# Chapitre 2 : Paradoxes de la diagonalisation et problème de l'arrêt



## 1. Cantor

Le chapitre précédent donne plusieurs exemples fondamentaux d'ensembles *dénombrables*, c'est-à-dire qui peuvent être mis en bijection avec l'ensemble des entiers : couples d'entiers, listes d'entiers, mots, etc. On peut alors se demander si tous les ensembles mathématiques "simples" sont dénombrables. La réponse est négative, et a été fournie par le mathématicien allemand (né à Saint-Petersbourg d'un père danois et d'une mère russe, vive l'Europe !) Cantor, fondateur de la théorie des ensembles. Cantor a prouvé en 1873 que l'ensemble des nombres rationnels est dénombrable (par la méthode de numérotation des couples d'entiers, vue au chapitre précédent), puis que l'ensemble des nombres réels ne l'est pas.

Les résultats négatifs sont souvent plus difficiles à prouver que les résultats positifs : ici montrer qu'il n'existe pas de numérotation est plus difficile que d'en fabriquer une. La méthode de Cantor revient à prouver un résultat comparable sur les fonctions $\mathbf{N} \to \mathbf{N}$, autrement dit sur les suites *infinies* d'entiers -- $u_n$ et $f(n)$ sont deux notations pour le même concept :

*Théorème (Cantor) :* l'ensemble des fonctions $\mathbf{N} \to \mathbf{N}$
n'est pas dénombrable.

La preuve de Cantor repose sur une méthode simple et astucieuse, dite de *diagonalisation*. On considère une numérotation des fonctions, et on montre qu'elle est forcément incomplète ; pour cela on imagine un tableau infini $U$, dont la *ligne* numéro $i$ contient les différentes valeurs

$$f_i(0), f_i(1), f_i(2), \ldots, f_i(j), \ldots$$

de la fonction numéro $i$ ; l'élément $f_i(j)$, à l'intersection de la ligne $i$ et de la colonne $j$ est aussi noté $U(i, j)$. Il reste alors à considérer la fonction $\gamma$ définie pour tout $n$ par :

$$\gamma(n) = U(n, n) + 1 = f_n(n) + 1$$

Le "1" qui termine la définition est une commodité, et peut être remplacé par n'importe quel terme *non nul* : l'important dans la définition de la fonction $\gamma$ est que, pour tout $n$, $\gamma(n)$ est *différent* du terme *diagonal* $U(n, n)$ — autrement dit $f_n(n)$. On voit alors immédiatement que $\gamma$ ne peut pas avoir de numéro (aucune ligne du tableau $U$ ne peut représenter la fonction $\gamma$) !

Remarque : on voit que les suites du type de $\gamma$ sont loin d'être rares, il y en a au moins autant que de suites $u$ "jamais nulles" (pour tout $n$, $u_n > 0$), puisqu'en ajoutant une telle suite à la suite diagonale $U(n, n)$, on fabrique une suite qui ne peut correspondre à une ligne de $U$ ; et les suites "jamais nulles" ne sont pas moins nombreuses que les suites quelconques (il suffit d'ajouter 1 à chaque terme d'une suite quelconque pour la transformer en une suite jamais nulle). Des théorèmes généraux de Cantor montrent qu'un ensemble non dénombrable est "radicalement plus gros" qu'un ensemble dénombrable.

## 2. Fonctions calculables

Si $p$ désigne une procédure, et $n$ un entier, $p(n)$ désigne (dans notre modèle de calcul, basé sur C, comme dans tout langage de programmation moderne) le résultat de l'*application* de $p$ à $n$ ; ce résultat

est un entier si l'exécution se termine par une instruction `return`, il est indéfini sinon (nous reviendrons bientôt sur ce point, loin d'être anecdotique). A une procédure $p$, est donc associée une fonction mathématique

$$f : n \to p(n)$$

Malgré la similitude (intentionnelle) des notations, il ne faut pas confondre la procédure $p$ et la fonction mathématique $f$, qui est, par définition, la fonction *calculée* par $p$ ; comme tout programmeur le sait, il existe bien des façons de calculer la même fonction, mais surtout mentionnons le résultat principal du chapitre 3 : savoir si deux procédures calculent la même fonction est carrément *indécidable* (ce terme sera défini rigoureusement un peu plus loin), aussi surprenant que cela puisse paraître ! C'est aussi la raison pour laquelle nous nommons *procédure*, dans ce cours, ce qui en C est appelé *fonction*.

