DS 2 option informatique
3 heures

Ce devoir est constitué de deux parties : un exercice issu d'une épreuve E3A et un sujet de CentraleSupélec. Vous pouvez sauter l'exercice E3A si vous êtes à l'aise en option informatique.

Exercice E3A

On suppose disposer d'une structure impérative de dictionnaire en Caml de type `('a,'b) dict` avec les primitives suivantes :

Primitive	Type	Description
`new`	`unit -> ('a,'b) dict`	Crée un nouveau dictionnaire vide
`add`	`('a,'b) dict -> 'a -> 'b -> unit`	`add d cl val` associe, dans le dictionnaire `d`, la clef `cl` à la valeur `val`
`find`	`('a,'b) dict -> 'a -> 'b`	`find d cl` lit, dans le dictionnaire `d`, la valeur associée à la clef `cl`
`keys`	`('a,'b) dict -> 'a list`	`keys d` renvoie la liste (dans un ordre arbitraire) des clefs du dictionnaire `d`

`'a` est le type des clefs, et `'b` le type des valeurs associées aux clefs.

On suppose disposer d'une structure persistante d'ensemble d'entiers de type `set` avec les primitives suivantes :

Primitive	Type	Description
`empty`	`set`	L'ensemble vide
`union`	`set -> set -> set`	L'union de 2 ensembles
`inter`	`set -> set -> set`	L'intersection de 2 ensembles
`card`	`set -> int`	Le cardinal d'un ensemble
`equal`	`set -> set -> bool`	Teste l'égalité de 2 ensembles
`mem`	`int -> set -> bool`	`mem i s` renvoie `true` si l'entier `i` appartient à l'ensemble `s` et `false` sinon
`singleton`	`int -> set`	`singleton i` renvoie l'ensemble à un élément ne contenant que `i`

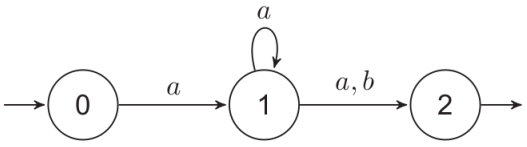
On représente un automate non-déterministe en Caml par le type suivant

```
type auto = {etats : set; init: set;
        trans: (int * char, set) dict ; final: set};;
```

Étant donné un automate `m`,

- `m.init` représente l'ensemble des états initiaux de `m` ;
- `m.final` l'ensemble de ses états finaux,
- `m.trans` sa fonction de transition qui à chaque couple `q,x` associe l'ensemble des états accessibles à partir de l'état `q` en lisant le caractère `x`.

Par exemple, considérons l'automate $\mathcal{M}_1$ suivant :



Si `m1` représente l'automate $\mathcal{M}_1$ en Caml alors `m1.final` est l'ensemble $\{2\}$ et `find m1.trans (1,`a`)` est l'ensemble $\{1;2\}$.

Un automate déterministe est un automate non-déterministe ayant un unique état initial et tel que pour tout état q et toute lettre a, en lisant a à partir de l'état q on peut aller dans au plus un état.

1. L'automate $\mathcal{M}_1$ est-il déterministe ?
2. Écrire une fonction `max_card : ('a, set) dict -> int` qui étant donné un dictionnaire d'ensembles, renvoie le cardinal maximal des ensembles stockés dans le dictionnaire.

3. Écrire une fonction `est_deterministe : auto -> bool` qui renvoie `true` si l'automate donné en argument est déterministe et `false` sinon.

4. Considérons le code incomplet suivant :

```
let etats_suivants m s x =                                    1
let rec parcours_clefs clefs acc = match clefs with           2
        | [] -> acc                                           3
        | (etat,y)::r -> if  ..............                   4
                 then  ..............                         5
                 else  ..............                         6
in parcours_clefs (keys m.trans) empty;;                      7
```

   Compléter le code de sorte que `etats_suivants m s x` renvoie l'ensemble des états accessibles à partir d'un état de l'ensemble `s` en lisant `x` dans l'automate `m`.

5. Écrire une fonction `reconnu : auto -> string -> bool` qui, étant donné un automate et une chaîne de caractères, renvoie `true` si l'automate reconnaît la chaîne et `false` sinon. (*)

6. Si au lieu d'une structure impérative de dictionnaire nous avions une structure persistante de dictionnaire, quel serait le type de la primitive `add` ? Si nous avions une structure impérative d'ensemble d'entiers et non pas une structure persistante, pourquoi le type de `empty` devrait-il être changé ?

