2022–2023

Informatique

MPI*

Hugo Salou

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

LISTE DES CODES

Partie I

Cours

CHAPITRE

–1 ORDRES ET INDUCTION

Sommaire

–1.1 Motivation

```
let rec fact n =
  if n = 0 then 1
  else n * (fact (n-1));;
```

CODE 1.1 – Calcul de factorielle

Ce programme calcule la factorielle d'un nombre ? En développant, l'expression de $3!$, on a

$$\begin{aligned}(\texttt{fact 3}) &= 3 \times (\texttt{fact 2}) \\ &= 3 \times (2 \times (1 \times 1))\end{aligned}$$

On en déduit que ce programme calcule la factorielle car ce développement s'arrête à un certain point. Comment en être sûr ? En effet, avec $\texttt{n} = -1$, on obtient une `Stack Overflow Error` ; on n'a plus de mémoire. Pour en être sûr, il faut définir un *invariant*.

À faire : Ajouter 2ème exemple

Un autre exemple :

```
let mystere2 n m =
  let rec aux c b =
    if c = 0 and b = m then 0
    else if c = 0 then aux (b * m) (b + 1)
    else 1 + aux (c - 1) b
  in aux n 0;;
```

CODE 1.2 – Un programme mystère (2)

Ce programme a beaucoup plus de variables : les variables augmentent dans certains cas, puis diminuent… On peut représenter l'état des variables `b` et `c` dans une figure :

À faire : Figure à faire

Figure 1.1 – État des variables `b` et `c`

On en conclut que ce programme calcule la valeur de

$$\mathtt{n} + \mathtt{m} + 2\mathtt{m} + \cdots + \mathtt{m}^2 = \mathtt{n} + \frac{\mathtt{m}^2 \times (\mathtt{m}-1)}{2}.$$

Cherchons un variant. On peut penser à $\mathtt{b} - \mathtt{m}$ mais il ne diminue pas à chaque étape. Nous verrons quel est ce variant plus tard dans le chapitre.

Nouvel exemple : retourner une liste. Essayons de distinguer les différents cas possibles : si la liste est vide, on la renvoie ; sinon, on extrait un élément `x` et on note le reste de la liste `xs`, on retourne `xs` puis on concatène à droite `x`. On peut donc écrire

```
let rec rev l =
  match l with
  | [] -> []
  | x :: xs -> (rev xs) @ [x];;
```

Code 1.3 – Inverser une liste

Cependant, le `@` est une opération lente en OCamL. En effet ce programme a une complexité en $\mathcal{O}(n^2)$, où n est la taille de la liste. Cette complexité est douteuse pour une opération aussi simple. On peut également se demander si cette opération se termine. Cela paraît très simple : la taille de la liste diminue mais nous n'avons pas le coté mathématique d'une liste. En effet, qu'est ce qu'une liste et la taille de cette liste ? On doit formaliser l'explication de pourquoi cet algorithme se termine.

Continuons avec un autre exemple :

```
let rec mystere3 m n =
  if m = 0 then n + 1
  else if n = 1 then mystere (n - 1) 1
  else mystere (m - 1) (mystere m (n-1));;
```

Code 1.4 – Un programme mystère (3)

Il s'agit de la fonction Ackermann. Sa complexité est très importante mais ce n'est pas le sujet de cette introduction. En effet, on a

$$A_{0,m} = n + 1$$
$$A_{m,0} = A_{n-1,1}$$
$$A_{m,n} = A_{m-1,A_{m,\ldots}}$$

Malgré ce que l'on peut penser, cette fonction se termine mais comment le prouver ?

À faire : Exemple arbres binaires

On en conclut que, avec les outils de l'année passée, il est difficile de prouver que ces algorithmes se terminent rigoureusement.

–1.2 Ordre

Définition (Élements minimaux)**:** Lorsque $(E, \preccurlyeq)$ est un espace ordonné, et $A \subseteq E$ ("inclut ou égal") est une partie de E, on appelle *élément minimal* de A un élément $x \in A$ tel que

$$\forall y \in A,\ y \preccurlyeq x \implies y = x.$$

Exemple:
La figure ci-dessous est un diagramme de Hasse : c'est un diagramme où les points représente les éléments de l'ensemble E et où les segments représentent une comparaison entre les deux éléments connectés : l'élément inférieur est représenté plus bas. Dans l'exemple ci-dessous, les éléments minimaux de A sont les points b et f.

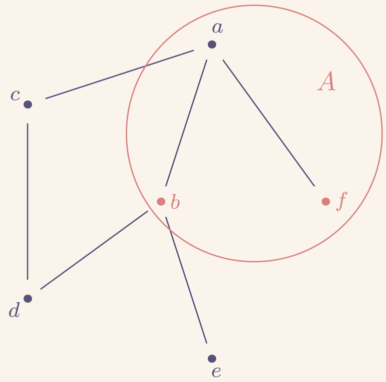


Figure 1.2 – Diagramme de Hasse

> **Définition** (Ordre bien fondé)**:** Un ordre est *bien fondé* s'il n'existe pas de suite infiniment strictement croissante.

Exemple:

Oui	Non
$(\mathbb{N}, \leqslant)$	$(\mathbb{Z}, \leqslant)$
	$(\mathbb{R}, \leqslant)$
	$(\mathbb{R}^+, \leqslant)$ $(1/2^n)$
$(E, \subseteq)$ (si E est fini)	$(E, \subseteq)$ (en général)

Table 1.1 – Exemples et non-exemples d'ordres bien fondés

Propriété: Une relation d'ordre $\preccurlyeq$ sur un ensemble E bien fondé si et seulement si toute partie non vide de E admet un élément minimal.

Preuve: " $\Longleftarrow$ " Supposons que toute partie non vide de E admet un élément minimal. Supposons, de plus, qu'il existe une suite $(x_n)_{n\in\mathbb{N}}$ infiniment strictement décroissante. Soit alors $A = \{u_n \mid n \in \mathbb{N}\}$ qui admet un élément minimal ; soit n_0 son indice. Or, $x_{n_0+1} \preccurlyeq x_{n_0}$ ce qui est absurde.

" $\Longrightarrow$ " Supposons que $(E, \preccurlyeq)$ est un ensemble bien fondé. Supposons également qu'il existe un sous-ensemble A de E non vide n'admettant pas d'élément minimal. Comme A est non vide, on pose alors $x_0 \in A$. Et, comme A n'admet pas d'élément minimal, donc il existe $x \in A$ tel que $x \preccurlyeq x_0$. Notons un tel x, x_1. En itérant ce procédé, on crée la suite $(x_i)_{i\in\mathbb{N}}$ qui est infiniment strictement décroissante ; ce qui est absurde.

□

Théorème (Induction bien fondée)**:** Soit $(E, \preccurlyeq)$ un ensemble ordonné et bien fondé. Soit P une propriété sur les éléments de E. Si $x \in E$, on note $E^{\preccurlyeq x} = \{y \in E \mid y \preccurlyeq x\}$. Si $\forall x \in E,\ \big(\forall y \in E^{\preccurlyeq x},\ P(y)\big) \implies P(x)$, alors $\forall x \in E,\ P(x)$.

Remarque:
Si $(E, \preccurlyeq) = (\mathbb{N}, \leqslant)$, alors le théorème précédent se traduit par :

$$\text{si } \forall n \in \mathbb{N},\ \big(\forall p < n,\ P(p)\big) \implies P(n), \text{ alors } \forall n \in \mathbb{N}, P(n).$$

Ce résultat correspond à la "récurrence forte." Décomposons ce "$\forall$" : on extrait le cas où $n = 0$

$$\text{si } P(0) \text{ et } \forall n \in \mathbb{N}^*,\ \big(\forall p < n, P(p)\big) \implies P(n), \text{ alors } \forall n \in \mathbb{N}, P(n).$$

On peut donc utiliser le principe de la récurrence pour tout ensemble ordonné bien fondé.

Preuve:
Soit $A = \{x \in E \mid P(x) \text{ n'est pas vrai}\}$.

Cas 1 $A = \varnothing$, alors OK.

Cas 2 $A \neq \varnothing$. Soit alors $x \in A$ un élément minimal de A (c.f. proposition d'avant). On a que $\forall y \in E,\ y \preccurlyeq x,\ P(y)$ est vrai donc $P(x)$ est vrai par hypothèse. Ce qui est absurde.

□

–1.2.1 Ordre produit

Définition: Soit $(A, \preccurlyeq_A)$ et $(B, \preccurlyeq_B)$ deux ensembles ordonnés on définit alors $\preccurlyeq_\times$ sur $A \times B$ par

$$\forall (a,b), (a',b') \in A \times B,\ (a,b) \preccurlyeq_\times (a',b') \overset{\text{def.}}{\iff} (a \preccurlyeq_A a' \text{ et } b \preccurlyeq_B b').$$

Propriété: $\preccurlyeq_\times$ est une relation d'ordre. □

La proposition précédente est facilement vérifiée comme $\preccurlyeq_A$ et $\preccurlyeq_B$ sont, elles aussi, des relations d'ordre.

Remarque (⚠):
Si $\preccurlyeq_A$ et $\preccurlyeq_B$ sont des ordres totaux, $\preccurlyeq_\times$ ne l'est pas forcément.

Propriété: Soient $(A, \preccurlyeq_A)$ et $(B, \preccurlyeq_B)$ bien fondés, alors $(A \times B, \preccurlyeq_\times)$ l'est aussi.

Preuve:
Supposons alors que $(A \times B, \preccurlyeq_\times)$ ne soit pas bien fondée. Nous avons donc une suite infiniment strictement décroissante

$$(a_0, b_0) \succ_\times (a_1, b_1) \succ_\times (a_2, b_2) \succ_\times \cdots.$$

On a donc $a_0 \succcurlyeq_A a_1 \succcurlyeq_A a_2 \succcurlyeq \cdots$. Or, $(A, \preccurlyeq_A)$ est bien fondée donc il existe $n_0 \in \mathbb{N}$ tel que $\forall i \geqslant n_0, a_i = a_{n_0}$. On a donc $\forall i \geqslant n_0,\ b_i \prec_B b_{i-1}$. Considérons $(b_i)_{i \geqslant n_0}$ est infiniment strictement décroissante dans $(B, \preccurlyeq_B)$, ce qui est absurde. □

Remarque:
On a défini une relation "produit," on peut se demander si ces résultats s'appliquent aussi si

la relation est “somme.” Ce n’est pas le cas : soit $\preccurlyeq_+$ définie comme

$$(a,b) \preccurlyeq_+ (a',b') \overset{\text{def.}}{\iff} a \preccurlyeq_A a' \text{ ou } b \preccurlyeq_B b'.$$

On peut démontrer que ce n’est pas une relation d’ordre.

Remarque:
Si $(A, \preccurlyeq_A)$ est une relation d’ordre alors $(A^n, \preccurlyeq_\times)$ a les même propriétés.

Remarque:
Sur $A^{\mathbb{N}}$, on définit $(u_n)_{n\in\mathbb{N}} \preccurlyeq_\times (v_n)_{n\in\mathbb{N}}$ si et seulement si

$$\forall i,\ u_i \preccurlyeq_A v_i.$$

L’ensemble ordonné $(A^{\mathbb{N}}, \preccurlyeq_\times)$ est il bien fondé ? La réponse est non.

Voici un contre-exemple : on pose $A = \{0, 1\}$.

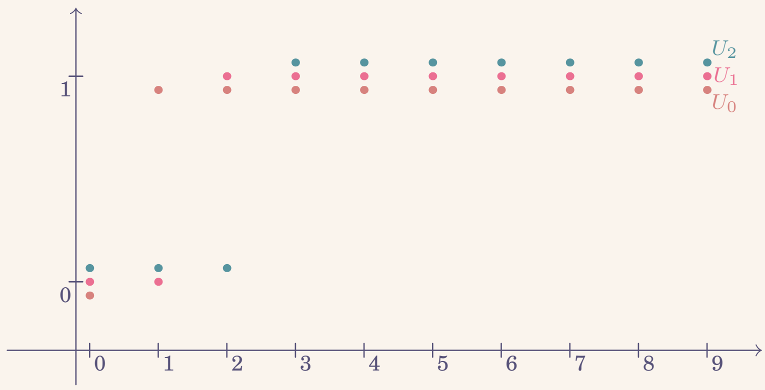


Figure 1.3 – Contre exemple : $(A^{\mathbb{N}}, \preccurlyeq_\times)$ est il bien fondé ?

On considère la suite U_0 qui a pour tout $n \in \mathbb{N}$ la valeur de 1. Puis, on considère la suite U_1 qui a, pour $n = 0$, la valeur de 0 puis pour les autres valeurs de n, la valeur de 1. Ensuite, on considère la suite U_2 qui, pour $n = 0, 1$, la valeur de 0 puis, pour les autres valeurs de n, la valeur de 1. En itérant ce procédé, on crée une suite de suite $(U_n)_{n\in\mathbb{N}}$ infiniment strictement décroissante :

$$U_0 \succcurlyeq_\times U_1 \succcurlyeq_\times U_2 \succcurlyeq_\times \cdots.$$

On considère le programme suivant :

```
let rec pgcd a b =
  if a = b then a
  else if a > b then pgcd (a-b) b
  else pgcd a (b-a);;
```

Code 1.5 – Calcul du PGCD

Étudions ce programme. Ce programme se termine si et seulement si `a` $= 0$ et `b` $= 0$ où si `a` > 0 et `b` > 0. Prouvons-le rigoureusement. On choisit comme variant (a, b) vivant dans l’ensemble ordonné $(\mathbb{N}^* \times \mathbb{N}^*, \preccurlyeq_\times)$ où $\preccurlyeq_\times$ est la relation d’ordre produit. À faire : Recopier une partie du cours ici. On a donc bien une décroissance stricte de la valeur de l’expression (a, b)) valeurs dans un espace bien fondé. D’où terminaison.

Démontrons maintenant la correction, c’est-à-dire, démontrons que

$$\forall (a,b) \in \mathbb{N}^* \times \mathbb{N}^*,\ (\texttt{pgcd}\ a\ b) = a \wedge b.$$

Pour cela, on procède par induction sur $\big((\mathbb{N}^*)^2, \preccurlyeq_\times\big)$ pour démontrer la proposition

$$P(a,b) = (\texttt{pgcd}\ a\ b) = a \wedge b.$$

— Soit $(a,b)=(1,1)$. On a
$$(\texttt{pgcd}\ a\ b) \underset{\text{(code)}}{=} a = a \wedge b.$$
— Soit $(a,b) \neq (1,1) \in (\mathbb{N}^*)^2$ tel que pour tout $(c,d) \in (\mathbb{N}^*)^2$ tel que $(c,d) \prec_\times (a,b)$ on ait $P(c,d)$. Montrons donc $P(a,b)$.
  — Si $a=b$:
$$(\texttt{pgcd}\ a\ b) \underset{\text{(code)}}{=} a = a \wedge b.$$
  — Si $a>b$:
$$\begin{aligned}(\texttt{pgcd}\ a\ b) &\underset{\text{(code)}}{=} (\texttt{pgcd}\ (a-b)\ b)\\ &\underset{\text{(hypothèse)}}{=} (a-b)\wedge b\\ &\underset{\text{(maths)}}{=} a\wedge b.\end{aligned}$$

–1.2.2 Ordre lexicographique

Définition: Soit $(A, \preccurlyeq_A)$ et $(B, \preccurlyeq_B)$ deux ensembles ordonnés, on définit alors sur $A \times B$ l'ordre
$$(a,b) \preccurlyeq_\ell (a',b') \overset{\text{def.}}{\iff} (a \prec_A a') \text{ ou } (a = a' \text{ et } b \preccurlyeq_B b').$$

Exemple:
Dans $(\mathbb{N}^2, \preccurlyeq_\times)$, on cherche les éléments $(x,y) \in \mathbb{N}^2$ tels que $(x,y) \preccurlyeq_\ell (3,4)$:

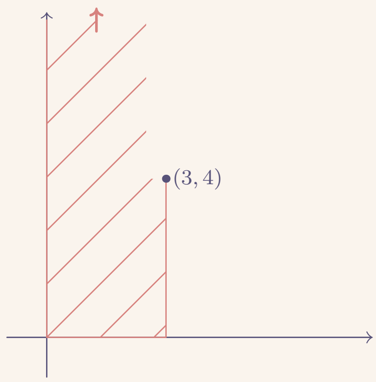


Figure 1.4 – Ordre lexicographique sur $\mathbb{N}^2$

Propriété: $\preccurlyeq_\ell$ est une relation d'équivalence.

Preuve:
À faire □

Propriété: Si $\preccurlyeq_A$ est totale et $\preccurlyeq_b$ est totale alors $\preccurlyeq_\ell$ est totale.

Preuve:
Soit (a,b) et $(c,d) \in (A \times B)$.

— Si $a \prec_A c$, alors $(a,b) \prec_\ell (c,d)$.
— Si $a = c$,
  — si $b \preccurlyeq_B d$ alors $(a,b) \preccurlyeq_\ell (c,d)$.
  — sinon ($b \succcurlyeq_B d$) alors $(a,b) \succ_\ell (c,d)$.
— si $a \succ_A c$ alors $(c,d) \prec_\ell (a,b)$.

□

Propriété: Si $(A, \preccurlyeq_A)$ et $(B, \preccurlyeq_B)$ sont bien fondés, alors $(A \times B, \preccurlyeq_\ell)$ l'est aussi.

Preuve:
À rédiger. □

Remarque:
On peut généraliser à un ensemble $(A^n, \preccurlyeq_\ell)$.

Par exemple, avec $n = 3$, on a

$$(a,b,c) \preccurlyeq_\ell (a',b',c') \overset{\text{def.}}{\iff} a \prec_A a' \text{ ou } (a = a' \text{ et } b \prec_B b') \text{ ou } (a = a' \text{ et } b = b' \text{ et } c \prec_C c').$$

Même question qu'avec l'ordre produit, l'ensemble $(A^{\mathbb{N}}, \preccurlyeq_\ell)$ est-il bien fondé ? De même, la réponse est non, la même suite de suite est un contre-exemple.

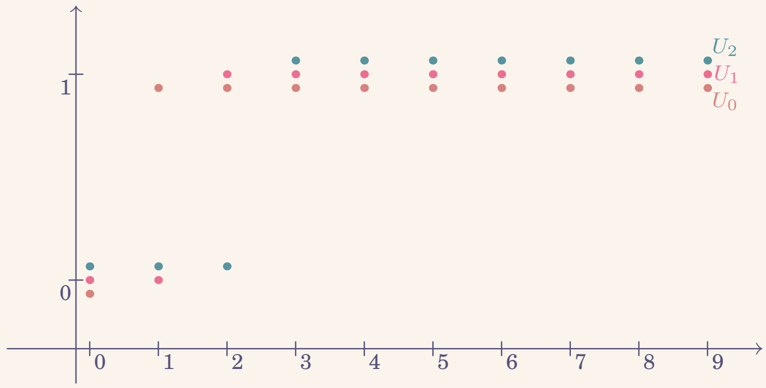


Figure 1.5 – Contre exemple : $(A^{\mathbb{N}}, \preccurlyeq_\ell)$ est il bien fondé ?

L'ordre lexicographique est, comme son nom l'indique, l'ordre utilisé dans le dictionnaire. La seule différence est que l'on peut comparer des mots de longueurs différentes.

Rappel:
Si A est un ensemble, alors $A^* = \bigcup_{n \in \mathbb{N}} A^n$. L'ensemble A^* contient toutes les suites finies d'éléments de A.

Par exemple, avec $A = \{0, 1\}$, on a

$$A^* \supseteq \{(\,), (1), (0), (0,1), (1,0), (1,1), (0,0), (1,1,0), (0,0,0), \dots\}.$$

Définition: Si $(A, \preccurlyeq_A)$ est un ensemble ordonné, on définit sur A^* :

$$(u_p)_{p\in[\![1,n]\!]} \prec_\ell (v_p)_{p\in[\![1,m]\!]} \overset{\text{def.}}{\iff} \begin{array}{l} \exists i \in [\![1, \min(n,m)+1]\!]\,, \\ (\forall j \in [\![1, i-1]\!]\,,\ u_j = v_j) \\ \text{et } (i = n+1 \text{ ou } u_i \prec_A v_i). \end{array}$$

Propriété: C'est une relation d'ordre. Elle est totale si $\preccurlyeq_A$ est totale. □

Même question avec cette nouvelle définition de l'ordre lexicographique, l'ensemble $\left((A^*)^{\mathbb{N}}, \preccurlyeq_\ell\right)$ est-il bien fondé ? De même, la réponse est non. Voici un contre-exemple : on considère la suite de suite $U_0 = (1)$, $U_1 = (0,1)$, $U_2 = (0,0,1)$. On crée une suite infiniment strictement décroissante.

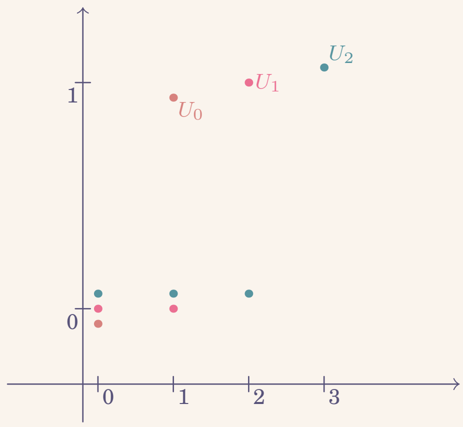


Figure 1.6 – Contre exemple : $\left((A^*)^{\mathbb{N}}, \preccurlyeq_\ell\right)$ est il bien fondé ? (2)

```
let rec mystere3 m n =
  if m = 0 then n + 1
  else if n = 1 then mystere (n - 1) 1
  else mystere (m - 1) (mystere m (n-1));;
```

Code 1.6 – La fonction Ackermann

La fonction Ackermann utilise l'ordre lexicographique ; dans ce cas ci, l'ensemble ordonné est bien fondé. C'est comme cela que l'on a la terminaison de cette fonction.

Prenons un autre exemple :

```
let rec mystere n m p =
  if m > 0 then 1 + mystere n (m - 1) p
  else if m = 0 && n > 0 then 1 + mystere (n
      -1) p (p+1)
  else 0
```

Code 1.7 – Une fonction mystère (5)

À faire :

- s'assurer de la terminaison (comme celle du PGCD)
- démontrer (par induction) que

$$\forall (m,n,p) \in \mathbb{N}^3,\ (\texttt{mystere}\ n\ m\ p) = \frac{n(n+1)}{2} + pn + m.$$

Les preuves de correction et de terminaison sont basées sur la supposition qu'un entier en OCaml est identique à un entier mathématique.

–1.3 Induction nommée

Définition (Règle de Construction nommée)**:** On appelle *Règle de Construction nommée* la donnée de

- un symbole S,
- un entier $r \in \mathbb{N}$,
- un ensemble non vide C.

On écrira alors cette règle

$$\text{“ } S\Big|_C^r \text{ ”} \qquad \text{ou encore} \qquad \text{“}S(y, \underbrace{\square, \square, \ldots, \square}_{r}) \text{ pour } y \in C.\text{”}$$

Remarque:
On a parfois besoin d'un ensemble C trivial (de taille 1 et contenant un objet inutile), on note alors la règle $S\Big|^r$.

Exemple:
Les symboles sont écrits en rouge afin de les différencier.

$$0\Big|^0 \qquad S\Big|^1$$
$$[\,]\Big|^0 \qquad ::\Big|_{\mathbb{N}}^1$$
$$V\Big|^0 \qquad N\Big|_{\mathbb{N}}^2$$

Définition:

- On appelle *règle de base* une règle de la forme $S|_C^0$.
- On appelle *règle d'induction* une règle de la forme $S|_C^n$.

Définition: Étant donné un ensemble <u>fini</u> de règles $R = \underbrace{B}_{\text{règle de base}} \cup \overbrace{I}^{\text{règle d'induction}}$ avec $B \neq \varnothing$, on définit alors

$$X_0 = \left\{ (S, a) \;\middle|\; S|_C^r \text{ et } a \in C \right\}$$

puis, pour tout $n \in \mathbb{N}$,

$$X_{n+1} = X_n \cup \left\{ (S, a, t_1, t_2, \ldots, t_r) \;\middle|\; S|_C^r \in \underbrace{R}_{C},\ a \in C,\ t_1 \in X_n,\ t_2 \in X_n, \ldots, t_r \in X_n \right\}.$$

On appelle alors $\bigcup_{n \in \mathbb{N}} X_n$ l'ensemble défini par induction nommée à partir des règles de R.

Remarque (Notation):
On note un n-uplet ayant pour premier élément un symbole S puis $n-1$ éléments $(a_1, \ldots, a_{n-1})$. Au lieu de $(S, a_1, \ldots, a_{n-1})$, on note $S(a_1, \ldots, a_{n-1})$.

Exemple:
On pose

$$R = \left\{ A|_{\mathbb{N}}^0, B|^1, C_{\{0,1\}}^2 \right\}.$$

À faire : À finir

Remarque:

Pour l'ensemble A défini par induction à partir de $R = \left\{0\big|^0, S\big|^1\right\}$, on dira plutôt

"Soit A l'ensemble défini par induction tel que $0 \in A$ et $\forall a \in A,\ S(a) \in A$."

Définition: Soit R un ensemble fini de règles et A l'ensemble défini par induction à partir de ces règles. Sur A, on définit la relation binaire $\diamond$ par

$$x \diamond y \overset{\text{def.}}{\iff} y = S(\ldots, x, \ldots) \text{ avec } S\big|_C^r \in R.$$

On définit alors

$$x \preccurlyeq y \overset{\text{def.}}{\iff} \exists p \in \mathbb{N},\ \exists (a_1, \ldots, a_p) \in A^p,\ x \diamond a_1 \text{ et } a_1 \diamond a_2 \text{ et } \ldots \text{ et } a_p \diamond y \text{ ou } x = y.$$

Définition (hauteur)**:** Soit R un ensemble fini de règles d'induction nommée définissant un ensemble $A = \bigcup_{n \in \mathbb{N}} X_n$. On définit alors

$$\begin{aligned} h : A &\longrightarrow \mathbb{N} \\ x &\longmapsto \min\{n \in \mathbb{N} \mid x \in X_n\}. \end{aligned}$$

Remarque:
Si $x \in a$, il existe alors $n_0 \in \mathbb{N}$ tel que $x \in X_{n_0}$ donc $\{n \in \mathbb{N} \mid x \in X_n\} \neq \varnothing$ donc le minimum existe.

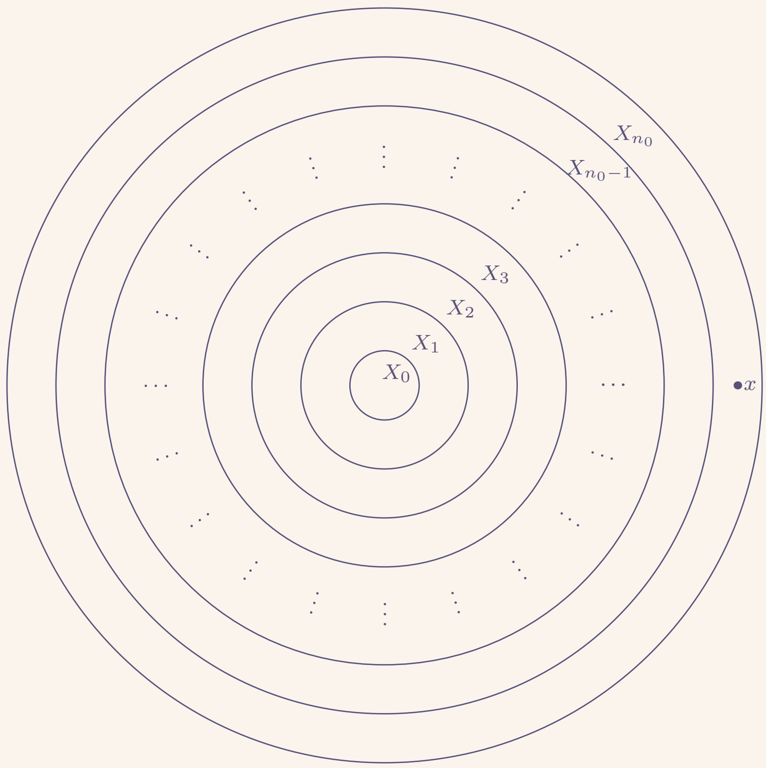


Figure 1.7 – Structure des ensembles $X_1, \ldots, X_n$

> **Propriété:** Si $x \diamond y$, alors $h(x) < h(y)$.

Preuve:
Soit $x \diamond y$. Alors, $y \in A$. Soit $n_0 = h(y) \neq 0$, on a donc $y \in X_{n_0}$.
- Si $y \in X_{n_0-1}$ ce qui est absurde par définition de h.
- Si $y \in \left\{ S(a, t_1, \dots, t_r) \;\middle|\; S|_C^r \in R \text{ et } a \in C \text{ et } t_1 \in X_{n_0-1} \text{ et } \dots \text{ et } t_r \in X_{n_0-1} \right\}$.
  Or, soit i_0 tel que $x = t_{i_0}$ donc $x \in X_{n_0-1}$ et donc $h(x) \leqslant n_0 - 1$.

$\square$

> **Corollaire:** Si $n \preccurlyeq y$, alors $h(x) = h(y) \iff x = y$ et $h(x) < h(y) \iff x \prec y$.

> **Corollaire:** La relation $\preccurlyeq$ est antisymétrique.

Remarque:
La relation $\preccurlyeq$ est trivialement transitive et reflective. Elle est donc d'ordre.

Exemple:
On prend $R = \left\{ A|^0, B|^0 \right\}$ et $X_0 = \{A, B\}$, $X_1 = X_0, \dots$ L'ensemble S défini par induction sur R est $\{A, B\}$. On en déduit que $\preccurlyeq$ n'est pas totale.

Exemple:
On prend $R = \left\{ 0|^0, S|^1 \right\}$, $X_0 = \{0\}$, $X_1 = \{0, S(0)\}$ (on a $0 \prec S(0)$), $X_2 = \{0, S(0), S(S(0))\}$ (on a $0 \prec S(0) \prec S(S(0))$). L'ensemble obtenu est

$$\begin{array}{c} 0 \\ | \\ S(0) \\ | \\ S(S(0)) \\ | \\ S(S(S(0))) \\ | \\ \vdots \end{array}$$

Figure 1.8 – Ensemble obtenu avec les règles S et 0

Exemple:
On pose $R = \left\{ ::|_{\mathbb{N}}^1, [\,]|^0 \right\}$ et $X_0 = \{[\,]\}$, $X_1 = \{::(0, [\,]), [\,], ::(1, [\,], ::(2, [\,])\}$, et $X_2 = \{[\,], ::(0, [\,]), ::(1, ::(0, [\,]))\}$. L'ensemble obtenu est

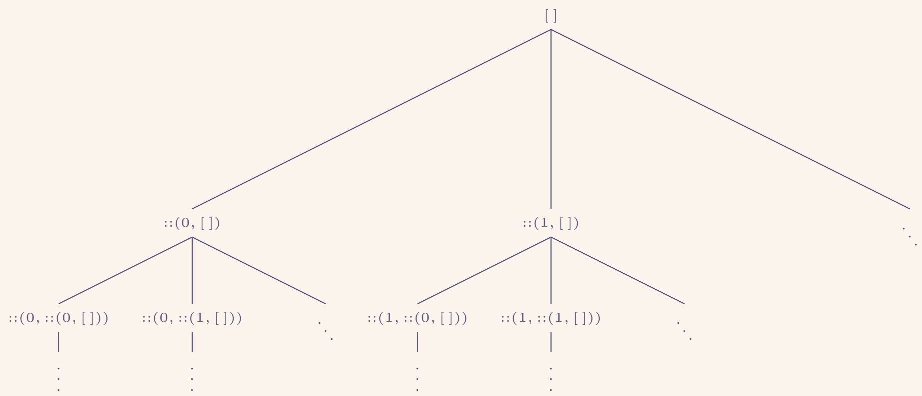


Figure 1.9 – Ensemble obtenu avec les règles :: et [ ]

> **Propriété:** La relation $\preccurlyeq$ est bien fondée.

Preuve:
Soit $(x_n)_{n\in\mathbb{N}}$ une suite telle que $x_0 \succ x_1 \succ x_2 \succ \cdots \succ x_n \succ \cdots$ alors

$$\underbrace{h(x_0) > h(x_1) > \cdots > h(x_n) > \cdots}_{\in(\mathbb{N},\leqslant)}.$$

Or, $(\mathbb{N}, \leqslant)$ est bien fondé, c'est donc absurde. $\square$

Remarque:
On peut faire des preuves par induction bien fondée.

Exemple:

— Soit l'ensemble trivial défini par $\left\{A\big|^0, B\big|^0\right\}$, le théorème est trivial.

— Soit $\mathcal{N}$ défini par $\mathcal{N} = \left\{0\big|^0, S\big|^1\right\}$. Le théorème donne :

si $P(0)$ vrai et $\forall n \in \mathbb{N}$, $(\forall p \in \mathbb{N},\ p < n-1,\ P(\underbrace{S(S(\cdots(S}_{p}(0)\cdots)))) \implies P(\underbrace{S(S(\cdots(S}_{n}(0)\cdots))))$

alors

$$\forall n \in \mathbb{N},\ P(\underbrace{S(S(\cdots(S}_{n}(0)\cdots)))).$$

— Soit $\mathcal{L}$ défini par $\mathcal{L} = \left\{[\,]\big|^0, ::\big|^1_{\mathbb{N}}\right\}$. Si $P([\,])$, et $\forall \ell \in \mathcal{L}, \forall n \in \mathbb{N}, P(\ell) \implies P(::(n, \ell))$ alors $\forall \ell \in \mathcal{L},\ P(\ell)$.

> **Propriété:** Étant donné,
>
> — un ensemble A défini par induction à partir d'un ensemble de règles nommé R,
>
> — un ensemble $\mathbb{I}$,
>
> — une fonction $f_T : C \times \mathbb{I}^r \to \mathbb{I}$ pour chaque règle $T = S\big|^r_C \in R$,
>
> on défini de manière unique une fonction $f : A \to \mathbb{I}$ telle que, pour tout $x = S(a, t_1, \ldots, t_r) \in A$, soit $T = S\big|^r_C \in R$ alors $f(x) = f_T\big(a, f(t_1), \ldots, f(t_r)\big)$.

Exemple:

Sur $\mathscr{L}$ défini par $\Big\{\underbrace{[\,]\big|^{0}}_{R_1}, \underbrace{::\big|^{1}_{\mathbb{N}}}_{R_2}\Big\}$, on choisit $\mathbb{I} = \mathbb{N}$ et

$$f_{R_1} : \overbrace{(\_,\_)}^{\in \text{Inutile} \times \mathbb{N}^0} \mapsto 0$$
$$f_{R_2} : (\underbrace{t}_{\mathbb{N}}, i) \mapsto t + i.$$

On définit alors la fonction

$$\begin{aligned} f : \mathscr{L} &\longrightarrow \mathbb{N} \\ ::(a_1,\, ::(a_2, ::\cdots ::(a_n, [\,])\cdots) &\longmapsto \sum_{i=1}^{n} a_i. \end{aligned}$$

CHAPITRE

0 LOGIQUE

Sommaire

0.1 Motivation

Considérons la grilles de Sudoku 2×2 suivant

3			2
	4	1	
	3	2	
4			1

Figure 0.1 – Grille de Sudoku 2×2

On modélise ce problème : on considère $P_{i,j,k}$ une variable booléenne, c'est à dire un élément de $\{\boldsymbol{V},\ \boldsymbol{F}\}$, définie telle que

$$P_{i,j,k} : \text{“}m(i,j) \overset{?}{=} k\text{”} \text{ avec } (i,j,k) \in [\![1,4]\!]^3 ..$$

On peut définir des contraintes logiques (des expressions logiques) pour résoudre le Sudoku. Les opérateurs ci-dessous seront définis plus tard.

$$\begin{aligned}
&P_{113}\\
\wedge\ &P_{1,4,2}\\
\wedge\ &P_{2,2,4}\\
\wedge\ &P_{2,3,1}\\
&\vdots\\
\wedge\ &P_{1,2,1} \to (\neg P_{1,2,2} \wedge \neg P_{1,2,3} \wedge \neg P_{1,2,4})\\
&\vdots
\end{aligned}$$

Pour résoudre le Sudoku, on peut essayer chaque cas possible. Mais, ces possibilités sont très nombreuses.

En mathématiques, on utilise une certaine logique. Il en existe d'autre, certaines où tout est vrai, certaines où il est plus facile de montrer des théorèmes, etc. On va définir une logique ayant le moins d'opérateurs possibles.

0.2 Syntaxe

Définition: On suppose donné un ensemble $\mathscr{P}$ de variables propositionnelles.

Définition: On définit alors l'ensemble des formules de la logique propositionnelle par induction nommée avec les règles :

— $\neg|^1$;
— $\wedge|^2$;
— $\vee|^2$;
— $\to|^2$;
— $\leftrightarrow|^2$;
— $\top|^0$;
— $\bot|^0$;
— $V|^0_{\mathscr{P}}$.

On nomme l'ensemble des formules $\mathscr{F}$.

Exemple:

$$\vee(\wedge(\to(V(P), \top(\,), \neg(\bot(\,))), \vee(\leftrightarrow(\top(\,), \top(\,)), V(r)).$$

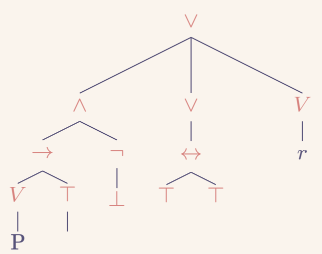


Figure 0.2 – Arbre syntaxique d'une expression logique

Pour simplifier la syntaxe, on écrit plutôt

$$\big((p \to \top) \wedge \neg\bot\big) \vee \big((\top \leftrightarrow \top) \vee r\big).$$

Définition (taille d'une formule)**:** On définit, par induction, la taille notée "taille" comme

$$\begin{aligned} \text{taille} : \mathcal{F} &\longrightarrow \mathbb{N} \\ p \in \mathcal{P} &\longmapsto 1 \\ \top &\longmapsto 1 \\ \bot &\longmapsto 1 \\ \neg G &\longmapsto 1 + \text{taille}(G) \\ G \to H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H) \\ G \leftrightarrow H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H) \\ G \wedge H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H) \\ G \vee H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H). \end{aligned}$$

Définition (Ensemble des variables propositionnelles)**:** On définit inductivement

$$\begin{aligned} \text{vars} : \mathcal{F} &\longrightarrow \wp(\mathcal{P}) \quad {}^{1} \\ p \in \mathcal{P} &\longmapsto \{p\} \\ \top, \bot &\longmapsto \varnothing \\ \neg G &\longmapsto \text{vars}(G) \\ G \odot H &\longmapsto \text{vars}(G) \cup \text{vars}(H) \end{aligned}$$

où $\odot$ correspond à $\cup$, $\cap$, $\to$ ou $\leftrightarrow$.

Définition: On appelle *substitution* une fonction de $\mathcal{P}$ dans $\mathcal{F}$ qui est l'identité partout sauf sur un ensemble fini de variables. On la note alors

$$(p_1 \mapsto H_1,\ p_2 \mapsto H_2, \ldots, p_n \mapsto H_n)$$

qui est la substitution

$$\begin{aligned} \mathcal{P} &\longrightarrow \mathcal{F} \\ p &\longmapsto \begin{cases} H_i & \text{si } p = p_i \\ p & \text{sinon.} \end{cases} \end{aligned}$$

Exemple:
La fonction

$$\sigma = (p \mapsto p \vee q,\ r \mapsto p \wedge \top)$$

est une substitution. On a $\sigma(p) = p \vee q$, $\sigma(r) = p \wedge \top$, $\sigma(q) = q$ et, pour toute autre variable logique a, $\sigma(a) = a$.

