éduSCOL
Informer et accompagner les professionnels de l'éducation

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ÉDUCATION NATIONALE ET DE LA JEUNESSE

VOIE GÉNÉRALE

2DE | 1RE | TLE

Numérique et Sciences informatiques

ENSEIGNEMENT
SPÉCIALITÉ

CALCULABILITÉ ET DÉCIDABILITÉ

- Histoire de l'informatique
- Structures de données
- Bases de données
- Architectures matérielles et systèmes d'exploitation
- Langages et programmation
- Algorithmique

1. La thèse de Church

Dans le cadre de la spécialité NSI, nous utilisons le langage Python, même si nous savons qu'il existe bien d'autres langages de programmation, comme JavaScript, C, Perl, Java, Fortran, etc.
Un même algorithme peut être traduit en un programme dans l'un ou l'autre des langages de notre choix, ce que met en évidence le document ressource Diversité et unité des langages de programmation
Cela revient à dire que tous les langages de programmation usuels ont la même puissance d'expression algorithmique, que chacun permet de programmer les mêmes fonctions que tous les autres. Cela définit ainsi *un modèle de calcul* : on dira qu'une fonction est calculable si elle peut être programmée dans l'un ou l'autre des langages de programmation usuels.
Dans ce document, nous utiliserons le langage Python comme témoin : une fonction est calculable si on peut la programmer en Python.

Il existe d'autres modèles de calcul, comme le λ-calcul, les fonctions récursives, les machines de Turing, que nous ne développerons pas ici, et qui ne font pas partie des attendus du programme.
La *thèse de Church* [1] postule que tous ces modèles de calcul sont équivalents : une fonction calculable pour un modèle l'est pour un autre. Cela nous permet d'utiliser le modèle des fonctions programmables en Python sans perdre de généralité.

2. Fixons le vocabulaire

2.1. Problèmes, instances, prédicats

La notion de problème n'est pas si simple à définir. Par exemple : le problème de savoir si un nombre est premier. De quoi s'agit-il précisément ?
Si j'ai calculé une table des nombres premiers jusqu'à un million, et si on me donne le nombre 870 671, il me suffit de consulter la table pour constater qu'il est effectivement premier. Mais bien entendu cette méthode a ses limites.
Si on note $\mathbf{P}$ l'ensemble des nombres premiers, on voudrait pouvoir dire de **tout** entier naturel n si oui ou non $n \in \mathbf{P}$.
Autrement dit, on voudrait avoir accès à une fonction $f : \mathbb{N} \to \{\text{Vrai,Faux}\}$ telle que $f(n)$ est Vrai si et seulement si $n \in \mathbf{P}$.

1. Alonzo Church est un logicien américain, né en 1903 et mort en 1995. Trois de ses étudiants sont également à l'origine des fondements théoriques de l'informatique : Stephen Kleene, John Barkley Rosser et Alan Turing.

Dire que le problème est décidable, c'est simplement dire qu'il existe un programme Python permettant de calculer $f(n)$ pour tout entier n.

[définition d'un prédicat] Un prédicat est une fonction qui prend des valeurs booléennes, c'est-à-dire dans l'ensemble $\{\text{Vrai,Faux}\}$.

[définition d'un problème de décision] Un problème de décision est la donnée d'un ensemble E, que l'on appelle l'ensemble des instances et d'une partie E^+ de E, que l'on appelle l'ensemble des instances positives.

Par exemple le problème de la primalité peut être défini par l'ensemble d'instances $E = \mathbb{N}$ et l'ensemble d'instances positives $E^+ = \mathbf{P}$.
On peut présenter un problème de décision de différentes façons.
D'abord à l'aide d'un schéma comme celui-ci, l'exemple de la primalité figurant à droite.

