# Logique et complexité

François Le Maître

3 décembre 2018

## Table des matières

# Première partie
# Calculabilité

Le but de cette première partie est d'arriver à une définition satisfaisante des fonctions calculables par ordinateur. Un premier candidat important est la classe des fonctions récursives primitives que voici.

## 1 Fonctions récursives primitives

Commençons par décrire les ensembles de départ et d'arrivée des fonctions qui nous intéressent : pour un nombre de paramètres $p \in \mathbb{N}$ on note $\mathcal{F}_p$ l'ensemble des applications de $\mathbb{N}^p$ dans $\mathbb{N}$. On adopte la convention que $\mathbb{N}^0 = \{\emptyset\}$ et ainsi $\mathcal{F}_0$ est l'ensemble des applications de $\{\emptyset\}$ dans $\mathbb{N}$ qui s'identifie naturellement à $\mathbb{N}$. La classe des fonctions récursives primitives sera un sous-ensemble de

$$\mathcal{F} := \bigcup_{p \in \mathbb{N}} \mathcal{F}_p.$$

**Définition 1.0.1.** Pour $p \in \mathbb{N}$ et $i \leqslant p$ on note $\pi_p^i$ l'application de **projection** sur la $i$-ème coordonnée de $\mathbb{N}^p$ dans $\mathbb{N}$, c'est-à-dire que pour $(x_1, ..., x_p) \in \mathbb{N}^p$,

$$\pi_p^i(x_1, ..., x_p) = x_i.$$

**Définition 1.0.2.** La fonction **successeur** est la fonction $S : \mathbb{N} \to \mathbb{N}$ donnée par $S(x) = x + 1$.

**Définition 1.0.3.** Étant donnée $f_1, ..., f_n \in \mathcal{F}_p$ et $g \in \mathcal{F}_n$, la fonction **composée** $g(f_1, ..., f_n)$ est l'élément de $\mathcal{F}_p$ défini par : pour tous $x_1, ..., x_p \in \mathbb{N}$,

$$g(f_1, ..., f_n)(x_1, ..., x_p) = g(f_1(x_1, ..., x_p), ..., f_n(x_1, ..., x_p)).$$

**Définition 1.0.4.** Étant donnée une fonction $g \in \mathcal{F}_p$ et $h \in \mathcal{F}_{p+2}$, il existe une unique fonction $f \in \mathcal{F}_{p+1}$ telle que pour tous $x_1, ..., x_p, y \in \mathbb{N}$ on ait

$$f(x_1, ..., x_p, 0) = g(x_1, ..., x_p)$$
$$f(x_1, ..., x_p, y+1) = h(x_1, ..., x_p, y, f(x_1, ..., x_p, y))$$

On appelle $f$ la fonction **définie par récurrence** à partir de $g$ et $h$.

L'unicité d'une telle fonction se prouve par induction sur $y$ (exercice). L'existence est intuitivement claire mais peut en fait être prouvée à partir des axiomes de la théorie des ensembles (cf. [CL93, Chap. 7, Sec. 3]).

**Remarque.** Dans la définition précédente, on autorise $p = 0$, ce qui veut simplement dire qu'il n'y a pas de variables $x_i$ dans la définition de $f$, que $g \in \mathcal{F}^0$ s'identifie à un entier $k$, et enfin que pour tout $y \in \mathbb{N}$,

$$f(0) = k$$
$$f(y+1) = h(y, f(y)).$$

Nous pouvons maintenant définir les fonctions récursives primitives.

**Définition 1.0.5.** L'ensemble $\mathcal{F}_{rp}$ des fonctions récursives primitives est le plus petit sous-ensemble de $\mathcal{F}$ qui :

i) contient les fonctions constantes $\mathbb{N}^p \to \mathbb{N}$,

ii) contient les projections $\pi_p^i$,

iii) contient la fonction successeur $S$,

iv) est stable par composition,

v) est stable par définition par récurrence.

Un sous-ensemble de $\mathbb{N}^p$ est récursif primitif si sa fonction caractéristique l'est.

## 1.1 Exemples de base

### 1.1.1 Addition

L'addition $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ est récursive primitive. En effet on a :
— $x + 0 = x$ (et la projection sur la première coordonnée est bien récursive)
— $x + (y+1) = S(x+y) = S(\pi_3^3(x, y, x+y))$ et $S \circ \pi_3^3$ est bien récursive.
Par composition, on voit alors que $(x_1, ..., x_p) \mapsto x_1 + \cdots + x_p$ est récursive primitive.

### 1.1.2 Multiplication

La multiplication $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ est récursive primitive. En effet on a :
— $x \times 0 = 0$
— $x \times (y+1) = (x \times y) + y$
Par composition, on voit alors que $(x_1, ..., x_p) \mapsto x_1 \times \cdots \times x_p$ est récursive primitive.

### 1.1.3 Soustraction

On note $x\dot{-}y = \max(x-y, 0)$. Montrons que c'est une fonctions récursive primitive. On commence par voir que la fonction $(x, y) \mapsto x\dot{-}1$ est récursive primitive. En effet on a la définition par récurrence sur $x$ suivante :

— $0\dot{-}1 = 0$

— $x + 1\dot{-}1 = x$.

Ensuite, $(x, y) \mapsto x\dot{-}y$ est récursive primitive car

— $x\dot{-}0 = x$

— $x\dot{-}(y + 1) = (x\dot{-}y)\dot{-}1$.

### 1.1.4 Relation d'ordre usuelle

L'ensemble des entiers strictement positifs est récursif primitif puisque sa fonction caractéristique $\chi_{\mathbb{N}^*}$ satisfait la relation de récurrence

— $\chi_{\mathbb{N}^*}(0) = 0$

— $\chi_{\mathbb{N}^*}(x + 1) = 1$.

L'ensemble des couples $(x, y)$ tels que $x > y$ est récursif primitif puisque $x > y \iff x\dot{-}y > 0$.

## 1.2 Propriétés de stabilité

**Proposition 1.2.1.** L'ensemble des sous-ensembles récursifs primitifs est clos par union finie, intersection finie et passage au complémentaire.

*Démonstration.* Remarquons d'abord que si $A$ et $B$ sont récursifs primitifs, alors $A \setminus B$ l'est aussi car $\chi_{A\setminus B} = \chi_A\dot{-}\chi_B$.

L'intersection de deux ensembles $A$ et $B$ récursifs primitifs est également récursive primitive car $\chi_{A\cap B} = \chi_A\chi_B$ et on a vu que la multiplication est récursive primitive.

Remarquons que $\mathbb{N}$ est récursif primitif car sa fonction caractéristique est constante.

On peut alors conclure quant à la stabilité des ensemble récursifs primitifs par union finie : on a $A \cup B = \mathbb{N} \setminus (\mathbb{N} \setminus A \cap \mathbb{N} \setminus B)$ et les résultats que nous venons d'établir permettent alors de conclure. □

**Exemple 1.2.2.** L'ensemble des $(x, y)$ tels que $x = y$ est récursif primitif car c'est l'intersection de l'ensemble des $(x, y)$ tels que $x \leqslant y$ avec l'ensemble des $(x, y)$ tels que $y \leqslant x$.

**Proposition 1.2.3.** L'ensemble des fonctions récursives primitives est stable par changement des variables : si $f : \mathbb{N}^p \to \mathbb{N}$ est récursive primitive alors pour toute application $\sigma : \{1, ..., p\} \to \{1, ..., n\}$ on a que

$$f_\sigma : (x_1, ..., x_n) \mapsto f(x_{\sigma(1)}, ..., x_{\sigma(p)})$$

est récursive primitive.

*Démonstration.* Il suffit de remarquer que $f_\sigma = f \circ (\pi_p^{\sigma(1)}, ..., \pi_p^{\sigma(p)})$. □

**Exemple 1.2.4.** Si $f : \mathbb{N}^3 \to \mathbb{N}$ est récursive primitive, alors la fonction $g : \mathbb{N} \to \mathbb{N}$ donnée par $g(x) = f(x, x, x)$ est récursive (appliquer la proposition précédente avec $\sigma : \{1, 2, 3\} \to \{1\}$ donnée par $\sigma(1) = \sigma(2) = \sigma(3) = 1$)

**Proposition 1.2.5.** L'ensemble des ensembles récursifs primitifs est stable par préimage via des applications récursives primitives : si $f_1, ..., f_n \in \mathcal{F}_p$ sont récursives primitives et $A \subseteq \mathbb{N}^n$ est récursif primitif, alors $\{(x_1, ..., x_p) : (f_1(x_1, ..., x_p), ..., f_n(x_1, ..., x_p)) \in A\}$ est récursif primitif.

*Démonstration.* Sa fonction caractéristique est $\chi_A(f_1, ..., f_n)$. □

### 1.2.1 Définition par cas

**Proposition 1.2.6.** L'ensemble des fonctions récursives primitives est stable par définition par cas portant sur des ensembles récursifs primitifs.

*Démonstration.* Si $\mathbb{N}^p = A_1 \sqcup \cdots \sqcup A_k$ alors la fonction définie par

$$f(x) = f_i(x) \text{ si } x \in A_i$$

peut se réécrire $f = \chi_{A_1} \times f_1 + \cdots + \chi_{A_k} \times f_k$ qui est bien récursive primitive dès lors que $A_1, ..., A_k$ et $f_1, ..., f_k$ le sont. □

**Exemple 1.2.7.** La fonction $(x, y) \mapsto \min(x, y)$ est récursive primitive puisque

$$\min(x, y) = \begin{cases} x & \text{si } x < y \\ y & \text{si } x \geqslant y \end{cases}.$$

De même la fonction max est récursive primitive.

### 1.2.2 Sommes et produits partiels

**Proposition 1.2.8.** Si $f \in \mathcal{F}_{p+1}$ est récursive primitive, alors les fonctions $g$ et $h$ définies par

$$g(x_1, ..., x_p, t) = \sum_{i=0}^{t} f(x_1, ..., x_p, i)$$

$$h(x_1, ..., x_p, t) = \prod_{i=0}^{t} f(x_1, ..., x_p, i)$$

sont également récursives primities.

*Démonstration.* La fonction $g$ est définie par récurrence par $g(x_1, ..., x_p, 0) = f(x_1, ..., x_p, 0)$ puis

$$g(x_1, ..., x_p, t+1) = g(x_1, ..., x_p, t) + f(x_1, ..., x_p, t+1)$$

donc $g$ est récursive primitive. De même $h$ est récursive primitive car définie par récurrence par $h(x_1, ..., x_p, 0) = f(x_1, ..., x_p, 0)$ puis

$$h(x_1, ..., x_p, t+1) = h(x_1, ..., x_p, t) \times f(x_1, ..., x_p, t+1). \quad \square$$

**Exercice.** Montrer que $n \mapsto n!$ est récursive primitive.

### 1.2.3 Schéma $\mu$ borné

Soit $A \subseteq \mathbb{N}^{p+1}$. Considérons la fonction $g \in \mathcal{F}_{p+1}$ définie par

$$g(x_1, ..., x_p, t) = \begin{cases} \min(\{z \leqslant t : (x_1, ..., x_p, z) \in A\}) & \text{si } \exists z \leqslant t : (x_1, ..., x_p, z) \in A \\ 0 & \text{sinon.} \end{cases}$$

On dit que la fonction $g$ est définie par schéma $\mu$-borné à partir de $A$. On notera

$$g(x_1, ..., x_p, z) = \mu\, t \leqslant z : (x_1, ..., x_p, t) \in A.$$

**Proposition 1.2.9.** Soit $A \subseteq \mathbb{N}^{p+1}$ récursif primitif. Alors la fonction $g$ définie par schéma $\mu$-borné à partir de $A$ est récursive primitive.

*Démonstration.* On voit que $g$ est définie par récurrence par $g(x_1, ..., x_p, 0) = 0$ puis

$$g(x_1, ..., x_p, z+1) = \begin{cases} g(x_1, ..., x_p, z) & \text{si } \sum_{i=0}^{z} \chi_A(x_1, ..., x_p, i) \geqslant 1; \\ z+1 & \text{si } \sum_{i=0}^{z} \chi_A(x_1, ..., x_p, i) = 0 \text{ et } (x_1, ..., x_p, z+1) \in A; \\ 0 & \text{dans les autres cas.} \end{cases}$$

Ainsi $g$ est bien récursive primitive (on a utilisé le fait que la définition par cas sur ensembles récursifs primitifs et la somme partielle préservent les fonctions récursives primitives). □

**Proposition 1.2.10.** La classe des ensembles récursifs primitifs est stable par quantification bornée.

*Démonstration.* Soit $A \subseteq \mathbb{N}^{p+1}$ récursif primitif, alors $\{(x_1, ..., x_p, z) : \forall t \leqslant z, (x_1, ..., x_p, t) \in A\}$ est récursif primitif car sa fonction caractéristique est

$$\prod_{t=0}^{z} \chi_A(x_1, ..., x_p, t).$$

L'ensemble $\{(x_1, ..., x_p, z) : \exists t \leqslant z, (x_1, ..., x_p, t) \in A\}$ est récursif primitif car c'est le complémentaire de $\{(x_1, ..., x_p, z) : \forall t \leqslant z, (x_1, ..., x_p, t) \notin A\}$. □

## 1.3 Nouveaux exemples

**Proposition 1.3.1.** La fonction $(x, y) \mapsto q(x, y)$ qui à $(x, y)$ associe le quotient entier de $x$ par $y$ est récursive primitive. (pour avoir une fonction définie partout on considère que le quotient entier de $x$ par 0 est 0).

*Démonstration.* En effet $q(x, y) = \mu\, q \leqslant x : (q+1) \times y > x$. □

**Corollaire 1.3.2.** La fonction qui à $(x, y)$ associe le reste $r(x, y)$ de la division euclidienne de $x$ par $y$ est récursive primitive.

*Démonstration.* On a $r(x, y) = x - q(x, y) \times y$. □

**Corollaire 1.3.3.** L'ensemble des $(x, y)$ tels que $y$ divise $x$ est récursif primitif.

*Démonstration.* C'est l'ensemble des $(x, y)$ tels que $r(x, y) = 0$. □

**Corollaire 1.3.4.** L'ensemble des nombres premiers est récursif primitif.

*Démonstration.* On a $x$ premier ssi $\forall y \leqslant x$, $y$ ne divise pas $x$ ou $y = 1$ ou $y = x$. □

**Corollaire 1.3.5.** La fonction $\pi(n)$ définie par $\pi(n)$ est le $n+1$-ème nombre premier est récursive primitive.

*Démonstration.* En effet on la définit par récurrence par $\pi(0) = 2$ puis $\pi(n+1) = \mu y \leqslant \pi(n)! + 1 : y$ est premier et $y > \pi(n)$. □

## 1.4 Deux codages

Dans cette section, on note quelques bijections utiles permettant de travailler avec des $p$-uplets d'entiers ou des suites finies.

**Définition 1.4.1.** On définit $\alpha : \mathbb{N}^2 \to \mathbb{N}$ par $\alpha(n,p) = \frac{1}{2}(n+p+1)(n+p)+p$.

$\alpha$ est récursive primitive, et on voit qu'elle est bijective (faire un dessin). De plus $\alpha(n,p) \geqslant \max(n,p)$. En particulier on peut retrouver $(n,p)$ à partir de $\alpha(n,p)$ en posant

$$\beta^1(x) = \mu z \leqslant x : \exists t \leqslant x, \alpha(z,t) = x \text{ et } \beta^2(x) = \mu z \leqslant x : \exists t \leqslant x, \alpha(t,z) = x$$

On a bien $(\beta^1(\alpha(n,p)), \beta^2(\alpha(n,p))) = (n,p)$, autrement dit $(\beta^1, \beta^2)$ est la bijection réciproque de $\alpha$.

