Python – Machines de Turing

Modélisation

Afin de modéliser les machines de Turing, nous allons utiliser une classe pour le ruban et une autre pour la machine. L'interface du ruban est :

Fonction	Description
`Ruban(contenu, position, defaut)`	Crée un ruban contenant chacun des symboles de `contenu`, place la tête à la position indiquée et toute nouvelle case contiendra le symbole `defaut`.
`ruban.a_gauche()`	Déplace la tête à gauche et rajoute une case si nécessaire.
`ruban.a_droite()`	Déplace la tête à droite et rajoute une case si nécessaire.
`ruban.lecture()`	Renvoie la valeur de la case sur laquelle se trouve la tête de lecture.
`ruban.ecriture(valeur)`	Place `valeur` sur la case sur laquelle se trouve la tête.
`ruban.afficher()`	Affiche le contenu de ruban et la position de la tête.
`ruban.verifier(contenu)`	Renvoie un booléen pour indiquer si le contenu du ruban est identique à `contenu`.

L'interface du Ruban est :

Fonction	Description
`Machine(contenu, position, etat)`	Crée une machine dans l'état `etat` avec un ruban correspondant à `contenu`, la tête se trouvant en `position`.
`machine.executer(programme)`	Éxecute une instruction du programme et modifie le ruban, l'état et la position de la tête, et renvoie un booléen indiquant si une commande a été exécutée ou pas.
`machine.afficher()`	Afiche l'état et le ruban.
`machine.verifier(contenu)`	Renvoie un booléen pour indiquer si le contenu du ruban de la machine est identique à `contenu`.

Implantation du ruban

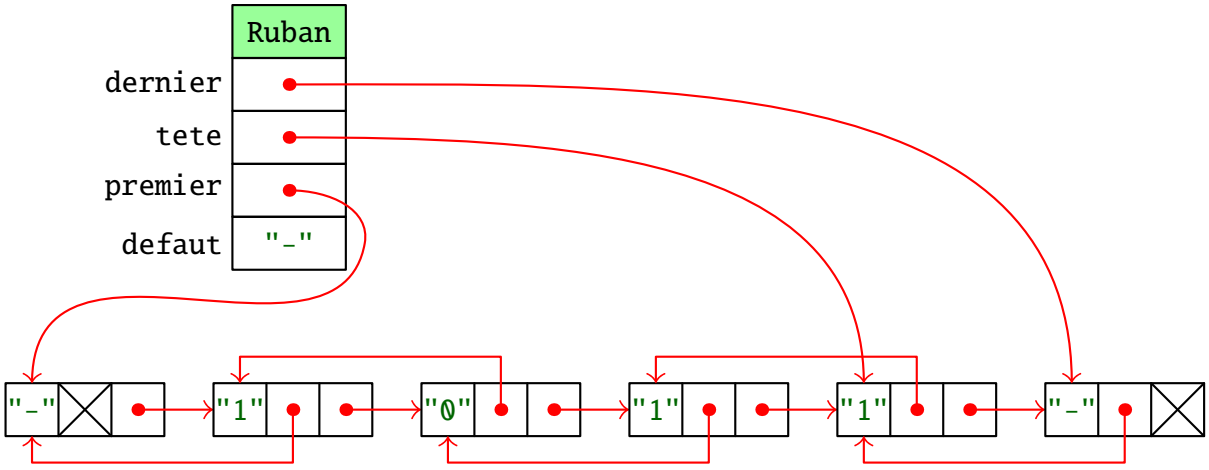
Le ruban est représenté par une liste doublement chaînée qui permet de se déplacer facilement à gauche ou à droite et d'ajouter des cases si nécessaires. Chaque case sera représenté par une `Cellule` :

```
class Cellule:
    def __init__(self, valeur, gauche, droite):
        self.valeur = valeur
        self.gauche = gauche
        self.droite = droite
```

