Calculabilité

TP3

Y. Deville
C-H. Bertrand Van Ouytsel - V. Coppé - A. Gerniers
N. Golenvaux - M. Parmentier

Mars 2021

Questions du test

1. Soit L un sous-ensemble récursif non trivial du langage Python qui ne permet de calculer que des fonctions totales. Est-il vrai que la fonction halt de L est calculable avec L ?

   Réponse : Oui
   Voici un programme de L qui calcule cette fonction (qui est bien une fonction totale) :

$$P_{\mathsf{HALT}_L}(n, x) \equiv [\texttt{return } 1$$

Questions du test

2. Soit L un sous-ensemble récursif non trivial du langage Python qui ne permet de calculer que des fonctions totales. Est-il vrai que la fonction halt de L est calculable avec Python ?

   Réponse : Oui
   Voici un programme Python qui calcule cette fonction :

   $$P_{\mathsf{HALT}_L}(n, x) \equiv [\mathtt{return}\ 1$$

Questions du test

3. Soit L un sous-ensemble récursif non trivial du langage Python qui ne permet de calculer que des fonctions totales. Est-il vrai que la fonction interpréteur de L est calculable avec L ?

   Réponse : Non
   Par le théorème d'Hoare-Allison, puisque L est un formalisme de calculabilité qui ne permet de calculer que des fonctions totales, la fonction interpréteur de L n'est pas calculable avec L.

Questions du test

4. Soit L un sous-ensemble récursif non trivial du langage Python qui ne permet de calculer que des fonctions totales. Est-il vrai que la fonction interpréteur de L est calculable avec Python ?

   Réponse : Oui
   L'interpréteur de Python PyPy (qui est un interpréteur de Python écrit en Python) convient. Sinon, plus explicitement :

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

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.

Questions du test

5. Soit L un sous-ensemble récursif non trivial du langage Python qui ne permet de calculer que des fonctions totales. Est-il vrai qu'il existe une fonction totale calculable qui n'est pas calculable avec L ?

   Réponse : Oui
   La fonction interpréteur de L (voir question 3 ci-dessus).

Questions du test

6. Existe-t-il un langage (éventuellement un sous-ensemble du langage Python) qui permet de calculer à la fois sa fonction halt et sa fonction interpréteur ?

   Réponse : Oui, mais il est nécessairement trivial/dégénéré (par le théorème d'Haore-Allison)
   Exemple : le langage constitué d'une unique instruction :
   $L = \{\texttt{return 1}\}$. Ce langage ne permet de créer qu'un seul programme et ce programme calcule à la fois la fonction halt et la fonction interpréteur de ce langage.

Questions du test

7. Est-il vrai qu'un ensemble non récursif ne peut pas être récursivement énumérable ?

   Réponse : Non
   K est un ensemble non récursif mais il est récursivement énumérable.

8. Est-il vrai qu'un ensemble non récursif ne peut pas être co-récursivement énumérable ?

   Réponse : Non
   $\overline{K}$ est un ensemble non récursif mais il est co-récursivement énumérable.

Questions du test

9. L'union de deux ensembles non récursifs est nécessairement non récursif.

   Réponse : Faux
   K et $\overline{K}$ sont deux ensembles non récursif mais $K \cup \overline{K} = \mathbb{N}$ est récursif.

10. L'intersection de deux ensembles non récursifs est nécessairement non récursif.

    Réponse : Faux
    K et $\overline{K}$ sont deux ensembles non récursif mais $K \cap \overline{K} = \emptyset$ est récursif.

Question 1 du TP

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 ?

Question 1 du TP

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

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

L'expression « $\text{interpret}(n, n) + 1$ » 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 $\text{interpret}(d, d) = \text{interpret}(d, d) + 1$ n'est alors plus nécessairement une contradiction (si $\text{interpret}(d, d) = \perp$).

Question 2 du TP

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

Question 2 du TP

Réponse :

Supposons par l'absurde que la fonction $\text{interpret}(n, x)$ (de L) est calculable dans L. Soit $P_{\text{interpret}}(n, x)$ 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 :

$$\text{diag}(n) = \text{interpret}(n, n)$$

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

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

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

Question 2 du TP

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$(d, d) = 1$, alors diag_mod$(d) =$ interpret$(d, d) + 1 =$ diag_mod$(d) + 1$, ce qui est absurde.
- Si halt$(d, d) = 0$, alors diag_mod$(d) = 0$. Mais halt$(d, d) = 0$ 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(n, x) (de L) est calculable dans L.

Question 3 du TP

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 ?

Question 3 du TP

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 :

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

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.

Question 4 du TP

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.

Question 4 du TP

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.

1. $S_1 = \{\, n \mid P_n(0) \text{ terminates}\}$
2. $S_2 = \{\, n \mid \varphi_n(k) = k \text{ for all } k\}$
3. $S_3 = \{(n, m) \mid \varphi_n = \varphi_m\}$
4. $S_4 = \{\, n \mid \varphi_n \text{ is a non-total function}\}$
5. $S_5 = \{(n, m) \mid \forall k, \varphi_n(k) \neq \varphi_m(k)\}$

Question 4 du TP

Réponse :

1. Supposons par l'absurde que $S_1 = \{\, n \mid P_n(0) \text{ terminates}\}$ est récursif. Soit $P_{S_1}(n)$ un programme qui décide S_1. Alors le programme suivant décide H.

$$P_H(n,k) \equiv \left[\texttt{return } P_{S_1}\left(P(x) \equiv [\texttt{return } P_n(k)]\right)\right.$$

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

Question 4 du TP

Réponse :

2. Supposons par l'absurde que $S_2 = \{\ n \mid \varphi_n(k) = k \text{ for all } k\}$ est récursif. Soit $P_{S_2}(n)$ un programme qui décide S_2. Alors le programme suivant décide H.

$$P_H(n,k) \equiv \Bigg[\texttt{return}\ \ P_{S_1}\Bigg(P(x) \equiv [P_n(k); \texttt{return}\ \ x]\Bigg)$$

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

Question 4 du TP

Réponse :

3. Supposons par l'absurde que $S_3 = \{(n, m) \mid \varphi_n = \varphi_m\}$ est récursif. Soit $P_{S_3}(n)$ un programme qui décide S_3. Alors le programme suivant décide H.

$$P_H(n,k) \equiv \Big[\texttt{return}\ \ P_{S_3}\Big(P(x) \equiv [\texttt{return}\ \ 1]\,,\ P(y) \equiv [P_n(k); \texttt{return}\ \ 1]\Big)$$

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

Question 4 du TP

Réponse :

4. Supposons par l'absurde que $S_4 = \{ \, n \mid \varphi_n$ is a non-total function$\}$ est récursif. Soit $P_{S_4}(n)$ un programme qui décide S_4. Alors le programme suivant décide H.

$$P_H(n,k) \equiv \left[\texttt{return } 1 - P_{S_4}\left(P(x) \equiv [\texttt{return } P_n(k)]\right)\right.$$

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

Question 4 du TP

Réponse :

5. Supposons par l'absurde que $S_5 = \{(n, m) \mid \forall k, \varphi_n(k) \neq \varphi_m(k)\}$ est récursif. Soit $P_{S_5}(n)$ un programme qui décide S_5. Alors le programme suivant décide H.

$$P_H(n,k) \equiv \Big[\texttt{return}\ \ P_{S_5}\Big(P(x) \equiv [\texttt{while TRUE : pass}]\,,\ P(y) \equiv [\texttt{return}\ \ P_n(k)]\Big)$$

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