**Définition 1.4.2.** On définit par récurrence sur $p \in \mathbb{N}^*$ des bijections $\alpha_p : \mathbb{N}^p \to \mathbb{N}$ par $\alpha_1(x) = x$ puis $\alpha_{p+1}(x_1, ..., x_{p+1}) = \alpha_p(x_1, ..., x_{p-1}, \alpha(x_p, x_{p+1})))$.

On définit également $\beta_1^1(n) = n$ puis par récurrence sur $p$, pour $1 \leqslant k \leqslant p$ des fonctions $\beta_p^k$ : $\beta_{p+1}^1(x) = \beta_p^1(x)$,..., $\beta_{p+1}^{p-1}(x) = \beta_p^{p-1}(x)$ puis $\beta_{p+1}^p = \beta^1(\beta_p^p(x))$ et $\beta_{p+1}^{p+1}(x) = \beta^2(\beta_p^p(x))$.

On montre par récurrence que toutes ces fonctions sont récursives primitives, et par construction $(\beta_p^1, ..., \beta_p^p)$ est la bijection réciproque de $\alpha_p$. On a $\alpha_2 = \alpha$ et $\beta^1 = \beta_2^1$, $\beta^2 = \beta_2^2$.

**Exemple 1.4.3.** La suite de Fibonacci est récursive primitive : considérons la fonction suivante définie par récurrence qui encode $(u_n, u_{n+1})$ ; on a $v(0) = \alpha_2(1,1)$ puis

$$v(n+1) = \alpha_2\left(\beta_2^2(v(n)), \beta_2^1(v(n)) + \beta_1^2(v(n))\right).$$

On va maintenant encoder les suites finies d'entiers, c'est-à-dire $\bigcup_{p\in\mathbb{N}} \mathbb{N}^p$, où par convention $\mathbb{N}^0$ est le singleton suite vide.

**Définition 1.4.4.** Etant donnée une suite finie d'entiers $(x_1, ..., x_p)$, on note $\Omega(x_1, ..., x_p) = \pi(0)^{x_1+1} \cdots \pi(p)^{x_p+1}$. On pose également $\Omega(\emptyset) = 1$.

On note $\delta(i,x) = \mu z \leqslant x : x$ n'est pas divisible par $\pi(i)^{z+1}$.

Remarquons qu'à $p$ fixé la fonction $\Omega$ est récursive primitive. De plus $\Omega$ est injective. Enfin $\delta(i,x)$ est l'exposant de $\pi(i)$ dans la décomposition de $x$ en produit de nombres premiers et permet donc de reconstituer $(x_1, ..., x_p)$ à partir de $\Omega(x_1, ..., x_p)$

## 1.5 Fonction d'Ackerman

Dans cette section, nous définissons une variante de la fonction d'Ackerman. Bien que définie par induction, nous allons voir qu'elle n'est pas récursive primitive.

**Définition 1.5.1.** La fonction d'Ackerman est la fonction $\xi \in \mathcal{F}_2$ définie par : pour tous $x, y \in \mathbb{N}$
- $\xi(0,x) = 2^x$
- $\xi(y,0) = 1$
- $\xi(y+1, x+1) = \xi(y, \xi(y+1, x))$

Pour $n \in \mathbb{N}$, notons $\xi_n \in \mathcal{F}_1$ la fonction $\xi(n, \cdot)$. Alors on montre par récurrence sur $n \in \mathbb{N}$ que $\xi_n$ est récursive primitive : en effet pour $n = 0$ on a que $\xi_0(x) = 2^x$ donc $\xi_0$ est récursive primitive, et si $\xi_n$ est récursive primitive alors comme la fonction $\xi_{n+1}$ est définie par récurrence par

$$\xi_{n+1}(0) = 1 \text{ et}$$
$$\xi_{n+1}(x+1) = \xi_n(\xi_{n+1}(x)),$$

elle est également récursive.

Nous commençons maintenant une série de lemmes qui nous mèneront au fait que la fonction d'Ackerman n'est pas récursive primitive.

**Lemme 1.5.2.** Pour tout $n \in \mathbb{N}$ et tout $x \in \mathbb{N}$ on a $\xi_n(x) > x$.

*Démonstration.* On le montre par récurrence sur $n$. Pour $n = 0$ on a bien $2^x > x$ pour tout $x \in \mathbb{N}$. Ensuite supposons que pour tout $x \in \mathbb{N}$ on a $\xi_n(x) > x$ et montrons par récurrence sur $x$ que pour tout $x \in \mathbb{N}$ on a $\xi_{n+1}(x) > x$. Pour $x = 0$ on a $\xi_{n+1}(0) = 1 > 0$, maintenant supposons $\xi_{n+1}(x) > x$, alors $\xi_{n+1}(x+1) = \xi_n(\xi_{n+1}(x))$. Par hypothèse de récurrence sur $n$ on a alors $\xi_{n+1}(x+1) > \xi_{n+1}(x) > x$ et puisque l'on a affaire à des entiers on a bien $\xi_{n+1}(x+1) > x+1$, ce qui conclut la récurrence sur $x$ puis sur $n$. □

**Lemme 1.5.3.** Pour tout $n \in \mathbb{N}$ la fonction $\xi_n$ est strictement croissante.

*Démonstration.* C'est clairement vrai pour $n = 0$, et pour $n \geqslant 1$ on écrit $\xi_n(x+1) = \xi_{n-1}(\xi_n(x)) > \xi_n(x)$ d'après le lemme précédent, d'où la stricte croissance voulue. □

**Lemme 1.5.4.** Pour tous $n, x \in \mathbb{N}$ on a $\xi_{n+1}(x) \geqslant \xi_n(x)$.

*Démonstration.* Pour $x = 0$ c'est vrai car $1 \geqslant 1$. Ensuite, pour $x \geqslant 1$ : écrivons $\xi_{n+1}(x) = \xi_n(\xi_{n+1}(x-1))$, alors comme d'après le lemme 1.5.2 $\xi_{n+1}(x-1)) \geqslant x$ et d'après le lemme 1.5.3 $\xi_n$ est croissante, on conclut $\xi_{n+1}(x) \geqslant \xi_n(x)$. □

On va maintenant montrer que la fonction d'Ackerman croit plus vite que toute fonction récursive. Pour ce faire, il faut d'abord préciser ce qu'on entend par croire plus vite.

**Définition 1.5.5.** On dit qu'une fonction $f \in \mathcal{F}_1$ **domine** $g \in \mathcal{F}_p$ s'il existe $M \in \mathbb{N}$ tel que pour tous $(x_1, ..., x_p)$, on a

$$g(x_1, ..., x_p) \leqslant f(\max(x_1, ..., x_p, M)).$$

Notons que si $f$ est croissante et $f(n)$ tend vers $+\infty$ quand $n$ tend vers $+\infty$, alors $f$ domine $g$ ssi on a $g(x_1, ..., x_p) \leqslant f(\max(x_1, ..., x_p))$ pour tous les $(x_1, ..., x_p)$ sauf pour un nombre fini d'entre eux. De plus, si $f$ est croissante, on a

$$f(\max(x_1, ..., x_p, M)) = \max(f(x_1), ..., f(x_p), f(M)).$$

On définit par récurrence sur $k$ la fonction $\xi_n^k$ par $\xi_n^0(x) = x$ puis $\xi_n^{k+1}(x) = \xi_n(\xi_n^k(x))$ (autrement dit $\xi_n^k$ itère $k$ fois la fonction $\xi_n$). Comme $\xi_n(x) > x$ pour tout $x$, on a $\xi_n^k < \xi_n^{k+1}$. De plus les fonctions $\xi_n^k$ sont strictement croissantes par le lemme 1.5.3. Enfin le fait que $\xi_n(x) > x$ donne par récurrence sur $y$ que pour tout $k$, $\xi_n^{k+y}(0) \geqslant \xi_n^k(y)$.

Soit alors

$$\mathcal{C}_n := \{f \in \mathcal{F} : \exists k, \xi_n^k \text{ domine } f\}.$$

Comme $\xi_{n+1} \geqslant \xi_n$ et les $\xi_n$ sont croissantes, on a $\xi_n^k \leqslant \xi_{n+1}^k$ et donc $\mathcal{C}_n \subseteq \mathcal{C}_{n+1}$.

**Lemme 1.5.6.** $\mathcal{C}_n$ est clos par composition.

*Démonstration.* Soient $f_1, ..., f_q \in \mathcal{F}_p \cap \mathcal{C}_n$, soit $g \in \mathcal{F}_q \cap \mathcal{C}_n$. Soit $M$ et $k$ suffisamment grand tel que pour tous $(x_1, ..., x_p)$ et tout $i \in \{1, ..., q\}$ on ait $f_i(x_1, ..., x_p) \leqslant \xi_n^k(\max(M, x_1, ..., x_p))$ et tous $(y_1, ..., y_q)$ on ait $g(y_1, ..., y_q) \leqslant \xi_n^k(\max(M, y_1, ..., y_q))$. Comme $\xi_n^k$ est croissante on a

$$g(f_1(x_1, ..., x_p), ..., f_q(x_1, ..., x_p)) \leqslant \xi_n^k(\max(\xi_n^k(\max(M, x_1, ..., x_p)), M)).$$

Toujours par croissance, $\xi_n^k(\max(M, x_1, ..., x_p)) = \max(\xi_n^k(x_1), ..., \xi_n^k(x_p), \xi_n^k(M))$ et donc

$$g(f_1(x_1, ..., x_p), ..., f_q(x_1, ..., x_p)) \leqslant \max(\xi_n^k\xi_n^k(x_1), ..., \xi_n^k\xi_n^k(x_p), \xi_n^k\xi_n^k(M), \xi_n^k(M)).$$

Ainsi $g(f_1(x_1, ..., x_p), ..., f_q(x_1, ..., x_p)) \leqslant \xi_n^{2k}(\max(x_1, ..., x_p, M))$ donc $g \in \mathcal{C}_n$. □

La définition par récurrence va nous faire passer de $\mathcal{C}_n$ à $\mathcal{C}_{n+1}$ ; pour cela il nous faut borner $\xi_n^k$ en fonction de $\xi_{n+1}$.

**Lemme 1.5.7.** On a pour tous $x$ et $k$

$$\xi_n^k(x) \leqslant \xi_{n+1}(x + k).$$

*Démonstration.* Par récurrence sur $k$. Pour $k = 0$ il n'y a rien à faire ; ensuite $\xi_{n+1}(x + k + 1) = \xi_n(\xi_{n+1}(x + k)) \geqslant \xi_n\xi_n^k(x) = \xi_n^{k+1}(x)$. □

**Lemme 1.5.8.** Si $g, h \in \mathcal{C}_n$ et $f$ est définie par récurrence à partir de $g$ et $h$ alors $f \in \mathcal{C}_{n+1}$.

*Démonstration.* Soient $M, k$ assez grand pour que $g(x_1, ..., x_p) \leqslant \xi_n^k(\max(x_1, ..., x_p, M))$ et $h(x_1, ..., x_p, y, z) \leqslant \xi_n^k(\max(x_1, ..., x_p, M))$. On montre par récurrence que que

$$f(x_1, ..., x_p, y) \leqslant \xi_n^{k+ky}(\max(x_1, ..., x_p, M))$$

C'est vrai pour $y = 0$, et l'hérédité se prouve comme pour la composition : on écrit

$$\begin{aligned} f(x_1, ..., x_p, y + 1) &= h(x_1, ..., x_p, y, f(x_1, ..., x_p, y)) \\ &\leqslant \max(\xi_n^k(x_1), ..., \xi_n^k(x_p), \xi_n^k(y), \xi_n^k(\xi_n^{k+ky}(\max(x_1, ..., x_p, M))), M) \\ &\leqslant \xi_n^{k+k(y+1)}(\max(x_1, ..., x_p, M)), \end{aligned}$$

où on a utilisé que $\xi_n^k(y) \leqslant \xi_n^{k+y}(0) \leqslant \xi_n^{2k+ky}(0)$. Ceci prouve la récurrence, maintenant on a $\xi_n^{2k+ky}(x) \leqslant \xi_{n+1}(x + 2k + y)$ d'après le lemme précédent, or la fonction $(x_1, ..., x_p, y) \mapsto \max(x_1, ..., x_p) + 2k + ky$ est dans $\mathcal{C}_0 \subseteq \mathcal{C}_{n+1}$ donc par stabilité par composition de $\mathcal{C}_{n+1}$, on a $(x_1, ..., x_p, y) \mapsto \xi_{n+1}(\max(M, \max_i(x_i + 2k + ky)))$ qui est dans $\mathcal{C}_{n+1}$, donc $f$ est bien dans $\mathcal{C}_{n+1}$. □

Soit alors $\mathcal{C} = \bigcup_n \mathcal{C}_n$. D'après les lemmes précédents $\mathcal{C}$ est stable par composition et récurrence, et contient les projections, les fonctions constantes et la fonction successeur. Ainsi l'ensemble des fonctions récursives primitives est inclus dans $\mathcal{C}$.

**Théorème 1.5.9.** *La fonction $g : x \mapsto \xi(x, 2x)$ n'est pas dans $\mathcal{C}$.*

*Démonstration.* Par l'absurde supposons qu'il existe $n \in \mathbb{N}$ et $k$ tel que $g$ soit dominée par $\xi_n^k$. Alors pour tout $x$ sauf un nombre fini, on a

$$\xi_x(2x) \leqslant \xi_n^k(x) \leqslant \xi_{n+1}(x + k).$$

Prenons alors $x$ strictement plus grand que $k$ et $n$ et suffisamment grand, alors $\xi_{n+1}(x + k) < \xi_{n+1}(2x) \leqslant \xi_x(2x)$, ce qui contredit l'inégalité précédente. □

# 2 Fonctions récursives

## 2.1 Fonctions partielles récursives

Dans cette section, on définit les fonctions récursives. Par rapport aux fonctions primitives récursives, on aura certaines fonctions qui seront seulement partiellement définies. On va en effet s'autoriser des schémas $\mu$ non bornés, et la fonction ne sera pas définie en $x$ si on ne peut trouver de $y$ tel que $(x, y) \in A$.

On définit donc $\mathcal{F}_p^*$ comme l'ensemble des fonctions $f : A \to \mathbb{N}$ où $A \subseteq \mathbb{N}^p$. On appelle $A$ le **domaine** de $f$. On dit que $f$ est **totale** si son domaine est $\mathbb{N}^p$. On note $\mathcal{F}^* = \bigcup_{p\in\mathbb{N}} \mathcal{F}_p^*$.

**Définition 2.1.1.** Étant donnée $f_1, ..., f_n \in \mathcal{F}_p^*$ et $g \in \mathcal{F}_n^*$, la fonction **composée** $g(f_1, ..., f_n)$ est l'élément de $\mathcal{F}_p^*$ défini par : pour tous $x_1, ..., x_p \in \mathbb{N}$,

$$g(f_1, ..., f_n)(x_1, ..., x_p) = g(f_1(x_1, ..., x_p), ..., f_n(x_1, ..., x_p)),$$

la fonction n'étant définie que là où l'expression ci-dessus est définie. Plus précisément, si $A$ est le domaine de $g$ le domaine de $g(f_1, ..., f_n)$ est $(f_1, ..., f_n)^{-1}(A)$.

**Définition 2.1.2.** Étant donnée une fonction $g \in \mathcal{F}_p^*$ et $h \in \mathcal{F}_{p+2}^*$, il existe une unique fonction $f \in \mathcal{F}_{p+1}^*$ telle que pour tous $x_1, ..., x_p, y \in \mathbb{N}$ on ait

$$\begin{aligned} f(x_1, ..., x_p, 0) &= g(x_1, ..., x_p) \\ f(x_1, ..., x_p, y + 1) &= h(x_1, ..., x_p, y, f(x_1, ..., x_p, y)), \end{aligned}$$

la fonction $f$ n'étant pas définie dès lors que l'un des termes ci-dessus n'est pas défini. On appelle $f$ la fonction **définie par récurrence** à partir de $g$ et $h$.

