DM5

L'*analyse lexicale* est la première étape effectuée par les outils d'analyse de texte et, en particulier, de programmes (compilateur, interprète, ...). Elle vise à reconnaître les différents mots composant une phrase sans s'attacher à la structure de la phrase elle-même. La technique la plus couramment employée pour implanter un analyseur lexical repose sur un *automate fini* dont les transitions sont étiquetées par les caractères composant les mots du vocabulaire. Les états initiaux correspondent aux débuts des mots. Les états terminaux correspondent à la fin des mots. Un mot est reconnu par un parcours de l'automate guidé par les caractères composant ce mot.

Les automates finis peuvent être *déterministes* ou *indéterministes*. Ces deux formes présentent chacune des avantages et des inconvénients au niveau des performances :

- le parcours d'un automate déterministe est peu coûteux alors que son occupation mémoire est coûteuse ;
- l'occupation mémoire d'un automate indéterministe est peu coûteuse alors que son parcours est coûteux.

L'outil `lex`, utilisé pour construire des analyseurs de programmes, exploite des *automates semi-indéterministes* pour obtenir un bon compromis entre les coûts du parcours et de l'occupation mémoire. Ce sujet étudie une catégorie d'automates semi-indéterministes, ses propriétés, ainsi qu'une technique de construction.

Soit Σ un alphabet et $\varepsilon \notin \Sigma$ le mot vide. L'ensemble des mots sur Σ est noté Σ^*. Un *automate fini* sur Σ est un quintuplet $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ composé d'un ensemble fini d'états Q, d'un ensemble d'états initiaux I, d'un ensemble d'états acceptants F, d'une relation de transition $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$. δ^* désigne l'extension de δ à $Q \times \Sigma^* \times Q$, définie par :

- $\forall q \in Q, (q, \varepsilon, q) \in \delta^*$;
- $\forall x \in \Sigma \cup \{\varepsilon\}, \forall m \in \Sigma^*, \forall (i, f) \in \Sigma^* \times \Sigma^*, (i, xm, f) \in \delta^* \iff \exists q \in Q, ((i, x, q) \in \delta) \wedge ((q, m, f) \in \delta^*)$.

Dans la suite, les fonctions sont codées en langage OCaml. Pour les besoins de codage, on suppose que tous les automates exploitent le même ensemble fini de symboles $\Sigma \subset \mathbb{N}$. Pour les variables ou les constantes dont les noms sont pris dans l'alphabet grec (δ, ε ...), l'identifiant OCaml sera leur nom en toute lettre (`delta`, `epsilon`, ...). Pour représenter les automates finis en OCaml, on doit disposer d'un codage de la structure d'ensemble. On adopte le choix de codage simple suivant : un ensemble est une liste contenant exactement un exemplaire de chaque valeur contenue dans l'ensemble. Les opérations de manipulation d'un ensemble devront préserver cette propriété. Le parcours d'un ensemble sera donc effectué de la même manière que celui d'une liste. On donne les types suivants.

```
type etat = int                                    (* état *)
type etats = etat list                             (* ensemble d'états *)
type etiquette = int                               (* étiquette de transition *)
let epsilon = -1                                   (* étiquette de epsilon *)
type transition = etat * etiquette * etat          (* transition *)
type relation = transition list                    (* relation *)
type paire = relation * relation                   (* paire de relations *)
type automate = etats * etats * etats * relation   (* automate *)
type mot = etiquette list                          (* mot *)
```

Question 1. On considère l'automate représenté par le graphe suivant. Quelle expression régulière dénote le langage reconnu par cet automate ?

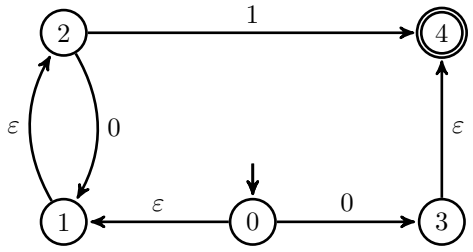

Figure 1 – Exemple d'automate

Question 2. Définir les identifiants OCaml `etatsQ`, `etatsI`, `etatsF`, `delta` associés à l'automate de la figure 1.