Sans surprise, une fonction $f$ est dite *calculable* s'il existe une procédure qui la calcule, au sens expliqué ci-dessus. Comme on peut numéroter les procédures (voir chapitre précédent), on déduit immédiatement du théorème de Cantor qu'il existe des fonctions $\mathbf{N} \to \mathbf{N}$ non calculables, et qu'elles sont même la grande majorité, d'après la remarque qui termine la section précédente. Elles correspondent aux suites *aléatoires*, dont chaque terme est imprévisible ; il semble normal que de telles suites ne puissent être produites par des procédures, qui sont finies par définition (suites finies d'instructions dans le modèle de Von Neumann).

L'argument de diagonalisation de Cantor semble maintenant conduire à un vrai paradoxe ; numérotons $p_0$, $p_1$, $p_2$, … les procédures, et définissons $\gamma$ par :

$$\gamma\,(n) = p_n\,(n) + 1$$

Cette fonction peut être calculée par une procédure simple (voir ci-dessous pour les détails de cette procédure), laquelle ne semble pourtant pas posséder de numéro : si $\gamma$ est calculée par $p_i$ , on aboutit à la contradiction :

$$p_i\,(i) = \gamma\,(i) = p_i\,(i) + 1\ !$$

En fait, l'examen attentif de ce paradoxe montre que la seule issue logique est la suivante : $\gamma\,(i)$ *n'est pas défini* ! Une fonction $f$ dont le domaine de définition est $\mathbf{N}$ tout entier (pour tout entier naturel $n$, $f\,(n)$ est défini), est dite *totale* ; si le domaine de définition est autorisé à être un sous-ensemble de $\mathbf{N}$, $f$ est dite *partielle* ; avec cette terminologie, les fonctions totales font partie des fonctions partielles, on spécifiera donc qu'une fonction est *strictement* partielle si elle n'est pas totale. Le paradoxe de la diagonalisation prouve donc le résultat suivant :

Dans tout modèle de calcul,
il existe des fonctions calculables partielles (strictement).

On peut être surpris de l'affirmation "dans tout modèle de calcul", à ce stade du cours ; mais cette affirmation est justifiée en remarquant que les seules propriétés du modèle de calcul que nous avons utilisées sont :

- les procédures sont numérotables ;
- la fonction *universelle*, qui à une procédure $p$ et à un entier $x$ associe $p(x)$ — résultat de l'application de $p$ à $x$, est calculable.

Remarque : la fonction universelle a deux arguments, une procédure et un entier ; mais on peut considérer qu'elle a un seul argument entier, en numérotant les couples formés d'une procédure et d'un entier. Dans les langages de programmation modernes, la calculabilité de la fonction universelle est masquée par l'usage direct de la notation $p(x)$ ; il faut se rappeler que la traduction d'un tel *appel de procédure* en langage machine ne va pas de soi : il faut une instruction pour empiler l'argument $x$, puis une autre pour placer l'adresse de $p$ dans le compteur ordinal, en empilant en même temps "l'adresse de retour" ; le code de $p$ doit se terminer par une instruction qui remplace l'argument par le résultat, puis par une instruction qui dépile l'adresse de retour dans le compteur ordinal ; à cet endroit du programme, le résultat de l'appel $p(x)$ se trouve sur la pile, et c'est ainsi que la *fonction universelle*

est calculable dans le modèle de Von Neumann !

Pour insister sur le caractère général des arguments employés, nous n'avons pas écrit les procédures utilisées ci-dessus en C, mais faisons-le maintenant :

```
typedef int proc (int);

proc *procedureNumero (int n) {…}

int gamma (int n) {
  proc *p = procedureNumero (n);
  return p(n) + 1;
}
```

Nous ne pouvons pas donner le code de `procedureNumero`, qui réalise la numérotation des procédures, car ce code serait bien trop long et inefficace si l'on suit les principes du chapitre précédent. Par contre l'existence d'un tel code est un axiome de tout modèle de calcul, et ici on pourrait aisément le simuler en utilisant un *tableau* de procédures ; un tableau est fini par définition, mais il suffit qu'il contienne `gamma` comme élément d'indice *i* pour vérifier que le calcul de `gamma(i)` boucle, puisque la seconde instruction devient `return gamma(i) + 1` .

Une autre approche encore plus directe utilise les adresses pour numéroter les procédures :

```
int gamma (proc p) {
  int n = (int) p;
  return p(n) + 1;
}
```

et l'appel `gamma ((proc *)gamma)` boucle (`gamma` n'a pas le bon type, d'où le forçage `(proc *)` quand on le passe en argument).

On peut objecter que c'est bien du travail pour fabriquer une procédure qui boucle, ce que tout débutant programmeur découvre par lui-même plus rapidement qu'il ne le voudrait. Mais cette section a prouvé que l'existence de procédures qui bouclent est *inévitable* ! En d'autres termes on a prouvé qu'il est illusoire de chercher à créer un langage de programmation "intelligent" dans lequel il serait impossible d'écrire des procédures qui bouclent.

