




Mathématiques

IUT INFO 1 / 2012-2013

Licence Creative
Commons   
MAJ: 15 décembre 2012

Mathématiques discrètes pour l'informatique (III)



TABLE DES MATIÈRES

| | | |
|----------|---|-----------|
| 5 | Machines de Turing | 5 |
| 5.1 | Approche historique | 6 |
| 5.2 | Les machines de TURING | 6 |
| 5.2.1 | Problème a posteriori | 6 |
| 5.2.2 | Description sommaire | 7 |
| 5.2.3 | La Machine de Turing par Alan Turing | 7 |
| 5.2.4 | Un exemple pour découvrir | 7 |
| 5.2.5 | La théorie en pratique | 8 |
| 5.2.6 | Définitions | 9 |
| 5.2.7 | Fonctions MT-calculables | 10 |
| 5.2.8 | Forme canonique | 11 |
| 5.2.9 | Machines universelles | 11 |
| 5.2.10 | Fonctions récursives | 12 |
| 5.3 | EXERCICES | 15 |
| 6 | Langages et automates | 17 |
| 6.1 | Langages | 18 |
| 6.1.1 | Un exemple pour découvrir | 18 |
| 6.1.2 | Définitions et notations | 20 |
| 6.1.3 | Langages | 22 |
| 6.1.4 | Langages et expressions rationnels (ou réguliers) | 23 |
| 6.2 | Automates | 27 |
| 6.2.1 | Définitions | 27 |
| 6.2.2 | Langage reconnaissable | 28 |
| 6.2.3 | Langage reconnu par un automate fini | 28 |
| 6.2.4 | Automates équivalents | 30 |
| 6.2.5 | Automates standards | 30 |
| 6.2.6 | Automates déterministes | 31 |
| 6.2.7 | Automates émondés | 32 |
| 6.2.8 | Automate minimal | 34 |
| 6.2.9 | Opérations rationnelles sur les automates | 37 |
| 6.2.10 | Théorème de KLEENE | 38 |
| 6.2.11 | Automate associé à un langage défini par une expression rationnelle | 38 |
| 6.2.12 | Automates séquentiels | 38 |
| 6.2.13 | Automates à pile | 39 |
| 6.2.14 | Machines de TURING : le retour | 41 |
| 6.3 | Grammaires | 42 |
| 6.3.1 | Syntaxe et sémantique | 42 |
| 6.3.2 | Grammaires algébriques (ou hors contexte) | 43 |
| 6.3.3 | Grammaire régulière | 43 |
| 6.3.4 | Lemme de la pompe | 44 |
| 6.3.5 | Analyse syntaxique (« parsing ») | 44 |
| 6.4 | Correspondance automate fini / grammaire régulière | 45 |

| | | |
|--------|---|----|
| 6.5 | EXERCICES | 46 |
| 6.5.1 | Langages | 46 |
| 6.5.2 | Langages et expressions rationnels | 47 |
| 6.5.3 | Généralités sur les automates | 48 |
| 6.5.4 | Expression rationnelle associée à un automate | 48 |
| 6.5.5 | Automates standards et émondés | 49 |
| 6.5.6 | Automates déterministes | 49 |
| 6.5.7 | Construction d'automates | 50 |
| 6.5.8 | Automates séquentiels | 50 |
| 6.5.9 | Automates à pile | 50 |
| 6.5.10 | Problèmes divers | 51 |
| 6.5.11 | Grammaires | 52 |
| 6.5.12 | Révisions | 53 |
| 6.6 | Automates et Caml | 55 |
| 6.6.1 | Modélisation | 55 |
| 6.6.2 | Visualisation | 56 |
| 6.6.3 | Standardisation | 57 |
| 6.6.4 | Opérations rationnelles | 59 |

CHAPITRE

Machines de TURING



En informatique, tout a commencé avant la construction du premier ordinateur, dans le cerveau fécond de quelques mathématiciens...

1 Approche historique



A.CHURCH (1903-1995)

Les essais de formalisation de la pensée ont commencé bien avant l'apparition des ordinateurs (notamment avec LEIBNIZ au XVII^e siècle) mais ont réellement pris leur essor avec les travaux de BOOLE (fin du XIX^e).

La période qui va surtout influencer le monde informatique se situe pendant l'entre-deux-guerres marquée par les travaux d'Alonzo CHURCH, Alan TURING, Stephen KLEENE, Haskell CURRY entre autres car elle marque les prémisses de l'informatique avant même la construction du premier ordinateur : λ -calcul, calculabilité, complexité algorithmique, équivalence entre fonctions mécaniquement calculables et définitions formelles des fonctions récursives.

Nous n'allons pas voguer dans les hautes sphères de la logique mathématique et de l'informatique théorique mais il est important pour un(e) informaticien(ne) d'aborder ces notions de mécanisation de la pensée qui sont l'essence de sa discipline.

Nous aborderons la logique comme un exemple de système formel que nous étudierons plus en détail à travers la théorie des langages, des grammaires et des automates qui sont à la base de la construction des compilateurs. Mais pour commencer par le commencement, nous étudierons brièvement l'ordinateur le plus simple qu'on puisse imaginer : la machine de TURING. Nous ferons le lien entre calculabilité et récursivité.

2 Les machines de Turing

2.1 Problème a posteriori

Contrairement à Alan TURING en 1936, nous savons aujourd'hui à quoi ressemble un ordinateur en pratique grâce aux merveilleux cours dispensés à l'IUT. Pour résumer, un programme est une décomposition d'une tâche complexe en tâches élémentaires que peut « comprendre » la machine, en fait ce que nous appellerons l'*automate* dont nous disposons : dans les faits, il s'agit du processeur. Le problème, c'est que les automates (les processeurs) sont différents : il faut donc leur donner des instructions différentes selon les cas. C'est ce que fait un compilateur qui traduit des instructions d'un langage évolué vers les instructions élémentaires (le langage machine) de l'automate-processeur dont nous disposons.

Cette dépendance au type de processeur utilisé est gênante pour obtenir des résultats généraux en informatique théorique sur la *complexité* (ça va être faisable ? en un temps raisonnable ? c'est performant ?), la *décidabilité* (ça va marcher ? Faire ce que nous voulons ?).

C'est pourquoi on a eu l'idée de revenir aux sources ante-ordinateur et de choisir une machine théorique suffisamment générale et abstraite pour pouvoir représenter tous les automates programmables : c'est la *machine de Turing*. Elle sert donc de mètre-étalon pour mesurer la complexité d'un programme et explique le fonctionnement de tout ordinateur dans sa version *universelle*.

2 2 Description sommaire



Alan TURING est le véritable homme à la pomme (dont il a utilisé un exemplaire pour s'empoisonner) de l'informatique car il en a posé les fondements et non pas exploité les capacités commerciales.

En 1936, il présente sa « machine » théorique afin de donner une définition précise du concept d'algorithme.

Il s'agit d'un ruban ayant un commencement mais une longueur infinie et qui représente la mémoire d'un ordinateur (qui, elle, est finie...).

Le ruban est constituée de cases pouvant contenir un symbole. À tout moment, seul un nombre fini de cases ne sont pas blanches (ou plutôt ne contenant pas le symbole « blanc »). Un pointeur identifie la case sur laquelle la machine « travaille » à un instant donné.

Ce que nous appelons un programme est une fonction qui prend comme argument le symbole pointé et le degré d'avancement du pointeur.

2 3 La Machine de Turing par Alan Turing

Voici un extrait de l'article fondateur paru en 1936 (dans une traduction de Julien BASH) :
Nous avons dit qu'un nombre est calculable lorsque son expression décimale est calculable avec des moyens finis, définition que nous allons maintenant expliciter [...] je me contenterai de suggérer que cette définition est justifiée par le fait que la mémoire humaine est nécessairement limitée.

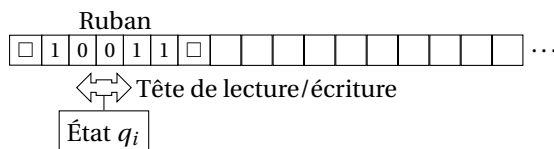
*Un homme en train de calculer la valeur d'un nombre réel peut être comparé à une **machine** susceptible de se trouver dans un nombre fini d'états q_1, q_2, \dots, q_R , que nous appellerons ses **m-configurations**. La machine est alimentée avec une **bande** (analogue au papier qu'utilise l'homme), divisée en sections (appelées **cases**), dans chacune desquelles peut être inscrit un **symbole**. Dans la case r est inscrit le symbole $S(r)$. À chaque instant, il n'y a qu'une seule case dans la machine : c'est la **case inspectée**, dans laquelle est inscrit le **symbole inspecté**, le seul dont la machine est pour ainsi dire « directement consciente ». Cependant, la machine peut garder trace de certains des symboles qu'elle aura vus (inspectés) précédemment en modifiant sa m-configuration. À chaque instant, la liste des comportements possibles de la machine est entièrement déterminé par sa m-configuration q_n et le symbole inspecté $S(r)$. Le couple $(q_n, S(r))$ est appelé la **configuration** de la machine, et c'est donc cette configuration qui détermine l'évolution possible de la machine, qui peut être, entre autres :*

- l'inscription d'un nouveau symbole sur la case inspectée (si celle-ci était blanche) ;
- l'effacement du symbole inspecté, la case inspectée devenant blanche ; le changement de case inspectée (uniquement par passage à la case suivante ou précédente) ;
- le passage à une autre m-configuration.

2 4 Un exemple pour découvrir

Commençons par un problème simple.

Il faut s'imaginer une machine de TURING comme ceci par exemple :



Un □ symbolise une case « blanche » et occupe la première case. Il est suivi par des 0 et des 1 et une autre case blanche marque la fin de la chaîne.

L'ensemble des symboles $\{0, 1, \square\}$ forme un **alphabet**.

Le pointeur indique la première case. Nous voudrions par exemple changer les 0 en 1 et vice-versa puis remettre le pointeur dans sa position initiale.

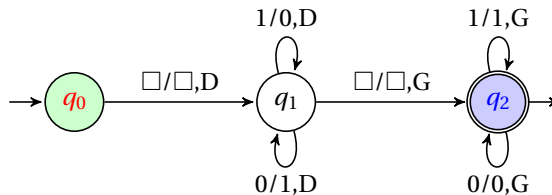
Il est possible d'effectuer trois actions qui dépendent de l'état du pointeur et du caractère pointé :

- changer l'état du pointeur ;
- changer le caractère pointé sur le ruban ;
- déplacer le pointeur d'une case vers la gauche ou vers la droite.

L'algorithme est donc :

- lorsque le pointeur rencontre le premier blanc, il se déplace vers la droite ;
- chaque fois que le pointeur rencontre un 0 ou un 1, il le change en son complémentaire et se déplace vers la droite ;
- lorsqu'il rencontre le deuxième blanc, il recule d'une case vers la gauche jusqu'à ce qu'il rencontre le premier blanc.

Ceci constitue un algorithme que nous prendrons l'habitude de représenter par l'**automate** suivant :



Les cercles correspondent aux différents états de la machine : il y en a trois. Les flèches décrivent, elles, les actions accomplies par la machine.

La flèche entrante en q_0 indique qu'il s'agit de l'état initial. Le double cercle autour de q_2 indique qu'il s'agit de l'état final.

Une flèche allant de q_i à q_j surmontée d'un couple $a/b, c$ signifie que si la machine est dans l'état q_i et qu'elle pointe sur une case contenant a alors elle remplace a par b et bouge d'une case vers la direction indiquée par c : Gauche, Droite, Reste sur place.

On voit qu'un nombre fini d'instructions permet de traiter une chaîne de longueur aussi longue que l'on veut.

Recherche

Testez cette machine sur la chaîne $\square 10011\square$.

2 5 La théorie en pratique

Voici une manière quasi directe de calculer avec une machine de TURING avec CAML. Commentez les fonctions suivantes :

```

exception Pas_lu;;
let rec phi q pos vec =
  match (q,vec.(pos)) with
  | ("q0", "b") -> phi "q1" (pos+1) vec
  | ("q0", _) -> raise Pas_lu
  | ("q1", "b") -> phi "q2" (pos-1) (en_place vec pos "b")
  | ("q1", "0") -> phi "q1" (pos+1) (en_place vec pos "1")
  | ("q1", "1") -> phi "q1" (pos+1) (en_place vec pos "0")
  | ("q2", "b") -> (en_place vec pos (vec.(pos)^^^)) (* sortie *)
  | ("q2", "1") -> phi "q2" (pos-1) (en_place vec pos "1")

```



```
|("q2", "0") -> phi "q2" (pos-1) (en_place vec pos "0")
|_ -> raise Pas_lu;;

let turing phi vec = phi "q0" 0 vec;;
```

On teste alors avec $\square 10011 \square$:

```
# turing phi [|"b"; "1"; "0"; "0"; "1"; "1"; "b"|];;
- : string array = [|"b^"; "0"; "1"; "1"; "0"; "0"; "b"|]
```

Nous venons donc de fabriquer une machine spécialement étudiée pour effectuer une tâche précise.

2 6 Définitions

Machine de TURING

Une machine de TURING standard (il en existe d'autres) est un triplet :

$$M = (Q, X, \varphi)$$

où

- Q est un ensemble fini appelé *alphabet d'état* ;
- X est un autre ensemble fini appelé *alphabet de ruban* ;
- φ est une fonction de domaine $D \subseteq Q \times X$ et à valeurs dans $Q \times X \times \{G, R, D\}$ avec G, R, D les options du mouvement du pointeur (gauche, reste immobile, droite).

On choisira en général Q et X de telle sorte que leur intersection soit vide. Quelques éléments particuliers :

- $q_0 \in Q$ est l'*état initial* ;
- $Q_f \subseteq Q$ est l'*ensemble des états finaux* ;
- $\square \in X$ est le symbole blanc.

Définition 5 - 1

Cela donne, avec les données de l'exemple précédent, $Q = \{q_0, q_1, q_2\}$, $X = \{\square, 0, 1\}$, $Q_f = \{q_2\}$ et la fonction φ est définie à l'aide de la **table de transition** suivante :

| $\varphi(q_i, x)$ | q_0 | q_1 | q_2 |
|-------------------|---------------------|---------------------|---------------|
| \square | (q_1, \square, D) | (q_2, \square, G) | |
| 0 | | $(q_1, 1, D)$ | $(q_2, 0, G)$ |
| 1 | | $(q_1, 0, D)$ | $(q_2, 1, G)$ |

En fait, une machine de TURING agit sur des *mots* construits à partir d'un *alphabet*. Un alphabet est en fait un ensemble fini. Définissons un peu mieux ce que peut être un mot :

Soit X un alphabet. Soit $\mathbb{1}_X$ le mot ne comportant aucun caractère. L'ensemble X^* des mots construits avec l'alphabet X est défini par :

1. $\mathbb{1}_X \in X^*$;
2. si $a \in X$ et $m \in X^*$, alors le mot construit à partir de m en ajoutant a à droite est un mot de X^* ;
3. μ est un élément de X^* seulement s'il peut être obtenu à partir de $\mathbb{1}_X$ par application de l'étape 2 un nombre fini de fois.

Définition 5 - 2

On peut donc à l'aide de cette définition construire un mot en ajoutant un par un des éléments de l'alphabet. Nous devons donc créer une fonction plus rapide pour créer des mots :

Définition 5 - 3

Concaténation

Soient m_1 et m_2 deux mots de X^* . La **concaténation** de ces deux mots est le mot $m_1 m_2 \in X^*$ obtenu en écrivant m_2 à la suite de m_1 .

2 7 Fonctions MT-calculables

La notion de fonction calculable par une fonction de TURING est primordiale en informatique. Une telle fonction sera dite **MT-calculable**.

Cette notion peut paraître assez abstraite alors mieux vaut étudier un exemple avant de l'affronter.

Par exemple, nous allons construire une machine de TURING qui calcule le successeur de la représentation unaire d'un entier.

Définition 5 - 4

Représentation unaire d'un entier

Un nombre entier $n \in \mathbb{N}$ a pour représentation unaire 1^{n+1} c'est-à-dire la concaténation de $n+1$ symboles de l'alphabet $X = \{1\}$.

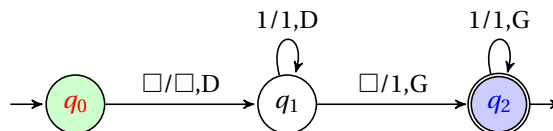
Ainsi, la représentation de 0 est 1, celle de 1 est 11, celle de 2 est 111, etc.

On notera \bar{n} la représentation de l'entier n .

Ainsi, le successeur de n est obtenu en ajoutant un 1 à son écriture unaire.

Décrivons une machine de TURING qui calcule cette fonction :

- la fonction successeur s est définie sur $D = \{1_X, \square 1 \square, \square 11 \square, \square 111 \square, \dots\}$: l'entrée sur le ruban est donc la représentation unaire du nombre n précédée du symbole \square , toutes les autres cases étant occupées par des \square ;
- l'alphabet de ruban est $X = \{1, \square\}$;
- l'alphabet d'état est $\{q_0, q_1, q_2\}$;
- $Q_f = \{q_2\}$;
- la fonction φ est décrite par l'automate suivant :



La configuration initiale est le mot $q_0 \square \bar{n} \square$.

Le pointeur rencontre le premier blanc et change donc d'état et se dirige vers la droite jusqu'à rencontrer un deuxième blanc qui est alors remplacé par un 1. Le pointeur change d'état et se déplace vers la gauche tant qu'il pointe sur un 1.

La machine s'arrête lorsqu'elle rencontre à nouveau un blanc car $\varphi(q_2, \square)$ n'est pas définie.

La configuration finale est $q_2 \square \bar{n} 1 \square$.

Cela vous aidera alors peut-être à comprendre la définition d'une fonction MT-calculable :

Fonction MT-calculable

Soit $f : D \subseteq X^* \rightarrow X^*$ une fonction et $M = (Q, X, \varphi)$ une machine de TURING. M calcule f si :

- il existe une unique transition de q_0 et que sa forme est $\varphi(q_0, \square) = (q_i, \square, D)$ avec $q_i \neq q_0$ (il faut quitter l'état initial à partir de \square) ;
- il n'existe pas de transition de la forme $\varphi(q_i, x) = (q_0, x', m)$ avec $i \neq 0$, $(x, x') \in X^2$ et $m \in \{G, R, D\}$ (il ne faut pas revenir à l'état initial) ;
- il n'existe pas de transition de la forme $\varphi(q_f, \square)$ où $q_f \in Q_f$ (il ne faut pas que la machine continue à fonctionner dans son état final en présence d'un blanc car il y en a une infinité et la machine ne s'arrêterait jamais) ;
- pour tout $\mu \in D$, notons $v = f(\mu)$. Le calcul effectué par la machine M en partant de la configuration initiale $q_0 \square \mu \square$ s'arrête après un nombre fini d'étapes dans la configuration finale $q_f \square v \square$ (la machine doit calculer $f(\mu)$) ;
- le calcul ne s'arrête jamais si $\mu \notin D$ (la machine ne calcule $f(\mu)$ que si ce nombre est défini).

