TD7 - Décidabilité

Compléments

Langages. Il existe un lien entre les notions de *problème de décision* et de *langage*. Tout objet informatique peut être représenté par une chaîne de caractères [1]. Dès lors, tout *problème de décision* peut être défini sur un domaine d'entrées E dont les éléments sont des chaînes de caractères. À tout *problème de décision* peut être associée une fonction calculable totale f ou, de façon équivalente, un *algorithme* A. Résoudre le problème de décision, c'est trouver A tel que, pour toute instance $e \in E$, A appliqué à e renvoie $f(e)$ en un temps fini [2].

Un langage L sur un alphabet Σ est défini par un ensemble de mots qui vérifient certaines propriétés. Considérons alors une fonction χ_L telle que :

$$\forall w \in \Sigma^* \quad \chi_L = \begin{cases} \mathsf{V} & \text{si } w \in L \\ \mathsf{F} & \text{si } w \notin L \end{cases}$$

Un telle fonction est appelée *fonction caractéristique*. Il s'agit d'une *fonction totale* sur Σ^*. On peut lui associer un algorithme A qui résout alors le *problème de décision* suivant : le mot $w \in \Sigma^*$ appartient-il à L ? Le langage L associé à A est alors l'ensemble des mots dont A résout le *problème de décision* associé. Ce qui permet d'écrire :

$$L(A) = \{w \in \Sigma^* \mid A(w)\}$$

et de traduire la relation entre un problème de décision A et un langage L par l'équivalence :

$$w \in L(A) \iff A(w)$$

Cette équivalence permet une définition de la classe P, en terme de langage.

La classe P est formée des langages reconnus par des algorithmes polynomiaux.

On comprend peut-être alors mieux l'importance et l'intérêt de la théorie des langages dans le cadre des enseignements d'informatique !

Langages finis et décidabilité. Il est une propriété des langages *finis* : ils sont tous décidables.

Tout langage fini est décidable.

Ce résultat n'exprime rien d'autre que la décidabilité d'un *ensemble fini*. Pour le démontrer, à chaque ensemble fini, on va associer un entier appelé numéro de l'ensemble. Cela est tout à fait possible et l'exemple suivant illustre comment y parvenir. À l'ensemble $E = \{1, 3, 4\}$, on peut associer l'entier 26 dont la représentation binaire est 11010. Les bits égaux à 1 y apparaissent en positions 1, 3 et 4, entiers qui sont les éléments de E. Il reste alors à montrer qu'un tel ensemble est décidable, à savoir qu'il existe un algorithme qui résout le problème de décision associé : un entier x donné appartient-il à E ? Le programme C suivant répond à la question en testant si `x` appartient à l'ensemble fini numéro `n`.

```
int is_in_set (int x, int n) {
  for (i = 0; i < x; i++)
    n = n / 2;
  return n % 2;
}
```

Semi-décidabilité. Le cours introduit la notion de *semi-décidabilité*. La différence essentielle avec la décidabilité est la possibilité pour l'algorithme qui résout le problème de décision associé de ne pas nécessairement terminer ou d'échouer. Pour les langages, la semi-décidabilité traduit la possibilité d'identifier ses éléments. On parle de *langage reconnaissable*. Et ce vocable peut être adopté pour parler de *problème semi-décidable* ou de *problème reconnaissable*.

Illustrons notre propos avec le problème de décision que nous désignons par Accept [3]. Une *instance* de ce problème est un couple formé d'un algorithme A et d'un mot w sur un alphabet Σ. Le *problème de décision* Accept cherche à déterminer si $w \in L(A)$. Ce problème est *indécidable* mais *semi-décidable*.

Montrer d'abord sa *semi-décidabilité*. Soient A un algorithme et w un mot. Si A et w sont dans Accept, alors :

- soit $A(w)$ renvoie V [4] ;
- soit $A(w)$ renvoie F [5] ou ne renvoie aucune réponse, l'algorithme ne s'arretant pas.

1. En fait, on pourrait faire encore plus simple : tout objet informatique est représentable par un entier naturel. Mais pour des raisons pratiques évoquées dans le cours, on adopte la représentation par des chaînes de caractères pour définir notre *modèle de calcul*.
2. On rappelle que $f(e)$ est soit V, soit F.
3. Dans la littérature anglo-saxonne, ce problème est connnu sous le nom de *Accept Problem for Turing Machine*, noté $\text{Accept}_{\text{TM}}$ ou A_{TM}.
4. On dit aussi que A *accepte* w.
5. On dit aussi que A *rejette* w.

Il suffit alors de *simuler* le problème, en proposant, par exemple, le programme suivant. Pour une entrée A et w :

1. exécuter A sur w ;
2. si l'algorithme accepte w alors renvoyer V

L'*indécidabilité* peut être établie par l'absurde. Considérons un algorithme H [6] qui décide Accept. Celà signifie qu'en un temps fini :

$$H(A,w) = \begin{cases} \mathsf{V} \text{ si } A \text{ accepte } w ; \\ \mathsf{F} \text{ si } A \text{ rejette } w. \end{cases}$$

Notons w_A la chaîne de caractères associée au code de A et soit D un second algorithme tel que :

$$D(A) = \text{négation de } H(A, w_A)$$

Le problème de décision associé à D est décidable puisque D est un algorithme. Il doit donc répondre *correctement* pour tout argument, notamment si ce dernier est D. Or :

$$D(D) = \text{négation de } H(D, w_D) = \begin{cases} \mathsf{V} \text{ si } D \text{ rejette } w_D, \text{ soit } D(D) = \mathsf{F} ; \\ \mathsf{F} \text{ si } D \text{ accepte } w_D, \text{ soit } D(D) = \mathsf{V} . \end{cases}$$

Clairement, ce résultat est absurde et D ne peut pas exister. Par voie de conséquence, ni H et ni A ne peuvent exister et finalement Accept est indécidable.