(*) On rappelle que si s est une chaîne de caractères, alors String.length s est la taille de s et s.[i] est son ième caractère.

CONCOURS CENTRALE•SUPÉLEC

Option informatique

MP 2022

Ce sujet aborde différents problèmes autour des automates et des expressions rationnelles. On explore dans une première partie des propriétés sur le miroir d'un langage, sur les palindromes et sur les automates correspondants. On implémente ensuite l'algorithme de déterminisation d'un automate, afin de construire de manière effective l'automate de Brzozowski qui a la propriété d'être minimal. Dans une deuxième partie, on travaille sur la syntaxe des expressions rationnelles, puis sur une construction de l'expression rationnelle associée à un automate donné, par un algorithme diviser pour régner, dû à Conway, en exploitant une représentation matricielle d'un automate et la construction de l'étoile d'une matrice d'expressions rationnelles. Enfin, dans une troisième partie, on introduit les dérivées d'Antimirov, qui permettent d'obtenir un automate fini non déterministe avec peu d'états qui reconnait le langage spécifié par une expression rationnelle. Les trois parties sont indépendantes, de difficulté progressive.

Langages et mots

On appelle *alphabet* tout ensemble fini de lettres. On note généralement l'alphabet Σ.

On note $\Sigma^\star$ l'ensemble de tous les mots formés sur l'alphabet Σ.

La *longueur* (ou la *taille*) d'un mot $w \in \Sigma^\star$ est son nombre de lettres et se note $|w|$. Le *mot vide*, noté ε, est le seul mot de longueur nulle.

Si un mot $w \in \Sigma^\star$ est de longueur $|w| = n$, on le note $w = a_0 a_1 ... a_{n-1}$, où les a_i sont des lettres de Σ.

Un *langage* sur l'alphabet Σ est un ensemble $L \subset \Sigma^\star$.

L'*étoile de Kleene* d'un langage L, notée $L^\star$, est le plus petit langage qui inclut L, qui contient ε et qui est stable par concaténation.

La concaténation de deux langages L et L' est notée $L \cdot L'$, souvent abrégé en LL' lorsqu'il n'y a pas d'ambiguïté.

Automates finis

Un *automate fini non déterministe* sur un alphabet Σ est un quadruplet $A = (Q, I, F, T)$, où Q est un ensemble fini d'*états*, $I \subset Q$ est le sous-ensemble des *états initiaux*, $F \subset Q$ est le sous-ensemble des *états finaux* et l'ensemble $T \subset Q \times \Sigma \times Q$ est l'ensemble des *transitions*, étiquetées par les lettres de l'alphabet Σ.

Si $(q, a, q') \in T$, on note $q \overset{a}{\to} q'$ cette transition.

Pour représenter graphiquement un automate, on utilise une flèche entrante pour désigner un état initial et une flèche sortante pour désigner un état final, comme l'illustre l'exemple de la figure 1.

Un mot $w = a_0 ... a_{n-1}$ est reconnu par l'automate A s'il existe une succession de transitions :

$$q_0 \overset{a_0}{\to} q_1 \overset{a_1}{\to} \cdots \; q_{n-1} \overset{a_{n-1}}{\to} q_n \qquad \text{avec} \qquad q_0 \in I \quad \text{et} \quad q_n \in F.$$

On dira que le mot w *étiquette un chemin* dans l'automate A allant de q_0 à q_n.

Le langage d'un automate A, noté L_A, est exactement l'ensemble des mots reconnus par l'automate A. On dit alors que A *reconnait* L_A. Un langage est dit *reconnaissable* s'il est le langage d'un automate fini.

Un *automate fini déterministe* sur un alphabet Σ est un quadruplet $A = (Q, \{q_0\}, F, \delta)$, où l'ensemble des états initiaux est un singleton (un unique état initial) et où l'ensemble des transitions T est remplacé par une fonction de transition δ définie sur un sous-ensemble de $Q \times \Sigma$ et à valeurs dans Q. Pour chaque couple $(q, a) \in Q \times \Sigma$, il existe au plus une transition (q, a, q') qui, si elle existe, est telle que $q' = \delta(q, a)$.

