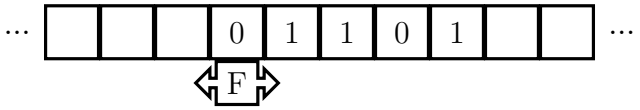
# Calculabilité et décidabilité
# Correction

**Exercice 1**
Pour multiplier par 2, il suffit d'ajouter un 0 à droite. Ainsi, la table est la suivante :

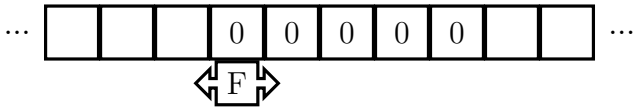
| état | lu | écrit | déplacement | état suivant |
|---|---|---|---|---|
| E1 | 0 | 0 | $\rightarrow$ | E1 |
| E1 | 1 | 1 | $\rightarrow$ | E1 |
| E1 | blanc | 0 | $\cdot$ | F |

**Exercice 2**

1. Dans ce cas, l'état final est le suivant :




2. Ici, l'état final du ruban est le même que l'état initial :




3. Cette machine effectue le complément à 2 du nombre binaire écrit sur le ruban à l'état initial. En effet, elle échange d'abord tous les 1 et les 0, puis elle ajoute 1, sans toutefois dépasser la taille du nombre initial.

**Exercice 3**
Voici la table demandée :

| état | lu | écrit | déplacement | état suivant |
|---|---|---|---|---|
| E1 | 0 | 0 | $\rightarrow$ | E1 |
| E1 | 1 | 1 | $\rightarrow$ | E1 |
| E1 | blanc | blanc | $\leftarrow$ | E2 |
| E2 | 0 | 1 | $\leftarrow$ | E2 |
| E2 | 1 | 0 | $\rightarrow$ | E3 |
| E2 | blanc | blanc | $\cdot$ | ERR |
| E3 | 0 | 0 | $\rightarrow$ | E3 |
| E3 | 1 | 1 | $\rightarrow$ | E3 |
| E3 | blanc | blanc | $\cdot$ | F |

Et la table de la question facultative :

| état | lu | écrit | déplacement | état suivant |
|---|---|---|---|---|
| E1 | 0 | 0 | $\rightarrow$ | E1 |
| E1 | 1 | 1 | $\rightarrow$ | E1 |
| E1 | blanc | blanc | $\leftarrow$ | E2 |
| E2 | 0 | 1 | $\leftarrow$ | E2 |
| E2 | 1 | 0 | $\leftarrow$ | Es |
| E2 | blanc | blanc | $\cdot$ | ERR |
| Es | 0 | 0 | $\rightarrow$ | E3 |
| Es | 1 | 1 | $\rightarrow$ | E3 |
| Es | blanc | blanc | $\rightarrow$ | Ez |
| Ez | 0 | blanc | $\rightarrow$ | E3 |
| Ez | 1 | 1 | $\rightarrow$ | ERR |
| Ez | blanc | blanc | $\rightarrow$ | ERR |
| E3 | 0 | 0 | $\rightarrow$ | E3 |
| E3 | 1 | 1 | $\rightarrow$ | E3 |
| E3 | blanc | blanc | $\cdot$ | F |

**Exercice 4**
Voici le code complété :

```
def ecrit(ruban,i,w):
    '''écrit sur le ruban la valeur w à l'indice i'''
    ruban[i]=w

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 i+1,ruban+[None]
        else:
            return i+1,ruban
    elif m=="<": # déplacement à gauche
        if i==0:
            return 0,[None]+ruban
        else:
            return i-1,ruban
    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=ruban[i] # récupère la valeur lue par la tête sur le ruban
        w,m,s=table[e][lu] # récupère l'action à effectuer selon la table
        ecrit(ruban,i,w)
        i,ruban=deplace(i,m,ruban)
        tete=(i,s) # nouvelle situation de la tête
        affiche_machine(tete,ruban)
```

en exécutant les deux lignes suivantes :

```
execute([1,0,0,1,1],(0,"E1"),["F"],tableEx2)
execute([0,0,0,0,0],(0,"E1"),["F"],tableEx2)
```

