Cours de 2ème année de Master Recherche
Logique et Théorie du Calcul

Paul Ruet

31 octobre 2005

Ces notes de cours sont une introduction à la logique, à la calculabilité et à la complexité algorithmique. On y présente deux des principaux modèles de calcul proposés dans les années 1930, les fonctions récursives et les machines de Turing, qui reposent sur des formalismes assez différents mais ont un pouvoir expressif équivalent. On donne des bases de logique (calcul des séquents et théorème de complétude) et d'arithmétique, en vue d'énoncer et prouver les théorèmes d'indécidabilité et d'incomplétude de Church et de Gödel. On termine par ce qui peut être considéré comme la forme moderne de la théorie de la calculabilité, à savoir la complexité algorithmique, dont on étudie les principales classes.

Notations générales

$\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{R}^+$	ensembles des entiers naturels, des entiers relatifs, des réels, des réels positifs
$[x]$	partie entière du réel x
$X + Y$	union disjointe $(X \times \{1\}) \cup (Y \times \{2\})$ de X et Y
$P(X)$	ensemble des sous-ensembles de X
$\mathrm{Seq}(X)$ ou X^*	ensemble des listes (suites finies) d'éléments de X
$\lvert u \rvert$	longueur de la liste u
$u * v$ ou uv	concaténée des listes u et v
ε	liste vide

Rappels sur les cardinaux infinis

0.1 Définition (Équipotence) Deux ensembles X et Y sont dits *équipotents* s'il existe une bijection $X \xrightarrow{\sim} Y$. On le note $X \sim Y$. Lorsque X est équipotent à $\mathbb{N}$, on dit que X est *dénombrable*.

Cela définit une relation d'équivalence entre ensembles. Remarquons que si X est fini, la seule partie de X équipotente à X est l'ensemble X lui-même, mais évidemment c'est faux dans le cas infini (par exemple l'ensemble des entiers pairs est dénombrable, et il est pourtant strictement inclus dans $\mathbb{N}$).

On a $\mathbb{N} \sim \mathbb{N}^2$, et on verra des exemples de telles bijections (calculables même) dans l'exercice 1.20. Plus généralement $\mathbb{N} \sim \mathbb{N}^k$ pour tout entier non nul k. Rappelons le

0.2 Théorème (Cantor-Bernstein) Deux ensembles X et Y sont équipotents si, et seulement s'il existe une injection de X dans Y et une injection de Y dans X, si, et seulement s'il existe une surjection de X dans Y et une surjection de Y dans X.

Ainsi, on montre que $\mathbb{N} \sim \mathrm{Seq}(\mathbb{N}) = \biguplus_{n \geqslant 0} \mathbb{N}^n$: on a une injection évidente $\mathbb{N} \hookrightarrow \mathrm{Seq}(\mathbb{N})$ qui associe à un entier n la liste à un seul élément $\langle n \rangle$, et d'autre part une injection $\mathrm{Seq}(\mathbb{N}) \hookrightarrow \mathbb{N}$ qui à la liste $\langle n_1, \ldots, n_k \rangle$ associe le produit des $p_i^{n_i}$, $i = 1, \ldots, k$, où p_i est le $i$ème nombre premier.

0.3 Théorème (Cantor) $\mathbb{N} \not\sim \mathrm{P}(\mathbb{N})$.

Preuve — On procède par l'absurde. Soit $f : \mathbb{N} \xrightarrow{\sim} \mathrm{P}(\mathbb{N}), n \mapsto f_n \subseteq \mathbb{N}$ et soit $X = \{n \in \mathbb{N} \mid n \notin f_n\}$. $X \subseteq \mathbb{N}$ donc $X = f_k$ pour un certain entier k. De deux choses l'une :
- soit $k \in f_k$ et alors $k \in \{n \in \mathbb{N} \mid n \notin f_n\}$ d'où $k \notin f_k$,
- soit $k \notin f_k$ et alors $k \notin \{n \in \mathbb{N} \mid n \notin f_n\}$, ce qui signifie $k \in f_k$.

On a une contradiction dans les deux cas. ∎

Une conséquence immédiate de ces deux théorèmes est que $\mathbb{N} \not\sim \mathbb{N}^{\mathbb{N}}$, l'ensemble des fonctions de $\mathbb{N}$ dans $\mathbb{N}$: on a en effet une injection $\mathbb{N} \hookrightarrow \mathrm{P}(\mathbb{N}), n \mapsto \{n\}$, et une injection $\chi : \mathrm{P}(\mathbb{N}) \hookrightarrow \mathbb{N}^{\mathbb{N}}, A \subseteq \mathbb{N} \mapsto \chi_A \in \{0,1\}^{\mathbb{N}}$, la fonction caractéristique de A, donc si on avait une injection $f : \mathbb{N}^{\mathbb{N}} \hookrightarrow \mathbb{N}$, leur composée $f \circ \chi$ serait une injection $\mathrm{P}(\mathbb{N}) \hookrightarrow \mathbb{N}$, absurde.

Une autre conséquence est que l'ensemble $\mathbb{R}$ des réels n'est pas dénombrable. En effet :

1. $\mathbb{R} \sim I$, l'intervalle $[0,1[$, puisque $x \mapsto \tan(x\pi/2)$ est une bijection $]-1,1[\xrightarrow{\sim} \mathbb{R}^*$ et que $[-1,1[\sim I$;
2. $(x,0) \mapsto x/2, (x,1) \mapsto x/2 + 1/2$ est une bijection $I + I = I \times \{0\} \cup I \times \{1\} \xrightarrow{\sim} I$;
3. la fonction qui à $A \subset \mathbb{N}$, $A \neq \mathbb{N}$, associe le couple $(\sum_{i \geqslant 0} \chi_A(i) 2^{-i-1}, 0)$ si A est cofinie (c'est-à-dire de complémentaire fini), le couple $(\sum_{i \geqslant 0} \chi_A(i) 2^{-i-1}, 1)$ si A n'est pas cofinie, et à $\mathbb{N}$ associe $(0,1)$ par exemple, est une injection $\mathrm{P}(\mathbb{N}) \hookrightarrow I + I$;
4. par conséquent une injection $\mathbb{R} \hookrightarrow \mathbb{N}$ induirait une injection

$$\mathrm{P}(\mathbb{N}) \hookrightarrow I + I \sim I \sim \mathbb{R} \hookrightarrow \mathbb{N},$$

ce qui est impossible d'après le théorème de Cantor.

En notant $X < Y$ lorsqu'il existe une injection de X dans Y et pas d'injection de Y dans X, on a :

$$\mathbb{N} \sim \mathbb{N} \times \mathbb{N} \sim \mathbb{N}^k \sim \mathrm{Seq}(\mathbb{N}) < \mathrm{P}(\mathbb{N}) \sim \mathbb{N}^{\mathbb{N}} \sim \mathbb{R}.$$

1 Fonctions récursives

1.1 Fonctions primitives récursives

1.1 Définition (Fonctions initiales) Ce sont :
- les fonctions *constantes* : $c_p^n : \mathbb{N}^n \to \mathbb{N}$, $\vec{x} = (x_1, \ldots, x_n) \mapsto p$,
- les *projections* : $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$, $\vec{x} \mapsto x_i$,
- la fonction *successeur* : $s : \mathbb{N} \to \mathbb{N}$, $x \mapsto x + 1$.

1.2 Définition (Fonctions p.r.) La classe des *fonctions primitives récursives (p.r.)* est la plus petite classe X de fonctions de toutes arités telle que :

(R1) X contient les fonctions initiales,

(R2) X est close par *composition généralisée* : si $f : \mathbb{N}^n \to \mathbb{N}$ et les $g_i : \mathbb{N}^k \to \mathbb{N}$, $i = 1, \ldots, n$, sont dans X, alors la fonction $f \circ \langle g_1, \ldots, g_n \rangle : \mathbb{N}^k \to \mathbb{N}$, $\vec{x} \mapsto f(g_1(\vec{x}), \ldots, g_n(\vec{x}))$ est dans X,

(R3) X est close par *(définition par) récurrence primitive* : si $f : \mathbb{N}^n \to \mathbb{N}$ et $g : \mathbb{N}^{n+2} \to \mathbb{N}$ sont dans X, alors la fonction $h = \mathrm{Rec}(f,g) : \mathbb{N}^{n+1} \to \mathbb{N}$ donnée par

$$h(\vec{x}, 0) = f(\vec{x})$$
$$h(\vec{x}, y+1) = g(\vec{x}, y, h(\vec{x}, y))$$

est dans X.

Formellement, la classe des fonctions p.r. est l'intersection des classes Y de fonctions qui satisfont (R1), (R2), (R3). On peut se convaincre que les fonctions p.r. sont bien calculables au sens intuitif : on peut par exemple les programmer dans un langage comme C, Caml... On verra en section 2 un modèle formel de calcul, les machines de Turing, qui permettent effectivement de les programmer.

1.3 Définition (Variante) Soit $f : \mathbb{N}^n \to \mathbb{N}$ une fonction. On dit que $g : \mathbb{N}^k \to \mathbb{N}$ est obtenue à partir de f par *substitution triviale*, ou est une *variante* de f, s'il existe $i_1, \ldots, i_n \in \{1, \ldots, k\}$ tels que $g(x_1, \ldots, x_k) = f(u_{i_1}, \ldots, u_{i_n})$, où u_{i_r} est soit un entier soit la variable x_{i_r}.

Une variante est donc obtenue en permutant, dupliquant, effaçant des variables et en en remplaçant par des constantes.

1.4 Proposition Toute variante d'une fonction p.r. est p.r.

Preuve — Une variante de f est de la forme $f \circ \langle h_1, \ldots, h_n \rangle$, où h_r est soit $\pi^k_{i_r}$ soit c^k_a pour un certain entier a. ∎

1.5 Définition (Sommes et produits bornés) Soit $f : \mathbb{N}^{n+1} \to \mathbb{N}$. On définit les fonctions $S_f, \overline{S}_f, P_f, \overline{P}_f : \mathbb{N}^{n+1} \to \mathbb{N}$ par :

$$\begin{array}{rclcrcl} S_f(\vec{x}, y) & = & \sum_{t<y} f(\vec{x}, t) & \qquad & \overline{S}_f(\vec{x}, y) & = & \sum_{t\leqslant y} f(\vec{x}, t) \\ P_f(\vec{x}, y) & = & \prod_{t<y} f(\vec{x}, t) & & \overline{P}_f(\vec{x}, y) & = & \prod_{t\leqslant y} f(\vec{x}, t). \end{array}$$

1.6 Proposition Si f est p.r., alors $S_f, \overline{S}_f, P_f, \overline{P}_f$ sont p.r.

Preuve — On utilise le schéma (R3). On a par exemple $S_f(\vec{x}, 0) = 0$ et $S_f(\vec{x}, y+1) = f(\vec{x}, y) + S_f(\vec{x}, y)$, etc. ∎

1.7 Exercice Les fonctions suivantes sont p.r. : l'addition $\mathbb{N}^2 \to \mathbb{N}$, la multiplication, la somme ou le produit de deux fonctions p.r., le prédecesseur $p : \mathbb{N} \to \mathbb{N}$ ($0 \mapsto 0$, $x+1 \mapsto x$), la soustraction restreinte ($x, y \mapsto 0$ si $x < y$, $x-y$ sinon ; on la notera $x \dot{-} y$), $d(x, y) = |x-y|$, l'exponentielle $\mathbb{N}^2 \to \mathbb{N}$ ($x, y \mapsto x^y$), la factorielle $x \mapsto x!$, le test à zéro ($\mathrm{zero}(x) = 1$ si $x = 0$, 0 sinon) et sa variante $\overline{\mathrm{zero}}$ (définie par $\overline{\mathrm{zero}}(x) = 0$ si $x = 0$, 1 sinon), les fonctions $\min_k, \max_k : \mathbb{N}^k \to \mathbb{N}$ ($k \geqslant 1$).

1.2 Relations primitives récursives

1.8 Définition Une *relation* n-aire est un sous-ensemble R de $\mathbb{N}^n$. On note $\chi_R : \mathbb{N}^n \to \mathbb{N}$ la *fonction caractéristique* de R : $\chi_R(\vec{x}) = 1$ si $R(\vec{x})$, 0 sinon. On dit que $Q \subseteq \mathbb{N}^k$ est une *variante* de R s'il existe $i_1, \ldots, i_n \in \{1, \ldots, k\}$ tels que $Q(x_1, \ldots, x_k) \Leftrightarrow R(u_{i_1}, \ldots, u_{i_n})$, où u_{i_r} est soit un entier soit la variable x_{i_r}.

1.9 Définition (Relation p.r.) On dit qu'une relation n-aire R est p.r. si χ_R est p.r.

1.10 Proposition La classe des relations p.r. contient la relation toujours fausse $\varnothing$ et les relations n-aires toujours vraies $\mathbb{N}^n$, et est close par les opérations booléennes (intersection $\cap$, union $\cup$, différence $\setminus$). Toute variante d'une relation p.r. est p.r.

Preuve — $\chi_\varnothing = 0$, $\chi_{\mathbb{N}^n} = 1$, $\chi_{A\cap B} = \chi_A \cdot \chi_B$, $\chi_{A\cup B} = \chi_A + \chi_B \dot{-} \chi_A \cdot \chi_B$, $\chi_{A\setminus B} = \chi_A \dot{-} \chi_B$. Par ailleurs, Q est une variante de R si, et seulement si, χ_Q est une variante de χ_R. ∎

1.11 Définition (Quantification bornée) Soit $R \subseteq \mathbb{N}^{n+1}$. On dit que $Q \subseteq \mathbb{N}^n$ est obtenue à partir de R par *quantification existentielle (resp. universelle) bornée* si Q est de la forme :

$$\begin{aligned} Q(x_1,\dots,x_n) &\Leftrightarrow \exists t < x_n, R(x_1,\dots,x_n,t) \\ (\text{resp. } Q(x_1,\dots,x_n) &\Leftrightarrow \forall t < x_n, R(x_1,\dots,x_n,t)\ ). \end{aligned}$$

La borne peut être stricte ou large. L'indice de la variable quantifiée peut être différent.

1.12 Proposition La classe des relations p.r. est close par quantification bornée.

Preuve — $\chi_Q(\vec{x}) = \overline{\text{zero}}(\sum_{t<x_n} \chi_R(\vec{x},t))$, etc. ∎

1.13 Proposition (Substitution de fonctions p.r.) La classe des relations p.r. est close par substitution de fonctions p.r. : si $R \subseteq \mathbb{N}^n$ est p.r. et si $f_1,\dots,f_n : \mathbb{N}^k \to \mathbb{N}$ sont p.r., alors $Q \subseteq \mathbb{N}^k$ définie par $Q(\vec{x}) \Leftrightarrow R(f_1(\vec{x}),\dots,f_n(\vec{x}))$ est p.r.

Preuve — $\chi_Q(\vec{x}) = \chi_R(f_1(\vec{x}),\dots,f_n(\vec{x})) = (\chi_R \circ \langle f_1,\dots,f_n\rangle)(\vec{x})$. ∎

1.14 Proposition (Définition par cas) Soient $R_1,\dots,R_n \subseteq \mathbb{N}^k$ des relations p.r. 2 à 2 disjointes, et $g_1,\dots,g_{n+1} : \mathbb{N}^k \to \mathbb{N}$ des fonctions p.r. La fonction définie par :

$$f(\vec{x}) = \begin{cases} g_1(\vec{x}) & \text{si } R_1(\vec{x}) \\ \vdots & \\ g_n(\vec{x}) & \text{si } R_n(\vec{x}) \\ g_{n+1}(\vec{x}) & \text{sinon} \end{cases}$$

est p.r.

Preuve — Posons $R_{n+1} = \mathbb{N}^k \setminus \bigcup_{i=1}^n R_i$. R_{n+1} est p.r. et $f(\vec{x}) = g_1(\vec{x}) \cdot \chi_{R_1}(\vec{x}) + \cdots + g_n(\vec{x}) \cdot \chi_{R_{n+1}}(\vec{x}) + g_{n+1}(\vec{x}) \cdot \chi_{R_n}(\vec{x})$. f est donc somme de produits de fonctions p.r., et est donc p.r. ∎

On rappelle qu'une fonction *partielle* h de A dans B, notée $h : A \rightharpoonup B$, est une fonction $A' \to B$ où $A' \subseteq A$. Si $h_1, h_2 : A \rightharpoonup B$, l'écriture $h_1(a_1) = h_2(a_2)$ signifie : $a_1 \in \text{dom}(h_1) \Leftrightarrow a_2 \in \text{dom}(h_2)$ et dans ce cas $h_1(a_1) = h_2(a_2)$. On dit que $h(a)$ *converge* quand $a \in \text{dom}(h)$, et on note aussi $h(a)\downarrow$.

1.15 Définition (Minimalisation) Soit $R \subseteq \mathbb{N}^{n+1}$. On définit la fonction partielle $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ par

$$f(\vec{x}) = (\mu y).R(\vec{x},y) = \begin{cases} \text{le premier } y \in \mathbb{N} \text{ tel que } R(\vec{x},y) & \text{s'il existe,} \\ \text{indéfini} & \text{sinon.} \end{cases}$$

On verra que si R est p.r., même si pour tout $\vec{x}$ il existe un y tel que $R(\vec{x},y)$, la fonction f ci-dessus n'est pas nécessairement p.r., encore qu'intuitivement calculable.

1.16 Définition (Minimalisation bornée) Pour $R \subseteq \mathbb{N}^{n+1}$, on définit la fonction totale $f : \mathbb{N}^{n+1} \to \mathbb{N}$ par

$$f(\vec{x},y) = (\mu t < y).R(\vec{x},t) = \begin{cases} \text{le premier } t < y \text{ tel que } R(\vec{x},t) & \text{s'il existe,} \\ y & \text{sinon.} \end{cases}$$

On remarque qu'on peut alors définir $(\mu t \leqslant y).R(\vec{x},t) = (\mu t < y+1).R(\vec{x},t)$.

1.17 Proposition Si $R \subseteq \mathbb{N}^{n+1}$ est p.r., alors la fonction $f(\vec{x},y) = (\mu t < y).R(\vec{x},t)$ est p.r.

Preuve — Schéma (R3) : $f(\vec{x},0)=0$ et $f(\vec{x},y+1)=f(\vec{x},y)$ si $R(\vec{x},f(\vec{x},y))$, $y+1$ sinon, donc $f(\vec{x},y+1)=h(\vec{x},y,f(\vec{x},y))$ où h est définie par cas. ∎

On remarque que si $g:\mathbb{N}^{n+1}\to\mathbb{N}$ est p.r., la fonction qui à $\vec{x},y$ associe $(\mu t<g(\vec{x},y)).R(\vec{x},t)$ est aussi p.r. (substitution).

1.18 Exercice 1. Les relations $=,\neq,<,\leqslant,\not<,\not\leqslant$ sont p.r.

2. Soit $R\subseteq\mathbb{N}^n$ et $g_R:\mathbb{N}^n\to\mathbb{N}$ définie par $g_R(\vec{x})=0$ si $R(\vec{x})$, 1 sinon. Est-ce que R est p.r. si, et seulement si, g_R l'est ? Soit $h_R:\mathbb{N}^n\to\mathbb{N}$ une fonction totale vérifiant $h_R(\vec{x})=0$ si, et seulement si, $R(\vec{x})$. Est-ce que R est p.r. si, et seulement si, h_R l'est ?
3. Les relations et fonctions suivantes sont p.r. : "x divise y", $\psi(x)=$ le nombre de diviseurs de x si $x\neq 0$ et 0 sinon, "x est un nombre premier", "x est pair", "x est impair", $f(x)=$ le nombre de nombres premiers $\leqslant x$.
4. Montrer que les fonctions et relations suivantes sont p.r. : le quotient ($x\operatorname{div} y=0$ si $y=0$, $[x/y]$ sinon) et le reste ($x\bmod y=x\dot{-}y\cdot(x\operatorname{div}y)$) de la division euclidienne, $n\mapsto[\sqrt{n}]$, $n,k\mapsto[\sqrt[k]{n}]$, $k\geqslant 1$, $n\mapsto[n\sqrt{2}]$, $x\mapsto 2x$ si x est un carré et $2x+1$ sinon, "x est la somme de deux carrés", l'indicatrice d'Euler ($\phi(n)=$ le nombre d'éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$), $n\mapsto p_n$ (le n-ème nombre premier), $x\mapsto[\log_a(x)]$ (a entier >1), $g(n)=$ la n-ème décimale du développement en base 10 de $\sqrt{3}$, $f(x)=0$ si la suite 0491269660 apparaît dans le développement en base 10 de π avant la x-ème décimale et 1 sinon.

1.3 Codages des suites

1.19 Définition (Codage des couples) On dit qu'une fonction $J:\mathbb{N}^2\to\mathbb{N}$ est une *fonction de codage des couples* si elle est bijective et croissante en chaque argument, c'est-à-dire pour tous $x,y\in\mathbb{N}$, $J(x,y)<J(x+1,y)$ et $J(x,y)<J(x,y+1)$.

1.20 Problème 1. Les fonctions p.r. suivantes sont-elles des fonctions de codage ?

$$\begin{aligned}J_1(x,y)&=2^x\cdot(2y+1)\dot{-}1\\ J_2(x,y)&=2^x\cdot 3^{xy}\\ J_3(x,y)&=(x+y)(x+y+1)/2+y\\ J_4(x,y)&=([(x+1)/2]+y)^2+x.\end{aligned}$$

2. Pour chaque fonction de codage trouvée :
   (a) décrire un algorithme de calcul des fonctions réciproques (appelées aussi *fonctions de décodage*) et constituées par le couple des deux fonctions $\pi_1^2\circ J^{-1}$ et $\pi_2^2\circ J^{-1}$;
   (b) les fonctions de décodage sont-elles p.r. ?

Correction —

1. Les 4 fonctions sont toutes strictement croissantes en chaque argument, donc il suffit de voir lesquelles sont bijectives.
   - J_2 n'est clairement pas bijective puisque son image contient uniquement des multiples de 2 et 3, donc ce n'est pas une fonction de codage.
   - J_1 est bijective : étant donné un entier z, on décompose $z+1$ en facteurs premiers ; soit x la puissance de 2 ; le reste est un nombre impair, donc de la forme $2y+1$ pour un certain entier y ; partant d'un z quelconque, y et z définis ainsi parcourent tous les entiers, et on voit que $J_1(x,y)=z$.

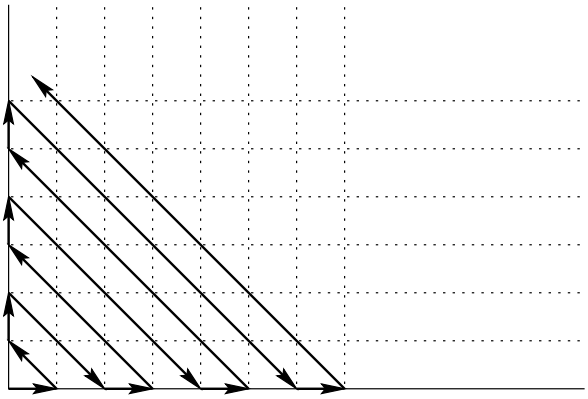


Fig. 1 – L'inverse du codage J_3.

– J_3 est bijective : étant donné un entier z, on cherche x et y tels que $J_3(x,y) = z$; en posant $s = x + y$, on a à résoudre $s(s+1)/2 + y = z$, soit $z - s(s+1)/2 = y \leqslant s$, c'est-à-dire

$$z - s \leqslant s(s+1)/2 \leqslant z\,;$$

il existe s tel que $s(s+1)/2 \leqslant z$, par exemple $s = 0$; maintenant s est maximal si, et seulement si, $z < (s+1)(s+2)/2 = s(s+1)/2 + s + 1$ si, et seulement si, $z \leqslant s(s+1)/2 + s$ si, et seulement si, $z - s \leqslant s(s+1)/2$; donc en prenant s maximal, on peut poser

$$y = z - s(s+1)/2 \quad \text{et} \quad x = s - y,$$

et c'est la seule solution. Donc J_3 est bien un codage. C'est le "fameux" codage qui balaie le plan comme sur la figure 1.

– J_4 est bijective : étant donné un entier z, on cherche x et y tels que $J_4(x,y) = z$; en posant $c = y + [(x+1)/2]$, on a à résoudre $c^2 + x = z$, soit $c^2 \leqslant z$ avec la condition $[(x+1)/2] \leqslant c$; comme $(x-1)/2 < [(x+1)/2]$, on a $x < 2c + 1$ et donc $z < c^2 + 2c + 1 = (c+1)^2$; d'où

$$c^2 \leqslant z < (c+1)^2,$$

et en prenant c maximal, on peut poser

$$x = z - c^2 \quad \text{et} \quad y = c - [(x+1)/2],$$

et c'est la seule solution. Donc J_4 est bien un codage.

2. Pour chaque fonction de codage J, les fonctions réciproques sont

$$\begin{aligned} \pi_1^2 \circ J^{-1} : z &\mapsto (\mu x \leqslant z).(\exists y \leqslant z, z = J(x,y)) \\ \pi_2^2 \circ J^{-1} : z &\mapsto (\mu y \leqslant z).(\exists x \leqslant z, z = J(x,y)). \end{aligned}$$

J est bijective, et par ailleurs, puisque J est croissante, on a $x \leqslant J(x,y)$ et $y \leqslant J(x,y)$ pour tous x, y, donc le x et le y dans les deux formules existent et sont uniques. On en déduit de plus que les fonctions de décodage sont p.r.

Avec une fonction de codage des couples, on peut bien sûr coder les suites finies de longueur fixée.

1.21 Proposition Soit J une fonction de codage des couples. On définit des fonctions $J^{(n)} : \mathbb{N}^n \to \mathbb{N}$ pour tout $n \geqslant 1$ de la manière suivante : $J^{(1)} = \mathrm{id}$, et pour tout $n \geqslant 1$, $J^{(n+1)}(x_1, \ldots, x_{n+1}) = J(J^{(n)}(x_1, \ldots, x_n), x_{n+1})$. Alors pour tout $n \geqslant 1$, $J^{(n)}$ est une bijection p.r., croissante en chaque argument, et telle que $x_i \leqslant J^{(n)}(x_1, \ldots, x_n)$ pour tout i, $1 \leqslant i \leqslant n$. De plus, les fonctions réciproques $\pi_i^n \circ (J^{(n)})^{-1} : \mathbb{N} \to \mathbb{N}$ $(1 \leqslant i \leqslant n)$ sont p.r.

Preuve — Les assertions sur $J^{(n)}$ se montrent par récurrence sur n. En ce qui concerne les réciproques, $\pi_i^n \circ (J^{(n)})^{-1}(y)$ vaut

$$(\mu x_i \leqslant y).(\exists x_1 \leqslant y, \ldots, \exists x_{i-1} \leqslant y, \exists x_{i+1} \leqslant y, \ldots, \exists x_n \leqslant y, y = J^{(n)}(x_1, \ldots, x_n)).$$

∎

1.22 Proposition (Récurrence croisée) Soient $f_1, f_2 : \mathbb{N}^{n+1} \to \mathbb{N}$ définies par :

$$\begin{aligned} f_1(\vec{x}, 0) &= g_1(\vec{x}) \\ f_1(\vec{x}, y+1) &= h_1(\vec{x}, y, f_1(\vec{x}, y), f_2(\vec{x}, y)) \\ f_2(\vec{x}, 0) &= g_2(\vec{x}) \\ f_2(\vec{x}, y+1) &= h_2(\vec{x}, y, f_1(\vec{x}, y), f_2(\vec{x}, y)). \end{aligned}$$

Si g_1, g_2, h_1, h_2 sont p.r., alors f_1, f_2 sont p.r.

Preuve — Soit $F : \mathbb{N}^{n+1} \to \mathbb{N}$ définie par $F(\vec{x}, y) = J(f_1(\vec{x}, y), f_2(\vec{x}, y))$, où J est un codage p.r. des couples. On vérifie par récurrence sur y que F est p.r. Or $f_1 = \pi_1^2 \circ J^{-1} \circ F$ et $f_2 = \pi_2^2 \circ J^{-1} \circ F$. ∎

Les suites finies sont notées $\langle n_0, \ldots, n_{l-1} \rangle$.

1.23 Définition (Codage des suites) Un *codage des suites (finies)* est une surjection $\sigma : \mathbb{N} \to \mathrm{Seq}(\mathbb{N})$.

On a évidemment :

1.24 Lemme Un codage des suites σ est déterminé par la donnée des deux fonctions suivantes : $\mathrm{lg}_\sigma : \mathbb{N} \to \mathbb{N}, n \mapsto$ la longueur de $\sigma(n)$, et $\mathrm{ap}_\sigma : \mathbb{N}^2 \to \mathbb{N}$ définie par :

$$(n, i) \mapsto \begin{cases} \sigma(n)(i) & \text{si } i < \mathrm{lg}_\sigma(n) \\ 0 & \text{sinon.} \end{cases}$$

1.25 Définition On dit d'un codage des suites σ qu'il est un *codage p.r. des suites* lorsque les fonctions associées lg_σ et ap_σ sont p.r.

