Problèmes complets en analyse calculable

7 mai 2025

Résumé

In the first part of this project, we were generally introduced to computable analysis : The model of type 2 Turing machines to compute functions $\mathbb{R} \to \mathbb{R}$ and $\Sigma^\omega \to \Sigma^\omega$ for Σ a finite alphabet and functions. We also studied the importance of representation which led us to topological considerations. Indeed, we examined an algorithm that given a representation of a type 2 function, actually computes this function. Now that we know that common functions on reals are computable (polynomials, exponential but also the universal Turing machine), the question of existence of complete problem arises, and that is the subject of the second part of this project : Are there complete problems in computable analysis ?

To answer this question, we will rephrase some works of Vasco Brattka in[2]. In the first part of this document, we will introduce some notions about classical computability (reduction, completeness, relativization), and then their version in type 2 : in particular, we will present the Weihrauch reduction, which will lead us to complete problems, and the arithmetical hierarchy in type 2, to classify these problems.

In the third part we define a first problem, called C, which is Σ^0_2-complete. Problem C will even allow us to show that a more natural problem, the problem lim, which returns the limit of a converging sequence, is also Σ^0_2-complete. This is even more interesting because, compared to C, lim is a natural problem.

Table des matières

Introduction

Lors de la première partie de ce projet, nous avons été introduits à l'analyse calculable. Ici, nous nous concentrons sur la thématique suivante : trouver des problèmes complets. Ce document est presque entièrement nouveau par rapport à la première partie du projet. Il est cette fois-ci écrit en français et devrait être autosuffisant.

Les connaissances générales d'analyse calculable nécessaires à notre développement sont rappelées au début de la partie 2. La partie 1, que le lecteur averti pourra parcourir rapidement, présente les versions classiques des notions de type 2 présentées dans la suite. Ces notions doivent beaucoup au

livre [1] (degrés Turing, hiérarchie arithmétique). Les résultats du projet, à savoir la présentation de problèmes complets, constituent la troisième partie de ce document.

1 Calculabilité classique

Nous nous plaçons dans le cadre de la calculabilité classique. Sauf mention contraire, le symbole Σ dénote un alphabet fini quelconque contenant au moins deux éléments, et $*$ est l'étoile de Kleene. On note $f :\subseteq X \to Y$ lorsque f est une fonction partielle, de domaine $\mathrm{dom}(f) \subseteq X$. Une fonction $f : X \to Y$ sera supposée totale ($\mathrm{dom}(f) = X$).

1.1 Représentation, Décidabilité, Indécidabilité

Notation

On fait appel aux connaissances du lecteur pour définir la machine de Turing. On précise que cette dernière contient au moins une bande d'entrée, des bandes de travail et une bande de sortie. Une machine de Turing est donc définie par un code fini que l'on, sans perte de généralité, identifier à un entier $e \in \mathbb{N}$. On note Φ_e la machine de Turing de code e. Elle travaille sur un alphabet d'entrée Σ et un alphabet de sortie Σ'. L'entrée est un mot $x \in \Sigma^*$ écrit sur la bande d'entrée avec un symbole de Σ par cellule. De la même façon, si la machine s'arrête, sa sortie est un mot $y \in \Sigma'^*$ écrit sur sa bande de sortie.
nn On note $\Phi_e(x) \downarrow$ si la machine s'arrête sur l'entrée x (écrite sur la bande d'entrée), $\Phi_e(x) \uparrow$ si elle ne s'arrête pas, et $\Phi_e(x) \downarrow = y$ si elle s'arrête sur l'entrée x.
- Dans cette section, sans plus de précisions, une "fonction partielle" est définie sur $:\subseteq \mathbb{N} \to \mathbb{N}$, et une fonction totale est définie sur $\mathbb{N} \to \mathbb{N}$.

Définition 1.1 (fonction calculable). *Une fonction partielle $f :\subseteq \Sigma^* \to \Sigma'^*$ est dite calculable s'il existe un code $e \in \mathbb{N}$ d'une machine de Turing telle que*

$$\forall x \in \mathrm{dom}(f),\ \Phi_e(x) \downarrow = f(x).$$ ∘

Lorsque x n'appartient pas au domaine de f, on considère que Φ_e ne s'arrête pas ; (on peut le faire de manière calculable car c'est une propriété syntaxique) et on note $\perp$ la valeur de retour.

Si le domaine d'arrivée d'une fonction f, noté $\mathrm{range}(f)$, est $\{0, 1\}$, on parle alors de problème de décision, et on pourra dire que f est décidable pour dire qu'elle est calculable dans ce cas particulier.