Définition 5 - 5

Nous définirons en TD :

- la fonction identiquement nulle ;
- la fonction addition ;
- les fonctions projections :

$$\pi_i^{(n)}(x_1, x_2, \dots, x_n) = x_i, \quad 1 \leq i \leq n$$

2 8 Forme canonique

On a pris certaines conventions : coder les entrées et sorties en unaire ou binaire, d'utiliser un seul ruban, fini à gauche et infini à droite, avec des opérations élémentaires simples (G, D, R, remplace et laisse inchangé) : c'est la forme canonique des MT.

On peut choisir plusieurs bandes, d'autres alphabets de bandes mais les résultats théoriques seront les mêmes, à une constante multiplicative près dépendant du type de machine. Nous en reparlerons au moment d'étudier la complexité.

2 9 Machines universelles

Pour l'instant, nous fabriquons une machine différente pour chaque tâche à accomplir. C'est peu pratique. La table de transition d'une MT constitue l'automate et n'est pas une donnée entrée sur le ruban.

Il faudrait avoir donc une machine avec une table de transition fixe qui puisse s'adapter à des tâches différentes uniquement en fonction des données d'entrée sans modifier son architecture : c'est alors l'idée de **programme** qui apparaît : non plus une machine pour chaque tâche mais une machine capable d'accomplir un grand nombre de tâches en fonction d'un programme donné en entrée.

Alan TURING a donc eu l'idée de séparer le ruban en plusieurs parties : une partie pour stocker le code de la MT que la MU doit simuler, une pour la mémorisation de la table de transition qui correspond à la MT et une autre permettant de gérer les entrées/sorties.

En fait, on écrit le *programme* sur le ruban (ou un ruban séparé, cela revient au même) ainsi que l'*argument* habituel lu puis calculé par la MT.

2 10 Fonctions récursives

Nous avons découvert des fonctions MT-calculables. Pour arriver jusqu'à la thèse de CHURCH-TURING, il faudrait déterminer quelles sont les fonctions qui sont MT-calculables. Nous allons montrer qu'il s'agit des fonctions récursives dont l'étude est donc indispensable pour étudier l'algorithmique...



J.HERBRAND
(1908-1931)

2 10 1 Fonctions primitives récursives

Il s'agit ici d'un modèle de fonctions récursives introduit par GÖDEL suite à une lettre que lui adressa HERBRAND juste avant de mourir à 23 ans en 1931 dans un accident de montagne.

Les fonctions de base des fonctions primitives récursives sont les fonctions précédemment étudiées :

- la fonction successeur $s : s(x) = x + 1$;
- la fonction identiquement nulle $z : z(x) = 0$;
- les fonctions projection $\pi_i^{(n)}$.

Nous appellerons **fonction arithmétique** une fonction de \mathbb{N}^k dans \mathbb{N} .

Les fonctions de base sont MT-calculables. Nous admettrons que les fonctions arithmétiques obtenues par composition et récurrence le sont aussi, en définissant ces opérations ainsi :

Composition de fonctions arithmétiques

Soient g_1, g_2, \dots, g_k k fonctions arithmétiques de k variables et h une fonction arithmétique de k variables. Soit f la fonction définie par :

$$f(x_1, x_2, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

alors f est appelée la **composition** de h et de g_1, g_2, \dots, g_n et se note :

$$f = h \circ (g_1, g_2, \dots, g_n)$$

Définition 5 - 6

Par exemple, si $h(x_1, x_2) = s(x_1) + s(x_2)$, $g_1(x) = 2x$ et $g_2(x) = x^2 + 37$ alors :

$$f(x) =$$

Récurrence

Soient g et h des fonctions arithmétiques totales de n et $n+2$ variables respectivement. La fonction f de $n+1$ variables définie par :

- $f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$;
- $f(x_1, \dots, x_n, y+1) = h(x_1, \dots, x_n, y, f(x_1, x_2, \dots, x_n, y))$,

est appelée **récurrence de base g et de pas h** .

On conviendra qu'une fonction de zéro variable est constante.

Définition 5 - 7

On peut alors définir les fonctions récursives primitives :

Fonctions récursives primitives

Une fonction est **primitive récursive** si elle peut être obtenue à partir des fonctions *successeur*, *zéro* et *projection* par l'application d'un nombre fini de compositions et de récurrences.

Définition 5 - 8

Par exemple, il est possible de définir l'addition de deux entiers à partir de la fonction s , des projections $\pi_1^{(1)}$ et $\pi_3^{(3)}$ et d'une récurrence de base $g(x) = \pi_1^{(1)}(x)$ et de pas $h = s \circ \pi_3^{(3)}$:

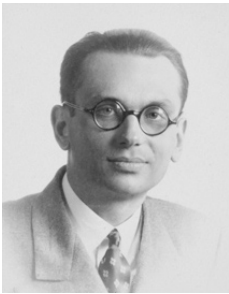
$$\begin{cases} \text{add}(m, 0) = g(m) = m \\ \text{add}(m, n + 1) = h(m, n, \text{add}(m, n)) = s(\text{add}(m, n)) \end{cases}$$

Nous verrons en exercice quelques exemples de fonctions primitives récursives. Cependant, certaines fonctions ne le sont pas comme par exemple la fonction d'ACKERMANN :

$$\begin{cases} A(0, y) = y + 1 \\ A(x + 1, 0) = A(x, 1) \\ A(x + 1, y + 1) = A(x, A(x + 1, y)) \end{cases}$$

qui est cependant MT-calculable mais la preuve est hors de notre portée.

2 10 2 La thèse de Church-Turing



Kurt GÖDEL
(1906 - 1978)

Si nous avons un peu plus de temps, nous pourrions définir les fonctions récursives (comme par exemple les fonctions primitives récursives et la fonction d'ACKERMANN) et montrer que toutes les fonctions récursives sont MT-calculables.

Ainsi les travaux de TURING tendant à définir un critère de « calculabilité » par une machine (ou un être humain muni d'une feuille et d'un crayon) rejoignent ceux de CHURCH et KLEENE sur le λ -calcul ou, ce qui revient au même, les fonctions récursives (notion introduite par GÖDEL au même moment).

C'est un domaine d'étude fort paradoxal : techniquement réduit (les machines de TURING sont simples à décrire !), il entraîne depuis 80 ans des recherches fort ardues.

Il faut cependant garder en tête que les machines de TURING sont des « humains qui calculent » comme le formulera le philosophe WITTGENSTEIN et évidemment TURING lui-même : « un être humain muni d'un papier, d'un crayon, d'un ruban et soumis à une stricte discipline est en fait une machine universelle ».

Comme tous nos ordinateurs *actuels* sont équivalents à des machines de TURING, ils sont donc également des modèles d'êtres humains en train d'effectuer des calculs de manière disciplinée mais non intelligente.

Cette notion est fondamentale également du côté matériel et du côté logiciel. Un ordinateur est une machine de TURING car, pourvu qu'on lui donne un bon code, il effectuera ce que vous voulez.

Le programmeur est lui-même une machine de TURING s'il a écrit son programme (et prouvé qu'il fonctionne...).

Un langage est une machine de TURING.

L'architecture même de l'ordinateur fonctionne selon ce principe : données et instructions sont de même nature dans la mémoire de l'ordinateur comme nombres et fonctions dans la machine de TURING.

Cependant, les machines de TURING sont des modèles théoriques (ruban infini) et donc éloignés du quotidien du programmeur. Savoir qu'il existe un algorithme qui résout notre problème est une chose, connaître sa complexité en est une autre...L'étude de la complexité des algorithmes est un domaine de recherche en soi.

Mais TURING aura surtout lancé le débat sur l'intelligence artificielle : est-ce qu'une machine sera un jour capable de raisonner comme un cerveau humain ? Il avait proposé un test : entamer un dialogue à distance avec une « entité » qui peut être un être humain ou une machine. Lorsque le comportement de la machine sera indiscernable de celui de l'être

humain par ce test, alors on pourra considérer la machine aussi intelligente que l'être humain.

Cette évolution passera peut-être par une remise en cause de l'ordinateur électronique : des essais d'ordinateur à ADN sont en cours depuis une quinzaine d'années par exemple mais leur base théorique est toujours la machine de TURING...

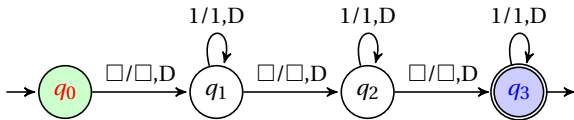
Pour finir, une nouvelle citation de Donald E. KNUTH :

Science is what we understand well enough to explain to a computer. Art is everything else we do.

EXERCICES

Exercice 5 - 1

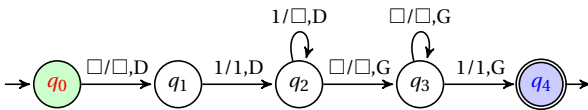
On considère la machine de TURING représentée par l'automate ci-dessous qui est dans l'état initial $q_0 \square 1111 \square 111111 \square 111111 \square 111 \square 1111 \square 111011 \square$:



Quelle est la configuration finale de la machine ?

Exercice 5 - 2

Quelle est la configuration finale de la machine de TURING décrite par l'automate suivant sachant que la configuration initiale du ruban est $q_0 \square \bar{x} \square$ (la représentation unaire d'un nombre x est notée \bar{x}) :



Exercice 5 - 3

Construire une machine de TURING qui calcule la fonction prédécesseur.

Exercice 5 - 4

Comment s'écrit 0 en système unaire ? Construire une machine de TURING qui calcule la fonction identiquement nulle : la configuration initiale est $q_0 \square \bar{x} \square$ et la configuration finale est $q_f \square 1 \square$

Exercice 5 - 5

Nous allons construire une machine à additionner. En base 1, c'est assez simple : écrivez l'addition en base 1 de 4 et 1 puis de 4 et 0.

On suppose qu'on a marqué en entrée $\square \bar{x} \square \bar{y} \square$ avec x et y les deux nombres à additionner.

La configuration finale attendue est $q_f \square \overline{x+y} \square$.

Exercice 5 - 6

Déterminer une MT qui renvoie $\square 1 \square$ si le nombre à tester, sous forme binaire, est pair et $\square 0 \square$ sinon.

Exercice 5 - 7

Voici les codes ASCII décimaux de quelques caractères affichables :

| Caractère | A | B | C | D | E | F | G | H | I | J |
|-----------|----|----|----|----|----|----|----|----|----|----|
| Code | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 |

| Caractère | a | b | c | d | e | f | g | h | i | j |
|-----------|----|----|----|-----|-----|-----|-----|-----|-----|-----|
| Code | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 |

1. Écrire les valeurs en base 2 des codes de A, B et C puis a, b et c.
2. Déterminer un automate décrivant une machine de TURING la plus simple possible qui transforme le code ASCII (en binaire) d'une lettre minuscule en le code ASCII (en binaire toujours) de la majuscule correspondante.

Exercice 5 - 8

Décrivez sommairement une MT qui reconnaisse le mot FOR, le vocabulaire de ruban étant $\{0, 1\}$: on utilisera les codes ASCII binaires des lettres F, O et R.

Exercice 5 - 9

Bon, maintenant, vous êtes bien persuadé(e) qu'un caractère, c'est en fait un entier. Nous allons gagner un peu de place en utilisant comme alphabet de ruban $V = \{a, z, X, \square\}$. Voici la table de transition d'une machine :

| $\phi(q_i, \cdot)$ | q_0 | q_1 | q_2 | q_3 |
|--------------------|---------------------|---------------|---------------------|---------------------|
| \square | | | (q_0, \square, D) | (q_4, \square, R) |
| a | (q_1, \square, D) | (q_1, a, D) | (q_2, a, G) | |
| z | | (q_2, X, G) | | |
| X | (q_3, X, D) | (q_1, X, D) | (q_2, X, G) | (q_3, X, D) |

Que se passe-t-il avec $az \square$, $aaaz \square$, $aaazzz \square$, $azz \square$, $aaaz \square$, $za \square$, cette machine en général ?

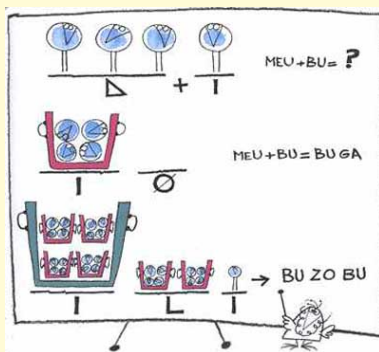
Exercice 5 - 10

L'alphabet de ruban est $\{a, b\}$. Déterminez une MT qui reconnaisse les mots du type ab^n puis une autre pour les mots $a^n b^p$ avec n et p des entiers strictement positifs.

Exercice 5 - 11 Filtre à rebond

Lorsqu'on transmet une chaîne de bits, voir un 1 ou un 0 isolé peut apparaître comme une anomalie (on parle de bruit), tout comme un petit pixel noir dans une zone blanche. Déterminez une machine de TURING qui élimine les éventuels 1 ou 0 isolés (au milieu de bits opposés) d'une chaîne donnée.

6 Langages et automates



Nous essaierons dans ce chapitre de découvrir comment parler à une machine... Quand nous parlons entre nous, nous utilisons un *langage naturel* dont les ambiguïtés en font un moyen à la fois riche et difficile de communication. Une machine n'a pas notre humour et notre goût pour la poésie : elle utilise un *langage formel* qui doit la renseigner de manière précise. Tout doit être correctement spé-ci-fié.

Notre but sera donc de jeter les bases des langages entre l'homme et la machine ou même entre machines.

Pour citer Jacques SAKAROVITCH, grand spécialiste de la théorie des automates : « *Les automates finis, c'est l'infini mis à la portée des informaticiens* » reprenant à son compte cette sentence de L.F. CÉLINE dans « Voyage au bout de la nuit » :

« *L'amour, c'est l'infini mis à la portée des caniches* »...

1 Langages

1 1 Un exemple pour découvrir

Reconnaître des mots est une tâche très importante en informatique qu'on retrouve dans des domaines aussi divers que :

- les compilateurs qui transforment un code de haut niveau, compréhensible par un humain, en un code machine directement utilisable par l'ordinateur ;
- les éditeurs de texte et les traitements de texte ;
- les bases de données, les moteurs de recherche ;
- les motifs de pixels sur un écran,...

Nous nous occuperons donc aussi bien de reconnaître un A dans ce code :

```

OOOXOOO
OOXXXXO
OOXOXOO
OXXOXXO
OXXXXXO
XXOOOXX
XOOOOOX

```

que de l'exemple suivant : nous avons déjà découvert que les machines de TURING utilisaient un *alphabet* assez sommaire pour former des *mots* ; nous avons donné des définitions générales de ces notions.

Voyons à présent un exemple plus complexe correspondant plutôt à l'action d'un compilateur (votre maîtresse d'école vous a peut-être présenté les choses ainsi). Nous voudrions que l'ordinateur puisse produire la phrase suivante (ou vérifier qu'elle est « correcte ») :

LA JEUNE FILLE ÉTUDIE LE BEAU COURS.

L'*alphabet des symboles* est :

$$A = \{LA, JEUNE, FILLE, ÉTUDIE, LE, BEAU, COURS, .\}$$

Vous avez noté que le mot « alphabet » n'est pas à prendre au sens usuel du terme. Par exemple, **def**, **return**, **if**, **for** sont des éléments de l'alphabet de symboles de Python. Maintenant, ces symboles ne peuvent pas être assemblés n'importe comment ; il faut suivre les règles d'une *grammaire* dont voici un exemple :

1. <phrase> → < sujet > < verbe > < complément > < ponctuation >
2. < sujet > → < groupe nominal >
3. < complément > → < groupe nominal >
4. < groupe nominal > → < article féminin > < adjectif féminin > < nom féminin >
5. < groupe nominal > → < article féminin > < nom féminin > < adjectif féminin >
6. < groupe nominal > → < article féminin > < nom féminin >
7. < groupe nominal > → < article masculin > < adjectif masculin > < nom masculin >
8. < groupe nominal > → < article masculin > < nom masculin > < adjectif masculin >
9. < groupe nominal > → < article masculin > < nom masculin >

10. <article masculin> → LE
11. <article féminin> → LA
12. <adjectif féminin> → JEUNE
13. <adjectif masculin> → JEUNE
14. <adjectif masculin> → BEAU
15. <nom féminin> → FILLE
16. <nom masculin> → COURS
17. <verbe> → ÉTUDIE
18. <ponctuation> → .

Ici, <phrase> est l'*axiome* : c'est notre point de départ.

Les *variables* sont les termes entre <>.

Les règles 1 à 9 sont les *règles syntaxiques*, la première étant toujours l'axiome.

Les règles 10 à 18 sont les *règles lexicales*.

Le *langage engendré* par la grammaire est l'ensemble de toutes les phrases que l'on peut former en respectant les règles. Voici quelques exemples de phrases « grammaticalement correctes » comme disait votre maîtresse :

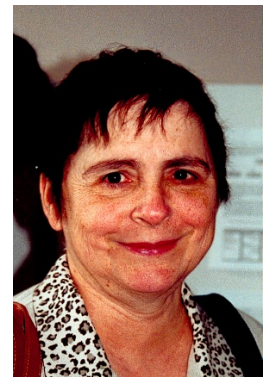
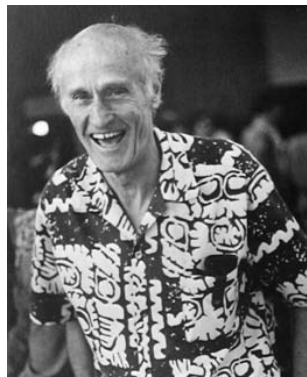
LE JEUNE COURS ÉTUDIE LA FILLE.
LA FILLE ÉTUDIE LE JEUNE COURS.
LA FILLE ÉTUDIE LE COURS.

La grammaire ne suffit pas : certaines phrases heurtent notre sens commun mais ceci est une autre histoire...

Remarque

Pour éviter le caractère ambigu des termes « mot », « lettre » employés dans de nombreux ouvrages et même un peu plus tôt dans ce cours, nous préférons dans le cours qui suit utiliser les termes « symbole » et « chaînes » car nous venons de voir qu'une « lettre » au sens mathématique peut être un « mot » au sens courant... De plus, les termes « caractères » et « chaînes » désignent habituellement, dans les langages informatiques, des types de variables qui correspondent aux notions que nous allons introduire.