Remarquons que par construction si $f(x_1, ..., x_p, y)$ n'est pas défini alors pour tous $z \geqslant y$ on a que $f(x_1, ..., x_p, z)$ n'est pas défini non plus.

**Définition 2.1.3.** Étant donné une fonction $f \in \mathcal{F}_{p+1}^*$, la fonction $g$ définie par **schéma** $\mu$ sur $f$ s'écrit

$$g(x_1, ..., x_p) = \mu y : [f(x_1, ..., x_p, y) = 0]$$

et est donnée par $g(x_1, ..., x_p) = y$ si pour tout $z \leqslant y$ $f(x_1, ..., x_p, z)$ est définie et non nul, tandis que $f(x_1, ..., x_p, y) = 0$, et sinon n'est pas définie. Si $A \subseteq \mathbb{N}^{p+1}$, on notera aussi par commodité

$$\mu y : [(x_1, ..., x_p, y) \in A] = \mu y : [1 \dot{-} \chi_A(x_1, ..., x_p, y) = 0].$$

Nous pouvons désormais définir l'ensemble des fonctions récursives partielles comme étant le plus petit ensemble de fonctions partielles dans $\mathcal{F}^*$ qui

i) contient les fonctions constantes $\mathbb{N}^p \to \mathbb{N}$,

ii) contient les projections $\pi_p^i$,

iii) contient la fonction successeur $S$,

iv) est stable par composition,

v) est stable par définition par récurrence,

vi) est stable par schéma $\mu$.

Par définition les fonctions récursives primitives sont bien récursives (totales). La stabilité par changement de variables, par somme partielle et produit partiel demeurent (même preuve que les Prop. 1.2.3 et 1.2.8). La stabilté par définition par cas sera vue plus tard. On verra à la fin de ce chapitre que la fonction d'Ackermann est récursive elle aussi.

Un ensemble $A \subseteq \mathbb{N}^p$ est dit **récursif** si sa fonction caractéristique est récursive (en particulier cette fonction est totale !). La même preuve que celle de la proposition 1.2.1 nous donne que le complémentaire de tout ensemble récursif est récursif, et que toute réunion ou intersection finie d'ensembles récursifs est récursive.

Un ensemble est dit **récursivement énumérable** si c'est le domaine d'une fonction récursive.

**Proposition 2.1.4.** Tout ensemble récursif est récursivement énumérable.

*Démonstration.* Soit $A \subseteq \mathbb{N}^p$ récursif. Alors $\chi_{A\times\mathbb{N}}$ est récursive (c'est la fonction $(x_1, ..., x_{p+1}) \mapsto \chi_A(x_1, ..., x_p)$). On définit alors $f$ par schéma $\mu$ total sur $A \times \mathbb{N}$, c'est-à-dire

$$f(x_1, ..., x_p) = \mu y : [(x_1, ...x_p, y) \in A \times \mathbb{N}]$$

Il est alors clair que le domaine de $f$ est $A$. □

On verra plus tard par un argument diagonal qu'il existe des ensembles récursivement énumérables non récursifs. Ceci nécessite de pouvoir énumérer récursivement les fonctions récursives, et on va faire ça en utilisant les machines de Turing.

## 2.2 Machines de Turing

Une **machine de Turing** consiste en un nombre fini $n$ de *bandes* infinies parallèles constituées de *cases* numérotées par les entiers naturels. Chaque case contient un symbole qui appartient à $\Sigma := \{0, 1, \#\}$ où # est le symbole de début de bande, uniquement présent sur la première case (case numéro 0). On a également une tête de lecture qui permet de lire et écrire sur les cases de chacune des bandes situées à sa position.

Une machine de Turing possède également un nombre fini d'états dont on notera $Q$ l'ensemble. Il y a deux états particuliers : l'état initial $q_i$ et l'état final $q_f$.

Enfin, la **table de transition** est une application $M : Q \times \Sigma^n \to Q \times \Sigma^n \times \{-1, +1, 0\}$. Afin de faciliter la lecture des tables de transitions, on notera systématiquement $[s_1, ..., s_n]$ un élément de $\Sigma^n$.

La machine fonctionne de la manière suivante, étant donné un contenu des bandes avec seulement un nombre fini de 1 :

- Au temps $t = 0$ elle est dans l'état $q_i$, la tête de lecture est au dessus des cases numéro 0 (et lit donc $(\#, ..., \#)$)
- Au temps $t$, étant dans un état $q \neq q_f$ et sa tête de lecture lisant $[s_1, ..., s_n]$, on considère $M(q, [s_1, ..., s_n]) = (q', [s'_1, ..., s'_n], f)$. La tête de lecture remplace alors $[s_1, ..., s_n]$ par $[s'_1, ..., s'_n]$, la tête de lecture est déplacée dans la direction indiquée par $f$ et enfin la machine se met dans l'état $q'$ au temps $t + 1$.
- Au temps $t$, si $q = q_f$, la machine s'arrête de fonctionner et $t$ est alors le temps de calcul de la machine étant donné le contenu initial des bandes.

La machine commence dans l'état $q_i$ et s'arrête de fonctionner si elle arrive dans l'état $q_f$. Elle peut très bien ne jamais s'arrêter de fonctionner. Une restriction s'impose : si la machine lit $(\#, ..., \#)$ (c'est-à-dire que la tête de lecture au dessus des cases numéro 0) elle ne peut le modifier et la tête de lecture ne peut pas aller à gauche ; et réciproquement si la tête ne lit pas $(\#, ..., \#)$ elle n'a pas

le droit d'écrire de # (ce qui permet de conserver le fait que $(\#, ..., \#)$ désigne seulement la case 0).

On dit qu'une bande est le **code** de l'entier $n$ si elle est commence par #, puis est suivie du symbole 1 $n$ fois, puis ne contient plus que des 0. On dit qu'une bande **représente** l'entier $n$ si elle commence par #, puis est suivie du symbole 1 $n$ fois, puis d'un 0 (mais elle peut contenir des nouveaux 1 ensuite).

Ainsi $(\#, 1, 0, 0...)$ code l'entier 1 (et le représente), et $(\#, 1, 0, 1, 0, 0, ...)$ représente l'entier 1 mais ne le code pas. Remarquons que chaque bande commençant par un # et dont les autres cases ne contiennent que des 0 et un nombre fini de 1 représente un unique entier, (il suffit de compter le nombre de 1 consécutifs à partir de la deuxième case).

**Définition 2.2.1.** Une machine de Turing à $n \geqslant p+1$ bandes **calcule** une fonction $f \in \mathcal{F}_p^*$ si pour tous $(x_1, ..., x_p)$, quand on lance la machine avec les $p$ premières bandes codant respectivement $x_1, ..., x_p$ et les autres codant 0,

— Si $f(x_1, ..., x_p)$ n'était pas définie, alors la machine ne s'arrête jamais

— Si $f(x_1, ..., x_p)$ est définie, la machine s'arrête et la $n$-ème bande *représente* $f(x_1, ..., x_p)$.

On appelle **T-calculable** une fonction calculable par une machine de Turing.

**Remarque.** Dès lors que $n \geqslant p+1$, chaque machine à $n$ bandes calcule bien une (unique !) fonction $f \in \mathcal{F}_p^*$, donnée par : $f(x_1, ..., x_p)$ n'est pas définie si la machine ne s'arrête jamais sur l'entrée $(x_1, ..., x_p)$, et sinon $f(x_1, ..., x_p)$ est l'unique entier représenté par la bande $p+1$ à la fin du calcul.

**Définition 2.2.2.** On dit qu'une machine de Turing à $n \geqslant p+1$ bandes **calcule proprement** une fonction $f \in \mathcal{F}_p^*$ si pour tous $(x_1, ..., x_p)$, quand on lance la machine avec les $p$ premières bandes représentant $x_1, ..., x_p$ et les autres codant 0,

— Si $f(x_1, ..., x_p)$ n'était pas définie, alors la machine ne s'arrête jamais

— Si $f(x_1, ..., x_p)$ est définie, la machine s'arrête et se retrouve avec les $p$ premières bandes qui codent $x_1, ..., x_p$, la $n$-ème bande qui *code* $f(x_1, ..., x_p)$ et les autres bandes qui codent 0.

On appelle **T-calculable proprement** une fonction calculable proprement par une machine de Turing.

**Remarque.** Clairement toute fonction T-calculable proprement est T-calculable ; la réciproque est vraie mais va nous occuper les deux prochaines sections.

Insistons sur le fait que pour le calcul propre on demande que les autres bandes codent 0 ; cette restriction nous permettra de montrer plus facilement que toutes les fonctions récursives sont T-calculables (proprement) au moment où il faudra traiter le cas des récurrences et des compositions.

### 2.2.1 Les fonctions récursives sont T-calculables proprement

**Lemme 2.2.3.** Les fonctions constantes sont T-calculables.

*Démonstration.* Soit $k \in \mathbb{N}$, montrons que la fonction constante égale à $k$ de $\mathbb{N}^p$ dans $\mathbb{N}$ est $T$-calculable. Notre machine a $p+1$ bandes On va se donner $k+2$ états $q_0, ..., q_{k+1}$, $q_0$ étant l'état initial et $q_{k+1}$ l'état final. On pose alors $M(q_0, [\#, ..., \#]) = (q_1, [\#, ..., \#], (+1))$ puis pour $i = 1, ..., k$ et $s_1, ..., s_{p+1} \in \{0, 1\}$, $M(q_i, [s_1, ..., s_{p+1}]) = (q_{i+1}, [s_1, ..., s_p, 1], +1)$. □

**Remarque.** Dans la preuve précédente, on a défini la table de transition uniquement sur les configurations que la machine sera effectivement amenée à rencontrer. C'est une convention que nous suivrons systématiquement par la suite, sachant que pour obtenir une véritable table de transition il suffit de prolonger notre fonction $M$ de manière arbitraire en une application $Q \times \Sigma^n \to Q \times \Sigma^n \times \{-1, +1, 0\}$.

**Lemme 2.2.4.** La fonction successeur est T-calculable proprement.

*Démonstration.* La machine a deux rubans. On a deux états $(q_0, q_1)$ où $q_0$ est l'état initial, $q_1$ est l'état final. On a $M(q_0, [\#, \#]) = (q_0, [\#, \#], +1)$, puis $M(q_0, [1, 0]) = (q_0, [1, 1], +1)$ puis $M(q_0, [0, 0]) = (q_1, [0, 1], 0)$. □

**Lemme 2.2.5.** Les fonctions projection sont T-calculables proprement.

*Démonstration.* Soit $p \in \mathbb{N}$ et $k \in \{1, ..., p\}$. La machine a $p+1$ rubans. On a deux états $(q_0, q_1)$ où $q_0$ est l'état initial, $q_1$ est l'état final. On a $M(q_0, [\#, ..., \#]) = (q_0, [\#, ..., \#], +1)$, puis pour chaque $s_1, .., s_{k-1}, s_{k+1}, ..., s_{p+1} \in \{0, 1\}$

$$M(q_0, [s_1, ..., s_{k-1}, 1, s_{k+1}, ..., s_{p+1}]) = (q_0, [s_1, ..., s_{k-1}, 1, s_{k+1}, ..., s_p, 1], +1) \text{ et}$$
$$M(q_0, [s_1, ..., s_{k-1}, 0, s_{k+1}, ..., s_{p+1}]) = (q_1, [s_1, ..., s_{k-1}, 0, s_{k+1}, ..., s_p, 0], 0).$$ □

**Remarque.** Dans ce qui suit, la numérotation des bandes de 1 à $n$ n'étant pas commode, on s'autorise à numéroter les bandes autrement (par exemple par des couples d'entiers) tout en spécifiant les $p$ bandes d'entrée et la bande de sortie. Pour rester dans le cadre de la définition du calcul propre, il suffit alors de renuméroter ses $n$ bandes par des entiers $1, ..., n$ de sorte que les $p$ bandes d'entrée soient les bandes 1,...,$p$ et la bande de sortie soit la bande $n$.

**Lemme 2.2.6.** Soient $f_1, ..., f_n \in \mathcal{F}_p^*$ T-calculables proprement, et soit $g \in \mathcal{F}_n^*$ également T-calculable proprement. Alors $g(f_1, ..., f_n)$ est T-calculable proprement.

*Démonstration.* Soient $p_1$,...,$p_n$ les nombres de bandes des machines de Turing $M_1, ..., M_n$ calculant $f_1$,..., $f_n$ respectivement, et $p_{n+1}$ le nombre de bandes de la machine de Turing $M_{n+1}$ calculant $g$. On ne va pas décrire précisément les tables de transitions et se contenter de décrire le comportement d'une machine de Turing calculant $g(f_1, ..., f_n)$.

Elle a $p_1 + ... + p_n + p_{n+1}$ bandes que l'on numérote $(i, j)$ avec $i \in \{1, ..., n+1\}$ et $j \in \{1, ..., p_i\}$. Ses bandes d'entrées sont les bandes $(1, j)$ pour $j \in \{1, ..., p\}$, sa bande de sortie.

Pour chaque $i \in \{1, ..., n + 1\}$, notre machine contient une copie de $M_i$ qui travaille sur les bandes $(i, j)$ avec $j \in \{1, ..., p_i\}$. Au début, pour $j = 1, ..., p$, la machine copie la bande $(1, j)$ sur toutes les bandes $(i, j)$ pour $i = 2, ..., n$ (avec la même méthode que pour la projection, à ceci près que l'on ramène la tête de lecture en 0 à la fin de chaque recopiage en allant à gauche jusqu'à retrouver le symbole $\#$ [1]).

Ensuite, les machines $M_1$,...,$M_n$ effectuent leurs calculs une par une à la suite (en particulier à la fin de chaque calcul on ramène la tête de lecture en position 0). Puis pour chaque $i \in \{1, ..., n\}$ on copie le contenu de la bande $(i, p_i)$ sur la bande $(n + 1, i)$ comme précédemment. Enfin on fait travailler $M_{n+1}$ et on efface toutes les bandes de lecture recopiées $(i, j)$ pour $i = 2, ..., n$ et $j = 1, ..., p$. □

**Lemme 2.2.7.** Soit $f \in \mathcal{F}_{p+1}^*$ définie par récurrence à partir de $g \in \mathcal{F}_p^*$ et $h \in \mathcal{F}_{p+2}^*$ toutes deux T-calculables. Alors $f$ est T-calculable proprement.

*Démonstration.* Supposons que la machine $M_g$ qui calcule $g$ a $p_1$ bandes et $M_h$ qui calcule $h$ en a $p_2$. La machine calculant $f$ a $p_1 + p_2$ bandes numérotées $(1, k)$ pour $k \in \{1, ..., p_1\}$ puis $(2, k)$ pour $k \in \{1, ..., p_2\}$ et enfin $(3, 1)$. La bande contenant le résultat final sera $(2, p_2)$. Au début $M_g$ effectue son calcul, puis le résultat est recopié sur $(2, p + 2)$. On recopie également $(1, i)$ sur $(2, i)$ pour $i = 1, ..., p$.

1. C'est là l'unique utilité de ce symbole.

Ensuite on arrive dans un état $q$, on recopie le contenu de $(1,p)$ sur $(3,1)$. On teste si $(2,p+1)$ est égal à $(3,1)$, si c'est le cas on s'arrête. Sinon on commence par faire calculer $M_h$, on augmente $(2,p+1)$ de 1 et on retourne à l'état $q$. □

**Lemme 2.2.8.** Soit $f \in \mathcal{F}_p^*$ définie par schéma $\mu$ à partir de $g$. Si $g$ est $T$-calculable proprement alors $f$ aussi.

*Démonstration.* Soit $n$ le nombre de bandes de $M_g$ calculant $g$, notre nouvelle machine aura elle aussi $n$ bandes numérotées de 1 à $n$ et consiste en une modification de $M_g$.

