Automate : les expressions régulières

Pour expliquer ce qu'est une expression régulière, reprenons la définition de langage. Un langage L sur un alphabet A est un ensemble de mots, c'est-à-dire un sous-ensemble de A^*. Plusieurs opérations sont possibles sur les langages, comme l'union (ensembliste), l'intersection (ensembliste) et le complémentaire. D'autres opérations sont spécifiques aux langages, comme la concaténation $L_1.L_2$, qui représente l'ensemble des mots obtenus en concaténant un mot de L_1 avec un mot de L_2. Plus formellement :

$$L_1.L_2 = \{w \in A^* \mid \exists u \in L_1, v \in L_2, u.v = w\}.$$

À partir de la concaténation, on peut définir la puissance par $L^0 = \{\varepsilon\}$ et $L^{n+1} = L.L^n$. Cela permet de définir la dernière opération qui est l'étoile $L^* = \bigcup_{i=0}^{\infty} L^i$. Toutes ces opérations sont ensemblistes uniquement.

L'idée derrière les expressions régulières est de définir un arbre de syntaxe (une sorte de langage de programmation) pour exprimer ces opérations. En OCaml, ça donnerait le type suivant :

```
type regex =
  | Vide                              (* Langage vide/ensemble vide *)
  | Mot of string                   (* Représente un langage singleton *)
  | Union of regex * regex
  | Concatenation of regex * regex
  | Etoile of regex
```

Cette description OCaml est notée plus concisément en maths avec 0 pour `Vide`, "$bonjour + coucou$" pour l'union, ab pour la concaténation, et $(a)^*$ pour l'étoile. Plus formellement, on définit les expressions rationnelles ainsi : 0 est une expression rationnelle, ε est une expression rationnelle, a pour chaque lettre est une expression rationnelle, et si E_1 et E_2 sont des expressions rationnelles, alors $(E_1)^*$, $E_1 + E_2$ et $(E_1.E_2)$ sont aussi des expressions rationnelles. Ce qui est intéressant, c'est qu'on peut définir un langage à partir d'une expression rationnelle.

```
let rec lang_of_regex r =
  match r with
  | Vide -> []
  | Lettre(x) -> [x]
  | Union (r1, r2) -> lang_of_regex r1 @ lang_of_regex r2
  | Concatenation (r1, r2) -> lang_concat (lang_of_regex r1) (lang_of_regex r2)
  | Etoile r -> lang_etoile (lang_of_regex r)
```

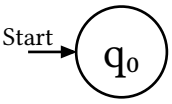
Dans le monde des maths, on définit une fonction $L : \mathrm{Regex}(A) \to \mathcal{P}(A^*)$ par induction structurelle :

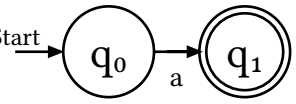
- $L(0) = \emptyset$;
- pour $a \in A$, $L(a) = \{a\}$;
- $L(E_1.E_2) = L(E_1).L(E_2)$;
- $L(E_1 + E_2) = L(E_1) \cup L(E_2)$;
- $L(E^*) = L(E)^*$.

Quel intérêt ? À partir des regex, on peut construire un automate en utilisant les opérations sur les automates que tu as implémentées dans le projet !

```
let rec auto_of_regex r =
  match r with
  | Vide -> Automate({})
  | Lettre(x) -> Automate({Transition(State(0, true, false), x, State(1, false, true))})
  | Union (r1, r2) -> union (auto_of_regex r1) (auto_of_regex r2)
  | Concatenation (r1, r2) -> concatenation (auto_of_regex r1) (auto_of_regex r2)
  | Etoile r -> etoile (auto_of_regex r)
```

En langage mathématique, c'est une fonction $\mathcal{A} : \text{Regex}(A) \to \text{Automate}$ définie par induction structurelle :