Exercice 1

Cet exercice a essentiellement pour objectif de vérifier la connaissance du cours de *décidabitilé*.

Question 1. Rappeler le cadre dans lequel le cours de MPI se place pour introduire la notion de *calculabité*. En particulier, quel *modèle de calcul* est adopté ?

Question 2. Qu'est une *fonction calculable* ? Combien en existe-t-il ? Justifier votre réponse.

Question 3. Qu'est un *problème de décision* ? Que dire d'un problème de décision sur un domaine d'entrées fini ?

Question 4. Définir les notions de *problème décidable* et de *problème indécidable*. Combien existe-t-il de problème décidables ? Citer au moins trois exemples de problèmes de décision décidables.

Question 5. Qu'est le *problème de l'arrêt* ? Montrer l'indécidabilité de ce problème.

Question 6. Qu'est un *problème semi-décidable* ? Montrer l'existence de problèmes qui ne sont pas semi-décidables.

Question 7. Qu'est un *algorithme universel* ? Que dit le *théorème d'existence d'un algorithme universel* ?

Question 8. Qu'est la *réduction calculatoire* d'un problème de décision, défini par une fonction f_1, à un autre problème de décision, défini par une fonction f_2 ? Qu'exprime le *théorème de réduction d'un problème indécidable* ? Le prouver.

Question 9. Qu'appelle-t-on une fonction booléenne *triviale* ? En partant de l'indécidabilité de l'arrêt, montrer l'indécidabilité de la trivialité d'un programme.

Question 10. Qu'appelle-t-on *équivalence sémantique* de deux algorithmes ? En partant de l'indécidabilité de la trivialité d'un programme, montrer l'indécidabilité de l'équivalence sémantique.

Exercice 2

Soit le langage L défini par l'ensemble des programmes qui reconnaissent des chaînes de caractères de taille 2023.

Question 1. L est-il décidable et reconnaissable ?

Question 2. L est-il indécidable et reconnaissable ?

Question 3. L est-il décidable et non reconnaissable ?

Question 4. L est-il indécidable et non reconnaissable ?

Exercice 3

Soit $A \leqslant_p B$. Parmi les affirmations suivantes, lesquelles sont incorrectes ?

Question 1. Si B est décidable alors A est décidable.

Question 2. Si A est indécidable alors B est indécidable.

Question 3. Si B est reconnaissable alors A est reconnaissable.

Question 4. Si B n'est pas reconnaissable alors A n'est pas reconnaissable.

6. Pour *Hypothèse*.

Exercice 4

Dans cet exercice, l'exécution d'un programme OCaml se fait sur un ordinateur à mémoire infinie. En outre, on suppose que la représentation des entiers est non bornée. On rappelle qu'une fonction $f : \mathbb{N} \to \mathbb{N}$ est *calculable* s'il existe une fonction OCaml `f : int -> int` dont l'exécution termine toujours et calcule les images par `f`. Un *castor affairé*[7] est un programme dont l'exécution termine toujours et qui calcule une valeur la plus grande possible parmi tous les programmes de même taille. Formellement, on s'intéresse au problème d'optimisation suivant noté BB.

- *Instance* : un entier naturel n.
- *Solution* : la valeur k renvoyée par un programme OCaml contenant n caractères[8], dont l'exécution termine toujours et renvoie un entier.
- *Optimisation* : maximiser k.

Par exemple, le programme ci-dessous, de taille 19, termine toujours et renvoie 970299. Ce n'est pas un castor affairé.

```
let k = 99 in k*k*k
```

Le programme suivant, de même taille, renvoie une valeur plus grande. Mais ce n'est toujours pas un castor affairé !

```
9999999999999999999
```

Pour un entier n donné, on note $C(n)$ la valeur maximale renvoyée par un programme OCaml de taille n qui termine et renvoie un entier. Par convention, on pose $C(0) = 0$. On cherche à montrer que BB n'est pas calculable, c'est-à-dire que la fonction C n'est pas calculable.

Question 1. Combien existe-t-il de programme OCaml à n caractères, en supposant qu'on dispose de 128 caractères différents possibles ? Expliquer pourquoi le fait qu'il y en ait un nombre fini ne permet pas de conclure que le problème est calculable.

Question 2. Montrer que la fonction C est correctement définie.

Question 3. Montrer que C est une fonction croissante et que pour tout entier naturel n, $C(n+2) > C(n)$.

Question 4. Que vaut $C(1)$? Le fait qu'on puisse déterminer cette valeur contredit-il la non-calculabilité du problème ?

Question 5. Si $f : \mathbb{N} \to \mathbb{N}$ une fonction calculable, montrer qu'il existe un entier k tel que pour tout entier naturel n :

$$f(n) \leqslant C\left(\left\lfloor \log_{10} n \right\rfloor + k\right)$$

Question 6. Montrer qu'il existe un entier naturel n_0 tel que $f(n_0) < C(n_0)$.

Question 7. Conclure.

Question 8. On considère le problème Arrêt.

- *Instance* : le code source d'un programme OCaml.
- *Question* : est-ce que l'exécution de ce programme termine ?

En utilisant la non-calculabilité de BB, montrer que le problème Arrêt est indécidable.

7. En anglais, *busy beaver*.
8. Parmi les 128 caractères possibles de l'encodage ASCII.