1 Expressions régulières

Definition 1 Un **alphabet** est un ensemble fini de symbole appelé **lettre**. On prend souvent la notation Σ pour représenter un **alphabet**.

Definition 2 Un **langage** sur un alphabet Σ, noté L est un sous-ensemble de Σ^*.

Definition 3 Soit Σ un alphabet, les **expressions régulières** sur Σ sont définis récursivement comme suit.
Cas de base :

- $\emptyset$, est l'**expression** représentant l'ensemble vide,
- ϵ est l'**expression** représentant l'ensemble $\{\epsilon\}$,
- $\forall a \in \Sigma$, a est l'**expression** représentant l'ensemble $\{a\}$,

Cas héréditaire :
Soit r, r' deux **expressions régulières** représentant respectivement les langages $R, R' \in (\Sigma^*)^2$, on a défini les opérateurs ci-dessous sur l'ensemble des **expressions régulières** de Σ que l'on note E_Σ,

- $r + r'$ représente le langage dénoté par $R \cup R'$,
- $r.r'$ représente le langage dénoté par $R.R'$,
- r^* représente le langage dénoté par R^*.

Definition 4 Sera noté $E_{(\Sigma,\mathbb{N})}$, l'ensemble des **expressions régulières** sur l'alphabet Σ au quelle aura été associé à chaque lettre son indice d'apparition dans l'ordre de lecture gauche-droite de chacune des expressions.

Definition 5 Avec $r \in E_\Sigma$, on note $L(r)$, le **langage** représenté par l'**expression régulière** r.

Definition 6 On définit la fonction **linéarisation** de la façon suivante,

$$linearisation :: E_\Sigma \to E_{(\Sigma,\mathbb{N})}$$

Definition 7 Sur une *expression régulière* $r \in E_Q$, on note la fonction renvoyant l'ensemble des premières lettres du langage représentées par r comme étant $first(r)$. Cette fonction est définie récursivement comme suit :

$$\begin{aligned}
&first(\emptyset) = first(\epsilon) = \emptyset \\
&first(x) = \{x\} \\
&first(f+g) = first(f) \cup first(g) \\
&first(f.g) = \begin{cases} first(f), & \text{si } \epsilon \notin L(f) \\ first(f) \cup first(g), & \text{si } \epsilon \in L(f) \end{cases} \\
&first(f^*) = first(f)
\end{aligned}$$

Avec $(f, g) \in (E_Q)^2$ et $x \in Q$

Definition 8 De même, on définit la fonction $last(r)$ permettant de renvoyer à partir d'une **expression régulière** l'ensemble des dernières lettres de tous les mots du langage représentés par r. Tout comme la fonction $first$, $last$ est définie de la même façon récursivement, sauf pour le cas suivant :

$$last(f.g) = \begin{cases} last(g), \text{ si } \epsilon \notin L(g) \\ last(f) \cup last(g), \text{ si } \epsilon \in L(g) \end{cases}$$

Definition 9 L'ensemble $Maybe(Q)$ est définie de la façon suivante :

$$Maybe(Q) = \{Nothing \vee Justv | v \in Q\}$$

Definition 10 On définit une Map, comme étant une structure de données. Elle stocke des valeurs sous-forme de couple clé-valeur. On ne peut d'accéder à une valeur à partir de sa clé et ceux grâce à la fonction *lookup*.

Soit $Map(K, V)$ une Map où les clés sont dans l'ensemble K et les valeurs dans l'ensemble V.

$$lookup :: Map(K, V) \to K \to Maybe(V)$$

Definition 11 Nous définissons la fonction *index* en prenant en paramètre une expression régulière indexée et retournant une Map de couple indice d'apparition symbole.

$$index :: E_{\Sigma,\mathbb{N}} \to Map(\mathbb{N}, \Sigma)$$

Definition 12 On définit la fonction $follow$, comme un moyen d'obtenir les possibles symboles qui peuvent suivre une certaine lettre dans une **expression régulière**. Cette fonction a la signature suivante :

$$follow :: E_{(\Sigma,\mathbb{N})} \to (Int \to 2^{\Sigma \times \mathbb{N}})$$

Retournant pour l'indice d'un élément d'une **expression régulière**, l'ensemble des éléments indicés qui peuvent le suivre.