L'automate est déterministe *complet* si la fonction de transition δ est définie sur $Q \times \Sigma$. Dans ce cas, on définit la *fonction de transition étendue* $\delta^\star$ sur $Q \times \Sigma^\star$ par

$$\forall q \in Q, \quad \begin{cases} \delta^\star(q, \varepsilon) = q \\ \delta^\star(q, wa) = \delta\big(\delta^\star(q, w), a\big) \end{cases} \quad \forall w \in \Sigma^\star, \forall a \in \Sigma$$

Les automates seront représentés par le type Caml suivant

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

l'ensemble d'états Q d'un automate implémenté étant toujours supposé être un intervalle d'entiers $[\![0, n-1]\!]$.

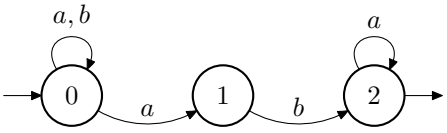


Figure 1 L'automate $\mathcal{A}_1$

Par exemple, l'automate $\mathcal{A}_1$ de la figure 1 est codé par

```
let a1 = { nb = 3 ;
           init = [0];
           final = [2];
           trans = [(0, 'a', 0); (0, 'a', 1); (0, 'b', 0); (1, 'b', 2); (2, 'a', 2)] } ;;
```

On accède au nombre d'états par `a1.nb`, à la liste des états initiaux par `a1.init`, à la liste des états finaux par `a1.final` et à la liste des transitions par `a1.trans`.

Expressions rationnelles

Soit Σ un alphabet. On définit la *syntaxe des expressions rationnelles* par :

— $\varnothing$, ε et a sont des expressions rationnelles, pour toute lettre $a \in \Sigma$;

— si E et F sont deux expressions rationnelles, alors $(E + F)$, $(E \cdot F)$ et $E^\star$ sont des expressions rationnelles.

La *sémantique des expressions rationnelles* est définie par l'application $\mathcal{L}$ qui associe à toute expression rationnelle un langage rationnel sur Σ par :

$$\begin{cases} \mathcal{L}(\varnothing) = \emptyset & \text{(langage vide)} \\ \mathcal{L}(\varepsilon) = \{\varepsilon\} & \text{(langage contenant le mot vide)} \\ \mathcal{L}(a) = \{a\} & \forall a \in \Sigma \end{cases}$$

et, si E et F sont deux expressions rationnelles,

$$\begin{cases} \mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F) \\ \mathcal{L}(E \cdot F) = \mathcal{L}(E) \cdot \mathcal{L}(F) \\ \mathcal{L}(E^\star) = \mathcal{L}(E)^\star \end{cases}$$

où $\star$ représente l'étoile de Kleene d'un langage et $\cdot$ représente la concaténation de deux langages.

Programmation

Le seul langage de programmation autorisé dans cette épreuve est Caml. Toutes les fonctions des modules `Array` et `List`, ainsi que les fonctions de la bibliothèque standard (celles qui s'écrivent sans nom de module, comme `max` ou `incr` ainsi que les opérateurs comme `/` ou `mod`) peuvent être librement utilisés.

Généralement, les objets mathématiques dans le texte seront notés A, m, i, n, ℓ, alors qu'ils seront représentés en Caml par `a`, `m`, `i`, `n`, `l`.

Les complexités demandées sont des complexités temporelles dans le pire des cas et seront exprimées sous la forme $O\big(f(n, m)\big)$, où f est une fonction usuelle simple et où n et m sont des paramètres correspondant aux tailles des objets en entrée de l'algorithme.

I Mots et automates

I.A – Miroir d'un mot et automate transposé

Pour tout mot $w = a_0 a_1 ... a_{n-1}$ de longueur $n \in \mathbb{N}^*$, on définit son *mot miroir* $\tilde{w}$ par $\tilde{w} = a_{n-1} ... a_1 a_0$. Par convention, le mot vide ε est son propre miroir.

Pour tout langage $L \subset \Sigma^\star$, on définit son *langage miroir* $\tilde{L}$ constitué de l'ensemble des mots miroirs du langage L :

$$\tilde{L} = \{\tilde{w} \mid w \in L\}.$$

Q 1. Décrire le langage L_1 de l'automate $\mathcal{A}_1$ de l'exemple de la figure 1 et décrire son langage miroir $\tilde{L}_1$.