Nom du problème

Donnée : spécification d'une instance

Réponse : propriété portant sur une instance qui spécifie les instances positives

Primalité d'un entier

Donnée : un entier naturel

Réponse : décider si ce nombre est premier

Une autre façon de définir un problème de décision est de fournir l'ensemble des instances (ici $\mathbb{N}$) et la description du prédicat testant l'appartenance aux instances positives (ici, le prédicat est une fonction, la fonction `isprime`).

Prenons un autre exemple : colorier un graphe avec un nombre fixé de couleurs, c'est colorier ses sommets de sorte que deux sommets reliés par une arête du graphe ne sont jamais de la même couleur.
On peut alors définir :
Coloration d'un graphe

Donnée : un graphe non orienté et un nombre c de couleurs

Réponse : décider si on peut colorier ce graphe avec c couleurs

Notons qu'ici les instances sont des couples (G, c) où G est un graphe et c un entier.

Nos exemples peuvent être plus exotiques, comme par exemple :
Programme Python bien formé

Donnée : une chaîne de caractères

Réponse : décider si le programme constitué de la chaîne de caractère donnée est un programme Python bien formé, c'est-à-dire sans erreur de syntaxe

2.2. Problèmes de décision décidables

Un problème de décision est décidable si on peut écrire un programme en Python qui permette de calculer le prédicat associé au problème.
Attention ! On ne s'intéresse ici pas du tout aux questions de complexité : que l'algorithme de calcul du prédicat soit coûteux ou pas n'intervient pas, c'est l'existence d'un tel algorithme qui permet de conclure, et pas ses performances.

Par exemple, on peut écrire un programme qui teste si un nombre est premier, comme celui-ci :

```
def isprime(n):
  # on élimine les cas triviaux
  if n < 2 : return False
  if n % 2 == 0 : return False
  # on teste tous les diviseurs possibles (il y a beaucoup plus efficace !)
  for d in range(3,n,2):
    if n % d == 0 : return False
  return True
```

On aura remarqué que le prédicat renvoie une réponse (`True` ou `False`) pour n'importe quelle donnée. En particulier, il ne sombre jamais dans une boucle infinie : il termine toujours. On peut donc affirmer que *le problème de la primalité est décidable*.

Le problème de la coloration des graphes pose un problème de progammation plus difficile, et nous ne détaillerons pas ici un programme Python pour le prédicat associé : contentons nous de dire qu'il suffit de balayer de façon systématique les coloriages de tous les sommets avec les couleurs disponibles, et, pour chaque coloriage, de vérifier, par un balayage systématique des arètes, si deux sommets reliés sont coloriés de la même façon. Mais, pour un graphe à n sommets et p arètes et c couleurs, cela va coûter cher : un nombre d'opérations de l'ordre de $c^n \times p$.
Mais on ne s'intéresse pas du tout ici à la complexité, et l'important est qu'on puisse affirmer que *le problème de la coloration des graphes est décidable*.

Notons au passage que savoir résoudre ce problème de décision permet de résoudre un autre type de problème : déterminer le nombre minimal de couleurs dont il faut disposer pour pouvoir colorer un graphe. Si on a écrit un programme qui résout le problème de décision de la coloration d'un graphe, on peut en déduire facilement :

```
def nbMinimalCouleurs(graphe):
  c = 1
  while not colorationPossible(graphe,c):
    c += 1
  return c
```

On est assuré de sortir de la boucle : en effet si le graphe possède n sommets, on peut le colorier à coup sûr avec n couleurs, il suffit de colorier chaque sommet par une couleur unique. Ainsi peut on garantir que la boucle terminera pour une valeur de `c` au plus égale à n.

Le troisième problème que nous avons présenté est lui aussi décidable : la preuve peut se limiter à rappeler qu'il existe un interpréteur du langage Python, qui en particulier est capable de décider si oui ou non une chaîne de caractères représente un programme Python syntaxiquement correct. Cet interpréteur est le plus souvent écrit dans le langage C, mais la thèse de Church nous indique qu'on pourrait aussi le programmer en Python.

