Rapport Projet IPI 2021-2022

Automates LR(1)

Teddy Alexandre

Janvier 2022

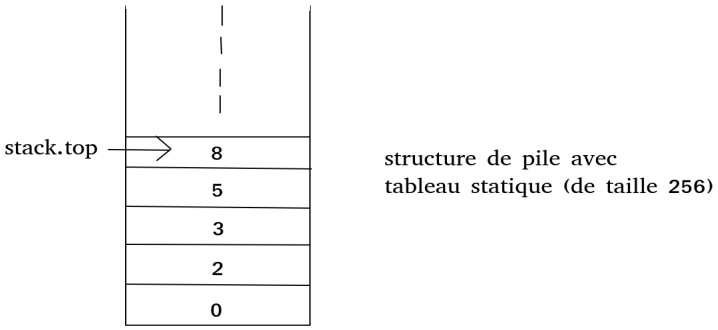
Table des matières

1 Introduction

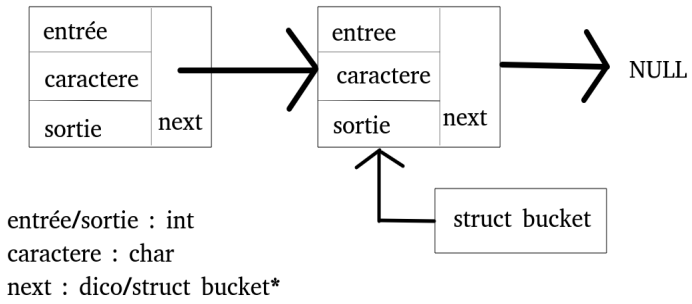
L'objectif du projet était d'exécuter des automates LR(1) afin de reconnaître certains langages. Il fallait procéder à la lecture d'un fichier binaire donnant le format l'automate en question, puis lire des mots/expressions entrés au clavier afin de savoir si ces derniers appartenaient au langage décrit par l'automate. Ces fichiers contenaient le nombre d'états de l'automate, l'ensemble des actions possibles, ainsi que des données correspondant aux fonctions de réduction, au décalage et au branchement.

2 Structures de données utilisées

Pile : Étant donné les contraintes de l'énoncé, l'utilisation d'une pile était je pense nécessaire. Le nombre d'états étant ici borné (256 au maximum), j'ai fait le choix d'une implémentation avec un tableau statique de taille 256, avec un attribut *top* qui permet l'accès au sommet de la pile en temps constant. Ceci fournit la structure de pile utilisée dans le programme. J'ai ensuite réalisé l'interface d'une pile, avec le test de vacuité, l'initialisation de la pile, l'affichage de la pile, l'empilement et le dépilement. On notera que l'implémentation aurait pu se baser sur celle des listes chaînées habituellement réalisée (une structure contenant un élément et un pointeur sur l'élément suivant).



Dictionnaire : Ce choix a été fait pour stocker le résultat des fonctions partielles *décale* et *branchement* d'une manière cohérente (ici pouvoir définir une fonction partielle discrète). La structure de dictionnaire est basée sur une implémentation similaire à celle des listes chaînées en C (i.e telle que décrite ci-dessus). On définit un dictionnaire comme un pointeur sur un "bucket". Les attributs d'un bucket sont l'état d'entrée s, le caractère c et l'état de sortie s' de sorte que *décale(s,c) = s'* et *branchement(s,c) = s'*, le tout suivi d'un pointeur vers l'élément suivant next. La clé ici est constitué d'un couple (s,c), associée à sa valeur s'. Les fonctions utilisées dans le projet sont l'initialisation d'un dictionnaire (pointeur valant *NULL*), l'ajout en tête de dictionnaire, l'affichage, la recherche de l'état de sortie et la suppression de tous les éléments (sert uniquement pour libérer la mémoire, mais pas dans le corps du programme).



