Calculabilité et incomplétude - Notes de cours

Arnaud Durand, Paul Rozière
Paris7 – M2 LMFI

2 décembre 2021
(v. provisoire — 19: 35)

Chapitre 1

Modèles de calcul

AVERTISSEMENT : version préliminaire des notes du cours fondamental 2, susceptible de corrections et d'ajouts.

L'objet de cette partie est de préciser les notions d'effectivité en mathématiques en proposant plusieurs modélisations de la notion de fonction et d'ensemble calculables. On va introduire essentiellement trois modèles différents, à savoir :

— la définition de Kleene des fonctions calculables : les fonctions μ-récursives ;
— les fonctions calculables par machines à registres ;
— les fonctions calculables par machines de Turing.

On montre ensuite que ces trois approches de la calculabilité (il en existe bien d'autres) qui mettent en jeux, a priori, des notions de ressources différentes mènent à des notions équivalentes de fonction calculable. Les notes se poursuivent ensuite par une introduction aux résultats de base de la théorie de la calculabilité.

La première approche présentée est celle des fonctions μ-récursives. Les objets de référence sont les fonctions sur les entiers naturels. On considère dans ce cadre qu'une fonction est calculable si elle appartient à un certain ensemble de base (fonctions constantes, successeur, ...) ou si elle peut être obtenue à partir de l'ensemble de fonctions de bases par composition, définition par récurrence ou d'autres schémas que l'on détaillera. Ces fonctions sont intuitivement calculables, mais, contrairement aux définitions que l'on verra ensuite, le calcul reste implicite. Cette approche élégante, fournit un intermédiaire commode pour montrer que les diverses notions de fonctions calculables sont équivalentes. Mais la formalisation du calcul s'avère très vite indispensable pour prolonger l'étude. Il s'avère que n'importe quel modèle de calcul convient et conduit aux mêmes résultats, pourvu qu'il soit suffisamment riche.

Notation. Dans la suite, un vecteur de paramètres sera alternativement désigné par $(x_1, ..., x_p)$ ou par $\overline{x}$ suivant les situations. On parle d'*arité* pour désigner le nombre de paramètres des fonctions et relations.

1.1 Fonctions récursives primitives

Les fonctions récursives primitives sont essentiellement les fonctions qui se calculent par récurrence sur un argument entier, et les composées de celles-ci. Elles ont été introduites dans les années 1920, et les mathématiciens se sont rendu compte assez vite qu'elles ne pouvaient représenter toutes les fonctions calculables (Ackermann, 1926), même si « beaucoup » de fonctions calculables « usuelles » sur les entiers sont récursives primitives.

Définition 1.1.1 L'ensemble des *fonctions récursives primitives* est le plus petit sous-ensemble de l'ensemble des fonctions à plusieurs arguments entiers à valeurs entières ($\bigcup_{p\in\mathbb{N}^*} \mathbb{N}^{\mathbb{N}^p}$) qui satisfait les conditions suivantes :

i. il contient la fonction *nulle* $\lambda x.0 : \mathbb{N} \to \mathbb{N}$, la fonction *successeur* $\mathrm{s} : \mathbb{N} \to \mathbb{N}$ et les *projections* $p_i^k : \mathbb{N}^k \to \mathbb{N}$ $(1 \leq i \leq k)$, définies par $p_k^i(x_1, \ldots, x_k) = x_i$;

ii. il est clos par le *schéma de composition* :
si $h : \mathbb{N}^p \to \mathbb{N}$, et $g_1, \ldots, g_p : \mathbb{N}^n \to \mathbb{N}$ sont récursives primitives, alors $f : \mathbb{N}^n \to \mathbb{N}$ définie par $f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_p(x_1, \ldots, x_n))$ est récursive primitive.

iii. il est clos par le *schéma de récurrence primitive* :
si $g : \mathbb{N}^p \to \mathbb{N}$ et $h : \mathbb{N}^{p+2} \to \mathbb{N}$ sont récursives primitives, alors $f : \mathbb{N}^{p+1} \to \mathbb{N}$ est récursive primitive, f définie par :

$$f(a_1, \ldots, a_p, 0) = g(a_1, \ldots, a_p)$$
$$f(a_1, \ldots, a_p, x+1) = h(a_1, \ldots, a_p, x, f(a_1, \ldots, a_p, x)) .$$

On parlera de *définition récursive primitive* d'une fonction, pour une définition de la fonction qui utilise ces trois clauses.

Un prédicat P sur $\mathbb{N}^p$ (resp. un sous-ensemble E de $\mathbb{N}^p$) est un *prédicat récursif primitif* (resp. *un sous-ensemble récursif primitif*), quand sa fonction caractéristique est récursive primitive.

On rappelle que la fonction caractéristique d'un ensemble est définie par $\chi_A(\overline{x}) = 1$ si $\overline{x} \in A$ et $\chi_A(\overline{x}) = 0$ sinon ; la fonction caractéristique d'un prédicat est celle de l'ensemble des uples pour lesquels le prédicat est vrai.

Plus généralement On dira d'un sous-ensemble de $\bigcup_{p\in\mathbb{N}^*} \mathbb{N}^{\mathbb{N}^p}$ qu'il est *clos par opérations récursives primitives* s'il satisfait les trois clauses **i**, **ii** et **iii** ci-dessus, sans être nécessairement le plus petit. Intuitivement, l'ensemble de toutes les fonctions calculables, doit être clos par opérations récursives primitives. Cela sera démontré quand nous aurons une caractérisation satisfaisante de la notion de fonction calculable.

1.1.1 Exemples de fonctions récursives primitives

Fonctions constantes Pour $n \in \mathbb{N}$, on note s^n la fonction successeur composée n fois. La fonction constante $c_n : \mathbb{N} \to \mathbb{N}$ telle que $c_n(x) = n$ se définit par $c_n(x) = \mathrm{s}^n(\lambda x.0(x))$, en utilisant donc n fois le schéma de composition.

Addition, Multiplication, exponentielle Une définition par récurrence de l'addition est :

$$x + 0 = x \text{ et } x + (y+1) = (x+y) + 1 .$$

et cette définition est récursive primitive. De façon plus explicite, la fonction $+ : \mathbb{N}^2 \to \mathbb{N}$ est définie par

$$+(x, 0) = p^1_1(x) \text{ et } +(x, y+1) = \mathrm{s}(p^3_3(x, y, +(x, y))) .$$

L'addition est bien obtenue à partir des fonctions initiales p^1_1, p^3_1, p^3_3, s, d'une occurrence du schéma de composition, et d'une occurrence du schéma de récurrence primitive.

De même la fonction multiplication $\times : \mathbb{N}^2 \to \mathbb{N}$, définie par récurrence par

$$x \cdot 0 = 0 \text{ et } x \cdot (y+1) = x + x \cdot y$$

est aussi récursive primitive :

$$\times(x, 0) = 0 \text{ et } \times(x, y+1) = +(p^3_1(x, y, +(x, y)), p^3_3(x, y, \times(x, y)))$$

ainsi que la fonction exponentielle définie par

$$x^0 = 0 \text{ et } x^{y+1} = x^y \cdot x .$$

Définitions par récurrence primitive de fonctions à un argument Le schéma de récurrence primitive donné en définition ne permet pas de définir directement des fonctions à un argument. Cependant, on montre facilement qu'il s'étend par :

iii$_0$ Si $b \in \mathbb{N}$, $h : \mathbb{N}^2 \to \mathbb{N}$ est récursive primitive, alors f est récursive primitive, $f : \mathbb{N} \to \mathbb{N}$ définie par

$$f(0) = b$$
$$f(x+1) = h(x, f(x)) .$$

En effet il suffit de définir par récurrence primitive une fonction auxiliaire $aux : \mathbb{N}^2 \to \mathbb{N}$, avec un second argument inutile :

$$aux(a,0) = c_b(a) \text{ et } aux(a,x+1) = h(p_2^3(a,x,f(x)), p_1^3(a,x,f(x))) \; ; f(x) = aux(p_1^1(x), p_1^1(x)).$$

Par exemple la fonction factorielle définie par

$$0! = 1 \text{ et } (x+1)! = (x+1)\cdot x!$$

est récursive primitive en suivant **iii$_0$**. Ce schéma est utilisé au paragraphe suivant pour les fonctions de signe et le prédécesseur.

Signe, prédécesseur, soustraction Les deux fonctions de signe sg et $\overline{\text{sg}}$ qui suivent sont récursives primitives

$$\text{sg}(0) = 0 \text{ et } \text{sg}(x+1) = 1 \; ; \; \overline{\text{sg}}(0) = 1 \text{ et } \overline{\text{sg}}(x+1) = 0$$

($\text{sg}(x+1) = c_1(p_1^2(x, \text{sg}(x))$).
Les fonctions récursives primitives sont partout définies. On définit un prédécesseur nul en 0 et une soustraction tronquée :

$$\text{pred}(0) = 0 \text{ et } \text{pred}(x+1) = x \; ; \; x \dot{-} 0 = x \text{ et } x \dot{-} (y+1) = \text{pred}(x \dot{-} y).$$

avec $\text{pred}(0) = 0$ et $\text{pred}(x+1) = p_1^2(x, \text{pred}(a,x))$.

Dès que l'on a une définition par récurrence sur un argument à partir de fonctions récursives primitives, la fonction obtenue est récursive primitive. Cette restriction aux récurrences à un argument ne peut être levée en toute généralité : on verra qu'il existe des fonctions comme la fonction d'Ackermann (voir 1.2.2 page 13) qui se définissent par récurrence double et ne sont pas récursives primitives. Ainsi une définition naturelle de la fonction $\dot{-}$ par récurrence double est

$$\dot{-}(x+1, y+1) = \dot{-}(x,y), \; \dot{-}(x,0) = x \text{ et } \dot{-}(0,y) = 0 \, .$$

cette définition induit manifestement un calcul, d'ailleurs plutôt plus naturel et plus efficace, de la fonction $\dot{-}$, mais elle ne met pas en évidence que cette fonction est récursive primitive (voir cependant l'exercice 8 page 11).

Comparaison Les prédicats de comparaison $\leq, \geq, <, >, =, \neq$ sont récursifs primitifs, c'est-à-dire que leurs fonctions caractérisques le sont. En effet : $\chi_>(x,y) = \text{sg}(x \dot{-} y)$, $\chi_\leq(x,y) = \overline{\text{sg}}(x \dot{-} y)$, et donc $\chi_=(x,y) = \overline{\text{sg}}(x \dot{-} y) \cdot \overline{\text{sg}}(y \dot{-} x)$, $\chi_\neq(x,y) = \overline{\text{sg}}(\chi_=(x,y))$.

1.1.2 Propriétés de clôtures

Au delà de la définition, l'ensemble des fonctions récursives primitives, et plus généralement tout ensemble clos par opérations récursives primitives, satisfait un grand nombre de propriétés de clôture. On a déjà vu la définition par récurrence primitive des fonctions à un argument. On en détaille quelques autres ci-dessous.

Définition par itération On peut ne pas utiliser tous les arguments dans la récurrence. Par exemple l'addition et la multiplication utilisent le schéma de définition par itération qui est de la forme $f(\overline{x}, 0) = g(\overline{x})$ et $f(\overline{x}, y+1) = h(\overline{x}, f(\overline{x}, y))$. On montre facilement que les fonctions définies ainsi sont récursives primitives.

Sommes et produits bornés Si f de $\mathbb{N}^{p+1} \to \mathbb{N}$ est une fonction récursive primitive, les fonctions g et h de $\mathbb{N}^{p+1} \to \mathbb{N}$ définies par

$$g(x_1, \ldots, x_p, y) = \sum_{i=0}^{y} f(x_1, \ldots, x_p, i) \quad \text{et} \quad h(x_1, \ldots, x_p, y) = \prod_{i=0}^{y} f(x_1, \ldots, x_p, i)$$

sont récursives primitives. En effet (pour la somme) :

$$g(\overline{x}, 0) = f(\overline{x}, 0) \text{ et } g(\overline{x}, y+1) = f(\overline{x}, y+1) + g(\overline{x}, y).$$

(Le schéma de récurrence utilisé est celui de la définition, ce n'est pas une définition par itération).

Opérations logiques L'ensemble des prédicats récursifs primitifs d'arité quelconque est clos par opération booléennes (conjonction, disjonction, négation). Ainsi, si A et B sont des prédicats récursifs primitifs, alors $P(\overline{x},\overline{y}) \wedge Q(\overline{x},\overline{z})$ (pensez à $\chi_P(\overline{x},\overline{y}) \cdot \chi_Q(\overline{x},\overline{z})$) est récursif primitif. La fonction $\overline{\text{sg}}$ permet d'obtenir la négation, et donc la disjonction.

Opérations ensemblistes Les résultats précédents se traduisent immédiatement de façon ensembliste : la classe des sous-ensembles récursifs primitifs de $\mathbb{N}^p$, p fixé, est close par intersection, réunion et passage au complémentaire. La classe des ensembles récursifs primitifs est close par produit cartésien.

Définition par cas Soient f et g deux fonctions récursives primitives et A un ensemble récursif primitif alors la fonction h suivante est récursive primitive :

$$\begin{array}{ll} h(\overline{x}) & = f(\overline{x}) \text{ si } \overline{x} \in A \\ h(\overline{x}) & = g(\overline{x}) \text{ sinon.} \end{array}$$

Cela se voit simplement en remarquant que $h(\overline{x}) = f(\overline{x}) \cdot \chi_A(\overline{x}) + f(\overline{x}) \cdot \chi_{\mathbb{N}^k \setminus A}(\overline{x})$. Plus généralement, si $A_1, ..., A_n$ sont des ensembles récursifs primitifs deux à deux disjoints et $f_1, ..., f_n, g$ des fonctions récursives primitives alors la fonction h suivante est récursive primitive :

$$\begin{array}{ll} h(\overline{x}) & = f_1(\overline{x}) \text{ si } \overline{x} \in A_1 \\ h(\overline{x}) & = f_2(\overline{x}) \text{ si } \overline{x} \in A_2 \\ \vdots & \vdots \\ h(\overline{x}) & = f_n(\overline{x}) \text{ si } \overline{x} \in A_n \\ h(\overline{x}) & = g(\overline{x}) \text{ si } \overline{x} \notin A_1 \cup \cdots \cup A_n. \end{array}$$

Minimisation bornée Soit h une fonction récursive primitive. Une fonction f est obtenue par schéma de *minimisation bornée* à partir de h si elle est définie par :

$$\begin{array}{lcll} f(x_1,\ldots,x_p,y) & = & \text{le plus petit entier } t \leq y \text{ tel que } h(x_1,\ldots,x_p,t) = 0 & \text{s'il existe un tel entier,} \\ f(x_1,\ldots,x_p,y) & = & 0 \text{ s'il n'existe pas de tel entier.} & \end{array}$$

On note l'opération de minimisation bornée :

$$f(\overline{x},y) = \mu t \leq y.[h(\overline{x},t) = 0] \ .$$

La fonction f ainsi obtenue est récursive primitive. En effet :

$$\begin{array}{lcl} f(\overline{x},0) & = & 0 \\ f(\overline{x},y+1) & = & \text{sg}\big(h(\overline{x},0)\big) \cdot \big(f(\overline{x},y) + \overline{\text{sg}}\big(f(\overline{x},y)\big) \cdot \overline{\text{sg}}\big(h(\overline{x},y+1)\big) \cdot (y+1)\big) \end{array}$$

Par composition, si k est récursive primitive, alors f définie par $f(\overline{x}) = \mu t \leq k(\overline{x}).[h(\overline{x},t) = 0]$ est aussi récursive primitive.

Quantifications bornées Si P est un prédicat récursif primitif alors les deux prédicats P_e et P_q définis comme suit sont récursifs primitifs :

$$\begin{array}{l} P_e\, x_1 \ldots x_p\, y \equiv \exists z \leq y\, P\, x_1 \ldots x_p\, y \\ P_q\, x_1 \ldots x_p\, y \equiv \forall z \leq y\, P\, x_1 \ldots x_p\, y \ . \end{array}$$

En effet

$$\begin{array}{l} \chi_{P_e}(\overline{x},y) = \text{sg}(\sum_{z=0}^{y} \chi_P(\overline{x},z)) \\ \chi_{P_q}(\overline{x},y) = \prod_{z=0}^{y} \chi_P(\overline{x},z) \qquad . \end{array}$$

Par composition avec les fonctions de projections, les prédicats (dépendants des mêmes variables que P)

$$\begin{array}{l} \exists z \leq x_i\, P\, x_1 \ldots x_p \\ \forall z \leq x_i\, P\, x_1 \ldots x_p \ . \end{array}$$

sont aussi récursifs primitifs. Plus généralement, on montre par composition que les prédicats

$$\exists z \leq f(\overline{x})\ P\overline{x}$$
$$\forall z \leq f(\overline{x})\ P\overline{x}$$

sont récursifs primitifs dès que le quantificateur est borné par une fonction récursive primitive f.

Les propriétés de clôture de ce paragraphe se généralisent évidemment à tout ensemble de fonctions arithmétiques clos par opérations récursives primitives, puisque seul cette partie de la définition des fonctions récursives primitives a été utilisée.

Exercice 1 Montrer que les sous-ensembles finis et cofinis des $\mathbb{N}^k$, $k \in \mathbb{N}$, sont récursifs primitifs.

Exercice 2 On a vu que l'ensemble des prédicats récursifs primitifs d'arité quelconque est clos sous les opérations booléennes (conjonction, disjonction, négation), (voir page 6). Détailler la démonstration et en déduire que la classe des ensembles récursifs primitifs est close par réunion, intersection, produit cartésien et passage au complémentaire.

1.1.3 Prédicats définissables au premier ordre par quantification bornée

On considère ici un sous-ensemble de prédicats récursifs primitifs qui contient la plupart des prédicats arithmétiques naturels. Appelons $\mathscr{R}$ le plus petit ensemble contenant les prédicats d'addition, de multiplication et clos par opérations booléennes et quantification bornée par un polynôme. En d'autres termes un prédicat $R(x_1, ..., x_k)$ est dans $\mathscr{R}$ s'il est définissable par une formule du premier ordre sur la signature $\{+, \times\}$ et dont toutes les quantifications sont bornées par un polynôme en $x_1, ..., x_k$.

Tout prédicat de $\mathscr{R}$ est récursif primitif (au vu de ce que l'on a déjà prouvé). La classe $\mathscr{R}$ nous fournit un moyen simple (par une définition logique) de montrer que certains prédicats sont récursifs primitifs. Par exemple, un nombre p est premier s'il satisfait :

$$p \text{ est premier } \equiv p \geq 2 \wedge \forall x \leq p\, \forall y \leq p\ \big(x \cdot y = p \rightarrow \big(x = 1 \vee x = p\big)\big)\ .$$

De même, x divise y, noté $x|y$ s'écrit : $x|y \equiv \exists z \leq y\ x \cdot z = y$. De façon générale, la plupart des prédicats arithmétiques naturels sont dans $\mathscr{R}$.

Enfin, en utilisant la clôture par minimisation bornée et par récurrence primitive on montre à partir de l'exemple précédent que la fonction $\mathrm{p} : \mathbb{N} \rightarrow \mathbb{N}$ qui à n associe $\mathrm{p}(n)$ (noté p_n) le $n+1$-ème nombre premier est bien récursive primitive ($\mathrm{p}_0 = 2$, $\mathrm{p}_1 = 3$, ...). Il suffit de remarquer que le $n+1$-ème nombre premier est forcément borné par $\mathrm{p}_n! + 1$ (la factorielle est récursive primitive). La définition de p se fait alors par récurrence :

$$\mathrm{p}(0) = 2 \text{ et } \mathrm{p}(n+1) = \mu x \leq \mathrm{p}(n)! + 1.[x \text{ est premier } \wedge\, x > \mathrm{p}(n)].$$

Exercice 3 (division euclidienne) Montrer que que le prédicat de divisibilité $|$ est récursif primitif, et que les fonctions quotient : $\mathbb{N}^2 \rightarrow \mathbb{N}$ (quotient de la division de n par p), reste : $\mathbb{N}^2 \rightarrow \mathbb{N}$ (reste de la division de n par p) sont des fonctions récursives primitives.

1.1.4 Premiers codages

Les paramètres des fonctions récursives primitives sont des entiers, et ce sera encore le cas pour les diverses notions de fonctions calculables que nous allons étudier. Il est cependant bien évident que la notion de calcul dépasse de loin ce cadre restreint. Dans un sens, un modèle de calcul général doit pouvoir prendre en entrée des objets finis de natures très différentes : nombres autres qu'entiers, mots sur un alphabet fini, structures algébriques finies, graphes, hypergraphes, arbres,... La notion de calcul a intuitivement un sens dans tous ces contextes. De même, clairement, les fonctions calculables doivent pouvoir, à travers une représentation finie (les programmes qui les calculent) être considérées comme des paramètres valides d'autres fonctions calculables.

Dans cette section, on va montrer comment représenter à l'aide d'entiers (on dira souvent « coder »), tout d'abord les couples et n-uplets, puis les listes — c'est-à-dire les suites finies — d'entiers. Ces codages se manipulent par des fonctions récursives primitives, c'est-à-dire que les fonctions usuelles,

par exemple sur les listes (longueur, i-ème élément, sous-liste, etc) sont représentées par des fonctions récursives primitives. De plus ces codages permettent d'établir de nouvelles propriétés de clôture pour les fonctions récursives primitives, définition par récurrence en fonction d'un entier strictement plus petit (qui n'est pas nécessairement le prédécesseur) par exemple. La méthode utilisée pour les listes se généralise à des structures plus complexes comme les arbres étiquetés, ce que l'on verra dans la partie 3.2.

La bijection de Cantor

On appelle α la bijection de Cantor $\mathbb{N}\times\mathbb{N}$ dans $\mathbb{N}$, définie par :

$$\alpha(n,p) = (\sum_{i=0}^{n+p} i) + p = \frac{(n+p+1)(n+p)}{2} + p$$

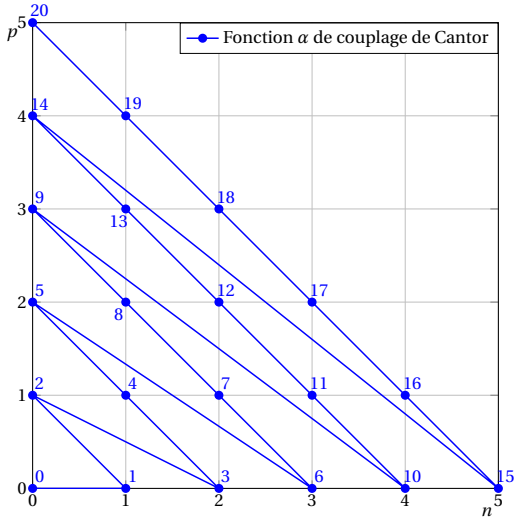
On vérifie facilement que α est bijective, strictement croissante pour ses deux entrées n et p et qu'elle est récursive primitive. Ona également $n \le \alpha(n,p)$ et $p \le \alpha(n,p)$.

L'application étant bijective, on peut définir deux projections π_1 et π_2 vérifiant pour tout entier c et tout couple d'entiers (n,p) :

$$\begin{aligned}&\alpha(\pi_1(c),\pi_2(c)) = c,\\ &\pi_1(\alpha(n,p)) = n\\ &\pi_2(\alpha(n,p)) = p\end{aligned}$$

qui sont également récursives primitives. En effet :

$$\begin{aligned}&\pi_1(x) = \mu z \le x.[\exists t \le x\ \alpha(z,t) = x]\\ &\pi_2(x) = \mu t \le x.[\exists z \le x\ \alpha(z,t) = x].\end{aligned}$$



On peut alors définir par récurrence sur $k \ge 1$ les fonctions $\alpha_k : \mathbb{N}^k \to \mathbb{N}$ par :

$$\alpha_1(n) = n$$
$$\alpha_{k+1}(n_1,\dots,n_{k+1}) = \alpha_2(n_1,\alpha_k(n_2,\dots,n_{k+1})) \quad (\text{ en particulier } \alpha_2 = \alpha)\ .$$

À nouveau on démontre par récurrence sur k que chaque α_k, $k \in \mathbb{N}^*$, est une bijection. Les projections correspondantes, notées π_i^k pour $1 \le i \le k$ sont définies par :

$$\text{pour } 1 \le i < k,\ \pi_i^k = \pi_1 \circ \underbrace{\pi_2 \circ \cdots \circ \pi_2}_{i-1}\ ; \quad \pi_k^k = \underbrace{\pi_2 \circ \cdots \circ \pi_2}_{k-1} \quad (\text{en particulier } \pi_1^2 = \pi_1 \text{ et } \pi_2^2 = \pi_2).$$

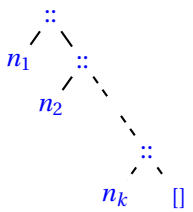
Chacune des fonctions α_k, de même que les projections π_i^k, sont récursives primitives par composition. On pose $\alpha_k(x_1,\dots,x_k) = \langle x_1,\dots,x_k\rangle$ et on utilisera le plus souvent cette dernière écriture.

Cette famille de fonctions ne permet pas, telle quelle, de définir une bijection de l'ensemble des suites finies vers $\mathbb{N}$ puisque précisement chacune de ces fonctions est bijective. Par exemple : $\alpha_3(1,0,0) = \alpha_2(1,\alpha_2(0,0)) = \alpha_2(1,0)$. On va modifier un tout petit peu l'approche pour obtenir un codage bijectif des suites finies, les listes de l'informatique.

Un codage bijectif des suites finies

Pour coder les suites finies d'entiers de taille arbitraire, ou listes d'entiers, on utilise à nouveau le codage des couples.

Le principe du codage est illustré par le schéma ci-contre. Le code de la liste vide (noté []) est 0, le code d'une liste non vide est donné par le couple constitué de premier élément de la liste et du code du reste de la liste, et ce couple doit être non nul.



La fonction cons : $\mathbb{N}^2 \to \mathbb{N}$ associe à x et y l'entier noté $x :: y$, elle est définie par :

$$x :: y = 1 + \alpha(x, y)$$

On obtient ainsi une fonction récursive primitive bijective de $\mathbb{N}^2 \to \mathbb{N}^*$. On appelle hd (pour *head*) et tl (pour *tail*) les fonctions vérifiant :

$$\begin{array}{ll} \mathrm{hd}(0) = 0 & \mathrm{tl}(0) = 0 \\ \mathrm{hd}(x :: y) = x & \mathrm{tl}(x :: y) = y \end{array}$$

La fonction liste de l'ensemble $\mathscr{S}$ des suites finies d'entiers dans $\mathbb{N}$ est définie inductivemement, on note $[a_0; \ldots; a_n] = \mathrm{liste}((a_0, \ldots, a_n))$:

$$\begin{array}{l} [\,] = 0 \\ [a_0; \ldots; a_n] = a_0 :: [a_1; \ldots; a_n] \end{array}$$

La fonction liste est bijective : on démontre par récurrence sur l'entier l que celui-ci possède un unique antécédent, en utilisant que α_2 est bijective. La fonction :: est récursive primitive (car α_2 l'est). Les fonctions hd et tl sont récursives primitives :

$$\mathrm{hd} = \pi_1 \circ \mathrm{pred} \quad \text{et} \quad \mathrm{tl} = \pi_2 \circ \mathrm{pred}\ .$$

La fonction nthl qui à l et i associe la suite codée par l à partie du $i+1$-ème élément (0 sinon), et la fonction nth qui à l et i associe le $i+1$-ème élément de la suite codée par l (0 sinon), sont récursives primitives :

$$\begin{array}{lll} \mathrm{nthl}(l, 0) = l & \text{puis} & \mathrm{nth}(l, 0) = 0 \\ \mathrm{nthl}(l, i+1) = \mathrm{tl}(\mathrm{nthl}(l, i)) & & \mathrm{nth}(l, i+1) = \mathrm{hd}(\mathrm{nthl}(l, i))\ . \end{array}$$

Ce codage bijectif des listes se généralise facilement à d'autres types de données inductifs [1], par exemple celui des arbres binaires étiquetés de la section 3.2.1 page 67 (utilisé pour la démonstration du premier théorème d'incomplétude).

Récurrence primitive sur la suite des valeurs On peut souhaiter faire dépendre la récurrence non seulement de la dernière valeur obtenue pour la fonction f mais de tout (ou partie) des précédentes. On parle de *récurrence sur la suite des valeurs*. L'ensemble des fonctions récursives primitives est clos par le schéma de récurrence sur la suite des valeurs. Même si elle fait appel au codage des listes introduit ci-dessus, la proposition suivante est pour une large part indépendante du codage choisi.

Lemme 1.1.2 *soient $g : \mathbb{N}^p \to \mathbb{N}$ et $h : \mathbb{N}^{p+2} \to \mathbb{N}$ deux fonctions récursives primitives, alors la fonction $f : \mathbb{N}^{p+1} \to \mathbb{N}$ suivante est récursive primitive :*

$$\begin{array}{l} f(a_1, \ldots, a_p, 0) = g(a_1, \ldots, a_p) \\ f(a_1, \ldots, a_p, x+1) = h\big(a_1, \ldots, a_p, x, \big[f(a_1, \ldots, a_p, x); \ldots; f(a_1, \ldots, a_p, 0)\big]\big)\ . \end{array}$$

L'appel fait appel à la suite des précédents, donc peut faire appel à n'importe quel d'entre eux avec la fonction nth du paragraphe précédente.

Démonstration. Appelons $\tilde{f}(\overline{a}, x) = \big[f(\overline{a}, x); \ldots; f(\overline{a}, 0)\big]$. On montre tout d'abord que $\tilde{f}$ est récursive primitive. En effet, $\tilde{f}(\overline{a}, 0) = g(\overline{a}) :: 0$ et

$$\tilde{f}(\overline{a}, x+1) = f(\overline{a}, x+1) :: \tilde{f}(\overline{a}, x) = h\big(\overline{a}, x, \tilde{f}(\overline{a}, x)\big) :: \tilde{f}(\overline{a}, x)$$

On a alors $f(\overline{a}, x) = \mathrm{hd}(\tilde{f}(\overline{a}, x))$. ■

1. Ces codages sont empruntés à KOMARA et VODA 1999

On laisse en exercice le fait de montrer que les fonctions suivantes (dont on se servira dans la suite) sont récursives primitives :

— la fonction mem fonction caractéristique de l'appartenance d'un entier n à une suite codée par l,
— la fonction @ vérifiant que $l@l'$ est le code de la concaténation des suites codées par l et l',
— la fonction len qui à un entier l associe la longueur de la suite codée par l,
— la fonction inc : $\mathbb{N}^2 \to \mathbb{N}$ qui à i et $l = [a_0;\ldots;a_i;\ldots;a_n]$ associe $\text{inc}(i,l) = [a_0;\ldots;a_i+1;\ldots;a_n]$ quand $i \leq n$, $\text{inc}(i,l) = l$ sinon.

Exercice 4 (Codage des listes par décomposition en nombres premiers) Un autre codage des listes s'appuie sur la décomposition en nombres premiers. On note $\mathscr{S}$ l'ensemble des suites finies d'entiers. Soit $\text{p} : \mathbb{N} \to \mathbb{N}$ la fonction qui à n associe le $n+1$-ème nombre premier noté p_n. Cette fonction est récursive primitive. La fonction de codage des listes $\text{seq} : \mathscr{S} \to \mathbb{N}$ associe à chaque suite de longueur finie $(x_1,\ldots,x_k)$ la valeur suivante

$$\text{seq}(x_1,...,x_k) = p_0^k \cdot p_1^{x_1} \cdot p_2^{x_2} \cdots p_k^{x_k},$$

avec la convention pour la suite vide (), seq() = 1.

1. Montrer que ce codage est injectif mais pas surjectif (il construit des nombres très grands ce qui le rendrait difficilement utilisable en pratique).

2. Montrer que la fonction de $\mathbb{N}^2 \to \mathbb{N}$ qui à (x,n) associe l'exposant de p_n dans la décomposition en facteurs premiers de x est récursive primitive.

3. En déduire que

3.a. il existe une fonction récursive primitive qui calcule le n-ième élément d'une suite représentée par x, quand x représente une suite de longueur supérieure ou égale à n (on ne se préoccupe pas de la valeur de la fonction dans les autres cas) ;

3.b. il existe une fonction récursive primitive qui calcule la longueur de la suite codée par x, quand x représente une suite ;

3.c. La fonction caractéristique de l'ensemble des codes de suites (l'ensemble image de la fonction seq) est récursive primitive.

4. Montrer qu'il existe une fonction récursive primitive qui, à deux entiers $\text{seq}(x_1,...,x_k)$ et $\text{seq}(y_1,...,y_h)$ codant des suites renvoie le nombre représentant la concaténation des deux listes $\text{seq}(x_1,...,x_k,y_1,...,y_h)$.

Exercice 5 (Définition par récurrences mutuelles) Utiliser la fonction α_k pour montrer que si les fonctions $g_1,\ldots,g_k : \mathbb{N}^n \to \mathbb{N}$, et $h_1,\ldots,h_k : \mathbb{N}^{n+k+1} \to \mathbb{N}$ sont récursives primitives, alors, les fonctions $f_1,\ldots,f_k$ définies ci-dessous sont récursives primitives (on écrit $\overline{a}$ pour $a_1,\ldots,a_n$).

$$\begin{array}{ll} f_1(\overline{a},0) = g_1(\overline{a}) & f_k(\overline{a},0) = g_k(\overline{a}) \\ f_1(\overline{a},x+1) = h_1(\overline{a},x,f_1(\overline{a},x),\ldots,f_k(\overline{a},x)) & \cdots \quad f_k(\overline{a},x+1) = h_k(\overline{a},x,f_1(\overline{a},x),\ldots,f_k(\overline{a},x)) \end{array}$$

Exercice 6 (récurrence sur les listes) Le but de l'exercice est de montrer que la définition par récurrence naturelle sur la structure de liste se code de façon récursive primitive.

1. Montrer que l'ensemble des fonctions récursives primitives est clos par le schéma de récurrence sur les listes (préciser pourquoi f est bien définie) : si $g : \mathbb{N}^p \to \mathbb{N}$ et $h : \mathbb{N}^{p+3} \to \mathbb{N}$ sont récursives primitives, alors $f : \mathbb{N}^{p+1} \to \mathbb{N}$ est récursive primitive, f définie par

$$\begin{array}{l} f(a_1,\ldots,a_p,[\,]) = g(a_1,\ldots,a_p) \\ f(a_1,\ldots,a_p,x::l) = h\big(a_1,\ldots,a_p,x,l,f(a_1,\ldots,a_p,l)\big)\,. \end{array}$$

et l'utiliser pour montrer que la fonction mem fonction caractéristique de l'appartenance d'un entier n à une suite codée par l, la fonction @ vérifiant que $l@l'$ est le code de la concaténation des suites codées par l et l', la fonction len qui à un entier l associe la longueur de la suite codée par l, sont récursives primitives.

2. Montrer que si $f : \mathbb{N}^{p+1} \to \mathbb{N}$ est récursive primitive, alors la fonction map_f qui à $a_1, \ldots, a_p, l$ codant $(u_i)_{i\leq n}$ associe $\text{map}_f(l)$ codant $\big(f(a_1, \ldots, a_p, u_i)\big)_{i\leq n}$ est récursive primitive.

Montrer que la fonction $concat$ qui à un entier l codant une liste de liste $\big((u_i)_{i\leq n_i}\big)_{i\leq p}$ associe l'entier $concat(l)$ codant la suite des entiers de chaque suite $(u_i)_{i\leq n_i}$ dans le même ordre est récursive primitive.

Montrer que la fonction $subst$ qui à trois entiers l, k, v, associe le code la suite obtenue en remplaçant dans la suite codée par l toutes les occurrences de v par les entiers de la suite codée par k est récursive primitive (on peut se servir des deux fonctions précédentes).

Exercice 7 (récurrence avec substitution de paramètre) On appelle *schéma de récurrence avec substitution de paramètre* le schéma de récurrence suivant (énoncé ici avec un seul paramètre) qui étant données g, $\gamma : \mathbb{N} \to \mathbb{N}$ et $h : \mathbb{N}^3 \to \mathbb{N}$ définit $f : \mathbb{N}^2 \to \mathbb{N}$.

$$\begin{aligned} f(a,0) &= g(a) \\ f(a,x+1) &= h(a,x,f(\gamma(a),x)) . \end{aligned}$$

Intuitivement, ce schéma conserve le fait d'être calculable (le calcul termine puisque la variable de récurrence décroit). La récurrence avec substitution de paramètre apparaît naturellement avec des structures de données plus complexes que les entiers, par exemple quand on définit une fonction par récurrence sur la longueur d'une liste (le paramètre est la liste elle-même), ou sur la hauteur d'un arbre (le paramètre est l'arbre).

On montre que l'ensemble des fonctions récursives primitives est clos sous ce schéma. On suppose dans la suite g, γ et h récursives primitives, et f définie comme ci-dessus. On note $\gamma^p(x) = \underbrace{\gamma \circ \cdots \circ \gamma}_{p}(x)$ (γ^0 est l'identité).

1. Montrer que la fonction F définie ci-dessous est récursive primitive

$$\begin{aligned} F(p,a,0) &= g\left(\gamma^p(a)\right) \\ F(p,a,x+1) &= h\left(\gamma^{p\dot{-}(x+1)}(a), x, F(p,a,x)\right) . \end{aligned}$$

2. Montrer que :

$$\forall x,a,p \in \mathbb{N}\ \left(x \leq p \to F(p,a,x) = f(\gamma^{p\dot{-}x}(a),x)\right)$$

et en déduire que f est récursive primitive.

3. Application : montrer que la fonction $\text{inc} : \mathbb{N}^2 \to \mathbb{N}$ qui à i et $l = [a_0;\ldots;a_i;\ldots;a_n]$ associe $\text{inc}(i,l) = [a_0;\ldots;a_i+1;\ldots;a_n]$ quand $i \leq n$, $\text{inc}(i,l) = l$ sinon, est récursive primitive.

Exercice 8 (récurrence double sans imbrication) On verra que la récurrence double ne conserve pas en général le fait d'être récursif primitif (la fonction d'Ackermann, voir 1.2.2 page 13, est définie par récurrence double). On peut montrer que s'il n'y a pas *imbrication* des appels récursifs dans la récurrence double, alors celle-ci reste « récursive primitive ».

On suppose que a et b sont des entiers, que h est une fonction récursive primitive à 4 arguments, que h_2 est une fonction primitive récursive à 2 arguments.

Pour simplifier les notations, les schémas qui suivent sont donnés sans paramètres, les démonstrations étant essentiellement les mêmes en présence de paramètres.

1. Montrer que la fonction f définie par :

$$\begin{aligned} f(0,y) &= a , \\ f(x+1,0) &= b , \\ f(x+1,y+1) &= h(x,y,f(x,y),f(x+1,y)) . \end{aligned}$$

est récursive primitive (on peut utiliser le codage des couples et la récurrence sur la suite des valeurs).

2. (généralisation, plus difficile) Montrer que la fonction f définie par :

$$\begin{aligned} f(0,y) &= a, \\ f(x+1,0) &= b, \\ f(x+1,y+1) &= h(x,y,f(x,h_2(x,y)),f(x+1,y)). \end{aligned}$$

est récursive primitive. On peut procéder comme à la question précédente, mais en cherchant pour h_2 donné une nouvelle fonction α' de codage des couples (injective mais non nécessairement bijective) vérifiant :

$$\alpha'(x+1,y) < \alpha'(x+1,y+1) \text{ et } \alpha'(x,h_2(x,y)) < \alpha'(x+1,y+1).$$

Rózsa Péter a étudié à partir des années 1930 de façon assez systématique la clôture de l'ensemble des fonctions récursives primitives sous divers schémas de récurrence dont ceux exposés ci-dessus [2].

1.2 Au delà des fonctions récursives primitives

1.2.1 Évaluation des fonctions récursives primitives

Dans ce paragraphe, « fonction calculable » signifie « fonction calculable au sens intuitif » : on dispose d'un procédé qui pour une entrée donnée permet d'obtenir en un temps fini le résultat de la fonction appliquée à cette entrée. On va voir que les fonctions récursives primitives ne résument pas à elles seules l'ensemble des fonctions calculables au sens intuitif, pour une raison qui tient au fait même que les fonctions récursives primitives sont calculables.

Pour s'en persuader, admettons tout d'abord qu'il est possible de *coder* les définitions des fonctions récursives primitives par des entiers, en utilisant par exemple les codages des suites (voir la section partie 1.1.4), ces définitions étant des suites finies de lettres, de façon analogue aux codages que l'on réalisera pour les machines en section 1.3 (ou mieux, ceux que l'on verra en section 3.2). Il est alors assez simple de se rendre compte que la fonction caractéristique de l'ensemble de ces codes est calculable (et même récursive primitive, sauf choix de codage aberrant), ce qui correspond à l'idée intuitive que l'on sait reconnaître si un assemblage de lettres est bien la définition d'une fonction récursive primitive. De la même façon on admet que l'arité de la fonction de code i se calcule en fonction de i (la fonction étant même récursive primitive, sauf, à nouveau, choix de codage aberrant).

La fonction d'évaluation des fonctions récursives primitives à un argument est alors une fonction Eval à deux arguments, telle que, si i est le code d'une fonction f récursive primitive à un argument, alors pour tout entier x, $\mathsf{Eval}(i,x) = f(x)$. La fonction peut être prolongée de façon arbitraire, par exemple elle vaut toujours 0, quand i n'est pas un code de fonction récursive primitive à un argument.

Dire des fonctions récursives primitives qu'elles sont calculables, c'est bien dire que cette fonction d'évaluation est calculable. Or cette fonction ne peut être récursive primitive. On le montre par diagonalisation : si elle l'était, la fonction à un argument $x \mapsto \mathsf{Eval}(x,x)+1$ le serait. Or si cette fonction était récursive primitive, elle aurait un code n, et on aurait $\mathsf{Eval}(n,n)+1 = \mathsf{Eval}(n,n)$.

On peut essayer d'analyser la définition de la fonction Eval, pour observer à quel endroit elle ne rentre pas dans le cadre des fonctions récursives primitives. Réaliser les codages qui permettent d'écrire en détail la fonction Eval est un peu long mais ne présente pas de difficulté particulière, et on verra ultérieurement plusieurs exemples de tels codages. Contentons nous de définir une syntaxe pour les fonctions récursives primitives, et de décrire l'évaluation pour cette syntaxe.

Les termes récursifs primitifs d'arité n ($n \in \mathbb{N}^*$) sont définis inductivement, sachant que même si on reprend pour simplifier les mêmes notations que pour les fonctions, un tel terme n'est qu'une suite de lettres :

i. les lettres $\lambda x.0$ et s sont des termes récursifs primitifs d'arité 1 ; pour $k \in \mathbb{N}^*$, $1 \le i \le k$, p_i^k est un terme récursif primitif d'arité k ;

ii. si h est un terme récursif primitif d'arité p et $g_1,\dots,g_p$ sont des termes récursifs primitifs d'arité n, alors $h \circ (g_1,\dots,g_p)$ est un terme récursif primitif d'arité n ;

2. voir PÉTER 1967

iii. si g et h sont des termes récursifs primitifs, g d'arité p et h d'arité $p+2$, alors $\mathrm{rec}(g,h)$ est un terme récursif primitif d'arité $p+1$.

L'interprétation de ces termes par des fonctions récursives primitives est celle que l'on imagine, la famille de fonctions (Eval_n) permet de l'expliciter (on retrouve la fonction Eval par $\mathsf{Eval} = \mathsf{Eval}_1$ et, grâce au codages de uples, il n'est pas difficile de définir (Eval_n) à partir de Eval_1).

i. $\mathsf{Eval}_1(\lambda x.0, x) = 0$, $\mathsf{Eval}_1(\mathrm{s}, x) = x+1$, $\mathsf{Eval}_k\left(p_i^k, x_1, \ldots, x_k\right) = x_i$;

ii. $\mathsf{Eval}_n\left(h \circ (g_1, \ldots, g_p), x_1, \ldots, x_n\right) = \mathsf{Eval}_p\left(h, \mathsf{Eval}_n(g_1, x_1, \ldots, x_n), \ldots, \mathsf{Eval}_n(g_p, x_1, \ldots, x_n)\right)$;

iii. $\mathsf{Eval}_{p+1}\left(\mathrm{rec}(g,h), a_1, \ldots, a_p, 0\right) = \mathsf{Eval}_p\left(g, a_1, \ldots, a_p\right)$
$\mathsf{Eval}_{p+1}\left(\mathrm{rec}(g,h), a_1, \ldots, a_p, x+1\right) = \mathsf{Eval}_{p+2}\left(h, a_1, \ldots, a_p, x, \mathsf{Eval}_{p+1}\left(\mathrm{rec}(g,h), a_1, \ldots, a_p, x\right)\right)$.

On suppose donc que les termes récursifs primitifs sont codés par des entiers, et que de plus un entier codant un terme est strictement supérieur aux entiers codant ses sous-termes stricts, ce qui est le cas pour les choix naturels de codage.

On peut voir alors la définition des fonctions (Eval_n) comme une définition par récurrence mutuelle, l'argument de récurrence, celui qui décroît, est le code de la fonction. En examinant celle-ci on s'aperçoit que

— l'évaluation des fonctions initiales est récursive primitive : ce sont de simples compositions des fonctions initiales ;

— La condition donnée par l'évaluation de la composition reste dans le cadre récursif primitif : c'est une généralisation de la définition par récurrences mutuelles (exercice 5 page 10) ;

— l'évaluation de la récurrence primitive utilise une récurrence double, avec imbrication des appels aux fonctions Eval : c'est cette clause de la définition de l'évaluation qui fait que celle-ci n'est pas récursive primitive.

1.2.2 Fonction d'Ackermann

La méthode qui précède est très générale, mais lourde à mettre en œuvre explicitement. Elle peut se simplifier, non plus en énumérant toutes les fonctions récursives primitives, mais en énumérant une suite Ack_n de fonctions récursives primitives d'arité 1, de façon que Ack_n soit supérieure à partir d'un certain rang à toutes les fonctions récursives primitives utilisant moins de n occurrences du schéma de récurrence [3]. La fonction à deux arguments définie par $\mathrm{Ack}(n,x) = \mathrm{Ack}_n(x)$ énumère les fonctions Ack_n, et la fonction diagonale $x \mapsto \mathrm{Ack}(x,x)$ sera alors supérieure à partir d'un certain rang à toute fonction récursive primitive, donc ne pourra être récursive primitive.

Cette fonction est appelée *fonction d'Ackermann* [4]. Elle se définit par récurrence double avec imbrication des appels récursifs :

$$\begin{aligned}
&\mathrm{Ack}(0,x) = x+2\\
&\mathrm{Ack}(1,0) = 0\\
&\mathrm{Ack}(n+2,0) = 1\\
&\mathrm{Ack}(n+1,x+1) = \mathrm{Ack}(n,\mathrm{Ack}(n+1,x))
\end{aligned}$$

Chaque fonction $\mathrm{Ack}_n : x \mapsto \mathrm{Ack}(n,x)$ est bien récursive primitive : la fonction Ack_{n+1} est obtenue en itérant la fonction Ack_n

$$\mathrm{Ack}_{n+1}(x+1) = \mathrm{Ack}_n(\mathrm{Ack}_{n+1}(x)) .$$

Toutes les fonctions Ack_n sont croissantes, et cette croissance est de plus en plus rapide

$$\mathrm{Ack}_1(x) = 2x,\ \mathrm{Ack}_2(x) = 2^x,\ \mathrm{Ack}_3(x) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}x},\ \ldots$$

On dit qu'une fonction $f : \mathbb{N} \longrightarrow \mathbb{N}$ *domine* une fonction $g : \mathbb{N}^p \longrightarrow \mathbb{N}$ si f est supérieure à g à partir d'un certain rang, i.e.

$$\exists K \in \mathbb{N}\ \forall \vec{x} \in \mathbb{N}^p\ g(\overline{x}) \leq f(\sup(\overline{x}, K))$$

3. La fonction Ack_n est même supérieure à partir d'un certain rang aux fonctions récursives primitives utilisant au plus n occurrences *imbriquées* du schéma de récurrence, voir la définition des $\mathscr{C}_n$ page 14.

4. La fonction originale avait 3 arguments, Rozsa Péter a donné une version a 2 arguments à laquelle celle donnée ici est quasi-identique).

Alors on peut montrer que pour toute fonction primitive récursive f, il existe n tel que Ack_n domine f, ce dont on déduit

Proposition 1.2.1 *La fonction d'Ackermann n'est pas récursive primitive.*

Les détails sont donnés dans la section suivante et en exercice.

Une hiérarchie des fonctions récursives primitives On définit une suite $\mathscr{C}_n$ d'ensembles de fonctions primitives récursives tous clos par composition, les fonctions de $\mathscr{C}_n$ étant les fonctions qui utilisent des suites imbriquées d'au plus n schémas de récurrence primitive. En voici une définition par induction :

(i) $\mathscr{C}_0$ est la clôture par composition de l'ensemble des fonctions de bases : constantes, projections, et successeur ;

(ii) $\mathscr{C}_{n+1}$ est la clôture par composition de la réunion de $\mathscr{C}_n$ et des fonctions obtenues par une seule occurrence du schéma de récurrence primitive à partir des fonctions de $\mathscr{C}_n$.

Il est clair que $\mathscr{C}_n \subset \mathscr{C}_{n+1}$ et que $\bigcup_{i=0}^{\infty} \mathscr{C}_n$ est l'ensemble de toutes les fonctions récursives primitives. On vérifie facilement que $\mathrm{Ack}_n \in \mathscr{C}_n$. On peut maintenant préciser l'énoncé du paragraphe précédent.

Proposition 1.2.2 *Si $f \in \mathscr{C}_n$, alors Ack_{n+1} domine f, en particulier $\mathrm{Ack}_{n+1} \notin \mathscr{C}_n$.*

La hiérarchie des $\mathscr{C}_n$ est donc stricte, puisque $\mathrm{Ack}_{n+1} \in \mathscr{C}_{n+1}$, mais $\mathrm{Ack}_{n+1} \notin \mathscr{C}_n$.

Remarque. En particulier, les fonctions de $\mathscr{C}_2$, niveau 3 de la hiérarchie, sont connues sous le nom de *fonctions élémentaires au sens de Kalmar*. La fonction Ack_3 n'est pas élémentaire.

Les démonstrations, en particulier celle de la proposition précédente, sont détaillées dans l'exercice qui suit.