Remarque importante : la question de la calculabilité d'une fonction ou de la décidabilité d'un problème est une question totalement différente de celle de la complexité des algorithmes concernés.

3. Le problème de l'arrêt

3.1. Les fonctions calculables

Une fonction $f : E \to V$ (où V n'est pas nécessairement l'ensemble $\{\text{Vrai,Faux}\}$) est dite calculable s'il existe un programme Python qui prend en entrée une valeur quelconque $x \in E$ et renvoie toujours la valeur $f(x)$ (ce qui signifie en particulier que le programme termine pour toute entrée x et renvoie $f(x)$).
On a déjà dit qu'il était équivalent de dire qu'un problème de décision est décidable ou que le prédicat associé est calculable.

3.2. Les programmes comme données

Un programme écrit en Python n'est autre qu'une chaîne de caractères : c'est le texte même du programme.
Bien sûr, pour un même programme, en ajoutant des espaces à la fin d'une ligne, ou entre deux mots, on obtient un autre programme qui calcule exactement les mêmes choses. Mais peu importe.
Cela permet de considérer des fonctions qui prennent en entrée des programmes.

3.3. Le problème de l'arrêt est indécidable

Voici tout d'abord la définition du problème :
Problème de l'arrêt

Donnée : le couple constitué d'un programme Python π pour une fonction f qui prend un argument, et d'une valeur x pour cet argument

Réponse : décider si le calcul de $f(x)$ par le programme π termine ou ne termine pas

Une instance est donc un couple (π, x) formé du texte π d'un programme Python pour une fonction f et d'une valeur x qui a vocation à être fournie en argument à f.
C'est Alan Turing qui, en 1936, dans le cadre de son fameux article *« On Computable Numbers, with an Application to the Entscheidungsproblem »*, démontre le théorème de l'indécidabilité du problème de l'arrêt qui s'énonce ainsi :

> Le problème de l'arrêt est indécidable.

Ce résultat met définitivement fin à tout espoir d'automatiser de façon algorithmique le calcul de n'importe quelle fonction. C'est ce qui fait son importance, à la fois historique, scientifique et philosophique.

Démonstration du théorème

La démonstration classique que nous présentons ici est une preuve par l'absurde. On suppose donc qu'il existe un programme Python qui décide du problème de l'arrêt :

```
def testARRET(programme,x):
  ...
  if ...:
  # si le programme s'arrête sur l'entrée x
    return True
  else:
    return False
```

On définit alors le programme suivant :

```
1 def testSurSoi(programme):
2   if testARRET(programme,programme):
3     while True:
4       continue
5   else:
6     return True
```

Les lignes 3–4 forment une boucle infinie.
Que se passe-t-il lors de l'appel `testSurSoi(testSurSoi)` ?

- ou bien cet appel termine, ce qui signifie qu'on n'est pas entré dans la boucle infinie des lignes 3–4. Mais alors cela signifie que l'appel `testARRET(testSurSoi,testSurSoi)` a renvoyé la valeur `False`, ce qui ne peut arriver que si l'appel `testSurSoi(testSurSoi)` ne termine pas : c'est contradictoire ;
- ou bien cet appel ne termine pas, ce qui signifie qu'on est entré dans la boucle.
  Cela ne peut arriver que si l'appel `testSurSoi(testSurSoi)` termine : c'est contradictoire.

On aboutit dans tous les cas à une contradiction, ce qui achève notre démonstration par l'absurde !

Rmarque : on peut regarder une vidéo très amusante qui illustre cette célèbre démonstration ici :
https://www.youtube.com/watch?v=92WHN-pAFCs

4. D'autres problèmes indécidables

On présente ici quelques problèmes indécidables célèbres, sans démonstration (elles sont hors de portée d'un élève de lycée), à titre culturel.
Rien de ce qui suit n'est au programme de NSI. Cette partie est simplement à l'intention du professeur curieux d'en apprendre un peu plus.

4.1. Notion de réduction

On peut présenter une méthode permettant de ramener la décidabilité d'un problème de décision à la décidabilité d'un autre : la réduction.

