TD7 - Décidabilité (éléments de réponses)

Exercice 1

Voir le cours.

Exercice 2

Pour répondre aux questions de cet exercice, il est préférable de ne pas le faire dans l'ordre. L'idée est d'abord de montrer que L est indécidable, puis de montrer qu'il est reconnaissable.

Tout d'abord, la décidabilité implique la semi-décidabilité, formulée par le qualificatif reconnaissable dans l'énoncé. Donc, la réponse à la **question 3** est toujours **négative**.

Ensuite, le problème de décision Accept établit l'indécidabilité de la reconnaissance de tout mot par n'importe quel algorithme. Le problème présent lui est similaire et on peut s'inspirer de la démonstration de l'indécidabilité de Accept pour établir celle de L. Donc, la réponse à la **question 1** est **négative**.

De ces deux premières réponses, il ressort que L est **indécidable**. Il reste à établir s'il est reconnaissable ou pas.

Pour ce faire, considérons un algorithme A qui accepte tout mot de taille 2023. Soit alors le programme suivant.

```
pour s variant de 1 à l'infini
  pour tout mot w de taille 2023
    calculer A(w) en s étapes
    si A accepte w en s étapes alors renvoyer V
```

Puisque le nombre de mots à tester est fini, la seconde boucle termine toujours. Par ailleurs, si `A` accepte `w`, il doit le faire en un nombre fini `s` d'étapes de sorte que la première boucle termine. Toutefois, cette boucle peut tourner indéfiniment si `A` rejette `w`. Ce programme établit ainsi la semi-décidabilité de L.

Par conséquent, L est indécidable et reconnaissable. La **question 4** admet une réponse **négative**. La **question 2** admet une réponse **positive**.

Exercice 3

Question 1. L'affirmation de la question 1 est correcte. Soit g une réduction polynomiale, fonction calculable, de A en B. Celà signifie que si $w \in A$ alors $g(w) \in B$ et réciproquement. Ainsi, si on veut décider A, on choisit un élément $w \in A$, on calcule $g(w)$ et on décide à l'aide de B.

Question 2. L'affirmation de la question 2 est correcte puisque c'est simplement la contraposée de l'affirmation de la question 1.

Question 3. L'affirmation de la question 3 est correcte pour une raison proche de celle invoquée pour la question 1. Bien que la semi-décidabilité de B puisse mener à un calcul qui ne s'arrête pas, ce n'est pas grave. Pour les situations où le calcul termine, et renvoie alors V, g^{-1} établit la terminaison du calcul pour A, qui renvoie également V. Les autres situations importent peu et sont en accord avec la définition de la semi-décidabilité.

Question 4. L'affirmation de la question 4 est incorrecte. Elle n'est pas la contraposée de celle de la question 3 qui s'écrit : *si A n'est pas reconnaissable alors B n'est pas reconnaissable.*

Exercice 4

Question 1. Désignons par $\text{prg}(n)$ l'ensemble des programmes de taille n, c'est-à-dire contenant n caractères parmi les 128 caractères disponibles. Il existe un nombre fini de tels programmes, donné par :

$$|\text{prg}(n)| = 128^n$$

Toutefois, on ne peut espérer tous les exécuter pour ne ne garder que celui qui s'exécute sans erreur, termine son calcul en temps fini et renvoie le plus grand entier possible, car certains de ces programmes peuvent ne pas se terminer !

Question 2. Désignons par $\text{res}(n)$ l'ensemble des valeurs entières renvoyées par les programmes de $\text{prg}(n)$ qui terminent.

$$\text{res}(n) = \{k \in \mathbb{N} \mid \exists A_n \in \text{prg}(n) \text{ tel que } A_n \text{ termine en temps fini et renvoie } k\}$$

Cet ensemble est non vide car il existe au moins un programme de taille n qui termine et qui renvoie un entier : le programme formé d'une répétition de n chiffres 1. Par ailleurs, $\text{res}(n)$ est inclus dans $\mathbb{N}$ et possède donc un maximum. Or $C(n) = \max(\text{res}(n))$. Donc C est correctement définie.

Question 3. Supposons l'existence d'un castor affairé de taille n qui renvoie un entier k_{max}. On peut ajouter une espace à la fin de son programme sans changer le déroulé de son exécution car l'espace sera ignorée. Il existe donc un programme de taille $n+1$ qui renvoie le même entier k_{max}, entier inférieur ou égal à celui renvoyé par un castor affairé de taille $n+1$. Par conséquent :

$$\forall n \in \mathbb{N}^* \qquad C(n+1) \geqslant C(n)$$

Par ailleurs, en complétant la dernière expression d'un castor affairé de taille n par `+1`, on obtient un programme modifié qui termine toujours son exécution en temps fini et qui renvoie un entier strictement plus grand que k_{max}. La taille de ce programme est exactement $n+2$. Ce qui mène nécessairement à $C(n+2) > C(n)$.

Question 4. Les seuls programmes à un seul caractère qui renvoient un entier sont des programmes constitués d'un des dix chiffres. Ainsi, $C(1) = 9$. Cela ne contredit pas la non-calculabilité car cela ne donne pas de méthode générale permettant de calculer $C(n)$ pour tout entier n.

