Calculabilité et décidabilité

1 Modèles de calcul

Au début du XX^{e} siècle le mathématicien allemand David Hilbert pose un certain nombre de problèmes à la communauté mathématique
Certains de ces problèmes n'ont pas été à ce jour résolus et ont été reconduits pour le XXI^{e} siècle
Le dixième problème de Hilbert (ou entscheidungsproblem ou problème de **décision**) est :
Existe t-il un **algorithme** permettant de **décider** si **toute** équation diophantienne admet une solution ?
Voici un exemple d'équation diophantienne
$3x - 5y = 1$ avec les inconnues x et y des **entiers relatifs**
Une solution particulière à cette équation particulière est $x = 2$ et $y = 1$ mais il se trouve que $x = 7$ et $y = 4$ est aussi solution , on trouve donc une infinité de solutions $x = 2 + 5k$ et $y = 1 + 3k$ avec $k \in \mathbb{Z}$
Par contre l'équation $2x^2 + 2y^2 = 1$ a des solutions réelles mais pas de solutions entières
Exercice 1
A l'aide d'un argument géométrique dire pourquoi cette équation n'a pas de solutions entières mais une infinité de solutions réelles
A l'époque où le problème est posé la notion d'algorithme est "intuitive" et les mathématiciens qui se sont attaqués à ce problème ont ressenti le besoin de donner un sens mathématique à la notion d'algorithme , à la notion de calculable et par conséquent **de définir un modèle de calcul**
En 1936, dans un article intitulé *On computable numbers with an application to the entscheidungsproblem* , (Sur les nombres calculables. Une réponse au problème de décision) Alan Turing introduit un modèle (théorique) de calcul, appelée depuis lors **machine de Turing**
On peut lire cet article en ligne ici `https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf`
(Thèse de Post) Une machine de Turing , une machine de Von Neumann, ou un langage de programmation comme Python , Java, C++, etc... sont des modèles de calcul équivalents dans le sens où "on calcule" les mêmes grandeurs avec chacun de ces modèles
D'où la définition particulière d'une fonction calculable

Définition 1 *Une fonction (au sens mathématique) est calculable si elle peut être programmée dans l'un ou l'autre des langages de programmation (Python au lycée)*

Existe-t-il des fonctions non calculables ?
Il nous reste aussi à définir le sens de "décider si toute équation diophantienne admet une solution"
A suivre ....

2 Problèmes de décision décidables

Définition 2 *Un problème de décision est la donnée d'un ensemble E d'instances et d'un sous ensemble P de E d'instances dites positives*

Exemple de problème de décision :

1. le problème de décision **Nombres premiers** a pour instances $E = \mathbb{N}$ et les instances positives P est l'ensemble des nombres premiers
2. le problème de décision **Graphes connexes** a pour instances l'ensemble de **tous** les graphes finis et les instances positives est l'ensemble des graphes connexes
3. le problème de décision **Equations diophantiennes** a pour instances l'ensemble de **toutes** les équations diophantiennes et les instances positives est l'ensemble des équations diophantiennes ayant des solutions

Définition 3 *Un problème de décision (E, P) est décidable s'il existe une fonction Python définie pour tout $x \in E$ et qui renvoie Vrai si $x \in P$ et Faux sinon*

Exercice 2
Montrer que le problème de décision **Nombres premiers** est **décidable**
Autrement dit définir une fonction Python `estPremier(nombre)` qui retourne Vrai si `nombre` est premier Faux sinon **(On ne soucie pas ici de la complexité donc on pourra définir une fonction "naïve")**
Exercice 3
Montrer que le problème de décision **Graphes connexes** est **décidable** autrement dit définir une fonction Python `estConnexe(graphe)` qui retourne Vrai si `graphe` est connexe Faux sinon

1. Dans un premier temps définir une fonction naïve
2. Dans un second temps faire un parcours en largeur du graphe en partant d'un des sommets si tous les sommets ont été visités au cours de ce parcours alors le graphe est connexe

En 1970 le mathématicien russe Matiassevitch a démontré que le problème de décision **Equations diophantiennes** est **indécidable**
En quelque sorte il a montré ainsi l'existence d'une fonction non calculable, mais Turing a démontré dès 1936 l'existence d'une fonction particulière non calculable
Avant d'étudier ce problème nous allons faire un détour par les quines ...

