Mots et langages

Ex. 1 – Préfixes et suffixes

Proposer une fonction `plpsc` de signature `string -> string` prenant en entrée une chaîne de caractères correspondant à un mot w sur l'alphabet ASCII et retournant le plus long mot v qui soit à la fois un préfixe et un suffixe de w. Quelle est sa complexité dans le pire des cas ?

Ex. 2 – Langage binaire

On s'intéresse à l'alphabet binaire $\Sigma = \{0,1\}$.

On définit le langage L comme la représentation binaire la plus courte des éléments de l'ensemble $\mathbb{N}$ des entiers positifs.

1. Définir formellement le langage L.

On définit le langage L_e sur l'alphabet Σ par $w \in L_e \iff |w|_0 = |w|_1$.

2. Déterminer $|\Sigma^n \cap L \cap L_e|$ en fonction de n.

Ex. 3 – Codes binaires

On considère l'alphabet Σ. On s'intéresse à un langage L sur Σ tel que, pour tout mots $w, w' \in L \times L$, si w est un préfixe de w', alors $w = w'$. Un tel langage est qualifié de *code*.

1. Déterminer une relation entre $\max_{w \in L}(|w|)$, $|L|$ et $|\Sigma|$.

On se place dans le cadre de l'alphabet binaire $\Sigma = \{0,1\}$.

2. Proposer un langage L tel que $|L| = 4$.

3. Même question pour $|L| = 5$.

Ex. 4 – Langages

Soit un alphabet Σ non vide. Décrire les langages définis par :

1. $L_1 = \left\{ w \in \Sigma^* \mid \exists n \in \mathbb{N} \setminus \{0,1\} \quad \text{et} \quad w' \in \Sigma^* \quad \text{tels quel} \quad w = w'^n \right\}$

2. $L_2 = \left\{ w \in \Sigma^* \mid \Pi_{c \in \Sigma} |w|_c = 0 \right\}$

3. $L_3 = \left\{ w \in \Sigma^* \mid \Pi_{c \in \Sigma} |w|_c = 1 \right\}$

4. L_2^*

5. L_3^*

Ex. 5 – Fonction génératrice

La fonction génératrice d'un langage L sur un alphabet Σ est la série formelle Φ_L définie par

$$\Phi_L \quad : \quad x \longmapsto \sum_{i=0}^{\infty} \left| L \cap \Sigma^n \right| x^n$$

1. Déterminer Φ_L pour $L = \{\varepsilon\}$

2. Même question pour $\Sigma = \{a, b, c\}$ et $L = \left\{ w \in \Sigma^* \mid |w| \leqslant 3 \right\}$.

3. Soit un langage L sur l'alphabet Σ, et $a \in \Sigma$. Déterminer la fonction génératrice de $a\,L$ en fonction de $\Phi_L(x)$.

4. Soit deux langages L et M sur l'alphabet Σ tels que $L \cap M$ soit le langage vide. Déterminer la fonction génératrice de $L \cup M$ en fonction de $\Phi_L(x)$ et $\Phi_M(x)$.

Ex. 6 – Langage construit à base d'un langage de Dyck

On considère $A = \{a\}$, $\Sigma = A \cup \overline{A} = \{a, \bar{a}\}$ et $\mathcal{D}_A$ le langage de Dyck sur Σ.

On définit le langage L par $L = (\mathcal{D}_A \bar{a})^* \mathcal{D}_A (a \mathcal{D}_A)^*$.

Déterminer L.

Ex. 7 – Suite de Thue-Morse

Soit Σ un alphabet. On s'intéresse aux mots de Σ^* ne contenant pas deux facteurs consécutifs égaux, c'est-à-dire aux mots qui ne peuvent pas s'écrire sous la forme $w = xy^2z$ où $x, z \in \Sigma^* \times \Sigma^*$ et $y \in \Sigma^+$.

1. Montrer que si $|\Sigma| = 2$, alors il n'existe aucun mot de quatre lettres ou plus respectant la condition précédente.

On suppose dans la suite $\Sigma = \{a, b\}$. On considère le morphisme σ défini par $\sigma(a) = ab$ et $\sigma(b) = ba$.