Définition (Application d'une substitution à une formule)**:** Étant donné une formule $G \in \mathcal{F}$ et une substitution σ, on définit inductivement $G[\sigma]$ par

$$\begin{cases} \top[\sigma] = \top \\ \bot[\sigma] = \bot \\ p[\sigma] = \sigma(p) \\ (\neg G)[\sigma] = \neg(G[\sigma]) \\ (G \odot H)[\sigma] = (G[\sigma]) \odot H[\sigma] \end{cases}$$

1. Le $\wp(E)$ représente ici l'ensemble des parties de E.

où $\odot$ correspond à $\cup$, $\cap$, $\to$ ou $\leftrightarrow$.

Exemple:
Avec $G = p \wedge (q \vee \top)$ et $\sigma = (p \mapsto p,\ q \mapsto r \wedge \top)$, on a

$$G[\sigma] = q \wedge ((r \wedge \top) \vee \bot).$$

Définition: On appelle parfois *clés* d'une substitution de σ, l'ensemble des variables propositionnelles sur lequel elle n'est pas l'identité.

Définition: On définit la *composée* de deux substitutions σ et σ' par

$$\begin{aligned} \sigma \cdot \sigma' : \mathcal{P} &\longrightarrow \mathcal{F} \\ p &\longmapsto (p[\sigma])[\sigma]. \end{aligned}$$

Exemple:
Avec $\sigma = (p \mapsto q)$ et $\sigma = (q \mapsto r)$, on a

$$\sigma' \cdot \sigma = (p \mapsto r,\ q \mapsto r).$$

En effet,

$$\sigma' \cdot \sigma(x) = \begin{cases} r & \text{si } x = p \\ r & \text{si } x = q \\ x & \text{sinon.} \end{cases}$$

Exemple:
Avec $\sigma = (p \mapsto q \wedge \top)$, $\sigma' = (q \mapsto \bot,\ r \mapsto p)$, on a

$$\begin{aligned} \sigma' \cdot \sigma(x) &= \begin{cases} \bot \wedge \top & \text{si } x = p \\ \bot & \text{si} x = q \\ p & \text{si } x = r \\ x & \text{sinon} \end{cases} \\ &= (p \mapsto \bot \wedge \top,\ q \mapsto \bot,\ r \mapsto p). \end{aligned}$$

Remarque:
L'opération $\cdot$ est associative.

Propriété: Soient σ et σ' deux substitutions, on a, pour toute formule $H \in \mathcal{F}$,

$$(H[\sigma])[\sigma'] = H[\sigma' \cdot \sigma].$$

Preuve:
Notons P_G la propriété "$(G[\sigma])[\sigma'] = G[\sigma' \cdot \sigma]$"

Montrons que, pour toute formule $G \in \mathcal{F}$, P_G est vraie par induction :
- $(\top[\sigma])[\sigma'] =^{(\text{def})} \top = \top[\sigma' \cdot]$
- $(p[\sigma])[\sigma'] = p[\sigma' \cdot \sigma]$
- à faire à la maison : le cas $\neg$ et un cas $\wedge$.

$\square$

Définition: On appelle *relation sous formule*, la relation définie Samedi.

À faire : Recopier cette formule (sinon ça va être drôle en Juin)

0.3 Sémantique

0.3.1 Algèbre de Boole

Définition: On note $\mathbb{B} = \{\boldsymbol{V}, \boldsymbol{F}\}$ l'*ensemble des booléens*.

Définition: Sur $\mathbb{B}$, on définit les opérateurs

a	b	$a \cdot b$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$

Table 0.1 – Opération $\cdot$ sur les booléens

a	b	$a + b$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$

Table 0.2 – Opération $+$ sur les booléens

a	$\bar{a}$
$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$

Table 0.3 – Opération $\bar{\cdot}$ sur les booléens

Nom	$\cdot$	$+$
Commutativité	$a \cdot b = b \cdot a$	$a + b = b + a$
Neutre	$\boldsymbol{V} \cdot a = a$	$\boldsymbol{F} + a = a$
Absorbant	$\boldsymbol{F} \cdot a = \boldsymbol{F}$	$\boldsymbol{V} \cdot a = \boldsymbol{V}$
Associativité	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$	$a + (b + c) = (a + b) + c$
Idempotence	$a \cdot a = a$	$a + a = a$
Distributivité	$a \cdot (b + c) = a \cdot b + a \cdot c$	$a + (b \cdot c) = (a + b) \cdot (a + c)$
Complémentaire	$a \cdot \bar{a} = \boldsymbol{F}$	$a + \bar{a} = \boldsymbol{V}$
Morgan	$\overline{a \cdot b} = \bar{a} + \bar{b}$	$\overline{a + b} = \bar{a} \cdot \bar{b}$

Table 0.4 – Règles dans $\mathbb{B}$

Remarque:

0.3.2 Fonctions booléennes

Définition (Environnement propositionnel)**:** On appelle *environnement propositionnel* une fonction de $\mathscr{P}$ dans $\mathbb{B}$.

Définition: On appelle *fonction booléenne* une fonction de $\mathbb{B}^{\mathscr{P}}$ dans $\mathbb{B}$. On note l'ensemble des fonctions booléennes $\mathbb{F}$.

Remarque:
Si $|\mathscr{P}| = n$, alors $\left|\mathbb{B}^{\mathscr{P}}\right| = 2^n$ et donc $|\mathbb{F}| = 2^{2^n}$.

Exemple:
La fonction

$$f : \begin{pmatrix} (p \mapsto \boldsymbol{F},\ q \mapsto \boldsymbol{F}) \mapsto \boldsymbol{F} \\ (p \mapsto \boldsymbol{F},\ q \mapsto \boldsymbol{V}) \mapsto \boldsymbol{V} \\ (p \mapsto \boldsymbol{V},\ q \mapsto \boldsymbol{F}) \mapsto \boldsymbol{V} \\ (p \mapsto \boldsymbol{V},\ q \mapsto \boldsymbol{V}) \mapsto \boldsymbol{V} \end{pmatrix} \in \mathbb{F}$$

est une fonction booléenne.

0.3.3 Interprétation d'une formule comme une fonction booléenne

Définition (Interprétation)**:** Étant donné une formule $G \in \mathscr{F}$ et un environnement propositionnel $\rho \in \mathbb{B}^{\mathscr{P}}$, on définit l'*interprétation* de G dans l'environnement ρ par

- $[\![\top]\!]^\rho = \boldsymbol{V}$;
- $[\![\bot]\!]^\rho = \boldsymbol{F}$;
- $[\![p]\!]^\rho = \rho(p)$ où $p \in \mathscr{P}$;
- $[\![\neg G]\!]^\rho = \overline{[\![G]\!]^\rho}$;
- $[\![G \wedge H]\!]^\rho = [\![G]\!]^\rho \cdot [\![H]\!]^\rho$;
- $[\![G \vee H]\!]^\rho = [\![G]\!]^\rho + [\![H]\!]^\rho$;
- $[\![G \to H]\!]^\rho = \overline{[\![G]\!]^\rho} + [\![H]\!]^\rho$;
- $[\![G \leftrightarrow H]\!]^\rho = \left(\overline{[\![G]\!]^\rho} + [\![H]\!]^\rho\right) \cdot \left(\overline{[\![H]\!]^\rho} + [\![G]\!]^\rho\right)$.

Exemple:
Avec $\rho = (p \mapsto \boldsymbol{V}, q \mapsto \boldsymbol{F})$, et $G = (p \wedge \top) \vee (q \wedge \bot)$, on a

$$\begin{aligned} [\![G]\!]^\rho &= [\![(p \wedge \top) \vee (q \wedge \bot)]\!]^\rho \\ &= [\![p \wedge \top]\!]^\rho + [\![q \wedge \bot]\!]^\rho \\ &= [\![p]\!]^\rho \cdot [\![\top]\!]^\rho + [\![q]\!]^\rho \cdot [\![\bot]\!]^\rho \\ &= \rho(p) \cdot \boldsymbol{V} + \rho(q) \cdot \boldsymbol{F} \\ &= \boldsymbol{V} + \boldsymbol{F} \\ &= \boldsymbol{V}. \end{aligned}$$

Définition (Fonction booléenne associée à une formule)**:** Étant donné une formule G, on note

$$\begin{aligned} \mathbb{F} \ni [\![G]\!] : \mathbb{B}^{\mathscr{P}} &\longrightarrow \mathbb{B} \\ \rho &\longmapsto [\![G]\!]^\rho . \end{aligned}$$

Exemple:

La fonction booléenne associée à $p \vee q$ est

$$f : \begin{pmatrix} (p \mapsto \boldsymbol{F},\ q \mapsto \boldsymbol{F}) \mapsto \boldsymbol{F} \\ (p \mapsto \boldsymbol{F},\ q \mapsto \boldsymbol{V}) \mapsto \boldsymbol{V} \\ (p \mapsto \boldsymbol{V},\ q \mapsto \boldsymbol{F}) \mapsto \boldsymbol{V} \\ (p \mapsto \boldsymbol{V},\ q \mapsto \boldsymbol{V}) \mapsto \boldsymbol{V} \end{pmatrix} \in \mathbb{F}.$$

La fonction booléenne associée à $p \vee (q \wedge \top)$ est aussi f ; tout comme $(p \vee \bot) \vee (q \wedge \top)$.

0.3.4 Liens sémantiques

Définition: On dit que G et H sont *équivalents* si et seulement si $[\![G]\!] = [\![H]\!]$. On note alors $G \equiv H$.

Définition (Conséquence sémantique)**:** On dit que H est *conséquence sémantique* de G dès lors que

$$\forall \rho \in \mathbb{B}^{\mathscr{P}},\ \big([\![G]\!]^\rho = \boldsymbol{V}\big) \implies \big([\![H]\!]^\rho = \boldsymbol{V}\big).$$

On le note $G \models H$.

Propriété: On a

$$G \equiv H \iff (G \models H \text{ et } H \models G).$$

Preuve: “ $\Longrightarrow$ ” On suppose $G \equiv H$. Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. On suppose $[\![G]\!]^\rho = \boldsymbol{V}$ alors $[\![H]\!]^\rho = \boldsymbol{V}$ car $[\![G]\!] = [\![H]\!]$. On suppose maintenant $[\![H]\!]^\rho = \boldsymbol{V}$, et alors $[\![G]\!]^\rho = \boldsymbol{V}$ car $[\![G]\!] = [\![H]\!]$.

“ $\Longleftarrow$ ” On suppose $G \models H$ et $H \models G$. Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. On suppose $[\![G]\!]^\rho = \boldsymbol{V}$ alors $[\![H]\!]^\rho = \boldsymbol{V}$ car $H \models H$ et donc $[\![G]\!] = [\![H]\!]$. On suppose maintenant $[\![H]\!]^\rho = \boldsymbol{V}$ alors $[\![G]\!]^\rho = \boldsymbol{V}$ car $G \models H$. Par contraposée, si $[\![G]\!]^\rho = \boldsymbol{F}$, alors $[\![H]\!]^\rho = \boldsymbol{F}$. On en déduit que $[\![G]\!] = [\![H]\!]$.

□

Remarque:
$\models$ n'est pas une relation d'ordre.

Remarque:
La relation $\equiv$ est une relation d'équivalence. De plus, si $G \equiv G'$ et $H \equiv H'$, alors
- $G \wedge H \equiv G' \wedge H'$;
- $G \vee H \equiv G' \vee H'$;
- $G \rightarrow H \equiv G' \rightarrow H'$;
- $G \leftrightarrow H \equiv G' \leftrightarrow H'$;
- $\neg G \equiv \neg G'$.

Une telle relation est parfois appelée une *congruence*.

Définition: On dit d'une formule $H \in \mathscr{F}$ qu'elle est
- *valide* ou *tautologique* dès lors que $\forall \rho \in \mathbb{B}^{\mathscr{P}},\ [\![H]\!]^\rho = \boldsymbol{V}$;
- *satisfiable* dès lors qu'il existe $\rho \in \mathbb{B}^{\mathscr{P}},\ [\![H]\!]^\rho = \boldsymbol{V}$;
- *insatisfiable* dès lors qu'il n'est pas satisfiable.

On dit de $\rho \in \mathbb{B}^{\mathscr{P}}$ tel que $[\![H]\!]^\rho = \boldsymbol{V}$ que ρ est un *modèle* de H.

Exemple:
- $p \vee \neg p$ est une tautologie. En effet, soit $\rho \in \mathscr{B}^{\mathscr{P}}$, on a
$$[\![p \vee \neg p]\!]^\rho = [\![p]\!]^\rho + \overline{[\![p]\!]^\rho} = \boldsymbol{V}.$$
- p est satisfiable mais non valide. En effet,
$$[\![p]\!]^{(p \mapsto \boldsymbol{V})} = \boldsymbol{V} \qquad \text{et} \qquad [\![p]\!]^{(p \mapsto \boldsymbol{F})} = \boldsymbol{F}.$$

— $p \wedge \neg p$ est insatisfiable. En effet, soit $\rho \in \mathbb{B}^{\mathscr{P}}$, on a

$$[\![p \wedge \neg p]\!]^\rho = [\![p]\!]^\rho \cdot \overline{[\![p]\!]^\rho} = \boldsymbol{F}.$$

Définition: Si Γ est un ensemble de formules, on écrit $\Gamma \models H$ pour dire que

$$\forall \rho \in \mathbb{B}^{\mathscr{P}},\ (\forall G \in \Gamma,\ [\![G]\!]^\rho = \boldsymbol{V}) \implies [\![H]\!]^\rho = \boldsymbol{V}.$$

Remarque:
Si Γ est fini, alors on a

$$\Gamma \models H \iff \Big(\bigwedge_{G \in \Gamma} G\Big) \models H.$$

On doit faire la preuve, pour $n \geqslant 1$,

$$\{G_1, G_2, \ldots, G_n\} \models H \iff (\cdots((G_1 \wedge G_2) \wedge G_3) \cdots \wedge G_n) \models H.$$

0.4 Le problème Sat – Le problème Validité

On définit le problème Sat comme ayant pour donnée une formule H et pour question "H est-elle satisfiable ?" et le problème Valide comme ayant pour donnée une formule H et pour question "H est-elle valide ?"

0.4.1 Résolution par tables de vérité

a	b	c	$a \wedge b$	$\neg b$	$\neg c$	$\neg b \vee \neg c$	$(a \wedge b) \to (\neg b \vee \neg c)$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$

Table 0.5 – Table de vérité de $(a \wedge b) \to (\neg b \vee \neg c)$

Exemple:

Le problème Sat lit la colonne résultat, on cherche un $\boldsymbol{V}$. Le problème Valide lit la colonne résultat et vérifie qu'il n'y a que des $\boldsymbol{V}$.

Remarque:
Deux formules sont équivalent si et seulement si elles ont la même colonne résultat.

On essaie d'énumérer toutes les possibilités : si $|\mathscr{P}| = n \in \mathbb{N}$, alors le nombre de classes d'équivalences pour $\equiv$ est au plus 2^{2^n}. On cherche donc un meilleur algorithme.

0.5 Représentation des fonction booléennes

0.5.1 Par des formules ?

p	q	r	S
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$

Table 0.6 – Table de vérité d'une formule inconnue

On regarde les cas où la sortie est $\boldsymbol{V}$ et on crée une formule permettant de tester cette combinaison de p, q et r uniquement. On unie toutes ces formules par des $\vee$. Dans l'exemple ci-dessus, on obtient

$$(\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r).$$

Théorème: Soit $f : \mathbb{B}^{\mathcal{P}} \to \mathbb{B}$ une fonction booléenne avec $\mathcal{P}$ fini. Il existe une formule $H \in \mathcal{F}$ telle que $[\![H]\!] = f$.

Avant de prouver ce théorème, on démontre d'abord les deux lemme suivants et on définit lit_ρ.

Définition: Soit $\rho \in \mathbb{B}^{\mathcal{P}}$. On définit

$$\mathrm{lit}_\rho(p) = \begin{cases} p & \text{si } \rho(p) = \boldsymbol{V}\,; \\ \neg p & \text{sinon.} \end{cases}$$

Lemme:

$$\forall \rho \in \mathbb{B}^{\mathcal{P}},\ \exists G \in \mathcal{F},\ \big(\forall \rho' \in \mathbb{B}^{\mathcal{P}},\ [\![G]\!]^{\rho'} = \boldsymbol{V} \iff \rho = \rho'\big).$$

On prouve ce lemme :

Preuve:
$\mathcal{P}$ est fini. Notons donc $\mathcal{P} = \{p_1, \ldots, p_n\}$ ses variables. Soit alors $\rho \in \mathbb{B}^{\mathcal{P}}$, on définit

$$H_\rho = \bigwedge_{i=1}^{n} \mathrm{lit}_\rho(p_i).$$

Montrons que $[\![H_\rho]\!]^{\rho'} = \boldsymbol{V} \iff \rho = \rho'$. Soit $\rho' \in \mathbb{B}^{\mathcal{P}}$.

— Si $\rho = \rho'$, alors

$$\begin{aligned}
[\![H_\rho]\!]^{\rho'} &= \left[\!\!\left[\bigwedge_{i=1}^{n} \mathrm{lit}_\rho(p_i) \right]\!\!\right]^{\rho'} \\
&= \mathop{\bullet}_{i=1}^{n} [\![\mathrm{lit}_\rho(p_i)]\!]^{\rho'}
\end{aligned}$$

Soit $i \in [\![1, n]\!]$. Si $\rho(p_i) = \boldsymbol{V}$ alors $\rho'(p_i) = \boldsymbol{V}$, or, $\mathrm{lit}_\rho(p_i) = p_i$ et donc $[\![\mathrm{lit}_\rho(p_i)]\!]^{\rho'} = [\![p_i]\!]^{\rho'} = \boldsymbol{V}$; sinon si $\rho(p_i) = \boldsymbol{F}$, alors $\rho'(p_i) = \boldsymbol{F}$, or, $\mathrm{lit}_\rho(p_i) = \neg p_i$ et donc

$$[\![\mathrm{lit}_\rho(p_i)]\!]^{\rho'} = [\![\neg p_i]\!]^{\rho'} = \overline{[\![p_i]\!]^{\rho'}} = \overline{\rho'(p_i)} = \bar{\boldsymbol{F}} = \boldsymbol{V}.$$

et comme ceci étant vrai pour tout $i \in [\![1, n]\!]$, on a

$$\mathop{\bullet}_{i=1}^{n} [\![\mathrm{lit}_\rho(p_i)]\!]^{\rho'} = \boldsymbol{V}.$$

— Sinon ($\rho \neq \rho'$), soit donc $p_i \in \mathscr{P}$ tel que $\rho(p_i) \neq \rho'(p_i)$. Si $\rho(p_i) = \boldsymbol{V}$ alors $\rho'(p_i) = \boldsymbol{F}$ et donc $\mathrm{lit}_\rho(p_i) = p_i$ et $[\![\mathrm{lit}_\rho\, p_i]\!]^{\rho'} = \rho'(p_i) = \boldsymbol{F}$; sinon si $\rho(p_i) = \boldsymbol{F}$, alors $\rho'(p_i) = \boldsymbol{V}$ et donc $\mathrm{lit}_\rho(p_i) = \neg p_i$ et $[\![\mathrm{lit}_\rho(p_i)]\!]^{\rho'} = [\![\neg p_i]\!]^{\rho'} = \overline{[\![p_i]\!]^{\rho'}} = \bar{\boldsymbol{V}} = \boldsymbol{F}$.

On en déduit donc que

$$[\![H_\rho]\!]^{\rho'} = \mathop{\bullet}_{j=1}^{n} [\![\mathrm{lit}_\rho(p_j)]\!]^{\rho'} = \boldsymbol{F}$$

car il existe $i \in [\![1, n]\!]$ tel que $[\![\mathrm{lit}_\rho(p_i)]\!]^{\rho'} = \boldsymbol{F}$. □

On peut donc maintenant prouver le théorème :

Lemme: Considérons alors la formule

$$H = \bigvee_{\substack{\rho \in \mathbb{B}^{\mathscr{P}} \\ f(\rho) = \boldsymbol{V}}} H_\rho.$$

On a $[\![H]\!] = f$.

Preuve: — Soit $\rho \in \mathbb{B}^{\mathscr{P}}$ tel que $f(\rho) = \boldsymbol{V}$, on a donc

$$[\![H]\!]^\rho = \left[\!\!\left[\bigvee_{\substack{\rho' \in \mathbb{B}^{\mathscr{P}} \\ f(\rho') = \boldsymbol{V}}} H_{\rho'} \right]\!\!\right]^\rho.$$

H_ρ apparaît donc dans cette disjonction. Or, $[\![H_\rho]\!] = \boldsymbol{V}$ et donc $[\![H]\!]^\rho = \boldsymbol{V}$.
Si $f(\rho) = \boldsymbol{F}$, alors on a vu que $\forall \rho'$ tel que $f(\rho') = \boldsymbol{V}$, alors $\rho' \neq \rho$ et donc $[\![H_{\rho'}]\!]^\rho = \boldsymbol{F}$ et donc

$$\left[\!\!\left[\bigvee_{\substack{\rho' \in \mathbb{B}^{\mathscr{P}} \\ f(\rho') = \boldsymbol{V}}} H_{\rho'} \right]\!\!\right] = \boldsymbol{F}.$$

Finalement $[\![H]\!] = f$.

□

Le théorème est prouvé directement à l'aide des deux lemmes précédents.

On connaît donc la réponse à la question du nom de ce paragraphe, à savoir “peut-on représenter les fonctions booléennes par des formules ?” Oui.

0.5.2 Par des formules sous formes normales ?

Définition: On dit d'une formule de la forme

— p ou $\neg p$ avec $p \in \mathcal{P}$, que c'est un *littéral* ;

— $\bigwedge_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux que c'est une *clause conjonctive* ;

— $\bigvee_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux que c'est une *clause disjonctive* ;

— $\bigwedge_{i=1}^{n} D_i$ où les D_i qui sont des clauses disjonctives est appelée une *forme normale conjonctive* ;

— $\bigvee_{i=1}^{n} C_i$ où les C_i qui sont des clauses conjonctives est appelée une *forme normale disjonctive*.

Remarque:
On prend, comme convention, que $\bigwedge_{i=1}^{0} G_i = \top$ et $\bigvee_{i=1}^{0} G_i = \bot$.

Exemple:
La formule $\overbrace{(p \land \neg q)}^{\text{clause conjonctive}} \lor \overbrace{(r \land p)}^{\text{clause conjonctive}}$ est donc une clause normale disjonctive.

Remarque:
On écrit FMD pour une forme normale disjonctive et FNC pour une forme normale conjonctive.

Exemple:
La formule $p \land q \land \neg r$ est une clause conjonctive donc une FNC mais c'est aussi une FND.

Exemple:
La formule $\top$ est une clause conjonctive de taille 0, donc c'est une FND. Mais, c'est aussi une clause conjonctive de taille 0, donc c'est une FNC. De même, la formule $\bot$ est une FNC et une FND.

Théorème: Toute formule est équivalente à une formule sous FND et à une formule sous FNC.

Preuve:
Soit $G \in \mathcal{F}$ une formule. Soit $[\![G]\!]$ la fonction booléenne associée à G. Alors, par le théorème précédent, il existe une formule H telle que $[\![H]\!] = [\![G]\!]$ (i.e. $H \equiv G$) avec H construit dans la preuve précédente sous forme normale disjonctive. □

Exemple:
La formule $G = p \land (\neg q \lor p)$ a pour table de vérité la table suivante.

p	q	$[\![G]\!]$
F	***F***	***F***
F	***V***	***F***
V	***F***	***V***
V	***V***	***V***

Table 0.7 – Table de vérité de $p \land (\neg q \lor p)$

La forme normale disjonctive équivalente à G est $(p \land \neg q) \lor (p \land q)$.

Nous n'avons pas encore prouvé la deuxième partie du théorème mais, on essaie de trouver une formule sous FNC :

Exemple:
On reprend l'exemple de la table de vérité d'une fonction inconnue.

p	q	r	f	$\bar{f}$
F	***F***	***F***	***V***	***F***
F	***F***	***V***	***F***	***V***
F	***V***	***F***	***F***	***V***
F	***V***	***V***	***V***	***F***
V	***F***	***F***	***V***	***F***
V	***F***	***V***	***F***	***V***
V	***V***	***F***	***V***	***F***
V	***V***	***V***	***F***	***V***

TABLE 0.8 – Table de vérité d'une formule inconnue (2)

On analyse la formule $\bar{f}$ au lieu de f. Grâce à la première partie du théorème (et de la méthode pour générer cette FND), on a

$$\bar{f} = (\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge r) \vee (p \wedge q \wedge r).$$

Et, à l'aide des lois de DE MORGAN, on a

$$\bar{f} = (p \vee q \vee \neg r) \wedge (p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r),$$

ce qui est une FNC.

À l'aide de cet algorithme, on prouve facilement la 2$^{\text{nde}}$ partie du théorème.

REMARQUE:
Il est en fait possible de transformer une formule en FND en appliquant les règles suivantes à toutes les sous-formules jusqu'à obtention d'un point fixe.

— $\neg\neg H \rightsquigarrow H$;
— $\neg(G \wedge H) \rightsquigarrow G \vee H$;
— $\neg(G \vee H) \rightsquigarrow G \wedge H$;
— $(G \vee H) \wedge I \rightsquigarrow (G \wedge I) \vee (H \wedge I)$;
— $I \wedge (G \vee H) \rightsquigarrow (I \wedge G) \vee (I \wedge H)$;

— $H \wedge \top \rightsquigarrow H$;
— $\top \wedge H \rightsquigarrow H$;
— $H \vee \bot \rightsquigarrow H$;
— $\bot \vee H \rightsquigarrow H$;
— $\neg\top \rightsquigarrow \bot$;
— $\bot \wedge H \rightsquigarrow \bot$;
— $H \wedge \bot \rightsquigarrow \bot$;
— $\neg\bot \rightsquigarrow \top$;
— $\top \vee H \rightsquigarrow \top$;
— $H \vee \top \rightsquigarrow \top$.

Propriété: Soit $n \geqslant 2$ et H_n la formule $H_n = (a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \cdots \wedge (a_n \vee b_n)$ avec $\mathscr{P}_n = \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$. Alors, par application de l'algorithme précédent on obtient

$$\bigvee_{P \in \wp(\llbracket 1,n \rrbracket)} \Big(\bigwedge_{j=1}^{n} \begin{cases} a_j & \text{si } j \in P \\ b_j & \text{sinon} \end{cases} \Big).$$

À faire :

Preuve (par récurrence)*:* □

REMARQUE:
Qu'en est-il du problème SAT ? Le problème est-il simplifié pour les FND ou les FNC ?

Oui, pour les FND, le problème se simplifie. On considère, par exemple, la formule

$$\begin{array}{cc} & (\ell_{11} \wedge \ell_{12} \wedge \cdots \wedge \ell_{1,n_1}) \\ \vee & (\ell_{21} \wedge \ell_{22} \wedge \cdots \wedge \ell_{2,n_2}) \\ \vdots & \vdots \\ \vee & (\ell_{m,1} \wedge \ell_{m,2} \cdots \ell_{m,n_m}). \end{array}$$

On procède en suivant l'algorithme suivant : (À faire : Mettre l'algorithme à part) Pour i fixé, je lis la ligne i, puis je fabrique un environnement ρ.

Par exemple, pour $(p \land \lnot q \land r \land \lnot p) \lor (q \land r \land \lnot q) \lor (p \land r)$, on a $\rho = (p \mapsto \boldsymbol{V}, r \mapsto \boldsymbol{V})$.

On en conclut que Sat peut être résolu en temps linéaire dans le cas d'une forme normale disjonctive. Le problème est de construire cette FND.

Remarque:
Après s'être intéressé au problème Sat, on s'intéresse au problème Valide.

Par exemple, on considère la formule $(p \lor q \lor \lnot r \lor \lnot p) \land (p \lor \lnot r \lor p \lor r) \land (q \lor r)$. On peut construire $\rho = (q \mapsto \boldsymbol{F}, r \mapsto \boldsymbol{F})$ est tel que $[\![H]\!]^\rho = \boldsymbol{F}$.

Si on ne peut pas construire un tel environnement propositionnel, la formule vérifie le problème Valide.

On en conclut que Valide peut être résolu en temps linéaire dans le cas d'une forme normale conjonctive. Le problème est de construire cette FNC.

0.6 Algorithme de Quine

Remarque:
Une forme normale peut être vue comme un ensemble d'ensembles de littéraux (c'est la représentation que nous allons utiliser en OCamL).

Exemple:

L'ensemble $\{\{p, q\}, \{p, r\}, \varnothing\}$, a pour formule sous FNC associée $(p \lor \lnot q) \land (q \lor r) \land \bot$.

L'ensemble $\{\{p, \lnot q\}, \{q, r\}, \varnothing\}$ a pour formule sous FND associée $(p \land \lnot q) \lor (q \land r) \lor \top$.

L'ensemble $\varnothing$ a pour formule sous FNC associée $\top$.

L'ensemble $\varnothing$ a pour formule sous FND associée $\bot$.

Lemme: Pour toute formule G, pour tout variable propositionnelle et pour tout environnement propositionnel ρ, tel que $\rho(p) = \boldsymbol{V}$, alors

$$\Big[\!\!\Big[G[p \mapsto \top]\Big]\!\!\Big]^\rho = [\![G]\!]^\rho .$$

CHAPITRE

1

LANGAGES RÉGULIERS ET AUTOMATES

Sommaire

1.1 Motivation

1.1.1 1ère motivation

Il y a une grande différence entre les mathématiques et l'informatique : la gestion de l'infini. On n'a pas de mémoire infinie sur un ordinateur. Par exemple, pour représenter π ou $\sqrt{2}$, on ne peut pas stocker un nombre infini de décimales. Ce n'est pas une question de base, ces nombres ont aussi des décimales infinies dans une base 2.

Par exemple, si on ne veut utiliser $\sqrt{2}$ seulement pour plus tard le mettre au carré. On peut définir une structure en `C` comme celle qui suit

```
typedef struct {
  int carre;
  bool sign;
};
```

Code 1.1 – $\sqrt{2}$ sous forme de structure

On a pu décrire $\sqrt{2}$ comme cela car il y a une certaine régularité dans ce nombre.

On définit des relations entre ces objets. Dans ce chapitre, on va commencer par étudier autre chose : les mots. Les mots sont utilisés, tout d'abord, pour entrer une liste de lettres mais aussi le programme lui-même. En effet, il y a une liste infinie de code possibles en `C`.

1.1.2 2nde motivation

Les ordinateurs sont complexes ; il peut être dans une multitude d'états. On représente une succession de tâches (le symbole représente une tâche) :

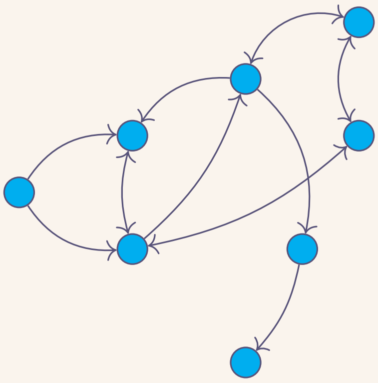


Figure 1.1 – États d'un ordinateur

Par exemple, pour une boucle infinie, on a un cycle dans le graphe ci-dessus. Ou, s'il atteint un certain nœud, on a un *bug*.

Mais, pour résoudre ce problème, on peut forcer le nombre d'état d'un ordinateur ; par exemple, dire qu'un ordinateur a 17 états.

On décide donc de représenter mathématiquement un ordinateur afin de pouvoir faire des preuves avec. Et, c'est l'objet de ce chapitre.

1.2 Mots et langages, rappels

Définition: On appelle *alphabet* un ensemble fini, d'éléments qu'on appelle *lettres*.

Définition: On appelle une *mot* sur Σ (où Σ est un alphabet) une suite finie de lettres de Σ.

La *longueur* d'un mot est le nombre de lettres, comptées avec leurs multiplicité. On la note $|w|$ pour un mot w. Si $|w| = n \in \mathbb{N}^*$, on indexe les lettres de w pour $(w_i)_{i\in[\![1,n]\!]}$ et on écrit alors

$$w = w_1 w_2 w_3 \dots w_n.$$

Il existe un unique mot de longueur 0 appelé *mot vide*, on le note ε.

Définition: Si Σ est un alphabet, on note

— Σ^n les mots de longueur n ;

— Σ^* les mots de longueurs positives ou nulle ;

— Σ^+ les mots de longueurs strictement positives.

Remarque:

$$\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n \qquad \Sigma^+ = \bigcup_{n\in\mathbb{N}^*} \Sigma^n.$$

Définition: Soit Σ un alphabet. Soit x et y deux mots de Σ^*. Notons

$$x = x_1 x_2 x_3 \dots x_n$$
$$y = y_1 y_2 y_3 \dots y_n.$$

On définit alors *concaténation* notée $x \cdot y$ l'opération définie comme

$$x \cdot y = x_1 x_2 x_3 \dots x_n y_1 y_2 \dots y_n.$$

Remarque: — $\cdot$ est une opération interne ;

— ε est neutre pour $\cdot$: $\forall n \in \Sigma^*,\ \varepsilon \cdot x = x \cdot \varepsilon = x.$

— $\cdot$ est associatif.

(On dit que c'est un monoïde.)

Remarque:
L'opération $\cdot$ n'est pas commutative.

Définition: Soit x et y deux mots sur l'alphabet Σ. On dit que

— x est un *préfixe* de y si $\exists v \in \Sigma^*,\ y = x \cdot v$;

— x est un *suffixe* de y si $\exists v \in \Sigma^*,\ y = v \cdot x$;

— x est un *facteur* de y si $\exists (v, w) \in \left(\Sigma^*\right)^2,\ y = v \cdot x \cdot w.$

Exemple: — a est facteur de aa ;

— ε est un facteur de a.

Définition: On dit que x est un *sous-mot* de y si x est une suite extraite de y. Par exemple

$$a\,\cancel{b}\,a\,a\,\cancel{b}\,a \qquad \longrightarrow \qquad a\,a\,a\,a.$$

Définition: Un *langage* est un ensemble de mots. C'est donc un élément de $\wp(\Sigma^*)$.

Remarque (⚠):
$\varnothing \neq \{\varepsilon\}$.

1.3 Langage régulier

1.3.1 Opérations sur les langages

Remarque:
Les langages sont des ensembles. On peut donc leurs appliquer des opérations ensemblistes.

Définition: Soient $L_1, L_2 \in \wp(\Sigma^*)$ deux langages. On définit la *concaténation* de deux langages, notée $L_1 \cdot L_2$:

$$L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1,\, v \in L_2\}.$$

Remarque:
- L'opération $\cdot$ (langages) a $\{\varepsilon\}$ pour neutre.
- L'opération $\cdot$ (langages) a $\varnothing$ pour élément absorbant :

$$\forall L \in \wp(\Sigma^*),\ L \cdot \varnothing = \varnothing \cdot L = \varnothing.$$

- L'opération $\cdot$ est distributive : soient K, L, M trois langages ; on a

$$K \cdot (L \cup M) = (K \cdot L) \cup (K \cdot M);$$
$$(L \cup M) \cdot K = (L \cdot K) \cup (M \cdot K).$$

Définition: Étant donné un langage L, on définit par récurrence :
- $L^0 = \{\varepsilon\}$;
- $L^{n+1} = L^n \cdot L = L \cdot L^n$.[1]

On note alors

$$L^* = \bigcup_{n\in\mathbb{N}} L^n \qquad \text{et} \qquad L^+ = \bigcup_{n\in\mathbb{N}} L^n.$$

Remarque:
On a $\Sigma^* = \Sigma^*$. On notera donc Σ^* dans tous les cas.

Remarque:
Si $\varepsilon \in L$, alors $L^* = L^+$. En effet, $L^* = L^+ \cup \{\varepsilon\}$.

Remarque:
On nomme l'application $L \mapsto L^*$ l'*étoile de* Kleene.

Remarque:
Avec L et K deux alphabets, on a

$$|L \cdot K| \leqslant |L|\,|K|.$$

En effet, avec $K = \{a, aa\}$, on a $K^2 = \{aa, aaa, aaaa\}$.

Exemple:
Avec $L = \{a, ab\}$, on a

1. La deuxième égalité est assurée par l'associativité de l'opération $\cdot$.

— $L^1 = L$;
— $L^2 = \{aa,\ aab,\ aba\,abab\}$;
— $L^* \supseteq \{\varepsilon,\ a,\ ab,\ aa, \dots, aaaaaaa\dots\}$.

Définition: Soit Σ un alphabet. On appelle *ensemble des langages réguliers*, noté LR. Le plus petit ensemble tel que

— $\varnothing \subset \mathrm{LR}$;
— Si $a \in \Sigma$, alors $a \in \mathrm{LR}$;
— Si $L_1 \in \mathrm{LR}$ et $L_2 \in \mathrm{LR}$, on a $L_1 \cup L_2 \in \mathrm{LR}$;
— Si $L_1 \in \mathrm{LR}$ et $L_2 \in \mathrm{LR}$, on a $L_1 \cdot L_2 \in \mathrm{LR}$;
— Si $L \in \mathrm{LR}$, alors $L^* \in \mathrm{LR}$.

Exemple: — Soit $\Sigma = \{t, o\}$. Est-ce que $\{toto\} \in \mathrm{LR}$? On sait déjà que $\{t\}$ et $\{o\}$ sont déjà des langages réguliers. Or,

$$\{toto\} = \{t\} \cdot \{o\} \cdot \{t\} \cdot \{o\}.$$

Et donc $\{toto\} \in \mathrm{LR}$.

— On a $\{\varepsilon\} = \varnothing^* \in \mathrm{LR}$.

— On a

$$\left\{ \begin{matrix} x_{11} & x_{12} & \dots & x_{1,n_1} \\ x_{21} & x_{22} & \dots & x_{2,n_2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,n_m} \end{matrix} \right\} \in \mathrm{LR}$$

car $\{x_{i,1}, x_{i,2}, \dots, x_{i,n_i}\} \in \mathrm{LR}$ et c'est stable par union finie.

— Avec $\Sigma = \{a, b\}$, on a

$$L = \{\varepsilon,\ a,\ aa,\ aaa, \dots\} = \underbrace{\Big(\overbrace{\{a\}}^{\in \mathrm{LR}}\Big)^*}_{\in \mathrm{LR}}.$$

— $\Sigma^* \in \mathrm{LR}$.

— Contre-exemple : $\{a^n \mid a \text{ premier }\} \notin \mathrm{LR}$.

1.3.2 Expressions régulières

On représente $\varnothing$ l'ensemble vide, _|_ l'union de deux expressions régulières, _ · _ la concaténation de deux expression régulières, et, _* l'étoile de Klein pour les expressions régulières.

On cherche à représenter informatiquement ces expressions à l'aide d'une règle de construction nommée.

Définition: Étant donné un alphabet Σ, on définit $\mathrm{Reg}(\Sigma)$ défini par induction nommée à partir des règles :

— $\varnothing\Big|^0$;
— $*\Big|^1$;
— $|\,\Big|^2$;
— $L\Big|^1_\Sigma$;
— $\cdot\Big|^2$;
— $\varepsilon\Big|^0$.