2 Automate fini

Definition 13 On définit un **automate fini**, comme étant un cinq-uplet. On note généralement l'**automate** M, $M = (\Sigma, Q, I, F, \delta)$.

— Σ, est l'alphabet d'entrée,

— Q, l'ensemble des états,

— I, est un sous-ensemble de Q, il s'agit des états initiaux de l'**automate**,

— F, est un sous-ensemble de Q, il s'agit des états finaux de l'**automate**,

— δ est la fonction de transition définie de la façon suivante :

$$\delta :: Q \times \Sigma \to 2^Q$$

Definition 14 Un mot $w \in \Sigma^*, w = \{a_0, a_1, \ldots, a_{n-1}\}$ est accepté par un **automate**, $M = (\Sigma, Q, I, F, \delta)$ si et seulement si, il existe au moins un sous-ensemble d'état $Q' = \{q_0, q_1, \ldots, q_{n-1}\}$ tel que :

$$q_0 \in I \wedge q_{n-1} \in F \wedge q_i \in Q', a_i \in \Sigma, i \in [1, n], \delta(q_{i-1}, a_i) = q_i$$

Definition 15 On définit le langage d'un automate $M = (\Sigma, Q, I, F, \delta)$, comme étant $L(M) = \{w \in \Sigma^* | w \text{ est accepté par } M\}$

Definition 16 Un automate est dit **homogène** si et seulement si toutes les transitions qui arrivent sur un état sont sur le même symbole de l'alphabet.

Definition 17 Un automate $M = (\Sigma, Q, I, F, \delta)$ est dit **standard** si et seulement si :

— $|I| = 1$

— $\{q \in Q | \forall a \in \Sigma, \delta(q, a) \cap I = \emptyset\}$

Definition 18 On peut supprimer d'un automate $M = (\Sigma, Q, I, F, \delta)$ un **état**. À l'aide de la fonction :

$$removeState((\Sigma, Q, I, F, \delta), q) = (\Sigma, Q/\{q\}, I/\{q\}, F/\{q\}, \delta')$$

Avec $\delta'(e, a) = \begin{cases} \emptyset, \text{ si } e = q \\ \delta(e, a)/\{q\}, \text{ sinon} \end{cases}$ On peut aussi supprimer une **transition** q, q', a avec $(q, q') \in Q^2, a \in \Sigma$.

$$removeTransition((\Sigma, Q, I, F, \delta), (q, q', a)) = (\Sigma, Q, I, F, \delta')$$

Avec $\delta'(e, a') = \begin{cases} \delta(e, a)/\{q'\}, \text{ si } e = q \wedge a' = a \\ \delta(e, a), \text{ sinon} \end{cases}$ On étend celle-ci à la suppression de toutes les transitions entre deux nœuds de la façon suivante :

$$removeTransitions((\Sigma, Q, I, F, \delta), (q, q')) = (\Sigma, Q, I, F, \delta')$$

Avec $\delta'(e, a') = \begin{cases} \delta(e, a)/\{q'\}, \text{ si } e = q \\ \delta(e, a), \text{ sinon} \end{cases}$

Remarque 1 *Les fonctions* $removeState$*,* $removeTransition$ *et* $removeTransitions$ *augmente le nombre d'opérations faites sur la fonction* δ *de l'automate. Il serait donc intéressant dans le cas d'une implémentation de cette méthode, de donner à cette implémentation une opération de* ***mémoization*** *qui permettrait quand voulu de réduire le temps de calcul de la fonction* δ*.*

Definition 19 On définit la fonction $\Omega^+(q)$ (respectivement $\Omega^-(q)$) sur l'ensemble des états d'un automate $M = (\Sigma, Q, I, F, \delta)$, comme étant l'ensemble des directes successeurs (respectivement prédécesseur) d'un état q tel que $q \in Q$.