Soit donc deux problèmes de décision A et B dont les ensembles d'instances respectifs sont E_A et E_B. Les instances positives forment les ensembles E_A^+ et E_B^+.

Soit f_A le prédicat associé au problème A, tel que pour toute instance $x \in E_A$, on a $f_A(x) = \begin{cases} \texttt{True}, & \text{si } x \in E_A^+ \\ \texttt{False}, & \text{si } x \notin E_A^+ \end{cases}$

Une réduction de A vers B est une fonction φ calculable (c'est-à-dire qu'on peut l'écrire par un programme Python) qui va de E_A dans E_B, telle que pour toute instance $x \in E_A$:

$$\varphi(x) \in E_B^+ \Leftrightarrow x \in E_A^+.$$

Ainsi si je sais programmer en Python le prédicat f_B associé au problème B, c'est-à-dire si B est décidable, je peux programmer un prédicat f_A associé au probème A : il suffit d'écrire

```
def f_A(x):
  # x est une instance du problème A
  y = phi(x)  # y est une instance du problèem B
  return f_B(y)
```

Avec ces notations, `f_B(y)` vaut `True` si et seulement si `y` est une instance positive du problème B. La définition de la réduction garantit que c'est équivalent à dire que `x` est une instance positive du problème A : on a bien programmé un prédicat `f_A` associé au problème A qui est donc également décidable.

Autrement dit : si on dispose d'une réduction de A à B, et si B est décidable, alors A est décidable. On peut reformuler cela en disant que le problème A est plus facile que le problème B.

Cet outil de la réduction permet donc de ramener la question de la décidabilité d'un problème à la recherche d'une réduction de ce problème à un problème décidable déjà connu.

Inversement si on dispose d'une réduction d'un problème A non décidable à un autre problème B, on peut en déduire que B n'est pas décidable. (Les mathématiciens parlent de raisonnement par contraposée.)

La réduction est au cœur de nombreuses démonstrations d'indécidabilité.

4.2. Le problème de la correspondance de Post

Né en Pologne en 1897, Emil Post émigre pour les États-Unis en 1904, où il mène de brillantes études à l'université de Columbia. Malgré une maladie très handicapante, il conduit des recherches en logique au City College de New York qui l'amènent à définir en 1946 le problème qui porte aujourd'hui son nom, dont il démontre l'indécidabilité. Il meurt en 1954.

Description du problème

Le problème de Post est un problème de décision.

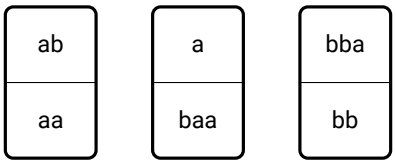
On se fixe un alphabet.

On construit des « dominos » sur chacun desquels on écrit deux mots. Une instance du problème est ainsi une suite finie $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ de couples de mots.

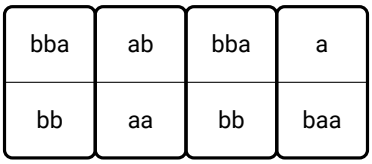
Le problème consiste à savoir si on peut aligner des dominos de sorte que les mots résultant sur chaque moitié sont les mêmes : existe-t-il une suite $k_1, k_2, \ldots, k_p$ d'indices à prendre dans $[\![1, n]\!]$ (on peut choisir plusieurs fois le même indice) telle que les mots $u_{k_1} u_{k_2} \ldots u_{k_p}$ et $v_{k_1} v_{k_2} \ldots v_{k_p}$ soient identiques.

Un exemple

On considère le jeu des 3 dominos suivants :



Cet ensemble de dominos constitue une instance du problème de Post.
Il s'agit d'une instance positive puisqu'on peut obtenir une correspondance (on lit le même mot bbaabbbaa en haut et en bas des dominos) :



Il existe des instances négatives : par exemple le jeu consitué de l'unique domino où on a écrit les mots a et aa, ne pourra évidemment fournir aucune correspondance.