Ces règles peuvent être définies en OCamL (douteux) de la façon suivante :

```
type regex =
  | ∅
  | |  of regex * regex
  | ·  of regex * regex
  | *  of regex
  | L  of char
  | ε
```

Code 1.2 – Règles des expressions régulières en OCamL

On peut donc écrire

$$\big((\varnothing^*) \mid (a \cdot b)\big) \quad \longrightarrow \quad |(*(\varnothing()), \cdot(L(a), L(b))).$$

A-t-on $|(L(a), L(b)) \overset{?}{=} |(L(b), L(a))$? Non, sinon on risque d'avoir une boucle infinie au moment de l'évaluation de cette expression.

On peut simplifier la notation : au lieu d'écrire $|(*(\varnothing()), \cdot(L(a), L(b)))$, on note $\varnothing^* \mid (a \cdot b)$.

Définition: On définit

$$\begin{aligned} \mathscr{L} : \mathrm{Reg}(\Sigma) &\longrightarrow \wp(\Sigma^*) \\ \varnothing &\longmapsto \varnothing \\ a &\longmapsto \{a\} \\ \varepsilon &\longmapsto \{\varepsilon\} \\ e_1 \cdot e_2 &\longmapsto \mathscr{L}(e_1) \cdot \mathscr{L}(e_2) \\ e_1 \mid e_2 &\longmapsto \mathscr{L}(e_1) \cup \mathscr{L}(e_2) \\ e^* &\longmapsto \mathscr{L}(e)^* \end{aligned}$$

Exemple:
Deux expressions régulières peuvent donner le même langage : on a $\mathscr{L}(\varnothing) = \varnothing$ mais également $\mathscr{L}(\varnothing \mid \varnothing) = \mathscr{L}(\varnothing) \cup \mathscr{L}(\varnothing) = \varnothing \cup \varnothing = \varnothing$. De même, $\mathscr{L}\big(a \mid (b \cdot b^*)\big) = \{a, b, bb, bbb, \dots\} = \mathscr{L}\big((bb^*) \mid a\big)$.

Définition: On définit sur $\mathrm{Reg}(\Sigma)$ la fonction “vars” définie comme

$$\begin{aligned} \mathrm{vars} : \mathrm{Reg}(\Sigma) &\longrightarrow \wp(\Sigma) \\ \varnothing &\longmapsto \varnothing \\ \varepsilon &\longmapsto \varnothing \\ a \in \Sigma &\longmapsto \{a\} \\ e_1 \cdot e_2 &\longmapsto \mathrm{vars}(e_1) \cup \mathrm{vars}(e_2) \\ e_1 \mid e_2 &\longmapsto \mathrm{vars}(e_1) \cup \mathrm{vars}(e_2) \\ e^* &\longmapsto \mathrm{vars}(e). \end{aligned}$$

Propriété: Un langage L est régulier si et seulement s'il existe $e \in \mathrm{Reg}(\Sigma)$ telle que $\mathscr{L}(e) = L$.

Preuve: “$\Longleftarrow$” Montrons que, pour toute expression régulière $e \in \mathrm{Reg}(\Sigma)$, P_e : “le langage $\mathscr{L}(e)$ est régulier.” On procède par induction.

- $P_\varnothing : \mathscr{L}(\varnothing) = \varnothing \in \mathrm{LR}$;
- $P_\varepsilon : \mathscr{L}(\varepsilon) = \{\varepsilon\} = \varnothing^* \in \mathrm{LR}$;
- Soit $a \in \Sigma$. Montrons $P_a : \mathscr{L}(a) = \{a\} \in \mathrm{LR}$;

— Soient e_1 et e_2 deux expressions régulières telles que P_{e_1} et P_{e_2} soient vrais. Montrons $P_{e_1 \cdot e_2}$: $\mathscr{L}(e_1 \cdot e_2) = \underbrace{\mathscr{L}(e_1)}_{\in \mathrm{LR}} \cdot \underbrace{\mathscr{L}(e_2)}_{\in \mathrm{LR}} \in \mathrm{LR}$

— Soient e_1 et e_2 deux expressions régulières telles que P_{e_1} et P_{e_2} soient vrais. Montrons $P_{e_1 | e_2}$: $\mathscr{L}(e_1 \mid e_2) = \underbrace{\mathscr{L}(e_1)}_{\in \mathrm{LR}} \cup \underbrace{\mathscr{L}(e_2)}_{\in \mathrm{LR}} \in \mathrm{LR}$

“ $\Longrightarrow$ ” Montrons que, pour tout langage régulier L, il existe une expression régulière e de l'alphabet Σ telle que $\mathscr{L}(e) = L$. Soit X l'ensemble des langages L tels qu'il existent une expression régulière e de l'alphabet Σ telle que $\mathscr{L}(e) = L$. On a

$$X \supseteq \underbrace{\{\varnothing\}}_{\mathscr{L}(\varnothing)} \cup \big\{ \underbrace{\{a\}}_{\mathscr{L}(a)} \bigm| a \in \Sigma \big\}.$$

De plus, si deux langages L_1 et L_2 sont dans X, alors il existent e_1 et e_2 deux expressions régulières de Σ telle que $\mathscr{L}(e_1) = L_1$ et $\mathscr{L}(e_2) = L_2$. Or, $\mathscr{L}(e_1 \mid e_2) = \mathscr{L}(e_1) \cup \mathscr{L}(e_2) = L_1 \cup L_2$ et donc $L_1 \cup L_2$.

De même pour $L_1 \cdot L_2$.

Si un langage L est dans X, alors il existe une expression régulière e d'un alphabet Σ telle que $\mathscr{L}(e) = L$, alors $\mathscr{L}(e^*) = L^* \in X$.

X contient les langages $\varnothing$ et $\{a\}$ (avec $a \in \Sigma$) et X est stable par $\cup$, $\cdot$ et $*$. Or, LR est défini comme le plus petit ensemble vérifiant les propriétés et donc $\mathrm{LR} \subseteq X$.

$\square$

Remarque (Notation):
Les notations $\mathrm{Reg}(\Sigma)$ et $\mathrm{Regexp}(\Sigma)$ sont équivalentes.

1.4 Automates finis (sans ε-transitions)

On considère l'automate représenté par les états suivants. L'entrée est représentée par la flèche sans nœud de départ et la sortie par celle sans nœud d'arrivée.

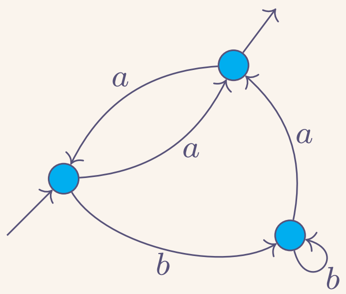


Figure 1.2 – Exemple d'automate

On représente une séquence d'état par un mot comme $aaba$ (qui correspond à une séquence valide) ou $bbab$ (qui n'est pas valide).

1.4.1 Définitions

Définition: Un *automate fini* (sans ε-transition) est un quintuplet $(Q, \Sigma, I, F, \delta)$ où

— Q est un ensemble d'états;

— Σ est son alphabet de travail;

— $I \subseteq Q$ est l'ensemble des états initiaux;

— $F \subseteq Q$ est l'ensemble des états finaux.

— $\delta \subseteq Q \times \Sigma \times Q$.

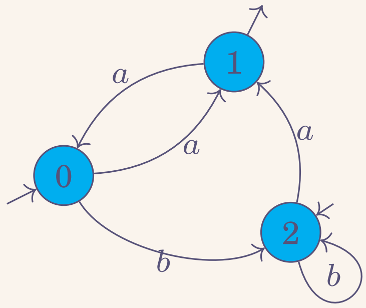


FIGURE 1.3 – Exemple d'automate (2)

EXEMPLE:
L'automate ci-dessus est donc représenté mathématiquement par

$$\Big(\underbrace{\{0,1,2\}}_{Q}, \underbrace{\{a,b\}}_{\Sigma}, \underbrace{\{0,2\}}_{I}, \underbrace{\{1\}}_{F}, \underbrace{\{(0,a,1),(0,b,2),(1,a,0),(2,a,1),(2,b,2)\}}_{\delta}\Big).$$

Définition: On dit d'une suite $(q_0, q_1, q_2, \dots, q_n) \in Q^{n+1}$ qu'elle est *une suite de transition de l'automate* $\mathscr{A} = (Q, \Sigma, I, F, \delta)$ dès lors qu'il existe $(a_1, a_2, \dots, a_n) \in \Sigma^n$ tels que, pour out $i \in [\![1, n]\!]$, $(q_{i-1}, a_i, q_i) \in \delta$. On note parfois une telle suite de transition par

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \to \cdots \to q_n - 1 \xrightarrow{a_n} q_n.$$

EXEMPLE:
$1 \xrightarrow{a} 0 \xrightarrow{b} 2 \xrightarrow{b} 2$ est une suite de transitions de l'automate ci-avant.

Définition: On dit qu'une suite $(q_0, q_1, \dots, q_n)$ est une *exécution* dans l'automate $\mathscr{A} = (Q, \Sigma, I, F, \delta)$ si c'est une suite de transition de A telle que $q_0 \in I$. On dit également qu'elle est *acceptante* si $q_n \in F$.

Lorsque $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \to \cdots \to q_{n-1} \xrightarrow{a_n} q_n$ est une suite de transitions d'un automate $\mathscr{A}$, on dit que le mot $a_1 a_2 \dots a_n$ est l'*étiquette* de cette transition.

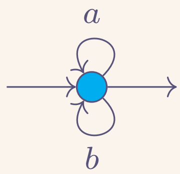
Définition (Langage reconnu par un automate)**:** On dit qu'un mot $w \in \Sigma^*$ est *reconnu* par un automate $\mathscr{A} = (Q, \Sigma, I, F, \delta)$ s'il est l'étiquette d'une exécution acceptante de $\mathscr{A}$. On note alors $\mathscr{L}(\mathscr{A})$ l'ensemble des mots reconnus par l'automate $\mathscr{A}$.

Définition: On dit d'un langage L qu'il est *reconnaissable* s'il existe un automate $\mathscr{A}$ tel que $\mathscr{L}(\mathscr{A}) = L$. On note $\mathrm{Rec}(\Sigma)$ l'ensemble des mots reconnaissables.

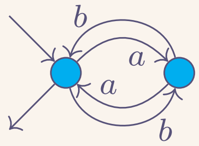
EXEMPLE:
Avec $\Sigma = \{a, b\}$, on cherche $\mathscr{A}$ tel que

— $\mathscr{L}(\mathscr{A}) = \Sigma^*$

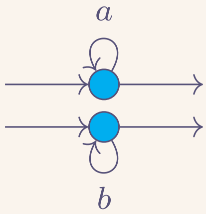
— $\mathscr{L}(\mathscr{A}) = (\Sigma^2)^*$



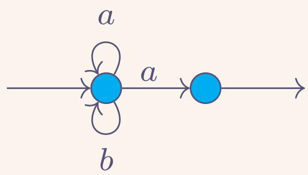
(a) Automate minimal pour $\mathscr{L}(\mathscr{A}) = \Sigma^*$



(b) Automate minimal pour $\mathscr{L}(\mathscr{A}) = (\Sigma^2)^*$



(c) Automate minimal pour $\mathscr{L}(\mathscr{A}) = \{a\}^* \cup \{b\}^*$



(d) Automate minimal pour $\mathscr{L}(\mathscr{A}) = \Sigma^* \cdot \{a\}$

FIGURE 1.4 – Automates minimaux pour différentes valeurs de $\mathscr{L}(\mathscr{A})$

Définition (Automate déterministe)**:** On dit d'un automate $\mathscr{A} = (Q, \Sigma, I, F, \delta)$ qu'il est *déterministe* si

1. $|I| = 1$;
2. $\forall (q, q_1, q_2) \in Q^3,\ \forall a \in \Sigma,\ (q, a, q_1) \in \delta$ et $(q, a, q_2) \in \delta \implies q_1 = q_2$;

REMARQUE:
(2) est équivalent à

$$\forall (q, a) \in Q \times \Sigma,\ \big|\{q' \in Q \mid (q, a, q') \in \delta\}\big| \leqslant 1.$$

Définition (Automate complet)**:** On dit d'un automate $\mathscr{A} = (Q, \Sigma, I, F, \delta)$ qu'il est *complet* si

$$\forall (q, a) \in Q \times \Sigma,\ \exists q' \in Q,\ (q, a, q') \in \delta.$$

EXEMPLE:
Les automates ci-avant sont

(a) complet et déterministe ;

(b) complet et déterministe ;

(c) non complet et non déterministe ;

(d) non complet et non déterministe.

1.4.2 Transformations en automates équivalents

On peut représenter le langage utilisé par l'automate ci-dessous avec une expression régulière : $a^* \cdot (a \mid bab)$. L'arbre ci-dessous n'est pas déterministe. On cherche à le rendre déterministe : pour cela, on trace un arbre contenant les nœuds accédés en fonctions de l'expression lue.

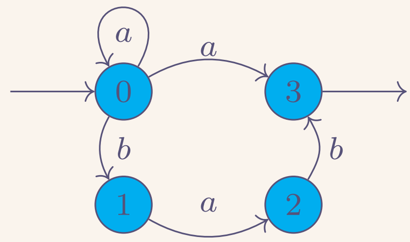


FIGURE 1.5 – Automate non déterministe ayant pour expression régulière $a^* \cdot (a \mid bab)$

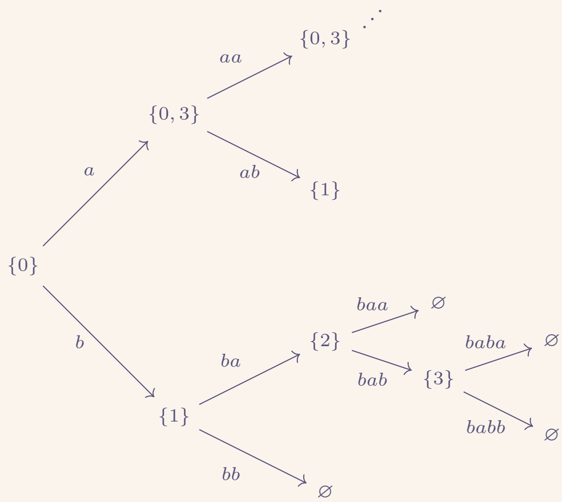


FIGURE 1.6 – Nœuds possibles par rapport à l'expression lue

À l'aide de cet arbre, on peut trouver un automate déterministe équivalent à l'automate précédent.

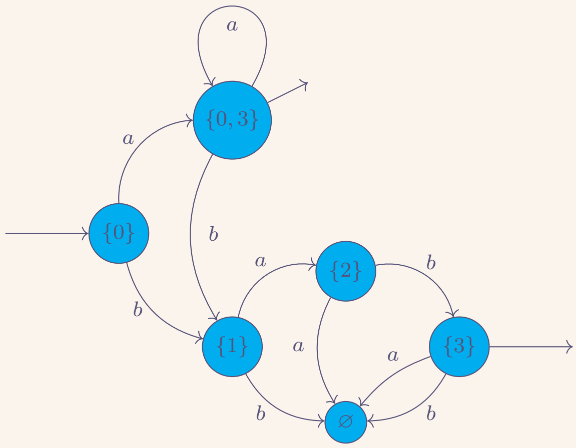


FIGURE 1.7 – Automate déterministe ayant pour expression régulière $a^* \cdot (a \mid bab)$

Définition: On dit de deux automates $\mathscr{A}$ et $\mathscr{A}'$ qu'ils sont *équivalents* si $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$.

Théorème: Pour tout automate $\mathscr{A}$, il existe un automate déterministe $\mathscr{A}'$ tel que $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$.

Preuve:
Soit $\mathscr{A} = (\mathbb{Q}, \Sigma, I, F, \delta)$ un automate. On pose $\Sigma' = \Sigma$, $\mathbb{Q}' = \wp(\mathbb{Q})$, $I' = \{I\}$, $F = \{Q \in \wp(\mathbb{Q}) \mid Q \cap F \neq \varnothing\}$ et

$$\delta' = \left\{(Q, a, Q') \in \mathbb{Q}' \times \Sigma \times \mathbb{Q}' \;\middle|\; Q' = \{q' \in \mathbb{Q} \mid \exists q \in Q,\ (q, a, q') \in \delta\}\right\}.$$

On pose alors l'automate $\mathscr{A}' = (\mathbb{Q}', \Sigma', I', F', \delta')$. Montrons que $\mathscr{L}(\mathscr{A}') = \mathscr{L}(\mathscr{A})$. On procède par double-inclusion.

"$\subseteq$" Soit $w \in \mathscr{L}(\mathscr{A})$. Il existe donc une exécution acceptante $I \ni q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n \in F$ telle que $w = w_1 w_2 \ldots w_n$. On pose $Q_0 = I$, et, pour tout entier $i \in [\![1, n]\!]$,

$$Q_i = \{q' \in \mathbb{Q} \mid \exists q \in Q_{i-1},\ (q, w_i, q') \in \delta\}.$$

Remarquons que, pour tout entier $i \in [\![1, n]\!]$, on a $(Q_{i-1}, w_i, Q_i) \in \delta'$. On a donc $I' \ni Q_0 \xrightarrow{w_1} Q_1 \to \cdots \to Q_{n-1} \xrightarrow{w_n} Q_n$ est une exécution de $\mathscr{A}'$. Montrons que, pour tout $i \in [\![0, n]\!]$, on a $q_i \in Q_i$ par récurrence finie.

— $q_0 \in I = Q_0$.

— Soit $p < n$ tel que $q_p \in Q_p$ alors q_{p+1} est tel que $(q_p, w_{p+1}, q_{p+1}) \in \delta$ et q_{p+1} est tel qu'il existe $q \in Q_p$ tel que $(q, w_{p+1}, q_{p+1}) \in \delta$. On en déduit $q_{p+1} \in Q_{p+1}$.

On a donc $q_n \in Q_n$ et $q_n \in F$ donc $Q_n \cap F \neq \varnothing$ et donc $Q_n \in F'$. L'exécution $Q_0 \xrightarrow{w_1} Q_1 \to \cdots \to Q_{n-1} \xrightarrow{w_n} Q_n$ est donc acceptante dans $\mathscr{A}'$ et donc $w = w_1 \ldots w_n \in \mathscr{L}(\mathscr{A}')$.

"$\supseteq$" Soit $w \in \mathscr{L}(\mathscr{A}')$. Soit donc $I' \ni Q_0 \xrightarrow{w_1} Q_1 \to \cdots \to Q_{n-1} \xrightarrow{w_n} Q_n \in F'$ une exécution acceptante de w dans $\mathscr{A}'$. $Q_n \cap F \neq \varnothing$. Soit donc $q_n \in Q_n \cap F$. Soit $q_{n-1} \in Q_{n-1}$ tel que $(q_{n-1}, w_n, q_n) \in \delta$ (par définition de $(Q_{n-1}, w_n, Q_n) \in \delta'$). "De proche en proche," il existe $q_0, q_1, \ldots, q_{n-2}$ tels que, pour tout $i \in [\![1, n]\!]$, $(q_{i-1}, w_i, q_i) \in \delta$ et $q_i \in Q_i$. Or, $q_0 \in Q_0 \in I' = \{I\}$ donc $Q_0 = I$ et donc $q_0 \in I$. On rappelle que $q_n \in F$. On en déduit donc que $q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n$ une exécution acceptante da ns $\mathscr{A}$ et donc $w \in \mathscr{L}(\mathscr{A}')$.

□

Pour comprendre la construction de l'automate dans la preuve, on fait un exemple. On considère l'automate non-déterministe ci-dessous.

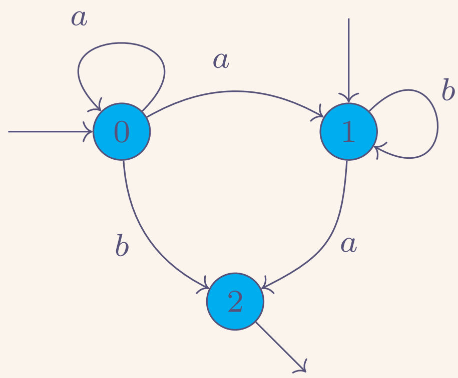


Figure 1.8 – Automate non déterministe

On construit la *table de transition* :

	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0,1\}$	$\{2\}$
$\{1\}$		

Table 1.1 – Table de transition de l'automate ci-avant

À faire : Finir la table de transition

Remarque:
L'automate $\mathscr{A}'$ construit dans le théorème précédent est complet mais son nombre de nœud suit une exponentielle.

Propriété: Soit $\mathscr{A}$ un automate fini à n états. Il existe un automate $\mathscr{A}'$ ayant $n+1$ états tel que $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$ avec $\mathscr{A}'$ complet.

Preuve:
Soit $\mathscr{A} = (\mathbb{Q}, \Sigma, I, F, \delta)$ un automate à n états. Soit $P \notin \mathbb{Q}$. On pose $\Sigma' = \Sigma$, $\mathbb{Q}' = \mathbb{Q} \cup \{P\}$, $I' = I$, $F' = F$ et

$$\delta' = \delta \cup \left\{ (q, \ell, P) \in \mathbb{Q}' \times \Sigma \times \{P\} \;\middle|\; \forall q' \in \mathbb{Q}',\ (q, \ell, q') \notin \delta \right\}.$$

Montrons que $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$. À faire : Preuve à faire

□

Définition: Soit $\mathscr{A} = (\mathbb{Q}, \Sigma, I, F, \delta)$ un automate. On dit d'un état $q \in \mathbb{Q}$ qu'il est

— *accessible* s'il existe une exécution $I \ni q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n = q$.

— *co-accessibles*'il existe une suite de transitions $q \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n \in F$.

Dans l'automate ci-dessous, l'état 0 n'est pas accessible et l'état 2 n'est pas co-accessible.

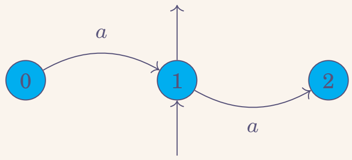


Figure 1.9 – Non-exemples d'états accessibles et co-accessibles

Définition: On dit d'un automate $\mathscr{A}$ qu'il est *émondé* dès lors que chaque état est accessible et co-accessible.

Propriété: Soit $\mathscr{A}$ un automate. Il existe $\mathscr{A}'$ un automate émondé tel que $\mathscr{L}(\mathscr{A}') = \mathscr{L}(\mathscr{A})$.

Preuve:
Soit $\mathscr{A} = (\mathbb{Q}, \Sigma, I, F, \delta)$. On pose $\Sigma' = \Sigma$, $\mathbb{Q}' = \{q \in \mathbb{Q} \mid q$ accessible ou co-accessible $\}$, $I' = I \cap R'$, $F' = F \cap R'$ et

$$\delta = \big\{(q, \ell, q') \in \mathbb{Q}' \times \Sigma \times \mathbb{Q}' \bigm| (q, \ell, q') \in \delta\big\} = (\mathbb{Q}', \Sigma, \mathbb{Q}') \cap \delta.$$

Montrons que $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$. On procède par double inclusion. On vérifie aisément que $\mathscr{L}(A') \subseteq \mathscr{L}(\mathscr{A})$. On montre maintenant $\mathscr{L}(\mathscr{A}) \subseteq \mathscr{L}(\mathscr{A}')$. Soit $w \in \mathscr{L}(\mathscr{A}')$. Soit $q_0, \dots, q_n$ tels que $q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n$ est une exécution acceptante. Or, pour tout $i \in [\![0, n]\!]$, q_i est accessible et co-accessible donc $q_i \in \mathbb{Q}'$. De plus, $q_0 \in I'$ et $q_n \in F'$. De plus, pour tout $i \in [\![0, n-1]\!]$, $(q_i, w_{i+1}, q_{i+1}) \in \delta'$. Donc, $q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n$ est une exécution acceptante de $\mathscr{A}'$. □

Parfois, on veut pouvoir "sauter" d'un état à un autre dans un automate. On utilise pour cela des ε-transitions.

1.5 Automates finis avec ε-transitions

Définition: On dit d'un automate sur l'alphabet $\Sigma \cup \{\varepsilon\}$ que c'est un automate avec ε-transition.

Exemple:
L'automate ci-dessous est un automate avec ε-transitions.

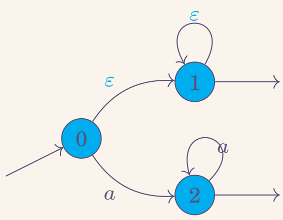


Figure 1.10 – Exemple d'automate avec ε-transition

> **Définition:** Soit $w \in \big(\Sigma \cup \{\varepsilon\}\big)^*$. On définit alors $\tilde{w}$ le mot obtenu en supprimant les occurrences de ε dans w.

Exemple:
Avec $\Sigma = \{a, b\}$ et $w = ab\varepsilon aa\varepsilon\varepsilon a$, on a $\tilde{w} = abaaa$.

On a également $\tilde{\varepsilon} = \varepsilon$; pour deux mots w_1 et w_2, on a $\widetilde{w_1 \cdot w_2} = \tilde{w}_1 \cdot \tilde{w}_2$; on a également $\tilde{a} = a$ pour $a \in \Sigma$ et $\tilde{\varepsilon} = \varepsilon$.

> **Définition:** Soit $\mathscr{A}$ un automate avec ε-transition. On pose $\tilde{\mathscr{L}}(\mathscr{A})$ est le langage de l'automate sur l'alphabet $\Sigma \cup \{\varepsilon\}$. On appelle *langage* de A, l'ensemble À faire : retrouver la formule.

Exemple:
On peut trouver un automate reconnaissant la concaténation des langages des deux automates ci-dessous.

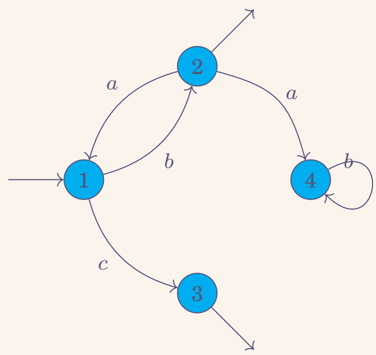


Figure 1.11 – Automate reconnaissant le langage $(ba)^* \cdot (c \mid a(ba)^*)$

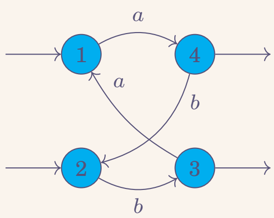


Figure 1.12 – Automate reconnaissant le langage $(a \mid baa)(bbaa)^* \mid (b \mid abb)(aabb)^*$

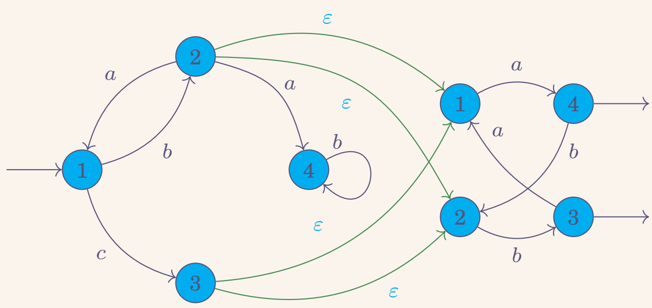


Figure 1.13 – Automate reconnaissant la concaténation des deux précédents

1.5.1 Cloture par concaténation

Propriété: Soient $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ et $\mathscr{A}' = (\Sigma', Q', I', F', \delta')$ deux automates avec $Q \cap Q' = \varnothing$. Alors $\mathscr{L}(\mathscr{A}) \cdot \mathscr{L}(\mathscr{A}')$ est un langage reconnaissable. Il est d'ailleurs reconnu par l'automate $\mathscr{A}^\cdot = (\Sigma^\cdot, Q^\cdot, I^\cdot, F^\cdot, \delta^\cdot)$ défini avec $\Sigma^\cdot = \Sigma \cup \Sigma'$, $Q^\cdot Q \cup Q'$, $I^\cdot = I$, $F^\cdot = F'$ et

$$\delta^\cdot = \{(q, \varepsilon, q) \mid q \in F,\ q' \in I'\}.$$

Preuve:
Montrons que $\mathscr{L}(\mathscr{A}^\cdot) = \mathscr{L}(\mathscr{A}) \cdot \mathscr{L}(\mathscr{A}')$. On procède par double inclusion.

"$\subseteq$" Soit $w \in \mathscr{L}(\mathscr{A}^\cdot)$ et soit

$$Q \supseteq I = I^\cdot \ni q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \to \cdots \to q_{n-1} \xrightarrow{u_n} q_n \in F^\cdot = F' \subseteq Q$$

une exécution acceptante de $\mathscr{A}^\cdot$ telle que $\tilde{u} = w$. On pose $i_0 = \min\{i \in [\![1, n]\!] \mid q_i \in Q\}$. On a alors, $\forall i \in [\![1, i_0 - 1]\!]$, $q_i \in Q$. Montrons que $\forall i \in [\![i_0, n]\!]$, $q_i \in Q'$ par récurrence finie. On a $q_{i_0} \in Q'$. De plus, si $q_i \in Q'$ et $(q_i, u_{i+1}, q_{i+1}) \in \delta^\cdot$ donc $q_{i+1} \in Q'$. Inspectons $(Q \ni q_{i_0-1}, u_{i_0}, q_{i_0} \in Q') \in \delta^\cdot$. On sait que $(q_{i_0-1}, u_{i_0}, q_{i_0}) \notin \delta$ car $q_{i_0} \in Q'$; de même, $(q_{i_0-1}, u_{i_0}, q_{i_0}) \notin \delta'$ car $q_{i_0-1} \in Q$ donc $(q_{i_0-1}, u_{i_0}, q_{i_0}) \in \{(q, \varepsilon, q') \mid q \in Fnq' \in I'\}$. On a donc que $q_{i_0-1} \in F$ et $q_{i_0} \in I$. Ainsi

$$I \ni \underbrace{q_0 \xrightarrow{u_1} q_1 \to \cdots \to \overset{\overset{F}{\rotatebox{90}{$\ni$}}}{q_i}} \xrightarrow{\varepsilon} \underbrace{\overset{\overset{I'}{\rotatebox{90}{$\ni$}}}{q_{i_0}} \xrightarrow{u_{i_0}} \to \cdots \to q_n} \in F$$

est une exécution acceptante de $\mathscr{A}$ d'étiquette $\widetilde{u_1 \dots u_{i_0-1}}$.

est une exécution acceptante de $\mathscr{A}'$ d'étiquette $\widetilde{u_{i_0+1} \dots u_n}$.

donc $\widetilde{u_{|[\![1, i_0-1]\!]}} \in \mathscr{L}(\mathscr{A})$ et $\widetilde{u_{|[\![i_0+1, n]\!]}} \in \mathscr{L}(\mathscr{A}')$.

$$\begin{aligned} w = \tilde{u} &= \widetilde{u_{|[\![1, i_0-1]\!]} \cdot u_{i_0} \cdot u_{|[\![i_0+1, n]\!]}} \\ &= \widetilde{u_{|[\![1, i_0+1]\!]}} \cdot \widetilde{u_{|[\![i_0+1, n]\!]}} \end{aligned}$$

"$\supseteq$" Montrons que $\mathscr{L}(\mathscr{A}) \cdot \mathscr{L}(\mathscr{A}') \subseteq \mathscr{L}(\mathscr{A}^*)$. Soit $w \in \mathscr{L}(\mathscr{A}) \cdot \mathscr{L}(\mathscr{A}^\cdot)$, il existe donc

$$I \ni q_0 \xrightarrow{u_1} q_1 \to \cdots \to q_n \in F$$

une exécution acceptante dans $\mathscr{A}$ d'étiquette $\widetilde{u_1 \dots u_n}$. Il existe également

$$I' \ni q_{n+1} \xrightarrow{u_{n+2}} q_{n+1} \to \cdots \to q_m \in F$$

une exécution acceptante dans $\mathscr{A}'$ d'étiquette $\widetilde{u_{n+2} \dots u_m}$. Or, $\delta \subseteq \delta^\cdot$ et $\delta' \subseteq \delta^\cdot$ donc $\forall i \in [\![1, n]\!]$, $(q_{i-1}, u_i, q_i) \in \delta^\cdot$ et $\forall i \in [\![n+2, m]\!]$, $(q_{i-1}, u_i, q_i) \in \delta^\cdot$. Or, $q_n \in F$ et $q_{n+1} \in I'$ donc $(q_n, \varepsilon, q_{n+1}) \in \delta^\cdot$. Finalement À faire : recopier.

□

1.5.2 Cloture par étoile

Propriété: Soit $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ un automate fini. Alors $\mathscr{L}(\mathscr{A})^*$ est un automate reconnaissable, il est de plus reconnu par l'automate $\mathscr{A}_* = (\Sigma_*, Q_*, I_*, F_*, \delta_*)$ défini avec $\Sigma_* = \Sigma$, $Q_* = Q \uplus \{V\}$ où $V \notin Q$. À faire : recopier ici...

Exemple:

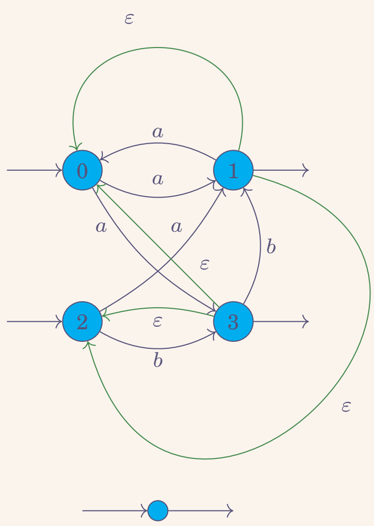


FIGURE 1.14 – Automate reconnaissant $\mathscr{L}(\mathscr{A})^*$

1.5.3 Cloture par union

Propriété: Soit $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ et $\mathscr{A}' = (\Sigma', Q', I', F', \delta')$ deux automates avec $Q \cap Q' = \varnothing$. Alors, $\mathscr{L}(\mathscr{A}) \cup \mathscr{L}(\mathscr{A}')$ est un langage reconnaissable. Il est, de plus, reconnu par $\mathscr{A}^\cup = (\Sigma^\cup, Q^\cup, I^\cup, F^\cup, \delta^\cup)$ avec $\Sigma^\cup = \Sigma \cup \Sigma'$, $Q^\cup = Q \cup Q'$, $I^\cup = I \cup I'$, $F^\cup = F \cup F'$ et $\delta^\cup = \delta \cup \delta'$.

Preuve:
Montrons que $\mathscr{L}(\mathscr{A}^\cup) \subseteq \mathscr{L}(\mathscr{A}) \cup \mathscr{L}(\mathscr{A}')$. Supposons, sans perte de généralité que les automates $\mathscr{A}$ et $\mathscr{A}'$ sont sans ε-transitions. Soit $w \in \mathscr{L}(\mathscr{A}^\cup)$. Il existe une exécution acceptante

$$I^\cup \ni q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n \in F^\cup$$

avec $w = w_0 \ldots w_n$.

Montrons que, en supposant $q_0 \in I$ sans perte de généralité, $\forall i \in [\![1, n]\!]$, $q_i \in Q$ de proche en proche.

On a donc $q_n \in Q \cap F^\cup = F$ et on a alors, pour tout $i \in [\![1, n]\!]$, $(q_{i-1}, w_i, q_i) \in \delta^\cup$. Or, $q_{i-1} \in Q$ et $(q_{i-1}, w_i, q_i) \in \delta'$ donc $(q_{i-1}, w_i, q_i) \in \delta$.

Finalement, $q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_n$ est un exécution acceptante de $\mathscr{A}$ donc $w \in \mathscr{L}(\mathscr{A})$. $\square$

REMARQUE:
Pour tout $a \in \Sigma$, $\{a\}$ est reconnaissable : par exemple,

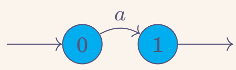


Figure 1.15 – Automate reconnaissant $\{a\}$ avec $a \in \Sigma$

Remarque:
$\varnothing$ est reconnaissable : par exemple,

Figure 1.16 – Automate reconnaissant $\varnothing$

Propriété: De ce qui précède, on en déduit que l'ensemble des langages reconnaissables par automates avec ε-transition est au moins l'ensemble des langages réguliers.

Théorème: Si $\mathscr{A}$ est un automate avec ε-transitions, alors il existe un automate $\mathscr{A}'$ sans ε-transition tel que $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$.

Exemple:

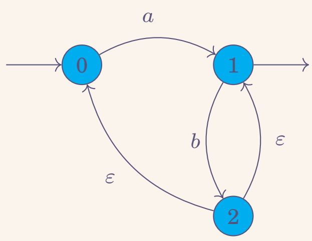


Figure 1.17 – Automate avec ε-transition

L'automate avec ε-transition ci-dessus peut être transformé en automate sans ε-transition comme celui ci-dessous.

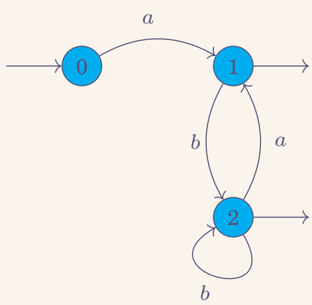


Figure 1.18 – Automate sans ε-transition

Propriété: Soit $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ un automate avec ε-transitions. Soit $q_r \in \mathbb{Q}$ un état de l'automate. Alors, l'automate $\mathscr{A}' = (\Sigma', \mathbb{Q}', I', F', \delta')$ défini par $\Sigma' = \Sigma$, $\mathbb{Q}' = \mathbb{Q}$,

$I' = I,$

$$F' = F \cup \begin{cases} \{q \in \mathbb{Q} \mid (q, \varepsilon, q_r) \in \delta\} & \text{si } q_r \in F \\ \varnothing & \text{sinon,} \end{cases}$$

$$\begin{aligned} \delta' = (&\delta \setminus \{(q, \varepsilon, q_r) \in \delta \mid q \in \mathbb{Q}\}) \\ &\cup \{(q, a, q') \mid (q, \varepsilon, q_r) \in \delta \,\text{et}(q_r, a, q') \in \delta \,\text{et}\, a \in \Sigma\} \\ &\cup \{(q, \varepsilon, q') \mid (q, \varepsilon, q_r) \in \delta \text{ et } (q_r, \varepsilon, q') \in \delta \text{ et } q_r \neq q' \in \mathbb{Q}\}, \end{aligned}$$

est tel que

— il n'y a pas d'ε-transitions entrant en q_r ;
— $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$;
— si $q \in \mathbb{Q}$ n'a pas d'ε-transition entrante dans $\mathcal{A}$, il n'en a pas dans $\mathcal{A}'$.

Algorithme 1.1 Suppression des ε-transitions

Entrée un automate $\mathcal{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$
Sortie un automate équivalent à $\mathcal{A}$ sans ε-transitions

1: $\delta' \leftarrow \delta$
2: $F' \leftarrow F$
3: $\mathbb{Q}' \leftarrow \mathbb{Q}$
4: **tant que** il existe $q \in \mathbb{Q}'$ avec une ε-transition entrante dans δ' **faire**
5: $(\Sigma', \mathbb{Q}', I', F', \delta') \leftarrow$ résultat de la proposition précédente avec $q_R = q$
6: **retourner** $(\Sigma', \mathbb{Q}', I', F', \delta')$

On a donc démontré que tout langage régulier peut être reconnu par un automate.

Exemple:
On veut, par exemple, reconnaître le langage $(a \cdot b)^* \cdot (a \mid b)$. À faire : Faire les automates

1.6 Théorème de Kleene

On s'intéresse à un autre ensemble de langages, les *langages locaux*.