Proposition 1.2. *Toute fonction $f :\subseteq \Sigma^* \to \Sigma'^*$ où $\mathrm{dom}(f)$ est fini est calculable.* ♢

Preuve. Prendre un code e contenant pour tout $x \in \mathrm{dom}(f)$ l'image $f(x)$, et renvoyer $f(x)$ quand l'entrée est x. Un tel code fini existe, puisqu'il y a autant de couples (antécédents, image) que d'antécédents : ces couples sont en nombres finis et chacun représentable avec une suite finie de symboles sur des alphabets finis. ■

Pour tout ensemble fini dénombrable X, il existe un alphabet fini Σ tel que $X = \Sigma^*$. Néanmoins, différents encodages peuvent permettre de travailler avec X de manière plus efficace, d'où l'intérêt de la notion suivante :

Définition 1.3 (Notation d'ensemble). *Une notation (ou fonction de décodage) d'un ensemble X est une fonction partielle* ***surjective*** *définie d'un langage sur un alphabet Σ^* vers X :*

$$\delta_X :\subseteq \Sigma^* \to X.$$ ∘

Théorème 1.4. *Soit $f : X \to Y$ avec des représentations $\delta_X : \Sigma^* \to X$ et $\delta_Y : \Sigma'^* \to Y$. Alors, f est calculable ssi il existe une fonction partielle $F :\subseteq \Sigma^* \to \Sigma'^*$ calculable vérifiant :*

$$\forall x \in \Sigma^*,\ \delta_Y F(x) = f\delta_X(x).$$

C'est-à-dire que le diagramme suivant est commutatif :

$$\begin{array}{ccc} \Sigma^* & \xrightarrow{F} & \Sigma'^* \\ \downarrow{\scriptstyle \delta_X} & & \downarrow{\scriptstyle \delta_Y} \\ X & \xrightarrow{f} & Y \end{array}$$

Preuve. ($\Rightarrow$) :
On choisit F la fonction calculée par une machine possédant le code suivant :

```
type sigma                          (* les symboles de l'alphabet Σ  *)
type sigma'                         (* les symboles de l'alphabet Σ' *)
val sigmastar : sigma list list     (* liste infinie des mots de Σ*  *)
val sigma'star : sigma' list list (* liste infinie des mots de Σ'* *)
val delta_X : sigma list -> X
val delta_Y : sigma list -> Y
val f : X -> Y

let realizer (u : sigma list) : sigma' list =
  List.find (fun v -> delta_X (f u) = delta_Y v) sigma'star
```

Donc, en utilisant le fait que les représentations sont totales, et le fait que nous pouvons énumérer les alphabets Σ^* et Σ'^*, on est assuré de finir par trouver pour tout $u \in \Sigma^*$ un mot $v \in \Sigma'^*$ tel que $F(u) = v$. Puisque f est calculable, la fonction `realizer` l'est aussi.

($\Leftarrow$) :
On fait la même chose pour construire f à partir de F et des représentations :

```
let f (x : X) : Y =
  let u = List.find (fun u -> delta_X u = x) sigmastar in
  delta_Y (F u)
```

Ce qui conclut la preuve. ∎

D'après le théorème 1.4 que nous venons de prouver, l'étude des fonctions calculables totales $f : X \to Y$ pour lesquelles il existe des notations calculables totales $\delta_X : \mathbb{N} \to X$ et $\delta_Y : \mathbb{N} \to Y$ se ramène à l'étude de la calculabilité des fonctions partielles $\subseteq \mathbb{N} \to \mathbb{N}$, sur lesquelles nous allons nous concentrer dans les pages suivantes.

Introduisons une fonction très célèbre (du moins en calculabilité), qui sera l'exemple canonique de fonction non-calculable.

Théorème 1.5 (Problème de l'arrêt). *Le problème de l'arrêt (ou simplement "l'arrêt"), la fonction totale définie par :*

$$\emptyset' : \left| \begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N} \\ n & \longmapsto & \begin{cases} 1 \text{ si } \Phi_n(n)\downarrow \\ 0 \text{ sinon} \end{cases} \end{array} \right.$$

est ***incalculable***. ♢

PREUVE. Montrons que l'existence d'une telle fonction mènerait à un paradoxe logique. Par l'absurde, supposons son existence. Définissons une fonction `diag`, calculable si la fonction `arret` l'est.

```
val arret : int -> bool (* renvoie vrai si pour l'entrée n, φ_n(n)↓,
    renvoie faux sinon. *)

let diag (e : int) =
  if arret e then (while true do () done; false)
  else true
```

On note `<diag>` le code de la fonction `diag`. Évaluons `diag(<diag>)` c'est à dire $\Phi_{<\text{diag}>}(<\text{diag}>)$. On distingue deux cas.

— Si $\emptyset'(<\text{diag}>)\downarrow = 1$, alors la fonction `diag` s'arrête quand elle prend son code en entrée. Mais par définition elle est supposée suivre la première branche du `if` et boucler à l'infini. On obtient une contradiction.

— Sinon, $\emptyset'(<\text{diag}>)\downarrow = 0$, donc `diag` ne s'arrête pas lorsqu'on lui donne son code en argument. Pourtant, l'exécution devrait terminer car on arrive dans la branche `else` qui termine. On obtient une contradiction aussi dans ce cas. ∎

1.2 Relativisation

Il y a donc des choses calculables, des choses incalculables et nous allons voir que le spectre est encore plus large que cela. Il y a des choses plus incalculables que d'autres : il y a différents degrés d'incalculabilité. Pour cela, il nous faut un outil conçut sur mesure pour une telle étude : la réduction. Mais d'abord, définissons la notion d'oracle.