Représentation d'un dictionnaire avec des listes chainées

Matrices/Tableaux/Chaînes de caractères : Afin de stocker le résultat des actions, représentés par des entiers, j'ai utilisé une matrice de taille *n * 128* (*n* étant nombre d'états de l'automate) pour stocker ces entiers. Les deux composantes de *Réduit* ont été représentées par un tableau/vecteur de taille *n* pour les stocker. La deuxième composante étant quant à elle un caractère (symbole non terminal), on y a accès facilement grâce à son code ASCII compris entre 0 et 127. Des chaînes de caractères sont utilisées dans le projet, sans pour autant en constituer le coeur (le projet fonctionnerait sans celles-ci)

3 Difficultés rencontrées au cours du projet - Solutions apportées

La principale difficulté à laquelle j'ai dû me confronter est ce qui à mes yeux constituait le cœur du programme, à savoir comment récupérer le plus simplement possible les données du fichier lors de la lecture. J'ai notamment perdu pas mal de temps à utiliser en vain la combinaison *fgets/sscanf* pour lire des données, sans me rendre compte qu'il s'agissait d'un fichier binaire, et que par conséquent les données lues ne s'affichaient pas convenablement à l'écran (ce qui est en réalité tout à fait normal, les caractères étant illisibles). J'ai ensuite utilisé *fread* pour lire les données, ce qui fonctionnait alors bien mieux.

J'ai également beaucoup réfléchi sur le type de retour des données : fallait-il les considérer comme des entiers (*int*) ou des caractères (*char*) ? J'ai fait le choix de tout considérer comme des entiers (sauf lors de la lecture de la deuxième composante de *Réduit*), quitte à effectuer la conversion en caractère derrière (notamment lors de l'ajout des deuxièmes arguments de la procédure *dico_add*), assez simple ici puisque les valeurs étaient comprises entre 0 et 127 (ce qui correspond au code ASCII de ces caractères).

La récupération des données pour les fonctions partielles *décale* et *branchement* était un peu délicate, il fallait bien s'y prendre pour bien lire les données d'une part, et ensuite effectuer les opérations sans perdre aucune donnée (gestion des séparateurs '255' '255' '255'). Les erreurs de segmentation m'ont également demandé pas mal de persévérance afin de les corriger.

4 Axes d'amélioration possibles

Avec davantage de recul, certaines parties du programme pouvaient être réalisées plus efficacement. Me vient en tête la récupération des séquences de groupement d'octets avec dicoSequence qui pouvait se faire très certainement de manière plus simple que celle que j'ai implémentée personnellement, utilisant pas mal de structures de données et de variables pour construire les dictionnaires. Ceci est, à mon avis, dû à l'idée d'implémenter un dictionnaire basé sur des listes chaînées. Stocker ces mêmes triplets dans des tableaux m'aurait très certainement évité d'utiliser autant de structures de données, avec en prime une fonction *find_etat_sortie* réalisable en temps constant avec des tableaux (ici le parcours se fait en temps linéaire en la taille du dictionnaire). Néanmoins, les fonctions *décale* et *branchement* étant partielles, elles ne sont pas définies pour toutes les valeurs d'états et de symboles possibles, ce qui rendait l'utilisation des tableaux peut-être moins naturelle (il fallait alors trouver une majoration du nombre de cases à réserver en mémoire). De même, certaines fonctions sont peut-être un peu longues (*lectureFichier* ou *dicoSequence*), et auraient éventuellement pu être raccourcies en définissant d'autres fonctions, notamment lors de la création de toutes les structures de données nécessaires avant la lecture des expressions en entrée standard.

5 Conclusion

Le projet semblait un peu obscur au départ, surtout dans la manière de lire les données, qui ne me paraissait pas immédiate. Néanmoins, dès lors que j'ai pu lire les données correctement et les stocker d'une manière satisfaisante, le reste me semblait plus naturel. En particulier, je n'ai eu aucun mal à traiter la lecture des données sur l'entrée standard. La difficulté principale était de considérer de bonnes structures de données pour stocker les données, et ensuite d'arriver à les lire.

Même si l'énoncé ne le suggérait pas, j'ai pris l'initiative d'insérer une condition (taper à l'entrée *"Fin de lecture"*) afin d'interrompre la lecture et de libérer la mémoire de toutes les structures de données utilisées (à chaque *malloc/calloc*, il est recommandé de *free* après utilisation des données), et de fermer le fichier avec *fclose*. C'est seulement à ce moment que les fonctions utilisant les chaînes de caractères sont utilisées (*tailleChaine* et *compareChaines*).