Calculabilité et décidabilité

Langages et programmation

Exercice 1 *QCM d'activation*

Pour chaque question, une seule réponse est correcte.

1. Le problème de l'arrêt consiste à déterminer :
   A. si un programme s'exécute en temps polynomial
   B. si un programme finira par s'arrêter pour une entrée donnée
   C. combien de temps un programme met à s'exécuter
   D. si un programme contient des erreurs de syntaxe
2. Un problème est dit « décidable » si :
   A. il admet au moins une solution
   B. il existe un algorithme qui répond toujours correctement par oui ou non en temps fini
   C. il peut être résolu uniquement par un humain
   D. il ne peut être résolu que par force brute
3. Le problème de l'arrêt est :
   A. décidable et de complexité polynomiale
   B. décidable mais de complexité exponentielle
   C. indécidable
   D. décidable uniquement pour les programmes récursifs
4. (Complément hors programme.) Un problème de la classe NP est un problème :
   A. impossible à résoudre
   B. dont on peut vérifier une solution en temps polynomial
   C. qui ne peut être résolu qu'en temps exponentiel
   D. qui n'a pas de solution
5. (Complément hors programme.) La question « P = NP ? » demande :
   A. si tous les problèmes ont une solution
   B. si tout problème dont la solution est vérifiable rapidement est aussi résoluble rapidement
   C. si les ordinateurs quantiques sont plus rapides
   D. si les problèmes indécidables peuvent devenir décidables

Exercice 2 *Un programme est une donnée*

La question 3 utilise la fonction suivante :

```
def double(x):
    return 2 * x
```

1. Expliquer en quelques phrases pourquoi un programme peut être considéré comme « une donnée comme une autre ».
2. Citer deux programmes qui reçoivent un programme comme donnée, et dire ce que chacun en fait.
3. Écrire une fonction `appliquer_deux_fois(f, x)` qui reçoit une fonction `f` et une valeur `x`, et renvoie `f(f(x))`. Par exemple, `appliquer_deux_fois(double, 5)` renvoie `20`.
4. Écrire une fonction `nombre_de_print(source)` qui reçoit le code source d'un programme sous forme de chaîne de caractères et renvoie le nombre de lignes qui contiennent `print(`. Par exemple, avec `source = "x = 3\nprint(x)\nprint(x + 1)\n"`, l'appel `nombre_de_print(source)` renvoie `2`.
5. La fonction `nombre_de_print` permet-elle de savoir combien de lignes un programme **affichera** quand on l'exécutera ? Justifier avec un exemple.

Exercice 3 *Un programme qui ne s'arrête pas toujours*

Une élève met 5 € de côté chaque semaine. La fonction suivante compte le nombre de semaines nécessaires pour atteindre un objectif :

```
def epargner(objectif):
    total = 0
    semaines = 0
    while total != objectif:
        total = total + 5
        semaines = semaines + 1
    return semaines
```

1. Que renvoie `epargner(20)` ? Donner les valeurs successives de `total`.
2. Que se passe-t-il lors de l'appel `epargner(12)` ? Justifier.
3. Que renvoie `epargner(0)` ?
4. Modifier la condition de la boucle pour que `epargner` s'arrête pour tout objectif positif ou nul. Que renvoie alors `epargner(12)` ?

Exercice 4 *La suite de Syracuse*

On reprend la fonction du cours :

```
def syracuse(n):
    etapes = 0
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        etapes = etapes + 1
    return etapes
```

1. Écrire à la main les valeurs successives de `n` lors de l'appel `syracuse(7)`. Que renvoie cet appel ?

2. Écrire une fonction `hauteur_max(n)` qui renvoie la plus grande valeur prise par `n` au cours du calcul, valeur de départ comprise. Par exemple, `hauteur_max(7)` renvoie `52`.
3. Les ordinateurs ont vérifié que `syracuse(n)` s'arrête pour tous les entiers jusqu'à plus de 10^{20}. Peut-on en conclure que `syracuse(n)` s'arrête pour tout entier `n` ? Justifier.

Exercice 5 *Le programme `meme` et le programme `contraire`*

On suppose, comme dans le cours, qu'il existe une fonction `arret(programme, donnee)` qui renvoie `True` si l'appel `programme(donnee)` s'arrête, et `False` sinon. On écrit le programme `meme`, qui **suit** la prédiction de `arret` au lieu de la contredire :

```
def meme(programme):
    if arret(programme, programme):
        return "terminé"
    else:
        while True:
            pass
```

1. On suppose que `arret(meme, meme)` renvoie `True`. Que fait l'appel `meme(meme)` ? La prédiction de `arret` est-elle juste ?
2. Même question si `arret(meme, meme)` renvoie `False`.
3. Le programme `meme` permet-il de démontrer que la fonction `arret` n'existe pas ? Justifier.
4. Modifier `meme` pour obtenir un programme qui met `arret` en défaut dans les deux cas, puis rédiger la démonstration.

Exercice 6 *Décidable ou indécidable ?*

Pour chacun des problèmes suivants, indiquer s'il est **décidable** ou **indécidable**. Justifier brièvement.

1. Déterminer si un nombre entier donné est premier.
2. Déterminer si un programme Python donné affiche `"bonjour"` au moins une fois.
3. Déterminer si une liste triée contient un élément donné.
4. Déterminer si deux programmes Python calculent toujours la même chose.
5. Déterminer si un entier donné est pair.
6. Déterminer si un programme Python donné contient une boucle `while`.

Exercice 7 *P, NP, NP-complet (complément hors programme)*

Classer les problèmes suivants dans la catégorie qui convient (P, NP, NP-complet). Justifier.

1. Trier une liste de n nombres.
2. Déterminer si un graphe admet un cycle hamiltonien.
3. Rechercher un élément dans une liste triée.
4. Le problème du sac à dos (version décision).
5. Calculer les distances les plus courtes depuis un sommet dans un graphe pondéré à poids positifs.

Exercice 8 *Vrai ou faux*

Pour chaque affirmation, indiquer si elle est **vraie** ou **fausse** et justifier. Les affirmations 2 à 5 portent sur les classes P et NP, complément hors programme.

1. « Le problème de l'arrêt montre que certains programmes ne s'arrêtent jamais. »
2. « Si un problème est dans NP, alors il est difficile à résoudre. »
3. « Tous les problèmes de P sont aussi dans NP. »
4. « NP signifie "non polynomial". »
5. « Si on prouvait que P = NP, la cryptographie RSA serait menacée. »
6. « Le théorème de Rice dit que toute question sur un programme est indécidable. »

Exercice 9 *Réduction au problème de l'arrêt*

On veut montrer que le problème « `prog(0)` renvoie-t-il 42 ? » est indécidable.

1. Supposons qu'il existe `test42(prog)`. Construire, à partir de `test42`, un programme capable de résoudre le problème de l'arrêt.

   Indication : pour un programme `p` et une entrée `x` donnés, construire un nouveau programme `q` tel que `q(0)` renvoie 42 si et seulement si `p(x)` s'arrête.

2. En déduire que le problème est indécidable.

Exercice 10 *Vérification vs résolution (classe NP, complément hors programme)*

Le sudoku généralisé, joué sur une grille de taille quelconque, est un problème NP-complet ; la grille 9×9 habituelle en est un cas particulier.

1. Expliquer ce que signifie « la vérification d'une solution est polynomiale » pour le sudoku. Quelle est la complexité ?
2. Pourquoi ne suffit-il pas de « vérifier toutes les grilles possibles » ? Estimer le nombre de grilles à tester.
3. Le problème du tri d'une liste est dans P. Est-il aussi dans NP ? Justifier.