Question 5. Soit n un entier dont l'écriture décimale est $(c_m c_{m-1} \dots c_1 c_0)_{10}$. Sa taille est donc $m = \lfloor \log_{10} n \rfloor$. La fonction f étant calculable, il existe un programme permettant de définir une fonction calculant $f(x)$ dont l'exécution termine toujours. Notons k_0 la taille de ce programme. En le complétant par f et $c_m c_{m-1} \dots c_1 c_0$, on obtient un programme de taille $k_0 + m + 1$ (avec l'espace), dont l'exécution termine toujours et calcule $f(n)$. En posant $k = k_0 + 1$, on obtient un programme de taille $m + k$ qui renvoie un entier. Cet entier est inférieur ou égal à celui renvoyé par un castor affairé de taille $m + k$. Ainsi : $f(n) \leqslant C(m+k)$.

Question 6. On a :

$$\lim_{n \to +\infty} \frac{n}{\lfloor \log_{10} n \rfloor + k + 2} = +\infty$$

Ainsi, pour n_0 assez grand, on a :

$$n_0 \geqslant \lfloor \log_{10} n_0 \rfloor + k + 2$$

Pour un tel n_0, à l'aide de la question 3, il vient :

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

soit finalement :

$$f(n_0) < C(n_0)$$

Question 7. La question précédente établit que pour toute fonction calculable f, on a $f \neq C$. C ne peut être égale à aucune fonction calculable. C n'est donc pas calculable.

Question 8. Supposons le problème Arrêt décidable. Pour tout entier naturel n, le programme suivant pourrait calculer $C(n)$.

- Énumérer tous les programmes de taille n.
- À l'aide d'un programme qui résout Arrêt, décider lesquels ont une exécution en temps fini.
- À l'aide d'une machine universelle, exécuter les programmes qui terminent, en gardant en mémoire les valeurs renvoyées, si ce sont des entiers.
- Renvoyer le maximum de tous ces entiers.

Dès lors, le problème du castor affairé serait calculable. Par la question précédente, on en déduit par l'absurde que Arrêt n'est pas décidable.

Exercice 5

Question 1. Supposons que Arrêt soit décidable par une fonction `arret : string * string -> string`. Considérons alors la fonction `paradoxe : string -> string` suivante.

```
let paradoxe s = match arret (s, s) with
  | false -> ()
  | true -> while true do () done
```

Notons `code_p` le code de la fonction `paradoxe` et considérons l'exécution de `paradoxe code_p`.

- Si `paradoxe code_p` termine, alors `arret (code_p, code_p)` renvoie `true`. On passe dans la ligne 3 et la boucle est infinie. Donc `paradoxe code_p` ne termine pas.
- Si `paradoxe code_p` ne termine pas, alors `arret (code_p, code_p)` renvoie `false`. On passe dans la ligne 2 : on ne fait rien. Donc `paradoxe code_p` termine.

Dans tous les cas, on a une contradiction. Une telle fonction `arret` n'existe pas : Arrêt est indécidable.

Question 2.

- $\psi(\texttt{toto}, \texttt{toto}) = (\texttt{toto} = \texttt{toto}) \leftrightarrow (\texttt{toto} \preccurlyeq \texttt{tototutu})$ est vraie.
- $\psi(\texttt{toto}, \texttt{tutu}) = (\texttt{toto} = \texttt{toto}) \leftrightarrow (\texttt{tutu} \preccurlyeq \texttt{tototutu})$ est fausse car `tutu` n'est pas un préfixe de `tototutu`.

Question 3.

- $\varphi_a(e, s) : s = \text{mpi}$.
- $\varphi_b(e, s) : s \preccurlyeq e$.

Question 4.

□ **4.1.** On considère la fonction `reduction : string * string -> string * string` telle que si `code_f` est le code source d'une fonction `f` et `code_e` est une entrée de `f`, alors `reduction (code_f, code_e)` renvoie un couple `(code_f, formule)` où `formule` représente une formule de postcondition :

```
let reduction (code_f, code_e) =
  let formule = "(e=" ^ code_e ^ ") $\to \bot$" in
  code_f, formule
```

Par exemple, `reduction (code_f, "abc")` renvoie `code_f` et la chaîne de caractères représentant la formule $(e = \text{abc}) \to \bot$.

□ **4.2.** Supposons que PostC soit décidable par une fonction `postc : string * string -> bool`. Alors la fonction suivante décide le problème de l'arrêt.

```
let arret (code_f, code_e) =
  let code_f, formule = reduction (code_f, code_e) in
  not (postc (code_f, formule))
```

En effet, en notant `<code_e>` la valeur contenue par la variable `code_e`, on a :

`arret(code_f, code_e)` renvoie `false`

⇕

`postc(code_f, formule)` renvoie `true`

⇕

`(e = <code_e>)` $\to \bot$ est une postcondition valide de `code_f`

⇕

l'exécution de `code_f` sur `code_e` ne termine pas.

La formule $(e =$ `<code_e>`$) \to \bot$ n'étant jamais vraie pour une fonction ayant pris `<code_e>` en argument, le seul moyen pour qu'une telle formule soit une postcondition valide est que la fonction ne termine pas sur cette entrée. Finalement, Arrêt étant indécidable, on en déduit que PostC est indécidable.