Exercice 9 (fonction d'Ackermann)

1. Vérifier qu'il existe bien une et une seule fonction de $\mathbb{N}^2 \to \mathbb{N}$ vérifiant les équations de la fonction d'Ackermann. Calculez explicitement les premières valeurs de Ack, par exemple $\{\mathrm{Ack}(n,x) \mid 0 \le n \le 3,\ 0 \le x \le 3\}$, et donner un argument informel pour la calculabilité (au sens intuitif) de Ack, c'est-à-dire la raison pour laquelle la suite des appels récursifs termine (indication : utiliser l'ordre lexicographique sur les couples).

2. Montrer que
$$\forall n \in \mathbb{N}\ \forall x > 0\ \mathrm{Ack}_{n+1}(x) = \underbrace{\mathrm{Ack}_n \circ \cdots \circ \mathrm{Ack}_n}_{x}(\mathrm{Ack}_{n+1}(0))$$
et vérifier les expressions des fonctions Ack_1, Ack_2 et Ack_3 données ci-dessus.

3. Vérifiez que chacune des fonctions Ack_n est récursive primitive, et donnez en une définition dont vous montrerez qu'elle utilise exactement n instances du schéma de définition par itération vu en section 1.1.2 page 5. On peut remarquer que les n schémas de récurrences sont imbriqués.

4. Montrer que $\forall n \in \mathbb{N} \forall x \in \mathbb{N}^*\ \mathrm{Ack}_n(x) > x$.

5. En déduire que pour tout entier n, Ack_n est strictement croissante.

6. Déduire de la question 4 que, à partir de 2, Ack est croissante au sens large sur son premier argument, le second étant fixé :
$$\forall x \ge 2\ \forall n \in \mathbb{N}\ \mathrm{Ack}(n,x) \le \mathrm{Ack}(n+1,x)\,.$$

La hiérarchie $\mathscr{C}_n$ des fonctions récursives primitives, ainsi que la définition de $f : \mathbb{N} \longrightarrow \mathbb{N}$ *domine* $g : \mathbb{N}^p \longrightarrow \mathbb{N}$ ont été données ci-dessus.
On pose pour k entier, $\mathrm{Ack}_n^k = \underbrace{\mathrm{Ack}_n \circ \cdots \circ \mathrm{Ack}_n}_{k}$.

7. Montrer que : $\forall n, k \in \mathbb{N}\ \mathrm{Ack}_n^k \in \mathscr{C}_n$

8. Montrer que $\forall n, k, x \in \mathbb{N}\ \mathrm{Ack}_n^k(x) \le \mathrm{Ack}_{n+1}(x+k)$.

9. Montrer par récurrence sur la définition de l'ensemble des fonctions récursives primitives que :

$$\text{si } f \in \mathscr{C}_n, \text{ alors } \exists k \in \mathbb{N}\ \mathrm{Ack}_n^k \text{ domine } f.$$

10. Montrer que Ack_n^k est dominée par Ack_{n+1} (on pourra montrer que pour $y > 0$, $\mathrm{Ack}_{n+1}(y) \geq 2y$, puis que pour $x > 2k$, $\mathrm{Ack}_{n+1}(x-k) \geq x$).

11. En déduire que si $f \in \mathscr{C}_n$, alors Ack_{n+1} domine f.

12. En déduire que la fonction d'Ackermann n'est pas récursive primitive.
On peut également montrer que la fonction diagonale $n \mapsto \mathrm{Ack}(n,n)$ domine toutes les fonctions récursives primitives.

1.2.3 Fonctions partielles μ-récursives

La méthode décrite (informellement) pour montrer que la fonction d'évaluation des fonctions récursives primitives n'est pas récursive primitive est très générale. Elle permet de montrer par diagonalisation que la fonction d'évaluation de n'importe quel ensemble de fonctions calculables n'appartient pas à cet ensemble, pourvu essentiellement que deux conditions soient réalisées.

1. La première que l'on a déjà indiquée, c'est que l'on puisse reconnaître de façon effective les codes des fonctions de l'ensemble en question.

2. La seconde qui est restée implicite jusqu'à présent, c'est que ces fonctions soient *partout définies*. En effet la contradiction n'arrive que si la fonction Eval est définie en (n,n) (ce qui est forcément le cas pour la fonction d'évaluation des fonctions récursives primitives).

Il est possible de remettre en cause la première condition. Mais le plus naturel du point de vue du calcul est de remettre en cause la seconde condition, c'est-à-dire d'étendre la notion de fonction calculable aux fonctions partielles. L'intuition est qu'une fonction partielle f n'est pas définie en une valeur x donnée, quand le calcul de $f(x)$ se poursuit indéfiniment sans jamais rendre une valeur. Dans les langages de programmation usuels cela peut arriver dès que l'on utilise une boucle `while`.

L'introduction de fonctions partielles calculables date des années 1930, bien avant les débuts de l'informatique. Il se trouve qu'un programme dont l'exécution ne termine pas est une chose indispensable en informatique. Par exemple une boucle interactive, qui attend une intervention de l'utilisateur n'a pas à terminer (sauf sur ordre de l'utilisateur). A fortiori un système d'exploitation ne termine pas sauf intervention extérieure. Cependant, même si l'activité des ordinateurs ne se résume pas, très loin de là, au calcul de fonctions, c'est la seule chose que formalise la calculabilité et la seule chose dont nous nous préoccuperons ici.

Le schéma de minimisation On va donc d'une part considérer des fonctions partielles de $\mathbb{N}^k$ dans $\mathbb{N}$ c'est à dire des fonctions définies sur un sous ensemble A de $\mathbb{N}^k$ (et non définies en dehors), d'autre part introduire un nouveau schéma de définition, qui peut produire des fonctions partielles à partir de fonctions totales. Il sera également nécessaire de généraliser la composition et la récurrence primitive aux fonctions partielles.

Notation : on écrira $f(x)\downarrow$ pour f est définie en x, $f(x)\uparrow$ pour f n'est pas définie en x.

Définition 1.2.3 L'opérateur μ est défini par le schéma suivant dit schéma μ ou schéma de minimisation. Soit f une fonction partielle calculable :

$$z = \mu y.\big(f(\overline{x},y) = 0\big) \quad \text{ssi} \begin{cases} f(\overline{x},z) = 0 \\ \text{pour tout } y < z, f(\overline{x},y)\downarrow \text{ et } f(\overline{x},y) \neq 0. \end{cases}$$

Le schéma de minimisation ci-dessus introduit une fonction de $\overline{x}$ qui est a priori partielle, même si la fonction f est totale (récursive primitive par exemple). On trouvera forcément des fonctions calculables f telles que que pour un certain uple $\overline{x}$, $f(\overline{x},y) \neq 0$ pour toute valeur de y, et alors $\mu y.\big(f(\overline{x},y) = 0\big)$ n'est simplement pas défini.

Clairement, le schéma de minimisation doit être défini de telle sorte que si $\mu y.\big(f(\overline{x},y) = 0\big)$ a une valeur z alors on peut obtenir ce z par un calcul. C'est bien le cas ici : trouver z, revient à calculer

successivement $f(\overline{x},0)$, $f(\overline{x},1)$, ... et à s'arrêter au premier y tel que $f(\overline{x},y)=0$. Vu ainsi, on a bien besoin que chacune des valeurs intermédiaires soit définie pour que le calcul arrive à son terme.

Imaginons la version suivante du schéma de minimisation :

$$z = \min y.\big(f(\overline{x},y)=0\big) \quad \text{ssi} \begin{cases} f(\overline{x},z)=0 \\ \text{pour tout } y<z, f(\overline{x},y)\neq 0. \end{cases}$$

Si z est le plus petit entier tel que $f(\overline{x},z)=0$ mais qu'il existe $y<z$ tel que $f(\overline{x},y)$ est non définie, il n'existe pas de procédure évidente pour calculer z. De fait, on montrera que ce schéma ne peut convenir quand on aura une caractérisation satisfaisante de fonction partielle calculable (exercice 17 page 42).

Si on souhaite obtenir des fonctions totales à l'issue d'une étape de minimisation, il faut imposer de n'appliquer ce schéma qu'à des fonctions totales calculables f telles que :

$$\forall \overline{x}\, \exists y\ f(\overline{x},y)=0.$$

Une telle fonction calculable f sera dite *régulière*.

Le schéma de minimisation conserve le fait d'être « calculable » (en un sens intuitif), pour les fonctions partielles comme totales. Il s'avère, qu'en ajoutant le schéma de minimisation aux schémas de clôtures des fonctions récursives primitives, on obtient en fait une caractérisation satisfaisante de la notion de fonction calculable (cela sera précisé à la section 1.5.3).

On appelle pour l'instant *fonctions μ-récursives* les fonctions calculables au sens de cette définition introduite par Kleene. Plus tard on les appelera simplement fonctions calculables.

Définition 1.2.4 (fonction partielle μ-récursive) L'ensemble des *fonctions partielles μ-récursives* est le plus petit sous-ensemble de fonctions partielles à plusieurs arguments entiers

i. contenant la fonction nulle, les projections et la fonction successeur

ii. clos par composition des fonctions partielles si $h:\mathbb{N}^p \longrightarrow \mathbb{N}$, et $g_1,\ldots,g_p:\mathbb{N}^n \longrightarrow \mathbb{N}$ sont des fonctions partielles μ-récursives, alors $f:\mathbb{N}^n \longrightarrow \mathbb{N}$ définie ci-dessous est une fonction partielle μ-récursive.

$$\begin{array}{l} \text{si } g_1(x_1,\ldots,x_n)\downarrow \ldots g_p(x_1,\ldots,x_n)\downarrow \text{ et } h(g_1(x_1,\ldots,x_n),\ldots,g_p(x_1,\ldots,x_n))\downarrow \\ \quad \text{alors } f(x_1,\ldots,x_n)\downarrow \text{ et } f(x_1,\ldots,x_n)=h(g_1(x_1,\ldots,x_n),\ldots,g_p(x_1,\ldots,x_n)) \\ \text{sinon } f(x_1,\ldots,x_n)\uparrow \end{array}$$

iii. clos par récurrence primitive pour les fonctions partielles si $g:\mathbb{N}^p \longrightarrow \mathbb{N}$ et $h:\mathbb{N}^{p+2} \longrightarrow \mathbb{N}$ sont des fonctions partielles μ-récursives, alors $f:\mathbb{N}^{p+1} \longrightarrow \mathbb{N}$ définie ci-dessous est une fonction partielle μ-récursive.

$$\begin{array}{l} \text{si } g(a_1,\ldots,a_p)\downarrow \text{ alors } f(a_1,\ldots,a_p,0)\downarrow \text{ et } f(a_1,\ldots,a_p,0)=g(a_1,\ldots,a_p) \\ \text{si } f(a_1,\ldots,a_p,x)\downarrow \text{ et } h(a_1,\ldots,a_p,x,f(a_1,\ldots,a_p,x))\downarrow \\ \quad \text{alors } f(a_1,\ldots,a_p,x+1)\downarrow \text{ et } f(a_1,\ldots,a_p,x+1)=h(a_1,\ldots,a_p,x,f(a_1,\ldots,a_p,x)) \\ \text{sinon } f(a_1,\ldots,a_p,x+1)\uparrow \end{array}$$

iv. clos par le schéma de minimisation μ : si $g:\mathbb{N}^{p+1} \longrightarrow \mathbb{N}$ est une fonction partielle μ-récursive, alors la fonction partielle $f:\mathbb{N}^p \longrightarrow \mathbb{N}$ définie ci-dessous notée $f(x_1,\ldots,x_p)=\mu z.g(x_1,\ldots,x_p,z)=0$ est μ-récursive.

$$\begin{array}{l} \text{si } \exists y_0\ \big[g(x_1,\ldots,x_p,y_0)=0 \wedge \forall z<y_0 (g(x_1,\ldots,x_p,z)\downarrow \wedge g(x_1,\ldots,x_p,z)\neq 0)\big] \\ \quad \text{alors } f(x_1,\ldots,x_p)\downarrow \text{ et } f(x_1,\ldots,x_p)=y_0 \\ \quad \text{sinon } f(x_1,\ldots,x_p)\uparrow \end{array}$$

Par exemple la fonction nulle part définie $x \mapsto \mu t.0=1$ est partielle μ-récursive, de même la fonction $x \mapsto \mu t.x=0$, définie seulement en 0.

En appliquant le schéma de minimisation aux fonctions régulières seulement, on évite de parler de fonction partielle.

Définition 1.2.5 (fonction totale μ-récursive) L'ensemble des *fonctions totales μ-récursives* est le plus petit ensemble de fonctions (partout définies) à plusieurs arguments entiers clos par opérations récursives primitives et schéma de minimisation.

Malgré les apparences, cette définition est très différente de celle des fonctions récursives primitives ou de celles des fonctions partielles μ-récursives. En effet, comme il ne s'agit que de fonctions totales, la condition de régularité doit être vérifiée pour chaque utilisation du schéma de minimisation. On peut donc donner une définition plus explicite.

Fait 1.2.6 *L'ensemble des fonctions totales μ-récursives est le plus petit sous-ensemble de $\mathscr{F}$*
- *clos par opérations récursives primitives (voir page 4) ;*
- *tel que si f récursive totale est μ-récursive, f d'arité $k+1$ et f régulière, alors $\overline{x} \mapsto \mu y \cdot \big(f(\overline{x}, y) = 0\big)$, qui est totale, est μ-récursive.*

Or la condition de régularité n'est pas, contrairement aux autres, une condition « syntaxique ». En fait elle ne peut-être vérifiable mécaniquement. On le verra de façon précise ultérieurement, mais l'argument est celui développé en début de cette section : la première des deux conditions de la page 15 ne peut être vérifiée.

Clairement toute fonction totale μ-récursive est une fonction partielle μ-récursive qui est totale. Par composition on peut définir des fonctions partielles μ-récursives qui s'avèrent totales à partir de fonctions qui ne le sont pas, comme par exemple en composant une fonction définie seulement en 0 et la fonction nulle. La réciproque n'est donc pas aussi évidente. Elle sera obtenue par codage du calcul à la section suivante (voir proposition 2.1.2 page 33), ce qui lèvera les ambiguïtés que pourrait susciter ces définitions. Jusqu'à cette proposition, on distingue entre fonction totale μ-récursive au sens de la définition 1.2.5, et fonction partielle μ-récursive (au sens donc de la définition 1.2.4) qui s'avère totale.

Définition 1.2.7 (prédicat décidable) Un prédicat d'arité p sur les entiers, un sous-ensemble de $\mathbb{N}^p$, est dit *décidable* ou *calculable* quand sa fonction caractéristique est calculable, c'est-à-dire μ-récursive avec la définition dont on dispose actuellement (noter qu'une fonction caractéristique est toujours totale).

L'ensemble des fonctions totales μ-récursives étant clos par opérations récursives primitives, non seulement contient les fonctions récursives primitives, mais a également les propriétés de clôture de la section 1.1.2 page 5. Par conséquent l'ensemble des prédicats décidables a aussi les propriétés de clôture de la section 1.1.2 vues pour les prédicats récursifs primitifs.

De façon analogue aux fonctions régulières, on appelle *prédicat régulier* un prédicat P d'arité $p+1$ calculable vérifiant

$$\forall \overline{x} \exists y P\, \overline{x}\, y.$$

et l'ensemble des fonction totales μ-récursives est clos par schéma de minimisation pour les prédicats réguliers, c'est à dire que si $P\,\overline{x}\,y$ est régulier, la fonction f qui à $\overline{x}$ associe le plus petit y vérifiant $P\overline{x}\,y$, notée $\overline{x} \mapsto \mu y \cdot P\,\overline{x}\,y$, qui est totale, est μ-récursive.

On aimerait disposer pour les fonctions partielles calculables de propriétés et schémas de clôture analogues à ceux obtenus pour les fonctions totales calculables (μ-récursives). Une contrainte importante au sujet des fonctions partielles calculables est que dans une composition $f \circ g$ n'est définie en x que si $g(x)$ est définie. Par exemple $x \mapsto f(x) - f(x)$ n'est définie que si f est définie. C'est l'appel par valeurs des langages de programmation fonctionnels. Or cela n'est pas toujours ce qui est attendu. Ainsi on souhaite la propriété de clôture suivante.

Proposition 1.2.8 *Si P est un prédicat décidable d'arité p, et f et g deux fonctions partielles d'arité p μ-récursives, alors la fonction partielle h définie ci-dessous est μ-récursive partielle*

$$\textit{si } P\,\overline{x} \textit{ alors } h(\overline{x}) = f(\overline{x}) \textit{ sinon } h(\overline{x}) = g(\overline{x})\ .$$

Si f et g sont totales on a vu que $h(\overline{x}) = \chi_P(\overline{x})\,f(\overline{x}) + \overline{\mathrm{sg}}(\chi_P(\overline{x}))\,g(\overline{x})$ convient, mais si f et g sont partielles ce n'est plus le cas. En effet quand on a $P\,\overline{x}$ et $f\,\overline{x})\downarrow$, on souhaite que h soit définie indépendamment de

ce qui se passe pour g. Or ce n'est pas le cas pour la fonction h définie ci-dessus si $g(\overline{x})\uparrow$. Il n'y a pas a priori de façon simple de définir la fonction h qui doit vérifier

$$\text{si } P\,\overline{x} \text{ et } f(\overline{x})\downarrow \text{ alors } h(\overline{x})\downarrow\ ;\ \text{ si } \neg P\,\overline{x} \text{ et } g(\overline{x})\downarrow \text{ alors } h(\overline{x})\downarrow\ ;\ \text{ dans les autres cas } h(\overline{x})\uparrow .$$

Démontrer cette proposition à partir de la caractérisation des fonctions μ-récursives n'a rien d'évident. La démonstration se fera facilement après avoir montré l'équivalence avec une caractérisation des fonctions partielles calculables où le calcul est explicite (exercice 14 page 34).

En effet ce formalisme des a été introduit par Kleene non pour son expressivité, mais parce qu'il fournit les primitives pour coder n'importe quelle notion de calcul sur machine, comme on va le constater dans le cas particulier des machines à registres.

1.3 Fonctions calculables par machines à registres

On va formaliser une notion de machine théorique très simple, les machines à registres. Ces machines ont une mémoire constituée d'un nombre fini de *registres* $R_0, R_1, \ldots, R_k$, chaque registre est de taille non bornée et peut donc contenir un entier arbitraire. Elles disposent d'autre part d'un programme qui est une suite finie d'instructions.

L'*état d'une machine* (à un instant donné) est le contenu des registres d'une part, un index de lecture du programme d'autre part. Cet index est un entier inférieur à la longueur du programme qui renvoie à l'instruction qui doit être exécutée : l'index de lecture est donc le numéro de cette instruction dans la suite finie constituant le programme (on choisit de numéroter de 1 en 1 à partir de 0). Chaque instruction agit sur l'état de la machine.

On va voir successivement plusieurs notions de programme, la première notion étant la plus primitive.

1.3.1 Programmes goto

Un *programme* goto est une suite finie constituée de 4 types d'instruction. à partir de 0.

1. incrémenter de 1 le registre numéro i, passer à l'instruction suivante :
$$R_i := R_i + 1$$
2. décrémenter de 1 le registre numéro i, passer à l'instruction suivante :
$$R_i := R_i - 1$$
(quand le registre est déjà à 0 il n'est pas modifié par l'instruction).
3. exécuter un goto conditionnel, si le registre numéro i est nul, aller à l'instruction numéro p (p est un entier inférieur à la longueur du programme) :
$$\text{if } R_i = 0 \text{ goto } p$$
4. une instruction d'arrêt qui apparait une et une seule fois en fin du programme :
$$\text{halt}$$

On considère que la numérotation des instructions est implicite : le numéro désigne la place de l'instruction dans le programme, même si dans les exemples, on l'explicitera pour la clarté. Le calcul d'une telle machine peut ne pas terminer, et l'instruction goto est la seule instruction susceptible de conduire le calcul à ne pas terminer.

Comme on peut accéder directement au registre numéro i : cela modélise plus ou moins la mémoire RAM (random access memory), mémoire à accès aléatoire. Vous verrez une autre notion de machine, les machines de Turing qui utilisent une mémoire à accès séquentiel : une machine de Turing écrit sur un ruban, il faut $|i-j|$ étapes pour aller de la case i à la case j.

Ces machines restent toutefois très théoriques : elles possèdent un nombre fini de registres qui n'est pas borné, de même que la taille de chaque registre, et la taille du programme.

Il manque par ailleurs des instructions essentielles pour manipuler les adresses de registres, même si elles ne permettraient pas de calculer de nouvelles fonctions (et ici le nombre fixé de registres rend ses instructions inutiles).

Définition 1.3.1 Une fonction partielle f de $\mathbb{N}^n \to \mathbb{N}$ est calculable par une machine M à k registres, signifie que quand on initialise la machine en affectant les entiers $x_1, \ldots, x_{\inf(n,k)}$ aux registres $R_1, \ldots, R_{\inf(n,k)}$, et la valeur 0 aux registres restant (s'il en existe), l'index de lecture étant à 0 (sur la première instruction), alors la machine termine son calcul si et seulement si $f(x_1, \ldots, x_n)\downarrow$, et dans ce cas dans le registre R_0 a la valeur $f(x_1, \ldots, x_n)$ à la fin du calcul.

On n'a pas supposé ci-dessus dans la définition que $n \leq k$, mais évidemment :

Lemme 1.3.2 *Si $f : \mathbb{N}^n \to \mathbb{N}$ est calculable par une machine à k registres, alors f est calculable par une machine à k registres avec $k \geq n+1$.*

Démonstration. En effet, si $k < n+1$, il suffit d'ajouter les registres manquant $R_k, \ldots, R_n$, qui ne seront pas modifiés lors du calcul. ■

Voyons quelques exemples.

La machine à un seul registre R_0 et dont le programme contient pour seule instruction halt calcule les fonctions constantes égales à 0, : $x_1 \ldots x_n \mapsto 0$.

$$\boxed{R_0} \quad 0 \quad halt$$

On calcule l'addition avec la machine à 4 registres dont le programme est le suivant :

$\boxed{R_0}\boxed{R_1}\boxed{R_2}\boxed{R_3}$

0 if $R_1 = 0$ goto 4
1 $R_1 := R_1 - 1$
2 $R_0 := R_0 + 1$
3 if $R_3 = 0$ goto 0
4 if $R_2 = 0$ goto 8
5 $R_2 := R_2 - 1$
6 $R_0 := R_0 + 1$
7 if $R_3 = 0$ goto 4
8 halt

Le registre R_3 ne sert qu'à pouvoir écrire un goto inconditionnel, instruction que l'on pourrait donc employer, sachant qu'on peut la simuler en ajoutant un registre à la machine. On va dans un premier temps montrer que l'on peut ainsi simuler quelques nouvelles instructions utiles.

Exercice 10

1. Décrire des machines qui calculent les fonctions sg et $\overline{\text{sg}}$.
2. Décrire une machine dont le calcul termine en 0 sur 0, et qui ne termine pas pour tout autre entier.

1.3.2 De nouvelles instructions

Proposition 1.3.3 *Si une fonction partielle de $\mathbb{N}^n \to \mathbb{N}$ est calculée par une machine à registres utilisant en plus des instructions usuelles l'une des instructions suivantes*

1. *le* goto *inconditionnel, aller à la ligne p :*

   goto p

2. *le registre numéro i est mis à* 0 *et l'on passe à l'instruction suivante :*

   $R_i := 0$

3. *l'assignation d'un entier, le registre numéro i reçoit le contenu du registre numéro j $(j \neq i)$ et l'on passe à l'instruction suivante :*
$$R_i := R_j$$

alors elle est calculable par une machine à registres usuelle.

Démonstration. On construit à chaque fois une machine avec programme goto qui simule une machine avec programme enrichi avec la nouvelle instruction.

1. goto p : on a vu qu'il suffisait d'ajouter un registre dont le contenu sera toujours nul, on suppose $n \leq k$ (Lemme 1.3.2), on ajoute alors un registre R_{k+1} et l'instruction devient if $R_{k+1} = 0$ goto p.
2. $R_i := 0$: la machine conserve les mêmes registres. On suppose que l'instruction $R_i := 0$ est à la place l. On remplace l'instruction $R_i := 0$ par la suite d'instructions :
$$\begin{array}{ll} l & \text{if } R_i = 0 \text{ goto } l+3 \\ l+1 & R_i := R_i - 1 \\ l+2 & \text{goto } l \end{array}$$
et dans le reste du programme on décale de 2 tous les goto l' pour $l' > l$ (remplacés par goto $l'+2$).
3. $R_i := R_j$: on suppose $n \leq k$ (Lemme 1.3.2), on ajoute alors un registre R_{k+1}. On suppose que l'instruction $R_i := R_j$ est à la place l. On remplace l'instruction $R_i := 0$ par la suite d'instructions :
$$\begin{array}{ll} l & R_i := 0 \\ l+1 & \text{if } R_j = 0 \text{ goto } l+6 \\ l+2 & R_j := R_j - 1 \\ l+3 & R_i := R_i + 1 \\ l+4 & R_{k+1} := R_{k+1} + 1 \\ l+5 & \text{goto } l+1 \\ l+6 & \text{if } R_{k+1} = 0 \text{ goto } l+10 \\ l+7 & R_{k+1} := R_{k+1} - 1 \\ l+8 & R_j := R_j + 1 \\ l+9 & \text{goto } l+6 \end{array}$$
et dans le reste du programme on décale de 9 les goto l' pour $l' > l$. ■

Par exemple, avec ces nouvelles instructions, la projection p_i^n est calculée par une machine à n registres de programme
$$\begin{array}{l} R_0 := R_i \\ \text{halt} \end{array}$$
la fonction successeur est calculée par la machine à 2 registres de programme :
$$\begin{array}{l} R_0 := R_1 \\ R_0 := R_0 + 1 \\ \text{halt} \end{array}$$
Ces fonctions sont donc calculables par une machine à registres avec programme goto.

1.3.3 Programmes structurés

On peut préferer utiliser des instructions plus complexes qui permettent de structurer les programmes.

Définition 1.3.4 Un *programme structuré* est une suite d'instructions, chaque instruction pouvant être (définition inductive) :

1. l'une des instructions d'assignation déjà décrites
$$R_i := 0,\ R_i := R_i + 1,\ R_i := R_i - 1,\ R_i := R_j\ (j \neq i)$$

2. une séquence d'instructions, elles sont exécutées séquentiellement, puis le programme passe à l'instruction suivante :

$$\text{begin } S_1; \ldots; S_p \text{ end}$$

3. une instruction "while", une boucle qui répète une instruction S donnée

$$\text{while } R_i \neq 0 \text{ do } S$$

l'instruction S est exécutée tant que le contenu du registre numéro i n'est pas nul, s'il est nul on passe à l'instruction suivante.

4. une instruction "for", une boucle de répétition bornée d'une instruction S donnée

$$\text{for } i = 1 \text{ to } R_j \text{ do } S$$

L'instruction S est exécutée un nombre de fois égal à l'entier contenu dans le registre R_j *avant exécution de ces instructions*, puis l'on passe à l'instruction suivante. Même si le registre R_j est modifié par l'instruction S, le nombre de répétitions de l'instruction n'est pas modifié.

On appelle programme while les programmes structurés qui n'utilisent pas l'instruction for.

L'exécution d'une instruction est plus complexe que pour un programme goto : chaque instruction peut avoir en fait la même complexité qu'un programme. En particulier on peut avoir imbrication des while et des for. L'instruction halt est devenue inutile : les instructions du programme sont exécutées l'une après l'autre, mais la machine ne termine pas toujours son calcul pour autant, puisqu'une boucle while peut ne pas terminer. C'est d'ailleurs la seule instruction susceptible de conduire le programme à ne pas terminer.

Les fonctions partielles calculables sur machines à registres avec programme structuré ou programme while se définissent de la même façon qu'au paragraphe précédent, on suppose de plus que la machine a toujours au moins autant de registres que n l'arité de la fonction.

Lemme 1.3.5 *Si une fonction de $\mathbb{N}^n \to \mathbb{N}$ est calculée par une machine avec programme structuré, elle est calculable par une machine avec programme* while.

Démonstration. On suppose $n \leq k$. On procède par induction sur la définition des instructions (les boucles for peuvent être imbriquées). On montre le résultat en simulant chaque instruction d'un programme structuré sur une machine M à k registres par une instruction ou une suite d'instructions sur une machine à $k+s$ registres, où s est le nombre d'instructions for dans le programme. À chaque instruction for est associé de façon univoque un registre R_{k+f}, $1 \leq f \leq s$. On suppose que S est simulée par une séquence d'instructions sans for $\mathscr{S}$, et on simule l'instruction

$$\text{for } i = 1 \text{ to } R_j \text{ do } S$$

par la séquence :

$$R_{k+f} := R_j$$
$$\text{while } R_{k+f} \neq 0 \text{ begin } R_{k+f} := R_{k+f} - 1; \mathscr{S} \text{ end}$$

On doit bien recopier le registre R_j pour le laisser dans le même état à la fin du calcul. D'autre part la suite d'instructions $\mathscr{S}$ doit laisser le registre R_{k+f} dans le même état. ■

Proposition 1.3.6 *Si une fonction de $\mathbb{N}^n \to \mathbb{N}$ est calculée par une machine à registres avec programme structuré, elle est calculable par une machine à registres avec programme* goto.

Démonstration. D'après le lemme il suffit de le montrer pour les programmes while. On le montre par induction. Il suffit de simuler chaque instruction d'un programme while par une séquence d'instructions avec goto.

1. Les instructions d'assignations sont des instructions de base des machines à registres ou sont simulables d'après la proposition 1.3.3.

2. On suppose que les instructions $S_1,\dots,S_p$ sont simulées par les suites d'instructions $\mathscr{S}_1,\dots,\mathscr{S}_p$, appelons $\mathscr{S}$ la suite obtenue en les concaténant. Alors la séquence

$$\text{begin } S_1;\dots;S_p \text{ end}$$

est simulée par la suite d'instructions $\mathscr{S}$.

3. On suppose que l'instruction S est simulée par la suite d'instructions $\mathscr{S}$ de longueur s. Alors l'instruction

$$\text{while } R_i \neq 0 \text{ do } S$$

est simulée par la séquence (les lignes sont numérotées pour que ce soit plus clair) :

$$\begin{array}{rl}
\vdots & \vdots \\
l & \text{if } R_i = 0 \text{ goto } l+(s+2) \\
\vdots & \mathscr{S} \\
l+s+1 & \text{goto } l \\
l+s+2 & \dots \\
\vdots & \vdots
\end{array}$$

Via la simulation, les programmes structurés apparaissent comme des cas particuliers de programme goto, au sens où ils restreignent l'usage de l'instruction goto. En fait les machines avec programmes goto et programmes structurés calculent la même classe de fonctions.

Cette équivalence apparaitra aussi comme conséquence d'un résultat ultérieur : on montrera que cette classe des fonctions est celle de toutes les fonctions μ-récursives partielles.

Pour la preuve directe, on utilisera le lemme suivant qui étend le jeu d'instructions utilisable par un programme structuré.

Lemme 1.3.7 *Soit une fonction partielle de $\mathbb{N}^n \to \mathbb{N}$ calculable par une machine à registres avec programme structuré utilisant en plus l'instruction*

$$\text{if } R_i = 0 \text{ then } S_1 \text{ else } S_2$$

qui exécute l'instruction S_1 ou S_2 suivant le résultat du test à zéro de R_i. Alors f est calculable par une machine à registres avec programme structuré.

Démonstration. On montre le résultat par induction sur le nombre d'instructions if dans le programme. Pour chaque instruction if, on a besoin de deux nouveaux registres que l'on ajoute à la machine.

Soit M une machine calculant f et utilisant if. On chosit une instruction :

$$\text{if } R_i = 0 \text{ then } S_1 \text{ else } S_2$$

telle que S_1 et S_2 ne contiennent pas de if. soient C_1 et C_2 les deux nouveaux registres associés. Alors cette instruction est simplement remplacée par

```
begin
  C1 := 1;
  C2 := 1;
  for j = 1 to Ri do C1 := 0;
  for j = 1 to C1 do begin S1; C2 := 0 end;
  for j = 1 to C2 do S2;
end
```

■

Proposition 1.3.8 *Si une fonction de $\mathbb{N}^n \to \mathbb{N}$ est calculable par une machine à registres avec programme* goto, *elle est calculable par machine à registres avec programme structuré.*

On obtient ce résultat comme conséquence de ceux obtenus par codage à la section 1.5 (en particulier la proposition 1.5.5). Cependant la démonstration directe qui suit est informative en soi.

Démonstration. Soit $f : \mathbb{N}^n \to \mathbb{N}$ une fonction calculable par une machine à registre avec programme goto M dont la séquence d'instructions est $S_1, ..., S_p$. On suppose que M utilise au plus k registres avec $k \geq n$. On suppose sans perte de généralité que S_p est l'instruction halt. On va construire un programme structuré N à $k+p$ registres calculant la fonction f. On appelera $I_1, ..., I_p$ les p registres supplémentaires. La signification de $I_i = 0$ est que l'instruction courante du programme est le numéro i. Un seul registre I_i, $i \leq p$, peut être à 0 à un instant donné du temps. Tous les autres doivent avoir la valeur 1.

$I_1 := 0; I_2 := 1; ...; I_p := 1$
while $I_p \neq 0$ do
  if $I_1 = 0$ then $\tilde{S}_1$
  else if $I_2 = 0$ then $\tilde{S}_2$
  $\vdots$
  else if $I_{p-1} = 0$ then $\tilde{S}_{p-1}$

où chaque instruction $\tilde{S}_i$ se définit à partir de S_i comme l'indique le tableau suivant.

S_i	$\tilde{S}_i$
$R_j := R_j + 1$	begin $R_j := R_j + 1; I_{i+1} := 0; I_i := 1$ end
$R_j := R_j - 1$	begin $R_j := R_j - 1; I_{i+1} := 0; I_i := 1$ end

if $R_j = 0$ goto p	begin
	  if $R_j = 0$ then $I_p := 0$ else $I_{i+1} := 0$;
	  $I_i := 1$
	end

■

Le résultat précédent présente une sorte de forme normale de programme structuré (« une boucle suffit »). Toutefois, le programme structuré fourni par la preuve est construit en fonction du programme simulé, et dépend donc de celui-ci. On verra un résultat plus fort : on peut obtenir un programme structuré de la même forme qui est « universel » pour toutes les machines, la machine étant passée en argument via codage.

Le lemme suivant sera commode pour montrer que les fonctions μ-récursives sont calculable par machine.

Définition 1.3.9 Une machine est dite *propre* quand elle termine son calcul avec tous ses registres à l'exception du registre de sortie R_0 dans le même état qu'au début du calcul.

Lemme 1.3.10 *Si une fonction est calculable par machine à registres avec programme structuré, alors elle est calculable par une machine propre (avec programme structuré).*

Démonstration. On suppose, en ajoutant éventuellement des registres, que le nombre initial de ceux-ci est supérieur ou égal au nombre d'arguments de la fonction calculée. Il suffit de copier les registres (sauf celui de sortie), de calculer sur ces copies sans toucher aux registres initiaux, puis de remettre les copies à 0. Soit M une machine à $k+1$ registres $R_0, \ldots, R_k$, dont le programme est une suite d'instructions $(S_1, \ldots, S_s)$ qui calcule f. On construit une machine propre à $2k+1$ registres $R_0, \ldots, R_k, R'_1, \ldots, R'_k$ (R'_i pour R_{i+k}) de la façon suivante :

$R'_1 := R_1$
$\vdots$
$R'_k := R_k$
$S_1[R'_i/R_i]$
$\vdots$
$S_s[R'_s/R_s]$
$R'_1 := 0$
$\vdots$
$R'_k = 0$

■

On ne se servira pas de ce lemme pour les machines avec programme goto, mais il reste valide : il faudrait alors décaler les goto dans les instructions supplémentaires.

Exercice 11

1. Simuler directement l'instruction for par un programme goto.
2. Simuler directement l'instruction

   repeat S until $R_i = 0$

   par un programme goto.

1.4 Les fonctions μ-récursives sont calculables par machines

On montre maintenant comment calculer n'importe quelle fonction partielle μ-récursive par un programme structuré.

1.4.1 Les fonctions partielles μ-récursives

Proposition 1.4.1 *Les fonction partielles μ-récursives sont calculables par machine à registres avec programme structuré. Par conséquent elles sont calculables par machine à registres avec programme* while *et programme* goto.

Démonstration. On procède par induction sur la définition des fonctions partielles μ-récursives. On a déjà traité en exemple les fonctions de base, la fonction nulle $\lambda x.0$, les projections p_k^i et la fonction successeur. On vérifie que les instruction utilisées sont bien celles des programmes structurés.

composition Supposons que les fonctions partielles $g_1,\dots,g_k : \mathbb{N}^n \to \mathbb{N}$ et $h : \mathbb{N}^k \to \mathbb{N}$ sont calculables par des machines $M_1,\dots,M_k$ et M_0, dont les programmes sont les suites d'instructions $\mathscr{S}_1,\dots,\mathscr{S}_k$ et $\mathscr{S}_0$. On suppose ces machines propres d'après le lemme 1.3.10. On suppose que chacune de ces machines utilisent au plus $m+1$ registres, et que $m \geq n$ et $m \geq k$. On construit une machine M qui utilise $m+k+1$ registres que l'on va noter $R_0, R_1,\dots,R_m, H_1,\dots,H_k$. Elle va calculer successivement $g_1(x_1,\dots,x_n)$,...,$g_k(x_1,\dots,x_n)$ dont elle va stocker les valeurs dans les registres $H_1,\dots,H_k$, puis calculer $f(H_1,\dots,H_k)$. Voici le programme de cette machine

$$
\begin{array}{l}
\mathscr{S}_1 \\
H_1 := R_0 \\
R_0 := 0 \\
\vdots \\
\mathscr{S}_k \\
H_k := R_0 \\
R_0 := 0 \\
R_1 := H_1 \\
\vdots \\
R_k := H_k \\
R_{k+1} := 0 \\
\vdots \\
R_m := 0 \\
\mathscr{S}_0
\end{array}
$$

Remarquez que dès que l'une des machines $M_1,\dots,M_k$ ne termine pas la machine M ne termine pas.

Récurrence primitive Soit M_1 une machine propre à k_1+1 registres dont le programme est $\mathscr{S}_1$ et qui calcule la fonction partielle $g : \mathbb{N}^n \to \mathbb{N}$ $(n \leq k_1)$. Soit M_2 une machine propre à k_2+1 registres dont le programme est $\mathscr{S}_2$ et qui calcule la fonction partielle $h : \mathbb{N}^{n+2} \to \mathbb{N}$ $(n+2 \leq k_2)$. Soit un

entier m tel que $m \geq k_1$ et $m \geq k_2$. La fonction f est définie par récurrence primitive à partir de g et h :

$$f(\overline{a},0) = g(\overline{a})$$
$$f(\overline{a},x+1) = h(\overline{a},x,f(\overline{a},x))\,.$$

La fonction f est calculée par une machine à $m+2$ registres dont voici les instructions :

$R_{m+1} := R_{n+1}$
$R_{n+1} := 0$
$\mathscr{S}_1$
for $i = 1$ to R_{m+1} do begin $R_{n+2} := R_0$; $R_0 := 0$; $\mathscr{S}_2$; $R_{n+1} := R_{n+1} + 1$ end

On calcule donc successivement $f(\overline{a},0)$ (avant la boucle for) puis dans l'ordre $f(\overline{a},1),\ldots,f(\overline{a},n)$. Dès que l'une de ces valeurs n'est pas définie, la machine ne termine pas, ce qui est bien le comportement souhaité.

Minimisation Soit M_0 une machine propre à $k+1$ registres dont le programme est $\mathscr{S}_0$ qui calcule la fonction partielle $g : \mathbb{N}^{n+1} \to \mathbb{N}$, avec $n+1 \leq k$. La fonction f :

$$f(x_1,\ldots,x_n) = \mu t.[g(x_1,\ldots,x_n,t) = 0]$$

est calculée par la machine à $k+1$ registres suivante :

$R_0 := R_0 + 1$
while $R_0 \neq 0$ do begin $R_0 := 0$; $\mathscr{S}_0$; $R_{n+1} := R_{n+1} + 1$ end
$R_{n+1} := R_{n+1} - 1$
$R_0 := R_{n+1}$ ■

1.4.2 Les fonctions récursives primitives

Si on examine la preuve, on se rend facilement compte que le while n'intervient que pour la minimisation. Le for suffit pour la récurrence primitive. Appelons programme for les programmes structurés qui n'utilisent pas de while, on a donc :

Proposition. Les fonctions récursives primitives sont calculables par programme for.

La réciproque de ce résultat est vraie, c'est-à-dire que :

Proposition 1.4.2 *Une fonction est récursive primitive si et seulement si elle est calculable par programme* for.

Il faut bien remarquer que les boucles for que nous avons définies ne peuvent modifier la variable d'itération lors du calcul, alors que c'est possible par exemple dans le langage C. Sans cette condition le résultat est manifestement faux (le programme ne termine pas forcément).

Exercice 12 Terminer la démontration de la proposition précédente, soit si une fonction est calculable par programme for, elle est récursive primitive. Procéder par induction sur le niveau d'imbrication des boucles for. Montrer que le contenu de chaque registre de la machine est une fonction récursive primitive des entrées en utilisant le schéma de récurrences mutuelles (exercice 5 page 10).

1.5 Les fonctions calculables par machine sont μ-récursives

Nous allons maintenant montrer que les fonctions partielles calculables par machine goto sont μ-récursives. La méthode consiste à coder par des entiers les configurations d'une machine quelconque (l'état de la machine à un instant donné), et à réfléter dans l'arithmétique par des fonctions μ-récursives le fonctionnement de cette machine. On parle parfois d'*arithmétisation*. Le premier exemple d'arithmétisation est celui des formules de l'arithmétique par Gödel dans son article paru en 1931.

Là on va arithmétiser plus que nécessaire pour le résultat annoncé : en codant également les machines par des entiers, plutôt que de montrer l'existence d'une fonction μ-récursive ad hoc décrivant le

fonctionnement de chaque machine, on montre que ce fonctionnement se décrit de façon uniforme en prenant le code de la machine en argument. La démonstration n'est pas plus compliquée, et cette arithmétisation plus poussée conduit aux résultats du chapitre suivant, en particulier à l'existence d'une machine universelle, une machine qui peut simuler n'importe quelle machine (disons en fixant le nombre d'entrées) en lui fournissant le bon argument.

1.5.1 Machine, état d'une machine

Une machine est déterminée par :

1. son nombre de registres ;
2. son programme qui est une suite finie d'instructions.

L'état d'une machine est déterminée par

1. un index de lecture (entier) : le numéro de l'instruction à exécuter ;
2. la suite finie des contenus des registres.

Le code d'une machine ou d'un état de machine est un entier, la fonction de codage doit être injective, mais pas nécessairement surjective.

Préliminaires et notations sur les fonctions récursives primitives

On va utiliser les différents codages récursifs primitifs des couples, k-uplets et listes (suite finies) définis à la section 1.1.4 page 7, ainsi que des fonctions usuelles sur ceux-ci. Les arguments de la fonction $\mathrm{nth}(l,i)$ qui calcule le i-ème élément de la suite de code l sont notés dans cet ordre.

On utilisera maintenant systématiquement $\langle , \rangle$ pour les bijections de Cantor α_k :

$$\alpha_k(x_1,\dots,x_k) = \langle x_1,\dots,x_k\rangle \ .$$

Par définition des α_i :

$$\langle x_1, \langle x_2,\dots,x_k\rangle\rangle = \langle x_1,\dots,x_k\rangle$$

et les projections vérifient $\pi_1 = \pi_1^2 = \pi_1^3 = \dots$ et plus généralement :

$$\pi_i^n = \pi_i^m \text{ pour } i < n \leq m$$

Comme α_k est une bijection on obtient une définition correcte de fonction en écrivant :

$$f(x_1,\dots,x_{i-1},\langle y_1,\dots,y_k\rangle,x_{i+1},\dots,x_n) = h(x_1,\dots,x_{i_1},y_1,\dots,y_k,x_{i+1},\dots,x_n)$$

et si h est récursive primitive alors f est récursive primitive. En effet cette définition équivaut à :

$$f(x_1,\dots,x_n) = h(x_1,\dots,x_{i-1},\pi_1^k(x_i),\dots,\pi_k^k(x_i),x_{i+1},\dots,x_n) \ .$$

Codage des instructions

On a quatre sortes d'instructions, chacune pouvant être paramétrée par un entier (le numéro de registre pour l'incrémentation et la décrémentation), ou un couple d'entiers (le numéro de registre et le numéro de ligne pour le goto conditionnel). On va donc coder chaque instruction par le code d'un couple, on note $\ulcorner S \urcorner$ l'entier qui code l'instruction S :

$$\begin{aligned}
\ulcorner R_i := R_i + 1 \urcorner &= \langle 0, i\rangle \\
\ulcorner R_i := R_i - 1 \urcorner &= \langle 1, i\rangle \\
\ulcorner \text{if } R_i = 0 \text{ goto } n \urcorner &= \langle 2, \langle i, n\rangle\rangle (= \langle 2, i, n\rangle) \\
\ulcorner \mathsf{halt} \urcorner &= \langle 3, 0\rangle
\end{aligned}$$

De fait, le codage n'a pas besoin d'être fonctionnel, et nous considérerons parfois dans la suite que tout entier c tel que $\pi_1^2(c) \geq 3$ est le code d'une instruction halt.

Codage des machines

Le code m d'une machine à $k+1$ registres de programme $S_0,\ldots,S_s$ est l'entier :

$$m = \langle k, [\ulcorner S_1 \urcorner;\ldots;\ulcorner S_s \urcorner]\rangle.$$

Une machine possède au moins un registre, la première projection est donc le nombre de registres moins 1. Le codage est évidemment injectif, mais pas surjectif, ce qui n'est pas gênant. On n'aura même pas besoin du résultat de l'exercice suivant.

Exercice 13 Montrer que l'ensemble des entiers qui sont des codes de machine est récursif primitif.

Codage de l'état d'une machine

On code l'état d'une machine dont les contenus des registres sont dans l'ordre $R_0, R_1, \ldots, R_k$ et l'index de lecture est i par l'entier :

$$e = \langle i, [R_0; R_1; \ldots; R_k]\rangle$$

1.5.2 Le calcul

Supposons que la machine de code m prenne en entrée n entiers. Pour coder le calcul, il nous faut essentiellement les deux fonctions et le prédicat suivant :

— Une fonction d'initialisation $\text{init}_n : \mathbb{N}^{n+1} \to \mathbb{N}$. Cette fonction calcule en fonction du code m de la machine et des entrées $x_1,\ldots,x_n$ l'état initial de cette machine (au départ du calcul).

— Une fonction de transition $\text{tr} : \mathbb{N}^2 \to \mathbb{N}$. Cette fonction calcule en fonction du code m d'une machine et de l'état de code e de cette machine l'état de la machine après une étape de calcul à partir de l'état de code e.

— Le prédicat Halt de terminaison à deux arguments est vrai pour les couples (m,e) où l'état codé par e indique que la machine codée par m est en fin de calcul.

Remarquez qu'il ne s'agit pas vraiment de définitions : on ne demande rien si les entrées ne sont pas cohérentes, ce qui est le cas par exemple pour ces fonctions et ce prédicat quand la première entrée n'est pas le code d'une machine, ou encore pour la fonction de transition et le prédicat de terminaison quand la seconde entrée n'est pas le code d'un état possible de la machine codée par la première entrée.

Il est intuitivement clair qu'on peut calculer de façon mécanique deux fonctions et un prédicat convenables. On montre explicitement dans les sections qui suivent qu'on peut les définir de façon récursive primitive.

Fonction d'initialisation

On définit d'abord une fonction auxiliaire qui crée en fonction de k la liste des $k+1$ premiers éléments de la suite infinie $0, x_1, \ldots, x_n, 0, \ldots$. Ensuite init_n construit l'état initial de la machine de code m, à savoir un index de lecture à 0, et si celle-ci possède $k+1$ registres ($k = \pi_1^2(m)$), le code de la liste de longueur $k+1$ $[0; x_1; \ldots; x_n; 0; \ldots; 0]$ (liste éventuellement tronquée si $k < n$) :

$$\begin{aligned}
&aux(x_1,\ldots,x_n,0) = [0] = 0 :: []\\
&aux(x_1,\ldots,x_n,1) = [0;x_1] = 0 :: x_1 :: []\\
&\vdots\\
&aux(x_1,\ldots,x_n,n) = [0;x_1;\ldots;x_n] = 0 :: x_1 :: \ldots :: x_n :: []\\
&aux(x_1,\ldots,x_n,n+r+1) = aux(x_1,\ldots,x_n,n+r)@[0]\\
&\text{init}_n(\langle k,s\rangle, x_1,\ldots,x_n) = \langle 0, aux(x_1,\ldots,x_n,k)\rangle
\end{aligned}$$

Lemme 1.5.1 *la fonction* $\text{init}_n(m,x_1,\ldots,x_n)$ *définie ci-dessus est récursive primitive. Elle calcule, quand m est le code d'une machine, l'état initial de la machine de code m avec $x_1,\ldots,x_n$ en entrées.*

Démonstration. La fonction se comporte comme souhaité si m est le code d'une machine. Le schéma de récurrence utilisé est récursif primitif. En effet on le ramène à une récurrence primitive en utilisant dans l'étape de récurrence n fois le «si ... alors ... sinon ...» sur des conditions récursives primitives (singleton). Notez bien que n est une constante. ■

Fonction de transition

On va avoir besoin pour écrire la fonction de transition de modifier le registre numéro i, ce qui demande, pour le codage considéré, de modifier le i-ème élément d'une liste.