Opérations sur la structure d'ensemble

Question 3.

□ **3.1.** Écrire une fonction `cadinal : etats -> int` telle que l'appel `(cardinal e)` renvoie le nombre d'états dans l'ensemble `e`. L'utilisation de la fonction `String.length` n'est pas autorisée.

□ **3.2.** Estimer sa complexité en fonction de la taille de son ensemble argument.

Question 4.
□ **4.1.** Écrire une fonction `appartient : etat -> etats -> boolean` telle que l'appel `(appartient q e)` renvoie `true` si l'ensemble `e` contient l'état `q`. L'utilisation de la fonction `List.mem` n'est pas autorisée.
□ **4.2.** Estimer sa complexité en fonction de la taille de son ensemble argument.

Question 5. Écrire une fonction `ajout : etat -> etats -> etats` telle que l'appel `(ajout q e)` renvoie les mêmes états que l'ensemble `e` ainsi que l'état `q` si ce dernier ne figurait pas dans `e`. L'ensemble renvoyé contiendra exactement une fois l'état `q`.

Question 6.
□ **6.1.** Écrire une fonction `egalite : etats -> etats -> boolean` telle que l'appel `(egalite e1 e2)` renvoie `true` si les deux ensembles `e1` et `e2` contiennent exactement les mêmes états.
□ **6.2.** Estimer sa complexité en fonction de la taille des deux ensembles arguments.

Question 7. Écrire une fonction `union : etats -> etats -> etats` telle que l'appel `(union e1 e2)` renvoie un ensemble contenant les états de `e1` et ceux de `e2`, sans doublons.

Question 8. Écrire une fonction `intersection : etats -> etats -> etats` telle que l'appel `(intersection e1 e2)` renvoie un ensemble contenant les états présents à la fois dans `e1` et dans `e2`.

Exploitation des relations de transition

La construction d'automates semi-indéterministes repose sur la manipulation des relations de transition et, en particulier, sur la construction de *fermetures transitives*. Nous allons définir des opérations de parcours des relations qui pourront être utilisées par la suite.

Question 9. Les *suivants* $\mathrm{s}(E, \delta)$ d'un ensemble E d'états selon une relation δ sont les ensembles des destinations des transitions de δ dont les états de E sont les origines, indépendamment des étiquettes.

$$\mathrm{s}(E, \delta) = \{q' \in Q \mid \exists q \in E, \exists x \in \Sigma \cup \{\varepsilon\}, (q, x, q') \in \delta\}$$

□ **9.1.** Quels sont les suivants des ensembles $\{0\}$, $\{2\}$, $\{4\}$, $\{1, 3\}$ construits à partir de l'automate de la figure 1 ?
□ **9.2.** Écrire une fonction `suivants : etats -> relation -> etats` telle que l'appel `(suivant e delta)` renvoie un ensemble d'états contenant les mêmes états que $\mathrm{s}(\mathrm{e}, \delta)$.

Question 10. L'ensemble des *états accessibles* $\mathrm{a}(E, \delta)$ depuis les états de E selon la relation δ est défini par :

$$\mathrm{a}(E, \delta) = \{q' \in Q \mid \exists q \in E, \exists m \in \Sigma^*, (q, m, q') \in \delta^*\}$$