Intuitivement, le calcul d'une fonction Φ_e avec un oracle g est un code e qui peut accéder, en plus de son jeu d'instruction, à une primitive spéciale permettant d'obtenir le résultat de l'oracle sur toute entrée, et ce même si ce dernier n'est pas calculable.

Notation (Calcul avec oracle)

On note $\Phi_e^g(x)$ le calcul de la machine de code e sur l'entrée x pouvant accéder à l'oracle $g :\subseteq \mathbb{N} \to \mathbb{N}$.

On rappel que pour $x \notin \text{dom}(g)$, $g(x) = \perp$, donc une machine calculant cette valeur ne termine pas, et la fonction utilisant l'oracle non plus. En d'autres termes, appeler lors d'un calcul un oracle en dehors de son domaine nous fait boucler.

Définition 1.6 (fonction calculable avec oracle). *On dit qu'une fonction $f :\subseteq \mathbb{N} \to \mathbb{N}$ est g-calculable s'il existe un code e pouvant avoir recours à l'oracle $g : \mathbb{N} \to \mathbb{N}$ tel que*

$$f(x) = \Phi_e^g(x),\ \forall x \in \text{dom}(f).$$ ∘

Définition 1.7 (Réduction Turing). *Pour deux fonctions partielles $f :\subseteq \mathbb{N} \to \mathbb{N}$, $g :\subseteq \mathbb{N} \to \mathbb{N}$, on dit que f se **réduit** à g s'il existe un code e qui, utilisant g comme oracle, permet de calculer f, c'est-à-dire :*

$$\exists e \in \mathbb{N},\ \forall x \in \text{dom}(f),\ \Phi_e^g(x) = f(x).$$

On note alors

$$f \leq_T g.$$ ∘

Proposition 1.8. *La relation $\leq_T$ est une relation d'ordre partielle (et non totale).* ♢

Pour montrer que la réduction Turing n'est pas totale, il faut construire des fonctions qui, jusqu'à ce que nous savons aujourd'hui, ne sont pas naturelles. Par exemple, par la méthode des extensions finies. Nous n'en parlerons pas plus ici.

Définition 1.9 (Degré Turing). *Deux fonctions partielles f et g sont dites de même degré Turing si $f \leq_T g$ et $g \leq_T f$. On note alors $f \equiv_T g$.*

Le degré Turing d'une fonction partielle f est l'ensemble des fonctions g qui lui sont équivalentes par la réduction Turing :

$$\deg_T(f) = \{g \in \mathbb{N} \to \mathbb{N} : f \equiv_T g\}$$

*On note les degrés Turing avec des lettres minuscules en gras (**a**, **b**,...).* ∘

Remarquons que la définition considère, en toute généralité, plus que les problèmes de décision. Cela diffère donc de certaines définitions dans la littérature qui restreignent leur étude aux ensemble dénombrables (ou fonctions de $\mathbb{N} \to \{0,1\}$, car ces deux notions sont très proches).

Notons aussi que la relation $\equiv_T$ est une relation d'équivalence sur les fonctions partielles. Si $X \leq_T Y$, alors toute fonction du degré de X est calculable avec n'importe quelle fonction du degré de Y comme oracle. La réduction Turing induit donc un ordre partiel sur les degrés Turing, on note cette structure $(\mathcal{D}, \leq)$.

Définition 1.10 (fonction calculatoirement énumérable). *Une fonction partielle $f :\subseteq \mathbb{N} \to \{0,1\}$ est dite calculatoirement énumérable (abrégé c.e.) s'il existe une fonction calculable* $\text{enum}_f : \mathbb{N} \to \mathbb{N}$ ***totale*** *telle que* $\text{enum}_f(\mathbb{N}) = \{x : f(x) = 1\}$. ∘

C'est-à-dire que nous pouvons **énumérer** effectivement les instances positives de f (les x tels que $f(x) = 1$). En effet, pour tout entier $x \in D = \{x \in \mathbb{N} : f(x) = 1\}$, il existe i tel que $\text{enum}_f(i) = x$: en faisant varier i de 0 jusqu'à $+\infty$ nous parcourons au moins un antécédent par enum_f de chacun des éléments de D.

Remarque

La calculabilité sur les ensembles infinis dénombrables se définit de manière analogue à celle sur les fonctions **totales** $\mathbb{N} \to \{0,1\}$, car en considérant la fonction indicatrice d'un tel ensemble, c'est le même objet qui est étudié.