On obtient la chose suivante :

```
état initial :
|   |   |   | 1 | 0 | 0 | 1 | 1 |   |   |
              E1
|   |   |   | 0 | 0 | 0 | 1 | 1 |   |   |
                  E1
|   |   |   | 0 | 1 | 0 | 1 | 1 |   |   |
                      E1
|   |   |   | 0 | 1 | 1 | 1 | 1 |   |   |
                          E1
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
                              E1
|   |   |   | 0 | 1 | 1 | 0 | 0 |   |   |
                                  E1
|   |   |   | 0 | 1 | 1 | 0 | 0 |   |   |
                              E2
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
                          E3
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
                      E3
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
                  E3
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
              E3
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
          E3
|   |   |   | 0 | 1 | 1 | 0 | 1 |   |   |
              F
```

```
état initial :
|   |   |   | 0 | 0 | 0 | 0 | 0 |   |   |
              E1
|   |   |   | 1 | 0 | 0 | 0 | 0 |   |   |
                  E1
|   |   |   | 1 | 1 | 0 | 0 | 0 |   |   |
                      E1
|   |   |   | 1 | 1 | 1 | 0 | 0 |   |   |
                          E1
|   |   |   | 1 | 1 | 1 | 1 | 0 |   |   |
                              E1
|   |   |   | 1 | 1 | 1 | 1 | 1 |   |   |
                                  E1
|   |   |   | 1 | 1 | 1 | 1 | 1 |   |   |
                              E2
|   |   |   | 1 | 1 | 1 | 1 | 0 |   |   |
                          E2
|   |   |   | 1 | 1 | 1 | 0 | 0 |   |   |
                      E2
|   |   |   | 1 | 1 | 0 | 0 | 0 |   |   |
                  E2
|   |   |   | 1 | 0 | 0 | 0 | 0 |   |   |
              E2
|   |   |   | 0 | 0 | 0 | 0 | 0 |   |   |
          E2
|   |   |   | 0 | 0 | 0 | 0 | 0 |   |   |
              F
```

Avec le ruban initial ne contenant que le nombre, soit :

```
execute([None,None,None,1,0,0,1,1,None,None],(3,"E1"),["F"],tableEx2)
execute([None,None,None,0,0,0,0,0,None,None],(3,"E1"),["F"],tableEx2)
```

On obtient :

```
état initial :
| 1 | 0 | 0 | 1 | 1 |
  E1
| 0 | 0 | 0 | 1 | 1 |
      E1
| 0 | 1 | 0 | 1 | 1 |
          E1
| 0 | 1 | 1 | 1 | 1 |
              E1
| 0 | 1 | 1 | 0 | 1 |
                  E1
| 0 | 1 | 1 | 0 | 0 |   |
                      E1
| 0 | 1 | 1 | 0 | 0 |   |
                  E2
| 0 | 1 | 1 | 0 | 1 |   |
              E3
| 0 | 1 | 1 | 0 | 1 |   |
          E3
| 0 | 1 | 1 | 0 | 1 |   |
      E3
| 0 | 1 | 1 | 0 | 1 |   |
  E3
|   | 0 | 1 | 1 | 0 | 1 |   |
  E3
|   | 0 | 1 | 1 | 0 | 1 |   |
      F
```

```
état initial :
| 0 | 0 | 0 | 0 | 0 |
  E1
| 1 | 0 | 0 | 0 | 0 |
      E1
| 1 | 1 | 0 | 0 | 0 |
          E1
| 1 | 1 | 1 | 0 | 0 |
              E1
| 1 | 1 | 1 | 1 | 0 |
                  E1
| 1 | 1 | 1 | 1 | 1 |   |
                      E1
| 1 | 1 | 1 | 1 | 1 |   |
                  E2
| 1 | 1 | 1 | 1 | 0 |   |
              E2
| 1 | 1 | 1 | 0 | 0 |   |
          E2
| 1 | 1 | 0 | 0 | 0 |   |
      E2
| 1 | 0 | 0 | 0 | 0 |   |
  E2
|   | 0 | 0 | 0 | 0 | 0 |   |
  E2
|   | 0 | 0 | 0 | 0 | 0 |   |
      F
```

**Exercice 5**

Il suffit que l'état dépende du nombre lu précédent. On appelle I l'état initial.

| état | lu | écrit | déplacement | état suivant |
|---|---|---|---|---|
| I | 0 | blanc | → | D0 |
| I | 1 | blanc | → | D1 |
| I | blanc | blanc | · | F |
| D0 | 0 | 0 | → | D0 |
| D0 | 1 | 0 | → | D1 |
| D0 | blanc | 0 | · | F |
| D1 | 0 | 1 | → | D0 |
| D1 | 1 | 1 | → | D1 |
| D1 | blanc | 1 | · | F |

```
table = {"I":{0:(None,">","D0"),
              1:(None,">","D1"),
              None:(None,".","F")},
         "D0":{0:(0,">","D0"),
               1:(0,">","D1"),
               None:(0,".","F")},
         "D1":{0:(1,">","D0"),
               1:(1,">","D1"),
               None:(1,"<","F")}}
```

**Exercice 6**

Ici l'idée est que dès que l'on rencontre un 0 on revient à gauche pour y écrire un 1, sinon, on continue et lorsque l'on tombe sur un blanc, on revient à gauche pour y écrire un 0.

