Projet de Programmation Fonctionnelle :

Machine de Turing

Gennuso Tàzio

29/03/2017

Table des matières

Introduction

L’objectif de ce projet est de modéliser le comportement d’une machine de Turing à un, puis deux rubans dans un paradigme de programmation fonctionnelle, en langage Caml.
Une machine de Turing est un modèle abstrait de calcul qui s’exécute sur un ruban (de longueur théoriquement infinie) composé de cases contenant des caractères, qui sont lus/écrits un par un par une tête de lecture/écriture qui se déplace case par case sur ce ruban.

Le fonctionnement d’une machine de Turing est le suivant :

(1) La machine est à l’état initial.
(2) Tant que l’était courant est différent de l’état final :
(3) On lit le caractère courant sur le ruban.
(4) Grâce à la table d’association, à partir du couple (état courant, caractère lu) on détermine le nouvel état, le caractère à écrire à l’emplacement de la tête, la direction de mouvement de la tête.
(5) L’état de la machine devient le nouvel état.
(6) On écrit le nouveau caractère sur le ruban à l’emplacement de la tête.
(7) On déplace la tête dans la direction donnée.
(8) On retourne à l’étape (2).
(9) On renvoie le ruban.

Ce rapport expose la façon dont j’ai modélisé une machine de Turing à un, puis à deux ruban en Caml.
Pour la machine à un ruban, on verra en premier lieu comment elle est définie, puis comment son exécution est gérée, et plus précisément comment est gérée une étape, ce qui passe par déterminer les modifications à apporter au ruban et déplacer la tête de lecture/écriture. Et finalement, comment l’affichage d’un ruban est généré.
Pour la machine à deux ruban, on retrouve en fait beaucoup la machine à un ruban dans son concept. Après avoir vu sa définition, on verra comment deux machines à un ruban sont réunies en une machine à deux rubans. Ensuite, l’exécution de la machine et l’exécution d’une étape.

Spécifications supplémentaires

Le type *direction* a été pourvu d’une direction « neutre » (notée N) pour résoudre une problématique de la machine à deux rubans qui sera détaillée plus bas.

I – Machine de Turing à un ruban.

Définir la machine.

Une machine de Turing est définie par :
- son état initial
- une liste d'associations, dont chaque élément est un couple clef-valeur dont le premier élément (la clef) est le couple (état courant, caractère lu), et le second (la valeur associée à la clef) le triplet (nouvel état, nouveau caractère, direction de la tête)
- son état final

Exécuter la machine.

Exécuter une machine de Turing consiste à exécuter l'étape suivante tant que l'état est différent de l'état final de la machine, en partant de l'état initial.

Pour chaque étape de la machine, on a besoin de connaître l'état courant. Cependant, il était demandé que la fonction *execute* n'ai pas de paramètre « état », seulement une machine et un ruban, alors qu'il était nécessaire de connaître l'état courant de la machine pour pouvoir déterminer (et exécuter) l'étape suivante. Ma solution a donc été de définir une sous-fonction *execute_rec* à l'intérieur de *execute*, qui est récursive et prend un état en argument (en plus de la machine et du ruban). Ainsi *execute* appelle une fois *execute_rec* avec l'état initial en paramètre, et c'est finalement *execute_rec* qui gère les étapes unes à unes.

Exécuter une étape de la machine.

Exécuter une étape de la machine de Turing consiste à appliquer le nouvel état, écrire le nouveau caractère et déplacer la tête d'après l'état actuel et le caractère lu.

La fonction *execute* décrite précédemment n'effectue pas elle-même les étapes de la machine de Turing. Elle appelle, à chaque étape de récursivité, la fonction *pas*, qui détermine et applique elle-même les transformations sur le ruban.
Une étape consiste à déterminer et appliquer le nouvel état à la machine, écrire le nouveau caractère sur le ruban, et déplacer la tête de lecture/écriture.

Déterminer le triplet (nouvel état, nouveau caractère, direction de la tête)

Les trois actions qu'exécute la fonction *pas* sont définies comme les valeurs associées au couple clef (état actuel, caractère lu) (cf définition de la machine

à un ruban). C'est la fonction *rechercher* qui renvoie à *pas* ce triplet de valeurs associé (par la liste d'associations qui définit la machine) au couple clef que *pas* lui passe en paramètre.

Déplacer la tête de lecture.

Les fonctions *deplace_droite* et *deplace_gauche* sont dédiées à déplacer la tête de lecture/écriture d'une case sur le ruban, respectivement vers la droite et vers la gauche.

Afficher un ruban.

