# Des expressions régulières aux automates

OPTION INFORMATIQUE - TP nº 4.2 - Olivier Reynet

## À la fin de ce chapitre, je sais :

- coder la linéarisation d'une expressions régulière
- déterminer les composantes P,S et F relatives à une expression régulière linéaire
- coder l'algorithme de Berry-Sethi et trouver l'automate de Glushkov associé à une expression régulière

## A Linéarisation d'une expression régulière

On souhaite réaliser la linéarisation d'une expression régulière dans le but d'implémenter l'algorithme de Berry-Sethi. On dispose du type regexp algébrique suivant :

```
type regexp = EmptySet

Epsilon

Letter of char

Sum of regexp * regexp

Concat of regexp * regexp

Kleene of regexp;;
```

On se donne le type algébrique l'regexp qui représente une expression régulière linéarisée :

```
type lregexp =

Letter_ind of char * int

SumL of lregexp * lregexp

ConcatL of lregexp * lregexp

KleeneL of lregexp;;
```

Un littéral est donc codé par une lettre associée à un numéro qui code l'ordre d'apparition de la lettre dans l'expression régulière.

#### A1. Écrire une fonction récursive de signature

```
linearize_and_count : regexp -> int -> lregexp * int
```

qui linéarise une expression régulière. Le paramètre de type int est le compteur de variable : on l'incrémente à chaque fois qu'on découvre un littéral ou une occurrence d'un littéral. La fonction renvoie l'expression linéarisée ainsi que l'état du compteur de variable. On choisira des caractères arbitraires <sup>1</sup> pour l'ensemble vide et le mot vide. La fonction s'utilise ainsi :

```
linearize_and_count e 1
```

en initialisant le compteur à 1. Pour l'expression régulière  $(a|b)^*c$ , la fonction renvoie :

<sup>1.</sup> Par exemple char\_of\_int 0xD8 et char\_of\_int 0x80

OPTION INFORMATIQUE TP nº 4.2

```
(ConcatL (KleeneL (SumL (Letter_ind ('a', 1), Letter_ind ('b', 2))),Letter_ind
('c', 3)),4)
```

- A2. Proposer une fonction de signature epsilon\_is\_in : lregexp -> bool qui teste si une expression régulière linéarisée contient le mot vide.
- A3. Écrire une fonction «wrapper» de signature linearize : regexp -> lregexp qui permette de ne récupérer que l'expression régulière linéarisée, sans le compteur.

## B Calcul des composantes P, S et F associées à un expression régulière

Toujours dans l'optique d'implémenter l'algorithme de Glushkov, on cherche maintenant à caractériser les ensembles P (préfixes à une lettre), S (suffixes à une lettre) et F (facteurs possibles de deux lettres) d'une expression régulière linéarisée.

- B1. Pour les expressions régulières suivantes, trouver les ensembles P, S et F tels que définis dans le cours :  $(a|b)^*c$  et  $((a|b)^*c)|b$ .
- B2. **Ensemble P.** Écrire une fonction récursive de signature

```
first_letter_prefix : lregexp -> (char * int)list
qui renvoie la liste des préfixes à une lettre d'une expression régulière linéarisée. Par exemple pour
(a|b)*c linéarisée, la fonction renvoie (char * int)list = [('a', 1); ('b', 2); ('c', 3)].
On utilisera la concaténation de liste @ et, si besoin, la fonction epsilon_is_in.
```

B3. **Ensemble S.** Écrire une fonction récursive de signature

```
last_letter_suffix : lregexp -> (char * int)list qui renvoie la liste des suffixes à une lettre d'une expression régulière linéarisée. Par exemple pour (a|b)^*c linéarisée, la fonction renvoie (char * int)list = [('c', 3)]. On utilisera la concaténation de liste @ et, si besoin, la fonction epsilon_is_in.
```

B4. Écrire une fonction de signature

```
cartesian_product : 'a list -> 'b list -> ('a * 'b)list
qui renvoie le produit cartésien de deux listes d'entiers. Par exemple, cartesian_product [1;3]
[2;4;6;8] renvoie [(1, 2); (1, 4); (1, 6); (1, 8); (3, 2); (3, 4); (3, 6); (3, 8)].
```

B5. **Ensemble F.** Écrire une fonction récursive de signature two\_factors : lregexp -> ((char \* int )\* (char \* int))list qui renvoie les facteurs possibles de longueur 2 d'une expression régulière linéarisée. On utilisera la fonction cartesian\_product et la concaténation de listes @.

