Projet Ocaml : Machine de Turing

Leo Perrat E156638J, Benjamin Scherdel E15F469A

23 Mai 2020

Sommaire

1 Introduction

Nous avons décidé pour ce projet d'implémenter une machine de turing en ocaml. Une machine de turing est composée d'un ruban infini où sont inscrits des symboles, d'un ensemble d'états dont un état initial et un ensemble d'état finaux, et une table de transition.
Le ruban possède une tête de lecture qui peut être lue et où l'on peut écrire un symbole.
Pour effectuer une transition on doit pouvoir lire et écrire sur la tête de lecture et la déplacer.
On effectue les transitions dans la table jusqu'à ce qu'on ne puisse plus faire de transition où qu'on arrive sur un état final.
Dans ce projet nous avons créé un ensemble de fonctions calculant une machine de turing plus des fonctions d'affichage et de parsing.

1.1 Utilisation

Le fichier "main.ml" est celui à executer, pour calculer une machine de turing il faut utiliser la fonction `calculer_machine` de type `ruban -> string -> string list -> transition list -> machine`
On peut aussi utiliser `parseFichier "nomDuFichier"` pour charger un fichier de machine avec la syntaxe suivante :

```
position initiale de la tête de lecture ; contenu du ruban
état initial
état final 1; état final 2; ... ; état final n
transition 1 (avec la syntaxe : état d'origine; symbole lu; symbole écrit;
              mouvement (->, <-, ou autre pour mouvement fixe);  état d'arrivée)
transition 2
...
transition n
```

2 Présentation des objets utilisés

Nous passons rapidement sur les choix des types pour les états et les symboles: l'utilisation des caractères et des chaînes de caractère nous a semblé la plus logique, les seules opérations à effectuer avec les états et les symboles sont des opérations d'égalité et les caractères permettent de représenter quasiment n'importe quoi. A symbole nous avons ajouté une valeur "Vide" pour représenter le symbole vide et "Wildcard" qui s'utilise dans le champ "symbole lu" des transitions, pour correspondre avec n'importe quel symbole, ou dans le champ "symbole écrit" pour ne pas changer le symbole.

2.1 Ruban

Le ruban de la machine de turing est censé être infini, la machine de turing étant un objet théorique cela ne pose pas de problème d'un point de vue mathématique mais évidemment nous travaillons avec des machines finies, nous ne pouvons donc pas avoir de ruban infini.
Malgré tout nous pouvons avoir l'illusion qu'il s'agit d'un ruban infini et c'est ce que nous avons fait

Nous avons tout d'abord pensé à une liste toute simple, ainsi on pourrait agrandir la liste au besoin tant qu'il nous reste de la place en mémoire, mais cela pose un problème, supposons que nous avons une liste avec les indices de 0 à 4 utilisés, notre tête de lecture est à 0 et se déplace à gauche, pour lire la tête de lecture c'est simple on peut juste renvoyer "Vide", mais pour écrire il faudrait déplacer toute la liste à droite .
Pour contourner ce problème nous avons créé un type ruban avec une liste à gauche et une liste à droite qui ressemble à ceci :

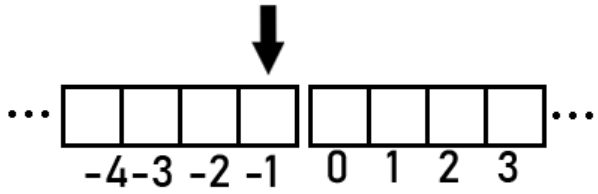

Figure 1: Un ruban avec la liste des indices négatifs à gauche et la liste des indices positifs à droite pointant sur -1

2.2 Transitions

Une transition dans une machine de turing se représente de manière formelle par :

```
état d'origine
symbole lu
symbole écrit
déplacement de la tête de lecture
état d'arrivée
```

Si l'état courant de la machine est l'état d'origine et le symbole à la tête de lecture du ruban est le "symbole lu" de la transition alors on effectue la transition et on écrit le "symbole écrit" de la transition puis on passe dans l'état d'arrivée.
Le type que nous avons créé se compose de tout ces éléments mais il dispose aussi d'une valeur "Null" qui n'a d'utilité que pour la fonction `chercher_transition`. Ensuite une table de transitions peut simplement être représentée par une liste de transitions ce que nous avons choisi de faire.
Nous avons aussi créé un type mouvement qui peut prendre 3 valeurs : Droite Gauche ou Fixe, la question s'est posée de savoir si il était nécessaire d'avoir des transitions qui peuvent avoir un mouvement fixe, la réponse est non mais nous l'avons quand même gardé pour une plus grande flexibilité d'utilisation.