Le domaine que nous allons étudier a été défriché par des linguistes, des mathématiciens, des informaticiens, des philosophes. Voici par exemple des portraits du linguiste Noam CHOMSKI, du mathématicien Stephen KLEENE et de l'informaticienne Sheila GREIBACH qui ont donné leur nom à de nombreuses notions que nous croiserons dans notre cours :



Noam CHOMSKI, né en 1928 et professeur de linguistique au MIT, est également connu pour son engagement politique (une recherche sur le nombre d'occurrences de la chaîne « Chomski » dans n'importe quel numéro du *Monde diplomatique* est un exercice de mise en pratique de notre cours et renverra toujours un résultat non nul...)

Stephen KLEENE (1909 - 1994) est un mathématicien américain dont nous avons déjà parlé car il a fondé la théorie de la récursion avec ALONZO CHURCH, Kurt GÖDEL et Alan TURING. Il est l'inventeur du concept d'expression rationnelle et de langage rationnel que nous retrouverons tout au long de ce cours.

Sheila GREIBACH, née en 1939 et professeur à UCLA a effectué d'importantes recherches dans le domaine des langages formels, des automates et des compilateurs.

1 2 Définitions et notations

Définition 6 - 1

Un *alphabet* est un ensemble fini non vide. Les éléments sont appelés des *symboles* ou des *caractères*.

Nous avons rencontré l'alphabet $\{\epsilon, 1, 0\}$ du ruban des machines de TURING, mais aussi l'ensemble $\{LA, JEUNE, FILLE, ÉTUDIE, LE, BEAU, COURS, .\}$ dans l'exemple d'introduction. On peut aussi penser à $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, \times, -, /, (,)\}$ pour écrire des expressions arithmétiques dans \mathbb{N} , à l'alphabet MORSE $\{., -, _.\}$. On peut également penser au *Petit Robert* comme un alphabet, une phrase étant une chaîne, un symbole étant un mot... Plus informatiquement, considérons des fichiers texte : l'alphabet peut être constitué des lignes se terminant par un retour chariot, une chaîne étant un fichier.

Une *chaîne* est une juxtaposition de symboles. On appelle cette juxtaposition la *concaténation*.

Plus rigoureusement :

Soit A un alphabet. Soit $\mathbb{1}_A$ la *chaîne vide* (ne comportant aucun symbole).

L'ensemble A^* des chaînes construites avec l'alphabet A est défini par :

1. $\mathbb{1}_A \in A^*$;
2. si $a \in A$ et $c \in A^*$, alors la chaîne construite à partir de c en concaténant a à droite est une chaîne de A^* ;
3. μ est un élément de A^* seulement s'il peut être obtenu à partir de $\mathbb{1}_A$ par application de l'étape 2 un nombre fini de fois.

Définition 6 - 2

La *longueur* d'une chaîne est le nombre de symboles qui la composent. On note souvent $|\mu|$ la longueur de la chaîne μ . En particulier $|\mathbb{1}_A| = 0$.

On note $|\mu|_a$ le nombre d'occurrences du symbole a dans la chaîne μ . On note A^n l'ensemble de toutes les chaînes de longueur n .

Par convention, $A^0 = \{\mathbb{1}_A\}$ qui n'est donc pas vide.

$A^* = \bigcup_{n \geq 0} A^n$: c'est l'ensemble de toutes les chaînes, éventuellement vides.

$A^+ = \bigcup_{n > 0} A^n$: c'est l'ensemble de toutes les chaînes non vides.

La concaténation des chaînes μ et ν sera notée $\mu\nu$.

Par exemple, si $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, \times, -, /, (,)\}$, alors $3 * (4 - 2) + 8$ et $)32 + - 7$ sont des chaînes de longueurs respectives 9 et 6.

Vous aurez noté que les notations rappellent la notion de fermeture transitive et transitive réflexive...

Vous aurez noté que $A^1 = A$.

Vous aurez noté que A a beau être fini, A^* est infini.

Notations

Par convention, nous noterons en général les symboles avec les premières lettres de l'alphabet latin (a, b, c, d, \dots), les chaînes avec des lettres grecques (μ, ν, ξ, \dots) et les alphabets avec des majuscules.

Remarque

La concaténation est une *loi de composition interne* sur A^* car en concaténant deux éléments de A^* , on obtient encore un élément de A^* .

Cette loi est également associative et possède un *élément neutre* $\mathbb{1}$. Cela confère à A^* la structure de *monoïde libre*, selon les standards informatiques. Les mathématiciens parlent de *magma associatif unitaire*. En anglais, on utilise le terme *lattice*.

L'associativité nous permet d'utiliser la notation *puissance* :

$$a^n = \underbrace{aa \dots a}_n; a^n b^k = \underbrace{aa \dots a}_n \underbrace{bb \dots b}_k; (ab)^n = \underbrace{ababab \dots ab}_n$$

Par convention on pose $a^0 = \mathbb{1}$ pour tout a élément de A .

Définition 6 - 3

- On dit que $\mu \in A^*$ est un *préfixe* de $\nu \in A^*$ si, et seulement si, il existe $\xi \in A^*$ telle que $\nu = \mu\xi$. Si $\xi \neq \mathbb{1}$, on dit que μ est un *préfixe propre* de ν .
- On dit que $\mu \in A^*$ est un *suffixe* de $\nu \in A^*$ si, et seulement si, il existe $\xi \in A^*$ telle que $\nu = \xi\mu$. Si $\xi \neq \mathbb{1}$, on dit que μ est un *suffixe propre* de ν .
- On dit que $\mu \in A^*$ est un *facteur* de $\nu \in A^*$ si, et seulement si, il existe $\xi \in A^*$ et $\xi' \in A^*$ telles que $\nu = \xi\mu\xi'$. Si $(\xi, \xi') \neq (\mathbb{1}, \mathbb{1})$, on dit que μ est un *facteur propre* de ν .
- Pour toutes chaînes μ et ν de A^* , le quotient à gauche de ν par μ est noté $\mu^{-1}\nu$ et défini par :

$$\mu^{-1}\nu = \begin{cases} \xi & \text{si } \nu = \mu\xi \\ \text{n'a pas de sens} & \text{si } \mu \text{ n'est pas un préfixe de } \nu \end{cases}$$

On définit de même le quotient à droite.

Si $\mu = aababc$, le quotient à gauche de μ par aab est abc .

Définition 6 - 4

Une chaîne μ étant une suite (finie) de symboles, on appelle *sous-chaîne* de μ toute sous-suite de la suite μ . Tout facteur de μ est une sous-chaîne de μ mais la réciproque est fautive, un sous mot de μ contient des lettres de μ dans le bon ordre mais pas forcément de façon consécutive.

Si $\mu = aabbaabc$, $abba$ est une sous-chaîne qui est aussi un facteur, bbb est une sous-chaîne qui n'est pas un facteur.

Définition 6 - 5

Si $\mu = a_{i_1} a_{i_2} \dots a_{i_n}$ est une chaîne de A^* , sa *transposée* (ou son image miroir) est la chaîne

$${}^t\mu = a_{i_n} a_{i_{n-1}} \dots a_{i_1}$$

Un mot qui est égal à son transposé est appelé *palindrome*.

On notera la propriété :

$${}^t(\mu\nu) = {}^t\nu{}^t\mu$$

1.3 Langages

Définition 6 - 6

Un *langage* sur A est un sous-ensemble de A^* .

Par exemple, sur l'alphabet $A = \{LA, JEUNE, FILLE, \acute{E}TUDIE, LE, BEAU, COURS, .\}$, les ensembles suivants sont des langages :

- $\{LA LA LE, LE FILLE BEAU, \acute{E}TUDIE COURS FILLE LA\}$;
- l'ensemble de toutes les cha\^nes contenant « FILLE » ;
- l'ensemble de toutes les cha\^nes de longueur multiple de 37.

Si nous travaillons sur l'alphabet $B = \{0, 1\}$, alors l'ensemble B^8 des cha\^nes de longueur 8 est l'ensemble bien connu des octets.

Un langage \^tant d\^fini comme un ensemble, il est alors naturel de s'int\^resser aux op\^rations ensemblistes sur les langages. L'union, l'intersection, la compl\^mentation et la diff\^rence s'introduisent naturellement :

Th\^or\^me 6 - 1

Si L et L' sont deux langages sur l'alphabet A (rappel, L et L' sont deux parties de A^*) alors $L \cup L'$, $L \cap L'$, $L - L'$ et $\complement_A L = \bar{L}$ sont des langages sur A .

Notations

La r\^eunion ou l'union $L \cup L'$ est aussi not\^ee $L|L'$ et on l'appelle le plus souvent la somme des langages L et L' .

Nous allons maintenant introduire deux autres op\^rations fondamentales qui vont nous permettre de d\^crire des langages int\^ressants. Ces op\^rations sont le **produit** ou la **concat\^nation** de deux langages et l'**\^toile** ou la **fermeture it\^rative** (appel\^ee aussi **\^toile de Kleene**) d'un langage. Commen\^ons par le **produit** :

D\^finition 6 - 7

Si L et L' sont deux langages sur A , on d\^finit le produit du langage L par le langage L' (attention \^ l'ordre) le langage not\^e LL' ou LL' d\^fini par

$$LL' = \{\mu.\mu' \mid \mu \in L \text{ et } \mu' \in L'\}$$

LL' est le langage obtenu en faisant le produit (concat\^nation) de toutes les cha\^nes de L avec toutes les cha\^nes de L' .

LL' est bien un langage et cette op\^ration est manifestement associative (l'int\^r\^et est de pouvoir se passer de parenth\^ses). Alors :

On d\^finit r\^ecursivement le langage L^n :

D\^finition 6 - 8

$$\begin{cases} L^0 = \{1\} \text{ par convention (m\^eme si } L = \emptyset) \\ L^{n+1} = L^n L \end{cases}$$

Pratiquement L^n n'est autre que l'ensemble des cha\^nes de longueur n construites avec l'« alphabet » L . (on accepte cette vision m\^eme si L est infini).

Par exemple, soient $L_1 = \{ab, ba\}$, $L_2 = \{c, cc\}$ et $L_3 = \{ac\}$. Nous avons

$$L_1L_2 = \{abc, abcc, bac, bacc\}$$

$$L_1L_2L_3 = \{abcac, abccac, bacac, baccac\}$$

$$L_1^2 = \{abab, abba, baab, baba\}$$

Si $n \in \mathbb{N}^*$, $L^{<n}$ désigne le langage

Définition 6 - 9

$$L^{<n} = \bigcup_{i=0}^{n-1} L^i = L^0 \cup L^1 \cup \dots \cup L^{n-1}$$

Pratiquement $L^{<n}$ est l'ensemble des mots de longueur $< n$ que l'on peut construire avec l'alphabet L .

Par exemple, si $L = \{a, ab\}$ alors $L^{<3} = \{\mathbb{1}, a, ab, aa, aab, abab, aba\}$.

On appelle **étoile de KLEENE** ou **itération** du langage L le langage

Définition 6 - 10

$$L^* = \bigcup_{k=0}^{+\infty} L^k = \bigcup_{k \in \mathbb{N}} L^k$$

L'union de toutes les puissances strictement positives de L étant notée L^+ :

$$L^+ = \bigcup_{k \in \mathbb{N}^*} L^k \text{ et } L^* = L^+ \cup \{\mathbb{1}\}$$

Remarque

1. L^* désigne l'ensemble de tous les produits possibles (penser concaténation) d'éléments de L y compris la chaîne vide, pratiquement L^* est le langage défini sur l'alphabet L y compris la chaîne vide.
2. Si $L = \{\mu\}$ où μ est une chaîne de A^* on a $L^* = \{\mu^n \mid n \in \mathbb{N}\}$. Le plus souvent L^* est noté simplement μ^* , en conséquence le langage noté a^*b^* est le langage $L^*L'^*$ avec $L = \{a\}$ et $L' = \{b\}$. Nous utiliserons ces abus de notations par la suite lors de la présentation des expressions rationnelles.
3. Si $L = \{\mathbb{1}\}$ alors $L^* = \{\mathbb{1}\}$. Si $L = \emptyset$ alors $L^* = \{\mathbb{1}\}$ puisque nous avons décidé (par convention) que, pour tout langage L , nous avons $L^0 = \{\mathbb{1}\}$.
4. $L^+ = LL^* = L^*L$
5. $L^* = L^+ \cup \{\mathbb{1}\} = L^+ | \{\mathbb{1}\}$, n'oublions pas que l'union ou la réunion de deux langages est aussi notée « | ».
6. Nous conviendrons qu'en l'absence de parenthèses, l'étoile a priorité sur la concaténation qui a elle-même priorité sur la réunion.

1 4 Langages et expressions rationnels (ou réguliers)

Comme souvent en mathématiques, il n'existe pas UNE SEULE définition d'une notion mais plusieurs définitions équivalentes. C'est à l'auteur du cours de choisir UNE PARMi les différentes définitions possibles : les autres deviendront des propriétés.

1 4 1 Langage rationnel

Définition 6 - 11

Langage rationnel (ou régulier)

Soit L un langage sur l'alphabet A . Il est dit rationnel s'il peut être obtenu récursivement uniquement à l'aide des opérations :

- réunion (somme),
- concaténation (produit),
- étoile de KLEENE

en partant :

- du langage vide \emptyset ,
- des langages du type $\{a\}$ avec $a \in A \cup \{\mathbb{1}_A\}$.

Voici un exemple pour illustrer cette notion.

Considérons l'alphabet $A = \{b, o, x\}$ et L l'ensemble des chaînes finissant par x .

Par exemple, box , $bobox$, $xxbobboxobbxxoxxxx$, x appartiennent à ce langage.

On peut le décrire ainsi :

$$L = (\{b\} \cup \{o\} \cup \{x\})^* \cdot \{x\}$$

C'est donc un langage rationnel car obtenu à partir de singletons et d'opérations rationnelles.

Par abus de notation, on se contentera souvent de l'écrire ainsi :

$$L = (b|o|x)^*x \quad \text{ou} \quad (b^*o^*x^*)^*x \quad \text{ou} \quad \{b, o, x\}^*x$$

Il est ensuite assez évident de montrer les propriétés suivantes :

Propriétés 6 - 1

- Tout langage **fini** est rationnel.
- Si L est rationnel, alors L^* est rationnel.
- Si L_1 et L_2 sont rationnels, alors $L_1|L_2$ et L_1L_2 sont rationnels.

1 4 2 Expression rationnelle

Ouvrons un shell bash dans le répertoire où se trouve un fichier contenant un dictionnaire français où les mots sont écrits un par ligne et suivis éventuellement d'une oblique puis d'un code d'utilisation :

```
a
à/L' D' Q' Qj
a/| |--
ab
abaca/S* ()
abacule/S* ()
abaissable/S* ()
abaissante/F* ()
abaissée/F* ()
abaissée/S* ()
abaisse-langue/L' D' Q'
abaissement/S* ()
abaisser/a4a+ ()
```


abaisseur/S* ()
 abajoue/S* ()
 abalober
 abalone/S* ()

et lançons la commande suivante :

```
$ egrep a[^/]*e[^/]*i[^/]*o[^/]*u ./Dictionnaire.txt
```

On obtient :

```
garde-chiourme
gardes-chiourme
quatre-vingt-douze
```

On obtient alors toutes les lignes où apparaît une chaîne contenant dans cet ordre les symboles a, e, i, o et u.

Le code `[^/]` signifie : « tous les caractères ASCII sauf / », le caractère `*` correspondant à l'étoile de KLEENE.

La commande **egrep** permet en effet d'effectuer une recherche d'*expression rationnelle* dans un fichier. Définissons cette notion :

Expression rationnelle

Soit A un alphabet. Une expression rationnelle définie sur A est :

- \emptyset ,
- $\mathbb{1}_A$,
- un élément quelconque de A,
- des « formules *bien formées* » à partir des éléments de A en utilisant les opérateurs rationnels. Ainsi, si R_1 et R_2 sont des expressions rationnelles, alors $(R_1|R_2)$, $R_1 \cdot R_2$ et R_1^* le sont aussi.

Définition 6 - 12

Disons qu'intuitivement, il s'agit de formules obéissant aux règles de « syntaxe » usuelles. Par exemple, les formules suivantes ne sont pas *bien formées* :

$$a| \quad)a| \star b(\quad | \star a$$

Définissons à présent l'ensemble des chaînes associées à une expression rationnelle de manière récursive :

Langage décrit par une expression régulière

À chaque expression rationnelle E on fait correspondre le langage $\mathcal{L}(E)$ de A^* défini par :

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\mathbb{1}_A) = \{\mathbb{1}_A\}$
- $a \in A, \mathcal{L}(a) = \{a\}$
- $\mathcal{L}(E|E') = \mathcal{L}(E) \cup \mathcal{L}(E') = \mathcal{L}(E) | \mathcal{L}(E')$
- $\mathcal{L}(E \cdot E') = \mathcal{L}(E) \cdot \mathcal{L}(E') = \mathcal{L}(E) \mathcal{L}(E')$
- $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

Propriété 6 - 2

Considérons $E = (a|b)^* \cdot c = (a|b)^* c$, et déterminons le langage $\mathcal{L}(E)$:

$$\begin{aligned}\mathcal{L}((a|b)^* \cdot c) &= \mathcal{L}((a|b)^*) \mathcal{L}(c) \\ \mathcal{L}((a|b)^* \cdot c) &= \mathcal{L}(a|b)^* \mathcal{L}(c) \\ \mathcal{L}((a|b)^* \cdot c) &= (\mathcal{L}(a)|\mathcal{L}(b))^* \mathcal{L}(c) \\ \mathcal{L}((a|b)^* \cdot c) &= (\{a\} \cup \{b\})^* \{c\} \\ \mathcal{L}((a|b)^* \cdot c) &= \{a, b\}^* \{c\} \\ \mathcal{L}((a|b)^* \cdot c) &= \{c, ac, bc, aac, aabbaabaac, bbabababc, \dots\}\end{aligned}$$

Le langage $\mathcal{L}(E)$ est l'ensemble des chaînes de la forme μc où μ est une chaîne quelconque du langage $\{a, b\}^*$ ou, si l'on préfère, μ est la chaîne vide ou une chaîne de longueur quelconque formé à l'aide des symboles a et b dans n'importe quel ordre.

Examinons de plus près l'expression rationnelle $E = (a|b)^* \cdot c$ pour appréhender rapidement le langage $\mathcal{L}(E)$ qu'elle décrit. Cette expression nous donne en fait une règle de construction des chaînes du langage $\mathcal{L}(E)$:

- on la lit de la gauche vers la droite ;
- le signe $|$ correspond au **ou** inclusif. On l'appelle parfois l'*alternative* ;
- le signe \cdot de la concaténation correspond au **et** logique ;
- le symbole \star correspond à une boucle que l'on peut ne pas faire ou faire un nombre quelconque de fois.

Les chaînes de $\mathcal{L}(E)$ vont toutes se terminer par c , les symboles qui vont précéder c sont décrits par $(a|b)^*$. Soit on ne rentre pas dans la boucle et le résultat est $\mathbb{1}$, soit on rentre dans la boucle autant de fois que l'on veut et, à chaque boucle, on choisit a ou bien b .