2. Montrer que pour tout $n \in \mathbb{N}$, $\sigma^{n-1}(a)$ est préfixe de $\sigma^n(a)$.

On définit le mot infini de Thue-Morse m comme le mot de longueur infinie dont tous les $\sigma^n(a)$ sont des préfixes.

On pose $L_1 = \{ab, ba\}$.

3. Montrer que $aL_1^*a \cap L_1^* = \emptyset$ et $bL_1^*b \cap L_1^* = \emptyset$

4. Montrer que $\forall n \in \mathbb{N}^*$, $\sigma^n(a) \in L_1^*$.

5. En déduire que m ne possède pas de facteur de la forme r^2c où $r \in \Sigma^+$ et $c \in \Sigma$, c étant le premier symbole de r.

On considère à présent le mot infini μ défini comme la suite du nombre de b compris entre deux a consécutifs dans le mot m.

6. Justifier que l'alphabet $\{0, 1, 2\}$ est suffisant pour écrire μ.

7. Montrer qu'il ne contient pas de facteur carré.

8. Proposer une fonction Caml qui affiche le mot μ (de façon évidente, la fonction ne terminera pas!)

Ex. 8 – Codes et algorithme de Sardinas-Patterson

Un *code* sur l'alphabet Σ est un langage L sur Σ tel que, pour tout mot $w \in L^*$, il existe une unique suite de mots $w_1, w_2, \ldots, w_n$ ($n \in \mathbb{N}$) de L tels que $w = w_1 \cdot w_2 \cdot \ldots \cdot w_n$.

1. Justifier que $\varepsilon \notin L$

2. Justifier que si $\forall u, v \in L \times L$, u n'est pas un préfixe de v, alors L est un code.

3. Le langage $L = \{a, ab\}$ est-il un code?

Si L est fini, on dispose d'un algorithme, dit de Sardinas-Patterson, pour vérifier si L est un code.

On rappelle que $M^{-1}L = \left\{ v \in \Sigma^* \mid \exists u \in M \quad \text{tel que} \quad uv \in L \right\}$

L'algorithme fonctionne de la façon suivante :

- on définit $S_1 = L^{-1}L \setminus \{\varepsilon\}$;
- puis on construit itérativement les $S_i = L^{-1}S_{i-1} \cup S_{i-1}^{-1}L$; si ε apparaît dans un des S_i, alors L n'est pas un code, et s'il existe $i \neq j$ tels que $S_i = S_j$, alors L est un code.

4. Justifier que l'algorithme termine (on admettra sa validité)

5. Vérifier avec l'algorithme précédent si les langages suivants sont des codes (si ce n'est pas le cas, on cherchera un mot de L^* qui peut se décomposer de deux façons différentes en une concaténation de mots de L) :

- $L = \{00, 01, 1, 10\}$;
- $L = \{011, 10, 1001, 11011\}$;
- $L = \{000, 010, 01001, 011\}$.

On représente en Caml un langage fini sur l'alphabet ASCII par une liste de chaînes de caractères correspondant aux mots de L (chaque mot apparaissant une fois et une seule.

6. Proposer une fonction `contientEpsilon` de signature `string list -> bool` indiquant si le langage fini passé en paramètre contient ε.

7. Écrire une fonction `estPrefixe` de signature `string -> string -> bool` indiquant si la première chaîne de caractères passée en argument est un préfixe de la seconde.

8. Écrire une fonction `reste` de signature `string -> string -> string` prenant en argument deux chaines de caractères u et v telles que $v = u.w$ (on supposera que u est bien un préfixe de v) et retournant la chaîne de caractères w.

9. Proposer une fonction `union` de signature `string list -> string list -> string list` prenant en argument deux langages finis L et M et retournant $M \cup M$.

10. Proposer une fonction `quotient` de signature `string list -> string list -> string lis` prenant en argument deux langages finis L et M et retournant $M^{-1}L$.

11. Écrire une fonction `egaux` de signature `string list -> string list -> bool` indiquant si les deux langages finis passés en argument sont identiques.

12. En déduire une fonction `estCode` de signature `string list -> bool` indiquant si le langage fini passé en argument est un code ou non, grâce à l'algorithme de Sardinas-Patterson.