- $\mathcal{A}(0) =$ 

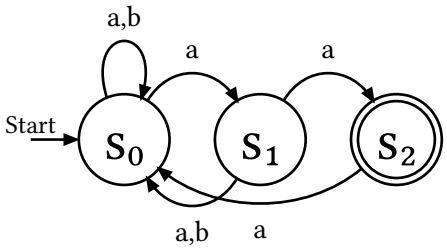
- pour $a \in A$, $\mathcal{A}(a) =$ 

- $\mathcal{A}(E_1.E_2) = \mathcal{A}(E_1).\mathcal{A}(E_2)$;
- $\mathcal{A}(E_1 + E_2) = \mathcal{A}(E_1) \cup \mathcal{A}(E_2)$;
- $\mathcal{A}(E^*) = \mathcal{A}(E)^*$.

Cette fonction correspond au premier sens du théorème de Kleene, qui dit que les langages décrits par des expressions régulières sont exactement les mêmes que ceux décrits par des automates.

En plus, je ne sais pas si tu as remarqué, mais quand j'ai défini les expressions régulières plus haut, je n'avais pas écrit de cas pour l'intersection ou le complémentaire. En fait, enlever ces opérations ne change pas l'ensemble des langages qu'on peut décrire. Si je veux calculer le complémentaire d'une expression régulière, je la convertis en automate, puis je calcule l'automate complémentaire, et enfin je reconstruis l'expression régulière correspondant à l'automate de départ.

Pour finir, on peut passer d'un automate fini vers une expression régulière en utilisant le lemme d'Arden. Je t'explique d'abord la construction, puis ensuite comment on utilise. Si on part d'un automate (S, I, F, T), l'idée est de construire un système d'équations entre différents langages obtenus en regardant le langage L_i reconnu si on remplace l'ensemble des états initiaux par le singleton $\{i\}$ pour chaque état i. Plus formellement, $L_i = L((S, \{i\}, F, T))$. Pour ça, il suffit de regarder les transitions. Par exemple, l'automate suivant :



	a	b
s0	s1, s0	s0
s1	s0, s2	s0
s2	s0	

$$\begin{cases} L_0 = (a+b)L_0 + aL_1 \\ L_1 = (a+b)L_0 + aL_2 \\ L_2 = \varepsilon + aL_0 \end{cases}$$

Pour chaque transition, il faut ajouter un terme correspondant. Par exemple, $0 \xrightarrow{a} 1$ implique $L_0 = aL_1 + \cdots$. Et pour chaque état final, il faut ajouter ε, c'est-à-dire $L_2 = \varepsilon + \cdots$. Cette construction est vraiment mécanique et ne devrait pas poser problème.

La partie plus compliquée est la résolution du système d'équations. Pour cela, il faut faire des substitutions et utiliser le fameux lemme d'Arden :

Théorème 1 (Lemme d'Arden) : Soit K et M deux langages sur un alphabet A avec $\varepsilon \notin K$. Alors $K^*.M$ est l'unique solution de l'équation $X = K.X + M$.

Preuve : La preuve est au programme mais est un peu compliquée, n'hésite pas à sauter la deuxième partie sur le fait que c'est l'unique solution.

Il faut montrer que $K^*.M$ est solution, puis que c'est l'unique solution.

On vérifie d'abord que $K^*.M$ est solution :

$$\begin{aligned} K^*M &= (\{\varepsilon\} + K.K^*).M \\ &= \{\varepsilon\}.M + K.K^*.M \\ &= M + K.(K^*.M) \end{aligned}$$

Pour montrer que c'est l'unique solution, on prend un X tel que $X = K.X + M$, et on va montrer $X = K^*.M$. Pour cela, on a besoin d'un petit lemme :

Exercice 1 : Montre par induction sur n que

$$X = M + ... + K^n.M + K^{n+1}.X = K^{n+1}.X \cup \bigcup_{i=0}^{n} K^i.M$$

Une fois ce lemme montré, on peut prouver que $X = K^*.M$ par double inclusion.