Proposition 1.5.2 *Les fonctions* inc, *et* dec *qui, appliquées à un entier i et au code d'une liste non vide, incrémente de* 1 *le $i+1$-ème élément de la liste pour* inc, *respectivement décrémente de* 1 *ce $i+1$-ème élément pour* dec, *et qui ne changent pas leur argument sinon, sont récursives primitives.*

Démonstration. On peut utiliser le schéma de récurrence avec substitution de paramètre (voir exercice 7 page 11), on a :

$$\begin{array}{ll} \text{si } \mathrm{len}(l) \leq n & \text{alors } \mathrm{inc}(n,l) = l \\ & \text{sinon} \\ & \quad \mathrm{inc}(0,l) = (\mathrm{hd}(l)+1) :: \mathrm{tl}(l) \\ & \quad \mathrm{inc}(n+1,l) = \mathrm{hd}(l) :: \mathrm{inc}(n, tl(l)) \end{array}$$

De même pour dec. ■

Par ailleurs on peut calculer de façon récursive primitive l'instruction courante : $\mathrm{inst}(m,e)$ est le code de l'instruction de la machine de code $m = \langle n,s\rangle$ auquel renvoie l'index de lecture indiqué par l'état de code $e = \langle i,r\rangle$ (si l'index de lecture i est supérieur au nombre d'instructions, on considère qu'il renvoie à la dernière instruction) :

$$\mathrm{inst}(\langle n,s\rangle, \langle i,r\rangle) = \mathrm{nth}(s, \mathrm{inf}(i, \mathrm{len}(s))) .$$

On décompose maintenant la fonction de transition en ses deux projections, soient tr_1 et tr_2, et on montre que l'on peut définir chacune de façon récursive primitive.

— $\mathrm{tr}_1(m,e)$ donne le numéro de l'instruction après exécution de l'instruction indiquée par $e = \langle i,r\rangle$:

$$\begin{array}{lll} \text{si } \pi_1^2(\mathrm{inst}(m,\langle i,r\rangle)) \in \{0,1\} & & \text{si incrémentation ou décrémentation} \\ \quad \text{alors } \mathrm{tr}_1(m,\langle i,r\rangle) = i+1 & & \text{on passe à l'instruction suivante} \\ \quad \text{sinon si } \pi_1^2(\mathrm{inst}(m,\langle i,r\rangle)) = 2 & & \text{si goto} \\ \qquad \text{alors si } \mathrm{nth}(r, \pi_2^3(\mathrm{inst}(m,\langle i,r\rangle))) = 0 & & \text{si } R_j = 0 \\ \qquad\quad \text{alors } \mathrm{tr}_1(m,\langle i,r\rangle) = \pi_3^3(\mathrm{inst}(m,\langle i,r\rangle)) & & \text{aller à la ligne indiquée} \\ \qquad\quad \text{sinon } \mathrm{tr}_1(m,\langle i,r\rangle) = i+1 & & \text{sinon on passe à l'instruction suivante} \\ \qquad \text{sinon } \mathrm{tr}_1(m,\langle i,r\rangle) = i & & \text{sinon (halt) arrêt} \end{array}$$

— $\mathrm{tr}_2(m,e)$ donne la liste des contenus des registres après exécution de l'instruction indiquée par $e = \langle i,r\rangle$:

$$\begin{array}{lll} \text{si } \pi_1^2(\mathrm{inst}(m,\langle i,r\rangle)) = 0 & & \text{si incrémentation} \\ \quad \text{alors } \mathrm{tr}_2(m,\langle i,r\rangle) = \mathrm{inc}(\pi_2^2(\mathrm{inst}(m,\langle i,r\rangle)), r) & & R_j := R_j + 1 \\ \quad \text{sinon si } \pi_1^2(\mathrm{inst}(m,\langle i,r\rangle)) = 1 & & \text{si décrémentation} \\ \qquad \text{alors } \mathrm{tr}_2(m,\langle i,r\rangle) = \mathrm{dec}(\pi_2^2(\mathrm{inst}(m,\langle i,r\rangle)), r) & & R_j := R_j - 1 \\ \qquad \text{sinon } \mathrm{tr}_2(m,\langle i,r\rangle) = r & & \text{sinon (goto ou halt) les registres ne sont pas modifiés} \end{array}$$

Les deux fonctions tr_1 et tr_2 sont bien définies de façon récursive primitive, la fonction de transition

$$\mathrm{tr}(m,e) = \langle \mathrm{tr}_1(m,e), \mathrm{tr}_2(m,e)\rangle$$

est donc récursive primitive. Résumons les résultats de cette section.

Lemme 1.5.3 *Les fonctions* inst *et* tr *définies ci-dessus sont primitives récursives. Quand m est le code d'une machine M et e le code d'un état E possible pour M,* $\mathrm{inst}(m,e)$ *est le code de l'instruction courante du programme de M selon l'état E,* $\mathrm{tr}(m,e)$ *est le code de l'état de M obtenu après exécution de cette instruction.*

Prédicat de terminaison

On rappelle que l'état d'une machine est terminal si l'instruction indiquée par l'état est l'instruction halt. On peut donc définir le prédicat de terminaison Halt ainsi :

$$\mathrm{Halt}[m,e] \equiv \pi_1^2(\mathrm{inst}(m,e)) > 2$$

Lemme 1.5.4 *Le prédicat binaire* Halt *défini ci-dessus est récursif primitif. Il est vrai pour les couples* (m,e) *où l'état codé par e indique que la machine codée par m est en fin de calcul, quand m est le code d'une machine, et e le code d'un état pour m.*

Les entiers qui ne sont pas des codes de machines

Nous avons défini ci-dessus les fonctions d'initialisation et de transition et le prédicat de terminaison pour tous les entiers. Parmi ceux-ci, certains ne codent pas de machine. Cela n'a pas d'importance : ces fonctions décrivent de toute façon un comportement mécanique, et finalement nous allons montrer que les fonctions partielles calculables par une classe plus étendue (au moins en apparence) de machines sont μ-récursives.

Nous décrivons maintenant le comportement de ces « machines » étendues, ce qui ne sera vraiment utile qu'au paragraphe 2.2.3.

Pour un entier donné codant une machine, le nombre de registres est bien défini ainsi que le programme comme suite, il se peut simplement que des éléments de la suite ne correspondent pas à des instructions valides.

Nous avons déjà supposé (Paragraphe 1.5.1) que tout entier c tel que $\pi_1^2(c) > 2$ codait une instruction halt, ce qui correspond bien au choix du prédicat de terminaison. La machine peut avoir des instructions halt n'importe où dans le programme, et la dernière instruction n'est pas forcément un halt, ce qui généralise un peu la notion initiale. La machine s'arrête dès qu'elle atteint un halt. Si la dernière instruction est exécutée et que ce n'est pas un halt ou un goto, alors cette instruction est répétée indéfiniment (voir définition de inst) : la machine ne s'arrête pas.

Les instructions d'incrémentation et de décrémentation ($\pi_1^2(c) = 0,1$) pourraient concerner un numéro de registre qui n'apparait pas dans la machine. Dans ce cas on considère que l'instruction ne fait rien en dehors de passer à l'instruction suivante. C'est bien ce que code tr (voir la définition des fonctions inc et dec).

La condition de l'instruction goto peut porter sur un registre qui n'apparait pas dans la machine : on le considère comme un goto inconditionnel (voir définition de tr_1). Elle peut renvoyer à une ligne de programme d'index supérieur à la longueur du programme. Dans ce cas cela revient à renvoyer à la dernière ligne du programme (voir définition de inst).

Temps de calcul

On a tout ce qu'il faut maintenant pour définir en fonction du code de la machine et des entrées le temps de calcul, c'est à dire le nombre d'étapes jusqu'à ce que la machine termine. Il se calcule par minimisation, et peut ne pas être défini, si la machine ne termine pas.

On définit d'abord de façon récursive primitive la fonction $\mathrm{st}_n(m,x_1,\dots,x_n,t)$ qui calcule l'état de la machine de code m au bout de t étapes de calcul avec les entrées $x_1,\dots,x_n$:

$$\begin{aligned}&\mathrm{st}_n(m,x_1,\dots,x_n,0) = \mathrm{init}_n(m,x_1,\dots,x_n)\\&\mathrm{st}_n(m,x_1,\dots,x_n,t+1) = \mathrm{tr}(m,\mathrm{st}_n(m,x_1,\dots,x_n,t))\end{aligned}$$

Enfin le prédicat de terminaison $H_n(m,x_1,\dots,x_n,t)$, qui indique que la machine de code m avec en entrée $x_1,\dots,x_n$ s'est arrêtée au bout de t étapes de calcul, se définit de façon primitive récursive.

$$H_n(m,x_1,\dots,x_n,t) \equiv \mathrm{Halt}[m,\mathrm{st}_n(m,x_1,\dots,x_n,t)]$$

Le *temps de calcul* de la machine de code m pour les entrées $x_1,\dots,x_n$ est donc obtenu par minimisation, c'est :

$$\mu t.H_n(m,x_1,\dots,x_n,t)$$

Étant donné un état e, le contenu du registre R_0 est donné par $\mathrm{hd}(\pi_2^2(e))$. La fonction calculée par la machine m est donc définie par :

$$\mathrm{hd}\circ\pi_2^2\circ\mathrm{st}_n(m,x_1,\dots,x_n,\mu t.H_n(m,x_1,\dots,x_n,t))$$

Elle est partielle μ-récursive. Remarquez bien que l'argument m est entier quelconque, pas forcément un « vrai » code d'une machine. Résumons les résultats obtenus.

Proposition 1.5.5 *Pour tout entier $n \geq 1$, il existe une fonction récursive primitive $U_n : \mathbb{N}^{n+2} \to \mathbb{N}$, et un prédicat récursif primitif H_n à $n+2$ arguments tels que, si m est le code d'une machine M avec programme* goto, *alors :*

- $\mathrm{st}_n(m, x_1, \ldots, x_n, t)$ *est le code de l'état de la machine M avec en entrées $x_1, \ldots, x_n$ au bout de t étapes de calcul ;*
- $H_n(m, x_1, \ldots, x_n, t)$ *ssi M avec en entrées $x_1, \ldots, x_n$ s'arrête au bout de t étapes de calcul.*
- *La fonction partielle n-aire f calculée par M est μ-récursive et vérifie :*

$$f(x_1, \ldots, x_n) = U_n(m, x_1, \ldots, x_n, \mu t.H_n(m, x_1, \ldots, x_n, t))$$

Corollaire 1.5.6 *Toute fonction partielle calculable par machine à registres avec programme* goto, *avec programme structuré, ou avec programme* while *est μ-récursive.*

Démonstration. D'après la proposition précédente et la proposition 1.3.8. ■

Du corollaire précédent et de la proposition 1.4.1 on déduit l'équivalence des tous les modèles de calcul introduits jusqu'ici pour les fonctions partielles.

Théorème 1.5.7 *Les propositions suivantes sont équivalentes :*

- *la fonction partielle f est μ-récursive ;*
- *la fonction partielle f est calculable par machine à registres avec programme* goto *;*
- *la fonction partielle f est calculable par machine à registres avec programme* while *;*
- *la fonction partielle f est calculable par machine à registres avec programme structuré.*

Temps de calcul d'une fonction récursive primitive

Les fonctions récursives primitives définissent d'une certain façon une classe de complexité (très étendue !) comme le montrent les propositions qui suivent.

Proposition 1.5.8 *Si le temps de calcul (sur une machine à registres) d'une machine est borné par une fonction récursive primitive (en fonction de ses entrées) alors la fonction calculée par la machine est primitive récursive.*

Démonstration. On reprend la Proposition 1.5.5. On peut borner la minimisation par une fonction récursive primitive, le prédicat H_n et la fonction U_n sont récursifs primitifs. ■

La réciproque de cette proposition est intuitivement vraie, mais plus pénible à démontrer.

Proposition 1.5.9 *Toute fonction récursive primitive est calculable par une machine dont le temps de calcul est une fonction récursive primitive des entrées.*

Démonstration. (indications). On montre d'abord que le temps de calcul d'une machine avec programme for est une fonction primitive récursive des entrées : c'est essentiellement le nombre de pas dans les boucles for, une succession de boucles for correspond à une addition, une imbrication à une multiplication. Le nombre de pas d'une boucle est le contenu d'un registre qui est une fonction primitive récursive des entrées (voir exercice 12). On montre ensuite que le temps de calcul reste une fonction récursive primitive par traduction d'un programme for en programme goto. ■

Évidemment une machine peut calculer une fonction récursive primitive sans que son temps de calcul soit une fonction récursive primitive des entrées ! Par exemple la fonction nulle est calculée par une machine obtenue en ajoutant en fin de programme l'instruction $R_0 := 0$ pour n'importe quelle machine calculant une fonction totale.

1.5.3 Thèse de Church

Nous venons de voir que les différentes classes de machines à registres et les fonctions μ-recursives définissent la même notion de fonction calculable. La méthode utilisée pour montrer que les fonctions partielles calculables par machine sont μ-récursives est tout à fait générale. Dès que l'on a une notion de machine (ou de programme) avec un calcul qui se ramène à une succession d'états, et un ou des états terminaux, on a juste besoin :

— de coder les machines et les états des machines ;
— de montrer que les fonctions d'initialisation et de transition induisent sur les codes des fonctions totales qui sont μ-récursives, et que le prédicat d'arrêt est décidable (au sens des fonctions μ-récursives).

Bien d'autres modélisations de la notion de fonction calculable existent. On citera entre autres les machines de Turing, le lambda calcul, des systèmes basés sur des principes de substitution de mots (Post), des systèmes équationnels (Herbrand-Gödel), des machines à pointeurs (Kolmogorov-Uspenski, Schönhage) etc. Tous ces modèles sont équivalents du point de vue de la calculabilité. À chaque fois la méthode résumée ci-dessus peut s'utiliser pour montrer que les fonctions partielles calculables avec l'un de ces modèles de calcul sont μ-récursives[5]. On a même du mal à imaginer comment une modélisation avec calcul explicite pourrait y échapper, même si les détails techniques peuvent s'avérer fastidieux. Il faut vérifier bien entendu à chaque fois la réciproque, mais si celle-ci n'est pas vérifiée on a simplement un modèle de calcul moins puissant.

En résumé cette série de résultats, mais aussi la grande généralité de la méthode mise en œuvre, semblent confirmer la conjecture suivante, dite thèse de Church.

Thèse de Church *Toute fonction calculable est μ-récursive (ou calculable par machine à registres, ou calculable par machine de Turing, etc.)*

Cette affirmation n'est pas démontrable puisqu'elle elle met en relation une notion intuitive (le fait d'être calculable) avec une notion mathématiquement précise (être calculable dans un modèle particulier). Mais on a tout de même donné des arguments pour sa validité, et elle n'a jamais pu être infirmée.

5. D'autres démonstrations par simulations directes des modèles de calcul entre eux sont aussi possibles.

Chapitre 2

Résultats fondamentaux de calculabilité

2.1 Introduction

Grâce à la caractérisation des fonctions calculables comme fonctions calculables par machines à registres [1] on a comme corollaire immédiat de la proposition 1.5.5 page 30.

Corollaire 2.1.1 *Toute fonction calculable s'écrit comme composée d'une fonction récursive primitive et d'une fonction définie par minimisation sur un prédicat récursif primitif.*

En particulier on obtient toutes les fonctions partielles calculables en restreignant dans la définition 1.2.4 des fonctions partielles μ-récursives le schéma de minimisation $\mu t.[g(\overline{x},t)=0]$ aux fonctions g totales.

Nous avions vu deux façons de définir les fonctions μ-récursives totales : soit ce sont les fonctions partielles μ-récursives de la définition 1.2.4 page 16 qui s'avèrent totales, soit elles sont obtenues par la définition inductive 1.2.5 page 17 qui restreint la minimisation aux fonctions ou les prédicats *réguliers*). Toujours comme corollaire de la proposition 1.5.5, on obtient que ces définitions sont équivalentes, et on parlera simplement de *fonction totale calculable*.

Proposition 2.1.2 *L'ensemble des fonctions totales μ-récursives (définition 1.2.5) est égal à l'ensemble des fonctions partielles μ-récursives (définition 1.2.4) qui s'avèrent totales.*

Démonstration. Si f est une fonction totale μ-récursive (définition 1.2.5), elle est évidemment aussi une fonction partielle μ-récursive (définition 1.2.4). Réciproquement, si f est une fonction partielle μ-récursive, alors pour un certain $m\in\mathbb{N}$:

$$f(x_1,\ldots,x_n)=U_n\left(m,x_1,\ldots,x_n,\mu t.H_n(m,x_1,\ldots,x_n,t)\right).$$

Si de plus f est une fonction totale le calcul de f sur la machine de code m s'arrête pour toute entrée, c'est-à-dire que pour tout uple $(x_1,\ldots,x_n)$ il existe t tel que $H_n(m,x_1,\ldots,x_n,t)$. Ceci signifie que $H_n(m,x_1,\ldots,x_n,s)$ est régulier. Comme f est définie par composition à partir de fonctions récursives primitives et d'une fonction définie par minimisation à partir d'un prédicat régulier, f est une fonction totale μ-récursive au sens de la définition 1.2.5. ■

Plus généralement, on peut, en utilisant la preuve précédente, dégager des propriétés indépendantes de la forme particulière du calcul. Ceci d'autant que nous avons fait plus qu'il n'en fallait pour le résultat de simulation obtenu : il aurait suffit de considérer une machine comme une constante du problème, alors que nous avons codé la machine par un entier et pris celui-ci comme paramètre, et c'est quelque chose que nous allons maintenant exploiter.

1. Les machines de Turing ou tout autre modèle de calcul « raisonnable » conviendrait.

2.2 Fonctions universelles.

On pourrait très bien continuer à exploiter la proposition 1.5.5, mais on obtient une forme un peu plus simple pour les fonctions calculables avec une démonstration alternative des résultats précédents qui fait intervenir le code de la suite des états de la machine, plutôt que le temps de calcul qui est la longueur de cette suite. C'est l'objet de la section suivante.

2.2.1 Forme normale de Kleene.

Proposition 2.2.1 *Pour chaque entier $n \geq 1$ il existe un prédicat récursif primitif noté T^n et une fonction récursive primitive $U : \mathbb{N} \to \mathbb{N}$ tels que pour toute machine de code m calculant une fonction à n arguments f :*

1. $\exists s\ T^n[m, x_1, \ldots, x_n, s] \Leftrightarrow f(x_1, \ldots, x_n)\downarrow$
2. $T^n[m, x_1, \ldots, x_n, s] \Rightarrow U(s) = f(x_1, \ldots, x_n)$
3. $\left(T^n[m, x_1, \ldots, x_n, s] \text{ et } T^n[m, x_1, \ldots, x_n, s']\right) \Rightarrow s = s'$

Démonstration. L'idée est que s désigne le code (en commençant par la fin) d'une suite d'états correcte pour m, soit $[e_n; \ldots; e_0]$. On commence par définir un prédicat P_n qui indique que la suite d'états est bien correcte (sans que le calcul soit forcément fini), pour une machine de code m avec $x_1, \ldots, x_n$ en entrées. Le prédicat P_n est défini par sa fonction caractéristique g_n :

$$\begin{aligned} &g_n(m, x_1, \ldots, x_n, []) = 1 \\ &g_n(m, x_1, \ldots, x_n, [e]) = \chi_=(\text{init}_n(m, x_1, \ldots, x_n), e) \\ &g_n(m, x_1, \ldots, x_n, e :: s) = g_n(m, x_1, \ldots, x_n, s) \cdot \chi_=(\text{tr}(m, \text{hd}(s)), e) \end{aligned}$$

la fonction g_n est définie par récurrence structurelle sur les listes, à partir de fonctions récursives primitives. on a vu qu'alors g_n est récursive primitive. Le prédicat P_n est donc récursif primitif. On définit ensuite

$$\begin{aligned} &T'^n[m, x_1, \ldots, x_n, s] \equiv P_n(m, x_1, \ldots, x_n, s] \text{ et } \text{Halt}[m, \text{hd}(s)] \\ &T^n[m, x_1, \ldots, x_n, s] \equiv T'^n[m, x_1, \ldots, x_n, s] \text{ et } \forall s' < s \neg T'^n[m, x_1, \ldots, x_n, s'] \end{aligned}$$

Le prédicat récursif primitif T'^n ne vérifie a priori que la première condition de la proposition (terminaison). Le prédicat T^n est récursif primitif et vérifie les deux premières conditions : terminaison et fonctionalité pour s en sortie.

La fonction U a besoin d'extraire la valeur de R_0 du code du premier état de s soit :

$$U(s) = \text{hd}(\pi_1^2(\text{hd}(s)))$$

fonction qui est bien récursive primitive. ■

Corollaire 2.2.2 (forme normale de Kleene) *Les prédicats T^n et la fonction U sont ceux de la proposition précédente. Pour toute fonction partielle calculable $f : \mathbb{N}^n \to \mathbb{N}$, il existe au moins un entier m tel que :*

$$f(x_1, \ldots, x_n) = U(\mu s.T^n[m, x_1, \ldots, x_n, s])\ .$$

Démonstration. Il suffit de prendre le code m d'une machine qui calcule f. ■

Une expression de la fonction f comme celle donnée dans le corollaire :

$$f(x_1, \ldots, x_n) = g(\mu s.P(x_1, \ldots, x_n, s))$$

où $g : \mathbb{N} \to \mathbb{N}$ est une fonction récursive primitive et P un prédicat récursif primitif est dite sous *forme normale de Kleene.*

Exercice 14 En utilisant les prédicats T^n et la fonction U, démontrer la proposition 1.2.8 page 17 (clôture par « si ... alors ... sinon ... » dans le cas des fonctions partielles calculables).

2.2.2 Propriété d'énumération.

Pour chaque $n \geq 1$ on définit la fonction d'arité $n+1$ φ^n :

$$\varphi^n(m, x_1, \ldots, x_n) = U(\mu s.T^n[m, x_1, \ldots, x_n, s])$$

et on note simplement φ pour φ^1.

Cette fonction partielle calculable permet de calculer toutes les fonctions partielles calculables à n arguments. On dit que c'est une *fonction universelle* pour les fonctions partielles calculables à n arguments. On peut réécrire ainsi le corollaire précédent :

Théorème 2.2.3 (Théorème d'énumération) *la famille de fonctions $(\varphi^n)_{n\in\mathbb{N}^*}$, définie ci-dessus est telle que pour toute fonction partielle calculable $f : \mathbb{N}^n \to \mathbb{N}$ il existe au moins un entier m tel que :*

$$f(x_1, \ldots, x_n) = \varphi^n(m, x_1, \ldots, x_n)$$

Un tel entier m sera appelé un *indice de f* (rappelons que c'est essentiellement le code d'une machine qui calcule f). On notera $f = \varphi^n_m$ si m est un code de f.

En fait chacune des fonctions $(\varphi^n)_{n\in\mathbb{N}^*}$ est universelle en un sens un peu plus fort. Par exemple, la fonction $\varphi = \varphi^1$ énumère modulo codage toutes les fonctions partielles calculables :

Corollaire 2.2.4 *Pour toute fonction partielle calculable $f : \mathbb{N}^n \to \mathbb{N}$, il existe un entier i tel que :*

$$f(x_1, \ldots, x_n) = \varphi(i, \langle x_1, \ldots, x_n\rangle)$$

Démonstration. On prend pour i un indice de la fonction : $x \mapsto f(\pi^n_1(x), \ldots, \pi^n_n(x))$. ■

Il est intuitivement clair (voir exercice suivant) qu'une fonction partielle calculable possède une infinité d'indices : on peut toujours ajouter des détours inutiles dans le programme d'une machine. On montrera plus tard que c'est en un certain sens inévitable.

Exercice 15 (Lemme de bourrage)

1. étant donné une machine M à k registres, construire une machine M', avec autant de registres, qui calcule la même fonction, et dont le programme a un code strictement plus grand que celui de M.
2. Soit $n \in \mathbb{N}^*$. Montrer qu'il existe une fonction récursive primitive $u : \mathbb{N} \to \mathbb{N}$ telle que

   $$\varphi^n_{u(m)} = \varphi^n_m \text{ et pour tout } m \; u(m) > m$$

   (u peut dépendre de n) ;
3. Montrer qu'il existe une fonction récursive primitive $v : \mathbb{N}^2 \to \mathbb{N}$ telle que v est strictement croissante en sur son second argument et pour tous entiers m et p, $\varphi^n_{v(m,p)} = \varphi^n_m$.

Une conséquence immédiate du théorème d'énumération est que la fonction partielle φ^n, qui est calculable à $n+1$ arguments, a un indice : le code d'une machine qui la calcule, que l'on appelle *machine universelle*.

Corollaire 2.2.5 *Pour tout $n \geq 1$ il existe un entier m_n tel que :*

$$\varphi^n(m, x_1, \ldots, x_n) = \varphi^{n+1}(m_n, m, x_1, \ldots, x_n)\ .$$

2.2.3 Propriété de paramétrisation.

Une propriété en quelque sorte réciproque de celle que l'on vient d'énoncer, est aussi caractéristique des familles de fonctions universelle, la propriété de *paramétrisation*.

Théorème 2.2.6 (théorème s_m^n ou de paramétrisation) *Pour tous entiers $m, n \geq 1$ il existe une fonction totale calculable $s_m^n : \mathbb{N}^{m+1} \to \mathbb{N}$ telle que :*

$$\varphi^{m+n}(i, x_1, \ldots, x_m, y_1, \ldots, y_n) = \varphi^m(s_m^n(i, x_1, \ldots, x_m), y_1, \ldots, y_n)$$

Pour la famille de fonctions universelle étudiée ici, la fonction s_m^n est récursive primitive.

Si on on revient aux machines, la fonction s_m^n permet donc, à partir du code i d'une machine M à $m+n$ entrées, et de m entiers $x_1, \ldots, x_m$, de calculer le code d'une machine M' à n entrées, qui fonctionne comme M, après avoir fixé les m premières entrées à $x_1, \ldots, x_m$.

Démonstration. On va suivre la remarque précédente. Il s'agit d'abord d'indiquer comment construire M', puis on montre que les constructions se codent de façon récursive primitive. Il faut veiller à ce que la fonction calculée soit bien celle cherchée même quand l'indice $i = \langle k, l \rangle$ n'est pas un « vrai » code de machine (cf. paragraphe 1.5.2). La machine M' va avoir les mêmes k registres. On construit son programme à partir de celui de M en 3 étapes.

1. On ajoute en tête de programme des instructions qui essentiellement (c'est-à-dire quand il y a suffisamment de registres) recopient les registres d'entrées $R_1, \ldots, R_n$ à partir de R_{m+1} (registres de travail), et mettent à 0 les n premiers registres d'entrées). il faut en fait discuter suivant le nombre de registres k. L'opération est donnée,
   si $k \leq m$ par la séquence
   $$R_k := 0, \ldots, R_1 := 0$$
   si $m < k < m+n$ par la séquence
   $$R_k := R_{k-m}, \ldots, R_{m+1} := R_1$$
   et si $k \geq m+n$ par la séquence
   $$R_{m+n} := R_n, \ldots, R_{m+1} := R_1 \ .$$
   Chacune des instructions $R_{m+i} := R_i$, respectivement $R_i := 0$, s'écrit

l	if $R_i = 0$ goto $l+4$	l	if $R_i = 0$ goto $l+3$
$l+1$	$R_{m+i} := R_{m+i} + 1$	$l+1$	$R_i := R_i - 1$
$l+2$	$R_i := R_i - 1$	$l+2$	goto l
$l+3$	goto l		

   avec, par exemple dans le dernier cas $l = 4(n-i)$. On ajoute $3k$ lignes dans le premier cas, $4(k-m)$ dans le second cas et $4n$ lignes dans le dernier.
2. On ajoute ensuite les instructions qui correspondent à :
   $$R_1 := x_1, \ldots, R_m := x_m$$
   chaque opération $R_i := x_i$ s'écrit (le registre R_i est à 0) :
   $$\underbrace{R_i := R_i + 1, \ldots, R_i := R_i + 1}_{x_i}$$
   on ajoute donc $x_1 + \cdots + x_m$ lignes.
3. On ajoute le programme de M modifié de façon à se comporter de la même façon que pour la machine M, c'est-à-dire que les instructions goto doivent pointer sur les numéros de lignes décalées du nombre d'instructions ajouté en tête de programme. Ce nombre est une fonction de $(k, x_1, \ldots, x_m)$ (m et n sont des constantes).

Il est à peu près clair qu'à chaque étape correspond une opération récursive primitive sur le code. Ceux qui n'en sont pas convaincus sont invités à le démontrer eux-mêmes, ou, en dernier ressort, à chercher de possibles erreurs dans les détails donnés ci-dessous.

1. la fonction $k, s \mapsto f_1(k,s)$ qui ajoute le code de la liste des instructions de l'étape 1 et la fonction $k \mapsto nb(k)$ qui donne le nombre d'instruction ajoutées, sont récursives primitives. Elles se définissent par cas et récurrence sur k comme indiqué.

2. Le code d'une suite de x incrémentations sur le registre i se définit par récurrence récursive primitive sur x

$$a(i,x) = []$$
$$a(i,x+1) = \langle 0, i\rangle :: a(i,x) \quad \text{0 est le code de l'incrémentation}$$

La fonction a est récursive primitive, et donc également la fonction $f_2(s,x_1,\ldots,x_m)$ qui ajoute la liste d'instructions souhaitée en tête du programme s (m est une constante) :

$$f_2(x_1,\ldots,x_m,s) = a(1,x_1)@\ldots@a(n,x_m)@s$$

3. La fonction $d(g,e)$ décale de g les numéro de ligne dans le code d'une instruction e s'il s'agit d'un goto, et ne fait rien sinon. Elle est récursive primitive.

si $\pi_1^2(e) = 2$ si goto
  alors $d(g,e) = \langle 2, \pi_2^3(e), \pi_3^3(e) + g\rangle$
sinon $d(g,e) = e$

Il suit que, par composition, la fonction $d'(x_1,\ldots,x_m,i,e) = d((nb(\pi_1^2(i)) + x_1 + \cdots + x_m, e)$ est récursive primitive, et donc également La fonction f_3 qui calcule le résultat de l'application de la fonction $e \mapsto d'(x_1,\ldots,x_m,i,e)$ à toutes les instructions du programme ; celui-ci étant une liste on utilise une fonction map, (voir page 11), soit $f_3(x_1,\ldots,x_m,i) = \mathrm{map}_{d'}(\pi_2^2(i))$. Cette dernière construit à partir du code i de la machine M le code du programme avec les renumérotations de registres et décalages des goto souhaités.

On obtient finalement la fonction s_m^n de façon récursive primitive :

$$\mathrm{s}_m^n(i,x_1,\ldots,x_m) = \langle \pi_1^2(i), f_1(\pi_1^2(i), f_2(x_1,\ldots,x_m, f_3(i,x_1,\ldots,x_m))))\rangle$$ ■

La famille de fonctions universelles telle qu'elle est construite est très arbitraire : elle dépend non seulement du choix des machines à registres pour représenter le calcul, mais aussi du choix du codage qui n'a rien d'intrinsèque.

On verra plus tard que la propriété d'énumération et l'existence de fonctions s_m^n, appelée aussi propriété de paramétrisation, axiomatisent les familles de fonctions partielles calculables universelles : toute famille de fonctions universelles $(\psi^n)_{n\geq 1}$ qui vérifient ces deux propriétés se déduit de $(\varphi^n)_{n\geq 1}$ par une permutation récursive h :

$$\psi_i^1 = \varphi_{h(i)}^1 .$$

L'importance de la propriété de paramétrisation va apparaître dans les chapitres qui suivent. Avec les fonctions universelles, les entiers jouent un double rôle : celui de donnée, mais aussi celui de programme. Mais les programmes peuvent aussi être vu comme des données pour d'autres programmes. La fonction s_m^n permet donc également de décrire des opérations calculables sur les programmes et de composer ceux-ci. Voyons un exemple simple. Soit e le code de la fonction $(i,j,x) \mapsto \varphi_i^1(x) + \varphi_j^1(x)$:

$$\varphi(i,x) + \varphi(j,x) = \varphi^3(e,i,j,x) = \varphi^1(\mathrm{s}_2^1(e,i,j),x) .$$

La fonction $(i,j) \mapsto \mathrm{s}_2^1(e,i,j)$ calcule donc, à partir des indices de deux fonctions, l'indice de la somme de ces deux fonctions. Il fabrique en quelque sorte le (code du) programme pour la somme de ces deux fonctions, en intégrant comme sous-programmes les programmes (de codes i et j) calculant chacune de ces deux fonctions.

2.3 Problèmes indécidables

2.3.1 Ensembles effectivement énumérables

Définition 2.3.1 (ensemble effectivement énumérable) Un sous-ensemble A de $\mathbb{N}^k$ est dit *semi-décidable* ou *effectivement énumérable* quand il est le domaine d'une fonction calculable (partielle).

Un tel ensemble est aussi appelé *récursivement énumérable* (quand « récursif » est utilisé au sens de « calculable »).

On parle aussi de *prédicat effectivement énumérable* ou *semi-décidable* pour un prédicat d'arité k dont le support est effectivement énumérable.

On note W_m^k le domaine de définition de la fonction (à k variables) d'indice m, φ_m^k. L'ensemble des prédicats effectivement énumérables est donc constitué des ensembles W_m^k.

Notons tout d'abord qu'il est toujours possible de se ramener aux sous-ensembles de $\mathbb{N}$ via codage.

Lemme 2.3.2 *Soit A un sous-ensemble de $\mathbb{N}^k$. On pose $A^{\langle\rangle} = \{\langle x_1,\dots,x_k\rangle \mid (x_1,\dots,x_k) \in A\}$ ($A \subset \mathbb{N}$). Alors :*

— *A est semi-décidable si et seulement si $A^{\langle\rangle}$ est semi-décidable ;*

— *A est décidable si et seulement si $A^{\langle\rangle}$ est décidable.*

On rappelle que le résultat est encore vrai en prenant récursif primitif pour décidable.

Démonstration. Par composition avec α_k pour un sens, $(\pi_1^k,\dots,\pi_k^k)$ pour l'autre. ■

Proposition 2.3.3 (caractérisation des ensembles semi-décidables) *Soit $A \subseteq \mathbb{N}^k$. Alors A est semi-décidable dès qu'il satisfait l'une des assertions suivantes qui sont équivalentes :*

1. *A est le domaine d'une fonction partielle calculable ;*
2. *A est le projeté d'un ensemble récursif primitif, B, c'est-à-dire que $A = \{\overline{x} : \exists y (\overline{x}, y) \in B\}$;*
3. *A est le projeté d'un ensemble décidable ;*
4. *A est l'image d'une fonction partielle calculable, modulo codage des k-uplets, soit il existe f partielle calculable telle que* $\operatorname{Im} f = \{\langle x_1,\dots,x_k\rangle \mid (x_1,\dots,x_k) \in A\}$.
5. *A est vide ou l'image d'une fonction récursive primitive à un argument modulo codage des k-uplets, soit il existe f fonction récursive primitive telle que* $\operatorname{Im} f = \{\langle x_1,\dots,x_k\rangle \mid (x_1,\dots,x_k) \in A\}$;
6. *A est vide ou l'image d'une fonction calculable totale à un argument, modulo codage des k-uplets.*

La dernière caractérisation justifie l'appellation « effectivement énumérable » pour les ensembles semi-décidables. Quand $A \subseteq \mathbb{N}$ elle s'écrit plus simplement

6'. A est vide où il existe une fonction totale calculable f telle que $A = \operatorname{Im} f$.

Démonstration. On peut supposer par codage (lemme 2.3.2) que A est un sous-ensemble de $\mathbb{N}$.

(1 ⇒ 2) : soit φ_m la fonction dont A est le domaine, alors $\{(x,s) \mid T^1[m,x,s]\}$ convient, où T^1 est le prédicat d'arrêt de Kleene, qui est récursif primitif, cf. proposition 2.2.1.

(2 ⇒ 3) : évident.

(3 ⇒ 6) et (2 ⇒ 5) : on suppose A non vide, soit donc $a \in A$. La fonction f qui à $z = \langle x,y\rangle$ associe x si $(x,y) \in B$, a sinon (c'est une définition par cas de fonction totale calculable, voir page 6), convient.

(6 ⇒ 4) et (5 ⇒ 4) : l'ensemble vide est l'ensemble image de la fonction nulle part définie.

(4 ⇒ 1) : soit φ_m la fonction partielle calculable dont A est l'image. Alors la fonction suivante convient (les notations sont celles de Kleene, cf. proposition 2.2.1)

$$f(y) = \mu t \cdot \left(T^1[m,\pi_1^2(t),\pi_2^2(t)] \text{ et } y = U(\pi_2^2(t))\right) .$$

Elle est bien définie si la machine de code m s'arrête avec y en valeur de sortie. ■

Un ensemble décidable A est par exemple le projeté de $A \times \mathbb{N}$ sur ses premières composantes.

Fait 2.3.4 *Un ensemble décidable est effectivement énumérable.*

De la caractérisation d'un semi-décidable comme projeté d'un ensemble décidable, c'est-à-dire que tout prédicat semi-décidable est obtenu par quantification existentielle sur un prédicat décidable, on en déduit la clôture par projection des semi-décidables. Certaines propriétés de clôture des prédicats décidables (qui sont clos par opérations récursives primitives donc a les propriétés de clôture du paragraphe 1.1.2) sont héritées par projection, mais pas toutes.

Proposition 2.3.5 (propriétés de clôture des effectivement énumérables) *La classe des prédicats effectivement énumérables est close par*

1. *conjonction et disjonction ;*
2. *quantification existentielle ;*
3. *quantification universelle bornée par une variable.*

Ceci se traduit par le fait que la classe des ensembles effectivement énumérables est close par intersection, produit cartésien, réunion, projection (et l'opération correspondant à la quantification universelle bornée sur une variable).

Démonstration.

1. Soit $P\,x_1,\ldots,x_p,z_1,\ldots,z_k$ et $Q\,y_1,\ldots,y_q,z_1,\ldots,z_k$ deux prédicats semi-décidables, caractérisés par les domaines des fonctions f et g, alors $f+g$ a un domaine caractérisé par la conjonction de ces deux prédicats.

   La disjonction a réellement un sens pour des prédicats de même arité, il suffit de compléter pour que les prédicats portent P et Q sur les mêmes variables $\overline{x}$. Soient A et B des prédicats décidables tels que
   $$P\overline{x} \equiv \exists z A\overline{x},z\,;\quad Q\overline{x} \equiv \exists z B\overline{x},z\,;$$
   On a $(P\vee Q)\overline{x} \equiv \exists z(A\overline{x},z \vee B\overline{x},z)$.
2. On a pour P semi-décidable A décidable tel que $P\overline{x},y \equiv \exists z A\overline{x},y,z$ et donc $\exists y\,P\overline{x},y \equiv \exists t\,A\overline{x},\pi_1^2(t),\pi_2^2(t)$, quantification existentielle sur un prédicat décidable par composition.
3. Avec les mêmes notations on a $\forall y \le t\,P\overline{x},y \equiv \forall y \le t \exists z\,A\overline{x},y,z$. Il s'agit d'échanger les deux derniers quantificateurs. Soient $\overline{x},t$ tels que $\forall y \le t\,P\overline{x},y$. Nous sommes dans $\mathbb{N}$, il n'y a qu'un nombre fini de $y \le t$, t étant fixé, pour lesquels on choisit un témoin z_y tel que $A\overline{x},y,z_y$. Soit $u = \sup_{y\le t} z_y$. On a alors $\forall y \le t \exists z \le u\,A\overline{x},y,z$. Ce prédicat est décidable et donc le prédicat $\exists u \forall y \le t \exists z \le u\,A\overline{x},y,z$, qui est satisfait par $\overline{x},t$, est semi-décidable Réciproquement si ce prédicat est satisfait par $\overline{x},t$ on a évidemment $\forall y \le t\,P\overline{x},y$, d'où le résultat cherché. ■

La proposition suivante relie les notions d'ensemble décidable et effectivement énumérable.

Proposition 2.3.6 *Un ensemble $A \subseteq \mathbb{N}^k$ est décidable si et seulement si A et son complémentaire $A^c = \mathbb{N}^k \setminus A$ sont effectivement énumérables.*

Démonstration. Si A est décidable alors clairement A et A^c sont effectivement énumérables. Pour la réciproque, si A et A^c sont les domaines (disjoints) respectives de deux fonctions partielles calculables d'indices m_0 et m_1, la fonction s définie par
$$s(\overline{x}) = \mu y.\left(T^k[m_0,\overline{x},y] \vee T^k[m_1,\overline{x},y]\right)$$
est totale calculable. On a alors : $A(\overline{x})$ si et seulement si $T^k[m_0,\overline{x},s(\overline{x})]$ et A est donc décidable[2]. On peut examiner l'argument ci-dessus avec un point de vue plus algorithmique. Supposons qu'il existe une machine M_1 ne terminant que sur les éléments de A et une machine M_2 ne terminant que sur les éléments de $\mathbb{N}^k\setminus A$. On a simplement construit une machine M qui, sur une entrée $\overline{x}$ exécute les deux machines M_1 et M_2 sur $\overline{x}$ en parallèle (en entrelaçant les instructions des machines[3]). L'entrée $\overline{x}$ est alors acceptée ou refusée suivant laquelle de M_1 et M_2 s'arrête. Noter qu'on ne peut pas lancer une exécution (celle de M_1, par exemple) puis l'autre car la première exécution peut ne pas terminer. ■

Dès que l'on aura montré l'existence d'un ensemble effectivement énumérable non décidable, ce qui va être fait en section 2.3.3, on pourra déduire de cette proposition et de la caractérisation des semi-décidables que

2. vous pouvez pousser les détails jusqu'à donner la définition de la fonction caractéristique de A

3. essayez de donner les détails pour une machine à registres. Aide : faites travailler les machines sur deux ensembles de registres différents. Utiliser 3 nouveaux registres en plus : deux pour garder, à tout instant le numéro de l'instruction de chacun des programmes en cours ; un dernier pour compter les pas de programme et permettre que tout étape impaire corresponde à une étape de simulation de M_1 et toute étape paire pour M_2

— la classe des ensembles effectivement énumérables n'est pas close par complémentation (contrairement à celle des ensembles décidables) ;
— la classe des ensembles décidables n'est pas close par projection (contrairement à celle des ensembles semi-décidables).

Nous avons vu qu'un sous-ensemble effectivement énumérable non vide de $\mathbb{N}$ était l'image d'une fonction totale calculable (proposition 2.3.3). Si l'ensemble est infini la fonction d'énumération peut être supposée injective. Mais si on la suppose de plus strictement croissante son ensemble image devient décidable. Les ensembles effectivement énumérables non décidables (voir section 2.3.3) ne peuvent donc être énumérés « par ordre croissant ».

Exercice 16

1. Montrer qu'un ensemble *effectivement énumérable infini* est l'image d'une fonction *calculable injective*, i.e. montrer que pour toute fonction totale calculable f dont l'image est infinie, il existe une fonction totale calculable injective g telle que $\mathrm{Im}\, f = \mathrm{Im}\, g$.
2. Montrer que l'image d'une fonction totale calculable croissante est un ensemble décidable.
3. Réciproquement montrer que tout ensemble *décidable infini* est l'image d'une fonction *calculable strictement croissante.*
4. En utilisant les questions précédentes, montrer que tout ensemble effectivement énumérable infini contient un sous-ensemble décidable infini.

2.3.2 Prédicats et problèmes

Dans un contexte algorithmique, il peut être plus naturel de parler de problème que d'ensemble ou de prédicat. Soit $A \subseteq \mathbb{N}^k$, le problème associé est :

PROBLÈME A (APPARTENANCE À A)
Entrée : $\overline{x} \in \mathbb{N}^k$
Question : $\overline{x}$ appartient-t-il à A ?

Lorsque A est un ensemble décidable, on dit alors que le problème A associé est *décidable*. Il est dit *indécidable* dans le cas contraire. Un problème *semi-décidable* est un problème associé à un prédicat semi-décidable.

2.3.3 Problème de l'arrêt : la méthode diagonale

On montre dans cette partie que certains problèmes sont indécidables par nature (bien qu'ils puissent être semi-décidables). La méthode employée est bien connue des mathématiciens depuis Cantor : il s'agit de la diagonalisation. Pour arriver à notre résultat le plus général, le problème de l'arrêt d'une machine, on commence par le problème suivant.

Théorème 2.3.7 (problème de l'arrêt diagonal) *L'ensemble $K = \{m \mid \varphi^1_m(m)\downarrow\}$ n'est pas décidable, plus précisément l'ensemble K est effectivement énumérable et son complémentaire K^c n'est pas effectivement énumérable.*

Démonstration. La fonction d'énumération φ^1 est une fonction partielle calculable à deux variables $\varphi^1_m(x) = \varphi^1(m,x)$. La fonction : $m \mapsto \varphi^1_m(m)$ est donc partielle calculable et son domaine K est effectivement énumérable. D'après la proposition 2.3.6 page précédente, il suffit donc de montrer que le complémentaire de K, noté K^c, n'est pas effectivement énumérable. Supposons le contraire [4]. Dans ce cas, K^c est le domaine d'une certaine fonction partielle calculable d'indice a, soit φ^1_a.

Une question naturelle est alors : est-ce que $a \in K^c$? On voit par définition que :

$$\begin{aligned} a \in K^c &\iff a \in \{x : \varphi^1_a(x)\downarrow\} \\ &\iff \varphi^1_a(a)\downarrow \\ &\iff a \in K \end{aligned}$$

ce qui est une contradiction. Donc K^c n'est pas semi-décidable et K ne peut être décidable. ■

4. cette assertion est déjà, intuitivement, douteuse : comment énumérer les entrées x sur lesquelles le calcul de l'image de $f(x)$ ne s'arrête pas... ?

Le prédicat considéré dans la preuve précédente est associé au problème suivant :

DIAG
Entrée : Une machine à registres M donnée par son code m
Question : le calcul de M sur l'entrée m s'arrête-t-il ?

Le théorème 2.3.7 dit que le problème DIAG est indécidable. Même si c'est redondant avec la preuve déjà donnée, on peut regarder comment l'argument diagonal principal se déploit dans le cas d'une preuve directe de l'indécidabilité de ce problème en termes de machine.

Supposons que le problème DIAG est décidable par une machine (mettons à registres avec programme structuré) M. Plus exactement, M calcule la fonction caractéristique Diag avec $\text{Diag}(m) = 1$ si la machine de code m s'arrête sur m, et $\text{Diag}(m) = 0$ sinon. On modifie la machine M en une nouvelle machine M' en la faisant boucler indéfiniment sur l'entrée m lorsque $\text{Diag}(m) = 1$. Cela peut se faire en ajoutant l'instruction suivante en fin de programme :

$$\text{while } R_0 \neq 0 \quad \text{do} \quad \text{end}$$

On a maintenant une machine M' qui s'arrête sur m si $\text{Diag}(m) = 0$, et qui ne s'arrête pas sinon. Que fait la machine M' sur l'entrée constituée de son propre code m' ? On a :

M' s'arrête sur l'entrée m' ssi $\text{Diag}(m') = 0$, c'est-à-dire ssi elle ne s'arrête pas sur m' (par définition de Diag). Ce qui nous amène à une contradiction.

Corollaire 2.3.8 *Pour tout k, il existe un sous-ensemble de $\mathbb{N}^k$ semi-décidable qui n'est pas décidable, il existe un sous-ensemble de $\mathbb{N}^k$ qui n'est pas semi-décidable.*

Démonstration. Immédiat pour $k = 1$ d'après le théorème 2.3.7, l'ensemble diagonal étant semi-décidable ; son complémentaire n'est donc pas semi-décidable (d'après la proposition 2.3.6 mais cela a été montré directement). On déduit le résultat pour k quelconque, car, par exemple, $K \times \{0\}$ est décidable si et seulement si K est décidable. ■

Un autre corollaire immédiat du Théorème 2.3.7 est l'indécidabilité du problème suivant, dont le problème de l'arrêt diagonal est un cas particulier.

ARRET
Entrée : Une machine à registres M, $x \in \mathbb{N}$
Question : le calcul de M sur l'entrée x s'arrête-t-il ?

Théorème 2.3.9 (problème de l'arrêt) *L'ensemble $\{(m,x) \mid \varphi^1(m,x)\downarrow\}$ n'est pas décidable.*

Démonstration. Si l'ensemble $\{(m,x) \mid \varphi^1(m,x)\downarrow\}$ était décidable, l'ensemble $\{m \mid \varphi^1(m,m)\downarrow\}$ le serait aussi. ■

Enfin une autre version est celui de l'arrêt sur une entrée donnée. On va prendre 0 comme entrée : dans ce cas la question est celle de l'arrêt d'une machine dont tous les registres sont initialisés à 0 et ne fait plus référence à une fonction calculée par la machine et à son arité. Mais le résultat d'indécidabilité est identique et se démontre de la même façon pour une entrée quelconque.

ARRET$_0$
Entrée : Une machine à registres M dont tous les registres sont initialisés à 0
Question : le calcul de M s'arrête-t-il ?

Théorème 2.3.10 (problème de l'arrêt des machines initialisées à 0) *L'ensemble $\{m \mid \varphi^1(m,0)\downarrow\}$ n'est pas décidable.*

Démonstration. Ce résultat est une conséquence immédiate du théorème de Rice 2.3.13 page 43, voir le corollaire 2.3.14, mais la preuve directe est un exemple particulièrement simple de méthode de réduction (méthode utilisée aussi pour le théorème de Rice). Il s'agit étant donné une machine de code m, de construire une machine avec un nouveau registre que l'on place en position R_1 et qui n'est jamais

utilisé par le programme : la nouvelle machine possède donc un registre supplémentaire, et le programme est obtenu à partir du programme de la machine de code m, d'abord en renumérotant dans les instructions tous les registres R_i, $i > 0$, en R_{i+1}, puis en ajoutant en tête de programme l'instruction $R_2 := m$ et en décalant les numéros de ligne de façon cohérente dans les instructions goto. Clairement le code $\alpha(m)$ de cette nouvelle machine se calcule à partir de m, on peut même dire que α est récursive primitive. Comme d'une partir le registre R_1 n'est jamais utilisé et d'autre part la machine calcule comme la machine de code m sur l'entrée m nous avons pour un entier x arbitraire :

$$\varphi^1(m,m)\downarrow \text{ ssi } \varphi^1(\alpha(m),x)\downarrow$$

en particulier

$$\varphi^1(m,m)\downarrow \text{ ssi } \varphi^1(\alpha(m),0)\downarrow .$$

La fonction totale calculable α *réduit* donc le problème Diag au le problème de l'arrêt des machines initialisées à 0, et ce dernier ne peut donc être décidable (sinon Diag le serait).

