UVSQ université PARIS-SACLAY

ISTY
Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Rapport du projet Compilation

Implémenter les outils de bases d'un analyseur lexical

Année scolaire 2020-2021

Réalisation:
DANG Duy
DRISSI Ali

Sommaire

Introduction:

Le but de ce projet est d'implémenter les outils de bases d'un analyseur lexical en langage C. Le but du projet est de nous faire travailler sur des automates. Nous devons implémenter une structure de données efficaces pour pouvoir travailler dessus. Par la suite, le projet est constitué de deux parties : une sur les automates finis non-déterministes où l'on fait des automates simples et des fonctions basiques sur deux automates, et une autre sur les automates finis déterministes où l'on reconnaît un mot, on détermine un automates non déterministe et la minimisation d'un automate.

Dans un souci de simplicité de test du code, nous avons pré-programmer des automates pour faire fonctionner nos fonctions dans le main.

Ce qui est attendue du projet

I - Faire une structure pour gérer un automate

Avoir une réflexion sur la structure qui sera essentielle pour la suite du projet.

II - Automates finis non-déterministes:

1. Construire un automate fini non-déterministe reconnaissant le langage vide.
2. Construire un automate fini non-déterministe reconnaissant le langage composé du seul mode vide.
3. Construire un automate fini non-déterministe reconnaissant le langage composé d'un mot d'un caractère passé en paramètre.
4. Implémenter une fonction qui renvoie un automate standard qui reconnaît la réunion des langages de deux automates passés en paramètre.
5. Implémenter une fonction qui renvoie un automate standard qui reconnaît la concaténation des langages de deux automates passés en paramètre.
6. Implémenter une fonction qui renvoie un automate standard qui reconnaît la fermeture itérative de Kleene du langage d'un automate passé en paramètre.

III - Automates finis déterministes :

1. Implémenter une fonction qui prend en paramètre un automate fini déterministe et un mot (chaîne de caractères) et qui exécute cet automate sur ce mot.
2. Implémenter une fonction qui déterminise un automate fini non-déterministe passé en paramètre.
3. Implémenter une fonction qui minimise un automate fini déterministe passé en paramètre.

Implémentation:

Avec une explication très brève, dans l'idée de construire une structure d'automate, nous avons eu l'idée au départ des tableaux dynamiques puis d'utiliser des listes chaînées. Mais on a fini par choisir des tableaux statiques dans un souci de temps. Vu que nous avons choisi de faire une structure de structure, cela nous semblait déjà bien assez compliqué.

Alors nous avons essayé de simplifier le travail, nous sommes conscients que ce n'est pas le plus efficace. Mais nous avons quand même essayé d'être un petit peu efficaces dans l'utilisation de nos tableaux, en renvoyant exactement la bonne taille de tableau à chaque fois, c'est pourquoi on l'est retrouve dans la structure.

Structure utilisé:

Nous utilisons un structure AFND (Automate Fini non déterministe) et AFD (Automate fini déterministe) qui est composé de:

- Un tableau d'alphabet
- Taille du alphabet
- Tableau d'état initial
- Tableau d'état
- La taille du tableau

La structure tableau d'état est constitué de:

- Le numéro de l'état
- Un booléen si l'état est accepteur
- Un booléen si l'état est final
- Un tableau de transition
- La taille de la transition

La structure de tableau de transition est constitué de:

- L'état de départ
- Le caractère de la transition
- L'état d'arrivée

Les structures TabTrans et TabTrans2 sont des structures pour aider à la minimisation.

Explication de notre structure

Chaque état possède ses transitions. Comme cela, nous retrouvons plus rapidement les transitions liées à son état. Il sera toujours facile pour nous plus tard de récupérer un tableau de toutes les transitions si jamais nous en avons besoin.
L'indice du tableau de l'état est le numéro l'état, nous avons donc un doublon qui nous permet d'accéder plus facilement à cette valeur.
De même pour la transition, le numéro de la transition est l'indice du tableau.
Avec une telle structure, cela nous permet de mettre un automate dans une variable afin de les manipuler facilement.