Inversement, on peut trouver une expression rationnelle qui correspond à un langage. Par exemple, nous pouvons décrire le langage sur $A = \{a, b, c, d\}$ constitué des chaînes qui commencent par c et qui contiennent au moins un a mais pas de d :

$$E = c(a^*b^*c^*)^* a(a^*b^*c^*)^*$$

ou bien

$$E = c(a|b|c)^* a(a|b|c)^*$$

Dans ce qui suit R , S et T sont des expressions rationnelles et l'égalité d'expressions rationnelles signifie qu'elles dénotent le même langage.

Propriétés 6 - 3

- $R|S = S|R, R|R = R$
- $R|\emptyset = R$
- $(R|S)|T = R|(S|T)$
- $R(ST) = (RS)T$
- $R\mathbb{1} = \mathbb{1}R = R$
- $R\emptyset = \emptyset R = \emptyset$
- $R(S|T) = RS|RT, (S|T)R = SR|TR.$
- $R^* = R^*R^* = (R^*)^* = (\mathbb{1}|R)^*$
- $(R|S)^* = (R^*S^*)^* = (R^*|S^*)^* = (R^*S)^*R^*$
- $R^+ = RR^* = R^*R$

Le théorème suivant relie langage et expressions rationnelles :

Théorème 6 - 2

Un langage de A^* est rationnel si, et seulement si, il est dénoté par une expression rationnelle.

En effet, la famille des langages dénotés par une expression rationnelle contient \emptyset , $\{1\}$ et tous les singletons $\{a\}$ avec $a \in A$ puisqu'il a bien été précisé que tout langage fini était rationnel.

Cette famille est fermée pour l'union, le produit et l'étoile, elle contient donc l'ensemble de tous les langages rationnels définis sur A .

Réciproquement tout langage dénoté par une expression rationnelle est obtenue à partir des symboles et de la chaîne vide par une suite finie d'unions, de produits ou d'étoiles, il est donc rationnel.

2 Automates

2.1 Définitions

Nous avons déjà abordé la notion d'automates lors de l'étude des machines de TURING (cf section 5.2.4 page 8).

Nous allons généraliser cette notion grâce à la définition suivante :

Automate fini

Un automate \mathcal{A} est caractérisé par un quintuplet $\langle A, Q, I, T, \tau \rangle$:

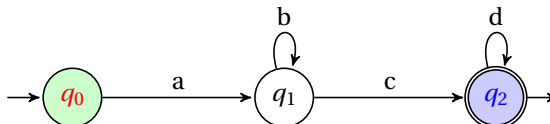
- un alphabet A appelé l'*alphabet d'entrée*;
- un alphabet Q appelé l'*alphabet d'état* (Q est donc **fini**);
- une partie I non vide de Q appelée *ensemble des états initiaux*;
- une partie T non vide de Q appelée *ensemble des états terminaux*;
- une relation τ de $Q \times A$ vers Q ou de $Q \times (A \cup \{1\})$ vers Q , appelée *relation de transition*.

Définition 6 - 13

Par exemple $\tau(q_1, a) = q_2$ signifie que quand l'automate est dans l'état q_1 et qu'il rencontre le symbole a , alors il passe dans l'état q_2 .

On a l'habitude de représenter à l'aide d'un graphe comme nous l'avons déjà vu lors de l'étude des machines de TURING.

Cette fois, les arcs vont porter les symboles de l'alphabet A (ce symbole sera nommé *étiquette de la transition*).



Ici, par exemple, on dit que (q_0, a, q_1) est une *transition* de l'automate, que a est l'*étiquette* de cette transition dont l'*origine* est q_0 et l'*extrémité terminale* est q_1 .

On pourra aussi noter cette transition $q_0 \xrightarrow{a} q_1$.

On peut aussi représenter l'automate par sa *table de transition* :

| μ | q_0 | q_1 | q_2 |
|-------|-------|-------|-------|
| a | q_1 | | |
| b | | q_1 | |
| c | | q_2 | |
| d | | | q_2 |

2 2 Langage reconnaissable

Reprenons l'automate précédent défini sur l'alphabet $A = \{a, b, c, d\}$.

Les chaînes ac , $abbbbcb$, $abbbbcbddddd$ semblent calculables par l'automate mais pas les chaînes ad , $ddddaa$, $aaabc$. L'automate étant simple, c'est assez facile à « voir ». Dans des cas plus compliqués, les mathématiques prendront le relais de nos yeux. Fabriquons donc nos « lunettes ».

Un **calcul c** de l'automate \mathcal{A} (ou une exécution ou une trace de l'automate \mathcal{A}) est une suite de transitions (des relations de $Q \times A$ dans Q) telles que l'origine de chacune (sauf la première) coïncide avec l'extrémité terminale de la précédente :

$$\mathbf{c} = q_{i_0} \xrightarrow{\alpha_1} q_{i_1} \xrightarrow{\alpha_2} q_{i_2} \xrightarrow{\alpha_3} \dots \xrightarrow{\alpha_n} q_{i_n}$$

que l'on note aussi

$$\mathbf{c} = q_{i_0} \xrightarrow{\alpha_1 \alpha_2 \dots \alpha_n} q_{i_n}$$

la longueur de ce calcul étant n (il y a n transitions) et $\alpha_1 \alpha_2 \dots \alpha_n$ est l'étiquette du calcul \mathbf{c} , c'est une chaîne de A^* . Il faut remarquer qu'un calcul a une seule étiquette mais qu'une même chaîne peut être l'étiquette de plusieurs calculs distincts.

Un calcul de \mathcal{A} est **réussi** si son origine est un état initial et son extrémité terminale un état final.

Une chaîne de A^* est donc reconnue par l'automate \mathcal{A} si, et seulement si, c'est l'étiquette d'un calcul réussi de \mathcal{A} .

Ici, $q_0 \xrightarrow{abbbbcbdd} q_2$ donc la chaîne $abbbbcbdd$ est reconnue par \mathcal{A} .

À retenir

Si l'automate se bloque pendant la lecture de la chaîne ou n'atteint pas l'état final, cette chaîne n'est pas reconnue par l'automate.

Langage reconnaissable par un automate

On note $\mathcal{L}(\mathcal{A})$ l'ensemble des chaînes reconnues par l'automate :

$$\mathcal{L}(\mathcal{A}) = \left\{ \mu \in A^* \mid \exists q_i \in I, \exists q_t \in T, q_i \xrightarrow{\mu} q_t \right\}$$

et on dit qu'un langage L est **reconnaisable par l'automate** \mathcal{A} ou que l'automate \mathcal{A} **reconnait** le langage L si, et seulement si, $\mathcal{L}(\mathcal{A}) = L$.

Un langage est **reconnaisable** si, et seulement si, il existe AU MOINS un automate FINI qui le reconnaît.

L'ensemble des langages reconnaissables sur l'alphabet A est noté **Rec**(A^*).

2 3 Langage reconnu par un automate fini

Une première question qui vient à l'esprit en ayant le graphe d'un automate sous les yeux : quel est le langage reconnu par cet automate ?

2 3 1 Méthode « manuelle »

On essaie de déterminer pas à pas la succession de langages permettant de passer des états initiaux aux états terminaux.

En utilisant toujours notre exemple, on voit qu'avec a on passe à l'état q_1 . On peut y rester en bouclant avec des b ou passer à l'état terminal q_2 avec c en bouclant éventuellement avec des d .

Le langage reconnu peut donc s'écrire à l'aide de l'expression rationnelle ab^*cd^* .

2 3 2 Technique d'élimination d'états

On tente cette fois d'éliminer les états intermédiaires pour obtenir un automate à deux états étiqueté par une expression déterminant le langage reconnu. On généralise ainsi les automates étiquetés par des symboles ou des chaînes.

Cette technique est dérivée de l'algorithme de MAC NAUGHTON et YAMADA.

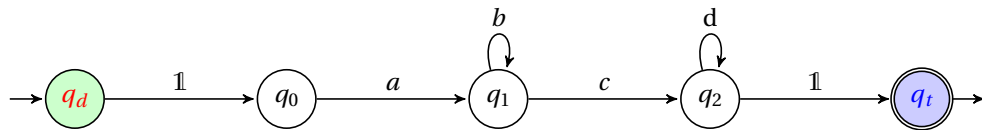
- On commence par créer deux nouveaux états q_d et q_t qui seront les uniques états initiaux et terminaux.

On relie q_d à tous les états initiaux par une transition étiquetée par $\mathbb{1}$ et on relie tous les états terminaux à q_t par le même type de transitions.

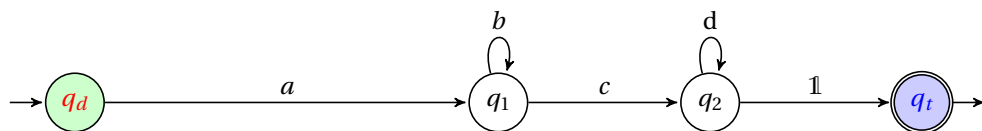
- Notons Q l'ensemble tel que l'alphabet d'états soit $\{q_d\} \cup Q \cup \{q_t\}$. Tant que Q n'est pas vide, on applique l'opération suivante : pour tout couple (p, r) d'états tels qu'il existe une transition directe de p à q et de q à r étiquetée respectivement par L_{pq} et L_{qr} , on crée une transition (p, L_{pqr}, r) avec $L_{pqr} = L_{pr} \cup L_{pq}L_{qr}^*L_{qr}$.

Reprenons notre automate de départ.

- on ajoute q_d et q_f :



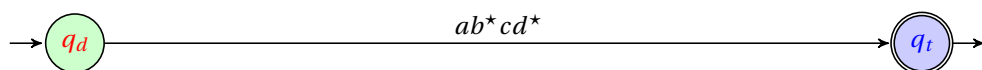
- on élimine l'état q_0 : $L_{d01} = L_{d1} \cup L_{d0}L_{00}^*L_{01} = \emptyset \cup \mathbb{1}\mathbb{1}^*a = a$



- on élimine l'état q_1 : $L_{d12} = L_{d2} \cup L_{d1}L_{11}^*L_{12} = \emptyset \cup ab^*c = ab^*c$



- on élimine l'état q_2 : $L_{d2t} = L_{dt} \cup L_{d2}L_{22}^*L_{2t} = \emptyset \cup ab^*cd^*\mathbb{1} = ab^*cd^*$



2 4 Automates équivalents

Théorème 6 - 3

Automates équivalents

On définit une relation \equiv sur l'ensemble des automates finis.

Deux automates \mathcal{A} et \mathcal{A}' sont en relation si, et seulement si, ils reconnaissent le même langage.

Cette relation est une relation d'équivalence. On note $\mathcal{A} \equiv \mathcal{A}'$ pour signifier que deux automates sont équivalents.

Il est assez évident de démontrer que la relation \equiv est une relation d'équivalence. Cette notion est très importante en mathématiques et très utilisée en informatique. Au lieu de considérer les automates finis dans leur ensemble, nous allons nous contenter de parler des **classes d'équivalence** qui sont les éléments de l'ensemble des automates quotienté par \equiv .

Aparté

Pensez justement aux fractions rationnelles qui renvoient naturellement à la notion de quotient : $\frac{1}{2}, \frac{2}{4}, \frac{3}{6}$, etc. sont des « représentants » de la classe d'équivalence de $(1, 2)$ pour la relation définie par $(a, b)\mathcal{R}(x, y) \Leftrightarrow ay = bx$ sur $\mathbb{Z} \times \mathbb{Z}$. L'ensemble quotient $\mathbb{Z} \times \mathbb{Z} / \mathcal{R}$ est justement l'ensemble... \mathbb{Q} !

La classe de $(1, 2)$ est $\{\dots, \frac{-2}{-4}, \frac{-1}{-2}, \frac{1}{2}, \frac{2}{4}, \frac{3}{6}, \dots\}$.

Deux automates pourront donc « avoir l'air » différents mais représenteront le même langage. Comme on préfère travailler avec $\frac{1}{2}$ plutôt qu'avec $\frac{2134}{4268}$, on essaiera de simplifier au maximum un automate pour débuser plus facilement le langage qu'il reconnaît.

2 5 Automates standards

Définition 6 - 15

Automate standard

Un automate $\langle A, Q, D, T, \tau \rangle$ est standard si, et seulement si, D est réduit à un singleton $\{d\}$ et aucune transition n'aboutit en d .

L'algorithme suivant va nous permettre de rendre standard tout automate fini. Notons \mathcal{A}' l'automate standardisé, alors $\mathcal{A}' = \langle A, Q \cup \{d\}, \{d\}, T', \tau' \rangle$ avec :

- $d \notin Q$;
- si $D \cap T = \emptyset$ alors $T' = T$ sinon $T' = T \cup \{d\}$;
- $\mathcal{G}_{\tau'} = \mathcal{G}_{\tau} \cup \{(d, a, q) \mid \exists q_d \in D, (q_d, a, q) \in \mathcal{G}_{\tau}\}$

Cela nous permet d'énoncer le théorème suivant :

Théorème 6 - 4

Tout automate fini est équivalent à un automate standard.

Ou bien « toute classe d'équivalence pour \equiv contient un automate standard ».

Recherche

Standardisez l'automate défini sur l'alphabet $\{a, b\}$ par la table de transition suivante :

| τ | a | b |
|--------|-------|------------|
| q_0 | q_1 | |
| q_1 | q_1 | q_0, q_2 |
| q_2 | q_0 | q_2 |

avec $D = Q$ et $T = \{q_0\}$.

2 6 Automates déterministes

Définition 6 - 16

Un automate *déterministe* :

- a un unique état initial ;
- la relation de transition τ est une fonction : il y a, à partir de chaque état, au plus une transition d'étiquette donnée.

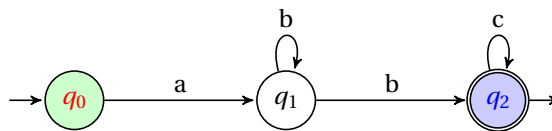
Si un automate n'est pas déterministe, il est...non-déterministe.

Définition 6 - 17

Un automate *déterministe complet* :

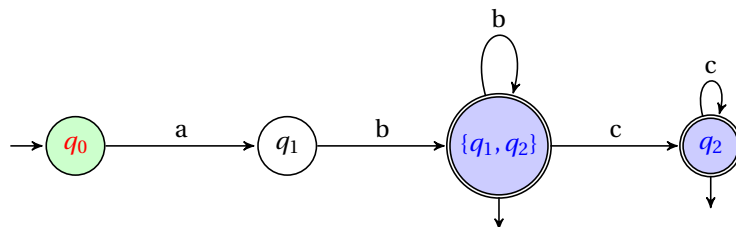
- est un automate déterministe ;
- la relation de transition τ est une application : il y a, à partir de chaque état, une et une seule transition d'étiquette donnée.

Par exemple :



n'est pas déterministe car (q_1, b) admet deux images par τ qui sont q_1 et q_2 .

Il est possible de rendre cet automate déterministe en regroupant toutes les images d'un couple donné en un seul état :



Théorème 6 - 5

Tout automate fini est équivalent à un automate déterministe.

ou bien, « toute classe d'équivalence pour \equiv » contient un automate déterministe.

Le problème d'un automate non déterministe est qu'il est moins efficace. Dans le cas d'un automate déterministe, pour chaque chaîne reconnue, il existe un unique calcul dont cette chaîne est l'étiquette.

Voici un algorithme qui permet de manière assez économique d'obtenir un automate déterministe équivalent à un automate donné.

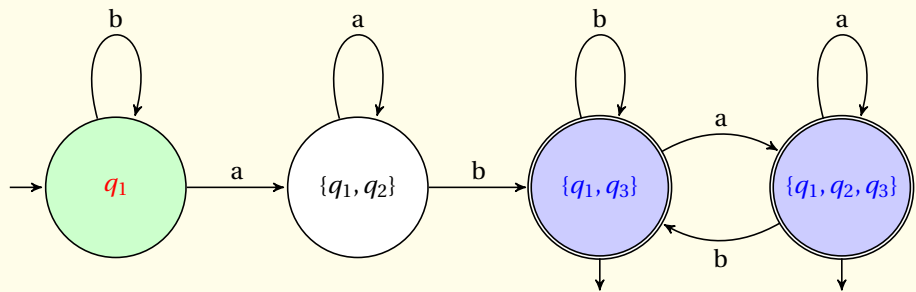
Considérons un automate non déterministe $\mathcal{A} = \langle A, Q, \{d\}, T, \tau \rangle$ mais standard et notons $\mathcal{A}^D = \langle A, Q^D, \{d\}, T^D, \tau^D \rangle$ l'automate construit de la manière suivante :

- au départ, $Q^D \leftarrow \{d\}$;
- on détermine les images non vides de chaque couple $(\{d\}, a)$ pour chaque a dans A . On obtient de nouveaux états q_i^D que l'on met dans Q^D ;
- on recommence ainsi avec les images des (q_i^D, a) pour chaque a dans A ;
- on s'arrête quand on ne crée plus de nouveaux états;
- T^D est l'ensemble des q_i^D qui contiennent un état terminal de \mathcal{A} .

Vérifiez que l'algorithme précédent permet de montrer que l'automate défini par :

| τ | a | b |
|--------|------------|-------|
| q_1 | q_1, q_2 | q_1 |
| q_2 | | q_3 |
| q_3 | q_3 | q_3 |

avec $I = \{q_1\}$ et $T = \{q_3\}$ est équivalent à :



Recherche

2 7 Automates émondés

Ouvrons le dictionnaire :

ÉMONDER v. tr. (lat. *emundare*, de *mundus*, propre; v, 1200, « purifier »). **1.** (1354). Couper les branches inutiles : *Émonder un peuplier*. — **2.** Débarrasser du superflu : *Émonder un récit des détails inutiles*.

Tout est dit...Enfin, pour avoir quelque chose de plus rigoureux afin de l'utiliser informatiquement, donnons quelques définitions intermédiaires.

États accessibles et co-accessibles

Soit $\mathcal{A} = \langle A, Q, D, T, \tau \rangle$ un automate fini.

- On dit que l'état q_j ($q_j \in Q$) est **accessible à partir d'un état** q_i ($q_i \in Q$) s'il existe (au moins) un calcul c dans \mathcal{A} dont l'origine est q_i et l'extrémité est q_j . On dit que l'état q_j est **accessible** s'il est accessible à partir d'un état initial.
- Un état q_i est **co-accessible à un état** q_j s'il existe un calcul c dans \mathcal{A} dont l'origine est q_i et l'extrémité est q_j . On dit que l'état q_i est **co-accessible** s'il est co-accessible d'un état final.
- On dit qu'un état q_i est **utile** s'il est à la fois accessible et co-accessible.

Définition 6 - 18

Ceci nous permet de définir un automate émondé :

Automate émondé

Un automate émondé est un automate dont tous les états sont utiles.