Afin de pouvoir suivre l'évolution de l'algorithme de la machine de Turing, on a souhaité afficher le ruban à chaque étape, de façon à en rendre la lecture aisée et intuitive.
La fonction *affiche_ruban* fait appel à deux autres fonctions : *inv_liste* qui renvoie une liste contenant les mêmes éléments que celle qui lui est donnée en paramètre, mais dans l'ordre inverse ; et *charList_to_string* qui produit une chaîne de caractères séparés par un espace à partir d'une liste de caractères.
L'usage de ces sous-fonctions a pour but de rendre la fonction d'affichage la plus simple possible en la libérant des tâches subsidiaires. Ainsi la fonction *affiche_ruban* ne fait qu'afficher à l'écran la concaténation de chaînes de caractères.
Par ailleurs, j'ai délibérément choisi de ne pas inclure de retour à la ligne après l'affichage du ruban dans cette fonction. Ce choix contraint d'ajouter un retour à la ligne après l'appel à cette fonction, pour plus de lisibilité (un état de ruban par ligne), mais a l'avantage de permettre d'afficher plusieurs rubans sur la même ligne si nécessaire (pour la machine à deux rubans, par exemple).

II – Machine de Turing à deux rubans.

Définir la machine.

La machine de Turing à deux rubans est définie de la même manière que la machine de Turing à un ruban, à savoir un triplet (état initial, liste d'associations, état final), seulement elle est capable d'exécuter deux machine à un ruban sur deux rubans distincts.
La différence avec la machine à un ruban réside dans la définition de l'état et le contenu de la liste d'association. Un état de la machine à deux rubans est en fait le couple des états des deux sous-machines à un ruban. De même, le couple clef-valeur de la liste d'associations regroupe les deux machines : la clef est le triplet (état, caractère lu sur le premier ruban, caractère lu sur le second ruban) et la valeur associée est le quintuplet (état, caractère à écrire sur le premier ruban, caractère à écrire sur le second ruban, direction de la tête du premier ruban, direction de la tête du second ruban).

Paralléliser deux machines à un ruban.

Paralléliser deux machines à un ruban consiste à les réunir en une machine à deux rubans, ce qui est le rôle de la fonction *parallelise*.
L'état initial (respectivement final) de la machine à deux ruban est facile à construire, il s'agit simplement du couple des états initiaux (respectivement finaux) des deux machines à un ruban.
La liste d'association de la machine à deux rubans est construite en distribuant chaque association de la liste d'associations de l'une des machine à un ruban sur la liste d'associations de l'autre machine à un ruban.
Par ailleurs, lors de l'exécution d'une machine à deux rubans, une des deux sous-machines peut se terminer avant l'autre, auquel cas il n'y a pas d'association correspondant à une machine en état final et l'autre non. La fonction *parallelise* ajoute donc à la liste d'associations de chacune des deux machines à un ruban les deux associations manquantes, à savoir les pour les couples clef (etat final de la machine, '1'), (etat final de la machine, 'B') avant de les fusionner. Le triplet valeur correspondant conserve l'état, réécrit le même caractère que celui qui est lu, et ne fait pas bouger la tête, grâce à la direction N.

Exécuter la machine.

La fonction *execute2*, qui exécute une machine à deux ruban fonctionne de la même façon que la fonction *execute*. La différence est que le test sur l'état final ne se fait pas sur le couple état en lui-même, mais sur chacun des états

des sous-machines à un ruban, parce que que l’opérateur != sur les couples ne fonctionne pas.

Exécuter une étape de la machine.

La fonction *pas2* fonctionne sur le même principe que *pas* décrite dans la première partie. Seulement, on y prend en compte la direction ‘N’ (pour « Neutre »), qui signifie que la tête de lecture/écriture ne doit pas être déplacée. Cette direction est utilisée lorsque l’une des deux sous-machines à un ruban de la machine à deux rubans a terminé et pas l’autre, afin de la maintenir dans son état final sans rien modifier.

Conclusion

Les machines à un et deux rubans fonctionnent. Le fonctionnement de la machine à deux rubans est finalement très semblable à celui de la machine à un ruban, la difficulté était surtout de paralléliser deux machines à un ruban. On pourrait maintenant penser à implémenter une machine à n rubans, avec une fonction parallelise_n qui prendrait une liste de n machine de Turing à un ruban pour générer la machine de Turing à n rubans correspondante. Les implémentations de toutes les fonctions en version « n » seraient bien plus lourdes car il faudrait qu’elles soient valables quel que soit n. Il y aurait plus de sous-fonctions à définir, pour éviter de surcharger en répétant des parcours de liste. Mais ces fonctions plus génériques ne seraient pas limitées dans leurs usages, dans le sens où on pourrait paralléliser autant de machines à un ruban qu’on voudrait sans avoir à utiliser de fonctions spécifiques selon le nombre de machine à paralléliser.