La classe `Ruban` est définie ainsi :

```python
class Ruban:
    def __init__(self, contenu, position_tete, defaut="-"):
        self.defaut = defaut  # Le symbole par défaut du ruban
        case_actuelle = None
        for i in range(len(contenu)):
            nouveau = Cellule(contenu[i], case_actuelle, None)
            if gauche is None: # On est sur la première case
                self.premier = nouveau
            else: # On rajoute le nouveau à droite
                case_actuelle.droite = nouveau
            if i == position_tete: # On est arrivé à la position de départ
                self.tete = nouveau
            case_actuelle = nouveau
        self.dernier = nouveau # On a mis la dernière case

    def a_gauche(self):
        if self.tete.gauche is None: # On doit créer une nouvelle case
            self.tete.gauche = Cellule(self.defaut, None, self.tete)
            self.premier = self.tete.gauche # qui devient la première case
        self.tete = self.tete.gauche # On y va

    def a_droite(self):
        ...

    def lecture(self):
        return self.tete.valeur

    def ecriture(self, valeur):
        ...

    def afficher(self):
        ligne1 = "" # La ligne du contenu
        ligne2 = "" # La ligne avec la tête
        case_actuelle = self.premier
        while case_actuelle is not None:
            ligne1 += str(case_actuelle.valeur)
            if case_actuelle is self.tete:
                ligne2 += "^"
            else:
                ligne2 += " "
            case_actuelle = case_actuelle.droite
        print(ligne1)
        print(ligne2)

    def verifier(self, contenu):
        case_actuelle = self.premier
        for symbole in contenu:
            ...
        return ...
```

Le ruban `Ruban("-1011-", 4)` correspond au schéma suivant :



Exercice 1 : Compléter la méthode `a_droite(self)` qui déplace la tête à droite sur le ruban. S'il n'y a pas de case à droite, il faut en créer une en mettant pour valeur `self.defaut`. Dans ce cas, il faut aussi penser à mettre à jour `self.dernier`.

```
>>> r = Ruban("-1011-", 4)
>>> r.afficher()
-1011-
    ^
>>> r.a_droite()
>>> r.afficher()
-1011-
     ^
>>> r.a_droite()
>>> r.afficher()
-1011--
      ^
```

Exercice 2 : Compléter la méthode `ecriture(self, valeur)` qui place le symbole `valeur` à l'emplacement de la tête.

```
>>> r = Ruban("-001-", 3)
>>> r.afficher()
-001-
   ^
>>> r.ecriture("0")
>>> r.afficher()
-000-
   ^
```

Modélisation des programmes

Les programmes sont représentés par des listes de quintuplets :

```
(etat1, symbole_lu, symbole_ecrit, direction, etat2)
```

Avec :

- `etat1` est un entier qui correspond à l'état de la machine avant l'instruction. L'état initial est 1 par défaut.
- `symbole_lu` est un texte d'un seul caractère. Pour cette feuille, on n'utilisera que `"0"`, `"1"` et `"-"`.
- `symbole_ecrit` est un texte d'un seul caractère.
- `direction` est un texte pouvant être `"gauche"` ou `"droite"`.
- `etat2` est un entier qui correspond à l'état de la machine après l'instruction.

Par exemple le programme ci-contre se traduit de la manière suivante :

```
prog = [(1, "1", "1", "droite", 1),
        (1, "0", "1", "gauche", 2),
        (2, "-", "-", "droite", 3),
        (2, "1", "1", "gauche", 2),
        (3, "1", "-", "droite", 1)]
```

État init.	Lu	Écrit	Dir.	État suiv.
q_1	1	1	$\rightarrow$	q_1
q_1	0	1	$\leftarrow$	q_2
q_2	–	–	$\rightarrow$	q_3
q_2	1	1	$\leftarrow$	q_2
q_3	1	–	$\rightarrow$	q_1

Afin de faciliter la saisie, nous allons utiliser la fonction suivante :

```
def compilation(prog):
    programme = []
    for ligne in prog:
        etat, lu, ecrit, direction, etat2 = ligne.split()
        e = int(etat)
        e2 = int(etat2)
        if direction == "G":
            direct = "gauche"
        else:
            direct = "droite"
        programme.append((e, lu, ecrit, direct, e2))
    return programme
```

Chaque instruction est donnée sous la forme d'un seul texte où `G` et `D` correspondent aux directions. Voici la traduction du programme précédent.

```
compacteur = ["1 1 1 D 1",
              "1 0 1 G 2",
              "2 - - D 3",
              "2 1 1 G 2",
              "3 1 - D 1"]

prog_compact = compilation(compacteur)
```