Définition 6 - 19

On peut définir deux algorithmes servant à émonder un automate donné. Il s'agit de supprimer les états non accessibles ou non co-accessibles et les transitions qui en dépendent. On considère toujours notre automate fini $\mathcal{A} = \langle A, Q, D, T, \tau \rangle$.

```

Fonction émondage aller( $\mathcal{A}$ : automate fini): automate fini
Début
    E ← D
    E' ← ∅
    TantQue E' ≠ E Faire
        E' ← E
        E ← E ∪ { $q \in Q \mid \exists e \in E, \exists a \in A: (e, a, q) \in \mathcal{G}_\tau$ }
    Fin TantQue
    Retourner  $\langle A, E, D, T, \tau' \rangle$ 
Fin
    
```

```

Fonction émondage retour( $\mathcal{A}$ : automate fini): automate fini
Début
    C ← T ∩ E
    C' ← ∅
    TantQue C' ≠ C Faire
        C' ← C
        C ← C ∪ { $q \in Q \mid \exists c \in C, \exists a \in A: (q, a, c) \in \mathcal{G}_\tau$ }
    Fin TantQue
    Retourner  $\langle A, C, D, T, \tau'' \rangle$ 
Fin
    
```

Recherche

Standardisez et émondez l'automate défini sur l'alphabet $\{a, b\}$ à l'aide de la table de transition suivante :

| τ | a | b |
|--------|------------|-------|
| q_1 | q_1, q_2 | |
| q_2 | | q_3 |
| q_3 | q_3 | q_3 |
| q_4 | q_3 | q_2 |

avec $D = \{q_1, q_3\}$ et $T = \{q_1, q_2\}$

On peut montrer qu'éliminer les états qui ne sont pas utiles ne change pas le langage reconnu par un automate et donc :

Théorème 6 - 6

Tout automate fini est équivalent à un automate émondé

ou bien « toute classe d'équivalence pour \equiv contient un automate émondé ».

En fait, émonder un automate déterministe, c'est tous les états inaccessibles à partir de l'état initial et tous ceux à partir desquels on ne peut pas atteindre un état terminal. C'est assez logique...

2 8 Automate minimal

2 8 1 Théorème de Myhill-Nerode

Minimiser un automate consiste à regrouper deux états q et q' si ce sont les mêmes chaînes qui permettent d'aller de q et de q' respectivement vers un état final. Il faudrait être sûr que cet automate (qui est équivalent à l'automate d'origine de manière évidente) est celui qui possède le plus petit nombre d'états parmi tous les éléments de sa classe d'équivalence.

Dans toute la suite on considère un alphabet A , un langage L sur A^* et un automate *déterministe* $\mathcal{A} = \langle A, Q, \{d\}, T, \tau \rangle$ qui reconnaît L .

États distinguables

Deux états q_i et q_j dans Q sont distinguables par la chaînes $\mu \in A^*$ si :

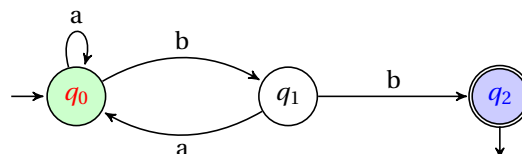
- $\tau(q_i, \mu) \in T$ et $\tau(q_j, \mu) \notin T$
- $\tau(q_i, \mu) \notin T$ et $\tau(q_j, \mu) \in T$

Deux états sont **indistinguables** s'ils ne sont distinguables par aucune chaîne.

Définition 6 - 20

Par exemple, q_0 et q_1 sont-ils distinguables par a ? Par b ?

Par quelle chaîne q_1 et q_2 sont-ils distinguables?



Équivalence de NERODE

On définit la relation :

$$q_i \equiv_N q_j \Leftrightarrow q_i \text{ et } q_j \text{ sont indistinguables}$$

C'est une classe d'équivalence.

Théorème 6 - 7

Les classes d'équivalence de cette relation sur Q réalisent une partition de Q. Les différentes classes d'équivalence permettent de définir un nouvel ensemble d'états Q' d'un nouvel automate $\mathcal{A}' = \langle A, Q', \{d\}, T', \tau' \rangle$ avec :

- $\forall q \in Q, \tau(q, a) = q_i \implies \tau'(q', a) = q'_i$ avec q' et q'_i les classes d'équivalences de q et q_i ;
- d' est la classe de d ;
- T' est l'ensemble des classes des éléments de T.

Théorème de MYHILL-NERODE (1958)

Soit L un langage reconnaissable. Parmi tous les automates déterministes finis qui reconnaissent L, il en existe un et un seul (au nom des états près) qui a un nombre minimum d'états. C'est l'*automate minimal* de L.

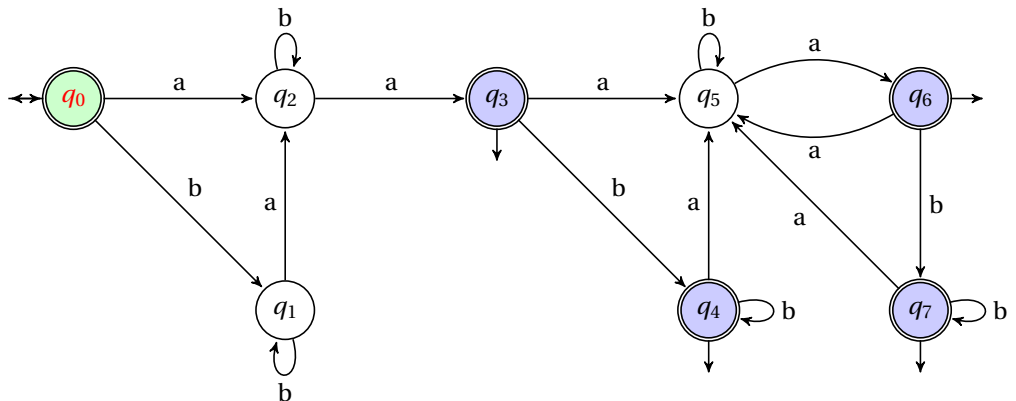
Théorème 6 - 8

On pourrait démontrer que l'algorithme minimal est celui défini par la relation d'équivalence de NERODE.

Nous allons plutôt proposer un **algorithme de minimisation** qui va nous permettre de construire l'automate minimal d'un automate déterministe fini donné.

2 8 2 Algorithme de minimisation de Moore

Considérons l'exemple suivant :



| | q_0 | q_1 | q_2 | q_3 | q_4 | q_5 | q_6 | q_7 |
|---|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | 1 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |

La classe 1 regroupe en fait les états terminaux et la classe 2 les autres.

On regarde maintenant pour chaque état vers quelle classe mène la transition par un caractère de A, à raison d'une ligne par caractère :

| | q_0 | q_1 | q_2 | q_3 | q_4 | q_5 | q_6 | q_7 |
|--------------|-------|-------|-------|-------|-------|-------|-------|-------|
| $\mathbb{1}$ | 1 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |
| a | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 |
| b | 2 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |

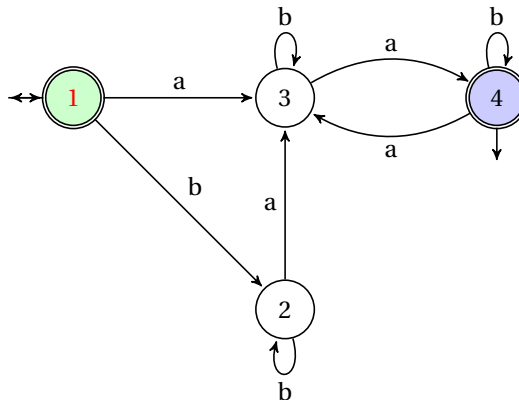
On regarde maintenant colonne par colonne et on crée une nouvelle classe dès qu'on rencontre une colonne différente :

| | q_0 | q_1 | q_2 | q_3 | q_4 | q_5 | q_6 | q_7 |
|--------------|-------|-------|-------|-------|-------|-------|-------|-------|
| $\mathbb{1}$ | 1 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |
| a | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 |
| b | 2 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |
| | 1 | 2 | 3 | 4 | 4 | 3 | 4 | 4 |

On réitère l'opération tant qu'on obtient des lignes de bilan différentes :

| | q_0 | q_1 | q_2 | q_3 | q_4 | q_5 | q_6 | q_7 |
|--------------|-------|-------|-------|-------|-------|-------|-------|-------|
| $\mathbb{1}$ | 1 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |
| a | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 |
| b | 2 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |
| | 1 | 2 | 3 | 4 | 4 | 3 | 4 | 4 |
| a | 3 | 3 | 4 | 3 | 3 | 4 | 3 | 3 |
| b | 2 | 2 | 3 | 4 | 4 | 3 | 4 | 4 |
| | 1 | 2 | 3 | 4 | 4 | 3 | 4 | 4 |

On s'arrête. Les transitions sont même données par la pénultième et l'antépénultième ligne.



Remarques

- Si l'automate n'est pas complet, on marquera les transitions manquantes par \emptyset .
- l'unicité de l'automate minimal permet de comparer deux automates : s'ils ont le même automate minimal, ils sont équivalents.
- on peut appliquer l'algorithme de MOORE pour vérifier qu'un automate est minimal.

2 9 Opérations rationnelles sur les automates

2 9 1 Union

Soit $\mathcal{A}_1 = \langle A, Q_1, D_1, T_1, \tau_1 \rangle$ et $\mathcal{A}_2 = \langle A, Q_2, D_2, T_2, \tau_2 \rangle$ deux automates finis reconnaissant deux langages L_1 et L_2 . On peut supposer que $Q_1 \cap Q_2 = \emptyset$ quitte à renuméroter les états (en effet les noms des états n'ont aucune importance). Alors on définit un nouvel automate

$$\mathcal{A}_1 | \mathcal{A}_2 = \langle A, Q_1 \cup Q_2, D_1 \cup D_2, T_1 \cup T_2, \tau_1 \cup \tau_2 \rangle$$

en notant abusivement $\tau_1 \cup \tau_2$ la relation de graphe $\mathcal{G}_{\tau_1} \cup \mathcal{G}_{\tau_2}$.

Théorème 6 - 9

L'automate $\mathcal{A}_1 | \mathcal{A}_2$ reconnaît le langage $L_1 | L_2$.

En général, il est ensuite nécessaire de rendre cet automate standard et émondé.

2 9 2 Produit

Soit $\mathcal{A}_1 = \langle A, Q_1, D_1, T_1, \tau_1 \rangle$ et $\mathcal{A}_2 = \langle A, Q_2, D_2, T_2, \tau_2 \rangle$ deux automates finis reconnaissant deux langages L_1 et L_2 . On peut supposer que $Q_1 \cap Q_2 = \emptyset$. Quitte à standardiser \mathcal{A}_2 , on peut supposer que $D_2 = \{d_2\}$.

On définit alors un nouvel automate :

$$\mathcal{A}_1 \cdot \mathcal{A}_2 = \langle A, (Q_1 \cup Q_2) \setminus \{d_2\}, D_1, T', \tau_1 \cup \tau'_2 \cup \tau''_2 \rangle$$

avec :

- $T' = (T_1 \cup T_2) \setminus \{d_2\}$ si $d_2 \in T_2$ et $T_1 \cup T_2$ sinon ;
- $\mathcal{G}_{\tau'_2} = \{(q, a, q') \in \text{GR}_{\tau_2} \mid q \neq d_2\}$;
- $\mathcal{G}_{\tau''_2} = \{(q_{t_1}, a, q') \mid q_{t_1} \in T_1, (d_2, a, q') \in \mathcal{G}_{\tau_2}\}$.

Recherche

Comment décrire concrètement la construction de $\mathcal{A}_1 \cdot \mathcal{A}_2$?

Théorème 6 - 10

L'automate $\mathcal{A}_1 \cdot \mathcal{A}_2$ reconnaît le langage $L_1 \cdot L_2$.

2 9 3 Étoile

Soit $\mathcal{A} = \langle A, Q, \{d\}, T, \tau \rangle$ un automate fini *standard* reconnaissant un langage L . On définit alors un nouvel automate

$$\mathcal{A}^* = \langle A, Q, \{d\}, T \cup \{d\}, \tau' \rangle$$

avec $\mathcal{G}_{\tau'} = \mathcal{G}_{\tau} \cup \{(q_t, a, q) \mid q_t \in T, (d, a, q) \in \mathcal{G}_{\tau}\}$

Recherche

Comment décrire concrètement la construction de \mathcal{A}^* ?

Théorème 6 - 11

L'automate \mathcal{A}^* reconnaît le langage L^* .

2 10 Théorème de Kleene

Nous pouvons déduire de ce qui précède que tout langage rationnel est reconnaissable par un automate.

En fait, la réciproque est vraie aussi mais assez compliquée à démontrer. Ces deux résultats constituent le théorème de KLEENE :

Théorème 6 - 12

Théorème de KLEENE (1956)

Pour tout alphabet A , $Rat(A^*) = Rec(A^*)$.

Ainsi, si un langage peut être décrit par une expression rationnelle, alors il existe un automate fini qui le reconnaît.

Réciproquement, si un langage est reconnaissable, alors il existe une expression rationnelle qui le décrit.

Notez bien que certains langages ne sont pas reconnaissables, comme par exemple l'ensemble des chaînes sur $\{A, B\}$ constituées d'autant de a que de b . En raisonnant par l'absurde, on peut montrer qu'aucun automate fini ne peut reconnaître ce langage.

2 11 Automate associé à un langage défini par une expression rationnelle

2 11 1 Méthode « manuelle »

Une expression rationnelle correspond à une construction récursive à partir d'opérations simples sur des langages réduits à des singletons et les constructions précédentes vont nous permettre de construire petit à petit un automate fini correspondant.

2 11 2 Algorithme de Gloushkov

La méthode suivante est attribuée à l'informaticien soviétique Viktor GLOUSHKOV qui l'a proposée en 1961. Il construit un automate ayant un état de plus que le nombre de symboles qui apparaissent dans l'expression rationnelle.

On *linéarise* l'expression rationnelle, i.e., on remplace tous les caractères par des symboles distincts, numérotés à partir de 1 de la gauche vers la droite.

Par exemple, $ab^*|a^2b^+$ devient $x_1x_2^*|x_3x_4x_5^+$.

On crée ensuite un état par variable ainsi qu'un état q_0 qui représente l'état initial.

On crée l'ensemble *posliedny* (dernier en russe) qui contient tous les caractères pouvant terminer une chaîne plus éventuellement l'état q_0 si $\mathbb{1}$ appartient au langage : il constitue l'ensemble des états terminaux de l'automate.

On crée l'ensemble *piervy* (premier en russe) des caractères pouvant commencer un mot : on relie l'état q_0 à chaque q_i de cet ensemble en l'étiquetant par x_i .

On crée une transition étiquetée par x_j de l'état q_i vers l'état q_j si le facteur $x_i x_j$ peut apparaître dans au moins une chaîne du langage.

Finalement, on remplace les x_i par les caractères qu'ils remplaçaient.

Il ne reste plus qu'à s'occuper d'une vingtaine d'exemples...

2 12 Automates séquentiels

Pour modéliser de nouvelles situations, il faut sortir du cadre restreint des automates à états finis sans sortie. Voici une première catégorie qui diffère peu de la précédente, à un petit

détail près : ils produisent des chaînes de sortie.

Un automate à états finis avec sorties est caractérisé par un sextuplet $\langle Q, q_0, A_I, A_S, f_t, f_s \rangle$ où :

- Q est un ensemble fini d'états ;
- q_0 est l'état initial ;
- A_I est l'alphabet d'entrée ;
- A_S est l'alphabet de sortie ;
- f_t est la fonction de transition de $Q \times A_I$ vers Q ;
- f_s est la fonction de sortie de $Q \times A_I$ vers A_S .

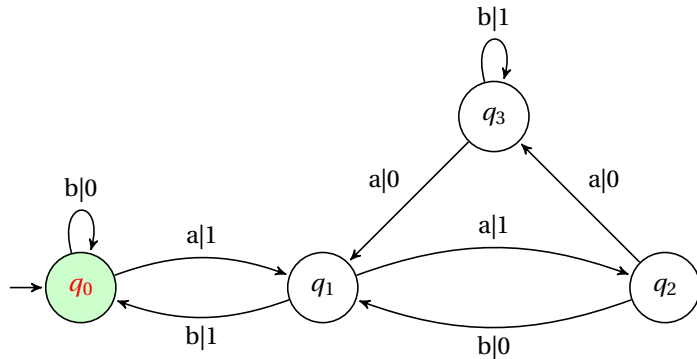
Définition 6 - 21

Vous aurez bien noté que l'automate ne contient qu'un seul état initial et que les fonctions de transition et de sortie sont des...fonctions, if you see what I mean...

Remarque

On peut également définir un automate séquentiel avec des états terminaux : une chaîne ne sera produite que si l'on atteint un état terminal.

Pour représenter l'automate par un diagramme, on étiquette chaque transition par un caractère d'entrée suivi du caractère de sortie correspondant.



Par exemple, la chaîne d'entrée babaaaaabbba donne en sortie 0111100101110.

2 13 Automates à pile

On a déjà évoqué le cas du langage $\{0^n 1^n \mid n \in \mathbb{N}\}$ qui ne peut être reconnu par aucun automate fini. Il existe des automates plus généraux qui reconnaissent ce genre de langages. Pour définir un automate à pile nous avons besoin :

1. d'un alphabet $A = \{..., \alpha_z, ...\}$, rappelons que A est forcément fini et non vide ; c'est sur cet alphabet que nous allons construire des mots ;
2. de savoir ce qu'est une pile, bien que cette pile soit rechargeable, elle ne peut en aucun cas être utilisée pour remplacer la pile de votre calculatrice le jour d'un examen. Pour simplifier une pile est une liste ordonnée d'éléments - cette liste peut être vide - sur laquelle deux opérations élémentaires peuvent être réalisées :
 - insertion d'un élément au sommet de la pile (empiler)
 - suppression de l'élément au sommet de la pile (dépiler)
 il est alors clair que le premier élément empilé sera le dernier à être dépilé ; C'est pourquoi « pile » est un synonyme de FILO (First In Last Out) ou de LIFO (Last In First Out).