1.6.1 Langages locaux

Définitions, propriétés

Définition (lettre préfixe, lettre suffixe, facteur de taille 2, non facteur)**:** Soit L un langage. On note l'ensemble $P(L)$ des lettres préfixes défini comme

$$\begin{aligned} P(L) &= \{\ell \in \Sigma \mid \exists w \in \ell, \exists v \in \Sigma^*, w = \ell \cdot v\} \\ &= \{\ell \in \Sigma, \{\ell\} \cdot \Sigma^* \cap L = \varnothing\}. \end{aligned}$$

On note l'ensemble $S(L)$ des lettres suffixes défini comme

$$S(L) = \{w_{|w|} \mid w \in L\} = \{\ell \in \Sigma \mid \Sigma^* \cdot \{\ell\} \cap L \neq \varnothing\}.$$

On note l'ensemble $F(L)$ des facteurs de taille 2 défini comme

$$F(L) = \{\ell_1 \cdot \ell_2 \in \Sigma^2 \mid \Sigma^* \cdot \{\ell_1, \ell_2\} \cdot \Sigma^* \cap L \neq \varnothing\}.$$

On note l'ensemble $N(L)$ des non-facteurs défini comme

$$N(L) = \Sigma^2 \setminus F(L).$$

On défini également l'ensemble

$$\Lambda(L) = L \cap \{\varepsilon\}.$$

Définition: Soit L un langage. On définit le langage local engendré par L comme étant

$$\rho(L) = \Lambda(L) \cup \Big(P(L) \cdot \Sigma^* \cap \Sigma^* \cdot S(L)\Big) \setminus \Sigma^* N(L) \Sigma^*.$$

Exemple:
Avec $L = \{aab, \varepsilon\}$, on a $P(L) = \{a\}$, $S(L) = \{b\}$, $F(L) = \{aa, ab\}$, $N(L) = \{ba, bb\}$, $\Lambda(L) = \{\varepsilon\}$. Et donc, on en déduit que

$$\rho(L) = \{\varepsilon\} \cup \{ab\} \cup \{aab\} \cup \cdots \{\varepsilon\} \cup \{a^n \cdot b \mid n \in \mathbb{N}^*\}.$$

Définition: Un langage est dit local s'il est son propre langage engendré i.e. $\rho(L) = L$.

Propriété: Soit L un langage. Alors, $\rho(L) \supseteq L$.

Preuve:
Soit $w \in L$. Montrons que $w \in \rho(L)$.

- Si $w = \varepsilon$, alors $\Lambda(L) = L \cap \{\varepsilon\} = \{\varepsilon\}$ donc $w \in \rho(L)$.
- Sinon, notons $w = w_1 w_2 \ldots w_n$. On doit montrer que $w_1 \in P(L)$, $w_n \in S(L)$, et $\forall i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F(L)$. Par définition de ces ensembles, c'est vrai.

□

Propriété: Soit L de la forme

$$\Lambda \cup (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N \Sigma^*)$$

avec $\Lambda \subseteq \{\varepsilon\}$, $P \subseteq \Sigma$, $S \subseteq \Sigma$, et $N \subseteq \Sigma^2$. Alors $\rho(L) = L$.

Preuve:
On a $L \subseteq \rho(L)$. Montrons donc $\rho(L) \subseteq L$.

- Montrons que $\Lambda(L) \subseteq L$.
  - Si $\Lambda(L) = \varnothing$, alors ok.
  - Sinon, $\Lambda(L) = \{\varepsilon\} = L \cap \{\varepsilon\}$ donc $\varepsilon \in L$ et donc $\varepsilon \in \Lambda$ parce que ce n'est possible.
- Montrons que $P(L) \subseteq P$. Soit $\ell \in P(L)$. Soient $v \in \Sigma^*$, et $w \in L$ tels que $w = \ell v$. On a donc $w \notin \Lambda$, donc $w \in (P\Sigma^* \cap \Sigma^* S)$ et donc $\ell v = w \in P\Sigma^*$ et donc $\ell \in P$.
- De même, $S(L) \subseteq L$
- À faire à la maison : $N \subseteq N(L)$ (ou $F(L) \subseteq F$)

□

Corollaire: On a $\rho^2 = \rho$.

À faire : Figure ensembles langages locaux, réguliers, ...

Preuve:
$\rho(\varnothing) = \varnothing$ et $\rho(\Sigma^*) = \Sigma^*$. □

REMARQUE:
Un langage L est local si et seulement s'il existe $S \subseteq \Sigma$, $P \subseteq \Sigma$, $N \subseteq \Sigma^2$ tel que

$$L \setminus \{\varepsilon\} = (P\Sigma^* \cap \Sigma^* S) \setminus \Sigma^* N \Sigma^*.$$

EXEMPLE:
Le langage $L = \{a\}$ est local avec $S = \{a\}$, $P = \{a\}$, $F = \varnothing$ et $\Lambda = \varnothing$.

Le langage $L = \{a, ab\}$ est local avec $S = \{b, a\}$, $P = \{a\}$, $F = \{ab\}$ et $\Lambda = \varnothing$.

Le langage $L = (ab)^*$ est local avec $S = \{b\}$, $P = \{a\}$, $F = \{ab, ba\}$. Soit $w \in \rho(L)$. Si $w = \varepsilon$, alors OK. Sinon, $w = abw_1$ et $w_1 \in \rho(L)$. Par récurrence, on montre que le langage est local.

Le langage $L = a \cdot (ab)^*$ n'est pas local.

Stabilité

Intersection

Propriété: Si L_1 et L_2 sont deux langages locaux, alors $L_1 \cap L_2$ est un langage local.

Preuve:
Soit $L_1 = \Lambda_1 \cup \big(P_1\Sigma^* \cap \Sigma^* S_1\big) \setminus \big(\Sigma^* N_1 \Sigma^*\big)$, et $L_2 = \Lambda_2 \cup \big(P_2\Sigma^* \cap \Sigma^* S_2\big) \setminus \big(\Sigma^* N_2 \Sigma^*\big)$. On pose $F_1 = \Sigma^2 \setminus N_1$ et $F_2 = \Sigma^2 \setminus N_2$. On pose alors $\Lambda_\cap = \Lambda_1 \cap \Lambda_2$; $P_\cap = P_1 \cap P_2$; $S_\cap = S_1 \cap S_2$; $F_\cap = F_1 \cap F_2$; $N_\cap = \Sigma^2 \setminus F_\cap$. On a

$$L_1 \cap L_2 = \Lambda_\cap \cup \big(P_\cap \Sigma^* \cap \Sigma^* S_\cap\big) \setminus \Sigma^* N_\cap \Sigma^*.$$

En effet,

$$\begin{aligned}
L_1 \cap L_2 = (&\Lambda_1 \cap \Lambda_2) \\
&\cap (\Lambda_1 \cap (P_2\Sigma^* \cap \Sigma^* S_2) \setminus \Sigma^* N_2 \Sigma^*) \\
&\cap \big(((P_1\Sigma^* \cap \Sigma^* S_1) \setminus \Sigma^* N_1 \Sigma^*) \cap \Lambda_2\big) \\
&\cap \big(((P_1\Sigma^* \cap \Sigma^* S_1) \setminus \Sigma^* N_1 \Sigma^*) \cap (P_2\Sigma^* \cap (P_2\Sigma^* \cap \Sigma^* S_2) \setminus \Sigma^* N_2 \Sigma^*\big) \\
= (&\Lambda_1 \cap \Lambda_2)\big((P_1 \cap P_2)\Sigma^* \cap \Sigma^*(S_1 \cap S_2)\big) \setminus \Sigma^*(N_1 \cap N_2)\Sigma^*
\end{aligned}$$

□

Union

CONTRE-EXEMPLE:
Avec $L_1 = ab$ et $L_2 = ba$, on a $\Lambda_1 = \Lambda_2 = \varnothing$, $P_1 = \{a\}$, $P_2 = \{b\}$, $S_1 = \{b\}$, $S_2 = \{b\}$, $F_1 = \{ab\}$ et $F_2 = \{ba\}$. Le langage $L_1 \cup L_2 = \{ab, ba\}$ n'est pas local : en effet, on a $\Lambda = \varnothing$, $P = \{a, b\}$, $S = \{a, b\}$, et $F = ab, ba$. Le mot aba est donc dans le langage local engendré.

On doit donc ajouter une contrainte afin d'éviter ce type de contre-exemples. L'intersection des alphabets est vide.

Propriété: Soient L_1 un langage local sur un alphabet Σ_1 et L_2 un langage local sur un alphabet Σ_2 avec $\Sigma_1 \cap \Sigma_2 = \varnothing$. Alors $L_1 \cup L_2$ est local.

Preuve:
Soient Λ_1, S_1, P_1, N_1, F_1 tels que L_1 soit défini par $(\Lambda_1, S_1, P_1, N_1, F_1)$. De même, soient Λ_2, S_2, P_2, N_2, F_2 tels que L_2 soit défini par $(\Lambda_2, S_2, P_2, N_2, F_2)$. Construisons alors $\Lambda_\cup = \Lambda_1 \cup \Lambda_2$, $P_\cup = P_1 \cup P_2$, $S_\cup = S_1 \cup S_2$, $F_\cup = F_1 \cup F_2$ et $N_\cup = (\Sigma_1 \cup \Sigma_2)^2 \setminus F_\cup$. On note $\Sigma = \Sigma_1 \cup \Sigma_2$. Montrons alors que

$$L_1 \cup L_2 = \underbrace{\Lambda_\cup \cup (P_\cup \Sigma^* \cap \Sigma^* S_\cup) \setminus (\Sigma^* N_\cup \Sigma^*)}_{L_\cup}.$$

On procède par double-inclusion.

"$\subseteq$" Soit $w \in L_1 \cup L_2$.

Cas 1 $w = \varepsilon$, alors $\Lambda_1 = \{\varepsilon\}$ ou $\Lambda_2 = \{\varepsilon\}$ donc $L_1 \cup L_2 = \{\varepsilon\}$ et donc $w \in L_\cup$.

Cas 2 $w \neq \varepsilon$. On pose $w = w_1 \dots w_n$. Sans perte de généralité, on suppose $w \in L_1$ et $w \notin L_2$. D'où $w_1 \in P_1$ et $w_n \in S_1$. Et, pour $i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F_1$ donc $w_1 \in P_1 \cup P_2$, $w_n \in S_1 \cup S_2$ et $\forall i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F_1 \cup F_2$. D'où $w \in L_\cup$.

"$\supseteq$" **Cas 1** $w = \varepsilon$ alors $w \in \Lambda_\cup = L_1 \cup L_2$ donc $w \in \Lambda_1$ ou $w \in \Lambda_2$ donc $w \in L_1$ ou $w \in L_2$.

Cas 2 $w \neq \varepsilon$. On pose $w = w_1 w_2 \dots w_n$ avec $w_1 \in P_\cup$, $w_n \in S_\cup$ et $\forall i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F_\cup$. Alors, sans perte de généralité, on suppose $w_1 \in \Sigma_1$. Montrons par récurrence que $\forall p \in [\![1, n]\!]$, $w_p \in \Sigma_1$.

- On sait que $w_1 \in \Sigma_1$ par hypothèse.
- On suppose que $w_p \in \Sigma_1$ avec $p < n$. Alors, $w_p w_{p+1} \in F_\cup = F_1 \cup F_2$. Or, $F_2 \subseteq (\Sigma_2)^2$ et $w_p \in \Sigma_1$ avec $\Sigma_1 \cap \Sigma_2 = \varnothing$ donc $w_{p+1} \in \Sigma_1$.

On conclut par récurrence que $\forall i \in [\![1, n]\!]$, $w_i \in \Sigma_i$. Or, $w_n \in S_1 \cup S_2$ et $S_2 \cap \Sigma_1 = \varnothing$ donc $w_n \in S_1$. De plus, pour $i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F_1 \cup F_2$ donc $w_i w_{i+1} \in F_1$ et donc $w \in L_1$.

□

Concaténation

Contre-exemple:
Avec $L_1 = \{ab\}$ et $L_2 = \{ab\}$, deux langages locaux, alors $L_1 \cdot L_2 = \{abba\}$ n'est pas local. En effet, $P = \{a\}$, $S = \{a\}$, $F = \{ab, bb, ba\}$; or $aba \notin L_1 \cdot L_2$.

Propriété: Soient L_1 un langage local sur un alphabet Σ_1 et L_2 un langage local sur un alphabet Σ_2, avec $\Sigma_1 \cap \Sigma_2 = \varnothing$. Alors $L_1 \cdot L_2$ est un langage local.

Preuve:
Soient Λ_1, S_1, P_1, N_1, F_1 définissant L_1 et soient Λ_2, S_2, P_2, N_2, F_2 définissant L_2. Construisons $\Lambda_\bullet = \Lambda_1 \cap \Lambda_2$, $P_\bullet = P_1 \cup \Lambda_1 \cdot P_2$, $S_\bullet = S_2 \cup \Lambda_2 \cdot S_1$, $F_\bullet = F_1 \cup F_2 \cup S_1 \cdot P_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$. Montrons que

$$L_1 \cdot L_2 = \underbrace{\Lambda_\bullet \cup (P_\bullet \Sigma^* \cap \Sigma^* S_\bullet) \setminus \Sigma^* N_\bullet \Sigma^*}_{L^\bullet}.$$

On procède par double inclusion.

"$\subseteq$" Soit $w \in L_1 \cdot L_2$.

- Si $w = \varepsilon$, alors $\varepsilon \in L_1$ et $\varepsilon \in L_2$ donc $w = \varepsilon \in \Lambda_\bullet \subseteq L^\bullet$.

- Sinon, $w = u \cdot v$ avec $u \in L_1$ et $v \in L_2$. On sait que $|u| > 0$ ou $|v| > 0$.
  - Si $u \neq \varepsilon$, alors $u = u_1 \dots u_p$ avec $p \geqslant 1$. On sait que $u_1 \in P_1$, $u_p \in S_1$ et, pour $i \in [\![1, p-1]\!]$, $u_1 u_{i+1} \in F_1$.

  Sous-cas 1 Si $v = \varepsilon$, alors $\Lambda_2 = \{\varepsilon\}$, et donc $S_1 \subseteq S_\bullet$. Or, $P_1 \subseteq P_\bullet$ et $F_1 \subseteq F_\bullet$. On en déduit que $w = u \in L^\bullet$.

  Sous-cas 2 $v \neq \varepsilon$, alors $v = v_1 \dots v_q$ avec $v_1 \in P_2$, $v_q \in S_2$ et, pour $i \in [\![1, nq-1]\!]$, $v_i v_{i+1} \in F_2$. Or, $u_p v_1 \in S_1 \cdot P_2$ et donc $w = u \cdot v \in L^\bullet$.
  - Si $u = \varepsilon$, on procède de la même manière.

"$\supseteq$" Soit $w \in L^\bullet$.

- Si $w = \varepsilon$, alors $\varepsilon \in \Lambda_\bullet$ et donc $\varepsilon \in L_1$ et $\varepsilon \in L_2$. D'où $\varepsilon \in L_1 \cdot L_2$.
- Sinon, on pose $w = w_1 \dots w_n$.

Sous-cas 1 Si $\{i \in [\![1, n]\!] \mid w_i \in \Sigma_2\} = \varnothing$, on a donc $\forall i \in [\![1, n]\!]$, $w_i \in \Sigma_1$. De plus, $w_1 \in P_\bullet$, $w_n \in S_\bullet$ et, pour $i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F_\bullet$. Or, $P_\bullet \cap \Sigma_1 = P_1$, $S_\bullet \cap \Sigma_1 = S_1$, $\Lambda_2 = \{\varepsilon\}$ et $\forall i \in [\![1, n-1]\!]$ $w_i w_{i+1} \in F_1$. On en déduit que $w = w_1 \dots w_n \cdot \varepsilon \in L_1 \cdot L_2$.

Sous-cas 2 Si $M = \{i \in [\![1, n]\!] \mid w_i \in \Sigma_2\} \neq \varnothing$. Soit $i_0 = \min(M)$. D'où

$$w = \underbrace{w_1 \dots w_{i_0 - 1}}_{\in \Sigma_1} \cdot \underbrace{w_{i_0}}_{\in \Sigma_2} \dots w_n.$$

Si, $\ell_1, \ell_2 \in F_\bullet$, et $\ell_1 \in \Sigma_2$, alors $\ell_2 \in \Sigma_2$. De proche en proche, on en déduit que $\forall i \in [\![i_0, n]\!]$, $w_i \in \Sigma_2$.

- Si $i_0 = 1$, alors $w_1 \in \Sigma_2$, $w_1 \in P_\bullet$, $w_1 \in P_2$, $w_1 \in P_2$, $\Lambda_1 = \{\varepsilon\}$, et $w_n \in S_\bullet \cap \Sigma_2$, et $w_n \in S_2$. De plus, pour $i \in [\![1, n-1]\!]$, $w_i w_{i+1} \in F_\bullet \cap (\Sigma_2)^2$ donc $w_i w_{i+1} \in F_2$. D'où $w = \varepsilon \cdot w_1 \dots w_n \in L_1 \cdot L_2$.
- Si $i_0 > 1$, alors $w_1 \in \Sigma_1 \cap P^\bullet = P_1$, $w_n \in \Sigma_2 \cap S^\bullet = S_2$, et $\forall i \in [\![1, i_0 - 2]\!]$, $w_i w_{i+1} \in F^\bullet \cap (\Sigma_1)^2 = F_1$. D'où $w_{i_0 - 1} w_{i_0} \in F^\bullet \cap \Sigma_1 \Sigma_2 = S_1 P_2$ donc $w_{i_0 - 1} \in S_1$ et $w_{i_0} \in P_2$ donc $w_1 \dots w_{i_0 - 1} \in L_1$. Finalement, $\forall i \in [\![i_0, n-1]\!]$, $w_i w_{i_1 \in F_\bullet \cap (\Sigma)^2}$ donc À faire : Finir la preuve.

□

Étoile

Propriété: Soit L un langage local, alors L^* est un langage local.

Preuve:
Soient Λ, P, S, F et N définissant L. Alors $\Lambda_* = \{\varepsilon\}$, $P_* = P$, $S_* = S$ et $F_* = F \cup S \cdot P$.
À faire : preuve à faire à la maison. □

Exemple:
Avec $L = \{a, b\}$, un langage local, on a $\Lambda_* = \{\varepsilon\}$, $P_* = S_* = \{a, b\}$ (car $P = \{a, b\} = S$), et $S_* = \{ab, ba, aa, bb\}$.

1.6.2 Expressions régulières linéaires

Définition: Un expression régulière est dite *linéaire* si chacune de ses lettres apparaît une fois au plus dans l'expression.

Exemple:

Oui	Non
a	aa
$a\|b$	$a\|ba$

Table 1.2 – Exemples et non-exemples d'expressions régulières linéaires

Exemple:
On définit une fonction booléenne permettant de vérifier si une expression régulière est linéaire :

$$\text{linéaire} : \left(\begin{array}{rcl} \varnothing & \mapsto & \top \\ \varepsilon & \mapsto & \top \\ a \in \Sigma & \mapsto & \top \\ e_1 \cdot e_2 & \mapsto & (\text{vars}(e_1) \cap \text{vars}(e_2) = \varnothing) \text{ et linéaire}(e_1) \text{ et linéaire}(e_2) \\ e_1 \mid e_2 & \mapsto & (\text{vars}(e_1) \cap \text{vars}(e_2) = \varnothing) \text{ et linéaire}(e_1) \text{ et linéaire}(e_2) \\ {e_1}^* & \mapsto & \text{linéaire}(e_1) \end{array}\right).$$

Propriété: Le langage d'une expression régulière est local.

Preuve:
Il nous suffit de montrer le résultat sur le cas de base.

$\mathscr{L}(\varnothing) : \varnothing$ qui correspond à $\Lambda = \varnothing$, $S = \varnothing$, $P = \varnothing$, $F = \varnothing$.

$\mathscr{L}(\varepsilon) = \{\varepsilon\}$ qui correspond à $\Lambda = \{\varepsilon\}$ et $S = P = F = \varnothing$.

$\mathscr{L}(a) = \{a\}$ qui correspond à $\Lambda = \varnothing$, $S = \{a\}$, $P = \{a\}$ et $F = \varnothing$. $\square$

Remarque:
Les grandeurs Λ, P, S et F sont de plus définies individuellement par la table suivante.

e	Λ	P	S	F
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
ε	$\{\varepsilon\}$	$\varnothing$	$\varnothing$	$\varnothing$
a	$\varnothing$	$\{a\}$	$\{a\}$	$\varnothing$
${e_1}^*$	$\{\varepsilon\}$	$P(e_1)$	$S(e_1)$	$F(e_1) \cup S(e_1) \cdot P(e_1)$
$e_1 \cdot e_2$	$\Lambda(e_1) \cap \Lambda(e_1)$	$P(e_1) \cup \Lambda(e_2) \cdot P(e_2)$	$S(e_2) \cup \Lambda(e_2) \cdot S(e_1)$	$F(e_1) \cup F(e_2) \cup S(e_1) \cdot P(e_2)$
$e_1 \mid e_2$	$\Lambda(e_1) \cup \Lambda(e_2)$	$P(e_1) \cup P(e_2)$	$S(e_1) \cup S(e_2)$	$F(e_1) \cup F(e_2)$

Table 1.3 – Construction de Λ, P, S et F dans différents cas

Remarque (Notation):
Si Σ_1 et Σ_2 sont deux alphabets et $\varphi : \Sigma_1 \to \Sigma_2$, alors on note $\tilde{\varphi}$ l'extension de φ aux mots de Σ_1^* :

$$\tilde{\varphi}(w_1 \dots w_n) = \varphi(w_1) \dots \varphi(w_n)$$

et, de plus, on note

$$\tilde{\varphi}(L) = \{\tilde{\varphi}(w) \mid w \in L\}.$$

Remarque:
On a $\tilde{\varphi}(L \cup M) = \tilde{\varphi}(L \cup M) = \tilde{\varphi}L \cup \tilde{\varphi}(M)$.

Propriété:

$$\tilde{\varphi}(L \cdot M) = \tilde{\varphi}(L) \cdot \tilde{\varphi}(M)$$

Preuve:

$$\begin{aligned} w \in \tilde{\varphi}(L \cdot M) &\iff \exists u \in L \cdot M,\ w = \tilde{\varphi}(u) \\ &\iff \exists (v,t) \in L \times M,\ w = \tilde{\varphi}(v \cdot t) \\ &\iff \exists (v,t) \in L \times M,\ w = \tilde{\varphi}(v) \cdot \tilde{\varphi}(t) \\ &\iff w \in \tilde{\varphi}(L) \cdot \tilde{\varphi}(M). \end{aligned}$$

□

Définition: Soient $e \in \mathrm{Reg}(\Sigma_1)$, $\varphi : \Sigma_1 \to \Sigma_2$. On définit alors inductivement e_φ comme étant

$$\varnothing_\varphi = \varnothing \qquad a_\varphi = \varphi(a) \text{ si } a \in \Sigma_1 \qquad (e_1 \mid e_2)_\varphi = (e_1)_\varphi \mid (e_2)_\varphi$$
$$\varepsilon_\varphi = \varepsilon \qquad (e_1 \cdot e_2)_\varphi = (e_1)_\varphi \cdot (e_2)_\varphi \qquad (e_1^*)_\varphi = ((e_1)_\varphi)^*.$$

Propriété: Si $\varphi : \Sigma_1 \to \Sigma_2$ et $e \in \mathrm{Reg}(\Sigma_1)$, alors

$$\mathscr{L}(e_\varphi) = \tilde{\varphi}(\mathscr{L}(e)).$$

Preuve (par incuction sur $e \in \mathrm{Reg}(\Sigma_1)$):cas $\varnothing$ $\mathscr{L}(\varnothing_\varphi) = \mathscr{L}(\varnothing) = \varnothing = \tilde{\varphi}(\varnothing) = \tilde{\varphi}(\mathscr{L}(\varnothing))$

cas ε $\mathscr{L}(\varepsilon_\varphi) = \mathscr{L}(\varepsilon) = \{\varepsilon\} = \tilde{\varphi}($ À faire : recopier ici

cas $e_1 \cdot e_2$ $\mathscr{L}((e_1 \cdot e_2)_\varphi) = \mathscr{L}((e_1)_\varphi \cdot (e_2)_\varphi)) = \mathscr{L}((e_1)_\varphi) \cdot \mathscr{L}((e_2)_\varphi) = \tilde{\varphi}(\mathscr{L}(e_1)) \cdot \tilde{\varphi}(\mathscr{L}(e_2)) = \tilde{\varphi}(\mathscr{L}(e_1) \cdot \mathscr{L}(e_2)) = \tilde{\varphi}(\mathscr{L}(e_1 \cdot e_2))$.

De même pour les autres cas

□

Propriété: Soit $e \in \mathrm{Reg}(\Sigma_1)$. Il existe $f \in \mathrm{Reg}(\Sigma)$ et $\varphi : \Sigma \to \Sigma_1$ tel que f est linéaire et $e = f_\varphi$.

Preuve:
Il suffit de numéroter les lettres (c.f. exemple ci-dessous). □

Exemple:
Avec $e = c^*((a \cdot a) \mid \varepsilon) \cdot ((a \mid c \mid \varepsilon)^*))^* \cdot b \cdot a \cdot a^*$, on a

$$f = c_1^*((a_1 \cdot a_2) \mid \varepsilon) \cdot (b_1((a_3 \mid c_2 \mid \varepsilon)^*))^* \cdot b_2 \cdot a_4 \cdot a_5$$

et

$$\varphi : \begin{pmatrix} a_1 & \mapsto & a \\ a_2 & \mapsto & a \\ a_3 & \mapsto & a \\ a_4 & \mapsto & a \\ a_5 & \mapsto & a \\ b_1 & \mapsto & b \\ b_2 & \mapsto & b \\ c_1 & \mapsto & c \\ c_2 & \mapsto & c \end{pmatrix}.$$

1.6.3 Automates locaux

Définition (automate local, local standard)**:** Un automate $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ est dit local dès lors que pour out $\forall(q_1, q_2, \ell, q_3, q_4) \in \mathbb{Q} \times \mathbb{Q} \times \Sigma \times \mathbb{Q} \times \mathbb{Q}$,

$$(q_1, \ell, q_3) \in \delta \quad \text{et} \quad (q_2, \ell, q_4) \in \delta \qquad \Longrightarrow \qquad q_3 = q_4.$$

L'automate $\mathscr{A}$ est dit, de plus, standard lorsque $\mathrm{Card}(I) = 1$ et qu'il n'existe pas de transitions entrante en l'unique état initial q_0.

Propriété: Un langage est local si et seulement s'il est reconnu par un automate local standard.

Exemple:

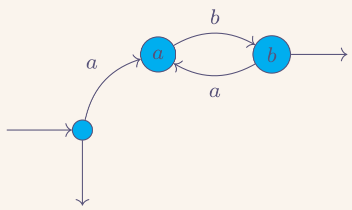


Figure 1.19 – Automate local reconnaissant le langage $(ab)^*$

Preuve: "$\Longrightarrow$" Soit L un langage local. Soit (Λ, S, P, F, N) tels que

$$L = \Lambda \cup (P\Sigma^* \cap \Sigma^*) \setminus (\Sigma^* N \Sigma^*).$$

Soit alors l'automate $\mathbb{Q} = \Sigma \cup \{\varepsilon\}$, $I = \{\varepsilon\}$, $F_{\mathscr{A}} = S \cup \Lambda$, et

$$\begin{aligned}\delta = \quad & \{(q, \ell, q') \in \mathbb{Q} \times \Sigma \times \mathbb{Q} \mid qq' \in F \text{ et } q' = \ell\} \\ & \cup \{(\varepsilon, \ell, q) \in \mathbb{Q} \times \Sigma \times \mathbb{Q} \mid \ell = q \text{ et } q \in P\}.\end{aligned}$$

On pose $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F_{\mathscr{A}}, \delta)$. Montrons que $\mathscr{L}(\mathscr{A}) = L$.

"$\subseteq$" Soit $w \in \mathscr{L}(\mathscr{A})$. Soit donc

$$q_1 \xrightarrow{w_1} q_2 \to \cdots \to q_{n-1} \xrightarrow{w_n} q_n$$

une exécution acceptante dans $\mathscr{A}$. Montrons que $w_1 \dots w_n \in L$.

Cas 1 $w = \varepsilon$ et $n = 0$. Ainsi $q_0 = q_n = \varepsilon$ et $F \cap I = \varnothing$. Or $F_{\mathscr{A}} = S \cup \Lambda$, et donc $\Lambda = \{\varepsilon\}$, d'où $\varepsilon \in L$.

Cas 2 $w \neq \varepsilon$. On sait que $w_1 \in P$; en effet, $(\varepsilon, w_1, q_1) \in \delta$ donc $w_1 = q_1 \in P$. De même, $(q_{n-1}, w_n, q_n) \in \delta$, d'où $S \ni w_n = q_n$. De plus, $\forall i \in [\![1, n-1]\!]$, $(q_{i-1}, w_i, q_i) \in \delta$ et $(q_i, w_{i+1}, q_{i+1}) \in \delta$. Ainsi, $w_i = q_i$ et $w_{i+1} = q_{i+1}$ avec $q_i q_{i+1} \in F$, d'où $w_i w_{i+1} \in F$. Donc $w \in L$.

"$\supseteq$" Soit $w = w_1 \dots w_n \in L$.

Cas 1 $w = \varepsilon$. On a $\Lambda = \{\varepsilon\}$, et donc ε est final (ou initial). On en déduit que $\varepsilon \in \mathscr{L}(\mathscr{A})$.

Cas 2 $w \neq \varepsilon$. Montrons, par récurrence finie sur $p \leqslant n$, qu'il existe une exécution

$$q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_p} q_p$$

dans $\mathscr{A}$.

— Avec $p = 1$, on a $w_1 \in P$ donc $(\varepsilon, w_1, w_1) \in \delta$. Ainsi, $\varepsilon \xrightarrow{w_1} w_1$ est une exécution dans $\mathscr{A}$.

— Supposons construit $\varepsilon \xrightarrow{w_1} q_1 \to \cdots \xrightarrow{w_p} q_p = w_p$ avec $p < n$. Or, $w_p w_{p+1} \in F$ donc $(w_p, w_{p+1}, w_{p+1}) \in \delta$. Ainsi,

$$\varepsilon \xrightarrow{w_1} q_1 \to \cdots \xrightarrow{w_p} q_p \xrightarrow{w_{p+1}} w_{p+1}$$

est une exécution acceptante de $\mathscr{A}$.

De proche en proche, on a

$$\varepsilon \xrightarrow{w_1} q_1 \to \cdots \to q_{n-1} \xrightarrow{w_n} w_n$$

une exécution dans $\mathscr{A}$. Or, $w_n \in S = F_{\mathscr{A}}$ et donc l'exécution est acceptante dans $\mathscr{A}$, et $w \in \mathscr{L}(\mathscr{A})$.

“ $\Longleftarrow$ ” Soit $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F_{\mathscr{A}}, \delta)$ un automate localement standard. Montrons que $\mathscr{L}(\mathscr{A})$ est local. Il suffit de montrer que $\rho(\mathscr{L}(\mathscr{A})) = \mathscr{L}(\mathscr{A})$. Or $\mathscr{L}(\mathscr{A}) \subseteq \rho(\mathscr{L}(\mathscr{A}))$. On montre donc $\rho(\mathscr{L}(\mathscr{A})) \subseteq \mathscr{L}(\mathscr{A})$.
Soit $w \in \rho(\mathscr{L}(\mathscr{A}))$. Ainsi,

$$w \in \Lambda(\mathscr{L}(\mathscr{A})) \cup \Big(P(\mathscr{L}(\mathscr{A}))\Sigma^* \cap \Sigma^* S(\mathscr{L}(\mathscr{A}))\Big) \setminus \Big(\Sigma^* N(\mathscr{L}(\mathscr{A}))\Sigma^*\Big).$$

Montrons que $w \in \mathscr{L}(\mathscr{A})$.

— Si $w \in \Lambda(\mathscr{L}(\mathscr{A}))$, alors $w = \varepsilon$. Or, $\Lambda(\mathscr{L}(\mathscr{A})) = \mathscr{L}(\mathscr{A}) \cap \{\varepsilon\}$. Ainsi $w \in \mathscr{L}(\mathscr{A})$.

— Sinon, $w = w_1 \dots w_n$ avec $w_1 \in P(\mathscr{L}(\mathscr{A}))$, donc il existe $u \in \Sigma^*$ tel que $w_1 \cdot u \in \mathscr{L}(\mathscr{A})$. Il existe donc une exécution acceptante

$$I \ni q_0 \xrightarrow{w_1} q_1 \overset{u}{- - \to} q_s \in F_{\mathscr{A}}.$$

Il existe donc une exécution $q_0 \xrightarrow{w_1} q_1$.
Supposons construit $q_1 \xrightarrow{w_1} q_1 \to \cdots \xrightarrow{w_p} q_p$ avec $p < n$. Or, $w_p w_{p+1} \in F(\mathscr{L}(\mathscr{A}))$, donc il existe $w \in \Sigma^*$ et $y \in \Sigma^*$ tels que $x \cdot w_p \cdot w_{p+1} \cdot y \in \mathscr{L}(\mathscr{A})$. Il existe donc une exécution acceptante

$$r_0 \overset{x}{- - \to} r_{p-1} \xrightarrow{w_p} r_p \xrightarrow{w_{p+1}} r_{p+1} \overset{y}{- - \to} r_s.$$

Or, par localité de l'automate, $q_p = r_p$. Il existe donc q_{p+1} $(= r_{p+1})$ tel que $(q_p, w_{p+1}, q_{p+1}) \in \delta$. On a donc une exécution

$$q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_p \xrightarrow{w_{p+1}} q_{p+1}.$$

De proche en proche, il existe une exécution

$$q_0 \xrightarrow{w_1} q_1 \to \cdots \to q_n.$$

Or, $w_n \in S(\mathscr{L}(\mathscr{A}))$, il existe donc $v \in \Sigma^*$ tel que $v \cdot w_n \in \mathscr{L}(\mathscr{A})$, donc il existe un exécution acceptante

$$I \ni r_0 \overset{v}{- - \to} r_{s-1} \xrightarrow{w_n} r_s \in F_{\mathscr{A}}.$$

Par localité, $r_s = q_n \in F_{\mathscr{A}}$.
Donc $\rho(\mathscr{L}(\mathscr{A})) \subseteq \mathscr{L}(\mathscr{A})$ et donc $\rho(\mathscr{L}(\mathscr{A})) = \mathscr{L}(\mathscr{A})$. On en déduit que $\mathscr{L}(\mathscr{A})$ est local.

□

Exemple:
Dans le langage local $(ab)^* \mid c^*$, on a $\Lambda = \{\varepsilon\}$, $S = \{c, b\}$, $P = \{a, c\}$ et $F = \{ab, ba, cc\}$. L'automate local reconnaissant ce langage est celui ci-dessous.

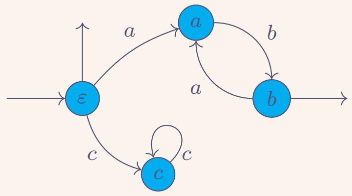


FIGURE 1.20 – Automate local reconnaissant $(ab)^* \mid c^*$

> **Propriété:** Soit $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ un automate et $\varphi : \Sigma \to \Sigma_1$. On pose
> $$\delta' = \{(a, \varphi(\ell), q') \mid (q, \ell, q') \in \delta\}.$$
> On pose $\mathscr{A}' = (\Sigma, \mathbb{Q}, I, F, \delta')$. On a $\mathscr{L}(\mathscr{A}') = \tilde{\varphi}(\mathscr{L}(\mathscr{A}))$.

EXERCICE:
Montrons que $\mathrm{LR} \subsetneq \wp(\Sigma^*)$ i.e. il existe des langages non reconnaissables.

$\mathbb{R}$ n'est pas dénombrable. On écrit un nombre réel comme une suite infinie

$$0{,}10110010101\ldots1010110\ldots$$

On pose $\Sigma = \{a\}$, on crée le langage L, associé au nombre ci-dessus comme l'ensemble contenant a, aaa, $aaaa$,...

REMARQUE (Notation):
On note A_φ, l'automate $\big(\varphi(E), \mathbb{Q}, I, F, \delta'\big)$ où $\delta' = \{(q, \varphi(\ell), q') \mid (q, \ell, q') \in \delta\}$.

1.6.4 Algorithme de Berry-Sethi : les langages réguliers sont reconnaissables

EXEMPLE:
On considère l'expression régulière $aab(a \mid b)^*$. On numérote les lettres : $a_1a_2b_1(a_3 \mid b_2)^*$, avec

$$\varphi : \begin{pmatrix} a_1 & \mapsto & a \\ a_2 & \mapsto & a \\ a_3 & \mapsto & a \\ b_1 & \mapsto & b \\ b_2 & \mapsto & b \end{pmatrix}.$$

	Λ	S	P	F
a_1	$\varnothing$	a_1	a_1	$\varnothing$
a_2	$\varnothing$	a_2	a_2	$\varnothing$
$a_1 \cdot a_2$	$\varnothing$	a_2	a_1	a_1a_2
b_1	$\varnothing$	b_1	b_1	$\varnothing$
$a_1a_2b_1$	$\varnothing$	b_1	a_1	a_1a_2, a_2b_1
a_3	$\varnothing$	a_3	a_3	$\varnothing$
b_2	$\varnothing$	b_2	b_2	$\varnothing$
$a_3 \mid b_2$	$\varnothing$	a_3, b_2	a_3, b_2	$\varnothing$
$(a_3 \mid b_2)^*$	ε	a_3, b_2	a_3, b_2	$a_3b_2, b_2a_3, a_3a_3, b_2b_2$
$a_1a_2b_1(a_3 \mid b_2)^*$	$\varnothing$	a_3, b_2ab_1	a_1	$a_3b_2, b_2a_3, a_3a_3, b_2b_2, a_1a_2, a_2b_1, b_1a_3, b_1b_2$

TABLE 1.4 – Λ, S, P et F pour les différents mots reconnus

On crée donc l'automate ci-dessous.

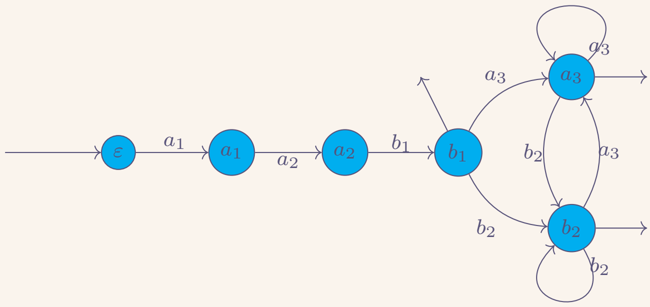


Figure 1.21 – Automate déduit de la table 1.4

On applique la fonction φ a tous les états et transitions pour obtenir l'automate ci-dessous. Cet algorithme reconnaît le langage $aab(a \mid b)^*$.

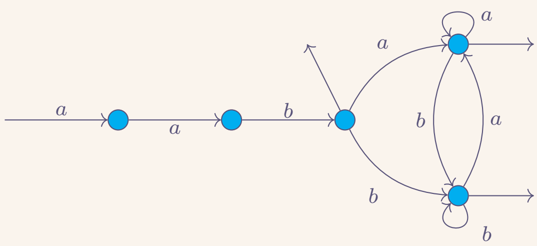


Figure 1.22 – Application de φ à l'automate de la figure 1.21

> **Théorème:** Tout langage régulier est reconnaissable. De plus, on a un algorithme qui calcule un automate le reconnaissant, à partir de sa représentation sous forme d'expression régulière.

Algorithme (Berry-Sethi)**:** Entrée : Une expression régulière e
Sortie : Un automate reconnaissant $\mathscr{L}(e)$

1. On linéarise e en f avec une fonction φ telle que $f_\varphi = e$.
2. On calcule inductivement $\Lambda(f)$, $S(f)$, $P(f)$, et $F(f)$.
3. On fabrique $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ un automate reconnaissant $\mathscr{L}(f)$.
4. On retourne $\mathscr{A}_\varphi$.

