GRANDS CONCEPTS DE L'INFORMATIQUE

OBJECTIFS :

1) Rattraper le temps perdu (un siecle depuis ces grands theoremes)
2) Satisfaction/fierté personnelle
3) Poser des limites théoriques de ce qu'on peut faire en informatique

PLAN DU COURS :

Introduction
A) Indénombrabilité (G. Cantor, 1891)
+ notion de finitude/dénombrabilité.
B) Indécidabilité (A. Turing, 1936)
Idée qu'il n'existe pas toujours d'algorithme pour tester une propriété sur toutes les instances possibles (cf. machine de Turing).
C) Incomplétude des systèmes formels (K. Gödel, 1931)

Remarques (avant de commencer) :

- Les propriétés que vous avez l'habitude de prouver pour un algorithme sont :
  - La terminaison (arrêt en un nombre fini d'étapes)
  - La correction
  - La complexité

  Les deux premieres sont plus fondamentales : on requiert d'un algorithme que toute execution finisse toujours en un nombre fini d'etapes et qu'elle fournisse le resultat attendu (correct)

- Hiérarchie des limites :
  Indécidable ⇒on n'y pense même pas
  NP difficile ⇒seulement pour des instances $n \leq 50$
  $O(n^2)$ ⇒gros jeux de données ▮

INTRODUCTION :

En mathématiques, toute propriété vraie est-elle prouvable ?
Cette question, à la limite de la métaphysique, est plutôt naturelle.

En effet, on a l'habitude d'énoncés de la forme :
- Montrer P1 : $\forall x, Q(x)$
- Montrer P2 : $\exists x, Q(x)$

Or, si P1 n'est pas vraie (i.e. $\exists x, \neg Q(x)$) on a une erreur d'énoncé puisqu'il est, par définition, impossible de montrer que P1 est vraie. À l'inverse, si P1 est vraie, on doit pouvoir la prouver (rien ne garantit que la personne qui s'y penche n'y arrive…).

Le problème réside alors dans le fait qu'il y a des choses qu'on pense vraies mais qu'on ne sait pas prouver. C'est ce qu'on appelle plus communément des conjectures, en voici quelques exemples :

- Conjecture de C. Goldbach :

$\forall n \in \mathbb{N}, n \geq 3, \quad n = p + q$ avec p et q des nombres premiers

- Les nombres premiers jumeaux :

Il existe une infinité de couple (p,q) avec p et q premiers tels que $q - p = 2$

- Suite de Syracuse d'un nombre entier N :

Il s'agit d'une suite à valeurs dans $\mathbb{N}$ définie par recurrence de la facon suivante:

- $U_0 \in \mathbb{N}^*$ et
- pour tout $n \in \mathbb{N}$:

si U_n pair, alors $U_{n+1} = U_n / 2$

sinon, $U_{n+1} = (3U_n + 1)/ 2$

La conjecture liée :

$\forall U_0 \in \mathbb{N}^*$, la suite de Syracuse tombe sur 1 en un nombre fini d'étapes.

Ainsi, le programme qui implémente cette suite peut s'écrire :

Tant que $n \neq 1$ faire :

si $n \% 2 = 0$ alors $n \leftarrow n / 2$

sinon $n \leftarrow (3n + 1) / 2$

Fin tant que

On ne sait pas prouver que ce programme termine toujours, quelle que soit la valeur initiale de n dans $\mathbb{N}^*$. Du coup, on en est pas sur…

- $P \neq NP$ (ou $P = NP$) :

Considérée comme l'une des conjectures les plus importantes des mathématiques et de l'informatique théorique, elle fait l'objet d'un des sept problèmes du prix du millénaire sélectionnés par l'institut mathématiques Clay en 2000 (le but étant de démontrer que P = NP, P ≠ NP ou même que ce n'est pas démontrable). On peut le résumer sommairement comme : « Ce que nous pouvons trouver rapidement lorsque nous avons de la chance, peut-il être trouvé aussi vite par un calcul intelligent ? » ou encore « Tout ce que l'on peut vérifier facilement, peut-il être découvert aisément ? »

Plus précisément, on pose P la classe de complexité des problèmes décisionnels (ceux dont la réponse est soit oui soit non) admettant un algorithme de résolution en temps polynomial sur une machine de Turing.
On pose NP la classe de complexité des problèmes de la décision dont la vérification du résultat (connu donc) se fait en temps polynomial.
Le but étant donc de savoir si ces classes de complexités peuvent être équivalentes.

La vérification de la validité d'une démonstration de longueur N se fait algorithmiquement en un temps polynomial (par rapport à N). En revanche, trouver une démonstration est un problème de classe NP dont la résolution se fait par force brute avec un algorithme de complexité exponentielle.
Ainsi, prouver que P = NP entrainerait que la recherche par force brute n'est pas la plus adaptée et qu'il est possible de trouver un algorithme qui travaille en un temps polynomial pour trouver ladite démonstration. Un grand nombre de problèmes mathématiques pourraient alors être résolus (dont certains autres problèmes du millénaire).
La preuve de P ≠ NP serait aussi particulièrement pertinente : on aurait la confirmation qu' un problème NP-Complet ne peut pas être résolu en un temps polynomial (ce qui aurait des avantages en cryptographie notamment).

