UNIVERSITÉ DE NANTES

Rapport de projet
Programmation fonctionnnelle (MP:X32I020))

Réalisé par

Mamadou Saliou DIALLO (685L, E18764T)

Ibrahima DIALLO (685L, E15E632Q)

May 27, 2020

Contents

1 Introduction

Dans le cadre de la réalisation du projet du cours programmation fonctionnelle, nous avons réalisé un outil de conception et d'exécution d'automate à pile.

Le choix de l'implémentation des automates à piles vient du faite qu'on les a vu en cours d'Informatique fondamentale 2 .
Nous avons donc choisi ce modèle de calcul car nous avons traités les TD de ce cours à distance pendant le confinement et c'est donc une occasion pour nous de renforcer notre connaissance en nous appuyant d'exemples précis et de pouvoir les programmer afin de mieux comprendre le fonctionnement et mieux les représenter dans notre tête.

2 Presentation

Comme annoncé dans l'introduction, le but de ce projet est de réaliser un outil de conception et d'exécution d'automates à pile.
L'outil réalisé permet de créer ses propres automates et de faire des tests de validité de mots sur ces automates. Lors des opérations de tests, on peut suivre les différents étapes de calcul dans le terminal avec un affichage bien soigné.

2.1 C'est quoi un automate à pile ?

Un automate à pile est une machine abstraite utilisée en informatique théorique et, plus précisément, en théorie des automates.

Un automate à pile est une généralisation des automates finis : il dispose en plus d'une mémoire infinie organisée en pile (last-in/first-out ou LIFO). Un automate à pile prend en entrée un mot et réalise une série de transitions. Il fait une transition pour chaque lettre du mot, dépendant de la lettre, de l'état de l'automate et du sommet de la pile, et peut aussi changer le contenu de la pile. Selon l'état de l'automate et de la pile à la fin du calcul, le mot est accepté ou refusé.

Les langages acceptés par les automates à piles sont exactement les langages algébriques, c'est-à-dire ceux engendrés par une grammaire algébrique.[1]

2.2 Definition formelle d'un automate à) pile

[2]Un automate à pile (non déterministe) est un 7-uplet $\mathcal{A} = (Q, A, Z, \delta, z_0, q_0, T)$ où :

- Q est l'ensemble d'états
- A est l'alphabet d'entrée
- Z est l'alphabet de pile
- $\delta : Q \times (A \cup \{\varepsilon\}) \times Z \to \mathcal{P}(Q \times Z^*)$ est la fonction de transition (la notation P désigne l'ensemble des parties)
- $z_0 \in$ Z est le symbole de fond de pile
- $q_0 \in$ Q est l'état initial
- T$\subset$ Q est l'ensemble des états terminaux

Au lieu de la fonction δ , on rencontre fréquemment l'ensemble $\Delta \subset Q \times (A \cup \{\varepsilon\}) \times Z \times Q \times Z^*$ défini par:

$$(q, y, z, p, h) \in \Delta \iff (p, h) \in \delta(q, y, z)$$

Les éléments de Δ sont les règles de transitions. Lorsque $y = \varepsilon$, on parle d'une ε-règle. Tous les ensembles dans la définition sont finis.

Une configuration interne de l'automate est un couple $(q, h) \in Q \times Z^*$. Une configuration de l'automate est un triplet $(q, w, h) \in Q \times A^* \times Z^*$. Un calcul de l'automate sur un mot $w \in A^*$ est une suite de transitions à partir de la configuration initiale (q_0, w, z_0). Il y a transition de la configuration (q, yw, zh), où $y \in A \cup \varepsilon$ et $z \in Z$ vers la configuration $(q', w, h'h)$ et on écrit :

$$(q, yw, zh) \vdash (q', w, h'h)$$

lorsque $(q', h') \in \delta(q, y, z)$. Lorsque $y = \varepsilon$, le mot d'entrée ne change pas. On parle alors d'une ε-transition ou d'une transition spontanée. Dans tous les cas, pour qu'une transition soit possible, la pile ne doit pas être vide.

