I Décidabilité

On rappelle que le problème de l'arrêt est indécidable :

ARRET
Instance : le code source d'un programme f et un argument x
Question : f termine-t-il sur l'entrée x ?

Pour chacun des problèmes suivants, dire s'il est décidable ou non en le prouvant.

1. **TERMINE-N**
   Instance : une fonction f, un argument x et un entier n.
   Question : est-ce que l'exécution de $f(x)$ termine en moins de n étapes ?

   Solution : Décidable : Il suffit de simuler l'exécution de $f(x)$ pendant n étapes.

2. **ZERO**
   Instance : une fonction f et un argument x.
   Question : est-ce que $f(x)$ renvoie 0 ?

   Solution : Indécidable, par réduction depuis le problème de l'arrêt. Soit `zero` une fonction OCaml résolvant le problème ZERO. Alors la fonction suivante résout le problème de l'arrêt : `let arret f x = zero (fun _ -> f x; 0)`

3. **SAC-A-DOS**
   Instance : un ensemble d'objets de poids $p_1, \dots, p_n$ et de valeurs $v_1, \dots, v_n$, un poids maximal P et une valeur minimale V.
   Question : existe-t-il un sous-ensemble d'objets de poids total inférieur à P et de valeur totale supérieure à V ?

   Solution : Décidable, car on peut tester tous les sous-ensembles possibles.

II Semi-décidabilité

Un problème de décision est dit semi-décidable s'il existe un algorithme qui :

- répond correctement (et en temps fini) pour toutes les instances positives du problème ;
- répond correctement ou ne termine pas pour les instances négatives.

On demande simplement que l'algorithme ne réponde jamais « Oui » pour une instance négative : il peut répondre « Non » pour certaines de ces instances et ne pas terminer pour d'autres.

Pour un problème de décision A, on définit le problème complémentaire de A, noté co-A, comme le problème de décision ayant les mêmes instances que A, mais des réponses opposées. Autrement dit, les instances positives de co-A sont exactement les instances négatives de A, et les instances négatives de co-A sont exactement les instances positives de A.

1. Le problème ARRET est-il semi-décidable ?
2. Montrer qu'un problème A est décidable si et seulement si A et co-A sont tous les deux semi-décidables.
3. Le problème co-ARRET est-il semi-décidable ?
4. **ARRET$_\forall$**
   Instance : une fonction f
   Question : le calcul de $f(x)$ termine-t-il pour tout x ?

   (a) Ce problème est-il décidable ?

(b) Est-ce que co-ARRET$_\forall$ est semi-décidable ?

(c) Est-ce que ARRET$_\forall$ est semi-décidable ?

Solution :

1. Le problème de l'arrêt est semi-décidable :

```
let arret f x =
      f x;
      true
```

   Si f termine sur l'entrée x (instance positive), l'appel `arret f x` renvoie `true`. Si f ne termine pas sur l'entrée x, l'appel `arret f x` ne termine pas non plus.

2. Si A est décidable, alors A est évidemment semi-décidable (par le même algorithme décidant A). De plus, si A est décidable, alors co-A est également décidable (il suffit d'inverser la réponse), donc semi-décidable : cela prouve le sens direct.

   Pour le sens indirect, soient f et g deux algorithmes permettant respectivement de semi-décider A et co-A. L'algorithme suivant décide une instance x de A :

   1 exécuter $f(x)$ pendant une unité de temps ;

   2 si $f(x)$ a répondu « Oui », renvoyer « Oui » ;

   3 sinon, exécuter $g(x)$ pendant une unité de temps ;

   4 si $g(x)$ a répondu « Oui », renvoyer « Non » ;

   5 sinon, repartir à l'étape 1.

   Si x est une instance positive, l'exécution de $f(x)$ terminera et renverra « Oui », sinon l'exécution de $g(x)$ terminera et renverra « Oui ». De manière plus intuitive, cet algorithme revient simplement à faire tourner f et g en parallèle sur l'entrée x et attendre que l'un des deux réponde (ce qui arrivera nécessairement).

3. On sait que ARRET n'est pas décidable et que ARRET est semi-décidable : la contraposée de la question précédente montre que co-ARRET n'est pas semi-décidable.

4. (a) Supposons que ARRET$_\forall$ soit décidable. La fonction suivante décide alors le problème de l'arrêt :

```
let arret f x =
      arret_pour_tout (fun y -> f x)
```

   Donc ARRET$_\forall$ n'est pas décidable.

