groupe
Tableau de pointeur vers ces états

LA STRUCTURE TAB_CORR

Voici la structure qui va réellement nous intéresser, comme il a été dit plus haut le but est de stocker les différents groupes que nous allons pouvoir créer. Ainsi la table de corrélation contiendra la structure précédente et une variable contenant le nombre de groupes créés.

Nombre de groupes
Tableau de groupes

Maintenant que nous avons les bonnes structures, nous allons énumérer les étapes de la minimisation.

SEPARER LES ETATS TERMINAUX ET NON-TERMINAUX

Comme vous le savez, la minimisation consiste à créer des « groupes » d'états qui auront les mêmes caractéristiques.

Or, la première étape primordiale consiste à séparer les états initiaux et terminaux.

Ainsi la méthode est la suivante :

-On commence par compter combien de terminaux et non-terminaux nous avons.

-On initialise notre tab_corr :

1 - On met le nombre de groupes à 2.

2 – On fait l'allocation de notre tableau de groupe pour qu'il y ait autant de groupes que d'états dans notre ADC (en effet, si l'automate est déjà minimal alors on aura autant de groupes que d'états à la fin)

3-Et on crée deux groupes avec nos Terminaux dans l'un et nos non-terminaux dans l'autre.

LA MINIMISATION EN ELLE-MEME

Pour commencer, nous allons créer un ADCM temporaire dans lequel on va retranscrire notre ADC sauf qu'au lieu de mettre le nom de nos états bruts, on va le réécrire avec le nom du groupe auquel chaque état appartient. Aussi, nous allons évidemment remplacer chaque transition par le groupe auquel appartient l'état cible.

Considérons la table suivante donc on perçoit les deux groupes 100/101:

Etats	Transitions		
	a	b	c
100	100	101	101
100	100	101	101
100	101	100	100
101	101	101	101

Etats	Transitions		
	a	b	c
100	100	101	101
→100	100	101	101
→100	101	100	100
→101	101	101	101

correspond on compare les transitions.

A partir de là, nous allons créer une nouvelle table de corrélation SANS ECRASER L'ANCIENNE selon la méthode suivante.

1 - On commence par comparer le nom du premier état à deux des autres, dès que ça

Etats	Transitions		
	a	b	c
100	100	101	101
100	100	101	101
100	101	100	100
101	101	101	101

2 - Si toutes les transitions correspondent, on enregistre l'indice de l'état correspondant et on incrémente un compteur qui comptera combien d'états composent le groupes

Ici, il n'y aura que ces deux états qui seront dans le même groupe, ainsi, à partir de là on crée le nouveau groupe composé de ces deux états dans notre tab_corr. Et on passe à l'état suivant.

Etats	Transitions		
	a	b	c
100	100	101	101
100	100	101	101
→100	101	100	100
→101	101	101	101

3 - Maintenant on compare l'état suivant aux états en dessous de lui puisqu'en théorie tous les états au-dessus sont déjà rangés

Mais... Attendez ! Cet état a déjà été rangé ! Voilà pourquoi nous avons enregistré son indice lors de l'étape 2, comme ça nous savons que celui-ci appartient déjà à un groupe et on passe direct au suivant.

On fait ces opérations ainsi de suite pour ensuite avoir fini de créer tous nos nouveau groupes.

Mais alors là, comment sait-on que nous avons fini de minimiser ?

Souvenez-vous ! Nous avons bien insisté que ces nouveau groupes ont été inscrits dans une nouvelle table de corrélation et que l'ancienne ne devait pas être écrasée !

Ainsi, il nous suffit de comparer le nombre de groupes de l'ancienne au nombre de groupes de la nouvelle !

-S'il y a autant de groupes on a fini de minimiser

-Sinon, on re-minimise notre nouvelle table avec nos nouveaux groupes.

Lire des mots

Pour reconnaître un mot, nous devons pouvoir utiliser un des trois automates suivants :

- Le Déterministe
- Le Déterministe complet
- Le Déterministe complet et minimal

Or, le point commun de ces trois automates est qu'ils ne contiennent qu'un seul et unique état initial. Par ailleurs, seul le déterministe aura potentiellement des transitions sans cible alors il ne faudra pas oublier de considérer ce cas-là.

Alors, la méthode suivra les règles suivantes :

- 1 – On recherche l'indice de l'état initial.
- 2 – On demande à l'utilisateur de rentrer le mot qu'il veut reconnaître.
 - Si le mot contient un caractère qui ne fait pas partie de l'alphabet, on lui demande de retaper un mot.
 - Si le premier caractère est '*' qui représente le mot vide, alors on regarde directement si l'état initial est aussi terminal.
- 3 – A partir de là on récupère caractère par caractère le mot et on avance dans l'automate
 - Dans l'état courant, si la transition correspondant au caractère n'a pas d'état cible, on stoppe et on dit que l'automate ne reconnaît pas le mot.
 - Et si elle a un état cible on va à l'état cible et on passe à la prochaine lettre.
- 4 – Une fois qu'on a fini de lire tous les caractères, il nous suffit de vérifier si l'état courant est terminal ou non !

Conclusion

Ce projet a réellement permis de faire un lien entre informatique et la logique propre à l'homme. Là où sur papier la résolution d'automates nous semblait plutôt instinctive et visuelle, il a fallu mettre en œuvre une réelle logique algorithmique afin de répondre à la problématique du TAI. Là où le nom de la matière « mathématiques pour l'informatique » semblait flou, ce terme prend tout son sens après ce projet.