

Calculabilité et décidabilité



Exercice 1 (Le problème de l’arrêt – Tiré d’un exercice d’épreuve écrite)

On se demande s’il est possible d’écrire une fonction `arret` qui prend en arguments :

- une chaîne de caractères `code_f` contenant le code d’une fonction `f` ;
- un argument `x` de `f`

et tel que `arret(code_f, x)` renvoie `True` si l’appel `f(x)` va terminer et `False` sinon.

1. Écrire une fonction Python `infini`, récursive, telle que l’appel `infini(n)` ne termine pour aucun entier `n`.

Dans la suite de cet exercice, on suppose disposer d’une telle fonction `arret` et on implémente la fonction suivante, utilisant cette fonction `arret`, ainsi que la fonction `infini` de la question précédente dont l’appel `infini(n)` ne termine jamais quelle que soit la valeur de `n`.

```
1 def paradoxe(x):
2     if arret(x, x):
3         infini(42)
4     else:
5         return 0
```

De même, on suppose disposer d’une variable `code_paradoxe` contenant le code de la fonction `paradoxe` sous la forme d’une chaîne de caractères.

On s’intéresse alors à l’appel `paradoxe(code_paradoxe)`.

Cet appel de fonction commence par effectuer le test `arret(code_paradoxe, code_paradoxe)` dans le `if` de la ligne 2.

2. Dans le cas où `arret(code_paradoxe, code_paradoxe)` renvoie `True`, préciser la prochaine instruction à être exécutée.
Dans ce cas, expliquer si l’appel `paradoxe(code_paradoxe)` termine.
3. Dans le cas où `arret(code_paradoxe, code_paradoxe)` renvoie `False`, préciser la prochaine instruction à être exécutée.
Dans ce cas, expliquer si l’appel `paradoxe(code_paradoxe)` termine.
4. En déduire qu’une telle fonction `arret` ne peut exister.

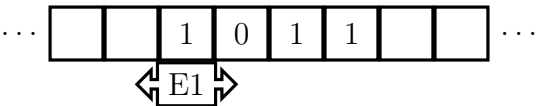
Exercice 2

Compléter la table de transition ci-dessous, afin que la machine effectue une multiplication par 2 d’un nombre binaire écrit sur le ruban à l’état initial.

état	lu	écrit	déplacement	état suivant
E1	0	...	...	...
E1	1	...	...	...
E1	blanc	...	...	...

Dans l’état initial, la tête est sur le bit de poids fort.

Ainsi, par exemple l’état initial est le suivant pour le nombre $11 = (1011)_2$:

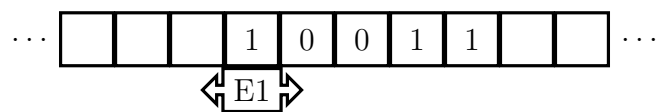


Exercice 3

La table de transition d'une machine de Turing est la suivante :

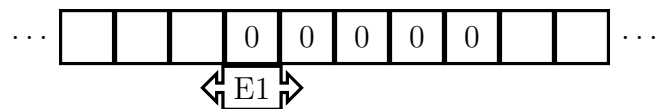
état	lu	écrit	déplacement	état suivant
E1	0	1	→	E1
E1	1	0	→	E1
E1	blanc	blanc	←	E2
E2	0	1	←	E3
E2	1	0	←	E2
E2	blanc	blanc	→	F
E3	0	0	←	E3
E3	1	1	←	E3
E3	blanc	blanc	→	F

1. Sachant que la machine est dans l'état initial suivant :



Quelle est la situation à la fin ?

2. Même question dans la situation initiale suivante :

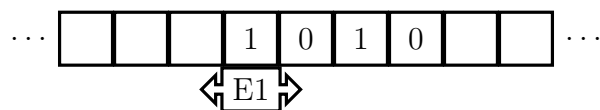


3. Que fait cette machine de Turing ?

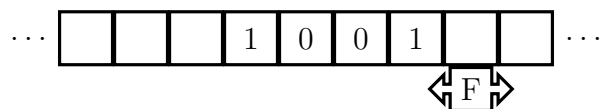
Exercice 4

Écrire la table de transition d'une machine de Turing qui effectue la soustraction de 1 d'un nombre entier (non nul) écrit sous forme binaire sur le ruban initial, la tête de lecture étant située sur le bit de poids fort au départ. La tête de lecture doit être située sur la case de droite du bit de poids faible du résultat à la fin.

Par exemple, si l'état initial est le suivant :



alors l'état final est le suivant :



Pour s'aider, réfléchir à ce qui se passe lorsque l'on soustrait 1 aux nombres suivants : $(101)_2$, $(1010)_2$ et $(100)_2$.

On pourra utiliser un état terminal ERR indiquant que le programme se termine sur une ERR (pour le cas où l'entrée serait un nombre égal à 0, par exemple).

Facultatif : Faire en sorte que si le bit de poids fort a été transformé en 0, celui-ci soit effacé du ruban.