On dit qu'un mot est accepté par l'automate s'il existe une série de transitions qui conduit à une configuration acceptante. Plusieurs modes de reconnaissance existent :

- Reconnaissance par pile vide. Les configurations acceptantes sont les configurations de la forme $(q, \varepsilon, \varepsilon)$ où $q \in Q$ est un état quelconque. Autrement dit, il est possible d'arriver à vider entièrement la pile au moment où on termine la lecture du mot.
- Reconnaissance par état final. Les configurations acceptantes sont les configurations de la forme (t, ε, h) où $t \in T$ est un état final. La pile n'est donc pas nécessairement vide à la fin de la lecture du mot.
- Reconnaissance par pile vide et état final. Les configurations acceptantes sont les configurations de la forme $(t, \varepsilon, \varepsilon)$ où $t \in T$ est un état final. La pile est vide à la fin de la lecture du mot.

Le langage reconnu par l'automate $\mathcal{A}$ est l'ensemble des mots de A^* qui sont acceptés.
Les trois modes d'acceptation sont équivalents, au sens que si un langage est reconnu par un automate à pile d'une espèce, on peut construire un automate reconnaissant ce langage, des autres espèces.

3 Les principaux types de données

Afin d'implémenter un outil permet de créer des automates à piles et de vérifier des mots, Nous avons défini les types de données suivants:

3.1 Le type etat

Ce type désigne la nature d'un état de l'automate.

$$type\ type\_etat = Initial|Final|Intermediaire|InitialFinal;;$$

- **Initial**: désigne un état initial
- **Final**: désigne un état final
- **Intermediaire**: désigne un état intermediaire (ni initial, ni final)
- **InitialFinal**: désigne un état à la fois initial et final

3.2 Le type regle_transition

Nous utilisons ce type pour définir le type transition, il désigne la règle de transition vers un autre etat.
Ce type nous permet de connaître la règle de transition à savoir s'il faut empiler ou non pour cette transition ainsi que la valeur à empiler (où à dépiler).

$$type\ regle\_transition = \{empiler : bool; valeur : char\};;$$

- **empiler**: est à true s'il faut effectuer un empilement, et à false lorsqu'il s'agit de dépilement.
- **valeur**: désigne le caractère devant être empiler ou dépiler de la pile de l'automate.

3.3 Le type transition

Ce type est utilisé pour désigner la transition entre deux etats de l'automate. Ce type contient la valeur qui sera lue sur la transition, l'état de destination (le prochain état de l'automate après la transition) ainsi que la règle associée à cette transition (empilement ou dépilement).

$$type\ transition = \{destination : etat;\ valeur : char;\ regle : regle\_transition\};;$$

- **destination** désigne le prochain état apès la transition.
- **valeur** désigne le caractère qui sera lue sur la transition.
- **regle** désigne la règle de transition.

3.4 Le type etat

Ce type désigne un etat donné de l'automate.

$$type\ etat = \{nom : string;\ nature : type\_etat;\ mutable\ transitions : transition\ list\};;$$

- **nom** : nom de l'état (ex: q1, A, ...)
- **nature** : désigne le type de l'état (Initial, Final, ...)
- **transitions**: liste de transitions partant de cet état.

3.5 Le type automate

Le type automate permet de definir un automate à pile. Il contient l'etat initial, l'alphabet de l'automate et de la pile. Ceux-ci pouvant être distincts.

$$type\ automate = \{etat\_init : etat;\ alphabet\_automate : char\ list;\ alphabet\_pile : char\ list\};;$$

- **etat_init** désigne l'état initial de l'automate
- **alphabet_automate** contient l'alphabet reconnu par l'automate
- **alphabet_pile** contient l'alphabet reconnu par la pile

4 Présentation des fonctions

4.1 La fonction ajouter_transition

Cette fonction permet d'ajouter une transition à la liste de transition (**transitions**) d'un état. Elle reçoit en argument l'état en question ainsi que la valeur sur la transition, l'état de destination et la règle associée à cette transition à savoir s'il faut empiler ou non et quelle est cette valeur.