Elle fonctionne ainsi : au début la bande $p+1$ contient 0 et représente $y$, notre machine calcule $g(x_1,...,x_p,y)$ qui apparaît alors sur la bande $p+2$. Elle teste ensuite le premier bit de la bande $p+1$, s'il est nul elle s'arrête, sinon elle incrémente $y$ et recommence après avoir effacé la bande $p+2$. □

**Remarque.** Dans les constructions précécentes, il faut bien voir que les machines de Turing construites donnent les mêmes fonctions partielles, en particulier elles ne s'arrêtent pas là où la fonction n'est pas définie.

### 2.2.2 Les fonctions T-calculables sont récursives

Pour montrer que les fonctions T-calculables sont récursives, on va devoir montrer qu'à chaque étape de calcul, le contenu des bandes dépend de manière récursive primitive de l'état initial. Pour rendre cette assertion précise, on doit d'abord coder l'état d'une machine de Turing. Pour cela, on commence par remplacer le symbole # par le symbole 2.

On peut alors coder facilement le contenu des bande : on les numérote de 0 à $n-1$ ; le contenu de $n$ bandes $(B_0,...,B_{n-1}) = ((b_i^j)_{i\in\mathbb{N}})_{j=0}^{n-1}$ est alors codé par l'entier

$\Gamma(B_0,...,B_{n-1}) = \sum_{j=0}^{n}\sum_{i=0}^{+\infty} 3^{ni+j}b_i^j$. Étant donné un entier $x$ qui code les bandes, le contenu la case $i$ de la bande $B_j$ est donné par la fonction

$$\gamma(x,i,j) = r(q(n,3^{ni+j}),3).$$

Étant donnée une machine du Turing à $n$ bande $B_0,...,B_{n-1}$ et $r$ états $0,1,...,r-1$ (où l'état 0 est initial, l'état 1 est final) et où la tête est en position $k$ et son état est $q$, sa configuration sera l'entier $\alpha_3(q,\Gamma(B_0,...,B_{n-1}),k)$.

**Lemme 2.2.9.** Étant donnée une machine de Turing à $n$ bandes de table de transition $M$, la fonction qui à $(x_1,...,x_p,t)$ associe la configuration de la machine au temps $t$ avec pour entrées $(x_1,...,x_p)$ est récursive primitive.

*Démonstration.* Il s'agit d'une fonction définie par récurrence, notons la $f$. Tout d'abord, remarquons que la fonction $c_p^n$ qui à $(x_1,...,x_p)$ associe le code de $n$ bandes codant la suite $(x_1,...,x_p,0,...,0)$ est récursive primitive, puisque

$$c_p^n(x_1,...,x_p) = 2\sum_{j=0}^{n-1} 3^j + \sum_{j=0}^{p-1}\sum_{i=1}^{x_{j+1}} 3^{j+ni}$$

Alors $f(x_1,...,x_p,0) = \alpha_3(0,c_p^n(x_1,...,x_p),0)$. Puis on va voir que $f(x_1,...,x_p,t+1)$ est définie par récurrence par cas, les cas étant donnés par la fonction de transition $M$ de notre machine de Turing.

Les cas vont porter sur ce que la tête observe et sur l'état actuel de la machine ; plus précisément pour chaque $(q', [s_0, ..., s_{n-1}], d) \in \{0, ..., r-1\} \times \{0,1,2\}^n \times \{-1, 0, 1\}$ on considère le cas où

$$M(\beta_3^1(y), [\gamma(z, \beta_3^3(y), 0), ..., \gamma(z, \beta_3^3(y), n-1)]) = (q', [s_0, ..., s_{n-1}], d),$$

et dans ce cas on pose alors

$$f(x_1, ..., x_p, t+1) = \alpha_3 \left( q', \ \sum_{j=0}^{n-1} 3^{j+n\beta_3^3(y)} s_j \dot{-} \sum_{j=0}^{n-1} 3^{j+n\beta_3^3(y)} \gamma(z, \beta_3^3(y), j), \ \ \beta_3^3(y) + d \right).$$

Donc $f$ est bien récursive primitive car définie par récurrence par cas récursifs primitifs. □

**Théorème 2.2.10.** *Toute fonction T-calculable est récursive.*

*Démonstration.* Soit une fonction $g \in \mathcal{F}_p^*$ calculable par une machine de Turing $M$ à $n$ bandes et $r$ états, quitte à renuméroter on peut supposer que les états sont $0, ..., r-1$, que l'état initial est 0 et que l'état final est 1. Soit $f$ la fonction récursive primitive du lemme précédent qui encode la configuration au temps $t$ sur les entrées $(x_1, ..., x_p)$. Remarquons qu'il faut au moins $t$ étapes pour que la bande de sortie de $M$ contienne l'entier $t$, en particulier la fonction $h(x_1, ..., x_p, t)$ qui retourne l'entier représenté par la bande de sortie après $t$ étapes est récursive primitive. En effet

$$h(x_1, ..., x_p, t) = \mu y \leqslant t : \left[ r\left(q\left(\beta_3^2(f(x_1, ..., x_p, t)), 3^{yn+p}\right), 3\right) = 0 \right] \dot{-} 1.$$

Soit alors

$$T(x_1, ..., x_p) = \mu t : \left[ \beta_3^2(f(x_1, ..., x_p, t)) = 1 \right]$$

$T$ est récursive, et alors $g(x_1, ..., x_p) = h(x_1, ..., x_p, T(x_1, ..., x_p))$. □

On peut donc rassembler les résultats précédents pour obtenir le théorème suivant.

**Théorème 2.2.11.** *Soit $f \in \mathcal{F}_*^p$, sont équivalentes :*

*(i) $f$ est récursive ;*

*(ii) $f$ est T-calculable proprement ;*

*(iii) $f$ est T-calculable.*

*Démonstration.* L'implication (iii)⇒(i) est le théorème précédent, (i)⇒(ii) provient de la section précédente et (ii)⇒(iii) est claire. □

## 2.3 Conséquences

**Corollaire 2.3.1.** Toute fonction récursive dont le temps de calcul est bornée par une fonction récursive primitive est récursive primitive.

*Démonstration.* En effet, dans la preuve ci-dessus, la fonction $T$ est alors définie par un schéma $\mu$-borné donc récursive primitive, donc $g$ est récursive primitive. □

La réciproque est vraie, comme on le voit par induction sur les fonctions récursives primitives en reprenant les preuves que les fonctions récursives sont T-calculables.

**Corollaire 2.3.2.** La classe des fonctions récursives totales est la plus petite classe de fonctions contenant les fonctions récursives primitives close par composition et schéma $\mu$ total e (c'est-à-dire qu'on ne s'autorise à faire des schémas $\mu$ que lorsque la fonction résultante est totale).

*Démonstration.* En effet dans la preuve du théorème la fonction $T$ est définie par un schéma $\mu$-total sur une fonction récursive primitive, et $g$ est ensuite obtenue en composant $T$ avec des fonctions récursives primitives. □

On va maintenant passer en revue des conséquences importantes sur les ensembles récursivement énumérables.

**Corollaire 2.3.3.** Les assertions suivantes sont vraies :

(1) Tout ensemble récursivement énumérable est la projection d'un ensemble récursif primitif.

(2) Toute projection d'un ensemble récursif est récursivement énumérable.

(3) Toute projection d'un ensemble récursivement énumérable est récursivement énumérable.

*Démonstration.* (1) Soit $A = \mathrm{dom}g$ où $g \in \mathcal{F}_p^*$ est récursive. On reprend les notations de la preuve du théorème. Soit $B = \{(x_1, ..., x_p, t) : \beta_3^2 f(x_1, ..., x_p, t) = 1\}$. Alors $B$ est récursif primitif, et $A$ est la projection sur $\mathbb{N}^p$ de $B$.

(2) Si $A \subseteq \mathbb{N}^{p+q}$ est récursif, la projection de $A$ sur les $p$ premières coordonnées est le domaine de la fonction récursive $(x_1, ..., x_p) \mapsto \mu y(x_1, ..., x_p, \beta_q^1(y), ..., \beta_q^q) \in A$.

(3) Comme la projection d'une projection est une projection (3) découle de (2) et (1) immédiatement. □

La proposition suivante justifie l'appellation *récursivement énumérable*.

**Proposition 2.3.4.** Soit $A$ un sous-ensemble de $\mathbb{N}$, alors sont équivalentes

(1) $A$ est récursivement énumérable

(2) $A$ est l'image d'une fonction récursive

(3) $A$ est l'image d'une fonction récursive primitive

*Démonstration.* (3)⇒(2) est claire, (2)⇒(1) découle de (2) de la proposition précédente car le graphe d'une fonction récursive est récursif et l'image est la projection sur la seconde coordonnée du graphe. Enfin montrons (1)⇒(3). Soit $A$ est récursivement énumérable, soit $g$ récursive telle que $A = \mathrm{dom}g$, alors on fixe une machine de Turing calculant $g$. On sait d'après la preuve du théorème que la fonction $h(x, t)$ qui retourne l'entier codé par la bande de sortie après $t$ étapes est récursive primitive, et que l'ensemble $B$ des $(x, t)$ tels que la machine soit à l'état final au temps $t$ avec pour entrée $x$ est récursif primitif. Soit $x_0 \in A$. Notre fonction récursive $f$ d'image $A$ est donnée par $f(x) = \begin{cases} h(\beta_2^1(x), \beta_2^2(x)) & \text{si } (\beta_2^1(x), \beta_2^2(x)) \in B \\ n & \text{sinon.} \end{cases}$. □

**Proposition 2.3.5.** Une fonction partielle est récursive ssi son graphe est récursivement énumérable.

*Démonstration.* L'ensemble des couples $(x, y)$ tels que $x = y$ est récursivement énumérable, c'est le domaine d'une fonction $h$. Si $f$ est récursive, son graphe est le domaine de la composée $h(f(x_1, ..., x_p), y)$ qui est récursive, donc le graphe est récursivement énumérable.

Réciproquement si le graphe $G$ de $f$ est récursivement énumérable, alors $\alpha_2(G)$ est également récursivement énumérable, donc par la proposition précédente c'est l'image de $h : \mathbb{N} \to \mathbb{N}$ récursive totale. On pose alors $g(n) = \mu m : [\beta_2^1 h(m) = n]$ qui est récursive, puis on a $f(n) = \beta_2^2 h(g(n))$. □

## 2.4 Machine de Turing universelle

On va maintenant construire une fonction récursive qui calcule toutes les fonctions récursives en encodant toutes les machines de Turing possibles. Jusqu'ici on a déjà codé les configurations, reste donc à coder les tables de transition et le nombre de bandes.

Pour chaque $(s_1, ..., s_n) \in \{0,1,2\}^n$, on note $\kappa(s_1, ..., s_n) = \sum_{i=0}^{n-1} s_{i+1} 3^i$ puis pour un $q \in \mathbb{N}$

$$\kappa_1(q, s_1, ..., s_n) = \alpha_2(q, \kappa(s_1, ..., s_n))$$

Pour $M(q, [s_1, ..., s_n]) = (q', [s'_1, ..., s'_n], d)$ on note $\kappa_2(M(q, [s_1, ..., s_n])) = \alpha_3(q', \kappa(s'_1, ..., s'_n), d+1)$. Le code de la table de transition est alors

$$C(M) = \prod_{(s_1,...,s_n)\in\{0,1,2\}^n, q\in\{0,1,...,r-1\}} \pi(\kappa_1(q, [s_1, ..., s_n]))^{\kappa_2(M(q,[s_1,...,s_n]))}.$$

Le **code** de la machine de Turing à $n$ bandes de table de transition $M$ avec $r$ états numérotés de 0 à $r-1$ est alors par définition $\alpha_3(n, r, C(M))$.

On peut montrer que l'ensemble des codes de machines de Turing est récursif primitif, il suffit de voir que les conditions qu'on a énoncées définissent des ensembles récursifs primitifs (par exemple fait que lorsqu'on lit un #, on ne le modifie pas). De plus, si on note $I_p$ l'ensemble des codes de machines de Turing ayant au moins $p+1$ bandes, alors $I_p$ est récursif primitif.

**Théorème 2.4.1.** *La fonction* Sit *qui à* $(i, t, x_1, ..., x_p)$ *associe* 0 *si* $i \notin I_p$ *et sinon la configuration de la machine de Turing d'indice* $i$ *au temps* $t$ *avec comme entrées* $(x_1, ..., x_p)$ *est récursive primitive.*

*Démonstration.* On commence par définir Sit par cas : si $i \notin I_p$ on pose $\text{Sit}(i, t, x_1, ..., x_p) = 0$.

Pour le cas intéressant où $i \in I_p$, il s'agit essentiellement de reprendre la preuve du lemme 2.2.9 sauf que cette fois-ci on a en plus comme paramètre la machine de Turing via son code $i$. Le nombre de bandes est $\beta_3^1(i)$. Le code des $\beta_3^1(i)$ bandes codant $(x_1, ..., x_p, 0, ..., 0)$ est récursif primitif car donné par la formule

$$c(i, x_1, ..., x_p) = 2 \sum_{j=0}^{\beta_3^1(i)-1} 3^j + \sum_{j=0}^{p-1} \sum_{k=1}^{x_{j+1}} 3^{j+\beta_3^1(i)k}.$$

Alors $\text{Sit}(i, x_1, ..., x_p, 0) = \alpha_3(0, c(i, x_1, ..., x_p), 0)$. Calculons $\text{Sit}(i, x_1, ..., x_p, t+1)$ en fonction de $s := \text{Sit}(i, x_1, ..., x_p, t)$ La position de la tête de lecture est $k := \beta_3^3(s)$. Le code de ce qu'elle lit est :

$$C := r(q(\beta_3^2(s), 3^{k\beta_3^1(i)}), 3^{\beta_3^1(i)})$$

La table de transition nous renvoie alors l'entier

$$m := \delta(C, \beta_3^3(i))$$

Le nouveau contenu des cases numéros $k$ est alors $\beta_3^2(m)$, le nouvel état est $\beta_3^1(m)$ et la nouvelle position de la tête de lecture est $k + \beta_3^3(m)$. Ainsi

$$\text{Sit}(i, x_1, ..., x_p, t+1) = \alpha_3\left(\beta_3^1(m),\ \ \beta_3^3(s) + 3^{k\beta_3^1(i)}\beta_3^2(m) - 3^{k\beta_3^1(i)}C,\ \ k + \beta_3^3(m)\right).$$

Ceci prouve que la fonction est bien récursive primitive comme annoncé car définie par cas puis récurrence à partir de composition de fonctions récursives primitives. □

**Définition 2.4.2.** On note $B^p$ l'ensembles des uplets $(i, x_1, ..., x_p, t)$ tels que $\beta_3^2(\text{Sit}(i, x_1, ..., x_p, t)) = 1$, c'est-à-dire que la machine de Turing de code $i$ a fini son calcul sur $(x_1, ..., x_p)$ au temps $t$.

On note $C^p$ l'ensemble des uplets $(i, x_1, ..., x_p, t, y)$ tels que $(i, x_1, ..., x_p, t) \in B^p$ et l'entier $y$ est codé sur la bande de sortie.

D'après le théorème précédent $B^p$ est récursif primitif. On pose alors $T(i, x_1, ..., x_p) = \mu t : (i, x_1, ..., x_p, t) \in B^p$. Alors $T$ est récursive.

Soit $h$ la fonction qui à $(i, x_1, ..., x_p, t)$ associe l'entier codé sur la bande de sortie au temps $t$ (et n'est pas définie si $i \notin I_p$). Par la même preuve que pour le lemme 2.2.10 on voit que $h$ est récursive primitive. Ainsi $C^p$ est récursif primitif.

**Définition 2.4.3.** On définit $\varphi^p$ par $\phi^p(i, x_1, ..., x_p) = h(i, x_1, ..., x_p, T(i, x_1, ..., x_p)$ qui calcule donc le résultat obtenu par la machine de Turing $i$ avec pour entrées $(x_1, ..., x_p)$, et qui n'est pas définie si un tel calcul ne termine pas où si $i \notin I_p$.