Exercice 5

On souhaite réaliser et exécuter des machines de Turing en Python.

Il existe bien sûr plusieurs manières de le faire.

On donne ici un code possible, dont les ... et **pass** sont à remplacer :

```

# Un exemple de table de transition (celle de l'exercice 3) :
tableEx3 = {"E1":{0:(1,">","E1"),
                  1:(0,">","E1"),
                  None:(None,"<","E2")},
            "E2":{0:(1,"<","E3"),
                  1:(0,"<","E2"),
                  None:(None,">","F")},
            "E3":{0:(0,"<","E3"),
                  1:(1,"<","E3"),
                  None:(None,">","F")}}

def affiche_ruban(ruban):
    print("|",end="")
    for v in ruban:
        if v is None:
            print("  |",end="")
        else:
            print(" "+str(v)+" |",end="")
    print()

def affiche_tete(tete,ruban):
    print(" ",end="")
    i,e = tete
    for j,v in enumerate(ruban):
        if j == i:
            lt = 4 if v is None else 3+len(str(v))
            s1=(lt-len(e))//2
            print(s1*" ",e,sep="")
            break
        else:
            lt = 4 if v is None else 3+len(str(v))
            print(lt*" ",end="")

def affiche_machine(tete,ruban):
    '''affiche la machine, c'est à dire le ruban et la tête'''
    affiche_ruban(ruban)
    affiche_tete(tete,ruban)

def ecrit(ruban,i,w):
    '''écrit sur le ruban la valeur w à l'indice i'''
    pass

def deplace(i,m,ruban):
    '''retourne le nouvel indice de la tête
    et le ruban éventuellement rallongé'''
    if m==">": # déplacement à droite
        if i==len(ruban)-1:
            return ...,...
        else:
            return ...,...
    elif m=="<": # déplacement à gauche
        if i==0:

```

```

        return ...,...
    else:
        return ...,...
else: # aucun déplacement
    return i, ruban

def execute(ruban, tete, etatsFinaux, table):
    '''ruban est une liste non vide
       tete est un couple (indice, état)
       etatsFinaux est une liste d'états finaux
       table est un dictionnaire de dictionnaires de triplets
       dont un exemple est donné plus haut.
       Cette fonction exécute la machine selon les règles
       données dans la table.
       Après chaque étape elle fait afficher l'état de la machine'''
    print("état initial :")
    affiche_machine(tete, ruban)
    while tete[1] not in etatsFinaux:
        i, e = tete
        lu = ... # récupère la valeur lue par la tête sur le ruban
        w, m, s = ... # récupère l'action à effectuer selon la table
        ecrit(ruban, i, w)
        i, ruban = deplace(i, m, ruban)
        tete = ... # nouvelle situation de la tête
        affiche_machine(tete, ruban)

execute([None, None, None, 1, 0, 0, 1, 1, None, None], (3, "E1"), ["F"], tableEx3)
execute([None, None, None, 0, 0, 0, 0, 0, None, None], (3, "E1"), ["F"], tableEx3)
# On peut aussi exécuter cela :
# execute([1, 0, 0, 1, 1], (0, "E1"), ["F"], tableEx3)
# execute([0, 0, 0, 0, 0], (0, "E1"), ["F"], tableEx3)

```

Compléter le code et l'exécuter pour vérifier son bon fonctionnement.

On pourra également définir la table de l'exercice 3 et la tester.

Exercice 6 (Facultatif)

Définir la table de transition d'une machine de Turing qui décale son entrée d'une case vers la droite. La position initiale est sur le bit de poids fort.

Exercice 7 (Facultatif)

Définir la table de transition d'une machine de Turing qui, si l'entrée contient un 0, affiche un 1 en tête de l'entrée, et sinon affiche un 0 en tête de l'entrée.

Exercice 8 (Facultatif)

Définir la table de transition d'une machine de Turing qui, si l'entrée contient deux 0 d'affilée, affiche un 1 en tête de l'entrée, et sinon affiche un 0 en tête de l'entrée.

Exercice 9 (Facultatif, difficile)

On considère la table de transition suivante :

état	lu	écrit	déplacement	état suivant
P	0	blanc	→	D0
P	1	blanc	→	D1
P	blanc	1	.	F
D0	0	0	→	D0
D0	1	1	→	D0
D0	blanc	blanc	←	V0
D1	0	0	→	D1
D1	1	1	→	D1
D1	blanc	blanc	←	V1
V0	0	blanc	←	R
V0	1	blanc	←	E
V0	blanc	1	.	F
V1	0	blanc	←	E
V1	1	blanc	←	R
V1	blanc	1	.	F
R	0	0	←	R
R	1	1	←	R
R	blanc	blanc	→	P
E	0	blanc	←	E
E	1	blanc	←	E
E	blanc	0	.	F

L'état initial est P. L'état terminal est F.

On considère qu'on initialise la machine en plaçant la tête sur le bit de poids fort.

Que fait-elle ?

On pourra dans un premier temps se demander ce qu'effectue la machine dans chacun de ses états.