On appelle C l'état initial (pour curseur). Cela donne :

| état | lu | écrit | déplacement | état suivant |
|---|---|---|---|---|
| C | 0 | 0 | $\leftarrow$ | U |
| C | 1 | 1 | $\rightarrow$ | C |
| C | blanc | blanc | $\leftarrow$ | Z |
| U | 0 | 0 | $\leftarrow$ | U |
| U | 1 | 0 | $\leftarrow$ | U |
| U | blanc | 1 | $\cdot$ | F |
| Z | 0 | 0 | $\leftarrow$ | Z |
| Z | 1 | 1 | $\leftarrow$ | Z |
| Z | blanc | 0 | $\cdot$ | F |

**Remarque** Normalement, on ne peut pas lire de 0 quand on est dans l'état U ou Z.

```
table = {"C":{0:(0,"<","U"),
              1:(1,">","C"),
              None:(None,"<","Z")},
         "U":{0:(0,"<","U"),
              1:(1,"<","U"),
              None:(1,".","F")},
         "Z":{0:(0,"<","Z"),
              1:(1,"<","Z"),
              None:(0,".","F")}}
```

**Exercice 7**
Ici il faut retenir, avec l'état, si un zéro a été lu précédemment ou non, et ne revenir que si l'on lit à nouveau un 0. On a alors :

| état | lu | écrit | déplacement | état suivant |
|---|---|---|---|---|
| C | 0 | 0 | $\rightarrow$ | C1 |
| C | 1 | 1 | $\rightarrow$ | C |
| C | blanc | blanc | $\leftarrow$ | Z |
| C1 | 0 | 0 | $\leftarrow$ | U |
| C1 | 1 | 1 | $\rightarrow$ | C |
| C1 | blanc | blanc | $\leftarrow$ | Z |
| U | 0 | 0 | $\leftarrow$ | U |
| U | 1 | 0 | $\leftarrow$ | U |
| U | blanc | 1 | $\cdot$ | F |
| Z | 0 | 0 | $\leftarrow$ | Z |
| Z | 1 | 1 | $\leftarrow$ | Z |
| Z | blanc | 0 | $\cdot$ | F |

Ici, dans l'état U, on tombe la première fois, et uniquement cette fois, sur 0.

```
table = {"C":{0:(0,">","C1"),
              1:(1,">","C"),
              None:(None,"<","Z")},
         "C1":{0:(0,"<","U"),
               1:(1,">","C"),
               None:(None,"<","Z")},
```

```
          "U":{0:(0,"<","U"),
               1:(1,"<","U"),
               None:(1,".","F")},
          "Z":{0:(0,"<","Z"),
               1:(1,"<","Z"),
               None:(0,".","F")}}
```

**Exercice 8**

La machine répond à la question de savoir si l'entrée est un palindrome (auquel cas elle répond 1) ou non (0). Dans tous les cas, elle efface le ruban, ne laissant sur celui-ci que la réponse au niveau de la tête de lecture.

On peut comprendre ainsi les états :

**P (fonction palindrome) :** S'il y a un 0 alors D0, s'il y a un 1 alors D1, sinon répond 1

**D0 (0 à droite ?) :** Va tout à droite et exécute V0

**D1 (1 à droite ?) :** Va tout à droite et exécute V1

**V0 :** S'il y a vraiment un 0, effectue la récursivité R, sinon E

**V1 :** S'il y a vraiment un 1, effectue la récursivité R, sinon E

**E :** Efface tout puis écrit 0 (ce n'est pas un palindrome)

**R :** Retourne en tête et exécute la fonction palindrome P

```
table = {"P":{0:(None,">","D0"),
              1:(None,">","D1"),
              None:(1,".","F")},
         "D0":{0:(0,">","D0"),
               1:(1,">","D0"),
               None:(None,"<","V0")},
         "D1":{0:(0,">","D1"),
               1:(1,">","D1"),
               None:(None,"<","V1")},
         "V0":{0:(None,"<","R"),
               1:(None,"<","E"),
               None:(1,".","F")},
         "V1":{0:(None,"<","E"),
               1:(None,"<","R"),
               None:(1,".","F")},
         "R":{0:(0,"<","R"),
              1:(1,"<","R"),
              None:(None,">","P")},
         "E":{0:(None,"<","E"),
              1:(None,"<","E"),
              None:(0,".","F")}}
```