D'après la discussion précédente, on a :

**Théorème 2.4.4.** *Pour tout $p \geqslant 1$ la fonction $\varphi^p$ est récursive. Pour toute fonction récursive partielle $f \in \mathcal{F}_p^*$ il existe $i \in I_p$ tel que $f = \varphi_i^p : (x_1, ..., x_p) \mapsto \varphi^p(i, x_1, ..., x_p)$. On dit que $i$ est un* ***indice*** *de $f$.*

## 2.5 Conséquences

On peut maintenant utiliser notre travail pour montrer que les fonctions récursives sont stables par définition par cas récursifs (on pouvait en fait le faire dès que l'on savait que les fonctions récursives sont T-calculables, mais la preuve ci-dessous permet de se familiariser avec le formalisme que l'on vient d'introduire). On se convaincra que la situation est plus compliquée que dans le cas des fonctions récursives primitives car on travaille avec des fonctions partielles : l'astuce que l'on avait utilisée dans la preuve de la proposition 1.2.6 donne une fonction dont l'ensemble de définition pourrait être trop petit. Notamment, si on a deux fonction $f_1, f_2$ récursives dont les domaines $A_1, A_2$ forment une partition [2] de $\mathbb{N}$, on veut pouvoir les recoller en une fonction récursive définie sur $\mathbb{N}$ tout entier, mais la fonction $f_1\chi_{A_1} + f_2\chi_{A_2}$ n'est définie nulle part !

**Proposition 2.5.1.** Soient $f, g \in \mathcal{F}_p^*$ récursives et soit $A \subseteq \mathbb{N}^p$ récursif. Alors la fonction

$$h : (x_1, ..., x_p) \mapsto \begin{cases} f(x_1, ..., x_p) & \text{si } (x_1, ..., x_p) \in A \\ g(x_1, ..., x_p) & \text{sinon.} \end{cases}$$

est récursive

*Démonstration.* Soit $i$ indice de $f$, $j$ indice de $g$, $k$ indice de $\chi_A$. On considère l'ensemble $C^p$ des $(i, x_1, ..., x_p, t, y)$ tels que $(i, x_1, ..., x_p, t) \in B^p$ et $h(i, x_1, ..., x_p, t) = y$ (la machine de Turing d'indice $i$ a fini son calcul et retourne $y$). Alors $C^p$ est récursif primitif.

On considère alors l'ensemble $C$ récursif primitif des $(x_1, ..., x_p, t, y)$ tels que $(i, x_1, ..., x_p, t, y) \in C^p$ et $(k, x_1, ..., x_p, 1) \in C^p$ ou $(j, x_1, ..., x_p, t, y) \in C^p$ et $(k, x_1, ..., x_p, 0) \in C^p$.

Alors $h(x_1, ..., x_p) = \mu y : (x_1, ..., x_p, \mu t : (x_1, ..., x_p, t, y) \in C, y) \in C$. □

2. Le lecteur attentif aura remarqué que $A_1$ et $A_2$ ne sont pas récursifs a priori mais seulement récursivement énumérables, mais on verra plus tard que lorsque deux ensembles récursivement énumérables partitionnent $\mathbb{N}$, ils sont automatiquement récursifs.

**Théorème 2.5.2.** *Un ensemble est récursif ssi il est récursivement énumérable et son complémentaire l'est également.*

*Démonstration.* Si $A$ est récursif, sont complémentaire l'est aussi, en particulier $A$ et son complémentaire sont récursivement énumérables.

Si $A \subseteq \mathbb{N}^p$ est récursivement énumérable de complémentaire $B$ récursivement énumérable, soit $i$ tel que $A = \text{dom}\varphi^p(i, \cdot)$ et $j$ tel que $B = \text{dom}\varphi^p(j, \cdot)$. Alors l'ensemble $D$ des $(x_1, ..., x_p, t)$ tels que $(i, x_1, ..., x_p, t) \in B^p$ ou $(j, x_1, ..., x_p, t) \in B^p$ est récursif primitif.

La fonction $T(x_1, ..., x_p) = \mu t : (x_1, ..., x_p, t) \in D$ est récursive. On pose alors

$$h(x_1, ..., x_p, t) = \begin{cases} 1 & \text{si } (i, x_1, ..., x_p, t) \in B^p \\ 0 & \text{sinon.} \end{cases}$$

qui est récursive primitive. La fonction $h(x_1, ..., x_p, T(x_1, ..., x_p))$ est alors comme voulue. □

**Théorème 2.5.3.** *L'ensemble* $\text{dom}\varphi^1(x, x)$ *est récursivement énumérable de complémentaire non récursivement énumérable. En particulier il existe un sous-ensemble de* $\mathbb{N}$ *récursivement énumérable non récursif.*

*Démonstration.* Soit $A = \text{dom}\varphi^1(x, x)$, alors $A$ est récursivement énumérable par définition. Supposons que son complémentaire $B$ l'est aussi. Mais alors il existe $n$ tel que $B = \text{dom}\varphi^1(n, \cdot)$.

Alors $n \in A \Leftrightarrow \varphi^1(n, n) \Leftrightarrow (n, n) \in B$ ce qui est manifestement contradictoire. □

**Définition 2.5.4.** Une propriété de $p$-uplets d'entiers est dite **décidable** si l'ensemble correspondant est récursif. On peut utiliser ce qu'on vient de faire pour voir que le problème de l'arrêt n'est pas décidable : l'ensemble des $(x, y) \in \text{dom}\varphi^1$ n'est pas récursif car sinon l'ensemble ci-dessus le serait aussi.

On dit de même qu'une propriété est **semi-décidable** si l'ensemble correspondant est récursivement énumérable. Cela veut dire que l'on a un algorithme qui, si on n'est pas dans l'ensemble ne s'arrête jamais, et si on est dans l'ensemble termine.

## 2.6 Trois théorèmes fondamentaux

### 2.6.1 Théorème s-n-m

Ce théorème permet de calculer de manière effective les indices des fonctions obtenues à partir d'autres fonctions dont on connait les indices (par example si on a les indices de deux fonctions, on saura calculer un indice de leur somme de manière récursive primitive). Son énoncé permet en fait de passer de $\varphi^p$ à $\varphi^q$ pour $q < p$ en voyant certains arguments comme des paramètres.

**Théorème 2.6.1.** *Soit* $n, m \geqslant 1$*, il existe une fonction récursive primitive* $s^n_m : \mathbb{N}^{m+1} \to \mathbb{N}$ *telle que pour tout* $i \in I_{n+m}$*, on ait*

$$\varphi^n(s^n_m(i, x_1, ..., x_m), y_1, ..., y_n) = \varphi^{n+m}(i, x_1, ..., x_m, y_1, ..., y_n)$$

*Démonstration.* Si $i \notin I_{n+m}$ on pose $s^n_m(i, x_1, ..., x_m) = 0$ (qui n'est pas un indice de machine de Turing).

Sinon, on change la machine de Turing d'indice $i$ en faisant en sorte d'écrire sur les $m$ premières bandes de lectures les entiers $x_1, ..., x_m$, puis en permutant les numéros de bandes de sorte à ce que l'ancienne bande de sortie se retrouve en position $n$ et que les bandes d'entrée correspondant aux

$n$ derniers arguments se retrouvent en premier. Enfin, on efface toutes les bandes correspondant au arguments $x_1, ..., x_m$.

La machine obtenue calcule la fonction $(y_1, ..., y_n) \mapsto \varphi^{n+m}(i, x_1, ..., x_m, y_1, ..., y_n)$. On peut vérifier (mais c'est très fastidieux !) que l'indice de la machine obtenue dépend de manière récursive primitive que $i$, si on le note $s^n_m(i, x_1, ..., x_m)$ on a le résultat voulu. □

**Exemple 2.6.2.** Montrons qu'on peut trouver une fonction récursive primitive qui à $(i, j)$ associe un indice de la machine de Turing calculant $x \mapsto \varphi^1(i, x) + \varphi^1(j, x)$. Considérons la fonction

$$(i, j, x) \mapsto \varphi^1(i, x) + \varphi^1(j, x)$$

Elle est récursive, soit donc $k$ un indice de cette fonction. Alors pour tout $x, i, j$,

$$\varphi^1(i, x) + \varphi^1(j, x) = \varphi^3(k, i, j, x) = \varphi^1(s^1_2(k, i, j), x)$$

Ainsi une fonction qui marche est $(i, j) \mapsto s^1_2(k, i, j)$.

### 2.6.2 Théorème de Rice

On a déjà vu que $\mathrm{dom}\varphi^1(x, x)$ est un ensemble récursif non récursivement énumérable. On va le réduire récursivement (cf. TD2 pour la définition) à tout ensemble non trivial d'indices pour obtenir :

**Théorème 2.6.3.** *Soit $\mathcal{G} \subseteq \mathcal{F}^*_1$ un ensemble de fonctions partielles récursives non vide et distinct de l'ensemble des fonctions partielles récursives. Alors*

$$A_{\mathcal{G}} := \{i \in I_1 : \varphi^1_i \in \mathcal{G}\}$$

*n'est pas récursif.*

*Démonstration.* Quitte à remplacer $\mathcal{G}$ par son complémentaire, on peut supposer que la fonction vide est dans $\mathcal{G}$. Soit $b$ l'indice d'une machine calculant une fonction qui n'est pas dans $\mathcal{G}$. Considérons la fonction

$$\psi : (x, y) \mapsto \varphi^1(b, y) + \varphi^1(x, x) \dot{-} \varphi^1(x, x)$$

Alors la fonction $\psi(x, \cdot)$ est égale à la fonction vide si $(x, x) \notin \mathrm{dom}\varphi^1(x, x)$ (et donc dans $\mathcal{G}$), et sinon elle est égale à $\varphi^1(b, y)$ (donc pas dans $\mathcal{G}$).

On va calculer un indice de cette fonction : soit $k$ un indice de $\psi$, alors $\psi(x, y) = \varphi^2(k, x, y) = \varphi^1(s^1_1(k, x), y)$. La fonction qui à $x$ associe $s^1_1(k, x)$ est récursive primitive, et par construction l'image réciproque de $A_{\mathcal{G}}$ est le complémentaire de $\mathrm{dom}\varphi^1(x, x)$. Comme ce dernier n'est pas récursif, $A_{\mathcal{G}}$ n'est pas récursif. □

**Remarque.** On en déduit une version plus satisfaisante du fait que le problème de l'arrêt soit non décidable : l'ensemble des indices de machines de Turing à 1 paramètre qui terminent pour l'entrée 0 est non récursif.

En faisant la même preuve et en se souvenant que le complémentaire de $\mathrm{dom}\varphi^1(x, x)$ n'est en fait pas récursivement énumérable, on obtient :

**Proposition 2.6.4.** Soit $\mathcal{G}$ un ensemble de fonctions récursives partielles à 1 variable contenant la fonction vide et distinct de l'ensemble de toutes les fonctions partielles récursives. Alors l'ensemble

$$A_{\mathcal{G}} := \{i \in I_1 : \varphi^1_i \in \mathcal{G}\}$$

n'est pas récursivement énumérable.

En particulier, l'ensemble des indices calculant la fonction vide n'est pas récursivement énumérable. On dit que le problème de savoir si une machine ne va pas s'arrêter pour chaque entrée n'est pas semi-décidable.

## 2.7 Théorème(s) de point fixe

Ces théorèmes, dus à Kleene, sont parfois appelés théorèmes de la récursion.

On peut montrer (en ajoutant des états inutiles à la machine de Turing) qu'il existe une fonction récursive primitive $\psi^p : \mathbb{N} \to \mathbb{N}$ telle que $\psi(i) > i$ et pour tout $i \in I^p$, $\varphi_i^p = \varphi_{\psi(i)}^p$. Réciproquement, le théorème de point fixe de Kleene affirme que l'on peut trouver un tel indice pour toute fonction $\psi$ récursive totale.

**Théorème 2.7.1.** *Soit $\alpha : \mathbb{N} \to \mathbb{N}$ récursive totale. Alors il existe $i \in I_p$ tel que*

$$\varphi_i^p = \varphi_{\alpha(i)}^p$$

*Démonstration.* Considérons la fonction $(y, x_1, ..., x_p) \mapsto \varphi^p(\alpha(s_1^1(y, y)), x_1, ..., x_p)$. Elle a un indice $k \in I_{p+1}$, donc

$$\varphi^{p+1}(k, y, x_1, ..., x_p) = \varphi^p(\alpha(s_1^1(y, y)), x_1, ..., x_p) = \varphi^p(s_1^p(k, y), x_1, ..., x_p)$$

d'après le théorème s-n-m. En posant $i = s_1^p(k, k)$ on obtient le résultat voulu en regardant l'équation ci-dessus pour $y = k$. □

**Exemple 2.7.2.** On va montrer que la fonction d'Ackerman est récursive. Considérons l'application $\psi \mapsto \psi^*$ de $\mathcal{F}_2^*$ dans $\mathcal{F}_2^*$ donnée par

$$\psi^*(x, y) = \begin{cases} 2^y & \text{si } x = 0 \\ 1 & \text{si } y = 0 \\ \psi(x - 1, \psi(x, y - 1)) & \text{sinon.} \end{cases}$$

On montre par induction que la fonction d'Ackerman est le seul point fixe de $\psi \mapsto \psi^*$. Montrons qu'il y a une fonction récursive (primitive) $\alpha$ qui à un indice de $\psi$ associe un indice de $\psi^*$. Considérons la fonction de 3 variables

$$\theta(i, x, y) = \begin{cases} 2^y & \text{si } x = 0 \\ 1 & \text{si } y = 0 \\ \varphi^2(i, x - 1, \varphi^2(i, x, y - 1)) & \text{sinon.} \end{cases}$$

Soit $k$ un indice de cette fonction, alors si $i$ est un indice de $\psi$ on voit que $\psi^*(x, y) = \theta(i, x, y) = \varphi^3(k, i, x, y) = \varphi^2(s_2^2(k, i), x, y)$. On pose donc $\alpha(i) = s_2^2(k, i)$, alors le théorème de point fixe dans sa première version nous donne un indice $i$ tel que $\varphi_{\alpha(i)}^2 = \varphi_i^2$, donc la fonction d'Ackerman est d'indice $i$, en particulier elle est récursive.

On donne maintenant une version où l'indice de $\alpha$ est un paramètre supplémentaire : on peut alors trouver $i$ de manière récursive primitive en fonction de l'indice de $\alpha$.

**Théorème 2.7.3.** *Il existe une fonction $h_p : \mathbb{N} \to \mathbb{N}$ récursive primitive telle que dès lors que $\varphi_j^1$ est totale, on a $h_p(j) \in I_p$ et*

$$\varphi_{h_p(j)}^p = \varphi_{\varphi_j^1(h_p(j))}^p.$$

*Démonstration.* On reprend la démonstration précédente avec paramètre : on considère un indice $k$ de la fonction $(j, y, x_1, ..., x_p) \mapsto \varphi^p(\varphi_j^1(s_1^1(y, y)), x_1, ..., x_p)$. Alors

$$\begin{aligned}\varphi^{p+2}(k, j, y, x_1, ..., x_p) &= \varphi^p(\varphi_j^1(s_1^1(y, y)), x_1, ..., x_p) \\ &= \varphi^{p+1}(s_1^1(k, j), y, x_1, ..., x_p) \\ &= \varphi^p(s_1^1(s_1^1(k, j), y), x_1, ..., x_p).\end{aligned}$$

On pose $h_p(j) = s_1^1(k, j)$, alors on a le résultat voulu en posant $y = s_1^1(k, j)$. □

Terminons avec le cas où $\alpha$ prend plusieurs paramètres.