## 3. Fonctions calculables totales

Nous n'avons pas encore épuisé les ressources de la méthode de diagonalisation, découverte par Cantor. En effet, une fois la section précédente assimilée, on peut revenir à la charge : pourquoi ne pas numéroter seulement les procédures $q_0$, $q_1$, $q_2$, … qui calculent des fonctions *totales* ? Définissons maintenant γ par :

$$\gamma\,(n) = q_n\,(n) + 1$$

γ est une fonction totale, et une procédure calculant γ ne peut pas avoir de numéro : on semble ramené exactement à la situation qui a permis à Cantor de démontrer que l'ensemble des fonctions $\mathbf{N} \to \mathbf{N}$ n'est pas dénombrable.

Il existe à nouveau une issue logique à ce paradoxe, et une seule : cette fois γ n'est pas calculable ! Alors qu'il existe un algorithme (terriblement inefficace, mais ce n'est pas la question) pour déterminer qui est $p_n$ , la procédure numéro $n$, il n'en existe pas pour déterminer qui est $q_n$ , la "procédure totale" numéro $n$ ! Autrement dit il n'existe pas de fonction C `procedureTotaleNumero`, analogue à la fonction C `procedureNumero`, qui calculerait la $n^{\text{ème}}$ "procédure totale" — dans ce paragraphe, "procédure totale" est une abréviation pour "procédure qui calcule une fonction totale"

Notons bien ce paradoxe : l'ensemble $T$ des procédures totales est un sous-ensemble de l'ensemble des procédures, donc est *dénombrable*, ce qui signifie par définition qu'il existe des bijections entre **N** et $T$ ; mais aucune de ces bijections n'est calculable !

Il est temps de revenir sur le chapitre précédent : chaque *numérotation* présentée dans le chapitre 1 peut être réalisée par une procédure, on dit aussi qu'elle est *effective* ; mais la plupart des sous-ensembles infinis de **N** ne sont pas *effectivement* numérotables, bien que dénombrables : les fonctions qui les dénombrent ne sont pas calculables. Cette distinction est subtile, mais essentielle pour la théorie de la calculabilité : elle sera approfondie au chapitre 4, où les ensembles effectivement numérotables seront désignés par leur nom traditionnel, à savoir ensembles *récursivement énumérables* (lors du développement historique de la théorie de la calculabilité, dans les années 1930, les fonctions *calculables* ont d'abord été appelées *récursives*). Pour notre part, dans ce cours, nous essayons de réserver le terme *numérotation* aux numérotations calculables.

## 4. Indécidabilité du problème de l'arrêt

Pourtant il semble exister un algorithme simple de numérotation des procédures qui calculent des fonctions totales : parcourir toutes les procédures, et éliminer celles à qui il arrive de boucler ! On appelle *problème de l'arrêt* (*halting problem* en anglais) le calcul de la fonction $h$ définie comme suit (les arguments sont une procédure $p$ et un entier naturel $x$) :

$$h(p, x) = \begin{cases} 1 \text{ si } p(x) \text{ est défini} \\ 0 \text{ sinon} \end{cases}$$

Cette fonction n'est pas calculable, sinon une légère modification de l'argument de la section précédente fournit une fonction totale $\gamma$ :

$$\gamma(n) = \begin{cases} p_n(n) + 1 \text{ si } h(p_n, n) = 1 \\ 0 \text{ sinon} \end{cases}$$

qui est calculable (si $h$ l'est), et pourtant une procédure qui calcule *gamma* ne peut pas avoir de numéro ; contradiction, qui prouve que $h$ n'est pas calculable.

Depuis la section 2 nous savons qu'il existe des fonctions non calculables (elles sont même l'écrasante majorité) ; mais voici la première que nous sommes capables de produire explicitement ! Les fonctions qui ne prennent que les valeurs 0 ou 1 sont appelées *fonctions booléennes*, ou, plus brièvement, *prédicats*, et les prédicats calculables sont dits *décidables* ; nous pouvons donc résumer cette section en formulant le théorème célèbre :

Le problème de l'arrêt est indécidable.

Il existe des variantes du problème de l'arrêt, en éliminant le paramètre $x$ :

- la procédure $p$ calcule-t-elle une fonction totale, c'est-à-dire a-t-on :
  pour tout $x$, $h(p, x) = 1$ ?
- existe-t-il $x$ tel que $h(p, x) = 1$ ?

Aucune de ces variantes n'est décidable : s'il existait un algorithme pour décider si une procédure calcule une fonction totale, on pourrait transformer immédiatement l'algorithme de numérotation des procédures (quelconques) en un algorithme de numérotation de $T$ (ensemble des procédures qui calculent une fonction totale), ce qui contredirait la section 3. L'indécidabilité de la seconde variante sera prouvée au chapitre suivant.