3. d'un alphabet de pile $P = \{p_1, p_2, p_3, \dots\}$, c'est tout simplement un ensemble fini de symboles spécifiques à la pile ; la pile va nous servir à mémoriser des étapes et, pour savoir où on en est ou ce qui a été fait avant, on va regarder l'état de la pile pour continuer ;
4. d'un ensemble d'états $E = \{e_1, e_2, \dots\}$, $s \in E$ est l'unique état initial de l'automate ; on note T la partie de E constituée des états terminaux, on dit aussi acceptants ;
5. μ est la relation de transition, c'est une relation :

$$(E \times (P \cup \{\mathbb{1}\})) \times (A \cup \{\mathbb{1}\}) \longrightarrow E \times (P \cup \{\mathbb{1}\})$$

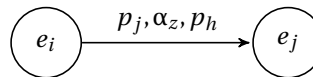
une transition est alors assimilée à un triplet (comme pour les automates à états finis). Si (e_k, p_h) est une image de $((e_i, p_j), \alpha_z)$ par μ on note

$$((e_i, p_j), \alpha_z, (e_k, p_h))$$

qui se lit : « si l'automate est dans l'état e_i et que le sommet de la pile est p_j et que l'automate lit α_z alors l'automate passe dans l'état e_k , dépile p_j de la pile et empile p_h ». Si $p_j = \mathbb{1}$ on ne s'occupe pas de savoir ce qu'il y a au sommet de la pile et on ne dépile rien, si $p_h = \mathbb{1}$ cela indique que l'on n'empile rien. La possibilité de pouvoir avoir $\alpha_z = \mathbb{1}$ nous permet, en cas de besoin, de modifier l'état de l'automate et du sommet de pile.

On le remarque immédiatement, un état d'un automate à pile n'est jamais isolé dans une transition, il doit être associé à un symbole de pile (ou $\mathbb{1}$), les états de notre automate sont alors des couples.

Comme pour les automates à états finis, α_z est l'étiquette de la transition :



On généralise cette définition en autorisant dans les transitions l'empilement d'une chaîne de plusieurs symboles de la pile :

$$((e_i, p_j), \alpha_z, (e_k, M))$$

ici on va empiler tous les symboles de M qui est un mot sur P l'alphabet de pile.

On peut, au lieu de présenter les transitions sous la forme $((e_i, p_j), \alpha_z, (e_k, p_h))$, utiliser lorsque μ est une fonction

$$\mu(e_i, p_j, \alpha_z) \text{ ou } \mu((e_i, p_j), \alpha_z) = (e_k, p_h)$$

qui se lit évidemment encore : si l'automate est dans l'état e_i et qu'il y a p_j en sommet de pile (si $p_j = \mathbb{1}$ on ne s'occupe pas du sommet de pile et on ne dépilera rien) et que l'automate lit α_z alors il passe dans l'état e_k , on dépile p_j et on empile p_h .

On dit qu'un mot m de A^* est reconnu par l'automate s'il existe une suite de transitions partant de l'état initial et **pile vide** dont la suite des étiquettes est m et aboutissant à un **état final** ou acceptant et **pile vide**. L'ensemble des mots reconnus par l'automate à pile est le langage engendré ou reconnu par cet automate.

Considérons l'automate défini par

$$A = \{a, b\}, E = \{s, 1, 2\}, T = \{2\}, P = \{u\}$$

et les transitions

$$T_1 : ((s, \mathbb{1}), a, (1, u))$$

$$T_2 : ((1, \mathbb{1}), a, (1, u))$$

$$T_3 : ((1, u), b, (2, \mathbb{1}))$$

$$T_4 : ((2, u), b, (2, \mathbb{1}))$$

T_1 : pour démarrer, l'automate doit être dans l'état initial s (c'est le seul état initial) et la pile doit être vide, si c'est le cas et qu'il lit a alors il passe dans l'état 1, on ne dépile rien ($\mathbb{1}$) et on empile u .

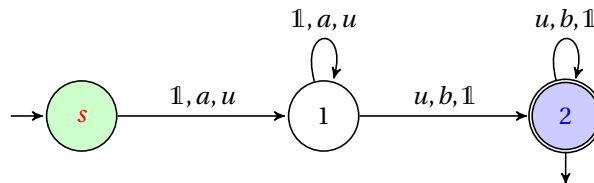
T_2 : si l'automate est dans l'état 1 et qu'il lit a alors il reste dans l'état 1, on ne dépile rien ($\mathbb{1}$) et on empile u .

T_3 : si l'automate est dans l'état 1 et que le sommet de la pile est u et qu'il lit b alors il passe dans l'état 2, on dépile u (le sommet de la pile) et on n'empile rien ($\mathbb{1}$).

T_4 : si l'automate est dans l'état 2 et que le sommet de la pile est u et qu'il lit b alors il passe (il reste) dans l'état 2, on dépile u (le sommet de la pile) et on n'empile rien ($\mathbb{1}$).

Vous aurez remarqué que la relation μ est une fonction, on dit dans ce cas que l'automate à pile est déterministe.

L'état de sortie de l'automate est ici l'état 2 ($T = \{2\}$) mais on ne peut utiliser cette sortie qu'au moment où la pile est vide.



On remarque assez facilement qu'à chaque fois que l'automate lit un a , on empile un u , il y aura alors dans la pile autant de u que de a lus ; à chaque fois que l'automate lit un b on dépile un u (si ce n'est pas possible, l'automate ne peut lire b), en conséquence l'automate sera dans l'état 2 avec la pile vide lorsqu'il aura lu autant de b que de a . Le langage reconnu par cet automate est $\{a^n b^n \mid n \in \mathbb{N}^*\}$, la pile nous permet de mémoriser le nombre de a lus ce que ne permet pas un automate à états finis.

Si nous décidons que s est aussi un état final et comme au départ la pile est vide, on conclut, comme pour les automates à états finis, que $\mathbb{1}$ fait partie du langage reconnu par l'automate.

Si on veut que la pile soit vide à un certain état e_i , pour vider la pile il suffit de rajouter les transition $((e_i, p), \mathbb{1}, (e_i, \mathbb{1}))$ avec p un symbole de pile quelconque.

2 14 Machines de Turing : le retour

On peut montrer qu'un automate à pile ne peut reconnaître le langage $\{0^n 1^n 2^n \mid n \in \mathbb{N}\}$: il faut imaginer quelque chose de plus puissant comme...les machines de TURING. La boucle est bouclée, les machines de TURING sont l'alpha et l'oméga de l'informatique théorique...

3 1 Syntaxe et sémantique

Pour commencer dans la joie, chantons gaiement :

*Mon père est marinier
 Dans cette péniche
 Ma mère dit : « La paix niche
 Dans ce mari niais »
 Ma mère est habile
 Mais ma bile est amère
 Car mon père et ses verres
 Ont les pieds fragiles*



Boby LAPOINTE (1969)

Boby LAPOINTE, ingénieur de Sup'Aero et chanteur à succès des années 1960, a su jouer entre syntaxe et sémantique comme dans la célèbre chanson **mon père et ses verres**.

L'ambiguïté n'est ici qu'orale. Elle peut être aussi écrite : « le vendredi matin, l'étudiant sent la rose ».

D'autres, selon le contexte, peuvent être non ambiguës : « Tous les matins, je mange un avocat ».

Il y a d'autres phrases non ambiguës mais au sens douteux : « Tous les avocats, je mange la Pologne ».

Encore un type d'ambiguïté : « Je regarde cette jolie fille qui se promène avec des jumelles ». Cependant, toutes ces phrases sont correctes *syntactiquement*.

Ces problèmes peuvent être également informatiques. Sur Python, la « phrase » **3 + 1.2** est correcte syntaxiquement :

```
>>> 3 + 1.2
4.2
```

mais pas sur CAML :

```
# 3 + 1.2 ;;
"Error: This expression has type float but an expression was expected of type int"
# 3. +. 1.2 ;;
- : float = 4.2
# 3 + 1 ;;
- : int = 4
```

Le dictionnaire propose les définitions suivantes :

- **syntaxe** : n. f. (bas latin syntaxis, du grec suntaxis, ordre) Partie de la grammaire qui décrit les règles par lesquelles les unités linguistiques se combinent en phrases.
- **sémantique** : n. f. (bas latin semanticus, du grec sêmantikos, qui signifie) **1.** Étude du sens des unités linguistiques et de leurs combinaisons. **2.** Étude des propositions d'une théorie déductive du point de vue de leur vérité ou de leur fausseté.

Nous allons dans cette section survoler un moyen d'analyser syntaxiquement des « phrases » qui pourront être de toute nature.

3 2 Grammaires algébriques (ou hors contexte)

3 2 1 Définitions

Définition 6 - 22

Une grammaire algébrique est définie par la donnée de :

- Σ : un ensemble fini de *symboles terminaux* ;
- V : un ensemble fini de *variables* (symboles non terminaux) ;
- $S \in V$: une variable particulière appelée axiome (pensez à « Start ») ;
- P : un ensemble fini de *règles de production* (ou de réécriture) qui sont de la forme $A \rightarrow w$ avec $A \in V$ et $w \in (\Sigma \cup V)^*$.

Dans la suite du cours, on désignera par une majuscule les variables, par une minuscule du début de l'alphabet romain les symboles terminaux, par une minuscule de la fin de l'alphabet des chaînes de $(\Sigma \cup V)^*$, par une minuscule grecque les chaînes de Σ^* .

Définition 6 - 23

Une chaîne μ *dérive* d'une chaîne ν , si, à partir de ν , on peut arriver à μ par un nombre fini d'applications des règles de G .

On note $\nu \Rightarrow \mu$ ou on peut même préciser le nombre n d'applications avec $\nu \xRightarrow{n} \mu$

Le *langage engendré* par une grammaire G est l'ensemble de toutes les chaînes terminales (appartenant donc à Σ^*) qui dérivent de l'axiome S . On le note $\mathcal{L}(G)$.

Définition 6 - 24

Une *dérivation à gauche* est une dérivation qui s'applique à la première variable rencontrée dans la chaîne (à gauche, donc...)

3 2 2 Arbre de dérivation

Pour toute règle de la forme $X \rightarrow ABCD\dots$, on construit un arbre de noeud X et de feuilles A, B, C, D, \dots

Pour toute règle du type $X \rightarrow A|B|C|\dots$, on construit un des arbres de noeud X et n'ayant qu'une seule feuille, celle-ci étant A ou B ou C ou...

On applique récursivement cette règle tant que les feuilles provisoires sont des symboles non terminaux.

Définition 6 - 25

Une grammaire est ambiguë si pour au moins une chaîne du langage, il existe au moins deux arbres de dérivation.

3 3 Grammaire régulière

Définition 6 - 26

Une grammaire algébrique est *régulière* si ses règles sont d'un des types suivant :

- $A \rightarrow aB$;
- $A \rightarrow a$;
- $A \rightarrow \mathbb{1}$, avec $A, B \in V$ et $a \in \Sigma$.

Dans chaque chaîne dérivant de l'axiome, il n'y a donc qu'une seule chaîne qui est en suffixe. Les arbres de dérivation associés sont des peignes.

On admettra le théorème suivant :

Théorème 6 - 13

Si G est une grammaire régulière, alors $\mathcal{L}(G)$ est régulier.
Tout langage régulier **peut** être engendré par une grammaire régulière.

3 4 Lemme de la pompe



Remarquons tout d'abord que si une chaîne a une longueur strictement supérieure au nombre de variables de la grammaire, alors il y a au moins une variable qui figure deux fois dans l'arbre de dérivation.

Théorème 6 - 14

Soit L un langage régulier. Il existe une constante k telle que toute chaîne μ de L dont la longueur est strictement supérieure à k peut s'écrire sous la forme $\mu = v\xi\rho$ avec ξ une chaîne non vide, la longueur de $v\xi$ étant inférieure ou égale à k et la chaîne $v\xi^n\rho$ appartient à L pour tout entier n .

3 5 Analyse syntaxique (« parsing »)

Étant donné une grammaire algébrique G , on voudrait savoir si une chaîne $\mu \in \Sigma^*$ appartient à $\mathcal{L}(G)$.

Définition 6 - 27

Le *préfixe terminal* d'une chaîne u de $(\Sigma \cup V)^*$ est la sous-chaîne des symboles terminaux qui précèdent la première variable de u .

Lors des dérivations successives, le préfixe terminal est soit inchangé, soit prolongé. Ainsi, si le préfixe terminal d'une chaîne u n'est pas identique au début de la chaîne μ à analyser, alors μ ne peut pas dériver de u .

3 5 1 Analyse syntaxique descendante en largeur

On utilise une file (premier entré, premier sortie : FIFO).

```

Procédure largeur(  $G$ : grammaire;  $\mu$ : chaîne à analyser )
Variable
  |
  | Q: liste { file }
  | c: chaîne
Début
  |
  | Q ← [S]
  | c ← S
  | TantQue  $c \neq \mu$  ou  $Q \neq \emptyset$  Faire
  |   |
  |   | Défiler Q
  |   | Appliquer successivement chacune des règles possibles à la première variable de c
  |   | Pour chaque chaîne obtenue Faire
  |   |   |
  |   |   | Si le préfixe terminal de la chaîne est un préfixe de  $\mu$  et la chaîne est non terminale Alors
  |   |   |   |
  |   |   |   | l'enfiler dans Q
  |   |   |   | FinSi
  |   |   |   | Si La chaîne est égale à  $\mu$  Alors
  |   |   |   |   |
  |   |   |   |   | C'est fini
  |   |   |   |   | FinSi
  |   |   |   |   | c ← la tête de Q
  |   |   |   | FinPour
  |   | FinTantQue
  |   | Retourner Ce n'est pas un mot du langage
  | Fin

```

3 5 2 Analyse syntaxique descendante en profondeur

On utilise cette fois une pile.

3 5 3 Analyse ascendante

On part de la chaîne et on essaie de remonter à S : on s'occupe cette fois des suffixes.

Remarque

Dans tous les cas, des grammaires mal formées peuvent rendre l'analyse syntaxique inopérante. Cependant, on peut réécrire les grammaires pour les rendre valables.

4

Correspondance automate fini / grammaire régulière

On s'arrange pour que la grammaire régulière ait des règles du type $A \rightarrow aB$ ou $A \rightarrow \mathbb{1}$.

À chaque variable de la grammaire on associe un état de l'automate.

L'axiome est l'unique état initial.

Les règles $A \rightarrow aB$ donnent les transitions $A \xrightarrow{a} B$.

Une règle $A \rightarrow \mathbb{1}$ donne un état terminal.

EXERCICES

Langages

Exercice 6 - 1

A est un alphabet :

1. Qu'est-ce qu'une chaîne sur A ?
2. Que désigne A^* ?
3. Qu'est-ce qu'un langage sur A ?
4. A est-il un langage sur A ?
5. A^* est-il un langage sur A ?
6. Toute partie de A est-elle un langage sur A ?
7. Qu'est-ce qu'un langage de A^* ?
8. \emptyset est-il un langage sur A ?
9. $\{\mathbb{1}\}$ est-il un langage sur A ?

Exercice 6 - 2

$A = \{a, b, c, d, e\}$ est un alphabet.

Expliciter les langages :

1. $L_1 = \{\mu \in A^* \mid |\mu|_c = 0\}$
2. $L_2 = \{\mu \in A^* \mid \mu = a\xi e \text{ avec } \xi \in L_1\}$
3. $L_3 = \{a, bc\} \cdot \{ad\}$
4. $L_4 = (\{a, bc\} \cdot \{ad\}) \mid (\{a, b, e\} \cdot \{cd, be\})$
5. $L_5 = L_4^0$
6. $L_6 = L_3^2$
7. $L_7 = \{a^n b^p \mid (n, p) \in \mathbb{N}^* \times \mathbb{N}\}$
8. $\{a\}^+$
9. $\{e, d\}^*$
10. $\{e \cdot d\}^* = \{ed\}^*$

Exercice 6 - 3

Soit un alphabet $A = \{a_1, a_2, \dots, a_n\}$ avec $n \in \mathbb{N}^*$.

On rappelle qu'un ensemble E est un sous-monoïde de A^* si :

- $E \subseteq A$;
- E est *stable* par concaténation, c'est-à-dire que la concaténation de deux éléments de E est encore dans E.

Ce sous-monoïde est dit *libre* s'il contient la chaîne vide $\mathbb{1}_A$.

Les ensembles suivants sont-ils des sous-monoïdes de A^* ? Si oui, sont-ils libres ? (En d'autres termes, sont-ce des langages ?...)

1. L'ensemble E_1 des chaînes de A de longueur paire.

2. L'ensemble E_2 des chaînes de A de longueur impaire.

3. $E_3 = \{(a_1 a_2)^p, p \in \mathbb{N}\}$.

4. $E_4 = \{a_1^p a_2^p, p \in \mathbb{N}\}$.

5. L'ensemble E_5 des mots où le nombre d'occurrences de a_1 est égal au nombre d'occurrences de a_2 .

Exercice 6 - 4

$L = \{a, ab, abab\}$ est un langage sur $A = \{a, b, c\}$

1. Expliciter L^2 .
2. Décrire en français le langage L^* .

Exercice 6 - 5

A est un alphabet, L est un langage sur A et μ est une chaîne de A^* . On appelle *résiduel* de L par rapport à μ le langage

$$\mu^{-1}L = \{\xi \in A^* \mid \mu \cdot \xi \in L\}$$

1. Vérifier que $\mu^{-1}L$ est bien un langage.
2. Démontrer que $\mathbb{1} \in \mu^{-1}L \Leftrightarrow \mu \in L$.
3. Déterminer $\mathbb{1}^{-1}L$.
4. Démontrer que $\nu^{-1}(\mu^{-1}L) = (\mu\nu)^{-1}L$
5. Si $L = \{ab^k \mid k \in \mathbb{N}^*\}$, déterminer $a^{-1}L$.
6. Si $L = \{b^k a \mid k \in \mathbb{N}^*\}$, déterminer $a^{-1}L$.

Exercice 6 - 6

On travaille avec l'alphabet $A = \{0, 1\}$ et L est le langage sur A formé des écritures normalisées des entiers naturels écrits en base 2, i.e. que 0 s'écrit avec 0 et si $n \neq 0$, son écriture en base 2 commence par un 1 ; on interdit les écritures du type 000001010111. Exprimer L à l'aide de langages de A^* et d'opérations rationnelles.

Exercice 6 - 7

$A = \{a, b, c, d\}$ et $L = \{a^n c b^p \mid (n, p) \in \mathbb{N}^2\}$. Exprimer L à l'aide de parties de A et d'opérations rationnelles.

Exercice 6 - 8

$A = \{a, b, c, d\}$ et L_1 est l'ensemble des chaînes qui s'écrivent uniquement avec des a ou/et des c. Exprimer L_1 à l'aide de parties de A et d'opérations rationnelles.

Exercice 6 - 9

$A = \{a, b, c, d\}$ et L_2 est l'ensemble des chaînes qui s'écrivent uniquement avec des a ou/et des c et qui contiennent une seule fois la lettre b. Exprimer L_2 à l'aide de parties de A et d'opérations rationnelles.

Exercice 6 - 10

Soient a et b deux symboles d'un alphabet A et $\mu \in A^*$. Montrer par récurrence (sur?) que si $\mu a = b\mu$ alors $a = b$ et $\mu \in \{a\}^*$.

