I Extrait Centrale 2022 : Palindromes et rationalité

Soit $w \in \Sigma^\star$. On dit que le mot w est un palindrome si $\tilde{w} = w$.

1. Écrire une fonction palindrome de signature `string -> bool` qui teste, en temps linéaire, si un mot est un palindrome.

Pour un alphabet Σ, on note $\text{Pal}(\Sigma)$ l'ensemble des palindromes de $\Sigma^\star$.

2. Montrer que si Σ est un alphabet à une lettre, alors $\text{Pal}(\Sigma)$ est rationnel.
3. Montrer que si Σ contient au moins deux lettres, alors $\text{Pal}(\Sigma)$ n'est pas rationnel.
   On pourra utiliser un automate et un mot de $\text{Pal}(\Sigma) \cap a^\star b a^\star$.

Soit $L \subset \Sigma^\star$ un langage reconnu par l'automate $A = (Q, I, F, T)$.

Pour $(q, q') \in Q^2$, on note $L_{q,q'}$ le langage de tous les mots w qui étiquettent un chemin dans A partant de q et arrivant en q'.

4. Montrer que $L_{q,q'}$ est reconnaissable et exprimer le langage L_A en fonction de langages $L_{q,q'}$.
5. Montrer que $\text{Pal}(\Sigma) \cap \left(\Sigma^2\right)^\star = \{u\tilde{u} \mid u \in \Sigma^\star\}$.

Soit L un langage rationnel reconnu par un automate $A = (Q, I, F, T)$. On définit les langages $D(L) = \{w\tilde{w} \mid w \in L\}$ et $R(L) = \{w \in \Sigma^\star \mid w\tilde{w} \in L\}$.

6. Décrire simplement les langages $D\left(a^\star b\right)$ et $R\left(a^\star b^\star a^\star\right)$.
7. Les langages $D(L)$ et $R(L)$ sont-ils reconnaissables? On pourra faire intervenir les langages $L_{q,q'}$, définis ci-dessus.

II Extrait Centrale 2022 : Algorithme de déterminisation

On souhaite implémenter l'algorithme de déterminisation d'un automate.

```
type automate = {
  nb : int; (* nombre d'états *)
  init : int list ; (* états initiaux *)
  final : int list; (* états finaux *)
  trans : (int * char * int) list (* transitions *)
}
```

Il faut d'abord choisir une représentation pour les parties de Q (c'est-à-dire des ensembles d'états). Une solution naïve consisterait à utiliser des listes d'états. Lors du déroulement de l'algorithme de déterminisation, on peut être amené à effectuer des réunions d'ensembles. Une concaténation simple des listes génère des doublons qu'il faut ensuite supprimer afin que les listes codent bien des ensembles d'états.

1. Écrire une fonction `supprimer` de signature `'a list -> 'a list` qui prend une liste en entrée et supprime toutes les occurrences multiples de ses éléments.

2. Donner la complexité de votre algorithme en fonction de la taille de la liste d'entrée.

On choisit plutôt de coder les ensembles d'états par des entiers. Pour un automate $A = (Q, I, F, T)$ tel que $Q = [\![0, n-1]\!]$, toute partie de Q va être représentée par un entier entre 0 et $2^n - 1$. Dans la suite, on supposera $n \leqslant 20$. Soit X une partie de $[\![0, n-1]\!]$. On définit le numéro de X par la fonction suivante

$$\text{numero}(X) = \sum_{i \in X} 2^i.$$

On se donne `pow` un tableau des puissances de 2 , qui contient toutes les puissances 2^k, pour $0 \leqslant k \leqslant 20$.

```
let pow = Array.make 21 1
for i = 1 to 20 do
  pow.(i) <- pow.(i - 1) * 2
done
```

Soit $q \in [\![0, n-1]\!]$ un état et $k \in [\![0, 2^n - 1]\!]$ le numéro d'un ensemble d'états X, c'est-à-dire numero $(X) = k$.

3. Écrire une fonction `est_dans` de signature `int -> int -> bool` qui teste, à l'aide d'opérations arithmétiques, si l'état q est dans l'ensemble d'états représenté par le numéro k en $O(1)$ opérations.

Soit ℓ une liste d'états contenant éventuellement plusieurs fois le même état, représentant l'ensemble X.