3 Quines

Définition 4 *Dans tout langage de programmation on peut écrire un programme qui affiche son propre code. On appelle quine un tel programme (vient du nom du logicien américain William Quine)*

Exercice 4
Vérifier que le programme Python suivant est un quine

```
x = ['print("x =", x)', 'for m in x: print(m)']
print ("x =", x)
for m in x: print(m)
```

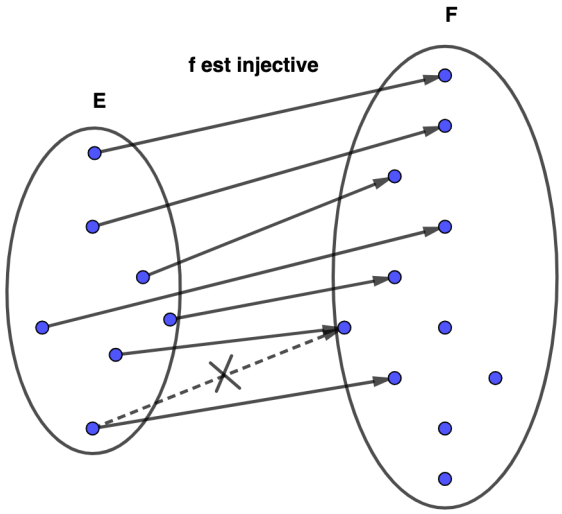
Tout ça pour insister sur le fait qu'**un programme Python est une chaine de caractères et donc peut être vu comme une donnée**
Une autre digression Le procédé diagonal de Cantor

4 Le procédé diagonal de Cantor

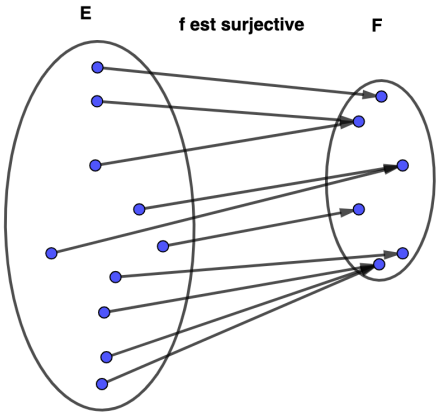
Dans le but de "classer les ensembles infinis" on pose comme première mesure étalon de l'infini l'ensemble $\mathbb{N}$

Définition 5 *Un ensemble E est dit dénombrable s'il existe une bijection de E dans $\mathbb{N}$ On dit alors qu'il a la même puissance que $\mathbb{N}$ en terme d'infini*

Définition 6 *Une fonction f de E dans F est une bijection de E dans F (ou bijective) si f est à la fois injective et surjective*
*f de E dans F est **injective** si deux éléments **distincts** de E ont deux images **distinctes***



par f
*f de E dans F est **surjective** si tout élément de F a au moins un antécédent dans E*



Théorème 1 $\mathbb{Z}$ *et* $\mathbb{Q}$ *ont la même puissance que* $\mathbb{N}$ ***même si*** $\mathbb{N}$ ***est strictement inclus dans*** $\mathbb{Z}$ ***qui est strictement inclus dans*** $\mathbb{Q}$

Théorème 2 *(Procédé diagonal de Cantor) On note E l'ensemble des nombres réels à [0,1[*
E n'est pas dénombrable. Donc $\mathbb{R}$ *n'est pas dénombrable*

Preuve par l'absurde
Chaque nombre de [0,1[a deux développements décimaux
Par exemple 0,2 = 0,199999....
Le premier (sans 9 jusqu'à l'infini !) est dit **propre**, l'autre impropre
On suppose que E est dénombrable donc on peut le ranger sous la forme d'une suite r_0, r_1, ...r_n où chaque r_i est le développement décimal propre
$r_0 = 0, c_{00}c_{01}....$
$r_1 = 0, c_{10}c_{11}....$
...
$r_n = 0, c_{n0}c_{n1}....c_{nn}$
Où chaque c_{ij} est un chiffre de 0 à 9
Soit le nombre $x = 0, \overline{c_{00}c_{11}}...\overline{c_{nn}}...$
où $\overline{c_{nn}} = 5$ si $c_{nn} \neq 5$ ou $\overline{c_{nn}} = 1$ si $c_{nn} = 5$
Par construction $x \in E$ mais

1. x n'est pas un développement impropre (il n'y a que des 1 et des 5) et donc ne peut pas coïncider à un des r_i en étant le développement impropre d'un des r_i
2. diffère de chaque r_i par la décimale en position i

donc $x \notin E$
Contradiction donc cet ensemble n'est pas dénombrable

Exercice 5
Justifier que :

1. $1 = 0,9999....$
2. $0,2 = 0,19999...$
3. $0, c_1..c_k = 0, c_1...(c_k - 1)9999...$ avec $c_k \geqslant 1$

Exercice 6
Soit D l'ensemble des nombres de [0,1[de la forme $\Sigma_{k=1}^{k=m} \frac{c_k}{2^k}$ où $c_k = 0$ ou 1 (c'est une somme finie de puissances de $\frac{1}{2}$
A tout nombre d de D on lui associe son développement dyadique $c_1c_2....c_m$

1. Montrer que chaque nombre de D est de la forme $\frac{a}{b}$
2. En déduire que D est dénombrable

Exercice 7
Justifier que si A et B sont dénombrables alors $A \cup B$ est dénombrable
On rappelle qu'un nombre réel est dit irrationnel s'il n'est pas rationnel
Pourquoi l'ensemble des irrationnels de [0, 1[n'est pas dénombrable ?