Langages et expressions rationnels

Exercice 6 - 11

Expliciter les langages associés aux expressions rationnelles suivantes :

1. $a(a|b)^* a$
2. $(aa|bb)^* (ab|ba) (aa|bb)^*$
3. $a^* ba^* ba^* ba^*$
4. $(abc)^*$
5. $(abc^*)^*$

Exercice 6 - 12 Vrai ou faux ?

1. $(a^*)^* = a^*$
2. $aa^* = a^* a$
3. $aa^* | \mathbb{1} = a^*$
4. $a(b|c) = ab|ac$
5. $(a|b)^* = (a^*|b^*)^*$
6. $a(ba)^* = (ab)^* a$
7. $(a|b)^* = (a^* b^*)^*$
8. $(a|b)^* = a^* (ba^*)^*$
9. $((a|b)^* | c)^* = (a|b|c)^*$

Exercice 6 - 13

Déterminez des expressions régulières correspondant aux définitions suivantes :

1. les chaînes formées de caractères de l'alphabet non accentués, où les caractères sont dans l'ordre alphabétique croissant et inférieurs à f ;
2. les chaînes de bits sans « 0 » ;
3. les chaînes de bits sans « 01 » ;
4. les chaînes de bits sans « 010 » ;
5. les commentaires dans un langage de programmation formés d'une chaîne encadrée par « [# » et « #] », sans « #] » à l'intérieur si ce n'est à l'intérieur d'une chaîne en étant encadré par « " » et « " » ;
6. les chaînes de bits constituées d'un nombre pair de 1.

Exercice 6 - 14 Expressions rationnelles et grep

Nous allons étudier quelques aspects de l'utilitaire **grep** des systèmes UNIX en rapport avec la notion d'expression rationnelle vue en cours.

Nous utiliserons les opérateurs « * » et « \ | » qui correspondent à l'étoile de KLEENE et à la réunion des chaînes.

Il existe également l'opérateur « \+ » qui correspond à la concaténation de la chaîne avec son étoile (**a\+** correspond à $a \cdot a^* = a^+$).

Les parenthèses seront précédées d'une contre-oblique : « \ (» et « \) ».

Nous travaillerons sur un fichier **.txt** contenant une chaîne par ligne. Pour signifier à **grep** que nous recherchons des chaînes correspondant en totalité à l'expression rationnelle donnée, nous l'encadrerons entre les symboles « ^ » et « \$ » qui marquent le début et la fin d'une chaîne.

Par exemple, considérons le fichier expressions.txt suivant :

```
a
b
aab
aaabbbb
abbbbbba
aaaaabaaaababababbbaaaaaaaaaabb
ababababababbabaaaaaaaa
bbbbbbbbbbba
abbbbbbbbbbbba
aaaaaaaaabaaaaaaaaaaaaaaaa
```

Considérons la première expression rationnelle de l'exercice **Exercice 6 - 11**, à savoir $a(a|b)^* a$. Sous **grep** cela donne :

```
$ egrep ^a\(a|b\) *a$ ./expression.txt
```

On obtient :

```
abbbbbba
ababababababbabaaaaaaaa
abbbbbbbbbbbba
aaaaaaaaabaaaaaaaaaaaaaaaa
```

Donnez un exemple de commande permettant de n'obtenir que la dernière chaîne, que la cinquième, que la première, que la troisième.

Que renverra $^a * b * a * \$? ^ a * b * a \ + \$ + ?$

Que fait le script Bash suivant :

```
#!/bin/sh
# -- coding: utf-8 --

read -p "Entrez une expression rationnelle " rat
```

```

read -p "Entrez une chaîne " chaîne
> ./chaîne_test.txt
echo $chaîne >> chaîne_test.txt
resultat=$(egrep '^$rat'$' ./chaîne_test.txt)
if [ ${#resultat} -gt 0 ]; then
    echo Reconnu ;
else
    echo Non_reconnu ;
fi

```

Généralités sur les automates

Exercice 6 - 15 Diagramme d'un automate

Considérons l'automate suivant :

- l'alphabet est $A = \{0, 1\}$;
- $Q = \{q_0, q_1, q_2, q_3, q_4\}$;
- $I = \{q_0\}$;
- $T = \{q_3\}$;
- la fonction de transition est définie par :

| τ | 0 | 1 |
|--------|-------|-------|
| q_0 | q_1 | q_3 |
| q_1 | q_4 | q_2 |
| q_2 | q_0 | q_2 |
| q_3 | q_2 | q_4 |
| q_4 | q_4 | q_4 |

Dessinez le diagramme correspondant.
 Donnez des chaînes reconnues par l'automate.
 Donnez des chaînes non reconnues par l'automate.
 Que peut-on dire à propos de l'état q_4 ?

Exercice 6 - 16

Pour chacun des automates suivants I (ou T) indique que l'état est initial (ou terminal), l'alphabet est $A = \{a, b\}$ et les états sont nommés à l'aide d'entiers. Donnez une représentation graphique et des chaînes qui appartiennent au langage engendré ou reconnu

par chacun des automates suivants :

1.

| τ | a | b |
|--------|-----|-----|
| 1(I) | 1,2 | 1 |
| 2 | 3 | 3 |
| 3 | 4 | 4 |
| 4(T) | | |

2.

| τ | a | b |
|--------|-----|-----|
| 1(I) | 2,3 | 2 |
| 2(T) | 3 | |
| 3(I,T) | | 1,2 |

3.

| τ | a | b |
|--------|-----|-----|
| 1(I) | 2,3 | 1 |
| 2(T) | 1,2 | |
| 3(I) | 1,3 | 2 |

Expression rationnelle associée à un automate

Exercice 6 - 17

On considère l'automate A sur l'alphabet $A = \{a, b\}$ qui a pour ensemble d'états $Q = \{e_1, e_2\}$, e_1 étant simultanément état initial et état final. Les transitions de A sont données par le tableau :

| τ | a | b |
|--------|-------|-------|
| e_1 | e_1 | e_2 |
| e_2 | e_2 | e_1 |

ce tableau traduit, par exemple, que si l'automate est dans l'état e_2 et qu'il lit b , alors il passe à l'état e_1 .

1. Cet automate est-il déterministe? Justifier votre réponse.
2. Cet automate est-il complet? Justifier votre réponse.
3. Donner une représentation graphique de cet automate.
4. Déterminer une expression rationnelle R qui décrit L.

- 5. On note L le langage engendré ou reconnu par cet automate, $L = \mathcal{L}(A)$. Ce langage est-il rationnel ?
- 6. ϵ désigne la chaîne vide. Expliquez pourquoi ϵ appartient à L.

Automates déterministes

Exercice 6 - 22

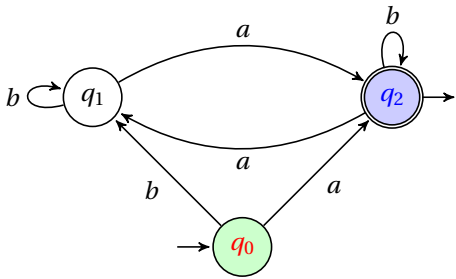
On considère l'automate $A = (A, E, I, T, U)$ avec

1.
 - $A = \{a, b\}$, l'alphabet de l'automate A.
 - $Q = \{1, 2, 3\}$ l'ensemble des états de A.
 - $I = \{1, 3\}$ l'ensemble des états initiaux de A.
 - $T = \{2, 3\}$ l'ensemble des états terminaux de A.
 - $U = \{(1, a, 2), (1, a, 3), (1, b, 2), (2, a, 3), (3, b, 1), (3, b, 2)\}$ l'ensemble des transitions de A, U pouvant être représenté par le tableau

| τ | a | b |
|---------|-----|-----|
| 1(I) | 2,3 | 2 |
| 2(T) | 3 | |
| 3(I, T) | | 1,2 |

Exercice 6 - 18

On note L le langage reconnu par cet automate :



Déterminez une expression régulière R qui décrit L.

Automates standards et émondés

Exercice 6 - 19

Standardisez et émondez les automates de l'Exercice 6 - 15 et de l'Exercice 6 - 16.

Exercice 6 - 20

Standardisez l'automate défini sur $A = \{a, b\}$ par :

| τ | a | b |
|--------|-------|-------|
| q_0 | q_1 | q_0 |
| q_1 | q_0 | q_1 |
| q_2 | q_2 | q_3 |
| q_3 | q_3 | q_2 |

sachant que $I = \{q_0, q_2\}$ et $T = \{q_1, q_2\}$.

Exercice 6 - 21

Standardisez et émondez l'automate défini sur $A = \{a, b\}$ par :

| τ | a | b |
|--------|------------|-------|
| q_0 | q_0, q_1 | |
| q_1 | | q_2 |
| q_2 | q_2 | q_2 |
| q_3 | q_2 | q_1 |

sachant que $I = \{q_0, q_2\}$ et $T = \{q_0, q_1\}$.

On précise dans ce tableau si un état est initial ou terminal.

2. Donnez une représentation graphique de A.
3. Déterminez un automate déterministe $A' = (A, E', I', T', U')$ équivalent à A, on présentera U' sous forme d'un tableau.

Exercice 6 - 23

« Déterminez » les automates suivants : I indique que l'état est initial et T que l'état est final

1.

| τ | a | b |
|---------|-----|-----|
| 1(I) | 2,3 | 2 |
| 2(T) | 3 | |
| 3(I, T) | | 1,2 |

2.

| τ | a | b |
|---------|-----|-----|
| 1(I) | 2 | 2,3 |
| 2(T) | | 1 |
| 3(I, T) | 1,2 | |

3.

| τ | a | b |
|--------|-----|-----|
| 1(I) | 2 | 1,2 |
| 2(I) | 3 | 3 |
| 3(T) | 4 | 4,1 |
| 4(T) | | |

Construction d'automates

Exercice 6 - 24

Construire des automates correspondant aux expressions rationnelles suivantes :

- $E = aab^*ab$
- $E = a^*b^*a^*$
- $E = a^*bA^+$
- $E = A^*abcA^*$
- $E = abA^*(A^4)^*$
- $E = (a|aba)^*b(a|b)^*aba(a|b)^*$

Exercice 6 - 25 Identités rationnelles

Tout comme pour les automates, on peut définir une relation d'équivalence sur l'ensemble des expressions rationnelles : deux expressions sont équivalentes si, et seulement si, les langages qui leurs sont associés sont égaux.

L'identité suivante est classique :

$$(a|b)^* \equiv (a^*b)^*a^*$$

Prouvez-la en passant aux automates associés.

Automates séquentiels

Exercice 6 - 26 avertissement d'erreur de transmission

Un protocole de codage de message prévoit d'indiquer une erreur de transmission si trois 1 apparaissent consécutivement dans le message. Construire un automate à états finis qui renvoie 1 si, et seulement si, la chaîne transmise contient trois 1 consécutifs.

Exercice 6 - 27 automate de rendu de monnaie

Un distributeur de boisson accepte les pièces de 5, 10 et 20 centimes.

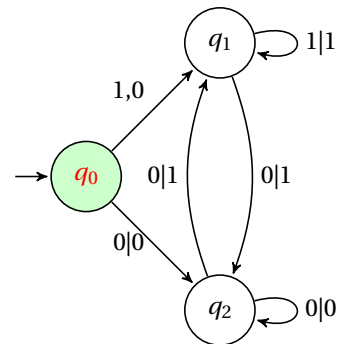
Une fois que l'utilisateur a introduit au minimum 30 centimes, le distributeur lui rend la somme dépassant

30 centimes et il peut choisir entre un café en appuyant sur la touche K ou un lait fraise en appuyant sur la touche L.

Construire un automate à états finis avec sortie qui modélise ce distributeur.

Exercice 6 - 28 Retard

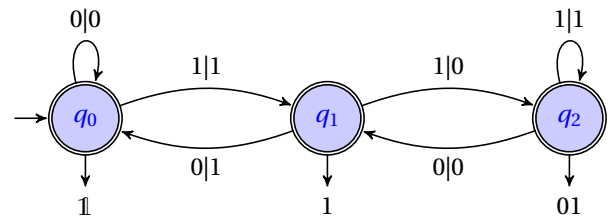
Décrivez le plus précisément possible l'action de cet automate :



Exercice 6 - 29 Table de 3

Écrivez la table de trois en binaire inversé.

Faites opérer l'automate suivant sur les nombres de un à dix représentés sous forme binaire inversée :



Automates à pile

Exercice 6 - 30

Soit l'automate à pile sur l'alphabet $A = \{a, b, \dots\}$ avec :

- $E = \{s, q\}$, s est l'état initial
- $T = \{t\}$
- $P = \{A, B\}$
- les transitions sont :
 - $T_1 = ((s, \mathbb{1}), a, (s, A))$
 - $T_2 = ((s, \mathbb{1}), b, (s, B))$
 - $T_3 = ((s, A), a, (s, AA))$
 - $T_4 = ((s, B), a, (s, AB))$
 - $T_5 = ((s, A), b, (s, BA))$
 - $T_6 = ((s, B), b, (s, BB))$
 - $T_7 = ((s, A), a, (t, \mathbb{1}))$
 - $T_8 = ((s, B), b, (t, \mathbb{1}))$
 - $T_9 = ((t, A), a, (t, \mathbb{1}))$
 - $T_{10} = ((t, B), b, (t, \mathbb{1}))$

1. Donner une représentation graphique de cet automate. On remarquera que cet automate n'est pas déterministe.
2. On note L le langage engendré par cet automate, démontrer que
 - a. $aaaa \in L$
 - b. $bbbbbb \in L$
 - c. $aaabaabbaabaaa \in L$, observer l'état de la pile au fur et à mesure de la reconnaissance du mot.
 - d. L est un langage particulier sur A en ce sens que tous les mots de L ont une particularité, deviner cette particularité.

Exercice 6 - 31

Déterminer un automate à pile qui reconnaît les langages :

1. $L_1 = \{u \in \{a, b\}^* \mid |u|_a = 2|u|_b\}$;
2. $L_2 = \{u \in \{a, b\}^* \mid \bar{u} = u\}$ avec \bar{u} le miroir de u .

Problèmes divers

Exercice 6 - 32 Minimalisation

Appliquez l'algorithme de MOORE pour minimaliser les automates étudiés dans les exercices précédents.

Exercice 6 - 33 Une conséquence du théorème de Kleene

Soit A un alphabet et $L \in A^*$ un langage reconnaissable de A^* donc il est rationnel. Le langage $L^c = \complement_E A^* = \{u \in A^* \mid u \notin L\}$ est aussi un langage rationnel donc il est reconnaissable.

Soit maintenant $\mathcal{A} = (A, Q, \{d\}, T, \tau)$ un automate fini déterministe et complet qui reconnaît un langage L . L'automate $\mathcal{A}' = (A, Q, \{d\}, Q - T, \tau)$ reconnaît le langage L^c . En effet, puisque \mathcal{A} et \mathcal{A}' sont tous deux complets et déterministes, chaque chaîne u de A^* est l'étiquette d'exactly un calcul commençant en d . Soit q_u l'extrémité terminale de ce calcul. Alors u appartient à L si, et seulement si, q appartient à T et u appartient à L^c si, et seulement si, q appartient à $\complement_{QF} = Q - F$.

On travaille sur l'alphabet $\{0, 1\}$. Déterminer alors une expression rationnelle représentant le langage L des chaînes NE contenant PAS la chaîne 001. Il sera utile de passer par la construction d'automates. La résolution de l' Exercice 6 - 13 page 47 se trouve ainsi facilitée...

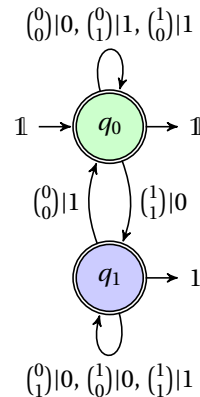
Exercice 6 - 34 Extrait de l'épreuve Mines-Ponts 2011

Pour tout entier n non nul on définit le langage L_n sur l'alphabet $A = \{a, b\}$ de la manière suivante : L_n est l'ensemble des mots de longueur supérieure ou égale à $2n$ dont le suffixe de longueur n est le miroir du préfixe de longueur n . Ainsi $abbbbaba$ appartient à L_1 et à L_2 mais pas à L_3 .

1. Expliquer pourquoi $abbbbaba$ appartient à L_1 et à L_2 mais pas à L_3 .
2. Donner une expression rationnelle décrivant L_1 .
3. Construire un automate \mathcal{A} non déterministe reconnaissant le langage L_2 . On impose que \mathcal{A} ait un seul état initial et un seul état final; par ailleurs, les transitions de \mathcal{A} seront étiquetées par les éléments de A .
4. Déterminer l'automate obtenu à la question précédente.
5. En s'inspirant des questions précédentes, montrer que L_n est un langage rationnel pour tout $n \geq 1$.

Exercice 6 - 35 Addition binaire

Le nombre 27 s'écrit 110011 en écriture binaire inversée. Le nombre 14 s'écrit 0111. On considère le couple (110011, 0111) que l'on écrit $\begin{pmatrix} 110011 \\ 011100 \end{pmatrix}$ soit encore $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. On peut donc le considérer comme une chaîne sur l'alphabet $A = \{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}\}$. Voici un automate séquentiel défini sur cet alphabet :



Faites lire $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ par l'automate. Que fait cet automate ?

Exercice 6 - 36 Parenthèses

Une chaîne bien parenthésée est une chaîne sur l'alphabet $\{(,)\}$ comportant autant de parenthèses ouvrantes que de parenthèses fermantes et telle que chacun de ses préfixes contient au moins autant de parenthèses ouvrantes que de parenthèses fermantes.

Déterminer un automate qui reconnaît les chaînes bien parenthésées.

Grammaires

Exercice 6 - 37

Essayer de caractériser le langage engendré par la grammaire suivante :

$$S \rightarrow ab|aS|Sb$$

Est-elle régulière ?

Proposer une grammaire régulière équivalente, puis un automate qui reconnaît le même langage puis une expression rationnelle associée.

Que dire de la grammaire :

$$S \rightarrow AB$$

$$A \rightarrow a|aA$$

$$B \rightarrow b|bB$$

Pour chacune des trois grammaires, proposer des dérivations qui donnent $aabbb$ puis les arbres de dérivation qui correspondent. Les grammaires sont-elles ambiguës ?

Donner maintenant une grammaire qui décrit le langage sur $\{a, b\}$ qui ne contient que des chaînes avec autant de a que de b avec tous les a au début et au moins un a .

Donner une grammaire qui décrit les chaînes formées de a et de b contenant au moins un b et deux arbres de dérivation associés à la chaîne abb .

Exercice 6 - 38 Palindromes

Donner une grammaire qui engendre les palindromes sur l'alphabet $\{a, b\}$.

Exercice 6 - 39 Chaînes bien parenthésées

Donner une grammaire qui engendre l'ensemble des chaînes bien parenthésées (voir Exercice 6 - 36 page précédente) et donner une dérivation gauche de $(())()$. Vérifier que $()(())$ est bien parenthésée par analyse syntaxique descendante en largeur.