4. Écrire une fonction `numero` de signature `int list -> int` qui calcule le numéro de l'ensemble X. Par exemple $\ell = [1; 5; 2; 5; 2; 5; 2; 2; 1; 2; 1]$ représente l'ensemble $X = \{1, 2, 5\}$, de numéro $38 = 2^1 + 2^2 + 2^5$. Soit ℓ une liste d'états et X un ensemble d'états représenté par son numéro k.

5. Écrire une fonction `intersecte` de signature `int list -> int -> bool` qui vérifie si un élément de ℓ est contenu dans l'ensemble X représenté par k.

On prépare désormais la fonction de transition de l'automate déterminisé accessible. Soit X un ensemble d'états de Q. On suppose désormais que l'automate est sur l'alphabet à deux lettres $\Sigma = \{a, b\}$ On cherche à calculer la fonction de transition $\delta : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ de l'automate déterminisé. On rappelle que, pour $c \in \{a, b\}$ et $X \in \mathcal{P}(Q)$

$$\delta(X, c) = \bigcup_{q \in X} \{q' \in Q \mid (q, c, q') \in T\}.$$

La transition $(X, c, \delta(X, c))$ sera alors dans l'automate déterminisé.

En parcourant l'ensemble des transitions T de l'automate, on va simultanément calculer les états $(\delta(X, a), \delta(X, b))$, ce qui correspond à la table de transition depuis l'état X.

6. Écrire une fonction `etat_suivant` de signature `int -> (int*char*int) list -> (int*int)` qui, étant donné en entrée un entier k tel que $k = \text{numero}(X)$ et la liste des transitions T, calcule le couple d'entiers (k_a, k_b) tels que $k_a = \text{numero}(\delta(X, a))$ et $k_b = \text{numero}(\delta(X, b))$.

Au moment de construire l'automate déterminisé accessible A_{det} , on va être amené à renommer (c'est-à-dire ici renuméroter) les états de A_{det} pour avoir au final un ensemble d'états Y de la forme $[\![0, N-1]\!]$ où N sera le nombre de parties de Q accessibles dans l'automate des parties. Pour cela, on va simplement utiliser une liste contenant des couples (k, v) où k est le numéro d'un ensemble d'états X et v le numéro final par lequel k sera remplacé. Par exemple, si à un moment donné de l'algorithme, la liste contient $(6, 2)$, «l'ensemble d'états 6» (qui correspond dans $\mathcal{P}(Q)$ à $\{1, 2\}$) est renuméroté 2 .

7. Écrire une fonction `cherche` de signature `int -> (int*int) list -> int` qui renvoie le nouveau numéro d'un ensemble d'états représenté par son numéro k dans une liste comme ci-dessus (-1 si k n'est pas présent).

8. Écrire une fonction `determinise` de signature `automate -> automate` qui calcule le déterminisé accessible de l'automate d'entrée. On expliquera brièvement la démarche utilisée.

9. Quelle est la complexité de votre fonction `determinise` en fonction du nombre d'états n de A et du nombre d'états N de A_{det} ?

III Extrait Centrale 2022 : Automate des dérivées d'Antimirov

On propose une méthode permettant de calculer un automate non déterministe ayant peu d'états à partir d'une expression rationnelle.

Si S et S' sont deux ensembles d'expressions rationnelles, on convient que

$$S \cdot S' = \{E \cdot E' \mid (E, E') \in S \times S'\}$$

En particulier, on a $\emptyset \cdot S = \emptyset$ et $\{\varepsilon\} \cdot S = S$

Soit E une expression rationnelle sur un alphabet Σ et soit $a \in \Sigma$ une lettre. On définit la dérivée partielle de E par a, notée $\partial_a(E)$, comme un ensemble d'expressions rationnelles défini inductivement par

$$\begin{aligned}
&\partial_a(\varnothing) = \emptyset \quad \text{(où } \emptyset \text{ est l'ensemble vide)}\\
&\partial_a(\varepsilon) = \emptyset\\
&\partial_a(b) = \begin{cases} \{\varepsilon\} & \text{si } a = b \\ \emptyset & \text{sinon} \end{cases}\\
&\text{ pour toute lettre } b \in \Sigma\\
&\partial_a(E+F) = \partial_a(E) \cup \partial_a(F)\\
&\partial_a\left(E^\star\right) = \partial_a(E) \cdot \{E^\star\}\\
&\partial_a(EF) = \begin{cases} \partial_a(E) \cdot \{F\} & \text{si } \varepsilon \notin \mathcal{L}(E) \\ \partial_a(E) \cdot \{F\} \cup \partial_a(F) & \text{sinon} \end{cases}
\end{aligned}$$

Notons bien que dans une dérivée partielle, on a une expression rationnelle, et que son résultat est un ensemble d'expressions rationnelles.

