Numérique et Sciences Informatiques
Chapitre XII - notion de programme en tant que donnée calculabilité

Classe de terminale

I. Notion de programme en tant que données

Nous avons vu sans le dire que des programmes utilisent d'autres programmes comme données.

En effet, par la modularité du langage Python, nos programmes créés utilisent des modules tels que *math*, *pylab*, *les_listes* (que nous avons créé), etc.
De plus, le débogueur de *Thonny* utilise notre programme comme donnée afin de vérifier si cette donnée est « correcte ». Le téléchargement d'un logiciel est un autre exemple ou encore le compilateur de certains langages de programmation qui transforme le programme en langage machine.

Remarque

Donc un programme est une donnée.

II. Calculabilité

Une fonction f est dite **calculable** s'il existe une méthode précise qui étant donné un argument x permet d'obtenir l'image $f(x)$ en un nombre fini d'étapes.
Donc en informatique, une fonction est calculable s'il existe un algorithme qui permet d'obtenir un résultat à partir de paramètres donnés.

Alan Turing(1912-1954) et Alonso Church(1903-1954) on démontré qu'il existait des fonctions non calculables.

Si une fonction est calculable, c'est qu'il existe un algorithme qui permet de trouver un résultat. Donc la calculabilité ne dépend pas du langage de programmation mais bel et bien de l'algorithme qu'on peut trouver et qui peut être implémenté dans différents langages.

III. Décidabilité

III.1. Généralités

Définition

Un problème donné qui peut être résolu par un algorithme est dit **décidable**.
Un problème donné qui ne peut pas être résolu par un algorithme est dit **indécidable**.

III.2. problème de l'arrêt

Elève : "Monsieur, ça mouline !"
Prof : "Ton programme fait une boucle infinie."
Elève : "Le débogueur ne l'a pas vu ?"
Prof : "C'est normal, le problème de l'arrêt est indécidable."

Le problème de l'arrêt est le problème de décision qui détermine, à partir d'une description d'un programme informatique, et d'une entrée, si le programme s'arrête avec cette entrée ou non.

Démontrons par l'absurde que le problème de l'arrêt est indécidable, c'est-à-dire qu'il n'existe pas de fonction qui puisse indiquer qu'un certain programme va s'arrêter.
Supposons qu'il existe une fonction calculable `halt` qui décide le problème de l'arrêt. C'est-à-dire, que pour tout programme `prog` et pour toute entrée `m` :

- Si `prog(m)` s'arrête, alors `halt` renvoie '*OUI*'
- Si `prog(m)` ne s'arrête pas, alors `halt` renvoie '*NON*'

La fonction `halt()` doit donc avoir deux paramètres : `prog` et `m`, `prog` étant une fonction (ou un programme) et `m` une donnée utilisable par `prog`.
La fonction `halt` est donc construit ainsi :

Algorithme 1 : halt(prog, m)

Entrées : prog, m
si *prog(m) s'arrête* **alors**
| **Sorties :** *OUI*
sinon
| **Sorties :** *NON*
fin

On construit le programme `diagonale` suivant :

Algorithme 2 : diagonale(x)

Entrées : x
si *halt(x, x) renvoie OUI* **alors**
| Création une boucle infinie
sinon
| Arrêt du programme
fin

`diagonale` est une fonction donc est aussi une donnée.

Que se passe-t-il si j'exécute `diagonale(diagonale)` ?

1 cas : supposons que `halt(diagonale,diagonale)` réponde *OUI*.

D'une part, `diagonale(diagonale)` crée une boucle infinie d'après son implémentation.

D'autre part, `halt(diagonale,diagonale)` répond *OUI* est équivalent à dire que `diagonale(diagonale)` s'arrête.
Il y a donc une contradiction.

2 cas : supposons que `halt(diagonale,diagonale)` réponde *NON*.

D'une part, `diagonale(diagonale)` s'arrête d'après son implémentation.

D'autre part, `halt(diagonale,diagonale)` réponde *NON* est équivalent à dire que `diagonale(diagonale)` ne s'arrête pas.
Il y a donc une contradiction.

Dans les deux cas, on obtient une contradiction.
Cela prouve que notre supposition de départ n'est pas possible.
Donc le programme 'halt' n'existe pas. On ne peut donc décider de l'arrêt d'un programme. Le problème de l'arrêt est donc indécidable.