□ **10.1.** Quels sont les états accessibles depuis les ensembles $\{0\}$, $\{2\}$, $\{4\}$, $\{1, 3\}$ construits à partir de l'automate de la figure 1 ?
□ **10.2.** Soit $E \subseteq Q$ un ensemble d'états. On définit la suite (A_i) d'ensembles d'états $A_i \subseteq Q$ par $A_0 = E$ et $A_{i+1} = A_i \cup \mathrm{s}(A_i, \delta)$.
▷ **10.2.1.** Montrer que la suite (A_i) est croissante pour la relation d'inclusion $\subseteq$.
▷ **10.2.2.** Montrer qu'il existe un entier $k \in \mathbb{N}$ tel que $A_k = A_{k+1}$.
▷ **10.2.3.** Montrer que : $\forall i \in \mathbb{N}$, $\mathrm{a}(A_i, \delta) = \mathrm{a}(A_{i+1}, \delta)$.
▷ **10.2.4.** Montrer que : $\forall E \subseteq Q$, $\mathrm{s}(E, \delta) \subseteq E \implies \mathrm{a}(E, \delta) \subseteq E$.
▷ **10.2.5.** Montrer que $\mathrm{a}(E, \delta) = A_k$.
□ **10.3.** Écrire une fonction `access : etats -> relation -> etats` telle que l'appel `(access e delta)` renvoie un ensemble d'états contenant les mêmes états que $\mathrm{a}(\mathrm{e}, \delta)$.
□ **10.4.** Les opérations de construction des automates semi-indéterministes étudiées par la suite engendrent de nouvelles transitions. Pour simplifier leurs définitions, on utilise des fonctions de construction de familles de transitions similaires ; par exemple, de transitions ayant la même origine, la même étiquette et plusieurs destinations distinctes.
Écrire une fonction `prefixe : etat -> etiquette -> etats -> relation` telle que l'appel `(prefixe q x e)` renvoie une relation contenant les transitisions d'origine `q`, d'étiquette `x` et de destination les états contenus dans `e`.

Automate fini semi-indéterministe

En général, dans le cadre d'un automate fini, un même mot du langage accepté par un automate peut être reconnu par plusieurs parcours distincts de ce dernier. L'algorithme naturel de parcours de l'automate pour analyser un mot est donc indéterministe car il existe plusieurs possibilités de parcours.

Trois causes d'indéterminisme peuvent apparaître dans un automate fini :

- l'existence de plusieurs états initiaux ;
- l'existence de plusieurs transitions avec une même étiquette et une même origine ;

- l'existence de transitions arbitraires (d'étiquette ε).

Les algorithmes indéterministes de parcours étant souvent plus coûteux, il est pertinent d'envisager la déterminisation de l'automate d'analyse. La construction d'un automate déterministe reconnaissant le même langage qu'un automate indéterministe donné peut faire apparaître un grand nombre de nouveaux états. L'occupation mémoire est donc souvent beaucoup plus coûteuse. Dans un objectif de compromis entre occupation mémoire et temps de parcours, on s'intéresse à des automates semi-indéterministes qui ne font pas apparaître la troisième forme d'indéterminisme. On étudie par la suite un algorithme de construction qui n'introduit pas de nouveaux états.

On définit une restriction des automates finis sans ε-transitions. Ces automates sont appelés *semi-indéterministes* car un degré d'indéterminisme est levé. Les définitions précédentes sont toujours valides en éliminant les ε-transitions. Plus formellement, si $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ est un automate fini, $\mathcal{A}$ est *semi-indéterministe* si et seulement si $\delta \subseteq Q \times \Sigma \times Q$.

Question 11. On décompose la relation de transition d'un automate fini quelconque en une *relation arbitraire* et une *relation non-arbitraire* qu'on recompose ensuite pour obtenir un automate semi-indéterministe. Soient Q un ensemble d'états.

- Une *relation de transition arbitraire* (ou ε-transition), notée δ_ε, est une relation dont toutes les transitions sont étiquetées par ε : $\delta_\varepsilon \subseteq Q \times \{\varepsilon\} \times Q$.
- Une *relation de transition non-arbitraire*, notée δ_Σ, est une relation dont toutes les transitions sont étiquetées par des symboles de Σ, donc différents de ε : $\delta_\Sigma \subseteq Q \times \Sigma \times Q$.

Si $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ est un automate fini, alors la relation δ peut donc se décomposer de manière unique en deux sous-relations disjointes δ_ε, relation arbitraire, et δ_Σ, une relation non-arbitraire.

$$\delta = \delta_\varepsilon \cup \delta_\Sigma$$

□ **11.1.** Quelles sont les relations arbitraire δ_ε et non-arbitraire δ_Σ associées à l'automate de la figure 1.
□ **11.2.** Écrire une fonction `decompose : relation -> paire` telle que `(decompose delta)` renvoie une paire contenant les deux relations δ_ε et δ_Σ.