Revenir aux codages des machines est assez intuitif mais s'avère vite pénible (d'ailleurs nous sommes loin d'avoir donné tous les détails). Comme pour d'autres fonctions codant des transformations de programme, il est en fait plus simple d'utiliser le théorème s^n_m. On pose $h(m,x) = \varphi^1(m,m)$ qui est une fonction partielle calculable de $\mathbb{N}^2 \to \mathbb{N}$, et possède donc un indice a :

$$\varphi^1(m,m) = h(m,x) = \varphi^2(a,m,x) = \varphi^1(s^1_1(a,m),x) .$$

En posant $\alpha(m) = s^1_1(a,m)$, on a bien que $\varphi^1(m,m)\downarrow$ si et seulement si $\varphi^1(\alpha(m),0)\downarrow$, et on conclut comme ci-dessus. ■

Ces diverses versions du problème de l'arrêt constituent des problèmes de base pour montrer l'indécidabilité d'autres problèmes algorithmiques.

2.3.4 Prolongement par une fonction totale calculable

Soit $A \subseteq \mathbb{N}^k$ et $f : A \longrightarrow \mathbb{N}$ une fonction partielle de domaine A. On dit que g est est un prolongement, ou une extension de f, si pour tout $\overline{x} \in A$, $f(\overline{x}) = g(\overline{x})$. Le résultat suivant est une conséquence assez directe de l'indécidabilité du théorème de l'arrêt. On en donne néanmoins une preuve indépendante jouant encore une fois avec la méthode de diagonalisation.

Proposition 2.3.11 *Il existe une fonction partielle calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ qui n'a pas de prolongement qui soit une fonction totale calculable.*

Démonstration. Soit φ^1 la fonction universelle à une variable. On définit une fonction calculable f qui diffère en tout point de la diagonale de φ^1, par exemple :

$$f(x) = 1 \dot{-} \varphi^1(x,x).$$

Soit g une fonction totale calculable. Montrons qu'elle ne peut prolonger f. Il existe un indice m tel que $g(x) = \varphi^1(m,x)$. La fonction g étant totale, elle est en particulier définie en m : $f(m)\downarrow$ et $f(m) = 1 \dot{-} g(m)$, donc $f(m) \neq g(m)$. On a bien que g ne peut être un prolongement de f car les deux fonctions ne coincident pas en m. ■

Exercice 17 Montrer que dans la définition 1.2.3 page 15 de $\mu y.\big(f(\overline{x},y) = 0\big)$, la condition $\forall z < y\ f(\overline{x},z)\downarrow$ est indispensable. Plus précisément montrer que, même si f est une fonction partielle calculable à $p+1$ arguments, la fonction partielle

$$g(\overline{x}) = \mu' y.f(\overline{x},y) = 0$$

qui est définie en $\overline{x}$ et vaut y si et seulement si

$$f(\overline{x},y) = 0\,;\ \ \forall z < y\ \big(f(\overline{x},z)\downarrow \Rightarrow f(\overline{x},z) \neq 0\big)$$

peut ne pas être calculable (utiliser le problème de l'arrêt diagonal).

Soient $A \subseteq \mathbb{N}^k$ et $B \subseteq \mathbb{N}^k$ deux ensembles disjoints. On dit qu'un ensemble $C \subseteq \mathbb{N}^k$ *sépare* A et B si $A \subseteq C$ et $B \subseteq \mathbb{N}^k \backslash C$; A et B sont dits *effectivement séparables* s'il existe un tel ensemble C qui est décidable.

Théorème 2.3.12 *Il existe deux ensembles disjoints effectivement énumérables $A \subseteq \mathbb{N}^k$ et $B \subseteq \mathbb{N}^k$ qui ne sont par effectivement séparables.*

Démonstration. Dans la preuve de la proposition 2.3.11, la fonction (partielle) f considérée a pour image l'ensemble $\{0,1\}$. Considérons les ensembles $A = \{x \mid f(x) = 1\}$ et $B = \{x \mid f(x) = 0\}$ (A et B ne forment pas une partition de $\mathbb{N}$ car f peut ne pas être défini pour certains x). A et B sont clairement effectivement énumérables. Supposons qu'il existe un ensemble décidable C tel que $A \subseteq C$ et $B \subseteq \mathbb{N}^k \backslash C$. Dans ce cas, la fonction caractéristique de C, qui est totale et calculable étend f en contradiction avec la Proposition 2.3.11 ■.

2.3.5 Théorème de Rice

Le résultat qui va suivre est très riche de conséquences. Il montre, en quelque sorte, que toute propriété des machines qui ne dépend que des entrées-sorties, toute propriété d'un ensemble de programmes qui ne dépend que de ce que calcule la machine (pas de la façon dont elle le calcule), est soit triviale soit indécidable.

Théorème 2.3.13 (Théorème de Rice) *Soit F un ensemble de fonctions (partielles) calculables à k variables tel que $F \neq \emptyset$ et tel qu'il existe une fonction partielle calculable qui n'est pas dans F. Alors l'ensemble $I_F = \{m \in \mathbb{N} \mid \varphi^k_m \in F\}$ n'est pas décidable.*

On peut réénoncer le théorème de Rice ainsi :

Autre énoncé du théorème de Rice. *Soit $k \in \mathbb{N}^*$ et $E \subset N$ tels que :*

i. $\left(i \in E \text{ et } \varphi^k_i = \varphi^k_j\right) \Rightarrow j \in E$,

ii. $E \neq \emptyset$ *et* $E \neq \mathbb{N}$,

alors E n'est pas décidable.

On dit d'un ensemble ou d'une prédicat qui satisfait la condition i qu'elle est *extensionnelle* (pour les fonctions partielles calculables d'arité k), ce qui traduit le fait que le prédicat sur les entiers ne dépend que de la fonction partielle (au sens ensembliste, le graphe de celle-ci, donc l'extension) dont l'indice est l'entier, et non de l'entier lui-même (ni donc de la machine dont il est le code). Le théorème de Rice exprime donc qu'une propriété des programmes extensionnelle est soit triviale, soit indécidable.

Le problème de l'arrêt fournit typiquement un exemple de propriété extensionnelle. Le théorème de Rice se démontre en se ramenant à celui-ci.

Démonstration. On appelle v la fonction nulle part définie à k arguments. On peut supposer sans perte de généralité, que $v \in F$: sinon on échange F et F^c son complémentaire parmi les fonctions partielles calculables à k arguments. Soit g d'arité k, $g \notin F$. Prenons K l'ensemble effectivement énumérable non décidable fourni par le théorème 2.3.7 (problème de l'arrêt diagonal), n'importe quel sous-ensemble effectivement énumérable non décidable de $\mathbb{N}$ conviendrait. On définit la fonction $f : \mathbb{N}^{k+1} \to \mathbb{N}$ suivante :

$$f(n, \overline{x}) = \begin{cases} g(\overline{x}) \text{ si } n \in K \\ \text{non définie si } n \notin K. \end{cases}$$

La fonction partielle f est bien calculable : la définition par cas sur l'appartenance à un ensemble semi-décidable ne conserve évidemment pas en général le fait d'être calculable — on pourrait définir ainsi la fonction caractéristique de n'importe quel ensemble effectivement énumérable —, mais pour cette forme particulière si. Il suffit de multiplier g par une fonction constante égale à 1 définie seulement sur K[5]). On remarque que, pour n fixé, la fonction f est soit égale à v soit égale à g. On va exploiter ce fait. Notons q un indice de la fonction f. Les deux situations suivantes sont possibles :

5. Vous pouvez aussi donner plus directement le principe d'un algorithme calculant f

— Soit $n \in K$. Dans ce cas, la fonction partielle $\overline{x} \mapsto \varphi^{k+1}(q,n,\overline{x})$ est la fonction g, soit par le Théorème s^n_m $\varphi^k_{s^k_1(q,n)} = g$ et donc $s^k_1(q,n) \notin I_F$.
— Soit $n \notin K$. Dans ce cas, $\varphi^{k+1}(q,n,\overline{x})$ n'est défini pour aucun $\overline{x}$, soit $\varphi^k_{s^k_1(q,n)} = v$ et donc $s^k_1(q,n) \in I_F$.

On a donc l'équivalence :

$$n \in K \iff s^k_1(q,n) \notin I_F$$

Supposons que l'ensemble I_F soit décidable. Dans ce cas, on peut déterminer l'appartenance de n à K indirectement en calculant $s^k_1(q,n)$ (cette fonction est totale calculable) et en testant son appartenance à I_F. Ceci contredit le fait que K n'est pas décidable. ■

Examinons maintenant quelques conséquences du Théorème de Rice.

Corollaire 2.3.14 *Les problèmes suivants sont indécidables :*
— *étant donnée une machine M, déterminer si la fonction calculée par M est une certaine fonction f fixée (i.e. l'ensemble des indices d'une fonction calculable f n'est pas décidable) ;*
— *étant donnée une machine M déterminer si son calcul s'arrête quand tous les registres sont initialisés à 0 (i.e. l'ensemble $\{i \mid \varphi_i(0)\downarrow\}$ n'est pas décidable) ;*
— *étant donnée une machine M, déterminer si l'ensemble des entrées acceptées par M (les entrées pour lesquelles le calcul de M s'arrête) est fini ;*
— *étant données deux machines M et N, déterminer si elles calculent la même fonction.*

2.4 Théorèmes du point fixe

2.4.1 Introduction : la fonction d'Ackermann

On utilise fréquemment des définitions récursives de fonctions calculables : une fonction est définie à partir d'elle même. Un exemple est la définition par récurrence primitive, qui est intégrée à la définition des fonctions μ-récursives, et dont on peut dériver d'autres schémas naturels de définition par récurrence comme la récurrence primitive sur la suite des valeurs 9 et d'autres.

Mais certaines définitions par récurrence ne rentrent pas dans ce cadre comme la récurrence double utilisée pour définir la fonction d'Ackermann définie section 1.2.2 page 13.

Le théorème du point fixe, d'ailleurs également appelé théorème de la récursion, permet entre autres d'assurer que de telles définitions fournissent bien des fonctions (éventuellement partielles) calculables.

Une définition récursive de fonction partielle) a la forme suivante :

$$f = \Phi(f)$$

où Φ est une *fonctionnelle*, soit $\Phi : (\mathbb{N}^k \to \mathbb{N}) \to (\mathbb{N}^k \to \mathbb{N})$, calculable en un sens qu'il faudrait définir. Mais plutôt que se lancer dans une théorie des fonctionnelles calculables, on va plutôt représenter cette fonctionnelle par une fonction totale calculable qui permet de passer d'un indice de la fonction partielle récursive f à un indice de la fonction partielle récursive $\Phi(f)$. Prenons l'exemple de la fonction d'Ackermann, la transformation Φ est donnée par :

$$\begin{aligned}
&\Phi(f)(0,x) = x+2\\
&\Phi(f)(1,0) = 0\\
&\Phi(f)(n+2,0) = 1\\
&\Phi(f)(n+1,x+1) = f(n, f(n+1,x))\,.
\end{aligned}$$

En prenant comme argument un indice i de f, on peut donc faire correspondre une fonction g à 3 arguments entiers :

$$\begin{aligned}
&g(i,0,x) = x+2\\
&g(i,1,0) = 0\\
&g(i,n+2,0) = 1\\
&g(i,n+1,x+1) = \varphi^2(i,n,\varphi^2(i,n+1,x))\,.
\end{aligned}$$

Soit a un indice de la fonction g, par la théorème s^n_m, $s^2_1(a,i)$ est un indice de $\Phi(f)$:

$$\begin{aligned}
&\varphi^2_{s^2_1(a,i)}(0,x) = x+2\\
&\varphi^2_{s^2_1(a,i)}(1,0) = 0\\
&\varphi^2_{s^2_1(a,i)}(n+2,0) = 1\\
&\varphi^2_{s^2_1(a,i)}(n+1,x+1) = \varphi^2_i(n,\varphi^2_i(n+1,x))\ .
\end{aligned}$$

La fonctionnelle Φ est donc représentée par la fonction récursive totale $\alpha : i \mapsto s^2_1(a,i)$. Montrer que la fonction d'Ackermann est récursive, demande de trouver un indice, c'est-à-dire le code d'un programme, pour une fonction qui soit point fixe de la fonctionnelle Φ, soit un e tel que :

$$\varphi^2_{\alpha(e)} = \varphi^2_e\ .$$

Nous allons démontrer le théorème suivant qui permet de conclure.

Théorème 2.4.1 (théorème du point fixe) *Soit $\alpha : \mathbb{N} \to \mathbb{N}$ une fonction totale calculable. Alors pour tout $k \in \mathbb{N}^*$ il existe un entier e tel que :*

$$\phi^k_e = \varphi^k_{\alpha(e)}$$

c'est-à-dire que e et $\alpha(e)$ sont des programmes qui calculent la même fonction (à k arguments).

Pour e obtenu comme ci-dessus, on dira que e est un point fixe de α pour φ^k.

Corollaire 2.4.2 *Il existe une unique fonction vérifiant les équations de la fonction d'Ackermann qui est une fonction totale calculable.*

Démonstration (corollaire). D'après la discussion précédente et le théorème du point fixe il existe une fonction partielle calculable Ack vérifiant les équations définissant la fonction d'Ackermann. Avec ces équations on montre par récurrence (double ou sur ω^2) :

$$\forall (n,x) \in \mathbb{N}^2\ \varphi(n,x)\downarrow\ .$$

L'unicité se vérifie également par récurrence. ■

La méthode utilisée pour la fonction d'Ackermann, s'étend à n'importe quel ensemble d'équations récursives. Le théorème du point fixe est d'ailleurs appelé également *théorème de la récursion*.

Mais le théorème du point fixe permet seulement de montrer l'existence d'une fonction récursive *partielle* φ^k_e vérifiant ces équations. Par exemple l'équation :

$$f(x) = 1 + f(x)$$

se traduit par la recherche d'un point fixe de $\alpha : i \mapsto s^1_1(a,i)$ pour φ, où a est un indice de : $i, c \mapsto \varphi(i,c) + 1$. Un tel point fixe e vérifie pour tout x :

$$\varphi_{\alpha(e)}(x) = \varphi_e(x) + 1\ ;\ \ \varphi_{\alpha(e)} = \varphi_e$$

cette fonction partielle n'est évidemment nulle part définie.

Il est tout à fait possible également d'appliquer la méthode quand l'unicité n'est pas assurée comme pour ces équations :

$$f(0) = f(1)\ ;\ \ f(x+1) = f(x)\ .$$

On a bien l'existence d'un point fixe e de β pour φ (β est obtenue par le théorème s^n_m) vérifiant :

$$\varphi_{\beta(e)}(0) = \varphi_e(1)\ ;\ \ \varphi_{\beta(e)}(x+1) = \varphi_e(x)\ ;\ \ \varphi_{\beta(e)} = \varphi_e\ .$$

La fonction φ_e est la fonction nulle part définie ou une fonction constante, le théorème du point fixe ne permet pas de déterminer de laquelle il s'agit.

Enfin, contrairement à ce que pourrait laisser penser la présentation et les exemples ci-dessus, le théorème du point fixe peut également s'appliquer à une fonction α qui ne correspond pas à une fonctionnelle : rien n'impose que si i et j sont des indices de la même fonction, il en soit de même pour $\alpha(i)$ et $\alpha(j)$.

Le théorème du point fixe permet finalement d'écrire un programme qui prend en argument le texte de ce même programme (enfin son code), c'est en le voyant comme cela que nous l'utiliserons.

2.4.2 Digression : méthode diagonale et point fixe

La démonstration du théorème du point fixe est très courte formellement. Elle s'appuie sur la méthode diagonale. Comme cela peut paraître un peu mystérieux, on commence par une petite digression.

Les applications de la méthode diagonale à la non dénombrabilité ou à la non calculabilité peuvent se voir comme une recherche de point fixe qui échoue. -² Prenons pour exemple le résultat suivant. Si $\psi : \mathbb{N}^2 \mapsto \mathbb{N}$ est une fonction, éventuellement partielle, on note ψ_i la fonction de $\mathbb{N} \to \mathbb{N}$, éventuellement partielle, $\psi_i : x \mapsto \psi(i,x)$.

Proposition 2.4.3 *Soit $\psi : \mathbb{N}^2 \mapsto \mathbb{N}$ une fonction totale calculable. Alors il existe une fonction totale calculable $d : \mathbb{N} \mapsto \mathbb{N}$ telle que $\forall\ i \in \mathbb{N}\ d \neq \psi_i$.*

Quand un ensemble de fonctions calculables F de $\mathbb{N} \to \mathbb{N}$ peut être décrit par une fonction calculable $\psi : \mathbb{N}^2 \to \mathbb{N}$, au sens où $F = \{\psi_i \mid i \in \mathbb{N}\}$, on dit que F est *effectivement énuméré*. Cela se généralise bien-sûr à une arité quelconque. Un exemple est celui de toutes les fonction partielles calculables à un argument, énuméré effectivement par φ^1. Un autre exemple est celui de toutes les fonctions récursives primitives (il faudrait un peu de codage pour le montrer formellement, mais il existe clairement une fonction d'énumération en précisant les fonctions d'évaluation du paragraphe 1.2.1 page 12, ou en utilisant les machines à registres avec programme for (proposition 1.4.2 page 25).

la démonstration de la proposition se fait par diagonalisation, c'est celle déjà esquissée au paragraphe 1.2.1 pour les fonctions récursives primitives. Il suffit de considérer la diagonale : $x \mapsto \psi(x,x)$ et de composer avec une fonction sans point fixe (au sens strict) comme $\alpha(x) = 1 \dot{-} x$, soit $d(x) = 1 \dot{-} \psi(x,x)$. Si d était dans l'énumération on aurait a tel que $\psi(a,a)$ est un point fixe de α, d'où la contradiction.

Pour résumer la méthode diagonale apparaît bien comme une méthode pour montrer que si la fonction construite en appliquant α à la diagonale apparaît dans l'énumération, elle possède un point fixe. Mais ici c'est la contraposée qui a été utilisée : la fonction α a été choisie pour ne pas avoir de point fixe (au sens strict, pas de e tel que $\alpha(e) = e$) donc la fonction construite sur la diagonale n'apparaît pas dans l'énumération.

2.4.3 Démonstration du théorème du point fixe

Soit $k \in \mathbb{N}^*$, et $\alpha : \mathbb{N} \to \mathbb{N}$ une fonction totale calculable. On applique donc la méthode diagonale, mais α est vue comme une transformation de programme, et il s'agit de construire un point fixe e de α au sens beaucoup plus relâché du théorème du point fixe où on demande juste l'égalité des fonctions calculées par les programmes de code e et $\alpha(e)$.

La fonction φ^{k+1} permet d'énumérer toutes les façons de lister effectivement (de façon calculable) des fonctions calculables à k arguments (on note $\overline{x}$ pour $x_1, \ldots, x_k$) :

$$i \mapsto ((j, \overline{x}) \mapsto \varphi^{k+1}(i, j, \overline{x}))$$

Ce qu'on veut énumérer, ce sont des (codes de) programmes pour ces fonctions calculables à k arguments, ce qui est donné par les fonctions :

$$i \mapsto \mathrm{s}_1^k(i, j)\ .$$

Quand on prend la diagonale à laquelle on applique α, on obtient la fonction :

$$i \mapsto \alpha(\mathrm{s}_1^k(i,i))$$

qui liste des programmes pour certaines fonctions à k arguments, et donc on va la retrouver dans l'énumération, au sens où il existe a tel que pour tout $i \in \mathbb{N}$:

$$\varphi^k_{\mathrm{s}_1^k(a,i)} = \varphi^k_{\alpha(\mathrm{s}_1^k(i,i))}\ .$$

Plus formellement, on prend pour a un indice de la fonction à $k+1$ arguments $i, \overline{x} \mapsto \varphi^k(\alpha(\mathrm{s}_1^k(i,i)), \overline{x})$, et par le théorème s_m^n, on a bien pour tout i et tout $\overline{x}$:

$$\varphi^k(\mathrm{s}_1^k(a,i), \overline{x}) = \varphi^k(\alpha(\mathrm{s}_1^k(i,i)), \overline{x})\ .$$

On prend $i = a$ et on obtient bien un point fixe de α pour φ^k :

$$\varphi^k_{s^k_1(a,a)} = \varphi^k_{\alpha(s^k_1(a,a))} .$$ ■

2.4.4 Raffinements

On peut donner divers raffinements du théorème du point fixe, comme le suivant.

Théorème 2.4.4 (théorème du point fixe avec paramètres) *Soit $k \in \mathbb{N}^*$, soit $\alpha : \mathbb{N}^{p+1} \to \mathbb{N}$ une fonction totale calculable. Alors il existe une fonction totale calculable $h : \mathbb{N}^p \to \mathbb{N}$ qui calcule en fonction de $y_1, \ldots, y_p$ un point fixe pour la fonction : $t \mapsto \alpha(y_1, \ldots, y_p, t)$, soit :*

$$\varphi^k_{\alpha(y_1,\ldots,y_p,h(y_1,\ldots,y_p))} = \varphi^k_{h(y_1,\ldots,y_p)}$$

Démonstration. Il suffit de reprendre la démonstration précédente. On note $\overline{y}$ pour $y_1, \ldots, y_p$, $\overline{x}$ pour $x_1, \ldots, x_k$. Soit a un indice de la fonction : $i, \overline{y}, \overline{x} \mapsto \phi^k(\alpha(\overline{y}, s^k_{p+1}(i, i, \overline{y})), \overline{x})$. On a pour tout i :

$$\forall \overline{y} \in \mathbb{N}^p \ \forall \overline{x} \in \mathbb{N}^k \ \varphi^k(s^k_{p+1}(a, i, \overline{y}), \overline{x}) = \varphi^k(\alpha(s^k_{p+1}(i, i, \overline{y})), \overline{x})$$

et en en faisant $i = a$, on montrer que la fonction définie par $h(\overline{y}) = s^k_{p+1}(a, a, \overline{y})$ calcule bien un point fixe pour φ^k de la fonction voulue. ■

Exercice 18 On peut montrer que la construction du point fixe est effective au sens qui suit.

1. Montrer que pour tout $k \in \mathbb{N}^*$ il existe une fonction g telle que, si $\alpha = \varphi^k_i$ est une fonction totale, alors :
$$\phi^k_{g(i)} = \varphi^k_{\alpha(g(i))}$$
(le point fixe de α est calculable en fonction d'un programme pour α).
2. Énoncer et démontrer le résultat analogue à celui énoncé à la question précédente pour le théorème du point fixe dans le cas du théorème de point fixe avec paramètres.

2.5 Système d'indices acceptable

2.5.1 Définitions

Les fonctions φ^k ont été construites de façon très dépendante du modèle de calcul choisi et du codage choisi pour ces machines. Cependant on conçoit bien que de changer le codage ou même le modèle de calcul devrait conduire aux mêmes résultats.

On peut remarquer que les théorèmes du point fixe, sans ou avec paramètres, ont été démontrés uniquement en utilisant que les fonctions d'énumérations sont calculables et la propriété de paramétrisation (théorème s^n_m).

Appellons *système d'indices* pour les fonctions partielles calculables la donnée pour chaque $k \in \mathbb{N}^*$ d'une famille de fonctions $(\psi^k_i)_{i\in\mathbb{N}}$ partielles calculables d'arité k.

Le but de cette section est de montrer qu'il est possible de ramener de façon calculable un système d'indices au système particulier des φ^k pourvu qu'il respecte certaines propriéts caractéristiques, essentiellement le théorème d'énumération 2.2.3 page 35 et le théorème s^n_m 2.2.6 page 36. La conséquence est que tous les systèmes que l'on pourrait construire de façon analogue seront équivalents. Précisons d'abord quelles sont ces propriétés caractéristiques des fonctions universelles.

Définition 2.5.1 Un système d'indices pour les fonctions partielles calculables, $(\psi^k_i)_{i\in\mathbb{N}}$, $k \in \mathbb{N}^*$, est appelé *système d'indices acceptable* quand il vérifie les trois propriété suivantes :

Surjectivité : pour chaque $k \in \mathbb{N}^*$, la famille $(\psi^k_i)_{i\in\mathbb{N}}$ parcourt toutes les fonctions partielles calculables à k arguments ;

Propriété d'énumération : pour chaque $k \in \mathbb{N}^*$, il existe un entier a qui est un indice de : $i, \overline{x} \mapsto \psi_i^k(\overline{x})$, c'est-à-dire :

$$\forall i \in \mathbb{N}\ \forall \overline{x} \in \mathbb{N}^k\ \psi_a^{k+1}(i, \overline{x}) = \psi_i^k(\overline{x})\ .$$

Propriété de paramétrisation : pour chaques $k, p \in \mathbb{N}^*$, il existe une fonction totale calculable σ_p^k telle que :

$$\forall i \in \mathbb{N}\ \forall \overline{y} \in \mathbb{N}^p\ \forall \overline{x} \in \mathbb{N}^k\ \psi_i^{p+k}(\overline{y}, \overline{x}) = \psi^k_{\sigma_p^k(i,\overline{y})}(\overline{x})\ .$$

La propriété d'énumération exprime bien que l'énumération des fonctions partielles calculables à k arguments est elle-même « calculable » (au sens précis donné par la définition).
Les $(\varphi_i^k)_{i\in\mathbb{N}}$, $k \in \mathbb{N}^*$ forment un système acceptable d'indices par les théorèmes 2.2.3 et 2.2.6.

2.5.2 Point fixe et bourrage

Les démonstrations du théorème du point fixe et du théorème du point fixe avec paramètres n'utilisent clairement que ces trois propriétés.

Proposition 2.5.2 *Tout système d'indices acceptable vérifie le théorème du point fixe et le théorème du point fixe avec paramètres.*

Dans le cas des φ^k, nous avons vu (exercice 15 page 35) qu'une fonction a forcément une infinité d'indices, et même qu'il est possible d'énumérer de façon calculable une infinité d'indices de cette fonction. La méthode indiquée s'appuie sur le modèle de calcul (ajouter des instructions inutiles) et le codage (qui croît en fonction de la longueur du programme de la machine). Nous allons montrer que cette propriété est vérifiée pour tout système acceptable.

Lemme 2.5.3 (lemme de bourrage) *Soit $(\psi_i^k)_{i\in\mathbb{N}}$, $k \in \mathbb{N}^*$, un système acceptable d'indices. Pour chaque $k \in \mathbb{N}^*$,*

1. *il existe une fonction totale calculable $u_k : \mathbb{N} \to \mathbb{N}$ telle que*

$$\psi^k_{u(i)} = \psi_i^k \text{ et pour tout } i,\ u(i) > i\ ;$$

2. *il existe une fonction totale calculable $v : \mathbb{N}^2 \to \mathbb{N}$ telle que v est strictement croissante en la seconde variable et pour tout entier i, $\psi^k_{v(i,n)} = \psi_i^k$.*

Démonstration. L'item **2** se déduit du **1** pour le même k, par récurrence primitive :

$$v(i,0) = i\ ;\quad v(i, n+1) = u(v(i,n), n)\ .$$

La démonstration du **1** utilise le théorème du point fixe avec paramètres[6].

Les propriétés d'énumération et de paramétrisation permettent de définir une fonction totale calculable $\alpha : \mathbb{N}^3 \to \mathbb{N}$ dont un point fixe (pour le système des (ψ_i^k) est donné par $h : \mathbb{N}^2 \to \mathbb{N}$ totale calculable de façon à vérifier :

$$\psi^k_{\alpha(i,n,h(i,n))} = \begin{cases} \psi_i^k & \text{si } h(i,n) > i \\ \lambda\overline{x}.n & \text{si } h(i,n) \leq i \end{cases} \quad \text{et } \psi^k_{\alpha(i,n,h(i,n))} = \psi^k_{h(i,n)}$$

où $\lambda\overline{x}.n$ est la fonction à k arguments constante égale à n. Pour un i donné, on voit que

$$\{\psi^k_{h(i,n)} \mid n \in \mathbb{N}\} \subset \{\lambda\overline{x}.n \mid h(i,n) \leq i\} \cup \{\psi_i^k\}\ .$$

Sur $\{n \mid h(i,n) \leq i\}$, $n \mapsto \psi^k_{h(i,n)}$ est injective, donc, en passant aux indices, a fortiori : $n \mapsto h(i,n)$ est injective, et elle est à valeurs dans $\{0, \ldots, i\}$. L'ensemble $\{n \mid h(i,n) \leq i\}$ est donc fini : il existe une infinité de n tels que $h(i,n) > i$. La fonction calculable u définie par :

$$u(i) = h(i, \mu n.h(i,n) > i\}$$

6. C'est un usage très différente de ceux donnés jusqu'à présent, qui visaient à montrer qu'une fonction définie par un système d'équations récursives était calculable : ici il ne s'agit pas de montrer qu'une nouvelle fonction est calculable, mais de trouver un nouvel indice pour une fonction.

est toujours définie, et on a bien $\psi_{u(i)} = \psi_i$ avec $u(i) > i$.

On peut donner un peu plus de détails pour la définition de α. On cherche α totale calculable vérifiant :

$$\psi^k_{\alpha(i,n,t)} = \begin{cases} \psi^k_i & \text{si } t > i \\ \lambda \overline{x}.n & \text{si } t \leq i \end{cases}$$

qui donne h par le théorème du point fixe avec paramètres.

Soit a tel que $\psi^k_i(\overline{x}) = \psi^{k+1}_a(i,\overline{x})$ (propriété d'énumération). La fonction α s'obtient par paramétrisation (théorème s^n_m) à partir d'un indice de la fonction partielle calculable définie par :

$$f(i,n,t,\overline{x}) = \begin{cases} \psi^{k+1}_a(i,\overline{x}) & \text{si } t > i \\ n & \text{si } t \leq i \end{cases}$$

■

2.5.3 Équivalence entre systèmes acceptables

Proposition 2.5.4 *Un système d'indices $(\psi^k_i)_{i\in\mathbb{N}}$, $k \in \mathbb{N}^*$ pour les fonctions partielles calculables est acceptable si et seulement si pour tout $k \in \mathbb{N}^*$ il existe deux fonctions totales calculables f_k, $g_k : \mathbb{N} \to \mathbb{N}$ telles que pour tout $i \in \mathbb{N}$:*

$$\psi^k_i = \varphi^k_{f_k(i)} \; ; \; \varphi^k_i = \psi^k_{g_k(i)} \; . \qquad (*)$$

Démonstration.

Sens direct : Il suffit d'indiquer comment construire f_k, la construction de g_k se faisant de façon symétrique. On a

$$\begin{array}{rcll} \psi^k_i(\overline{x}) & = & \psi^{k+1}_a(i,\overline{x}) & \text{par propriété d'énumération} \\ & = & \varphi^{k+1}_b(i,\overline{x}) & \text{par surjectivité} \\ & = & \varphi^k_{s^1_k(b,i)}(\overline{x}) & \text{par paramétrisation.} \end{array}$$

Réciproque : soit un système d'indices $(\psi^k_i)_{i\in\mathbb{N}}$ avec pour chaque k des fonctions f_k et g_k vérifiant les deux égalités $(*)$ ci-dessus.

— La surjectivité du système d'indices se déduit de celle des $(\varphi^k_i)_{i\in\mathbb{N}}$ en utilisant $\varphi^k_e = \psi^k_{g_k(e)}$.

— la propriété d'énumération se démontre par le théorème d'énumération 2.2.3, soit $\overline{x} \in \mathbb{N}^k$

$$\psi^k_i(\overline{x}) = \varphi^k(g_k(i),\overline{x}) = \varphi^{k+1}_a(i,\overline{x}) = \psi^{k+1}_{f_{k+1}(a)}(i,\overline{x})$$

où a est un indice de : $i,\overline{x} \mapsto \varphi^k(g_k(i),\overline{x})$) ;

— La propriété de paramétrisation se démontre par le théorème s^n_m 2.2.6, soient $\overline{y} \in \mathbb{N}^k$, $\overline{x} \in \mathbb{N}^p$:

$$\psi^{p+k}_i(\overline{y},\overline{x}) = \varphi^{p+k}(f_{p+k}k(i),\overline{y},\overline{x}) = \varphi^k(s^k_{p+1}(a,i,\overline{y}),\overline{x}) = \psi^k_{f(s^k_{p+1}(a,i,\overline{y}))}(\overline{x})$$

où a est un indice de : $i,\overline{x} \mapsto \varphi^{p+k+1}(f_{p+k}(i),\overline{y},\overline{x})$). ■

Proposition 2.5.5 *Un système d'indices pour les fonctions partielles calculables $(\psi^k_i)_{i\in\mathbb{N}}$, $k \in \mathbb{N}^*$, est acceptable si et seulement si pour tout $k \in \mathbb{N}^*$ il existe une fonction totale bijective calculable $h : \mathbb{N} \to \mathbb{N}$ telle que pour tout $i \in \mathbb{N}$:*

$$\psi^k_i = \varphi^k_{h(i)} \; .$$

Démonstration. La réciproque est un cas particulier de la réciproque de la proposition précédente, sachant que si h est totale calculable et bijective, par minimisation h^{-1} est également totale calculable.

Pour le sens direct, on utilise pour chaque $k \in \mathbb{N}^*$ les fonctions f_k et g_k données par la proposition précédente, dont on modifie l'image par le lemme de bourrage, que l'on note f et g (k étant fixé pour la suite de la démonstration). Par exemple on obtient facilement deux fonctions $\tilde{f}$ et $\tilde{g}$ injectives vérifiant la proposition précédente, en prenant pour $\tilde{f}(i)$ un indice de la fonction d'indice $f(i)$ différent de tous les $\tilde{f}(j)$, $0 \leq j < i$, ce qui est possible par le lemme de bourrage pour (ψ^k_i). De même pour g avec le lemme de bourrage pour (φ^k_i).

Pour obtenir une fonction bijective h il suffit de mener alternativement ces deux constructions, c'est-à-dire que l'on construit alternativement h (à partir de f) et h^{-1} (à partir de g).

Plus précisément on construit par récurrence sur la suite des valeurs une fonction calculable l qui énumère les (codes des) couples du graphe de la bijection cherchée.

— Au rang pair, on ajoute le couple constitué du plus petit entier i qui n'a pas encore d'image, et d'une image j choisie par le lemme de bourrage pour (ψ_i^k) de façon que $\psi_j^k = \psi_{f(i)}^k$ et j est différent de toutes les images déjà atteintes.
— Au rang impair on fait l'inverse, c'est-à-dire que l'on ajoute le couple constitué du plus petit entier j qui n'a pas encore d'antécédent, et d'un antécédent i choisi par le lemme de bourrage pour φ_i^k de façon que $\varphi_i^k = \varphi_{g(j)}^k$ et i est différent de toutes les images déjà atteintes.

On appelle ν la fonction donnée par le **2** du lemme de bourrage 2.5.3 pour (φ_i^k) et ν' celle donnée de la même façon pour (ψ_i^k), on obtient :

$$\begin{array}{l} l(0) = \langle 0, f(0)\rangle \\ l(2n+1) = \langle \nu'(g(j_n), \mu p.[\forall\, t \le 2n\ \pi_1(l(t)) \neq \nu'(g(j_n), p)]), j_n\rangle \quad \text{où} \quad j_n = \mu j.[\forall\, t \le 2n\ \pi_2(l(t)) \neq j] \\ l(2n+2) = \langle i_n, \nu(f(i_n), \mu p.[\forall\, t \le 2n+1\ \pi_2(l(t)) \neq \nu(f(i_n), p)])\rangle \quad \text{où} \quad i_n = \mu i.[\forall\, t \le 2n+1\ \pi_1(l(t)) \neq i]\ . \end{array}$$

la fonction calculable énumère bien les couples d'une fonction qui est bijective par construction, et qui a la propriété cherchée par choix de f et g.

On définit ensuite facilement h à partir de l :

$$h(i) = \pi_2(l(\mu n.\pi_1(l(n)) = i))\ .$$ ■

2.6 Réductions

Les preuves de l'indécidabilité du problème de l'arrêt (théorème 2.3.9) ou du théorème de Rice 2.3.13 sont d'une nature particulière : elles sont obtenues par *réduction* à un autre problème. On peut formaliser la notion de réduction de plusieurs manières, plus ou moins libérales suivant les moyens mis en jeu.

2.6.1 Réduction *many-one*

Définition 2.6.1 (réduction many-one) Soient $A \subseteq \mathbb{N}^k$ et $B \subseteq \mathbb{N}^l$. On dit que A se réduit à B par *réduction many-one*[7], noté $A \le_m B$, s'il existe une fonction[8] $f : \mathbb{N}^k \to \mathbb{N}^l$ totale calculable, au sens où chacune de ses composantes est totale calculable, telle que pour tout $\overline{x} \in \mathbb{N}^k$:

$$\overline{x} \in A \iff f(\overline{x}) \in B\ .$$

La réduction many-one est aussi notée en abrégé *m-réduction*.

Clairement, la réduction *transmet* la difficulté : si A n'est pas décidable et que A se réduit à B par réduction many one, alors B n'est pas décidable. Sinon, un algorithme évident pour énumérer A serait obtenu, sur toute entrée n, en calculant $f(n)$ et en testant si $f(n) \in B$. Dans la démontration du théorème de Rice 2.3.5, on a réduit par $\le_m$ l'ensemble K associé au problème de l'arrêt diagonal à un ensemble F vérifiant les hypothèses du théorème, la fonction de réduction étant fournie par le théorème s_m^n.

L'argument montre aussi que l'on peut utiliser *positivement* la réduction pour décider A si on sait décider B. Donnons en résumé la proposition suivante.

Proposition 2.6.2 *Soient $A \subset \mathbb{N}^k$, $B \subset \mathbb{N}^l$ tels que $A \le_m B$:*
— *si B est semi-décidable, alors A est semi-décidable ;*
— *si B est décidable, alors A est décidable.*

Démonstration. Soit $f = (f_1, \ldots, f_l) : \mathbb{N}^k \to \mathbb{N}^l$ la fonction de réduction, en particulier les f_i sont toutes calculables totales. Supposons B semi-décidable et domaine de la fonction $g : \mathbb{N}^l \to \mathbb{N}$, alors A est le domaine de la fonction partielle calculable $g \circ f$. Supposons B décidable de fonction caractéristique $\chi_B : \mathbb{N}^l \to \mathbb{N}$, alors la fonction caractéristique de A est $\chi_B \circ f$ qui est calculable. ■

7. La réduction a été appelée *many-one* car f est une fonction quelconque, qui peut réduire plusieurs instances du problème de l'appartenance à A à une même instance du problème de l'appartenance à B. Nous n'étudierons pas la réduction *one-one* où l'on demande à f d'être de plus injective.

8. On peut voir une fonction $f : \mathbb{N}^k \to \mathbb{N}^l$ comme une collection de fonctions $f_1, ..., f_l : \mathbb{N}^k \to \mathbb{N}$

Il est également immédiat par composition que la relation $\leq_m$ est transitive.

Fait 2.6.3 *La relation $\leq_m$ est un un préordre (relation réflexive et transitive).*

On note $\equiv_m$ la relation d'équivalence associée au préordre $\leq_m$: $A \equiv_m B$ ssi $A \leq_m B$ et $B \leq_m A$.

On parle également, et avec les mêmes notations, de réduction many-one entre prédicats ou problèmes.

Une dernière caractéristique fondamentale des réductions est qu'elles permettent parfois de montrer que certains ensembles ou prédicats sont représentatifs des problèmes les plus difficiles d'une classe autrement dit qu'ils sont *complets* pour cette classe.

Définition 2.6.4 Un ensemble semi-décidable A est dit m-complet (sous-entendu pour les semi-décidables) quand, pour tout ensemble semi-décidable B, $B \leq_m A$.

On peut toujours se ramener par codage aux sous-ensembles de $\mathbb{N}$, et en fait, quand il s'agit d'ensembles m-complets, on s'intéressera la plupart du temps aux sous-ensembles de $\mathbb{N}$. Par codage (lemme 2.3.2) on a le lemme suivant.

Lemme 2.6.5 *Un sous-ensemble semi-décidable A de $\mathbb{N}$ est m-complet (sous-entendu pour les semi-décidables) quand, pour tout sous-ensemble semi-décidable B de $\mathbb{N}$, $B \leq_m A$.*

Clairement un ensemble décidable ne peut être m-complet. Il s'avère que les ensembles semi-décidables non décidables rencontrés jusqu'à présent sont m-complets, à commencer par celui de l'arrêt diagonal.

Proposition 2.6.6 *Un ensemble A est semi-décidable si et seulement si $A \leq_m K$, où K est l'ensemble associé au problème* DIAG*, autrement dit l'ensemble K, est m-complet.*

Démonstration. Si $A \leq_m K$, alors A est semi-décidable par la proposition2.6.2 car K est semi-décidable.

La preuve de réduction vers le problème ARRET serait plus immédiate. Pour DIAG un petit détour est nécessaire. Soit $A \subset \mathbb{N}$ (on utilise le lemme) semi-décidable, donc domaine d'une fonction partielle calculable $f : \mathbb{N} \to N$. Pour pouvoir se ramener au problème de l'arrêt diagonal, il suffit d'ajouter un argument dont ne dépend pas la fonction : posons $h(x, y) = f(x)$ qui est partielle calculable. Soit m un indice de h. Par le théorème s^n_m, on a :

$$\varphi(m, x, y) = \varphi(s^1_1(m, x), y)$$

Par définition de h, $x \in A \Leftrightarrow \varphi(s^1_1(m, x), y) \downarrow$ pour n'importe quel y, en particulier pour $y := s^1_1(m, x)$. On a bien montré :

$$x \in A \Leftrightarrow s^1_1(m, x) \in K\,.$$

■

Corollaire 2.6.7 *Le problème de l'arrêt et le problème de l'arrêt en 0 sont m-complets.*

Démonstration. Les ensembles associés sont semi-décidables, et K se réduit à chacun d'entre eux, par les preuves des théorèmes 2.3.9 et 2.3.10. ■

2.6.2 Calculabilité relative

Pour définir une notion de réduction plus générale, la réduction de Turing, on introduit d'abord une notion de *calculabilité relative* à une fonction ou un ensemble (ou prédicat).

Définition 2.6.8 Soit g une fonction totale, l'ensemble des fonctions partielles calculables relativement à g, plus brièvement calculables en g, est le plus petit sous-ensemble de fonctions :

1. contenant les fonctions de base usuelles : la fonction nulle, la fonction successeur et les projections ainsi que la fonction g ;
2. clos par schéma de composition, récursion primitive et schéma de minimisation.

Une fonction totale calculable en g est une fonction calculable en g qui est totale.
La définition s'étend aux ensembles ou prédicats.

— Une fonction calculable relativement à un prédicat est une fonction calculable relativement à la fonction caractéristique de ce prédicat. De même pour une fonction calculable relativement à un ensemble.
— Un prédicat ou un ensemble est décidable ou calculable en une fonction, un prédicat ou un ensemble A quand sa fonction caractéristique l'est.

La définition précédente exprime que f est calculable modulo « un *oracle* » qu'on peut interroger en lui fournissant des arguments pour g et qui renvoie alors la valeur de g pour ces arguments. L'oracle est une « boîte noire ». La fonction f n'est effectivement calculable que si g l'est. On peut imaginer par exemple que, dans un programme, l'instruction qui demande de saisir une valeur par clavier est une espèce d'appel à un oracle.

Comme déjà vu, à une fonction $g : \mathbb{N}^k \to \mathbb{N}$, on peut associer par codage des k-uplets une fonction $g^{\langle\rangle} : \mathbb{N} \to \mathbb{N}$, telle que $g^{\langle\rangle}(\langle x_1, \ldots, x_k\rangle) = g(x_1, \ldots, x_k)$. On laisse en exercice la démonstration des résultats suivant.

Fait 2.6.9
— *La fonction g est calculable en $g^{\langle\rangle}$;*
— *La fonction $g^{\langle\rangle}$ est calculable en g ;*
— *le graphe de la fonction g est calculable en la fonction g ;*
— *la fonction g est calculable en son graphe.*

On peut donc, sans perte de généralité, ne s'intéresser qu'aux fonctions calculables en une fonction (totale) à un argument, voire même aux fonctions calculables en un sous-ensemble A de $\mathbb{N}$.

On peut introduire une notion de machine à registres avec oracle g. Un façon de faire (il n'y en pas pas qu'une !) est d'étendre la définition donnée section 1.3 page 18 en ajoutant une nouvelle instruction oracle(j). Si l'oracle g est une fonction à k arguments, l'effet de l'instruction oracle(j) est de recopier dans le registre R_j l'entier $g(x_1, \ldots, x_k)$ où $(x_1, \ldots, x_k)$ sont les contenus des registres $R_1, \ldots, R_k$ (avant appel à l'instruction), en complétant à 0 s'il n'y a pas assez de registres.

On définit ensuite introduire ensuite exactement de la même façon les programmes structurés avec oracle. La démonstration qu'une fonction partielle calculable en g est calculable par machine à registres avec oracle g est quasi-identique à celle de la proposition 1.4.1 pour les machines ordinaires.

Le codage de l'état d'une machine avec oracle est identique à celui d'une machine ordinaire. Pour coder une machine avec oracle, il suffit d'intégrer la nouvelle instruction (codée par exemple par $\langle 4, i\rangle$, voir le codage des instructions page 26).

Le codage du calcul se fait également de la même façon. La fonction d'initialisation et le prédicat d'arrêt se définissent de façon identique. Seule la fonction de transition doit intégrer l'appel à l'oracle : elle est totale calculable en g. Les résultats de la section 1.3 et ceux des sections précédentes du présent chapitre (2.2 à 2.4 en particulier) se généralisent donc naturellement aux fonctions calculables en g, avec des démonstrations quasi identiques. En particulier :

Proposition 2.6.10 *Une fonction (partielle) est calculable en g si et seulement si elle est calculable sur machine à registres avec oracle g.*

Beaucoup de résultats sur les fonctions récursives se généralisent au cas des fonctions récursives relativement à une fonction ou un prédicat (forme normale de Kleene, notion d'indice de machine etc...).

Voyons la mise sous forme normale de Kleene relativisée (voir section 2.2.1 page 34), dont on donne ci-dessous une version qui met en évidence qu'une fonction partielle calculable en g n'a besoin pour chaque entrée que d'un nombre fini d'appels à l'oracle.

Soit $g : \mathbb{N} \to \mathbb{N}$ une fonction totale calculable et $y \in \mathbb{N}$. Remarquons que comme g est totale, la fonction $s \mapsto [g(0); \ldots; g(s)]$ est totale, et qu'elle est clairement calculable en g.

Proposition 2.6.11 (forme normale de Kleene) *Pour chaque entier n, il existe un prédicat primitif récursif $T^{g,n} \subseteq \mathbb{N}^{n+3}$ et une fonction récursive primitive $U : \mathbb{N} \to \mathbb{N}$ tels que pour toute fonction $f : \mathbb{N}^n \to N$ partielle calculable en une fonction totale g, il existe un entier m tel que :*

1. $\exists s\ T^{g,n}[m, x_1, \ldots, x_n, [g(0); \ldots; g(s)], s] \Leftrightarrow f(x_1, \ldots, x_n)\downarrow$

2. $T^{g,n}[m, x_1, \ldots, x_n, [g(0); \ldots; g(s)], s] \Rightarrow U(s) = f(x_1, \ldots, x_n)$

3. $\left(T^{g,n}[m,x_1,\ldots,x_n,[g(0);\ldots;g(s)],s] \text{ et } T^{g,n}[m,x_1,\ldots,x_n,[g(0);\ldots;g(s')],s']\right) \Rightarrow s'=s$.

Démonstration. Quand le calcul termine, il n'y a qu'un nombre fini d'appels à l'oracle, et donc l'ensemble des entiers auquel l'oracle fait appel est borné. Chaque appel à la fonction g se fait sur le contenu de R_1 qui est un entier inférieur au code de l'état donc au code s de la suite des codes des états. Chaque appel à l'oracle se calcule donc bien de façon récursive primitive en fonction de $[g(0);\ldots;g(s)]$, plus formellement on doit définir une fonction de transition (voir la section 1.5.2 page 27) avec un argument supplémentaire (interprété comme représentant une liste $[g(0);\ldots;g(i)]$) qui est utilisé quand l'instruction est un appel à l'oracle. Le reste de la démonstration est analogue à celle de la proposition 2.2.1. ■

La propriété d'énumération se généralise à la calculabilité relative, à l'image de ce que l'on a fait pour les fonctions calculables. On définit la famille universelle $\left(\varphi^{g,k}\right)_{n\in\mathbb{N}^*}$ ($\varphi_i^{A,k}$ quand g est la fonction caractéristique de l'ensemble A) des fonctions partielles calculables en g :

$$\varphi^{g,k}(i,\overline{x}) = \varphi_i^{g,k}(\overline{x}) = U(\mu s.\left(T^{g,k}[i,\overline{x},[g(0);\ldots;g(s)],s]\right)).$$

où $\varphi_i^{g,k}$ est la fonction partielle à k variables calculable en g d'indice i.

Proposition 2.6.12 *Soit $g:\mathbb{N}\to\mathbb{N}$ une fonction totale. Les fonctions $\varphi^{g,n}$ sont des fonctions partielles calculables en g à $n+1$ arguments qui énumèrent toutes les fonctions partielles calculables en g, au sens où pour toute fonction partielle $f:\mathbb{N}^k\to\mathbb{N}$ calculable en g, il existe au moins un entier i, tel que i est un indice de f pour $\varphi^{g,n}$, soit $f=\varphi_i^{g,n}$.*

La propriété de paramétrisation se démontre de la même façon que dans le cas des fonctions calculables.

On se restreint maintenant (sans réelle perte de généralité, comme on l'a vu) à la calculabilité relativement à un ensemble $A\subset\mathbb{N}$ (g ci-dessus est la fonction caractéristique de A). Un ensemble est *semi-décidable*, ou *effectivement énumérable en A* quand il est le domaine d'une fonction calculable en A. On a de la même façon une énumération des sous-ensembles semi-décidables en A, et on note $W_i^{A,n}$ le domaine de la fonction partielle $\varphi_i^{A,n}$.

Proposition 2.6.13 *Pour tout $i\in\mathbb{N}$:*

$$W_i^{A,n} = \left\{\overline{x}\in\mathbb{N}^n \mid \exists s\exists l\left[T^{A,n}[i,\overline{x},l,s] \text{ et } \forall j\le s\left((\text{nth}(l,j)=0\Rightarrow j\notin A) \text{ et } (\text{nth}(l,j)=1\Rightarrow j\in A)\right)\right]\right\} .$$

Et on a de la même façon la version relativisée de l'indécidabilité du problème de l'arrêt (diagonal).