2.3 Machine

Nous avons créé un type qui peut représenter une machine de turing, qui utilise les types que nous avons cité précédemment, il est composé de :

```
un ruban
l'état courant
les états finaux
la table de transition
un booléen qui indique si la machine est acceptée ou non
```

3 Fonctions principales

Il y a 3 opérations principales dans une machine de turing :

- Lire la tête de lecture
- Écrire sur la tête de lecture
- Déplacer la tête de lecture

Pour effectuer ces opérations nous avons créé beaucoup de fonctions auxiliaires que nous avons déplacé dans un fichier à part : `fonctions_utilitaires.ml`

3.1 Lecture du ruban

Le ruban d'une machine de turing est infini et le symbole vide ou symbole blanc est inscrit partout par défaut, pour simuler ce comportement on utilise le type ruban décrit précédemment. Nous avons créé deux fonctions, une plus générique qui fonctionne uniquement sur une liste : `lire` ,qui renvoie la valeur de la liste à un indice donné, et renvoie le symbole Vide si l'indice est hors de la liste, ainsi cela produit le même résultat que si l'on remplissait tout le ruban du symbole Vide.
Ensuite pour la fonction `lire_ruban` on lit soit la liste de droite si l'indice donné est positif soit la liste de gauche si l'indice est négatif.

La fonction `lire_ruban` effectue un simple parcours de liste, elle a donc un coût en O(n)

3.2 Écriture sur le ruban

Notre fonction d'écriture sur la tête de lecture est assez analogue à celle de la fonction de lecture, quand nous voulons écrire sur un indice du ruban qui dépasse la taille des deux listes, on agrandit la liste correspondante (celle des indices négatifs ou positifs) en remplissant les cases de symbole Vide, puis on écrit le symbole à l'indice donné.

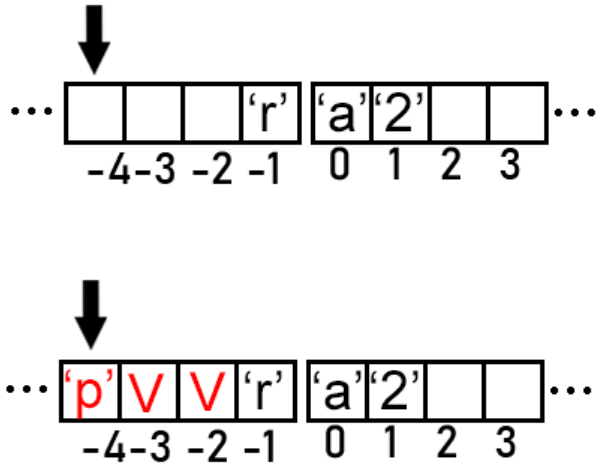

Figure 2: Écriture du symbole 'p' à l'indice -4, les symboles en rouges sont ceux écrits, V étant le symbole vide

Là aussi c'est un parcours de liste simple donc le coût est en O(n)

3.3 Déplacement de la tête de lecture

Sûrement la fonction la plus simple, en fonction des trois valeurs du type mouvement (Droite Gauche ou Fixe) ou déplace ou non la tête de lecture, cela ce fait en augmentant ou diminuant l'indice de la tête de lecture de 1. La fonction peut porter à confusion si par "déplacer la tête de lecture" on entend déplacer le ruban, ici on déplace bien la tête de lecture donc un mouvement à droite signifie augmenter de 1.