**Théorème 2.7.4.** *Soit $p \geqslant 1$ et $n \in \mathbb{N}$, soit $\alpha : \mathbb{N}^{n+1} \to \mathbb{N}$ récursive totale. Alors il existe $h : \mathbb{N}^n \to \mathbb{N}$ récursive primitive à valeur dans $I_p$ telle que pour tous $(x_1, ..., x_n)$,*

$$\varphi^p_{\alpha(x_1,...,x_n,h(x_1,...,x_n))} = \varphi^p_{h(x_1,...,x_n)}$$

# Deuxième partie
# Complexité

## 3 Machines de Turing déterministes

### 3.1 Modèle et classes de complexité en temps déterministe

Pour parler de complexité, nous allons affiner le modèle des machines de Turing de manière à pouvoir définir efficacement la complexité en temps et en taille d'algorithmes. Par rapport à la section précédente, on se donne un **alphabet d'entrée** fini $\Sigma$ (précédemment $\{0, 1\}$) et un **alphabet de travail** $\Gamma$ avec un symbole spécial **blanc** $B \in \Gamma \setminus \Sigma$ qui représentera une case vide. Rappelons qu'étant donné un alphabet $A$, $A^*$ désigne l'ensemble des mots sur $A$, c'est-à-dire des suites finies d'éléments de $A$. On note $\epsilon$ le mot vide.

On travaille avec $k$ bandes biinfinies, c'est-à-dire dont les cases seront indexées par $\mathbb{Z}$. On aura deux bandes particulières : une bande de lecture et une bande d'écriture. La bande de lecture sera la première et contient le mot initial qui sera écrit dans l'alphabet d'entrée à partir de la case 1.

La bande d'écriture sera la dernière et contiendra le résultat du calcul. Les bandes de calcul ont des têtes qui font à la fois lecture et écriture, contrairement aux deux autres (lecture et écriture). Autrement dit les $k-1$ premières bandes peuvent être lues, les $k-1$ dernières peuvent être écrites. Les têtes se déplacent indépendamment les unes des autres.

On aura également un ensemble fini $Q$ d'états, avec un état $q_0$ initial, et cette fois-ci deux états finaux : un état $q_a$ d'**acceptation** et un état $q_r$ de **rejet**. Enfin on aura une fonction de transition

$$\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma^{k-1} \to Q \times \Gamma^{k-1} \times \{-1, 0, 1\}^k.$$

Si $\delta(q, a_1, ..., a_{k-1}) = (q', b_2, ..., b_k, \epsilon_1, ..., \epsilon_k)$ alors $(b_2, ..., b_k)$ indique ce que les têtes écrivent sur les $k-1$ dernières bandes dans l'état, $\epsilon_i$ indique dans quelle direction elles se déplacent et $q'$ est le nouvel état quand elles lisent les symboles $(a_1, ..., a_{k-1})$ sur les $k-1$ premières bandes et sont dans l'état $q$.

**Définition 3.1.1.** Une **machine de Turing déterministe** à $k \geqslant 2$ bandes est un uplet $M = (k, \Sigma, \Gamma, B, Q, q_0, q_a, q_r, \delta)$ satisfaisant les conditions ci-dessus.

Une **configuration** d'une machine de Turing $M$ à $k$ bandes est un uplet $(q, (c_j^1, ..., c_j^k)_{j\in\mathbb{Z}}, (i_1, ..., i_k))$ où $q$ désigne l'état actuel et pour chaque $l \in \{1, ..., k\}$, la suite $(c_j^l)_{j\in\mathbb{Z}}$ désigne le contenu de la bande $j$ et $i_l$ la position de sa tête de lecture. Une configuration **finale** est une configuration où l'état est $q_a$ ou $q_r$.

La **configuration initiale** d'entrée $(x_1, ..., x_n) \in \Sigma^*$ est donnée par $(q_0, (c_j^1, ..., c_j^k)_{j\in\mathbb{Z}}, (1, ..., 1)))$ où les $c_j^l$ sont tous égaux à $B$ sauf pour $l = 1$ et $j = 1, ..., n$ auquel cas $c_j^1 = x_j$.

Le calcul de $M$ sur l'entrée $x = (x_1, ..., x_n)$ est est la suite potentiellement infinie $C_0, C_1, ...$ où $C_0$ est la configuration intiale d'entrée $(x_1, ..., x_n)$, $C_{i+1}$ est la configuration obtenue à partir de $C_i$ en suivant les instructions données par $\delta$ (en particulier si $C_i$ est dans un état final ($q_a$ ou $q_r$) le calcul est la suite finie $(C_0, ..., C_i)$). Le passage de $C_i$ à $C_{i+1}$ est une étape de calcul. On dit que la machine $M$ **s'arrête** sur l'entrée $x$ (ou que $M(x)$ s'arrête) si le calcul de $M$ sur $x$ est fini (c'est-à-dire qu'on atteint une configuration finale).

Dans le cas où $M$ s'arrête, on dit que $M$ accepte $x$ si $M$ s'arrête sur $x$ et l'état de la configuration finale est $q_a$, sinon l'état de la configuration finale est $q_r$ et on dit que $M$ rejette $x$. Le résultat du calcul de $M$ sur $x$, noté $M(x)$, est le mot dans l'alphabet sur $\Gamma$ qui apparait sur la bande d'écriture dans la configuration finale. Plus précisément, si $j$ est le plus petit indice tel que $c_j^k \neq B$ et $l$ le plus grand tel que $c_l^k \neq B$ alors $M(x) = (c_j^k, ..., c_l^k)$.

**Définition 3.1.2.** Soit $M$ une machine de Turing. Le **langage accepté** par $M$ est l'ensemble des $x \in \Sigma^*$ tels que $M$ accepte $x$. La **fonction calculée** par $M$ est la fonction partielle $f_M : \Sigma^* \to \Gamma^*$ définie lorsque $M$ s'arrête sur $x$ par $f_M(x) = M(x)$.

**Définition 3.1.3.** Si $M(x)$ s'arrête, le **temps de calcul** de $M(x)$ est l'indice de la configuration finale, autrement dit c'est le nombre d'étapes de calcul. L'espace de calcul de $M(x)$ est le nombre totale de cases visitées par les têtes *sur les rubans de travail*.

Un langage est **décidable** s'il existe une machine de Turing qui s'arrête sur tout mot et le reconnait. On ne considérera que des langages décidables.

**Définition 3.1.4.** Soit $t : \mathbb{N} \to \mathbb{N}$. La classe $\mathsf{DTIME}(t(n))$ est l'ensemble des langages reconnus par une machine de Turing $M$ telle qu'il existe $\alpha > 0$ telle que pour tout entrée $x$, $M(x)$ fonctionne en temps $\leqslant \alpha t(|x|)$.

Si $\mathcal{T}$ est un ensemble de fonctions, on pose $\mathsf{DTIME}(\mathcal{T}) = \bigcup_{t\in\mathcal{T}} \mathsf{DTIME}(t)$.

On pose $\mathsf{P} = \mathsf{DTIME}(\{n \mapsto 1 + n^k : k \in \mathbb{N}\})$, $\mathsf{EXP} = \mathsf{DTIME}(\{n \mapsto 2^{n^k} : k \in \mathbb{N}\})$ et $\mathsf{E} = \mathsf{DTIME}(\{n \mapsto 2^{kn} : k \in \mathbb{N}\})$.

## 3.2 Codages

Par rapport à ce qui a été fait dans la section calculabilité, on va se donner un codage plus efficace des entiers et bien connu : le codage binaire. Sauf mention contraire, un entier $n$ sera donc représenté par un mot de longueur $\lceil \log_2 n \rceil$, par exemple 8 sera représenté par le mot 100.

**Exercice.** Décrire une machine de Turing qui détermine si un entier est pair. Quel est son temps de calcul en fonction de la longueur du mot représentant l'entier ?

On utilisera souvent un symbole # comme délimiteur, ce qui nous permettra de coder une suite finie d'entiers par leurs codes binaires entrecoupés de #, par exemple $(2, 3)$ sera codé par 10#11.

En particulier un rationnel sera codé par la suite (numérateur,dénominateur). Une matrice sera codée par la suite de ses coefficients (en utilisant ## pour signifier que l'on passe à la ligne suivante), un graphe sera codé par sa matrice d'adjacence et enfin un polynôme par la liste de des coefficients.

## 3.3 Changements d'alphabet

Comme on a plusieurs alphabets possibles, on cherche une manière de se ramener à un alphabet fixé sans trop perdre de temps. Commençons par préciser ce que l'on entend par se ramener à. Un morphisme entre $\Sigma_1^*$ et $\Sigma_2^*$ est une application $\varphi : \Sigma_1^* \to \Sigma_2^*$ tel que pour tous $w, w' \in \Sigma_1^*$, on a $\varphi(ww') = \varphi(w)\varphi(w')$. Remarquons qu'un morphisme est complètement déterminé par les valeurs $\varphi(s)$ où $s \in \Sigma_1^*$, et que réciproquement si on fixe les valeurs de $\varphi$ sur $\Sigma_1$, on a une unique manière d'étendre $\varphi$ en un morphisme $\Sigma_1^* \to \Sigma_2^*$.

On dit qu'un morphisme $\varphi : \Sigma_1^* \to \Sigma_2^*$ **préserve les longueurs** si il existe $n \in \mathbb{N}$ tel que pour tout $s \in \Sigma_1$, $|\varphi(s)| = n$. Remarquons qu'alors pour tout mot $w$, la longueur de $\varphi(w)$ est $n\,|w|$, et qu'étant donné un élément $y$ de l'image de $\varphi$, on peut facilement trouver un antécédent de $y$ en trouvant successivement des lettres antécédentes des segments de $n$ lettres dans $y$.

**Définition 3.3.1.** Soit $M_1$ et $M_2$ deux machines de Turing. On dit que $M_2$ simule $M_1$ si on a un morphisme $\varphi : \Gamma_1^* \to \Gamma_2^*$ *injectif qui préserve les longueurs* et qui envoie $B$ sur $B$, $\Sigma_1$ sur $\Sigma_2$, tel que $M_1$ accepte $w$ ssi $M_2$ accepte $\varphi(w)$ et si $M_1$ termine sur $x$, alors $M_2(\varphi(x))) = \varphi(M_1(x))$.

**Théorème 3.3.2.** *Soit $M$ une machine de Turing sur un alphabet $\Sigma$. Il existe une machine de Turing $\tilde{M}$ sur l'alphabet $\{0, 1\}$, d'alphabet de travail $\{0, 1, B\}$ simulant $M$ avec morphisme $\varphi$ telle que si $M(x)$ termine en temps $t$ et en espace $s$, alors $\tilde{M}(\varphi(x))$ termine en temps $\leqslant 3t\lceil\log_2|\Gamma|\rceil$ et en espace $\leqslant s\lceil\log_2|\Gamma|\rceil$.*

*De plus, il existe une machine de Turing qui au code de $M$ associe le code de $\tilde{M}$.*

*Démonstration.* Commençons par coder notre alphabet en binaire. Soit $\psi : \Gamma \setminus \{B\} \to \{0, 1\}^{\lceil\log_2|\Gamma|\rceil}$ injective, on pose $\psi(B) = B$ puis on étend $\psi$ en un morphisme $\Gamma^* \to \{0, 1, B\}^*$.

Notre nouvelle machine a autant de rubans ($n$) que l'ancienne, et chaque case de l'ancienne sera représentée par $\log_2|\Sigma|$ cases de la nouvelle. Il va falloir à chaque étape commencer par lire $\log_2|\Sigma|$ cases de gauche à droite, puis se placer dans l'état correspondant, écrire de droite à gauche, et enfin déplacer les têtes en prenant soin au fait qu'on doit toujours écrire quelque chose sur le ruban d'écriture, mais on ne peut pas le lire pour le recopier. Ainsi, lorsque l'on doit déplacer la tête d'écriture vers la droite, il faut retenir ce qu'on venait d'écrire dessus pour l'écrire une nouvelle fois (si on la déplace vers la gauche, on peut écrire n'importe quoi). Avec ceci en tête, voici une liste des états de notre nouvelle machine : pour chaque état $q$ de notre ancienne machine (y compris l'état initial et les deux états terminaux), on a :

- $q^{lec}_{w_1,\dots,w_{n-1}}$ pour chaque mots $w_i$ sur $\{0, 1, B\}$ tous de même longueur $< \lceil\log_2|\Gamma|\rceil$.
- L'état $q_0{}^{lec}_{\epsilon,\dots,\epsilon}$ est l'état initial de $\tilde{M}$.
- $q^{ecr}_{w_2,\dots,w_n,w,d_1,\dots,d_n}$ pour chaque mots $w_2, ..., w_n$ sur $\{0, 1, B\}$ tous non vides et de même longueur $\leqslant \lceil\log_2|\Gamma|\rceil$, $w$ de longueur $\lceil\log_2|\Gamma|\rceil$ (on garde en mémoire le contenu entier de la case d'écriture car on devra peut-être repasser dessus) et $d_i \in \{-1, 0, +1\}$,
- $q^{depl}_{d_1,\dots,d_n,w}$ où $d_i \in \{-1, 0, +1\}$ et $w$ mot sur $\{0, 1, B\}$ de longueur $\leqslant \lceil\log_2|\Gamma|\rceil$
- Un état acceptant $\tilde{q}_a$ et rejetant $\tilde{q}_r$.

Voici la table de transition, où $q$ est un état de l'ancienne machine :

— États de lecture : si $w_1$,..., $w_{n-1}$ sont des mots tous de la même longueur $l$ :
  — Si $l \leqslant \lceil \log_2 |\Gamma| \rceil - 2$, alors $\tilde{M}(q^{lec}_{w_1,...,w_{n-1}}, a_1, .., a_{n-1}) = (q^{lec}_{w_1 a_1,...,w_{n-1}a_{n-1}}, a_2, ..., a_{n-1}, B, +1, ..., +1)$
  — Si $l = \lceil \log_2 |\Gamma| \rceil - 1$, alors pour $a_1, ..., a_{n-1} \in \{0,1\}$ et $i \in \{1, ..., n-1\}$ soit $v_i = w_i a_i$ (on vient de finir la lecture, et $v_i$ est ce qu'on a lu). Soit $b_i = \psi^{-1}(v_i)$, et soit $(r, c_2, ..., c_n, d_1, ..., d_n) := M(q, b_1, ..., b_{n-1})$. Soit, pour $i = 2, ..., n$, $u_i = \psi(c_i)$. Alors on pose
$$\tilde{M}(q^{lec}_{w_1,...,w_{n-1}}, a_1, .., a_{n-1}) = (r^{ecr}_{u_2,...,u_n,u_n,d_1,...,d_n}, B, ..., B, 0, ..., 0).$$

— États d'écriture : si $w_2$,...,$w_n$ sont des mots tous de la même longueur $l$ que l'on écrit $w_i = v_i a_i$ pour $a_i \in \{0, 1, B\}$, $w$ est un mot de longueur $\lceil \log_2 |\Gamma| \rceil$ et $d_1, ..., d_n$ sont des éléments de $\{-1, 0, 1\}$ :
  — Si $l \geqslant 2$ et $q \neq q_a, q_r$ on pose
$$\tilde{M}(q^{ecr}_{w_2,...,w_n,w,d_1,...,d_n}, b_1, ..., b_{n-1}) = (q^{ecr}_{v_1,...,v_n,w,d_1,...,d_n}, a_2, ..., a_n, -1, ..., -1)$$
  — Si $l = 1$ et $q \neq q_a, q_r$, on a $v_i = \epsilon$ et $w_i = a_i$ et $w = av$ et on pose, si $d_n = +1$
$$\tilde{M}(q^{ecr}_{w_2,...,w_n,w,d_1,...,d_n}, b_1, ..., b_{n-1}) = (q^{depl}_{w,d_1,...,d_n}, a_2, ..., a_n, d_1, ..., d_n),$$
  et si $d_n = 0$, alors
$$\tilde{M}(q^{ecr}_{w_2,...,w_n,w,d_1,...,d_n}, b_1, ..., b_{n-1}) = (q^{depl}_{a^{|w|-1},d_1,...,d_n}, a_2, ..., a_n, d_1, ..., d_n),$$
  et si $d_n = -1$
$$\tilde{M}(q^{ecr}_{w_2,...,w_n,w,d_1,...,d_n}, b_1, ..., b_{n-1}) = (q^{depl}_{B^{|w|-1},d_1,...,d_n}, a_2, ..., a_n, 0, ..., 0, -1),$$
  enfin si $q = q_a$ ou $q = q_r$ on pose
$$\tilde{M}(q^{ecr}_{w_2,...,w_n,w,d_1,...,d_n}, b_1, ..., b_{n-1}) = (\tilde{q}, a_2, ..., a_n, 0, ..., 0).$$

— États de déplacement : si $d_1, ..., d_{n-} \in \{-1, 0, +1\}$,
  — si $w$ de longueur $\geqslant 2$, on écrit $w = av$ avec $a \in \{0, 1, B\}$, alors
$$\tilde{M}(q^{depl}_{w,d_1,...,d_n}, a_1, ..., a_{n-1}) = (q^{depl}_{v,d_1,...,d_n}, a_2, ..., a_{n-1}, a, d_1, ..., d_n)$$
  — si $w = a$ est de longueur 1, alors
$$\tilde{M}(q^{depl}_{w,d_1,...,d_n}, a_1, ..., a_{n-1}) = (q^{lec}_{\epsilon,...,\epsilon}, a_2, ..., a_{n-1}, a, d_1, ..., d_n)$$

On voit que chaque étape de $M$ est accomplie en au plus $3\lceil \log_2 |\Gamma| \rceil$ étapes de la nouvelle machine, et que l'espace occupé est au plus $3\lceil \log_2 |\Gamma| \rceil$ fois plus grand.