À faire : refaire la mise en page pour les algorithmes

1.6.5 Les langages reconnaissables sont réguliers

On fait le « sens inverse » : à partir d'un automate, comment en déduire le langage reconnu par cet automate ?

L'idée est de supprimer les états un à un. Premièrement, on rassemble les états initiaux en les reliant à un état i, et de même, on relie les états finaux à f. Pour une suite d'états, on concatène les lettres reconnus sur chaque transition :

Figure 1.23 – Succession d'états

De même, lors de « branches » en parallèles, on les concatène avec un $|$. En appliquant cet algorithme à l'automate précédent, on a

$$(aab) \cdot \Big((\varepsilon \mid aa^*) \mid (b \mid aa^*b) \cdot (b \mid aa^*b)^* (aa^* \mid \varepsilon) \Big).$$

Définition: Un automate généralisé est un quintuplet $(\Sigma, \mathbb{Q}, I, F, \delta)$ où
- Σ est un alphabet ;
- $\mathbb{Q}$ est un ensemble fini ;
- $I \subseteq \mathbb{Q}$;
- $F \subseteq \mathbb{Q}$;
- $\delta \subseteq \mathbb{Q} \times \mathrm{Reg}(\Sigma) \times \mathbb{Q}$, avec

$$\forall r \in \mathrm{Reg}(\Sigma),\ \forall (q, q') \in \mathbb{Q}^2,\ \mathrm{Card}(\{(q, r, q') \in \delta\}) \leqslant 1.$$

Définition (Langage reconnu par un automate généralisé)**:** Soit $(\Sigma, \mathbb{Q}, I, F, \delta)$ un automate généralisé. On dit qu'un mot w est reconnu par l'automate s'il existe une suite

$$q_0 \xrightarrow{r_1} q_1 \xrightarrow{r_2} q_2 \to \cdots \to q_{n-1} \xrightarrow{r_n} q_n$$

et $(u_i)_{i \in [\![1,n]\!]}$ tels que $\forall i \in [\![1, n]\!]$, $u_i \in \mathscr{L}(r_i)$ et $w = u_1 \cdot u_2 \cdot \ldots \cdot u_n$.

Définition: Un automate généralisé $(\Sigma, \mathbb{Q}, I, F, \delta)$ est dit « bien détouré [2] » si $I = \{i\}$ et $F = \{f\}$, avec $i \neq f$, tels que i n'a pas de transitions entrantes et f n'a pas de transitions sortantes.

Lemme: Tout automate généralisé est équivalent à un automate généralisé « bien détouré. » En effet, soit $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ un automate généralisé. Soit $i \notin \mathbb{Q}$ et $f \notin \mathbb{Q}$. On pose $\Sigma' = \Sigma$, $I' = \{i\}$, $F' = \{f\}$, $\mathbb{Q}' = \mathbb{Q} \cup \{i, f\}$ et

$$\delta' = \delta \cup \{(i, \varepsilon, q) \mid q \in I\} \cup \{(q, \varepsilon, f) \mid q \in F\}.$$

Alors, l'automate $\mathscr{A}' = (\Sigma', \mathbb{Q}', I', F', \delta')$ est équivalent à $\mathscr{A}$ et « bien détouré. »

Lemme: Soit $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ un automate généralisé « bien détouré » tel que $|\mathbb{Q}| \geqslant 3$. Alors il existe un automate généralisé « bien détouré » $\mathscr{A}' = (\Sigma, \mathbb{Q}', I, F, \delta')$ avec $\mathbb{Q}' \subsetneq \mathbb{Q}$ et $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$.

Preuve:
Étant donné qu'il existe au plus une transition entre chaque pair d'état $(q, q') \in \mathbb{Q}^2$, il est possible de le représenter au moyen d'une fonction de transition

$$T : \mathbb{Q} \times \mathbb{Q} \longrightarrow \mathrm{Reg}(\Sigma).$$

2. Cette notation n'est pas officielle.

À faire : Recopier la def de T Soit $q \in \mathbb{Q} \setminus \{i, f\}$. Soit alors T' défini, pour $(q_a, q_b) \in \mathbb{Q} \setminus \{q\}$, par

$$T'(q_a, q_b) = T(q_a, q_b) \mid T(q_a, q) \cdot T(q, q)^* \cdot T(q, q_b).$$

On considère l'automate $\mathbb{Q}' = \mathbb{Q} \setminus \{q\}$ et δ' construit à partir de T'. □

Exemple:
On considère l'automate ci-dessous.

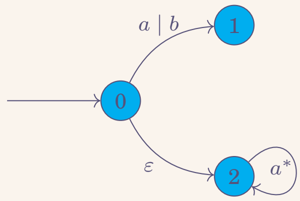


Figure 1.24 – Automate exemple

La fonction T peut être représentée dans la table ci-dessous.

	0	1	2
0	$\varnothing$	$a \mid b$	ε
1	$\varnothing$	$\varnothing$	$\varnothing$
2	$\varnothing$	$\varnothing$	a^*

Table 1.5 – Fonction T équivalente à l'automate de la figure 1.24

Exemple:
On applique l'algorithme à l'automate suivant.

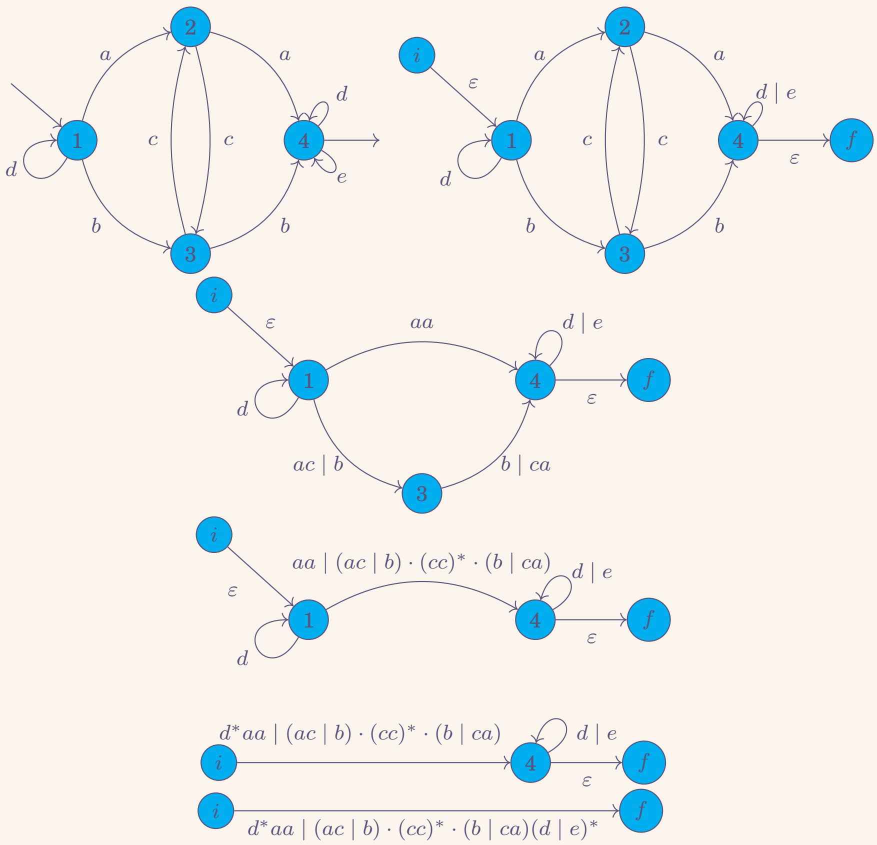


Figure 1.25 – Application de l'algorithme à un exemple

On a donc que le langage de l'automate initial est

$$\mathscr{L}(d^*(aa) \mid (ac \mid b)(cc)^*(b \mid ca)(d \mid e)^*).$$

Théorème: Un langage reconnaissable est régulier.

Preuve:
On itère le lemme précédent depuis un automate généralisé $\mathscr{A}$ jusqu'à obtention d'un automate comme celui ci-dessous.

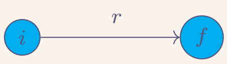


Figure 1.26 – Automate résultat de l'application du lemme

On a alors $\mathscr{L}(\mathscr{A}) = \mathscr{L}(r)$. □

Théorème (Kleene)**:** Un langage est régulier si et seulement s'il est reconnaissable. Et, on a donné un algorithme effectuant ce calcul dans les deux sens.

1.7 La classe des langages réguliers

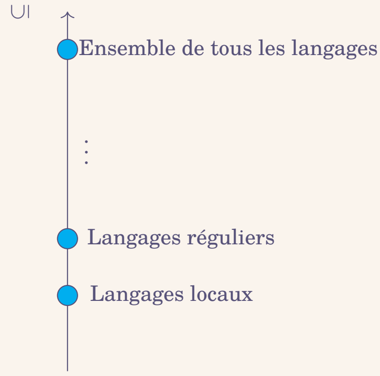


FIGURE 1.27 – Ensembles de langages

Propriété: La classe des langages réguliers/reconnaissables est stable par passage au complémentaire.

Preuve:
Soit $L \in \mathrm{LR}$. Soit $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ un automate reconnaissant le langage L. Soit $\mathscr{A}' = (\Sigma, \mathbb{Q}', I', F', \delta')$ un automate déterministe et complet équivalent à $\mathscr{A}$. Soit $\mathscr{A}'' = (\Sigma, \mathbb{Q}', I', \mathbb{Q}' \setminus F', \delta')$. Alors (à prouver à la maison) $\mathscr{L}(\mathscr{A}'') = \Sigma^* \setminus \mathscr{L}(\mathscr{A}) = \Sigma^* \setminus L$ et donc $\Sigma^* \setminus L$ est reconnaissable/régulier. $\square$

Corollaire: On a la stabilité par intersection. En effet,

$$L \cap L' = \left(L^{\mathrm{c}} \cup (L')^{\mathrm{c}}\right)^{\mathrm{c}}$$

où L^{c} est le complémentaire de L.

Corollaire: Si L et L' sont deux langages réguliers (quelconques), alors $L \setminus L'$ est un langage régulier. En effet,

$$L \setminus L' = L \cap (L')^{\mathrm{c}}.$$

Corollaire: Si L et L' sont deux langages réguliers. Alors $L \bigtriangleup L'$[3] est un langage régulier. En effet

$$L \bigtriangleup L' \overset{(\mathrm{def})}{=} (L \cup L') \setminus (L \cap L').$$

1.7.1 Limite de la classe/Lemme de l'étoile

Théorème (Lemme de l'étoile)**:** Soit L un langage reconnu par un automate à n états. Pour tout mot $u \in L$ de longueur supérieure ou égale à n, il existe trois mots x, y et z

3. $\bigtriangleup$ est la différence symétrique

tels que
$$u = x\cdot y\cdot z, \qquad |x\cdot y| \leqslant n, \qquad y \neq \varepsilon, \qquad \text{et} \qquad \forall p \in \mathbb{N},\ x\cdot y^p\cdot z \in L.$$

Preuve:
Soit L un langage reconnu par un automate $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ à n états. Soit u un mot d'un alphabet Σ de longueur supérieure ou égale à n ($u \in \Sigma^{\geqslant n}$) tel que $u \in L$. Alors, il existe un exécution acceptante
$$q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \to \cdots \to q_{m-1} \xrightarrow{u_m} q_m$$
avec $m \geqslant n$. Par principe des tiroirs, l'ensemble $\{(i,j) \in [\![0,m]\!]^2 \mid i < j \text{ et } q_i = q_j\}$ est non vide. Et donc $A = \{j \in [\![0,m]\!] \mid \exists i \in [\![0, j-1]\!],\ q_i = q_j\}$ est non vide. Soit alors $j_0 = \min A$ bien défini. Alors, par définition de A, il existe $i_0 \in [\![0, j_0 - 1]\!]$ tel que $q_{i_0} = q_{j_0}$ et $j_0 \leqslant n$. On pose donc
$$\underbrace{q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} \cdots \xrightarrow{u_{i_0}}}_{x} q_{i_0} \underbrace{\xrightarrow{u_{i_0+1}} q_{i_0+1} \to \cdots \xrightarrow{u_{j_0}}}_{y} q_{j_0} \underbrace{\xrightarrow{u_{j_0+1}} q_{j_0+1} \to \cdots \xrightarrow{u_m}}_{z} q_m \ :$$
$x = u_1u_2\dots u_{i_0}$, $y = u_{i_0+1}\dots u_{j_0}$ et $z_{j_0+1}\dots u_m$. On a donc $y \neq \varepsilon$: en effet $i_0 < j_0$. Également, on a $|x\cdot y| = j_0 \leqslant n$ et $u = x\cdot y \cdot z$. Montrons alors que $\forall p \in \mathbb{N},\ x\cdot y^p \cdot z \in L$. La suite de transitions
$$q_0 \xrightarrow{u_1} q_1 \to \cdots \xrightarrow{u_{i_0}} q_{i_0} \xrightarrow{u_{j_0+1}} q_{j_0+1} \to \cdots \xrightarrow{u_m} q_m$$
est une exécution acceptante donc $x\cdot z \in L$. De proche en proche, on en déduit que $x\cdot y^p\cdot z \in L$ pour tout $p \in \mathbb{N}$. □

Corollaire: Il y a des langages non réguliers/reconnaissables.

Preuve:
Soit $L = \{a^n \cdot b^n \mid n \in \mathbb{N}\}$. Montrons que L n'est pas régulier par l'absurde. Supposons L reconnaissable par un automate $\mathscr{A}$ à n états, et soit $u = a^n\cdot b^n$. Alors $|u| \geqslant n$. D'où, d'après le lemme de l'étoile, il existe un triplet $(x,y,z) \in (\Sigma^*)^3$ tel que $y \neq \varepsilon$, $u = x\cdot y\cdot z$, $|x\cdot y| \leqslant n$ et $x\cdot y^* \cdot z \subseteq L$ $(*)$. Il existe donc $p \in [\![1,n]\!]$ tel que $y = a^p$. De même, il existe $q \in [\![0, n-p]\!]$ tel que $x = a^q$ et $z = a^{n-p-q}\cdot b^n$. Donc, d'après $(*)$, $x\cdot y\cdot y\cdot z \in L$ et donc $a^q\cdot a^p \cdot a^p \cdot a^{n-p-q}\cdot b^n \in L$, d'où $a^{n+p}\cdot b^n \in L$. Or, comme $p \neq 0$, $n+p \neq n$: une contradiction. □

Exercice:
On considère le langage $L_2 = \{w \in \Sigma^* \mid |w|_a = |w|_b\}$. *Le langage L_2 est-il régulier ?* La même démonstration fonction en remplaçant L par L_2. Mais, nous allons procéder autrement, par l'absurde : on suppose L_2 régulier. Or, on sait que, d'après la preuve précédente, $L = L_2 \cap a^*\cdot b^*$, et $a^*\cdot b^*$ est régulier. D'où L régulier, ce qui est absurde.

Exercice:
On considère le langage $L = \{w \in \Sigma^* \mid |w|_a \equiv |w|_b\ [3]\}$. *Le langage L est-il régulier ?* Oui, l'automate de la figure suivante reconnait le langage L (les états représentent la différence $|w|_a - |w|_b \bmod 3$).

Montrons à présent qu'un automate à moins de trois états n'est pas possible : si $\delta^*(i_0, a^x) = \delta^*(i_0, a^y)$ avec $[\![0,2]\!] \ni x < y \in [\![0,2]\!]$, alors pour tout $z \in \mathbb{N}$, $\delta^*(i_0, a^{x+z}) = \delta^*(i_0, a^{y+z})$. On pose $z = 3 - y$. Alors
$$\underbrace{\delta^*(i_0, a^{x+3-y})}_{\not\in F} = \underbrace{\delta^*(i_0, a^3)}_{\in F}.$$

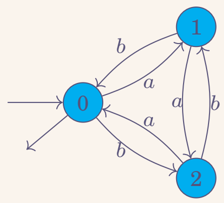


FIGURE 1.28 – Automate reconnaissant le langage $\{w \in \Sigma^* \mid |w|_a \equiv |w|_b \ [3]\}$

EXERCICE:
Soit $\Sigma = \{0, 1, `(', `)', `\{', `\}', `,'\}$. On écrit en OCaml la fonction `to_string` définie telle que si (`affiche` $\mathscr{A}$) et (`affiche` $\mathscr{A}'$) donnent le même affichage, alors $\mathscr{A} = \mathscr{A}'$.

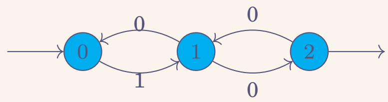


FIGURE 1.29 – Codage d'un automate par une chaîne de caractères

Par exemple, on représente l'automate ci-dessus par

$$\text{“}(\{0, 1, 10\}, \{0\}, \{10\}, \{(0, 0, 1), (1, 0, 10), (10, 0, 1), (1, 1, 0)\}).\text{”}$$

```
let affiche (Q,I,F,δ) =
```

CODE 1.3 – Fonction `affiche` affichant un automate

À faire : Recopier le code

EXERCICE:
Supposons que tout langage est reconnaissable. Soit $L = \{w \in \Sigma^* \mid \exists \mathscr{A},\ w \leftarrow$ `affiche` $\mathscr{A}$ et $w \notin \mathscr{L}(\mathscr{A})\}$. Soit B un automate tel que $L = \mathscr{L}(B)$. Soit $w \in$ `affiche` B. Si $w \in L$, alors il existe un automate tel que $w =$ `affiche` $\mathscr{A}$ et $w \notin \mathscr{L}(\mathscr{A})$. D'où $\mathscr{A} = B$ par injectivité et donc $w \notin \mathscr{L}(B) = L$, ce qui est absurde. Sinon, si $w \notin L$, alors $w =$ `affiche` B avec $w \notin \mathscr{L}(B)$ et $w \in L$, ce qui est absurde.

Annexe 1.A Comment prouver la correction d'un programme ?

Avec $\Sigma = \{a, b\}$. Comment montrer qu'un mot a au moins un a et un nombre pair de b.

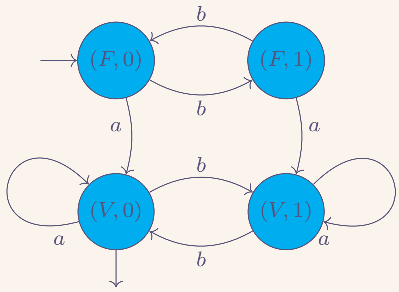


Figure 1.30 – Automate reconnaissant les mots valides

On veut montrer que

P_w : « $\forall w \in \Sigma^*, \forall q \in \mathbb{Q}$, (il existe une exécution par w menant à q) $\iff$ w satisfait I_q »

où

$$I_{(\underset{\mathbb{B}}{\underset{\cap}{v}},\ \underset{\{0,1\}}{\underset{\cap}{r}})} \quad : \quad (|w|_a \geqslant 1 \iff v) \text{ et } (r = |w|_b \bmod 2).$$

On le montre par récurrence sur la longueur de w :

“ $\Longrightarrow$ ”

— Pour $w = \varepsilon$, alors montrons que $\forall q \in \mathbb{Q}$, il existe un exécution menant à q étiquetée par w (noté $\xrightarrow[\mathscr{A}]{w} q$) si et seulement si w satisfait I_q.

  — $\xrightarrow[\mathscr{A}]{\varepsilon} (\boldsymbol{F}, 0)$ est vrai, de plus ε satisfait $I_{(\boldsymbol{F},0)}$;

  — sinon si $q \neq (\boldsymbol{F}, 0)$, alors $\xrightarrow[\mathscr{A}]{\varepsilon} q$ est fausse, de plus ε ne satisfait pas I_q.

— Supposons maintenant P_w vrai pour tout mot w de taille n. Soit $w = w_1 \dots w_n w_{n+1}$ un mot de taille $n+1$. Notons $\underline{w} = w_1 \dots w_n$. Montrons que P_w est vrai. Soit $q \in \mathbb{Q}$. Supposons $\xrightarrow[\mathscr{A}]{w} q$.

  — Si $q = (\boldsymbol{F}, 0)$ et $w_{n+1} = b$. On a donc $\xrightarrow[\mathscr{A}]{\underline{w}} (\boldsymbol{F}, 1)$, et, par hypothèse de récurrence, $\underline{w}$ satisfait. Donc $|\underline{w}|_a = 0$ et $|\underline{w}|_b \equiv 1\ [2]$ donc $|w|_a = 0$ et $|w|_b \equiv 0\ [2]$ donc w satisfait $I_{(\boldsymbol{F},0)}$.

  — De même pour les autres cas.

“ $\Longleftarrow$ ” Réciproquement, supposons que w satisfait I_q.

— Si $w = (\boldsymbol{V}, 0)$ et $w_{n+1} = a$. Alors,

  — si $|\underline{w}|_a = 0$, alors $\underline{w}$ satisfait $I_{(\boldsymbol{F},0)}$. Par hypothèse de récurrence, on a donc $\xrightarrow[\mathscr{A}]{\underline{w}} (\boldsymbol{F}, 0)$ et donc $\xrightarrow[\mathscr{A}]{w} (\boldsymbol{V}, 0)$.

  — si $|\underline{w}|_b \geqslant 1$, alors $\underline{w}$ satisfait $I_{(\boldsymbol{V},0)}$ donc $\xrightarrow[\mathscr{A}]{\underline{w}} (\boldsymbol{V}, 0)$ et donc $\xrightarrow[\mathscr{A}]{w} (\boldsymbol{V}, 0)$.

— De même pour les autres cas.

On a donc bien

$$\forall w \in \Sigma^*, \forall q \in \mathbb{Q}, \xrightarrow[\mathscr{A}]{w} q \iff w \text{ satisfait } I_q.$$

Finalement,

$$\begin{aligned}\mathscr{L}(\mathscr{A}) &= \{w \in \Sigma^* \mid \exists f \in F,\ \xrightarrow[\mathscr{A}]{w} f\} \\ &= \{w \in \Sigma^* \mid \xrightarrow[\mathscr{A}]{w} (\boldsymbol{V}, 0)\} \\ &= \{w \in \Sigma^* \mid w \text{ satisfait } I_{(\boldsymbol{V},0)}\} \\ &= \{w \in \Sigma^* \mid |w|_a \geqslant 1 \text{ et } |w|_b \equiv 0 \ \ [2]\}\end{aligned}$$

Annexe 1.B Hors-programme

Définition: On appelle monoïde un ensemble M muni d'une loi "·" interne associative admettant un élément neutre 1_M.

Définition: Étant donné deux monoïdes M et N, on appelle morphisme de monoïdes une fonction $\mu : M \to N$ telle que

1. $\mu(1_M) = 1_N$;
2. $\mu(x \cdot_M y) = \mu(x) \cdot_N \mu(y)$.

Exemple:
$|\cdot| : (\Sigma^*, \cdot) \to (\mathbb{N}, +)$ est un morphisme de monoïdes.

Définition: Un langage L est dit reconnu par un monoïde M, un morphisme $\mu : \Sigma^* \to M$ et un ensemble $P \subseteq M$ si $L = \mu^{-1}(P)$.

Exemple:
L'ensemble $\{a^{n^3} \mid n \in \mathbb{N}\}$ est reconnu par le morphisme $|\cdot|$ et l'ensemble $P = \{n^3 \mid n \in \mathbb{N}\}$.

Théorème: Un langage est régulier si et seulement s'il est reconnu par un monoïde fini.

Exemple:
L'ensemble $\{a^{2n} \mid n \in \mathbb{N}\}$ est un langage régulier. En effet, on a $M = \mathbb{Z}/2\mathbb{Z}$, $P = \{0\}$ et

$$\begin{aligned}\mu : \Sigma^* &\longrightarrow \mathbb{Z}/2\mathbb{Z} \\ w &\longmapsto |w| \bmod 2.\end{aligned}$$

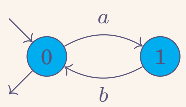


Figure 1.31 – Automate reconnaissant $\mu^{-1}(P) = L$

Preuve: "$\Longrightarrow$" Soit $L \in \wp(\Sigma^*)$ reconnu par un monoïde M fini, un morphisme μ et un ensemble $P : L = \mu^{-1}(P)$. Posons $\mathscr{A} = (\Sigma', \mathbb{Q}, I, F, \delta)$ avec

$$\Sigma' = \Sigma \qquad \mathbb{Q} = M \qquad I = \{1_M\} \qquad F = P$$
$$\delta = \{(q, \ell, q') \in \mathbb{Q} \times \Sigma \times \mathbb{Q} \mid q \cdot \mu(\ell) = q'\}.$$

Montrons que $\mathscr{L}(\mathscr{A}) = L$. Soit $w \in \mathscr{L}(\mathscr{A})$. Il existe une exécution acceptante

$$1_M = q_0 \xrightarrow{w_1} q_1 \to \cdots \xrightarrow{w_n} q_n \in P.$$

Or, $\mu(w_1 \dots w_n) = \prod_{i=1}^n \mu(w_i) = q_0 \prod_{i=1}^n \mu(w_i) = q_0\mu(w_1) \cdot \prod_{i=1}^n \mu(w_i) = q_1 \prod_{i=1}^n \mu(w_i) = q_n \in P$.

$\square$

CHAPITRE

2

ALGORITHMES PROBABILISTES

Sommaire

2.1 Introduction

Définition: Un algorithme *déterministe* est un algorithme tel que pour chaque entrée I de l'algorithme, l'exécution de l'algorithme produit toujours exactement la même suite d'états.

Remarque:
Un algorithme déterministe produit donc toujours les même sorties sur les mêmes entrées.

Définition: Un algorithme *probabiliste* est un algorithme opérant sur un ensemble $\mathscr{E}$, tel que la suite d'états obtenus par exécution de l'algorithme sur une entrée $e \in \mathscr{E}$ est une variable aléatoire.

Remarque:
Avec cette définition, un algorithme déterministe est un algorithme probabiliste.

Exemple:
On considère le problème suivant :

$$\text{Problème Tri} : \begin{cases} \text{Entrée} & : \text{ un tableau } T \text{ de taille } n \\ \text{Sortie} & : T \text{ trié.} \end{cases}$$

Une réponse à ce problème est l'algorithme nommé Bozosort décrit ci-dessous. On le nomme aussi « tri aléatoire. »

Algorithme 2.2 BOZOSORT

Entrée T un tableau

tant que T non trié **faire**
- $i \leftarrow \mathcal{U}(\llbracket 1, n-1 \rrbracket)$
- $j \leftarrow \mathcal{U}(\llbracket 1, n-1 \rrbracket)$
- Échanger i et j dans le tableau T

On étudie l'algorithme ci-dessus : il est trivialement partiellement correct (i.e. s'il est correct). En effet, par négation de la condition de boucle, on a T trié.

Le temps d'exécution de l'algorithme est difficile à estimer. L'algorithme peut ne pas terminer.

EXEMPLE:
On considère à présent le problème ci-dessous : approximer π. L'algorithme tire des points au hasard dans un carré unité, et regarde si le point est dans le disque unité.

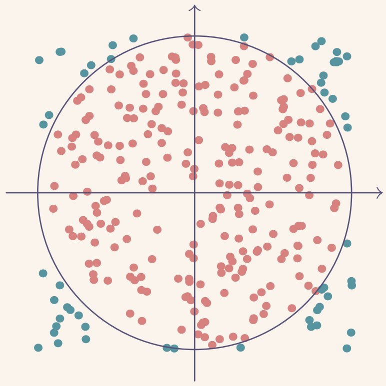


FIGURE 2.1 – Algorithme de MONTE-CARLO pour approximer π

On compte le nombre de points dans le disque, et ceux en dehors. Avec un grand nombre de points, on approxime le ratio de l'aire du disque et de l'aire du carré. Puis, on calcule

$$4 \times \left(\frac{\#\bullet}{\#\bullet + \#\bullet} \right) \approx \pi.$$

Le temps d'exécution dépend uniquement des paramètres de précision de l'algorithme, pas des tirages. Par contre, la qualité de la réponse dépend des choix aléatoires.

Définition (Algorithme de type LAS VEGAS)**:** Étant donné un problème P, un algorithme probabiliste répondant au problème P est dit de type LAS VEGAS dès lors que, s'il se termine, c'est en donnant une réponse correcte.

Définition (Algorithme de type MONTE-CARLO)**:** Étant donné un problème P, un algorithme probabiliste répondant au problème P est dit de type MONTE-CARLO dès lors que son temps d'exécution dépend uniquement de son entrée. L'algorithme peut cependant répondre de manière erronée au problème P avec une « certaine » probabilité.

REMARQUE:
Dans le cas d'un problème de décision (la réponse de l'algorithme est OUI ou NON), un algorithme de type MONTE-CARLO est dit

— « à erreur unilatérale » s'il existe une des réponses (OUI ou NON) r telle que, si l'algorithme répond r, alors il a raison (r est la réponse au problème) ;

— « à erreur bilatérale » si pour chaque réponse l'algorithme se trompe avec une probabilité non nulle.

Exemple:
On considère le problème : étant donné un tableau $T \in \{0,1\}^n$ tel que T contient p fois la valeur '0', avec $0 < p < n$, on cherche si, pour $i \in [\![1, n-1]\!]$, $T[i] = 1$.

Une réponse à ce problème est un algorithme de type Monte-Carlo, comme celui ci-dessous.

Algorithme 2.3 Algorithme de Monte-Carlo pour répondre au problème

Entrée $k \in \mathbb{N}$ et T un tableau
```
i ← 0
pour j ∈ ⟦1, k⟧ faire
    i ← 𝒰(⟦1, n − 1⟧)
    si T[i] = 1 alors
        retourner i
retourner i
```

Mais, on peut également donner un algorithme de Las-Vegas répondant aussi au même problème.

Algorithme 2.4 Algorithme de Las-Vegas pour répondre au problème

Entrée T un tableau
```
i ← 𝒰(⟦1, n − 1⟧)
tant que T[i] ≠ 1 faire
    i ← 𝒰(⟦0, n − 1⟧)
retourner i
```

Étudions l'algorithme de Las-Vegas : la correction partielle est validée. Étudions la terminaison : fixons T un tableau de taille n contenant p occurrences de '0' avec $0 < p < n$. Notons $(X_\ell)_{\ell \in \mathcal{D}}$ la suite des variables aléatoires donnant la valeur produite par le ℓième appel à $\mathcal{U}([\![0,1]\!])$. Remarquons que $\mathcal{D}$ est une variable aléatoire : en effet c'est $[\![0, N]\!]$ pour un certain $N \in \mathbb{N}$ si l'algorithme se termine; sinon, on a $\mathcal{D} = \mathbb{N}$. Notons A_q l'événement « l'algorithme s'arrête après q appels au générateur $\mathcal{U}([\![0, n-1]\!])$. On a donc

$$A_q = \text{“}T[X_0] = 0 \wedge T[X_1] = 0 \wedge \cdots \wedge T[X_{q-2}] = 0 \wedge T[X_{q-1}] = 1\text{”}.$$

D'où en passant aux probabilités, on a

$$P(A_q) = P(\text{“}T[X_0] = 0 \wedge T[X_1] = 0 \wedge \cdots \wedge T[X_{q-2}] = 0 \wedge T[X_{q-1}] = 1\text{”})$$

et, par indépendance, on a donc

$$P(A_q) = P(T[X_{q-1} = 1) \times \Big(\prod_{j=0}^{q-2} P(T[X_j] = 0) \Big).$$

Or, $\forall j \in [\![0, q-2]\!]$, $P(T[X_j] = 0) = \frac{p}{n}$ par uniformité, et, de plus, $P(T[X_{q-1}] = 1) = \frac{n-p}{n}$ par uniformité. On pose $\rho = \frac{p}{n}$, et donc $P(A_q) = \rho^{q-1}(1-\rho)$. Notons N l'événement « l'algorithme ne se termine pas » et calculons

$$\begin{aligned} P(N) &= 1 - P(\bar{N}) \\ &= 1 - P\Big(\bigvee_{q \in \mathbb{N}^*} A_q \Big) \\ &= 1 - \sum_{q \in \mathbb{N}^*} P(A_q) \\ &= 1 - \sum_{q \in \mathbb{N}^*} \rho^{q-1}(1-\rho) \\ &= 1 - \frac{1-\rho}{1-\rho} \\ &= 0 \end{aligned}$$

Soit $\mathcal{T}$ la variable aléatoire indiquant le temps d'arrêt de l'algorithme (en nombre d'itérations). On calcule l'espérance de $\mathcal{T}$:

$$\begin{aligned}
\mathrm{E}(\mathcal{T}) &= \sum_{t=1}^{+\infty} t \times P(\mathcal{T} = t) \\
&= \sum_{t=1}^{+\infty} t \times \rho^{t-1}(1-\rho) \\
&= (1-\rho) \sum_{t=1}^{+\infty} t \times \rho^{t-1} \\
&= (1-\rho) \sum_{t=1}^{+\infty} \sum_{k=0}^{t-1} \rho^{t} \\
&= (1-\rho) \sum_{k=0}^{+\infty} \sum_{t=k+1}^{+\infty} \rho^{t-1} \\
&= \cdots\cdots [1] \\
&= \frac{1}{1-\rho}
\end{aligned}$$

Étudions maintenant l'algorithme de Monte-Carlo. Il se termine trivialement, et la probabilité d'erreur est $\underbrace{\rho \times \cdots \times \rho}_{k}$. Par exemple, pour $\rho = \frac{1}{2}$, et $k = 80$, la probabilité d'erreur est de $\frac{1}{2^{80}}$.

2.2 Algorithme de Monte-Carlo

On considère le problème : « étant donné trois matrices A, B, C de $\mathcal{M}_n(\mathbb{Z}/2\mathbb{Z})$, a-t-on $A \cdot B = C$? »

Un algorithme trivial serait de calculer $A \cdot B$ et on vérifie, point à point, que $A \cdot B = C$. La complexité cet algorithme est en $\Theta(n^3)$ à cause du produit matriciel.

Un algorithme de Monte-Carlo serait le suivant.

Algorithme 2.5 Algorithme de Monte-Carlo répondant au problème

Entrée A, B, C trois matrices et $k \in \mathbb{N}$

pour $j \in [\![1, k]\!]$ **faire**
- $r \leftarrow \mathcal{U}\big((\mathbb{Z}/2\mathbb{Z})^n\big)$ $\quad \triangleright\ n$
- $r_1 \leftarrow B \cdot r$ $\quad \triangleright\ n^2$
- $r_2 \leftarrow A \cdot r_1$ $\quad \triangleright\ n^2$
- $r_3 \leftarrow C \cdot r$ $\quad \triangleright\ n^2$
- **si** $r_3 \neq r_2$ **alors**
  - **retourner** Non

retourner Oui

Dans le pire cas, la complexité est en $k \times n^2$. On cherche la probabilité d'erreur de cet algorithme. Pour cela, on utilise le lemme suivant.

> **Lemme:** Si $D \neq 0$, et $r \sim \mathcal{U}\big((\mathbb{Z}/2\mathbb{Z})^n\big)$, alors $P(D \cdot r = 0) \leqslant \frac{1}{2}$.

1. à faire

Preuve:
Si $D \in \mathscr{M}_n(\mathbb{Z}/2\mathbb{Z}) \setminus \{0\}$, alors il existe i et j tels que $D_{i,j} \neq 0$. Si $D \cdot r = 0$, on a

$$\sum_{k=1}^n D_{i,k} r_k = 0 \quad \text{et donc} \quad r_k = -\sum_{\substack{k=1\\k\neq j}}^n D_{i,k} r_k.$$

Donc, si $r_j \neq \sum_{\substack{k=1\\k\neq j}}^n D_{ik} r_k$, alors $P(D \cdot r \neq 0) \geqslant P\Big(r_j \neq \sum_{\substack{k=1\\k\neq j}}^n D_{i,k} r_k\Big)$. On note E_0 l'événement « $r_j = 0$ et $\sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k = 1$ » et E_1 l'événement « $r_j = 1$ et $\sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k = 0$. » D'où

$$P\Big(r_j \neq \sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k\Big) = P(E_0 \vee E_1).$$

Par incompatibilité, on a $P(E_0 \vee E_1) = P(E_0) + P(E_1)$, d'où

$$\forall a \in \{0,1\}, \quad P(E_a) = P\Big(r_j = a \wedge \sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k = 1 - a\Big)$$

et, par indépendance,

$$\forall a \in \{0,1\} \quad P(E_a) = P(r_j = a) \cdot P\Big(\sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k = 1 - a\Big) = \frac{1}{2} P(\ldots).$$

D'où

$$P\Big(r_j \neq \sum_{\substack{k=1\\j\neq j}} D_{i,k} r_k\Big) = \frac{1}{2}\Big[P\Big(\sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k = 1\Big) + P\Big(\sum_{\substack{k=1\\j\neq j}}^n D_{i,k} r_k = 0\Big)\Big]$$

et, par incompatibilité,

$$P\Big(r_j \neq \sum_{\substack{k=1\\j\neq j}} D_{i,k} r_k\Big) = \frac{1}{2}\frac{1}{2} P\Big(\sum_{\substack{k=1\\j\neq j}} D_{i,k} r_k \in \{0,1\}\Big) = \frac{1}{2}.$$

□

D'où, l'algorithme ci-dessous est tel que sa probabilité d'échec est de $\frac{1}{2^k}$. Or, l'algorithme a une complexité de $\mathcal{O}(k\, n^2)$.

2.3 Algorithme de type Las-Vegas

On étudie le tri rapide. On considère les fonctions "Partitionner," puis "Tri Rapide."

Algorithme 2.6 Fonction “Partitionner” utilisée dans le tri rapide

Entrée T le tableau à trier, g, d et p trois entiers (bornes du tableau)

Sortie un entier J et le sous-tableau $T[g..d]$ est modifié en $\bar{T}$ de sorte que $\bar{T}$[2] $[J] = \underline{T}[p]$, et $\forall i \in [\![g, J-1]\!]$, $\bar{T}[i] \leqslant \bar{T}[J]$, et $\forall i \in [\![J+1, d]\!]$, $\bar{T}[i] \geqslant \bar{T}[J]$, et $\forall i \in [\![0, n-1]\!] \setminus [\![g, d]\!]$, $\bar{T}[i] = \underline{T}[i]$, et $\bar{T}$ est une permutation de $\underline{T}$.

Échanger(T, g, d)
$J \leftarrow g$
$I \leftarrow g$
tant que $I < d$ **faire**
    si $T[I] > T[d]$ **alors** ▷ *Cas* “$T[I] >$ pivot”
        $I \leftarrow I + 1$
    sinon ▷ *Cas* “$T[I] \leqslant$ pivot”
        Échanger(T, I, J)
        $J \leftarrow J + 1$
        $I \leftarrow I + 1$
Échanger(T, J, d)
retourner J

Remarque:
On admet que $\bar{T}$ est une permutation de $\underline{T}$. On admet également que, $\forall i \in [\![0, n-1]\!] \setminus [\![g, d]\!]$, $\bar{T}[i] = \underline{T}[i]$.

> **Lemme:** “Partitionner” est correct.

Preuve:
On considère

$$(\mathcal{I}) : \begin{cases} \forall k \in [\![g, I]\!]\,,\ T[k] \leqslant T[d] & (1) \\ \forall k \in [\![J, I-1]\!]\,,\ T[k] > T[d] & (2) \\ g \leqslant J \leqslant I \leqslant d & (3) \end{cases}.$$

— Montrons que $\mathcal{I}$ est vrai initialement : à l'initialisation, $I = g$ et $J = g$, donc les trois propriétés $(\mathcal{I})$ sont trivialement vraies.