Q 2. Dessiner un automate $\tilde{\mathcal{A}}_1$, reconnaissant le langage miroir $\tilde{L}_1$.

Soit $A = (Q, I, F, T)$ un automate non déterministe et $L = L_A$ le langage qu'il reconnait.

Q 3. Donner, en justifiant, la construction de l'automate miroir $\tilde{A} = (Q, I', F', T')$ qui reconnait le langage $\tilde{L}$.

Q 4. Écrire une fonction `transpose` de signature `automate -> automate` qui étant donné un automate A non déterministe en entrée, renvoie un automate non déterministe qui reconnait le miroir de L_A.

Q 5. Quelle est la complexité de cette fonction ?

I.B – Palindromes et rationalité

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

Q 6. Écrire une fonction `palindrome` de signature `string -> bool` qui teste, en temps linéaire, si un mot est un palindrome.

On rappelle que pour tout $0 \leqslant i <$ (`String.length s`), le i-ième caractère de la chaine de caractères `s` est obtenu par l'expression `s.[i]`.

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

Q 7. Montrer que si Σ est un alphabet à une lettre, alors $\mathrm{Pal}(\Sigma)$ est rationnel.

Q 8. Montrer que si Σ contient au moins deux lettres, alors $\mathrm{Pal}(\Sigma)$ n'est pas rationnel.

On pourra utiliser un automate et un mot de $\mathrm{Pal}(\Sigma) \cap a^\star b a^\star$.

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'.

Q 9. Montrer que $L_{q,q'}$ est reconnaissable et exprimer le langage L_A en fonction de langages $L_{q,q'}$.

Q 10. Montrer que $\mathrm{Pal}(\Sigma) \cap (\Sigma^2)^\star = \{u\tilde{u} \mid u \in \Sigma^\star\}$.

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 $R(L) = \{w \in \Sigma^\star \mid w\tilde{w} \in L\}$.

Q 11. Décrire simplement les langages $D(a^\star b)$ et $R(a^\star b^\star a^\star)$.

Q 12. Les langages $D(L)$ et $R(L)$ sont-ils reconnaissables ?

On pourra faire intervenir les langages $L_{q,q'}$, définis ci-dessus.

I.C – Déterminisation

On rappelle que pour tout automate $A = (Q, I, F, T)$ non déterministe, on peut définir l'automate déterminisé accessible $A_{\mathrm{det}} = (Y, \{I\}, F', \delta)$ où $Y \subset \mathcal{P}(Q)$ est l'ensemble des états accessibles depuis l'état initial $\{I\}$ dans l'automate des parties. Cet automate déterminisé accessible reconnait le même langage que l'automate A.

Q 13. Écrire un automate $\mathcal{A}_2$ non déterministe à 4 états qui reconnait le langage $L_2 = (b + ab)^\star ba$. Cet automate devra avoir un unique état initial et un unique état final.

Q 14. Appliquer l'algorithme de déterminisation sur l'automate miroir $\widetilde{\mathcal{A}_2}$ afin d'obtenir l'automate $\mathcal{A}_3 = \left(\widetilde{\mathcal{A}_2}\right)_{\mathrm{det}}$. Les états de $\mathcal{A}_3$ seront renommés $e_0, e_1, ...$

Q 15. Appliquer l'algorithme de déterminisation sur l'automate miroir $\widetilde{\mathcal{A}_3}$ afin d'obtenir l'automate $\mathcal{A}_4 = \left(\widetilde{\mathcal{A}_3}\right)_{\mathrm{det}}$. Ses états seront renommés $q_0, q_1, ...$

Q 16. Quel doit être le langage reconnu par l'automate $\mathcal{A}_4$?

On cherche à généraliser cette construction de façon effective. Pour cela, on va implémenter l'algorithme de déterminisation.

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.

Q 17. 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.

Q 18. 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

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

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 $\mathrm{numero}(X) = k$.

Q 19. 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.

Q 20. 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.

Q 21. 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.

Q 22. 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 $k_a = \text{numero}\big(\delta(X, a)\big)$ et $k_b = \text{numero}\big(\delta(X, b)\big)$.