Le fait q'il y a une machine de Turing qui au code de toute machine $M$ associe le code de $\tilde{M}$ serait très fastidieux, on espère que le fait d'avoir explicité précisément la construction de $\tilde{M}$ suffira à convaincre le lecteur. □

## 3.4 Machine universelle avec perte de temps quadratique

Nous allons construire une machine de Turing capable de simuler toutes les autres avec, par rapport à ce qui a été fait en calculabilité, un contrôle le plus fin possible sur le temps et l'espace de calcul. Pour cela, comme en calculabilité, on doit commencer par coder les machines de Turing et leurs entréess.

Voyons déjà comment nous allons coder une machine de Turing à $k$ bandes ; pour simplifier l'écriture on commence par se ramener au cas où $\Sigma = \{0, 1, ..., |\Sigma| - 1\}$, $\Gamma = \{0, 1, ..., |\Gamma| - 1\}$, $Q = \{0, 1, ..., |Q|-1\}$ et le symbole de blanc est $|\Gamma|-1$, alors le code de la machine est $1^k 01^{|\Sigma|} 01^{|\Gamma|} 01^{|Q|} 0 u_\delta$ où $u_\delta$ est définit ainsi :

— Pour chaque $(s_1, ..., s_k) \in \Gamma^k$ et $q \in Q$, si $\delta(q, s_1, ..., s_{k-1}) = (q', s'_2, ..., s'_k, d_1, ..., d_k)$ alors on pose

$$w_{\delta,q,s_1,...,s_k} = 1^{q'} 01^s_1 01^{s_2} 0 \cdots 1^{s_k} 01^q 01^{s'_1} 0 \cdots 1^{s'_k} 01^{d_1+1} 0 \cdots 1^{d_k+1} 0,$$

— Puis on obtient $u_\delta$ en concaténant les $w_{\delta,s_1,...,s_k}$ en suivant l'ordre lexicographique sur l'ensemble des $(s_1, ..., s_k)$ (l'ordre importe peu, il faut juste en fixer un).

**Définition 3.4.1.** Étant donnée une machine de Turing $M$, on note $\langle M \rangle$ le **code** de $M$, obtenu par la procédure ci-dessus.

Pour coder une entrée $x = x_1 \cdots x_n$ de $M$, qui est donc un mot sur $\{0, ..., |\Sigma| - 1\}$, on utilise encore la notation unaire : on pose

$$\langle x \rangle = 01^{x_1} 0 \cdots 01^{x_n}$$

En particulier le code du mot vide est 0. Remarquons que $|\langle x \rangle| \leqslant (|\Sigma| + 1)\,|x|$.

**Définition 3.4.2.** Le code du couple $(M, x)$ est par définition la concaténation de $\langle M \rangle$ et $\langle x \rangle$, que l'on notera simplement $\langle M, x \rangle$.

Nous pouvons maintenant énoncer le théorème fondamental de cette section.

**Théorème 3.4.3.** *Il existe une machine de Turing $U$ à cinq bandes d'alphabet d'entrée $\{0, 1\}$ et d'alphabet de travail $\{0, 1, B\}$ telle que pour toute machine de Turing $M$, on a une constante $\alpha_M > 0$ telle que pour tout entrée $x$ de $M$, $U(\varphi\langle M, x\rangle)$ simule $M(x)$ en un temps $\leqslant \alpha_M st$ et un espace $\leqslant s^2$, où $s$ et $t$ sont les espaces et temps d'exécution de $M(x)$. En particulier comme $s \leqslant t$, le temps de calcul de $U(\varphi\langle M, x\rangle)$ est borné par $\alpha_M t^2$.*

*Démonstration.* Nous allons nous contenter d'une description informelle de $U$. Précisons d'entrée de jeu le rôle des cinq bandes de $U$.

— La première contient l'entrée $\langle M, x \rangle$.
— La seconde contient le code de $\tilde{M}$ machine sur l'alphabet $\{0, 1, B\}$ obtenue via le théorème 3.3.2.
— La troisième contient l'état actuel de $\tilde{M}$, ainsi que ce que $\tilde{M}$ lira et écrira dans la simulation.
— La quatrième contient les bandes simulées de $\tilde{M}$ marquées avec les positions des têtes.
— La cinquième sera le ruban de sortie de $\tilde{M}$.

D'après le théorème 3.3.2, on peut enrichir l'alphabet de travail de $U$ sans modifier le temps de calcul à une constante multiplicative près, et c'est ce que nous ferons en ajoutant un symbole de séparation # et des symboles $(0, *)$, $(1, *)$ et $(B, *)$ qui permettront de connaître l'emplacement des têtes de $\tilde{M}$.

□

## 3.5 Théorème de hiérarchie en temps déterministe

**Définition 3.5.1.** Une fonction $t : \mathbb{N} \to \mathbb{N}$ est constructible en temps s'il existe $\alpha > 0$ et une machine de Turing $M$ qui, pour tout $n \in \mathbb{N}$, retourne $1^{t(n)}$ sur l'entrée $1^n$, et ce en temps $\leqslant \alpha t(n)$.

**Théorème 3.5.2.** *Soient $f, g : \mathbb{N} \to \mathbb{N}*$ des fonctions telles que $g$ soit constructible en temps, $g(n) \geqslant n$ et $f^2 = o(g)$. Alors $\mathsf{DTIME}(f) \subsetneq \mathsf{DTIME}(g)$.*

*Démonstration.* On va utiliser un argument diagonal. On construit une machine de Turing $V$ qui utilise la machine universelle $U$ de la section précédente, dont le comportement est le suivant :

- Elle prend en entrée $\langle M, x\rangle$.
- Elle commence par calculer $g(|\langle M, x\rangle|)$ sur un ruban.
- Elle simule sur $U$ pendant $g(|\langle M, x\rangle|)$ étapes le calcul de $M$ sur $x$ (attention, ce sont des étapes pour $U$, pas pour $M$).
- A la fin, si $M$ n'a pas fini son calcul elle rejette.
- Et si $M$ a fini son calcul, elle accepte si $M$ rejette et rejette si $M$ accepte.

Notons que par construction le langage reconnu par notre machine est dans $\mathsf{DTIME}(g)$. Si il était également dans $\mathsf{DTIME}(f)$, soit $M$ le reconnaissant en temps $\leqslant \alpha f(n)$. Considérons $M(\langle M, x\rangle)$ où $x$ est un mot sur $\{0, 1\}$. Par hypothèse, la simulation du calcul de $M(\langle M, x\rangle)$ sur $U$ prend un temps $\leqslant \alpha f(|\langle M, x\rangle|)$, ce qui pour $x$ suffisamment grand est plus petit que $g(|\langle M, x\rangle|)$ car $f^2 = o(g)$.

Mais alors par construction $V(\langle M, x\rangle)$ accepte ssi $M(\langle M, x\rangle)$ rejette, ce qui est absurde car $M$ et $V$ doivent reconnaitre le même langage. □

**Corollaire 3.5.3.** $\mathsf{P} \subsetneq \mathsf{E} \subsetneq \mathsf{EXP}$.

*Démonstration.* Considérer $f(n) = 2^{\sqrt{n}}$, comme $n^{2^k} = o(2^n)$ on a $\mathsf{P} \subseteq \mathsf{DTIME}(f)$, mais $\mathsf{DTIME}(f) \subsetneq \mathsf{DTIME}(2^n)$ d'après le théorème précédent, or ce dernier est inclus dans $\mathsf{E}$.

Pour $\mathsf{E} \subsetneq \mathsf{EXP}$, considérer $f(n) = 2^{n^2}$. □

# 4 Machines de Turing non déterministes

Les machines de Turing **non déterministes** diffèrent des machines déterministes vue précédemment uniquement au niveau de la table de transition : essentiellement, au lieu d'avoir une seule transition possible en fonction de l'état et de ce qui est lu, on en a plusieurs, ce qui se formalise ainsi.

**Définition 4.0.1.** Étant donné un alphabet de travail $\Gamma$, un nombre $k$ de bandes et un ensemble $Q$ d'états, une **table de transition non déterministe** est un sous-ensemble

$$\delta \subseteq (Q \times \Gamma^{k-1}) \times (Q \times \Gamma^{k-1} \times \{-1, 0, +1\}^k)$$

tel que pour tous $q \in Q$, tous $(s_1, ..., s_{k-1}) \in \Gamma^{k-1}$ il existe $q' \in Q$, $(s'_2, ..., s'_k) \in \Gamma^{k-1}$ et $(d_1, ..., d_k) \in \{-1, 0, +1\}^k$ tels que

$$((q, [s_1, ..., s_{k-}]), (q', [s'_2, ..., s'_k], d_1, ..., d_k)) \in \delta,$$

ce que l'on notera

$$(q, [s_1, ..., s_{k-}]) \xrightarrow{\delta} (q', [s'_2, ..., s'_k], d_1, ..., d_k)$$

afin d'alléger un peu la notation.

**Remarque.** Une table de transition déterministe $\delta$ peut se voir comme une table non déterministe telle que pour tous $q \in Q$, tous $(s_1, ..., s_{k-1}) \in \Gamma^{k-1}$ il existe un *unique* uplet $(q', [s'_2, ..., s'_k], d_1, ..., d_k) \in Q \times \Gamma^{k-1} \times \{-1, 0, +1\}^k$ tel que

$$(q, [s_1, ..., s_{k-}]) \xrightarrow{\delta} (q', [s'_2, ..., s'_k], d_1, ..., d_k)$$

Une **machine de Turing non déterministe** à $k \geqslant 2$ bandes est un uplet $M = (k, \Sigma, \Gamma, B, Q, q_0, q_a, q_r, \delta)$ satisfaisant les mêmes conditions qu'une machine déterministe, à ceci près que l'on demande maintenant que $\delta$ soit une table de transition non déterministe comme défini ci-dessus.

Définissons maintenant le fonctionnement d'une machine de Turing non déterministe ; comme on a plusieurs possibilités à chaque étape de calcul, on n'aura plus une simple suite de configurations mais un **arbre de calcul**, défini comme suit par induction. Étant donnée une entrée $x \in \Sigma^*$,

- La racine de notre arbre est la **configuration initiale**, eon rappelle qu'elle est donnée par $(q_0, (c_j^1, ..., c_j^k)_{j\in\mathbb{Z}}, (1, ..., 1)))$ où les $c_j^l$ sont tous égaux à $B$ sauf pour $l = 1$ et $j = 1, ..., n$ auquel cas $c_j^1 = x_j$.
- Étant donnée une configuration $(q, (c_j^1, ..., c_j^k)_{j\in\mathbb{Z}}, (x_1, ..., x_k)$ avec $q \neq q_a, q_r$, ses **configurations filles** sont tous les uplets $(q', [s'_2, ...s'_k], x_1 + d_1, ..., x_k + d_k)$ tels que

$$(q, [s_1, ..., s_{k-}]) \xrightarrow{\delta} (q', [s'_2, ..., s'_k], d_1, ..., d_k)$$

- Pour une configuration terminale $(q_a, (c_j^1, ..., c_j^k)_{j\in\mathbb{Z}}, (x_1, ..., x_k)$ ou $(q_r, (c_j^1, ..., c_j^k)_{j\in\mathbb{Z}}, (x_1, ..., x_k)$, on déclare que c'est une feuille de l'arbre de calcul, c'est-à-dire qu'elle n'a pas de descendants.

Un **chemin de calcul** est une suite éventuellement infinie de configurations $(C_i)$ qui commence par la configuration initiale, et telle que $C_{i+1}$ est une fille de $C_i$, et si la suite est finie son dernier élément est terminal.

**Définition 4.0.2.** Une machine de Turing non déterministe **termine** sur une entrée $x$ si tous ses chemins de calcul sur l'entrée $x$ sont finis. De plus, on dit qu'elle

- accepte $x$ s'il existe un chemin de calcul qui termine sur l'état acceptant
- rejette $x$ si tous les chemins de calcul terminent sur l'état rejetant

**Remarque.** du fait que Il est important de noter que si une machine termine sur $x$, alors soit elle l'accepte, soit elle le rejette, et que ces deux cas s'excluent. De plus, contrairement à ce qui se passait dans le cas déterministe, les définitions d'accepter et de rejeter ne sont plus symmétriques.

**Définition 4.0.3.** Soit $M$ une machine de Turing non déterministe d'alphabet d'entrée $\Sigma$ qui termine sur toute entrée. Le **langage reconnu** par $M$ est l'ensemble des mots $x \in \Sigma^*$ tels que $M(x)$ accepte.

**Remarque.** Si on note $L$ le langage reconnu par $M$, et si on échange $q_a$ et $q_r$ et note $\tilde{L}$ le langage reconnu par la nouvelle machine, il n'est pas vrai que $\tilde{L}$ est le complémentaire de $L$. En effet, par définition $\tilde{L}$ est l'ensemble des mots $x$ tels qu'il existe un chemin de calcul sur $x$ qui termine sur un état rejetant, tandis que le complémentaire de $L$ est l'ensemble des mots $x$ tels que *tout* chemin de calcul sur $x$ termine sur un état rejetant.

C'est une différence fondamentale entre les machines déterministes et non déterministes, due à la non symétrie de la définition de l'acceptation et du rejet dans le cas non déterministe. Cependant, comme on le verra bientôt, les machines non déterministes ne reconnaissent pas plus de langages que les machines déterministes.

Nous pouvons d'ors et déjà reprendre la définition des classes DTIME mot pour mot, en ajoutant le qualificatif *non déterministe*.

**Définition 4.0.4.** Soit $t : \mathbb{N} \to \mathbb{N}$ la classe $\mathsf{NTIME}(t(n))$ est l'ensemble des langages reconnus par une machine de Turing $M$ non déterministe telle qu'il existe $\alpha > 0$ telle que pour tout entrée $x$, $M(x)$ fonctionne en temps $\leqslant \alpha t(|x|)$.

Si $\mathcal{T}$ est un ensemble de fonctions, on pose $\mathsf{NTIME}(\mathcal{T}) = \bigcup_{t\in\mathcal{T}} \mathsf{NTIME}(t)$.