Définition 1.11 (Ensemble calculatoirement énumérable). *Un ensemble $X \subseteq \mathbb{N}$ est dit calculatoirement énumérable s'il existe un code de fonction e tel que*

$$\forall x \in \mathbb{N},\ \Phi_e(x) \downarrow = 1 \text{ si } x \in X$$ ∘

Proposition 1.12. *Une partie de $X \subseteq \mathbb{N}$ est c.e. ssi sa fonction caractéristique associée*

$$\chi_X : \left| \begin{array}{ccc} \mathbb{N} & \longrightarrow & \{0,1\} \\ x & \longmapsto & \begin{cases} 1 \text{ si } x \in X \\ \perp \text{ sinon} \end{cases} \end{array} \right.$$

est calculable. ♢

PREUVE. Les fonctions $enum_{\chi_X}$ de la définition 1.10 et celle définie par le code e de la définition 1.11 sont les mêmes, ce qui permet de conclure. ■

Notation (Partie de $\mathbb{N}$ comme mot binaire infini)

On représente une part P de $\mathbb{N}$ par un mot binaire infini w. Pour $x \in \mathbb{N}$, on choisit $w[i] = 1$ si $x \in P$, 0 sinon. On voit bien que l'on peut établir une bijection calculable entre $\mathcal{P}$ et $2^{\mathbb{N}}$.

Théorème 1.13 (Caractérisation des ensembles calculatoirement énumérables). *Un ensemble X est c.e. ssi il existe une fonction* ***totale*** $f : \mathbb{N} \to \{0,1\}^{\mathbb{N}}$ ***calculable*** *telle que*

$$f(\mathbb{N}) = X.$$

La réduction Turing n'est pas très adaptée pour l'étude des ensembles (ou degrés) c.e. car elle n'est pas assez fine pour comparer un ensemble c.e. de son complémentaire. En particulier, $\emptyset' \equiv_T$ co-$\emptyset'$.

Définition 1.14 (Réduction many-one). *La réduction many-one est la plus petite relation binaire $\leq_m$ telle que pour tous ensembles $X \subseteq \mathbb{N}$ et $Y \subseteq \mathbb{N}$, s'il existe une fonction* ***calculable*** *$f : \mathbb{N} \to \mathbb{N}$ telle que $x \in X \Leftrightarrow f(x) \in Y$ alors $X \leq_m Y$. C'est une relation d'ordre. On dit alors que X est many-one réductible à Y.* ◦

Définition 1.15 (problème complet). *Soit $\mathcal{C}$ une classe de problèmes, et $\leq$ une réduction sur les problèmes de cette classe. Un problème X est complet pour la classe $\mathcal{C}$ par la réduction $\leq$ si :*

- *X appartient à la classe : $X \in \mathcal{C}$*
- *X est $\mathcal{C}$-dur, c'est-à-dire que tout problème de $\mathcal{C}$ se réduit à X : $\forall Y \in \mathcal{C},\ Y \leq X$.* ◦

Le problème universel est similaire au problème de l'arrêt, la différence réside en ce fait qu'il ne prend pas un mais deux paramètres : un code e et une entrée n, et renvoie la terminaison de la fonction de code e sur l'entrée n, au lieu d'utiliser le même entier pour le code et l'entrée. Ce problème est aussi indécidable.

Définition 1.16 (Problème universel). *Le problème universel est l'ensemble*

$$\{< e, n >\in \mathbb{N} : \Phi_e(n) \downarrow\}$$ ◦

Proposition 1.17. *Le problème universel est c.e.* ♢

PREUVE. Donnons un code pour le problème universel.

```
val universal_tm : int * int -> int (* On suppose l'existence d'une
    machine universelle. *)
let arret (e, x : int * int) -> int =
  ignore(universal_tm(e, x)); 1
```

Si $\Phi_e(x) \downarrow$ alors `universal_tm(e, x)` s'arrête et on renvoie alors 1. Sinon, on boucle indéfiniment ($\perp$). ■

Théorème 1.18 (Le problème universel est c.e.-complet pour $\leq_m$). *Le problème universel $\emptyset'$ est calculatoirement énumérable complet pour la réduction many-one ($\leq_m$).* ♢

PREUVE. Le problème universel est c.e. Montrons qu'il est c.e-difficile pour la réduction $\leq_m$. Soit X un ensemble *c.e.*. Par définition, la fonction partielle calculable suivante existe :

$$f_X \left| \begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N} \\ x & \longmapsto & \begin{cases} 1 \text{ si } x \in X \\ \perp \text{ sinon} \end{cases} \end{array} \right. .$$

Alors, la fonction totale **partielle** $g : x \mapsto < f_X, x >$ est une réduction many-one c'est-à- dire que

$$x \in X \leftrightarrow g(x) \in \emptyset'.$$

($\Leftarrow$) : Si $x \in X$, alors $f_X(x) \downarrow$ donc on a bien $< f_X, x > \in \emptyset'$.
($\Rightarrow$) : Si $< f_X, x > \in \emptyset'$, alors $f_X(x) \downarrow$. Mais par construction de f_X, c'est forcément que $f_X(x) = 1$ avec $x \in X$. ■

Plus généralement, toute fonction r.e. mais non calculable qui est "naturelle" est r.e.-complète pour la réduction many-one.

2 Calculabilité de type 2

2.1 Un modèle

On note $2^{\mathbb{N}}$ l'ensemble des mots binaires infinis sur l'alphabet $\{0, 1\}$ parfois noté "2". $p[m]$ dénote le m-ième symbole du mot p.