Par exemple, pour $E = a^\star(a+b) = (a^\star) \cdot (a+b)$, on calcule $\partial_a(E)$ et $\partial_b(E)$ ainsi : comme $\varepsilon \in \mathcal{L}\left(a^\star\right)$,

$$\begin{aligned}
\partial_a(E) &= \left(\partial_a\left(a^\star\right)\right) \cdot \{a+b\} \cup \partial_a(a+b)\\
&= (\partial_a a) \cdot \{a^\star\} \cdot \{a+b\} \cup \partial_a(a) \cup \partial_a(b)\\
&= \{\varepsilon\} \cdot \{a^\star(a+b)\} \cup \{\varepsilon\} \cup \emptyset\\
&= \{a^\star(a+b); \varepsilon\}\\
\partial_b(E) &= \left(\partial_b\left(a^\star\right)\right) \cdot \{a+b\} \cup \partial_b(a+b)\\
&= (\partial_b a) \cdot \{a^\star(a+b)\} \cup \emptyset \cup \{\varepsilon\}\\
&= \emptyset \cup \{\varepsilon\}\\
&= \{\varepsilon\}
\end{aligned}$$

1. Pour $E = (ab+b)^\star ba$, calculer $\partial_a(E)$ et $\partial_b(E)$.

Cette définition de dérivée partielle est étendue à tout mot $w \in \Sigma^\star$ et à des ensembles d'expressions rationnelles par : pour $a \in \Sigma, w \in \Sigma^\star$ et S un ensemble d'expressions rationnelles,

$$\partial_\varepsilon(E) = \{E\} \quad \partial_{wa}(E) = \partial_a\left(\partial_w(E)\right) \quad \partial_w(S) = \bigcup_{E \in S} \partial_w(E)$$

On construit alors l'automate d'Antimirov à partir des dérivées partielles d'une expression rationnelle.

Partons de E une expression rationnelle. L'automate d'Antimirov de l'expression E est $A = (Q, I, F, T)$ défini par

$$\begin{cases}
Q = \{E_1 \mid \exists w \in \Sigma^\star, E_1 \in \partial_w(E)\}\\
I = \{E\}\\
F = \{E_1 \in Q \mid \varepsilon \in \mathcal{L}\left(E_1\right)\}\\
T = \{(E_1, c, E_2) \in Q \times \Sigma \times Q \mid E_2 \in \partial_c\left(E_1\right)\}
\end{cases}$$

On rappelle la notation suivante : pour tout mot $w \in \Sigma^\star$ et tout langage $L \subset \Sigma^\star$,

$$w^{-1}L = \{u \in \Sigma^\star \mid wu \in L\}$$

2. Dessiner l'automate obtenu à partir de l'expression rationnelle $E = (ab+b)^\star ba$. On indiquera précisément l'ensemble d'états Q.

3. Montrer que pour tous mots u, v et tout langage L, $v^{-1}u^{-1}L = (uv)^{-1}L$.

Pour S ensemble d'expressions rationnelles, on note $\mathcal{L}(S)$ la réunion des langages des expressions de S. On admet que, si E est une expression rationnelle et x une lettre,

$$\mathcal{L}\left(\partial_x(E)\right) = x^{-1}\mathcal{L}(E)$$

4. Soit S un ensemble d'expressions rationnelles sur Σ et w un mot de $\Sigma^\star$. Montrer que

$$\mathcal{L}\left(\partial_w(S)\right) = w^{-1}\mathcal{L}(S).$$

5. Montrer que pour tout mot $w \in \Sigma^\star$, l'ensemble $\partial_w(E)$ est l'ensemble des états accessibles depuis l'état E en lisant le mot w.