1.26 Exemple 1. Soit $J : \mathbb{N}^2 \to \mathbb{N}$ un codage des couples qui est p.r. On pose $G = \pi_1^2 \circ J^{-1}$ et $D = \pi_2^2 \circ J^{-1}$:

$$\sigma(n) = \begin{cases} \left(J^{(G(n))}\right)^{-1}(D(n)) & \text{si } G(n) \neq 0 \\ \varepsilon & \text{sinon.} \end{cases}$$

Alors σ est un codage des suites. Ce codage est p.r. car les fonctions associées $\mathrm{lg}_\sigma = G$ et ap_σ, donnée par

$$\mathrm{ap}_\sigma(n, i) = \begin{cases} 0 & \text{si } G(n) = 0 \\ J^{(G(n))}(D(n)) & \text{si } i < G(n) \\ 0 & \text{sinon,} \end{cases}$$

sont p.r.

2. Soit $\sharp\langle n_0, \ldots, n_{k-1} \rangle = 2^{1+n_0} 3^{1+n_1} \cdots p_{k-1}^{1+n_{k-1}}$, où $p_0, p_1, \ldots$ est une énumération croissante des nombres premiers. On convient que $\sharp\varepsilon = 0$. La fonction $\sharp : \mathrm{Seq}(\mathbb{N}) \to \mathbb{N}$ est clairement injective. Si l'on pose

$$\sigma(n) = \begin{cases} \sharp^{-1}(u) & \text{si } n \text{ est de la forme } \sharp u \\ \varepsilon & \text{sinon,} \end{cases}$$

alors σ est un codage p.r. des suites.

1.27 Proposition Soit σ est un codage p.r. des suites. Les relations suivantes sont p.r. :

$$\begin{aligned} I_\sigma(n,m) &\quad \text{si, et seulement si,} \quad \sigma(n) \text{ est un segment initial de } \sigma(m),\\ E_\sigma(n,m) &\quad \text{si, et seulement si,} \quad \sigma(n) = \sigma(m),\\ C_\sigma(n,m,p) &\quad \text{si, et seulement si,} \quad \sigma(p) = \sigma(n) * \sigma(m). \end{aligned}$$

Preuve — $I_\sigma(n,m)$ si, et seulement si, $lg_\sigma(n) \leqslant lg_\sigma(m)$ et pour tout $i < lg_\sigma(n)$, $ap_\sigma(n,i) = ap_\sigma(m,i)$. $E_\sigma(n,m)$ si, et seulement si, $I_\sigma(n,m)$ et $I_\sigma(m,n)$. $C_\sigma(n,m,p)$ si, et seulement si, $lg_\sigma(p) = lg_\sigma(n) + lg_\sigma(m)$, pour tout $i < lg_\sigma(n)$, $ap_\sigma(p,i) = ap_\sigma(n,i)$, et pour tout $j < lg_\sigma(m)$, $ap_\sigma(p, j + lg_\sigma(n)) = ap_\sigma(m,j)$. ∎

1.28 Exercice Soit σ un codage des suites. Montrer que l'application qui à $u \in \mathrm{Seq}(\mathbb{N})$ associe le *code canonique de* u *(pour* σ*)*, à savoir le plus petit n tel que $\sigma(n) = u$, est injective. Montrer que si σ est p.r., alors la fonction $k_\sigma : \mathbb{N} \to \mathbb{N}$ qui à n associe le code canonique de $\sigma(n)$ est p.r.

La proposition suivante est une application de l'existence de codages p.r. des suites.

1.29 Définition (Bon codage p.r.) Soit $\sigma : \mathrm{Seq}(\mathbb{N}) \to \mathbb{N}$ est un codage p.r. des suites. On dit que σ est un *bon codage p.r.* s'il existe une fonction p.r. $b : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ telle que $\sigma_{b(n,x)} = \sigma_n * \langle x \rangle$, où l'on note $\sigma_n = \sigma(n)$.

1.30 Lemme Soit σ l'un des deux codages de 1.26. Alors :
- σ est un bon codage p.r.
- Pour tout $x \in \mathbb{N}$, $x \leqslant$ au code canonique de $\langle x \rangle$.
- Pour tous $n_0, \ldots, n_p \in \mathbb{N}$ et tous i, j, $0 \leqslant i \leqslant j \leqslant p$, le code canonique de $\langle n_i, \ldots, n_j \rangle$ est $\leqslant$ au code canonique de $\langle n_0, \ldots, n_p \rangle$.
- Il existe une fonction p.r. $\mathrm{conc}_\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ telle que $\sigma_n * \sigma_m = \sigma_{\mathrm{conc}_\sigma(n,m)}$.

Preuve — Pour le deuxième codage, on pose :

$$b(n,x) = \begin{cases} 2^{1+x} & \text{si } n \text{ est un code de } \varepsilon,\\ n \cdot p_{lg_\sigma(n)}^{1+x} & \text{sinon.} \end{cases}$$

On laisse au lecteur le soin de montrer les autres assertions. ∎

1.31 Définition (Invariance pour un codage) Soient $f : \mathbb{N} \to \mathbb{N}$ et $\sigma : \mathbb{N} \to \mathrm{Seq}(\mathbb{N})$ est un codage p.r. des suites. On dit que f est *invariante pour le codage* σ lorsque pour tous entiers m, n, $\sigma_m = \sigma_n$ implique $f(m) = f(n)$.

1.32 Proposition (Récurrence complète) Soit $g : \mathbb{N} \to \mathbb{N}$ une fonction p.r. invariante pour un bon codage p.r. σ. Alors la fonction f définie par *récurrence complète* par

$$f(x) = g\big(\langle f(0), \ldots, f(x-1) \rangle_\sigma\big),$$

où $\langle n_0, \ldots, n_p \rangle_\sigma$ dénote le code canonique de $\langle n_0, \ldots, n_p \rangle$ pour σ, est p.r.

Preuve — Posons $F(x) = \langle f(0), \ldots, f(x) \rangle_\sigma$. On a alors :

$$\begin{aligned} F(0) &= \langle f(0) \rangle_\sigma\\ F(x+1) &= \langle f(0), \ldots, f(x), f(x+1) \rangle_\sigma = (k \circ b)\big(\langle f(0), \ldots, f(x) \rangle_\sigma, f(x+1)\big), \end{aligned}$$

où b est la fonction p.r. associée au bon codage σ et k est la fonction p.r. qui à n associe le code canonique de σ_n. Comme $F(x+1) = (k \circ b)\big(F(x), (g \circ F)(x)\big)$, F est p.r. Or $f(x) = ap_\sigma(F(x), x)$, d'où f est p.r. ∎

1.33 Exercice 1. On appelle fonction de Fibonacci la fonction $F : \mathbb{N} \to \mathbb{N}$ définie par $F(0) = F(1) = 1$ et $F(n+2) = F(n+1) + F(n)$. Montrer que F est p.r.

2. Soit $u : \mathbb{N}^{n+1} \to \mathbb{N}$ une fonction p.r. telle que $u(\vec{x}, y) < y$ pour tout $y > 0$. Soient $g : \mathbb{N}^n \to \mathbb{N}$ et $h : \mathbb{N}^{n+2} \to \mathbb{N}$ des fonctions p.r. La fonction suivante est p.r. :

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, y) &= h(\vec{x}, y, f(\vec{x}, u(\vec{x}, y))) \quad \text{si } y > 0. \end{aligned}$$

1.4 Fonctions récursives

1.34 Problème (Fonction d'Ackermann) Soit $A : \mathbb{N}^2 \to \mathbb{N}$ la fonction définie par :

$$\begin{aligned} A(0, n) &= n + 1 \\ A(m+1, 0) &= A(m, 1) \\ A(m+1, n+1) &= A(m, A(m+1, n)). \end{aligned}$$

Notons A_p la fonction $\mathbb{N} \to \mathbb{N}$ définie par $A_p(n) = A(p, n)$. On remarque que A est bien calculable au sens intuitif (on le verra formellement en section 2). Démontrer les propriétés suivantes.

1. A_0, A_1, A_2, A_3 satisfont $A_0(n) = n + 1$, $A_1(n) = n + 2$, $A_2(n) = 2n + 3$ et $A_3(n) = k \cdot 2^n + c$ (k et c à déterminer).
2. Pour tous n, x, $A_n(x) > x$.
3. Pour tout p, les fonctions A_p sont strictement croissantes.
4. Pour tous n, x, $A_{n+1}(x) \geqslant A_n(x)$.
5. Pour tous n, x, $A_{n+1}(x) \geqslant A_n(x+1)$.
6. Pour tous n_1, n_2 il existe m tel que pour tout x, $A(n_1, A(n_2, x)) \leqslant A(m, x)$.
7. Pour tous n_1, n_2 il existe m tel que pour tout x, $A(n_1, x) + A(n_2, x) \leqslant A(m, x)$.
8. Soit m un entier. On dira qu'une fonction $f : \mathbb{N}^p \to \mathbb{N}$ est m-bornée si pour tous $x_1, \ldots, x_p$, $f(x_1, \ldots, x_p) \leqslant A(m, x_1 + \cdots + x_p)$. Montrer que pour toute fonction p.r. f, il existe un entier m tel que f est m-bornée.
9. En déduire que A n'est pas p.r.

Correction — A est bien calculable au sens intuitif : en effet, la première ligne permet de calculer $A(0, n)$ pour tout $n \in \mathbb{N}$; maintenant si on sait calculer $A(m, n)$ pour tout $n \in \mathbb{N}$ et pour m fixé, alors 1) on sait calculer $A(m+1, 0)$ par la deuxième ligne, 2) si on sait calculer $A(m+1, n)$ la troisième ligne permet de calculer $A(m+1, n+1)$, par conséquent on sait calculer $A(m+1, n)$ pour tout n. Remarquons aussi que la définition de A n'a pas la forme d'une récurrence primitive.

1. (a) Pour A_0 c'est évident.
   (b) $A_1(0) = A_0(1) = 2$ et si $A_1(n) = n + 2$, on a bien $A_1(n+1) = A_0(A_1(n)) = A_1(n) + 1 = n + 3$.
   (c) $A_2(0) = A_1(1) = 3$ et si $A_2(n) = 2n + 3$, on a bien $A_2(n+1) = A_1(A_2(n)) = A_2(n) + 2 = 2n + 3 = 2(n+1) + 3$.
   (d) $A_3(0) = 5 = k + c$ et si $A_3(n) = k \cdot 2^n + c$, on a $A_3(n+1) = A_2(A_3(n)) = 2(k \cdot 2^n + c) + 3$, qui est bien de la forme $k \cdot 2^{n+1} + c$ si, et seulement si, $2c + 3 = c$ et $k + c = 5$, c'est-à-dire $k = 8$ et $c = -3$.

   On peut remarquer plus généralement que

$$A_{n+1}(x) = \underbrace{A_n \circ \cdots \circ A_n}_{x+1}(1) = (A_n)^{x+1}(1).$$

2. Si $A_n(x) > x$ pour tout x, alors $A_{n+1}(0) = A_n(1) \geqslant 1 > 0$ et $A_{n+1}(x) > x$ implique $A_{n+1}(x+1) = A_n(A_{n+1}(x)) > A_{n+1}(x)$ d'où $A_{n+1}(x+1) \geqslant A_{n+1}(x) + 1 > x+1$. On a prouvé que $\forall x, A_n(x) > x$ implique $\forall x, A_{n+1}(x) > x$. Comme c'est vrai pour $n = 0$, on en déduit que pour tous n, x, $A_n(x) > x$.
3. A_0 est strictement croissante. Si A_p est strictement croissante, alors d'après le 2, $A_{p+1}(x+1) = A_p(A_{p+1}(x)) > A_{p+1}(x)$ et donc A_{p+1} est strictement croissante.
4. $A_{n+1}(0) = A_n(1) > A_n(0)$ d'après le 3. D'après le 2, $A_{n+1}(x) > x$ donc $A_{n+1}(x) \geqslant x+1$. Puisque A_n est strictement croissante, on a $A_{n+1}(x+1) = A_n(A_{n+1}(x)) \geqslant A_n(x+1)$.
5. $A_{n+1}(0) = A_n(0+1)$. Si $A_{n+1}(x) \geqslant A_n(x+1)$, alors $A_{n+1}(x) > x+1$ d'après le 2, d'où $A_{n+1}(x) \geqslant x+2$, et comme A_n est croissante, $A_{n+1}(x+1) = A_n(A_{n+1}(x)) \geqslant A_n(x+2)$, d'où le résultat.
6. On devine que $A_{n_1} \circ A_{n_2} \leqslant A_{A(n_2,n_1)}$ en essayant avec $n_2 = 0, 1 \ldots$. On le prouve par récurrence sur n_2. Pour $n_2 = 0$, on a bien $A_{n_1} \circ A_0 \leqslant A_{n_1+1} = A_{A(0,n_1)}$ d'après le 5. On montre que $A_{n_1} \circ A_{n_2} \leqslant A_{A(n_2,n_1)}$ implique $A_{n_1} \circ A_{n_2+1} \leqslant A_{A(n_2+1,n_1)}$. On procède par cas. Pour $x = 0$, on a :

$$\begin{array}{rcll} A_{n_1} \circ A_{n_2+1}(0) & = & A_{n_1} \circ A_{n_2}(1) & \\ & \leqslant & A_{A(n_2,n_1)}(1) & \text{par hyp. de récurrence} \\ & \leqslant & A_{A(n_2,n_1)+1}(0) & \text{par 5} \\ & \leqslant & A_{A(n_2+1,n_1)}(0) & \text{par 5 et 3.} \end{array}$$

Pour $x+1$, on a :

$$\begin{array}{rcll} A_{n_1} \circ A_{n_2+1}(x+1) & = & A_{n_1} \circ A_{n_2} \circ A_{n_2+1}(x) & \\ & \leqslant & A_{A(n_2,n_1)} \circ A_{n_2+1}(x) & \text{par hyp. de récurrence} \end{array}$$

et par ailleurs :

$$\begin{array}{rcll} A_{A(n_2+1,n_1)}(x+1) & \geqslant & A_{A(n_2,n_1)+1}(x+1) & \text{par 5 et 3} \\ & = & A_{A(n_2,n_1)} \circ A_{A(n_2,n_1)+1}(x). & \end{array}$$

Pour conclure il suffit donc de remarquer que $A(n_2, n_1)+1 \geqslant A(0, n_2+n_1)+1 = n_2+n_1+2 > n_2+1$.
7. $A_{n_1}(x)+A_{n_2}(x) < 2A_{n_1+n_2}(x)+3 = (A_2 \circ A_{n_1+n_2}(x)$, donc $A_{n_1}+A_{n_2} < A_2 \circ A_{n_1+n_2} \leqslant A_{A(n_1+n_2,2)}$ d'après le 6.
8. (a) Les fonctions initiales sont m-bornées : $c_0^n(\vec{x}) = 0 \leqslant A_0(\sum \vec{x})$ par exemple, $s(x) = x+1 = A_0(x)$ et $\pi_i^n(\vec{x}) = x_i < \sum \vec{x} + 1 = A_0(\sum \vec{x})$.
   (b) Si f est m-bornée et les g_i sont m_i-bornées, alors

$$f \circ \langle g_1, \ldots, g_n \rangle(\vec{x}) \leqslant A_m\left(\sum_1^n g_i(\vec{x})\right) \leqslant A_m\left(\sum_1^n A_{m_i}\left(\sum \vec{x}\right)\right),$$

   et donc $f \circ \langle g_1, \ldots, g_n \rangle$ est m'-bornée, avec m' donné par les résultats 6 et 7.
   (c) Soit $h = \mathrm{Rec}(f, g)$ avec f m-bornée et g k-bornée : d'une part $h(\vec{x}, 0) \leqslant A_m(\sum \vec{x}) \leqslant A_l(0+\sum \vec{x})$ dès que $l \geqslant m$. D'autre part, si l'on pose $y = n + \sum \vec{x}$, alors $h(\vec{x}, n) \leqslant A_l(y)$ implique $h(\vec{x}, n+1) \leqslant A_k(A_l(y)+y)$ et donc $h(\vec{x}, n+1) \leqslant A_l(y+1)$ dès que $A_k(A_l(y)+y) \leqslant A_l(y+1)$; c'est le cas si $l-1 \geqslant A(2,k)$ car alors

$$\begin{array}{rcl} A_k(A_l(y)+y) & < & A_k(2A_l(y)+3) \\ & = & (A_k \circ A_2)(A_l(y)) \\ & \leqslant & A_{l-1}(A_l(y)) \\ & = & A_l(y+1). \end{array}$$

   En prenant $l = \max(m, 1+A(2,k))$, h est donc l-bornée.
9. On procède par l'absurde en utilisant un argument de diagonalisation : si A était p.r., la fonction $f : x \mapsto A(x,x)$ le serait aussi, donc d'après le 8, il existerait un entier m tel que pour tout x, $f(x) \leqslant A(m,x)$, en particulier pour $x = m+1$, on aurait $f(m+1) = A(m+1, m+1) \leqslant A(m, m+1)$. Or d'après 5, $A_{m+1}(m+1) \geqslant A_m(m+1+1)$ d'où $A_{m+1}(m+1) > A_m(m+1)$: contradiction.

1.35 Définition (Fonctions récursives) La classe des *fonctions récursives partielles* est la plus petite classe X de fonctions partielles de toutes arités close par (R1), (R2), (R3) et telle que :

(R4) X est close par *minimalisation* : si $g : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ est dans X, la fonction f définie par $f(\vec{x}) = (\mu y).(g(\vec{x}, y) = 0)$ est dans X.

Remarquons qu'on considère des fonctions *partielles* dans la définition uniquement à cause de (R4).

1.36 Remarque (Sur les fonctions partielles)
- L'égalité $f = g$ entre fonctions partielles signifie que f et g ont le même domaine de définition et prennent la même valeur en chaque point du domaine. Des égalités comme $f = (f + g) \dot{-} g$ entre fonctions partielles ne sont donc plus vraies.
- De même, si $g_1, \ldots, g_n : \mathbb{N}^k \rightharpoonup \mathbb{N}$ et $h : \mathbb{N}^n \rightharpoonup \mathbb{N}$, la fonction composée $f(\vec{x}) = h(g_1(\vec{x}), \ldots, g_n(\vec{x}))$ est définie si, et seulement si, $g_1(\vec{x}) \downarrow, \ldots, g_n(\vec{x}) \downarrow$ et $h(g_1(\vec{x}), \ldots, g_n(\vec{x})) \downarrow$.
- Si $g : \mathbb{N}^n \rightharpoonup \mathbb{N}$, $h : \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$, et si la fonction f est définie par récurrence par $f(\vec{x}, 0) = g(\vec{x})$ et $f(\vec{x}, y+1) = h(\vec{x}, y, f(\vec{x}, y))$, on voit que $f(\vec{x}, y) \downarrow$ implique $f(\vec{x}, y-1) \downarrow, \ldots, f(\vec{x}, 0) \downarrow$.
- Enfin, soit $g : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, la fonction définie par $f(\vec{x}) = (\mu y).(g(\vec{x}, y) = 0)$ diverge si par exemple $g(\vec{x}, 0)$ diverge, $g(\vec{x}, 1) = 2$, $g(\vec{x}, 2) = 0$; f est donnée par : $f(\vec{x}) \downarrow$ et $= t$ si, et seulement si, $g(\vec{x}, 0) \downarrow$ et $\neq 0$, ..., $g(\vec{x}, t-1) \downarrow$ et $\neq 0$, $g(\vec{x}, t) \downarrow$ et $= 0$.

1.37 Définition (Relations récursives) Une relation R est *récursive* si χ_R l'est.

1.38 Lemme Soit $S \subseteq \mathbb{N}^n$ une relation récursive, alors il existe une fonction récursive partielle $K_S : \mathbb{N}^n \rightharpoonup \mathbb{N}$ telle que $K_S(\vec{x}) \downarrow \Leftrightarrow \vec{x} \in S$, et dans ce cas $K_S(\vec{x}) = 1$.

Preuve — $K_S(\vec{x}) = (\mu y).(y = 1 \text{ et } S(\vec{x}))$. ∎

1.39 Proposition Toute fonction p.r. est récursive (et totale). La classe des fonctions et relations récursives est close pour les opérations déjà étudiées pour la classe des fonctions p.r. (pour la substitution dans les relations, on ne substitue que par des fonctions totales).

Preuve — La première assertion est évidente. Les propriétés de clôture sont démontrées comme pour les fonctions p.r., sauf pour ce qui suit :

1. Définition par cas. $f(\vec{x}) = g_1(\vec{x})$ si $S(\vec{x})$, $g_2(\vec{x})$ sinon. Dans cette définition on veut : pour $\vec{x} \in S$, $f(\vec{x}) = g_1(\vec{x})$, donc $f(\vec{x}) \downarrow \Leftrightarrow g_1(\vec{x}) \downarrow$, et pour $\vec{x} \notin S$, $f(\vec{x}) = g_2(\vec{x})$, donc $f(\vec{x}) \downarrow \Leftrightarrow g_2(\vec{x}) \downarrow$. D'après le lemme 1.38 il y a 3 possibilités :
   (a) $\operatorname{dom} g_2 = \varnothing$: $f(\vec{x}) = g_1(\vec{x}) \cdot K_S(\vec{x})$ et f est bien récursive,
   (b) $\operatorname{dom} g_1 = \varnothing$: similaire,
   (c) sinon, soient $\vec{a_1} \in \operatorname{dom} g_1$ et $\vec{a_2} \in \operatorname{dom} g_2$, et posons $I_1(\vec{x}) = \vec{x}$ si $\vec{x} \in S$, $\vec{a_1}$ sinon, et $I_2(\vec{x}) = \vec{x}$ si $\vec{x} \notin S$, $\vec{a_2}$ sinon. I_1 et I_2 sont récursives totales (à valeurs vectorielles) : définition par cas triviale. On a $f(\vec{x}) = (g_1 \circ I_1)(\vec{x}) \cdot \chi_S(\vec{x}) + (g_2 \circ I_2)(\vec{x}) \cdot \chi_{\mathbb{N}^n \setminus S}(\vec{x})$ donc f est récursive.
2. Minimalisation bornée. Soit g une fonction récursive. Dans la définition $f(\vec{x}, t) = (\mu y < t).(g(\vec{x}, y) = 0)$, $f(\vec{x}, t) = y_0$ si, et seulement si,

   $$(y_0 < t, \text{ et } g(\vec{x}, 0) \downarrow \neq 0, \ldots, g(\vec{x}, y_0 - 1) \downarrow \neq 0, \text{ et } g(\vec{x}, y_0) = 0) \quad \text{ou}$$
   $$(y_0 = t, \text{ et } g(\vec{x}, 0) \downarrow \neq 0, \ldots, g(\vec{x}, y_0 - 1) \downarrow \neq 0).$$

   Par exemple, si $\operatorname{dom} g = \varnothing$, $f(\vec{x}, 0) = 0$. Maintenant, on pose $g^+(\vec{x}, y, t) = g(\vec{x}, y)$ si $y < t$, 0 sinon : g^+ est récursive (définition par cas) et $f(\vec{x}, t) = (\mu y).(g^+(\vec{x}, y, t) = 0)$, donc f est récursive.

∎

On a vu que la fonction d'Ackermann n'est pas p.r. On verra en section 2 qu'elle est en revanche Turing-calculable (exemple 2.23), et que la classe des fonctions Turing-calculables concide avec celle des fonctions récursives (théorème 2.29). Donc la fonction d'Ackermann est récursive et on a :

1.40 Théorème La classe des fonctions p.r. est strictement incluse dans celle des fonctions récursives.

1.41 Proposition Si $R \subseteq \mathbb{N}^{n+1}$ est récursive, alors la fonction $f(\vec{x}) = (\mu y).R(\vec{x}, y)$ est récursive.

Preuve — $f(\vec{x}) = (\mu y).(1 \dot{-} \chi_R(\vec{x}, y) = 0)$. ∎

1.42 Exercice Un entier p est appelé premier jumeau si p et $p+2$ sont premiers. Une des plus anciennes conjectures non prouvées affirme qu'il y a une infinité de premiers jumeaux. Soit $T(n)$ le n-ième nombre premier jumeau. Montrer que T est une fonction récursive partielle, totale si la conjecture est vraie.

On va maintenant montrer qu'à condition d'étendre un tout petit peu la classe des fonctions initiales, on peut se passer de la récurrence primitive dans la construction des fonctions récursives. Pour cela on introduit le codage de Gödel des suites, qui repose sur un lemme d'algèbre rappelé ci-dessous.

1.43 Lemme (Lemme chinois) Si p et q sont des entiers premiers entre eux, on a un isomorphisme $\mathbb{Z}/pq\mathbb{Z} \cong \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$. Plus généralement, si $d_1, \ldots, d_n$ sont des entiers deux à deux premiers entre eux, on a un isomorphisme $\mathbb{Z}/d_1 \cdots d_n\mathbb{Z} \cong \mathbb{Z}/d_1\mathbb{Z} \times \cdots \times \mathbb{Z}/d_n\mathbb{Z}$.

Preuve — L'application qui à la classe modulo pq associe le couple des classes modulo p et q est un morphisme injectif car p et q sont premiers entre eux. Puisque les deux groupes ont même cardinalité, c'est un isomorphisme. Le cas général se déduit trivialement par induction. ∎

1.44 Lemme Les nombres $1 + (1+i) \cdot n!$, $i = 0, \ldots, n-1$, sont deux à deux premiers entre eux.

Preuve — Posons $d_i = 1 + (1+i) \cdot n!$, $i = 0, \ldots, n-1$. Tout nombre premier p qui divise d_i est $> n$: en effet, sinon, p divise $d_i - (1+i) \cdot n! = 1$, absurde. Supposons qu'il existe un premier p qui divise d_i et d_j, $i < j$: alors p divise $d_j - d_i = (j-i) \cdot n!$, et comme $p > n$, p divise $j - i$, donc $p \leqslant j - i < n$, contradiction. ∎

1.45 Définition (Codage de Gödel) Les applications $\tilde{\beta} : \mathbb{N}^3 \to \mathbb{N}$, $\beta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ et $\beta^* : \mathbb{N} \to \mathrm{Seq}(\mathbb{N})$ (appelée le *codage de Gödel*) sont définies par :

$$\begin{aligned} \tilde{\beta}(x, y, i) &= x \bmod \big(1 + (1+i)y\big) \\ (n)_i = \beta(n, i) &= G(n) \bmod \big(1 + (1+i)D(n)\big) \\ \beta^*(u) &= \langle \beta(D(u), 0), \ldots, \beta(D(u), G(u) - 1) \rangle \end{aligned}$$

où $G = \pi_1^2 \circ J^{-1}$ et $D = \pi_2^2 \circ J^{-1}$ sont les inverses d'un codage p.r. des couples $J : \mathbb{N}^2 \to \mathbb{N}$.

1.46 Lemme $\tilde{\beta}$, β et β^* sont p.r., et pour toute suite $\langle x_0, \ldots, x_{k-1} \rangle$ il existe un entier n tel que $(n)_i = x_i$, $i = 0, \ldots, k-1$.

Preuve — Il est clair que $\tilde{\beta}$, β et β^* sont p.r. Soit $\langle x_0, \ldots, x_{k-1} \rangle$ une suite. On choisit $p \geqslant k$ tel que $1 + (1+i) \cdot p! > x_i$, $i = 0, \ldots, k-1$. Posons $d_i = 1 + (1+i) \cdot p!$. D'après le lemme chinois 1.43, il existe un entier a tel que $a \bmod d_i = x_i$, $i = 0, \ldots, k-1$. On a $\tilde{\beta}(a, p!, i) = x_i$ car $x_i < d_i$, donc on prend $n = J(a, p!)$. ∎

1.47 Proposition La classe des fonctions récursives partielles est la plus petite classe X de fonctions partielles de toutes arités close par (R2), (R4) et :

(R'1) X contient les projections, les polynomes à coefficients entiers et la comparaison $\chi_< : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ ($\chi_<(x, y) = 1$ si $x < y$, 0 sinon).

Preuve — La classe des fonctions récursives est évidemment close par les schémas (R'1), (R2), (R4). Réciproquement, on exprime le schéma (R3) au moyen de (R'1), (R2), (R4). Soit $f = \mathrm{Rec}(g,h) : \mathbb{N}^{n+1} \to \mathbb{N}$, où $g : \mathbb{N}^n \to \mathbb{N}$ et $h : \mathbb{N}^{n+2} \to \mathbb{N}$ sont construites avec les schémas (R'1), (R2), (R4). Soit la fonction

$$F(\vec{x},y) = (\mu s)\Big((s)_0 = g(\vec{x}) \text{ et } (\forall z < y)\big((s)_{z+1} = h(\vec{x},z,(s)_z)\big)\Big).$$

On vérifie que $F(\vec{x},y)$ est défini si, et seulement si, $f(\vec{x},0),\dots,f(\vec{x},y)$ le sont, et qu'alors $(F(\vec{x},y))_z = f(\vec{x},z)$ pour tout $z = 0,\dots,y$.