Annexes

Interfaces des fonctions

```
(** deplace_gauche
@param Tete (rub_g, car, rub_d), un ruban
@return un ruban identique au ruban d'entrée, mais avec la tête de lecture décalée d'un caractère vers la gauche.
*)

(** deplace_droite
@param Tete (rub_g, car, rub_d), un ruban
@return un ruban identique au ruban d'entrée, mais avec la tête de lecture décalée d'un caractère vers la droite.
*)

(** charList_to_string
@param l, une liste de char
@return une chaine de caractère composée de la concaténation des char de la liste, dans l'ordre et séparés par des espaces.
*)

(** inv_liste
@param l, une liste
@return une liste contenant les même éléments que l, mais dans l'ordre inverse.
*)

(** affiche_ruban
@param Tete (rub_g, car, rub_d), un ruban
@return un type unit, et affiche le ruban sur la sortie standard
*)

(** rechercher
@param liste, une liste de couples (clef, valeur)
@param clef, une clef à rechercher la liste
@return une erreur "Not_found" si la clef n'est pas trouvée, la valeur associée à la clef sinon.
*)

(** pas
@param Machine (etat_init, liste_assoc, etat_final), une machine de Turing
@param Tete (rub_g, car_tete, rub_d), un ruban
@param etat, l'état actuel de la machine
@return le couple composé du nouvel état et du ruban obtenu après avoir effectué une étape de transition de la machine de turing en partant de l'état et du ruban donné
*)
```

```
(** execute
@param Machine (etat_init, liste_assoc, etat_final), une machine de Turing
@param rub, un ruban
@return le ruban une fois que la machine a atteint son état final.
*)

(** pas2
@param Machine2 (etat_init, liste_assoc, etat_final),une machine de turing à deux ruban
@param Tete (rub_g1, car_tete1, rub_d1), un premier ruban
@param Tete (rub_g2, car_tete2, rub_d2), un second ruban
@param etat, l'état actuel de la machine
@return un couple (nouvel état, (nouveau ruban 1, nouveau ruban 2))
*)

(** execute2
@param Machine2 (etat_init, liste_assoc, etat_final), une machine à deux rubans
@param rub1, le premier ruban
@param rub2, le second ruban
@return le couple des deux rubans une fois que la machine a atteint son état final
*)

(** parallelise
@param Machine (etat_init1, liste_assoc1, etat_final1), une première machine de Turing à un ruban
@param Machine (etat_init2, liste_assoc2, etat_final2), une seconde machine de Turing à un ruban
@return une machine de Turing à deux rubans composée des deux machines à un rubanfournies en paramètre
*)
```

Tests

Le ruban blanc et les machines de Turing à un ruban du castor affairé et mystères sont définies ainsi :

```
let blancs = Tete ([], 'B', []);;

let castor_affaire = Machine ('a', [    (('a','B'),('b','1',D));
                                        (('a','1'),('c','1',G));
                                        (('b','B'),('a','1',G));
                                        (('b','1'),('b','1',D));
                                        (('c','B'),('b','1',G));
                                        (('c','1'),('h','1',D))
                                ], 'h');;

let mystere = Machine (1, [     ((1,'B'),(0,'B',D));
                                ((1,'1'),(2,'B',D));
                                ((2,'B'),(3,'B',D));
                                ((2,'1'),(2,'1',D));
                                ((3,'B'),(4,'1',G));
                                ((3,'1'),(3,'1',D));
                                ((4,'B'),(5,'B',G));
                                ((4,'1'),(4,'1',G));
                                ((5,'B'),(1,'1',D));
                                ((5,'1'),(5,'1',G));
                        ], 0);;
```

