Automates

Petit préliminaire mathématiques sur les relations binaires

Dans cette partie préliminaire on conidère deux ensembles A et B et une relation $\mathcal{R}$ de A dans B, c'est-à-dire une partie de $A \times B$.

Pour rappel, on note $x\mathcal{R}y$ pour $(x, y) \in \mathcal{R}$.

Définition 1 : Relation fonctionnelle, totale

- La relation $\mathcal{R}$ est dite fonctionnelle lorsque, pour tout $x \in A$, il existe au plus un $y \in B$ tel que $x\mathcal{R}y$.
- La relation $\mathcal{R}$ est dite totale lorsque, pour tout $x \in A$, il existe au moins un $y \in B$ tel que $x\mathcal{R}y$.

Comment caractériser une relation fonctionnelle et totale ? Seulement fonctionnelle ?

I Automates finis déterministes

I.1 Définition

Définition 2 : Automate fini déterministe

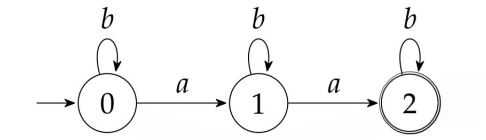
On appelle automate (fini) déterministe un quintuplet $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$, où :

- Σ est un alphabet ;
- Q est un ensemble fini, qu'on appelle ensemble des états
- q_0 est un élément remarquable de Q qu'on appelle l'état initial
- F est un sous-ensemble de Q qu'on appelle ensemble des états terminaux
- δ une relation fonctionnelle de $Q \times \Sigma$ dans Q.

Pour y comprendre quelque chose, il faut faire des dessins.

Un automate fini déterministe peut se représenter comme un graphe orienté, mais dont les arcs sont étiquettés et qui peut présenter plusieurs arcs d'un sommet vers un autre. Plus précisément : les sommets correspondent aux états (certains sont distingués et marqués comme initial ou terminaux) et les arcs correspondent aux transitions, étiquetés par des symboles de Σ (on a un arc étiquetté par a entre q_1 et q_2 si et seulement si on a $(q_1, a)\,\delta\,q_2$, ce qu'on notera $q_1 \xrightarrow[(\delta)]{a} q_2$.

Exemple 1 : Je représente un automate fini déterministe qu'on appellera dans la suite $\mathcal{A}_1$: on annote le dessin en spécifiant Σ, Q, q_0, F, δ, et en précisant si δ est ou pas totale.



Exemples 2 : Proposez-moi deux autres exemples qu'on appelera $\mathcal{A}_2$ et $\mathcal{A}_3$.

Implémentation : on peut définir le type afd des automates finis déterministes par :

```
type afd = {initial : int ;
terminaux : int list ;
transitions : int*char -> int};;
```

où la relation de transition (puisqu'elle est fonctionnelle) est représentée par une fonction, en convenant (puisqu'elle n'est pas nécessairement totale) qu'elle provoque l'exception `Not_found` là où elle n'est pas définie.

On pourrait aussi utiliser le module `Hashtbl`, on se le garde sous le coude pour les automates non-déterministes.

Application 1 : Implémenter $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$.

I.2 Langage reconnu par un automate fini déterministe

Définition 3 : Langage reconnu par un automate

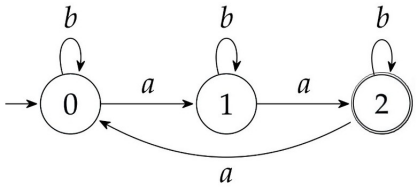
• Soit $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ un automate. On appelle chemin dans $\mathcal{A}$, ou calcul dans $\mathcal{A}$, une suite finie de transitions consécutives : $e_0 \xrightarrow[(\delta)]{a_0} e_1 \xrightarrow[(\delta)]{a_1} \cdots \xrightarrow[(\delta)]{a_{n-1}} e_n$ où $e_0, e_1, \ldots, e_n \in Q$ et $a_1, a_2, \ldots, a_n \in \Sigma$.
On le note $e_0 \xrightarrow[(\delta^*)]{a_0 a_1 \cdots a_{n-1}} e_n$.

• Soit $u \in \Sigma^*$. S'il existe un calcul dans $\mathcal{A}$ de la forme $q_0 \xrightarrow[(\delta^*)]{u} q_f$ avec q_0 l'état initial et q_f un état terminal, alors on dit que le mot u est accepté par $\mathcal{A}$.

