CALCULABILITÉ − DÉCIDABILITÉ : SYNTHÈSE DE COURS

I) Introduction

1) L'informatique a-t-elle des limites ?

Dans ce chapitre, nous élargissons notre regard sur l'informatique en évoquant quelques problèmes difficiles d'informatique théorique très liés à l'histoire des mathématiques du 20ème siècle.

La question est : Y a-t-il des limites à ce qu'un algorithme peut faire ?
En effet, parmi tous les problèmes que l'on cherche à résoudre habituellement avec des ordinateurs, on trouve :

- Des problèmes pour lesquels on a trouvé des algorithmes efficaces (complexité « acceptable »)
- Des problèmes pour lesquels on a trouvé des algorithmes, mais ces derniers demandent trop de ressources pour être considérés comme satisfaisants (complexité trop importante) et on cherche des solutions plus satisfaisantes.
- Des problèmes pour lesquels les informaticiens n'ont pas encore trouvé d'algorithme et cherchent encore.

Mais il se trouve qu'il existe encore une autre catégorie de problèmes : **Ceux dont on a réussi à démontrer qu'aucun algorithme ne pouvait les résoudre !**

On sait par exemple qu'on ne trouvera jamais d'algorithme universel capable de déterminer si un ensemble quelconque de polygones peut paver le plan entier. Pas la peine de chercher : un mathématicien a démontré en 1966 que ça n'existe pas et ça n'existera jamais !

2) Un peu d'histoire des mathématiques

En mathématiques, on se dit en général que si une propriété est vraie, alors il y a bien un mathématicien qui un jour réussira à la démontrer. Mais est-ce si sûr ?

C'est en se posant ce genre de questions, que dans les années 1900, David Hilbert et d'autres mathématiciens cherchent à montrer que si une affirmation mathématique est vraie, alors elle est démontrable.

Mais en 1931, Kurt Gödel met en évidence avec ses « théorèmes d'incomplétude » que dans le domaine du calcul arithmétique, il existera toujours des énoncés vrais, et pourtant indémontrables !

En 1936, Alan Turing cherche à définir ce qui est calculable. Alors que les ordinateurs n'existent pas encore, il imagine une machine abstraite formée d'un ruban (= mémoire et entrées sorties) sur lequel on peut lire ou écrire à partir des instructions précisées sur une table de transition (= programme). Cette machine qu'il n'a jamais cherché à construire et que l'on appelle « machine de Turing » lui permet d'établir que tout algorithme de calcul peut être implémenté par une machine de Turing et vice-versa. Ensuite il met en évidence ce que l'on appellera le « problème de l'arrêt » : Il ne peut exister de machine de Turing (ou d'algorithme) capable de décider si n'importe quelle machine de Turing (ou algorithme) s'arrêtera de façon certaine.

Il y a donc des choses qui ne sont pas « calculables », c'est à dire des choses pour lesquelles il n'existe pas d'algorithme de calcul.

A noter que Kurt Gödel avec les fonctions récursives et Alonso Church avec le lambda calcul démontreront la même chose chacun de leur côté et avec des approches complètement différentes (mais qui se révéleront équivalentes).

Une conséquence importante de ces travaux pour aujourd'hui est que l'on a pu montrer depuis que **tous les langages de programmation usuels ont la même puissance algorithmique** : chacun permet de programmer les mêmes fonctions que tous les autres.

3) Un peu de vocabulaire

- Calculer, c'est appliquer un algorithme de calcul. Pour les mathématiciens il y a équivalence entre ce qui est « calculable » et ce pour quoi on peut établir un algorithme de calcul.
  Exemple : La fonction *f* qui a *x* fait correspondre $2x + 1$ est calculable et l'algorithme est simple : On prend *x*, on le multiplie par 2 et on ajoute 1.
  En d'autres termes : une fonction est calculable si elle est programmable en Python ou tout autre langage de programmation.
- En mathématiques, un « problème de décision » est une question dont la réponse est soit oui, soit non.
  Exemple : « 101 est-il premier ? »
- Un problème de décision est dit « décidable » s'il existe un algorithme qui permet de répondre.
  Exemple : « 101 est-il premier ? » est décidable car on a un algorithme pour répondre à la question : il suffit de chercher à diviser 101 par tous les entiers compris entre 2 et 100 pour pouvoir répondre oui.
  Exemple : « 102 est-il premier ? » est décidable car il suffit d'appliquer le même algorithme pour pouvoir répondre non.
- Un problème de décision est dit « indécidable » s'il n'existe pas d'algorithme permettant de répondre. Ce n'est pas que l'on n'a pas encore trouvé l'algorithme, c'est que l'on a montré qu'un tel algorithme ne peut pas exister !

II) Le problème de l'arrêt

Dans la pratique, il y a assez peu de problèmes dont on a montré qu'ils étaient indécidables et ils sont pour la plupart hors de notre portée ! Nous allons toutefois nous intéresser au plus célèbre d'entre eux : le fameux « problème de l'arrêt » proposé par Alan Turing.

1) Préliminaire : Un programme peut aussi être une donnée !

On a l'habitude de distinguer les programmes et les données sur lesquelles on les fait travailler. En Python, par exemple, on écrit les programmes dans des fonctions et on stocke les données dans des variables.
Dans la réalité, un programme peut aussi être une donnée pour un autre programme : L'interpréteur Python par exemple est un programme qui prend pour données les scripts Python (programmes) que l'on écrit.

2) Problème de l'arrêt

Supposons qu'il existe une fonction **`arrêt`** capable de déterminer pour n'importe quel algorithme s'il s'arrête toujours ou non. Alan Turing faisait son raisonnement en utilisant des machines de Turing, nous le ferons en utilisant le langage Python :

```
def arrêt(prog, x):
    """Fonction qui renvoie Vrai si prog(x) termine, et False sinon."""
    # Désolé, je n'ai pas encore écrit le code !
```

Définissons la fonction paradoxe suivante :

```
def paradoxe(prog):
    if arrêt(prog, prog):
        while True:
            pass
    else:
        return False
```

Quel est le résultat de l'appel `paradoxe(paradoxe)` ?

Il n'y a que deux cas de figure :

- 1er cas : Si `paradoxe(paradoxe)` termine, alors `arrêt(paradoxe, paradoxe)` est vrai et `paradoxe(paradoxe)` rentre dans une boucle infinie et donc ne termine pas...
- 2ème cas : Si `paradoxe(paradoxe)` ne termine pas, alors `arrêt(paradoxe, paradoxe)` est faux et `paradoxe(paradoxe)` termine tout de suite...

Bilan, on ne peut écrire une fonction `arrêt` donc le problème de l'arrêt est indécidable.

III) Pour en savoir plus :

- Article de David Louapre sur le théorème d'incomplétude de Göedel : https://sciencetonnante.wordpress.com/2013/01/14/le-theoreme-de-godel/
- Article de Jean-Gabriel Ganascia sur la décidabilité et la calculabilité : https://interstices.info/alan-turing-du-calculable-a-lindecidable/