   (b) Supposons que l'on dispose d'une fonction `coarret_pour_tout` qui semi-décide ce problème. On définit alors :

```
let coarret f x =
      coarret_pour_tout (fun y -> f x)
```

   Cette fonction semi-décide co-ARRET, qui n'est pas semi-décidable : absurde.

   (c) Supposons que l'on dispose d'une fonction `arret_pour_tout` qui semi-décide ce problème. On peut alors considérer l'algorithme suivant :

Entrée	une instance (f, x) de co-ARRET
Comportement	Définir la fonction g qui prend en entrée un entier n et qui : – simule les n premières étapes du calcul de $f(x)$; – si ce calcul a terminé (en au plus n étapes, donc), boucle à l'infini ; – sinon, termine (et ne renvoie rien). Renvoyer le résultat de l'appel `arret_pour_tout g`

   On remarque que :

- si (f, x) est une instance positive de co-ARRET (c'est-à-dire si f ne termine pas sur l'entrée x), alors g est une instance positive de $\text{ARRET}_\forall$, et comme la fonction `arret_pour_tout` semi-décide ce problème, l'algorithme renvoie « Oui » en temps fini ;
- si (f, x) est une instance négative de co-ARRET, alors g est une instance négative de $\text{ARRET}_\forall$, donc l'algorithme renvoie « Non » ou ne termine pas.

Ainsi, on a obtenu un algorithme qui semi-décide co-ARRET : ce problème n'étant pas semi-décidable, c'est absurde. On en déduit que $\text{ARRET}_\forall$ n'est pas semi-décidable.

III Castor affairé

On considère une version idéalisée du langage OCaml où la mémoire est illimitée et le type `int` permet de représenter des entiers arbitrairement grands.

On dit qu'une fonction $f : \mathbb{N} \longrightarrow \mathbb{N}$ est calculable s'il existe une fonction OCaml `f : int -> int`, dont l'exécution termine pour tout argument positif ou nul, qui calcule les images par f.

On appelle programme une expression OCaml de type `int`. Si l'évaluation de cette expression termine (sans erreur), on dit que le programme calcule la valeur de l'expression. On suppose que les programmes OCaml sont écrits en utilisant les 128 caractères ASCII, et l'on appelle taille d'un programme son nombre de caractères.

Par exemple, les deux programmes suivants calculent respectivement les valeurs 13 et 120 :

```
7 + 6
```

```
let rec fact n =
      if n = 0 then 1
      else n * fact (n - 1)
      in
      fact 5
```

alors que le programme ci-dessous ne termine pas, et ne calcule donc pas de valeur :

```
let rec fact n = n * fact (n - 1) in
fact 5
```

Un castor affairé (busy beaver en anglais) est un programme dont l'exécution termine et qui calcule une valeur la plus grande possible parmi tous les programmes de même taille. On définit le problème de décision CASTOR :

CASTOR

Instance : le code source d'un programme OCaml P
Question : P est-il un castor affairé ?

Par exemple, le programme suivant :

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

est un programme de taille 17, qui termine et renvoie 970299. Ce n'est pas un castor affairé, puisque le programme suivant (qui n'est toujours pas un castor affairé) renvoie une valeur plus grande :

```
99999999999999999
```

Pour un entier $n \geq 0$, on notera $C(n)$ la valeur maximale renvoyée par un programme OCaml de taille n qui termine et renvoie un entier. On pose $C(0) = 0$ par convention.

On cherche à montrer que la fonction C n'est pas calculable, et que le problème CASTOR n'est pas décidable.

1. Combien existe-t-il de programmes 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 la fonction C est calculable.
2. Montrer que la fonction C est correctement définie.
3. Montrer que C est une fonction croissante, puis que pour $n \in \mathbb{N}$, on a $C(n+2) > C(n)$.
4. Que vaut $C(1)$?

On considère $f : \mathbb{N} \longrightarrow \mathbb{N}$ une fonction calculable.

5. Montrer qu'il existe un entier k tel que pour tout $n \in \mathbb{N}$, $f(n) \leq C(\lfloor \log_{10} n \rfloor + k)$.
6. Montrer que C n'est pas calculable.
7. En déduire que CASTOR n'est pas décidable.
8. Montrer, en utilisant la non-calculabilité de C, que le problème de l'arrêt est indécidable. On ne demande pas simplement de prouver la non-décidabilité du problème de l'arrêt, comme cela a pu être fait dans le cours, mais bien de la déduire de la non-calculabilité de C.