Exercice 8
Soit F l'ensemble des nombres de [0,1[de la forme $\Sigma_{k=1}^{+\infty} \frac{c_k}{2^k}$ où $c_k = 0$ ou 1 (c'est une somme finie ou infinie de puissances de $\frac{1}{2}$

1. Justifier que $\lim_{n\to+\infty} \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^n} = 1$
2. En déduire que tout nombre de D a deux sommes
   $\Sigma_{k=1}^{k=m} \frac{c_k}{2^k}$ avec $c_m = 1$ (somme propre)
   ou
   $\Sigma_{k=1}^{k=m} \frac{c_k}{2^k} + \Sigma_{k=m+1}^{+\infty} \frac{1}{2^k}$ avec $c_m = 0$ (somme impropre)
   A partir de maintenant on considère uniquement les sommes propres dans F
3. Pourquoi le procédé diagonal de Cantor ne peut pas être utilisé pour montrer que F n'est pas dénombrable ?
4. Soit la fonction de codage c de [0,1[dans F qui à tout réel x de [0,1[associe la somme propre $\Sigma_{k=1}^{+\infty} \frac{c_k}{2^k}$
   Montrer que c est définie puis injective
5. En déduire que F n'est pas dénombrable

5 Un problème de décision célèbre : le problème de l'arrêt

Le problème de l'arrêt est défini par :
Les instances E sont toutes les fonctions Python f ayant un argument x
Les instances **positives** sont toutes les fonctions Python f ayant un argument x, qui s'arrêtent au bout d'un certain moment.
Ce qu'il faut comprendre ici c'est que même si nous nous efforçons de programmer correctement à priori un programme Python quelconque peut ne pas s'arrêter

Théorème 3 *(Turing 1936) Le problème de l'arrêt est indécidable*

Preuve
On va faire une démonstration par l'absurde utilisant le procédé diagonal de Cantor
Supposons (*) qu'il existe un programme Python `arret(f,x)` *qui renvoie Vrai si $f(x)$ est un programme Python qui s'arrête avec f un programme Python quelconque et x un argument quelconque et Faux si $f(x)$ ne s'arrête pas*
Or puisque l'argument x est quelconque et puisqu'un programme peut être vu comme une donnée on peut remplacer x par f
On construit alors la fonction Python `diagonal(f)` ainsi

```
def diagonal(f):
    if arret(f,f):
        while True:
            continue
    else:
        return True
```

Ensuite on exécute `diagonal(diagonal)`
Que se passe-t-il ?
D'après notre hypothèse `arret(diagonal,diagonal)` doit renvoyer soit Vrai soit Faux

Si c'est Vrai alors après le if la fonction `diagonal(diagonal)` "entre dans une boucle infinie" **donc ne s'arrête pas**
Si c'est Faux d'après (*) `diagonal(diagonal)` ne s'arrête pas donc après le else on renvoie Vrai et la fonction `diagonal(diagonal)` **s'arrête**
Il y a contradiction dans tous les cas
On rejette donc l'hypothèse autrement dit le problème de l'arrêt est indécidable

6 Machine de Turing

La machine de Turing est un outil théorique apparu en 1936 pour donner un sens à "calculable"
Elle est constituée des éléments suivants :

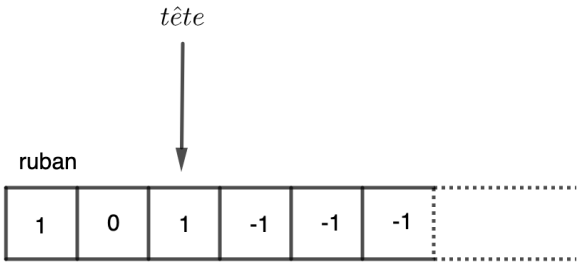
1. Un **ruban** infini , constitué d'une infinité de cases pouvant contenir les symboles 0,1 ou -1 (représentant la case vide)

