ENSIIE

Promotion 2024

Rapport Projet IPI Semestre 1

Valentin Buchon

Année 2021-2022

Table des matières

Voici le rapport du projet de semestre 1 dans l'UE IPI par Valentin Buchon

Partie I

Introduction

Sommaire

1 Présentation du sujet

a) Objectif

Le **sujet** du projet a pour objectif d'implémenter un programme en C qui exécute et utilise des automates LR(1) dans l'optique de reconnaître des langages.

b) Utilisation

L'utilisateur devra passer en paramètre de l'exécutable un fichier .aut qui contiendra la description d'un automate dans un format que nous détaillerons plus tard. Après cela, l'utilisateur pourra donner sur l'entrée standard la ligne qu'il veut vérifier. Ainsi le programme affichera `Accepted` ou `Rejected` si le langage proposé est reconnu par l'automate.

c) Description de l'automate

Un automate LR1 est donné par les éléments suivants :

- Un ensemble fini d'états
- Un état initial
- Un alphabet d'entrée
- Un autre alphabet de symboles dits non-terminaux
- Une fonction action qui à chaque état et chaque lettre associe une action : cette action peut être soit Rejette, soit Accepte, soit Décale, soit Réduit
- Une fonction (partielle) décale qui à un état et une lettre associe un état ; cette fonction n'a besoin d'être définie que quand l'action associée à l'état et la lettre est Décale
- une fonction (partielle) réduit qui à un état associe un entier et un symbole non-terminal ; cette fonction n'a besoin d'être définie que quand l'action associée à l'état et la lettre est Réduit
- Une fonction (partielle) branchement qui à un état et un symbole non-terminal associe un état

Un tel automate fonctionne à l'aide d'une pile d'états. Initialement, cette pile contiendra un unique élément, à savoir l'état initial. Au cours de l'exécution, l'état courant sera celui situé au sommet de la pile.

d) Format du fichier

Les états seront représentés par un entier sur un octet. (On aura donc au maximum 256 états.) L'état initial sera l'état 0.

Pour l'alphabet d'entrée mais aussi pour les symboles non-terminaux, on utilisera les caractères dont le code ASCII est compris entre 0 et 127 inclus.

Les actions seront représentées par des entiers : Rejette = 0, Accepte = 1, Décale = 2 et Réduit = 3.

On supposera qu'on ne dépilera jamais plus de 256 états lors d'une réduction, on pourra donc représenter la première composante de réduit(s,c) sur un octet.

Un fichier contenant une description d'un automate LR(1) respectera le format suivant :

- une première ligne contenant a n où n est le nombre d'états de l'automate
- n × 128 octets représentant les valeurs de action(s,c) pour tout état s et toute lettre c (dans l'ordre action(0,0) action(0, 1) action(0, 2) ... action(0, 127) action(1, 0) action(1, 1) ... action(n,127)), le tout suivi d'un retour à la ligne
- n octets représentant la première composante de réduit(s) pour tout état s, le tout suivi d'un retour à la ligne
- n octets représentant la deuxième composante de réduit(s) pour tout état s, le tout suivi d'un retour à la ligne
- Une séquence de groupement de trois octets représentant la fonction partielle décale ; un groupement de trois octets s c s' indique que décale(s,c) = s' ; cette séquence se terminera par le groupement de trois octets '\255' '\255' '\255'
- Une séquence de groupement de trois octets représentant la fonction partielle branchement ; un groupement de trois octets s A s' indique que branchement(s,A) = s' ; cette séquence se terminera par le groupement de trois octets '\255' '\255' '\255'.

Partie II

Conception du projet

Sommaire

1 Lecture et Stockage de l'automate

Toute la lecture de l'automate se fera via la fonction `fgetc` afin de lire caractère par caractère.

a) Lecture de n

La lecture de n est un problème car il peut être écrit sur plusieurs caractères, nous devrons y faire attention afin d'avoir la bonne valeur. Pour cela nous utiliserons la méthode suivante : "1 2 3" sera lu de gauche à droite. Ainsi nous utiliserons une formule n = n×10 + caractère-Lu. Avec n = 0 nous aurons au départ n = 1 puis n = 1×10 + 2 = 12 puis n = 12×10 + 3 = 123

b) Lecture des fonctions

L'accès aux données du fichier est une question à laquelle il a fallu répondre. J'ai ainsi choisi de parcourir tout le fichier dans l'ordre de lecture classique et de stocker chaque valeur dans le tableau lui correspondant : `action`, `reduit1`, `reduit2`, `decale`, `branchement`. A noter que `decale` et `branchement` sont des matrices de taille 256×256. Ainsi avec les 256 états possibles et les 256 caractères, ceux qui seront utiles pourront être retrouvés grâce aux coordonnées. Le stockage des informations, bien que un peu gourmand en mémoire, m'a semblé être plus simple que d'utiliser un curseur dans la lecture du fichier et de le reparcourir à chaque fois.