Proposition 2.6.14 *L'ensemble $K^A=\left\{i\in\mathbb{N}\mid i\in W_i^A\right\}$, qui est effectivement énumérable en A, n'est pas calculable en A.*

Terminons par la version relativisée à A de la proposition 2.3.6 page 39, dont là encore il suffit de calquer la démonstration.

Proposition 2.6.15 *Un ensemble $B\subset\mathbb{N}^k$ est calculable en A si et seulement si B et son complémentaire B^c sont calculables en A.*

2.6.3 Réduction de Turing

On définit maintenant la réduction dite de Turing.

Définition 2.6.16 On dit qu'un ensemble A se réduit à un ensemble B, quand A est décidable en B i.e. quand la fonction caractéristique de A est calculable en celle de B, et on note $A\le_T B$. On parle aussi de T-réduction.

Une fonction calculable en un ensemble décidable B est évidemment une fonction calculable. On en déduit :

Proposition 2.6.17 *Si B est décidable et $A \leq_T B$, alors A est décidable.*

On ne peut plus généraliser aux semi-décidables, contrairement à la réduction many-one. En effet la fonction caractéristique du complémentaire d'un ensemble A se calcule de façon évidente en la fonction caractéristique de A. En particulier $K^c \leq_T K$ (on rappelle que $K = \{i \mid \varphi(i,i)\downarrow\}$ est l'ensemble associé au problème diagonal de l'arrêt). Or K^c n'est pas semi-décidable.

La réduction many-one est clairement un cas particulier de la réduction de Turing, celui ou l'appel à l'oracle se fait une seule fois en fin de calcul. Mais, comme $K^c \not\leq_m K$, la réduction de Turing est vraiment plus générale que la réduction many-one. En résumé :

Fait 2.6.18

1. *Si $A \leq_m B$ alors $A \leq_T B$;*
2. $A^c \leq_T A$.

Définition 2.6.19 Un ensemble A est dit complet pour la réduction de Turing, ou Turing-complet, ou T-complet, sous-entendu : pour les semi-décidables, quand pour tout ensemble semi-décidable B, $B \leq_T A$.

De K m-complet (proposition 2.6.6 page 51) on déduit immédiatement :

Proposition 2.6.20 *Si un prédicat A est semi-décidable alors $A \leq_T K$ et $A^c \leq_T K$, en particulier K est Turing-complet.*

La réciproque de la proposition 2.6.6 ne peut passer à la réduction de Turing, puisque la classe des ensembles A tels que $A \leq_T K$ comprend non seulement les semi-décidables mais aussi leurs complémentaires. On verra au chapitre suivant qu'il s'agit de la classe des ensemble Δ^0_2, qui contient aussi des ensembles qui ne sont ni semi-décidables, ni complémentaires de semi-décidables.

De même que pour les réductions many-one, on note $A \equiv_T B$ ssi $A \leq_T B$ et $B \leq_T A$. La relation $\equiv_T$, tout comme $\equiv_m$ est une relation d'équivalence.

Chapitre 3

Arithmétique

3.1 Définissabilité dans $\mathbb{N}$

Dans cette section on s'intéresse aux sous-ensembles de $\mathbb{N}$ (ou de $\mathbb{N}^p$) que l'on peut définir par une formule de l'arithmétique, et on fait le rapport avec la calculabilité.

On a vu (proposition 2.3.3 page 38) que les ensembles semi-décidables sont les projetés des ensembles récursifs primitifs : un sous-ensemble A de $\mathbb{N}$ est semi-décidable si et seulement si il existe une fonction récursive primitive f telle que :

$$n \in A \text{ ssi } \exists d\ f(n,d) = 0\,.$$

ce résultat reste vrai pour des classes de fonctions plus restreintes. Il n'est pas nécessaire d'utiliser toute la puissance des fonctions récursives primitives pour ce résultat. Par exemple en utilisant essentiellement les mêmes méthodes, on peut demander que f soit élémentaire au sens de Kalmar (c'est-à-dire qu'elle n'utilise pas plus de trois schémas de récurrence imbriqués).

Si on autorise plusieurs quantifications existentielles, on montre que la fonction f peut être choisie polynomiale à coefficients dans $\mathbb{Z}$. Le prédicat s'exprime alors par plusieurs quantifications existentielles sur une égalité entre polynômes à coefficients dans $\mathbb{N}$. C'est le théorème de Matiassevitch (1970), dont la démonstration demande des codages astucieux, nettement plus complexes que ceux déjà présentés.

Cependant, sans aller jusqu'aux équations polynomiales, on peut montrer qu'un ensemble semi-décidable peut se définir par une formule $\exists d\ \Phi[n,d]$ où Φ est une formule qui, en plus du zéro et du successeur, n'utilise que l'addition et la multiplication comme symboles de fonctions (les termes sont donc des polynômes), les opérateurs booléens, et les quantifications bornées.

Ce résultat suffit pour les théorèmes d'incomplétude et d'indécidabilité de l'arithmétique, et se trouve déjà implicitement dans l'article de Gödel de 1931 (il le démontre pour les projetés d'ensembles récursifs primitifs). Il s'agit essentiellement de montrer que l'on peut simuler les définitions par récurrence dans l'arithmétique.

3.1.1 formules et ensembles Σ_0 et Σ

Le langage est celui de l'arithmétique, il a pour signature $(0, \mathrm{s}, +, \times, \leq)$.

La classe Σ_0

Une formule est Σ_0 (ou Π_0) quand toutes ses quantifications sont bornées, c'est-à-dire apparaissent sous la forme $\exists x \leq t\ \Phi$ ou $\forall x \leq t\ \Phi$, où t est un terme. On rappelle que

$$\exists x \leq t\ \Phi \equiv_d \exists x(x \leq t \wedge \Phi)\ ;\ \ \forall x \leq t\ \Phi \equiv_d \forall x(x \leq t \to \Phi).$$

La classe des formules Σ_0 peut donc se définir inductivement : c'est la plus petite classe de formules

i. qui contient les formules atomiques (égalités et inégalités polynomiales) ;

ii. qui est close par opérations booléennes (négation, disjonction et conjonction) ;

iii. qui est close par quantification universelle et existentielle bornée.

Un sous-ensemble de $\mathbb{N}^k$, ou un prédicats d'arité k sur n, est dit Σ_0, quand il est définissable par une formule Σ_0. Pour, par exemple, un ensemble E, cela signifie qu'il existe une formule Σ_0, soit $\Phi[x_1,\dots,x_k]$, telle que pour tous $n_1,\dots,n_k$ entiers

$$(n_1,\dots,n_k) \in E \text{ ssi } \mathbb{N} \models \Phi[n_1,\dots,n_k] \,.$$

On se persuade facilement que la vérité ou la fausseté dans $\mathbb{N}$ d'une formule close Σ_0 est décidable : à chaque étape de la définition inductive correspond un nombre fini connu de vérifications. L'appartenance à un ensemble Σ_0 est donc décidable. Plus formellement et plus précisément, on a comme conséquence immédiate des résultats de la section 1.1.2 le fait suivant.

Fait 3.1.1 *Les ensembles Σ_0 sont récursifs primitifs.*

Nous avons déjà vu un certain nombre d'ensembles (ou prédicats) Σ_0, et de fonctions dont le graphe est Σ_0.

Fait 3.1.2 *Les ensembles suivant sont Σ_0.*
- *les graphes du successeur, de l'addition et de la multiplication ;*
- *la divisibilité $x \mid y \equiv \exists z \leq y \; y = x \cdot z$;*
- *le graphe du reste de la division euclidienne de x par y, $\exists d \leq x \; (x = d \cdot y + r \wedge r < x)$.*
- *le graphe des couples de Cantor $z = \langle x, y\rangle \equiv 2z = (x+y+1)(x+y)+2y$;*
- *les graphes des projections associées, par exemple $x = \pi_1(z) \equiv \exists y \leq z \; z = \langle x, y\rangle$.*

En raisonnant à équivalence près on peut simplifier la définition des formules Σ_0.

Lemme 3.1.3 (définition alternative) *Une formule est équivalente à une formule Σ_0 si et seulement si elle est équivalente à une formule de la plus petite classe*

i. *qui contient les formules atomiques égalitaires et leurs négations ;*

ii. *qui est close par conjonction et disjonction ;*

iii. *qui est close par quantification bornée.*

Démonstration. Appelons Σ_0' la classe définie par induction ci-dessus. Il s'agit de montrer que toute formule équivalente à une formule Σ_0' est Σ_0.

D'une part une formule Σ_0' est Σ_0.

D'autre part, la classe des formules équivalentes à une formule Σ_0' contient toutes les formules atomiques, car $u \leq v \equiv \exists z \leq v \; u = z$. Elle est stable par négation (démonstration facile par induction). Elle contient donc toutes les formules Σ_0 (les autres cas viennent par définition). ■

La classe Σ

Définition 3.1.4 La classe des formules Σ est la classe des formules de l'arithmétique dont toutes les quantifications universelles sont bornées, qui n'utilise que les connecteurs propositionnels « $\wedge$, $\vee$ » et la négation seulement devant une formule atomique. Elle se définit inductivement comme la plus petite classe

i. qui contient les formules atomiques et leurs négations ;

ii. qui est close par conjonction et disjonction ;

iii. qui est close par quantification universelle bornée ;

iv. qui est close par quantification existentielle.

De la même façon que pour les Σ_0, un sous-ensemble de $\mathbb{N}^k$, ou un prédicat d'arité k sur les entiers, est dit Σ, quand il est définissable par une formule Σ.

À équivalence près, on peut se restreindre aux formules construites sur les seules formules atomiques égalitaires et leurs négations, ce qui définit la même classe d'ensembles Σ, en raisonnant exactement comme pour les formules Σ_0.

D'après le lemme 3.1.3 tout ensemble Σ_0 est Σ, mais les formules Σ autorisent la quantification existentielle non bornée, et donc la classe des ensembles Σ est stable par projection. Le fait suivant est une conséquence immédiate des propriétés de clôture de la proposition 2.3.5 page 39.

Fait 3.1.5 *Tout ensemble Σ est semi-décidable.*

On va montrer que les ensembles Σ sont exactement les semi-décidables. C'est l'objet de la section qui suit.

3.1.2 Fonctions calculables et Σ-définissabilité

On va tout d'abord montrer que les fonctions totales calculables ont un graphe Σ. Ceci se démontre par induction, le seul cas non évident étant celui de la définition par récurrence. Voyons comment procéder sur un cas simple.

Soit une fonction $f : \mathbb{N} \to \mathbb{N}$ définie par itération d'une fonction $h : \mathbb{N} \to \mathbb{N}$:

$$f(0) = a$$
$$f(n+1) = h(f(n)) \ .$$

On cherche à représenter son graphe par une formule arithmétique Σ, en supposant que c'est déjà le cas pour h. Par exemple si h est la multiplication par 2, de graphe évidemment Σ, on a alors $f(n) = 2^n$. Il s'agit de montrer que cette fonction a un graphe Σ.

C'est possible si on a un codage des suites qui est Σ. En effet, supposons que l'on dispose d'un codage analogue à celui déjà utilisé (voir section 1.1.4), et d'une fonction nth, où $\mathrm{nth}(l,i)$ calcule le i-ème terme de la liste codée par l. On peut représenter par un entier la suite des calculs successifs pour arriver à $f(n)$, soit la suite $[a; h(a); \ldots; h^n(n)]$. On a donc :

$$z = f(n) \text{ ssi } \exists l\, [z = \mathrm{nth}(l,n) \wedge a = \mathrm{nth}(l,0) \wedge \forall i < n \ \mathrm{nth}(l, i+1) = h(\mathrm{nth}(l,i))] \ .$$

Cette méthode se généralise facilement à des définitions par récurrence primitive quelconques. Il suffirait que la fonction nth puisse se définir sans récurrence pour éliminer cette dernière. Mais ni le codage des listes de la page 8, ni plus généralement ceux de la section 1.1.4, n'ont naturellement cette propriété.

Le codage plus astucieux qui suit est celui de l'article original de Gödel. La fonction qui joue le rôle de la fonction nth est appelée (en suivant la notation de l'article de Gödel) fonction β.

Fonction β de Gödel

Le codage, contrairement à ceux introduits dans la section 1.1.4, n'est pas fonctionnel, mais il est partout défini : toute suite finie doit avoir au moins un code, et deux suites différentes ne peuvent avoir le même code. En fait une suite finie donnée aura une infinité de codes, ce qui ne gêne pas pour le résultat cherché. Gödel le construit en utilisant le *théorème des restes chinois*, à savoir que le produit des $\mathbb{Z}/d_i\mathbb{Z}$, $i \in \{1, \ldots, n\}$, est isomorphe à $\mathbb{Z}/(\prod_{i=1}^n d_i)\mathbb{Z}$ quand les d_i sont premiers entre eux 2 à 2. En voici une version calculatoire adaptée au résultat cherché.

Théorème 3.1.6 (Théorème des restes chinois) *Si $d_0, \ldots, d_n$ sont des entiers premiers entre eux 2 à 2, et si $a_0, \ldots, a_n$ sont des entiers vérifiant*

$$0 \le a_i < d_i \ \ \textit{pour} \ 0 \le i \le n$$

*alors (*reste(a,q) *étant le reste de la division de a par q), il existe un entier a tel que pour tout $0 \le i \le n$:*

$$a \equiv a_i \mod d_i \ \ \textit{c'est-à-dire} \ \ a_i = \mathrm{reste}(a, d_i)$$

Pour coder une suite d'entiers $a_0, \ldots, a_n$ on va donc utiliser deux entiers a et d : l'entier a est fourni par le théorème des restes chinois, pour $n+1$ entiers premiers entre eux 2 à 2 construits polynomialement à partir de d.

Lemme 3.1.7 *Pour tout entier n, il existe une infinité d'entiers d tels que*

$$d_0 = 1+d,\ d_1 = 1+2d,\ \ldots,\ d_n = 1+(n+1)d$$

sont premiers entre eux 2 à 2.

Démonstration. Pour $s \geq n$, on pose $d = s!$. Alors un diviseur premier p commun de $d_i = 1+id$ et de $d_j = 1+jd$, $1 \leq i < j \leq n+1$, divise également $(j-i)d$, soit d, puisque $0 < j-i \leq n \leq s$. Comme $p \mid d_i$, $p \mid 1$: contradiction. ■

La fonction β de Gödel est définie par :

$$\beta(d,a,i) = \text{reste}(a, 1+(i+1)d)$$

Lemme 3.1.8 (Propriétés de la fonction β)

i. *Le graphe de la fonction β est Σ_0.*

ii. *Pour toute suite finie d'entiers $(a_0,\ldots,a_n)$ il existe au moins deux entiers a et d tels que :*

$$\beta(d,a,i) = a_i \quad \textit{pour } 0 \leq i \leq n$$

Démonstration.

i. Par définition des formules Σ_0, l'argument est le même que pour le fait 3.1.2.

ii. D'après le lemme 3.1.8, il existe un entier d tels que les $d_i = 1+(i+1)d$, $0 \leq i \leq n$, sont premiers entre eux, et qui peut être choisi supérieur à tous les a_i. On a $a_i < 1+(i+1)d$. D'après le théorème des restes chinois 3.1.6 page précédente, il existe donc un entier a ayant les propriétés requises. ■

Exemple d'utilisation de la fonction β Notons $\underline{y = \beta(a,d,i)}$ la formule Σ_0 qui définit le graphe de la fonction β. On montre alors que le graphe de la fonction exponentielle (définie par récurrence à partir de la multiplication) est défini par une formule Σ :

$$\underline{z = x^y} \equiv_d \exists a \exists d \left(\begin{array}{c} \underline{1 = \beta(a,d,0)} \\ \wedge \\ \underline{z = \beta(a,d,y)} \\ \wedge \\ \forall i \leq y\ \exists t' \leq a\ \exists t \leq a \Big(\underline{t' = \beta(a,d,i+1)} \wedge \underline{t = \beta(a,d,i)} \wedge t' = x \cdot t \Big) \end{array} \right)$$

Élimination de la récurrence

On établit d'abord une caractérisation des fonctions totales calculables qui n'utilise pas le schéma de définition par récurrence primitive.

Proposition 3.1.9 (Caractérisation des fonctions calculables sans récurrence) *L'ensemble des fonctions totales calculables est le plus petit ensemble de fonctions à plusieurs arguments entiers,*

— *contenant l'addition, la multiplication, les fonctions de projections p_i^k (voir page 3) et la fonction caractéristique de l'égalité $\chi_=$ (définie par $\chi_=(x,y) = 1$ si $x = y$, 0 sinon) ;*

— *clos par composition ;*

— *clos par minimisation.*

Démonstration. On nomme temporairement $\mathscr{C}$ le plus petit ensemble de fonctions à plusieurs arguments entiers vérifiant les propriétés de la proposition. Il est à peu près évident que $\mathscr{C}$ est contenu dans l'ensemble des fonctions totales calculables : on a enlevé un schéma de clôture, celui de définition par récurrence, et modifié l'ensemble des fonctions de base, celles qui sont ajoutées ont été démontrées récursives primitives.

Pour la réciproque on montre que $\mathscr{C}$ satisfait la caractérisation des fonctions totales calculables (définition 1.2.5 page 17).

Vu la définition de $\mathscr{C}$, il suffit de vérifier que $\mathscr{C}$ contient les fonctions de bases pour cette caractérisation, et que $\mathscr{C}$ est clos par récurrence primitive, ce qui est l'objet des trois lemmes suivant.

Lemme 3.1.10 *La fonction constante égale à* 1*, la fonction successeur et la fonction nulle sont dans* $\mathscr{C}$*.*

Démonstration. On a $1 = \chi_=(x,x)$, $\mathrm{s}(x) = x+1$, $0 = \chi_=(x,\mathrm{s}(x))$ (on utilise à chaque fois la composition et les projections). ■

Lemme 3.1.11 *L'ensemble des prédicats sur les entiers dont la fonction caractéristique est dans* $\mathscr{C}$ *est clos par opérations booléennes et quantifications universelles et existentielles bornées.*

Démonstration. On a déjà vu que la fonction caractéristique de la conjonction s'obtenait par produit. Pour la négation $\chi_{\neg A}(x) = \chi_=(\chi_A(x),0)$.

Comme on a la clôture par négation, il suffit de montrer le résultat pour l'une des quantifications bornées, prenons la quantification universelle bornée.

On utilise une forme ad hoc de minimisation bornée que l'on note $\tilde{\mu}$. Si S est un prédicat à $p+1$ arguments dont la fonction caractéristique est dans $\mathscr{C}$ alors, f définie par :

$$f(x_1,\dots,x_k,z) = \tilde{\mu}y<z.S[x_1,\dots,x_k,y] = \begin{cases} \mu y.S[x_1,\dots,x_k,y] & \text{s'il existe } y<z \\ & \quad \text{tel que } S[x_1,\dots,x_k,y] \\ z & \text{sinon} \end{cases}$$

est une fonction totale qui est dans $\mathscr{C}$.

En effet

$$\tilde{\mu}y<z.S[x_1,\dots,x_k,y] = \mu y.[S[x_1,\dots,x_k,y] \vee y = z]$$

et on a vu la clôture par opérations booléennes. On suppose maintenant que le prédicat R d'arité $k+1$ est dans C. Le prédicat défini sur les variables $x_1,\dots,x_k,z$ par $\forall y < z\ R[x_1,\dots,x_k,y]$ est de fonction caractéristique :

$$(x_1,\dots,x_k) \mapsto \chi_=\left(\tilde{\mu}y<z.\neg R[x_1,\dots,x_k,y],z\right) .$$

On a le résultat pour la quantification bornée avec une inégalité large, en bornant strictement par le successeur. ■

Lemme 3.1.12 (clôture par récurrence primitive) *L'ensemble* $\mathscr{C}$ *est clos par par schéma de récurrence primitive pour des fonctions totales, c'est-à-dire que, si* $g : \mathbb{N}^p \to \mathbb{N}$ *et* $h : \mathbb{N}^{p+2} \to \mathbb{N}$ *sont des fonctions totales de* $\mathscr{C}$*, alors* $f : \mathbb{N}^{p+1} \to \mathbb{N}$ *définie ci-dessous est une fonction totale de* $\mathscr{C}$*.*

$$\begin{aligned} f(a_1,\dots,a_p,0) &= g(a_1,\dots,a_p) \\ f(a_1,\dots,a_p,x+1) &= h(a_1,\dots,a_p,x,f(a_1,\dots,a_p,x)) \end{aligned}$$

Démonstration. La démonstration du lemme suit à peu près le schéma décrit au début de la section 3.1.2 page 57. On utilise la fonction β, qui est dans $\mathscr{C}$ par les lemmes 3.1.10 et 3.1.11 car :

$$\beta(d,a,i) = \mathrm{reste}(a,1+(i+1)d) = \mu r.\left[r<a \wedge \exists q \le a\ a = q\cdot(1+(i+1)d)+r\right] .$$

Les fonctions π_1 et π_2 qui calculent les composantes x et y du couple de Cantor sont également dans $\mathscr{C}$:

$$\langle x,y\rangle = \frac{(x+y)(x+y+1)}{2}+y\ ;\quad \pi_1(c) = \mu x.(\exists y \le c)\ \langle x,y\rangle = c\ ;\quad \pi_2(c) = \mu y.(\exists x \le c)\ \langle x,y\rangle = c\ .$$

La fonction $\beta' : \mathbb{N}^2 \to \mathbb{N}$, définie par

$$\beta'(c,i) = \beta(i,\pi_1(c),\pi_2(c))$$

est donc dans $\mathscr{C}$ par composition. Elle permet de définir la suite des calculs menant à $f(\overline{a},x)$:

$$F(\overline{a},x) = \mu\, l.\left[\beta'(l,0) = g(\overline{a},0) \wedge \forall i < x\, \beta'(l,i+1) = h(\overline{a},x,\beta'(l,i))\right]$$

fonction qui est dans $\mathscr{C}$ par le lemme 3.1.11, et donc f est dans $\mathscr{C}$ car

$$f(\overline{a},x) = \beta'(F(\overline{a},x),x)\ .$$

■

On conclut en revenant à la caractérisation des ensembles Σ.

Proposition 3.1.13

i. *Le graphe d'une fonction totale calculable est* Σ.

ii. *La classe des ensembles* Σ *est exactement la classe de tous les ensembles semi-décidables.*

Démonstration. On déduit d'abord le ii du i. On sait qu'un ensemble Σ est semi-décidable (fait 3.1.5 page 57).

Réciproquement soit A un sous-ensemble semi-décidable de $\mathbb{N}^p$. Il est le projeté d'un ensemble décidable (proposition 2.3.3 page 38), dont la fonction caractéristique a un graphe $G \subset \mathbb{N}^{p+1}$, qui est Σ d'après i, donc A est Σ puisque

$$\overline{x} \in A \text{ ssi } \exists y \exists z \left((\overline{x}, y, z) \in G \wedge z = 1 \right) .$$

On montre que toute fonction totale calculable a un graphe Σ en utilisant la caractérisation de la proposition 3.1.9.

— Les fonctions de base ont un graphe Σ : les graphes de + et × sont définis par $y = x_1 + x_2$ et $y = x_1 \times x_2$, les graphes des p_i^k par $x_i = z$; le graphe de la fonction caractéristique de l'égalité par $(x = y \wedge z = 1) \vee (x \neq y \wedge z = 0)$.

— La clôture par composition des fonctions de graphe Σ résulte de la clôture par quantification existentielle. En effet si $f(\overline{x}) = h\left(g_1(\overline{x}), \ldots, g_p(\overline{x})\right)$

$$y = f(\overline{x}) \text{ ssi } \exists z_1 \ldots \exists z_p \left[y = h(z_1, \ldots, z_p) \wedge z_1 = g_1(\overline{x}) \wedge \cdots \wedge z_p = g_p(\overline{x}) \right] .$$

La clôture par minimisation utilise la quantification existentielle et la quantification universelle bornée :

$$y = \mu z. f(x_1, \ldots, x_p, z) = 0 \text{ ssi } f(x_1, \ldots, x_p, y) = 0 \wedge \forall z < y \, \exists t \left(f(x_1, \ldots, x_p, z) = t \wedge t \neq 0 \right) . \quad \blacksquare$$

Exercice 19

1. Montrer que si une fonction totale $f : \mathbb{N}^p \to \mathbb{N}$ a un graphe G qui est semi-décidable, alors le complémentaire de ce graphe $G^c = \mathbb{N}^{p+1} \setminus G$ est aussi semi-décidable.

2. Montrer qu'un fonction totale de graphe semi-décidable a un graphe décidable.

3. Montrer que l'ensembles des fonctions totales de graphe Σ est exactement l'ensemble des fonctions totales calculables.

Exercice 20

1. Montrer que le graphe d'une fonction partielle calculable est un ensemble semi-décidable.

2. Montrer que si le graphe d'une fonction partielle est semi-décidable, alors cette fonction est partielle calculable.

3. Montrer que l'ensemble des fonctions partielles de graphe Σ est exactement l'ensemble des fonctions partielles calculables.

3.1.3 Hiérarchie arithmétique

Un *ensemble arithmétique* est un ensemble définissable par une formule de l'arithmétique du premier ordre, soit un sous-ensemble E de $\mathbb{N}^p$, $p > 0$, tel qu'il existe une formule F du langage de l'arithmétique ayant au plus $x_1, \ldots, x_p$ pour variables libres vérifiant :

$$(n_1, \ldots, n_p) \in E \text{ ssi } \mathbb{N} \models F \left[\underline{n_1} / x_1; \ldots, \underline{n_p} / x_p \right] .$$

On prend pour langage le langage égalitaire de signature $(0, \mathrm{s}, +, \times, \leq)$, c'est celui par exemple de l'arithmétique de Peano. Mais on verra que la définition est inchangée quand on ajoute à la signature des fonctions calculables et des prédicats décidables. On parle également de *prédicat arithmétique* pour un prédicat qui définit un ensemble arithmétique (c'est-à-dire qu'il s'exprime par une formule du premier ordre de l'arithmétique).

Une première remarque est que la classe des ensembles arithmétiques est dénombrable, puisque la signature étant finie, l'ensemble des formules est dénombrable. Il s'agit maintenant de classer ces ensembles suivant la complexité d'une formule qui les définit.

Formules et ensembles Σ_n^0 et Π_n^0

Le point de départ de la hiérarchie est la classe Σ_0 (ou Π_0). Les formules Σ_n^0 et Π_n^0 sont des formules Σ_0 précédées d'un préfixe de n alternances de quantificateurs ; les Σ_n^0 commencent par un quantificateur existentiel (et donc éventuellement plusieurs), les Π_n^0 par un universel. Plus formellement, on peut en donner une définition inductive :

i. si F est une formule Σ_0, alors $\exists x\ F$ est une formule Σ_1^0, et $\forall x\ F$ est une formule Π_1^0 ;

ii. Pour $n \geq 1$, si F est une formule Σ_n^0, alors $\exists xF$ est une formule Σ_n^0 et $\forall xF$ est une formule Π_{n+1}^0 ;

iii. Si F est une formule Π_n^0, alors $\forall xF$ est une formule Π_n^0 et $\exists xF$ est une formule Σ_{n+1}^0.

Ainsi, si F est une formule Σ_0 :

$$\begin{array}{rcl@{\qquad}rcl} \forall xF & \text{est} & \Pi_1^0 & \exists xF & \text{est} & \Sigma_1^0 \\ \forall y\exists xF & \text{est} & \Pi_2^0 & \exists y\forall xF & \text{est} & \Sigma_2^0 \\ \forall z\exists y\forall xF & \text{est} & \Pi_3^0 & \exists z\forall y\exists xF & \text{est} & \Sigma_3^0 \end{array}$$
etc.

Un sous-ensemble Σ_n^0, respectivement Π_n^0, de $\mathbb{N}^p$ est un sous-ensemble de $\mathbb{N}^p$ définissable par une formule Σ_n^0, respectivement Π_n^0. La classe des ensembles Δ_n^0 est l'intersection des classes Σ_n^0 et Π_n^0. On utilise le même vocabulaire pour les prédicats. Ces classes définissent ce que l'on appelle la *hiérarchie arithmétique*. On commence par énoncer des conséquences assez immédiates de la définition.

Proposition 3.1.14

i. *Tout ensemble arithmétique apparaît dans la hiérarchie arithmétique ;*

ii. *un ensemble est Σ_n^0 ssi son complémentaire est Π_n^0 ;*

iii. $\Sigma_n^0 \cup \Pi_n^0 \subset \Delta_{n+1}^0 (= \Sigma_{n+1}^0 \cap \Pi_{n+1}^0)$ *;*

iv. *la classe des ensembles arithmétiques est*

$$\bigcup_{n=1}^{\omega} \Delta_n^0 = \bigcup_{n=1}^{\omega} \Sigma_n^0 = \bigcup_{n=1}^{\omega} \Pi_n^0.$$

Démonstration.

i. Toute formule peut se mettre sous forme prénexe, qui définit bien un ensemble dans la hiérarchie ;

ii. le passage à la négation échange les quantificateurs ;

iii. par ajout de quantificateurs « inutiles », en tête ou en queue du préfixe de quantificateurs de la formule définissant l'ensemble ;

iv. conséquence de i et iii. ∎

Le 0 en exposant indique que les quantifications sont du premier ordre. Comme on ne considère que ce cas (pas de quantification du second ordre, c'est-à-dire portant sur des ensembles d'entiers), on va noter dans la suite Σ_n, Π_n, Δ_n pour, respectivement, Σ_n^0, Π_n^0 et Δ_n^0.

Exemple. Si on examine l'exemple page 58 on s'aperçoit que l'on a montré que le graphe de l'exponentielle est un ensemble Σ_1. Le complémentaire du graphe de l'exponentielle est donc Π_1, la classe Σ_0 étant stable par négation. Mais la fonction exponentielle étant totale, son complémentaire est également Σ_1 :

$$z \neq x^y \equiv_{\mathbb{N}} \exists z' \left(\underline{z' = x^y} \wedge z \neq z' \right) .$$

On a donc montré que le graphe de l'exponentielle est un ensemble Δ_1.

Par définition la classe des prédicats Σ_n est close par quantificatione existentielle, et celle des prédicats Π_n par quantification universelle. La proposition suivante donne d'autres propriétés de clôture, ainsi que des définitions alternatives pour chacune des classes de la hiérarchie.

Proposition 3.1.15

i. *Pour tout entier n les classes de prédicats Σ_n et Π_n sont chacune close par conjonctions, disjonctions, quantifications bornées, les classes de prédicats Δ_n sont chacune close par toutes les opérations booléennes, et quantifications bornées ;*

ii. *Pour tout entier n, un sous-ensemble Σ_{n+1} de $\mathbb{N}^p$ est le projeté d'un sous-ensemble Π_n de $\mathbb{N}^{p+1}$, et donc dans la définition des prédicats Σ_n et Π_n, on peut supposer que chaque alternance comporte un seul quantificateur ;*

iii. *La classe des ensembles Δ_1 est la classe des ensembles décidables ; la classe des ensembles Σ_1 est la classe des ensembles Σ, soit la classe des ensembles semi-décidables ; la classe des ensembles Π_1 est la classe des ensembles co-semi-décidables (complémentaires de semi-décidables) ;*

iv. *Si l'on ajoute à la signature des prédicats décidables et des fonctions calculables, la hiérarchie arithmétique reste identique à partir du niveau 1 et donc la classe des ensembles définissables est inchangée.*

Démonstration.

i. On procède par récurrence sur n. Pour $n = 0$, c'est la définition de $\Sigma_0 = \Pi_0$. Supposons le résultat pour n. On a la clôture par disjonction et conjonction de la classe Σ_{n+1} en se ramenant à la même propriété pour la classe Π_n : il suffit d'utiliser autant de fois que chacune des deux formules a de quantificateurs existentiels en tête les équivalences du type (x n'apparaît pas libre dans B) :

$$\exists x\ A \vee B \equiv \exists x(A \vee B)\ ;\quad \exists x\ A \wedge B \equiv \exists x(A \wedge B)\ .$$

et les équivalences symétriques en A et B.

Le résultat se déduit par négation et loi de de Morgan pour la classe Π_{n+1}.

La classe Σ_{n+1} est close par quantification existentielle bornée, puisque close par quantification existentielle :

$$\exists x \leq t\ \exists y\ A \equiv \exists x\ \exists y(x \leq t \wedge A)\ .$$

Finalement l'équivalence suivante, qui est vérifiée dans $\mathbb{N}$ en prenant pour z un majorant des y en nombre fini ($x \leq t$) :

$$\forall x \leq t\ \exists y\ A \equiv_{\mathbb{N}} \exists z\ \forall x \leq t\ \exists y \leq z\ A$$

permet de déduire la clôture des Σ_{n+1} par quantification universelle bornée de celle des Π_n par quantification existentielle bornée. On a par passage à la négation le résultat pour la classe Π_{n+1}.

ii. On a

$$\exists x_1 \ldots \exists x_p\ A \equiv_{\mathbb{N}} \exists z\ \exists x_1 \leq z\ \ldots \exists y_p \leq z\ A\ .$$

On peut donc définir un prédicat Σ_{n+1} par une seule quantification à partir d'un prédicat Π_n d'après le i. De même pour la classe Π_{n+1} en passant à la négation. On peut donc se ramener à un seul quantificateur par alternance pour la définition des Σ_n et Π_n.

iii. La classe Σ_1 est incluse dans la classe Σ définie page 56. Elle contient évidemment tous les ensembles définis par des formules atomiques et négation de formules atomiques. On a montré au i qu'elle est stable par les opérations de clôture de celle-ci (voir la définition 3.1.4 page 56) : c'est la classe Σ, et donc la classe des ensembles semi-décidables d'après la proposition 3.1.13.

Les ensembles Δ_1 sont donc les ensembles semi-décidables de complémentaire semi-décidable, c'est-à-dire les ensembles décidables (proposition 2.3.6 page 39).

iv. Le projeté d'un ensemble décidable est semi-décidable, la classe Σ_1 n'est donc pas modifiée par un tel ajout. Toutes les classes de la hiérarchie d'indice non nul se définissent à partir de la classe Σ_1. ■

En particulier on déduit le corollaire suivant des (ii), (iii) de cette proposition et de la proposition 3.1.14 (ii).

Corollaire 3.1.16 *La hiérarchie arithmétique des prédicats Σ_n et Π_n, $n \geq 1$, s'obtient par passage à la négation (de Σ_n à Π_n) et une seule quantification existentielle (de Π_n à Σ_{n+1}) à partir de la classe Σ_1 qui est celle des prédicats semi-décidables.*

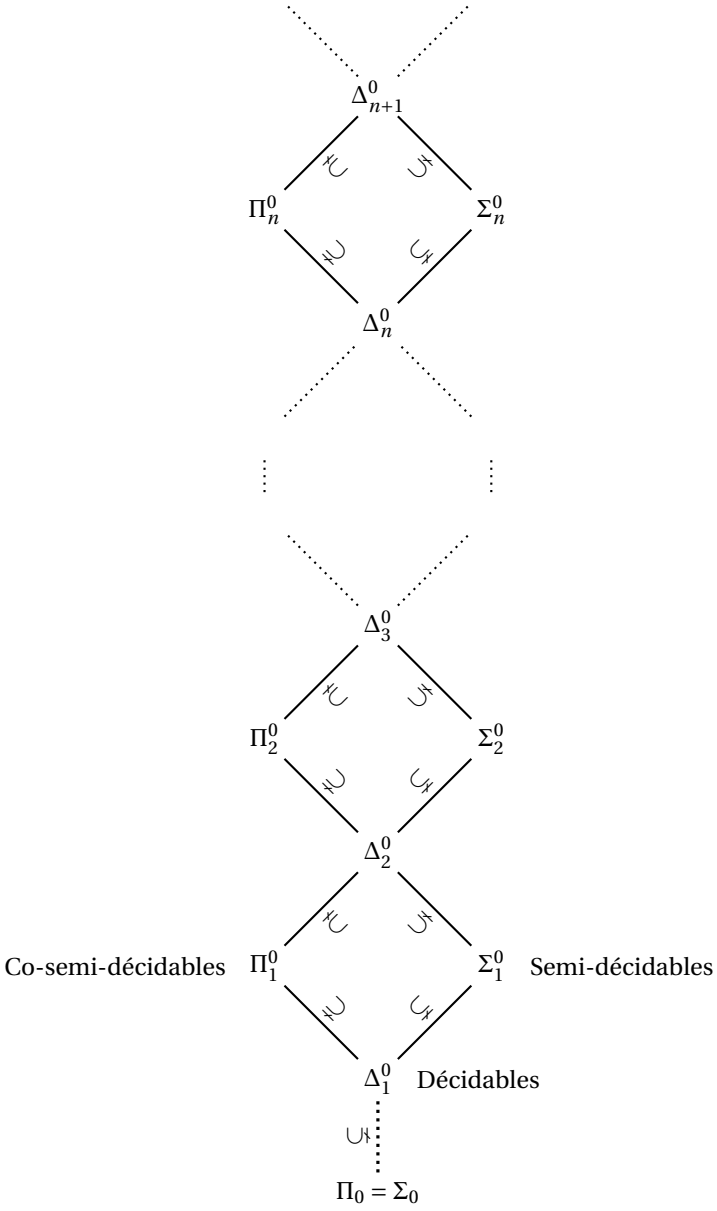


FIGURE 3.1 – Hiérarchie arithmétique

Les relations d'inclusion de la hiérarchie arithmétique sont décrites par la figure 3.1. Pour le moment on n'a montré que des inclusions larges. Grâce au iii de la proposition précédente, on sait maintenant que $\Delta_1 \subsetneq \Sigma_1$, puisque l'ensemble diagonal K du problème de l'arrêt diagonal (voir le théorème 2.3.7 page 40) est semi-décidable non décidable, et $\Delta_1 \subsetneq \Pi_1$ puisque K^c est co-semi-décidable non décidable. Ceci se généralise en utilisant les mêmes méthodes. Tout d'abord on peut énumérer les sous-ensembles Σ_n, resp. Π_n, de Π_n par un sous-ensemble Σ_n, resp. Π_n, de $\mathbb{N}^{p+1}$.

Lemme 3.1.17 *Soit A un sous-ensemble de $\mathbb{N}^p$ ($p > 0$), et $n \geq 1$. On a*

— *A est Σ_n si et seulement s'il existe i (indice de A) tel que*

$$A = \left\{ \overline{x} \mid \exists y_1 \forall y_2 \ldots Q_n\, y_n\ T_n^{n+p}[i, \overline{x}, y_1, \ldots, y_n] \right\}$$

— *A est Π_n si et seulement s'il existe i (indice de A) tel que*

$$A = \left\{ \overline{x} \mid \forall y_1 \exists y_2 \ldots Q'_n y_n \neg T_n^{n+p}[i, \overline{x}, y_1, \ldots, y_n] \right\}$$

où T_n^m est soit le prédicat de terminaison de Kleene T^m pour les fonctions à m arguments, soit sa négation, plus précisément :
— *si n est impair, T_n^m est T^m, Q_n est $\exists$ et Q'_n est $\forall$;*
— *si n est pair, T_n^m est $\neg T^m$, Q_n est $\forall$ est Q'_n est $\exists$.*

Démonstration. Tout sous-ensemble semi-décidable W de $\mathbb{N}^m$, a un indice i qui vérifie :

$$\overline{x} \in W \Leftrightarrow \exists y \; T^m[i, \overline{x}, y]$$

par le théorème de forme normale de Kleene. La proposition suit par le corollaire 3.1.16. ■

On dit qu'un sous-ensemble E de $\mathbb{N}^{p+1}$ *énumère une classe* $\mathscr{C}$ de sous-ensembles de $\mathbb{N}^p$ quand :

$$\forall A \in \mathscr{C} \; \exists i \in \mathbb{N} \; \forall \overline{x} (x \in A \Leftrightarrow (i, x) \in E) \;\; \text{(un tel } i \text{ est appelé indice de } A)$$

et on utilise le même vocabulaire pour les prédicats. Par exemple $\{(i, x) \in \mathbb{N}^2 \mid \exists s \; T^1(i, x, s)\}$ énumère les sous-ensembles semi-décidables de $\mathbb{N}$.

La proposition suivante est une conséquence immédiate du lemme 3.1.17.

Proposition 3.1.18 (Énumération des ensembles Σ_n et Π_n) *Pour, tout entier $p > 0$, pour tout entier $n > 0$, il existe un sous-ensemble Σ_n de $\mathbb{N}^{p+1}$, soit S_n^p, et un sous-ensemble Π_n de $\mathbb{N}^{p+1}$, soit P_n^p, tels que S_n^p énumère tous les sous-ensembles Σ_n de $\mathbb{N}^p$, et P_n^p énumère tous les sous-ensembles Π_n de $\mathbb{N}^p$.*

Proposition 3.1.19 *La hiérarchie arithmétique est stricte pour l'inclusion, c'est-à-dire que toutes les inclusions de la figure 3.1 sont strictes, et qu'il n'y a pas d'autres relations d'inclusion que celles décrites. Plus précisément :*

i. $\Pi_n \setminus \Sigma_n \neq \emptyset$ *et* $\Sigma_n \setminus \Pi_n \neq \emptyset$ *;*

ii. $\Sigma_n \cup \Pi_n \subsetneq \Delta_{n+1}$ *;*

Démonstration.

i. La démonstration de $\Pi_n \setminus \Sigma_n \neq \emptyset$ et $\Sigma_n \setminus \Pi_n \neq \emptyset$ est la même que celle de l'indécidablité du diagonal du problème de l'arrêt (qui donne $\Pi_1 \setminus \Sigma_1 1 \neq \emptyset$ et $\Sigma_1 \setminus \Pi_1 \neq \emptyset$). Soit S_n un ensemble Σ_n qui énumère les sous-ensembles Σ_n de $\mathbb{N}$. On pose

$$K_n = \{i \mid (i, i) \in \mathrm{S}_n\} .$$

L'ensemble K_n est Σ_n. Son complémentaire K_n^{c} est Π_n. S'il était Σ_n, il aurait un indice i_0 :

$$i \in K_n^{\mathrm{c}} \text{ ssi } (i_0, i) \in \mathrm{S}_n$$

d'où une contradiction avec la définition de K_n puisque

$$i_0 \in K_n^{\mathrm{c}} \text{ ssi } (i_0, i_0) \in \mathrm{S}_n .$$

L'ensemble K_n^{c} est Π_n et non Σ_n, donc par passage au complémentaire l'ensemble K_n est Σ_n et non Π_n.

ii. Posons

$$E_n = \left\{ \langle i, b \rangle \mid (i \in K_n \text{ et } b = 0) \text{ ou } (i \in K_n^{\mathrm{c}} \text{ et } b = 1) \right\}.$$

Cet ensemble est Δ_{n+1} comme réunion de deux ensembles Δ_{n+1} mais n'est ni Σ_n, car K_n^{c} le serait, ni Π_n, car K_n le serait. ■

Réduction et hiérarchie arithmétique

On sait déjà (proposition 2.6.6 page 51) que K est complet pour la classe Σ_1, et donc K^c est complet pour la classe Π_1 par passage au complémentaire. On généralise aux classes Σ_n et Π_n en suivant la même démonstration.

Proposition 3.1.20

— *L'ensemble K_n est m-complet pour la classe des ensembles Σ_n, c'est-à-dire que :*

$$E \in \Sigma_n \Leftrightarrow E \leq_m K_n \,;$$

— *l'ensemble K_n^c est m-complet pour la classe des ensembles Π_n, c'est-à-dire que :*

$$E \in \Pi_n \Leftrightarrow E \leq_m K_n^c.$$

Démonstration. Exercice.

On peut maintenant identifier la classe des ensembles qui se réduisent à K par réduction de Turing.

Proposition 3.1.21

i. *Un ensemble est Σ_{n+1} si et seulement s'il est semi-décidable en un ensemble Σ_n ou Π_n ;*

ii. *un ensemble E est Δ_{n+1} si et seulement s'il est décidable en un ensemble Σ_n ou Π_n, c'est-à-dire si et seulement s'il existe un ensemble F qui est Σ_n ou Π_n, tel que $E \leq_T F$;*

iii. *la classe des ensembles E tels que $E \leq_T K_n$ est exactement la classe des ensembles Δ_{n+1} ;*

iv. *en particulier la classe des ensembles E tels que $E \leq_T K$ est exactement la classe des ensembles Δ_2.*

Démonstration.

i. ($\Rightarrow$) Supposons $E \subset \mathbb{N}$ et $E \in \Sigma_{n+1}$. Alors, E est le projeté d'un ensemble Π_n, soit F. Cet ensemble F est semi-décidable en lui-même, donc E également qui est son projeté, par la version relativisée de la clôture des semi-décidables par projection (proposition 2.3.5 page 39). L'ensemble F est également semi-décidable en F^c, qui est Σ_n, donc E également, avec le même argument.

($\Leftarrow$) Supposons $E \subset \mathbb{N}$ et E semi-décidable en F, où $F \subset \mathbb{N}$ et F est Σ_n. Alors, d'après la version relativisé de l'énumération des ensembles semi-décidables, $E = W_i^F$ pour un certain i, et d'après la proposition 2.6.13 page 53 :

$$E = \left\{x \in \mathbb{N} \mid \exists s \exists l \left[T^F[i,x,l,s] \text{ et } \forall j \leq s \left((\text{nth}(l,j) = 0 \Rightarrow j \in F^c) \text{ et } (\text{nth}(l,j) = 1 \Rightarrow j \in F) \right) \right] \right\}$$

L'ensemble F est Σ_n, donc lui-même et son complémentaire sont Δ_{n+1}. La classe Δ_{n+1} est close par opérations booléennes et quantification bornée. L'ensemble $E = W_i^F$, est donc défini par deux quantifications existentielles sur un prédicat Δ_{n+1}, soit un prédicat Σ_{n+1} ($\Delta_{n+1} \subset \Sigma_{n+1}$ et clôture par quantification existentielle de la classe Σ_{n+1}).

Pour $F \in \Pi_n$, il suffit d'échanger F et F^c dont les rôles sont symétriques dans la démonstration.

ii. ($\Rightarrow$) Supposons que E est Δ_{n+1}. Alors E et E^c sont Σ_{n+1}. On a donc, d'après **i**, des ensembles Σ_n F et G tels que E est semi-décidable en F et E^c est semi-décidable en G, donc tous deux sont semi-décidables en $F \times \{0\} \cup G \times \{1\}$, donc E est décidable en $F \times \{0\} \cup G \times \{1\}$ (proposition 2.6.15 page 53) qui est encore Σ_n, et en son complémentaire qui est Π_n.

($\Leftarrow$) Supposons que $E \leq_T F$ où F est Σ_n (si F est Π_n, on se ramène à ce cas car $E \leq_T F^c$). Alors E est semi-décidable en F, donc d'après **i**, E est Σ_{n+1}. Comme $E^c \leq_T F$, de même E^c est Σ_{n+1} donc E est Δ_{n+1}.

iii. Tout ensemble Σ_n se réduit à K_n par réduction many-one, donc par réduction de Turing, on a le résultat d'après **ii**. ■

3.1.4 Prédicats arithmétiques

On rappelle (proposition 3.1.14 page 61) que tout ensemble arithmétique (i.e. définissable au premier ordre dans le langage de l'arithmétique) est dans l'une des classes de la hiérarchie arithmétique. On a le corollaire suivant de la proposition 3.1.19.

Corollaire 3.1.22 *Il n'existe pas de sous-ensemble arithmétique de* $\mathbb{N}^2$ *qui énumère tous les sous-ensembles arithmétiques de* $\mathbb{N}$.

Démonstration. En effet un tel sous-ensemble R serait dans la hiérarchie à un certain niveau, disons Σ_n. Tous les sous-ensembles E arithmétiques de $\mathbb{N}$, vérifiant pour un certain i que

$$n \in E \text{ ssi } (i,n) \in R$$

seraient alors Σ_n. ■

Supposons maintenant que l'on ait définit un codage des formules de l'arithmétique du premier ordre $F \mapsto \ulcorner F \urcorner$, et que les fonctions qui permettent de manipuler ces formules sont calculables, donc de graphe définissable dans l'arithmétique (par une formule Σ_1). On pourrait songer à refléter dans l'arithmétique la vérité (dans le modèle standard $\mathbb{N}$) des formules de l'arithmétique du premier ordre, soit chercher un prédicat V, dit *prédicat de vérité* tel que :

$$\mathbb{N} \models V[\ulcorner F \urcorner / x_0] \text{ ssi } \mathbb{N} \models F\,.$$

La définition de la vérité de Tarski permet de construire de tels prédicats pour les formules jusqu'à un certain niveau de la hiérarchie, par exemple Σ_n. Le prédicat de vérité correspondant sera au même niveau.

En utilisant que les fonctions qui permettent de manipuler ces codages sont calculables donc de graphe définissable dans l'arithmétique du premier ordre, en particulier la fonction de substitution d'un entier à une variable dans une formule, il n'est pas très difficile de se rendre compte qu'un prédicat de vérité pour toutes les formules arithmétiques permettrait d'énumérer tous les sous-ensembles arithmétiques ce qui contredit le corollaire 3.1.22.

Théorème de Tarski. Le prédicat de vérité des énoncés de l'arithmétique du premier ordre n'est pas arithmétique, autrement dit, la vérité (dans $\mathbb{N}$) des énoncés de l'arithmétique du premier ordre n'est pas définissable dans $\mathbb{N}$ en arithmétique du premier ordre.

La démonstration est précisée plus loin en donnant un codage explicite (voir théorème 3.3.1 page 70).

Ce théorème est un peu le prototype du premier théorème d'incomplétude de Gödel, bien qu'il n'ait été publié qu'après ce dernier par Tarski. Mais on sait maintenant par la correspondance de Gödel que celui-ci le connaissait déjà à l'époque où il a découvert son théorème d'incomplétude, et que c'est même de ce résultat qu'il est parti pour élaborer ce dernier. Il semble qu'à l'époque, il ait été méfiant vis-à-vis de la notion de vérité, qui a été étudiée et vraiment formalisée par Tarski, et a préféré publier seulement un énoncé qui ne parlait que de démonstration.

On va expliciter un codage des formules de l'arithmétique, ce qui permet de préciser l'énoncé du théorème et sa démonstration, mais il est clair que celui-ci vaut pour tout choix de codage « raisonnable ».

3.2 Le codage des formules

Les termes et les formules de l'arithmétiques sont des objets syntaxiques, des suites de caractères, que l'on va représenter par des entiers de la même façon que l'on a codé les suites de calcul sur machine à registre par des entiers.

