Rapport du projet de programmation impérative

Yoann Boulinguez-Bonnet ⋆ ENSIIE

9 janvier 2022

Résumé

Le but de ce projet est d'implémenter un programme qui exécute des automates LR(1). Ces derniers servent à reconnaître des langages de programmation. Le programme va lire un fichier (dont le nom sera passé en paramètre de l'exécutable) qui contiendra une description de l'automate. Il va ensuite lire des lignes sur l'entrée standard. Après chaque ligne lue, il indiquera si la ligne est correcte pour le langage correspondant à l'automate, auquel cas il affichera `Accepted`, sinon il affichera `Rejected`.

Table des matières

Compilation

- Pour extraire le projet, il suffit de taper la commande `tar zxvf yoann_boulinguez.tgz` dans un terminal ;
- Pour lancer le projet, taper, dans un premier temps, `make` dans le terminal ;
- Ensuite, taper `./automaton filename.aut` où `filename` est le nom du fichier contenant une description d'automate.

1. Implémentation des matrices

Pour implémenter les matrices, j'ai choisi d'utiliser une structure contenant le nombre de colonnes (`nc`), le nombre de lignes (`nr`) ainsi qu'un pointeur vers chacune des valeurs de la matrice.

```
typedef struct {
    int rows;
    int columns;
    int** cells;
} matrix;
```

Trois fonctions sont ensuite implémentées : la *création* de matrices (à travers `createMatrix`), l'*initialisation* avec des zéros (à travers `matrixZeros`) et l'*affichage* des matrices (à travers `printMatrix`).

Cette dernière fonction a été utile durant la phase d'implémentation du projet pour afficher les valeurs stockées dans les différentes matrices et ainsi détecter d'éventuelles erreurs.

2. Implémentation des piles

Pour implémenter les piles, on utilise deux structures. Une première qui contient les éléments de la pile, définie à la manière des listes chaînées : la valeur de l'élément, et un pointeur vers l'élément suivant.

En plus de cela, une autre structure a été définie, une *structure de contrôle*, qui contient un pointeur vers le premier élément de la pile.

```
typedef int value;

typedef struct Element Element;
struct Element{
    value num;
    Element* next;
};

typedef struct stack stack;
struct stack{
    Element* first;
};
```

Diverses fonctions ont également été implémentées :

- une fonction qui teste si la pile est *vide* : `isEmpty` ;
- une fonction qui *crée* une pile vide : `emptyStack` ;
- une fonction qui *empile* une valeur, *i.e.* ajoute une valeur au sommet de la pile : `push` ;
- une fonction qui *dépile* une valeur, *i.e.* supprime la valeur au sommet de la pile, sans la retourner ;
- une fonction d'*affichage* de la pile : `printStack`.

Cette implémentation des piles est reprise d'un cours en ligne sur la plateforme *OpenClassRoom* et prend toute son utilité dans la mesure où l'on accède facilement au premier élément de la pile.

3. Démarche générale

3.1. Lecture des fichiers

On déclare un pointeur vers un fichier que l'on nomme `automate`. On peut ensuite effectuer toutes les actions nécessaires sur ce fichier : la première est l'ouverture du fichier en *lecture* et en mode *binaire* car les fichiers sont écrits dans ce mode, comme le précise le sujet.

Ensuite, on utilise différentes fonctions pour lire lignes et caractères :

- `fgets` permet de lire la première ligne et ainsi de récupérer le nombre d'états en utilisant par la suite un `atoi` ;
- `fgetc` s'utilise pour récupérer des caractères un par un pour remplir la matrice `action` et les listes correspondant aux première et deuxième composantes de `reduit` ;
- `fread` permet de lire des séquences de trois octets pour remplir `decale` et `branchement`.

3.2. Récupération des entrées de l'utilisateur

Pour récupérer les entrées de l'utilisateur, j'ai utilisé les conseils donnés dans le sujet, à savoir l'utilisation d'un `fgets` suivi d'un `sscanf`.

L'étape suivante est la création - puis l'initialisation avec des zéros - d'une pile, qui contiendra les états, avec l'état courant qui se trouve au sommet.

Ensuite, nous nous intéressons au texte entré par l'utilisateur. On parcourt la chaîne de caractère en commençant par le premier caractère avec une boucle `while`, qui se terminera car on passera obligatoirement par, au pire, tous les états.

On stocke l'état actuel de la pile en prenant son sommet ; ensuite on effectue un comportement différent en fonction de la valeur de `actions` pour l'état actuel et le caractère lu.

On initialise une variable de vérification `check` à `0`. Dans les cas *acceptés* et *rejetés*, on change la valeur de cette variable pour lui affecter la valeur `1`. Cette variable est intégrée à la boucle `while` pour résoudre un soucis d'acceptation [1].

On parcourt ensuite la matrice des actions en fonction de l'état actuel et du caractère courant. Si la valeur est `0`, on affiche `Rejected`, si la valeur est `1`, on affiche `Accepted`. Dans ces deux derniers cas, on change la valeur de `check` en `1` et comme ceci la boucle s'arrête dans tous les cas.

Dans le cas où l'action est égale à `2`, on réalise bien l'empilement de `decale(s,c)` et on passe à la lettre suivante.

Dans le cas où l'action est égale à `3`, on stocke la première et la deuxième composante de `reduit`, puis on dépile n fois où n est la première composante de `reduit`. On obtient ensuite un nouvel état courant, et on empile la valeur de `branchement`.

4. Problèmes rencontrés et solutions trouvées

Le premier problème rencontré était la lecture du nombre d'état : je n'avais pas bien compris comment récupérer le nombre d'états ; ainsi, pour les automates `dyck`, `word` et `word_bis`, il n'y avait pas de problèmes car le nombre d'états était plus petit que 10 mais cela n'était plus valable pour `arith`. J'ai ensuite compris qu'il fallait utiliser la commande `atoi`, ce qui m'a servi pour résoudre ce soucis et pour mieux comprendre

1. voir section suivante

le format des fichiers.

J'ai eu un soucis d'implémentation de matrices qui était en fait une erreur d'inattention : dans ma boucle `for` pour initialiser les colonnes, je parcourais les colonnes au lieu de parcourir les lignes et j'avais ainsi une erreur de type `corrupted size vs prev_size \\ Aborted`. Ce problème apparaissait à la compilation et disparaissait quand je commentais la fermeture du fichier ; une fois l'implémentation des matrices corrigée, ce problème a disparu mais je ne comprends pas pourquoi le fait de ne pas écrire `fclose` à la fin résolvait le problème.

Le dernier problème rencontré était lors de la dernière étape du projet, lorsque l'on doit *switcher* selon la valeur de la matrice `actions`. En effet, le programme rejettait bien les mots faux pour chaque automate, mais, pour les mots qui devaient être acceptés, le message `Accepted` n'était jamais affiché. Une solution pour résoudre ce problème a été de mettre en place la variable `check` et de l'intégrer dans la boucle `while`, mais ceci ne m'apparaît pas optimal.