Langage vide:

Elle renvoie l'automate du langage vide. C'est-à-dire, avec un état initial, sans transition et sans état accepteur.

Mot vide:

Elle renvoie l'automate du mot vide. C'est-à-dire, un automate sans transition avec un état accepteur et final.

Reunion d'automate:

Cette fonction permet de réunir deux automates. C'est-à-dire, que notre automate qu'on va retourner peut reconnaître les deux automates proposés en paramètre. Dans notre cas, le deuxième automate est soustrait de son état initial et ses transitions sont remises à l'état initial du premier automate.

La Concaténation:

Elle prend en entrée deux automates, l'automate 1 est concaténé au l'automate 2 dans cette ordre la.
Le premier automate doit lui retirer sont état final pour lui ajouter les transitions de l'état initial de l'automate 2. Dans l'automate 2, on doit retirer l'état initial. On fait faire attention au transition sur elle-même dans l'automate 2, car dans ce cas là, il y aura des transitions supplémentaires.

La mise en étoile:

Cette fonction est appelée par la méthode appelée Fermeture Itérative de Kleene. Elle prend en entrée un automate fini non-déterministe. On boucle que sur les états accepteurs, on récupère leurs numéros et le caractère de la transition avec l'état initial, et on crée des transitions vers les autres états accepteurs puis on l'ajoute dans la liste des transitions.

Déterminisation:

Malheureusement, nous n'avons pas réussi à faire cette fonction. Mais nous allons quand même expliquer ce que nous avons voulu faire en détail.
Tout d'abord, nous avons voulu créer un tableau de 2 dimensions avec les éléments de l'alphabet de l'automate en lignes, et les groupes d'états en colonnes. (La première colonne est le groupe d'état avec l'état initial seulement). Puis, pour chaque case, on ajoute au groupe d'état qui correspond à la colonne l'état d'arrivée de chaque transition qui a comme départ chaque état du groupe d'état correspondant. On refait cette action jusqu'à ce qu'on ait plus de nouveaux groupes d'états. Après, on crée notre nouvel automate avec les groupes d'états de chaque colonne comme état, et on crée des nouvelles transitions avec le départ qui est le groupe d'état qui correspond à la colonne, l'arrivée est celui dans une case du tableau et le caractère est celui qui correspond à la ligne où se situe cette case. Tous les groupes d'états qui ont un état accepteur sont aussi accepteurs.

La minimisation:

Etant donné qu'on travaille énormément sur les transitions. Nous avons créé une fonction permettant de récupérer un tableau de transition à deux dimensions.
L'idée est de séparer les états en différent groupe. Les états qui ont des transitions vers le même groupe et ceux qui ont le même comportement se retrouvent ensemble.

Ici, dans un réel souci de bien simplifier la compréhension du code.

Nous avons construit deux tableaux, un en une dimension et l'autre à deux dimensions: Dans le premier tableau, les indices sont les états et leurs valeurs sont le groupe auquel il appartient.
Dans le deuxième tableau, les indices sont l'état de départ + le caractère de la transition et elle retourne le groupe.
L'idée est de faire plusieurs fois le processus, avec un nombre de groupes différent au départ. Ce qui permet à la fin de bien séparer tous les comportements différents.

A la fin de la fonction, nous créons l'automate que nous retournons.

Exécution d'un automate

Cette partie est gérée par la fonction execution_afd. AJOUTE TES TRUCS

Fonction supplémentaire:

set_etat_AFND et set_etat_AFD

Ce sont des fonctions qui permettent de set un état d'un automate fini non déterministe et déterministe.

set_transition_AFND et set_transition_AFD

Ce sont des fonctions qui permettent de set une transition d'un automate fini non déterministe et déterministe

Find_etat

Cette fonction permet de rendre l'indice du tableau de l'état.

getTransition

Permet de récupérer la structure transition de la lettre.

Toutes les fonctions "afficher" permettent d'afficher dans le terminal.
Toutes les fonctions "init" permettent d'initialiser les tableaux.

Les implémentation supplémentaire:

Afin d’avoir un vrai analyseur lexical, nous aurions dû rajouter la possibilité de mots clés qui permettra de retourner ce qui est recherché.