Question 12. La *fermeture transitive* de δ_ε, notée $\bar{\delta}_\varepsilon$, est définie par :

$$\begin{cases} \forall q \in Q, (q, \varepsilon, q) \in \bar{\delta}_\varepsilon \\ \forall q \in Q, \forall q' \in Q, (q, \varepsilon, q') \in \bar{\delta}_\varepsilon \iff \exists q'' \in Q, ((q, \varepsilon, q'') \in \delta_\varepsilon) \wedge \left((q'', \varepsilon, q') \in \bar{\delta}_\varepsilon\right) \end{cases}$$

$\bar{\delta}_\varepsilon$ est la plus petite relation vérifiant la relation donnée.
□ **12.1.** Construire $\bar{\delta}_\varepsilon$ à partir de la relation δ_ε obtenue à la question 11.1.
□ **12.2.** Calculer un majorant de la taille de $\bar{\delta}_\varepsilon$ en fonction de la taille de δ_ε.
□ **12.3.** Montrer que :

$$\bar{\delta}_\varepsilon = \{(q, \varepsilon, q') | q \in Q, q' \in \mathrm{a}(\{q\}, \delta_\varepsilon)\}$$

□ **12.4.** Écrire une fonction `fermeture : relation -> relation` telle que l'appel `(fermeture delta)` renvoie une relation contenant toutes les transitions engendrées par la fermeture transitive $\bar{\delta}$ des transitions de la relation arbitraire δ.

Question 13. Soient δ_ε une relation de transition arbitraire, δ_Σ une relation de transition non-arbitraire. L'*opérateur de composition*, noté $\triangleright$ est défini par :

$$\delta_\Sigma \triangleright \delta_\varepsilon = \{(q, x, q') | \exists q'' \in Q, (q, x, q'') \in \delta_\Sigma, (q'', \varepsilon, q') \in \delta_\varepsilon\}$$

□ **13.1.** Calculer un majorant de la taille de la relation $\delta_\Sigma \triangleright \delta_\varepsilon$ en fonction des tailles de δ_Σ et de δ_ε.
□ **13.2.** Construire la composition $\delta_\Sigma \triangleright \bar{\delta}_\varepsilon$ de la relation δ_Σ et de la fermeture transitive de la relation δ_ε données dans la question 11.1.
□ **13.3.** Écrire une fonction `compose : relation -> relation -> relation` telle que l'appel `(compose delta1 delta2)` renvoie une relation résultant de la composition $\delta_1 \triangleright \delta_2$ de la relation non-arbitraire δ_1 et de la relation arbitraire δ_2. On suppose que δ_1 ne contient que des transitions non-arbitraires et que δ_2 ne contient que des transitions arbitraires.

Question 14. Soit l'automate fini $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ avec $\delta = \delta_\Sigma \cup \delta_\varepsilon$. L'*automate semi-indéterministe* associé $\mathcal{E}(\mathcal{A})$ est défini par :

$$\mathcal{E}(\mathcal{A}) = \left(Q, \Sigma, I \cup \mathrm{s}(I, \bar{\delta}_\varepsilon), F, \delta_\Sigma \cup (\delta_\Sigma \triangleright \bar{\delta}_\varepsilon)\right)$$

□ **14.1.** Montrer que l'automate semi-indéterministe associé reconnaît le même langage que l'automate initial, c'est-à-dire que $L(\mathcal{A}) = L(\mathcal{E}(\mathcal{A}))$.
□ **14.2.** Calculer un majorant de la taille de l'automate semi-indéterministe associé en fonction de la taille de l'automate initial. Comparer cette valeur avec celle de l'automate résultant de la déterminisation de l'automate initial.
□ **14.3.** Donner la représentation graphique de l'automate semi-indéterministe associé à l'automate de la figure 1.
□ **14.4.** Écrire une fonction `semi : automate -> automate` telle que l'appel `(semi automateA)` renvoie l'automate semi-indéterministe associé à l'automate `automateA`.