

Les énoncés des travaux pratiques seront disponibles sur internet quelques jours avant la séance correspondante, à l'adresse indiquée ci-dessus. Les questions en rapport avec les TP, Caml Light ou l'informatique sont les bienvenues et peuvent être envoyées à [victor.nicollet@ens.fr](mailto:victor.nicollet@ens.fr)

L'algorithme de Thompson permet de construire un automate à partir d'une expression rationnelle de manière automatique. Dans ce TP, on choisit de représenter les expressions rationnelles à l'aide du type `regexp` donné ci-contre. À titre d'échauffement, et pour réviser les concepts d'expressions rationnelles d'il y a deux semaines, on se donne les deux expressions rationnelles suivantes:

```
type regexp =
| Mot of string
| Ou of regexp * regexp
| Et of regexp * regexp
| Etoile of regexp;;
```

$$E_1 = (ba|a)^*(b|\varepsilon) \quad E_2 = (a|b)^*(ab|ba)$$

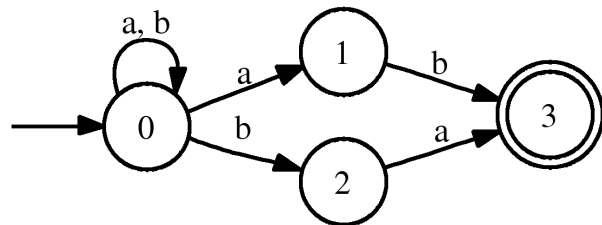
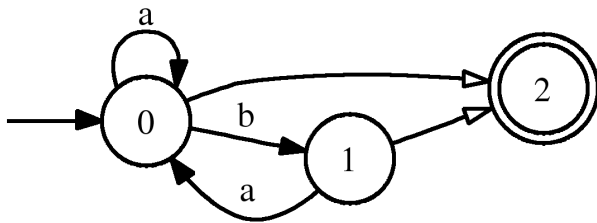
▷ **Question 1.** Quels sont les langages reconnus par les expressions rationnelles  $E_1$  et  $E_2$  ? Représenter ces expressions à l'aide du type `regexp` donné précédemment. ◀

## 1 CONSTRUCTION

On représente les automates non-déterministes à  $\varepsilon$ -transitions par le type `auto_ndet` défini ci-contre, avec la convention que l'état 0 est l'état initial, et que l'état `a.etats - 1` est l'unique état final de l'automate. On représente l'ensemble complet des transitions et  $\varepsilon$ -transitions par des listes, plutôt que de constituer une liste pour chaque état.

```
type auto_ndet = {
  etats : int;
  char : (char * int * int) list;
  epsilon : (int * int) list;
};;
```

▷ **Question 2.** Représenter les automates non-déterministes à  $\varepsilon$ -transitions donnés ci-dessous à l'aide du type `auto_ndet`. ◀



L'algorithme de Thompson consiste à construire un automate pour chaque sous-expression d'une expression rationnelle donnée, puis de combiner ces automates en un seul automate représentant l'expression rationnelle dans son ensemble. Par exemple, la concaténation de deux expressions rationnelles  $\alpha\beta$  peut se transformer récursivement en automate non-déterministe à  $\varepsilon$ -transitions en transformant  $\alpha$  en un automate  $A(\alpha)$ ,  $\beta$  en un automate  $A(\beta)$ , et en reliant l'état final de  $A(\alpha)$  à l'état initial de  $A(\beta)$  par une  $\varepsilon$ -transition (et rendant au passage l'état final de  $A(\alpha)$  non-final).

▷ **Question 3.** Écrire deux fonctions d'aide, `decaler auto n` et `fusionner n a b liens`.

La fonction `decaler` crée un nouvel automate à partir de `auto`, de taille `auto.etats + n`, et décale ensuite toutes les transitions de `n` états (ainsi, si une transition partait ou arrivait sur l'état  $i$ , elle part ou arrive désormais sur l'état  $i + n$ ).