Les prédicats utiles, par exemple celui qui dit d'une formule qu'elle est close, doivent être définissables par des formules de l'arithmétique, et les fonctions qui manipulent ces formules, par exemple celle qui substitue un terme à une variable libre, doivent avoir leur graphe définissable par une formule de l'arithmétique, ce qui va permettre de refléter dans l'arithmétique les calculs sur les formules.

Les fonctions et les prédicats dont on a besoin sont intuitivement calculables. Si on vérifie que leurs équivalents sur les codes de formules le sont également, on aura montré qu'elles se définissent dans l'arithmétique par des formules Σ_1.

Les méthodes sont analogues à celles déjà utilisées pour coder les machines. La principale différence vient de la structure des formules, qui est arborescente, et de la liaison de variable, dont il faut tenir compte pour la substitution.

Le choix et les détails du codage n'ont pas grande importance : l'essentiel du travail a été fait en montrant que la fonction β permettait de coder la récurrence. Le reste, qui était tout à fait nouveau en 1930 quand Gödel présentait pour la première fois son théorème, est devenu banal à l'époque de l'informatique : on sait bien que l'on peut représenter des structures syntaxiques par des entiers et les manipuler par des fonctions calculables.

Çà n'est aujourd'hui qu'un exercice, proche d'un exercice de programmation (ou de préparation à celle-ci puisque l'on n'exécute rien), mais sans aucun souci d'efficacité. Le langage est juste un peu contraint, du moins si on veut être précis et ne pas faire appel à la thèse de Church. Le codage des formules présenté ci-dessous passe par un codage des arbres binaires : les définitions syntaxiques usuelles par induction sur la structure des formules se traduisent alors directement en des définitions récursives primitives.

3.2.1 Codage des arbres binaires

Les arbres binaires étiquetés

Les termes et les formules peuvent être vus comme des arbres binaires finis étiquetés. L'ensemble $\mathscr{B}$ des arbres binaires étiquetés est défini inductivement :

arbre vide $() \in \mathscr{B}$;

nœud Si $e \in \mathbb{N}$ est un entier, si $T_1 \in \mathscr{B}$ et $T_2 \in \mathscr{B}$, alors $(T_1, T_2)_e \in \mathscr{B}$.

L'arbre $(T_1, T_2)_e$ a pour fils gauche T_1, pour fils droit T_2 et pour étiquette n. Les feuilles sont les arbres dont les deux fils sont l'arbre vide. Les nœuds unaires sont ceux dont un des deux fils est l'arbre vide.

Codage des arbres binaires étiquetés

Il y a une seule façon d'écrire un arbre de $\mathscr{B}$, soit comme l'arbre vide, soit comme un arbre d'étiquette n, de fils gauche T_1, et de fils droit T_2, ce qui autorise les définitions par induction sur la structure des arbres de $\mathscr{B}$. Pour le codage, On introduit, de façon analogue au codage des listes page 8, une nouvelle fonction ternaire (pour les notations des couples et n-uplets voir page 8) :

$$\langle t_1, t_2\rangle_e = 1 + \langle e, t_1, t_2\rangle \ .$$

On définit inductivement une fonction : $T \mapsto \ulcorner T \urcorner$ de $\mathscr{B}$ dans $\mathbb{N}$.

arbre vide $\ulcorner () \urcorner = 0$;

nœud $\ulcorner (T_1, T_2)_e \urcorner = \langle \ulcorner T_1 \urcorner, \ulcorner T_2 \urcorner\rangle_e$ $(= 1 + \langle e, \ulcorner T_1 \urcorner, \ulcorner T_2 \urcorner\rangle)$.

Comme la fonction de codage des couples de Cantor est bijective, pour tout entier naturel n, un et un seul des cas suivants est réalisé :

— soit $n = 0$;

— soit $n = \langle n_1, n_2\rangle_e$ où $e, n_1, n_2 \in \mathbb{N}$ sont déterminés de façon unique par n.

Lemme 3.2.1 *la fonction* : $T \mapsto \ulcorner T \urcorner$ *est bijective de* $\mathscr{B}$ *dans* $\mathbb{N}$.

Démonstration. Par induction sur $\mathscr{B}$, en utilisant la remarque qui précède. ■

Un sous-arbre d'un arbre binaire étiqueté T est un arbre qui apparaît dans la construction inductive de celui-ci, la définition de l'ensemble des sous-arbres apr induction est immédiate. Un sous-arbre strict de T est un sous-arbre de T différent de T.

Le lemme suivant est une propriété de « croissance » du codage assez évidente mais plusieurs fois utilisée ensuite.

Lemme 3.2.2 (croissance du code)

— *Si S est un sous-arbre de T, alors $\ulcorner S \urcorner < \ulcorner T \urcorner$;*
— *si une étiquette e apparaît dans la construction de T, alors $e < \ulcorner T \urcorner$.*

Démonstration. Induction immédiate sur la définition des sous-arbres pour l'inégalité large, dont on déduit l'inégalité stricte pour les sous-arbres stricts. ■

Fonctions sur les arbres

Lemme 3.2.3 *Les trois fonctions de $\mathbb{N} \to \mathbb{N}$, qui au code d'un arbre de $\mathscr{B}$ associent pour chacune 0 si l'arbre est vide, et sinon son étiquette pour la première, le code de son fils droit pour la seconde, le code de son fils gauche pour la troisième, sont récursives primitives.*

Démonstration. Par composition des projections π_1, π_2 et du prédécesseur. ■

On a besoin de représenter les définitions usuelles sur les termes et les formules qui sont presque toutes des définitions par récurrence structurelle, et s'avèrent être des cas particuliers d'un schéma de recurrence primitive sur les arbres.

Lemme 3.2.4 (Récurrence primitive structurelle sur les arbres) *Si g est une fonction récursive primitive de $\mathbb{N}^p \to \mathbb{N}$, h une fonction récursive primitive de $\mathbb{N}^{p+5} \to \mathbb{N}$, alors la fonction $f : \mathbb{N}^{p+1} \to \mathbb{N}$ définie ci-dessous est récursive primitive.*

— $f(n_1, \ldots, n_p, 0) = g(n_1, \ldots, n_p)$;
— $f(n_1, \ldots, n_p, \langle a, b \rangle_e) = h\big(n_1, \ldots, n_p, e, a, b, f(a), f(b)\big)$.

Démonstration. On peut réécrire cette définition par récurrence avec les fonctions récursives primitives fournies par le lemme 3.2.3 précédent. C'est alors un cas particulier de la récurrence sur la suite des valeurs (section 1.1.4 page 9), puisque $a < \langle a, b \rangle_e$, $b < \langle a, b \rangle_e$ et a, b et e se calculent de façon récursive primitive à partir de $\langle a, b \rangle_e$. ■

Par exemple la fonction $h : \mathbb{N} \to \mathbb{N}$, qui calcule la hauteur d'un arbre codé par son argument, se définit de façon récursive primitive, de façon strictement analogue à la définition sur les arbres :

$$h(()) = 0 \; ; \quad h(\langle t_1, t_2 \rangle_n) = 1 + \sup(t_1, t_2) \; .$$

En logique, on utilise parfois des définitions par récurrence sur la hauteur d'un arbre, qui est plus générale que celle qui précède car l'appel récursif peut se faire sur un arbre de hauteur plus petite qui n'est pas nécessairement un sous-arbre, et n'a donc pas un code forcément plus petit. On ne l'utilisera pas pour les résultats qui nous intéressent, mais on pourrait obtenir celle-ci comme cas particulier d'une définition par récurrence primitive sur la suite des valeurs avec substitution de paramètre [1] (la variable de récurrence est la hauteur de l'arbre, le paramètre substituable est le code de l'arbre) dont on montre facilement qu'elle reste récursive primitive.

1. Voir exercice 7 page 11 pour la substitution de paramètre

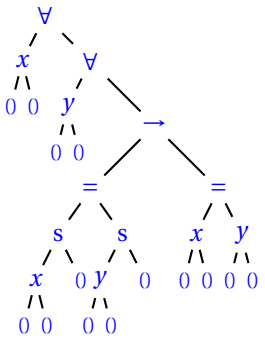
3.2.2 Codage des expressions arithmétiques

On code les expressions du langage, c'est à dire les termes et les formules, via le codage précédent des arbres binaires étiquetés. Un exemple est donné ci-contre. il suffit donc de décider d'un choix d'étiquettes pour les opérateurs du langage. Le langage considéré est le langage (égalitaire) de l'arithmétique de Peano, de signature $\mathscr{L}_{\mathscr{P}} = (0, \mathrm{s}, +, \times, \leq)$. Il a la syntaxe suivante :

— les termes utilisent des symboles de constantes et de fonctions 0, s, + et ×, et une infinité dénombrable de symboles de variables, $\{x_i\}_{i\in\mathbb{N}}$;

— les formules atomiques utilisent = ou ≤ ;

— les formules du calcul des prédicats utilisent les connecteurs → et ⊥ et le quantificateur ∀.



Arbre binaire étiqueté pour la formule $\forall x \forall y \left(s(x) = s(y) \to x = y\right)$

Les étiquettes des symboles primitifs du langage, notées $\#\{\text{symbole}\}$, sont (c'est évidemment arbitraire) :

Termes					Formules				
$\#\{0\}$	$\#\{x_i\}$	$\#\{\mathrm{s}\}$	$\#\{+\}$	$\#\{\times\}$	$\#\{=\}$	$\#\{\leq\}$	$\#\{\bot\}$	$\#\{\to\}$	$\#\{\forall\}$
0	$9+i$	1	2	3	4	5	6	7	8

On note $\ulcorner E \urcorner$ le code d'une expression (terme ou formule). Celui-ci est défini inductivement :

Termes				
$\ulcorner 0 \urcorner$	$\ulcorner x_i \urcorner$	$\ulcorner \mathrm{s}\, t \urcorner$	$\ulcorner t_1 + t_2 \urcorner$	$\ulcorner t_1 \times t_2 \urcorner$
$\langle 0,0\rangle_{\#\{0\}}$	$\langle 0,0\rangle_{\#\{x_i\}}$	$\langle \ulcorner t \urcorner, 0\rangle_{\#\{\mathrm{s}\}}$	$\langle \ulcorner t_1 \urcorner, \ulcorner t_2 \urcorner\rangle_{\#\{+\}}$	$\langle \ulcorner t_1 \urcorner, \ulcorner t_2 \urcorner\rangle_{\#\{\times\}}$

Formules				
$\ulcorner t_1 = t_2 \urcorner$	$\ulcorner t_1 \leq t_2 \urcorner$	$\ulcorner \bot \urcorner$	$\ulcorner F \to G \urcorner$	$\ulcorner \forall x_i\, F \urcorner$
$\langle \ulcorner t_1 \urcorner, \ulcorner t_2 \urcorner\rangle_{\#\{=\}}$	$\langle \ulcorner t_1 \urcorner, \ulcorner t_2 \urcorner\rangle_{\#\{\leq\}}$	$\langle 0,0\rangle_{\#\{\bot\}}$	$\langle \ulcorner F \urcorner, \ulcorner G \urcorner\rangle_{\#\{\to\}}$	$\langle \ulcorner x_i \urcorner, \ulcorner F \urcorner\rangle_{\#\{\forall\}}$

Il s'agit bien d'un cas particulier du codage des arbres binaires étiquetés vu au dessus.

Lemme 3.2.5 *Les ensembles suivants sont récursifs primitifs :*
— *l'ensemble des codes de variables* $\mathtt{Var} = \{\ulcorner x_i \urcorner \mid i \in \mathbb{N}\}$;
— *l'ensemble des codes de termes* $\mathtt{Term}$;
— *l'ensemble des codes de formules* $\mathtt{Form}$.

Démonstration. Pour $\mathtt{Var}$ on remarque que : $i \mapsto \ulcorner x_i \urcorner$ est récursive primitive et croissante.
Les fonctions caractéristiques de $\mathtt{Term}$ et $\mathtt{Form}$ se définissent par récurrence sur les arbres, en utilisant le lemme 3.2.4. ■

Lemme 3.2.6 *Les fonctions* $\mathtt{var} : i \mapsto \ulcorner x_i \urcorner$ *et* $\mathtt{num} : n \mapsto \ulcorner \underline{n} \urcorner$ *(on rappelle que* $\underline{n} = \underbrace{s \ldots s}_{n} 0$*) sont récursives primitives.*

Démonstration. La fonction $\mathtt{var}$ se définit par composition de fonctions récursives primitives, la fonction $\mathtt{num}$ par itération de $x \mapsto \langle x, 0\rangle_{\#\{s\}}$ en partant de $\ulcorner 0 \urcorner$. ■

Lemme 3.2.7 *Il existe une fonction récursive primitive* $\mathtt{libre} : \mathbb{N}^2 \to \mathbb{N}$ *telle que*
— $\mathtt{libre}(i, \ulcorner E \urcorner)$ *est le nombre d'occurrences libres de la variable* x_i *dans l'expression* E ;
les ensembles suivants sont récursifs primitifs :
— $\mathtt{Libre} = \{(i,n) \mid x_i$ *apparait libre dans l'expression* E *de code* $n\}$;
— *l'ensemble* $\mathtt{Clos} = \{n \mid n$ *est le code d'une formule close*$\}$;

Démonstration. La première assertion se démontre par récurrence primitive sur les arbres (lemme 3.2.4). Il suffit de suivre la définition inductive habituelle de variable libre, sachant que la fonction : $i \mapsto \ulcorner x_i \urcorner$ est récursive primitive (lemme 3.2.7) :

$$\begin{array}{ll}
\texttt{libre}(i,0) = 0 & \\
\texttt{libre}(i,\ulcorner x_i \urcorner) = 1 & \\
\texttt{libre}\big(i,\langle \ulcorner x_i \urcorner, n_2\rangle_{\#\{\forall\}}\big) = 0 & \\
\texttt{libre}\big(i,\langle \ulcorner x_j \urcorner, n_2\rangle_{\#\{\forall\}}\big) = \texttt{libre}(i,n_2) & \text{si } j \neq i \\
\texttt{libre}(i,\langle n_1,n_2\rangle_e) = \texttt{libre}(i,n_1) + \texttt{libre}(i,n_2) & \text{dans tous les autres cas.}
\end{array}$$

On a alors :

$$(i,n) \in \texttt{Libre} \text{ ssi } n \in \texttt{Term} \cup \texttt{Form} \text{ et } \texttt{libre}(i,n) > 0\,.$$

Pour la seconde assertion (codes de formules closes), il suffit de remarquer que le code d'une variable qui apparait dans une expression est strictement inférieur au code de celle-ci. L'ensemble des formules closes se définit par quantification bornée :

$$(i,n) \in \texttt{Clos} \text{ ssi } n \in \texttt{Form} \text{ et } \forall i < n\ (i,n) \notin \texttt{Libre}\,.$$

Il est donc récursif primitif. ■

On aura besoin d'une fonction qui code la substitution d'un terme t à toute occurrence libre d'une variable x_i dans une expression E, terme ou formule, notée $E[t/x_i]$. On va se contenter d'une fonction qui ne renomme pas les variables en cas de capture, c'est-à-dire que la substitution n'est correcte que si les variables libres dans le terme t ne sont pas capturées lors de la substitution par des quantificateurs de E (on dira qu'une telle substitution est propre). Les occurrences liées de la variable x_i ne sont bien-sûr pas substituées.

Lemme 3.2.8 *Il existe une fonction récursive primitive* $\texttt{subst} : \mathbb{N}^3 \to \mathbb{N}$*, telle que si E est une expression où la variable x_i n'apparait pas libre dans le champ d'un quantificateur sur une variable qui apparait dans t :*

$$\texttt{subst}(\ulcorner E \urcorner, \ulcorner t \urcorner, i) = \ulcorner E[t/x_i] \urcorner\,.$$

Démonstration. La définition de la fonction subst qui suit est récursive primitive d'après le lemme 3.2.4, sachant que la fonction : $i \mapsto \ulcorner x_i \urcorner$ est récursive primitive (lemme 3.2.7).

$$\begin{array}{ll}
\texttt{subst}\big(0,p,i\big) = 0 & \\
\texttt{subst}\big(\ulcorner x_i \urcorner, p, i\big) = p & \\
\texttt{subst}\big(\ulcorner x_j \urcorner, p, i\big) = \ulcorner x_j \urcorner & \text{si } \ulcorner x_j \urcorner \neq \ulcorner x_i \urcorner \\
\texttt{subst}\big(\langle \ulcorner x_i \urcorner, n_2\rangle_{\#\{\forall\}}, p, i\big) = \langle \ulcorner x_i \urcorner, n_2\rangle_{\#\{\forall\}} & \\
\texttt{subst}\big(\langle \ulcorner x_j \urcorner, n_2\rangle_{\#\{\forall\}}, p, i\big) = \langle \ulcorner x_j \urcorner, \texttt{subst}\big(n_2,p,i\big)\rangle_{\#\{\forall\}} & \text{si } \ulcorner x_j \urcorner \neq \ulcorner x_i \urcorner \\
\texttt{subst}\big(\langle n_1,n_2\rangle_e, p, i\big) = \langle \texttt{subst}\big(n_1,p,i\big), \texttt{subst}\big(n_2,p,i\big)\rangle_e & \text{dans tous les autres cas.}
\end{array}$$

■

3.3 La vérité dans l'arithmétique n'est pas définissable

On peut maintenant revenir de façon plus précise sur la non définissabilité de la vérité (théorème de Tarski page 66). Un *prédicat de vérité* est un prédicat V à une variable libre, prenons x_0, vérifiant

$$\mathbb{N} \models V[\ulcorner F \urcorner / x_0] \text{ ssi } F \text{ est une formule close et } \mathbb{N} \models F\,.$$

Théorème 3.3.1 (théorème de Tarski) *Il n'existe pas de prédicat de vérité pour les formules de l'arithmétique qui soit définissable dans l'arithmétique.*

On va en donner deux démonstrations.

Démonstration (1). La première est celle déjà évoquée : un prédicat de vérité V fournirait une énumération des sous-ensembles arithmétiques de $\mathbb{N}$ par substitution :

$$\left\{(i,n) \mid V\left[\ulcorner F[\underline{n}/x_0]\urcorner/x_0\right] \text{ pour } F \text{ formule de code } i \text{ ayant pour seule variable libre } x_0\right\}$$

$$= \{(i,n) \mid V\left[\mathtt{subst}(i,\mathtt{num}(n),0)/x_0\right] \wedge \mathtt{subst}(i,\ulcorner 0\urcorner,0) \in \mathtt{Clos}\} \ . \qquad \blacksquare$$

On donne maintenant la démonstration originale de Tarski, qui suit celle de Gödel pour son premier théorème d'incomplétude. L'argument essentiel reste un argument diagonal (qui a été utilisé pour montrer que la hiérarchie est stricte dans la première démonstration).

Démonstration (2). Soit un prédicat représenté par une formule à une variable libre V. On va construire par diagonalisation une formule D à une variable libre x_0 telle que, si V est un prédicat de vérité, $D[\ulcorner F\urcorner/x_0]$ signifie « $F[\underline{\ulcorner F\urcorner}/x_0]$ est fausse » (F appliquée à son propre code est fausse). On aura une contradiction en appliquant D à son propre code.

On appelle r la fonction récursive primitive $r : (x,n) \mapsto \mathtt{subst}(x,\mathtt{num}(n),1)$, en particulier :

$$r(\ulcorner F\urcorner, n) = \ulcorner F[\underline{n}/x_1]\urcorner \ .$$

Le graphe de r est définissable dans $\mathbb{N}$ par une formule que l'on note $\underline{z = r(x,y)}$ (on pourra choisir les nom des variables), et on peut supposer que x_0 et x_1 n'apparaissent pas liées dans cette formule.

On suppose l'existence d'un prédicat de vérité arithmétique V avec x_0 pour seule variable libre, et tel que x_1 n'apparaisse pas liée dans V. Soit alors

$$D[x_1] \equiv_d \forall x_0 \left(\underline{x_0 = r(x_1,x_1)} \to \neg V[x_0]\right).$$

On a bien

$$D[n/x_1] \equiv_{\mathbb{N}} \neg V[r(n,n)/x_0]$$

d'où

$$D[\ulcorner F\urcorner/x_1] \equiv_{\mathbb{N}} \neg V[\ulcorner F[\underline{\ulcorner F\urcorner}/x_1]\urcorner/x_0]$$

en particulier

$$D[\ulcorner D\urcorner/x_1] \equiv_{\mathbb{N}} \neg V[\ulcorner D[\underline{\ulcorner D\urcorner}/x_1]\urcorner/x_0] \ .$$

Pour tout prédicat $V[x_0]$ à une variable libre on peut donc construire une formule close $G = D[\underline{\ulcorner D\urcorner}/x_1]$ telle que $V[\ulcorner G\urcorner/x_0]$ ne peut représenter la vérité de G, d'où la conclusion. $\blacksquare$

La démonstration utilise une construction de point fixe pour les formules. On a que si A est une formule à une seule variable libre x_0. Alors il existe un énoncé G tel que :

$$\mathbb{N} \models G \leftrightarrow A[\ulcorner G\urcorner/x_0] \ .$$

Le résultat, pour la formule $\neg V$, est démontré comme étape intermédiaire dans la seconde démonstration du théorème de Tarski ci-dessus. On l'adaptera à la prouvabilité avec les lemmes 3.5.9 page 78 et 3.6.13 page 87.

3.4 Démontrabilité et décidabilité

La notion de démonstration a beaucoup à voir avec la calculabilité. Une démonstration est un objet syntaxique fini dont il doit être possible mécaniquement, du moins s'il s'agit d'une preuve formelle, de vérifier qu'elle est correcte. Dit autrement on s'attend à la décidabilité du problème de reconnaître si une suite (ou un arbre) de formules logiques est ou non une démonstration. L'ensemble des théorèmes, les énoncés pour lesquels il existe une preuve, est donc effectivement énumérable (tout ceci modulo codage des formules).

Il est bien sûr nécessaire d'être plus précis. Ainsi on peut supposer qu'une démonstration se fait dans une certaine théorie axiomatique qui est elle même finie, ou présentée de façon finie, à l'aide par exemple de schémas d'axiomes (on va plutôt donner une définition utilisant la calculabilité). Cependant, on peut d'ores et déjà remarquer que si l'ensemble des théorèmes d'une théorie arithmétique

comme l'arithmétique de Peano est effectivement énumérable, soit Σ_1, en particulier arithmétique, l'ensemble des énoncés vrais dans $\mathbb{N}$ ne l'est pas, comme on l'a vu au chapitre précédent. En supposant qu'on ne peut démontrer que des énoncés vrais dans l'arithmétique de Peano, on a donc nécessairement des énoncés vrais non démontrables dans cette théorie. C'est un premier résultat d'incomplétude, plus faible que celui de Gödel, que l'on va préciser puis renforcer dans ce chapitre pour obtenir le premier théorème d'incomplétude.

3.4.1 Théories décidables et théories complètes

Une *théorie* du calcul des prédicats égalitaire dans un certain langage de signature $\mathscr{S}$ est un ensemble d'énoncés (formules closes) qui est clos par déduction (on renvoie à la section suivante pour la définition de la déduction). Un énoncé de la théorie est appelé *théorème* de la théorie. On a choisi d'identier une théorie à l'ensemble de ses théorèmes.

Un système d'axiomes pour une théorie est un ensemble d'énoncés dont la clôture par déduction est la théorie.

Il existe bien entendu plusieurs systèmes d'axiomes pour une même théorie : il est toujours possible d'ajouter ou d'enlever un énoncé universellement valide comme $\forall x\ x = x$ aux axiomes sans changer les théorèmes. Pour le même genre de raison une théorie, vue on le rappelle comme l'ensemble de ses théorèmes, est nécessairement infinie.

— Une théorie $\mathscr{T}$ est dite *cohérente*, ou *consistante* quand il existe un énoncé du langage de la théorie, qui n'est pas un théorème de $\mathscr{T}$;
— comme $\bot \to F$ est toujours démontrable, $\mathscr{T}$ est cohérente si et seulement si $\bot$ n'est pas un théorème de $\mathscr{T}$;
— une *hypothèse de cohérence* pour $\mathscr{T}$ est n'importe quelle hypothèse sur $\mathscr{T}$ qui a pour conséquence que toutes les formules ne sont pas démontrables, appelée dans ce contexte *cohérence simple* de la théorie ;
— une théorie $\mathscr{T}$ est dite *complète* quand pour tout énoncé du langage de la théorie, soit lui même, soit sa négation est un théorème de $\mathscr{T}$;
— un ensemble de formules logiques est dit décidable quand l'ensemble des codes de ses formules est décidable (de fait, cette définition ne dépend pas du choix du codage, tant qu'il est raisonnable) ;
— en particulier une théorie est dite *décidable* quand l'ensemble de ses théorèmes est décidable, *indédicable* sinon.
— un ensemble de formules logiques est dit effectivement énumérable quand l'ensemble des codes de ses formules est effectivement énumérable ;
— une théorie est dite *effectivement énumérable* quand l'ensemble de ses théorèmes est effectivement énumérable ;
— une théorie est dite *récursivement axiomatisable*, ou *effectivement axiomatisable*, quand elle possède un système d'axiomes décidable.

On va montrer que ces deux dernières notions sont équivalentes. Le lemme suivant donne l'une des implication. Remarquons que l'ensemble des théorèmes d'une théorie est toujours non vide (et même toujours infini), donc si la théorie est effectivement énumérable, l'ensemble des codes de ses théorèmes est l'ensemble image d'une fonction totale calculable.

Lemme 3.4.1 (astuce de Craig) *Une théorie dont l'ensemble des (codes de) théorèmes est effectivement énumérable, est effectivement axiomatisable, plus précisément, si $(A_i)_{i\in\mathbb{N}}$ est une énumération des théorèmes d'une théorie telle que la fonction : $i \mapsto \ulcorner A_i \urcorner$ est totale calculable, alors $(\bigwedge_{j=1}^{i} A_j)_{i\in\mathbb{N}}$ est un système d'axiomes décidable pour cette théorie.*

Démonstration. Ce lemme repose sur le fait que la la déduction permet de démontrer effectivement quelques propriétés évidentes de la conjonction, à savoir que B (ainsi que A) se déduit de $A \wedge B$.

On vérifie facilement que, si la fonction totale : $i \mapsto \ulcorner A_i \urcorner$ est calculable, la fonction totale $i \mapsto \ulcorner \bigwedge_{j=1}^{i} A_j \urcorner$ est calculable et strictement croissante. L'ensemble image d'une telle fonction est décidable (l'appartenance à l'ensemble image se définit par minimisation bornée, voir exercice 16). ■

La réciproque est démontrée à la proposition 3.4.7. Si on veut être précis, elle demande de formaliser la déduction, mais elle est assez intuitive : on peut voir un système de déduction comme une machine pour produire la suite des théorèmes à partir d'un ensemble d'axiomes décidable.

L'astuce de Craig permet également d'associer à un ensemble d'axiomes dont l'ensemble des codes est effectivement énumérables, un ensemble d'axiomes décidable de la même théorie, et donc il suffit de parler de théorie effectivement axiomatisable.

L'astuce de Craig est plutôt artificielle, mais, de fait, les systèmes d'axiomes des théories ordinairement utilisés en mathématiques sont décidables : cela signifie simplement que l'on peut reconnaître de façon mécanique si un énoncé est ou non un axiome. Les théories usuelles sont présentées de façon finie. Elles peuvent être *finiment axiomatisables*, c'est-à-dire qu'elles possèdent un système d'axiomes fini, comme la théorie des groupes par exemple, et sont donc évidemment effectivement axiomatisables. Elles peuvent être infinies et utiliser des *schémas d'axiomes*, comme le schéma d'axiomes de récurrence de l'arithmétique de Peano (premier ordre). On reconnait clairement de façon décidable un axiome de récurrence, et donc une telle théorie est effectivement axiomatisable.

Admettons qu'une théorie effectivement axiomatisable a un ensemble de théorèmes effectivement énumérable (ce sera démontré rigoureusement à la proposition 3.4.7, mais on vient d'esquisser une démonstration). L'ensemble des énoncés dont la négation est démontrable est alors l'ensemble des énoncés qui ne sont pas des théorèmes : on obtient le résultat suivant comme conséquence de la proposition 2.3.6 page 39.

Proposition 3.4.2 *Une théorie effectivement axiomatisable et complète est décidable.*

3.4.2 Codage des démonstrations

Pour faciliter le codage, on choisit un système pour lequel la structure des démonstrations est la plus simple possible, comme le système « à la Hilbert » qui suit. Pour éviter les confusions, dans la suite on utilisera plutôt le mot « preuve » pour les preuves formelles dans le système de déduction considéré.

Un système de preuves

On rappelle que l'on s'est restreint aux formules construites sur $\{\to, \bot, \forall\}$, et que $\neg A$ est une abréviation de $A \to \bot$. Le système de déduction est donné par cinq schémas d'axiomes logiques, deux règles, un axiome et un schéma d'axiomes pour l'égalité.

Schémas d'axiomes

- **i.** $A \to (B \to A)$
- **ii.** $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
- **iii.** $\neg\neg A \to A$
- **iv.** $\forall x A \to A[t/x]$ pour t un terme tel que si x apparaît dans le champ d'un quantificateur portant sur une certaine variable y, y n'apparaît pas dans t (pas de capture de variable)
- **v.** $\forall x(A \to B) \to (A \to \forall x B)$ si x n'apparaît libre dans A

Règles

Modus ponens. Des formules A et $A \to B$ on déduit B :

$$\frac{A \to B \quad A}{B}$$

Généralisation. De A on déduit $\forall x\ A$ (x peut bien-sûr apparaître libre dans A).

$$\frac{A}{\forall x\ A}$$

Égalité On ajoute les axiomes de l'égalité (en calcul des prédicats égalitaires)

- **vi.** $\forall x\ x = x$
- **vii.** $\forall x \forall y [x = y \to (A[x/z] \to A[y/z])]$ où z n'est pas dans le champ d'un quantificateur sur x, ni sur y.

Une preuve dans le système d'axiomes $\mathscr{A}$ est par définition une suite finie de formules $(F_0,\dots,F_n)$ telle que pour tout $i \le n$, l'une des condition suivante est réalisée

a. F_i est un axiome logique (c'est-à-dire un instance d'un des schémas d'axiomes logique) ;

b. F_i est un axiome de l'égalité ;

c. F_i est un axiome de $\mathscr{A}$;

d. il existe $j < i$ et $k < i$ tels que $F_j = F_k \to F_i$ (*modus ponens*) ;

e. il existe $j < i$ et une variable x tels que $F_i = \forall x\, F_j$ (*généralisation*).

Une preuve de la formule F est une suite finie de formules dont la dernière est F. Une formule F est dite prouvable (ou démontrable) dans une théorie $\mathscr{T}$ axiomatisée par $\mathscr{A}$, s'il existe une preuve de F dans $\mathscr{T}$, et on écrira

$$\vdash_{\mathscr{T}} F\,.$$

Par définition tout segment initial d'une preuve est une preuve, et donc, comme on s'y attend, toute formule qui intervient dans une preuve est elle même prouvable.

Exercice 21 Montrer que pour toute formule F

1. la formule $F \to F$ est prouvable (appliquer une instance de ii à deux instances de i) ;

2. la formule $\bot \to F$ est prouvable (utiliser iii).

Codage

On sait coder les formules, les suites finies, et donc les preuves. On choisit de coder les preuves $(F_0,\dots,F_n)$ par l'entier $[\ulcorner F_0\urcorner;\dots;\ulcorner F_n\urcorner]$ (voir le codage des suites finies page 8). Les codes de preuve en calcul des prédicats égalitaire (pas d'axiome non logique) se manipulent alors par des fonctions récursives primitives.

Lemme 3.4.3 *Soit* $\mathtt{Capt}$ *l'ensemble des triplets* $(i,j,\ulcorner A\urcorner)$ *d'entiers naturels tels que la variable* x_j *apparaît libre dans* A *dans le champ d'un quantificateur de nom* x_i*, c'est-à-dire que pour* $(i,j,\ulcorner A\urcorner) \in \mathtt{Capt}$ *il y aura capture de la variable* x_j *si on substitue un terme contenant cette variable à* x_i *dans* A*. Alors l'ensemble* $\mathtt{Capt}$ *est récursif primitif.*

Démonstration. La démonstration utilise la définition par récurrence primitive sur la structure d'arbre (lemme 3.2.4), la fonction `libre` du lemme 3.2.7 qui compte le nombre d'occurrences libres d'une variable dans une formule, et le fait que l'ensemble des codes de formules `Form` est récursif primitif (lemme 3.2.5).

$\mathtt{capt}\big(i,j,0\big) = 0$
$\mathtt{capt}\big(i,j,\ulcorner x_i\urcorner\big) = 0$
$\mathtt{capt}\big(i,j,\langle\ulcorner x_i\urcorner,n_2\rangle_{\#\{\forall\}}\big) = \mathtt{libre}(j,n_2)$
$\mathtt{capt}\big(i,j,\langle n_1,n_2\rangle_e\big) = \mathtt{capt}\big(i,j,n_1\big) + \mathtt{capt}\big(i,j,n_2\big)$ dans tous les autres cas.

On a alors $(i,j,n) \in \mathtt{Capt}$ ssi $\mathtt{capt}(i,j,n) > 0$ et $n \in \mathtt{Form}$. ■

Lemme 3.4.4 *Les ensembles* $\mathtt{Ax_i}$, $\mathtt{Ax_{ii}}$, $\mathtt{Ax_{iii}}$, $\mathtt{Ax_{iv}}$, $\mathtt{Ax_v}$, $\mathtt{Ax_{vi}}$, $\mathtt{Ax_{vii}}$ *des codes des formules qui sont instances des schémas d'axiomes ci-dessus (numérotés de la même façon) sont récursifs primitifs.*

Démonstration. On ne détaille pas les démonstrations qui sont toutes similaires. Le lemme 3.4.3 est utile pour n'engendrer que des instances correctes (sans capture de variable) de $\mathtt{Ax_{iv}}$ et $\mathtt{Ax_{vii}}$, le lemme 3.2.8 pour $\mathtt{Ax_{iv}}$, et le lemme 3.2.7 pour $\mathtt{Ax_v}$. ■

Lemme 3.4.5 *Soit* $\mathscr{A}$ *un ensemble d'axiomes décidable, alors l'ensemble des codes de preuves dans* $\mathscr{A}$ *est un ensemble décidable.*

Démonstration. On utilise le lemme précédent, et que A, l'ensemble des codes des axiomes de $\mathscr{A}$ est décidable. L'ensemble $\mathtt{Ax} = \mathtt{Ax}_{\mathrm{i}} \cup \mathtt{Ax}_{\mathrm{ii}} \cup \mathtt{Ax}_{\mathrm{iii}} \cup \mathtt{Ax}_{\mathrm{iv}} \cup \mathtt{Ax}_{\mathrm{v}} \cup \mathtt{Ax}_{\mathrm{vi}} \cup \mathtt{Ax}_{\mathrm{vii}} \cup A$ est donc décidable. Un entier n code une preuve si et seulement si

$$\forall i \leq \mathrm{len}(n) \left(\begin{array}{c} \mathrm{nth}(n,i) \in \mathtt{Ax} \\ \text{ou} \\ \exists j < i \; \exists k < i \; \langle \mathrm{nth}(n,k), \mathrm{nth}(n,i) \rangle_{\#\{\rightarrow\}} = \mathrm{nth}(n,j) \\ \text{ou} \\ \exists j < i \; \exists k \leq n \; \langle \ulcorner x_k \urcorner, \mathrm{nth}(n,j) \rangle_{\#\{\forall\}} = \mathrm{nth}(n,i) \end{array} \right)$$

et donc l'ensemble des preuves est décidable. ■

Le lemme suivant correspond à l'idée simple que la vérification d'une preuve formelle dans une théorie ordinaire est quelque chose de mécanique.

Lemme 3.4.6 *Soit $\mathscr{T}$ une théorie effectivement axiomatisable, alors l'ensemble $\mathtt{Prov}_{\mathscr{T}}$ des couples d'entiers m et n vérifiant :*

$$(m,n) \in \mathtt{Prov}_{\mathscr{T}} \quad \textit{ssi} \quad n \textit{ est le code d'une preuve de la formule } F \textit{ de code } n$$

est décidable.

On obtient les théorèmes par quantification existentielle sur cet ensemble : ce sont les énoncés dont il existe une preuve, d'où la proposition suivante.

Proposition 3.4.7 *Une théorie $\mathscr{T}$ est effectivement axiomatisable si et seulement si l'ensemble (des codes) de ses théorèmes est effectivement énumérable. En particulier, si $\mathscr{T}$ est effectivement axiomatisable, il existe une formule Σ_1 du langage de l'arithmétique à une variable libre telle que pour tout entier n*

$$\mathrm{Dem}_{\mathscr{T}}[n] \quad \textit{ssi} \quad n \textit{ est le code d'une formule close } E \textit{ prouvable dans } \mathscr{T}$$

ce qui, dans le cas d'un énoncé (formule close) E, donne :

$$\mathbb{N} \models \mathrm{Dem}_{\mathscr{T}}[\ulcorner E \urcorner] \quad \textit{ssi} \quad \vdash_{\mathscr{T}} E \,.$$

Démonstration. On a déjà vu la réciproque (lemme de Craig 3.4.1 page 72). Pour le sens direct il suffit de définir $\mathrm{Dem}_{\mathscr{T}}$:

$$\mathrm{Dem}_{\mathscr{T}}[n] \equiv_d \exists m \; (m,n) \in \mathtt{Prov}_{\mathscr{T}} \,.$$

■

Remarque. On a écrit $\mathbb{N} \models F$ (dans le cas particulier où F est $\mathrm{Dem}_{\mathscr{T}}[\ulcorner E \urcorner]$ pour bien marquer la différence entre cette notion sémantique et la notion syntaxique de démontrabilité qui suit, mais comme $\mathbb{N}$ est le modèle standard de l'arithmétique, c'est ce que l'on écrivait simplement jusqu'à présent « F », au sens où on affirme F, ou peut-être plus explicitement « F est vrai ».

3.5 Le premier théorème d'incomplétude de Gödel

3.5.1 Une version faible du théorème de Gödel

On peut maintenant préciser et démontrer la version faible du premier théorème d'incomplétude évoquée en introduction.

Proposition 3.5.1 (version faible du premier théorème d'incomplétude) *Soit $\mathscr{T}$ une théorie dans le langage de l'arithmétique de Peano, effectivement axiomatisable, et qui a pour modèle $\mathbb{N}$, l'ensemble des entiers naturels muni de ses opérations usuelles, alors il existe un énoncé G qui est vraie dans $\mathbb{N}$ et non démontrable dans $\mathscr{T}$, la négation de G n'est pas non plus démontrable :*

$$\textit{Si } \mathbb{N} \models \mathscr{T}, \textit{ alors il existe un énoncé } G \textit{ tel que } \mathbb{N} \models G, \nvdash_{\mathscr{T}} G \textit{ et } \nvdash_{\mathscr{T}} \neg G \,.$$

Démonstration. L'ensemble des théorèmes de $\mathscr{T}$ est effectivement énumérable. L'ensemble des formules vraies dans $\mathbb{N}$ n'est pas arithmétique d'après le théorème de Tarski (3.3.1), a fortiori pas effectivement énumérable, et contient l'ensemble des théorèmes de $\mathscr{T}$. Cette inclusion est donc stricte, c'est-à-dire qu'il existe une formule close G vraie dans $\mathbb{N}$ et non démontrable dans $\mathscr{T}$. La négation de G est fausse donc non démontrable dans $\mathscr{T}$ par hypothèse. ■

L'hypothèse $\mathbb{N} \models \mathscr{T}$ est une *hypothèse de cohérence* : elle a bien pour conséquence que certaines formules ne sont pas démontrables dans $\mathscr{T}$, en l'occurrence les formules fausses dans $\mathbb{N}$.

Mais cette hypothèse de cohérence est très forte, beaucoup plus forte que la cohérence simple, qui s'exprime dans l'arithmétique par la formule $\neg \mathrm{Dem}_{\mathscr{T}}[\ulcorner \bot \urcorner]$, donc par une formule Π_1.

En revanche dans le cas de l'arithmétique de Peano, à cause du schéma de récurrence qui met en jeu des formules de complexité logique arbitraire, $\mathbb{N} \models \mathscr{T}$ ne s'exprime même pas dans l'arithmétique, alors que l'on a vu comment coder la démontrabilité, et que la vérité dans $\mathbb{N}$ d'une formule donnée se code dans l'arithmétique par la définition de la satisfaction de Tarski. On a finalement introduit une hypothèse beaucoup plus complexe que la conclusion. On va donc améliorer ce théorème de deux façons :

— en proposant une hypothèse de cohérence beaucoup plus faible ;
— en précisant la complexité logique de la formule obtenue.

Dorénavant nous nous intéressons à des théories dont le langage est, sauf précision, le langage $\mathscr{L}_{\mathrm{PA}}$ de l'arithmétique de Peano (le langage égalitaire du premier ordre de signature $(0, s, +, \times, \leq)$).

3.5.2 Les formules Π_1

Si on reprend la démonstration de la version faible du théorème, on observe que l'on a juste besoin d'un ensemble d'énoncés vrais qui n'est pas effectivement énumérable. Pour cela il suffit de considérer les formules Π_1.

Lemme 3.5.2 *L'ensemble des énoncés Π_1 de l'arithmétique est récursif primitif.*

Démonstration. L'ensemble des formules Σ_0 est récursif primitif. En effet la définition inductive de la classe Σ_0 (section 3.1.1 page 55) se traduit en une définition de la fonction caractéristique des codes de formules Σ_0 utilisant la récurrence primtive sur les arbres du lemme 3.2.4 page 68. L'ensemble des formule Π_1 se définit aussi inductivement : une formule Σ_0, est Π_1, si une formule F est Π_1, alors $\forall x\, F$ est Π_1 (voir section 3.1.3 page 61), l'ensemble des codes de formules Π_1 est donc récursif primitif. L'ensemble des énoncés (formules closes) Π_1 est donc récursif primitif (lemme 3.2.7 page 69). ■

Lemme 3.5.3 *Soit $\mathscr{T}$ une théorie dans le langage de l'arithmétique de Peano qui est effectivement axiomatisable, alors l'ensemble des énoncés Π_1 de l'arithmétique démontrables dans $\mathscr{T}$ est effectivement énumérable.*

Démonstration. Conséquence du lemme précédent et de la proposition 3.4.7 page précédente . ■

Pour la version faible de l'incomplétude on a utilisé que la vérité des énoncés arithmétiques n'est pas arithmétique. Il suffit en fait du résultat suivant.

Lemme 3.5.4 *L'ensemble des énoncés Π_1 vrais de l'arithmétique (les énoncés Π_1 F tels que $\mathbb{N} \models F$, où $\mathbb{N}$ est le modèle standard) n'est pas effectivement énumérable.*

Démonstration. On montre que si l'ensemble des énoncés Π_1 vrais était effectivement énumérable, l'ensemble K^{c} des codes i des machines qui ne s'arrêtent pas sur i, serait effectivement énumérable.

On note $\underline{x_0 \in K^{\mathrm{c}}}$ une formule Π_1 de l'arithmétique (langage $\mathscr{L}_{\mathrm{PA}}$) qui définit l'ensemble K^{c} dans $\mathbb{N}$, formule qui existe d'après la proposition 3.1.15.iii page 62. Alors :

$$n \in K^{\mathrm{c}} \Leftrightarrow \mathbb{N} \models \underline{x_0 \in K^{\mathrm{c}}}[\underline{n}/x_0]$$

L'énoncé $\underline{x_0 \in K^{\mathrm{c}}}[\underline{n}/x_0]$ est Π_1, comme la fonction de substitution (voir lemme 3.2.8 page 70) `subst` ainsi que $\mathtt{num} : n \mapsto \ulcorner \underline{n} \urcorner$ sont récursives primitives, et que $\mathtt{subst}(\ulcorner \underline{x_0 \in K^{\mathrm{c}}} \urcorner, \ulcorner \underline{n} \urcorner, 0)$ est $\ulcorner \underline{x_0 \in K^{\mathrm{c}}}[\underline{n}/x_0] \urcorner$, par composition si l'ensemble des énoncés Π_1 vrais était semi-décidable (domaine d'une fonction partielle calculable), K^{c} le serait également ce qui contredit l'indécidabilité du problème de l'arrêt (proposition 2.3.7 page 40). ■

La proposition suivante, qui est une première version du théorème d'incomplétude de Gödel, suit immédiatement des deux derniers lemmes.

Théorème 3.5.5 (premier théorème d'incomplétude, version 1) *Soit $\mathscr{T}$ une théorie arithmétique effectivement axiomatisable telle que tous les énoncés Π_1 démontrables dans $\mathscr{T}$ sont vrais :*

$$\textit{pour tout énoncé } \Pi_1 \ F, \textit{ si } \vdash_{\mathscr{T}} F, \textit{ alors } \mathbb{N} \models F \ .$$

Alors il existe un énoncé Π_1, soit G, qui est vrai, mais pas démontrable dans $\mathscr{T}$:

$$\mathbb{N} \models G \textit{ mais } \nvdash_{\mathscr{T}} G \ .$$

L'hypothèse que tous les énoncés Π_1 démontrables sont vrais est bien une hypothèse de cohérence (alors les énoncés Π_1 faux ne sont pas démontrables), qui de plus est arithmétique, plus précisément elle est elle-même Π_1. Mais nous allons voir surtout qu'elle est équivalente à la cohérence simple ($\nvdash_T \bot$) modulo une hypothèse très naturelle sur la la théorie $\mathscr{T}$: la Σ-complétude.

3.5.3 Σ-complétude

On rappelle que la classe des formules Σ_0 est la plus petite classe de formules contenant les égalités et inégalités polynomiales et close par opérations booléennes et quantifications bornées.

Les formules atomiques closes de $\mathscr{L}_{\mathrm{PA}}$ sont des égalités ou des inégalités polynomiales sur des numéraux, et donc leur vérité ou leur fausseté dans $\mathbb{N}$ se vérifie par un calcul simple. On montre facilement étape par étape que la vérité ou la fausseté d'une formule close obtenue par opérations booléennes et quantifications bornées à partir de telles formules peut également se calculer. Les formules closes Σ_0 vraies dans $\mathbb{N}$ sont donc « démontrables » au sens intuitif, et les formules closes Σ_0 fausses dans $\mathbb{N}$ ont donc leurs négations démontrables.

Une formule Σ_1 vraie dans $\mathbb{N}$, soit $\exists x\ A[x]$ est également intuitivement démontrable puisqu'il suffit de vérifier successivement $A[0],\ A[1],\ldots$ jusqu'à trouver un n tel que $A[n]$ est vrai, ce qui arrivera au bout d'un nombre fini de vérifications si $\exists x\ A[x]$ est vraie dans $\mathbb{N}$.

Tout ceci ne passe plus à la vérité des formules closes Π_1 (ou la fausseté des formules Σ_1). L'approche naïve, pour vérifier $\forall x A[x]$, consisterait en une infinité de vérifications :

$$A[0],\ A[1],\ldots,A[n],\ldots$$

et ce n'est plus du domaine de la preuve, qui, au sens usuel, doit rester un objet fini. Et pour cause : le théorème d'incomplétude 3.5.5 nous dira justement que pour les énoncés Π_1, vérité et démontrabilité ne peuvent pas coïncider.

Venons en à ce que l'on peut considérer raisonnablement comme une théorie pour l'arithmétique : le minimum requis est que les formules Σ_1 vraies dans $\mathbb{N}$, dont nous avons vu qu'elles sont intuitivement prouvables, le soient effectivement.

Nous appelerons donc *théorie Σ-complète* une théorie dans laquelle toutes les formules Σ_0 vraies dans $\mathbb{N}$ sont démontrables, et dorénavant nous n'envisagerons que des théories arithmétiques Σ-complètes.

Proposition 3.5.6 *Si la théorie arithmétique $\mathscr{T}$ est Σ-complète, alors tous les énoncés Σ_1 vrais dans $\mathbb{N}$ sont démontrables dans $\mathscr{T}$.*

Démonstration. Soit une formule Σ_1, $\exists x\ A[x]$, vraie dans $\mathbb{N}$, alors pour un certain entier $n \in \mathbb{N}$ (standard), la formule $A[\underline{n}]$, qui est Σ_0, est vraie dans $\mathbb{N}$ (l'entier standard n est l'interprétation du terme $\underline{n} = s^n\ 0$). Donc par hypothèse $\vdash_{\mathscr{T}} A[\underline{n}]$. On en déduit $\vdash_{\mathscr{T}} \exists x\ A$. ■

Proposition 3.5.7 *Une théorie arithmétique Σ-complète $\mathscr{T}$ est cohérente si et seulement si tous les énoncés Π_1 démontrables dans $\mathscr{T}$ sont vrais.*

Démonstration. On suppose $\mathscr{T}$ Σ-complète. Soit E un énoncé Π_1 tel que $\vdash_{\mathscr{T}} E$. Si E est faux, c'est-à-dire $\mathbb{N} \not\models \neg E$, l'énoncé $\neg E$, qui est Σ_1, est vrai, donc $\vdash_{\mathscr{T}} \neg E$ par Σ-complétude : la théorie $\mathscr{T}$ n'est pas cohérente.
Réciproquement, si $\mathscr{T}$ n'est pas cohérente, tous les énoncés sont démontrables, en particulier des énoncés Π_1 faux. ■

La première version du théorème d'incomplétude 3.5.5 page précédente se reformule donc de la façon suivante pour les théories Σ-complètes.

Théorème 3.5.8 (premier théorème d'incomplétude) *Soit $\mathscr{T}$ une théorie arithmétique effectivement axiomatisable Σ-complète et cohérente. Alors il existe un énoncé Π_1, soit G, qui est vrai, mais n'est pas démontrable dans $\mathscr{T}$:*

$$\mathbb{N} \models G \text{ mais } \nvdash_{\mathscr{T}} G\,.$$

Remarque. Comme la théorie $\mathscr{T}$ est Σ-complète, et la formule G est Π_1, on peut encore reformuler la conclusion du théorème précédent de la façon suivante. Il existe une formule Σ_0 à une variable libre H telle que :

$$\text{pour tout entier } n,\ \vdash_{\mathscr{T}} H[\underline{n}/x] \text{ mais } \nvdash_{\mathscr{T}} \forall x\, H\,.$$

3.5.4 Démonstration directe du premier théorème d'incomplétude

Il est possible de donner une démonstration du premier théorème d'incomplétude qui exhibe directement la formule G vraie et non démontrable. C'est l'argument de la démonstration originale de Gödel, qui est également utile en vue de la démonstration du second théorème d'incomplétude.