Pour parler de calculabilité "de type 2", sur des ensembles possédant la cardinalité du continu comme $2^{\mathbb{N}}$, nous devons définir un modèle de calcul. Nous choisirons la machine de Turing dite "de type 2", généralisant la machine de Turing en calculabilité classique.

Définition 2.1 (Machine de Turing de type 2). *Une machine de Turing de type 2 est une machine de Turing en calculabilité classique (de "type 1") qui possède une entrée de taille infinie.* ∘

Il faut adapter la définition de calculabilité sur ce nouveau modèle.

Définition 2.2. *(Calculabilité $2^{\mathbb{N}}$) Une fonction partielle $f :\subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ est dite **calculable** s'il existe une machine de Turing de type 2 qui, pour tout mot binaire infini $p \in 2^{\mathbb{N}}$, ne s'arrête pas d'écrire $f(p)$ sur la bande de sortie si $p \in \text{dom}(f)$, et qui s'arrête si $p \notin \text{dom}(f)$.* ∘

Sur la bande d'entrée comme sur la bande de sortie, les mots infinis de $2^{\mathbb{N}}$ sont représentés comme une suite infinie des symboles qui les composent, c'est-à-dire que chaque cellule des bandes d'entrée et de sortie contient un symbole de l'alphabet $\{0, 1\}$.

Nos précédentes définitions ne nécessitent pas de considérations particulières sur les représentations (généralisant au type 2 le concept de "notations" défini pour le type 1) étudiées dans le document précédent celui-ci, car nos machines ont directement les symboles 0 et 1 dans leurs alphabets, permettant de représenter parfaitement (telles quelles) les chaînes binaires infinies de $2^{\mathbb{N}}$. Il y a unicité de la représentation des chaînes de $2^{\mathbb{N}}$. Nous pourrions utiliser la même stratégie pour des mots sur Σ^ω avec Σ un alphabet fini, mais cela serait une autre affaire si nous étudions par exemple la calculabilité sur l'ensemble $\mathbb{R}$.

Définition 2.3. *(Ensemble calculable) Un ensemble $R \subseteq 2^{\mathbb{N}}$ est dit calculable s'il existe une*

fonction totale calculable $F : 2^{\mathbb{N}} \to \{0,1\}$ *telle que*

$$\forall p \in 2^{\mathbb{N}},\ F(p) = \begin{cases} 1 \text{ si } p \in R \\ 0 \text{ sinon} \end{cases} .$$

∘

Notation (projection sur $2^{\mathbb{N}} \times 2^{\mathbb{N}}$)

On note $< p, q >: 2^{\mathbb{N}} \to 2^{\mathbb{N}} \times 2^{\mathbb{N}}$ la bijection calculable restituant deux mots binaires infinis à partir d'un seul. Par exemple, nous pouvons construire la chaîne s telle que $s[2n] = p[n]$ et $s[2n+1] = q[n]$, pour tout $n \in \mathbb{N}$.

Notation (projection sur $2^{\mathbb{N}} \times \mathbb{N}$)

On note $< p, n >: 2^{\mathbb{N}} \to 2^{\mathbb{N}} \times \mathbb{N}$ la bijection calculable faisant la même chose que la notation précédente mais avec un mot binaire infini et un mot binaire fini. On peut considérer une chaîne $s = p[0]0n[0]0p[1]0n[1]0...0p[k]0n[k]0...$ où les $p[i]$ et $n[i]$ sont écrits en unaire, les "0" servant de séparateur, avec la convention que $n[i] = \epsilon$, la chaîne vide, pour $i > \log_2(n)$.

Avec ces projections, on se convainc que nous pouvons construire des bijections calculables entre $2^{\mathbb{N}}$ et $(2^{\mathbb{N}})^n \times (\mathbb{N})^m$, pour tous entiers $n \geq 1$ et $m \geq 0$. Par convention, p et q seront des mots infinis quand n, m ... seront des entiers.

On utilisera aussi cette notation comme sucre syntaxique pour nos codes en `OCaml`.

2.2 La réduction Weihrauch

Pour comparer et classifier des problèmes, la notion de réduction est fondamentale en calculabilité. Nous définissons ici une réduction prometteuse (ou plutôt qui a déjà fait ses preuves) pour le type 2.

Définition 2.4. *(Réduction Weihrauch faible) Pour deux fonctions partielles* $F, G :\subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$*, F est dite Weihrauch-réductible à G, ce qui se note $F \leq_W G$, s'il existe une machine de Turing de type 2 calculant F en faisant un seul appel à l'oracle G lors de son exécution.* ∘

Puis, une version plus forte :

Définition 2.5. *(Réduction Weihrauch forte) Une réduction Weihrauch est forte (noté $F \leq_{fW} G$) si nous n'utilisons pas l'entrée (ni un résidu d'entrée) après notre appel à l'oracle. Formellement, il existe deux fonctions partielles* $A, B :\subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ *telles que l'égalité suivante est juste et bien définie :*

$$\forall p \in \mathrm{dom}(F),\ F(p) = (A \circ G \circ B)(p)$$

∘

Nous nous contenterons d'utiliser la réduction Weihrauch faible pour nos preuves, en utilisant parfois le fait que la réduction Weihrauch forte implique la réduction Weihrauch (faible). Il suffit de relâcher l'hypothèse que nous n'utilisons plus l'entrée après avoir appelé l'oracle.