6. En déduire que l'automate d'Antimirov reconnait bien le langage de l'expression rationnelle E.

Pour tout mot $w \in \Sigma^\star$ et $w \neq \varepsilon$, et pour toutes expressions rationnelles E et F sur Σ, on vérifie que

$$\partial_w(E+F) = \partial_w(E) \cup \partial_w(F)$$
$$\partial_w(EF) \subset \partial_w(E) \cdot F \cup \bigcup_{v \in S^+(w)} \partial_v(F)$$
$$\partial_w\left(E^\star\right) \subset \bigcup_{v \in S^+(w)} \partial_v(E) \cdot E^\star$$

où $S^+(w)$ est l'ensemble des suffixes non vides d'un mot w.

Pour une expression rationnelle E, on note

$$Q(E) = \bigcup_{w \in \Sigma^*, w \neq \varepsilon} \partial_w(E).$$

7. Montrer que pour toute expression rationnelle E, le cardinal de $Q(E)$ est majoré par le nombre de lettres présentes dans l'écriture syntaxique de E (qu'on notera $\|E\|$). Qu'en déduit-on sur l'automate d'Antimirov?

IV Ulm 2022 : Inégalité de Kraft

Dans ce qui suit, on se fixe un alphabet fini Σ de cardinalité $K \geq 2$. On rappelle qu'un mot $u \in \Sigma^*$ est préfixe d'un mot $w \in \Sigma^*$ s'il existe $u' \in \Sigma^*$ tel que $uu' = w$. On dit qu'un ensemble de mots $\Gamma \subseteq \Sigma^*$ est sans préfixe si aucun mot u de Γ n'est préfixe d'un autre mot v de $\Gamma(v \neq u)$.

1. Donner un exemple d'ensemble Γ sans préfixe sur l'alphabet $\Sigma = \{a, b, c\}$, contenant au moins 6 mots. Donner maintenant un exemple d'ensemble sans préfixe infini.

2. Soit $\Gamma \subseteq \Sigma^*$ un ensemble de mots sans préfixe ne contenant pas le mot vide. Montrer que Γ possède la propriété de décomposabilité unique : si un mot w peut s'écrire à la fois $w = u_1 \dots u_n$ et $w = v_1 \dots v_m$ avec tous les u_i et v_j dans Γ, alors $n = m$ et $u_i = v_i$ pour tout i.

3. Etant donné un ensemble fini de mots Γ, proposer un algorithme naïf pour tester si Γ est sans préfixe. Quelle est sa complexité?

4. Un arbre K-aire est un arbre dont chaque noeud a au plus K fils. Une feuille de l'arbre est un noeud qui n'a aucun fils. On appelle hauteur d'un noeud sa distance à la racine (cette dernière a donc hauteur 0). Dit autrement, la racine a hauteur 0 et si un noeud a hauteur h, ses fils ont hauteur $h + 1$.

   Soit T un arbre K-aire fini. Montrer que

   $$\sum_{\sigma \text{ feuille de } T} K^{\text{-hauteur } (\sigma)} \leq 1$$

   Dans quel cas a-t-on égalité?

5. En utilisant la question précédente, montrer que si $\Gamma \subseteq \Sigma^*$ est un ensemble fini sans préfixe, alors

   $$\sum_{w \in \Gamma} K^{-|w|} \leq 1$$

   On appelle cette inégalité l'inégalité de Kraft. Dans quel cas a-t-on égalité? Est-ce que l'inégalité de Kraft reste vraie pour les ensembles infinis (sans préfixe)?

6. Proposer maintenant un algorithme en temps linéaire pour tester si un ensemble de mots finis est sans préfixe.

7. On va maintenant donner une preuve algorithmique de la réciproque de l'inégalité de Kraft : si $(l_i)_{i=1}^n$ est une famille d'entiers telle que

   $$\sum_{i=1}^{n} K^{-l_i} \leq 1$$

   alors il existe une famille de mots distincts $(u_i)_{i=1}^n$ de Σ^* formant un ensemble sans préfixe avec $|u_i| = l_i$ pour tout i.

   Donner un algorithme prenant en entrée une telle suite l_i qui retourne une telle suite u_i. On demande de plus que l'algorithme soit 'en ligne', c'est-à-dire que l'algorithme lit les l_i un par un et produit u_i immédiatement après la lecture de l_i, sans attendre la valeurs des l_j pour $j > i$. Indication: Utiliser un invariant faisant intervenir, à l'étape $t + 1$, l'écriture en base K de $1 - \sum_{i=1}^{t} K^{-l_i}$.