2 La structure de pile

Comme indiqué dans le sujet, concernant les états nous implémenterons une structure de pile pour les états. Elle permet d'accéder facilement au sommet, de retirer un ou plusieurs éléments ainsi que d'en rajouter.

3 Application de l'algorithme

a) Lecture de l'entrée

Afin de lire l'entrée nous utiliserons la fonction `fgets`, avec un buffeur d'une taille limitée arbitrairement à 256, celle-ci pourra être augmentée s'il le faut. Ici, il fallait bien faire attention d'avoir à la fin \n\0 pour que les automates reconnaissent la fin de la chaîne à tester. Dans le sujet il était indiqué d'utiliser un `fgets` puis un `sscanf` cependant je n'ai pas réussi à appliquer cette méthode. En effet il me manquait le \n dans la chaine de caractères, ainsi j'ai opté pour la solution décrite juste avant en utilisant seulement le `fgets`.

b) Ecart à la forme du sujet

Dans l'énoncé, l'exemple du code montre bien une boucle infini au niveau des entrées. J'ai pris l'initiative de changer un peu la forme et de demander à l'utilisateur si il veut continuer après chaque proposition de langage. Il y a deux raisons à cela, la première est technique, en quittant la boucle `While` infini avec un <`CTRL+C`> on ne quitte pas correctement l'algorithme et ainsi on exécute pas la fin qui consiste justement a libérer la mémoire. La seconde raison est simplement esthétique, je trouve que quitter le programme proprement est plus agréable.

c) Algorithme

L'algorithme est simplement une transcription de celui décrit dans le sujet. J'utilise une variable `cpt` qui me permet de changer de lettre dans le tableau de caractères à tester quand il le faudra. Afin d'appliquer l'algorithme, j'utilise une boucle `while`. La conditions d'arrêt est de tomber sur une valeur d'action qui vaut 0 (`Rejected`) ou 1 (`Accepted`). Par construction, cette action arrivera au moins à la fin du tableau, donc cette boucle sera finie.

d) Les noms des variables

Par commodité, l'ensemble du code est en anglais, sauf les tableaux qui contiennent les valeurs des fonctions de l'automate. J'ai jugé qu'il était plus simple de comprendre le code si la fonction "décale" du sujet était transcrite par la matrice "`decale`" par exemple.

Partie III

Les améliorations

Sommaire

1 Améliorations possibles

a) Gestion des erreurs

Ne sachant pas comment gérer les erreurs dans un projet un peu plus professionnel, j'ai garder l'habitude de mettre des `exit(1)`. Une plus grande diversité et description des erreurs pourrait être utile à corriger le code.

b) Makefile

Je pense que le `Makefile` pourrait être amélioré, même s'il fonctionne. Il n'a surement pas une forme conventionnelle et il ne supprime pas automatiquement les fichiers en `*.o`

c) Bonus

Par manque de temps et de compréhension de la bibliothèque `DOT`, je n'ai pas pu travailler sur l'aspect graphique de l'automate, cela pourrait être fait plus tard.

d) Les commentaires

J'ai eu quelques difficultés à savoir quels commentaires mettre et à quel emplacement. De plus, il est difficile d'être concis et précis à la fois. En effet certaines lignes qui me semblent claires peuvent ne pas l'être pour d'autres, et inversement.

e) Notes supplémentaires

Mon projet fonctionne apriori correctement sur les 4 exemples proposés. Cependant je n'ai pu le tester dans des conditions particulières. Par exemple, un automate qui peut contenir plus de 173 caractères, peut-être un soucis si je n'arrive pas à bien faire la distinction entre les trois '\255' qui séparent `decale` et `branchement` dans le fichier et l'état 173 (car le numéro ascii de \255 est 173).
Il aurait peut être été préférable de réduire la quantité de code dans le `main` et de faire un autre fichier pour contenir toutes les lignes qui servent à la lecture et au stockage de l'automate. Ici ce n'est pas encore trop gênant car la taille `main` reste raisonnable, c'est pour cela que je ne l'ai pas fait.