Complexité temporelle : $O(1)$

4.2 La fonction creer_automate

Cette fonction nous permet de créer un automate. Nous lui passons en argument l'état initial, l'alphabet de l'automate et l'alphabet de la pile. Et à partir de ces arguments elle crée un nouveau type **automate**.

Complexité temporelle : $O(1)$

4.3 La fonction ieme_transition

Cette fonction retourne la i-ème transition contenu dans la liste de transition d'un **état**. Elle prend en paramètre la liste de transition et l'index de la i-ème transition à laquelle on veut accéder. Cette fonction nous permet de nous assurer de parcourir toutes les transitions d'un état donné ce qui est utile pour la reconnaissance des mots acceptés par l'automate.

Complexité temporelle : $O(n)$ où n est l'index de la n-ième transition.

4.4 La fonction depiler_pile

La fonction depiler_pile nous permet de dépiler la pile, c'est à dire retourner la pile en supprimant l'élément au dessus.Elle prend en paramètre la pile qui est de type **char list** et un booléen qui nous indique s'il faut supprimer ou non le bas de la pile afin de retourner une pile vide **[ ]** car notre pile en début d'exécution contient l'élément **'*'** qui représente notre grand **Z**. Nous dépilerons ensuite **Z** quand la pile ne contiendra que **Z** et que nous serons à la fin de l'exécution du mot à reconnaître afin de retourner la pile vide **[ ]**

Complexité temporelle : $(O(1)$

4.5 La fonction est_final

La fonction est_final nous permet de tester si un état est final ou non. Elle prend en paramètre l'état, fais un test sur sa nature et retourne un booléen.

Complexité temporelle : $(O(1)$

4.6 La fonction mot_est_trouve

Cette fonction vérifie si le mot recherché est retrouvé à partir de l'état courant de l'automate et de la pile. Elle prend en paramètre l'état en question, la pile de l'automate et retourne un booléen. Ainsi cette fonction retourne vrai lorsque l'état est final et que la pile de l'automate ne contient que **Z (['*'])**.

Complexité temporelle : $(O(1)$

4.7 La fonction decouper_mot

Cette fonction prend en paramètre un mot et un index et retourne une chaîne de caractères. Nous nous servons de cette fonction pour l'affichage en console afin de montrer les différentes étapes de la reconnaissance d'un mot. Ainsi nous découpons le mot de sorte à afficher les caractères restants du mot devant être exécutés par l'automate.

Complexité temporelle : $O(n)$ où n est la longueur du mot à découper.

4.8 La fonction afficher_pile

La fonction afficher_pile comme son nom l'indique nous permet d'afficher l'état de la pile dans la console. Elle prend donc en paramètre une pile **(une char list)** et ne retourne rien vu qu'on fait juste un affichage.

Complexité temporelle : $O(n)$ où n est la taille de la pile.

4.9 La fonction afficher_etat_automate

La fonction afficher_etat_automate est notre fonction principale d'affichage. Elle nous permet d'afficher l'état des automates ; ainsi de suivre les différentes étapes d'exécution de la reconnaissance d'un mot. On affiche le nom de l'état courant,les caractères restants du mot à reconnaître, les transitions effectuées et l'état de la pile après transition. Elle prend donc en paramètre l'état courant, la transition courante et l'état de la pile de l'automate.

Complexité temporelle : $O(1)$

4.10 La fonction tester_mot

La fonction tester_mot est la fonction la plus longue et la plus importante de notre projet. Elle Effectue une itération sur les états en partant d'un état (l'état initial) pour vérifier l'appartenance d'un mot. Elle prend en paramètre l'état de l'automate, le mot à vérifier, un index sur les caractères du le mot à verifier, un index sur la liste des transitions de l'état (afin de parcourir tous le mot et toutes les transitions) et elle prend enfin en paramètre la pile de l'automate. Elle retourne un booléen qui nous permet de savoir si le mot est valide ou non.

Complexité temporelle : $O(n * m)$ où n est la taille du mot à vérifier et m est le nombre de transition.