- Si $F(\vec{x},y)$ est défini, il existe un entier s tel que $(F(\vec{x},y))_0 = g(\vec{x}) = f(\vec{x},0)$ et pour tout $z \leqslant y$, $(F(\vec{x},y))_{z+1} = h(\vec{x},z+1,(F(\vec{x},y))_z)$, donc $(F(\vec{x},y))_z = f(\vec{x},z)$ est défini pour tout z.
- Si $f(\vec{x},0),\dots,f(\vec{x},y)$ sont définis, soit σ la suite $\langle f(\vec{x},0),\dots,f(\vec{x},y)\rangle$: d'après le lemme 1.46, il existe un entier s tel que $(s)_z = f(\vec{x},z)$, $z = 0,\dots,y$, donc $F(\vec{x},y)$ est défini. De plus $(F(\vec{x},y))_0 = g(\vec{x}) = f(\vec{x},0)$, et de $(F(\vec{x},y))_z = f(\vec{x},z)$ on déduit

$$(F(\vec{x},y))_{z+1} = h(\vec{x},z,(F(\vec{x},y))_z) = h(\vec{x},z,f(\vec{x},z)) = f(\vec{x},z+1).$$

Il reste à montrer que F est construite avec les schémas (R'1), (R2), (R4).

Disons qu'une relation est constructible au moyen de (R'1), (R2), (R4) lorsque sa fonction caractéristique l'est. On vérifie aisément que la classe des relations constructibles au moyen de (R'1), (R2), (R4) est close par opérations booléennes et quantification bornée : pour la quantification existentielle bornée par exemple, on a $(\exists x < y)P(x,\vec{a})$ si, et seulement si, $(\mu x)P(x,\vec{a}) < y$. Si le codage des couples sous-jacent est le polynome J_3 de l'exemple 1.20, G et D sont donc constructibles au moyen de (R'1), (R2), (R4). De plus la soustraction s'écrit $x \dot{-} y = \mu z(x < y + z + 1)$ et :

$$\begin{aligned} x \bmod y &= \mu z\big(z < y \text{ et } y \text{ divise } x \dot{-} z\big) \\ &= \mu z\big(z < y \text{ et } (\exists t < x \dot{-} z + 1)(ty = x \dot{-} z)\big). \end{aligned}$$

On en conclut que $(n)_i$, composée de polynomes à coefficients entiers, de G et D et de la fonction mod, et par conséquent F sont constructibles avec les schémas (R'1), (R2), (R4). ∎

2 Machines de Turing

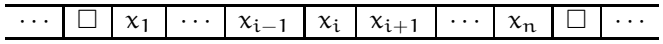
2.1 Machines

On fixe un ensemble à quatre éléments $\{\square,\vdash,\top,\bot\}$. $\square$ s'appelle le *caractère blanc*, $\vdash$ l'*état initial*, $\top$ l'*état final acceptant* et $\bot$ l'*état final rejetant*.

2.1 Définition (Machine de Turing, 1936) Une *machine de Turing (déterministe à un ruban)* est un triplet $\mathcal{M} = (\Sigma,S,T)$ où Σ et S sont des ensembles finis disjoints et disjoints de $\{\square,\vdash,\top,\bot\}$, et T est une application de $(S\cup\{\vdash\})\times(\Sigma\cup\{\square\})$ dans $(S\cup\{\top,\bot\})\times(\Sigma\cup\{\square\})\times\{-1,0,+1\}$. On appelle Σ l'*alphabet* de $\mathcal{M}$, on note $\Sigma^\square = \Sigma\cup\{\square\}$, on appelle S l'*ensemble des états* de $\mathcal{M}$ et T sa *fonction de transition*.

2.2 Définition (Configuration) Soit $\mathcal{M} = (\Sigma,S,T)$ une machine de Turing. Une *configuration* est un triplet (s,f,n) où $s \in S\cup\{\vdash,\top,\bot\}$ est un état, $n \in \mathbb{Z}$ (appelé la *position du curseur*) et $f : \mathbb{Z} \to \Sigma^\square$ est une fonction (appelée le *contenu du ruban*) qui vaut $\square$ sauf pour un nombre fini d'entiers. Si la configuration $\kappa = (s,f,n)$ est telle que $f(n+p+i) = f(n-q-i) = \square$ pour tout $i > 0$, on pourra utiliser pour κ la notation (s,v,w), où v est le mot $f(n-1) * f(n-2) * \cdots * f(n-q) \in \Sigma^{\square *}$ et w est le mot $f(n) * f(n+1) * \cdots * f(n+p) \in \Sigma^{\square *}$.

On représente souvent une configuration $(s, x_{i-1} \cdots x_1, x_i x_{i+1} \cdots x_n)$ par le diagramme suivant :



où les caractères sont écrits sur un ruban infini presque partout rempli de □. On peut indiquer la case numérotée 0 ou bien le numéro n de la case courante (désignée par l'état courant s). Observons que la deuxième notation pour une configuration, (s, v, w), n'est pas une représentation univoque : on peut ajouter des □, par exemple (s, v, w) et $(s, v\square\square, w\square)$ représentent la même configuration. De plus, cette deuxième notation oublie la position de la case numérotée 0. Cet oubli est justifié par le lemme 2.5 qui exprime une invariance par translation de la relation d'exécution, et par le fait que les propriétés observées ne dépendent pas de la position du curseur (définition 2.15).

2.3 Définition (Calcul ou exécution) On dit qu'une machine de Turing $\mathcal{M} = (\Sigma, S, T)$ *fait passer de* la configuration $\kappa = (s, f, n)$ à la configuration $\kappa' = (s', f', n')$ lorsque :
- $T(s, x) = (s', x', d)$,
- $f(n) = x$ et $f'(n) = x'$,
- $f'(i) = f(i)$ pour tout $i \neq n$,
- $n' = n + d$.

On dit aussi que $\mathcal{M}$ *mène de* κ à κ' ou que κ' *découle* de κ, et on le note

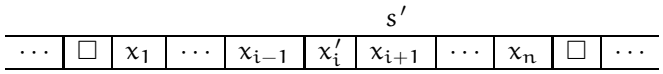
$$\kappa \xrightarrow{\mathcal{M}} \kappa'.$$

On appelle cette relation entre configurations la relation d'*exécution*. Une suite de configurations $\kappa_0, \kappa_1, \ldots$, finie ou non, telle que $\mathcal{M}$ mène de κ_i à κ_{i+1} pour tout $i \geqslant 0$, est appelée un *calcul* ou une *séquence d'exécution*. On écrit

$$\kappa \xrightarrow{\mathcal{M}^k} \kappa'$$

et on dit que κ' découle de κ en k étapes quand il existe une séquence d'exécution finie $\kappa_0, \kappa_1, \ldots, \kappa_k$ telle que $\kappa = \kappa_0$ et $\kappa_k = \kappa'$.

2.4 Exemple Si $\mathcal{M} = (\Sigma, S, T)$ est une machine de Turing, κ est la configuration représentée par le diagramme précédent, soit $(s, x_{i-1} \cdots x_1, x_i x_{i+1} \cdots x_n)$, et $T(s, x_i) = (s', x_i', +1)$, alors $\mathcal{M}$ fait passer de κ à la configuration suivante :



2.5 Lemme (Invariance par translation) Soient $\mathcal{M} = (\Sigma, S, T)$ une machine de Turing et $p \in \mathbb{Z}$. Si $\kappa = (s, f, n)$ est une configuration quelconque, soit κ' la configuration $(s, f', n+p)$ où $f'(i+p) = f(i)$. Alors :

$$\kappa_0 \xrightarrow{\mathcal{M}} \kappa_1 \quad \text{si, et seulement si,} \quad \kappa_0' \xrightarrow{\mathcal{M}} \kappa_1'.$$

La preuve est triviale.

2.2 Algorithmes

Une machine de Turing sert à décider des langages et calculer des fonctions. On peut d'ailleurs noter que la décision d'un langage $L \subseteq \Sigma^*$ d'alphabet Σ revient au calcul de sa fonction caractéristique $\chi_L : \Sigma^* \to \{0, 1\}$.

2.6 Définition (Machines et mots) Soit $\mathcal{M}$ une machine de Turing d'alphabet Σ. La *configuration initiale* associée à $v \in \Sigma^*$ est $(\vdash, \varepsilon, v)$. Une configuration dont l'état est un des deux états finaux est appelée une *configuration finale*, *acceptante* ou *rejetante* selon le cas. On dit que $\mathcal{M}(v)$ *converge sur l'entrée* v ou simplement que $\mathcal{M}(v)$ *converge*, et on le note

$$\mathcal{M}(v) \downarrow$$

s'il existe un entier k tel que $\mathcal{M}$ fait passer en k étapes de la configuration initiale associée à v à une configuration finale : on peut préciser alors que $\mathcal{M}(v)$ *converge en* k *étapes*. Dans le cas contraire, on dit que $\mathcal{M}(v)$ *diverge* et on le note

$$\mathcal{M}(v) \uparrow .$$

En cas de convergence, si la configuration finale est acceptante (resp. rejetante), on dit plus précisément que $\mathcal{M}$ *accepte* (resp. *rejette*) v et on note

$$\mathcal{M}(v) \downarrow_o \text{ (resp. } \mathcal{M}(v) \downarrow_n).$$

En cas d'acceptation, si la configuration finale est $(\top, \varepsilon, w)$ avec $w \in \Sigma^*$, on dit que $\mathcal{M}(v) = w$.

L'utilisation de la notation $\mathcal{M}(v)$ dans tous les cas de figure (divergence, convergence avec acceptation, rejet ou résultat) signifie qu'on interprète en l'occurrence une machine comme une fonction de Σ^* dans l'ensemble $\{\uparrow, \downarrow_n\} \cup (\{\downarrow_o\} \times \Sigma^*)$. Il est intéressant de noter qu'on donne comme entrée à une machine $\mathcal{M}$ d'alphabet Σ un mot sur Σ (et non sur $\Sigma^\square$) : l'important est que $\mathcal{M}$ sache où commence et s'arrête son entrée, c'est pourquoi on ne part pas d'une configuration trop quelconque $(\vdash, v, w)$ avec $v, w \in \Sigma^{\square *}$.

2.7 Définition (Temps de calcul) Soient une machine de Turing $\mathcal{M}$ d'alphabet Σ et une fonction $f : \mathbb{N} \to \mathbb{R}^+$. On dit que $\mathcal{M}$ *opère en temps* f (ou $f(n)$) si pour tout $v \in \Sigma^*$, $\mathcal{M}(v)$ converge en au plus $f(|v|)$ étapes.

L'utilisation de fonctions à valeurs réelles pour mesurer le temps n'est qu'une commodité qui dispense d'écrire d'inutiles parties entières.

2.8 Définition (Machines, langages et relations) Soient $L \subseteq \Sigma^*$ un langage et $\mathcal{M}$ une machine de Turing d'alphabet contenant Σ. On dit que $\mathcal{M}$ *accepte* L lorsque pour tout $v \in \Sigma^*$, $v \in L$ si, et seulement si, $\mathcal{M}$ accepte v. On dit que $\mathcal{M}$ *décide le langage* L lorsque pour tout $v \in L$, $\mathcal{M}$ accepte v et pour tout $v \in \Sigma^* \setminus L$, $\mathcal{M}$ rejette v. Plus généralement, soient $R \subseteq (\Sigma^*)^p$ une relation et $\mathcal{M}$ une machine de Turing d'alphabet contenant $\Sigma \cup \{\#\}$, avec $\# \notin \Sigma$. On dit que $\mathcal{M}$ *accepte la relation* R lorsque pour tout $(v_1, \ldots, v_p) \in (\Sigma^*)^p$, $(v_1, \ldots, v_p) \in R$ si, et seulement si, $\mathcal{M}$ accepte $v_1 \# \cdots \# v_p$, et on dit que $\mathcal{M}$ *décide la relation* R lorsque pour tout $(v_1, \ldots, v_p) \in R$, $\mathcal{M}$ accepte $v_1 \# \cdots \# v_p$ et pour tout $(v_1, \ldots, v_p) \in (\Sigma^*)^p \setminus R$, $\mathcal{M}$ rejette $v_1 \# \cdots \# v_p$.

Un langage est *semi-décidable* (ou *récursivement énumérable*) s'il existe une machine de Turing qui l'accepte. Un langage est *décidable* (ou *récursif*) s'il existe une machine de Turing qui le décide. Un langage est *co-récursivement énumérable* si son complémentaire est récursivement énumérable. Un langage qui n'est pas décidable est dit *indécidable*. On note RE (resp. coRE) l'ensemble des langages récursivement énumérables (resp. co-récursivement énumérables) et R celui des langages récursifs.

Un langage L dont on souhaite déterminer s'il est semi-décidable, décidable, etc, on l'appelle souvent un *problème*.

On peut observer que pour décider un langage $L \subseteq \Sigma^*$, on s'autorise à prendre une machine d'alphabet plus grand. En fait, ce n'est pas très important, et cette convention est surtout destinée à uniformiser la définition dans le cas de relations R d'arité quelconque : là, on a besoin d'un séparateur # qui ne soit pas dans l'alphabet Σ de R. En revanche, si l'arité est constante et connue, disons p, on peut remplacer les séparateurs # par des blancs $\square$.

2.9 Exemple Le langage $L = \{a^p b^p, p \in \mathbb{N}\}$ sur l'alphabet $\Sigma = \{a, b\}$ est décidable. Pour montrer cela, on définit une machine de Turing $\mathcal{M}$ qui, étant donné un mot $w \in \{a, b\}^*$, efface si possible la lettre a le plus à gauche (et rejette sinon), se déplace le plus possible à droite du mot, efface si possible la lettre b le plus à droite, retourne à gauche, et recommence si le mot n'est pas vide. Ce qu'on formalise de la manière suivante : $\mathcal{M} = (\Sigma, S, T)$, où $S = \{0, 1, 2, 3\}$ et T est définie par la table

	a	b	$\square$
$\vdash$	$0, a, 0$	$0, b, 0$	$0, \square, 0$
0	$1, \square, +1$	$\bot$	$\top$
1	$1, a, +1$	$1, b, +1$	$2, \square, -1$
2	$\bot$	$3, \square, -1$	$\bot$
3	$3, a, -1$	$3, b, -1$	$0, \square, +1$

où on n'a indiqué que l'état lorsque c'est un état final ($\top$ ou $\bot$), sans se préoccuper du caractère écrit et du déplacement qui sont alors sans intérêt.

2.10 Exercice Écrire les deux suites de configurations de la machine $\mathcal{M}$ définie dans l'exemple 2.9, en partant de la configuration initiale associée au mot $aabb$ (accepté), puis au mot aab (refusé). Montrer, par récurrence sur la longueur du mot $w \in \{a, b\}^*$, que w est accepté s'il est de la forme $a^p b^p$, refusé sinon. En déduire que $\mathcal{M}$ décide L.

2.11 Exercice Montrer que tout langage automatique (ou, ce qui revient au même, régulier) est récursif.

L'exemple 2.9 montre donc que l'ensemble des langages réguliers sur Σ est strictement inclus dans l'ensemble des langages récursifs sur Σ.

2.12 Proposition R est une algèbre de Boole et $\mathsf{R} = \mathsf{RE} \cap \mathsf{coRE}$.

Preuve — Le langage vide et le langage Σ^* sont trivialement récursifs. Si L est récursif, son complémentaire l'est aussi : on échange acceptation et rejet dans une machine de Turing qui décide L. Enfin, R est close par intersection et union, ce qu'on montre facilement en combinant les réponses des deux machines.

Il est clair que $\mathsf{RE} \supseteq \mathsf{R}$, et donc que $\mathsf{coRE} \supseteq \mathsf{R}$ d'après ce qui précède. Réciproquement, si L et $\Sigma^* \setminus L$ sont acceptés par des machines $\mathcal{M}$ et $\mathcal{M}'$, on peut construire une machine $\mathcal{N}$ qui entrelace les calculs de $\mathcal{M}$ et de $\mathcal{M}'$, c'est-à-dire alterne une étape de $\mathcal{M}$ et une étape de $\mathcal{M}'$; $\mathcal{N}$ accepte si une étape de $\mathcal{M}$ aboutit à un état acceptant, et rejette si une étape de $\mathcal{M}'$ aboutit à un état acceptant. ∎

2.13 Exercice Définir explicitement la machine invoquée dans la preuve de la proposition 2.12.

2.14 Définition (Machines et fonctions) Soient $f : (\Sigma^*)^p \to (\Sigma^*)^q$ une fonction et $\mathcal{M}$ une machine de Turing d'alphabet contenant $\Sigma \cup \{\#\}$, avec $\# \notin \Sigma$. Pour tout $(v_1, \ldots, v_p) \in (\Sigma^*)^p$, on pose $\mathcal{M}(v_1, \ldots, v_p) = \mathcal{M}(v_1 \# v_2 \# \cdots \# v_p)$. On dit que $\mathcal{M}$ *calcule* f lorsque pour tout $(v_1, \ldots, v_p) \in (\Sigma^*)^p$, $\mathcal{M}(v_1, \ldots, v_p) = w_1 \# \cdots \# w_q$ si $f(v_1, \ldots, v_p) = (w_1, \ldots, w_q)$. Une fonction est dite *Turing-calculable* s'il existe une machine de Turing qui la calcule.

2.15 Définition (Équivalence dénotationnelle) Deux machines de Turing $\mathcal{M}$ et $\mathcal{M}'$ de même alphabet Σ sont *dénotationnellement équivalentes*, ce qu'on note $\mathcal{M} \sim \mathcal{M}'$, si pour tout $v \in \Sigma^*$:

- $\mathcal{M}(v) \uparrow$ si, et seulement si, $\mathcal{M}'(v) \uparrow$,
- $\mathcal{M}(v) \downarrow_n$ si, et seulement si, $\mathcal{M}'(v) \downarrow_n$,
- $\mathcal{M}(v) \downarrow_o$ si, et seulement si, $\mathcal{M}'(v) \downarrow_o$,
- $\mathcal{M}(v) = w$ si, et seulement si, $\mathcal{M}'(v) = w$.

On a évidemment :

2.16 Lemme Si deux machines de Turing d'alphabet Σ sont dénotationnellement équivalentes, elles décident le même langage sur Σ. Si deux machines de Turing d'alphabet $\Sigma + \{\#\}$ sont dénotationnellement équivalentes et $p, q \in \mathbb{N}$, elles calculent la même fonction de $(\Sigma^*)^p$ dans $(\Sigma^*)^q$.

2.17 Exercice Définir une machine de Turing déterministe à un ruban qui attend en entrée un mot $w \in \Sigma^*$ et le duplique, c'est-à-dire s'arrête dans la configuration $(\top, \varepsilon, w\#w)$.

2.18 Exercice
1. Montrer que toute machine de Turing est dénotationnellement équivalente à une machine de Turing dont la fonction de transition n'utilise que les déplacements $+1$ et -1 (jamais de sur-place).
2. Montrer que toute machine de Turing est dénotationnellement équivalente à une machine de Turing où la contrainte ci-dessus sur les déplacements est remplacée par la contrainte suivante sur l'écriture : on suppose que le langage est $\{0,1\}$ et on demande, étant donnée une machine machine $\mathcal{M}$, que la nouvelle machine $\mathcal{M}'$ change systématiquement la lettre lue, autrement dit $T(s,0) = (s',1,d)$ et $T(s,1) = (s',0,d)$.
3. Les deux contraintes ci-dessus sont-elles compatibles (peut-on construire $\mathcal{M}'$ satisfiant les deux) ?

2.3 Représentation des données

On s'intéresse particulièrement aux fonctions et problèmes sur deux types de données : les entiers naturels et les graphes finis. Sauf mention contraire, les entiers naturels sont représentés en binaire, et les graphes sont représentés de l'une des deux manières suivantes :

- soit par le couple (V, E) où V est la liste des sommets (par exemple une liste d'entiers $1, \dots, n$ eux-mêmes représentés en binaire) et E est l'ensemble des arcs représentés comme des couples de sommets ;
- soit par leur matrice d'adjacence.

Ces deux représentations sont *polynomialement équivalentes* dans la mesure où la taille d'une représentation est majorée par un polynôme en la taille de l'autre, et vice-versa. De même, pour tout $b \geqslant 2$, la représentation des entiers en base b est polynomialement équivalente à la représentation binaire. Ce n'est évidemment pas le cas de la représentation unaire, qui est exponentiellement plus coûteuse.

La question du coût d'une représentation ne nous intéresse toutefois qu'en vue de la section 4 : tant qu'on ne s'occupe pas de complexité, on peut à la limite se contenter de remarquer que les fonctions de passage d'une représentation à l'autre sont calculables. Ainsi, les fonctions de passage entre représentations unaire et binaire des entiers sont évidemment calculables.

2.19 Exercice Définir une machine de Turing calculant la fonction, de $\{1\}^*$ dans $\{0,1\}^*$, qui associe au mot $1^n = 1 * \cdots * 1$ constitué de n lettres 1 et représentant l'entier n en unaire, le mot sur $\{0,1\}$ qui représente n en binaire.

2.20 Exercice Définir une machine de Turing qui calcule le prédécesseur d'un entier représenté en unaire. Même question en représentation binaire.

2.21 Exemple (Accessibilité) On se donne un graphe (dirigé, fini) $G = (V, E)$ et deux sommets $v_1, v_2 \in V$. Le problème dit d'accessibilité, noté Acc, est de déterminer s'il existe un chemin (orienté) de v_1 vers v_2. Notons que Acc est un langage, par exemple sur l'alphabet $\{0,1\}$ si l'on convient que les sommets sont les entiers $1, \dots, n$ représentés en binaire, et il s'agit donc de définir un algorithme qui décide le langage Acc. Une méthode est de construire deux suites d'ensembles de sommets $A_p, B_p, p \geqslant 0$: on part de $A_0 = B_0 = \{v_1\}$; à l'étape p, on choisit un sommet $i_p \in A_p$ et on pose

$$A_{p+1} = A_p \setminus \{i_p\} \cup C_p \quad \text{et} \quad B_{p+1} = B_p \cup C_p,$$

où C_p est l'ensemble des sommets $j \notin B_p$ tels que G ait une arète de i_p vers j. On s'arrête quand A_p est vide. Alors, on accepte si $v_2 \in B_p$ et on rejette dans le cas contraire.

2.22 Exercice Définir une machine de Turing qui formalise l'algorithme de l'exemple 2.21. Montrer que cette machine décide Acc.

2.23 Exemple (Fonction d'Ackermann) La fonction d'Ackermann $A : \mathbb{N}^2 \to \mathbb{N}$, définie dans le problème 1.34, est calculée par l'algorithme suivant. On utilise la représentation unaire des entiers. L'alphabet est donc $\{1\}$. Étant donnés des entiers m et n, on part de la configuration initiale $(\vdash, \varepsilon, 1^m\#1^n)$. Une configuration obtenue à partir d'une telle configuration initiale sera de la forme

$$(s, w_1\# \cdots \#w_r\#, u\#v),$$

où s est un état et $u, v, w_1, \ldots, w_r \in \{1\}^*$. Une étape de calcul consiste, si disons $u = 1^p$ et $v = 1^q$, à calculer $A(p, q)$ pour se retrouver dans la configuration

$$(g, w_1\# \cdots \#w_r\#, 1^{A(p,q)}).$$

Ici, g est un état particulier dans lequel la machine teste s'il y a quelque chose d'écrit à gauche du curseur, c'est-à-dire si $r > 0$. Si oui, on décale le premier mot à gauche du curseur vers la droite pour se retrouver dans la configuration

$$(\vdash, w_1\# \cdots \#w_{r-1}\#, w_r\#1^{A(p,q)})$$

et recommencer. Sinon, on a terminé l'algorithme.

Décrivons maintenant une étape de calcul à partir de $(s, w_1\# \cdots \#w_r\#, u\#v)$. Si $u = \varepsilon$, la machine recopie v au début du ruban, ajoute un 1 et l'étape est finie (on a bien calculé $A(0, q) = q+1$, si disons $v = 1^q$). Si $u \neq \varepsilon$ et $v = \varepsilon$, la machine échange le dernier 1 de u avec le # qui suit (c'est-à-dire remplace $1^{p+1}\#$ par $1^p\#1$, si $u = 1^{p+1}$) et l'étape est finie (l'étape suivante va alors calculer $A(p, 1) = A(p+1, 0)$). Si enfin $u \neq \varepsilon$ et $v \neq \varepsilon$, disons $u = 1^{p+1}$ et $v = 1^{q+1}$, la machine recopie 1^p à gauche du curseur et $1^{p+1}\#1^q$ à droite pour se retrouver dans la configuration

$$(\vdash, w_1\# \cdots \#w_r\#1^p\#, 1^{p+1}\#1^q),$$

et recommencer, afin de calculer $A(p+1, q+1) = A(p, A(p+1, q))$; cette étape termine car la taille de la partie du ruban à droite du curseur décroit strictement (de $p+1+q+1$ à $p+1+q$).

2.4 Variantes

2.24 Définition (Machine à plusieurs rubans) Une *machine de Turing déterministe à* k *rubans* est un triplet $\mathcal{M} = (\Sigma, S, T)$, où Σ et S sont comme dans la définition 2.1 et T est une application

$$(S \cup \{\vdash\}) \times (\Sigma \cup \{\square\})^k \to (S \cup \{\top, \bot\}) \times \big((\Sigma \cup \{\square\}) \times \{-1, 0, +1\}\big)^k.$$

Une *configuration à* k *rubans* est un $k+2$-uplet $(s, f_1, \ldots, f_k, n)$ où $s \in S \cup \{\vdash, \top, \bot\}$ est un état, $n \in \mathbb{Z}^k$ (la *position des curseurs*) et $f_j : \mathbb{Z} \to \Sigma^{\square}$, $j = 1, \ldots, k$, est une fonction qui vaut $\square$ sauf pour un nombre fini d'entiers. On utilise pour une configuration la notation $(s, v_1, w_1, \ldots, v_k, w_k)$ où les $v_j, w_j \in \Sigma^{\square}$, $j = 1, \ldots, k$, sont comme dans la définition 2.2 ; on appelle f_j ou (v_j, w_j) le *contenu du* j*ème ruban*.

On dit que $\mathcal{M}$ *fait passer de* la configuration $(s, f_1, \ldots, f_k, n)$ à la configuration $(s', f'_1, \ldots, f'_k, n')$ lorsque :

$$T(s, x_1, \ldots, x_k) = (s', x'_1, d_1, \ldots, x'_k, d_k)$$

et pour tout $j = 1, \ldots, k$:

– $f_j(n_j) = x_j$ et $f'(n_j) = x'_j$,
– $f'_j(i) = f_j(i)$ pour tout $i \neq n_j$,
– $n'_j = n_j + d_j$.

La notion de séquence d'exécution est calquée sur 2.3. La *configuration initiale* (resp. *finale*) associée à $v \in \Sigma^*$ a tous ses rubans vides sauf le premier qui contient (ε, v). Les notions de configuration acceptante, rejetante sont identiques, ainsi que celle de temps de calcul, et on note $\mathcal{M}(v) = w$ si la configuration finale est acceptante et le contenu du premier ruban soit (ε, w). Les définitions de langage accepté, refusé, de fonction calculée et d'équivalence dénotationnelle sont alors identiques.

Comme on le verra, la possibilité de calculer sur plusieurs rubans est souvent bien commode. Mais comme l'affirme la proposition suivante, cette libéralisation du modèle de calcul ne change rien d'essentiel au pouvoir calculatoire (facteur quadratique).

Rappelons les notations $\mathcal{O}$, o et $\sim$ d'analyse. Soient f et g deux applications de $\mathbb{N}$ ou $\mathbb{R}$ dans $\mathbb{R}$. Si $f \geqslant 0$, on dit que $f = \mathcal{O}(g)$ (f est un *grand "o"* de g) s'il existe une constante $c \geqslant 0$ telle que pour tout $x \in X$, $|f(x)| \leqslant c \cdot g(x)$. On dit que $f = o(g)$ (f est un *petit "o"* de g) si $f(x)/g(x) \to_{x \to +\infty} 0$. On dit que $f \sim g$ (f est *équivalente* à g) si $f = \mathcal{O}(g)$ et $g = \mathcal{O}(f)$.

2.25 Proposition Toute machine de Turing déterministe $\mathcal{M}$ à k rubans est dénotationnellement équivalente à une machine de Turing déterministe $\mathcal{M}'$ à 1 ruban. Si $\mathcal{M}$ opère en temps $f(n)$, $\mathcal{M}'$ opère en temps $\mathcal{O}(f(n)^2)$.