Le problème de correspondance de Post est indécidable

La preuve classique utilise les machines de Turing.
On ne peut pas trouver d'algorithme général qui dise si on peut ou pas obtenir une correspondance à partir d'un jeu de dominos.
Notons que ce résultat n'a rien d'évident *a priori*. Beaucoup de personnes sont persuadées du contraire avant d'essayer … et de ne pas trouver !

4.3. Le dixième problème de Hilbert

Lors du deuxième congrès international des mathématiciens, tenu à Paris en août 1900, le grand mathématicien allemand David Hilbert (1862–1943) a présenté en août 1900, à Paris, une liste de 23 problèmes à ses collègues. Ces problèmes interrogent en particulier les fondements des mathématiques et de la logique, et seule une moitié d'entre eux ont été résolus depuis.
Le dixième problème de Hilbert consiste en la décidabilité d'un problème de décision.
Il s'agit de déterminer s'il existe un algorithme qui décide si un polynôme à plusieurs variables dont tous les coefficients sont entiers admet ou non des solutions entières.
Par exemple, l'équation $3x^2 - 2xy + 4y^2z = 31$ admet comme solution $x = 1$, $y = 2$ et $z = 2$, alors que l'équation $3x^2 + 6xy + y^2 + z^2 + 1 = 0$ n'admet aucune solution entière.
Il a fallu attendre 1970 pour que le mathématicien russe Iouri Vladimirovitch Matiassevitch démontre, à l'âge de 23 ans, que ce problème est indécidable.

Des cas simples

Il n'est pas trop difficile de décider si une équation polynômiale à une seule variable entière x a ou non des solutions entières.
Le professeur de mathématiques pourra facilement donner une preuve générale, mais considérons un exemple pour s'en convaincre.
Soit l'équation $3x^5 - 2x^4 + 7x^3 + 22x^2 - 31x + 7 = 0$.
Comme le coefficient constant n'est pas nul, $x = 0$ n'est pas solution. Cherchons s'il peut y avoir des solutions entières non nulles.
En isolant le terme de degré nul, on obtient une équation équivalente $-3x^5 + 2x^4 - 7x^3 - 22x^2 + 31x = 7$, qu'on peut réécrire sous la forme $x(-3x^4 + 2x^3 - 7x^2 - 22x + 31) = 7$, ce qui prouve qu'une solution entière x non nulle doit être un diviseur de 7 : il suffit donc de tester quelques entiers parmi $-7, -6, -5, \ldots, -1, 0, 1, \ldots, 7$. (Comme 7 est premier, on ne teste ici que $-7, -1, 1$ et 7.)
Il n'est donc besoin que d'une simple boucle pour programmer le prédicat.
Donc le problème de décider si un polynôme à coefficients entiers et à une seule variable admet ou non des solutions entières est décidable.

Ça se complique

Dès que l'équation polynomiale considérée possède deux variables, tout se complique. Par exemple il est vrai que l'équation $x^2 - 991y^2 - 1 = 0$ a des solutions entières, mais elles ne sont pas faciles à trouver : la plus petite d'entre elles est $x = 379516400906811930638014896080$ et $y = 12055735790331359447442538767$.
On sait démontrer que le problème de savoir si une équation polynomiale à coefficients entiers à p inconnues a des solutions entières est décidable pour $p = 1$ (on l'a vu plus haut), on sait aussi qu'il est indécidable pour $p \geqslant 9$. Mais on ne sait pas si le problème est décidable pour $p \in [\![2, 8]\!]$.
Il y a des sous-problèmes décidables : si on se restreint à des équations où le degré total est inferieur ou égal à 2, le problème est décidable ; le problème devient indécidable pour des degrés inférieurs ou égaux à 4. En revanche, on ne sait dire si le dixième problème de Hilbert restreint aux polynômes de degré 3 est décidable ou non décidable !