• L'ensemble des mots acceptés par $\mathcal{A}$ est appellé le langage reconnu par l'automate $\mathcal{A}$. On le note $\mathcal{L}(\mathcal{A})$.

Exemples 3 : Essayons de décrire les langages reconnus par $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$.

Exemple 4 : Quel est le langage reconnu par l'automate $\mathcal{A}_4$ suivant :



Implémentation : Écrire une fonction `reconnu_par_afd : string → afd → bool` qui détermine si un mot (codé ici par une chaîne de caractères) est reconnu ou pas par un afd.
On commencera par écrire pour cela une fonction auxiliaire locale `accepte` qui indique si on peut accéder à un état terminal en lisant un mot **à partir d'un état donné** (pas nécessairement l'état initial).
Il semble adapté de rattraper les exceptions pour gérér facilement le fait que la relation de transition n'est pas totale.

Ok donc là il semble que les langages reconnnus par des automates finis déterministes sont tous rationnels. Mais les automates finis déterministes, c'est trop simple, donc faisons plus compliqué.

II Automates finis non-déterministes avec ε-transitions

II.1 Définition

Définition 4 : Automate fini non-déterministe avec ε-transitions

On appelle automate (fini) non-déterministe avec ε-transition un quintuplet $\mathcal{A} = (\Sigma, Q, I, F, \delta)$, où :

- Σ est un alphabet ;
- Q est un ensemble fini, qu'on appelle ensemble des états ;
- I est un sous-ensemble de Q qu'on appelle ensemble des états initiaux ;
- F est un sous-ensemble de Q qu'on appelle ensemble des états terminaux ;
- δ une relation quelconque de $Q \times \big(\Sigma \cup \{\varepsilon\}\big)$ dans Q.

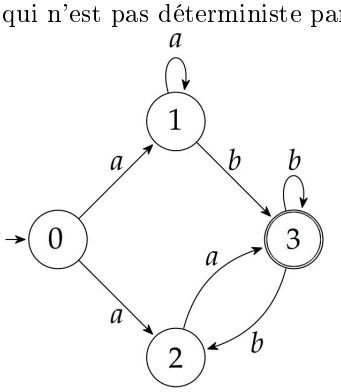
Qu'est-ce qui a changé ?

1. On s'autorise à avoir plusieurs états initiaux.
2. La relation de transition n'est plus nécessairement fonctionnelle : on peut, depuis un même état, en lisant une même lettre, transitionner potentiellement vers plusieurs états différents.
3. On s'autorise à étiquetter une transition par ε et plus seulement par une lettre de Σ.

(On a une notion intermédiaire : les automates finis non-déterministes mais sans ε-transitions.)

C'est donc une **généralisation** de la notion précédente : on peut voir les automates finis déterministes comme des cas particuliers d'automates finis non-déterministes avec ε-transitions (c'est le cas particulier où on a un unique état initial, où on n'a aucune ε-transition, et où la relation de transition se trouve être fonctionnelle). Mais même dans le cas plus général, on peut garder notre méthode de représentation précédente !

Exemple 5 : Un exemple d'automate fini non-déterministe, avec un unique état initial, avec aucune ε-transition, mais qui n'est pas déterministe parce que sa relation de transition n'est pas fonctionnelle : appelons-le $\mathcal{A}_5$.



Exemple 6 : Je laisse ton imagination me proposer un automate fini non-déterministe avec ε-transitions $\mathcal{A}_6$.

Implémentation : on peut définir le type afnd des automates finis non-déterministes avec ε-transitions par :

```
type afd = {initiaux : int list ;
terminaux : int list ;
transitions : (int * (char option), int) Hashtbl.t};;
```

Ici la relation de transition est représentée par un dictionnaire (lui-même codé par une table de hachage).
Les clés du dictionnaire sont de type `int * (char option) * bool` : le premier élément correspond à l'état de départ, le second élément à la lettre lue (`Some c` pour une vraie lettre, `None` pour ε), le troisième à un booléen précisant l'existence ou pas de la transition.
Les valeurs du dictionnaire sont de type `int` : la valeur correspond à l'état d'arrivée.

Application 2 : Implémenter $\mathcal{A}_5$ et $\mathcal{A}_6$. Commencer par définir les relations de transition en bourrinant les `Hashtbl.add`.