Exercice 6 - 40 Priorité et ambiguïté

On étudie une grammaire décrivant des expressions arithmétiques sur des nombres réduits à un chiffre en

base 10 et les opérateurs binaires d'addition, de soustraction et de multiplication.

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \times\}$$

Les opérations sont notées sous forme *infixée*, c'est-à-dire que si E_1 et E_2 sont des expressions arithmétiques, alors $E_1 + E_2$, $E_1 - E_2$ et $E_1 \times E_2$ aussi.

Voici une grammaire possible :

$$S \rightarrow C|S + S|S - S|S \times S$$

$$C \rightarrow 0|1|2|3|4|5|6|7|8|9$$

Donner deux arbres syntaxiques correspondant à $1 - 2 - 3$ puis deux autres correspondant à $1 + 2 \times 3$.

Que dire alors de la grammaire ?

Pour contrer ce désagrément (pourquoi est-ce un désagrément en pratique ?), on peut envisager plusieurs stratégies.

Par exemple, on peut décider que soustraction et addition sont associatives à gauche. On remplace donc l'axiome par :

$$S \rightarrow C|S + C|S - C$$

Que se passe-t-il maintenant pour $1 - 2 - 3$? $1 - 2 + 3$?

Comment changer la grammaire pour avoir cette fois l'associativité à droite ?

On choisira par la suite la priorité à gauche.

On s'occupe maintenant de la priorité : \times a priorité sur $+$ et $-$.

Ainsi, une expression est d'abord une somme ou une différence puis, à un second niveau d'abstraction, chaque terme est un produit de facteurs.

Donner une grammaire traduisant cette construction puis l'arbre de dérivation de $1 + 2 \times 3$.

Une possibilité pour « contrer » la règle de priorité est d'utiliser des parenthèses.

Comment réorganiser la grammaire avec la règle :

$$F \rightarrow C|(E)$$

Donner alors les arbres de dérivation de $1 \times 2 + 3 \times 4$ et $1 \times (2 + 3) \times 4$.

Exercice 6 - 41 Notation polonaise inverse

La notation préfixée a été introduite en 1920 par le polonais Jan ŁUKASIEWICZ puis « inversée » par l'australien Charles HAMBLIN dans les années 1950.

Elle est utilisée en informatique car elle permet de se passer de parenthèses (machines HP, PostScript, processeurs).

$1 + 2$ s'écrit $12+$ et 1×2 s'écrit $12\times$. Ensuite, on écrit les triplets (nb,op,nb) dans le bon ordre de calcul.

Par exemple, $(1 + 2) \times (3 + 4)$ s'écrit $12 + 34 \times$.

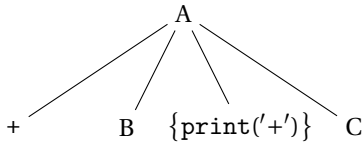
$((1 + 2) \times 3) + 4$ s'écrit $12 + 3 \times 4+$ mais aussi $4312 + \times +$ ce qui est moins évident peut-être.

Comment décrire ce mécanisme à l'aide d'une pile ?

Comment le décrire à l'aide d'une grammaire ? Est-elle ambiguë ?

Exercice 6 - 42 **Système de traduction**

Un système de traduction est une grammaire algébrique à laquelle on ajoute des *actions sémantiques*. Par exemple, la règle $A \rightarrow +B\{\text{print}('+\')\}C$ se traduira par l'arbre de dérivation :



Donner un système de traduction effectuant la conversion notation infix avec associativité à gauche vers NPI. On se contentera de traiter le cas d'additions et de soustractions de nombres d'un chiffre. Donner l'arbre de dérivation correspondant à 9-8+1.

Exercice 6 - 43 **Lemme de pompage**

Le lemme de pompage (ou « de gonflement » ou « de l'étoile » a été vu en cours et donne une condition nécessaire pour vérifier si un langage défini par une grammaire est régulier et dit en gros que si un arbre de dérivation est plus « haut » que le nombre de variables, il y a au moins une variable répétée sur un chemin issu de S et la portion de branche située entre ces deux occurrences peut être répétée autant de fois qu'on veut sans « quitter » le langage.

Voyons une application classique qui va nous permettre de montrer que le sempiternel langage $L = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas régulier. On va raisonner par l'absurde. Quelque soit l'entier k , on considère la chaîne $\mu = a^k b^k$ qui est de longueur supérieure à k . Si le langage est régulier, alors μ contient un préfixe $v\xi$ de longueur inférieur à k tel que $\mu = v\xi\rho$ et $nu\xi\xi\rho$ appartient encore à L. En quoi cette condition aboutit à une contradiction ?

Appliquer cette méthode pour montrer que le langage des palindromes sur $\{a, b\}$ n'est pas régulier. On pourra distinguer les palindromes de longueur paire et ceux de longueur impaire.

Exercice 6 - 44

Déterminer un automate qui reconnaît le langage $\{a^{2n} \mid n \geq 0\}$. Donner une grammaire régulière qui reconnaît ce langage.

Exercice 6 - 45 **Ambiguïté**

Soit $\Sigma = \{\text{if, then, else, } a, b\}$ et la grammaire :
 $S \rightarrow \text{if } a \text{ then } S \text{ else } S \mid \text{if } a \text{ then } S \mid b$
 Cette grammaire est-elle ambiguë ?

Exercice 6 - 46

Soit G et G' les deux grammaires suivantes :
 $G : S \rightarrow aSb \mid ab$
 $G' : S \rightarrow cSd \mid cd$
 Comparer $\mathcal{L}(G) \cup \mathcal{L}(G')$ et $\mathcal{L}(G \cup G')$.

Exercice 6 - 47 **Automate → grammaire**

À chaque transition, on associe une règle de grammaire. Un état terminal renvoie à un symbole terminal en doublant éventuellement la règle. Donner une grammaire associée à l'automate suivant :

| | | |
|------|-----|-----|
| | a | b |
| 1(I) | 1,2 | 1,3 |
| 2 | 4 | |
| 3 | | 5 |
| 4(T) | 4 | 4 |
| 5(T) | 5 | 5 |

En déduire une simplification de l'automate puis le rendre déterministe.
 En déduire une simplification de l'automate puis le rendre déterministe.

Révisions

Exercice 6 - 48

Donner un automate qui reconnaît le langage de $\{a, b\}^*$ dont les chaînes n'ont pas trois a consécutifs.

Exercice 6 - 49 **Glushkov et minimalisation**

Soit $E = (a|b)^* abb$
 Écrire un automate qui reconnaît le langage associé à l'expression rationnelle E. Est-il minimal ?

Exercice 6 - 50

Donner une expression rationnelle pour le langage reconnu par cet automate :

| | | |
|------|-------|---|
| | a | b |
| 0(I) | 1,2 | 0 |
| 1(T) | 0,1,2 | 1 |
| 2 | | 2 |

Déterminer cet automate et trouver une autre expression rationnelle associée.

Minimiser ensuite l'automate et trouver une autre expression rationnelle associée.

Exercice 6 - 51

Minimaliser l'automate suivant :

| | a | b |
|------|---|---|
| 0(I) | 1 | 5 |
| 1 | 2 | 3 |
| 2(T) | 2 | 4 |
| 3 | 1 | 6 |
| 4(T) | 1 | 4 |
| 5 | 5 | 5 |
| 6 | 1 | 6 |

Exercice 6 - 52 Machine de Turing plus forte qu'un automate à pile

Un automate à pile ne peut reconnaître le langage $\{a^n b^n c^n \mid n \in \mathbb{N}\}$: un lemme de pompage généralisé permet de le montrer.

Déterminer une machine de TURING qui va reconnaître ce langage à l'aide de 5 états plus deux états (accepté, refusé).

Le principe est de barrer le premier a , aller au bout du mot et barrer le dernier c puis revenir en arrière et barrer le dernier b et recommencer en allant vers le rejet dès que l'occasion s'en présente et sachant que la chaîne vide est accepté.

Exercice 6 - 53 Machine de Turing

Construisez les machines de Turing suivantes :

1. M qui écrit 0 1 0 1 0 1 0 ... sur un ruban blanc. *Pour ceux qui douteraient de l'intérêt de cette question, s'adresser à A. Turing.*
2. M à un ruban sur l'alphabet $\{0, 1, \varepsilon\}$ qui multiplie par 2 son entrée binaire.
3. M à un ruban sur l'alphabet $\{0, 1, \varepsilon\}$ qui multiplie par 2 et ajoute 1 à son entrée binaire.
4. M à un ruban sur l'alphabet $\{0, 1, \varepsilon\}$ qui ajoute 1 à son entrée binaire.

6

Automates et Caml

6 1 Modélisation

La modélisation adoptée suit la définition du cours, mais nous remplacerons la relation de transition par son graphe.

Ainsi, un automate \mathcal{A} est caractérisé par un quintuplet $\langle \Sigma, Q, D, T, \tau \rangle$:

- un alphabet Σ appelé l'*alphabet d'entrée*;
- un alphabet Q appelé l'*alphabet d'état* (Q est donc **fini**);
- une partie D non vide de Q appelée *ensemble des états initiaux*;
- une partie T non vide de Q appelée *ensemble des états terminaux*;
- une relation τ de $Q \times \Sigma$ vers Q ou de $Q \times (\Sigma \cup \{\emptyset\})$ vers Q , appelée *relation de transition* qui est elle-même caractérisée par son graphe qui est une partie de $((Q \times \Sigma) \times Q)$.

Cela donne :

```
(* un état construit à partir d'un entier *)
type etat = int;;

(* un symbole construit à partir d'un caractère *)
type symbole = char;;

(* automate sous forme d'enregistrement (record) *)
type automate =
{
  q      : etat list;
  d      : etat list;
  t      : etat list ;
  graphe : ((etat * symbole) * etat) list;
  s      : symbole list;
};;
```

Voici un exemple d'automate : dessinez-le :

```
let graphe1 =
  [((1, 'a'), 3)      ;((1, 'b'), 2)
   ;((2, 'a'), 3)      ;((2, 'b'), 2)
   ;((3, 'a'), 4)      ;((3, 'b'), 3)
   ;((4, 'a'), 3)      ;((4, 'b'), 4)]
;;
let otto1 =
{
  q      = [1; 2; 3; 4];
  d      = [1];
  t      = [4];
  graphe = graphe1;
  s      = ['a'; 'b'];
};;
```

Voici un extrait de la documentation Ocaml sur le module **List** :

```
val assoc : 'a -> ('a * 'b) list -> 'b
"assoc a l returns the value associated with key a in the list of pairs l. That is, assoc
 a [ ...; (a,b); ...] = b if (a,b) is the leftmost binding of a in list l. Raise
 Not_found if there is no value associated with a in the list l."
```

Déduisez-en une fonction `tau` de signature :

```
val tau : automate -> etat * symbole -> etat = <fun>
```

N'oubliez pas que la curryfication est associative à droite : cette signature correspond donc à `val tau : automate -> ((etat*symbole) -> etat) = <fun>`

Par exemple, vous devriez obtenir :

```
# tau otto1 (1, 'a');;
- : etat = 3
```

Il s'agit maintenant de construire le cheminement d'une liste d'étiquettes dans l'automate. On veut donc construire une fonction `parcours` de signature :

```
val parcours : automate -> symbole list -> etat list = <fun>
```

On se remémorera le filtrage par motif vu il y a quelques semaines (« *pattern matching* »). Par exemple, on obtiendra :

```
# parcours otto1 ['a'; 'a'; 'a'; 'b'; 'a'];;
- : etat list = [1; 3; 4; 3; 3; 4]
```

Pour cela, il faudra par exemple utiliser une fonction *locale*. On aura donc une structure du type :

```
let parcours auto chaine =
  let rec parcours_local t e chaine =
    match chaine with
    | ... -> ...
    | ... -> ...
  in
  parcours_local ... .. .
;;
```

On pourra en déduire une fonction `lit` de signature :

```
val lit : automate -> symbole list -> bool = <fun>
```

qui teste si une liste de symboles correspond à un parcours vers un état final :

```
# lit otto1 ['a'; 'a'; 'a'; 'b'; 'a'];;
- : bool = true
# lit otto1 ['a'; 'a'; 'a'; 'b'];;
- : bool = false
```

6 2 Visualisation

Nous utiliserons le logiciel GraphViz qui permet de visualiser un graphe à partir d'un fichier d'extension `dot`. Nous vous invitons à parcourir sa documentation.

Voici des fonctions qui font le boulot :

```
let trans_to_string = fun ((e1, s), e2) ->
  String.concat ""
  (string_of_int(e1)::"->"::string_of_int(e2)::"[label=\\""::(String.make 1 s)::"\"];\\n"
   ::[]);;

let ini_to_string = fun auto ->
  let rec parcours = fun liste ->
    match liste with
    | [] -> ""
```



```

|t::q -> (string_of_int(t)^" [color = yellow,style=filled];\n ")^(parcours q)
in parcours auto.d;;

let fin_to_string = fun auto ->
  let rec parcours = fun liste ->
    match liste with
    |[] -> ""
    |t::q -> (string_of_int(t)^" [color = blue,style=filled,fontcolor=white];\n ")^(
      parcours q)
  in parcours auto.t;;

let fini_to_string = fun auto ->
  let rec parcours = fun liste ->
    match liste with
    |[] -> ""
    |t::q -> (string_of_int(t)^" [color = red,style=filled,fontcolor=white];\n ")^(
      parcours q)
  in parcours (List.filter (fun x-> List.mem x auto.d) auto.t);;

let rec auto_to_string = fun graphe ->
  match graphe with
  |[] -> ""
  |t::q -> (trans_to_string t) ^ (auto_to_string q);;

let auto2dot nomFich auto =
  let chan = open_out nomFich in
  let print_str = output_string chan in
  print_str "digraph G {\n";
  print_str (auto_to_string (auto.graphe));
  print_str (ini_to_string auto);
  print_str (fin_to_string auto);
  print_str (fini_to_string auto);
  print_str "}\n";
  close_out chan
;;

```

Commentez-les!

Ainsi, la commande `auto2dot "otto1.dot" otto1;;` produit un fichier «point dot» que l'on peut ouvrir. Emacs se met alors en mode GraphViz. On peut compiler le fichier avec C-c c et visualiser l'automate avec C-c p.

Visualisez `otto1`.

6.3 Standardisation

Nous allons dans un premier temps nous occuper d'une semi-standardisation : un automate fini sera dit semi-standard s'il admet un unique état initial.

Explorons à nouveau la documentation du module `List` :

```

val length : 'a list -> int
"Return the length (number of elements) of the given list."

val mem : 'a -> 'a list -> bool
"mem a l is true if and only if a is equal to an element of l."

val exists : ('a -> bool) -> 'a list -> bool
"exists p [a1; ...; an] checks if at least one element of the list satisfies the
predicate p. That is, it returns (p a1) || (p a2) || ... || (p an)."
```

```

val filter : ('a -> bool) -> 'a list -> 'a list
"filter p l returns all the elements of the list l that satisfy the predicate p.
  The order of the elements in the input list is preserved."

val map : ('a -> 'b) -> 'a list -> 'b list
"List.map f [a1; ...; an] applies function f to a1, ..., an, and builds the list [f
  a1; ...; f an] with the results returned by f. Not tail-recursive."

val fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
"List.fold_left f a [b1; ...; bn] is f (... (f (f a b1) b2) ...) bn."

val append : 'a list -> 'a list -> 'a list
"Catenate two lists. Same function as the infix operator @. Not tail-recursive (length of
  the first argument). The @ operator is not tail-recursive either."

```

Comment traduiriez-vous à l'aide de conditionnelles et de boucles les fonctions **exists**, **filter**, **map** et **fold_left** ?

Comment les traduiriez-vous à l'aide d'un filtrage par motif et d'une fonction récursive ?
 Créez un test **est_semi_standard** de signature :

```
val est_semi_standard : automate -> bool = <fun>.
```

Encore un extrait de la documentation :

```

val fst : 'a * 'b -> 'a
"Return the first component of a pair."

val snd : 'a * 'b -> 'b
"Return the second component of a pair."

```

Pour simplifier notre travail et clarifier nos fonctions, nous allons utiliser les variables suivantes :

```

let src      t = fst(fst t);;
let label    t = snd(fst t);;
let tgt      t = snd t;;

```

Que représentent-elles ?

Créez alors un test **est_standard** en utilisant la définition complète du cours.

Créez ensuite une fonction **sous_graphe** qui a pour arguments un automate et un sous-ensemble de ses états et qui renvoie la partie du graphe contenant les transitions issues des états du sous-ensemble :

```
automate -> etat list -> ((etat*symbole)*etat) list = <fun>
```

Voici un nouvel extrait de la documentation au sujet d'une fonction au nom explicite :

```

val max : 'a -> 'a -> 'a
"Return the greater of the two arguments. The result is unspecified if one of the
  arguments contains the float value nan."

```

Que représente alors **id** dans le code suivant :

```
let id = (List.fold_left (fun m e -> (max m e)) 0 auto.q) + 1
```

Vous êtes maintenant armé(e) pour créer une fonction **standard** qui va standardiser un automate donné en argument et dont la signature est :

```
- : automate -> automate = <fun>.
```

6.4 Opérations rationnelles

Un nouvel extrait de la doc :

```
val combine : 'a list -> 'b list -> ('a * 'b) list
"Transform a pair of lists into a list of pairs: combine [a1; ...; an] [b1; ...; bn] is
[(a1,b1); ...; (an,bn)]. Raise Invalid_argument if the two lists have different
lengths. Not tail-recursive."
```

Commentez alors les fonctions suivantes :

```
let l_reorder_from = fun liste dep ->
  let rec aux = fun liste i acc ->
    match liste with
    | [] -> acc
    | t::q -> aux q (i+1) (i::acc)
  in aux liste dep []
;;

let g_reorder_from = fun graphe l_qa dep ->
  let l_combi = List.combine l_qa (l_reorder_from l_qa dep) in
  List.map (fun ((q1,a),q2) -> ((List.assoc q1 l_combi,a),(List.assoc q2 l_combi)) )
    graphe;;

let q_reorder_from = fun q_liste l_qa dep->
  let l_combi = List.combine l_qa (l_reorder_from l_qa dep) in
  List.map (fun q1 -> List.assoc q1 l_combi) q_liste;;

let a_reorder = fun auto m ->
{
  s = auto.l.s;
  q = q_reorder_from auto.q auto.q m;
  d = q_reorder_from auto.d auto.q m;
  t = q_reorder_from auto.t auto.q m;
  graphe = g_reorder_from auto.graphe auto.q m;
};;
```

Vous êtes maintenant prêt(e) pour l'union, la concaténation et l'étoile :

```
val auto_union : automate -> automate -> automate = <fun>
val auto_concat : automate -> automate -> automate = <fun>
val auto_star : automate -> automate = <fun>
```