Solution :

1. Le nombre de programmes contenant n caractères est 128^n (qui est bien fini). Malheureusement, on ne peut pas se contenter de tous les exécuter et ne garder que celui qui s'exécute sans erreur, termine son calcul en temps fini et renvoie un entier, le plus grand possible, car certains de ces programmes peuvent avoir une exécution qui ne termine pas.
2. La fonction C est correctement définie, car chaque l'exécution de chaque programme peut soit terminer en temps fini, soit ne pas terminer (il n'y a pas d'entre deux, les programmes contenant des erreurs terminent en temps fini). De plus, le programme contenant n répétitions du caractère `1` est correct, termine en temps fini et renvoie un entier. Ainsi, l'ensemble des entiers renvoyés par des programmes de taille n qui terminent est un ensemble fini, non vide, inclus dans $\mathbb{Z}$, il possède donc un maximum.
3. On peut rajouter une espace à la fin d'un programme sans changer le déroulé de son exécution (l'espace sera ignorée). On en déduit que pour $n \in \mathbb{N}^*$, $C(n+1) \geq C(n)$: si on dispose d'un castor affairé de taille n, alors il existe un programme de taille $n+1$ qui renvoie le même entier, qui est donc inférieur ou égal à l'entier renvoyé par un castor affairé de taille $n+1$.
   Par ailleurs, en ajoutant `+1` (deux caractères) à la fin d'un castor affairé de taille n, on obtient un programme de taille $n+2$ renvoyant $C(n)+1$. On en déduit que $C(n+2) \geq C(n)+1$.
4. Les seuls programmes corrects de taille 1 sont ceux constitués d'exactement un chiffre, donc $C(1) = 9$. Cela ne contredit pas la non-calculabilité, car cela ne donne pas de méthode générale permettant de calculer $C(n)$ pour tout entier n.
5. Notons $m = \lfloor \log_{10} n \rfloor$, et $n = (c_m c_{m-1} c_1 c_0)_{10}$ l'écriture décimale de n. f étant calculable, il existe une fonction OCaml commençant par `let f x =` permettant de définir une fonction calculant $f(x)$, dont l'exécution termine toujours. Notons k_0 la taille de son code source. En le complétant par `in f` $c_m c_{m-1} c_1 c_0$, on obtient un programme de taille $k_0 + 2 + m + 1$, dont l'exécution termine toujours et qui calcule $f(n)$. En posant $k = k_0 + 3$, on obtient un programme de taille $m + k$ qui renvoie un entier, donc cet entier est inférieur ou égal à l'entier renvoyé par un castor affairé de taille $m + k$, soit $f(n) \leq C(m+k)$.
6. On a $\dfrac{n}{\lfloor \log_{10} n \rfloor + k + 2} \underset{n \to +\infty}{\longrightarrow} +\infty$, donc pour n_0 assez grand, $n_0 \geq \lfloor \log_{10} n_0 \rfloor + k + 2$. Pour un tel n_0, par la question 3, on a :

$$f(n_0) \leq C(\lfloor \log_{10} n_0 \rfloor + k) < C(\lfloor \log_{10} n_0 \rfloor + k + 2) \leq C(n_0)$$

   On a donc $f(n_0) < C(n_0)$, d'où $f \neq C$. Comme ce résultat est valable pour toute fonction calculable f, on en déduit que C n'est pas calculable.
7. Supposons le problème décidable, et considérons alors l'algorithme suivant, qui prend en entrée un entier $n \geq 0$:
   - on énumère tous les programmes de taille n ;
   - pour chacun de ces programmes, on décide s'il s'agit d'un castor affairé (cette décision se fait en temps fini par hypothèse) ;
   - si c'est le cas, on l'exécute (en temps fini, puisque c'est un castor affairé) à l'aide d'une machine universelle et l'on renvoie son résultat.

   Cet algorithme renvoie toujours une valeur (puisqu'il existe un castor affairé de taille n) en temps fini d'après les remarques, et cette valeur est exactement $C(n)$. Cela contredit la non-calculabilité de cette fonction, donc le problème CASTOR n'est pas décidable.
8. Supposons que le problème Arrêt est décidable. Alors le programme suivant pourrait calculer $C(n)$, pour tout n :

- on énumère (sans les exécuter) tous les programmes OCaml de taille n ;
- on décide, à l'aide d'un programme qui résout Arrêt, lesquels ont une exécution en temps fini ;
- on exécute les programmes qui terminent à l'aide d'une machine universelle, en gardant en mémoire les valeurs renvoyées, si ce sont des entiers ;
- on renvoie le maximum de tous ces entiers.

Ainsi, le problème du castor affairé serait calculable. C'est faux, donc Arrêt n'est pas décidable.