2.3 Hiérarchie arithmétique

Comme généralisation au type 2 de la hiérarchie arithmétique en calculabilité classique, nous proposons les définitions suivantes :

Définition 2.6. *Une fonction* $F :\subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ *est dite* Σ^0_k *s'il existe un ensemble calculable* $R \subseteq 2^{\mathbb{N}}$ *tel que*

$$\forall p \in \mathrm{dom}(F),\ \forall n \in \mathbb{N},\ F(p)(n) = \begin{cases} 1 \text{ si } \exists n_1,\ \forall n_2,\ \ldots,\ Qn_k,\ < p, n, n_1, \ldots, n_k > \in R \\ \perp \text{ sinon} \end{cases}$$

où $Q = \forall$ *si* k *est pair, et* $Q = \exists$ *sinon.* ∘

Le symbole "$\perp$" indique que nous n'inscrirons jamais de symbole sur la n-ième cellule de la sortie d'une machine de Turing de type 2 calculant $F(p)$.

On précise que $F(p)(n)$ signifie "le n-ième bit de l'image de la chaîne p par F", et que $F(p)$ peut être vue comme une fonction qui à un entier associe un bit $\in \{0, 1\}$. C'est-à-dire que les ensembles $2^{\mathbb{N}}$ et $[\mathbb{N} \to \{0, 1\}]$ sont confondus. En fait, en théorie des ensembles c'est le même objet, donc le lecteur averti ne sera pas surpris.

De la même manière, nous définissons les ensembles Π^0_k :

Définition 2.7. *Une fonction* $F :\subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ *est dite* Π^0_k *s'il existe un ensemble calculable* $R \subseteq 2^{\mathbb{N}}$ *tel que*

$$\forall p \in \mathrm{dom}(F),\ \forall n \in \mathbb{N},\ F(p)(n) = \begin{cases} 1 \text{ si } \forall n_1,\ \exists n_2,\ \ldots,\ Qn_k,\ < p, n, n_1, \ldots, n_k > \in R \\ \perp \text{ sinon} \end{cases}$$

où $Q = \exists$ *si* k *est pair, et* $Q = \forall$ *sinon.* ∘

Théorème 2.8. *Nous pouvons relier le début de la hiérarchie arithmétique aux ensembles de fonctions déjà rencontrés.*

— $\Sigma^0_0 = \Pi^0_0$ *est l'ensemble des fonctions calculables.*

— Σ^0_1 *est l'ensemble des fonctions récursivement énumérables.*

— Π^0_1 *est l'ensemble des fonctions co-récursivement énumérables.*

En notant $<_W$ la réduction Weihrauch stricte, c'est-à-dire telle que $\Sigma_k \leq_W \Sigma_{k+1}$ mais que nous ne pouvons pas réduire Σ_{k+1} à Σ_k ($\Sigma_{k+1} \not\leq_W \Sigma_k$), nous établissons les résultats suivants :

Théorème 2.9. *La hiérarchie arithmétique est stricte, c'est-à-dire que pour tout* $k \in \mathbb{N}$ *:*

$$\Sigma^0_k <_W \Sigma^0_{k+1}$$

De plus,

$$\Pi^0_k <_W \Sigma^0_{k+1}.$$

Corollaire 2.10. *Pour tout* $k \in \mathbb{N}$*,*

$$\Pi^0_k <_W \Pi^0_{k+1}$$

et

$$\Sigma^0_k <_W \Pi^0_{k+1}.$$

3 Deux problèmes complets

3.1 Le problème C

Développons plusieurs résultats qui serviront pour la deuxième partie de cette section.

Définition 3.1 (Brattka, 5.3). *On définit*

$$C : \left| \begin{array}{ccc} 2^{\mathbb{N}} & \longrightarrow & 2^{\mathbb{N}} \\ p & \longmapsto & n \mapsto \begin{cases} 0 \text{ si } \exists m,\ p(< n, m >) \neq 0 \\ 1 \text{ sinon} \end{cases} \end{array} \right. .$$

∘

Remarque : le problème C tel que nous l'avons défini est noté C_1 dans [2].

Théorème 3.2 (Complétude, Brattka 5.5). *Montrons que C est Σ^0_2-difficile, c'est-à-dire que pour toute fonction $F :\subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$,*

$$F \text{ est } \Sigma^0_2\text{-calculable} \implies F \leq_W C. \tag{1}$$

Idée de preuve

Nous construisons en parallèle deux chaînes p' et q', chacune faisant un appel à C, permettant de déterminer la valeur de $F(p)(n)$ pour une instance donnée (p, n) : s'il existe un entier m vérifiant $p' < n, m > = 1$, alors c'est que $q' < n, m > = 1$ ne peut être vérifié et que $F(p)(n) = 0$, sinon c'est l'inverse.
On montre ensuite que nous pouvons faire deux appels simultanés en un appel à C.