Preuve — Soit $\mathcal{M} = (\Sigma, S, T)$. Soit $\dot{\Sigma} + \{\dot{\square}\}$ une copie de $\Sigma + \{\square\}$ constituée des nouvelles lettres "pointées" $\dot{x}$ pour $x \in \Sigma + \{\square\}$. Le langage Σ' de $\mathcal{M}'$ est l'ensemble $\left(\Sigma + \dot{\Sigma} + \{\square, \dot{\square}\}\right)^k$, et on identifie le k-uplet $(\square, \ldots, \square)$ avec le caractre vide $\square$. Une configuration $(s, f_1, \ldots, f_k, n)$ de $\mathcal{M}$ est simulée par $(s, g, \min_{i=1}^k n_i)$, où $g(p) = (g_1(p), \ldots, g_k(p)) \in \Sigma'$ est défini comme suit pour tout $p \in \mathbb{Z}$:

$$g_i(p) = \begin{cases} x & \text{si } p \neq n_i \text{ et } x = f_i(p), \\ \dot{x} & \text{si } p = n_i \text{ et } x = f_i(p). \end{cases}$$

Pour simuler une étape de $\mathcal{M}$, $\mathcal{M}'$

1. lit le ruban de gauche à droite en enregistrant successivement dans l'état les k lettres pointées $\dot{x}_1, \ldots, \dot{x}_k$,
2. calcule $T(s, x_1, \ldots, x_k) = (s', x'_1, d_1, \ldots, x'_k, d_k)$,
3. revient de droite à gauche en remplaçant chaque $\dot{x}_i$ par $\dot{x}'_i$ et en effectuant les déplacements,
4. effectue éventuellement un déplacement à gauche pour placer le curseur sur la plus petite case p contenant une lettre pointée,
5. enfin passe dans l'état s'.

Si $\mathcal{M}$ opère en temps $f(n)$, puisqu'alors les longueurs des rubans en jeu sont majorées par $f(n)$ (on ne peut pas écrire plus de t lettres dans une durée t), la taille d'une configuration de $\mathcal{M}'$ est majorée par $1 + f(n)$. En comptant l'aller-retour plus les changements de lettres pointées, une étape de $\mathcal{M}$ est donc simulée en au plus $2 + 2k(f(n) + 1) + 2k$ étapes. Soit en tout un temps de calcul de $\mathcal{O}(f(n)^2)$. ∎

2.26 Définition (Machine non-déterministe) Une *machine de Turing non-déterministe à k rubans* est un triplet $\mathcal{M} = (\Sigma, S, T)$, où Σ et S sont comme dans la définition 2.1 et

$$T \subseteq (S \cup \{\vdash\}) \times (\Sigma \cup \{\square\})^k \times (S \cup \{\top, \bot\}) \times \left((\Sigma \cup \{\square\}) \times \{-1, 0, +1\}\right)^k$$

est une relation. Les définitions relatives aux *configurations* sont comme en 2.24. En considérant T comme une fonction de $(S \cup \{\vdash\}) \times (\Sigma \cup \{\square\})^k$ dans $\mathrm{P}\big((S \cup \{\top, \bot\}) \times ((\Sigma \cup \{\square\}) \times \{-1, 0, +1\})^k\big)$, on a une notion de *séquence d'exécution* non-déterministe calquée sur 2.24. Une séquence d'exécution est *convergente* (resp. *acceptante*, *rejetante*) si elle est finie et ne peut être continuée (resp. se termine

par une configuration acceptante, rejetante), *divergente* si elle est infinie. On dit que $\mathcal{M}$ *accepte* v s'il existe une séquence d'exécution acceptante partant de la configuration initiale associée à v. On dit que $\mathcal{M}$ *rejette* v dans tous les autres cas. Les notions de langage *accepté, refusé* s'en déduisent.

On dit que $\mathcal{M}$ *calcule* $f : (\Sigma^*)^p \to \Sigma^*$ lorsque pour tout $(v_1, \ldots, v_p) \in (\Sigma^*)^p$, toute séquence d'exécution partant de la configuration initiale associée à $(v_1, \ldots, v_p)$

- ou bien diverge,
- ou bien termine sur la configuration finale associée à $f(v_1, \ldots, v_p)$.

Étant donnée une fonction $f : \mathbb{N} \to \mathbb{R}^+$, $\mathcal{M}$ *opère en temps* $f(n)$ si pour tout $v \in \Sigma^*$, toute séquence d'exécution de $\mathcal{M}$ à partir de v converge en au plus $f(|v|)$ étapes.

Le point important dans la définition ci-dessus est l'asymétrie entre acceptation d'une entrée (au moins une des séquences d'exécution partant de cette entrée est acceptante) et rejet (aucune séquence d'exécution n'est acceptante, c'est-à-dire toute séquence d'exécution est rejetante ou divergente).

En tant que modèle de calcul, les machines de Turing non-déterministes ne sont pas très réalistes, mais elles donnent lieu à des classes de complexité très importantes, et on verra que dans certains cas, notamment la complexité en temps polynomial, on peut remplacer cette notion par des vérifieurs (définition 4.12), qui correspondent mieux à l'intuition. La proposition suivante affirme que les machines non-déterministes n'apportent rien du point de vue de la calculabilité, mais leur simulation par des machines déterministes implique une croissance a priori exponentielle du temps de calcul.

2.27 Proposition Toute machine de Turing non-déterministe $\mathcal{M}$ à 1 ruban est dénotationnellement équivalente à une machine de Turing déterministe $\mathcal{M}'$ à 3 rubans. Si $\mathcal{M}$ opère en temps $f(n)$, $\mathcal{M}'$ opère en temps $\mathcal{O}(c^{f(n)})$ pour une certaine constante $c > 1$.

Preuve — On se contente de l'idée, qui est très simple. Les séquences d'exécution de $\mathcal{M} = (\Sigma, S, T)$ à partir de l'entrée v forment un arbre planaire A. En considérant T comme une fonction non-deterministe et donc $T(s,x) \in P\big((S \cup \{\top, \bot\}) \times ((\Sigma \cup \{\Box\}) \times \{-1, 0, +1\})^k\big)$ comme un ensemble de transitions possibles, on voit que A est à branchement borné par le maximum d des cardinalités de $T(s,x)$ pour $s \in S$ et $x \in \Sigma^\Box$. $\mathcal{M}'$ va donc essentiellement parcourir l'arbre A, en prenant soin de procéder par profondeur croissante (toutes les branches de longueur 1, puis de longueur 2, etc) afin d'éviter de partir dans une branche infinie alors qu'une autre branche mènerait à un état acceptant. $\mathcal{M}'$ accepte si une branche de $\mathcal{M}$ accepte, refuse sinon. Si $\mathcal{M}$ opère en temps $f(n)$, le temps de calcul de $\mathcal{M}'$ est

$$\mathcal{O}\left(\sum_{1 \leqslant i \leqslant f(n)} d^i \cdot i\right) = \mathcal{O}(c^{f(n)})$$

pour une certaine constante $c > 1$. ∎

2.28 Exercice Préciser la preuve de la proposition 2.27.

2.5 Thèse de Church

2.29 Théorème La classe des fonctions Turing-calculables coïncide avec celle des fonctions récursives.

Le théorème 2.29 accrédite la thèse, dite *thèse de Church*, selon laquelle les fonctions récursives ou de manière équivalente les machines de Turing capturent bien la notion intuitive de "calculabilité". D'autres définitions ont été proposées : fonctions λ-définissables de Church (1941), fonctions calculables par des machines à registre..., qu'on ne verra pas dans ce cours, et donnent lieu la même classe de fonctions calculables.

Preuve — On laisse au lecteur le soin de montrer que toute fonction récursive est Turing-calculable,

par une facile induction sur sa construction au moyen des schémas (R1) à (R4) (définitions 1.2 et 1.35). D'après la proposition 2.25, on pourra pour plus de commodité utiliser plusieurs rubans.

On prouve la réciproque, à savoir que toute fonction Turing-calculable $f : \mathbb{N}^p \to \mathbb{N}$ est récursive. Soit donc $\mathcal{M} = (\Sigma, S, T)$ une machine de Turing (déterministe) à un seul ruban calculant f. On suppose que les entiers sont représentés en unaire, et pour simplifier, on se débarrasse tout de suite du symbole auxiliaire # en supposant qu'un p-upluet d'entiers $(n_1, \dots, n_p)$ est entré sous la forme $1^{n_1}\square\cdots\square 1^{n_p}$ (au lieu de $1^{n_1}\#\cdots\#1^{n_p}$) : on peut toujours le faire, car à p fixé on sait où sont les bornes de l'entrée (les p mots de $\{1\}^*$ séparés par $p-1$ $\square$). L'alphabet est donc $\Sigma = \{1\}$. On suppose aussi que l'ensemble d'états S est $\{1, \dots, e\}$. On code l'état initial $\vdash$ par 0 et les états finaux $\top, \bot$ respectivement par $e+1, e+2$. On code une configuration (s, v, w) de $\mathcal{M}$ ($v, w \in \{1, \square\}^*$) par le quadruplet d'entiers

$$\Psi(s, v, w) = (s, \psi(v), x, \psi(w')),$$

où $w = xw'$ (avec $x = 0$ si $w = \varepsilon$) et $\psi(v)$ est l'entier dont le mot v est la représentation binaire, $\square$ étant remplacé par 0. Par exemple, la configuration

						7				
$\cdots$	$\square$	1	$\square$	1	$\square$	$\square$	1	1	$\square$	$\cdots$

est codée par le quadruplet $(7, 5, 0, 6)$. Soit $g : \mathbb{N}^{p+1} \to \mathbb{N}^4$ la fonction qui à $(\vec{n}, t) = (n_1, \dots, n_p, t)$ associe le code $\Psi(s, v, w)$ de la configuration obtenue en t étapes à partir de la configuration initiale associée à $\vec{n}$. Soient $g_1, g_2, g_3, g_4 : \mathbb{N}^{p+1} \to \mathbb{N}$ les 4 composantes de g.

On montre que g_1, g_2, g_3, g_4 sont p.r. Pour $t = 0$, on a $g_1(\vec{n}, 0) = g_2(\vec{n}, 0) = 0$. De plus, $g_3(\vec{n}, 0) = 1$ si $n_1 \geqslant 1$, 0 sinon. Ensuite, l'entier dont la représentation unaire est 1^n est $2^n - 1$, l'entier dont la représentation unaire est $1^{n_1-1}01^{n_2}$ est $2^{n_1-1} - 1 + 2^{n_1}(2^{n_2} - 1) = 2^{n_1+n_2} - 1 - 2^{n_1-1}$, et un petit calcul montre que

$$g_4(\vec{n}, 0) = 2^{n_1+\cdots+n_p+p-2} - 1 - \sum_{i=1}^{p-1} 2^{n_1+\cdots+n_i+i-1}.$$

Passons à $t > 0$ et notons $T(s, x) = (s', x', d) = (T_1(s, x), T_2(s, x), T_3(s, x))$. On veut calculer $g(\vec{n}, t+1)$, donc on part de $s = g_1(\vec{n}, t)$ et $x = g_3(\vec{n}, t)$. Dans le cas où $d = T_3(s, x) = +1$, on a

$$\begin{aligned}
g_1(\vec{n}, t+1) &= T_1(s, x) \\
g_2(\vec{n}, t+1) &= 2g_2(\vec{n}, t) + T_2(s, x) \\
g_3(\vec{n}, t+1) &= g_4(\vec{n}, t) \bmod 2 \\
g_4(\vec{n}, t+1) &= g_4(\vec{n}, t) \operatorname{div} 2,
\end{aligned}$$

et on a des formules semblables dans les deux autres cas $d = -1$ ou 0. Ce qui montre que g_1, g_2, g_3, g_4 sont obtenues par récurrence croisée sur t (proposition 1.22), et sont donc p.r.

Maintenant, pour reconstruire f à partir de g, introduisons la fonction partielle $h : \mathbb{N}^p \to \mathbb{N}$ qui à $\vec{n}$ associe l'unique entier t, s'il existe, tel que $g(\vec{n}, t)$ est le code d'une configuration acceptante, c'est-à-dire tel que $g_1(\vec{n}, t) = e + 1$ et $g_1(\vec{n}, t') \neq e + 1$ pour tout $t' < t$. Donc h est définie par minimalisation à partir d'une relation p.r., et est récursive. Par conséquent, la fonction de $\mathbb{N}^p$ dans $\mathbb{N}$ qui associe à $\vec{n}$ l'entier $g_3\big(\vec{n}, h(\vec{n})\big) + 2g_4\big(\vec{n}, h(\vec{n})\big)$ est également récursive. Puisque $\mathcal{M}$ calcule f, cet entier vaut $2^{f(\vec{n})} - 1$, et donc

$$f(\vec{n}) = \left[\log_2\left(1 + g_3\big(\vec{n}, h(\vec{n})\big) + 2g_4\big(\vec{n}, h(\vec{n})\big)\right)\right],$$

qui est bien récursive puisque la fonction $x \mapsto [\log_2(x)]$ l'est (exercice 1.18). ∎

2.30 Exercice Donner des formules explicites pour la récurrence croisée dans la preuve du théorème 2.29.

2.31 Proposition Soit $L \subseteq \Sigma^*$ un langage d'alphabet $\Sigma = \{0, \ldots, b-1\}$. En identifiant un mot d'alphabet Σ avec l'entier qu'il représente en base b, les assertions suivantes sont équivalentes :

1. L est récursivement énumérable,
2. L est le domaine d'une fonction récursive,
3. L est vide ou l'image d'une fonction récursive totale.

Preuve — (1⇒2) La fonction partielle qui à $v \in \Sigma^*$ associe 1 si $v \in L$ et est indéfinie sinon, est Turing-calculable partielle de domaine L, donc récursive (partielle) d'après le théorème 2.29.

(2⇒1) Évident.

(1⇒3) Supposons L non vide et soit donc $v \in L$. Soit $\mathcal{M}$ une machine de Turing qui accepte L. On définit une machine de Turing $\mathcal{M}'$ qui, étant donnés deux mots w et n, se comporte comme $\mathcal{M}$ sur l'entrée w tout en décrémentant n, vu comme un entier, et s'arrête quand $n = 0$: la sortie est alors w si le calcul simulé de $\mathcal{M}$ a déjà accepté w, v sinon. $\mathcal{M}'$ converge sur toute entrée et calcule le langage L. Donc, d'après le théorème 2.29, L est bien l'image d'une fonction récursive totale.

(3⇒1) Si L est vide, L est clairement accepté par une machine de Turing qui rejette systématiquement. Sinon, on suppose que $w \in L$ si, et seulement s'il existe v tel que $w = f(v)$, f étant récursive totale, donc Turing-calculable. Soit $\mathcal{M}$ une machine de Turing qui, étant donné un mot w, énumère les entiers en base b, c'est-à-dire les mots de Σ^*, dans l'ordre croissant en partant de 0, calcule leur image par f, et teste si le résultat égale w. Alors $\mathcal{M}$ accepte L. ∎

2.6 Machines universelles

Un caractère essentiel d'une machine de Turing, et de tout modèle de calcul, est que c'est un objet fini, ce qui offre la possibilité de la représenter par un mot. On peut alors définir des machines, dites universelles, qui interprètent une entrée comme le code d'une machine de Turing $\mathcal{M}$ suivi d'une entrée w pour $\mathcal{M}$, et simulent le calcul de $\mathcal{M}$ à partir de w. Dans ce paragraphe on se limite à des machines déterministes à 1 ruban, ce qui est suffisant pour des questions de calculabilité d'après les propositions 2.25 et 2.27, mais le lecteur doit se convaincre qu'on peut adapter la définition aux machines non-déterministes à plusieurs rubans. En notant $\mathcal{M}^\bullet$ le code de la machine de Turing $\mathcal{M}$ et $w^\bullet$ le code de l'entrée w pour $\mathcal{M}$, la propriété que doit satisfaire une machine de Turing universelle $\mathcal{U}$ est :

$$\mathcal{U}(\mathcal{M}^\bullet \# w^\bullet) = \mathcal{M}(w)^\bullet.$$

2.32 Exemple L'alphabet de la machine de Turing universelle $\mathcal{U}$ qu'on définit est constitué des entiers 0, 1, des parenthèses ouvrante et fermante, et de la virgule, par exemple.

On commence par fixer un codage des machines de Turing. Soit $\mathcal{M} = (\Sigma, S, T)$ une machine de Turing déterministe à un ruban. On peut supposer que $\square = 1$, $\Sigma = \{2, 3, \ldots, k\}$, $\vdash = k+1$, $S = \{k+2, \ldots, k+\ell\}$, $\top = k+\ell+1$, $\bot = k+\ell+2$, et que les déplacements $+1, 0, -1$ sont codés par les entiers $k+\ell+3, k+\ell+4, k+\ell+5$. Pour simplifier, tous ces entiers sont représentés en binaire sur exactement $1 + [\log_2(k + \ell + 6)]$ bits, en ajoutant si nécessaire suffisamment de 0 à gauche pour avoir des codes de même longueur. Le code $\mathcal{M}^\bullet$ de $\mathcal{M}$ est constitué de 3 mots : l'entier k en binaire, suivi de l'entier ℓ en binaire, suivi d'une énumération des couples $((s, x), (s', x', d))$ appartenant à la fonction de transition T, séparés les uns des autres par des virgules.

$\mathcal{U}$ a 2 rubans, encore une commodité. Le deuxième ruban contient la configuration courante (s, u, v) écrite ainsi, en binaire. $\mathcal{U}$ commence par chercher le quatrième mot de son entrée, w donc (et échoue si l'entrée n'est pas de la forme voulue), puis écrit la configuration initiale $(\vdash, \varepsilon, w)$ sur le deuxième ruban. Dans le cours de l'exécution, $\mathcal{U}$ cherche successivement dans le premier ruban les couples $((s, x), (s', x', d))$ commençant par l'état s écrit sur le deuxième ruban (s, u, v), et teste si la lettre lue x est la première lettre de v : si oui, la règle est appliquée et la nouvelle configuration est écrite sur le deuxième ruban ; sinon, $\mathcal{U}$ cherche un nouveau couple de T. Quand $\mathcal{M}$ s'arrête, $\mathcal{U}$ fait de même.

Il faut remarquer que n'importe quel ensemble non vide peut être pris comme alphabet d'une machine de Turing universelle. En effet, étant donné un ensemble non vide Γ, n'importe quel ensemble peut être codé par des mots de longueur fixe sur $\Gamma + \{\square\}$ (autrement dit des listes de mots sur Γ). Par exemple, si $\Gamma = \{1\}$, l'alphabet Σ à cinq lettres de l'exemple 2.32 peut être codé de la manière suivante : $0 \mapsto \square\square\square, 1 \mapsto \square\square 1$, "("$\mapsto \square 1\square$, ")"$\mapsto \square 11$, ","$\mapsto 1\square\square$.

2.7 Indécidabilité

2.33 Théorème (Problème de l'arrêt) Étant donné un codage des machines de Turing dans l'alphabet Σ, le langage $\texttt{Halt} = \{\mathcal{M}^\bullet \# w^\bullet, \mathcal{M}(w)\downarrow\} \subseteq (\Sigma \cup \{\#\})^*$, appelé problème de l'arrêt, est **RE** mais pas **R**.

Le langage `Halt` est donc constitué des mots qui codent une machine et une entrée telles que la machine s'arrête sur cette entrée.

Preuve — Soit $\mathcal{U}$ une machine de Turing universelle d'alphabet Σ : $\mathcal{U}(\mathcal{M}^\bullet \# w^\bullet) = \mathcal{M}(w)^\bullet$, donc $\mathcal{U}$ accepte $\mathcal{M}^\bullet \# w^\bullet$ si $\mathcal{M}(w)$ converge et diverge sinon. Par conséquent, `Halt` est **RE**.

Montrons par l'absurde que `Halt` n'est pas **R**. Supposons au contraire qu'il existe une machine de Turing $\mathcal{N}$ qui décide `Halt`. On utilise une méthode très classique, dite de "diagonalisation". On construit une machine $\mathcal{D}$ qui, sur l'entrée $\mathcal{M}^\bullet$, simule le calcul de $\mathcal{N}$ sur l'entrée $\mathcal{M}^\bullet \# \mathcal{M}^\bullet$ jusqu'au moment d'entrer dans une configuration finale, ce qui arrive toujours puisque $\mathcal{N}$ décide `Halt` ; à ce moment, si $\mathcal{N}$ rejette, $\mathcal{D}$ accepte, et si $\mathcal{N}$ accepte, $\mathcal{D}$ diverge (par exemple en entrant dans un état qui bouge indéfiniment vers la droite). Appliquons $\mathcal{D}$ à $\mathcal{D}^\bullet$. Si $\mathcal{D}(\mathcal{D}^\bullet)\uparrow$, c'est que $\mathcal{N}$ accepte l'entrée $\mathcal{D}^\bullet \# \mathcal{D}^\bullet$, et donc que $\mathcal{D}^\bullet \# \mathcal{D}^\bullet \in \texttt{Halt}$, ce qui signifie que $\mathcal{D}(\mathcal{D}^\bullet)\downarrow$. Si $\mathcal{D}(\mathcal{D}^\bullet)\downarrow$, c'est que $\mathcal{N}$ rejette l'entrée $\mathcal{D}^\bullet \# \mathcal{D}^\bullet$, et donc que $\mathcal{D}^\bullet \# \mathcal{D}^\bullet \notin \texttt{Halt}$, ce qui signifie que $\mathcal{D}(\mathcal{D}^\bullet)\uparrow$. On a une contradiction dans les deux cas. ∎

2.34 Corollaire **R** est strictement incluse dans **RE**.

2.35 Exemple Les langage $L = \{\mathcal{M}^\bullet, \mathcal{M}(w)\downarrow \text{ pour tout } w\}$ n'est pas récursif. Pour montrer cela, on utilise une autre méthode très classique, dite de réduction, qui sera largement développée dans la section 4.

On réduit donc `Halt` au langage L. On procède par l'absurde en supposant qu'il existe une machine de Turing qui décide L, et on définit une fonction calculable $f : \Sigma^* \to \Sigma^*$ telle que

$$f(u) \in L \text{ si, et seulement si, } u \in \texttt{Halt}.$$

Étant donné $u = \mathcal{M}^\bullet \# w^\bullet$, $f(u)$ est le code $\mathcal{M}'^\bullet$ de la machine $\mathcal{M}'$ suivante : sur l'entrée v, $\mathcal{M}'$ simule $\mathcal{M}$ sur w si $v = w$, s'arrête sinon. Si u n'est pas de la forme $\mathcal{M}^\bullet \# w^\bullet$, $f(u)$ est indéfinie, par exemple. Il est clair que la machine $\mathcal{M}'$ obtenue converge sur toutes les entrées si, et seulement si, $\mathcal{M}(w)$ converge, c'est-à-dire $\mathcal{M}'^\bullet \in L$ si, et seulement si, $\mathcal{M}^\bullet \# w^\bullet \in \texttt{Halt}$. Ainsi, on a obtenu un algorithme qui décide `Halt`, en contradiction avec le théorème 2.33.

2.36 Exercice Les langages suivants ne sont pas récursifs.

1. $\{\mathcal{M}^\bullet \# w^\bullet, \mathcal{M}(w) = v \text{ pour un certain } v\}$,
2. $\{\mathcal{M}^\bullet \# w^\bullet, \text{ le calcul de } \mathcal{M} \text{ sur l'entrée } w \text{ utilise tous les états de } \mathcal{M}\}$,
3. $\{\mathcal{M}^\bullet \# w^\bullet \# v^\bullet, \mathcal{M}(w) = v\}$.

2.37 Théorème (Rice) Si X est un ensemble de langages non vide et strictement inclus dans **RE**, alors le langage constitué des codes $\mathcal{M}^\bullet$, où $\mathcal{M}$ est une machine de Turing qui accepte un langage de X, est indécidable.

Autrement dit, toute propriété non triviale des machines de Turing est indécidable.

Preuve — Soit R le langage en question. On peut supposer que $\varnothing \notin X$, et on prend $L \in X$ (si $\varnothing \in X$, on prend $L \notin X$).

Soit $\mathcal{M}_{\texttt{Halt}}$ (resp. $\mathcal{M}_L$) une machine de Turing qui accepte `Halt` (resp. L). On va définir une réduction du problème de l'arrêt au problème considéré R, et pour cela, construire une machine $\mathcal{N}$. Sur l'entrée w, $\mathcal{N}$ calcule le code $\mathcal{M}_w^\bullet$ de la machine suivante $\mathcal{M}_w$. Sur l'entrée v, $\mathcal{M}_w$ simule $\mathcal{M}_{\texttt{Halt}}$ sur w ; si $\mathcal{M}_{\texttt{Halt}}(w) \uparrow$, alors $\mathcal{M}_w(v) \uparrow$; si en revanche $\mathcal{M}_{\texttt{Halt}}$ accepte w, alors $\mathcal{M}_w$ continue et simule $\mathcal{M}_L$ sur v. Autrement dit :

$$\mathcal{M}_w(v) = \text{ si } \mathcal{M}_{\texttt{Halt}}(w) \downarrow_o \text{ alors } \mathcal{M}_L(v) \text{ sinon } \uparrow .$$

Clairement, $\mathcal{M}_w$ accepte soit le langage vide $\varnothing \notin X$ si $w \notin \texttt{Halt}$, soit le même langage que $\mathcal{M}_L$, c'est-à-dire le langage $L \in X$, si $w \in \texttt{Halt}$. On en conclut que $\mathcal{M}_w^\bullet \in R$ si, et seulement si, $w \in \texttt{Halt}$. Donc `Halt` se réduit à R et R n'est pas décidable. ∎

2.38 Définition (Inséparabilité récursive) Soient L_1 et L_2 deux langages de même alphabet. On dit que L_1 et L_2 sont *récursivement inséparables* s'il n'existe pas de langage récursif R tel que $L_1 \cap R = \varnothing$ et $L_2 \subseteq R$.

Le lemme suivant est évident.

2.39 Lemme 1. Soient $L_1, L_2, L_1', L_2' \subseteq \Sigma^*$ tels que $L_1 \subseteq L_2$ et $L_1' \subseteq L_2'$. Si L_1 et L_1' sont récursivement inséparables, alors il en est de même de L_2 et L_2'.

2. Soient $L_1, L_1' \subseteq \Sigma_1^*$ et f une fonction Turing-calculable de Σ_1^* dans Σ_2^*. Si $f(L_1)$ et $f(L_1')$ sont récursivement inséparables, alors il en est de même de L_1 et L_1'.

2.40 Proposition 1. Les deux langages $L_1 = \{\mathcal{M}^\bullet, \mathcal{M}(\mathcal{M}^\bullet) \downarrow_o\}$ et $L_2 = \{\mathcal{M}^\bullet, \mathcal{M}(\mathcal{M}^\bullet) \downarrow_n\}$ sont récursivement inséparables.

2. Les deux langages $L_1' = \{\mathcal{M}^\bullet, \mathcal{M}(\varepsilon) \downarrow_o\}$ et $L_2' = \{\mathcal{M}^\bullet, \mathcal{M}(\varepsilon) \downarrow_n\}$ sont récursivement inséparables.

Preuve —

1. Supposons qu'il existe un langage récursif R qui sépare L_1 et L_2. Soit donc $\mathcal{M}_R$ une machine de Turing qui décide R. En particulier, $\mathcal{M}_R(\mathcal{M}_R^\bullet) \downarrow$.
   - Si $\mathcal{M}_R(\mathcal{M}_R^\bullet) \downarrow_o$, c'est que $\mathcal{M}_R^\bullet \in L_1$, donc $\mathcal{M}_R^\bullet \notin R$ puisque $L_1 \cap R = \varnothing$; comme $\mathcal{M}_R$ décide R, cela signifie que $\mathcal{M}_R(\mathcal{M}_R^\bullet) \downarrow_n$, une contradiction.
   - Si $\mathcal{M}_R(\mathcal{M}_R^\bullet) \downarrow_n$, c'est que $\mathcal{M}_R^\bullet \in L_2 \subseteq R$, c'est-à-dire $\mathcal{M}_R(\mathcal{M}_R^\bullet) \downarrow_o$, une contradiction.
2. Supposons qu'il existe un langage récursif R′ qui sépare L_1' et L_2'. Si $\mathcal{M}$ est une machine de Turing quelconque, soit $\mathcal{M}'$ la machine qui, sur n'importe quelle entrée, écrit $\mathcal{M}^\bullet$ et simule $\mathcal{M}$ sur $\mathcal{M}^\bullet$. Donc $\mathcal{M}(\mathcal{M}^\bullet) \downarrow_o$ si, et seulement si, $\mathcal{M}'(\varepsilon) \downarrow_o$, et $\mathcal{M}(\mathcal{M}^\bullet) \downarrow_n$ si, et seulement si, $\mathcal{M}'(\varepsilon) \downarrow_n$:

   $$\mathcal{M}^\bullet \in L_1 \Leftrightarrow \mathcal{M}'^\bullet \in L_1' \text{ et } \mathcal{M}^\bullet \in L_2 \Leftrightarrow \mathcal{M}'^\bullet \in L_2'.$$

   On définit alors la machine N suivante : sur l'entrée $\mathcal{M}^\bullet$, N calcule $\mathcal{M}'^\bullet$; N accepte si $\mathcal{M}'^\bullet \in R'$, rejette sinon ; sur une entrée qui n'est pas de la forme $\mathcal{M}^\bullet$, N accepte. Le langage décidé par N sépare alors L_1 et L_2, en contradiction avec 1.

∎

Pour compléter cette lecture sur la théorie de la calculabilité, avec en particulier un œil sur d'autres modèles de calcul comme les automates, on pourra consulter [3, 6].