Lemme 3.5.9 (point fixe pour les formules) *Soit A une formule à une seule variable libre x_0. Alors il existe un énoncé G tel que :*

$$\mathbb{N} \models G \leftrightarrow A[\ulcorner G \urcorner / x_0]$$

et si $\mathscr{T}$ est une théorie Σ-complète alors :

$$\vdash_{\mathscr{T}} G \to A\left[\underline{\ulcorner G \urcorner} / x_0\right]\,.$$

De plus si A est une formule Π_1, G peut être choisie Π_1.

Démonstration. On reprend le résultat de point fixe utilisé dans la démonstration du théorème de Tarski 3.3.1 page 70. La démonstration est aussi très similaire à celle du théorème de point fixe des fonctions calculable 2.4.1 page 45. On appelle r la fonction récursive primitive $r : (x,n) \mapsto \mathtt{subst}(x, \mathtt{num}(n), 1)$, en particulier :

$$r(\ulcorner F \urcorner, n) = \ulcorner F[\underline{n}/x_1] \urcorner\,.$$

Le graphe de r est définissable dans $\mathbb{N}$ par une formule Σ_1 que l'on note $\underline{z = r(x,y)}$ (on pourra choisir les nom des variables), et on peut supposer que x_0 et x_1 n'apparaissent pas liées dans cette formule.

La formule D à une seule variable libre x_1 est définie par :

$$D[x_1] \equiv_d \forall x_0 \left(\underline{x_0 = r(x_1,x_1)} \to A[x_0]\right)$$

On a bien :

$$D[\underline{n}/x_1] \equiv_{\mathbb{N}} A[\underline{r(n,n)}/x_0]\,.$$

On pose maintenant $G \equiv_d D[\underline{\ulcorner D \urcorner}/x_1]$, d'où $\ulcorner G \urcorner = r(\ulcorner D \urcorner, \ulcorner D \urcorner)$. Du fait que $\underline{x_0 = r(x_1,x_1)}$ définit le graphe de r on a bien :

$$\mathbb{N} \models G \leftrightarrow A[\ulcorner G \urcorner / x_0]\,.$$

Le sens direct est démontrable, en effet, la formule $\underline{x_0 = r(x_1,x_1)}$ étant Σ_1 et $\mathscr{T}$ étant Σ-complète :

$$\vdash_{\mathscr{T}} \underline{x_0 = r(x_1,x_1)}\left[\underline{r(\ulcorner D \urcorner, \ulcorner D \urcorner)}/x_0, \underline{\ulcorner D \urcorner}/x_1\right]$$

et en reprenant la définition de D :

$$\vdash_{\mathscr{T}} D\left[\underline{\ulcorner D \urcorner}/x_1\right] \to A\left[\underline{r(\ulcorner D \urcorner, \ulcorner D \urcorner)}/x_0\right]$$

soit :

$$\vdash_{\mathscr{T}} G \to A\left[\underline{\ulcorner G \urcorner}/x_0\right]$$

À nouveau en reprenant la définition de D on voit que si A est Π_1, la formule D équivaut logiquement à une formule Π_1, donc la formule G également. ■

On utilise maintenant ce lemme pour une démonstration plus directe du premier théorème d'incomplétude.

Démonstration (premier théorème d'incomplétude, démonstration directe). Soit $\mathscr{T}$ une théorie Σ-complète. Soit $\mathrm{Dem}_{\mathscr{T}}$ une formule Σ_1 à une seule variable libre x_0 qui définit la prouvabilité dans $\mathscr{T}$. Le lemme de point fixe est appliqué à la formule Π_1 $\neg\mathrm{Dem}_{\mathscr{T}}$. On obtient une formule G vérifiant en particulier :

$$\mathbb{N} \models \neg\mathrm{Dem}_{\mathscr{T}}\left[\ulcorner G\urcorner / x_0\right] \to G \tag{1}$$

$$\vdash_{\mathscr{T}} G \to \neg\mathrm{Dem}_{\mathscr{T}}\left[\underline{\ulcorner G\urcorner} / x_0\right] . \tag{2}$$

c'est-à-dire que G n'est pas prouvable dans $\mathscr{T}$, alors G est vraie, et la réciproque est démontrable dans $\mathscr{T}$: de G on déduit dans $\mathscr{T}$ que G n'est pas prouvable dans $\mathscr{T}$.. De plus :

On va montrer que si G est prouvable dans $\mathscr{T}$, alors la théorie est incohérente. La démonstration est rédigée volontairement de façon très formelle en vue de la démonstration du second théorème d'incomplétude (qui consistera essentiellement à la formaliser).

On suppose $\vdash_{\mathscr{T}} G$, c'est-à-dire :

$$\mathbb{N} \models \mathrm{Dem}_{\mathscr{T}}\left[\ulcorner G\urcorner / x_0\right] .$$

Comme $\mathrm{Dem}_{\mathscr{T}}$ est Σ_1 :

$$\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}}\left[\underline{\ulcorner G\urcorner} / x_0\right] . \tag{3}$$

Par ailleurs, de (2) et à nouveau de $\vdash_{\mathscr{T}} G$, on déduit par *modus ponens* que :

$$\vdash_{\mathscr{T}} \neg\mathrm{Dem}_{\mathscr{T}}\left[\underline{\ulcorner G\urcorner} / x_0\right] . \tag{4}$$

Toujours par *modus ponens* ($\neg A$ étant une abréviation pour $A \to \bot$) on déduit de (4) et (3) que :

$$\vdash_{\mathscr{T}} \bot .$$

On a bien par contraposée que si T est cohérente, la formule G n'est pas prouvable, et donc G est vraie d'après (1). ■

3.5.5 Quelques théories Σ-complètes

Les formules Σ_0 vraies.

La théorie Σ-complète minimale est celle dont les axiomes sont toutes les formules Σ_0 vraies dans $\mathbb{N}$. C'est une théorie dont il serait assez facile de voir directement qu'elle est effectivement axiomatisable : vérifier qu'une formule Σ_0 est vraie est décidable par la procédure brièvement décrite au paragraphe précédent.

Il est cependant possible de donner un système d'axiomes, qui reste infini, et bien entendu décidable, mais plus simple à décrire formellement. La théorie R^- est la théorie axiomatisée par les cinq schémas d'axiomes :

$\mathrm{R}_{1,n,p}$ $\neg\ \underline{n} = \underline{p}$, pour $n \neq p$, $n, p \in \mathbb{N}$;

$\mathrm{R}_{2,n,p}$ $\underline{n+p} = \underline{n} + \underline{p}$, pour $n, p \in \mathbb{N}$;

$\mathrm{R}_{3,n,p}$ $\underline{n \cdot p} = \underline{n} \cdot \underline{p}$, pour $n, p \in \mathbb{N}$;

$\mathrm{R}_{4,n,p}$ $\underline{n} \leq \underline{p}$, pour $n \leq p$, $n, p \in \mathbb{N}$;

$\mathrm{R}_{5,n}$ $\forall x \left[x \leq \underline{n} \to (x = \underline{0} \vee \cdots \vee x = \underline{n})\right]$, pour $n \in \mathbb{N}$.

Lemme 3.5.10 *La théorie* R^- *a pour conséquence les énoncés suivant :*

i. $\mathrm{R}^- \vdash \forall x \left[x \leq \underline{n} \leftrightarrow (x = \underline{0} \vee \cdots \vee x = \underline{n})\right]$, *pour* $n \in \mathbb{N}$.

ii. $\mathrm{R}^- \vdash \neg\ \underline{n} \leq \underline{p}$, *pour* $p < n$, $n, p \in \mathbb{N}$.

Démonstration. On a **i** par R_5 pour le sens direct, R_4 pour la réciproque. On a **ii** par R_5 et R_1. ■

Proposition 3.5.11 *La théorie* R^- *de Robinson est exactement la théorie qui a pour axiomes tous les énoncés* Σ_0 *vrais. Elle est* Σ*-complète et c'est la plus petite théorie cohérente* Σ*-complète.*

Démonstration. Tous les axiomes de R^- sont évidemment Σ_0. Réciproquement on montre par induction qu'un énoncé Σ (voir section 3.1.4 page 56) vrai est conséquence de R^-, a fortiori un énoncé Σ_0 vrai. Plus précisément on montre par induction que pour toute formule F qui est Σ et a pour variables libres $x_1,\dots,x_k$, si $\mathbb{N}\models F[\underline{n_1},\dots,\underline{n_k}]$ alors $\mathrm{R}^-\vdash F[\underline{n_1},\dots,\underline{n_k}]$.

- Supposons que F soit une formule atomique ou une négation de formule atomique. Par les axiomes R_2 et R_3 on se ramène au cas où les termes en jeu sont des numéraux. Dans le cas d'une égalité, comme la formule est vraie les deux termes de l'égalité sont identiques. Dans le cas d'une inégalité on conclut par R_4. Dans le cas d'une négation d'égalité on conclut par R_1. Dans le cas d'une négation d'inégalité, on conclut par le lemme précédent (**ii**).
- Supposons que F est $(F_1\wedge F_2)$, ou bien $(F_1\vee F_2)$. Le résultat se déduit de l'hypothèse d'induction sur F_1 et F_2.
- Supposons que F est $\forall\, y\le t\ G$. On est ramené après substitution à une formule du type $\forall\, y\le \underline{n}\ G'$ par R_2 et R_3, on conclut par le lemme précédent (**i**), et l'hypothèse d'induction sur G utilisée $n+1$ fois.
- Supposons que F est $\exists y\ G$, et (on note $\overline{x}$ pour $x_1,\dots,x_k$), $\mathbb{N}\models F[\overline{\underline{n}}/\overline{x}]$. Alors pour un certain entier p, $\mathbb{N}\models G[\underline{p}/y,\overline{\underline{n}}/\overline{x}]$ et on conclut par hypothèse d'induction. ∎

Cette théorie très faible est naturellement incomplète. Par exemple il est quasi immédiat, en construisant un modèle ad hoc qu'une formule aussi simple que $\forall x\ \neg s\,x = x$ n'y est pas démontrable, ni bien sûr sa négation qui est fausse dans $\mathbb{N}$. Mais le premier théorème de Gödel –il existe une formule vraie dans $\mathbb{N}$ qui n'est pas démontrable –, vaut pour toute extension effectivement axiomatisable et cohérente de R^-.

D'un point de vue sémantique, les axiomes $\mathrm{R}_{1,n,p}$, $\mathrm{R}_{2,n,p}$, $\mathrm{R}_{3,n,p}$ et $\mathrm{R}_{4,n,p}$ expriment que la partie standard de tout modèle de ces axiomes est isomorphe à $\mathbb{N}$. Les axiomes $\mathrm{R}_{5,n}$ expriment qu'un entier inférieur à un entier standard est forcément standard, ou, dit autrement, qu'un entier non standard ne peut être inférieur à un entier standard.

Le système R de Robinson.

Pour certains résultats, en particulier pour le théorème de Gödel-Rosser, un renforcement du théorème de Gödel, on aura besoin d'ajouter à R^- un schéma d'axiomes qui n'est pas Σ_1. Le système R est le système R^- auquel on ajoute le schéma d'axiomes :

$\mathrm{R}_{6,n}$ $\quad\forall x(\underline{n}\le x\vee x\le\underline{n})$ pour $n\in\mathbb{N}$.

La théorie R est donc évidemment Σ-complète.

On montrera le théorème de Gödel-Rosser — il existe une formule qui n'est pas démontrable et dont la négation n'est pas démontrable — pour tout extension cohérente effectivement axiomatisable de R.

D'un point de vue sémantique les axiomes $\mathrm{R}_{6,n}$ expriment que dans un modèle de R, un entier standard et un entier non standard sont toujours comparables, soit, modulo les autres axiomes de R, un entier non standard est toujours supérieur à un entier standard.

L'arithmétique finie Q de Robinson.

Un système fini d'axiomes très simple a pour conséquence les schémas précédents. C'est la théorie Q définie par les neuf axiomes qui suivent.

Q_1 $\quad\forall x\ \neg\,\mathrm{s}\,x = 0$;

Q_2 $\quad\forall x\forall y(\mathrm{s}\,x=\mathrm{s}\,y\to x=y)$;

Q_3 $\quad\forall x(\neg x=0\to\exists y\ x=\mathrm{s}\,y)$;

Q_4 $\quad\forall x\ x+0=x$;

Q_5 $\quad\forall x\forall y\ x+\mathrm{s}\,y=\mathrm{s}\,(x+y)$;

Q_6 $\quad\forall x\ x\cdot 0=0$;

Q_7 $\quad\forall x\forall y\ x\cdot\mathrm{s}\,y=x\cdot y+x$;

Q_8 $\quad\forall x\forall z\ x\le z+x$;

Q_9 $\quad\forall x\forall y(x\le y\to\exists z\ y=z+x)$.

Les deux derniers axiomes établissent juste la définition usuelle de la relation $\le$ à partir de l'addition.

Il serait également possible, de restreindre le langage à la signature $(0,\ \mathrm{s},\ +,\ \cdot)$, d'axiomatiser la théorie par les 7 premiers axiomes, qui est d'ailleurs la théorie Q originale de Robinson, et de définir l'ordre par $\exists z\ y=z+x$.

Proposition 3.5.12 *La théorie* R *est conséquence de la théorie* Q*. En particulier la théorie* Q *est* Σ*-complète.*

Démonstration. Les preuves se font la plupart du temps par récurrence (sur les entiers du meta-langage, la théorie Q ne possède pas d'axiome de récurrence !).

R_1 On montre par récurrence sur p que pour tout entier n, si $n \neq p$, $\vdash_Q \neg \underline{n} = \underline{p}$.

$p = 0$ **:** on a bien pour $n \neq 0$, $\vdash_Q \neg \underline{n} = 0$ par Q_1 (tout successeur est non nul) ;

$p \to p+1$ **:** : on suppose $\vdash_Q R_{1,n,p}$ pour tout n. On veut montrer que pour tout entier n, $\vdash_Q R_{1,n,p+1}$, soit

$$\text{si } n \neq p+1, \text{ alors } \vdash_Q \neg \underline{n} = s\,\underline{p}\,.$$

Si $n = 0$, on a le résultat par Q_1.

Si $n \neq 0$, alors $\underline{n} = s\,\underline{n-1}$. Par injectivité du successeur (Q_2), il suffit de montrer $\neg \underline{n-1} = \underline{p}$ qui suit de l'hypothèse de récurrence.

R_2 Par récurrence sur p en utilisant Q_4 et Q_5.

R_3 Par récurrence sur p en utilisant Q_6, Q_7 et les $R_{2,n,p}$.

R_4 Par les $R_{2,n,p}$ et Q_8.

R_5 Par récurrence sur n.

$n = 0$ **:** de $x \leq 0$ on déduit par Q_9 un z tel que $z + x = 0$. Or par Q_3 , Q_5 et Q_1

$$\vdash_Q \forall u \forall v (\neg u = 0 \to \neg v + u = 0)\,.$$

La contraposée donne bien $x = 0$.

$n \to n+1$ **:** on suppose $\vdash_Q \forall x \left[x \leq \underline{n} \to (x = 0 \vee \cdots \vee x = \underline{n}) \right]$ ($R_{5,n}$).

On raisonne dans Q. Supposons $x \leq s\,\underline{n}$. Si $\neg x = 0$, par Q_3, on a un y tel que $x = s\,y$, et par Q_9 un z tel que $z + x = z + s\,y = s\,\underline{n}$. Par Q_5 et l'injectivité du successeur (Q_2) on a $z + y = \underline{n}$, donc par Q_8 et l'hypothèse de récurrence, $y = 0 \vee \cdots \vee y = \underline{n}$:

$$\neg x = 0, x \leq s\,\underline{n} \vdash_Q x = s\,0 \vee \cdots \vee x = \underline{n+1}\,.$$

On a bien montré $R_{5,n+1}$.

R_6 Par récurrence sur n.

$n = 0$ **:** $\vdash_Q \forall x\, 0 \leq x$ par Q_4 et Q_8 ;

$n \to n+1$ **:** on suppose $\vdash_Q \forall x (\underline{n} \leq x \vee x \leq \underline{n})$ ($R_{6,n}$). Pour montrer $R_{6,n+1}$, distinguons suivant que x est nul ou non ;

$x = 0$ **:** on a $\vdash_Q 0 \leq s\,\underline{n}$ par $R_{4,n}$;

$\neg x = 0$ **:** par Q_3, on a un y tel que $x = s\,y$; par hypothèse de récurrence $\underline{n} \leq y \vee y \leq \underline{n}$; pour conclure il suffit de montrer que $\forall x \forall y (x \leq y \to s\,x \leq s\,y)$, qui découle par Q_8 et Q_9 de :

$$z + x = y \vdash_Q z + s\,x = s\,(z + x) = s\,y$$

relation elle-même obtenue par Q_5. ■

À nouveau cette théorie est naturellement incomplète. Un énoncé aussi simple que $\forall x\, 0 + x = x$ n'est pas démontrable dans Q.

Exercice 22 En exhibant un contre-modèle ad hoc, montrer que $\nvdash_Q \forall x\, 0 + x = x$.

Mais l'incomplétude vaut pour toute théorie qui étend Q, et le fait que Q est finiment axiomatisable sera exploité en particulier pour des résultats d'indécidabilité.

L'arithmétique de Peano.

L'arithmétique de Peano PA est essentiellement l'arithmétique de Robinson à laquelle on ajoute le schéma d'axiomes de récurrence :

schéma d'axiomes de récurrence Pour tout prédicat $P(x, x_1, \ldots, x_p)$ du langage de l'arithmétique (on note $\overline{a} = a_1, \ldots, a_p$) :

$$\forall \overline{a}\big(\, P[0,\overline{a}], \forall y \big(P[y,\overline{a}] \Rightarrow P[s\,y,\overline{a}]\big) \Rightarrow \forall x\, P[x,\overline{a}]\,\big)\ .$$

On peut ommettre l'axiome Q_3 qui se démontre par récurrence. On vérifie facilement que l'ensemble des axiomes de récurrence, et donc de l'arithmétique de Peano, est décidable.

Proposition 3.5.13 *L'arithmétique de Peano est effectivement axiomatisable et a pour conséquence la théorie* R*, en particulier elle est* Σ*-complète.*

Démonstration. Il suffit de montrer que Q est conséquence de PA, ce qui est évident. ■

3.5.6 Σ-cohérence

Le premier théorème d'incomplétude avait été présenté par Gödel comme un résultat d'indécidabilité logique : la conclusion est qu'il existe une formule G qui n'est pas démontrable et dont la négation n'est pas démontrable. Pour cela Gödel utilisait une hypothèse de cohérence supplémentaire, la ω-cohérence (voir exercice 23) qui ne lui est utile que pour assurer que la négation d'une formule Π_1 vraie n'est pas démontrable.

On introduit une hypothèse analogue, la Σ-cohérence, qui est conséquence de la ω-cohérence mais un peu plus faible (voir exercice 23).

Une théorie $\mathscr{T}$ est dite Σ*-cohérente* quand tous les énoncés Σ_1 démontrables dans $\mathscr{T}$ sont vrais :

$$\text{Pour } F\ \Sigma_1,\ \text{si } \vdash_{\mathscr{T}} F,\ \text{alors } \mathbb{N} \models F\ .$$

C'est bien une hypothèse de cohérence, puisqu'une conséquence est que les énoncés Σ_1 faux ne sont pas démontrables. Elle est plus forte que l'hypothèse de cohérence simple. Prenons par exemple l'arithmétique de Peano PA, dont on suppose qu'elle est cohérente (mais n'importe quelle théorie effectivement axiomatisable Σ-complète et cohérente conviendrait). D'après le théorème de Gödel il existe une formule vraie G Π_1 telle que $\mathrm{PA} \cup \{\neg G\}$ est cohérente. Alors $\neg G$ est démontrable dans $\mathrm{PA} \cup \{\neg G\}$, mais $\neg G$ est Σ_1 et fausse.

L'hypothèse de Σ-cohérence est l'hypothèse $\mathbb{N} \models E$ pour tout énoncé E démontrable, restreinte aux énoncés E Σ_1. Elle bien plus faible que cel'hypothèse que tous les énoncés démontrables sont vrais. En particulier elle peut se coder dans l'arithmétique : la vérité des formules Σ_1 est arithmétique, et même Σ_1.

Lemme 3.5.14 *Si* $\mathscr{T}$ *est une théorie* Σ*-cohérente, et* G *est un énoncé* Π_1 *vrai, alors* $\neg G$ *n'est pas démontrable dans* $\mathscr{T}$ *.*

Démonstration. Si G est un énoncé Π_1 vrai, alors $\neg G$ est un énoncé Σ_1 faux, qui ne peut donc être démontrable dans une théorie Σ-cohérente. ■

En appliquant ce lemme à la conclusion du premier théorème d'incomplétude 3.5.8 page 78, on en obtient un avatar plus proche de la formulation originale.

Corollaire 3.5.15 (premier théorème d'incomplétude de Gödel pour les théories Σ-cohérentes) *Soit* $\mathscr{T}$ *une théorie effectivement axiomatisable cohérente,* Σ*-complète et* Σ*-cohérente. Alors il existe une formule* Π_1 G *telle que :*

$$\nvdash_{\mathscr{T}} G \ \text{ et } \nvdash_{\mathscr{T}} \neg G\ .$$

On appelle *énoncé indécidable*[2] dans $\mathcal{T}$ un énoncé G vérifiant $\nvdash_{\mathcal{T}} G$ et $\nvdash_{\mathcal{T}} \neg G$. Si G est un énoncé Π_1 qui est indécidable dans une théorie Σ-complète, alors G est forcément vrai. En effet sinon sa négation, étant Σ_1 et vraie, serait démontrable dans $\mathcal{T}$ par Σ-complétude. L'énoncé d'incomplétude 3.5.15 a donc facilement pour conséquence l'énoncé 3.5.8 dans le cas des théories Σ-cohérentes.

Exercice 23 (ω-cohérence) Une théorie T dans le langage de l'arithmétique est dite *ω-cohérente* s'il n'existe pas de formule à une variable libre F telle que $\vdash_T \exists x F$ et pour tout entier n, $\vdash_T \neg F[\underline{n}/x]$.

1. Montrer que si T est ω-cohérente, alors T est cohérente.
2. Montrer que si T est cohérente et étend R^-, toutes les formules Π_1 prouvables dans T sont vraies dans $\mathbb{N}$.
3. Montrer que si T est ω-cohérente et étend R^-, toutes les formules Π_2 (en particulier les formules Σ_1) prouvables dans T sont vraies dans $\mathbb{N}$, et donc que T est Σ_1-cohérente.
4. En déduire que si T étend R^- et est ω-cohérente, alors il existe une formule G telle que ni G, ni $\neg G$ ne sont démontrables dans T (l'énoncé original du théorème de Gödel était sous hypothèse d'ω-cohérence de la théorie).

3.6 Les théorèmes de Gödel-Rosser et de Church

L'objet de cette section est :
— d'une part d'obtenir, sous des hypothèses analogues à celles du théorème de Gödel, des résultats d'indécidabilité (au sens algorithmique), par exemple une théorie qui satisfait les mêmes hypothèses que celle du théorème d'incomplétude 3.5.15 est indécidable, c'est le corollaire 3.6.4 qui est à peu de choses près la première version historique du théorème de Church ;
— d'autre part, au prix d'une hypothèse supplémentaire sur les conséquences de la théorie, de remplacer l'hypothèse de Σ-cohérence du théorème d'incomplétude 3.5.15 par une hypothèse de simple cohérence. Ce résultat dû à Rosser, donne une version du théorème de Church plus simple, et plus facile à utiliser pour démontrer l'indécidabilité d'autres théories. En ce qui concerne le théorème d'incomplétude, il s'agit de l'étendre à des théories arithmétiques assez étranges, les théories cohérentes qui ne sont pas Σ-cohérentes : c'est le théorème de Gödel-Rosser.

3.6.1 Ensembles représentables dans une théorie

Définition 3.6.1 Nous dirons qu'un sous-ensemble E de $\mathbb{N}^p$ est *faiblement représentable* dans une théorie $\mathcal{T}$ quand il existe une formule F dont les variables libres sont parmi $x_1, \ldots, x_p$ telle que :

$$(n_1, \ldots, n_p) \in E \text{ ssi } \vdash_{\mathcal{T}} F[\underline{n_1}/x_1, \ldots, \underline{n_p}/x_p] \,.$$

Nous dirons que E est *fortement représentable* dans $\mathcal{T}$ quand il existe une formule F dont les variables libres sont parmi $x_1, \ldots, x_p$ telle que :

$$\begin{array}{l} \text{si } (n_1, \ldots, n_p) \in E, \text{ alors } \vdash_{\mathcal{T}} F[\underline{n_1}/x_1, \ldots, \underline{n_p}/x_p] \\ \text{si } (n_1, \ldots, n_p) \notin E, \text{ alors } \vdash_{\mathcal{T}} \neg F[\underline{n_1}/x_1, \ldots, \underline{n_p}/x_p] \,. \end{array}$$

On dit que la formule F indiquée représente, faiblement dans le premier cas, fortement dans le second, l'ensemble E.

On remarque immédiatement que :

Fait 3.6.2 *Si E est fortement représentable par F dans une théorie $\mathcal{T}$* cohérente, *alors E est faiblement représentable par la même formule F.*

2. Ce sens de « indécidable » est tout à fait différent de son sens algorithmique, celui invoqué, par exemple, quand on parle de théorie indécidable à la section 3.4.1.

Dans une théorie cohérente, la faible représentabilité est donc une notion plus faible, mais la forte représentabilité a l'avantage de rester stable quand on renforce la théorie $\mathscr{T}$.

On avait pu jusqu'à présent ne se servir que de la la définissabilité dans $\mathbb{N}$ qui est une notion purement sémantique. Même s'il y a une analogie formelle, la notion de représentabilité est très différente puisqu'il s'agit d'une notion syntaxique. En particulier dans une théorie effectivement axiomatisable (le cas de toutes les théories qui nous intéressent ici), la démontrabilité est semi-décidable quelle que soit la complexité des formules en jeu, contrairement à la définissabilité.

3.6.2 Ensembles semi-décidables

Dans une théorie $\mathscr{T}$ Σ-complète, si F est une formule Σ_1 à une variable libre

$$\{n \in \mathbb{N} \mid \mathbb{N} \models F[n]\} \subset \{n \in \mathbb{N} \mid \vdash_{\mathscr{T}} F[\underline{n}]\} .$$

On ne peut espérer l'égalité sans supposer la cohérence de la théorie, mais cette hypothèse ne suffit pas, il faut la Σ-cohérence.

Proposition 3.6.3 *Soit $\mathscr{T}$ une théorie Σ-complète et Σ-cohérente, alors tout ensemble $A \subset \mathbb{N}^p$ semi-décidable est faiblement représentable dans la théorie $\mathscr{T}$ par une formule Σ_1, c'est-à-dire qu'il existe une formule F à p variables libres $\overline{x}$ qui est Σ_1 telle que :*

$$\overline{n} \in A \ \ ssi \ \ \vdash_T F[\overline{\underline{n}}/\overline{x}] .$$

Démonstration. Dans une telle théorie $\mathbb{N} \models F[\overline{n}/\overline{x}]$ si et seulement si $\vdash_T F[\overline{\underline{n}}/\overline{x}]$, d'où le résultat par la définissabilité des sesmi-décidables (proposition 3.1.15 iii page 62). ■

On obtient comme corollaire un premier résultat d'indécidabilité, que l'on va renforcer à la section suivante.

Corollaire 3.6.4 *Une théorie $\mathscr{T}$ Σ-complète et Σ-cohérente est indécidable.*

Démonstration. Si $\mathscr{T}$ était décidable, en prenant une formule qui représente faiblement l'ensemble semi-décidable K du problème de l'arrêt, on pourrait décider de l'appartenance à K. ■

Exercice 24 Montrer que si la théorie $\mathscr{T}$ est cohérente mais pas Σ-cohérente, alors il existe une formule F à une variable libre Σ_1 pour laquelle :

$$\{n \in \mathbb{N} \mid \mathbb{N} \models F[n]\} \subsetneq \{n \in \mathbb{N} \mid \vdash_{\mathscr{T}} F[\underline{n}]\} .$$

3.6.3 Ensembles décidables

Dans une théorie arithmétique effectivement axiomatisable cohérente, l'ensemble des théorèmes, ainsi que l'ensemble des énoncés dont la négation est un théorème sont tous les deux semi-décidables. Comme l'opération de substitution se code de façon récursive primitive, un ensemble représentable est semi-décidable et de complémentaire semi-décidable, donc décidable par la proposition 2.3.6 page 39.

Fait 3.6.5 *Si un sous-ensemble E de $\mathbb{N}^p$ est fortement représentable dans une théorie arithmétique effectivement axiomatisable cohérente, alors il est décidable.*

La réciproque est vraie pour une théorie qui étend la théorie R de Robinson (section 3.5.5 page 80).

Proposition 3.6.6 *Un sous-ensemble décidable de $\mathbb{N}^p$ est fortement représentable par une formule Σ_1 dans une théorie cohérente qui étend* R.

La proposition est conséquence du lemme suivant.

Lemme 3.6.7 (Astuce de Rosser) *Soit $A \subset N^p$, on note A^c son complémentaire dans $\mathbb{N}^p$. On suppose que :*

— *A est définissable par une formule Σ_1, soit $\exists y\, F[x, x_1, \ldots, x_p]$ où F est Σ_0,*
— *A^c est définissable par une formule Σ_1, soit $\exists y\, G[y, x_1, \ldots, x_p]$ où G est Σ_0.*

Alors A est fortement représentable dans toute théorie cohérente qui étend R *par la formule Σ_1 :*

$$\exists y \big(F[y, x_1, \ldots, x_p] \wedge \forall z < y \neg G[z, x_1, \ldots, x_p)\big) \ .$$

Démonstration (proposition). Si A est décidable, il est semi-décidable et de complémentaire semi-décidable par le fait 2.3.4. On a donc des formules F et G qui satisfont les hypothèses du lemme par les proposition3.1.15 (iii) et 3.1.16 page 62, et on conclut par le lemme 3.6.7. ■

Dans la preuve du lemme on note $H[y, x_1 \ldots, x_p]$ la formule Σ_0 $F[y, x_1, \ldots, x_p] \wedge \forall z < y \neg G[z, x_1, \ldots, x_p]$. Suivant une convention déjà utilisée, on note $\overline{x}$ pour $x_1, \ldots, x_p$. On a une intuition sur la formule $H[y, \overline{x}]$ en remarquant que sur $\mathbb{N}$, celle-ci exprime qu'un entier a tel que $H[\underline{a}, \overline{n}]$ vérifie

$$a = \mu y.(F[y, \overline{n}] \vee \neg G[y, \overline{n}]) \ .$$

C'est donc une trace de la preuve de la proposition 2.3.6 qu'un ensemble semi-décidable et de complémentaire semi-décidable est décidable.

Démonstration (lemme). Soit $\mathcal{T}$ une théorie arithmétique qui étend le système R de Robinson, en particulier elle est Σ-complète. Soit $\overline{n}$ un p-uplet d'entiers.

— Supposons $\overline{n} \in A$. Alors par hypothèse pour un certain entier m, $\mathbb{N} \models F[m, \underline{\overline{n}}]$, et pour tout $k \in \mathbb{N}$, $\mathbb{N} \models \neg G[\underline{k}, \underline{\overline{n}}]$, en particulier si $k < m$, c'est-à-dire que :

$$\mathbb{N} \models \ H[\underline{m}, \underline{\overline{n}}] \text{ d'où } \mathbb{N} \models \exists y\, H[y, \underline{\overline{n}}] \ .$$

Cette dernière formule étant Σ_1, par Σ-complétude, $\vdash_{\mathcal{T}} \exists y\, H[y, \underline{\overline{n}}]$.

— Supposons $\overline{n} \notin A$. On a

$$\neg H[y, \overline{x}] \equiv F[y, \overline{x}] \to \exists z < y\ G[z, \overline{x}] \ .$$

Par hypothèse on a un m tel que $\mathbb{N} \models G[m, \underline{\overline{n}}]$ et pour tout entier k $\mathbb{N} \models \neg F[k, \underline{\overline{n}}]$, donc par Σ-complétude

$$\vdash_{\mathcal{T}} G[\underline{m}, \underline{\overline{n}}] \ \text{ et pour tout entier } k \vdash_{\mathcal{T}} \neg F[\underline{k}, \underline{\overline{n}}] \ . \qquad (*)$$

Il s'agit de démontrer $\forall y \neg H[y, \underline{\overline{n}}]$. On distingue dans $\mathcal{T}$ suivant que $y \leq \underline{m}$ ou $\underline{m} < y$: c'est là où $R_{6,n}$ (seul schéma d'axiomes non Σ_1) intervient.

— $y \leq \underline{m} \vdash_{\mathcal{T}} \neg F[y, \underline{\overline{n}}]$ par $R_{5,n}$ et par $(*)$, donc :

$$y \leq \underline{m} \vdash_{\mathcal{T}} F[y, \underline{\overline{n}}] \to \exists z < y G[z, \underline{\overline{n}}] \ ;$$

— $\underline{m} < y \vdash \exists z < y G[z, \underline{\overline{n}}]$ par $(*)$ on a donc :

$$\underline{m} < y \vdash_{\mathcal{T}} F[y, \underline{\overline{n}}] \to \exists z < y G[z, \underline{\overline{n}}] \ .$$

On déduit de $R_{6,m}$ le résultat souhaité, à savoir que :

$$\vdash_{\mathcal{T}} \forall y \big(F[y, \underline{\overline{n}}] \to \exists z < y G[z, \underline{\overline{n}}]\big) \ .$$ ■

3.6.4 Indécidabilité

Théorème 3.6.8 (théorème d'indécidabilité de Church) *Soit $\mathcal{T}$ une théorie cohérente dans le langage de l'arithmétique $(0, s, +, \times, \leq)$ ayant pour conséquence la théorie R de Robinson, alors $\mathcal{T}$ est indécidable.*

Démonstration. On procède par l'absurde. La démonstration procède de façon analogue à celle du théorème de Gödel de la page 79, par le lemme de point fixe 3.5.9 sur les formules. Soit donc une fonction r récursive primitive telle que :

$$r(\ulcorner F \urcorner, n) = \ulcorner F[\underline{n}/x_1] \urcorner \ .$$

Si l'ensemble $\mathrm{Th}_{\mathcal{T}}$ des théorèmes de $\mathcal{T}$ est décidable, alors l'ensemble $\Delta = \{n \in \mathbb{N} \mid r(n, n) \notin \mathrm{Th}_{\mathcal{T}}\}$ est décidable et donc fortement représentable par une formule D à une variable libre x_1 :

— si $n \in \Delta$, alors $\vdash_{\mathcal{T}} D[\underline{n}/x_1]$;

— si $n \notin \Delta$, alors $\vdash_{\mathcal{T}} \neg D[\underline{n}/x_1]$.

On obtient une contradiction en prenant le code de D :

— $\ulcorner D \urcorner \in \Delta$ signifie que $r(\ulcorner D \urcorner, \ulcorner D \urcorner) \notin \mathrm{Th}_{\mathcal{T}}$, soit $\nvdash_{\mathcal{T}} D\left[\underline{\ulcorner D \urcorner}/x_1\right]$, contradiction ;

— $\ulcorner D \urcorner \notin \Delta$, signifie que $r(\ulcorner D \urcorner, \ulcorner D \urcorner) \in \mathrm{Th}_{\mathcal{T}}$, soit $\vdash_{\mathcal{T}} \neg D\left[\underline{\ulcorner D \urcorner}/x_1\right]$, ce qui contredit la cohérence de la théorie. ■

Par la proposition 3.4.2 page 73 on a le corollaire suivant.

Théorème 3.6.9 (théorème d'incomplétude de Gödel-Rosser) *Soit $\mathcal{T}$ une théorie effectivement axiomatisable cohérente ayant pour conséquence la théorie R de Robinson, alors $\mathcal{T}$ est incomplète, au sens où il existe une formule G telle que :*

$$\nvdash_{\mathcal{T}} G \quad et \quad \nvdash_{\mathcal{T}} \neg G\,.$$

3.6.5 Fonctions représentables

Souvent il suffit pour représenter une fonction de représenter son graphe. Clairement, le graphe d'une fonction totale calculable est un ensemble décidable, et une fonction totale dont le graphe est décidable est calculable. On peut décliner le fait 3.6.5 page 84 et la proposition 3.6.6 page 84 pour les graphes des fonction totales calculables. Mais on a parfois besoin d'une notion de représentabilité plus forte pour les fonctions.

Définition 3.6.10 (représentabilité des fonctions) Une fonction (totale) $f : \mathbb{N}^p \to \mathbb{N}$ est représentable dans une théorie $\mathcal{T}$ par une formule H à au plus $p+1$ variables libres $x_1, \ldots, x_p, y$ quand pour tout p-uplet $(n_1, \ldots, n_p)$:

$$\vdash_{\mathcal{T}} \forall y \left(H[\underline{n_1}/x_1, \ldots, \underline{n_p}/x_p] \leftrightarrow y = \underline{f(n_1, \ldots, n_p)} \right).$$

Une fonction représentable dans une théorie $\mathcal{T}$ est une fonction telle qu'il existe une telle formule H.

On a clairement la proposition suivante.

Proposition 3.6.11 *Si f est une fonction totale représentable par une formule H dans une théorie $\mathcal{T}$ qui a pour conséquence les axiomes R_1 (page 79), alors*

— *son graphe est fortement représentable dans $\mathcal{T}$ par la même formule H ;*

— *si $\mathcal{T}$ est de plus effectivement axiomatisable et cohérente, alors f est calculable.*

Une réciproque s'obtient avec les mêmes hypothèses que pour la proposition 3.6.6 page 84.

Proposition 3.6.12 (représentation des fonctions calculables) *Une fonction totale calculable est représentable par une formule Σ_1 dans une théorie cohérente qui étend R.*

La démonstration est proposée à l'exercice suivant. on pourrait simplifier s'il s'agissait seulement d'obtenir une formule Σ.

Exercice 25 (Représentation des fonctions calculables) Soit f une fonction totale calculable. On suppose que G_f, le graphe de la fonction *totale* $f : \mathbb{N}^p \longrightarrow \mathbb{N}$ est fortement représenté dans R par la formule $\exists z\, A[x_1, \ldots, x_p, y, z]$ où A est Σ_0, on écrira dans la suite $A[\overline{x}, y, z]$. Posons $A'[\overline{x}, y, z] \equiv_d \exists t \leq z (A[\overline{x}, y, t] \wedge y \leq z)$.

1. Montrer que A' est une formule Σ_0 qui représente G_f (**a** et **b**) :

1.a. si $m \neq f(\overline{n})$ alors $\vdash_R \neg \exists z\, A'[\underline{\overline{n}}, \underline{m}, z]$;

1.b. si $m = f(\overline{n})$ alors $\vdash_R \exists z\, A'[\underline{\overline{n}}, \underline{m}, z]$

1.c. et vérifie de plus $\forall \overline{x}, y, z\ (A'[\overline{x}, y, z] \to y \leq z)$.

2. Montrer que la formule $H[\overline{x}, y] := \exists z H_0[\overline{x}, y, z]$ avec :

$$H_0[\overline{x}, y, z] := A'[\overline{x}, y, z] \wedge \forall z' \leq z \forall y' \leq z' (A'[\overline{x}, y', z'] \to y' = y)$$

est Σ_1 et représente la fonction f dans R au sens suivant :

$$\vdash_R \forall y \left(H[\underline{\overline{n}}, y] \leftrightarrow y = \underline{f(\overline{n})} \right) \qquad (*)$$

Pour tout p-uplet d'entiers $\overline{n}$, on montrera successivement :

2.a. $\vdash_R H[\underline{\overline{n}}, \underline{f(n)}]$;

2.b. Il existe un entier d tel que $\vdash_R H_0[\underline{\overline{n}}, \underline{f(n)}, \underline{d}]$;

2.c. $\vdash_R \forall y(H[\underline{\overline{n}}, y] \to y = \underline{f(\overline{n})})$ (on utilisera le **b**, $\vdash_R \forall z(z \leq \underline{d} \vee \underline{d} \leq z)$ et la question 1).

On obtient ainsi un lemme de point fixe pour la démontrabilité.

Lemme 3.6.13 (point fixe pour les formules) *Soit T une théorie qui étend la théorie R de Robinson. Soit A une formule à une seule variable libre x_0. Alors il existe un énoncé G tel que :*

$$\vdash_{\mathscr{T}} G \leftrightarrow A[\ulcorner G \urcorner / x_0] \ .$$

De plus si A est une formule Π_1, G peut être choisie Π_1.

Démonstration. On reprend la démonstration du lemme de point fixe 3.5.9 page 78, avec les mêmes notations, mais, au lieu d'un formule qui définit le graphe de la fonction récursive primitive r (qui code la substitution), on choisit une formule Σ_1 $\underline{x_0 = r(x, y)}$ qui représente cette fonction dans $\mathscr{T}$. On obtient de façon analogue que :

$$\vdash_{\mathscr{T}} G \to A\left[\underline{\ulcorner G \urcorner} / x_0\right] \ .$$

Pour la réciproque, la condition de représentabilité assure que

$$\vdash_{\mathscr{T}} \underline{x_0 = r(\ulcorner D \urcorner, \ulcorner D \urcorner)} \to x_0 = \underline{r(\ulcorner D \urcorner, \ulcorner D \urcorner)}$$

donc, comme $G \equiv_d D[\underline{\ulcorner D \urcorner} / x_1]$, $\ulcorner G \urcorner = r(\ulcorner D \urcorner, \ulcorner D \urcorner)$, et $D[x_1] \equiv_d \forall x_0(\underline{x_0 = r(x_1, x_1)} \to A[x_0])$ on a bien :

$$\vdash_{\mathscr{T}} G \to A[\underline{\ulcorner G \urcorner} / x_0] \ .$$

■

3.7 Le second théorème d'incomplétude

Dans l'article où il démontre le premier théorème d'incomplétude, Gödel énonce également son second théorème d'incomplétude : la cohérence d'une théorie arithmétique effectivement axiomatisable n'est pas démontrable dans cette théorie même, sauf si elle est incohérente.

Une fois que l'on a un prédicat de prouvabilité, la cohérente de la théorie $\mathscr{T}$ s'exprime facilement : c'est le fait que l'absurde ($\bot$) n'est pas prouvable, soit $\neg \mathrm{Dem}_T[\ulcorner \bot \urcorner / x_0]$, que l'on note aussi Coh_T.

Gödel donne juste une esquisse de démonstration. En effet celle-ci peut-être vue comme une formalisation de sa démonstration du premier théorème d'incomplétude (essentiellement celle de la section 3.5.4 page 79), dans une théorie arithmétique suffisamment riche pour permettre cette formalisation. On va prendre l'arithmétique de Peano, mais une théorie arithmétique plus faible, obtenue par exemple en restreignant la classe des formules pour le schéma de récurrence, pourrait suffire.

On observe que la démonstration du premier théorème d'incomplétude donnée page 79 consiste essentiellement à montrer que si la formule G est démontrable dans la théorie $\mathscr{T}$, alors la théorie $\mathscr{T}$ est incohérente. Par contraposée on obtient que la cohérence de $\mathscr{T}$ entraîne que G n'est pas démontrable dans $\mathscr{T}$, en formalisant $\neg \mathrm{Dem}_T[\ulcorner G \urcorner / x_0]$, qui a pour conséquence G. En choisissant une théorie $\mathscr{T}$ dans laquelle on peut formaliser cette démonstration, on aura montré que $\vdash_{\mathscr{T}} \mathrm{Coh}_T \to G$, et comme G n'est pas démontrable dans $\mathscr{T}$, $\mathrm{Coh}_{\mathscr{T}}$ non plus.

Théorème 3.7.1 (second théorème d'incomplétude) *Soit $\mathscr{T}$ une théorie cohérente effectivement axiomatisable qui étend l'arithmétique de Peano. Alors la cohérence de la théorie $\mathscr{T}$ n'est pas démontrable dans $\mathscr{T}$:*

$$\nvdash_{\mathscr{T}} \mathrm{Coh}_T \ .$$

Démonstration. On détaille l'esquisse donnée juste au dessus en reprenant la démonstration de la page 79. On simplifie la notation en écrivant $\mathrm{Dem}_{\mathscr{T}}[\underline{n}]$ pour $\mathrm{Dem}_{\mathscr{T}}[\underline{n} / x_0]$. On part d'une formule G vérifiant (voir le lemme 3.6.13) :

$$\vdash_{\mathscr{T}} G \leftrightarrow \neg \mathrm{Dem}_{\mathscr{T}}\left[\underline{\ulcorner G \urcorner}\right]$$

Les conditions (1) et (2) de la page 79 deviennent en formalisant :

$$\vdash_{\mathscr{T}} \neg \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner G \urcorner}\right] \to G \tag{1}$$

$$\vdash_{\mathscr{T}} \mathrm{Dem}_T \left[G \to \neg \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner G \urcorner}]\right] . \tag{2}$$

La première partie de la démonstration de la page 79, celle qui aboutit à (3), se formalise en :

$$\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner G \urcorner}\right] \to \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner [\mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner G \urcorner}] \urcorner\right] . \tag{3}$$

La partie suivante, qui aboutit à (4), utilise (2) et le *modus ponens* (qu'il faut donc formaliser dans la théorie), et donne :

$$\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner G \urcorner}\right] \to \mathrm{Dem}_{\mathscr{T}} [\ulcorner \neg \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner G \urcorner}\right] \urcorner] . \tag{4}$$

La partie suivante, qui aboutit à l'incohérence, utilise (3), (4) et le *modus ponens* et donne :

$$\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner G \urcorner}\right] \to \neg \mathrm{Coh}_{\mathscr{T}} \tag{5}$$

soit par contraposée :

$$\vdash_{\mathscr{T}} \mathrm{Coh}_{\mathscr{T}} \to \neg \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner G \urcorner}\right]$$

et d'après (1) :

$$\vdash_{\mathscr{T}} \mathrm{Coh}_{\mathscr{T}} \to G .$$

On conclut comme déjà indiqué, par le premier théorème d'incomplétude. Ce schéma de démonstration permet ainsi d'isoler trois conditions portant sur le prédicat de prouvabilité qu'il faut démontrer pour la compléter.

i. Si $\vdash_{\mathscr{T}} A$, alors $\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \urcorner}]$;

ii. $\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \to B \urcorner}] \to (\mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \urcorner}] \to \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner B \urcorner}])$;

iii. $\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \urcorner}] \to \mathrm{Dem}_{\mathscr{T}} \left[\underline{\ulcorner \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \urcorner}] \urcorner}\right]$.

En effet, on a bien une formule G qui satisfait (1). Comme G satisfait aussi la réciproque de (1), par la condition **i** on obtient (2). La condition **iii** donne immédiatement (3). Le (4) s'obtient à partir de (2) par **ii**, qui est le *modus ponens* formalisé. Le (5) s'obtient à partir de (3) et (4) à nouveau par **ii** (rappelons que par définition $\neg A$ est $A \to \bot$).

Examinons ces trois conditions.

— la première **i** a en fait déjà été démontrée et utilisée pour le premier théorème d'incomplétude : c'est un cas particulier de la Σ-complétude. En effet le prédicat de prouvabilité est Σ_1, et la condition **i** peut se reformuler :

$$\text{si } \mathbb{N} \models \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \urcorner}], \text{ alors } \vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner A \urcorner}] .$$

— La seconde **ii** demande juste de formaliser la construction de la preuve par *modus ponens* de $A \to B$ à partir des deux preuves de $A \to B$ et de B : c'est une simple concaténation, et la démonstration se formalise sans difficultés.

— La troisième condition **iii** est la version formalisée de la première. On peut la déduire du résultat suivant :

$$\text{si } F \text{ est un énoncé } \Sigma_1, \text{ alors } \vdash_{\mathscr{T}} F \to \mathrm{Dem}_{\mathscr{T}} [\underline{\ulcorner F \urcorner}] .$$

Elle se démontre par induction. Dans le cas où la théorie $\mathscr{T}$ est l'arithmétique de Peano (ou une théorie qui étend la théorie Q et permet de faire « suffisament de récurrence »), la démonstration qu'il s'agit de formaliser est donnée à la section 3.5.5. Comme celle-ci n'utilise manifestement pas de récurrence au delà du premier ordre, il ne fait guère de doute qu'elle peut se formaliser dans l'arithmétique de Peano, même si les détails peuvent s'avérer pénibles. Que faut-il formaliser exactement ?

— Les axiomes de Peano ont pour conséquence la théorie R^-, soit la démonstration de la proposition 3.5.12 (les axiomes de R à R_5) : ce sont des récurrences portant sur des formules Σ_1 qui ne posent aucune difficulté de formalisation ;

— la théorie R^- est Σ-complète (proposition 3.5.11) : c'est une démonstration par induction sur la structure d'une formule Σ, donc faisant intervenir des formules avec variables libres, ce qui demande de formaliser la substitution, ou une notion d'environnement (affectation d'entiers aux variables libres) ; la récurrence se fait sur une formule universellement quantifiée (toute susbtitution d'entiers aux variables libres) ; cette démonstration serait la plus délicate à formaliser en toute précision [3]. ■

On a démontré que pour une formule G équivalente dans $\mathscr{T}$ à sa non démontrabilité dans $\mathscr{T}$, $\vdash_{\mathscr{T}} coh_T \to G$. On a en fait l'équivalence.