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.

Q 23. 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).

Q 24. Écrire une fonction `determinise` de signature `automate -> automate` qui calcule le déterminisé accessible de l'automate d'entrée. On expliquera brièvement la démarche utilisée.

Q 25. 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} ?

I.D – Algorithme de Brzozowski

L'algorithme de Brzozowski permet d'obtenir un automate déterministe ayant un nombre minimal d'états, reconnaissant le même langage que l'automate initial.

On se donne un automate $A = (Q, I, \{f\}, T)$ qui reconnait le langage L et tel que l'automate miroir $\tilde{A}$ est déterministe et accessible.

On note $A_{\text{det}} = (Y, \{I\}, F, \delta)$ le déterminisé accessible de A.

Si u est un mot et L un langage, on note $u^{-1}L = \{w \in \Sigma^\star \mid uw \in L\}$.

Q 26. Soit $q \in Q$ un état et $u \in \Sigma^\star$ un mot. Montrer que si $q \in \delta^\star(\{I\}, u)$, alors il existe un mot $w \in \Sigma^\star$ tel que $uw \in L$.

Q 27. Montrer la propriété $(*)$: si l'on prend deux mots u et v dans $\Sigma^\star$ tels que $u^{-1}L = v^{-1}L$, alors dans l'automate A_{det} déterminisé, $\delta^\star(\{I\}, u) = \delta^\star(\{I\}, v)$.

Q 28. En déduire que si A est un automate quelconque reconnaissant L, alors en posant $B = \left(\tilde{A}\right)_{\text{det}}$, montrer que $\left(\tilde{B}\right)_{\text{det}}$ reconnait L et vérifie la propriété $(*)$.

Q 29. Écrire une fonction `minimal` de signature `automate -> automate` appliquant la construction de Brzozowski sur l'automate d'entrée. On fera abstraction de la taille des automates générés, possiblement problématique.

II Expression rationnelle associée à un automate

Dans cette partie, on introduit un algorithme, dû à Conway, pour le calcul de l'expression rationnelle associée au langage d'un automate, via l'utilisation de matrices dont les coefficients sont des expressions rationnelles.

II.A – Simplification d'expressions rationnelles équivalentes

On se donne en Caml le type `exprat` des expressions rationnelles

```
type exprat = Vide
            | Epsilon
            | Lettre of char
            | Union of exprat * exprat ;;
            | Concat of exprat * exprat
            | Etoile of exprat ;;
```

II.A.1)

Q 30. Écrire une fonction `lettre` de signature `exprat -> int` qui renvoie le nombre de lettres présentes dans l'expression rationnelle en argument. Par exemple, si $E = (a^\star b) + abba(a + \varepsilon)^\star + \varnothing$, `(lettre e)` doit renvoyer 7.

Q 31. Écrire une fonction `est_vide` de signature `exprat -> bool` qui teste si le langage rationnel représenté par l'expression rationnelle en argument est vide.

II.A.2) Dans cette section, on travaille formellement sur la syntaxe des expressions rationnelles.

On utilise les équivalences évidentes suivantes

$$\varnothing + E \equiv E + \varnothing \equiv E \qquad E \cdot \varepsilon \equiv \varepsilon \cdot E \equiv E \qquad E \cdot \varnothing \equiv \varnothing \cdot E \equiv \varnothing \qquad \varnothing^\star \equiv \varepsilon \qquad \varepsilon^\star \equiv \varepsilon \qquad (E^\star)^\star \equiv E^\star$$

où la notation $E \equiv E'$ signifie que les langages représentés sont égaux : $\mathcal{L}(E) = \mathcal{L}(E')$.

La fonction suivante réalise une simplification à la racine sur une expression du type `Union` en suivant la règle donnée.

```
let su expr = match expr with
   | Union( Vide , e ) -> e
   | Union( e , Vide ) -> e
   | _ -> expr;;
```

De même, on peut écrire une fonction `sc : exprat->exprat` qui simplifie à la racine une expression de type `Concat`. On suppose codées ces fonctions.

Q 32. Écrire une fonction `se : exprat -> exprat` qui simplifie à la racine une expression de type `Etoile` avec les règles données.