3 Logique et arithmétique

3.1 Calcul des séquents

3.1 Définition (Langage du 1er ordre) Un *langage du 1er ordre* est consitué d'un ensemble dénombrable V de *variables*, d'un ensemble au plus dénombrable de *symboles de fonction* et d'un ensemble au plus dénombrable de *symboles de relation*. Chaque symbole de fonction ou de relation est muni d'une *arité*, qui est un entier.

Soit donc fixé un langage du premier ordre.

3.2 Définition (Termes, formules) Les *termes (du 1er ordre)* sont construits de la façon suivante : une variable est un terme ; si $t_1, \ldots, t_n$ sont des termes et f est un symbole de fonction n-aire, alors $f(t_1, \ldots, t_n)$ est un terme.

Une *formule atomique (du 1er ordre)* est soit v (vrai), soit f (faux), soit $R(t_1, \ldots, t_n)$ pour des termes $t_1, \ldots, t_n$ et un symbole de relation n-aire R. Les *formules (du 1er ordre)* sont construites à partir des formules atomiques, des *connecteurs* et des *quantificateurs* suivants : les formules atomiques sont des formules ; si A et B sont des formules, $\neg A$ (négation), $A \wedge B$ (et), $A \vee B$ (ou) et $A \Rightarrow B$ (implique) sont des formules ; si A est une formule et x une variable, $\forall x A$ (quel que soit) et $\exists x A$ (il existe) sont des formules.

On considère donc d'emblée le *calcul des prédicats* (ou logique du 1er ordre), fondé sur un langage du 1er ordre. Le *calcul propositionnel* peut être obtenu en prenant comme ensemble de symboles de fonction l'ensemble vide et des symboles de relation tous d'arité 0.

3.3 Définition (Calcul des séquents, Gentzen, 1934) Un *séquent* est la donnée de deux listes de formules ; un séquent est noté $\Gamma \vdash \Delta$ où Γ et Δ sont deux listes de formules. Le symbole $\vdash$ se lit : thèse. Une *preuve dans le calcul des séquents LK* est un arbre dont les sommets sont étiquetés par des séquents et construit au moyen des règles décrites dans la table 1.

Intuitivement, un séquent $\Gamma \vdash \Delta$, où Γ est la liste $A_1, \ldots, A_n$ et Δ est la liste $B_1, \ldots, B_p$, signifie que de la conjonction des A_i on déduit la disjonction des B_j.

Il existe plusieurs autres systèmes de déduction, notamment les systèmes de Hilbert (antérieurs) et la déduction naturelle (due à Gentzen et Prawitz, plus proche du λ-calcul). Un avantage du calcul des séquents est la symétrie des règles (gauche, droite) ; le fait que toutes les règles logiques du calcul introduisent un connecteur ou un quantificateur (à droite ou à gauche) le rend aussi particulièrement adapté à la démonstration du théorème de complétude (théorème 3.8).

Observons que les règles d'introduction de $\wedge$ et $\vee$, ainsi que les règles d'introduction de $\exists$ à gauche et de $\forall$ à droite sont *réversibles*, c'est-à-dire que la conclusion est prouvable si, et seulement si, ses prémisses le sont : dans un sens, c'est trivial ; pour montrer l'autre direction, on utilise la règle de coupure. Par exemple pour la règle $\vee$ à gauche, on déduit $\Gamma, A \vdash \Delta$ et $\Gamma, B \vdash \Delta$ de $\Gamma, A \vee B \vdash \Delta$:

$$\dfrac{\Gamma, A \vee B \vdash \Delta \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A}}{A \vdash A, B}\,\text{aff.}}{A \vdash A \vee B}\,\vee}{\Gamma, A \vdash \Delta}\,\text{coupure} \qquad\qquad \dfrac{\Gamma, A \vee B \vdash \Delta \qquad \dfrac{\dfrac{\dfrac{}{B \vdash B}}{B \vdash A, B}\,\text{aff.}}{B \vdash A \vee B}\,\vee}{\Gamma, B \vdash \Delta}\,\text{coupure}$$

Désormais, en général, on n'écrira pas les occurrences de la règle d'échange. Notons la propriété suivante, immédiate, de LK sans coupure.

3.4 Proposition (Propriété de la sous-formule) Dans une preuve sans coupure, toute formule apparaissant dans un séquent est sous-formule d'une formule du séquent conclusion.

	Identité	*Coupure*
	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma \vdash C, \Delta \qquad \Gamma', C \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$
	Règles logiques	
ν	$\dfrac{\Gamma \vdash \Delta}{\Gamma, \nu \vdash \Delta}$	$\dfrac{}{\Gamma \vdash \nu, \Delta}$
f	$\dfrac{}{\Gamma, f \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash f, \Delta}$
$\neg$	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$
$\wedge$	$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$
$\vee$	$\dfrac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}$
$\Rightarrow$	$\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \Rightarrow B \vdash \Delta, \Delta'}$	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}$
$\forall$	$\dfrac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x A, \Delta}$ $(*)$
$\exists$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}$ $(*)$	$\dfrac{\Gamma \vdash A[t/x], \Delta}{\Gamma \vdash \exists x A, \Delta}$
$(*)$ x n'a pas d'occurrence libre dans le contexte Γ, Δ		
	Règles structurelles	
affaiblissement	$\dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}$
contraction	$\dfrac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}$	$\dfrac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}$
échange	$\dfrac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}$

Tab. 1 – Règles du calcul des séquents LK.

La formule $A \Leftrightarrow B$ est une abbréviation de $(A \Rightarrow B) \wedge (B \Rightarrow A)$. On prouvera à titre d'exercice la proposition suivante :

3.5 Proposition Les formules suivantes sont prouvables :
- Involutivité de la négation : $\neg\neg A \Leftrightarrow A$, et loi de Pierce : $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$. L'implication $\neg\neg A \Rightarrow A$ et la loi de Pierce sont des formules importantes non prouvables dans le calcul des séquents *intuitionniste*, c'est-à-dire dans lequel on se limite à des séquents ayant au plus une formule à droite.
- Règles de De morgan : $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$, $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$, $\neg\exists x A \Leftrightarrow \forall x \neg A$, $\neg\forall x A \Leftrightarrow \exists x \neg A$, $\neg\nu \Leftrightarrow f$, $\neg f \Leftrightarrow \nu$.
- Implication : $(A \Rightarrow B) \Leftrightarrow (B \vee \neg A)$, $(A \Rightarrow \neg B) \Leftrightarrow (B \Rightarrow \neg A)$.
- Associativité : $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$, $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$.
- Commutativité : $(A \wedge B) \Leftrightarrow (B \wedge A)$, $(A \vee B) \Leftrightarrow (B \vee A)$.
- Constantes : $(A \wedge \nu) \Leftrightarrow A$, $(A \wedge f) \Leftrightarrow f$, $(A \vee \nu) \Leftrightarrow \nu$, $(A \vee f) \Leftrightarrow A$, $f \Rightarrow A$, $A \Rightarrow \nu$.
- Idempotence : $(A \wedge A) \Leftrightarrow A \Leftrightarrow (A \vee A)$.
- Distributivité : $(A \wedge B) \vee C \Leftrightarrow ((A \vee C) \wedge (B \vee C))$, $(A \vee B) \wedge C \Leftrightarrow ((A \wedge C) \vee (B \wedge C))$.
- Commutations de quantificateurs : $\forall x \forall y A \Leftrightarrow \forall y \forall x A$, $\exists x \exists y A \Leftrightarrow \exists y \exists x A$, $\exists x \forall y A \Rightarrow \forall y \exists x A$.
- Quantificateurs et connecteurs (si x n'a pas d'occurrence libre dans B et Q est $\forall$ ou $\exists$) : $Qx(A \wedge B) \Leftrightarrow (QxA) \wedge B$, $Qx(A \vee B) \Leftrightarrow (QxA) \vee B$.
- $(A \Rightarrow B) \vee (B \Rightarrow A)$.

3.6 Corollaire Toute formule est équivalente à une formule sous forme prénexe normale conjonctive (resp. disjonctive).

3.2 Modèles et théorème de complétude

3.7 Définition Un *modèle* M est la donnée ;
- d'un ensemble $|M|$,
- pour tout symbole de fonction f d'arité k, d'une fonction $f_M : |M|^k \to |M|$,
- pour tout symbole de relation R d'arité k, d'une relation $R_M \subseteq |M|^k$.

Une *valuation* est une fonction de l'ensemble des variables du 1er ordre dans $|M|$. Si ν est une *valuation*, l'*interprétation* $t_{M,\nu} \in |M|$ d'un terme quelconque t : si $t = x$ est une variable, $t_{M,\nu} = \nu(x)$, et si $t = f(t_1, \ldots, t_k)$, $t_{(M,\nu)} = f_M((t_1)_{M,\nu}, \ldots, (t_k)_{M,\nu})$. La *valeur de vérité* $A_{M,\nu}$ d'une formule A quelconque dans (M, ν) est définie comme suit :
- si $A = R(t_1, \ldots, t_k)$ est atomique, $A_{M,\nu}$ est vrai si, et seulement si, $((t_1)_{M,\nu}, \ldots, (t_k)_{M,\nu}) \in R_M$;
- si A se termine par un connecteur, on définit $A_{M,\nu}$ de façon évidente ;
- si $A = \forall x B$, $A_{M,\nu}$ est vrai si, et seulement si, $B_{M,\nu'}$ est vrai pour toute valuation ν' telle que $\nu'(y) = \nu(y)$ quand $y \neq x$;
- si $A = \exists x B$, $A_{M,\nu}$ est vrai si, et seulement s'il existe une valuation ν' telle que $B_{M,\nu'}$ soit vrai et $\nu'(y) = \nu(y)$ quand $y \neq x$.

Observons que $A_{M,\nu}$ ne dépend pas des valeurs que prend ν pour des variables non libres dans A. En particulier, si A est close, $A_{M,\nu} = A_M$ est indépendant de la valuation.

Soit A une formule close : on dit qu'un modèle M *satisfait* A, et on le note $M \models A$, lorsque A_M est vrai, que A est *satisfiable* lorsqu'il existe un modèle qui la satisfait, *contradictoire* sinon. Si $\Gamma \vdash \Delta$ est le séquent $A_1, \ldots, A_n \vdash B_1, \ldots, B_p$, on dit que M satisfait $\Gamma \vdash \Delta$ lorsque M satisfait $\bigwedge_{i=1}^{n} A_i \Rightarrow \bigvee_{j=1}^{p} B_j$. Si T est un ensemble de formules closes, on dit que T est satisfiable ou contradictoire lorsque la conjonction des formules de T l'est.

Fixons une énumération $(t_n)_{n \geqslant 1}$ de l'ensemble dénombrable des termes du 1er ordre.

3.8 Théorème (Complétude, Gödel, 1930) Un séquent $\Gamma \vdash \Delta$ est prouvable sans coupure dans LK si, et seulement si, tout modèle M satisfait la clôture de $\Gamma \vdash \Delta$.

Preuve — La partie condition suffisante est appelée la propriété de *correction* et se prouve par une induction évidente.

Dans l'autre direction, on se limite aux formules construites sans l'implication (c'est possible en vertu de la proposition 3.5) et on montre la contraposée : si un séquent $\Gamma \vdash \Delta$ n'est pas prouvable sans coupure dans LK, alors il existe un modèle M qui ne satisfait pas la clôture de $\Gamma \vdash \Delta$. On définit pour cela une suite $((\Gamma_n \vdash \Delta_n), \omega_n)_{n \geqslant 0}$, où $\Gamma_n \vdash \Delta_n$ est un séquent non-prouvable sans coupure dans LK et ω_n est un ordre linéaire sur les formules de Γ_n, Δ_n. Pour $\Gamma_0 \vdash \Delta_0$, on prend $\Gamma \vdash \Delta$, et ω_0 est la liste Γ, Δ. Étant donné $\Gamma_n \vdash \Delta_n$, avec $\omega_n = A_1, \ldots, A_p$, on définit $\Gamma_{n+1} \vdash \Delta_{n+1}$ par induction sur A_1.

- Si A_1 est atomique, $\Gamma_{n+1} = \Gamma_n$ et $\Delta_{n+1} = \Delta_n$ et $\omega_{n+1} = A_2, \ldots, A_p, A_1$.
- Si $A_1 = A \vee B$ et est dans Δ_n, alors $\Gamma_n \vdash \Delta_n, A, B$ n'est pas prouvable (sans coupure), car sinon :
$$\dfrac{\dfrac{\Gamma_n \vdash \Delta_n, A, B}{\Gamma_n \vdash \Delta_n, A \vee B}\vee}{\Gamma_n \vdash \Delta_n}\text{contr.}$$
donnerait une preuve (sans coupure) de $\Gamma_n \vdash \Delta_n$. On prend $\Gamma_n \vdash \Delta_n, A, B$ comme $\Gamma_{n+1} \vdash \Delta_{n+1}$, avec $\omega_{n+1} = A_2, \ldots, A_p, A, B, A_1$.
- Si $A_1 = A \vee B$ et est dans Γ_n, alors l'un des deux séquents $\Gamma_n, A \vdash \Delta_n$ ou $\Gamma_n, B \vdash \Delta_n$ n'est pas prouvable, car sinon :
$$\dfrac{\dfrac{\Gamma_n, A \vdash \Delta_n \qquad \Gamma_n, B \vdash \Delta_n}{\Gamma_n, A \vee B \vdash \Delta_n}\vee}{\Gamma_n \vdash \Delta_n}\text{contr.}$$
donnerait une preuve de $\Gamma_n \vdash \Delta_n$. On prend comme $\Gamma_{n+1} \vdash \Delta_{n+1}$ celui qui n'est pas prouvable, avec $\omega_{n+1} = A_2, \ldots, A_p, A$ (ou B), A_1.
- Si $A_1 = \forall z A[z]$ et est dans Δ_n, alors soit x_n une variable sans occurrence libre dans $\Gamma_n \vdash \Delta_n$; $\Gamma_n \vdash \Delta_n, A[x_n]$ n'est pas prouvable et on prend ce séquent comme $\Gamma_{n+1} \vdash \Delta_{n+1}$, avec $\omega_{n+1} = A_2, \ldots, A_p, A[x_n], A_1$.
- Si $A_1 = \forall z A[z]$ et est dans Γ_n, alors, $t_1, \ldots, t_n$ étant les n premiers termes du langage, le séquent $\Gamma_n, A[t_1], \ldots, A[t_n] \vdash \Delta_n$ n'est pas prouvable et on le prend comme $\Gamma_{n+1} \vdash \Delta_{n+1}$, avec $\omega_{n+1} = A_2, \ldots, A_p, A[t_1], \ldots, A[t_n], A_1$.
- Si $A_1 = A \wedge B, A \Rightarrow B, \neg A$ ou $\exists z A[z]$, on procède de façon similaire.

Remarquons que pour construire le séquent suivant dans la liste, on utilise quand c'est possible une règle réversible. Les seuls cas où on ne peut le faire sont le cas de $\exists$ à droite et $\forall$ à gauche, et c'est la raison de la complexité de la preuve. Dans le calcul propositionnel, on peut se restreindre à ne construire que des listes finies de séquents ; ici il faut, pour chaque occurrence de $\forall$ à gauche et de $\exists$ à droite, "essayer" tous les termes possibles.

La construction à venir du modèle repose sur les propriétés suivantes :

1. Pour tout $n \geqslant 0$, $\Gamma_n \subseteq \Gamma_{n+1}$ et $\Delta_n \subseteq \Delta_{n+1}$.
2. La suite est infinie.
3. Toute occurrence de sous-formule de $\Gamma \vdash \Delta$ apparaît une infinité de fois en 1ère position dans la liste.
4. Pour tout $n \geqslant 0$, une occurrence de formule apparaît dans au plus une des deux listes Γ_n ou Δ_n (car $\Gamma_n \vdash \Delta_n$ n'est pas prouvable).

On définit un modèle M. $|M|$ est l'ensemble des termes du 1er ordre. Pour éviter les confusions, lorsqu'on fait usage d'un terme t en tant qu'élément de M, on le souligne : $\underline{t}$. On interprète un symbole de fonction f d'arité k par

$$f_M(\underline{u_1}, \ldots, \underline{u_k}) = \underline{f(u_1, \ldots, u_k)}.$$

L'interprétation R_M d'un symbole de relation R d'arité k est un sous-ensemble de $|M|^k$: on pose $R_M(\underline{u_1}, \ldots, \underline{u_k})$ si, et seulement si, la formule atomique $R(u_1, \ldots, u_k)$ apparaît dans Γ_n pour un certain n. Donc si $R(u_1, \ldots, u_k)$ apparaît dans un Δ_n, $R_M(\underline{u_1}, \ldots, \underline{u_k})$ est faux puisque les séquents $\Gamma_n \vdash \Delta_n$ ne sont pas prouvables sans coupure (remarque 4).

Fixons une valuation $\nu : V \to |M|$ en posant $\nu(x) = \underline{x}$ pour toute variable x. On a donc $t_{M,\nu} = \underline{t}$ pour tout terme t. Montrons que pour toute formule A dans $\bigcup_n \Gamma_n$, $A_{M,\nu}$ est vrai et que pour toute formule A dans $\bigcup_n \Delta_n$, $A_{M,\nu}$ est faux. On procède par induction sur A.

- Si A est atomique, A est de la forme $R(u_1, \ldots, u_k)$, et par définition, $A_{M,\nu} = R_M(\underline{u_1}, \ldots, \underline{u_k})$ est vrai si, et seulement si, $A \in \bigcup_n \Gamma_n$. Si $A \in \bigcup_n \Delta_n$, alors $A \notin \bigcup_n \Gamma_n$ et donc $A_{M,\nu}$ est faux.
- Si $A = B \vee C$ et $A \in \bigcup_n \Gamma_n$, alors par construction B ou C est dans $\bigcup_n \Gamma_n$, donc $B_{M,\nu}$ ou $C_{M,\nu}$ est vrai (hyp. d'induction) et $A_{M,\nu}$ est vrai. Si au contraire $A \in \bigcup_n \Delta_n$, alors B et C sont dans $\bigcup_n \Delta_n$, donc $B_{M,\nu}$ et $C_{M,\nu}$ sont faux et $A_{M,\nu}$ est faux.
- Si $A = \forall z B[z]$ et $A \in \bigcup_n \Gamma_n$, alors pour tout terme t, $B[t] \in \bigcup_n \Gamma_n$: en effet, la remarque 3 assure que A apparaît alors une infinité de fois en 1ère position dans la liste, et forcément toujours dans un Γ_n (remarque 4) ; donc $\bigcup_n \Gamma_n$ contient, pour tout $j \geqslant 0$, une occurrence de $B[t_j]$, et donc une occurrence de $B[t]$. Par conséquent, pour tout t, $B[t]_{M,\nu}$ est vrai (hyp. d'induction) et $A_{M,\nu}$ est vrai. Si au contraire $A \in \bigcup_n \Delta_n$, alors A est en 1ère position dans Δ_p pour un ecrtain p, et soit x_p une variable qui n'est pas libre dans $\Gamma_p \vdash \Delta_p$: ainsi, $B[x_p] \in \Delta_{p+1}$, donc faux (hyp. d'induction) et $A_{M,\nu}$ est faux.
- Les autres cas sont similaires.

Alors le séquent $\Gamma \vdash \Delta$, qui n'est autre que $\Gamma_0 \vdash \Delta_0$, n'est pas valide dans (M, ν) et la clôture de $\Gamma \vdash \Delta$ n'est pas valide dans M. ∎

Une version plus forte du théorème de complétude, pour un séquent infini $\Gamma \vdash \Delta$ (ou, ce qui revient au même, pour des preuves en LK avec des séquents hypothèses), se prouve de la même manière, mais cette fois :

1. Dans le cas des règles $\forall$ à droite et $\exists$ à gauche, il se peut que toutes les variables soient libres dans le contexte, ce qui empêcherait de décomposer la formule, comme dans $\exists x A \vdash \{A[t], t \text{ un terme quelconque}\}$: on étend donc le langage avec une infinité dénombrable de nouvelles variables : soit V' un ensemble (ses éléments sont appelés témoins de Henkin) disjoint de l'ensemble des termes du 1er ordre, on étend le modèle en ajoutant évidemment les $\underline{x}$ pour $x \in V'$ et on étend la valuation en posant, pour un tel x, $\nu(x) = \underline{x}$; partant d'un séquent $\Gamma \vdash \Delta$ dans le langage initial (sans V'), un nombre fini de témoins de Henkin apparaît donc dans un séquent quelconque de la liste.
2. De plus, on ne peut plus parcourir cycliquement le séquent : à la place, on considère une énumération $(A_n)_{n \geqslant 1}$ de toutes les formules du 1er ordre et on initialise un compteur à $n = 1$. Au niveau n, on prend successivement comme occurrences de formules actives les formules du séquent qui sont parmi $A_1, \ldots, A_n$, dans cet ordre ; puis on incrémente n. À chaque niveau, on allonge donc la suite d'au plus n séquents. De plus, toute formule apparaissant dans un séquent de la suite est active au moins une fois, et dès qu'elle est active à un niveau elle l'est à tous les niveaux suivants, soit une infinité de fois. Donc les propriétés utiles dans la preuve ci-dessus sont préservées.

3.3 Corollaires du théorème de complétude

Comme conséquences du théorème de complétude, on obtient les théorèmes suivants :

3.9 Théorème (Hauptsatz, Gentzen) Si le séquent $\Gamma \vdash \Delta$ est prouvable dans LK alors il est prouvable sans coupure dans LK.

Preuve — Si $\Gamma \vdash \Delta$ est prouvable, alors il est universellement valide (correction), donc prouvable sans coupure dans LK (contraposée de la complétude). ∎

Le Hauptsatz est une version faible du théorème d'élimination des coupures : il affirme juste l'existence, pour toute preuve, d'une preuve sans coupure de même conclusion, mais ne fournit pas d'algo-

rithme explicite d'élimination des coupures. Expliciter cette transformation de preuves et en comprendre les mécanismes (en lien avec l'informatique) est le principal objectif de la théorie de la démonstration.

3.10 Définition (Théorie) Une *théorie* est un ensemble de formules. On dit qu'une théorie T est *cohérente* lorsque $T \nvdash f$, et *contradictoire* dans le cas contraire.

3.11 Théorème (Löwenheim-Skolem, version descendante faible) Soit T une théorie. T admet un modèle si, et seulement si, T admet un modèle dénombrable.

Preuve — Si T admet un modèle, alors le séquent $T \vdash$ n'est pas prouvable (correction) et donc T admet le modèle dénombrable donné par la preuve du théorème de complétude. ∎

3.12 Théorème (Compacité) Soit T une théorie. T est contradictoire si, et seulement s'il existe un sous-ensemble fini de T qui est contradictoire. De manière équivalente, T est satisfiable si, et seulement si, tout sous-ensemble fini de T est satisfiable.

Preuve — Si T n'est pas satisfiable (est contradictoire), alors le séquent $T \vdash$ est prouvable (complétude), donc il existe un sous-ensemble fini T' de T tel que $T' \vdash$ est prouvable, puisqu'une preuve est constituée d'un nombre fini de règles. Dès lors, par le théorème de correction, T' est contradictoire. ∎

3.13 Théorème (Löwenheim-Skolem, version ascendante) Soit T une théorie. Si T admet un modèle infini, alors T admet un modèle de cardinalité quelconque (en particulier un modèle non-dénombrable).

Preuve — Soient κ un cardinal quelconque, $X = \{c_x, x \in \kappa\}$ un ensemble de constantes de cardinalité κ et $T' = T \cup \{c_x \neq c_y, x, y \in \kappa, x \neq y\}$. Il suffit de démontrer que T' admet un modèle, ce qui est évident d'après le théorème de compacité puisque tout sous-ensemble fini de T' admet un modèle. ∎

Du Hauptsatz, on déduit :

3.14 Théorème (Séquent médian) Soit un séquent $\Gamma \vdash \Delta$ formé de formules prénexes. Si $\Gamma \vdash \Delta$ est prouvable dans LK alors il en existe une preuve de la forme :

$$\begin{array}{c} \text{règles logiques prop.} \\ + \text{ règles structurelles} \\ \vdots \\ \Gamma' \vdash \Delta' \\ \vdots \\ \text{règles sur } \forall, \exists \\ + \text{ contractions} \\ \vdots \\ \Gamma \vdash \Delta \end{array}$$

$\Gamma' \vdash \Delta'$ est appelé le séquent médian.

Preuve — On peut supposer que les axiomes et les affaiblissements portent sur des formules atomiques. Par le Hauptsatz, il existe une preuve sans coupure π de $\Gamma \vdash \Delta$. On construit la preuve π' recherchée par induction sur π.

– Si π est un axiome, $\pi' = \pi$.

– Si π se termine par une règle sur un quantificateur, on applique trivialement l'hyp. d'induction.
– π est

$$\dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \Gamma_1 \vdash A, \Delta_1 & \Gamma_2, B \vdash \Delta_2\end{array}}{\Gamma_1, \Gamma_2, A \Rightarrow B \vdash \Delta_1, \Delta_2}$$

A et B sont sans quantificateur puisque $A \Rightarrow B$ est prénexe. Donc, par hyp. d'induction, on a des preuves π_1' de $\Gamma_1 \vdash A, \Delta_1$, et π_2' de $\Gamma_2, B \vdash \Delta_2$:

$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ \Gamma_1' \vdash A, \Delta_1' & \Gamma_2', B \vdash \Delta_2'\end{array}}{\begin{array}{cc}\vdots & \vdots\\ \Gamma_1 \vdash A, \Delta_1\ 1 & \Gamma_2, B \vdash \Delta_2\ 2\end{array}}$$

avec des séquents médians $\Gamma_1' \vdash A, \Delta_1'$ et $\Gamma_2', B \vdash \Delta_2'$. De plus, quitte à les renommer, on peut supposer distinctes les variables libres de ces deux séquents. Alors π' est :

$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ \Gamma_1' \vdash A, \Delta_1' & \Gamma_2', B \vdash \Delta_2'\end{array}}{\Gamma_1', \Gamma_2', A \Rightarrow B \vdash \Delta_1', \Delta_2'}$$

$$\dfrac{}{\Gamma_1, \Gamma_2', A \Rightarrow B \vdash \Delta_1\ 1, \Delta_2'}$$

$$\dfrac{}{\Gamma_1, \Gamma_2, A \Rightarrow B \vdash \Delta_1, \Delta_2\ 2}$$

– Le cas des autres règles est similaire.

∎

3.15 Théorème (Herbrand, 1928) Soit $A = \exists x_1 \cdots \exists x_n B[x_1, \ldots, x_n]$, où B est sans quantificateur. Si A est prouvable, alors il existe un entier k et des suites de termes $(u_j^i)_{i \leqslant k, j \leqslant n}$ telles que $B[u_1^1, \ldots, u_n^1] \vee \cdots \vee B[u_1^k, \ldots, u_n^k]$ soit prouvable.

Preuve — C'est un corollaire immédiat du théorème du séquent médian. ∎

Le lecteur pourra trouver une plus longue introduction à la logique dans [1, 2] par exemple.

3.4 Arithmétique de Peano

3.16 Définition (Théorie de l'égalité) Soit un langage du 1er ordre contenant le symbole de relation binaire $=$ (*égalité*). La *théorie de l'égalité* est constituée des formules suivantes :

$t = t$	pour tout terme t
$\big(t = u \wedge \varphi(t)\big) \Rightarrow \varphi(u)$	pour tous termes t, u et toute formule φ.

3.17 Exercice Montrer que la relation d'égalité est symétrique et transitive, et que les formules suivantes sont conséquences de la théorie de l'égalité : $(t_1 = u_1 \wedge \cdots \wedge t_n = u_n) \Rightarrow f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)$ pour tout entier n et tout symbole de fonction n-aire f, et $(t_1 = u_1 \wedge \cdots \wedge t_n = u_n \wedge R(t_1, \ldots, t_n)) \Rightarrow R(u_1, \ldots, u_n)$ pour tout entier n et tout symbole de relation n-aire R.

3.18 Définition (Langage de l'arithmétique) Le *langage de l'arithmétique* est consitité d'un ensemble dénombrable V de variables du 1er ordre, des symboles de fonction 0 (*zéro*, 0-aire, constante), s (*successeur*, unaire), $+, \times$ (*addition* et *multiplication*, binaires), et du symbole de relation $=$.

On notera simplement n le terme $s^n(0) = s(\cdots s(0))$ avec n occurrences du symbole s.

3.19 Définition (Arithmétique élémentaire, arithmétique de Peano) La théorie consituée de la théorie de l'égalité et des axiomes suivants (tous supposés universellement quantifiés) :

$$\begin{array}{ll}(A_1) & \neg(s(x) = 0)\\ (A_2) & s(x) = s(y) \Rightarrow x = y\\ (A_3) & x = 0 \vee \exists y(x = s(y))\\ (A_4) & x + 0 = x\\ (A_5) & x + s(y) = s(x + y)\\ (A_6) & x \times 0 = 0\\ (A_7) & x \times s(y) = x + (x \times y)\end{array}$$

est appelée l'*arithmétique élémentaire* et est notée AE. L'*arithmétique de Peano* AP est consituée des axiomes de l'arithmétique élémentaire et du *schéma de récurrence* :

$$\Big(\varphi(0, \vec{p}) \wedge \forall x\big(\varphi(x, \vec{p}) \Rightarrow \varphi(s(x), \vec{p})\big)\Big) \Rightarrow \forall x \varphi(x, \vec{p})$$

pour toute formule $\varphi = \varphi(x, \vec{p})$, où $\vec{p}$ est une liste de variables.