— Montrons que l'invariant $(\mathcal{I})$ se propage. Notons $\underline{I}$, $\underline{J}$, et $\underline{T}$ les valeurs de I, J et T avant itération de boucle. Notons également $\bar{I}$, $\bar{J}$ et $\bar{T}$ les valeurs de I, J et T après cette même itération de boucle. Supposons que $\underline{I}$, $\underline{J}$ et $\underline{T}$ vérifient $(\mathcal{I})$, et la condition de boucle. Montrons que $\bar{I}$, $\bar{J}$ et $\bar{T}$ vérifient $(\mathcal{I})$. On a donc, d'après $(\mathcal{I})$,

$$\begin{cases} \forall k \in [\![g, \underline{J}-1]\!]\,,\ \underline{T}[k] \leqslant \underline{T}[d] \\ \forall k \in [\![\underline{T}, \underline{I}-1]\!]\,,\ \underline{T}[k] > \underline{T}[d] \\ g \leqslant \underline{J} \leqslant \underline{I} \leqslant d. \end{cases}$$

Mais aussi, d'après la condition de boucle, $\underline{I} \leqslant d$. Mais également, d'après le programme,

— si $\underline{T}[\underline{I}] > \underline{T}[d]$, alors $\bar{I} = \underline{I} + 1$, $\bar{T} = \underline{T}$, et $\bar{J} = \underline{J}$;

— sinon si $\underline{T}[\underline{I}] \leqslant \underline{T}[d]$, et donc $\bar{J} = \underline{J} + 1$, $\bar{I} = \underline{I} + 1$, $\forall k \in [\![0n, -1]\!] \setminus \{\underline{I}, \underline{J}\}$, $\bar{T}[k] = \underline{T}[k]$, et $\bar{T}[\bar{I}] = \underline{T}[\underline{J}]$, et $\bar{T}[\underline{J}] = \underline{T}[\underline{I}]$.

Cas 1 $\underline{T}[\underline{I}] > \underline{T}[d]$, alors

(3) $g \leqslant \underline{J} = \bar{J} \leqslant \underline{I} < \bar{I} \leqslant d$.

(1) Soit $k \in [\![g, \bar{J}-1]\!]$, on a donc $k \in [\![g, \underline{J}-1]\!]$, et donc $\bar{T}[k] = \underline{T}[k] \leqslant \underline{T}[d] = \bar{T}[d]$.

(2) Soit $k \in [\![\bar{J}, \bar{I}-1]\!]$,

— si $k \in [\![\underline{J}, \underline{I}-1]\!]$, alors $\bar{T}[k] = \underline{T}[k] > \underline{T}[d] = \bar{T}[d]$.

2. La notation $\bar{T}$ représente le tableau T après l'algorithme, et la notation $\underline{T}$ représente le tableau T avant l'algorithme.

— si $k = \bar{I} - 1 = \underline{I}$, par condition `if`, alors $\bar{T}[\underline{I}] = \underline{T}[\underline{I}] > \underline{T}[d] = \bar{T}[d]$.

Cas 2 $\underline{T}[\underline{I}] \leqslant \underline{T}[d]$

(3) On a $\underline{J} \leqslant \underline{I}$, donc $\underline{J} + 1 \leqslant \underline{I} + 1$, d'où $g \leqslant \underline{J} + 1 = \bar{J} \leqslant \bar{I} \leqslant d$.

(1) Soit $k \in \llbracket g, \bar{J} - 1 \rrbracket$, donc

— si $k \in \llbracket g, \underline{J} - 1 \rrbracket$, alors $\bar{T}[k] = \underline{T}[k] \leqslant \underline{T}[d] = \bar{T}[d]$.

— si $k = \bar{J} - 1 = \underline{J}$, alors $\bar{T}[k] = \bar{T}[\underline{J}] = \underline{T}[\underline{I}] \leqslant \underline{T}[d] = \bar{T}[d]$.

(2) Soit $k \in \llbracket \bar{J}, \bar{I} - 1 \rrbracket$, alors

— si $k \in \llbracket \bar{J}, \underline{J} - 1 \rrbracket$, alors, comme $\bar{J} \geqslant \underline{J}$, et donc $\bar{T}[k] = \underline{T}[k] > \underline{T}[d] = \bar{T}[d]$.

— si $k = \bar{I} - 1 = \underline{I}$, et donc $\bar{T}[k] = \bar{T}[\underline{I}] = \bar{T}[\underline{J}] > \underline{T}[d] = \bar{T}[d]$.

Ainsi, $(\mathscr{I})$ est un invariant, et donc, en sortie de boucle, I, J et T sont tels que

$$\left.\begin{array}{l} \forall k \in \llbracket g, J-1 \rrbracket\,,\ T[k] \leqslant T[d] \\ \forall k \in \llbracket J, I-1 \rrbracket\,,\ T[k] > T[d] \\ g \leqslant J \leqslant I \leqslant d \end{array}\right\} \quad \text{et} \quad I \geqslant d,$$

la négation de la condition de boucle. On a donc $I = d$, et donc en fin de programme,

$$\forall k \in \llbracket g, J-1 \rrbracket\,,\ T[k] \leqslant T[J] \quad \text{et} \quad \forall k \in \llbracket J+1, I \rrbracket\,,\ T[k] > T[J].$$

□

Algorithme 2.7 Tri rapide

Entrée T un tableau, g et d les bornes de ce tableau

```
1: si d > g alors
2:     p ← ChoixPivot(T, g, d)
3:     J ← Partition(T, g, d, p)
4:     TriRapide(T, g, J − 1)
5:     TriRapide(T, J + 1, d)
```

La fonction "Tri(T)" est donc définie comme TriRapide$(T, 0, n-1)$ si T est un tableau de taille n.

Étudions rapidement l'influence du choix du pivot.

Cas 1 On définit "$\mathrm{ChoixPivot}(T, g, d) = g$." Ainsi

À faire : Figure

Figure 2.2 – Arbre des appels récursifs de "TriRapide" avec le pivot à gauche

Ainsi, la complexité de cet algorithme, avec ce choix de pivot, est en $(n-1) + (n-2) + (n-3) + \cdots + 2 = \Theta(n^2)$.

Cas 2 On définit maintenant le choix du pivot comme l'indice de la médiane.

À faire : Figure

Figure 2.3 – Arbre des appels récursifs de "TriRapide" avec le pivot à la médiane

Rédigeons-le rigoureusement : soit $C_n = \max_{T \text{ tableau de taille } n} C(T)$. Posons $(u_p)_{p \in \mathbb{N}} =$

$(C_{2^p})_{p\in\mathbb{N}}$. D'après l'algorithme de "TriRapide," on a

$$\begin{aligned} u_{p+1} &= 2^{p+1} - 1 + u_p + u_p \\ &= 2^{p+1} - 1 + 2u_p \\ &= (2^{p+1} - 1) + 2(2^p - 1) + 2^2 u_{p-1} \\ &= 2^{p+1} - 1 + 2^{p+1} - 2 + 2^2 u_{p-1} \\ &= 2^{p+1} - 1 + 2^{p+1} - 2 + 2^2(2^{p-1} - 1 + 2u_{p-2}) \\ &= 2^{p+1} - 1 + 2^{p+1} - 2 + 2^{p+1} - 2^2 + 2^3 u_{p-2}. \end{aligned}$$

On a donc $u_0 = 1$ et $u_p = p \times 2^p - (2^p - 1)$. Or, la suite $(c_n)_{n\in\mathbb{N}}$ est croissante. Or,

$$\forall n \in \mathbb{N},\ 2^{\lfloor \log_2 n\rfloor} \leqslant n \leqslant 2^{\lfloor \log_2 n\rfloor + 1}$$

donc

$$u_{\lfloor \log_2 n\rfloor} \leqslant C_n \leqslant u_{\lfloor \log_2 n\rfloor + 1}.$$

D'où

$$c_n \leqslant (\lfloor \log_2 n\rfloor + 1) \times 2^{\lfloor \log_2 n\rfloor + 1} - 2^{\lfloor \log_2 n\rfloor - 1}.$$

Et donc, on en déduit que $c_n = \Theta\big(n \log_2(n)\big)$.

Remarque (Notations):
On fixe un tableau T de taille n. De plus, on suppose dans toute la suite, que $T \in \mathfrak{S}_n$.[3] On note alors $X_g^d[T]$ la variable aléatoire indiquant le nombres de comparaisons effectuées par l'algorithme TriRapide(T, g, d), dès lors que $T(\llbracket g, d\rrbracket) \subseteq \llbracket g, d\rrbracket$.

On note de plus, $\mathrm{E}\big[X_g^d[T]\big]$ l'espérance de cette variable aléatoire.

Théorème: Le nombre moyen de comparaisons effectuées par l'algorithme de tri rapide pour une entrée T de taille n est équivalent à $2n \ln n$. Autrement dit,

$$\mathrm{E}\big[X_0^{n-1}[T]\big] \sim 2n \ln n.$$

Preuve:
- Lorsque $d \leqslant g$, alors $X_g^d[T] = 0$.
- Lorsque $g < d$,
  - dans l'éventualité d'un choix de pivot d'indice $p \in \llbracket g, d\rrbracket$, le nombre de comparaisons est alors

$$\underbrace{d - g - 1}_{\text{coût de Partition}(T,\,g,\,d,\,p)} + \overbrace{X_g^{T[p]-1}\left[T^{g,d,p}\right]}^{\text{coût du 1}^{\text{er}}\text{ appel récursif}} + \underbrace{X_{T[p]+1}^{d}\left[T^{g,d,p}\right]}_{\text{coût du 2}^{\text{nd}}\text{ appel récursif}}$$

3. T est une permutation de n éléments. Ici, $\mathfrak{S}_n$ représente l'ensemble des permutations de $\llbracket 1, n\rrbracket$.

où $T^{g,d,p}$ est le tableau T après appel à Partition(T, g, d, p). D'où

$$\begin{aligned}
\mathrm{E}\left[X_g^d[T]\right] &= \sum_{j=g}^{d} \mathrm{E}\left[X_g^d[T]\Big|_{p=j}\right] \cdot P(p=j) \\
&= \frac{1}{d-g+1} \sum_{j=g}^{d} \mathrm{E}\left[X_g^d[T]\Big|_{p=j}\right] \\
&= \frac{1}{d-g+1} \sum_{j=g}^{d} \Big(d-g-1 + \mathrm{E}\left[X_g^{T[j]-1}\left[T^{g,d,j}\right]\right] \\
&\qquad\qquad + \mathrm{E}\left[X_{T[j]+1}^{d}\left[T^{g,d,j}\right]\right]\Big) \\
&= (d-g-1) + \frac{1}{d-g+1} \sum_{k=g}^{d} \Big(\mathrm{E}\left[X_g^{k-1}\left[T^{g,d,T^{-1}[k]}\right]\right] \\
&\qquad\qquad + \mathrm{E}\left[X_{k+1}^{d}\left[T^{g,d,T^{-1}[k]}\right]\right]\Big)
\end{aligned}$$

car $T : [\![g, d]\!] \to [\![g, d]\!]$ est une bijection.
Soit donc la suite $(c_\ell)_{\ell \in \mathbb{Z}}$ définie par

$$\begin{cases} \forall \ell \in \mathbb{Z}^-, & c_\ell = 0 \\ \forall \ell \in \mathbb{N}^*, & c_\ell = \mathrm{E}\left[X_0^\ell[\mathrm{id}]\right]. \end{cases}$$

On a alors

$$\begin{aligned}
\forall \ell \in \mathbb{N}^*, \quad c_\ell &= (\ell-1) + \frac{1}{\ell-1} \sum_{k=0}^{\ell} (c_{k-1} - c_{\ell-k-1}) \\
c_\ell &= (\ell-1) + \frac{2}{\ell+1} \sum_{k=1}^{\ell-1} c_k \\
(\ell+1) c_\ell &= (\ell+1)(\ell-1) + 2 \sum_{k=1}^{\ell-1} c_k
\end{aligned}$$

D'où, $\ell c_\ell = \ell(\ell-2) + 2\sum_{k=1}^{\ell-2} c_k$, et donc

$$(\ell+1)c_\ell - \ell c_{\ell-1} = \ell^2 - 1 - \ell^2 + 2\ell + 2c_{\ell-1}.$$

On en déduit donc que

$$(\ell+1)c_\ell - (\ell+2)c_{\ell-1} = 2\ell - 1$$

et donc

$$\frac{c_\ell}{\ell+2} - \frac{c_{\ell-1}}{\ell+1} = \frac{2\ell-1}{(\ell+1)(\ell+2)}.$$

Soit alors $(u_\ell)_{\ell \in \mathbb{N}} = \big(c_\ell/(\ell+2)\big)_{\ell \in \mathbb{N}}$, et $u_0 = 0$. Alors

$$u_\ell = \sum_{k=1}^{n} (u_k - u_{k-1}) = \sum_{k=1}^{n} \frac{2k-1}{(k+1)(k+2)}$$

or $\frac{2k-1}{(k+1)(k+2)} \sim \frac{2}{k}$, et $\sum_{k \geqslant 1} \frac{2}{k}$ diverge donc $u_\ell \sim \sum_{k=1}^{\ell} \frac{2}{k} \sim 2\ln \ell$. On en déduit donc que $c_\ell \sim 2\ell \ln \ell$.

$\square$

Dans la preuve précédente, on a utilisé le lemme suivant.

Lemme: Soit $(g, d) \in \mathbb{N}^2$ et soit $T \in \mathfrak{S}_n$ une permutation telle que $T(\llbracket g, d \rrbracket) \subseteq \llbracket g, d \rrbracket$.

$$\mathrm{E}\left[X_g^d[T]\right] = \mathrm{E}\left[X_0^{d-g}[\mathrm{id}]\right].$$

Preuve (par récurrence forte sur $d - g = \ell \in \mathbb{N}$): — Soient $(g, d) \in \mathbb{N}^2$ tel que $d - g = 0$. Soit $T \in \mathfrak{S}_n$ telle que $T(\llbracket g, d \rrbracket) \subseteq \llbracket g, d \rrbracket$. On a bien $X_g^d[T] = 0 = X_0^{d-g}[\mathrm{id}]$.

— On remarque, par hypothèse de récurrence,

$$\mathrm{E}\left[X_g^{k-1}\left[\overbrace{T^{g,d,T^{-1}[k]}}^{k-1-g<d-g}\right]\right] = \mathrm{E}\left[X_0^{k-1-g}[\mathrm{id}]\right]$$

et

$$\mathrm{E}\left[X_{k-1}^d\left[T^{g,d,T^{-1}[k]}\right]\right] = \mathrm{E}\left[X_0^{d-k-1}[\mathrm{id}]\right].$$

On a alors

$$\begin{aligned}
&\mathrm{E}\left[X_g^d[T]\right] \\
&= (d-g-1) + \frac{1}{d-g+1} \sum_{k=g}^{d} \left(\mathrm{E}\left[X_g^{k-1}\left[T^{g,d,T^{-1}[k]}\right]\right] + \mathrm{E}\left[X_{k-1}^d\left[T^{g,d,T^{-1}[k]}\right]\right]\right) \\
&= (d-g-1) + \frac{1}{d-g-1} \sum_{k=g} \left(\mathrm{E}\left[X_0^{k-1-g}[\mathrm{id}]\right] + \mathrm{E}\left[X_0^{d-k-1}[\mathrm{id}]\right]\right).
\end{aligned}$$

Ceci est vrai pour tout $T \in \mathfrak{S}_n$ telle que $T(\llbracket g, d \rrbracket) \subseteq \llbracket g, d \rrbracket$, donc

$$\begin{aligned}
\mathrm{E}\left[X_g^d[\mathrm{id}]\right] &= (d-g-1) + \frac{1}{d-g-1} \sum_{k=g} \left(\mathrm{E}\left[X_0^{k-1-g}[\mathrm{id}]\right] + \mathrm{E}\left[X_0^{d-k-1}[\mathrm{id}]\right]\right) \\
&= \mathrm{E}\left[X_g^d[T]\right]
\end{aligned}$$

□

Annexe 2.A Hors-programme

CHAPITRE

3 APPRENTISSAGE

Sommaire

3.1 Motivation

L'intelligence artificielle est vu comme un « objet magique » mais ce n'est pas le cas : c'est ce que nous allons étudier dans ce chapitre. Il existe plusieurs méthodes permettant l'apprentissage : descente de gradient, ?, ...

La base de donnée la plus utilisée est MNIST : elle contient 60 000 images de 28×28 pixels représentant un chiffre, et le chiffre correspondant. L'idée de l'apprentissage est de « deviner » le chiffre dessiné en connaissant l'image.

3.2 Vocabulaire

Définition: On appelle *signature de données* un n-uplet de paires nom, ensemble ; on le typographie

$$\big(\mathrm{nom}_1 : S_1, \mathrm{nom}_2 : S_2, \ldots, \mathrm{nom}_n : S_n\big).$$

EXEMPLE:

1. $\mathbb{S}_1 = \big(\text{titre} : \text{string}, \text{longueur} : \mathbb{N}, \text{date} : \mathbb{N}\big)$,
2. $\mathbb{S}_2 = \big(x : \mathbb{R}, y : \mathbb{R}\big)$,
3. $\mathbb{S}_3 = \big(R : [\![0, 255]\!]\,, G : [\![0, 255]\!]\,, B : [\![0, 255]\!]\,\big)$.

Définition: Étant donné une signature de données $\mathbb{S} = (\text{nom}_1 : S_1, \ldots, \text{nom}_n : S_n)$, on appelle *donnée* un vecteur

$$\bar{v} = (v_1, v_2, \ldots, v_n) \in S_1 \times S_2 \times \cdots \times S_n.$$

Exemple: 1. ("2001, a space odyssey", $139, 1968$) est une donnée/un vecteur de signature $\mathbb{S}_1$.

2. $(\pi, \sqrt{2})$ est une donnée/un vecteur de signature $\mathbb{S}_2$.

Définition: Étant donné une signature de données $\mathbb{S}$, on appelle *jeu de données* un ensemble fini de vecteurs de signature $\mathbb{S}$.

Définition: Étant donnée une signature de données $\mathbb{S}$ et un ensemble de classes $\mathscr{C}$, on appelle *jeu de données classifié* la donnée

— d'un jeu de données S,

— d'une fonction $f : S \to \mathscr{C}$ de classification.

3.3 Apprentissage supervisé

L'objectif de cette section est de construire des fonctions de classification, à partir d'un jeu de données classifié.

Définition: Étant donné une signature de données $\mathbb{S}$, et un ensemble de classes $\mathscr{C}$, on appelle *fonction de classification* une fonction des données de de signature $\mathbb{S}$ dans $\mathscr{C}$.

Remarque:
On discutera de la « qualité » d'une fonction de classification en fonction de ses résultats sur les données d'un jeu de données et sur des exemples de tests.

3.3.1 k plus proches voisins

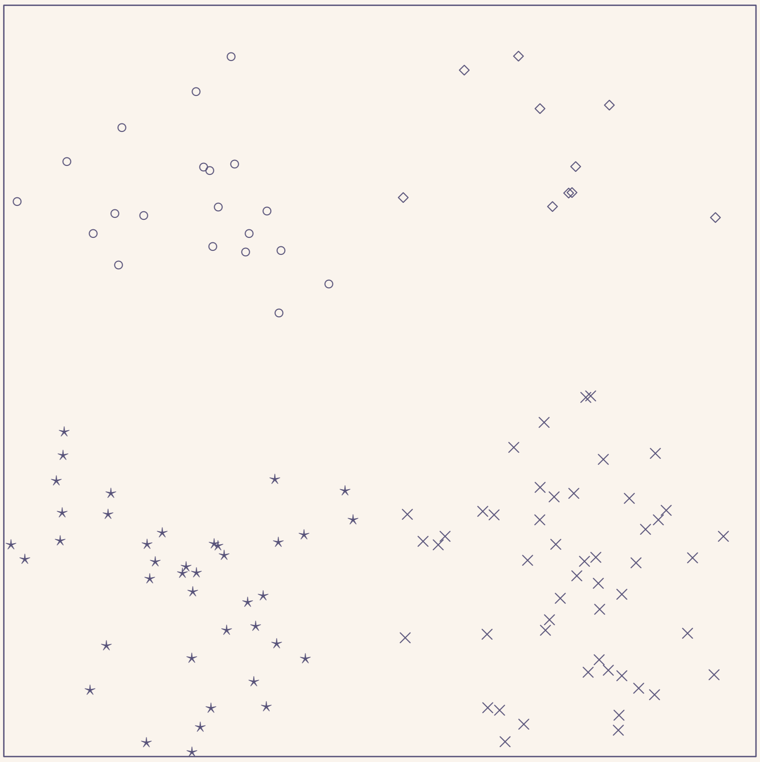


FIGURE 3.1 – Représentation de l'algorithme des k plus proches voisins

Algorithme 3.8 k-NN (k *nearest neighbors*)

Entrée Un jeu de données classifié (S, c), un vecteur d'entrée $\bar{v}$

On trie S par distance à croissante de v en $d_1, d_2, \ldots, d_k, d_{k+1}, \ldots$
Soit D un dictionnaire de $\mathscr{C}$ vers $\mathbb{N}$ initialisé à 0 [1]
pour $j \in [\![1, k]\!]$ **faire**
  $D\big[c(d_j)\big] \leftarrow D\big[c(d_j)\big] + 1$
retourner $\operatorname{argmax}_{d \in \mathscr{C}} D[d]$

REMARQUE:
On doit avoir $k \leqslant n$, et l'espace doit être muni d'une distance. Les résultats de l'algorithme dépendent fortement du jeu de données, du paramètre k et de la distance choisie.

Matrice de confusion

Définition: On appelle *matrice de confusion* d'un algorithme de prédiction $\mathscr{A}$ sur un jeu de données classifié (T, c), la matrice

$$\Big(\operatorname{Card}\{t \in T \mid \mathscr{A}(t) = i \text{ et } c(t) = j\}\Big)_{(i,j) \in \mathscr{C}^2}.$$

Dans le cas particulier dans le cas d'une classification $(\boldsymbol{V}, \boldsymbol{F})$, on nomme

1. où toutes les valeurs sont initialisées à 0, pas un dictionnaire vide

vrai \ $\mathscr{A}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{F}$	vrai négatif	faux négatif
$\boldsymbol{V}$	faux positif	vrai positif

TABLE 3.1 – Matrice de confusion dans le cas d'une classification en $\boldsymbol{V}$ et $\boldsymbol{F}$

Comment améliorer la performance de l'algorithme des k plus proches voisins ? En dimension 1, on peut utiliser une dichotomie. Mais, dans des dimensions plus grandes, l'ordre lexicographique, et l'ordre produit ne fonctionnent pas. Mais, on peut appliquer une "dichotomie" en changeant de dimension. Par exemple, en deux dimension, on a

À faire : Représenter le schéma

FIGURE 3.2 – Représentation de la "dichotomie" en dimension 2

Pour représenter cette structure de données, on utilise un arbre binaire comme montré ci-dessous. Cet arbre est appelé un arbre k-dimensionnels.

À faire : Faire l'arbre

FIGURE 3.3 – Arbre 2-dimensionnel représentant la "dichotomie" précédente

3.3.2 Arbres k-dimensionnels

REMARQUE (Notations):
Étant donné un jeu de données S, on note pour $v \in S$,

$$S^{\leqslant_i v} = \{u \in S \mid u_i \leqslant v_i\} \qquad \text{et} \qquad S^{>_i v} = \{u \in S \mid u_i > v_i\}.$$

Algorithme 3.9 "F" : Fabrication d'un arbre k-dimensionnel

Entrée $\mathscr{V}$ un jeu de données et $i \in [\![0, n-1]\!]$, où n est la dimension des données

```
si 𝒱 = ∅ alors
    retourner Vide
sinon
    On cherche v ∈ 𝒱 tel que v_i est la médiane de {u_i | u ∈ 𝒱}
    retourner Nœud((v, i), F((𝒱 \ {v})^{≤_i v}, i + 1 mod n), F((𝒱 \ {v})^{>_i v}, i + 1 mod n))
```

Algorithme 3.10 "R" : Recherche du point le plus proche

Entrée Un arbre k-dimensionnel et un vecteur v

si T est vide **alors**
  retourner $\varnothing$
sinon
  Nœud$\big((u, i), G, D\big) \leftarrow T$
  si $u_i \leqslant v_i$ **alors**
    $W \leftarrow \mathrm{R}(D, v)$
    si $W = \mathrm{None}$ **alors**
      $W' \leftarrow \mathrm{R}(G, v)$
      si $W' = \varnothing$ **alors**
        retourner $\mathrm{Some}(u)$
      sinon
        $\mathrm{Some}(z) \leftarrow W'$
        retourner le plus proche de v entre u et z
    sinon
      $\mathrm{Some}(w) \leftarrow W$
      si $v_i - u_i \leqslant d(w, v)$ **alors** ▷ *$d(w, v)$ représente la distance entre w et v*
        $W' \leftarrow \mathrm{R}(G, v)$
        si $W' = \mathrm{None}$ **alors**
          retourner Some(plus proche de v entre u et w)
        sinon
          $\mathrm{Some}(z) \leftarrow w$
          retourner Some(plus proche de v entre u, z, et w)
      sinon
        retourner W

3.3.3 Algorithme ID3

L'algorithme des k plus proches voisins (sans arbres k-dimensionnels) n'a pas de *phase d'apprentissage* : en effet, les données ne sont pas réorganisées. Mais, par exemple, pour l'utilisation des arbres k-dimensionnels, les données sont réorganisées dans un arbre.

Ce qu'on aimerai avoir, c'est des *bordures* entre les différentes classes. Par exemple, dans l'exemple précédent, on aimerai avoir les différentes zones ci-dessous.

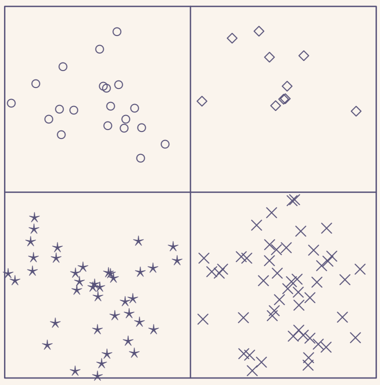


FIGURE 3.4 – Représentation de *bordures* entre les différentes classes

De ces zones, on peut construire un algorithme qui classifie les données, que l'on représente sous forme d'arbre. Ce type d'arbre est un *arbre de décision*. Dans l'exemple précédent, on peut donc créer l'arbre ci-dessous.

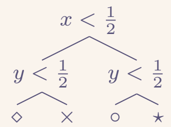


Figure 3.5 – Arbre de décision pour la classification
À faire : Refaire l'arbre plus proprement

Dans le reste de cette section, on s'intéresse uniquement à des données de $\mathbb{B}^n$ (une liste de n booléens) pour un certain $n \in \mathbb{N}$.

On considère l'exemple dont les données ci-dessous.

Transport	Moteur	Rails	Sous-terre	$\geqslant 320$ km/h	Train ?
A380	✓	×	×	✓	×
TGV	✓	✓	×	✓	✓
Métro	✓	✓	✓	×	✓
Wagonnet	×	✓	✓	×	×
Draisine	×	✓	×	×	×
Tram	✓	✓	×	×	✓

Table 3.2 – Exemple de données

Entropie

Définition: Étant donné un variable aléatoire finie X à valeurs dans E. On note $p_X : E \to [0,1]$ sa loi de probabilité :

$$\forall x \in E, \quad p_X(x) = P(X = x).$$

On définit l'*entropie* $H(X)$ de cette variable aléatoire comme

$$\mathrm{H}(X) = -\sum_{x \in E} p_X(x)\ \ln\big(p_X(x)\big).$$

On prolonge $p_X(x)\ln(p_X(x))$ par continuité à la valeur 0 lorsque $p_X(x) = 0$.

Exemple:
On considère la variable aléatoire X à valeur dans $\{\bullet, \bullet\}$ telle que $P(X = \bullet) = 1$ et $P(X = \bullet) = 0$. On a

$$\mathrm{H}(X) = -0 \ln 0 - 1 \ln 1 = 0.$$

Exemple:
On considère la variable aléatoire X à valeur dans $\{\bullet, \bullet\}$ telle que $P(X = \bullet) = p$ et $P(X = \bullet) = 1 - p$, avec $p \in [0,1]$. On a alors

$$\mathrm{H}(X) = -p \ln p - (1-p)\ln(1-p)$$

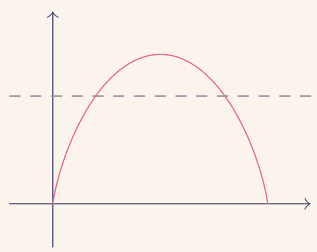


Figure 3.6 – Représentation graphique de $\mathrm{H}(X)$ en fonction de p

Lemme: Si $(p_i)_{i\in[\![1,n]\!]} \in \,]0,1]^n$ sont tels que $\sum_{i=1}^n p_i = 1$. Soit $(q_i)_{i\in[\![1,n]\!]}$ tels que $\forall i$, $q_i = \frac{1}{n}$. On a alors

$$-\sum_{i=1}^n p_i \ln p_i \leqslant -\sum_{i=1}^n p_i \ln(q_i).$$

Preuve:
On a

$$\sum_{i=1}^n p_i \ln(q_i) - \sum_{i=1}^n p_i \ln(p_i) = \sum_{i=1}^n p_i \ln\left(\frac{p_i}{q_i}\right) \leqslant \ln\left(\sum_{i=1}^n p_i \frac{q_i}{p_i}\right) = 0.$$

□

Propriété: Soit $n = |E|$. L'entropie d'une variable aléatoire à valeurs dans E est maximale lorsque

$$\forall e \in E,\ P(X = e) = \frac{1}{n}.$$

Preuve:
On conclut, d'après le lemme précédent, que … □

Définition: Étant donné un jeu de données classifiés (S, c) où $c : \mathbb{S} \to E$, on appelle *entropie* de ce jeu de données l'entropie de la variable aléatoire $c(Y)$ où $Y \sim \mathcal{U}(S)$. On a donc

$$\mathrm{H}\big((S,c)\big) = -\sum_{e\in E} \frac{\mathrm{Card}\ c^{-1}(\{e\})}{\mathrm{Card}\ S} \ln\left(\frac{\mathrm{Card}\ c^{-1}(\{e\})}{\mathrm{Card}\ S}\right).$$

Définition: Étant donné un jeu de données une partition $\{S_1, S_2, \dots, S_p\}$ d'un jeu de données classifié (S, c), l'*entropie* de cette partition est la moyenne (pondérée par les cardinaux et renormalisée) :

$$\mathrm{H}\big((\{S_1,\dots,S_p\},c)\big) = \sum_{i=1}^p \frac{\mathrm{Card}\ S_i}{\mathrm{Card}\ S} \mathrm{H}\big((S_i,c)\big).$$

L'entropie de $\{w,a,t,d,r,m\}$ est $\mathrm{H} = -\frac{3}{6}\ln\left(\frac{3}{6}\right) - \frac{3}{6}\ln\left(\frac{3}{6}\right) = \ln 2 \simeq 0{,}69$. Mais, avec le découpage de l'arbre de décision ci-dessous, on obtient l'entropie

$$\begin{aligned}\mathrm{H} &= \frac{2}{6}\mathrm{H}(\{w;d\}) + \frac{4}{6}\mathrm{H}(\{a,t,r,m\}) \\ &= \frac{2}{6}\times 0 + \frac{4}{6}\times\left(-\frac{1}{4}\ln\left(\frac{1}{4}\right) - \frac{3}{4}\ln\left(\frac{3}{4}\right)\right) \\ &\simeq 0{,}37.\end{aligned}$$

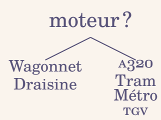


FIGURE 3.7 – Arbre de décision possible se basant sur le moteur

Avec un autre arbre (comme celui ci-dessous), on obtient une entropie différente :

$$\begin{aligned}
\mathrm{H} &= \frac{1}{6}\mathrm{H}(\{a\}) + \frac{5}{6}\mathrm{H}(\{w,d,t,r,m\}) \\
&= \frac{5}{6}\left(-\frac{2}{5}\ln\left(\frac{2}{5}\right) - \frac{3}{5}\ln\left(\frac{3}{5}\right)\right) \\
&\simeq 0{,}56.
\end{aligned}$$

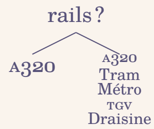


Figure 3.8 – Arbre de décision possible se basant sur les rails

Pour le sous-terrain, on a $\mathrm{H} = \ln 2$.

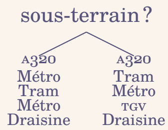


Figure 3.9 – Arbre de décision possible se basant sur sous-terrain

À faire : Vérifier

À faire : Autre cas

Ainsi, on choisit de commencer avec la condition “moteur” car l’entropie est la plus faible avec cette condition. Ainsi, l’arbre de décision ressemble à celui ci-dessous.

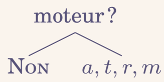


Figure 3.10 – Arbre de décision partiel

On réitère avec les autres conditions. L’entropie en se basant sur la vitesse est $\frac{1}{2}\ln 2 \simeq 0{,}34$. En effet, l’arbre de décision possible ressemble à celui ci-dessous.

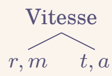


Figure 3.11 – Arbre de décision possible se basant sur le moteur puis la vitesse

Mais, en se basant sur sous-terrain, on obtient une entropie de $\frac{3}{4}\left(-\frac{2}{3}\ln\left(\frac{2}{3}\right) - \frac{1}{3}\ln\left(\frac{1}{3}\right)\right) \simeq 0{,}48$.

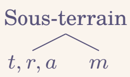


Figure 3.12 – Arbre de décision possible se basant sur le moteur puis sous-terrain

Et, en se basant sur les rails, on obtient une entropie de 0.

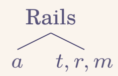


Figure 3.13 – Arbre de décision possible se basant sur le moteur puis les rails

On en déduit que l'arbre final de décision est celui ci-dessous.

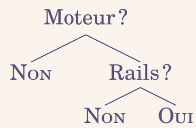


Figure 3.14 – Arbre de décision final pour la classification de trains

Les données que l'on a utilisées sont pour l'apprentissage. On teste notre arbre de décision sur les données ci-dessous.

Nom	Moteur	Rail	Sous-terre	Vitesse	Résultat de l'algorithme
Bus	✓	×	×	×	×
TER	✓	✓	×	×	✓
Cheval	×	×	×	×	×
Ascenseur spatial	✓	✓	×	×	✓

Table 3.3 – Test de l'arbre de décision créé

Attention : il ne faut pas faire du *sur-apprentissage*, comme montré sur la figure ci-dessus. À faire : Figure sur-apprentissage Aussi, il faut faire attention aux critères : par exemple, lors de la classification de photos de chats et de chiens, les photos de chiens sont en général prises en extérieur et l'algorithme ID3 aurait donc pu choisir de baser sa décision sur l'emplacement de la photo, même si elle n'importe pas dans la différenciation chat/chiens.

Autre exemple : on considère la table de données ci-dessous. Utilisons l'algorithme ID3 sur ces données, et trouvons l'arbre de décision.

A	B	C	D	Classification
✓	×	×	✓	●
✓	✓	×	✓	●
✓	×	✓	✓	●
✓	✓	✓	✓	●
×	×	✓	✓	●
×	✓	✓	×	●
×	×	×	×	●
×	✓	×	✓	●

Table 3.4 – Table de données d'exemple

Pour les conditions A, B et C, on a $\mathrm{H} \simeq 0{,}63$ et, pour D, on a $\mathrm{H} = 0{,}65$. Comme on prend le 1$^{\text{er}}$ dans l'ordre lexicographique, on choisit la condition A. De même, on construit l'arbre ci-dessous.

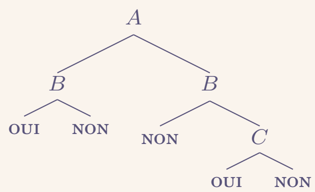


Figure 3.15 – Arbre de décision pour la table de données précédente

Partie II

Travaux Dirigés

TRAVAUX DIRIGÉS

1 ORDRE & INDUCTION

TD 1.1 Listes, listes, listes !

1. On a $\forall \ell \in \mathscr{L}$, $@([\,], \ell) = \ell$; $\forall \ell_1, \ell \in \mathscr{L}$, $@\big(::(x, \ell_1), \ell\big) = ::\big(x, @(\ell_1, \ell)\big)$.

2. On fait une induction. Comme dans l'énoncé, on passe le '@' en infixe. Notons P_ℓ : "$\ell \mathbin{@} [\,] = \ell$".

   Montrons $P_{[\,]}$: on sait que $[\,] \mathbin{@} [\,] = [\,]$ par définition de @.

   On suppose P_ℓ est vraie pour une certaine liste $\ell \in \mathscr{L}$. Montrons que, $\forall x \in \mathbb{N}$, $P_{::(x,\ell)}$ vrai. Soit $x \in \mathbb{N}$.

$$\begin{aligned}\big(::(x,\ell)\big) \mathbin{@} [\,] &\overset{\text{(def)}}{=} ::(x, \ell \mathbin{@} [\,])\\ &\overset{(P_\ell)}{=} ::(x,\ell).\end{aligned}$$

3. Notons P_{ℓ_1} : "$\forall \ell_2, \ell_3 \in \mathscr{L}$, $(\ell_1 \mathbin{@} \ell_2) \mathbin{@} \ell_3 = \ell_1 \mathbin{@} (\ell_2 \mathbin{@} \ell_3)$", où $\ell_1 \in \mathscr{L}$ est une liste.

   Soient $\ell_2, \ell_3 \in \mathscr{L}$ deux listes. On a, par définition de @, $([\,] \mathbin{@} \ell_2) \mathbin{@} \ell_3 = \ell_2 \mathbin{@} \ell_3$ et $[\,] \mathbin{@} (\ell_2 \mathbin{@} \ell_3) = \ell_2 \mathbin{@} \ell_3$.

   Soit $\ell_1 \in \mathscr{L}$ une liste telle que P_{ℓ_1}. Soient $\ell_2, \ell_2 \in \mathscr{L}$ deux listes. Soit $x \in \mathbb{N}$. Montrons que $P\big(::(x,\ell_1)\big)$:

$$\begin{aligned}(::(x,\ell_1) \mathbin{@} \ell_2) \mathbin{@} \ell_3 &= ::(x, \ell_1 \mathbin{@} \ell_2) \mathbin{@} \ell_3\\ &= ::\big(x, (\ell_1 \mathbin{@} \ell_2) \mathbin{@} \ell_3\big)\\ &\overset{(H)}{=} ::\big(x, \ell_1 \mathbin{@} (\ell_2 \mathbin{@} \ell_3)\big)\\ &= ::(x,\ell_1) \mathbin{@} (\ell_2 \mathbin{@} \ell_3).\end{aligned}$$

4. Notons P_{ℓ_1} : "$\forall \ell_2 \in \mathscr{L}$, $\texttt{rev}(\ell_1 \mathbin{@} \ell_2) = \texttt{rev}(\ell_2) \mathbin{@} \texttt{rev}(\ell_1)$. Soit $\ell_2 \in \mathscr{L}$.

   On a $\texttt{rev}([\,] \mathbin{@} \ell_2) = \texttt{rev}(\ell_2) = \texttt{rev}(\ell_2) \mathbin{@} \texttt{rev}([\,])$.