II.2 Langage reconnu par un automate fini non-déterministe avec ε-transition

Définition 5 : Langage reconnu par un automate

Les trois points de la définition 3 gardent tout leur sens pour les automates non-déterministes, avec ou sans ε-transition !

Notation 1 Soit $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ un automate.
Pour $q \in Q$ et $u \in \Sigma^*$, on notera $\delta^*(q,u)$ l'ensemble des états q' tels que $q \xrightarrow[(\delta^*)]{u} q'$.
Si, pour tout état, la restiction de δ à cet état à la source est fonctionnelle alors on a toujours $|\delta^*(q,u)| \leqslant 1$.
Si, pour tout état, la restiction de δ à cet état à la source est totale alors on a toujours $|\delta^*(q,u)| \geqslant 1$.
Le langage $\mathcal{L}(\mathcal{A})$ reconnu par $\mathcal{A}$ est $\mathcal{L}(\mathcal{A}) = \left\{u \in \Sigma^* \mid \exists q_0 \in I,\ \delta^*(q_0,u) \cap F \neq \emptyset\right\}$.

Exemples 7 : Langage reconnu par $\mathcal{A}_5$? Voire par $\mathcal{A}_6$?

Implémentation trop naïve : pour écrire une fonction `reconnu_par_afnd : string → afd → bool` qui détermine si un mot est ou pas reconnu par un automate fini non-déterministe avec ε-transition, la stratégie la plus naïve est de mimer ce qu'on a fait pour un automate fini déterministe : on définit une fonction auxiliaire locale `accepte : string → int → int → bool`. Dans celle-ci, on teste si, parmi les chemins qu'il est possible de parcourir en lisant le sous-mot de `chaine` commencant au caractère `i`, à partir de l'état `etat` , il y en a au moins un qui conduit à un état terminal. On commencera par écrire `images : (int * char * int) list → int * char → int list` telle que `images delta (i,c)` donne l'ensemble des états accessibles à partir de l'état `i` en lisant le caractère `c` pour la relation de transition `delta`.

Pour faire mieux, on va essayer de transformer les automates compliqués en automates plus simples.

Définition 6 : Automates équivalents

Deux automates $\mathcal{A}$ et $\mathcal{A}'$ (déterministes ou pas, avec ou sans ε-transitions) définis sur un même alphabet Σ sont dits équivalents lorsqu'ils reconnaissent le même langage, i. e. lorsqu'on a $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

III Transformation d'un automate en automate équivalent

III.1 Automate complet. Complétion.

Définition 7 : Automate fini complet

Un automate fini $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ est dit complet lorsque la relation δ de $Q \times \Sigma$ dans Q est totale (en particulier, un automate fini est déterministe et complet lorsque c'est une application).

Autrement dit, un automate est complet lorsque, pour tout état q, pour toute lettre ℓ, on a toujours une transition depuis l'état q étiquettée par la lettre ℓ.

Exemple 8 : L'automate $\mathcal{A}_1$ sur l'alphabêt $\Sigma = \{a, b\}$ n'est pas complet (c'est pas passé loin).

En effet .....

Théorème 1 : Automate complet équivalent

Tout automate est équivalent à un automate complet (déterministe ssi l'automate de départ l'est).

Démonstration. Il suffit de

Exemples 9 : Des automates complets équivalents à $\mathcal{A}_1$, à $\mathcal{A}_5$.

III.2 Automate émondé. Émondage.

On complète un automate en rajoutant un état inutile mais qui permet artificiellement d'éviter tout blocage dans l'automate lors de la lecture d'un mot. À l'inverse, on pourrait enlever tous les états inutiles d'un automate, quitte à perdre l'éventuel caractère complet, pour obtenir un automate moins volumineux.

Définition 8 : États accessibles, co-accessibles, utiles ; automate émondé

Soit $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ un automate fini.

1. Un état $q \in Q$ est dit accessible lorsqu'il existe un état initial $q_0 \in I$ et un mot $u \in \Sigma^*$ tel que le calcul $q_0 \xrightarrow[(\delta^*)]{u} q$ soit correct.
2. Un état $q \in Q$ est dit co-accessible lorsqu'il existe un état initial $q_f \in F$ et un mot $u \in \Sigma^*$ tel que le calcul $q \xrightarrow[(\delta^*)]{u} q_f$ soit correct.
3. Un état $q \in Q$ est dit utile lorsqu'il est à la fois accessible et co-accessible.
4. L'automate $\mathcal{A}$ est dit émondé lorsque tous ses états utiles.