   Ce ruban correspond à la mémoire d'un ordinateur

   On peut implémenter le ruban par une liste Python

2. Une **tête de lecture-écriture** positionnée à tout moment sur une des cases du ruban.

   La tête peut se déplacer d'une case vers la droite ou vers la gauche



3. Une **table de transitions** permettant de passer d'un couple

   (état actuel ,symbole lu) à un triplet (état suivant,symbole écrit,déplacement de la tête)

   Les états sont codés par les entiers

   Il y a un **état initial 0** et un **état final** n le plus grand des entiers codant les états

   A l'état initial la tête se trouve sur la case la plus à gauche du ruban

4. On peut matérialiser une machine de Turing par un graphe particulier appelé **automate fini** où

   (a) **les sommets sont les états**

   (b) **Les arcs partent d'un état vers un autre état et sont étiquetés par un triplet (symbole lu, symbole écrit, décalage)**

   Pour les décalages :

   (a) aller vers la droite est codé par D

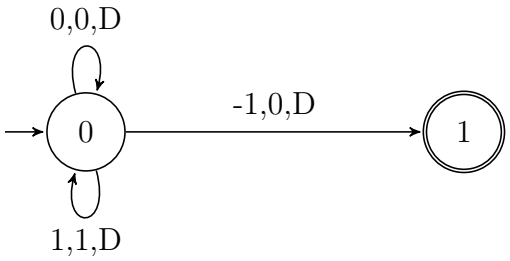
   (b) aller vers la gauche par G

Voici un exemple d'automate représentant une machine de Turing qui multiplie un entier codé en binaire par 2

(a) Au départ la tête de lecture se trouve sur le premier symbole le plus à gauche
Quelque soit le bit lu, la machine recopie le même bit et se décale vers la droite, c'est le sens de la transition 0,b -> 0,b,D

(b) A un moment la tête de lecture va lire le symbole vide -1 dans ce cas :
La machine passe de l'état 0 à l'état 1 et la tête de lecture écrit 0 à la place de -1 et il y a décalage vers la droite
0,-1 -> 1,0,D

(c) L'état 1 est final , la machine s'arrête

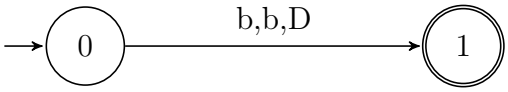


Exercice 9
Construire l'automate d'une machine de Turing effectuant la division par 2 d'une entier codé en binaire

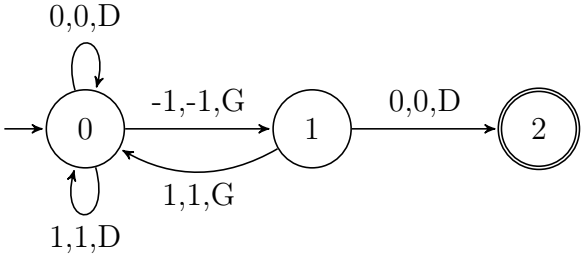
6.1 Acceptation

Définition 7 *Un mot m (suite finie de 0 et 1) placé au début du ruban est* ***accepté*** *par une machine de Turing M si on peut passer de l'état initial à l'état final par une suite de transitions de M*
L'ensemble des mots acceptés par une machine de Turing M est le langage accepté par M, noté L(M)

Par exemple la machine de Turing suivante accepte tous les mots finis



En effet quelque soit le premier bit b lu le plus à gauche elle le recopie et va dans un état final et la machine accepte le mot sans le traiter !
Par exemple **la machine de Turing suivante accepte les nombres pairs**

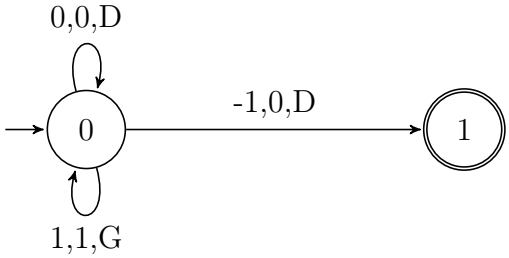


En effet lorsque la tête lit le premier caractère blanc -1, elle recopie le caractère -1 et va dans l'état 1 et revient sur le symbole 0, 1 le plus à droite

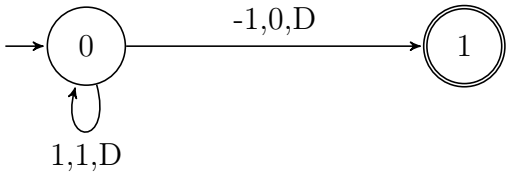
Si ce caractère est 0 alors le nombre est pair et donc la prochaine transition est d'aller dans l'état final 2
Si ce caractère est 1 alors le nombre est impair et la prochaine transition est d'aller dans l'état 0 tout en se décalant de 1 vers la droite donc la tête revient sur le premier caractère vide -1 et donc il va y avoir une boucle infinie !
Les nombres impairs ne sont donc pas acceptés.