En interprétant le langage de l'arithmétique de façon évidente, on constate que les axiomes de AP sont bien vrais dans $\mathbb{N}$ (c'est pour ça qu'on les a choisis), autrement dit :

3.20 Proposition $\mathbb{N}$ est un modèle de AP.

On dit que $\mathbb{N}$ est le modèle standard de l'arithmétique. D'après le théorème de Löwenheim-Skolem 3.13, AP admet aussi des modèles non-dénombrables : l'étude de tels modèles est l'objet de l'arithmétique non-standard. On note qu'à cause de l'axiome (A_2) qui exprime l'injectivité du successeur, tout modèle de AP est infini.

3.21 Proposition La formule suivante est conséquence de AP : $x + z = y + z \Rightarrow x = y$.

Preuve — On procède par récurrence sur z. C'est prouvable pour $z = 0$ par (A_4). On suppose $\forall xyz(x + z = y + z \Rightarrow x = y)$ et on veut prouver $\forall xyz(x + s(z) = y + s(z) \Rightarrow x = y)$, ce qui découle de (A_5) et (A_2). ∎

3.22 Exercice Montrer que les formules suivantes sont conséquences de AP : $0 + x = x$, $s(x) + y = s(x+y)$, $x+y = y+x$ (commutativité de l'addition), $x+(y+z) = (x+y)+z$ (associativité de l'addition).

3.23 Définition (Ordre) $x \leqslant y$ est (une abréviation de) la formule $\exists z(y = x + z)$, et $x < y$ est la formule $x \leqslant y \wedge x \neq y$, où $x \neq y$ est la formule $\neg(x = y)$.

3.24 Proposition Les formules suivantes, qui expriment que $\leqslant$ est bien une relation d'ordre, sont conséquences de AP : $x \leqslant x$ (réflexivité), $x \leqslant y \wedge y \leqslant x \Rightarrow x = y$ (anti-symétrie), $x \leqslant y \wedge y \leqslant z \Rightarrow x \leqslant z$ (transitivité).

Preuve — Réflexivité : on a bien $x + 0 = x$. Transitivité : $y = x + a$ et $z = y + b$ donnent $z = x + a + b$. Anti-symétrie : $y = x + a$ et $x = y + b$ donnent $x + a + b = x$, d'où $a + b = 0$ d'après (A_4) et la proposition 3.21, donc $a = b = 0$ par $(A_3), (A_5)$ et (A_1), par conséquent $x = y$. ∎

3.25 Exercice 1. Montrer que les formules suivantes sont conséquences de AP : $x \leqslant 0 \Rightarrow x = 0$, $x \leqslant s(y) \Rightarrow x = s(y) \vee x \leqslant y$, $x \leqslant y \vee y \leqslant x$ (ordre total).

2. Soit $\varphi(x)$ une formule quelconque. Montrer que la formule suivante est conséquence de AP :

$$\exists x \varphi(x) \Rightarrow \exists x \Big(\varphi(x) \wedge \forall y \big(y < x \Rightarrow \neg \varphi(y) \big) \Big).$$

Autrement dit, toute propriété non vide est vraie pour un plus petit élément.

Dans la suite, si T est une théorie dans le langage de l'arithmétique, on suppose systématiquement que T contient la théorie de l'égalité.

3.5 Définissabilité et représentabilité

3.26 Définition ($\Delta_0^0, \Delta_1^0, \Sigma_1^0, \Pi_1^0$) L'ensemble des formules Δ_0^0 est le plus petit ensemble X de formules tel que :
- X contient les formules atomiques (du langage de l'arithmétique),
- X est clos par utilisation des connecteurs,
- X est clos par quantification bornée : si $\phi \in X$, t est un terme et x une variable qui n'apparaît pas dans t, $(\exists x < t)\phi = \exists x(x < t \wedge \phi)$ et $(\forall x < t)\phi = \forall x(x < t \Rightarrow \phi)$ sont dans X.

Une formule est dite Σ_1^0 (resp. Π_1^0) si elle est de la forme $\exists x_1 \cdots \exists x_n \phi$ (resp. $\forall x_1 \cdots \forall x_n \phi$) avec $\phi \in \Delta_0^0$. Par abus, on dit qu'une formule est Δ_0^0 (Σ_1^0, Π_1^0) si elle est équivalente à une formule qui l'est.

Une relation $R \subseteq \mathbb{N}^n$ est dite Δ_0^0 (Σ_1^0, Π_1^0) s'il existe une formule $\phi \in \Delta_0^0$ (Σ_1^0, Π_1^0) qui la *définit*, c'est-à-dire telle que $R(x_1, \ldots, x_n)$ si, et seulement si, $\mathbb{N} \models \phi(x_1, \ldots, x_n)$. Une relation $R \subseteq \mathbb{N}^n$ est dite Δ_1^0 si elle est à la fois Σ_1^0 et Π_1^0. Une fonction $f : \mathbb{N}^n \to \mathbb{N}$ est *définie* par une formule ϕ lorsque $y = f(\vec{x})$ si, et seulement si, $\mathbb{N} \models \phi(\vec{x}, y)$.

3.27 Lemme
- La négation échange Σ_1^0 et Π_1^0.
- $\Delta_0^0 \subseteq \Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$.
- Σ_1^0 et Π_1^0 sont closes par conjonction, disjonction et quantification bornée.
- Σ_1^0 (resp. Π_1^0) est close par quantification existentielle (resp. universelle).
- $\Delta_0^0 \subseteq R$.

Preuve — Pour la quantification bornée, on note que $(\forall x < t)\exists y_1 \cdots \exists y_n \phi$ si, et seulement si, $\exists \nu (\forall x < t)(\exists y_1 < \nu) \cdots (\exists y_n < \nu)\phi$: dans le sens de gauche à droite, on prend pour ν le maximum des y_i pour $i = 1, \ldots, n$ et $x < \nu$. Pour montrer la dernière inclusion, on peut par exemple définir une machine de Turing appropriée. Les autres assertions sont immédiates. ∎

3.28 Théorème (Définissabilité)
1. Toute fonction récursive est définissable par une formule Σ_1^0.
2. $RE = \Sigma_1^0$.

Preuve —

1. Induction sur la construction de la fonction récursive au moyen des schémas (R'1), (R2) et (R4) (proposition 1.47). La projection $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$ est définie par $y = x_i$, la constante $c_p^n : \mathbb{N}^n \to \mathbb{N}$ par $y = s^p(0)$, l'addition par $y = x_1 + x_2$, la multiplication par $y = x_1 \times x_2$ et la comparaison par $(x_1 < x_2 \wedge y = 1) \vee (\neg(x_1 < x_2) \wedge y = 0)$. Si $h = f \circ \langle g_1, \ldots, g_n \rangle : \mathbb{N}^k \to \mathbb{N}$ avec $f : \mathbb{N}^n \to \mathbb{N}$ et $g_i : \mathbb{N}^k \to \mathbb{N}$, $i = 1, \ldots, n$, récursives, alors par hypothèse d'induction il existe des formules Σ_1^0 $\phi(\vec{x}, y)$ définissant f et $\psi_i(\vec{t}, x_i)$ définissant g_i, $i = 1, \ldots, n$, donc h est définie par :

$$\exists x_1 \cdots \exists x_n \left(\psi_1(\vec{t}, x_1) \wedge \cdots \wedge \psi_n(\vec{t}, x_n) \wedge \phi(\vec{x}, y) \right),$$

qui est bien Σ_1^0. Si $f(\vec{x}) = (\mu t).(g(\vec{x}, t) = 0)$ avec $g : \mathbb{N}^{n+1} \to \mathbb{N}$ récursive, par hypothèse d'induction il existe une formule $\phi(\vec{x}, t, y)$ définissant g qui est Σ_1^0, donc f est définie par :

$$\phi(\vec{x}, y, 0) \wedge (\forall t < y)\exists z \phi(\vec{x}, t, s(z)),$$

qui est bien Σ_1^0 par le lemme 3.27 (on n'a pas écrit $\neg\phi(\vec{x}, t, 0)$ car a priori cette formule n'est pas Σ_1^0).

2. Soit $R \subseteq \mathbb{N}^n$, qu'on suppose non vide (sinon le résultat est trivial). Si $R \in \mathbf{RE}$, d'après la proposition 2.31, il existe une fonction résursive $f : \mathbb{N}^n \to \mathbb{N}$ telle que $R(\vec{x})$ si, et seulement si, $f(\vec{x})$ est définie, c'est-à-dire $\exists y(f(\vec{x}) = y)$. D'après le 1, il existe donc une formule ϕ qui est Σ_1^0 telle que $R(\vec{x})$ si, et seulement si, $\mathbb{N} \models \exists y \phi(\vec{x}, y)$, donc $R \in \Sigma_1^0$. Réciproquement, si $R \in \Sigma_1^0$, $R(\vec{x})$ est équivalente à $\exists t_1 \cdots \exists t_k \phi(\vec{x}, t_1, \ldots, t_k)$ pour une certaine formule $\phi \in \Delta_0^0$. D'après le lemme 3.27, $\phi \in R$, donc il existe une machine $\mathcal{M}$ qui décide ϕ. Alors la machine qui essaie les valeurs successives de $t_1, \ldots, t_k$ et lance $\mathcal{M}$ sur l'entrée $\vec{x}, t_1, \ldots, t_k$, accepte R, donc $R \in \mathbf{RE}$.

∎

3.29 Corollaire $\mathbf{coRE} = \Pi_1^0$, $R = \Delta_1^0$.

Preuve — Conséquences immédiates du théorème 3.28 combiné avec la proposition 2.12 et le lemme 3.27. ∎

3.30 Définition Soient $R \subseteq \mathbb{N}^n$ et T une théorie dans le langage de l'arithmétique. On dit que R est *représentable* dans T s'il existe une formule $\phi(x_1, \ldots, x_n)$ telle que pour tout $(a_1, \ldots, a_n) \in \mathbb{N}^n$:

$$R(a_1, \ldots, a_n) \Rightarrow T \vdash \phi(a_1, \ldots, a_n)$$
$$\neg R(a_1, \ldots, a_n) \Rightarrow T \vdash \neg\phi(a_1, \ldots, a_n).$$

On dit alors plus précisément que R est représentable par ϕ.

3.31 Lemme Soit $T \supseteq AE$ une théorie cohérente. Alors, pour tous entiers m, n, p :
- $m \neq n$ si, et seulement si, $T \vdash m \neq n$.
- $m + n = p$ si, et seulement si, $T \vdash m + n = p$.
- $mn = p$ si, et seulement si, $T \vdash m \times n = p$.
- $m \leqslant n$ si, et seulement si, $T \vdash m \leqslant n$.

Preuve — On montre par récurrence sur n que pour tout $k > 0$, $T \vdash n + k \neq n$. Si $n = 0$, on a $T \vdash s(k-1) \neq 0$ donc $T \vdash k \neq 0$. Sinon, on suppose que $T \vdash n + k \neq n$ pour tout $k > 0$: donc $T \vdash s(n + k) \neq s(n)$, d'où $T \vdash (n+1) + k \neq n + 1$. Réciproquement, si $T \vdash m \neq n$ et $m = n$, on a $T \vdash m = n$, ce qui est impossible car T est cohérente.

Les deuxième et troisième équivalences sont prouvées par récurrence sur n. La quatrième se déduit immédiatement. ∎

3.32 Lemme Soit t un terme clos du 1er ordre dans le langage de l'arithmétique et $t_{\mathbb{N}} \in \mathbb{N}$ son interprétation dans $\mathbb{N}$. On a $AE \vdash t = t_{\mathbb{N}}$. Si $T \supseteq AE$ une théorie cohérente, alors pour tout entier k, $T \vdash t = k$ implique $t_{\mathbb{N}} = k$.

Preuve — La première assertion se montre par récurrence sur t. Si $t = 0$, on a $t_{\mathbb{N}} = 0$ et $AE \vdash 0 = 0$. Si par exemple $t = s(t')$, alors $t_{\mathbb{N}} = t'_{\mathbb{N}} + 1$; par hypothèse d'induction, $AE \vdash t' = t'_{\mathbb{N}}$, donc $AE \vdash s(t') = t'_{\mathbb{N}} + 1$. Idem pour l'addition et la multiplication.

Pour la deuxième assertion, supposons $T \vdash t = k$ mais $t_{\mathbb{N}} \neq k$. D'après ce qui précède, $T \vdash t = t_{\mathbb{N}}$, donc $T \vdash t_{\mathbb{N}} = k$. Mais $t_{\mathbb{N}} \neq k$, donc d'après le lemme 3.31, $T \vdash t_{\mathbb{N}} \neq k$, absurde car T est cohérente. ∎

3.33 Lemme Pour tout entier n, $AE \vdash \forall x(x < n \Leftrightarrow (x = 0 \vee \cdots \vee x = n-1))$. Si $T \supseteq AE$ est une théorie cohérente et $\phi(x, \vec{y})$ est une formule, alors pour tout entier n :

$$T \vdash (\forall x < n)\phi(x, \vec{y}) \Leftrightarrow \phi(0, \vec{y}) \wedge \cdots \wedge \phi(n-1, \vec{y})$$
$$T \vdash (\exists x < n)\phi(x, \vec{y}) \Leftrightarrow \phi(0, \vec{y}) \vee \cdots \vee \phi(n-1, \vec{y}).$$

Preuve — La première assertion se prouve par récurrence sur n. Pour $n = 0$, c'est immédiat. On a $AE \vdash x < n+1 \Rightarrow x < n \vee x = n$. Donc si $AE \vdash \forall x(x < n \Leftrightarrow (x = 0 \vee \cdots \vee x = n-1))$, on a bien $AE \vdash \forall x(x < n+1 \Rightarrow (x = 0 \vee \cdots \vee x = n))$. Réciproquement, il suffit de remarquer que $AE \vdash x = k \Rightarrow x < n$ pour $k = 0, \ldots, n-1$ par le lemme 3.31. La deuxième assertion s'en déduit facilement. ∎

3.34 Théorème Soit $T \supseteq AE$ une théorie cohérente. Alors :
- Pour toute formule close $\phi \in \Delta_0^0$, $\mathbb{N} \models \phi$ si, et seulement si, $T \vdash \phi$.
- Pour toute formule close $\phi \in \Sigma_1^0$, si $\mathbb{N} \models \phi$, alors $T \vdash \phi$.
- Pour toute formule close $\phi \in \Pi_1^0$, si $T \vdash \phi$, alors $\mathbb{N} \models \phi$.

Preuve —
- On montre par induction sur les formules Δ_0^0 que si $\phi \in \Delta_0^0$, $\mathbb{N} \models \phi$ implique $T \vdash \phi$, et $\mathbb{N} \models \neg\phi$ implique $T \vdash \neg\phi$. Si ϕ est atomique, ϕ est $t = u$ pour des termes clos t et u : d'après le lemme 3.32, $T \vdash t = t_{\mathbb{N}}$ et $T \vdash u = u_{\mathbb{N}}$, donc $t_{\mathbb{N}} = u_{\mathbb{N}}$ si, et seulement si, $T \vdash t = u$ d'après le lemme 3.31. L'utilisation de connecteurs ne pose aucun problème. Regardons le cas de la quantification bornée, par exemple universelle (l'existentielle étant similaire) : si ϕ est $(\forall x < t)\theta$ avec $\theta \in \Delta_0^0$, comme ϕ est close et x n'apparaît pas dans t, t est clos, donc :

$\mathbb{N} \models \phi$	implique	$\mathbb{N} \models \theta[0/x], \ldots, \mathbb{N} \models \theta[t_{\mathbb{N}} - 1/x]$	
	implique	$T \vdash \theta[0/x], \ldots, T \vdash \theta[t_{\mathbb{N}} - 1/x]$	par hypothèse de récurrence
	implique	$T \vdash \theta[0/x] \wedge \cdots \wedge \theta[t_{\mathbb{N}} - 1/x]$	
	implique	$T \vdash (x = 0 \vee \cdots \vee x = t_{\mathbb{N}} - 1) \Rightarrow \theta$	
	implique	$T \vdash (\forall x < t)\theta$	par le lemme 3.33.

- Soit $\phi = \exists x_1 \cdots \exists x_n \theta(\vec{x})$ avec $\theta \in \Delta_0^0$. Si $\mathbb{N} \models \phi$, il existe $\vec{k} \in \mathbb{N}^n$ tel que $\mathbb{N} \models \theta(\vec{k})$, d'où $T \vdash \exists\vec{x}\theta(\vec{x})$.
- Si $\phi \in \Pi_1^0$, $\neg\phi \in \Sigma_1^0$. Si $T \vdash \phi$, $T \nvdash \neg\phi$ puisque T est cohérente, donc $\mathbb{N} \not\models \neg\phi$ d'après le point précédent, c'est-à-dire $\mathbb{N} \models \phi$.

∎

3.35 Théorème (Représentabilité, Rosser) Toute relation récursive est représentable dans AE, donc dans toute théorie contenant AE.

Preuve — Soit T une théorie contenant AE. Si T est incohérente, l'assertion est triviale. On suppose donc T est cohérente. Soit $R \subseteq \mathbb{N}^n$ une relation récursive, donc définie par une relation $\phi \in \Delta_1^0$ d'après le corollaire 3.29, et soit $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$. Comme $\phi \in \Sigma_1^0$, on a $R(\vec{a})$ implique $\mathbb{N} \models \phi(\vec{a})$ implique $T \vdash \phi(\vec{a})$ d'après le théorème 3.34. D'autre part, si $\neg R(\vec{a})$, on a $\mathbb{N} \not\models \phi(\vec{a})$, autrement dit $\mathbb{N} \models \neg\phi(\vec{a})$, d'où $T \vdash \neg\phi(\vec{a})$ puisque $\neg\phi \in \Sigma_1^0$. ∎

3.6 Théorèmes d'indécidabilité et d'incomplétude

On arithmétise la logique, c'est-à-dire qu'on code les expressions logiques (termes, formules) sous forme d'entiers et leurs propriétés sous forme d'ensembles d'entiers. Pour cela, on utilise le deuxième bon codage p.r. des suites considéré en section 1.3 dans l'exemple 1.26. Le code canonique de la suite $\langle n_0, \ldots, n_{k-1} \rangle$ est alors :

$$\begin{cases} \sharp\langle n_0, \ldots, n_{k-1} \rangle = 2^{1+n_0} 3^{1+n_1} \cdots p_{k-1}^{1+n_{k-1}} & \text{si } k \geqslant 1, \\ \sharp\varepsilon = 1 & \text{si } k = 0, \end{cases}$$

où $p_0, p_1, \ldots$ est une énumération croissante des nombres premiers. (En fait le premier entier qui code la suite vide ε est 0 et non 1, mais il est plus commode de choisir $\sharp\varepsilon = 1$ comme code canonique.) On a remarqué dans le lemme 1.30 que la composition des suites induit une fonction p.r. de $\mathbb{N} \times \mathbb{N}$ dans $\mathbb{N}$,

ce qu'on peut reformuler et préciser ainsi : la fonction de $\mathbb{N}\times\mathbb{N}$ dans $\mathbb{N}$, notée abusivement $*$ comme la composition et définie par $\sharp u * \sharp v = \sharp(u * v)$ et $a * b =$ par exemple 0 si a ou b n'est pas un code canonique, est p.r. On a évidemment :

3.36 Lemme Si $v = w_1 * u * w_2$ (c'est-à-dire u est une sous-suite de v), alors $\sharp u \leqslant \sharp v$.

À chaque symbole x du langage de l'arithmétique (l'ensemble dénombrable de variables étant $V = \{v_0, v_1, v_2, \ldots\}$) et chaque accessoire de ponctuation, on associe un entier $\ulcorner x \urcorner$:

x	v_i	0	s	$+$	$\times$	$=$	$\neg$	$\wedge$	$\forall$	$($	$)$	$,$
$\ulcorner x \urcorner$	$2i$	1	3	5	7	9	11	13	15	17	19	21

On s'est limité aux connecteurs $\neg$ et $\wedge$ et au quantificateur $\forall$ car on sait que tout formule est équivalente à une formule écrite avec ces symboles.

3.37 Définition Une *expression logique* est un mot sur l'alphabet ci-dessus. Le *code* d'une expression logique $u = \langle x_1, \ldots, x_{k-1}\rangle$ est l'entier $u^\bullet = \sharp\langle \ulcorner x_1 \urcorner, \ldots, \ulcorner x_{k-1} \urcorner\rangle$.

3.38 Lemme Les ensembles suivants sont p.r. :

$$\begin{aligned} \mathit{Var} &= \{2i, i \in \mathbb{N}\} & \mathit{Symb} &= \{1, 3, \ldots, 21\} \cup \mathit{Var} \\ \mathit{TerA} &= \{\ulcorner 0 \urcorner\} \cup \mathit{Var} & \mathit{ForA} &= \{\varphi^\bullet, \varphi \text{ une formule atomique}\} \\ \mathit{Ter} &= \{t^\bullet, t \text{ un terme}\} & \mathit{For} &= \{\varphi^\bullet, \varphi \text{ une formule}\} \end{aligned}$$

Le prédicat et l'ensemble suivants sont p.r. :

$$\begin{aligned} \mathit{Lib}(x, y) \Leftrightarrow\ & \mathit{For}(x), \mathit{Var}(y) \text{ et la variable codée par } y \text{ a une occurrence libre} \\ & \text{dans la formule codée par } x \\ \mathit{ForCl} =\ & \{\varphi^\bullet, \varphi \text{ une formule close}\}. \end{aligned}$$

sont p.r. La fonction suivante $\mathbb{N}^3 \to \mathbb{N}$ est p.r. :

$$\mathit{Sub}(x, u, t) = \big(\exists v_{(u \operatorname{div} 2)}\big((v_{(u \operatorname{div} 2)} = \tau_t) \wedge \varphi_x\big)\big)^\bullet,$$

où τ_t est le terme codé par t si $\mathit{Ter}(t)$ et 0 sinon, et φ_x est la formule codée par x si $\mathit{For}(x)$ et v sinon.

Preuve — Les cas de *Var*, *Symb* et *TerA* sont triviaux. Pour *Ter*, d'après le lemme 3.36, on a $\mathit{Ter}(x)$ si, et seulement si, $\mathit{TerA}(x)$ ou il existe $y_1, y_2 < x$ tels que $\mathit{Ter}(y_1)$, $\mathit{Ter}(y_2)$ et

$$x = s^\bullet({}^\bullet{*}y_1{*})^\bullet \text{ ou } x = ({}^\bullet{*}y_1 * +^\bullet * y_2{*})^\bullet \text{ ou } x = ({}^\bullet{*}y_1 * \times^\bullet * y_2{*})^\bullet,$$

donc *Ter* est p.r. Pour *ForA*, on a $\mathit{ForA}(x)$ si, et seulement s'il existe $y_1, y_2 < x$ tels que $\mathit{Ter}(y_1)$, $\mathit{Ter}(y_2)$ et

$$x = ({}^\bullet{*}y_1{*} =^\bullet {*}y_2{*})^\bullet \text{ ou } x = ({}^\bullet{*}y_1{*} <^\bullet {*}y_2{*})^\bullet.$$

Le cas de *For* se traite comme celui de *Ter*. Pour *Lib*, on vérifie aisément que la troisième condition est bien p.r. Enfin $\mathit{ForCl}(x)$ si, et seulement si, $\mathit{For}(x)$ et pour tout $y \leqslant x$, non $\mathit{Lib}(x, y)$. ∎

3.39 Définition (Théorie complète, récursivement axiomatisée, décidable) Soit T une théorie dans le langage de l'arithmétique. On dit que T est :
- *complète* lorsque pour toute formule close ϕ, $T \vdash \phi$ ou $T \vdash \neg\phi$,
- *récursivement axiomatisée* lorsque $\{\phi^\bullet, \phi \in T\}$ est récursif,
- *décidable* lorsque $\mathit{Thm}_T = \{\phi^\bullet, T \vdash \phi\}$ est récursif.

3.40 Exercice Expliquer pourquoi la décidabilité d'une théorie est une propriété intrinsèque, c'est-à-dire ne dépend pas du codage choisi.

3.41 Lemme Thm_T est récursif si, et seulement si, $ThmCl_T = \{x, ForCl(x) \text{ et } T \vdash \varphi_x\}$ est récursif.

Preuve — $ThmCl_T = Thm_T \cap ForCl$, donc si Thm_T est récursif, $ThmCl_T$ l'est aussi. Réciproquement, il suffit de remarquer que l'application qui à n associe le code de la clôture universelle de φ_n est évidemment récursive. ∎

3.42 Théorème (Indécidabilité, Church) Toute théorie cohérente contenant AE est indécidable.

Preuve — Soit T une telle théorie. Par le théorème de représentabilité 3.35, pour chaque $R \subseteq \mathbb{N}$ récursif, il existe ψ_R telle que pour tout $a \in \mathbb{N}$, $R(a)$ implique $T \vdash \psi_R(a)$, et $\neg R(a)$ implique $T \vdash \neg\psi_R(a)$. Comme T est cohérente, ceci donne : $R(a)$ si, et seulement si, $T \vdash \psi_R(a)$.

Posons $U_T(n, a)$ si, et seulement si, $T \vdash \varphi_n[a/v_0]$. On a $R(a)$ si, et seulement si, $U_T(\psi_R^\bullet, a)$. Procédons par l'absurde et supposons Thm_T récursif. Alors U_T est récursif puisque $U_T(n, a)$ si, et seulement si, $Thm_T(Sub(n, \ulcorner v_0 \urcorner, a^\bullet))$. Soit $D_T(n)$ si, et seulement si, $\neg U_T(n, n)$. Alors D_T est récursif, donc en prenant $k = \psi_{D_T}^\bullet$, on a $D_T(n)$ si, et seulement si, $U_T(k, n)$. Mais alors $D_T(k)$ si, et seulement si, $U_T(k, k)$, en contradiction avec la définition de D_T. ∎

3.43 Corollaire AP est une théorie indécidable. L'ensemble des (codes des) formules vraies dans $\mathbb{N}$ n'est pas récursivement énumérable.

Preuve — Dans le deuxième cas, il suffit d'appliquer le théorème 3.42 à la théorie complète constituée des formules vraies dans $\mathbb{N}$. ∎

3.44 Corollaire Le calcul des prédicats pur dans le langage de l'arithmétique est indécidable.

Preuve — Soit $A = A_1 \wedge \cdots \wedge A_7$ la conjonction des axiomes de AE. Pour toute formule ϕ, on a évidemment $AE \vdash \phi$ si, et seulement si, $\vdash A \Rightarrow \phi$, donc $Thm_{AE}(n)$ si, et seulement si, $Thm_\varnothing((A \Rightarrow \varphi_n)^\bullet)$. La fonction $n \mapsto (A \Rightarrow \varphi_n)^\bullet$ étant clairement p.r., si $Thm_\varnothing$ est récursif, Thm_{AE} l'est aussi. ∎

Quelques théories pas inintéressantes sont toutefois décidables, comme l'arithmétique de Presburger ou la théorie des corps réels fermés. On peut consulter [2]. On montre facilement le lemme suivant, par exemple en définissant une machine de Turing.

3.45 Lemme Si T est une théorie récursivement axiomatisée, alors la relation $Prv_T(p, n)$ si, et seulement si, p est le code d'une preuve de φ_n dans T, est récursive.

3.46 Lemme Toute théorie récursivement axiomatisée et complète est décidable.

Preuve — Soit T une théorie récursivement axiomatisée et complète. Alors $Thm_T(n)$ est équivalente à $\exists p(Prv_T(p, n))$, qui est Σ_1^0 car $Prv_T(p, n)$ est récursif donc Δ_1^0 (lemme 3.41). De plus, comme T est complète, $\neg Thm_T(n)$ est équivalente à $Thm_T(\neg^\bullet * n)$, donc à $\exists p(Prv_T(p, \neg^\bullet * n))$. Par conséquent, Thm_T est récursif et T est décidable. ∎

3.47 Théorème (Incomplétude, Gödel)
1. Toute théorie cohérente et récursivement axiomatisée contenant AE est incomplète.
2. Soient T une théorie récursivement axiomatisée contenant AP, $\pi_T(x, y)$ une formule Σ_1^0 définissant dans $\mathbb{N}$ la relation $Prv_T \subseteq \mathbb{N} \times \mathbb{N}$ et coh_T la formule $\forall x(\neg\pi_T(x, f^\bullet))$. Si T est cohérente, alors $T \nvdash coh_T$.