Remarque : Le problème de la décision a été posé en 1928 par le mathématicien allemand David Hilbert. Il consiste à se demander s'il existe un algorithme qui, si on lui présente une question mathématique dont la réponse est « Oui » ou « Non » trouvera automatiquement et infailliblement la réponse. En 1936, A. Turing et A. Church ont démontré que dans le cas général, ce problème de la décision n'a pas de réponse (autrement dit qu'il existe toujours des questions algorithmiquement indécidables). ∎

Une question sensée découle immédiatement : comment savoir qu'une propriété est vraie si on n'en a pas la preuve ? L'idée est de montrer qu'il existe une propriété vraie telle qu'on ne peut la prouver (cf. K. Gödel en 1931 selon lequel toute théorie mathématique cohérente contient au moins un énoncé qui est vrai et qui n'admet pas de preuve).
Une idée quelque peu abusive réside dans le fait qu'une propriété est soit vraie soit fausse. En effet, on peut trouver des contre-exemples assez simples :

- "Cet énoncé est faux" :
  Si on considère que l'énoncé est vrai, la propriété nous dit que l'énoncé est faux : contradiction.
  Si on considère que l'énoncé est faux, la propriété nous dit que l'énoncé est vrai : contradiction
  On ne peut donc pas attribuer de valeur logique «vrai » ou « faux » a cet enonce.

- “L’énoncé est vrai” :
  Si on considère que l’énoncé est vrai, la propriété nous dit que l’énoncé est vrai, pas de probleme d’incoherence.
  En revanche, si on considère que l’énoncé est faux, la propriété nous dit que l’énoncé est faux ce qui ne pose pas de probleme non plus.
  On peut donc decider que l’enonce est vrai ou est faux sans avoir de probleme de contradiction, ce qui est troublant.

- B. Russel et le paradoxe du barbier :
  Si dans une ville où tout le monde doit être parfaitement rasé, un barbier rase toutes les personnes qui ne se rasent pas elles-mêmes et seulement celles-ci, se rase-t-il lui-même ?
  Si oui, on a un problème puisque le barbier ne rase pas les personnes se rasant elles-mêmes (or il est le barbier).
  Si non, on a aussi un problème puisque que le barbier le rase (et donc il se rase lui-même).

Au regard des exemples apportés, on constate que les problèmes de vérité viennent du fait qu’on a une auto-référence dans les énoncés. Il s’agit de les éviter pour construire une théorie mathématique et ses énoncés bien formés.

Remarque : Le paradoxe du barbier est une version “imagée” du paradoxe de Russel qui concerne la théorie des ensembles (et qui a d’ailleurs joué un certain rôle dans la formalisation de celle-ci). On peut le formuler ainsi :
“ L'ensemble des *ensembles n'appartenant pas à eux-mêmes* appartient-il à lui-même ? ”
Plus formellement :
On pose E l’ensemble de tous les ensembles. Alors, si un tel ensemble existe, il est forcément élément de lui-même (ce qui ne pose pas de problème en soit).
Soit maintenant C un sous ensemble de E qui contient tous les ensembles qui n’appartiennent pas a eux-mêmes. Autrement dit : $C = \{ X \in E \mid X \notin X \}$ (où X un ensemble donc).
Alors est-ce que C appartient à lui-même ?
Etant donné que le principe dit du tiers exclus (soit $C \in C$, soit $C \notin C$) s’applique à cette théorie, vérifions les deux alternatives :
- $C \in C \Rightarrow C \notin C$ par définition de C et donc contradiction.
- $C \notin C \Rightarrow C \in C$ par définition de C et donc contradiction.

Une solution apportée a été la distinction entre ensemble et classe (E. Zermelo, 1908).
On remplace ce qu’on a appelé “ensemble” jusque-là par la notion de “classe” et on redéfinit la notion “d’ensemble” par “les classes qui peuvent être éléments d’une autre

classe". On parlera de classe propre lorsqu'une classe n'est pas un ensemble (elle n'est l'élément d'aucune autre classe).

Ainsi, si on redéfini ce qu'on a précédemment appelé C par " la classe de tous les ensembles " (qui est donc une classe propre) on évite par définition que C ne soit un ensemble et donc qu'il se contienne lui-même.

Une autre solution a été la théorie des types de Russel (1908 aussi) selon laquelle les ensembles sont de types hiérarchisés qui ne peuvent contenir que des objets (pouvant être des ensembles) de types strictement inférieurs au type de l'ensemble initial. Ceci entraîne notamment le fait qu'on ne puisse plus écrire que $X \in X$ (avec X un ensemble) et donc par la même rend nul le paradoxe. ∎

Ainsi, on constate que d'un point de vue mathématique, on ne peut pas toujours prouver nos propriétés. En revanche, en informatique, il nous est possible d'écrire un algorithme qui va, pour une instance donnée (notons x), déterminer si la propriété appliquée à ce cas précis (notons P(x)) est vraie. Malheureusement, on a déjà évoqué que même ce fait ne sera pas toujours possible (cf. Turing, 1936) …