Definition 20 On peut construire en extrayant d'un automate tous les états n'appartenant pas à une liste d'état. On appellera cette automate, un **sous-automate d'état**. On définit alors la fonction *extractListStateAutomata*, permettant d'obtenir le $sous - automate$ de $M = (\Sigma, Q, I, F, \delta)$ ne contenant que les états de la liste $L = (q_0, \ldots, q_{n-1})$.

$$extractListStateAutomata(M, L) = (\Sigma, Q \cap L, I \cap L, F \cap L, \delta')$$

Avec $\delta'(q, a) = \begin{cases} \emptyset, \text{ si } q \notin L \\ \delta(q, a) \cap L, \text{ sinon} \end{cases}$

Definition 21 Soit un automate $M = (\Sigma, Q, I, F, \delta)$, une **orbite** est un sous-ensemble de Q tel que,

$$\forall (q_0, q_n) \in Q^2, \exists Q' = \{q, q_0, \ldots, q_{n-1}, q'\}, \text{ tel que}$$

$$\forall i \in [1, n], \exists a \in \Sigma, \delta(q_{i-1}, a) = q_i$$

Definition 22 Une **orbite** d'un automate $M = (\Sigma, Q, I, F, \delta), \mathcal{O} \subset Q$ est dite **maximale** si et seulement si, $\forall q \in \mathcal{O}, \forall q' \notin \mathcal{O}$, il n'existe pas en même temps de chemin de q vers q' et de q' vers q.

Definition 23 Pour une **orbite** $\mathcal{O}$ d'un automate $M = (\Sigma, Q, I, F, \delta)$, on définit les **portes d'entrée** (respectivement de **sortie**), noté $In(\mathcal{O})$ (respectivement $Out(\mathcal{O})$) de la façon suivante :

$$In(\mathcal{O}) = \{x \in \mathcal{O} | x \in I \vee \Omega^-(x) \neq \emptyset\}$$

$$Out(\mathcal{O}) = \{x \in \mathcal{O} | x \in F \vee \Omega^+(x) \neq \emptyset\}$$

Definition 24 Une **orbite** $\mathcal{O}$ d'un automate $M = (\Sigma, Q, I, F, \delta)$ est dite **stable** si et seulement si, $In(\mathcal{O}) \times Out(\mathcal{O}) \subset \{(q, q') | \forall q \in Q, \forall a \in \Sigma, \delta(q, a) = q'\}$.

Definition 25 Une **orbite maximale** $\mathcal{O}$ d'un automate $M = (\Sigma, Q, I, F, \delta)$ est dite **hautement stable** si et seulement si, $\mathcal{O}$ est stable et qu'après avoir supprimé toutes les transitions des états présents dans $In(\mathcal{O}) \times Out(\mathcal{O})$, chaque orbite maximal obtenu est **hautement stable**.

Definition 26 Une **orbite** $\mathcal{O}$ d'un automate $M = (\Sigma, Q, I, F, \delta)$ est dite **transverse** si et seulement si, $\forall (q, q') \in Out(\mathcal{O}), \Omega^+(x) = \Omega^+(y)$ et $\forall (q, q') \in In(\mathcal{O}), \Omega^-(x) = \Omega^-(y)$.

Definition 27 Une **orbite maximale** $\mathcal{O}$ d'un automate $M = (\Sigma, Q, I, F, \delta)$ est dite **hautement transverse** si $\mathcal{O}$ est **transverse** et que chaque sous-orbite **maximale** est **hautement transverse** une fois les transitions d'états $In(\mathcal{O}) \times Out(\mathcal{O})$ supprimer.

[1] Valentin Antimirov, *Partial derivatives of regular expressions and finite automaton constructions*

[2] Marc Chemillier, *TL Support De Cours*, https://infosetif.do.am/S4/TL/TL-SupportDeCours-MarcChemillier.pdf. Consulté le 12 juin 2024.

[3] Pascal Caron, Djelloul Ziadi, *Characterization of Glushkov automata*

[4] E. Desmontils, *Propriétés des AFNs* http://www.desmontils.net/emiage/Module209EMiage/c5/Ch5_7.htm. Consulté le 12 juin 2024.