

Bases formelles du TAL

corrections d'exercices

Pierre-Léo Bégay

February 1, 2020

Contents

1	Langages	2
1.1	Mots	2
1.2	Langage	5
2	Expressions régulières	6
2.1	Lexique et idée générale	6
2.1.1	Les lettres et ϵ , la base	6
2.1.2	\cdot , la concaténation	6
2.1.3	$*$, l'itération	6
2.1.4	$+$, la disjonction	6
2.2	Syntaxe	8
2.3	Sémantique	8
2.3.1	Les cas de base	8
2.3.2	Sémantique de la concaténation	8
2.3.3	Sémantique de la disjonction	8
2.3.4	Sémantique de l'itération	8
2.4	Mise en application	8
2.4.1	Quelques astuces	8
2.4.2	Syntaxe en pratique	8

Chapter 1

Langages

1.1 Mots

Exercice 1.1.1. Combien de préfixes et suffixes admet un mot w quelconque ?

Correction. Soit un mot w qui se décompose en $c_1c_2\dots c_n$ avec $\forall i \in [1 - n], c_i \in \Sigma$. Il y a $n + 1$ préfixes : $\epsilon, c_1, c_1c_2, c_1c_2c_3, \dots$, et $c_1\dots c_n$ entier. Pareil pour les suffixes avec $\epsilon, c_n, c_{n-1}c_n, \dots, c_1\dots c_n$. Pour tout mot w , il y a donc $|w| + 1$ préfixes et suffixes.

Exercice 1.1.2. Donner l'ensemble des facteurs du mot *abbba*.

Correction. On note en *rouge* les facteurs :

- *abbba* (ϵ)
- *abbba*
- *abbba*
- *abbba*
- *abbba*
- *abbba*
- *abbba*
- *abbba*
- *abbba*
 - On saute *abbba* et *abbba*
- *abbba*
 - On saute *abbba*
- *abbba*
 - On saute *abbba*

Exercice 1.1.3. (*) Donner la borne la plus basse possible du nombre de facteurs d'un mot w . Donner un mot d'au moins 3 lettres dont le nombre de facteurs est exactement la borne donnée.

Correction. Soit $w = c_1 \dots c_n$. L'ensemble des **facteurs** de w est l'ensemble des $c_1 \dots c_{i-1} \mathbf{c_i} \dots \mathbf{c_j} c_{j+1} \dots c_n$ ainsi qu' ϵ . Le nombre de ces facteurs non-nuls est borné par

$$|\{(i, j) \mid 0 \leq i < j \leq n\}| = {}^1 1 + 2 + 3 + \dots + n = {}^2 \frac{n(n+1)}{2} \in O(n^2)$$

Il ne s'agit que d'une borne, puisqu'il y aura des répétitions à partir du moment où une même lettre apparaît deux fois (cf. l'exercice précédent). Duale, le mot abc par exemple contient bien $1 + \frac{3 \times 4}{2} = 7$ facteurs.

Exercice 1.1.4. Montrer que tout facteur d'un mot en est également un sous-mot. A l'inverse, montrer qu'un sous-mot n'est pas forcément un facteur.

Correction. Soit f facteur d'un mot w . D'après la définition, ça veut dire qu'il existe v_1 et v_2 tels que $w = v_1.f.v_2$. On a alors bien $w = v_0.s_0.v_1$ avec $s_0 = f$.

Soit $w = abc$. ac en est clairement un sous-mot, alors qu'il n'en est pas un facteur.

Exercice 1.1.5. Donner toutes les façons de voir $abba$ comme sous-mot de $baaabaabbbaa$.

Correction. On a la liste (beaucoup trop longue (22 éléments !)) suivante :

- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa

¹Quand i vaut 0, il y a n possibilités pour j . Quand i vaut i , il y a $n - 1$ possibilités pour j et quand i vaut $n - 1$, il y a une possibilité pour j .

²Prouvable assez facilement par induction sur n (bon entraînement si vous n'avez pas l'habitude)

- baabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa
- baaabaabbaa

Exercice 1.1.6. Donner l'ensemble des sous-mots de *abba*

Correction. On a la liste suivante de *sous-mots* :

- *abba* (ϵ)
- *abba*
- *abba*
- *abba*
 - On saute *abba* et *abba*
- *abba*
- *abba*
 - On saute *abba*
- *abba*
- *abba*
- *abba*
 - On saute *abba* et *abba*
- *abba*
- *abba*

Exercice 1.1.7. (*)

Donner la borne la plus basse possible du nombre de sous-mots d'un mot w . Donner un mot dont le nombre de sous-mots est exactement la borne donnée.

Correction. Pour construire l'ensemble des sous-mots d'un mot w , on choisit de garder ou non chaque lettre du mot. On a donc $2^{|w|}$ choix. On a d'ailleurs, pour énumérer l'ensemble des sous-mots de l'exercice précédent, "généralisé" la suite des séries de 5 bits (00000, 00001, 00010, 00011, etc) qu'on a collées sur *abba*.

Il peut y avoir des répétitions, comme dans l'exercice précédent, ce $2^{|w|}$ n'est donc qu'une borne maximale, cependant atteinte par un mot comme *abc*.

Exercice 1.1.8. (*) Dans l'exercice 1.1.1, on demande le nombre exact de préfixes et suffixes d'un mot, alors que dans les exercices 1.1.3 et 1.1.7, on demande une borne, pourquoi ?

Correction. Le nombre de préfixes et de suffixes est toujours le même, puisqu'on n'y trouve pas de problèmes de répétitions, contrairement aux facteurs et sous-mots (cf. les exercices associés).

