Des automates aux expressions régulières

OPTION INFORMATIQUE - TP n° 4.3 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ Coder un automate généralisé
- ☞ Programmer l'élimination des états
- ☞ Construire l'expression régulière au fur et à mesure de l'élimination

A Construction d'un automate associé à une expression régulière

On dispoes des types `regexp` et `ndfsm` suivants :

```
type regexp =
        EmptySet
        | Epsilon
        | Letter of char
        | Sum of  regexp * regexp
        | Concat of regexp * regexp
        | Kleene of regexp ;;

type ndfsm =
       { states : int list;
        alphabet : char list;
        initial : int list;
        transitions : (int * char * int) list;
        accepting : int list};;
```

A1. En procédant intuitivement, proposer un automate non-déterministe associé à l'expression régulière $a(b^*|c)a^*$.

A2. En utilisant l'algorithme de Berry-Sethi, construire l'automate de Glushkov associé à $a(b^*|c)a^*$.

A3. Comparer les deux automates précédents.

A4. En OCaml, construire un automate de type `ndfsm` correspondant à $a(b^*|c)a^*$.

B Fonctions auxiliaires

Pour construire l'automate généralisé, toutes les transitions de l'automate vont devenir des expressions régulières. Par ailleurs, on cherche à traiter toutes les transitions au départ d'un même état. On s'appuie donc pour cela sur quelques fonctions intermédiaires. Pour les écrire, on pourra au choix utiliser les fonctions de la bibliothèque `List` ou écrire des fonctions récursives auxiliaires : il est important de savoir faire les deux.

B1. Écrire une fonction de signature `letter_to_regexp : char -> regexp` qui transforme une lettre `a` de type `char` de l'alphabet en une expression régulière de type `Letter a`.

B2. En utilisant la fonction précédente, écrire une fonction de signature
`trans_to_regexp : ('a * char * 'b)list -> ('a * regexp * 'b)list`
qui transforme une liste de transitions labellisées par des `char` en une liste de transitions labellisées par des `regexp`.

B3. Écrire une fonction de signature
`trans_from_q : ('a * 'b * 'c)list -> 'a -> ('a * 'b * 'c)list`
qui extrait les transitions au départ de d'un certain état `q`.

B4. Écrire une fonction de signature
`trans_from_q_no_loop : ('a * 'b * 'a)list -> 'a -> ('a * 'b * 'a)list`
qui extrait les transitions au départ de d'un certain état `q` mais pas les boucles.

B5. Écrire une fonction de signature
`find_loop : ('a * 'b * 'a)list -> ('a * 'b * 'a)option`
qui renvoie la première boucle d'une liste de transitions s'il une boucle existe et `None` sinon.

C Construire l'expression régulière

Deux grandes étapes sont nécessaires pour construire l'expression régulière équivalent à un automate. Pour chaque état `q` à éliminer, c'est à dire les états autres que l'état initial ou l'état final,

1. fusionner les expressions régulières des transitions au départ de q_s et à destination du même état q_n comme illustré sur la figure 1. Formellement, si on a les transitions (q_s, e_1, q_n) et (q_s, e_2, q_n), alors on fusionne les deux expressions en faisant leur somme : $(q_s, e_1|e_2, q_n)$. On ne conserve ainsi qu'une seule expression par destination au départ de q_s.

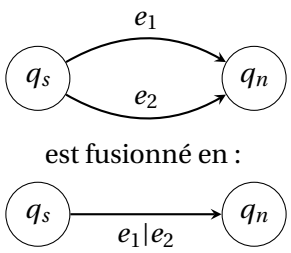

FIGURE 1 – Fusion de deux arcs au départ d'un état q_s et à destination du même état q_n

2. éliminer l'état q_s en mettant à jour les transitions au départ des états précédents. Considérons les transitions (q_p, e_1, q_s) et (q_s, e_2, q_n). Si on souhaite éliminer q_s, deux cas sont à considérer :
   (a) une transition boucle (q_s, e_b, q_s) existe : alors il est nécessaire d'ajouter la transition $(q_p, e_1 e_b^* e_2, q_n)$,
   (b) dans le cas contraire, il est nécessaire d'ajouter la transition $(q_p, e_1 e_2, q_n)$.

C1. Écrire une fonction de signature
`merge_mult_trans : ('a * regexp * 'b)list -> 'a -> ('a * regexp * 'b)list`
qui permet de fusionner les expressions régulières associées à des transitions multiples au départ de `q` et à destination d'un même état. L'utilisation d'une table de hachage du module Hashtbl pour

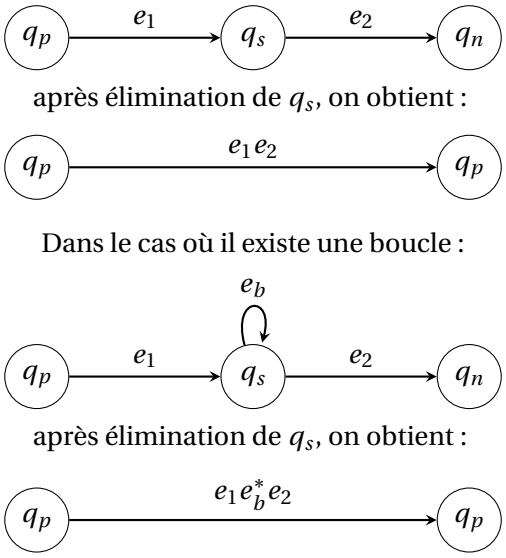

FIGURE 2 – Élimination d'un état q_s.

mémoriser les états suivants `q` déjà rencontrés et leur associer une expression régulière est recommandée. On pourra utiliser les fonction `mem`, `find` et `add` de ce module et transformer une `Hashtbl` nommé `dict` en une liste comme suit :

```
Hashtbl.fold (fun key value acc -> value::acc)dict [].
```

C2. Écrire une fonction de signature `fsm_to_regexp : ndfsm -> regexp` qui renvoie l'expression régulière associée à un automate [1]. On pourra procéder comme suit :

1. Mettre à jour au fur et à mesure une référence vers la liste des transitions restantes.
2. Supprimer les états intermédiaires qui ne sont ni accepteurs ni initial.
3. S'il existe des boucles sur l'état initial ou accepteurs, les gérer puis fusionner les expressions régulières.

1. Bonne chance!