On suppose P_{ℓ_1} vraie pour une certaine liste $\ell_1 \in \mathscr{L}$. Soit $x \in \mathbb{N}$.

$$\begin{aligned}
\texttt{rev}(::x, \ell_1) \mathbin{@} \ell_2) &= \texttt{rev}(::(x, \ell_1 \mathbin{@} \ell_2) \\
&= \texttt{rev}(\ell_1 \mathbin{@} \ell_2) \mathbin{@} ::(x, [\,]) \\
&= \big(\texttt{rev}(\ell_2) \mathbin{@} \texttt{rev}(\ell_1)\big) \mathbin{@} ::(x, [\,]) \\
&= \texttt{rev}(\ell_2) \mathbin{@} \big(\texttt{rev}(\ell_1) \mathbin{@} ::(x, [\,])\big) \\
&= \texttt{rev}(\ell_2) \mathbin{@} \texttt{rev}\big(::(x, \ell_1)\big)
\end{aligned}$$

5. Notons, pour toute liste $\ell \in \mathscr{L}$, P_ℓ : “$\texttt{rev}\big(\texttt{rev}(\ell)\big) = \ell$”.
Montrons que $P_{[\,]}$ est vraie : $\texttt{rev}(\texttt{rev}([\,])) = \texttt{rev}([\,]) = [\,]$.
Soit une liste $\ell \in \mathscr{L}$ telle que P_ℓ soit vraie. Soit $n \in \mathbb{N}$. Montrons que $P_{::(x,\ell)}$ vraie :

$$\begin{aligned}
\texttt{rev}(\texttt{rev}(::(x, \ell))) &= \texttt{rev}(\texttt{rev}(\ell) \mathbin{@} ::(x, [\,])) \\
&= \texttt{rev}(::(x, [\,])) \mathbin{@} ::(x, \ell) \mathbin{@} \ell \\
&= [\,] \mathbin{@} ::(x, [\,]) \mathbin{@} \ell \\
&= ::(x, [\,]) \mathbin{@} \ell \\
&= ::(x, \ell).
\end{aligned}$$

TD 1.2 Ensembles définis inductivement

La correction est disponible sur *cahier-de-prepa*.

TD 1.3 Arbres, Arbres, Arbres !

1. On pose $R = \left\{ V\big|^0, N\big|^2_{\mathbb{N}} \right\}$. Ainsi, par induction nommée, on crée l'ensemble $\mathscr{A}$ des arbres.
2. On pose

$$\begin{aligned}
h : \mathscr{A} &\longrightarrow \mathbb{N} \cup \{-1\} \\
V &\longmapsto -1 \\
N(x, f_1, f_2) &\longmapsto 1 + \max\big(h(f_1), h(f_2)\big)
\end{aligned}$$

et

$$\begin{aligned}
t : \mathscr{A} &\longrightarrow \mathbb{N} \\
V &\longmapsto 0 \\
N(x, f_1, f_2) &\longmapsto 1 + t(f_1) + t(f_2)
\end{aligned}$$

3. On rappelle les relations taille/hauteur (vues l'année dernière) :

$$h(a) + 1 \leqslant t(a) \leqslant 2^{h(a)+1} - 1.$$

Soit, pour tout arbre $a \in \mathscr{A}$, P_a la propriété ci-dessus. Montrons que P_a est vraie pour tout arbre $a \in \mathscr{A}$ par induction.
Montrons que P_V vraie : on a $h(V) + 1 = 1 - 1 = 0$, $t(V) = 0$ et $2^{h(V)+1} - 1 = 1 - 1 = 0$ d'où $h(V) + 1 \leqslant t(V) \leqslant 2^{h(V)+1} - 1$.

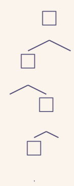
Supposons P_g vraie et P_d vraie pour deux arbres $g, d \in \mathscr{A}$. Soit $x \in \mathbb{N}$. Montrons que $P_{N(x,g,d)}$ est vraie :

$$\begin{aligned} h\big(N(x,g,d)\big) - 1 &= 1 + \max\big(h(g), h(d)\big) + 1 \\ &\leqslant \max(t(g) - 1, t(d) - 1) + 2 \\ &\leqslant \max\big(t(g), t(d)\big) + 1 \\ &\leqslant t(g) + t(d) + 1 \\ &= t\big(N(x,g,d)\big) \end{aligned}$$

et

$$\begin{aligned} t\big(N(x,g,d)\big) &= t(g) + t(d) + 1 \\ &\leqslant 2^{h(g)+1} + 2^{h(d)+1} - 1 \\ &\leqslant 2 \times 2^{\max(h(g),h(d))+1} - 1 \\ &\leqslant 2^{\max(h(g),h(d))+2} - 1 \\ &\leqslant 2^{h(N(x,g,d))+1} - 1. \end{aligned}$$

4. Je pense qu'il y a une erreur d'énoncé : les arbres crées sont de la forme



où □ représente un nœud. Il ne sont pas de la forme “peigne.”

TD 1.4 Ordre sur *powerset*

1. Soient A et B deux parties d'un ensemble ordonné $(S, \preccurlyeq)$. Si $A = B$, alors $A \dot{\preccurlyeq} B$ et donc A et B sont comparables. Si $A \neq B$, alors $A \triangle B \neq \varnothing$, et donc $A \triangle B$ admet un plus petit élément m. Par définition de $\triangle$, on a $m \in A$ (et donc $A \dot{\succcurlyeq} B$) ou $m \in B$ (et donc $A \dot{\preccurlyeq} B$). On en déduit que A et B sont comparables. La relation $\dot{\preccurlyeq}$ est donc totale.
2. On a
$$\varnothing \dot{\preccurlyeq} \{2\} \dot{\preccurlyeq} \{1\} \dot{\preccurlyeq} \{1,2\} \dot{\preccurlyeq} \{0\} \dot{\preccurlyeq} \{0,2\} \dot{\preccurlyeq} \{0,1\} \dot{\preccurlyeq} \{0,1,2\}.$$
3. Non, l'ordre $(\wp(S), \dot{\preccurlyeq})$ n'est pas forcément bien fondé. Par exemple, on pose $(S, \preccurlyeq) = (\mathbb{N}, \leqslant)$. Toute partie non vide de $\mathbb{N}$ admet bien un plus petit élément. Mais, la suite $(u_n)_{n\in\mathbb{N}} = (\{n\})_{n\in\mathbb{N}}$ est strictement décroissante : en effet, pour $n \in \mathbb{N}$, on a $u_n \triangle u_{n+1} = \{n, n+1\}$ qui admet pour élément minimal $n \in A$, d'où $u_n \dot{\succcurlyeq} u_{n+1}$.

TD 1.5 Ordres bien fondés en vrac

1. Non, l'ensemble $(\mathbb{N}, \sqsubseteq)$ n'est pas un ensemble ordonné. En effet, en posant $n = 4$ et $m = 8$, on a $\forall i \in \mathbb{N},\ \frac{n}{2^i} \pmod 2 = 0$, et $\forall i \in \mathbb{N},\ \frac{m}{2^i} \pmod 2 = 0$. Ainsi, on a $n \sqsubseteq m$, et $m \sqsubseteq n$, mais comme $n \neq m$, la relation “$\sqsubseteq$” n'est pas anti-symétrique, ce n'est donc pas une relation d'ordre (et donc encore moins un ordre bien fondé).
2. Non, l'ensemble $(\Sigma^*, \sqsubseteq)$ n'est pas un ensemble ordonné. En effet, en posant $\Sigma = \{a, b\}$, et $u = aa$ et $v = ab$ deux mots de Σ, on a $|u| = |v|$ et donc $u \sqsubseteq v$ et $u \sqsupseteq v$ mais comme $u \neq v$, la relation “$\sqsubseteq$” n'est pas anti-symétrique, ce n'est donc pas une relation d'ordre (et donc encore moins un ordre bien fondé).

3. Oui, l'ensemble $(\Sigma^*, \sqsubseteq)$ est un ensemble ordonné, et cet ordre est total. En effet, soit u, v et w trois mots. On a bien $u \sqsubseteq u$ (avec $\phi = \mathrm{id}_{[\![0,|u|-1]\!]}$). Également, si $u \sqsubseteq v$ et $v \sqsubseteq u$, alors $|u| = |v|$, et par stricte croissance de ϕ, on a bien $u = v$. Aussi, si $u \sqsubseteq v$ et $v \sqsubseteq w$, alors soit ϕ l'extractrice de la suite v de u, et soit φ l'extractrice de la suite w de v. Alors, la fonction $\phi \circ \varphi$ est strictement croissante, et $\forall i \in [\![|u| - 1]\!]$, $u_i = v_{\phi(i)} = w_{\phi(\varphi(i))}$, et donc $u \sqsubseteq w$. Ainsi, la relation "$\sqsubseteq$" est une relation d'ordre.

   Montrons à présent que l'ordre est bien fondé. Soit L une partie non vide de Σ^*. Soit $x \in L$. Si $\varepsilon \in L$, alors $x \sqsupseteq \varepsilon$.
4. Non, l'ensemble $(\wp(E), \sqsubseteq)$ n'est pas un ensemble ordonné. En effet, on pose $E = \mathbb{N}$. Soit A une partie finie de E. On sait que son plus grand élément existe, et on le note m. Par définition du maximum, $\forall y \in A$, $y \preccurlyeq m$. Et donc $A \not\sqsubseteq A$. La relation "$\sqsubseteq$" n'est donc pas une relation d'ordre (et donc encore moins un ordre bien fondé).

TD 1.6 Définition inductive des mots et ordre préfixe

1. On pose $X_0 = \{\varepsilon\}$, et pour $n \in \mathbb{N}$,
$$X_{n+1} = X_n \cup \Big(\bigcup_{a \in \Sigma} \{a \cdot w \mid w \in X_n\}\Big).$$
   Ainsi, on définit par induction l'ensemble des mots Σ^*.
2. Soient u et v deux mots. Montrons $u \preccurlyeq_1 v \iff u \preccurlyeq_2 v$.

" $\Longrightarrow$ " Supposons $u \preccurlyeq_1 v$. Soit $w \in \Sigma^*$ tel que $v = uw$. Par définition de $\preccurlyeq_2$, on a bien $\varepsilon \preccurlyeq_2 w$. On décompose u en $u = u_1 \cdot u_2 \cdot \ldots \cdot u_n \cdot \varepsilon$ (avec la définition de mot de la question précédente). D'où, toujours par définition de $\preccurlyeq_2$, on a $u_n \cdot \varepsilon \preccurlyeq_2 u_n \cdot w$, puis $u_{n-1} \cdot u_n \cdot \varepsilon \preccurlyeq_2 u_{n-1} \cdot u_n \cdot w$. En itérant ce procédé, on obtient
$$\underbrace{u_1 \cdot u_2 \cdot \ldots \cdot u_n \cdot \varepsilon}_{u} \preccurlyeq_2 \overbrace{\underbrace{u_1 \cdot u_2 \cdot \ldots \cdot u_n}_{u} \cdot w}^{v}.$$
Et donc $u \preccurlyeq_2 v$.

" $\Longleftarrow$ " Supposons à présent que $u \preccurlyeq_2 v$. On pose $u = u_1 u_2 \ldots u_n$, et $v = v_1 v_2 \ldots v_m$. Par définition de $\preccurlyeq_2$, on a $u_1 \cdot (u_2 \ldots u_n) \preccurlyeq_2 u_1 \cdot (v_2 \ldots v_m)$. Puis, toujours par définition de $\preccurlyeq_2$, on a $u_1 \cdot u_2 \cdot (u_3 \ldots u_n) \preccurlyeq u_1 \cdot u_2 \cdot (v_3 \ldots v_n)$. En itérant ce procédé, on a $u \cdot \varepsilon \preccurlyeq u \cdot (v_n \ldots v_m)$. On pose $w = v_n \ldots v_m$, et on a bien $v = uw$. D'où $v \succcurlyeq_1 u$.

TD 1.7 $\mathcal{N}$

1. On définit par induction la fonction suivante
$$\begin{aligned} \oplus : \mathcal{N}^2 &\longrightarrow \mathcal{N} \\ (\boldsymbol{S}(x), y) &\longmapsto \oplus(x, \boldsymbol{S}(y)) \\ (\boldsymbol{0}, x) &\longmapsto x. \end{aligned}$$
2. Soit $(x, y) \in \mathcal{N}^2$.
   - Si $f(x) = 0$, alors $\oplus(x, y) = y$ et donc $f(\oplus(x, y)) = f(y) = f(x) + f(y)$.
   - Si $f(x) \geqslant 1$, alors $x = \boldsymbol{S}(z)$ avec $z \in \mathcal{N}$. Ainsi, $\oplus(x, y) = \oplus(z, \boldsymbol{S}(y))$. Or, $f(z) = f(x) - 1 \leqslant f(x)$. Et donc, par définition de $\oplus$ puis par hypothèse d'induction, on a $f(\oplus(x, y)) = f(\oplus(z, \boldsymbol{S}(y)) = f(z) + f(\boldsymbol{S}(y))$. On en déduit que $f(\oplus(x, y)) = f(x) - 1 + f(y) + 1 = f(x) + f(y)$.

   Par induction, on a bien $\forall (x, y) \in \mathcal{N}^2,\ f\big(\oplus(x, y)\big) = f(x) + f(y)$.
3. On définit par induction la fonction suivante
$$\begin{aligned} \otimes : \mathcal{N}^2 &\longrightarrow \mathcal{N} \\ (\boldsymbol{S}(x), y) &\longmapsto \oplus\big(y, \otimes(x, y)\big) \\ (\boldsymbol{0}, y) &\longmapsto \boldsymbol{0}. \end{aligned}$$

4. Soit $(x, y) \in \mathcal{N}^2$.
   - Si $f(x) = 0$, alors $\otimes(x, y) = \mathbf{0}$, et donc $f(\otimes(x, y)) = 0 = f(x) \times f(y)$.
   - Si $f(x) \geqslant 1$, alors $x = \boldsymbol{S}(z)$ avec $z \in \mathcal{N}$. Ainsi, par définition de $\otimes$, on a $\otimes(x, y) = \oplus(y, \otimes(z, y))$. Or, par hypothèse d'induction, $f(\otimes(z, y)) = f(z) \times f(y)$ (car $f(z) < f(x)$), et donc $f(\otimes(x, y)) = f(y) + f(\otimes(z, y)) = f(y) + f(z) \times f(y) = f(y) \times (1 + f(z)) = f(y) \times f(x)$.

   Par induction, on a bien $\forall (x, y) \in \mathcal{N}^2,\ f\big(\otimes(x, y)\big) = f(x) \times f(y)$.
5. On définit par induction la fonction suivante
$$\begin{aligned} ! : \mathcal{N} &\longrightarrow \mathcal{N} \\ \mathbf{0} &\longmapsto \boldsymbol{S}(\mathbf{0}) \\ \boldsymbol{S}(x) &\longmapsto \otimes\big(\boldsymbol{S}(x), !(x)\big). \end{aligned}$$
6. Soit $x \in \mathcal{N}$.
   - Si $f(x) = 0$, alors $!(x) = \boldsymbol{S}(\mathbf{0})$ par définition, et donc $f(!(x)) = 1 = 0! = f(x)!$.
   - Si $f(x) \geqslant 1$, alors $x = \boldsymbol{S}(z)$ avec $z \in \mathcal{N}$. Ainsi, par définition de $!$, on a $!(x) = \otimes(x, !(z))$, et donc, par hypothèse de récurrence, $f(!(x)) = f(x) \times f(!(z)) = f(x) \times \big(f(z)!\big)$. Or, comme $f(z) = f(x) - 1$, on a donc $f(!(x)) = f(x) \times \big(f(x) - 1\big)! = f(x)!$.

   Par induction, on a bien $\forall x \in \mathcal{N},\ f\big(!(x)\big) = f(x)!$.

TD 1.8 Résultats manquants du cours

TRAVAUX DIRIGÉS

2

LOGIQUE PROPOSITIONNELLE

TD 2.1 Logique avec `If`

TD 2.1.1 Représentatibilité des fonctions booléennes par formules de $\mathscr{F}_{\texttt{if}}$

1. On pose $G = \texttt{if}\ p\ \texttt{then}\ \top\ \texttt{else}\ (\underbrace{\texttt{if}\ q\ \texttt{then}\ \top\ \texttt{else}\ r}_{A})$, et on a

$$\begin{aligned}
[\![G]\!]^\rho &= [\![p]\!]^\rho \cdot [\![\top]\!]^\rho + \overline{[\![p]\!]^\rho} \cdot [\![A]\!]^\rho \\
&= \rho(p) + \overline{\rho(p)} \cdot \left([\![q]\!]^\rho \cdot [\![\top]\!]^\rho + \overline{[\![q]\!]^\rho} \cdot [\![r]\!]^\rho\right) \\
&= \rho(p) + \overline{\rho(p)} \cdot \left(\rho(q) + \overline{\rho(q)} \cdot \rho(r)\right) \\
&= \rho(p) + \overline{\rho(p)} \cdot \rho(q) + \overline{\rho(p)} \cdot \overline{\rho(q)} \cdot \rho(r) \\
&= \rho(p) + \rho(q) + \rho(r).
\end{aligned}$$

2.
$$[\![\texttt{if}\ C\ \texttt{then}\ G\ \texttt{else}\ H]\!]^\rho = \begin{cases} [\![G]\!]^\rho & \text{si } [\![C]\!]^\rho = \boldsymbol{V} \\ [\![H]\!]^\rho & \text{if } [\![C]\!]^\rho = \boldsymbol{F} \end{cases}.$$

3. Soit $G \in \mathbb{F}$.

   Cas 1 Soit $\mathscr{P} = \{p\}$

   - Sous-cas 1 : $f : \rho \mapsto \boldsymbol{V}$ est associée à $\top$.
   - Sous-cas 2 : la fonction dont la table de vérité est ci-dessous est associée à p.

p	f
$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{V}$

   - Sous-cas 3 : la fonction dont la table de vérité est ci-dessous est associée à $\bar{p}$.

p	f
$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$

   - Sous-cas 4 : $f : \rho \mapsto \boldsymbol{F}$ est associée à $\bot$.

Cas 2 Soit $\mathscr{P} = \{p_1, \dots, p_n\}$ et on pose

$$P_r : \text{“}\forall f : \mathbb{B}^{\{p_1,\dots,p_r\}} \to \mathbb{B},\ \exists G \in \mathscr{F}_{\mathtt{if}},\ [\![G]\!] = f.\text{”}$$

Soit $r \in [\![2, n]\!]$ et f une fonction booléenne définie sur $\mathbb{B}^{\{p_1,\dots,p_r\}}$ à valeurs dans $\mathbb{B}$. Soit

$$\begin{aligned} g : \mathbb{B}^{\{p_1,\dots,p_{r-1}\}} &\longrightarrow \mathbb{B} \\ \rho' &\longmapsto f\big(\rho' \uplus (p_r \mapsto \boldsymbol{V})\big). \end{aligned}$$

où $\uplus$ est défini comme dans l'exemple $(p \mapsto \boldsymbol{V}, q \mapsto \boldsymbol{F}) \uplus (r \mapsto \boldsymbol{V}) = (p \mapsto \boldsymbol{V}, q \mapsto \boldsymbol{F}, r \mapsto \boldsymbol{V})$. Soit alors G par hypothèse de récurrence tel que $[\![G]\!] = g$. Soit

$$\begin{aligned} h : \mathbb{B}^{\{p_1,\dots,p_{r-1}\}} &\longrightarrow \mathbb{B} \\ \rho' &\longmapsto f\big(\rho' \uplus (p_r \mapsto \boldsymbol{F})\big). \end{aligned}$$

Soit alors H par hypothèse de récurrence tel que $[\![H]\!] = h$.
On pose alors $A = \mathtt{if}\ p_r\ \mathtt{then}\ G\ \mathtt{else}\ H$. Montrons que $[\![A]\!] = f$. Soit $\rho \in \mathbb{B}^{\{p_1,\dots,p_r\}}$.

— Si $\rho(p_r) = \boldsymbol{V}$ alors

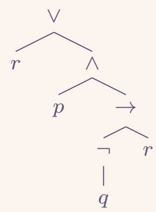
$$\begin{aligned} [\![A]\!]^\rho &= [\![G]\!]^\rho \\ &= [\![G]\!]^{\rho_{|\{p_1,\dots,p_{r-1}\}}} \\ &= g\left(\rho_{|\{p_1,\dots,p_{r-1}\}}\right) \\ &= f\left(\rho_{|\{p_1,\dots,p_{r-1}\}} \uplus (p_r \mapsto \boldsymbol{V})\right) \\ &= f(\rho) \end{aligned}$$

— Si $\rho(p_r) = \boldsymbol{F}$, alors

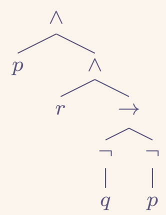
$$\begin{aligned} [\![A]\!]^\rho &= [\![H]\!]^\rho \\ &= [\![H]\!]^{\rho_{|\{p_1,\dots,p_{r-1}\}}} \\ &= h\left(\rho_{|\{p_1,\dots,p_{r-1}\}}\right) \\ &= f\left(\rho_{|\{p_1,\dots,p_{r-1}\}} \uplus (p_r \mapsto \boldsymbol{F})\right) \\ &= f(\rho) \end{aligned}$$

TD 2.2 Définitions de cours : syntaxe

1. On considère la formule $H_1 = r \vee (p \wedge (\neg q \to r))$. Son arbre syntaxique est

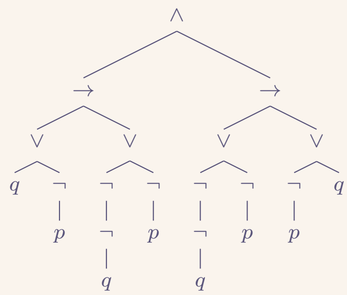


   Ses sous-formules sont $r \vee (p \wedge (\neg q \to r))$, r, $p \wedge (\neg q \to r)$, p, $\neg q \to r$, $\neg q$, et q. Ses variables sont p, q et r.
2. On considère la formule $H_2 = p \wedge (r \wedge (\neg q \to \neg p))$. Son arbre syntaxique est



Ses sous-formules sont $p \wedge (r \wedge (\neg q \to \neg p))$, p, $r \wedge (\neg q \to \neg p)$, r, $\neg q \to \neg p$, $\neg q$, q, et $\neg p$. Ses variables sont p, q et r.

3. On considère la formule $H_3 = ((q \vee \neg p) \to (\neg\neg q \vee \neg p)) \wedge ((\neg\neg q \vee \neg p) \to (\neg p \vee q))$. Son arbre syntaxique est



Ses sous-formules sont $((q \vee \neg p) \to (\neg\neg q \vee \neg p)) \wedge ((\neg\neg q \vee \neg p) \to (\neg p \vee q))$, $(q \vee \neg p) \to (\neg\neg q \vee \neg p)$, $(\neg\neg q \vee \neg p) \to (\neg p \vee q)$, $q \vee \neg p$, $\neg\neg q \vee \neg p$, $\neg p \vee q$, q, $\neg p$, $\neg\neg q$, p, et $\neg q$. Ses variables sont p et q.

TD 2.3 Formules duales

1. On définit par induction $(\cdot)^\star$ comme

— $\top^\star = \bot$;
— $\bot^\star = \top$;
— $(G \vee H)^\star = G^\star \wedge H^\star$;
— $(G \wedge H)^\star = G^\star \vee H^\star$;
— $(\neg G)^\star = \neg G^\star$;
— $p^\star = p$.

2. Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. Montrons, par induction, $P(H)$: "$[\![H^\star]\!]^\rho = [\![\neg H]\!]^{\bar\rho}$" où $\bar\rho : p \mapsto \overline{\rho(p)}$.

— On a $[\![\bot^\star]\!]^\rho = [\![\top]\!]^\rho = \boldsymbol{V}$, et $[\![\neg\bot]\!]^{\bar\rho} = [\![\top]\!]^{\bar\rho} = \boldsymbol{V}$, d'où $P(\bot)$.
— On a $[\![\top^\star]\!]^\rho = [\![\bot]\!]^\rho = \boldsymbol{F}$, et $[\![\neg\top]\!]^{\bar\rho} = [\![\bot]\!]^{\bar\rho} = \boldsymbol{F}$, d'où $P(\top)$.
— Soit $p \in \mathscr{P}$. On a $[\![p^\star]\!]^\rho = [\![p]\!]^\rho = \rho(p)$, et $[\![\neg p]\!]^{\bar\rho} = \overline{[\![p]\!]^{\bar\rho}} = \overline{\bar\rho(p)} = \overline{\overline{\rho(p)}} = \rho(p)$, d'où $P(p)$.

Soient F et G deux formules.

— On a

$$\begin{aligned} [\![(F \wedge G)^\star]\!]^\rho &= [\![F^\star \vee G^\star]\!]^\rho \\ &= [\![F^\star]\!]^\rho + [\![G^\star]\!]^\rho \\ &= [\![\neg F]\!]^{\bar\rho} + [\![\neg G]\!]^{\bar\rho} \\ &= [\![\neg F \vee \neg G]\!]^{\bar\rho} \\ &= [\![\neg (F \wedge G)]\!]^{\bar\rho} \end{aligned}$$

d'où $P(F \wedge G)$.

— On a

$$\begin{aligned}
[\![(F \vee G)^\star]\!]^\rho &= [\![F^\star \wedge G^\star]\!]^\rho \\
&= [\![F^\star]\!]^\rho \cdot [\![G^\star]\!]^\rho \\
&= [\![\neg F]\!]^{\bar\rho} \cdot [\![\neg G]\!]^{\bar\rho} \\
&= [\![\neg F \wedge \neg G]\!]^{\bar\rho} \\
&= [\![\neg (F \vee G)]\!]^{\bar\rho}
\end{aligned}$$

d'où $P(F \vee G)$.

— On a

$$[\![(\neg F)^\star]\!]^\rho = [\![\neg (F^\star)]\!]^\rho = \overline{[\![F^\star]\!]^\rho} = \overline{[\![\neg F]\!]^{\bar\rho}} = [\![\neg(\neg F)]\!]^{\bar\rho} \,.$$

d'où $P(\neg F)$.

Par induction, on en conclut que $P(F)$ est vraie pour toute formule F.

3. Soit G une formule valide. Alors, par définition, $G \equiv \top$. Or, d'après la question précédente, $G^\star \equiv (\top)^\star = \bot$. Ainsi, $G^\star$ n'est pas satisfiable.

TD 2.4 Conséquence sémantique

1. Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. On suppose $[\![A \vee B]\!]^\rho = \boldsymbol{V}$, $[\![A \to C]\!]^\rho = \boldsymbol{V}$ et $[\![B \to C]\!]^\rho = \boldsymbol{V}$.
   — Si $[\![A]\!]^\rho = \boldsymbol{V}$, alors, comme $[\![A \to C]\!]^\rho = \boldsymbol{V}$, $[\![C]\!]^\rho = \boldsymbol{V}$.
   — Si $[\![B]\!]^\rho = \boldsymbol{V}$, alors, comme $[\![B \to C]\!]^\rho = \boldsymbol{V}$, $[\![C]\!]^\rho = \boldsymbol{V}$.

   D'où $\{A \vee B, A \to C, B \to C\} \models C$.
2. Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. On suppose $[\![A \to B]\!]^\rho = \boldsymbol{V}$. Si $[\![B]\!]^\rho = \boldsymbol{F}$ (i.e. $[\![\neg B]\!]^\rho = \boldsymbol{V}$), alors $[\![A]\!]^\rho = \boldsymbol{F}$, par implication. Et donc, $[\![\neg A]\!]^\rho = \boldsymbol{V}$. On a donc bien $[\![\neg B \to \neg A]\!]^\rho$. On en déduit que $A \to B \models \neg B \to \neg A$.

3. oui
4. non
5. oui
6. non
7. oui
8. non
9. oui
10. non

TD 2.5 Axiomatisation algèbre de Boole

1. On pose, pour $t \in \mathbb{T}$, $P(t)$: "$t \simeq 0$ ou $t \simeq 1$," et on démontre cette propriété par induction.
   — On a bien $P(\mathbf{0})$ et $P(\mathbf{1})$.
   — Soient $t_1, t_2 \in \mathbb{T}$.
   (a) Si $t_1 \simeq \mathbf{1}$, alors $\bar t_1 \simeq \bar{\mathbf{1}} \simeq \bar{\bar{\mathbf{0}}} \simeq \mathbf{0}$; si $t_1 \simeq \mathbf{0}$, alors $\bar t_1 \simeq \bar{\mathbf{0}} \simeq \mathbf{1}$.
   (b) Si $t_1 \simeq \mathbf{0}$, alors $t_1 \cdot t_2 \simeq \mathbf{0} \cdot t_2 \simeq \mathbf{0}$; si $t_1 \simeq \mathbf{1}$, alors $t_1 \cdot t_2 \simeq \mathbf{1} \cdot t_2 \simeq t_2$ (qui est équivalent à $\mathbf{0}$ ou $\mathbf{1}$).
   (c) Si $t_1 \simeq \mathbf{1}$, alors $t_1 + t_2 \simeq \mathbf{1} + t_2 \simeq \mathbf{1}$; si $t_1 \simeq \mathbf{0}$, alors $t_1 + t_2 \simeq \mathbf{0} + t_2 \simeq t_2$ (qui est équivalent à $\mathbf{0}$ ou $\mathbf{1}$)
2. En reprenant les relations trouvées dans la question précédente, on construit les tables ci-dessous.

t_1	t_2	$t_1 \cdot t_2$
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

t_1	t_2	$t_1 + t_2$
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

t_1	$\bar t_1$
0	**1**
1	**0**

TD 2.6 Exercice 6 : Barre de Scheffer

1. — $\neg H_1 \equiv H_1 \texttt{ nand } H_1$;
   — $H_1 \wedge H_2 \equiv \neg(H_1 \texttt{ nand } H_2) \equiv (H_1 \texttt{ nand } H_1) \texttt{ nand } (H_2 \texttt{ nand } H_2)$;
   — $H_1 \vee H_2 \equiv \neg H_1 \texttt{ nand } \neg H_2 \equiv (H_1 \texttt{ nand } H_1) \texttt{ nand } (H_2 \texttt{ nand } H_2)$;
   — $H_1 \to H_2 \equiv H_1 \vee (\neg H_2) \equiv (H_1 \texttt{ nand } H_1) \texttt{ nand } H_2$;
   — $H_1 \leftrightarrow H_2 \equiv (H_1 \to H_2) \wedge (H_2 \to H_1)$.
2. — $\neg H_1 \equiv H_1 \texttt{ nor } H_1$;
   — $H_1 \vee H_2 \equiv \neg(H_1 \texttt{ nor } H_2) \equiv (H_1 \texttt{ nor } H_2) \texttt{ nor } (H_1 \texttt{ nor } H_2)$;
   — $H_1 \wedge H_2 \equiv (\neg H_1) \texttt{ nor } (\neg H_2) \equiv (H_1 \texttt{ nor } H_1) \texttt{ nor } (H_2 \texttt{ nor } H_2)$;
   — $H_1 \to H_2 \equiv H_1 \vee (\neg H_2) \equiv \neg(H_1 \texttt{ nor } (\neg H_2)) \equiv (H_1 \texttt{ nor } (H_2 \texttt{ nor } H_2)) \texttt{ nor } (H_1 \texttt{ nor } (H_2 \texttt{ nor } H_2))$;
   — $H_1 \leftrightarrow H_2 \equiv (H_1 \to H_2) \wedge (H_2 \to H_1)$.

TD 2.7 Énigmes en logique propositionnelle

TD 2.7.1 Fraternité

1. Ou les deux mentent, ou les deux disent la vérité. D'où $(A_1 \wedge C_1) \vee (\neg A_1 \wedge \neg C_1)$.
2. On a $A_1 = G \vee R$, et $C_1 = \neg G$.
3. On développe l'expression trouvée dans la question 1. :

$$\begin{aligned}(A_1 \wedge C_1) \vee (\neg A_1 \wedge \neg C_1) \equiv& \big((G \vee R) \wedge \neg G\big) \vee \big(\neg(G \vee R) \wedge \neg\neg G\big)\\ \equiv&(G \wedge \neg G \vee R \wedge \neg G) \vee \big((\neg G \wedge \neg R) \wedge G\big)\\ \equiv&(\bot \vee R \wedge \neg G) \vee (\neg G \wedge G \wedge \neg R)\\ \equiv&(R \wedge \neg G) \vee (\bot \wedge \neg R)\\ \equiv&(R \wedge \neg G) \vee \bot\\ \equiv&R \wedge \neg G.\end{aligned}$$

   La cérémonie se tiendra donc dans le réfectoire et non dans le gymnase.
4. $H = (A_2 \wedge B_2 \wedge C_2) \vee (\neg A_2 \wedge \neg B_2 \wedge \neg C_2)$.
5. $A_2 = E_1 \vee E_3$, $B_2 = E_2 \to \neg E_3$, $C_2 = E_1 \wedge \neg E_2$.
6.

E_1	E_2	E_3	A_2	B_2	C_2	H
F	**F**	**F**	**F**	**V**	**F**	**F**
F	**F**	**V**	**F**	**V**	**F**	**F**
F	**V**	**F**	**F**	**V**	**F**	**F**
F	**V**	**V**	**F**	**F**	**F**	**V**
V	**F**	**F**	**F**	**V**	**V**	**F**
V	**F**	**V**	**V**	**V**	**V**	**V**
V	**V**	**F**	**F**	**V**	**F**	**F**
V	**V**	**V**	**V**	**F**	**F**	**F**

   L'escalier 3 conduit bien à l'intronisation.
7. Si les trois mentent, alors on peut aussi prendre l'escalier 2.

TD 2.7.2 Alice au pays des merveilles

1. $I_R = \bar{J} \wedge B$, $I_J = \bar{R} \to \bar{B}$, et $I_B = B \wedge (\bar{R} \vee \bar{J})$.
2. Oui, la situation où le flacon jaune contient le poison ($\bar{J}$) satisfait ces formules.
3. Oui, on a $I_B \models I_R$.

4. Oui, les instructions sur le flacon rouge et le bleu sont fausses. En effet, le flacon jaune ne contient pas de poison, et il n'y a pas "au moins l'un des deux autres flacons contenant du poison."
5. Oui, comme vu précédemment, la configuration où le poison est seulement dans le flacon jaune est valide. Si le flacon rouge contient du poison ou le flacon bleu contient du poison, on a une contradiction avec l'une des instructions. La configuration trouvée précédemment est la seule valide.
6. À cette condition, deux autres configurations sont possibles : le poison est dans les flacons jaune et rouge, ou le poison est dans les flacons rouge et bleu.

TD 2.7.3 Socrate et Cerbère

1. On a $H = (I_1 \wedge I_2 \wedge I_3) \vee (\neg I_1 \wedge \neg I_2 \wedge \neg I_3)$.
2. On a $I_1 = C_1 \wedge C_3$, $I_2 = C_2 \to \bar{C}_3$, et $I_3 = C_1 \wedge \bar{C}_2$.
3.

C_1	C_2	C_3	I_1	I_2	I_3	H
F	***F***	***F***	***F***	***V***	***F***	***F***
V	***F***	***F***	***F***	***V***	***V***	***F***
F	***V***	***F***	***F***	***V***	***F***	***F***
V	***V***	***F***	***F***	***V***	***F***	***F***
F	***F***	***V***	***F***	***V***	***F***	***F***
V	***F***	***V***	***V***	***V***	***V***	***V***
F	***V***	***V***	***F***	***F***	***F***	***V***
V	***V***	***V***	***V***	***F***	***F***	***F***

   Socrate doit suivre le couloir 3.
4. En supposant que Cerbère ait menti, on suppose que les trois têtes ont menti, et donc que I_1, I_2 et I_3 sont fausses. On aurait aussi pu choisir le couloir 2.

TD 2.8 Compléments de cours, en vrac

1. *Montrer que $\models$ est une relation d'ordre sur $\mathscr{F}$.*
   - Soit F une formule. On sait que $F \models F$. En effet, pour tout $\rho \in \mathbb{B}^{\mathscr{P}}$, si $[\![F]\!]^\rho = \boldsymbol{V}$, alors $[\![F]\!]^\rho = \boldsymbol{V}$. La relation $\models$ est donc reflective.
   - Soient F et G deux formules. On suppose $F \models G$ et $G \models H$. Montrons que $F \equiv G$. On reprend la démonstration du cours : soit $\rho \in \mathbb{B}^{\mathscr{P}}$. On suppose $[\![G]\!]^\rho = \boldsymbol{V}$, alors $[\![F]\!] = \boldsymbol{V}$ car $G \models H$; et donc $[\![G]\!]^\rho = [\![F]\!]^\rho$. On suppose à présent que $[\![G]\!]^\rho = \boldsymbol{F}$, alors, par contraposée, $[\![F]\!]^\rho = \boldsymbol{F}$; on a donc $[\![G]\!]^\rho = [\![F]\!]^\rho$. On en déduit que $[\![G]\!] = [\![F]\!]$, i.e. $G \equiv F$. La relation est donc *quasi*–anti-symétrique.
   - Soient F, G et H trois formules. On suppose $F \models G$ et $G \models H$. Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. Si $[\![F]\!]^\rho = \boldsymbol{V}$, alors $[\![G]\!]^\rho = \boldsymbol{V}$. Or, comme $G \models H$, si $[\![G]\!]^\rho = \boldsymbol{V}$, alors $[\![H]\!]^\rho = \boldsymbol{V}$. D'où $[\![F]\!]^\rho = \boldsymbol{V} \implies [\![H]\!]^\rho = \boldsymbol{V}$. On a donc $F \models H$. La relation $\models$ est donc transitive.
2. Soit $A \in \mathscr{F}$, soit $\rho \in \mathbb{B}^{\mathscr{P}}$, et soit $\sigma \in \mathscr{P}^{\mathscr{F}}$. On pose, pour $p \in \mathscr{P}$, $\tau(p) = [\![\sigma(p)]\!]^\rho$. *Montrons que $[\![A[\sigma]]\!]^\rho = [\![A]\!]^\tau$.*

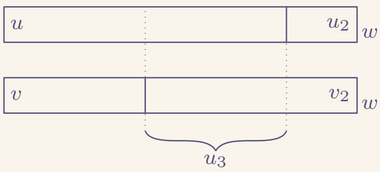
TRAVAUX DIRIGÉS

3

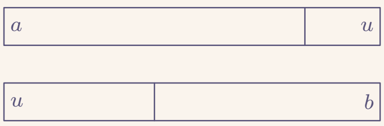
LANGAGES ET EXPRESSIONS RÉGULIÈRES

TD 3.1 Propriétés sur les mots

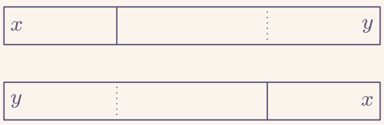
1. Soit $u_2, v_2 \in \Sigma^*$ tels que $w = uu_2$ et $w = vv_2$. Si $|u_2| = |v_2|$, alors $u = v = v\varepsilon$ donc v est préfixe de u. Si $|u_2| < |v_2|$, u_2 est suffixe de v_2. Soit $u_3 \in \Sigma^*$ tel que $v_2 = u_2u_3$. Ainsi, $w = vu_3u_2 = uu_2$, d'où $u = vu_3$. On en déduit que v est un préfixe de u. Similairement, si $|u_2| > |v_2|$, par symétrie du problème, en inversant u et v, puis u_2 et v_2, on se trouve bien dans le cas précédent. Ainsi, on a bien u est un préfixe de v.



2. Soit $u = u_1 \dots u_n$ avec, pour tout $i \in [\![1, n]\!]$, $u_i \in \Sigma$. Or, $au = ub$ donc $au_1 \dots u_n = u_1 \dots u_n b$ donc, pour tout $i \in [\![1, n-1]\!]$, $u_i = u_{i+1}$. Or, $u_1 = a$. De proche en proche, on a $\forall i \in [\![1, n]\!]$, $u_i = a$. Or, $u_n = b$ et donc $a = b$. On en déduit également que $u \in a^*$.



3. La suite de la correction de cet exercice est disponible sur *cahier-de-prepa*.



TD 3.2 Une équivalence sur les mots

La correction de cet exercice est disponible sur *cahier-de-prepa*.