Théorème 2

Tout automate est équivalent à un automate émondé (déterministe ssi l'automate de départ l'est).

Démonstration. Il suffit d'

III.3 Suppression des ε-transitions

Un premier résultat spectaculaire :

Théorème 3 : Automate sans ε-transition équivalent

Tout automate non-déterministe avec ε-transition est équivalent à un automate non-déterministe sans ε-transition.

Pourtant, s'autoriser des ε-transitions semblait tout sauf innocent ! Bin si.

Démonstration. Procédons par *fermeture arrière* (une méthode équivalente est la fermeture avant, mais on pert l'unicité éventuelle de l'état initial lorsqu'on dispose de cellle-ci, c'est dommage) :

1. Pour chaque transitions $q_1 \xrightarrow[(\delta^*)]{\varepsilon} q_2$ et $q_2 \xrightarrow[(\delta)]{\ell} q_3$ (avec $\ell \in \Sigma$), on rajoute une transition de q_1 à q_3 étiquettée par ℓ.
2. Si de plus q_3 est un état terminal, on rajoute q_2 à l'ensemble des états terminaux.
3. Et bien sûr on supprime les ε-transitions.

Pour tout calcul réussi dans l'automate initial, on obtient par récurrence immédiate sur le nombre d'ε-transitions de ce calcul un calcul réussi dans l'automate ainsi contruit, ce qui montre que ce dernier est équivalent à l'automate inital.

Exemples 10 : Ok comme on a rien compris traitons deux exemples.

III.4 Déterminisation via l'automate des parties

Montrons maintenant un truc de taré :

Théorème 4 : Automate déterministe équivalent
Tout automate est équivalent à un automate déterministe.

Je dis truc de taré parce que la définition d'automate fini non-déterministe est tellement plus souple que celle d'automate fini déterministe qu'on aurait légitimement pu s'attendre à ce que les automates finis non-déterministes reconnaissent davantage de langages que les automates finis déterministes ! Bin non.

Démonstration. Soit $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ un automate (non-déterministe, avec ε-transitions) fini. On construit un automate déterministe équivalent $\mathcal{A}_d$ sur Σ comme suit :

- les états de $\mathcal{A}_d$ sont des parties de $\mathcal{P}(Q)$;
- l'état initial de $\mathcal{A}_d$ est $I \cup J$ où J est l'ensemble des états q pour lequels il existe $q_0 \in I$ avec $q_0 \xrightarrow[(\delta)]{\varepsilon} q$;
- pour $P_1, P_2 \subset Q$ il existe une transition $P_1 \xrightarrow[(\delta_d)]{\ell} P_2$ si et seulement si il existe $e_1 \in P_1$, $e_2 \in P_2$ et une transition $e_1 \xrightarrow[(\delta)]{\ell} e_2$;
- les états terminaux de $\mathcal{A}_d$ sont les parties P pour lesquelles $P \cap F \neq \emptyset$.

L'automate $\mathcal{A}_d$ ainsi contruit est déterministe et complet par construction.
De plus, par récurrence immédiate, on obtient que pour $P_1, P_2 \subset Q$ et $u \in \Sigma^*$, on a $P_1 \xrightarrow[(\delta_d^*)]{u} P_2$, si et seulement si il existe $q_1 \in P_1$ et $q_2 \in P_2$ pour lesquels on a $q_1 \xrightarrow[(\delta^*)]{u} q_2$, ce qui, sachant qu'un état de $\mathcal{A}_d$ est terminal si et seulement si c'est une partie qui comprend un état terminal de $\mathcal{A}$, montre que les deux automates reconnaissent le même langage. 

Exemples 11 : Ok comme on a rien compris traitons plein d'exemples. Je commence.

IV Théorème de Kleene et applications

Dans cette section on va enfin montrer le théorème intéressant du chapitre : *un langage est régulier si et seulement si il est reconnaissable par un automate.* Je ne précise plus pour automate, puisqu'on a vu précedemment qu'un langage est reconnu par un automate truc si et seulement si il est reconnu par un automate machin, pour tous les trucs et tous les machins qu'on a définis dans le chapitre.

IV.1 Régulier implique reconnaissable

Théorème 5 : Sens direct du théorème de Kleene

Tout langage régulier est reconnu par un automate.