6.2 Blocage

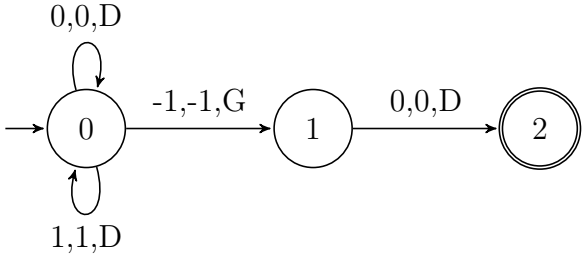
Une machine de Turing est un concept général donc dans l'ensemble des machines de Turing certaines peuvent se **bloquer** par exemple



En effet si le premier symbole le plus à gauche à l'état initial est 1 dans ce cas la tête de lecture se décale vers la gauche ce qui est impossible à l'état initial on dit alors qu'il y a blocage, ce qui est différent de l' état final qui est un arrêt "normal"
Voici un autre exemple de blocage



Cette fois ci si la machine est dans l'état 0 et lit un 0 alors il n'y a pas de transition possible prévu et il y a un blocage
On peut maintenant définir une machine de Turing qui accepte les nombres pairs sans boucle infini mais qui bloque sur un nombre impair, autrement dit les nombres impairs ne sont pas acceptés



Exercice 10
Construire l'automate d'une machine de Turing qui accepte les puissances de 2
Exercice 11 Construire l'automate d'une machine de Turing qui accepte les mots ne contenant que des 1
Exercice 12 Construire l'automate d'une machine de Turing qui accepte les mots contenant autant de 0 que des 1

7 Codage des machines de Turing

De la même manière qu'un programme Python peut être considéré comme une donnée, une machine de Turing peut être considéré comme un mot de $\{0,1\}^*$ l'ensemble de tous les mots écrits avec des 0 et des 1

Définition 8 *Soit M une machine de Turing et w un mot on note*

1. *<M> le codage de cette machine par un mot de* $\{0,1\}^*$
2. *<M,w> le codage de la paire (M,w) de la manière suivante <M>sw où s est un symbole n'apparaissant ni dans <M> ni dans w*
3. $L_U = \{< M, w > | M \text{ quelconque}, w \in L(M)\}$ *le langage des mots obtenus en concaténant à la suite du codage de M un mot quelconque accepté par M pour M une machine de Turing quelconque*
4. *Une machine de Turing* M_U *telle que* $L(M_U) = L_U$ *est appelée machine universelle. C'est une machine capable de simuler n'importe quelle machine de Turing fournie en entrée*

Théorème 4 *Il est possible de construire une machine de Turing universelle*

Exercice 13
Comment pourrait coder avec des 0 et des 1 une machine de Turing ?
Exercice 14
Justifier que $\{0,1\}^*$ est dénombrable

8 Langages décidables

Définition 9 *On note* $\{0,1\}^*$ *l'ensemble de tous les mots écrits avec des 0 et des 1*
On appelle ***langage*** *n'importe quelle partie de* $\{0,1\}^*$

Définition 10 *Un langage L est* ***semi-décidable*** *s'il est accepté par une machine de Turing M*

Théorème 5 *Il existe des langages qui ne sont pas semi-décidables*

Preuve (Procédé diagonal de Cantor voir exercice)

Définition 11 *Un langage L est* ***décidable*** *s'il est accepté par une machine de Turing M* ***sans calcul infini***

Théorème 6 L_U *n'est pas décidable*

Exercice 15
On définit dans cet exercice le langage diagonal ainsi
$\{0,1\}^*$ l'ensemble de tous les mots écrits avec des 0 et des 1 est dénombrable donc on peut le ranger sous la forme d'une suite m_0, m_1 etc...
On admet que l'ensemble des machines de Turing est dénombrable donc on peut le ranger sous la forme d'une suite M_0, M_1 etc...

Soit L_D le langage diagonal défini par
$L_D = \{m_i | m_i \notin L(M_i)\}$
Démontrer par l'absurde que L_D n'est pas semi-décidable
(Supposer qu'il l'est en supposant l'existence d'une machine de Turing M telle que $L(M) = L_D$, cette machine M est donc une de celles de la suite M_0, M_1 , noter la M_k puis considérer le mot m_k de même rang et discuter de l'appartenance de m_k à $L(M_k)$ et aboutir à une contradiction...)