TD 3.3 Langages

La correction de cet exercice est disponible sur *cahier-de-prepa*.

TD 3.4 Propriétés sur les opérations régulières

1. On a
$$\varnothing^* = \{\varepsilon\}\ ; \qquad \varnothing \cdot A = \varnothing\ ; \qquad \{\varepsilon\} \cdot A = A.$$

2. *(1)* On procède par double-inclusion.

“$\subseteq$” Soit $w \in (A \cdot B) \cdot C$. On pose $w = u \cdot v$ avec $u \in A \cdot B$ et $v \in C$. On pose ensuite $u = x \cdot y$ avec $x \in A$ et $y \in B$. Or, comme l'opération “$\cdot$,” pour les mots, est associative, on a bien $w = (x \cdot y) \cdot v = x \cdot (y \cdot v)$, et donc $w \in A \cdot (B \cdot C)$.

“$\supseteq$” Soit $w \in A \cdot (B \cdot C)$. On pose $w = u \cdot v$ avec $u \in A$ et $v \in B \cdot C$. On pose ensuite $v = x \cdot y$ avec $x \in B$ et $y \in C$. Or, comme l'opération “$\cdot$,” pour les mots, est associative, alors $w = u \cdot (x \cdot y) = (u \cdot x) \cdot y$ et donc $w \in A \cdot (B \cdot C)$.

(2) On suppose $A \subseteq B$. On a donc $B = A \cup (B \setminus A)$, et par définition $A^* = \bigcup_{n \in \mathbb{N}} A^n$, et $B^* = \bigcup_{n \in \mathbb{N}} B^n$. Montrons par récurrence, pour $n \in \mathbb{N}$, $P(n)$: “$A^n \subseteq B^n$.”

— On a $A^0 = \{\varepsilon\} \subseteq B^0 = \{\varepsilon\}$ d'où $P(0)$.

— Soit $n \in \mathbb{N}$ tel que $A^n \subseteq B^n$. On a $A^{n+1} = A^n \cdot A$ et $B^{n+1} = B^n \cdot B$. Or, comme $A^n \subseteq B^n$ et $A \subseteq B$, et que “$\cdot$”est croissant (dans l'inclusion), on en déduit que $A^{n+1} \subseteq B^{n+1}$. D'où $P(n+1)$.

(3) On procède par double-inclusion.

“$\supseteq$” On a $A^* = (A^*)^1 \subseteq \bigcup_{n \in \mathbb{N}} (A^*)^n = (A^*)^*$.

“$\subseteq$” Soit $w \in (A^*)^*$. On pose donc $w = u_1 \dots u_n$ avec, pour tout $i \in [\![1, n]\!]$, $u_i \in A^*$. On pose également, pour tout $i \in [\![1, n]\!]$, $u_i = v_{i,1} \dots v_{i,m_i}$ où, pour tout $j \in [\![1, m_i]\!]$, $v_{i,j} \in A$. D'où, $w = v_{11} \dots v_{1,m_1} v_{21} \dots v_{2,m_2} \dots v_{n,m_n} \in A^*$. On en déduit que $(A^*)^* \subseteq A^*$.

(4) On procède par double-inclusion.

“$\subseteq$” On a $\{\varepsilon\} \subseteq A^*$ et donc $A^* = A^* \cdot \{\varepsilon\} \subseteq A^* \cdot A^*$. D'où $A^* \subseteq A^* \cdot A^*$.

“$\supseteq$” Soit $w \in A^* \cdot A^*$. On décompose ce mot : soient $u_1, u_2 \in A^*$ tels que $w = u_1 \cdot u_2$. On pose $n = |u_1|$, et $m = |u_2|$. On décompose également ces deux mots : soient $(w_1, w_2, \dots, w_n) \in A^n$ et $(w_{n+1}, w_{n+2}, \dots, w_{n+m}) \in A^m$ tels que $u_1 = w_1 \cdot w_2 \cdot \dots \cdot w_n$ et $u_2 = w_{n+1} \cdot w_{n+2} \cdot \dots \cdot w_{n+m}$. Ainsi,
$$w = w_1 \cdot w_2 \cdot \dots \cdot w_n \cdot w_{n+1} \cdot \dots \cdot w_{n+m} \in A^*.$$
D'où $A^* \cdot A^* \subseteq A^*$.

(5) On procède par double-inclusion.

“$\subseteq$” Soit $w \in A \cup B$.

— Si $w \in A$, alors $w \in A^*$, et donc $w = w \cdot \varepsilon \in A^* \cdot B^*$.

— Si $w \in B$, alors $w \in B^*$, et donc $w = \varepsilon \cdot w \in A^* \cdot B^*$.

On a donc bien $A \cup B \subseteq A^* \cdot B^*$, et par croissance de l'étoile, on a bien $(A \cup B)^* \subseteq (A^* \cdot B^*)^*$.

“$\supseteq$” Soit $w \in (A^* \cdot B^*)^*$. On pose
$$\begin{aligned} w = \ & u_{11} \dots u_{1,n_1} v_{11} \dots v_{1,m_1} \\ & \cdot u_{21} \dots u_{2,n_2} v_{21} \dots v_{2,n_2} \\ & \vdots \\ & \cdot u_{p,1} \dots u_{p,n_p} v_{p,1} \dots v_{p,m_p} \end{aligned}$$
où, $u_{i,j} \in A$ et $v_{i,j} \in B$. On a donc $w \in (A \cup B)^*$.

(6) On procède par double-inclusion.

"$\subseteq$" Soit $w \in A \cdot (B \cup C)$. On pose $w = u \cdot v$ avec $u \in A$ et $v \in B \cup C$.

— Si $v \in B$, alors $w = u \cdot v \in A \cdot B$ et donc $w \in (A \cdot B) \cup (A \cdot C)$.

— Si $v \in C$, alors $w = u \cdot v \in A \cdot C$ et donc $w \in (A \cdot B) \cup (A \cdot C)$.

On a bien montré $A \cdot (B \cup C) \subseteq (A \cdot B) \cup (A \cdot C)$.

"$\supseteq$" Soit $w \in (A \cdot B) \cup (A \cdot C)$.

— Si $w \in A \cdot B$, on pose alors $w = u \cdot v$ avec $u \in A$ et $v \in B \subseteq B \cup C$. Ainsi, on a bien $w = u \cdot v \in A \cdot (B \cup C)$.

— Si $w \in A \cdot C$, on pose alors $w = u \cdot v$ avec $u \in A$ et $v \in C \subseteq B \cup C$. Ainsi, on a bien $w = u \cdot v \in A \cdot (B \cup C)$.

On a bien montré $A \cdot (B \cup C) \supseteq (A \cdot B) \cup (A \cdot C)$.

3. *(1)* Soit $A = \{a\}$ et $B = \{b\}$ avec $a \neq b$. On sait que $abab \in (A \cdot B)^*$. Or, $abab \not\in A^* \cdot B^*$ donc $L_1 \not\subseteq L_2$. De plus, $a \in A^* \cdot B^*$ et $a \not\in (A \cdot B)^*$ donc $L_2 \not\subseteq L_1$. Il n'y a aucune relation entre L_1 et L_2.

(2) On sait que $(A \cdot B)^* \subseteq (A^* \cdot B^*)^*$ (car $A \cdot B \subseteq A^* \cdot B^*$ et par croissance de l'étoile). Mais, $(A \cdot B)^* \not\supseteq (A^* \cdot B^*)^*$. En effet, avec $A = \{a\}$ et $B = \{b\}$ où $a \neq b$, on a $ba \in (A^* \cdot B^*)^*$ (d'après la question précédente) mais $ba \not\in (A \cdot B)^*$. On a donc seulement $L_1 \subseteq L_2$.

(3) On a $L_1 \subseteq L_2$. En effet, $A \cap B \subseteq B$ donc $(A \cap B)^* \subseteq B^*$ par croissance l'étoile. De même, $A \cap B \subseteq A$ donc $(A \cap B)^* \subseteq A^*$. D'où $(A \cap B)^* \subseteq A^* \cap B^*$. Mais, $L_1 \not\supseteq L_2$. En effet, avec $A = \{a\}$ et $B = \{aa\}$, on a $A \cap B = \varnothing$ et donc $L_1 = (A \cap B)^* = \{\varepsilon\}$, mais, $L_2 = A^* \cap B^* = B^*$ (car $A^* \subseteq B^*$), et donc $L_2 \not\subseteq L_1$.

(4) Comme $A^* \subseteq (A \cup B^*)$ et $B^* \subseteq (A \cup B)^*$, alors $A^* \cup B^* \subseteq (A \cup B)^*$. Mais, $A^* \cup B^* \not\supseteq (A \cup B)^*$. En effet, si $A = \{a\}$ et $B = \{b\}$ où $a \neq b$, alors on a $ba \in (A \cup B)^*$ mais $ba \not\in A^* \cup B^*$. On a donc seulement $L_1 \subseteq L_2$.

(5) On a $L_1 \subseteq L_2$. En effet, soit $w \in A \cdot (B \cap C)$. On pose $w = u \cdot v$ avec $u \in A$ et $v \in B \cap C$. Comme $v \in B$, alors $w = u \cdot v \in A \cdot B$. De même, comme $v \in C$, alors $w = u \cdot v \in A \cdot C$. On a donc bien $w \in (A \cdot B) \cap (A \cdot C)$. D'où $L_1 \subseteq L_2$. Mais, $L_1 \not\supseteq L_2$. En effet, avec $A = \{a, aa\}$, $B = \{b\}$ et $C = \{ab\}$ où $a \neq b$, on a $aab \not\in B \cap C = \varnothing$ mais, $aab \in A \cdot B$ et $aab \in A \cdot C$, donc $aab \in L_2$. On a donc seulement $L_1 \subseteq L_2$.

(6) On a, d'après la question 2.

$$L_1 = (A^* \cup B)^* = \big((A^*)^* \cdot B^*\big)^* = (A^* \cdot B^*)^* = (A \cup B)^* = L_2.$$

TD 3.5 Habitants d'expressions régulières

1. *(1)* Les mots de taille 1, 2, 3 et 4 de $\big((ab)^* \mid a\big)^*$ sont a, aa, ab, aaa, $aaaa$, $abab$, aba, $abaa$, aab, $aaab$ et $aaba$.

(2) On sait, tout d'abord, que l'expression régulière $\big(a \cdot \big((b \cdot b)^* \mid (a \cdot \varnothing)\big) \cdot b\big) \mid \varepsilon$ est équivalente à $\big(a \cdot (bb)^* \cdot b\big)$. Les mots de taille 1, 2, 3 et 4 sont donc $abbb$ et ab.

2. *(1)* Les mots de taille 1, 2 et 3 de $(a \mid b)^* \cdot (a \mid c)^*$ sont a, b, c, aa, bb, cc, ab, ac, bc, ba, ca, aaa, bbb, ccc, aac, aab, bbc, bba, cca, aba, abc, baa, aca, acc, abb, bca, bcc, cac, bab et caa.

(2) Les mots de taille 1, 2 et 3 de $(a \cdot b)^* \mid (a \cdot c)^*$ sont ab et ac.

TD 3.6 Regexp Crossword

`https://regexcrossword.com/`

TD 3.7 Description d'automates au moyen d'expression régulières

1. $(a \mid b)^* \cdot a \cdot b \cdot b \cdot a \cdot (a \mid b)^*$;
2. $a^* \cdot a \cdot b^*$;
3. $\big(a \cdot (ab)^*\big) \mid \big(a \cdot a \cdot (ba)^*\big)$;
4. $(aa) \cdot (aa)^*$.

TD 3.8 Vocabulaire des automates

On représente, ci-dessous, l'automate $\mathscr{A}$ décrit dans l'énoncé.

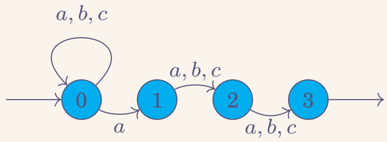


FIGURE TD 3.1 – Automate décrit dans l'énoncé de l'exercice 8

1. Cet automate n'est pas complet : à l'état 0, la lecture d'un a peut conduire à l'état 0 ou bien à l'état 1.
2. Le mot *baba* est reconnu par $\mathscr{A}$ mais pas le mot *cabcb*.
3. L'automate reconnaît les mots dont la 3ème lettre du mot, en partant de la fin, est un a.

TD 3.9 Complétion d'automate

1. Non, cet automate n'est pas complet. Par exemple, la lecture d'un b à l'état 1 est impossible.
2. Cet automate reconnaît le langage $L = \mathscr{L}\big(a \cdot b \cdot (a \mid b)^*\big)$.
3.

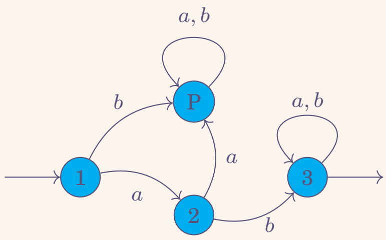
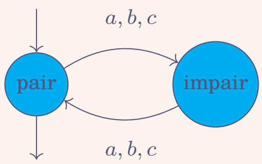
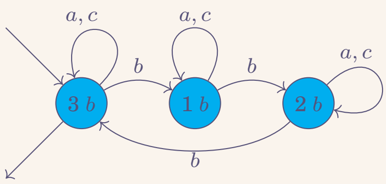
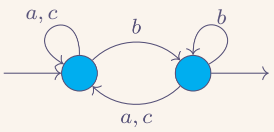


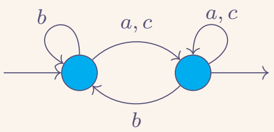
FIGURE TD 3.2 – Automate complet équivalent à $\mathscr{A}$

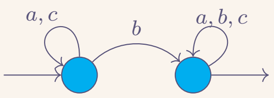
TD 3.10 Construction d'automates

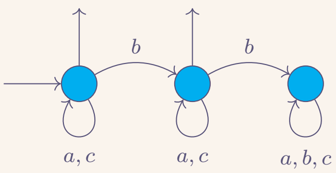
1. 

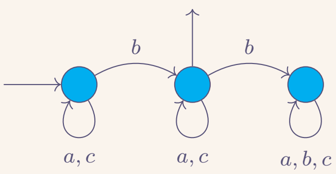
2. 

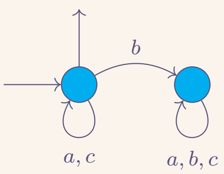
3. 

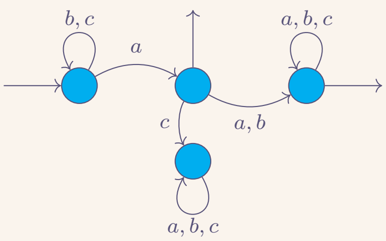
4. 

5. 

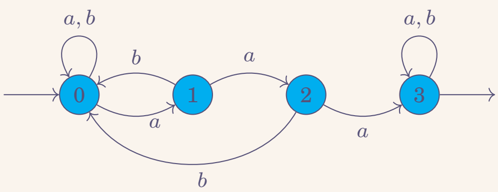
6. 

7. 

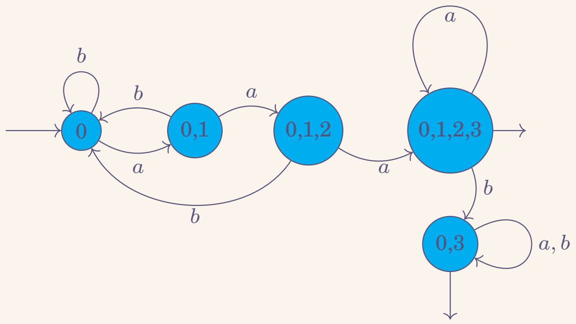
8. 

9. 

TD 3.11 Déterminisation 1

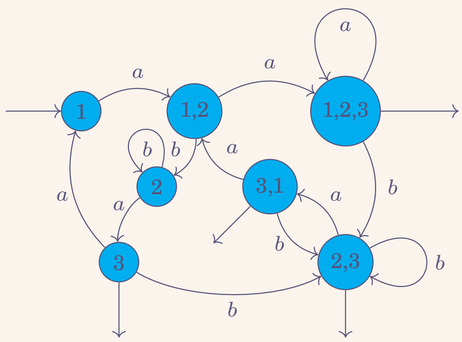
1. 

2.

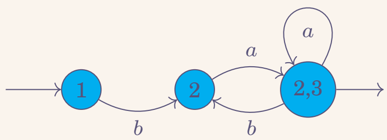


TD 3.12 Déterminisation 2

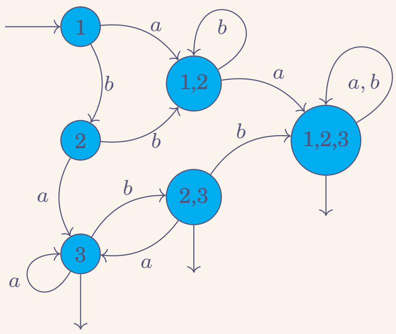
1.



2.



3.



TD 3.13 Exercice supplémentaire 1

1. *Montrer que l'ensemble des langages reconnaissables est stable par complémentaire.*
2. *Montrer que l'ensemble des langages reconnaissables est stable par intersection.*

1. Soient $\mathscr{A} = (\Sigma, \mathbb{Q}, I, F, \delta)$ et $\mathscr{A}' = (\Sigma, \mathbb{Q}', I', F', \delta')$ deux automates déterministes complets, tels que $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$. Alors

$$\mathscr{L}\big(\Sigma, \mathbb{Q}', I', \mathbb{Q}' \setminus F', \delta'\big) = \Sigma^* \setminus \mathscr{L}(\mathscr{A}).$$

2. On utilise les lois de De Morgan en passant au complémentaire les deux automates, puis l'union (que l'on a vu en cours), et on repasse au complémentaire.

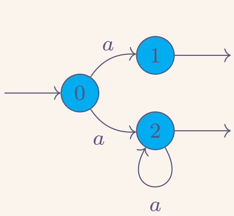
TRAVAUX DIRIGÉS

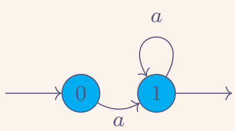
4

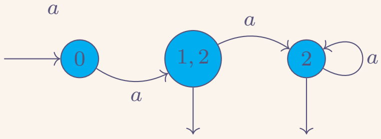
LANGAGES ET EXPRESSIONS RÉGULIÈRES (2)

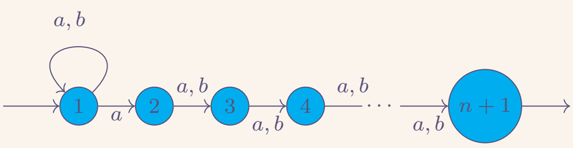
TD 4.1 Déterminisation de taille exponentielle

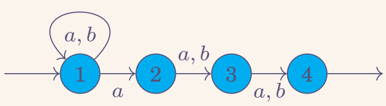
1. En notant n le nombre d'états de $\mathscr{A}$, alors le nombre d'états de $\det(\mathscr{A})$ est, au plus, 2^n. En effet, les états sont des éléments de $\wp(Q)$ et $|\wp(Q)| = 2^n$.

2. $\mathscr{A}$: 

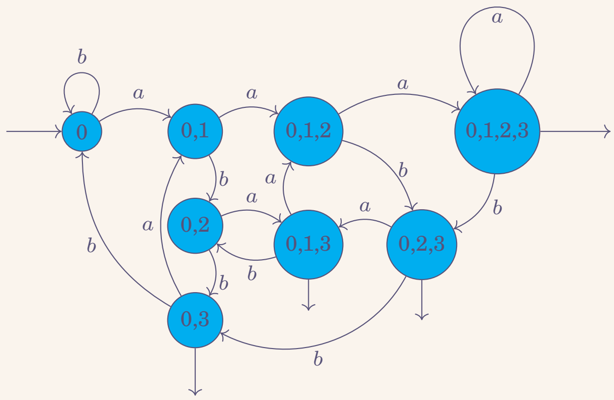
$\mathscr{B}$: 

   mais $\det(\mathscr{A})$: 

3. $\mathscr{A}_n$: 

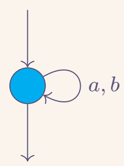
4. $\mathscr{A}_3$: 

et, $\det(\mathscr{A}_3)$:



5. Soit $i_0 = \max\{k \in [\![1,n]\!] \mid u_k \neq v_k\}$. Soit $m \in \Sigma^{i_0}$ tel que $u \cdot m \in L_n$ mais $v \cdot m \notin L_n$. Or, $\delta^*(i, u \cdot m) = \delta^*(\delta^*(i,u), m)$ et $\delta^*(i, v \cdot m) = \delta^*(\delta^*(i,v), m)$. D'où $\delta^*(i, u \cdot m) \in F$ et $\delta^*(i, v \cdot m) \notin F$. Ce qui est absurde.
6. Ainsi, l'application
$$\begin{aligned} f : \Sigma^* &\longrightarrow Q \\ u &\longmapsto \delta^*(i,u) \end{aligned}$$
est injective. D'où, $\mathscr{D}_n = |Q| \geqslant |\Sigma^*| = 2^n$.
7. D'où, d'après les questions 1 et 6, on en déduit que le nombre d'états utilisés pour la déterminisation de $\mathscr{A}_n$ est de $\mathscr{D}_n \geqslant 2^n$.

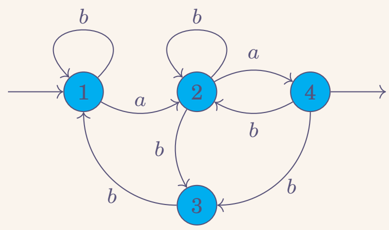
TD 4.2 Suppression des ε-transitions



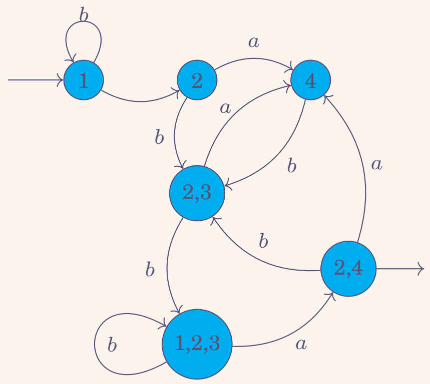
TD 4.3 Déterminisation d'automates avec ε-transitions

Pour les deux automates, on commence par supprimer les ε-transitions, puis on le déterminise.

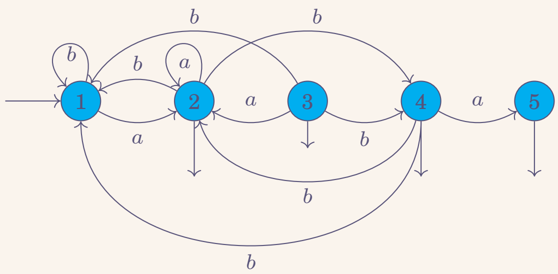
1. L'automate équivalent sans ε-transitions est le suivant.



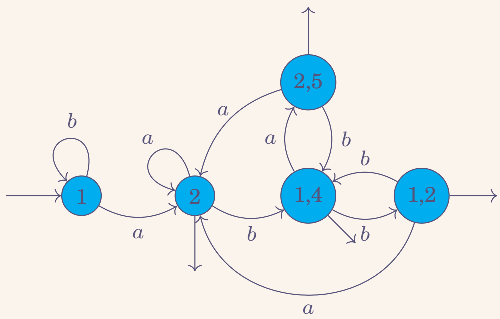
Une fois déterminisé, on obtient l'automate ci-dessous.



2. L'automate équivalent, sans ε-transitions, est le suivant.



Une fois déterminisé, on obtient l'automate ci-dessous.

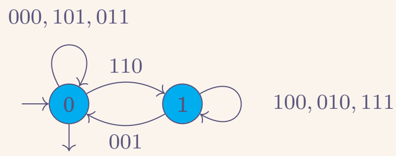


TD 4.4 Automates pour le calcul de modulo

TD 4.5 Automates pour le calcul de l'addition en binaire

TD 4.5.1 Nombres de même tailles

Q. 1



Q. 2 Pour $r \in \{0, 1\}$, il existe une exécution dans $\mathscr{A}$ étiquetée par

$$(u_0, v_0, w_0)(u_1, v_1, w_1) \dots (u_{n-1}, v_{n-1}, w_{n-1})$$

menant à r si et seulement si

$$\overline{u_0 \dots u_{n-1}}^2 + \overline{v_0 \dots v_{n-1}}^2 = \overline{w_0 \dots w_{n-1}}^2 + r\, 2^n,$$

ce qui est équivalent à si et seulement si

$$\overline{u_0 \dots u_{n-1} 0}^2 + \overline{v_0 \dots v_{n-1} 0}^2 = \overline{w_0 \dots w_{n-1} r}^2.$$

Q. 3 Prouvons-le par récurrence.

— Pour $n = 0$, il existe une exécution dans $\mathscr{A}$ étiquetée par ε menant à $r = 0$ si et seulement si $\overline{\varepsilon}^2 + \overline{\varepsilon}^2 = 0 = \overline{\varepsilon}^2 + 0 \times 2^0$. De même, il existe une exécution dans $\mathscr{A}$ étiquetée par ε menant à $r = 1$ si et seulement si $\overline{\varepsilon}^2 + \overline{\varepsilon}^2 = 0 = 1 = \overline{\varepsilon}^2 + 1 \times 2^0$.

TRAVAUX DIRIGÉS

5

LANGAGES ET EXPRESSIONS RÉGULIÈRES (3)

TD 5.1 Exercice 5

1. On a $e = a(ab \mid b^*) \mid a$, $f = a_1(a_2b_1 \mid b_2^*) \mid a_3$et $f_\varphi = e$ où

$$\varphi : \begin{pmatrix} \forall i,\ a_i & \longmapsto & a \\ \forall i,\ b_i & \longmapsto & b \end{pmatrix}.$$

D'où

	Λ	P	S	F
a_1	$\varnothing$	a_1	a_1	$\varnothing$
a_2	$\varnothing$	a_2	a_2	$\varnothing$
b_2^*	ε	b_2	b_2	b_1b_2
$a_2b_1 \mid b_2^*$	ε	a_2, b_2	b_1, b_2	a_2b_1, b_2b_2
a_3	$\varnothing$	a_3	a_3	$\varnothing$
$a_1(a_2b_1 \mid b_2^*)$	$\varnothing$	a_1	b_1, b_2, a_1	$a_1a_2, a_1b_2, a_2b_1, b_2b_2$
f	$\varnothing$	a_1, a_3	b_1, b_2, a_3, a_1	$a_1a_2, a_1b_2, a_3b_1, b_2b_2$

Automate à faire...

2. On pose $e = (\varepsilon \mid a)^* \cdot ab \cdot (a \mid b)^*$ et $f = (\varepsilon \mid a_1)^* \cdot a_2b_1 \cdot (a_3 \mid b_2)^*$ et

$$\varphi : \begin{pmatrix} \forall i,\ a_i & \longmapsto & a \\ \forall i,\ b_i & \longmapsto & b \end{pmatrix}$$

d'où $f_\varphi = e$.

TD 5.2 Exercice 4

Q. 1

> **Algorithme:** *Entrée* : Un automate $\mathscr{A}$;
> *Sortie* : $\mathscr{L}(\mathscr{A}) = \varnothing$;
> On fait un parcours en largeur depuis les états initiaux et on regarde si on atteint un état final.

> **Algorithme** (Nathan F.)**:** *Entrée* : Deux automates $\mathscr{A}$ et $\mathscr{B}$
> *Sortie* : $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{B})$; Soit $\mathscr{C}$ l'automate reconnaissant $\mathscr{L}(\mathscr{A}) \triangle \mathscr{L}(\mathscr{B})$. On retourne $\mathscr{L}(\mathscr{C}) \stackrel{?}{=} \varnothing$ à l'aide de l'algorithme précédent.

Autre possibilité, on procède par double inclusion :

> **Algorithme** ($\subseteq$)**:** *Entrée* : Deux automates $\mathscr{A}$ et $\mathscr{B}$
> *Sortie* : $\mathscr{L}(\mathscr{A}) \subseteq \mathscr{L}(\mathscr{B})$; On retourne $\mathscr{A} \setminus \mathscr{B} \stackrel{?}{=} \varnothing$.

Q. 2 L'algorithme reconnaissant $\mathscr{L}(\mathscr{A}) \triangle \mathscr{L}(\mathscr{B})$ doit être déterminisé, sa complexité est donc au moins de 2^n.

TD 5.3 Exercice 6 : Langages reconnaissables ou non

Q. 7 *Le carré d'un langage est le langage $L_2 = \{u \cdot u \mid u \in L\}$. Si L est reconnaissable, L_2 est-il nécessairement reconnaissable ?*

Avec $\Sigma = \{a, b\}$, soit $L = \mathscr{L}(a^* \cdot b^*)$. On a donc $L_2 = \{a^n \cdot b^m \cdot a^n \cdot b^m \mid (n, m) \in \mathbb{N}^2\}$. Supposons L_2 reconnaissable. Soit $\mathscr{A}$ un automate à n états reconnaissant L_2. On pose $u = a^{2n} \cdot b^n \cdot a^{2n} \cdot b^n \in L_2$. D'après le lemme de l'étoile, il existe $(x, y, z) \in (\Sigma^*)^3$ tel que $u = x \cdot y \cdot z$, $|xy| \leqslant n$, $\mathscr{L}(x \cdot y^* \cdot z) \subseteq L_2$, et $y \neq \varepsilon$. Ainsi, il existe $m \in [\![1, n]\!]$ et $p \in [\![1, n]\!]$ tels que $y = a^m$, $x = a^p$ et $z = a^{2n-m-p} \cdot b^n \cdot a^{2n} \cdot b^n$. Et alors, $x \cdot y^2 \cdot z = a^p \cdot a^{2m} \cdot a^{n-m-p} \cdot b^n \cdot a^{2n} \cdot b^n = a^{2n+m} \cdot b^n \cdot a^{2n} \cdot b^n \notin L_2$.

Q. 5 *Le langage $L_5 = \{a^{n^3} \mid n \in \mathbb{N}\}$ est-il reconnaissable ?* Soit $\mathscr{A}$ un automate à N états, et soit $u = a^{N^3}$. D'après le lemme de l'étoile, il existe $(x, y, z) \in (\Sigma^*)^3$ tel que $u = x \cdot y \cdot z$, $|xy| \leqslant N$, $\mathscr{L}(x \cdot y^* \cdot z) \subseteq L_5$ et $y \neq \varepsilon$. D'où $x \cdot y^0 \cdot z \in L$, et donc $a^{N^3 - i} \in L$, avec $i \leqslant N$. Or, $\forall k \in \mathbb{N},\ N^3 - i \neq k^3$, ce qui est absurde.

TRAVAUX DIRIGÉS

6 ALGORITHMES PROBABILISTES

TD 6.1 Exercice 1 : Vérification d'égalité polynomiale

1. Étant donnés deux tableaux représentant deux polynômes, on peut calculer leurs produit en concaténant ce tableau. La complexité du produit de polynômes avec cet algorithme est en $\mathcal{O}(nm)$ où n est le degré du 1er polynôme, et m est le degré du second. En effet, *dans le pire des cas*, tous les polynômes représentant les deux polynômes sont des monômes, or, la concaténation étant en $\mathcal{O}(nm)$ (pour un tableau de taille n et un de taille m). D'où la complexité en $\mathcal{O}(nm)$.
2. Afin d'évaluer ces polynômes, on utilise l'algorithme de Horner, qui est en $\mathcal{O}(n)$, donc en temps linéaire.
3. En développant ces polynômes, la complexité serait en $\mathcal{O}(n^3)$. En effet, la multiplication de deux polynômes de degrés n a une complexité en $\mathcal{O}(n^2)$. D'où la complexité en $\mathcal{O}(n^3)$ pour la multiplication de deux polynômes ayant chacun un degré n.
4. Un polynôme de degré n a, au plus, n racines. D'où, le polynôme $P - Q$, a au plus n racines (où $n = \max(\deg P, \deg Q)$). Ainsi, s'il a $n + 1$ racines, c'est alors le polynôme nul, et donc $P = Q$.

Algorithme TD 6.11 Algorithme déterministe pout tester l'égalité polynomiale en $\mathcal{O}(n^2)$

Entrée : $P = (P_i)_{i \in [\![1,m]\!]}$ et $Q = (Q_j)_{j \in [\![1,p]\!]}$ deux polynômes
$n \leftarrow \deg P$
pour $i \in [\![0, n]\!]$ **faire**
  si $P(i) \neq Q(i)$ **alors** ▷ *Avec l'algorithme de Horner, évaluation en $\mathcal{O}(n)$*
    retourner Non
retourner Oui

5.

Algorithme TD 6.12 Algorithme probabiliste pout tester l'égalité polynomiale en $\mathcal{O}(n)$

Entrée $P = (P_i)_{i \in [\![1,n]\!]}$ et $Q = (Q_j)_{j \in [\![1,n]\!]}$ deux polynômes, et $k \in \mathbb{N}$ un entier
$x \leftarrow \mathcal{U}([\![1, k \times n]\!])$
si $P(x) \neq Q(x)$ **alors**
  retourner Non
retourner Oui

Soit X la variable aléatoire de $\mathcal{U}(\llbracket 1, k \times n \rrbracket)$. L'événement "$P \neq Q$ mais l'algorithme retourne Oui" arrive si $X \in \{j \in \llbracket 1, kn \rrbracket \mid P(j) = Q(j)\} = A$. Or $|A| \leqslant n$, et $A \subseteq \llbracket 1, kn \rrbracket$. Ainsi, l'événement a une probabilité de $\frac{1}{k}$.

TD 6.2 Test de primalité probabiliste

TD 6.2.1 Résultats mathématiques

1. — Élément neutre : soit $x \in G_n$, d'où $x \cdot 1 = 1 \times x \bmod n = x \bmod n$, et donc $1 \in G_n$ est l'élément neutre de G_n.
   — Associativité : par associativité de $\times$, et par le fait que "mod" soit une congruence, on en conclut que $\cdot$ est associative.
   — Soient $x, y \in G_n$. Ainsi, $x \cdot y = x \times y \bmod n$. Or, $x \times y \wedge n = 1$, et donc $x \cdot y \wedge n = 1$.
   — Soit $x \in G_n$, donc $x \wedge n = 1$. D'où, d'après le théorème de Bézout, il existe u et $v \in \mathbb{Z}$ deux entiers tels que $u \times x + v \times n = 1$. D'où $1 \bmod n = u \times x + v \times n \bmod n$ et donc $1 = u \times x \bmod n$. Ainsi $x^{-1} = u \in G_n$, car $u \neq 0$.
2. On sait que $1 \in E_n$. Soit $y \in E_n$, d'où $y^{n-1} \equiv 1\ [n]$, i.e. $y \times (y^{n-2}) \equiv 1\ [n]$, donc $y^{n-2} \in E_n$ est l'inverse de y. Soient x et $y \in E_n$. On a $(x \cdot y^{-1})^{n-1} \equiv x^{n-1} \cdot y^{n-1}\ [n] \equiv 1\ [n]$. D'où $x \cdot y^{-1} \in E_n$. Ainsi, E_n est un sous-groupe de $(G_n, \cdot)$.
3. Soit n composé. Il existe $a \in \llbracket 1, n-1 \rrbracket$ tel que $a^{n-1} \not\equiv 1\ [n]$, et donc $E_n \subsetneq G_n$. Or, *le cardinal d'un sous-groupe divise le cardinal du groupe*, et donc $|E_n| \mid |G_n| \leqslant n-1$, donc $|E_n| \leqslant \frac{n-1}{2}$.

TD 6.2.2 Algorithme

4.

Algorithme TD 6.13 Algorithme Monte-Carlo testant la primalité d'un nombre en $\mathcal{O}(k\,(\ln k)^3$

Entrée $n \in \mathbb{N}$ et $k \in \mathbb{N}$ deux entiers.

pour $j \in \llbracket 1, k \rrbracket$ **faire**
  $a \leftarrow \mathcal{U}(\llbracket 1, n-1 \rrbracket)$
  si $a^{n-1} \bmod n \neq 1$ **alors**
    retourner `Non`
retourner Oui

En effet, si $|E_n| \leqslant \frac{n-1}{2}$, donc si $a \sim \mathcal{U}(\llbracket 1, n-1 \rrbracket)$, d'où $P(a \in E_n) \leqslant \frac{1}{2}$. La probabilité que l'algorithme échoue est inférieure à $\frac{1}{2^k}$.

TD 6.2.3 Implémentation

Indications Pour calculer $a^b \bmod c$, on décompose b en base 2 : $b = \sum_{i=1}^{p} b_i 2^i$, et donc

$$a^b \bmod c = \left(\prod_{i=1}^{p} a^{b_i 2^i} \right) \bmod c = \prod_{i=0}^{p} \left(a^{b_i 2^i} \bmod c \right).$$

Et, $p \sim \log_2(n)$.

TD 6.3 Exercice 3 : Échantillonnage

Q. 1

Algorithme TD 6.14 Échantillonnage naïf

Entrée T un tableau à n éléments, et $k \in \mathbb{N}$ avec $k \leqslant n$

```
1: T ← Mélanger(T)
2: R ← T[0..k]
3: retourner R
```

Q. 2 Un invariant de boucle est « $\forall p \in [\![0, I-1]\!]\,,\ P(T[p] \in \mathsf{Res}) = \frac{k}{I}$ et $\forall p \in [\![I, n]\!]\,,\ T[p] \notin \mathsf{Res}$ »

Q. 3 Notons $\underline{I}$ et $\underline{\mathsf{Res}}$ l'état des variables avant un tour de boucle ; et, $\bar{I}$ et $\overline{\mathsf{Res}}$ l'état des variables après un tour de boucle.

— Pour $k = I$, on a
  1. $\forall p \in [\![0, k-1]\!]$, $P(T[p] \in \mathsf{Res}) = 1$,
  2. $\forall p \in [\![k, n-1]\!]$, $T[p] \notin \mathsf{Res}$,
  3. $I \leqslant n$.

— Supposons $\underline{I}$, et $\underline{\mathsf{Res}}$ vérifiant l'invariant et la condition de boucle. Alors, on a
  1. $\forall p \in [\![0, \underline{I}-1]\!]$, $P(T[p] \in \underline{\mathsf{Res}}) = \frac{k}{\underline{I}}$,
  2. $\forall p \in [\![\underline{I}, n-1]\!]$, $T[p] \notin \underline{\mathsf{Res}}$,
  3. $\underline{I} < n$, la condition de boucle.

  Soit $j \in [\![0, \underline{I}]\!]$. On a $\bar{I} = \underline{I} + 1$.

Cas 1 $j < k$, et donc $\overline{\mathsf{Res}}(j) = T[\underline{I}]$, et $\forall \ell \neq j$, $\overline{\mathsf{Res}}[\ell] = \underline{\mathsf{Res}}[\ell]$.

Cas 2 $j \geqslant k$, et donc $\forall \ell$, $\overline{\mathsf{Res}}[\ell] = \underline{\mathsf{Res}}[\ell]$.

  1. Soit $p \in [\![0, \underline{I}]\!]$. Montrons $P(T[p] \notin \overline{\mathsf{Res}}) = \frac{k}{\bar{I}}$. Si $p < \underline{I}$, alors

$$\begin{aligned} P(T[p] \in \overline{\mathsf{Res}}) &= P\big(T[p] \in \underline{\mathsf{Res}} \cap j \neq p\big) \\ &= \frac{k}{\underline{I}} \times \frac{\underline{I}}{\underline{I}+1} \\ &= \frac{k}{\bar{I}}. \end{aligned}$$

  Si $P = \underline{I}$, alors d'après 2. $T[p] \notin \underline{\mathsf{Res}}$, donc $P(T[p] \in \overline{\mathsf{Res}}) = P(j < k) = \frac{k}{\underline{I}+1}$.