- $\boxed{K^*.M \subseteq X}$ Soit $u \in K^*.M$. Alors il existe n tel que $u \in K^n.M$. On peut utiliser le lemme pour écrire :

$$\begin{aligned} & u \in K^n.M \\ \iff & u \in \bigcup_{i=0}^{n} K^i.M \cup K^{n+1}.X \\ \iff & u \in X. \end{aligned}$$

- $\boxed{X \subseteq K^*.M}$ L'autre sens est plus subtil. Soit $u \in X$. On pose n comme la longueur de u. Grâce au lemme, u appartient à $\bigcup_{i=0}^{n} K^i.M$. Puisque $\bigcup_{i=0}^{n} K^i.M \subseteq K^*.M$, on conclut que $u \in K^*.M$.

Par double inclusion, $X = K^*.M$. ■

On peut utiliser ce lemme pour résoudre le système suivant :

$$\begin{cases} L_0 = (a+b)L_0 + aL_1 \\ L_1 = (a+b)L_0 + aL_2 \\ L_2 = \varepsilon + aL_0 \end{cases} \iff \begin{cases} L_0 = (a+b)L_0 + aL_1 \\ L_1 = (a+b)L_0 + a(\varepsilon + aL_0) \\ L_2 = \varepsilon + aL_0 \end{cases}$$

$$\iff \begin{cases} L_0 = (a+b)L_0 + a(a+b)L_0 + a(\varepsilon + aL_0) \\ L_1 = (a+b)L_0 + a(\varepsilon + aL_0) \\ L_2 = \varepsilon + aL_0 \end{cases} \iff \begin{cases} L_0 = (a+b)L_0 + a(a+b)L_0 + a + aaL_0 \\ L_1 = (a+b)L_0 + a(\varepsilon + aL_0) \\ L_2 = \varepsilon + aL_0 \end{cases}$$

$$\iff \begin{cases} L_0 = \underbrace{(a+b+a(a+b)+aa)}_{K}L_0 + \underbrace{a}_{M} \\ L_1 = (a+b)L_0 + a(\varepsilon + aL_0) \\ L_2 = \varepsilon + aL_0 \end{cases} \iff \begin{cases} L_0 = (a+b+a(a+b)+aa)^*a \\ L_1 = (a+b)L_0 + a(\varepsilon + aL_0) \\ L_2 = \varepsilon + aL_0 \end{cases}$$

$$\iff \begin{cases} L_0 = (a+b+a(a+b)+aa)^*a \\ L_1 = (a+b)(a+b+a(a+b)+aa)^*a + a(\varepsilon + a(a+b+a(a+b)+aa)^*a) \\ L_2 = \varepsilon + a(a+b+a(a+b)+aa)^*a \end{cases}$$

Et donc, une expression régulière de l'automate est $(a+b+a(a+b)+aa)^*a$. Lorsqu'il y a plusieurs états initiaux, il suffit de prendre l'union entre les différents L_i. Il peut y avoir plusieurs expressions régulières pour un même langage. Par exemple :

$$L_0 = (a + b + a(a + b) + aa)^* a = (a + b + aa + ab)^* a.$$

Ce qui compte, c'est surtout le déroulé du calcul. Dans l'exemple ci-dessus, j'ai détaillé toutes les étapes, mais en pratique, sur une copie, tu peux te contenter de noter les étapes clés avec une justification (substitution, calcul ou application du lemme d'Arden).

Exercice 2 : Donne un automate sur l'alphabet $\{a, b\}$ qui reconnaît $\{a\}^*$. Calcule son complémentaire, et retrouve une expression régulière correspondant au langage $\{a, b\}^* \setminus \{a\}^*$. Même chose pour l'intersection de $(a + b)^* a (a + b)^*$ (au moins un a) et $b^*(\varepsilon + a)b^*$ (au plus un a). Et puis pour finir, le complémentaire de l'intersection de $(a + b)^* a (a + b)^*$ avec $b^*(\varepsilon + a)b^*$.

Voila, j'espère que ce document t'aura été utile ☺