Prenons par exemple $E_n = (a + (b.(b.(b.(b....\varnothing)...)$, où n lettres b concaténées se succèdent.

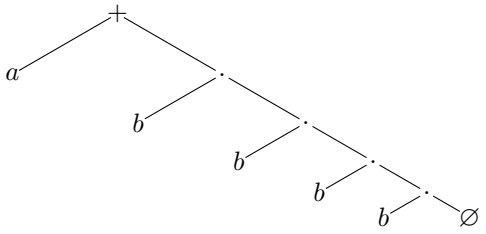


Figure 2 Exemple de l'arbre syntaxique de l'expression E_4

Q 33. Combien d'applications de règles décrites ci-dessus sont-elles nécessaires pour obtenir à partir de E_n l'expression équivalente a ?

Q 34. Écrire une fonction `simplifie : exprat -> exprat` qui simplifie une expression rationnelle selon les règles données.

II.B – Matrices d'expressions rationnelles

Dans la suite, on considère des matrices d'expressions rationnelles

```
type mat = exprat array array ;;
```

La *matrice nulle* de taille n est la matrice de taille n où chaque coefficient vaut $\varnothing$.

La *matrice identité* de taille n est la matrice de taille n où chaque coefficient vaut ε sur la diagonale et $\varnothing$ en dehors de la diagonale.

On définit la *somme de deux matrices* A et B de taille $(n \times m)$ par la matrice $A + B$ de taille $(n \times m)$ où $[A + B]_{i,j} = A_{i,j} + B_{i,j}$ où le $+$ représente l'opération rationnelle d'union et $0 \leqslant i \leqslant n-1$, $0 \leqslant j \leqslant m-1$.

On définit le *produit de deux matrices* A et B de taille $(n \times p)$ et $(p \times q)$ à la manière du produit matriciel usuel AB, de taille $(n \times q)$, où la somme de coefficients est remplacée par l'union et où le produit de coefficients est remplacé par la concaténation des expressions rationnelles.

II.B.1)

Q 35. Écrire une fonction `somme` de signature `mat -> mat -> mat` effectuant la somme de deux matrices d'expressions rationnelles de même taille $(n \times p)$. Quelle est sa complexité ?

Q 36. Écrire une fonction `produit` de signature `mat -> mat -> mat` effectuant le produit de deux matrices d'expressions rationnelles, en supposant que les tailles sont bien compatibles (la première de taille $(n \times p)$ et la seconde de taille $(p \times q)$.) On ne vérifiera pas la compatibilité des tailles. Quelle est sa complexité ?

On cherche désormais à définir l'étoile de Kleene d'une matrice carrée d'expressions rationnelles.

II.B.2) Étude de l'étoile d'une matrice de taille 2

Plaçons-nous dans le cas d'une matrice carrée de taille 2, $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, où a, b, c et d sont quatre lettres.

On associe à cette matrice M le graphe étiqueté à deux sommets de la figure 3.

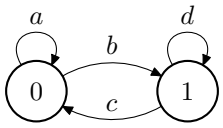


Figure 3

On note $L_{i,j}$ le langage de l'automate $\mathcal{A}_{i,j} = (\{0,1\}, \{i\}, \{j\}, T)$, où $T = \{(i, M_{i,j}, j) \mid (i,j) \in \{0,1\}^2\}$.

Q 37. Donner une expression rationnelle sur l'alphabet $\{a, b, c, d\}$ pour décrire chaque langage $L_{i,j}$.

II.B.3) Étoile d'une matrice carrée de taille quelconque

Pour la définition de l'étoile d'une matrice carrée d'expressions rationnelles, on procède récursivement, sur la taille n de la matrice carrée M :

— si la taille vaut 1, $M = (e)$, donc $M^\star = (e^\star)$;

— sinon, pour M de taille $n \geqslant 2$, on découpe M par blocs

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

où A et D sont carrées de taille $\geqslant 1$, et on définit

$$M^\star = \begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix}$$

où

$$A' = (A + BD^\star C)^\star \qquad B' = A^\star B(D + CA^\star B)^\star \qquad C' = D^\star C(A + BD^\star C)^\star \qquad D' = (D + CA^\star B)^\star$$

On suppose déjà codées les deux fonctions suivantes sur les matrices d'expressions rationnelles :

