3 Réduction et théorème d'Hoare-Allison

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 ?*

   Réponse : Lorsqu'on modifie la fonction $diag(n) = interpret(n, n)$:

   $$diag\_mod(n) = interpret(n, n) + 1$$

   L'expression « $interpret(n, n) + 1$ » ne fait pas nécessairement toujours sens (si pour un certain $n \in \mathbb{N}$, $P_n(n)$ ne termine pas).

   Si on décide de forcer cette étape en posant $\perp + 1 = \perp$, c'est l'étape suivante de la démonstration qui ne fonctionne plus. En effet, l'égalité $diag\_mod(n) = interpret(n, n)+1$ n'est alors plus nécessairement une contradiction (si $interpret(d, d) = \perp$).

2. *Let L be a (non-trivial) programming language in which the function :*

   $$halt_L(n, x) = \begin{cases} 1 & \text{if } P_n \text{ stops on } x \\ 0 & \text{otherwise} \end{cases}$$

   is computable. Using the diagonalization, prove that the function $interpret_L(n, x)$ is not computable in L.

   Réponse : Supposons par l'absurde que la fonction $interpret_L(n, x)$ est calculable dans L. Soit $P_{interpret_L}(n, x)$ le programme qui calcule cette fonction. Comme pour la preuve du théorème d'Hoare-Allison, on réalise un tableau listant tous les programmes de L et toutes les entrées possibles et on sélectionne la diagonale :

   $$diag(n) = interpret_L(n, n)$$

   Cette fonction est calculable dans L puisque nous avons supposé (par l'absurde) que $interpret(n, x)$ est calculable dans L. Modifions cette fonction de la façon suivante :

   $$diag\_mod(n) = \begin{cases} interpret_L(n, n) + 1 & \text{si } halt_L(n, n) = 1 \\ 0 & \text{si } halt_L(n, n) = 0 \end{cases}$$

   Nous pouvons affirmer que cette fonction $diag\_mod(n)$ est calculable dans L car la fonction $halt_L(n, x)$ est calculable dans L (par hypothèse).

   Comme $diag\_mod(n)$ est calculable dans L, soit d un programme de L qui calcule cette fonction. Étudions la valeur de $diag\_mod(d)$:

   - Si $halt_L(d, d) = 1$, alors $diag\_mod(d) = interpret_L(d, d) + 1 = diag\_mod(d) + 1$, ce qui est absurde.
   - Si $halt_L(d, d) = 0$, alors $diag\_mod(d) = 0$. Mais $halt_L(d, d) = 0$ nous dit également que $P_d(d)$ ne termine pas, autrement dit que la fonction calculée par P_d (qui est $diag\_mod$) n'est pas définie en d, ce qui est absurde.

   En conclusion, on arrive à une contradiction dans tous les cas. Il ne peut donc être vrai que la fonction $interpret_L(n, x)$ est calculable dans 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 ?*

Réponse : Un programme qui calcule cette fonction universelle n'est rien d'autre qu'un interpréteur de Python. Il est tout à fait possible de coder un interpréteur de Python en Python (`PyPy` est un exemple), même si c'est tout sauf simple en pratique. Plus explicitement, voici une version élémentaire d'un tel programme :

$$P_{interpret}(n, x) \equiv \texttt{return compile}(n)[x]$$

Remarque. `compile` est une fonction native de Python potentiellement très dangereuse ! Faites très attention si vous décidez de jouer avec. Le code du programme Python ci-dessus est volontairement syntaxiquement incorrect.

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 :*

   1. *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.*
   2. *We conclude that B is not decidable, since if B were decidable, then A would also be decidable, which is impossible by hypothesis.*

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

   (a) $S_1 = \{n \mid P_n(0)$ terminates$\}$

   Réponse : Supposons par l'absurde que S_1 est récursif. Soit $P_{S_1}(n)$ un programme qui décide S_1. Alors le programme suivant décide *HALT* :

   $$P_H(n, k) \equiv \texttt{return } P_{S_1}\Big(P(x) \equiv \texttt{return } P_n(k)\Big)$$

   C'est absurde. Il ne peut donc être vrai que S_1 est récursif.

   (b) $S_2 = \{n \mid \varphi_n(k) = k \ \forall k\}$

   Réponse : Supposons par l'absurde que S_2 est récursif. Soit $P_{S_2}(n)$ un programme qui décide S_2. Alors le programme suivant décide *HALT* :

   $$P_H(n, k) \equiv \texttt{return } P_{S_2}\left(P(x) \equiv \begin{bmatrix} P_n(k) \\ \texttt{return } x \end{bmatrix}\right)$$

   C'est absurde. Il ne peut donc être vrai que S_2 est récursif.

   (c) $S_3 = \{(n, m) \mid \varphi_n = \varphi_m\}$

   Réponse : Supposons par l'absurde que S_3 est récursif. Soit $P_{S_3}(n, m)$ un programme qui décide S_3. Alors le programme suivant décide *HALT* :

   $$P_H(n, k) \equiv \texttt{return } P_{S_3}\left(P_a(x) \equiv \texttt{return } 1\,,\ P_b(y) \equiv \begin{bmatrix} P_n(k) \\ \texttt{return } 1 \end{bmatrix}\right)$$

   C'est absurde. Il ne peut donc être vrai que S_3 est récursif.

(d) $S_4 = \{n \mid \varphi_n \text{ is a non-total function}\}$

Réponse : Supposons par l'absurde que S_4 est récursif. Soit $P_{S_4}(n)$ un programme qui décide S_4. Alors le programme suivant décide $HALT$:

$$P_H(n,k) \equiv \texttt{return } 1 - P_{S_4}\Big(P(x) \equiv \texttt{return } P_n(k)\Big)$$

C'est absurde. Il ne peut donc être vrai que S_4 est récursif.

(e) $S_5 = \{(n,m) \mid \forall k : \varphi_n(k) \neq \varphi_m(k)\}$

Réponse : Supposons par l'absurde que S_5 est récursif. Soit $P_{S_5}(n)$ un programme qui décide S_5. Alors le programme suivant décide $HALT$:

$$P_H(n,k) \equiv \texttt{return } P_{S_5}\left(P_a(x) \equiv \left[\begin{array}{l}\texttt{while True:}\\ \quad\texttt{pass}\end{array}\right], P_b(y) \equiv \texttt{return } P_n(k)\right)$$

C'est absurde. Il ne peut donc être vrai que S_5 est récursif.