TP 1 : Langages Réguliers et Automates

5 septembre 2024

1 Expressions Régulières

1.1 type regex

1) Définir un type somme `regex` dont les constructeurs sont les suivants : `Vide`, `Epsilon`, `Lettre`, `Union`, `Concat` et `Etoile`, avec des arguments si nécessaire (`Lettre` prend un argument de type `char`).

2) Soient e_1 et e_2 deux expressions régulières. Si $e_1 = \emptyset$, quel est le langage dénoté par $e_1|e_2$, e_1e_2 et e_1^* ?

3) En déduire une fonction `est_vide` qui prend en entrée un élément de type `regex` est renvoie un booléen indiquant si le langage dénoté par l'expression régulière est vide.

4) Définir une exception `Est_vide`, puis définir un fonction `exemple_mot` qui prend en entrée une expression régulière de type `regex` et renvoie un mot du langage dénoté par l'expression régulière si celui-ci est non-vide (et lève l'exception `Est_vide` sinon).

5) On définit le type `taille` de la façon suivante :

```
type taille = Non_borne | Langage_vide | Longueur of int ;;
```

Définir une fonction `longueur_max` qui prend en entrée une expression régulière et renvoie un élément de type `taille` représentant la taille maximale d'un mot du langage dénoté (`Non_borne` s'il n'y a pas de limite de taille pour les mots dans le langage et `Langage_vide` si le langage est vide).

1.2 Module Stringset

En OCaml, on peut donner des définitions, de type et de fonctions par exemple, au sein d'un module. La syntaxe est la suivante (avec le nom de module en majuscule) :

```
module Nondumodule =
    struct
        (* Définitions *)
    end ;;
```

On peut ensuite utiliser les fonctions à l'intérieur du module à l'aide de la syntaxe suivante, par exemple pour une fonction `fonction1` définie dans un module `Nomdumodule` :

```
Nomdumodule.fonction1
```

6) On souhaite utiliser un module pour manipuler des langages, c'est-à-dire des ensembles de séquences de caractères (que l'on représentera par des listes de chaînes de caractères). Créer un module `Stringset` contenant :

- La définition d'un type `t` représentant un ensemble de chaînes de caractères
- La définition d'un élément `ens_vide` représentant l'ensemble vide
- Une fonction `add : string -> t -> t` qui renvoie l'ensemble des chaînes de caractères obtenu en ajoutant une chaîne de caractère à un ensemble.
- Une fonction `union` qui prend en entrée deux ensembles de chaînes de caractères et renvoie l'ensemble correspondant à leur union.

On souhaite pouvoir déterminer si un mot appartient au langage dénoté par une expression régulière. Comme ce langage est potentiellement infini, on peut se restreindre à son appartenance à l'ensemble des mots du langage de même longueur n. On va donc coder une fonction qui calcule, pour une expression régulière, l'ensemble des mots de longueur n dans langage dénoté par l'expression régulière.

7) Définir une fonction `concat` qui a deux langages associe la concaténation des deux langages.

Soit e une expression régulière et $n \in \mathbb{N}$. On note $\mathcal{L}_n(e)$ le langage des mots de longueur n dans le langage dénoté par e.

8) Montrer que pour toutes expressions e et f, on a $\mathcal{L}_n(e.f) = \bigcup_{k=0}^{n} \mathcal{L}_k(e) \cdot \mathcal{L}_{n-k}(f)$.

9) Montrer que $\mathcal{L}_0(e^*) = \varepsilon$ et que pour tout $n \in \mathbb{N}^*$, on a $\mathcal{L}_n(e^*) = \bigcup_{k=1}^{n} \mathcal{L}_k(e) \cdot \mathcal{L}_{n-k}(e^*)$

10) En déduire une fonction `langage_taille_n` qui a une expression régulière e et un entier n associe l'ensemble des mots de taille n dans $\mathcal{L}(e)$.

11) En déduire une fonction `appartient_langage` qui prend en entrée une chaîne de caractère et une expression régulière, et détermine si la chaîne de caractère appartient au langage dénoté par l'expression régulière.