On pose $\mathsf{NP} = \mathsf{NTIME}(\{n \mapsto 1 + n^k : k \in \mathbb{N}\})$, $\mathsf{NEXP} = \mathsf{NTIME}(\{n \mapsto 2^{n^k} : k \in \mathbb{N}\})$ et $\mathsf{NE} = \mathsf{NTIME}(\{n \mapsto 2^{kn} : k \in \mathbb{N}\})$.

Une des questions ouverte les plus importantes en complexité est de savoir si $\mathsf{P} = \mathsf{NP}$. On reviendra dessus plus tard.

On a plusieurs fois décrit le fonctionnement de machines de Turing informellement, en utilisant parfois des instructions proches de celles d'un langage de programmation. Pour capturer ce qu'apporte le non déterminise, on se donne l'instruction suivante.

**Définition 4.0.5.** L'instruction `deviner` $x_i$ affecte à $x_i$ la valeur 0 ou 1 de manière non déterministe.

Il est clair que cette instruction est représentable dans une machine de Turing non déterministe : il suffit, sur l'état courant $q$, d'avoir deux transitions, une qui affecte 0 à $x_i$ et l'autre qui lui affecte 1.

**Exemple 4.0.6.** Voici un algorithme pour le problème Composé qui demande de déterminer, pour un entier $x = x_1 \cdots x_n$ en binaire, si $x$ n'est pas un nombre premier :

— Pour $i$ allant de 1 à $n$, deviner $b_i \in \{0, 1\}$ puis $b'_i \in \{0, 1\}$.

— Si $b_1 \cdots b_n \times b'_1 \cdots b'_n = x$ et $1 < b_1 \cdots b_n < x$, accepter.

On se permettra de deviner également des lettres d'un langage plus étendu, et de deviner des mots. Ainsi l'exemple précédent se réécrit :

— Deviner $b \in \{0, 1\}^n$ et $b' \in \{0, 1\}^n$

— Si $b \times b' = x$ et $1 < b < x$, accepter.

Comme le calcul d'un produit et la comparaison prennent un temps polynomial en le nombre de bits des entrées, on conclut que le problème Composé est dans NP. Il a été montré assez récemment (2002 ) que le problème de déterminer si un entier en binaire est premier est dans P, ce qui entraîne que Composé est également dans P.

## 4.1 Liens entre temps déterministe et temps polynomial

Le résultat qui suit nous assure que les machines non déterministes ne reconnaissent pas plus de langages que les machines déterministes, avec de plus une borne explicite sur le temps de calcul déterministe nécessaire pour reconnaître le même langage qu'une machine non déterministe.

**Théorème 4.1.1.** *Soit $t : \mathbb{N} \to \mathbb{N}$. Alors on a les inclusions suivantes :*

$$\mathsf{DTIME}(t(n)) \subseteq \mathsf{NTIME}(t(n)) \subseteq \mathsf{DTIME}(2^{O(t(n))})$$

**Corollaire 4.1.2.** On a l'inclusion $\mathsf{NP} \subseteq \mathsf{EXP}$.

On peut utiliser le même genre d'idées pour donner une caractérisation existentielle de NP et de NEXP comme suit.

**Théorème 4.1.3.** *Soit $A \subseteq \Sigma^*$ un langage. On a les caractérisations suivantes :*

1. *$A \in \mathsf{NP}$ ssi il existe un langage $B \in \mathsf{P}$ et un polynôme $p$ à coefficients dans $\mathbb{N}$ tels que pour tout $x \in \Sigma^*$ :*
$$x \in A \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}, (x,y) \in B.$$
2. *$A \in \mathsf{NEXP}$ ssi il existe un langage $B \in \mathsf{EXP}$ et un polynôme $p$ à coefficients dans $\mathbb{N}$ tels que pour tout $x \in \Sigma^*$ :*
$$x \in A \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}, (x,y) \in B.$$

## 4.2 Machine non déterministe universelle optimale en temps

**Théorème 4.2.1.** *Il existe une machine de Turing $U_N$ à six bandes d'alphabet d'entrée $\{0,1\}$, d'alphabet de travail $\{0,1,B\}$, telle que pour toute machine de Turing $M$ non déterministe sur un alphabet $\Sigma$, il existe un morphisme $\varphi : \Sigma^* \to \{0,1\}$ et une constante $\alpha_M > 0$ tels que pour tout $x \in \Sigma^*$, $M(x)$ accepte (resp. rejette) ssi $U_N(\langle M \rangle, \varphi(x))$ accepte (resp. rejette), et si on note $t$ le temps de calcul de $M(x)$, alors le temps de calcul de $U_N$ sur l'entrée $(\langle M \rangle, \varphi(x)$ est borné par $\alpha_M t$.*

**Remarque.** On ne demande pas que le morphisme préserve les longueurs (même si on pourrait le faire) : en effet cette restriction n'a d'utilité que pour décoder facilement le contenu de la bande de sortie, qui dans le cas non déterministe ne nous intéresse pas (notamment parce qu'il dépend du chemin de calcul).

*Démonstration.* Comme dans le cas déterministe, notre machine $U_N$ commence par écrire sur son premier ruban de travail le code de $\tilde{M}$ qui simule $M$ mais a pour alphabet d'entrée $\{0,1\}$ et pour alphabet de travail $\{0,1,B\}$. De plus, on s'autorise à utiliser un langage de travail plus riche que $\{0,1,B\}$, sachant que l'on peut toujours s'y ramener d'après le théorème 3.3.2 de réduction de l'alphabet, dont la preuve s'adapte sans difficulté au cas non déterministe. Le principe général de notre machine universelle, par rapport au cas déterministe, est d'économiser du temps en devinant les contenus des rubans.

Décrivons de suite les six rubans de notre machine universelle :

1. Le premier contient $\langle M \rangle, x$.
2. Le second va contenir le code de $\tilde{M}$.
3. Le troisième contiendra la liste des contenus de case devinés et des transitions effectuées.
4. Le quatrième servira à la vérification ruban par ruban que les contenus devinés sont cohérents.
5. Le cinquième est le ruban de sortie.

La machine simule une étape de calcul de $\tilde{M}$ ainsi : elle devine un contenu de cases et une transition, elle vérifie que la transition est dans la table de transition et est cohérente avec la transition précédente, qui est inscrite sur le ruban 3 (c'est-à-dire que l'état d'où la nouvelle transition part est celui où arrivait l'anicenne transition). Si elle voie que la transition est impossible elle rejette. Si ce n'est pas le cas elle note sur le ruban 3 le contenu des cases et la transition à la suite de ce qui était déjà écrit. Notons que chacune de ces étapes prend un temps constant : il existe $\beta_M > 0$ tel que pour tout $n$, $n$ étapes de $\tilde{M}$ seront simulées en temps $\leqslant \beta_M n$.

Voyons maintenant quand la machine accepte. À chaque fois que le nombre d'étapes simulées est une puissance de deux (donc aux étapes 1,2, 4, 8, 16...), la machine *vérifie* que le contenu deviné est cohérent : pour chaque ruban $k$ de la machine $\tilde{M}$, elle suit les transitions indiquées par le ruban

3 et travaille sur le ruban 4 comme si c'était le ruban $k$ de $\tilde{M}$. Elle vérifie à chaque étape que ce qu'elle est censée lire sur le ruban $k$ y est bien écrit en lisant le ruban 4. Si à un moment elle voit une incohérence, elle rejette. De plus, si à un moment elle arrive dans un état acceptant/rejetant de $\tilde{M}$, elle accepte/rejette. Remarquons qu'on a une constante $\kappa_M > 0$ telle que la vérification d'*une* étape prend un temps de calcul au plus $\kappa_M$

Voyons maintenant que le temps de calcul de la machine $U_N$ décrite ci-dessus est comme voulu. Soit donc $t$ le temps de calcul de $M$ sur une entrée $x$. Soit $k$ le plus petit entier tel que $2^k \geqslant x$, alors $2^k \leqslant 2t$. D'après ce qui vient d'être dit, le temps de calcul de $U_N$ est au plus $\beta_M 2^k + \kappa_M \sum_{i=0}^{k} 2^i$, le premier terme provenant de la simulation elle-même, et le deuxième des vérifications aux temps de la forme $2^i$. Ainsi le temps de calcul est majoré par

$$\beta_M 2^k + 2^{k+1}\kappa_M = 2^k(\beta_M + 2\kappa_M) \leqslant 2(\beta_M + 2\kappa_M)t.$$

On a donc le résultat voulu avec $\alpha_M = 2(\beta_M + 2\kappa_M)$. □

### 4.3 Hiérarchie en temps non-déterministe

### 4.4 Comparaison des classes P, NP, EXP et NEXP

## 5 NP-complétude

### 5.1 Définitions et exemple de base

**Théorème 5.1.1.** *Le langage*

$$L := \{(\langle M \rangle, x, 1^t) : M \text{ non déterministe et } M(x) \text{ a un chemin acceptant de longueur} \leqslant t\}$$

*est* NP*-complet.*

### 5.2 SAT est NP-complet

### 5.3 3-SAT et IND-SET sont NP-complets

## 6 Complexité en espace

On rappelle que l'espace de calcul d'une machine de Turing déterministe sur une entrée $x$ est le nombre de cases visitées par les têtes *des rubans de travail*. Si la machine est non déterministe, on regarde le nombre maximum de cases visitées sur les rubans de travail en suivant un chemin de calcul sur l'entrée $x$.

Afin de simplifier certaines preuves, on va faire une restriction naturelle sur les calculs des machines de Turing : pour qu'un mot soit accepté ou rejeté, on demandera en plus qu'à chaque étape de calcul, la tête de lecture soit à distance au plus 1 du mot d'entrée. Cela permettra d'avoir un contrôle direct sur l'espace de travail lorsque l'on voudra composer des machines de Turing, bien que ce problème puisse être contourné.

**Définition 6.0.1.** Soit $s : \mathbb{N} \to \mathbb{N}$. La classe DSPACE($s$) est l'ensemble des langages $L$ tels qu'il existe une machine de Turing $M$ déterministe reconnaissant $L$ et $\alpha_M > 0$ tel que sur toute entrée $x$, l'espace de calcul de $M(x)$ est borné par $\alpha_M s(|x|)$.

La classe $\mathsf{NSPACE}(s)$ est l'ensemble des langages $L$ tels qu'il existe une machine de Turing $M$ non déterministe reconnaissant $L$ et $\alpha_M > 0$ tel que sur toute entrée $x$, l'espace de calcul de $M(x)$ est borné par $\alpha_M s(|x|)$.

Afin de comparer les classes de complexité en espace et en temps, on introduit le **graphe des configurations** d'une machine de Turing $M$ a priori non déterministe : étant donnée une entrée $x$, les sommets du graphe des configurations sont des uplets décrivant l'état, les positions des têtes *sauf celle de sortie* et contenus des rubans de travail sur l'entrée $x$ à différentes étapes du calcul. C'est un graphe dirigé, où l'on relie une configuration à une autre si une transition permet de passer de l'une à l'autre.

Si on sait que le calcul de $M(x)$ se fait en espace $s(|x|)$, les positions des têtes seront comprises entre $-s(|x|)$ et $s(|x|)$, si $r$ est le nombre de rubans on a $|\Gamma|^{r[2s(|x|)+1]}$ possibilités pour leur contenu. D'où le lemme suivant.

**Lemme 6.0.2.** Sur une entrée $x$, si l'espace de calcul de $M$ est donnée par $s(|x|)$, alors le nombre de sommets du graphe de configuration sur l'entrée $x$ est $\leqslant 2^{O(s(|x|))}$.

Nous pouvons désormais comparer les classes de complexité en temps et en espace.

**Proposition 6.0.3.** Soient $s : \mathbb{N} \to \mathbb{N}$ et $t : \mathbb{N} \to \mathbb{N}$. On a les inclusions suivantes :

$$\mathsf{NTIME}(t) \subseteq \mathsf{DSPACE}(t)$$
$$\mathsf{NSPACE}(s) \subseteq \mathsf{DTIME}(2^{O(s)})$$

*Démonstration.* Pour la première inclusion, remarquer qu'un parcours en profondeur de l'arbre de calcul de $M$ non déterministe de temps de calcul $t$ prend un espace $O(t)$.

Pour la deuxième, on cherche à déterminer si la configuration initiale peut être reliée à une configuration acceptante dans le graphe (dirigé) des configurations, ce qui se fait en un temps polynomial en le nombre de sommets, et donc en temps $\leqslant 2^{O(s)}$ d'après le lemme précédent. □

**Définition 6.0.4.** Définissons quatre classes de complexité en espace :
- $\mathsf{L} = \mathsf{DSPACE}(\log n)$
- $\mathsf{NL} = \mathsf{NSPACE}(\log n)$
- $\mathsf{PSPACE} = \bigsqcup_{k\in\mathbb{N}} \mathsf{DSPACE}(n^k)$
- $\mathsf{NPSPACE} = \bigsqcup_{k\in\mathbb{N}} \mathsf{NSPACE}(n^k)$

On a alors les inclusions suivantes d'après la proposition précédente :

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}\,.$$

On a des théorèmes de hiérarchie en espace via la notion de fonction constructible en espace ; ces derniers permettent de voir que l'inclusion de $\mathsf{L}$ dans $\mathsf{PSPACE}$ est stricte. On va maintenant montrer que $\mathsf{PSPACE} = \mathsf{NPSPACE}$. Pour ça, on fait un détour par le problème d'accessibilité.

On définit $\leqslant_m^L$ la réduction many-one en espace logarithmique. Remarquons que d'après la proposition précédente, une machine qui fonctionne en espace logarithmique Pour voir que c'est transitif, on a besoin du lemme suivant.

**Lemme 6.0.5.** Soient $f$ et $g$ deux fonctions calculables en espace logarithmique, alors $f \circ g(x)$ est calculable en espace logarithmique.

*Démonstration.* On garde un compteur en binaire qui correspond à la position de la tête de lecture en entrée de $M_f$ calculant $f$. À chaque fois que l'on lit, on simule $g$ et renvoie uniquement la lettre à la bonne position. Pour ce faire, on a besoin de savoir la position de la lettre la plus à gauche du ruban de sortie de $M_g$, puis de garder en tête la position de la tête de sortie et écrire quand on est à la bonne position. Comme $g$ fonctionne en espace logarithmique, elle fonctionne en temps polynomial et donc les positions, comptées en binaire, prennent un espace en $O(\log|x|)$. Ceci prend donc un espace $O(\log|x|)$. La simulation de $f \circ g$ prendra au final un espace $O(\log|x|)+O(\log|g(x)|$ or $|g|(x)$ est polynomial en $|x|$ ; d'où le résultat.

□

On a donc bien la transitivité de $\leqslant_m^L$.
On définit le problème Access :
— **donnée** : Un graphe orienté et deux sommets $v_1, v_2$
— **question** : Existe-t-il un chemin orienté de $v_1$ à $v_2$ ?

**Proposition 6.0.6.** Access est NL-complet.

*Démonstration.* C'est dans NL car il suffit de parcourir le graphe en faisant une boucle en $n$ étapes où à chaque étape on choisit un voisin du prédécesseur (codé en binaire). Si à un moment on arrive sur le sommet cible on accepte.

C'est NL-complet car étant donnée $M$ fonctionnant en espace logarithmique, son graphe des configurations sur un entrée $x$ a une taille polynomiale en la taille de l'entrée, et peut être produit en espace logarithmique (on a juste besoin de compteurs en binaire). □

**Proposition 6.0.7.** Access$\in$ DSPACE$(\log^2 n)$

*Démonstration.* □

**Théorème 6.0.8.** PSPACE = NPSPACE

# Références

[CL93] René Cori and Daniel Lascar. *Logique mathématique. Cours et exercices. II : Fonctions récursives, théorème de Gödel, théorie des ensembles, théorie des modèles. Préface de J.-L. Krivine.* Paris : Masson, 1993.