Démonstration. Rappelle-toi, un langage est régulier si et seulement si il est rationnel, et l'ensemble des langages rationnels est le plus petit sous-ensemble de l'ensemble de tous les langages qui comprend $\emptyset$, $\{\varepsilon\}$ et les singletons de Σ qui soit stable par réunion, concaténation et étoile de Kleene.

Il suffit donc de montrer que l'ensemble des langages reconnus par un automate comprend $\emptyset$, $\{\varepsilon\}$ et les singletons de Σ, et est stable par réunion, concaténation et étoile de Kleene.

Remarque 1 : La démonstration précédente est **constructive**, elle donne un algorithme pour construire un automate à partir d'une expression régulière. Cet algorithme est appelé algorithme de Thompson, mais franchement, c'est l'enfer sur terre. Sur une expression régulière aussi gentille que $ab+c^*$, c'est déjà bien pénible de produire l'automate. Faisons-le quand même.

Remarque 2 : Il existe un algorithme plus raisonnable, *l'algorithme de Berry-Sethi*, permettant d'obtenir un tel automate, *l'automate de Glushkov*. On garde ça pour la dernière séance (est-ce aujourd'hui parce qu'on est en retard ?).

IV.2 Reconnaissable implique régulier

Théorème 6 : Sens direct du théorème de Kleene

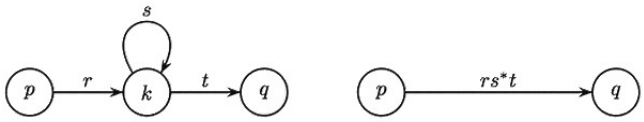
Tout langage reconnu par un automate est régulier.

Le programme indique de seulement donner des exemples d'application de l'algorithme d'élimination pour montrer ce sens.

L'algorithme d'élimination procède comme suit : on s'autorise au cours de l'algorithme d'étiquetter les transitions non plus seulement par des lettres de Σ mais aussi par des expressions régulières. À partir d'un l'automate on produit une expression régulière en rajoutant un état initial source avec ε-transition vers tous les autres états initiaux, un état terminal but avec ε-transitions depuis tous les états terminaux, puis en réduisant successivement ses transitions et ses états par application des deux règles suivantes :

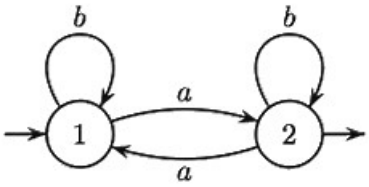
1. Réduction des transitions : on remplace deux transitions $p \xrightarrow[(\delta)]{r} q$ et $p \xrightarrow[(\delta)]{r} q$ par la transition $p \xrightarrow[(\delta)]{r} q$.

2. Réduction des états : on enlève un état k, et pour tous p, q, r, s, t tels que $p \xrightarrow[(\delta)]{r} k$, $k \xrightarrow[(\delta)]{s} k$ et $k \xrightarrow[(\delta)]{t} q$ on ajoute la transition $p \xrightarrow[(q)]{rs^*t}$ (où $rs^*t = rt$ s'il n'existe pas de telle transition s).



Lorsqu'il ne reste plus qu'un unique état initial, la source, un unique état terminal, le but, et une unique transition de l'état initial vers l'état terminal, celle-ci est étiquettée par une expression rationnelle reconnaissant le même langage que l'automate initial.

Exemple 12 : Déterminons une expression régulière reconnaissant le même langage que l'automate suivant :



IV.3 Applications

Théorème 7

L'ensemble des langages rationnels est stable par complémentaire et par intersection.

Démonstration.

Théorème 8 : Lemme de l'étoile

Soit L un langage rationnel, alors il existe un entier n tel que pour tout $u \in L$ tel que $|u| \geqslant n$ il existe $x, y, z \in \Sigma^*$ pour lesquels $u = xyz$, $y \neq \varepsilon$ et $xy^*z \subset L$.

Plus précisément, si $\mathcal{A}$ est un automate reconnaissant L, on peut prendre pour n le nombre d'états de $\mathcal{A}$ et x, y, z tels que $|xy| \leqslant n$.

Démonstration.

Application 3 : Le langage des mots de Dyck $\left\{a^n b^n,\ n \in \mathbb{N}\right\}$ n'est pas rationnel.

Démonstration.