## C Algorithme de Berry-Sethi

L'algorithme de Berry-Sethi permet d'obtenir l'automate de Glushkov qui n'est pas déterministe a priori. C'est pourquoi on choisit de modéliser l'automate comme suit :

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

On choisit de représenter les états par un numéro. Le zéro est l'état initial. Les états sont ensuite numérotés d'après l'indice des lettres de l'expression régulière linéarisée. On s'appuie par ailleurs sur toutes les fonctions précédemment écrites.

OPTION INFORMATIQUE TP nº 4.2

C1. Linéariser à la main l'expression régulière  $(ab|b)^*ba$  telle que l'effectue la fonction linearize déjà programmée.

- C2. Déterminer à la main l'automate de Glushkov associé à l'expression régulière  $(ab|b)^*ba$ .
- C3. Écrire une fonction de signature all\_states : regexp -> int list qui renvoie la liste de tous les états de l'automate de Glushkov associés à une expression régulière. On utilisera la fonction linearize. Par exemple, pour  $(a|b)^*c$ , cette fonction renvoie [0; 1; 2; 3].
- C4. Les états accepteurs de l'automate de Glushkov sont déterminés par l'ensemble S obtenu grâce à la fonction last\_letter\_suffix. Écrire une fonction de signature accepting\_states : ('a \* 'b)list -> 'b list qui prend comme paramètre un ensemble S lié à une expression régulière linéarisée et qui renvoie l'ensemble des états accepteurs de l'automate de Glushkov.
- C5. Écrire une fonction récursive de signature initial\_transitions : ('a \* 'b)list -> (int \* 'a \* 'b)list dont le paramètre est un ensemble P et qui renvoie la liste des transitions depuis l'état initial de l'automate de Glushkov.
- C6. Écrire une fonction récursive de signature inner\_transitions : (('a \* 'b)\* ('c \* 'd))list -> ('b \* 'c \* 'd)list dont le paramètre est un ensemble F et qui renvoie la liste des transitions internes de l'automate de Glushkov.
- C7. Écrire une fonction de signature all\_transitions : lregexp -> (int \* char \* int)list qui renvoie la liste des transitions de l'automate de Glushkov.
- C8. Écrire une fonction de signature rm\_dup : 'a list -> 'a list qui élimine les doublons dans une liste.
- C9. Écrire une fonction de signature get\_alphabet\_from\_trans : ('a \* 'b \* 'c)list -> 'b list qui renvoie l'alphabet de l'automate de Glushkov d'après ses transitions.
- C10. Écrire une fonction de signature glushkov : regexp -> ndfsm qui renvoie l'automate de Glushkov associé à une expression régulière.
- C11. Déterminiser à la main l'automate de Glushkov obtenu grâce la fonction précédente à partir de l'expression régulière  $(ab|b)^*ba$ .

#### D Entraînement

- D1. En utilisant l'algorithme de Berry-Sethi, trouver l'automate associé aux expressions régulières suivantes :
  - (a)  $aab^*ab$
  - (b)  $a(ab)^*|b*a$
  - (c)  $(b|ab)^*(\epsilon|ab)$
- D2. En utilisant l'algorithme de Thompson, trouver l'automate associé aux expressions régulières suivantes :
  - (a)  $a^*b$
  - (b)  $aab^*ab$
  - (c)  $(a|b)^*a^*b^*$
  - (d)  $(b|ab)^*(\epsilon|ab)$
  - (e)  $a(ab)^*|b^*a$