Lemme 3.3. *Pour toute fonction $F \subseteq 2^{\mathbb{N}} \to 2^{\mathbb{N}}$, $F \leq_W C \times C$.* ♢

Preuve. Il existe deux ensembles calculables P et Q tels que

$$F : \left| \begin{array}{ccc} 2^{\mathbb{N}} & \longrightarrow & 2^{\mathbb{N}} \\ p & \longmapsto & n \mapsto \begin{cases} 0 \text{ if } \exists n_2,\ \forall n_1,\ < p, n, n_1, n_2 > \in P \\ 1 \text{ if } \exists n_2,\ \forall n_1,\ < p, n, n_1, n_2 > \in Q \end{cases} \end{array} \right.$$

En effet, puisque $F \in \Sigma^0_2$, il existe un ensemble R satisfaisant la définition 2.6 pour $k = 2$. On peut alors prendre $P = R$ et $Q = \{p \in 2^{\mathbb{N}} : p \notin R\}$.

```
type chaine = int -> bool

val c : chaine -> chaine
val set_p : chaine -> bool
val set_q : chaine -> bool

exception Return of bool
let red (p: chaine) : chaine =
  let bp p <<n, m>, n1> = not (set_p <p, n, m, n1>)     (* chaîne B_P(p) *)
  and bq p <<n, m>, n1> = not (set_q <p, n, m, n1>) in (* chaîne B_Q(p) *)

  let p' = c bp p
  and q' = c bq p in
```

```
fun n ->
  try
    for m = 0 to max_int do
      if p' <n, m> then Return false;
      if q' <n, m> then Return true
    done
  with Return b -> b
```

Par hypothèse, P et Q sont des ensembles calculables ce qui justifie que nous pouvons calculer leur n-ième caractère pour tout n, et les chaînes $B_P(p)$ et $B_Q(p)$ sont donc calculables. Ensuite, nous faisons deux appels simultanés à C pour construire des chaînes p' et q', dont nous allons montrer qu'elles permettent de calculer la chaîne $F(p)$.

Pour n fixé, montrons :

$$(\exists m,\ p' < n, m >= 1) \oplus (\exists m,\ q' < n, m >= 1) \tag{2}$$

où $\oplus$ est le symbole du ou exclusif.

$$\begin{aligned}
\exists m,\ p' < n, m >= 1 &\iff \exists m,\ C(B_P(p)) < n, m > = 1 && \textit{(par définition de } p'\textit{)}\\
&\iff \exists m,\ \neg(\exists m',\ B_P(p) << n, m >, m' > \neq 0) && \textit{(par définition de } C\textit{)}\\
&\iff \exists m,\ \neg(\exists m',\ < p, n, m, m' > \notin P) && \textit{(par définition de } B_P(p)\textit{)}\\
&\iff \exists m,\ \forall m',\ < p, n, m, m' > \in P && \textit{(négation logique)}\\
&\iff F(p)(n) = 0 && \textit{(par définition de } F\textit{)}
\end{aligned}$$

De la même manière,

$$(\exists m,\ q' < n, m > = 1) \iff F(p)(n) = 1$$

Puisque F est bien définie, on conclut (2). Ainsi, en énumérant les entiers dans la boucle "for" de notre programme, on est assuré de trouver m tel que $p' < n, m >= 1$ ou bien $q' < n, m >= 1$, de manière exclusive d'après ce que nous venons de montrer, donc notre réduction est bien définie (c'est-à-dire que peu importe l'énumération choisie, nous aboutirons au même résultat). Puisque nous avons fait deux appels simultanés à C lors de la réduction, nous avons montré que $F \leq_W C \times C$ où $\leq_W$ est la réduction Weihrauch. ∎

Lemme 3.4. *Faire un appel à C est aussi difficile que de faire un nombre fini d'appels simultanés à C. C'est-à-dire :*

$$C \times C \leq_W C$$

Preuve. Écrivons le code d'une telle réduction.

```
type chaine = int -> bool

val c : chaine -> chaine

let red (<p, q> : chaine) -> chaine * chaine =
  let s <n, m> =
    if n % 2 = 0 then
      p<n, m>
    else
      q<n, m>
  in
  c s                              (* du type chaine -> chaine *)
  |> fun <x, y> n -> (x n, y n)    (* du type chaine * chaine  *)
```

On a red $< p, q > 2n = p(n)$ et red $< p, q > 2n + 1 = p(n)$, pour tout $n \in \mathbb{N}$. De plus, nous avons bien fait qu'un seul appel à C. ∎

Preuve. (Théorème 3.2) Par transitivité de la réduction Weihrauch, les deux lemmes précédents nous permettent de conclure que $F \leq_W C$. ∎

Proposition 3.5. *L'ensemble C appartient à Σ^0_2.* ♢

Preuve. On construit l'ensemble calculable

$$R = \{< p, n, m, m' > \in 2^{\mathbb{N}} : p < n, m > \neq 0\}.$$

On a alors $C(p)(n) = \begin{cases} 0 \text{ si } \exists m,\ \forall m',\ < p, n, m, m' > \in R \\ \perp \text{ sinon} \end{cases}$, ce qui montre que C est Σ^0_2. On peut aussi montrer que C est Π^0_2. ∎

On conclut que C est Σ^0_2-complet.

