Computability and Complexity
Problem Set 3
Diagonalization, reduction method and Hoare-Allison theorem

Y. Deville
C. Bertrand Van Ouytsel & V. Coppé & A. Gerniers & N. Golenvaux & M. Parmentier
March, 2021

1. Look at the Hoare-Allision diagonalization proof in the lecture slides. Why does the Hoare-Allison theorem not apply to a formalism that allows one to compute non-total functions ?

2. Let L be a (non-trivial) programming language in which the function $\text{halt}(n, x) = 1$ if P_n stops on x, 0 otherwise, is computable.
Using the diagonalization, prove that the function $\text{interpret}(n, x)$ is not computable in L.

3. Any complete formalism of computability must allow to compute its own interpreter. Given that the Python language is a complete formalism of computability, it implies that it is theoretically possible to compute the universal function of Python with Python. In practice, how would you proceed to write a program in Python which would compute this function ?

4. In order to prove the undecidability of a problem, we have so far used the diagonalization method. We now show a more practical method, called the *reduction method*. It is used to prove the undecidability (i.e. the non recursivity) of a set B, knowing the undecidability of the set A. Its principle is simple :

méthode de réduct° à HALT

   (a) We build an algorithm P_A deciding A assuming the existence of an algorithm P_B deciding B. Algorithm P_A can thus use P_B as a subroutine. We say that the decidability of A is *reduced* to the decidability of B.

   (b) We conclude that B is not decidable, since if B were decidable, then A would also be decidable, which is impossible by hypothesis.

   Let $H = \{(n, k) \mid P_n(k) \text{ terminates}\}$. Using the reduction method, prove that the following sets are undecidable because H is undecidable.

   (a) $S_1 = \{\ n \mid P_n(0) \text{ terminates}\}$
   (b) $S_2 = \{\ n \mid \varphi_n(k) = k \text{ for all } k\}$
   (c) $S_3 = \{(n, m) \mid \varphi_n = \varphi_m\}$
   (d) $S_4 = \{\ n \mid \varphi_n \text{ is a non-total function}\}$
   (e) $S_5 = \{(n, m) \mid \forall k, \varphi_n(k) \neq \varphi_m(k)\}$

1. Le résultat peut être $\perp$ (fct non totale)

$\Rightarrow$ interpret $(P_n, x) + 1 = \perp + 1 = \perp$

$\Rightarrow$ pas de contradict° (voir théorème H-A)

2. Commençons de la m̂ manière que H-A

$\Rightarrow$ Supposons interpret calculable

1) Table

	0	k
L_0	interpret(0,0)	interpret(0,k)
⋮		
L_k	interpret(k,1)	interpret(k,k)

2) Sélect° de la diag(n) = interpret(n,n)

3). Pour cette diagonale ; construisons

diag_mod(n) = interpret(n,n) + 1

diag_mod calculable ds L

4) on n'a plus que des fcts totales

$\Rightarrow$ 2 cas : diag_mod(n) = $\begin{cases} \text{interpret}(n,n)+1 & \text{si halt}(n,n)=1 \ \checkmark \\ 0 & \text{si halt}(n,n)=0 \end{cases}$

3. C'est possible de coder un interpréteur python en python.

4. ⚠ méthode de réduct°, grand classique de l'exam !

1) Supposer S_1 récursif

$\Rightarrow \exists$ pgm qui décide S_1

construire pgm

On écrit dans P_H

$P(x) = [\text{return } P_n(k)]$

return $P_{S_1}(P(x))$

↳ regarde si P est dans S_1

P_H(n,k) = [
construire programme P(x) = [return P_n(k)]
d = numéro de programme de P(x)
return P_S1(d)
]