Proposition 3.7.2 *Soit $\mathscr{T}$ une théorie que vérifie les hypothèses du second théorème d'incomplétude de Gödel,et G une formule vérifiant*

$$\vdash_{\mathscr{T}} G \leftrightarrow \neg \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner G \urcorner\right]$$

dont on sait que $\mathbb{N} \models G$ et $\nvdash G$ (premier théorème d'incomplétude, voir section 3.5.4 page 79). Alors :

$$\vdash_{\mathscr{T}} \mathrm{Coh}_{\mathscr{T}} \leftrightarrow G\,.$$

Démonstration. Il reste à démontrer que $\vdash_{\mathscr{T}} \to \mathrm{Coh}_{\mathscr{T}}$, ce qui se déduit de

$$G \to \neg \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner G \urcorner\right] \qquad (*)$$

en suivant l'intuition que s'il existe une formule qui n'est pas démontrable, c'est que la théorie est cohérente. On part donc de

$$\vdash \bot \to G$$

donc par la première condition de Bernays :

$$\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner \bot \to G \urcorner\right]$$

donc par la seconde (formalisation du Modus Ponens) :

$$\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner \bot \urcorner\right] \to \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner G \urcorner\right]$$

d'où le résultat en prenant la contraposée et $(*)$. ■

Une conséquence immédiate du second théorème d'incomplétude est qu'il existe des théories arithmétiques cohérentes dans lesquelles on peut démontrer qu'elles sont elles-mêmes incohérentes (remarquez que si on a une théorie suffisante pour le second théorème d'incomplétude dans laquelle on démontre au contraire qu'elle est elle-même cohérente, alors elle est forcément incohérente).

Il suffit de prendre une théorie cohérente $\mathscr{T}$ suffisante pour le second théorème d'incomplétude, la théorie $\mathscr{T} + \neg \mathrm{Coh}_{\mathscr{T}}$, qui est cohérente par le second théorème d'incomplétude, est une telle théorie.

On a vu qu'une autre façon de formuler le second théorème d'incomplétude est d'affirmer que si une théorie satisfait les hypothèses suffisantes est qu'elle permet de démontrer sa propre cohérence, alors elle est incohérente. On reformule cet énoncé.

Second théorème d'incomplétude. Soit une théorie $\mathscr{T}$ effectivement axiomatisable et qui étend l'arithmétique de Peano, alors :

$$\text{si } \vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner \bot \urcorner\right] \to \bot \text{ alors } \vdash_{\mathscr{T}} \bot\,.$$

Cette proposition se généralise à une formule close quelconque.

Proposition 3.7.3 (Théorème de Löb) *Soit $\mathscr{T}$ une théorie cohérente effectivement axiomatisable qui étend l'arithmétique de Peano. Alors pour tout énoncé E :*

$$\textit{si } \vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}} \left[\ulcorner E \urcorner\right] \to E, \textit{ alors } \vdash_{\mathscr{T}} E\,.$$

3. Voir GIRARD 1987 pour une formalisation assez détaillée d'une partie de cette démonstration.

Démonstration. On remarque d'abord que $\mathrm{Dem}_{\mathscr{T}}[\ulcorner A \to B \urcorner] \equiv_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}+A}[\ulcorner B \urcorner]$. Sans rentrer dans les détails de codage, il s'agit pour le sens direct d'ajouter A aux axiomes et d'ajouter à la preuve de $A \to B$ une application du Modus Ponens, et pour la réciproque de formaliser le lemme de déduction dans un système à la Hilbert.

Il suffit ensuite d'appliquer à la théorie $\mathscr{T} + \neg E$, le second théorème d'incomplétude sous la forme que l'on vient d'énoncer.

De $\vdash_{\mathscr{T}} \mathrm{Dem}_{\mathscr{T}}[\ulcorner E \urcorner] \to E$, on déduit $\vdash_{\mathscr{T}+\neg E} \neg\mathrm{Dem}_{\mathscr{T}}[\ulcorner E \urcorner]$, donc $\vdash_{\mathscr{T}+\neg E} \neg\mathrm{Dem}_{\mathscr{T}+\neg E}[\ulcorner \bot \urcorner]$, d'où d'après le second théorème d'incomplétude $\vdash_{\mathscr{T}+\neg E} \bot$. Par l'absurde $\vdash_{\mathscr{T}} E$. ■

Chapitre 4

Problèmes de décision en logique

4.1 Résultats d'indécidabilité

Nous avons obtenu déjà obtenu une résultat d'indécidabilité, le théorème de Church 3.6.8 page 85), qui est un résultat d'indécidabilité pour certaines $\mathscr{L}$-théories arithmétiques cohérentes, le langage $\mathscr{L}$ étant celui de l'arithmétique : $(0, s, +, \times, \leq)$. Nous allons dans un premier temps étendre ces résultats en restant dans le même langage, en particulier au calcul des prédicats égalitaires pur. Nous verrons ensuite comment on peut étendre ce résultat à d'autres langages et à d'autres théories, en *interprétant* une théorie dans une autre.

Nous nous intéresserons ensuite à la satisfaction dans les modèles finis : on obtiendra des résultats d'indécidabilité directement à partir de l'indécidabilité du problème de l'arrêt d'une machine à registres. La méthode donne d'ailleurs une autre démonstration du théorème de Church.

4.1.1 Premières conséquences du théorème de Church

Théorie essentiellement indécidable

Le théorème de Church 3.6.8 énonce plus que l'indécidabilité de la théorie Q de Robinson, ou de l'arithmétique de Peano : toutes les extensions cohérentes de Q, ou de l'arithmétique de Peano, sont indécidables. Une telle théorie, qui est cohérente et dont toutes les extensions cohérentes sont indécidables, est dite *essentiellement indécidable*.

On exploite maintenant que Q est essentiellement indécidable et finiment axiomatisable. Deux théories cohérentes $\mathscr{T}$ et $\mathscr{T}'$ sont dites *compatibles* quand la théorie $\mathscr{T} \cup \mathscr{T}'$ est cohérente.

Théorème 4.1.1 *Soit $\mathscr{T}$ une théorie dans un langage dont la signature σ contient le langage de l'arithmétique $(0, s, +, \times, \leq)$ et qui est compatible avec la théorie* Q *de Robinson. Alors $\mathscr{T}$ est indécidable.*

Démonstration. La théorie $\mathscr{T}+\mathrm{Q}$, donc la théorie des énoncés clos du langage de signature $(0, s, +, \times, \leq)$ conséquences de $\mathscr{T}+\mathrm{Q}$ est cohérente, et c'est une extension de Q, elle est donc indécidable d'après le théorème de Church 3.6.8. Comme il est évidemment décidable de reconnaître si une formule est dans le langage $(0, s, +, \times, \leq)$, la théorie $\mathscr{T}+\mathrm{Q}$ est indécidable. Soit $\mathscr{A}_{\mathrm{Q}}$ des axiomes en nombre fini pour Q. Pour toute formule close F :

$$\vdash_{\mathscr{T}+\mathrm{Q}} F \text{ si et seulement si } \vdash_{\mathscr{T}} \bigwedge_{A \in \mathscr{A}_{\mathrm{Q}}} \rightarrow F .$$

La théorie $\mathscr{T}+\mathrm{Q}$ étant indécidable, la théorie $\mathscr{T}$ est indécidable. ■

La théorie sans axiomes, celle des énoncés universellement valides, est évidemment compatible avec Q. On a donc le corollaire suivant.

Corollaire 4.1.2 *L'ensemble des énoncés universellement valides d'un langage de signature contenant $(0, s, +, \times, \leq)$ est semi-décidable et indécidable.*

Le problème de la satisfaisabilité, qui est dual par complétude du calcul des prédicats, est donc indécidable et co-semi-décidable.

Si $\mathbb{N} \models \mathscr{T}$, alors $\mathscr{T}$ est compatible avec Q. Le corollaire suivant est donc un cas particulier de 4.1.1.

Corollaire 4.1.3 *Soit $\mathscr{T}$ une théorie du langage de signature $(0, \mathrm{s}, +, \times, \leq)$, telle que $\mathbb{N} \models \mathscr{T}$. Alors $\mathscr{T}$ est indécidable.*

On dit que qu'un tel modèle $\mathbb{N}$ est *fortement indécidable*. Un modèles *fortement indécidable* est un modèle $\mathscr{M}$ tel que pour toute théorie $\mathscr{T}$ satisfaite par $\mathscr{M}$, $\mathscr{T}$ est indécidable. Dit autrement la théorie de toutes les formules closes satisfaites par le modèle est indécidable, et toutes ses sous-théories sont indécidables.

On savait déjà que la théorie du modèle standard $\mathbb{N}$ (signature $(0, \mathrm{s}, +, \times, \leq)$) n'est pas décidable, ni même semi-décidable : d'après le théorème de Tarski 3.3.1 page 70 que cette théorie n'est pas même définissable dans l'arithmétique.

4.1.2 Méthode d'interprétation

La méthode d'interprétation de Tarski [1] ; permet d'étendre les résultats d'indécidabilité comme ceux de la section précédente à d'autres langages dans lesquels il est possible d'*interpréter* une théorie arithmétique essentiellement indécidable et finiment axiomatisable telle que la théorie Q. On en donne quelques éléments dans cette section.

On commence par un exemple très simple : il s'agit de montrer qu'on peut se passer de la relation d'ordre dans la signature pour les résultats d'indécidabilité.

Définition de l'ordre

Appelons σ-formule, une formule du langage de signature σ, σ-théorie une théorie du langage de signature σ.

Prenons la théorie Q_0 dans le langage $(0, \mathrm{s}, +, \times)$ axiomatisée par les axiomes Q_1 à Q_7 de la section 3.5.5 page 80 (c'est la théorie Q de Robinson d'origine). La relation d'ordre se définit dans Q_0 par $\exists z\ y = z + y$. Précisons en quel sens.

On associe à toute $(0, \mathrm{s}, +, \times, \leq)$-formule F une $(0, \mathrm{s}, +, \times)$-formule $\langle F \rangle$ obtenue en remplaçant toutes les occurrences de formules atomiques $t_1 \leq t_2$ dans F par $\exists z\ t_2 = z + t_1$. Si $\mathscr{A}$ est un ensemble de formules, $\langle \mathscr{A} \rangle = \{\langle F \rangle \mid F \in \mathscr{A}\}$.

Une première remarque est que cette traduction et calculable : la définition de $\langle F \rangle$ se fait par induction, et $\ulcorner F \urcorner \mapsto \ulcorner \langle F \rangle \urcorner$ est une fonction calculable (elle est récursive primitive, en utilisant le lemme 3.2.4 page 68).

Lemme 4.1.4 *La traduction $F \mapsto \langle F \rangle$ qui précède est calculable.*

Les lemmes suivant montrent que cette traduction est correcte, en un sens à chaque fois précisé.

Lemme 4.1.5 *On suppose que la $(0, \mathrm{s}, +, \times, \leq)$-structure $\mathscr{M}$ vérifie $\mathscr{M} \models \forall x \forall y (x \leq y \leftrightarrow \exists z\ z + x = y)$. Soit $\mathscr{N}$ la $(0, \mathrm{s}, +, \times)$-structure obtenue à partir de $\mathscr{M}$ en oubliant l'interprétation de $\leq$. Alors pour toute formule $F[x_1, \ldots, x_p]$, tout p-uplet d'entiers $(a_1, \ldots, a_p)$:*

$$\mathscr{M} \models F[a_1, \ldots, a_p] \text{ si et seulement si } \mathscr{N} \models \langle F \rangle [a_1, \ldots, a_p] .$$

Démonstration. Immédiat par induction sur F. ∎

On a clairement $\vdash_{\mathrm{Q}} \forall x \forall y (x \leq y \leftrightarrow \exists z\ z + x = y)$ (par Q_8 et Q_9), et donc le lemme suivant par induction F.

Lemme 4.1.6 *Pour toute $(0, \mathrm{s}, +, \times, \leq)$-formule F*

$$\vdash_{\mathrm{Q}} F \leftrightarrow \langle F \rangle .$$

1. voir TARSKI, MOSTOWSKI et Raphael Mitchel ROBINSON 1953.

On peut remarquer aussi que $\langle \mathrm{Q} \rangle \equiv \mathrm{Q}_0$.

Lemme 4.1.7 *Pour toute* $(0, \mathrm{s}, +, \times, \leq)$*-formule* F*, et toute théorie* $\mathscr{T}$ *qui étend* Q *:*

$$\vdash_{\mathscr{T}} F \text{ ssi } \vdash_{\langle \mathscr{T} \rangle} \langle F \rangle .$$

Démonstration.

(⇒) On l'obtient Par induction sur la longueur de la preuve de $\vdash_{\mathscr{T}} F$, sachant que $\vdash_{\mathrm{Q}_0} \langle \mathrm{Q}_8 \rangle$ et $\vdash_{\mathrm{Q}_0} \langle \mathrm{Q}_9 \rangle$. C'est aussi très simple en utilisant la complétude et le lemme 4.1.5.

(⇐) D'après le lemme 4.1.6. ■

On a finalement :

Proposition 4.1.8

i. *La théorie* Q_0 *est finiment axiomatisable et essentiellement indécidable ;*

ii. *Toute* $(0, \mathrm{s}, +, \times)$*-théorie compatible avec* Q_0 *est indécidable ;*

iii. *le modèle* $\mathbb{N}$ *restreint à la signature* $(0, \mathrm{s}, +, \times)$ *est fortement indécidable.*

Démonstration. Il suffit de démontrer **(i)**, car **(ii)** et **(iii)** s'en déduisent facilement comme à la section 4.1.1.

Soit $\mathscr{T}_0$ une $(0, \mathrm{s}, +, \times)$-théorie cohérente qui étend Q_0. On considère la $(0, \mathrm{s}, +, \times, \leq)$-théorie $\mathscr{T}_0 + \mathrm{Q}$: comme $\mathscr{T}$ étend Q_0, $\langle \mathscr{T}_0 + \mathrm{Q} \rangle \equiv \mathscr{T}_0$. On déduit donc du lemme précédent que pour toute $(0, \mathrm{s}, +, \times)$-formule F_0 (donc telle que $\langle F_0 \rangle = F_0$) :

$$\vdash_{\mathscr{T}_0} F_0 \text{ si et seulement si } \vdash_{\mathscr{T}_0 + \mathrm{Q}} F_0 .$$

En particulier la théorie $\mathscr{T}_0 + \mathrm{Q}$ est cohérente. Par ailleurs comme elle étend Q, elle est indécidable. La prouvabilité restreinte aux $(0, \mathrm{s}, +, \times)$-formules est encore indécidable, car le cas général se ramène à celui-ci d'après le lemme 4.1.6, et car la traduction d'une $(0, \mathrm{s}, +, \times, \leq)$-formule $F \mapsto \langle F \rangle$ est calculable (lemmme 4.1.4). Mais alors la théorie $\mathscr{T}_0$ est également indécidable par l'équivalence ci-dessus. ■

On retrouve de même les autres résultats de la section précédente, par exemple l'ensemble des formules universellement valides d'un langage dont la signature contient $(0, \mathrm{s}, +, \times)$ (un symbole de constante, un symbole de fonction à un argument, deux symboles de fonction à deux arguments) est indécidable.

Cas général

On peut maintenant s'appuyer sur les résultats obtenus à la section précédente, pour la signature $(0, \mathrm{s}, +, \times)$ et la théorie Q_0. La méthode développée au paragraphe précédent ne peut être reprise telle quelle , car elle utilise fortement que la signature d'arrivée est une partie de la signature initiale, que c'est un symbole de prédicat qui a été éliminé, et que les modèles de la théorie d'arrivée peuvent avoir même support.

On veut pouvoir éliminer des symboles de constante, comme 0 qui est clairement définissable comme neutre de l'addition avec suffisamment d'axiomes, des symboles de fonction, comme le successseur qui est définissable à partir de l'addition et de 1 (neutre de la multiplication). On veut aussi pouvoir interpréter une théorie arithmétique dans la théorie des ensembles, par exemple, avec une signature complètement différente, et une théorie qui ne parle pas que d'entiers.

Il s'agit, pour une certaine signature σ, de disposer :

— de formules qui permettent d'interpréter l'arithmétique dans le langage de signature σ, soit :
  — une σ-formule à une variable libre qui permet de définir dans un modèle quelconque les objets qui seront pris pour représenter les entiers ;
  — de σ-formules pour définir chaque symbole de la signature utilisée pour l'arithmétique.

  On dispose ainsi d'une traduction calculable d'une formule arithmétique en une formule sur la signature σ.
— d'axiomes pour assurer que l'interprétation est correcte : démontrer une formule arithmétique équivaut à démontrer sa traduction.

Cela ne coûte rien pour la définition de prendre un langage de signature *finie* quelconque, soit σ_0, pour le langage de l'arithmétique. On suppose aussi que la signature σ est finie.

On suppose qu'il existe :

— des formules pour traduire les σ_0-formules en des σ-formules, soit :
  — une σ-formule $N[x]$ à une variable libre (qui permet de définir dans un σ-modèle quelconque les objets qui seront pris pour représenter les éléments d'un σ_0-modèle donné) ;
  — pour chaque symbole ζ de σ_0 d'une σ-formules ϕ_ζ :
    c un symbole de constante, $\phi_c[x]$ a une seule variable libre ;
    f un symbole de fonction d'arité k, $\phi_f[y, x_1, \dots, x_k]$ a $k+1$ variables libres ;
    R un symbole de prédicat d'arité k, $\phi_R[x_1, \dots, x_k]$ a k variables libres.
— d'un ensemble d'axiomes $\mathscr{D}$, un ou deux axiomes étant associés à chaque symbole de σ_0 :
  c symbole de constante, $\exists! x\, \phi_c[x]$, $\forall x(\phi_c[x] \to N[x])$;
  f symbole de fonction d'arité k, $\forall x_1 \dots x_k \exists! y \phi_f[y, x_1, \dots, x_k]$,
  $\forall y \forall x_1 \dots x_k \big(\phi_f[y, x_1, \dots, x_k] \to (N[y] \wedge N[x_1] \wedge \dots \wedge N[x_k])\big)$;
  R un symbole de prédicat d'arité k, $\forall x_1 \dots x_k \big(\phi_R[x_1, \dots, x_k] \to (N[x_1] \wedge \dots \wedge N[x_k]\big)$.

La traduction $F \mapsto \langle F \rangle$, des σ_0-formules vers les σ-formules, se définit par induction sur la structure de F, en définissant tout d'abord $t \mapsto \langle x = t \rangle$, où x est une variable quelconque, par induction sur la structure du σ_0-terme t.

— $\langle x = y \rangle$ est $x = y \wedge N[x] \wedge N[y]$ pour y une variable quelconque ;
— $\langle x = f\, t_1 \dots t_k \rangle$ est $\exists x_1 \dots x_k \big(\langle x_1 = t_1 \rangle \wedge \dots \wedge \langle x_k = t_k \rangle \wedge \phi_f[x, x_1, \dots, x_k]\big)$ pour f un symbole de fonction de σ_0 ;
— $\langle x = c \rangle$ est $\phi_c[x]$ pour c un symbole de constante de σ_0 ;
— $\langle t_1 = t_2 \rangle$ est $\exists z(\langle z = t_1 \rangle \wedge \langle z = t_2 \rangle$, où t_1 est un terme qui n'est pas une variable, z une variable « fraîche » (non encore utilisée, en particulier elle n'apparait ni dans t_1, ni dans t_2) ;
— $\langle R t_1 \dots t_k \rangle$ est $\exists x_1 \dots x_k \big(\langle x_1 = t_1 \rangle \wedge \dots \wedge \langle x_k = t_k \rangle \wedge \phi_R[x_1, \dots, x_k]\big)$ où R est un symbole de relation de σ_0 d'arité k ;
— $\langle \bot \rangle$ est $\bot$;
— $\langle (A \to B) \rangle$ est $(\langle A \rangle \to \langle B \rangle)$;
— $\langle \forall x\, A \rangle$ est $\forall x(N[x] \to \langle A \rangle)$.

Soit $\mathscr{M}$ une σ-structure telle que $\mathscr{M} \models \mathscr{D}$. On peut alors définir une σ_0-structure $\mathscr{M}_{\mathscr{D}}$ de support

$$| \mathscr{M}_{\mathscr{D}} | = \{a \in | \mathscr{M} | \, | \, \mathscr{M} \models \mathbb{N}[a]\}$$

et où les symboles de la signature σ_0 sont interprétés en suivant $\mathscr{D}$, plus précisément :

— Si c est un symbole de constante, il est interprété par l'unique m de $\mathscr{M}$ tel que $\mathscr{M} \models \phi_c[m]$ (unique et dans $| \mathscr{M}_{\mathscr{D}} |$ car $\mathscr{M} \models \mathscr{D}$:
— si f est un symbole de fonction d'arité k ; il est interprété par la fonction de $| \mathscr{M}_{\mathscr{D}} |^k \to | \mathscr{M}_{\mathscr{D}} |$, qui à $m_1, \dots, m_k \in | \mathscr{M}_{\mathscr{D}} |$ associe l'unique n tel que $\mathscr{M} \models \Phi_f[n, m_1, \dots, m_k]$ (unique et dans $| \mathscr{M}_{\mathscr{D}} |$ par $\mathscr{D}$) ;
— si R est un symbole de relation, R est interprétée par Φ_R.

On dit que la σ_0-structure $\mathscr{M}_{\mathscr{D}}$ obtenue (ou une structure isomorphe) est *définissable* dans la σ-structure $\mathscr{M}$.

Par exemple, en prenant pour σ_0 la signature du langage de l'arithmétique de Peano (sans la relation d'ordre), ($\sigma_0 = (0, \mathrm{s}, +, \times)$, et $\sigma = (\in)$ (un seul symbole de prédicat binaire), on a bien des formules de la théorie des ensembles pour $N[x]$ qui définit « x est un entier » (un ordinal fini), des formules pour définir 0 et s, et par le théorème de définition par récurrence sur les ordinaux, des formules pour définir + et ×, et les axiomes $\mathscr{D}$ correspondant se démontrent bien dans la théorie de Zermelo-Fraenkel ZF, sans l'axiome de l'infini (ni non plus l'axiome du choix et l'axiome de fondation).

0n peut donc définir dans tout modèles de ZF (éventuellement sans l'axiome de l'infini) une σ_0-structure $\mathscr{M}_{\mathscr{D}}$ dont on vérifie facilement qu'elle satisfait les axiomes de Peano (voir un livre d'introduction à la théorie des ensembles).

On se restreint dans la suite à des langages de signature finie. Clairement la traduction : $F \to \langle F \rangle$ est calculable, et cela peut se formaliser en utilisant les méthodes développées à la section 3.2. La traduction est même récursive primitive en utilisant le lemme 3.2.4.

Lemme 4.1.9 *La traduction $F \mapsto \langle F \rangle$ des σ_0-formules vers les σ-formules est calculable.*

Lemme 4.1.10 *On suppose que la σ-structure $\mathscr{M}$ satisfait $\mathscr{D}$, et que $\mathscr{M}_{\mathscr{D}}$ est la σ_0-structure définie dans $\mathscr{M}$ comme ci-dessus. Alors pour toute formule $F[x_1,\dots,x_p]$ à au plus p variables libres, pour tous $a_1,\dots,a_p \in \mathscr{M}_{\mathscr{D}}$:*

$$\mathscr{M}_{\mathscr{D}} \models F[a_1,\dots,a_p] \text{ si et seulement si } \mathscr{M} \models \langle F\rangle[a_1,\dots,a_p]\ .$$

Démonstration. La démonstration se fait d'abord pour les formules atomiques du type $x = t$ par induction sur la structure du terme t. À chaque étape les axiomes associés au symbole de σ_0 considéré donnent le résultat. Le résultat s'en déduit pour une formule atomique égalitaire quelconque, pour une éventuelle formule atomique non égalitaire en utilisant $\mathscr{D}$, puis pour une formule F quelconque, par une induction dont chaque étape est évidente. ■

Lemme 4.1.11 *Soit $\mathscr{A}$ un ensemble d'axiomes pour une σ_0-théorie, $\mathscr{D}$ les axiomes associés à une signature σ qui définissent la signature σ_0 dans σ. On note $\langle\mathscr{A}\rangle$ l'ensemble des traductions des axiomes de $\mathscr{A}$. Alors pour toute formule F :*

$$\text{si } \mathscr{A} \vdash F \text{ alors } \langle\mathscr{A}\rangle + \mathscr{D} \vdash \langle F\rangle$$

De plus si $\mathscr{A}$ est fini, et si la signature σ_0 est finie, alors $\langle\mathscr{A}\rangle + \mathscr{D}$ est fini.

Démonstration. On utilise le théorème de complétude et le lemme 4.1.10 : si tout σ_0-modèle de $\mathscr{A}$ est modèle de F, c'est le cas en particulier de tous les σ_0-modèles $\mathscr{M}_{\mathscr{D}}$ associés à un σ-modèle $\mathscr{M}$ de $\mathscr{D}$. Si de plus $\mathscr{M} \models \langle\mathscr{A}\rangle$, alors $\mathscr{M}_{\mathscr{D}} \models A$ d'après le lemme 4.1.10, et donc $\mathscr{M}_{\mathscr{D}} \models F$, et toujours d'après le même lemme 4.1.10, $\mathscr{M} \models \langle F\rangle$. ■

On ne peut pas espérer de réciproque pour n'importe quel jeu d'axiomes de définition $\mathscr{D}$ (comme on l'avait dans le cas particulier précédent). Mais pour l'indécidabilité il suffit de revenir à une théorie cohérente extension de Q_0. On définit donc pour une σ-théorie $\mathscr{T}$, la σ_0-théorie $\mathscr{T}^-$ des énoncés dont la traduction est démontrable dans $\mathscr{T}$, c'est-à-dire que pour toute formule close F :

$$F \in \mathscr{T}^- \text{ si et seulement si } \vdash_{\mathscr{T}} \langle F\rangle\ .$$

Si $\mathscr{T}$ étend $\mathscr{D}$, alors $\mathscr{T}^-$ est close déductivement.

Lemme 4.1.12 *On suppose que $\mathscr{T}$ étend $\mathscr{D}$, alors les trois propositions suivantes sont équivalentes :*

i. $F \in \mathscr{T}-$; **ii.** $\vdash_{\mathscr{T}^-} F$; **iii.** $\vdash_{\mathscr{T}} \langle F\rangle$.

Démonstration. Il suffit de démontrer **(ii)** ⇒ **(iii)**. Si $\vdash_{\mathscr{T}^-} F$, d'après le lemme 4.1.11 ; $\vdash_{\langle\mathscr{T}\rangle^-} \langle F\rangle$, mais par définition de $\mathscr{T}^-$ toutes les éléments de $\mathscr{T}^-$ sont démontrables dans $\mathscr{T}$, donc $\vdash_{\mathscr{T}} \langle F\rangle$. ■

Proposition 4.1.13 *Soit $\mathscr{A}$ un ensemble d'axiomes pour une σ_0-théorie cohérente essentiellement indécidable, $\mathscr{D}$ les axiomes associés à une signature σ qui définissent la signature σ_0 dans σ. On suppose que $\langle\mathscr{A}\rangle + \mathscr{D}$ est une σ-théorie cohérente. Alors :*

i. *la σ-théorie axiomatisée par $\langle\mathscr{A}\rangle + \mathscr{D}$ est essentiellement indécidable ;*

ii. *Si $\mathscr{A}$ est fini, et si la signature σ_0 est finie, alors la théorie $\langle\mathscr{A}\rangle + \mathscr{D}$ est finiment axiomatisable.*

Démonstration. Soit $\mathscr{T}$ une théorie σ-cohérente qui étend $\langle\mathscr{A}\rangle + \mathscr{D}$. Soit $\mathscr{T}^-$ la théorie associée comme au lemme précédent. La théorie $\mathscr{T}^-$ est cohérente car $\bot$ n'est pas changé par traduction, donc d'après le lemme précédent :

$$\vdash_{\mathscr{T}^-} \bot \text{ si et seulement si } \vdash_{\mathscr{T}} \bot$$

et $\mathscr{T}$ est cohérente. Par définition de $\mathscr{T}^-$, $\mathscr{T}^-$ étend $\mathscr{A}$, donc est indécidable. À nouveau par le lemme précédent, et en utilisant que la traduction est calculable (lemme 4.1.9 page précédente), la théorie $\mathscr{T}$ est indécidable. La théorie $\langle\mathscr{A}\rangle + \mathscr{D}$ est donc essentiellement indécidable. Elle est clairement finiment axiomatisée (par $\langle\mathscr{A}\rangle + \mathscr{D}$) si $\mathscr{A}$ et σ_0, donc $\mathscr{D}$, sont finis. ■

En reprenant l'exemple de la théorie des ensembles, et en interprétant la théorie Q_0 de Robinson, finiment axiomatisable, dans la théorie des ensembles, on obtient une théorie $\langle Q_0\rangle + \mathscr{D}$ dans le langage de l'appartenance, qui reste finiment axiomatisable, conséquence de la la théorie ZF sans l'axiome de l'infini, et donc cohérente si celle-ci est cohérente. D'après la proposition précédente l'ensemble des énoncés de l'arithmétique E tels que :

$$\langle Q_0\rangle + \mathscr{D} \vdash \langle E\rangle$$

Par conséquent

Proposition 4.1.14 *Les théorie suivantes, du langage de signature* $(\in)$ *sont indécidables :*

- *la théorie* ZF *sans axiome de l'infini (ni axiome du choix, ni axiome de fondation), supposée cohérente, est indécidable ainsi que toutes ses extensions cohérentes ;*
- *la prouvabilité en calcul des prédicats égalitaire pur, pour une signature qui comporte au moins un symbole de prédicat binaire, est indécidable ;*
- *il existe une sous-théorie finie de la théoire* ZF*, sans axiome de l'infini, ni axiome du choix, ni axiome de fondation, et supposée cohérente, qui est indécidable.*

Exercice 26 [2]

1. On suppose que σ_0 est une signature finie, que la σ_0-structure $\mathscr{N}$ est fortement indécidable.
Soit σ une signature, et $\mathscr{M}$ une σ-structure dans laquelle $\mathscr{N}$ est définissable (voir page 4.1.2). On nomme $\mathscr{D}$ les axiomes de définition associés, et la traduction est notée de la même façon que ci-dessus.
Montrer que $\mathscr{M}$ est fortement indécidable.

2. En déduire que

2.a. $(\mathbb{N}, +, \times, 0, 1)$ est fortement indécidable ;

2.b. $(\mathbb{N}, +, \times)$ est fortement indécidable ;

2.c. $(\mathbb{Z}, +, \times)$ est fortement indécidable (utiliser le théorème des quatre carrés de Lagrange) ;

2.d. les théories des semi-anneaux, des anneaux, des anneaux unitaires, des anneaux commutatifs, des anneaux intègres, le calcul des prédicats égalitaire pur avec au moins deux symboles de fonction binaire, sont indécidables.

3. Montrer que (avec $(\cdot)^2$ pour l'élévation au carré) :

3.a. $(\mathbb{N}, +, (\cdot)^2)$ est fortement indécidable ;

3.b. le calcul des prédicats égalitaire pur avec au moins un symbole de fonction binaire et un symbole de fonction unaire est indécidable.

2. Solution : voir CORI et LASCAR 1994b, chapitre 6, exercice 11.

4.2 Satisfaisabilité dans les modèles finis

Le problème auquel nous allons nous intéresser est le suivant.

SAT FINI PO
Entrée : Une signature finie σ, une formule close du premier-ordre φ sur une signature σ
Question : φ a-t-elle un modèle fini ?

Étant donnée une formule φ du premier ordre et une structure $\mathscr{S}$ finie, décider si $\mathscr{S} \models \varphi$ est clairement (primitif) récursif : il suffit de considérer chaque $\forall x$ comme une grande conjonction $\bigwedge_{a \in Dom(\mathscr{S})}$, chaque $\exists x$ comme $\bigvee_{a \in Dom(\mathscr{S})}$, de remplacer chaque x par sa valeur a et de tester si la formule sans quantificateur[3] ainsi obtenue est vraie dans $\mathscr{S}$. Ceci implique le résultat suivant.

Proposition 4.2.1 *Le problème* SAT FINI PO *est semi-décidable.*

Démonstration. Il suffit d'énumérer toutes les structures potentielles finies $\mathscr{S}$ et de vérifier si $\mathscr{S} \models \varphi$. ■

Nous allons démontrer que la satisfaction dans les modèles finis n'est pas en général décidable (comme pour la satisfaction dans les modèles quelconques, cela dépend de la signature) : c'est le théorème de Trakhtenbrot. De façon surprenante, la restriction aux structures finies amène à une situation duale du cas général :

— le problème SAT FINI PO est semi-décidable mais n'est pas co-semi-décidable ;
— le problème SAT PO est co-semi-décidable mais pas semi-décidable.

En effet la satisfaisabilité (cas général) d'une formule équivaut à la non prouvabilité de sa négation par le théorème de complétude. Dans le cas des modèles finis, on peut décrire le problème dual de la satisfaisabilité qui est celui de la validité dans tous les modèles finis.

VALID FINI PO
Entrée : Une signature finie σ, une formule close du premier-ordre φ sur une signature σ
Question : φ est-t-elle satisfaite dans tous les modèles finis ?

Par dualité avec la satisfaisabilité, le problème VALID FINI PO est co-semi-décidable mais pas semi-décidable. Il n'y a donc pas de notion de preuve satisfaisante, pour l'interprétation dans les modèles finis : une notion de preuve comme objet fini dont on peut vérifier mécaniquement qu'il est correct, conduit à une prouvabilité semi-décidable, et il ne peut donc y avoir complétude pour la satisfaction dans les modèles finis puisque le problème VALID FINI PO n'étant pas semi-décidable.

Théorème 4.2.2 (Trakhtenbrot) *Le problème* SAT PO FINI *est indécidable. En particulier,* SAT FINI PO *n'est pas co-semi-décidable.*

La démonstration que l'on va en donner illustre un principe de codage du calcul d'une machine par une interprétation logique.

Démonstration. Pour montrer ce résultat, on va procéder par réduction à partir du problème de l'arrêt des machines à registres initialisées à 0, voir le théorème 2.3.10 page 41. À peu de frais la démonstration fournit également une preuve du théorème de Church pour le langage considéré.

Soit donc une machine à registres M utilisant k registres $R_1, \ldots, R_k$ et dont le programme (programme goto) est la suite d'instructions $I_O, \ldots, I_l$. En particulier la dernière instruction I_l est l'instruction halt (voir section 1.3 page 18), et aucune des instructions I_n, $n < l$ n'est l'instruction halt. On suppose que tous les registres sont initialisés à 0.

On veut une signature σ_M, et une formule Φ_M sur σ_M, telle que :

$$M \text{ s'arrête} \iff \Phi_M \text{ a un modèle fini .} \qquad (*)$$

Pour modéliser le fonctionnement de M, il faut pouvoir parler des entiers contenus dans les registres, et du temps qui est également un entier (le nombre d'instructions). Comme la machine est initialisée à

3. Régler les détails éventuels en exercice

0, vu le jeu d'instructions des programmes goto, un entier x contenu dans un registre à l'instant t vérifie nécessairement $x \leq t$.

Ces entiers ont besoin d'être incrémentés, décrémentés, et comparés entre eux.

Le langage utilisé est celui des ordres discrets : $(0, \mathrm{s}, \mathrm{pred}, <)$, où 0 est un symbole de constante, $<$ un symbole de relation binaire qui sera interprété comme l'ordre naturel sur les entiers, et et s et pred comme les fonctions successeur et prédécesseur compatible avec $<$. On utilise l'abréviation usuelle $x \leq y$ pour $x < y \vee x = y$.

À chaque registre R_i est associé un prédicat binaire noté aussi R_i, et à chaque entier n, $0 \leq n \leq l$, un prédicat unaire noté I_n, comme la n-ième instruction I_n du programme. Ces prédicats sont interprétés intuitivement de la façon suivante :

— $R_i(x,t)$ signifie que le registre contient l'entier x a l'instant t ;
— $I_n(t)$ signifie que l'instruction I_n est lue à l'instant t ;
— $\exists t\ I_l(t)$ exprime l'arrêt de la machine.

La signature obtenue est $\sigma_M = (0, \mathrm{s}, \mathrm{pred}, <, R_1, \ldots, R_k, I_0, \ldots, I_l)$, et on va construire sur cette signature une formule Φ_M qui va décrire le comportement de la machine M au cours du calcul. Elle doit aussi axiomatiser l'ordre pour s'assurer de son interprétation. La formule Φ_M est la conjonction de quatre formules :

$$\textsc{Ordre}, \textsc{Init}, \textsc{Contrainte}, \textsc{Transition}_M.$$

— La formule Ordre axiomatise l'ordre, le successeur et le prédécesseur ;
— la formule Init code la configuration initiale de la machine ;
— la formule la formule Contrainte est un ensemble de clauses qui doivent être satisfaites par toute machine ;
— la formule $\textsc{Transition}_M$ code le fonctionnement de la machine M.

L'ordre : la formule Ordre est la conjonction des 4 formules suivantes :

— $O_1 : \forall x\ \neg(x < x) \wedge \forall y\ (x = y \vee x < y \vee y < x) \wedge \forall z (x < y \wedge y < z \rightarrow x < z)$
soit le prédicat $<$ est un ordre total strict ;
— $O_2 : \forall x\ (x = 0 \vee x > 0)$
soit la constante 0 est le plus petit élément pour $<$, on ne dit rien sur l'existence d'un plus grand élément éventuel ;
— $O_3 : \forall x\ \big(x \leq \mathrm{s}(x) \wedge (\exists y\ x < y \rightarrow x < \mathrm{s}(x)) \wedge \forall y (y \leq x \vee \mathrm{s}(x) \leq y)\big)$;
— $O_4 : \forall x \big(x < \mathrm{s}(x) \rightarrow\ \mathrm{pred}(\mathrm{s}(x)) = x\big) \wedge \mathrm{pred}(0) = 0$;
Les deux formules O_3 et O_4 définissent les fonctions pred et s comme attendu.

On déduit de O_1 et O_3 que :

— $O_5 : \forall x \big(x = \mathrm{s}\, x \leftrightarrow \forall y\ y \leq x\big)$

c'est-à-dire qu'un élément est égal à son successeur si et seulement si c'est le plus grand élément.

Les conditions initiales : la formule Init exprime que les registres sont vides à l'instant 0 et que c'est l'instruction I_1 qui est en lecture :

$$\textsc{Init} \equiv R_1(0,0) \wedge \cdots \wedge R_k(0,0) \wedge I_1(0)$$

Les contraintes des machines : dans cette partie, on exprime le fait que les prédicats R_i et I_n codent bien le comportement d'une machine à registres. La formule Contrainte est la conjonction des formules suivantes.

— Au plus une instruction est lue à l'instant t :

$$C_{1,i} \equiv \forall t \left(I_i(t) \rightarrow \bigwedge_{j \neq i} \neg I_j(t) \right),\ 1 \leq i \leq l\ ;$$

— à un instant t donné, un registre ne peut contenir qu'un seul entier :

$$C_{2,i} \equiv \forall t \forall x \forall y \big((R_i(x,t) \wedge R_i(y,t)) \rightarrow x = y \big),\ 0 \leq i \leq k\ .$$

Les transitions de la machine : cette partie est la seule dépendante de la machine M. À chaque instruction I_n on associe une formule qui décrit celle-ci :

— I_n est $R_i := R_i + 1$:

$$\forall t \left(I_n(t) \to [t < \mathrm{s}\, t \wedge I_{n+1}(\mathrm{s}\, t) \wedge \forall x (R_i(x,t) \to R_i(\mathrm{s}\, x, \mathrm{s}\, t) \wedge \bigwedge_{j \neq i} \forall x (R_j(x,t) \to R_j(x, \mathrm{s}\, t))] \right)$$

— I_n est $R_i := R_i - 1$:

$$\forall t \left(I_n(t) \to [t < \mathrm{s}\, t \wedge I_{n+1}(\mathrm{s}\, t) \wedge \forall x (R_i(x,t) \to R_i(\mathrm{pred}\, x, \mathrm{s}\, t) \wedge \bigwedge_{j \neq i} \forall x (R_j(x,t) \to R_j(x, \mathrm{s}\, t))] \right)$$

— I_n est if $R_i = 0$ goto p :

$$\forall t \left(I_n(t) \to \left[t < \mathrm{s}\, t \wedge (R_i(0,t) \to I_p(\mathrm{s}\, t)) \wedge \forall x \, (R_i(\mathrm{s}\, x, t) \to I_{n+1}(\mathrm{s}\, t)) \wedge \bigwedge_{j \neq i} t \forall x \left(R_j(x,t) \to R_j(x, \mathrm{s}\, t) \right) \right] \right)$$

— I_n est halt (alors $n = l$) :

$$\forall t \forall t' \left(I_l(t) \to t' \leq t \right) .$$

On appelle « partie standard » d'un modèle $\mathcal{V}$ de Φ_M, la partie de $\mathcal{V}$ constitué des interprétations des $\mathrm{s}^n\, 0$, $n \in \mathbb{N}$ dans $\mathcal{V}$, qui est encore l'interprétation de tous les termes clos dans $\mathcal{V}$ d'après l'axiome O_4, et cette partie standard définit alors une sous-structure modèle de Φ_M car tous les axiomes constituant Φ_M sont universels (ou équivalents à une formule universelle). Un modèle standard de Φ_M est un modèle réduit à sa partie standard. On observe facilement que les axiomes de Φ_M définissent entièrement l'interprétation sur la partie standard d'un modèle, c'est-à-dire que tous les modèles standards d'une machine donnée sont isomorphes.

Lemme 4.2.3 *Soit une machine à registres M, et la formule Φ_M définie ci-dessus. alors Φ_M possède un modèle standard $\mathcal{U}$:*

— *si la machine M s'arrête à l'étape θ, alors ce modèle standard $\mathcal{U}$ peut être défini de support l'ensemble des $\theta + 1$ premiers entiers $\{0, \ldots, \theta\}$, l'ordre étant interprété usuellement ;*

— *si la machine M ne s'arrête jamais, alors le modèle standard peut être défini de support $\mathbb{N}$, l'ordre étant interprété usuellement ;*

— *la partie standard de tout modèle de Φ_M définit une sous-structure isomorphe à $\mathcal{U}$, en particulier tous les modèles standards sont isomorphes.*

Démonstration (lemme). Si M s'arrête à l'étape θ, cela signifie que l'instruction I_l est en lecture à l'instant θ, sinon I_l n'est jamais en lecture. Si la machine s'arrêt, le modèle $\mathcal{U}$ est de support l'ensemble d'entiers $U = \{0, \ldots, \theta\}$, sinon de support $U = \mathbb{N}$.

— l'ordre est l'ordre usuel sur les entiers, on a $\mathrm{pred}_{\mathcal{U}}(0) = 0$ et, quand la machine s'arrête en θ, $\mathrm{s}_{\mathcal{U}}(\theta) = \theta$, sinon le successeur et le prédécesseur sont interprétés usuellement ;

— supposons qu'à l'instant t, l'instruction lue est I_{n_t}, et chaque registre R_i contient l'entier $x_{i,t}$, alors :

  — le prédicat interprétant I_{n_t} prend la valeur 1 en t, les prédicats interprétant I_n, $n \neq n_t$ prennent la valeur 0 en t ;

  — le prédicat interprétant R_i prend la valeur 1 en $(x_{i,t}, t)$, la valeur 0 en (x, t) quand $x \neq x_{i,t}$.

On a bien décrit une σ_M-structure $\mathcal{U}$, qui est un modèle standard de Φ_M satisfaisant les deux premières assertions du lemme.

Supposons maintenant que $\mathcal{V}$ est un modèle de Φ_M, sa partie standard est $\{\mathrm{s}^t_{\mathcal{V}}(0) \mid t \in \mathbb{N}\}$. Montrons que l'application : $t \mapsto \mathrm{s}^t_{\mathcal{V}}(0)$ définit un isomorphisme de $\mathcal{U}$ sur la partie standard de $\mathcal{V}$. Deux cas sont possibles, soit pour tout entier t, $\mathrm{s}^t_{\mathcal{V}}(0) < \mathrm{s}^{t+1}_{\mathcal{V}}(0)$, et alors la partie standard $\mathrm{Std}(\mathcal{V})$ de $\mathcal{V}$ est en bijection avec $\mathbb{N}$, soit pour un certain entier θ, on a a $\mathrm{s}^{\theta+1}_{\mathcal{V}}(0) = \mathrm{s}^{\theta}_{\mathcal{V}}(0)$, choisissons le plus petit tel θ, et alors la partie standard de $\mathcal{V}$ est finie de cardinal $\theta + 1$: $\mathrm{Std}(\mathcal{V}) = \left\{\mathrm{s}^t_{\mathcal{V}}(0) \mid 0 \leq t \leq \theta\right\}$. Les formules INIT, $\mathrm{TRANSITION}_M$ et CONTRAINTES assurent assurent que $\mathcal{V} \models I_n(\mathrm{s}^t_{\mathcal{V}}(0))$ si et seulement si l'instruction lue à l'instant t est I_n. Si $\mathrm{Std}(\mathcal{V})$ est finie, ces formules assurent que $\mathcal{V} \models I_l(\mathrm{s}^{\theta}_{\mathcal{V}}(0))$ et donc la machine s'arrête bien à l'étape θ, et $\mathcal{U}$ est de support $\{0, \ldots, \theta\}$. Ces formules assurent également que l'interprétation des R_i sur $\mathrm{Std}(\mathcal{V})$ est bien celle correspondant au fonctionnement de la machine, et on a bien un isomorphisme de $\mathcal{U}$ sur la partie standard de $\mathcal{V}$. ∎

Démonstration (théorème de Trakhtenbrot). On reprend maintenant la démonstration du théorème de Trakhtenbrot. On déduit du lemme que la machine M s'arrête si et seulement si Φ_M est satisfaisable par un modèle fini. En effet si M s'arrête, le modèle standard de Φ_M est fini. Si la machine ne s'arrête pas le modèle standard de Φ_M est infini et tout modèle de Φ_M contient une copie de ce modèle standard donc est infini.

Comme la construction de la formule Φ_M à partir de M est clairement effective (si on veut se ramener aux entiers, les codages sont ceux pratiqués dans les chapitres précédents, ou analogues), on conclut à l'indécidabilité de la satisfaction dans les modèles finis. ■

Considérons maintenant le problème suivant.

VALID PO
Entrée : Une signature σ, une formule close du premier-ordre φ du langage de signature σ
Question : φ est-elle valide ?

Corollaire 4.2.4 (Church) *Le problème* VALID PO *est indécidable.*

Démonstration. Soit la formule Ψ définie par $\Psi \equiv_{\text{def}} \exists t\, \forall t'\ t' \leq t$. D'après le lemme 4.2.3 :

— si la machine M initialisée à 0 s'arrête, tout modèle de Φ_M possède un élément égal à son successeur, donc maximal d'après 0_5, donc satisfait également Ψ ;
— si la machine M initialisée à 0 ne s'arrête pas, le modèle standard de la formule Φ_M, d'ensemble de base $\mathbb{N}$, vérifie $\forall t\, \exists t'\, t < t'$ soit $\neg\Psi$.

La formule $\Phi_M \wedge \neg\Psi$ a donc un modèle si et seulement si M ne s'arrête pas. Autrement dit, $\Phi_M \to \Psi$ est valide si et seulement si M s'arrête. ■

Dans la démonstration précédente la signature est une donnée du problème car le nombre de prédicats unaires I_i et de prédicats binaires R_j dépend de la machine qui est codée. Il est facile d'adapter la démonstration en les remplaçant par un prédicat binaire I (pour représenter les instructions) et un prédicat ternaire R (pour les registres) dont le premier élément est un index du registre. Pour qu'il y ait suffisamment de tels index, il faut que le temps de calcul soit supérieur au nombre de registres et au nombre d'instructions, et pour cela on montre facilement, en ajoutant en tête de programme suffisamment d'instructions sans effet, et en renumérotant les instructions goto, l'indécidabilité de l'arrêt pour des machines dont le temps de calcul est supérieur ou égal au nombre de registres et au nombre d'instructions.

On a donc le résultat suivant.

Proposition 4.2.5 *Le problème de la satisfaisabilité dans les modèles finis pour une signature fixée, qui comprend un symbole de constante, deux symboles de fonction unaire, deux symboles de prédicat binaire et un symbole de prédicat ternaire est indécidable.*

Par des méthodes analogues à celles utilisées dans le cas du théorème de Church,on peut montrer, que le résultat d'indécidabilité est conservé pour toute signature fixée contenant au moins un symbole de relation binaire.

Index

Bibliographie

COLSON, Loïc (21 juin 1991). “About primitive recursive algorithms”. In : *Theoretical Computer Science* 83.1, p. 57-69.

CORI, René et Daniel LASCAR (1994a). *Logique mathématique : Cours et exercices I. Calcul propostionnel, algèbres de Boole, calcul des prédicats*. 2e éd. Masson. 420 p.

— (1994b). *Logique mathématique : Cours et Exercices II. Fonctions récursives, théorème de Gödel, théorie des ensembles, théorie des modèles*. 2e éd. Masson. 380 p.

DEDEKIND, Richard (1888). *Was sind und was sollen die Zahlen ?* Brunswick : Vieweg.

GIRARD, Jean-Yves (1987). *Proof Theory and Logical Complexity*. Bibliopolis. 516 p.

GÖDEL, Kurt (1er déc. 1931). “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I”. In : *Monatshefte für Mathematik und Physik* 38.1, p. 173-198.

KOMARA, Ján et Paul J. VODA (1999). “Theorems of Péter and Parsons in Computer Programming”. In : *Computer Science Logic*. Sous la dir. de Georg GOTTLOB, Etienne GRANDJEAN et Katrin SEYR. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 204-223.

ODIFREDDI, P. (1999a). *Classical Recursion Theory, Volume II*. 1re éd. Studies in logic and the foundations of mathematics 143. North Holland.

— (1999b). *Classical Recursion Theory : The Theory of Functions and Sets of Natural Numbers*. 2e éd. Studies in logic and the foundations of mathematics 125. Elsevier. 669 p.

PEANO, Giuseppe (1889). *Arithmetices principia, nova methodo exposita*. Turin : Bocca.

PÉTER, Rózsa (1967). *Recursive functions*. Academic Press. 308 p.

ROBINSON, Raphael M. (oct. 1947). “Primitive recursive functions”. In : *Bulletin of the American Mathematical Society* 53.10, p. 925-942.

SHOENFIELD, Joseph Robert (1967). *Mathematical logic*. Association for Symbolic Logic. 364 p.

SMORYNSKI, Craig (1991). *Logical Number Theory I : An Introduction*. Universitext, Universitext Smorynski,C. :Logical Number Theory. Berlin Heidelberg : Springer-Verlag.

TARSKI, Alfred, Andrzej MOSTOWSKI et Raphael Mitchel ROBINSON (1953). *Undecidable Theories*. Elsevier. 110 p.

TURING, Alan (1936). “On Computable Numbers, with an Application to the Entscheidungsproblem”. In : *Proceedings of the London Mathematical Society* 42.1, p. 230-265.

Table des matières