1.2 Langage

Chapter 2

Expressions régulières

2.1 Lexique et idée générale

2.1.1 Les lettres et ϵ , la base

2.1.2 $.$, la concaténation

2.1.3 $*$, l'itération

Exercice 2.1.1. Donner 5 autres mots appartenant au langage dénotée par l'expression $ab(bab)^*b(ca)^*b$.

Correction. $abbabbabb$ (première étoile instanciée à 2 et seconde à 0), $abbabbabbcab$ (2 et 1), $abbabbabbcacacab$ (2 et 3), $abbabbabbcacacab$ (3 et 3) et $abbabbabbbabbbabbcacacacacab$ (5 et 5).

Exercice 2.1.2. Pourquoi le changement de formulation dans les exemples 2.1.5 et 2.1.6 par rapport aux exemples précédents ("c'est à dire $\{x, y, z, \dots\}$ " qui devient "contenant notamment x, y ou z ") ?

Correction. Parce qu'on se met à étudier des *regex* qui dénotent des langages infinis, et dont il est donc assez peu pratique de faire une liste exhaustive des mots.

2.1.4 $+$, la disjonction

Exercice 2.1.3. Donner un mot acceptant deux dérivations avec la regex $(aa)^* + (bb)^*$ (justifier en donnant les dérivations). Existe-t-il un autre mot admettant plusieurs dérivations ?

Correction. On a les deux dérivations suivantes du mot ϵ :

$$\begin{array}{c} \underbrace{\epsilon}_{(bb)^0} \\ \underbrace{(bb)^0}_{(bb)^*} \\ \underbrace{(bb)^*}_{(aa)^* + (bb)^*} \end{array}$$

$$\begin{array}{c} \underbrace{\epsilon}_{(aa)^0} \\ \underbrace{(aa)^0}_{(aa)^*} \\ \underbrace{(aa)^*}_{(aa)^* + (bb)^*} \end{array}$$

Il n'existe pas d'autre mot acceptant plusieurs dérivations : dans une dérivation, on choisit d'abord si le mot sera composé de a ou de b , puis on choisit sa longueur (paire). Deux dérivations différentes généreront donc deux mots qui diffèrent par leur longueur ou les lettres qui le composent, et donc deux mots qui seront forcément différents à moins que la longueur soit 0.

Exercice 2.1.4. Existe-t-il un mot acceptant plusieurs dérivations pour la regex $(aa + bb)^*$?

Correction. Non, dans cette regex on choisit d'abord la longueur (paire) du mot, puis les lettres qui le composent. On n'a donc plus le cas de l'exercice précédent.

Exercice 2.1.5. Donner un mot accepté par la regex $(aa + bb)^*$ mais pas $(aa)^* + (bb)^*$. Est-il possible de trouver un mot qui, à l'inverse, est accepté par la deuxième mais pas la première ?

Correction. La première regex accepte par exemple $bbaa$, qui ne fait pas partie du langage dénoté par la seconde.

Tout mot accepté par la seconde regex sera accepté par la première : si on a une dérivation de la forme $(aa)^* + (bb)^* \rightarrow (aa)^* \rightarrow (aa)^n$, alors on a également $(aa + bb)^* \rightarrow (aa + bb)^n \rightarrow^n (aa)^n$ (le \rightarrow^n indique qu'il y a n étapes de dérivation, en l'occurrence n fois le choix de aa dans $aa + bb$). Même raisonnement si on part sur les b .

Exercice 2.1.6. (*) Exprimer, en langue naturelle et de façon concise, le langage dénoté par la regex $(a^*b^*)^*$. Traduire ensuite ce langage en une regex non-ambiguë, c'est-à-dire où il n'y aura qu'une dérivation pour chaque mot.

Correction. La regex permet d'engendrer n'importe quel mot. En effet, soit le mot $c_1c_2\dots c_n$, où $c_i \in \{a, b\}$ pour tout i , on peut par exemple commencer la dérivation par $(a^*b^*)^* \rightarrow (a^*b^*)^n$ et, quand $c_i = a$ (resp. b), instancier le $i^{\text{ème}}$ facteur par a^1b^0 (resp. a^0b^1).

Le langage accepté, Σ^* , est plus simplement reconnu par la regex $(a + b)^*$.

2.2 Syntaxe

2.3 Sémantique

2.3.1 Les cas de base

2.3.2 Sémantique de la concaténation

2.3.3 Sémantique de la disjonction

2.3.4 Sémantique de l'itération

2.4 Mise en application

2.4.1 Quelques astuces

Exercice 2.4.1. Donner une *regex* pour les mots qui commencent par a .

Correction. $a\Sigma^*$

Exercice 2.4.2. Donner une *regex* pour les mots qui finissent par b .

Correction. Σ^*b

Exercice 2.4.3. Donner une *regex* pour les mots qui commencent par a finissent par b .

Correction. $a\Sigma^*b$

Exercice 2.4.4. Donner une *regex* pour les mots de longueur paire.

Correction. $(\Sigma\Sigma)^*$

Exercice 2.4.5. Donner une *regex* pour les mots de longueur impaire qui contiennent au moins 4 lettres.

Correction. $\Sigma^4(\Sigma\Sigma)^*\Sigma$

Exercice 2.4.6. Donner une *regex* pour les mots de longueur impaire, qui contiennent au moins 4 lettres, commencent par a et finissent par b .

Correction. $a\Sigma^3(\Sigma\Sigma)^*b$

2.4.2 Syntaxe en pratique