3.2 Le problème lim

Le précédent problème C n'est, tel quel, pas très naturel. Cependant, il va nous permettre de montrer la complétude d'un second problème plus naturel.

Définition 3.6. *(Suite de Cantor) On appelle suite de Cantor une suite de mots binaires infinis indexée par des réels, ce que l'on note $(p_n)_{n\in\mathbb{N}} \in (2^{\mathbb{N}})^{\mathbb{N}}$.* ○

Définition 3.7. *(Convergence) Une suite de Cantor $(p_n)_{n\in\mathbb{N}}$ est dite **convergente** si pour tout entier k, la suite est "stable" jusqu'à k à partir d'un certain rang. C'est-à-dire que*

$$\forall k \in \mathbb{N},\ \exists n \in \mathbb{N},\ \forall m \geq n,\ p_m[i] = p_n[i],\ \forall i \leq k$$ ○

On pourrait aussi étudier la convergence où le nombre de bits corrects augmente, mais pas forcément à partir du début. C'est-à-dire que, pour l la limite de la suite, nous avons le résultat suivant :

$$\forall k \in \mathbb{N},\ \exists n \in \mathbb{N},\ \forall m \geq n,\ \{i \leq \mathbb{N} : l[i] = p_m[i]\} \geq k.$$

Cette définition est plus faible que la précédente.

Définition 3.8. *(Problème de la limite) On appelle problème de la limite la fonction suivante :*

$$\lim : \left| \begin{array}{ccc} \subseteq (2^{\mathbb{N}})^{\mathbb{N}} & \longrightarrow & 2^{\mathbb{N}} \\ (p_n)_{n\in\mathbb{N}} & \longmapsto & \lim\limits_{n\to\infty} p_n \end{array} \right.$$

qui, à tout suite de mots binaire infinis qui converge au sens de la définition précédente, associe sa limite. ○

Théorème 3.9. *(Complétude de* lim*, ([2], 9.2)) Le problème de la limite est Σ^0_2-complet.* ♢

Preuve. Montrons que $C \leq_W \lim$. On construit une fonction calculable

$$G = \left| \begin{array}{ccc} 2^{\mathbb{N}} & \longrightarrow & (2^{\mathbb{N}})^{\mathbb{N}} \\ p & \longmapsto & n \mapsto \begin{cases} 0 \text{ si } \exists k \leq m,\ p < n, k > \neq 0 \\ 1 \text{ sinon} \end{cases} \end{array} \right. .$$

On a

$$\begin{aligned} \lim(G(p)(n)) = 0 &\iff \exists k,\ p < n, k > \neq 0 && \textit{(car } \lim_{m\to\infty} m = \infty\textit{)} \\ &\iff C(p)(n) && \textit{(par définition de } C\textit{)} \end{aligned}$$

Donc $C \leq_m \lim$ ce qui implique $C \leq_W \lim$ (renvoyer la sortie de l'oracle sans plus de calculs).

Il manque à montrer que $\lim \in \Sigma^0_2$, nous renvoyons le lecteur à la proposition 9.2 de [2] pour une preuve nécessitant d'introduire de nombreuses définitions préliminaires, sur des considérations topologiques dont nous ne nous sommes pas encore penchés. ∎

Conclusion

Après avoir été introduits à l'analyse calculable avec[3], nous avons étudié ce qui existait dans la littérature à propos des problèmes complets au type 2. Mais, des livres comme[4] ne propose que des problèmes pour les fonctions $2^{\mathbb{N}} \to \mathbb{N}$ ce qui est restrictif puisque des machines pour de telles fonctions n'ont pas besoin de lire toute leur entrée pour calculer leur sortie. Cela nous a poussés à considérer le papier[2] énonçant des résultats prometteurs. Dès lors, le principal travail fut de comprendre et de reformuler les notions décrites dans ce document, pour les rendre plus accessibles. Ce rapport n'est aucunement un substitut à ces papiers déjà existants, puisqu'il est moins riche, mais se propose comme un entre-deux, permettant une introduction aux thématiques étudiés. Finalement, nous avons montré qu'il existe des problèmes complets en type 2, et que certains de ces problèmes sont même "naturels" (comme la limite d'une suite), ce qui était l'objectif du projet.

Ouverture

Une autre approche pertinente pour découvrir des problèmes complets serait d'étudier les degrés, par exemple pour la réduction Weihrauch.

Références

[1] Benoît Monin et Ludovic Patey. *Calculabilité*. Calvage et Mounet, 2022.

[2] Vasco Brattka. Effective borel measurability and reducibility of functions. *Mathematical Logic Quarterly*, 51(1) :19–44, 2005.

[3] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. *A Tutorial on Computable Analysis*, pages 425–491. Springer New York, New York, NY, 2008.

[4] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.