Tests d’exécution des machines à un ruban :

```
# execute castor_affaire blancs;;
[B]
1 [B]
[1] 1
[B] 1 1
[B] 1 1 1
[B] 1 1 1 1
1 [1] 1 1 1
1 1 [1] 1 1
1 1 1 [1] 1
1 1 1 1 [1]
1 1 1 1 1 [B]
1 1 1 1 [1] 1
1 1 1 [1] 1 1
1 1 1 1 [1] 1
- : ruban = Tete (['1'; '1'; '1'; '1'], '1', ['1'])

# execute mystere (Tete([],'B',[]));;
[B]
B [B]
- : ruban = Tete (['B'], 'B', [])
# execute mystere (Tete([],'1',[]));;
[1]
B [B]
B B [B]
B [B] 1
[B] B 1
1 [B] 1
1 B [1]
- : ruban = Tete (['B'; '1'], '1', [])
```

```
# execute mystere (Tete(['1'],'1',[]));;
1 [1]
1 B [B]
1 B B [B]
1 B [B] 1
1 [B] B 1
1 1 [B] 1
1 1 B [1]
- : ruban = Tete (['B'; '1'; '1'], '1', [])

# execute mystere (Tete([],'1',['1']));;
[1] 1
B [1]
B 1 [B]
B 1 B [B]
B 1 [B] 1
B [1] B 1
[B] 1 B 1
1 [1] B 1
1 B [B] 1
1 B B [1]
1 B B 1 [B]
1 B B [1] 1
1 B [B] 1 1
1 [B] B 1 1
1 1 [B] 1 1
1 1 B [1] 1
- : ruban = Tete (['B'; '1'; '1'], '1', ['1'])

# execute mystere (Tete(['1'],'1',['1']));;
1 [1] 1
1 B [1]
1 B 1 [B]
1 B 1 B [B]
1 B 1 [B] 1
1 B [1] B 1
1 [B] 1 B 1
1 1 [1] B 1
1 1 B [B] 1
1 1 B B [1]
1 1 B B 1 [B]
1 1 B B [1] 1
1 1 B [B] 1 1
1 1 [B] B 1 1
1 1 1 [B] 1 1
1 1 1 B [1] 1
- : ruban = Tete (['B'; '1'; '1'; '1'], '1', ['1'])
```

Tests de parallélisation de deux machines à un ruban, et exécution de la machine à deux rubans :

```
# let machine_double = parallelise castor_affaire mystere;;
val machin_double : (char * int) turing_machine2 =
  Machine2 (('a', 1),
   [((('h', 0), '1', '1'), (('h', 0), '1', '1', N, N));
    ((('h', 0), 'B', '1'), (('h', 0), 'B', '1', N, N));
    ((('a', 0), 'B', '1'), (('b', 0), '1', '1', D, N));
    ((('a', 0), '1', '1'), (('c', 0), '1', '1', G, N));
    ((('b', 0), 'B', '1'), (('a', 0), '1', '1', G, N));
    ((('b', 0), '1', '1'), (('b', 0), '1', '1', D, N));
    ((('c', 0), 'B', '1'), (('b', 0), '1', '1', G, N));
    ((('c', 0), '1', '1'), (('h', 0), '1', '1', D, N));
    ((('h', 0), '1', 'B'), (('h', 0), '1', 'B', N, N));
    ((('h', 0), 'B', 'B'), (('h', 0), 'B', 'B', N, N));
    ((('a', 0), 'B', 'B'), (('b', 0), '1', 'B', D, N));
    ((('a', 0), '1', 'B'), (('c', 0), '1', 'B', G, N));
    ((('b', 0), 'B', 'B'), (('a', 0), '1', 'B', G, N));
    ((('b', 0), '1', 'B'), (('b', 0), '1', 'B', D, N));
    ((('c', 0), 'B', 'B'), (('b', 0), '1', 'B', G, N));
    ((('c', 0), '1', 'B'), (('h', 0), '1', 'B', D, N));
    ((('h', 1), '1', 'B'), (('h', 0), '1', 'B', N, D));
    ((('h', 1), 'B', 'B'), (('h', 0), 'B', 'B', N, D));
    ((('a', 1), 'B', 'B'), (('b', 0), '1', 'B', D, D));
    ((('a', 1), '1', 'B'), (('c', ...), ...)); ...],
   …)

# execute2 machine_double blancs (Tete(['1'],'1',['1']));;
[B] | 1 [1] 1
1 [B] | 1 [1] 1
[1] 1 | 1 B [1]
[B] 1 1 | 1 B 1 [B]
[B] 1 1 1 | 1 B 1 B [B]
[B] 1 1 1 1 | 1 B 1 [B] 1
1 [1] 1 1 1 | 1 B [1] B 1
1 1 [1] 1 1 | 1 [B] 1 B 1
1 1 1 [1] 1 | 1 1 [1] B 1
1 1 1 1 [1] | 1 1 B [B] 1
1 1 1 1 1 [B] | 1 1 B B [1]
1 1 1 1 [1] 1 | 1 1 B B 1 [B]
1 1 1 [1] 1 1 | 1 1 B B [1] 1
1 1 1 1 [1] 1 | 1 1 B [B] 1 1
1 1 1 1 [1] 1 | 1 1 [B] B 1 1
1 1 1 1 [1] 1 | 1 1 1 [B] 1 1
- : ruban * ruban =
(Tete (['1'; '1'; '1'; '1'], '1', ['1']),
 Tete (['B'; '1'; '1'; '1'], '1', ['1']))
```