Cette fonction effectue un simple calcul et ne tient pas compte de la taille des données donc le coût est constant (O(1))

3.4 Algorithme principal

Une fois les opérations de la machine de turing implémentées il ne nous reste plus grand chose à faire, pour vérifier nous renvoyer la transition correspondante à un symbole lu et un état courant nous avons fait la

fonction `chercher_etat` , tant que l'on trouve des transitions on les effectue (donc on écrit l'état de la transition, on déplace la tête de lecture puis on change d'état courant) jusqu'à soit ne plus en trouver (la fonction `chercher_etat` renvoie la transition Null) soit on arrive sur un état final (d'où l'utilité de la fonction `recherche` qui vérifie qu'un élément est dans une liste ou non)
Ensuite on renvoie la machine correspondante, si elle s'est arreté sur un etat final le booléen du type machine qui est renvoyé est vrai.
Une critique qui peut être apportée sur notre code est qu'on peut calculer des machines qui ne sont pas vraiment déterministes si on met dans la table de transition deux transitions qui ont le même état d'origine et le même symbole lu, mais différentes du reste. Techniquement nos fonctions ne vont choisir qu'une transition mais on aurait pu renvoyer une exception dans le cas où une table de transition n'est pas compatible avec une machine déterministe.

Pour ce qui est du coût on ne peut pas vraiment savoir ça dépend vraiment de chaque machine, ce qu'on peut dire c'est qu'il y a un appel à 4 fonctions qui effectuent un parcours de liste.

4 Fonctions d'affichage et de parsing

Pour commencer, une visualisation et un parseur signifie qu'il faut choisir un caractere pour le symbole Vide, nous avons donc créé une variable `separateur` qu'on peut changer de valeur si l'on souhaite représenter le symbole vide par un autre caractère qu'un espace.

4.1 Affichage

L'affichage d'un ruban affiche 20 cases autour de la tête de lecture, l'affichage d'une transition montre simplement ses différents champs. Nous avons fait appel à ces fonctions d'affichage dans l'algorithme principal à chaque transition effectuée

4.2 Parsing

Pour les fonctions de parsing la principale difficulté a été de traiter les chaînes de caractères, nous avons créé une fonction qui convertit une chaîne de caractères en liste de caractères, plus simple à utiliser sur ocaml. A la base nous voulions utiliser la fonction `String.split_on_char` mais elle n'est pas disponible dans toutes les versions d'ocaml donc nous avons recréé une fonction similaire `split_string`
D'abord nous avons fait le parseur du ruban, à la base nous voulions juste utiliser `List.map` pour convertir tout les caractères en symboles, mais après avoir testé plusieurs exemples, certaines machines de turing nécessitent d'avoir la tête de lecture à une certaine position, et il était embêtant d'avoir à modifier la valeur initiale de la tête de lecture dans la fonction à chaque fois. La syntaxe que nous avons adopté est de donner la position initiale de la tête de lecture séparé du contenu du ruban par un point virgule.
Toutes les fonctions de parsing renvoient la même exception : `Syntaxe_fichier` quand le fichier n'est pas conforme à la syntaxe.

5 Tests et exemples utilisés

Des exemples sont fournis dans le dossier Exemples sous forme de fichiers à parser, parmis eux on retrouve des exemples simples comme l'addition binaire, une machine qui accepte le langage qui contient autant de a que de b, mais nous avons aussi testé notre programme avec une machine de turing universelle.

5.1 Machine de turing universelle

Nous ne l'avons pas réalisé nous même par manque de temps (la source est fournie ci dessous). Une machine de turing universelle est une machine de turing qui prend en entrée (le ruban) une autre machine de turing avec son entrée. Pour cela il faut choisir comment encoder la machine de turing et réussir à répliquer le fonctionnement de la machine via la table de transition. Évidemment, exécuter la machine de turing universelle

est beaucoup plus lent que d'exécuter la machine elle même.

Source de la machine de turing universelle : https://github.com/awmorp/jsturing/tree/gh-pages/machines (universal.txt)

6 Conclusion

Ce que peut faire notre programme :

- Calculer n'importe quel algorithme (thèse de Church) qui respecterait la taille de la pile
- Simuler le comportement d'une autre machine de turing

Ce que notre programme ne peut pas faire :

- Détecter les boucles infinies
- Détecter les doublons dans la table de transition