Preuve —

1. Ce premier théorème est conséquence immédiate du lemme 3.46 et du théorème d'indécidabilité 3.42.
2. Soit T contenant AP. Soit $U_T(n,a)$ la propriété du théorème d'indécidabilité 3.42 : U_T est une relation **RE**, donc Σ_1^0 d'après le théorème de définissabilité 3.28. La relation $D_T(n) \Leftrightarrow \neg U_T(n,n)$ est Π_1^0. Soient donc θ_T une formule Π_1^0 définissant D_T, et δ_T la formule $\theta_T(\theta_T^\bullet)$. Si maintenant T est cohérente et $T \vdash \delta_T$, on a $\mathbb{N} \models \delta_T$ d'après le théorème 3.34 (formule Π_1^0), donc $D_T(\theta_T^\bullet)$, soit $\neg U_T(\theta_T^\bullet)$, c'est-à-dire $T \nvdash \theta_T(\theta_T^\bullet)$, contradiction. Par conséquent, $T \nvdash \delta_T$, c'est-à-dire $D_T(\theta_T^\bullet)$, autrement dit $\mathbb{N} \models \delta_T$.

   On a montré que si T est cohérente, alors $\mathbb{N} \models \delta_T$ et $T \nvdash \delta_T$.

   Pour achever la démonstration de ce deuxième théorème, on se convainc que la preuve ci-dessus est formalisable dans AP : à partir de la définition 3.30, tout est purement syntaxique ; on a besoin de la récurrence à cause de l'utilisation du théorème 3.34. On a donc $AP \vdash \mathit{coh}_T \Rightarrow \delta_T$, et donc $T \vdash \mathit{coh}_T \Rightarrow \delta_T$ puisque $T \supseteq AP$. On a montré aussi que si T est cohérente, $T \nvdash \delta_T$. Donc $T \nvdash \mathit{coh}_T$.

   ∎

Remarquons que d'après le deuxième théorème d'incomplétude ci-dessus, si T est cohérente, $T \nvdash \mathit{coh}_T$, donc d'après le théorème de complétude, il existe un modèle M dénombrable de T tel que $M \not\models \mathit{coh}_T$. Et pourtant T est cohérente si, et seulement si, $\mathbb{N} \models \mathit{coh}_T$.

4 Éléments de complexité algorithmique

4.1 Temps et espace

4.1 Définition (Classe de complexité, complémentaire) Une *classe de complexité* est un ensemble de langages (d'alphabet quelconque). Si X est une classe de complexité, l'ensemble des langages $\Sigma^* \setminus L$ pour $L \in X$ et $L \subseteq \Sigma^*$ est noté **co**X.

Le lemme suivant est évident.

4.2 Lemme Si X est une classe de complexité, alors **co**co**X** = X. Si X et Y sont deux classes de complexité, alors $X \subseteq Y$ si, et seulement si, **co**$X \subseteq$ **co**Y.

4.3 Définition (Temps) Soit une fonction $f : \mathbb{N} \to \mathbb{R}^+$. L'ensemble des langages décidables par une machine de Turing déterministe (resp. non-déterministe) opérant en temps f (définition 2.7) est noté **Time**(f) (resp. **NTime**(f)).

Il est à noter que dans la définition ci-dessus, on ne limite ni le nombre de rubans des machines, ni leur alphabet.

4.4 Théorème Si $L \in$ **Time**$(f(n))$, alors pour tout $\epsilon > 0$, on a $L \in$ **Time**$(\epsilon f(n) + n + 2)$.

Preuve — Soit $M = (\Sigma, S, T)$ une machine déterministe à k rubans qui décide L en temps f. On définit une machine déterministe $M' = (\Sigma', S', T')$ à k' rubans qui décide L en temps $\epsilon f(n) + n + 2$, où :

$$k' = \begin{cases} 2 & \text{si } k = 1 \\ k & \text{si } k > 1. \end{cases}$$

Soit $m = 1 + [6/\epsilon]$. On prend $\Sigma' = \Sigma + \Sigma^{\square m}$ et

$$S' = S + \Sigma^1 + \cdots + \Sigma^{m-1} + \left(S \times \{1,\ldots,m\}^k\right) + \left(S \times \{1,\ldots,m\}^k \times (\Sigma^\square)^{mk}\right) + \left(S \times \{1,\ldots,m\}^k \times (\Sigma^\square)^{2mk}\right).$$

$\mathcal{M}'$ commence par recopier l'entrée sur le deuxième ruban sous forme condensée, par paquets de m lettres : le mot $w = w_1 \cdots w_p \in \Sigma^*$, avec $w_i = x_{i,1} \cdots x_{i,m} \in \Sigma^m$ pour $i < p$ et $w_p = x_{p,1} \cdots x_{p,m'}$ pour un certain $m' \leqslant m$, est écrit $x'_1 \cdots x'_p \in \Sigma'^*$, où $x'_i = (x_{i,1}, \ldots, x_{i,m}) \in \Sigma^m \subseteq \Sigma'$ pour $i < p$ et $x'_p = (x_{p,1}, \ldots, x_{p,m'}, \square, \ldots, \square) \in (\Sigma + \{\square\})^m \subseteq \Sigma'$. Cette opération est effectuée au moyen des états dans $\Sigma^1 + \cdots + \Sigma^{m-1} \subseteq S'$, et prend $n+2$ étapes.

Puis $\mathcal{M}'$ simule une étape de $\mathcal{M}$ en au plus 6 étapes. Au début de la simulation d'une étape, l'état est $(q, j_1, \ldots, j_k) \in S \times \{1, \ldots, m\}^k$, où j_i est la position du i-ème curseur dans le m-uplet lu sur le i-ème ruban. En utilisant les états dans $(S \times \{1, \ldots, m\}^k \times (\Sigma^\square)^{mk}) + (S \times \{1, \ldots, m\}^k \times (\Sigma^\square)^{2mk})$, $\mathcal{M}'$ enregistre les symboles de Σ' écrits dans les cases adjacentes à celles pointées par les curseurs, soit $2k$ cases contenant chacune un m-uplet, ce qui requière 4 étapes à $\mathcal{M}'$. Cette information est alors suffisante pour déterminer les m étapes suivantes de $\mathcal{M}$: $\mathcal{M}'$ passe donc directement à la simulée de la configuration de $\mathcal{M}$ après m étapes, ce qui requière au plus 2 étapes à $\mathcal{M}'$ car en m étapes, $\mathcal{M}$ ne peut altérer dans chaque ruban que la case de $\mathcal{M}'$ pointée et au plus une case adjacente.

En tout, $\mathcal{M}'$ décide L en au plus $6 + 6\lceil f(n)/m \rceil + n + 2 \leqslant \epsilon f(n) + n + 2$ étapes. ∎

4.5 Définition (Machine avec entrée et sortie) Soit k un entier $\geqslant 2$. Une *machine de Turing à k rubans avec entrée et sortie* est une machine de Turing à k rubans (déterministe ou non) $\mathcal{M} = (\Sigma, S, T)$ telle que pour tout $((s, x_1, \ldots, x_k), (s', x'_1, \ldots, x'_k, d_1, \ldots, d_k)) \in T$ on ait $x_1 = x'_1$ (le premier ruban est un ruban de lecture : entrée) et $d_k \neq -1$ (le dernier ruban est un ruban d'écriture : sortie). Dans une *configuration finale*, le résultat est écrit sur le dernier ruban. Toutes les autres notions de la définition 2.24 sont identiques.

4.6 Définition (Espace) Soit une fonction $f : \mathbb{N} \to \mathbb{R}^+$. On dit qu'une machine de Turing $\mathcal{M}$ avec entrée et sortie *opère en espace* f (ou $f(n)$) si pour tout $v \in \Sigma^*$ tel que $\mathcal{M}$ mène en plusieurs étapes de la configuration initiale associée à v à la configuration $(s', v'_1, w'_1, \ldots, v'_k, w'_k)$, alors $\sum_{i=2}^{k-1} |v'_i| + |w'_i| \leqslant f(|v|)$. L'ensemble des langages décidables par une machine de Turing déterministe (resp. non-déterministe) avec entrée et sortie opérant en espace f est noté $\mathbf{Space}(f)$ (resp. $\mathbf{NSpace}(f)$).

L'intérêt de considérer des machines avec entrée et sortie dans la définition de l'espace de calcul en 4.6 (somme de 2 à $k-1$) est que certains problèmes requièrent nettement moins d'espace que la taille n de leur entrée, typiquement $\log n$.

4.7 Exemple (Palindromes) Soit un ensemble fini Σ. Un palindrome sur Σ est un mot $x_1 \cdots x_n \in \Sigma^*$ tel que pour tout $i > 0$, $x_i = x_{n+1-i}$. La machine déterministe $\mathcal{M} = (\Sigma, S, T)$, où $S = \{0, 1\} + \{s_x, x \in \Sigma\} + \{s'_x, x \in \Sigma\}$ et T est définie par la table suivante pour, disons $\Sigma = \{a, b\}$:

	a	b	$\square$
$\vdash$	$0, a, +1$	$0, b, +1$	$\top$
0	$s_a, \square, +1$	$s_b, \square, +1$	$\top$
s_a	$s_a, a, +1$	$s_a, b, +1$	$s'_a, \square, -1$
s_b	$s_b, a, +1$	$s_b, b, +1$	$s'_a, \square, -1$
s'_a	$1, \square, -1$	$\bot$	$\bot$
s'_b	$\bot$	$1, \square, -1$	$\bot$
1	$1, a, -1$	$1, b, -1$	$0, \square, +1$

décide l'ensemble L des palindromes. On peut évidemment considérer $\mathcal{M}$ comme une machine avec entrée et sortie (à 2 rubans, qui n'écrit rien sur le ruban de sortie), qui décide alors L en espace logarithmique (même en espace constant...), ce qui montre que $L \in \mathbf{Space}(\log n)$.

4.8 Théorème Si $L \in \mathbf{Space}(f(n))$, alors pour tout $\epsilon > 0$, on a $L \in \mathbf{Space}(\epsilon f(n) + 2)$.

Preuve — On procède comme pour le théorème 4.4. On écrit des m-uplets de lettres de $\Sigma^\square$, où $m = 1 + \lceil 1/\epsilon \rceil$. ∎

4.9 Lemme Si $f : \mathbb{N} \to \mathbb{R}^+$, alors $\mathsf{Time}(f) = \mathsf{coTime}(f)$ et $\mathsf{Space}(f) = \mathsf{coSpace}(f)$.

4.10 Théorème Soit $f : \mathbb{N} \to \mathbb{R}^+$. Alors :

$$\mathsf{Time}(f) \subseteq \mathsf{NTime}(f) \subseteq \mathsf{Space}(f) \subseteq \mathsf{NSpace}(f)$$
$$\mathsf{NSpace}(f) \subseteq \bigcup_{k>0} \mathsf{Time}(k^{\log n + f(n)}).$$

Preuve — Les inclusions $\mathsf{Time}(f) \subseteq \mathsf{NTime}(f)$ et $\mathsf{Space}(f) \subseteq \mathsf{NSpace}(f)$ sont triviales. On montre $\mathsf{NTime}(f) \subseteq \mathsf{Space}(f)$ en utilisant la simulation des machines de Turing non-déterministes par des machines déterministes introduite dans la preuve de la proposition 2.27. Soient donc L un langage décidé par une machine non-déterministe à k rubans $\mathcal{M} = (\Sigma, S, T)$ et d le maximum des cardinalités de $T(s, x_1, \ldots, x_k)$ pour $s \in S$ et $x_1, \ldots, x_k \in \Sigma^\square$. La machine $\mathcal{M}'$ de la proposition 2.27 génère les suites de choix non-déterministes pour $\mathcal{M}$, c'est-à-dire les entiers successifs $i = 1, 2, \ldots$ en unaire sous la forme 0^i et pour chaque i les suites $\langle n_1, \ldots, n_i \rangle$ d'entiers $\in \{0, \ldots, d-1\}$ de longueur i. Pour chaque suite $\langle n_1, \ldots, n_i \rangle$, $\mathcal{M}'$ simule $\mathcal{M}$ avec les choix non-déterministes correspondants ; comme $\mathcal{M}$ opère en temps f, l'espace utilisé pour une suite est aussi majoré par $f(n)$. Puis, si la simulation de $\mathcal{M}$ n'a pas encore abouti à un état final, $\mathcal{M}'$ efface le ruban qui sert à la simulation de $\mathcal{M}$ (c'est l'idée importante, qui permet d'économiser l'espace), et calcule la prochaine suite dans l'ordre lexicographique ou bien incrémente i si la suite considérée était $d-1 * \cdots * d-1$. Comme $\mathcal{M}$ s'arrête en au plus $f(n)$ étapes, $\mathcal{M}'$ n'écrit que des suites de longueur majorée par $i \cdot \log(d-1) \leqslant f(n) \cdot \log(d-1)$. Par conséquent, $L \in \mathsf{Space}(f)$.

Montrons que $\mathsf{NSpace}(f) \subseteq \bigcup_{k>0} \mathsf{Time}(k^{\log n + f(n)})$. Soit donc L un langage décidé par une machine non-déterministe à ℓ rubans avec entrée et sortie $\mathcal{M} = (\Sigma, S, T)$ opérant en espace $f(n)$. L'idée est de construire, sur l'entrée v de longueur n, le graphe G_n des configurations de $\mathcal{M}$, les sommets étant les configurations et les arcs étant les étapes d'exécution, et de réduire le problème L au problème d'accessibilité dans ce graphe. Puisque $\mathcal{M}$ est une machine avec entrée et sortie, le premier ruban est déterminé par l'entrée v et la position $i \leqslant |v| = n$ du curseur et le dernier ruban, qui n'est pas lu, est sans effet sur l'exécution. Donc une configuration $(s, v_1, w_1, \ldots, v_\ell, w_\ell)$ peut être représentée par le $(2\ell+2)$-uplet

$$\langle s, i, v_2, w_2, \ldots, v_{\ell-1}, w_{\ell-1} \rangle$$

où les mots v_p, w_p $(p = 2, \ldots, \ell-1)$ sont de longueur au plus $f(n)$. Si c est la cardinalité de Σ, la machine déterministe $\mathcal{M}'$ que nous définissons commence par construire le graphe G_n, dont le nombre de sommets est donc majoré par

$$|s| \cdot (n+1) \cdot c^{(2\ell-2)f(n)} \leqslant k_1^{\log n + f(n)}$$

pour un certain $k_1 > 0$ indépendant de n. Il est clair que $v \in L$ si, et seulement s'il existe un chemin du sommet $\langle s_0, 0, \varepsilon, v, \ldots, \varepsilon, \varepsilon \rangle$ vers un sommet $\langle s_1, i, v_2, w_2, \ldots, v_{\ell-1}, w_{\ell-1} \rangle$ représentant une configuration acceptante. Cette deuxième question est résoluble en temps polynomial en la taille du graphe (exemple 2.21), donc aussi en temps polynomial, même quadratique, en fonction du nombre de sommets, et on a le résultat souhaité. ∎

La méthode de réduction au problème `Acc` utilisée dans la preuve ci-dessus est appelée la méthode d'accessibilité. Pour simplifier la notation des classes de complexité usuelles, on convient que n est toujours la variable principale (dénotant la longueur de l'entrée) et on prend l'union sur les autres variables. Donc, si $f : \mathbb{N}^{p+1} \to \mathbb{N}$:

$$\mathsf{Time}(f(n, k_1, \ldots, k_p)) \quad \text{dénote} \quad \bigcup_{k_1>0, \ldots, k_p>0} \mathsf{Time}(f(n, k_1, \ldots, k_p))$$
$$\mathsf{Space}(f(n, k_1, \ldots, k_p)) \quad \text{dénote} \quad \bigcup_{k_1>0, \ldots, k_p>0} \mathsf{Space}(f(n, k_1, \ldots, k_p)),$$

et de même pour les classes non-déterministes. On considère les classes de complexité classiques suivantes, en temps :

$$\mathsf{P} = \mathsf{Time}(n^k) \qquad \mathsf{NP} = \mathsf{NTime}(n^k)$$
$$\mathsf{Exp} = \mathsf{Time}(2^{n^k}) \qquad \mathsf{NExp} = \mathsf{NTime}(2^{n^k})$$

et en espace :

$$\mathsf{L} = \mathsf{Space}(\log n) \qquad \mathsf{NL} = \mathsf{NSpace}(\log n)$$
$$\mathsf{PSpace} = \mathsf{Space}(n^k) \qquad \mathsf{NPSpace} = \mathsf{NSpace}(n^k)$$
$$\mathsf{ExpSpace} = \mathsf{Space}(2^{n^k}) \qquad \mathsf{NExpSpace} = \mathsf{NSpace}(2^{n^k}).$$

4.11 Corollaire $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSpace} \subseteq \mathsf{Exp} \subseteq \mathsf{NExp} \subseteq \mathsf{ExpSpace}$.

On a une caractérisation alternative de **NP** :

4.12 Définition (Vérifieur polynomial) Soit Σ un ensemble fini. Un *vérifieur polynomial d'aphabet* Σ est une relation binaire $R \subseteq \Sigma^* \times \Sigma^*$ telle que $R \in \mathsf{P}$.

4.13 Théorème Soient Σ un ensemble fini et $L \subseteq \Sigma^*$. Alors $L \in \mathsf{NP}$ si, et seulement s'il existe $k \in \mathbb{N}$ et un vérifieur polynomial R d'aphabet Σ tel que pour tout $v \in \Sigma^*$:

$$v \in L \text{ si, et seulement s'il existe } w \in \Sigma^* \text{ tel que } |w| \leqslant |v|^k \text{ et } R(v, w).$$

Preuve — S'il existe $k \in \mathbb{N}$ et un tel vérifieur polynomial R, alors L est décidé par la machine $\mathcal{M}$ qui, sur l'entrée v, commence par "deviner" un mot w de longueur au plus $|v|^k$: $\mathcal{M}$ écrit la borne $|v|^k$ sur un ruban auxiliaire, et construit le mot w en choisissant (non-déterminisme) à chaque étape l'une des possibilités suivantes : soit ajouter une lettre et décrémenter la borne, soit stopper la construction. Puis $\mathcal{M}$ utilise la machine pour le vérifieur polynomial R, et décide si $R(v, w)$.

Réciproquement, si $L \in \mathsf{NP}$, L est décidé par une machine non-déterministe $\mathcal{M}$ opérant en temps polynomial. Soit R la relation suivante : $R(v, w)$ si, et seulement si, w est le code d'un calcul acceptant de $\mathcal{M}$ sur l'entrée v. Comme $\mathcal{M}$ opère en temps polynomial, $R(v, w)$ implique $|w| \leqslant |v|^k$ pour une certaine constante k. Clairement, $R \in \mathsf{P}$. Enfin, puisque $\mathcal{M}$ décide L, on a bien $v \in L$ si, et seulement s'il existe $w \in \Sigma^*$ tel que $R(v, w)$. ∎

4.14 Théorème (Savitch, 1970) $\mathtt{Acc} \in \mathsf{Space}(\log^2 n)$.

Preuve — L'idée est de procéder par dichotomie. Soit G un graphe dont l'ensemble X des sommets a pour cardinalité n. Soit $P_G \subseteq X \times X \times \mathbb{N}$ la propriété suivante : $P_G(x, y, i)$ si, et seulement s'il existe un chemin dans G de x à y de longueur au plus 2^i. Clairement, $(G, x, y) \in \mathtt{Acc}$ si, et seulement si, $P_G(x, y, 1 + [\log n])$. On définit une machine déterministe avec entrée et sortie à 3 rubans $\mathcal{M}$ qui décide P_G. L'entrée consiste en la description de G suivie d'un triplet $(x, y, i) \in X \times X \times \mathbb{N}$ en représentation binaire.

- Si $i = 0$, $\mathcal{M}$ teste si $x = y$ ou si G contient un arc de x vers y.
- Si $i > 0$, $\mathcal{M}$ génère sur le troisième ruban les représentations binaires successives des sommets, c'est-à-dire les entiers de 0 à $n-1$. Pour chaque sommet z, $\mathcal{M}$ teste si $P_G(x, z, i-1)$ et $P_G(z, y, i-1)$ de la façon suivante : $\mathcal{M}$ ajoute le triplet $(x, z, i-1)$ sur le deuxième ruban, teste si $P_G(x, z, i-1)$; en cas de réponse négative, $\mathcal{M}$ l'efface et incrémente z ; en cas de réponse positive, $\mathcal{M}$ l'efface et le remplace par $(z, y, i-1)$ (z est lu sur le troisième ruban, y et i sont lus sur le triplet précédent (x, y, i) sur le deuxième ruban) et teste $P_G(z, y, i-1)$; de nouveau, en cas de réponse négative, $\mathcal{M}$ l'efface et incrémente z ; en cas de réponse positive, $\mathcal{M}$ vérifie qu'il s'agit du deuxième test en comparant le triplet courant (ici $(z, y, i-1)$) avec le triplet précédent (ici (x, y, i)) et c'est une réponse positive à $P_G(x, y, i)$. Cet algorithme est correct parce que $P_G(x, y, i)$ si, et seulement s'il existe un sommet z tel que $P_G(x, z, i-1)$ et $P_G(z, y, i-1)$.

Le deuxième ruban (vide au départ) contient donc au plus $1+\lceil \log n \rceil$ triplets dans $X \times X \times \mathbb{N}$, chacun de longueur au plus $3 \log n$, et le troisième ruban contient est de longueur au plus $\log n$. Donc $\mathcal{M}$ opère en temps $\mathcal{O}(\log^2 n)$. ∎

4.15 Corollaire Si $f(n) \geqslant \log n$, $\mathsf{NSpace}(f(n)) \subseteq \mathsf{Space}(f(n)^2)$. En particulier, $\mathsf{NL} \subseteq \mathsf{Space}(\log^2 n)$, $\mathsf{PSpace} = \mathsf{NPSpace} = \mathsf{coNPSpace}$ et $\mathsf{ExpSpace} = \mathsf{NExpSpace} = \mathsf{coNExpSpace}$.

Preuve — On reprend la preuve de $\mathsf{NSpace}(f) \subseteq \bigcup_{k>0} \mathsf{Time}(k^{\log n + f(n)})$ dans le théorème 4.10 et on teste l'accessibilité avec l'algorithme du théorème 4.14. Le graphe de configuration est de taille au plus $c^{\log n + f(n)}$ pour une certaine constante c, donc l'espace utilisé est majoré par :

$$\log^2 \left(c^{\log n + f(n)} \right) \leqslant \log^2 \left(c^{2f(n)} \right) = \mathcal{O}(f(n)^2).$$

Les cas particuliers sont conséquences du lemme 4.9. ∎

4.16 Théorème (Immerman-Szelepscényi, 1987) La fonction qui à un graphe G et un sommet x de G associe le nombre de sommets accessibles à partir de x est calculable en espace logarithmique non-déterministe.

Preuve — Soit n le nombre de sommets de G. La machine $\mathcal{M}$ calcule pour chaque $k = 0, \ldots, n-1$, la cardinalité s_k de l'ensemble S_k des sommets accessibles à partir de x par un chemin de longueur au plus k. Le résultat du calcul est s_{n-1}. Clairement, $S_0 = \{x\}$ et $s_0 = 1$. Pour calculer s_{k+1} en fonction de s_k (et G, x bien sûr), $\mathcal{M}$ initialise un compteur ℓ à 0, parcourt tous les sommets v de G et incrémente ℓ si $v \in S_{k+1}$. Pour décider si $v \in S_{k+1}$, $\mathcal{M}$ initialise un compteur m à 0 et :

1. parcourt de nouveau tous les sommets u de G ;
2. devine pour chaque u un chemin de longueur au plus k de x à u (c'est la partie non-déterministe) ;
3. en cas d'échec, la branche est rejetée ;
4. en cas de succès, incrémente m (le nombre de sommets de S_k trouvés dans cette branche) ; si G ne contient pas d'arc de u à v, la branche est rejetée.

À la fin de la boucle, dans une branche qui n'est pas encore rejetée, si $m < s_k$, c'est-à-dire si la branche n'a pas essayé tous les sommets de $u \in S_k$, la branche est rejetée, sinon $\mathcal{M}$ accepte. Le résultat d'une branche à la question $v \in S_{k+1}$ est donc une acceptation si, et seulement si, $v \in S_{k+1}$.

$\mathcal{M}$ utilise les 6 variables auxiliaires k, s_k, ℓ, v, m, u, et un compteur $p \in \{0, \ldots, k-1\}$, et deux sommets u', u'' pour deviner le chemin de longueur au plus k de x à u. Ces 9 variables sont majorées par n, donc de longueur majorée par $\log n$. ∎

La routine non-déterministe pour décider si $v \in S_{k+1}$ dans la preuve du théorème 4.16 donne immédiatement :

4.17 Corollaire $\mathtt{Acc} \in \mathsf{NL}$.

4.18 Corollaire Si $f(n) \geqslant \log n$, $\mathsf{NSpace}(f) = \mathsf{coNSpace}(f)$. En particulier, $\mathsf{NL} = \mathsf{coNL}$.

Preuve — Si L est un langage décidé en espace f par une machine non-déterministe $\mathcal{M}$, soit $\mathcal{M}'$ une machine non-déterministe qui applique l'algorithme du corollaire 4.17 sur le graphe de configuration de $\mathcal{M}$ (de taille majorée par $c^{f(n)}$ pour une certaine constante c), et inverse les résultats : $\mathcal{M}'$ rejette si une configuration acceptante de $\mathcal{M}$ est accessible, accepte sinon. Clairement, $\mathcal{M}'$ décide $\Sigma^* \setminus L$ en espace f. ∎

4.19 Définition (Fonction de complexité) Une *fonction de complexité* est une fonction $f : \mathbb{N} \to \mathbb{N}$ croissante au sens large telle qu'il existe une machine de Turing (d'alphabet contenant 1) avec entrée et sortie, qui sur l'entrée v calcule $1^{f(|v|)}$ en espace $\mathcal{O}(f(|v|))$ et en temps $\mathcal{O}(|v| + f(|v|))$.

L'intérêt de se limiter à des fonctions de complexité est illustré par le résultat suivant, très contre-intuitif.

4.20 Théorème (Trou noir, Trakhtenbrot, 1964) Il existe une fonction $f : \mathbb{N} \to \mathbb{N}$ récursive telle que $\mathsf{Time}(f(n)) = \mathsf{Time}(2^{f(n)})$.

Preuve — On définit une fonction f telle que pour tout entier n, toute machine $\mathcal{M}$ et toute entrée pour $\mathcal{M}$ de longueur n, soit $\mathcal{M}$ s'arrête en au plus $f(n)$ étapes, soit $\mathcal{M}$ s'arrête en plus de $2^{f(n)}$ étapes, soit $\mathcal{M}$ ne s'arrête pas.

Soit $\mathcal{M}_i, i \geqslant 1$, une énumération récursive des machines de Turing, par exemple ordonnées lexicographiquement par leur code (pour un certain codage $\mathcal{M} \mapsto \mathcal{M}^\bullet$). Étant donnés des entiers n, k, i et une entrée v pour $\mathcal{M}_i$ de longueur n, on appelle $P(n, k, i, v)$ la propriété suivante : sur l'entrée v, soit $\mathcal{M}_i$ s'arrête en au plus k étapes, soit $\mathcal{M}_i$ s'arrête en plus de 2^k étapes, soit $\mathcal{M}_i$ ne s'arrête pas. On appelle $P(n, k)$ la conjonction des $P(n, k, i, v)$ pour $i \leqslant n$ et v une entrée quelconque pour $\mathcal{M}_i$ de longueur n : $P(n, k)$ est clairement décidable.

Soit $n \in \mathbb{N}$. Posons $k_1 = n$ et pour $j \geqslant 1$, $k_{j+1} = 1 + 2^{k_j}$. Soit $N(n)$ le nombre d'entrées de longueur n des n premières machines $\mathcal{M}_1, \ldots, \mathcal{M}_n$. Pour chaque entrée v pour $\mathcal{M}_i$ de longueur n, $i \leqslant n$, il existe au plus un entier j tel que $P(n, k_j, i, v)$ soit fausse, par définition des k_j. Donc au plus $N(n)$ formules parmi les $P(n, k_1), \ldots, P(n, k_{N(n)+1})$ sont fausses, et il existe $\ell \leqslant N(n) + 1$ tel que $P(n, k_\ell)$ soit vraie : soit $f(n)$ le premier tel k_ℓ. La fonction f est clairement récursive.

Si maintenant $L \in \mathsf{Time}(2^{f(n)})$, L est décidé par une machine, disons $\mathcal{M}_i$, en temps au plus $2^{f(n)}$. Pour toute entrée de longueur $n \geqslant i$ pour $\mathcal{M}_i$, on a $P(n, f(n))$, donc $\mathcal{M}_i$ ne s'arrête pas en un temps t tel que $f(n) < t \leqslant 2^{f(n)}$. Or $\mathcal{M}_i$ s'arrête en au plus $2^{f(n)}$ étapes, donc $\mathcal{M}_i$ s'arrête en au plus $f(n)$ étapes. Pour décider les entrées de longueur $n < i$, on ajoute un nombre fini d'états, ce qui permet de conclure que $L \in \mathsf{Time}(f(n))$. ∎

On se limite donc à classes définies à partir de fonctions de complexité. Le lemme suivant, dont la preuve est laissée en exercice, implique que les classes considérées jusqu'à présent sont bien définies à partir de fonctions de complexité.

