

Les énoncés des travaux pratiques seront disponibles sur internet quelques jours avant la séance correspondante, à l'adresse indiquée ci-dessus. Les questions en rapport avec les TP, Caml Light ou l'informatique sont les bienvenues et peuvent être envoyées à [victor.nicollet@ens.fr](mailto:victor.nicollet@ens.fr)

On manipule ici des **mots**, qui sont des suites finies de lettres d'un **alphabet**. Par exemple, *car* est un mot sur l'alphabet latin. On appelle **facteur** d'un mot tout mot qui apparaît tel quel dans celui-ci. Par exemple *car* est un facteur de *decarcasser*. Il existe un unique mot ne contenant aucune lettre, appelé le mot vide et noté  $\varepsilon$ .

## 1 MOTS ET FACTEURS

En Caml Light, on représente les mots par le type `string` des chaînes de caractères. En particulier, `a^b` représente la concaténation, et `sub_string a i l` représente le facteur  $a_i \dots a_{i+l-1}$ .

Dans cet exercice, on va avoir besoin de travailler de manière importante sur les facteurs d'un mot. Pour gagner en performance, on va représenter les facteurs d'un mot non pas avec le type `string`, mais avec le type `mot` ci-contre: le facteur est déterminé par le mot entier, la lettre de début, et la lettre de fin (exclue).

```
type mot = {
  lettres : string ;
  debut : int ; fin : int
}
```

▷ **Question 1.** Écrire la fonction `mot` qui représente un mot complet à l'aide de ce type (en observant que tout mot est facteur de lui-même). Elle est de type `string -> mot`. ◀

L'intérêt de cette représentation est que couper un facteur en deux est très rapide (il n'implique pas de recopier les lettres du facteur).

▷ **Question 2.** Écrire la fonction `couper f i` qui coupe le facteur  $f$  au niveau de la lettre  $i$  du mot complet. Ainsi, si l'on coupe  $f_a \dots f_{b-1}$ , on obtiendra le mot  $f_a \dots f_{i-1}$  et le mot  $f_i \dots f_{b-1}$ . Cette fonction est de type `mot -> int -> mot * mot`. ◀

## 2 RECONNAISSANCE

Un **langage** est un sous-ensemble de  $A^*$ . La représentation informatique usuelle d'un langage  $L$  se fait à travers une fonction de reconnaissance  $f : A^* \rightarrow \{v, f\}$  telle que  $L = f^{-1}(v)$  (autrement dit, la fonction renvoie vrai si le mot fourni est dans le langage, et faux sinon).

On va travailler ici avec des fonctions de reconnaissance, dont le type sera donc `mot -> bool`.

### 2.1 LANGAGE SINGLETON

▷ **Question 3.** Écrire la fonction `facteur(f)` qui reconnaît le langage singleton  $\{f\}$  ne contenant que le mot  $f$ , de type `string -> mot -> bool`.

On appelle `let vide = facteur ""`.

On appelle `let rien = fun _ -> false`. ◀

### 2.2 LANGAGES FINIS

Si  $L_1$  et  $L_2$  sont des langages reconnus par les fonctions  $f_1$  et  $f_2$ , alors on note  $f_1|f_2$  la fonction reconnaissant le langage  $L_1 \cup L_2$ .

▷ **Question 4.** Écrire la fonction `ou a b` qui renvoie la fonction  $a|b$ , où  $a$  et  $b$  sont des fonctions de reconnaissance. ◀

À l'aide des fonctions `facteur` et `ou`, on peut construire une fonction pour reconnaître n'importe quel langage fini.

## 2.3 LANGAGES RATIONNELS

Si  $L_1$  et  $L_2$  sont des langages, on définit le langage concaténé :

$$L_1L_2 = \{ab : a \in L_1, b \in L_2\}$$

On note par ailleurs  $f_1f_2$  la fonction de reconnaissance de ce langage.

▷ **Question 5.** Écrire la fonction `puis a b` qui renvoie la fonction  $ab$ , où  $a$  et  $b$  sont des fonctions de reconnaissance. ◁

Si  $L$  est un langage, on note  $L^0 = \{\varepsilon\}$ ,  $L^1 = L$ ,  $L^2 = LL$  et plus généralement  $L^n$  la concaténation appliquée  $n$  fois au même langage.

On définit alors l'étoile :

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

▷ **Question 6.** Montrer que cette notation est cohérente si l'on considère  $\mathcal{A}$  comme l'ensemble des mots à une seule lettre. ◁

▷ **Question 7.** Écrire la fonction `etoile a` qui renvoie la fonction  $a^*$ , où  $a$  est une fonction de reconnaissance. ◁

L'ensemble des langages que l'on peut reconnaître à l'aide des seules fonctions définies dans cet énoncé est appelé ensemble des *langages rationnels*.

## 3 EXPRESSIONS RÉGULIÈRES

---

Puisque toute fonction reconnaissant un langage rationnel peut être construite à l'aide de ces opérations, on peut représenter le langage de manière simplifiée par la suite des opérations à accomplir pour créer sa fonction de reconnaissance.

On appelle alors expression régulière tout objet de la forme :

$$e = \begin{cases} \emptyset & \text{Rien} \\ \varepsilon & \text{Vide} \\ a \in \mathcal{A} & \text{Lettre} \\ e|e & \text{Alternative} \\ ee & \text{Concatenation} \\ e^* & \text{Etoile} \end{cases}$$

Chaque expression régulière reconnaît un langage rationnel, et chaque langage rationnel est reconnu par une expression régulière.

▷ **Question 8.** Le langage des mots sur l'alphabet  $\{a, b\}$  qui ne contiennent pas deux  $b$  consécutifs est-il un langage rationnel? Qu'en est-il du langage des mots de la forme  $a^n b^n$ ? ◁

## 4 OUVERTURES

---

Parmi les questions auxquelles on répond en général sur les langages :

**Un langage  $L$  est-il rationnel?** Si oui, on exhibe l'expression régulière ou l'*automate* le reconnaissant. Si non, on utilise le *lemme de l'étoile*. Dans les deux cas, on peut utiliser les *résiduels*.

**Comment améliorer la vitesse de reconnaissance?** Actuellement très faible en cas de concaténation ou d'étoile (temps quadratique, voire pire). On utilise des *automates* qui reconnaissent ou rejettent un mot en temps linéaire, en les construisant rapidement grâce à l'algorithme de *Thompson*.