— `(decouper m n1 n2)` renvoie quatre matrices blocs A, B, C et D telles que A est carrée de taille n_1, D est carrée de taille n_2 et $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, où la matrice d'entrée M est carrée de taille $n = n_1 + n_2$.

```
decouper : mat -> int -> int -> (mat*mat*mat*mat)
```

— `(recoller a b c d)` renvoie la matrice $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ à partir des quatre blocs A, B, C et D codés par `a, b, c, d` dont les tailles sont compatibles.

```
recoller : mat -> mat -> mat -> mat -> mat
```

On propose la décomposition récursive suivante pour une matrice M carrée de taille $n \geqslant 2$,

$$M = \begin{pmatrix} a & B \\ C & D \end{pmatrix}$$

où a est une lettre et D est carrée de taille $n-1$. Ainsi,

$$M^\star = \begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix}$$

où

$$A' = (a + BD^\star C)^\star \qquad B' = a^\star B(D + Ca^\star B)^\star \qquad C' = D^\star C(a + BD^\star C)^\star \qquad D' = (D + Ca^\star B)^\star$$

Q 38. Évaluer les différentes complexités des sommes et produits effectués, et en déduire que si $C(n)$ est la complexité du calcul de l'étoile pour une matrice de taille n, alors $C(n) = 2C(n-1) + O(n^2)$. En déduire la complexité de cet algorithme.

On se place dans le cas où la taille n de la matrice M est une puissance de 2, avec $n \geqslant 2$.

On propose désormais la décomposition récursive suivante

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

où les matrices A et D sont carrées de taille $n/2$ chacune.

Q 39. Évaluer les différentes complexités des sommes et produits effectués, et en déduire que si $C(n)$ est la complexité du calcul de l'étoile pour une matrice de taille n, alors $C(n) = 4C(n/2) + O(n^3)$. En déduire la complexité de cet algorithme.

Q 40. Comment gérer le cas des matrices M de taille n quelconque ? Quelle complexité peut-on obtenir pour le calcul de $M^\star$?

Q 41. Écrire la fonction `etoile` de signature `mat -> mat` qui renvoie l'étoile d'une matrice en utilisant l'algorithme récursif le plus adéquat.

II.C – Algorithme de Conway

Soit $A = (Q, I, F, T)$ un automate, où l'ensemble d'états est $Q = [\![0, n-1]\!]$.

On définit M_A la *matrice de transition* de l'automate A par la matrice d'expressions rationnelles de taille $(n \times n)$ telle que pour $0 \leqslant i, j \leqslant n-1$, $[M_A]_{i,j} = \sum_{c \in \Sigma | (i,c,j) \in T} c$ s'il existe au moins une telle lettre c, $\varnothing$ sinon.

On admet la propriété suivante : pour tout état $(i, j) \in Q^2$, $\mathcal{L}([M_A^\star]_{i,j}) = L_{i,j}$, où $L_{i,j}$ est le langage défini en question 9.

Q 42. Montrer que

$$L_A = \mathcal{L}([X\, M_A^\star\, Y]_{0,0})$$

où X est une matrice ligne d'expressions rationnelles de la forme $X = \begin{pmatrix} x_0 & \cdots & x_{n-1} \end{pmatrix}$ où chaque $x_i \in \{\varnothing, \varepsilon\}$, et Y est une matrice colonne d'expressions rationnelles de la forme $Y = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}$ où chaque $y_j \in \{\varnothing, \varepsilon\}$. On précisera les valeurs de X et Y en fonction de l'automate A.

Q 43. Écrire la fonction `langage` de signature `automate -> exprat` prenant en entrée un automate et renvoyant une expression rationnelle représentant le langage de cet automate. Quelle est la complexité de cette fonction ?

III Automate des dérivées d'Antimirov

On propose pour conclure 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

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

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 $\varepsilon \in \mathcal{L}(a^\star)$,

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

Q 44. 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,

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

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

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

On rappelle la notation $w^{-1}L$ de la partie I : pour tout mot $w \in \Sigma^\star$ et tout langage $L \subset \Sigma^\star$,

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

Q 45. Dessiner l'automate obtenu à partir de l'expression rationnelle $E = (ab + b)^\star ba$. On indiquera précisément l'ensemble d'états Q.

Q 46. 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,

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