4.21 Lemme L'ensemble des fonctions de complexité est clos par composition ($f, g \mapsto f \circ g$), addition ($f, g \mapsto f+g$), multiplication ($f, g \mapsto f \times g$) et exponentiation ($f \mapsto (n \mapsto 2^{f(n)})$). Les fonctions constantes, l'identité $n \mapsto n$, la fonction $n \mapsto \lfloor \log n \rfloor$, les polynômes, la fonction $n \mapsto 2^{n^k}$ sont des fonctions de complexité.

4.22 Définition (Problème de l'arrêt en temps limité) Soit f une fonction de complexité telle que pour tout n, $f(n) \geqslant n$. On appelle *problème de l'arrêt en temps* f le langage suivant :

$$\mathtt{Halt}(f) = \{\mathcal{M}^\bullet \# w^\bullet, \mathcal{M}(w) \downarrow_o \text{ en au plus } f(|w|) \text{ étapes}\}.$$

4.23 Lemme Si f est une fonction de complexité telle que pour tout n, $f(n) \geqslant n$, on a $\mathtt{Halt}(f) \in \mathsf{Time}(f(n)^3)$ et $\mathtt{Halt}(f) \notin \mathsf{Time}(f(\lfloor n/2 \rfloor))$.

Preuve — On définit une machine déterministe U_f à 4 rubans qui décide $\mathtt{Halt}(f)$ en temps $\mathcal{O}(f(n)^3)$. Comme f est une fonction de complexité, U_f écrit sur le quatrième ruban le mot $1^{f(|w|)}$, qui va servir de compte à rebour. U_f vérifie que l'entrée est bien le code d'une machine $\mathcal{M}$, et copie $\mathcal{M}^\bullet$ sur le troisième ruban et le code de l'état initial sur le deuxième, ce qui dure $\mathcal{O}(f(|w|)+n) = \mathcal{O}(f(n))$ étapes.

Ensuite, U_f simule $\mathcal{M}$ comme la machine universelle U de l'exemple 2.32, mais décrémente le compte à rebour à chaque étape de $\mathcal{M}$; U_f rejette si le compte à rebour est à 0. Chaque étape de $\mathcal{M}$ est simulée en un temps $\mathcal{O}(k_{\mathcal{M}}^2 \ell_{\mathcal{M}} f(|w|))$, où $k_{\mathcal{M}}$ est le nombre de rubans de $\mathcal{M}$ et $\ell_{\mathcal{M}}$ est la longueur de chaque description de symbole de $\mathcal{M}$. Comme ces deux constantes sont $\leqslant \log n$ et que $\log^3 n \leqslant n \leqslant f(n)$ asymptotiquement, le temps de simulation d'une étape de $\mathcal{M}$ est $\mathcal{O}(f(n)^2)$. Le compte à rebour limitant le nombre d'étapes à $f(n)$, U_f opère en temps $\mathcal{O}(f(n)^3)$, on en conclut que $\mathtt{Halt}(f) \in \mathsf{Time}(f(n)^3)$.

Pour montrer que $\mathtt{Halt}(f) \notin \mathsf{Time}(f([n/2]))$, on procède par l'absurde en supposant qu'il existe une machine de Turing $\mathcal{M}_{\mathtt{Halt}(f)}$ qui décide $\mathtt{Halt}(f)$ en temps $f([n/2])$. On définit alors par diagonalisation la machine D_f suivante :

$$D_f(\mathcal{M}^\bullet) = \text{ si } \mathcal{M}_{\mathtt{Halt}(f)}(\mathcal{M}^\bullet \# \mathcal{M}^\bullet) \downarrow_o \text{ alors } \downarrow_n \text{ sinon } \downarrow_o,$$

et $D_f(v) \downarrow_n$ si v n'est pas de la forme $\mathcal{M}^\bullet$. Sur l'entrée $\mathcal{M}^\bullet$, D_f opère en temps $f([(2n+1)/2]) = f(n)$. Si D_f accepte $D_f^\bullet$, c'est que $\mathcal{M}_{\mathtt{Halt}(f)}$ rejette $D_f^\bullet \# D_f^\bullet$, et donc $D_f^\bullet \# D_f^\bullet \notin \mathtt{Halt}(f)$, ce qui signifie que D_f n'accepte pas $D_f^\bullet$ en temps $f(n)$. Comme D_f converge en temps $f(n)$, D_f rejette $D_f^\bullet$. Si au contraire D_f rejette $D_f^\bullet$, on en déduit de même que D_f accepte $D_f^\bullet$. Dans les deux cas, on a une contradiction. ∎

On en déduit :

4.24 Théorème (Hiérarchie pour le temps déterministe) Si f est une fonction de complexité telle que pour tout n, $f(n) \geqslant n$, on a $\mathsf{Time}(f(n)) \subsetneq \mathsf{Time}(f(2n+1)^3)$.

4.25 Corollaire $\mathsf{P} \subsetneq \mathsf{Exp}$.

Preuve — Clairement, $\mathsf{P} \subseteq \mathsf{Time}(2^n)$. Or, par le théorème 4.24, on a $\mathsf{Time}(2^n) \subsetneq \mathsf{Time}(2^{3(2n+1)}) \subseteq \mathsf{Time}(2^{n^2}) \subseteq \mathsf{Exp}$. ∎

4.26 Théorème (Hiérarchie pour l'espace déterministe) Si f est une fonction de complexité, alors on a $\mathsf{Space}(f(n)) \subsetneq \mathsf{Space}(f(2n)\log n)$.

Preuve — On définit un *problème de l'arrêt en espace* f :

$$\mathtt{Halt}_{sp}(f) = \{\mathcal{M}^\bullet \# w^\bullet, \mathcal{M}(w) \downarrow_o \text{ en espace au plus } f(|w|)\}.$$

On montre que $\mathtt{Halt}_{sp}(f) \notin \mathsf{Space}(f([n/2]))$ comme dans le lemme 4.23. On montre que $\mathtt{Halt}_{sp}(f) \in \mathsf{Space}(f(n)\log n)$ en utilisant une machine universelle avec un compteur pour l'espace. Le facteur $\log n$ provient du codage des mots. ∎

4.27 Corollaire $\mathsf{NL} \subsetneq \mathsf{PSpace} \subsetneq \mathsf{ExpSpace}$.

Preuve — Par le corollaire 4.15, $\mathsf{NL} \subseteq \mathsf{Space}(\log^2 n)$, et par le théorème 4.26, $\mathsf{Space}(\log^2 n) \subsetneq \mathsf{Space}(\log^3 n) \subsetneq \cdots \subseteq \mathsf{Space}(n) \subsetneq \mathsf{PSpace}$. La deuxième inclusion est triviale. ∎

Il existe aussi des théorèmes de hiérarchie pour le temps et l'espace non-déterministes.

4.3 Réductions et complétude

4.28 Définition (Réduction) Soient Σ_1, Σ_2 deux alphabets et $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$ deux langages. Une *l-réduction (resp. p-réduction, r-réduction) de* L_1 *vers* L_2 est une fonction $R : \Sigma_1^* \to \Sigma_2^*$ calculable en espace logarithmique (resp. calculable en temps polynomial, récursive) et telle que $v \in L_1$ si, et seulement si, $R(v) \in L_2$. Pour $x = l,p,r$, on dit qu'un langage L_1 est x-*réductible* à un langage L_2 s'il existe une x-réduction de L_1 vers L_2, et on le note $L_1 \leqslant_x L_2$.

4.29 Définition (Clôture par réduction) Soient X une classe de complexité et $x = l,p$ ou r. On dit que X est *close par* x*-réduction* si pour tous $L \in X$ et $L' \leqslant_x L$, on a $L' \in X$.

Chacune des classes de complexité considérées X a une classe naturellement associée $x = l,p$ ou r, et qu'on appellera la classe de réduction standard de X. Par exemple :

l	pour	**NL** et **P**,
p	pour	**NP**, **coNP**, **PSpace**, **Exp**, **NExp**, **coNExp** et **ExpSpace**,
r	pour	**RE** et **coRE**.

La preuve de la proposition suivante est laissée en exercice.

4.30 Proposition **L**, **NL**, **P**, **NP**, **coNP**, **PSpace**, **Exp**, **NExp**, **coNExp**, **ExpSpace**, **RE**, **coRE** sont closes par réduction.

4.31 Définition (Complétude) Soient X une classe de complexité, x sa classe de réduction standard et L un langage. On dit que L est X-*difficile* si pour tout $L' \in X$, $L' \leqslant_x L$. On dit que L est X-*complet* si $L \in X$ et L est X-difficile.

4.32 Théorème `Halt` est **RE**-complet.

Preuve — D'après le théorème 2.33, `Halt` $\in$ **RE**. Fixons un codage des machines de Turing : si $L \in$ **RE**, soit $\mathcal{M}$ une machine qui l'accepte ; on a $w \in L$ si, et seulement si, $\mathcal{M}^\bullet \# w^\bullet \in$ `Halt`, donc $L \leqslant_r$ `Halt`. Par conséquent, `Halt` est **RE**-complet. ∎

4.33 Définition (Circuit booléen) Soit V un ensemble dénombrable (de variables). Un *circuit booléen* C est un triplet $(|C|, \leqslant, e)$, où $(|C|, \leqslant)$ est un ordre partiel fini avec minorant (appelé la *sortie* de C) tel que $|C|$ soit non vide et que chaque élément ait au plus 2 prédécesseurs, et e est une application (appelée *étiquetage*) de $|C|$ dans $V + \{\neg, \wedge, \vee, v, f\}$ telle que pour tout $s \in |C|$:
- s a 0 prédécesseur si, et seulement si, $e(s) \in V + \{v, f\}$ (un tel s est appelé une *entrée* de C),
- s a 1 prédécesseur si, et seulement si, $e(s) = \neg$,
- s a 2 prédécesseurs si, et seulement si, $e(s) = \wedge$ ou $\vee$.

À un circuit booléen C on associe une formule propositionnelle ϕ_C définie par induction sur la cardinalité de $|C|$:
- si $|C| = \{s\}$ est un singleton, alors $e(s) \in V + \{v, f\}$ et $\phi_C = e(s)$,
- si la sortie de C est étiquetée par $\neg$ et C' est le circuit booléen obtenu en enlevant la sortie de C, alors $\phi_C = \neg\phi_{C'}$,
- si la sortie de C est étiquetée par $\wedge$ (resp. $\vee$) et C_1, C_2 sont les circuits booléens obtenus en enlevant la sortie de C, alors $\phi_C = \phi_{C_1} \wedge \phi_{C_2}$ (resp. $\phi_{C_1} \vee \phi_{C_2}$).

On dit que C *calcule* ϕ_C.

En particulier, tout sommet d'un circuit booléen sauf sa sortie a au moins un successeur. En retirant de la définition ci-dessus l'existence d'un minorant, on obtient des circuits calculant des fonctions booléennes de $\{v, f\}^n$ dans $\{v, f\}^p$ (au lieu de propositions, c'est-à-dire de fonctions booléennes de $\{v, f\}^n$ dans $\{v, f\}$). Observons qu'une formule propositionnelle est un cas particulier de circuit booléen, dans lequel tout sommet sauf la sortie a exactement un successeur.

4.34 Définition `CircuitSat` est le problème suivant : étant donné un circuit booléen C, existe-t-il une valuation qui satisfasse ϕ_C ? `CircuitValue` est le problème suivant : étant donné un circuit booléen C dont toutes les entrées sont v ou f, ϕ_C est-elle vraie ?

4.35 Théorème `CircuitValue` est **P**-complet.

Preuve — `CircuitValue` est dans **P** : le circuit C étant donné par exemple sous la forme de son graphe à n sommets et de son étiquetage (liste d'étiquettes), on calcule la valeur de ϕ_C en calculant les valeurs de tous les sommets (on cherche les sommets dont on connaît les valeurs des prédécesseurs).

Pour montrer qu'il est **P**-complet, on définit pour tout langage $L \in \mathbf{P}$ une réduction R de L à `CircuitValue`. On sait que L est décidé par une machine déterministe $\mathcal{M} = (\Sigma, S, T)$ à un ruban en temps au plus n^k pour une certaine constante k, et on admet que $\mathcal{M}$ peut être supposée s'arrêter sur la case numérotée 0, ce qui se démontre facilement. La *table de calcul* de $\mathcal{M}$ sur l'entrée ν de longueur n, notée $t(\mathcal{M}, \nu)$, est la matrice $(n^k + 1) \times 2(n^k + 1)$ dont la ligne i représente la configuration (s, f, p) qui découle de la coniguration initiale en i étapes :

$$t(\mathcal{M}, \nu)_{i,j} = \begin{cases} f(j) & \text{si } j \neq p \\ (f(p), s) & \text{si } j = p. \end{cases}$$

On observe que $t(\mathcal{M}, \nu)_{i,j}$, $i \geqslant 1$, $-n^k \leqslant j \leqslant n^k$, est une fonction F des 3 valeurs voisines de $t(\mathcal{M}, \nu)$ sur la ligne précédente, soit :

$$t(\mathcal{M}, \nu)_{i-1,j-1},\ t(\mathcal{M}, \nu)_{i-1,j} \text{ et } t(\mathcal{M}, \nu)_{i-1,j+1},$$

F ne dépendant que de la machine $\mathcal{M}$. En codant les entrées de $t(\mathcal{M}, \nu)$ par des suites de 0 et de 1 de longueur m, où m est une constante qui ne dépend que de $\mathcal{M}$, et en identifiant 0 avec f et 1 avec ν, on peut donc définir une famille, notée C, constituée de m circuits à $3m$ entrées qui calculent F.

On peut alors construire un circuit $R(\nu)$ constitué de $n^k \cdot 2(n^k - 1)$ copies de C, une pour chaque ligne $i \geqslant 1$ et chaque colonne j telle que $-n^k \leqslant j \leqslant n^k$. Si la copie numéro (i, j) est notée $C_{i,j}$, on identifie les entrées de $C_{i,j}$ avec les sorties de $C_{i-1,j-1}$, $C_{i-1,j}$ et $C_{i-1,j+1}$. Les entrées de $R(\nu)$ sont les valeurs de la première ligne et les codes des $\square$ situés sur les première et dernière colonnes de $t(\mathcal{M}, \nu)$. On ajoute un sous-circuit qui calcule 0 ou 1 suivant que la valeur de $C_{n^k,0}$ est le code d'une configuration rejetante ou acceptante. Ce résultat, 0 ou 1, est la sortie de $R(\nu)$.

On vérifie aisément que $R(\nu)$ a pour valeur 1 si, et seulement si, $\nu \in L$, et que la construction de $R(\nu)$ se fait en espace logarithmique. ∎

4.36 Théorème (Cook, 1971) `CircuitSat` est **NP**-complet.

Preuve — Il est facile de définir un vérifieur polynomial pour `CircuitSat` : étant donné un circuit C avec k variables, on devine une valuation à k variables et on vérifie en temps polynomial si elle satisfait ou non la proposition ϕ_C, comme dans l'algorithme pour `CircuitValue` (théorème 4.35). Donc d'après le théorème 4.13, `CircuitSat` est dans **NP**.

Pour montrer qu'il est **NP**-complet, on définit pour tout langage $L \in \mathbf{NP}$ une réduction R de L à `CircuitSat`. On sait que L est décidé par une machine non-déterministe $\mathcal{M} = (\Sigma, S, T)$ à un ruban en temps au plus n^k pour une certaine constante k. On peut supposer sans perte de généralité que $\mathcal{M}$ toujours a exactement 2 choix, c'est-à-dire que pour tout $(s, x) \in (S \cup \{\vdash\}) \times (\Sigma \cup \{\square\})$, la cardinalité de $T(s, x)$ est 2. Comme dans la preuve du théorème 4.35, étant données une entrée ν de longueur n et en plus une suite de choix $c \in \{0,1\}^{n^k}$, on peut définir la table de calcul $t(\mathcal{M}, \nu, c)$ de $\mathcal{M}$ sur l'entrée ν avec les choix $c_1, \ldots, c_{n^k}$ qui est une matrice $(n^k + 1) \times 2(n^k + 1)$. Pour $i \geqslant 1$, $-n^k \leqslant j \leqslant n^k$, $t(\mathcal{M}, \nu, c)_{i,j}$ est une fonction $t(\mathcal{M}, \nu, c)_{i-1,j-1}$, $t(\mathcal{M}, \nu, c)_{i-1,j}$, $t(\mathcal{M}, \nu, c)_{i-1,j+1}$ et de c_{i-1}. On construit alors le circuit $R(\nu)$ en espace logarithmique comme dans la preuve du théorème 4.35, en combinant des circuits (à $3m + 1$ entrées cette fois).

La proposition associée $\phi_{R(\nu)}$ est satisfiable si, et seulement s'il existe une suite de choix $c \in \{0,1\}^{n^k}$ telle que $\phi_{R(\nu)}(c)$ a pour valeur 1 (on identifie proposition à n^k variables et fonction de $\{0,1\}^{n^k}$ dans $\{0,1\}$) si, et seulement si, $\nu \in L$. ∎

4.37 Définition `Sat` est le problème suivant : étant donné une proposition ϕ, existe-t-il une valuation qui la satisfasse ? `2-Sat` (resp. `3-Sat`) est la restriction de `Sat` aux propositions sous forme normale

conjonctive $\phi_1 \wedge \cdots \wedge \phi_k$ dont toutes les disjonctions ϕ_i (appelées clauses) ont 2 (resp. 3) littéraux (variables ou négations de variable).

Dans la définition ci-dessus, la restriction à des propositions sous forme normale conjonctive n'enlève pas de généralité d'après le corollaire 3.6. C'est la condition sur la taille des clauses qui est spécifique.

4.38 Corollaire `Sat` et `3-Sat` sont **NP**-complets.

Preuve — `CircuitSat`, qui est dans **NP** d'après le théorème 4.36, étant plus général que `Sat` et `3-Sat`, `Sat` et `3-Sat` sont clairement dans **NP**. Pour conclure, on montre que `CircuitSat` se réduit à `3-Sat` (donc à `Sat` qui est plus général que `3-Sat`).

Étant donné un circuit $C = (|C|, \leqslant, e)$, on construit en espace logarithmique une proposition sous forme normale conjonctive dont les clauses ont au plus 3 littéraux (cela suffit, car on peut alors remplacer par exemple $x \vee y$ par $x \vee y \vee f \ldots$), et qui est satisfiable si, et seulement si, C l'est. La proposition a pour variables celles étiquetant des entrées de C, ainsi que les sommets de C, et est la conjonction des formules suivantes ψ_s où s varie parmi les sommets de C :

$$\begin{cases} (\neg s \vee x) \wedge (s \vee \neg x) & \text{si } e(s) = x, \text{ une variable} \\ s & \text{si } e(s) = v \\ \neg s & \text{si } e(s) = f \\ (s \vee s_1) \wedge (\neg s \vee \neg s_1) & \text{si } e(s) = \neg \text{ et } s_1 \text{ est le prédécesseur de } s \\ (\neg s \vee s_1) \wedge (\neg s \vee s_2) \wedge (\neg s_1 \vee \neg s_2 \vee s) & \text{si } e(s) = \wedge \text{ et } s_1, s_2 \text{ sont les prédécesseurs de } s \\ (\neg s_1 \vee s) \wedge (\neg s_2 \vee s) \wedge (s_1 \vee s_2 \vee \neg s) & \text{si } e(s) = \vee \text{ et } s_1, s_2 \text{ sont les prédécesseurs de } s \\ s & \text{si } s \text{ est la sortie.} \end{cases}$$

∎

4.39 Définition Étant donnés un graphe non-dirigé G, une *clique* (resp. une *anticlique*) est un sous-ensemble X de sommets tels qu'il existe une arête (resp. il n'existe pas d'arête) de G entre 2 sommets quelconques de X. `Clique` (resp. `Anticlique`) est le problème suivant : étant donnés un graphe non-dirigé G et un entier n, G contient-il une clique (resp. anticlique) de cardinalité n ? `3-Color` est le problème suivant : étant donné un graphe non-dirigé G, existe-t-il une application de l'ensemble des sommets de G dans $\{1, 2, 3\}$ telle que deux sommets adjacents quelconques aient des images différentes ?

4.40 Corollaire `Clique` et `Anticlique` sont **NP**-complets.

Preuve — Il est clair que `Clique` et `Anticlique` ont des vérifieurs polynomiaux, et sont donc dans **NP**. De plus, `Clique` et `Anticlique` se réduisent l'un à l'autre en espace logarithmique.

Pour montrer que `Anticlique` est **NP**-difficile, on réduit `3-Sat` à `Anticlique`. À une conjonction $\psi = \psi_1 \wedge \cdots \wedge \psi_m$ de m clauses $\psi_i = x_i^1 \vee x_i^2 \vee x_i^3$, on associe le graphe G ayant $3m$ sommets s_i^1, s_i^2, s_i^3, $i = 1, \ldots, m$, et les arêtes suivantes : une arête entre s_i^1 et s_i^2, une arête entre s_i^2 et s_i^3, une arête entre s_i^1 et s_i^3 pour tout $i = 1, \ldots, m$; une arête entre s_i^α et s_j^β lorsque $i \neq j$ et $x_i^\alpha = \neg x_j^\beta$. Il est clair que G a une anticlique de cardinalité m si, et seulement si, ψ est satisfiable. ∎

4.41 Exercice Montrer que `3-Color` est **NP**-complet.

On trouvera de nombreux autres problèmes **NP**-complets dans [4].

4.42 Théorème `Acc` est **NL**-complet.

Preuve — D'après le corollaire 4.17, Acc est dans **NL**. Pour montrer qu'il est **NL**-complet, on définit pour tout langage $L \in \mathbf{NL}$ une réduction R de L à Acc. Soit $\mathcal{M}$ une machine non-déterministe qui décide L en espace logarithmique. Comme dans la preuve de l'inclusion $\mathbf{NSpace}(f) \subseteq \bigcup_{k>0} \mathbf{Time}(k^{\log n + f(n)})$ (théorème 4.10), on construit à partir de l'entrée v le graphe G_v des configurations de $\mathcal{M}$ à partir de v ; on lui ajoute un sommet t et des arcs vers t depuis chaque configuration terminale. Si s est le sommet de G_v associé à la configuration initiale, on a bien $v \in L$ si, et seulement s'il existe un chemin de s vers t dans G_v. ∎

4.43 Corollaire 2-Sat est **NL**-complet.

Preuve — 2-Sat est dans **NL** : si ϕ, une proposition sous forme normale conjonctive dont toutes les clauses ont 2 littéraux, a pour variables $x_1, \ldots, x_n$, on construit le graphe dont les sommets sont les littéraux $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$, et comportant, pour tous littéraux x et y, un arc de x vers y si, et seulement si, une clause de ϕ est équivalente à $x \Rightarrow y$ (par exemple de $\neg x_i$ vers $\neg x_j$ si une clause de ϕ est $\neg x_j \vee x_i$, etc). Clairement, ϕ n'est pas satisfiable si, et seulement s'il existe i et deux chemins, l'un de x_i vers $\neg x_i$ et l'autre de $\neg x_i$ vers x_i, ce qui se teste en essayant (partie non-déterministe) un numéro de sommet, codé par exemple en binaire, donc de taille logarithmique. Par conséquent, 2-Sat est dans $\mathbf{coNL} = \mathbf{NL}$ (corollaire 4.18).

D'après le théorème 4.42, Acc est **NL**-complet. Or d'après le corollaire 4.18, $\mathbf{NL} = \mathbf{coNL}$, donc le problème complémentaire Inacc (pas de chemin de s vers t) est aussi **NL**-complet. On réduit Inacc à 2-Sat. Soit G un graphe. On peut le supposer acyclique (sinon, on le modifie en prenant un cycle quelconque, en enlevant un arc quelconque de ce cycle, et on recommence jusqu'à obtenir un graphe G' acyclique ; alors entre deux sommets s et t, il existe un chemin dans G si, et seulement s'il en existe un dans G'). On associe à G, s, t la conjonction ψ des propositions s, $\neg t$, et $\neg x \vee y$ pour tout couple de sommets (x, y) tel que G contienne un arc de x vers y. Il est clair que ψ est satisfiable si, et seulement s'il n'existe pas de chemin de s vers t dans G. ∎

La figure 2 présente les relations entre les classes de complexité considérées, avec quelques problèmes complets. **Elem** est la classe des langages *élémentaires* :

$$\mathbf{Elem} = \bigcup_{p \geqslant 0} \mathbf{Time}\,(f_p(n))\,,$$

où $f_0(n) = n$ et pour tout $p \geqslant 0$, $f_{p+1}(n) = 2^{f_p(n)}$. Pour compléter cette courte introduction à la théorie de la complexité algorithmique, on pourra lire notamment [5, 6].

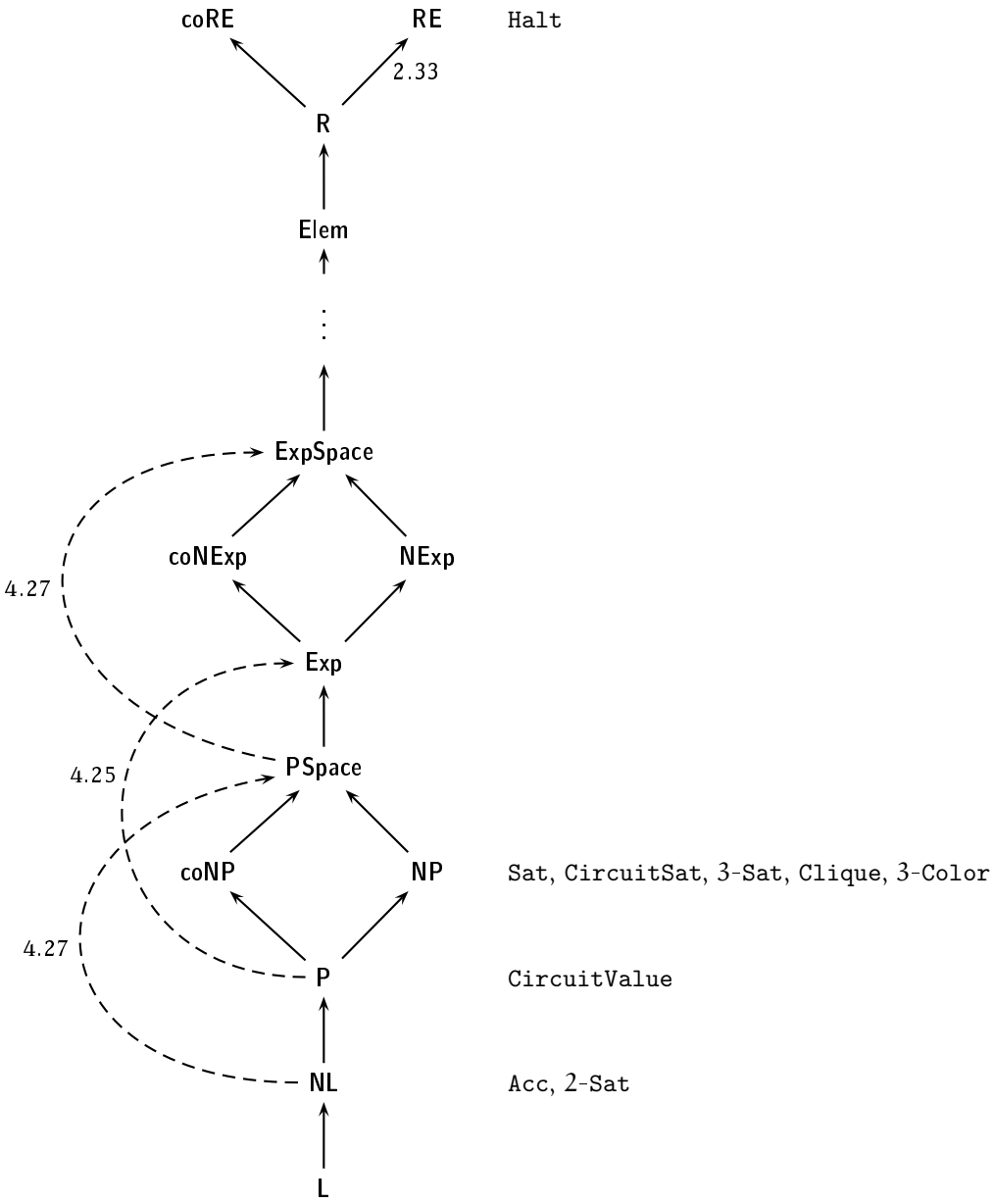


Fig. 2 – Panorama des principales classes de complexité, avec quelques problèmes complets. Les références indiquent les résultats d'inclusion stricte mentionnés dans le texte.

Références

[1] R. Cori and D. Lascar. *Logique mathématique*. Axiomes. Masson, 1993.

[2] R. David, K. Nour, and C. Raffalli. *Introduction à la logique*. Dunod, 2001.

[3] P. Dehornoy. *Mathématiques de l'informatique*. Dunod, 2000.

[4] M. R. Garey and D. S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. Freeman and Co, 1979.

[5] C. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[6] M. Sipser. *Introduction to the theory of computation*. PWS, 1997.

Table des matières