La fonction `fusionner` construit un automate de taille `n` vierge, puis y insère toutes les transitions des automates `a` et `b` (ce qui revient à combiner les automates en question au sein d'un même automate).

plus grand) puis ajoute liens à la liste des  $\varepsilon$ -transitions de cet automate (pour relier, par exemple, un état final à un état initial).

En déduire des fonctions `et a b`, ou `a b` et `etoile a` construisant les automates de concaténation, d'alternative et d'étoile à partir de sous-automates `a` et `b`. ◀

Une fois qu'il est possible de combiner des automates, on peut passer à l'étape suivante: traverser récursivement l'expression rationnelle pour construire l'automate complet la représentant.

▷ **Question 4.** Écrire une fonction `transforme e` qui renvoie un automate non-déterministe à  $\varepsilon$ -transitions qui reconnaît le même langage que l'expression rationnelle `e`. On pourra, pour s'aider, commencer par écrire une fonction `mot s` qui renvoie un automate reconnaissant le langage  $\{s\}$  constitué du seul mot `s`. ◀

## 2 RECONNAISSANCE

---

On souhaite maintenant écrire un algorithme de reconnaissance d'une expression rationnelle. Pour cela, on transforme cette expression en un algorithme non-déterministe à  $\varepsilon$ -transitions par l'algorithme de Thompson, et on utilise cet automate pour reconnaître des mots.

On choisit de représenter l'ensemble des états dans lesquels se trouve l'automate à un moment donné par une liste triée d'entiers. On utilisera, pour ajouter un élément à une liste, la fonction `insere` donnée ci-contre, qui permet de conserver l'ordre de la liste tout en évitant les doublons.

```
let rec insere e = fonction
| [] -> [e]
| t::q ->
  if t = e then t :: q
  else if t < e then t :: insere e q
  else e :: t :: q;;
```

▷ **Question 5.** Écrire une fonction `applique_transition c etats transitions` qui, en supposant que l'automate se trouve dans les états `etats` et lit une lettre `c`, calcule le nouvel ensemble d'états à partir des transitifs de l'automate.

Écrire une fonction `applique_epsilon etats transitions` qui, en supposant que l'automate se trouve dans les états `etats`, calcule le nouvel ensemble d'états (contenant le précédent) qui peut être atteint à partir de cette position initial en suivant des  $\varepsilon$ -transitions. Attention, c'est moins évident qu'il n'y paraît.

En déduire une fonction `reconnaitre auto mot` qui renvoie vrai si et seulement si le mot fourni est reconnu par l'automate fourni. ◀

## 3 QUESTION DIFFICILE — DÉTERMINISATION

---

On reprend maintenant la définition d'un automate déterministe telle qu'elle a été fournie au TP précédent, et on se fixe pour objectif de construire un automate déterministe à partir d'une expression rationnelle. Puisqu'un dispose déjà, grâce à l'algorithme de Thompson, d'un automate non-déterministe à  $\varepsilon$ -transitions, on peut se contenter de déterminer celui-ci.

```
type automate = {
  transitions : (char * int) list vect;
  initial : int;
  final : bool vect;
};;
```

Pour déterminer l'automate, on considère l'ensemble des sous-ensembles d'états de l'automate non-déterministe, chacun de ces sous-ensembles étant vu comme un état de l'automate déterministe. On calcule les transitions entre eux suivant la règle que si l'automate non-déterministe, dans les états  $A$ , passe dans les états  $B$  après lecture du caractère  $c$  (et application des  $\varepsilon$ -transitions), alors  $A \xrightarrow{c} B$  dans l'automate déterministe. Un état est final si l'ensemble correspondant contient l'état final  $n - 1$ . Pour des raisons de performances, on ne choisit que les états qui sont accessibles à partir de l'état initial.

▷ **Question 6.** Choisir des structures de données pour représenter l'automate durant sa construction, et programmer l'algorithme de détermination. ◀