8. Montrer que l'inégalité de Kraft reste vraie pour tout ensemble de mots Γ (fini ou infini) vérifiant la propriété de décomposabilité unique de la

9. Indication: On pourra considérer la série entière $\sum_n a_n x^n$ ò a_n est le nombre de mots de Γ de longueur n - que sait-on de son rayon de convergence? - et étudier $\left(\sum_n a_n x^n\right)^N$ pour un 'grand' entier N.

V Ulm 2022 : Automate partiellement ordonné

Soit $\mathcal{A} = (Q, A, \delta, I, F)$ un automate fini (non nécessairement déterministe). Pour $q \in Q$ et $u \in A^*$ on note $q \cdot u$ l'ensemble des états accessibles depuis q en lisant u. Si $\mathcal{A}$ est déterministe, alors on note également $q \cdot u$ (l'unique) état accessible depuis q en lisant u.

L'automate $\mathcal{A}$ est partiellement ordonné si tous ses cycles sont de taille au plus un. Formellement, si pour tout $q, q' \in Q$ et $u, v \in A^*$, tel que $q' \in q \cdot u$ et $q \in q' \cdot v$, alors $q = q'$.

1. a) Justifiez le nom partiellement ordonné pour ces automates.

   b) Montrez que le langage $\{abaa, baaa, abb\}$ est calculable par un automate déterministe partiellement ordonné.

   c) Proposez un exemple de langage non fini calculable par un automate déterministe partiellement ordonné.

   d) Montrez que le langage sur l'alphabet $A = \{a, b, c\}, K = A^*ac^*aA^*$ est calculable par un automate non déterministe partiellement ordonné.

2. Montrez que l'intersection et l'union de deux langages calculables par des automates partiellement ordonnés sont également calculables par des automates partiellement ordonnés. Est-ce que les opérations d'union et d'intersection sont toujours vérifiées quand on considère des automates partiellement ordonnés déterministes?

3. Les langages finis. Un automate $\mathcal{A} = (Q, A, \delta, I, F)$ est strictement partiellement ordonné s'il est partiellement ordonné et si pour tout etat q et lettre $a \in A$, on a $q \notin \delta(q, a)$.

   a) Montrer qu'un langage est calculable par un automate strictement partiellement ordonné si et seulement s'il est fini.

   b) Illustrer par un exemple que la taille de l'automate d'un langage fini peut être beaucoup plus petit que la taille du langage.

4. Langages clos par sur-mots. Soit $u = a_1 \cdots a_n \in A^*$.

   On note $\uparrow u$ le langage $A^*a_1A^*a_2A^* \cdots a_nA^*$. Un langage $L \subseteq A^*$ est dit clos par sur-mots si pour tout mot $u \in L$ on a $\uparrow u \subseteq L$. On souhaite montrer qu'un langage clos par sur-mots est régulier et calculable par un automate déterministe partiellement ordonné. On admettra dans la suite le résultat suivant.

   Théorème : Higman

   Soit $(w_i)_{i \in \mathbb{N}}$ une suite infinie de mots sur l'alphabet A. Il existe deux entiers i et j tels que w_i est un sur-mot de w_j.

   a) Montrez que l'intersection de deux langages clos par sur-mots est clos par sur-mots.

   b) Soit $F \subset A^*$ un ensemble fini de mots. Montrer que $\bigcup_{u \in F} \uparrow u$ est calculable par un automate déterministe partiellement ordonné.

   c) Montrez qu'un langage L est clos par sur-mots si et seulement s'il existe un ensemble fini F tel que $L = \bigcup_{u \in F} \uparrow u$ et conclure.

5. Monômes et automates non-déterministe partiellement ordonnés. On appelle monôme un langage régulier de la forme $B_0^*a_0B_1^*a_1 \cdots B_n^*$ avec $B_i \subseteq A$ et $a_i \in A$ pour tout i. On utilise la convention $\varnothing^* = \{\varepsilon\}$ avec ε le mot vide.

   a) Montrez qu'un langage régulier est calculable par un automate non-déterministe partiellement ordonné si et seulement s'il est égal à une union finie de monômes.

   b) Montrez que l'intersection de deux monômes est une union finie de monômes.

   c) Montrez que le langage $T = (ab)^*$ n'est pas calculable par un automate non-déterministe partiellement ordonné.

   d) Conclure qu'il existe un langage L tel que L est calculable par un automate non-déterministe partiellement ordonné mais pas L^c.