4.11 La fonction valider_mot

La fonction valider_mot vérifie si un mot est reconnue par un automate. Elle prend en paramètre l'automate à exécuter ainsi que le mot à tester. Cette fonction s'assure que le mot à tester n'est pas la chaîne vide ; dans ce cas elle retourne directement vrai car la chaîne vide est reconnu par tous les automates à pile. Dans le cas où le mot à tester n'est pas une chaîne vide, la fonction **valider_mot** fait appel à la fonction **tester_mot** à partir de l'état initial de l'automate.

Complexité temporelle : $O(n * m)$ où n est la taille du mot à vérifier et m est le nombre de transition. (Donc celle de la fonction tester_mot).

4.12 La fonction afficher_resultat

La fonction afficher_resultat est une fonction d'affichage qui affiche si le mot est reconnu ou non par l'automate. Elle prend en paramètre le mot et un booléen qui affiche que le mot est reconnu et non sinon. Le booléen qu'il prend en paramètre est le résultat de fonction **valider_mot** sur l'automate et le mot.

Complexité temporelle : $O(1)$

5 Exemples

Pour réaliser les tests de fonctionnalité de l'outil développé, nous avons créer trois(3) automates différents et avons testé différents mots sur ceux-ci pour s'assurer qu'uniquement le langage accepté par ces automates est reconnu. Les trois automates réalisés sont les suivants :

- Automate 1: $a^n b^n$ avec $n >= 0$
- Automate 2: $a^n b^m$ avec $n, m >= 0$ *et* $n <= m$
- Automate 3: $a^i b^j c^k$ avec $i = j$ *ou* $i = k$

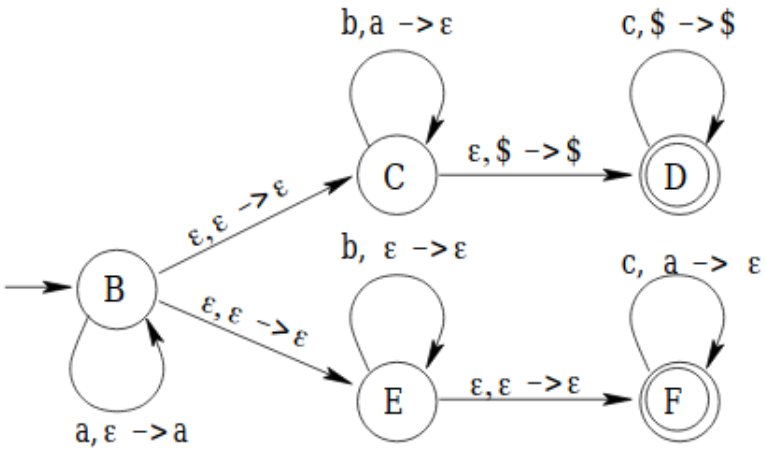

Figure 1 : Figure de l'automate 3. Source

Le code de tests de ces trois automates est dans le code du projet.

6 Support de présentation

Comme dit précédemment dans la présentation, nous avons choisi de présenter le travail en console de manière esthétique en utilisant les couleurs de la norme ANSI.

7 Conclusion

Nous sommes satisfait du travail effectué, car nous avons réussi à réaliser un outil fonctionnel en utilisant la programmation fonctionnelle. A la lecture du libellé du projet, on avait l'impression que le sujet allait être compliqué à réaliser surtout le terme "Modèle de calcul" nous faisait un peu peur, car on ne savait pas par quoi commencer. Après le choix du modèle à réaliser et la définition des types de données, le problème est devenu plus abordable. Nous savons que c'est optionnel mais notre travail serait encore plus propre si nous arrivions à représenter les automates et les différentes étapes d'exécution à partir d'une interface graphique.

Comme amélioration, on pourrait ajouter une interface graphique afin de pouvoir créer des automates à partir d'une palette comme sur JFLAP.

La réalisation de ce projet nous permis de consolider nos connaissance en programmation fonctionnelle, particulièrement en Ocaml.

References

[1] Wikipédia. *Automates à pile*. URL: https://fr.wikipedia.org/wiki/Automate_%C3%A0_pile.

[2] Wikipédia. *Automates à pile*. URL: https://fr.wikipedia.org/wiki/Automate_%C3%A0_pile#D%C3%A9finition_formelle.