Implantation de la machine

La machine est définie par une classe `Machine` qui contiendra un ruban, un état et deux autres attributs qui servent à stocker le nombre d'étapes exécutées et le nombre de fois où on a remplacé une valeur par une autre valeur sur le ruban.

```python
class Machine:
    def __init__(self, contenu_Ruban, position, etat_init=1):
        self.etat = etat_init
        self.ruban = Ruban(contenu_Ruban, position)
        self.ecritures = 0 # Pour compter le nombre d'écriture
        self.etapes = 0 # Pour contrer le nombre d'étapes

    def afficher(self):
        print("etat",self.etat)
        self.ruban.afficher()

    def executer(self, programme):
        for instr in programme:
            (etat, lu, ecrit, direction, etat2) = instr
            if ...: # On peut exécuter cette instruction
                ... # Il faut écrire et se déplacer
                if ecrit != lu: # On a changé la valeur
                    self.ecritures += 1
                self.etat = ...  # Nouvel état
                self.etapes += 1 # On met à jour
                return True
        return False

    def verifier(self, contenu):
        return self.ruban.verifier(contenu)
```

Exercice 3 : Compléter la méthode `executer(self, programme)` qui exécute la première instruction de `programme` qui correspond à l'état et à la valeur se trouvant sous la tête de lecture. La fonction renvoie un booléen indiquant si une instruction a pu être exécutée ou pas.

```
>>> m = Machine("-1011-", 1, 1)
>>> m.afficher()
etat 1
-1011-
 ^
>>> m.executer(prog_compact)
True
>>> m.afficher()
etat 1
-1011-
  ^
>>> m.executer(prog_compact)
True
>>> m.afficher()
etat 2
-1111-
 ^
>>> m.executer(prog_compact)
True
```

Au bout de plusieurs étapes, le programme fini par arriver dans un état final.

```
>>> m.executer(prog_compact)
True
>>> m.executer(prog_compact)
False
>>> m.afficher()
etat 1
--111-
     ^
```

La fonction `execution(programme, position, contenu_debut)` permet d'exécuter un programme jusqu'à ce qu'il se termine. Chacune des étapes est affichée.

```
def execution(programme, position, contenu_debut):
    m = Machine(contenu_debut, position, 1)
    continuer = True
    m.afficher()
    while continuer:
        continuer = m.executer(programme)
        m.afficher()
    print(f"Étapes : {m.etapes}")
    print(f"Écritures : {m.ecritures}")
```

Exercice 4 : Tester la fonction avec le programme `prog_compact`, la position 1 et le contenu `"-1011-"`.

Exercice 5 : Compléter la méthode `verifier(self, contenu)` de la classe `Ruban` qui renvoie un booléen indiquant si le contenu du ruban est exactement celui de `contenu`.

```
>>> ruban = Ruban("-1101-", 1)
>>> ruban.verifier("-1101-")
True
>>> ruban.verifier("-1011-")
False
>>> ruban.verifier("-1101--")
False
>>> ruban.verifier("-1011")
False
>>> ruban.verifier("1011")
False
```

Une fois cette méthode complétée, il est possible d'utiliser la méthode `verifier` de la classe `Machine`.

```
>>> m = Machine("-1011-", 1, 1)
>>> m.verifier("-1011-")
True
>>> m.verifier("-1011001-")
False
```

Afin de tester les programmes pour la machine de Turing, nous allons faire une fonction qui prend un programme et un jeu de tests et affiche le résultat de chaque test. Les tests sont de la forme `(position_initiale, contenu_initial, contenu_final)`. Les tests pour le programme précédent est :

```
tests_compact = [(1, "-1011-", "--111-"),
                 (1, "-01101-", "---111-"),
                 (1, "-10010-", "----11-")]
```

Exercice 6 : Compléter la fonction `validation(programme, liste_tests)` qui affiche un texte pour chaque test de `liste_tests` indiquant si `programme` valide le test ou pas.

```
def validation(programme, liste_tests):
    for i in range(len(liste_tests)):
        position, contenu_debut, contenu_fin = liste_tests[i]
        m = Machine(contenu_debut, position, 1)
        ...
```

```
>>> validation(prog_compact, tests_compact)
Test 0 réussi
Test 1 réussi
Test 2 réussi
```

Exercice 7 : Le programme ci-contre permet d'inverser le bit le plus à gauche sur le ruban. On suppose que la tête se trouve initialement sur ce bit ou à sa droite. On pourra, par exemple, prendre le test : `(3, "-101-", "-001-")`.

État init.	Lu	Écrit	Dir.	État suiv.
q_1	0	0	$\leftarrow$	q_1
q_1	1	1	$\leftarrow$	q_1
q_1	–	–	$\rightarrow$	q_2
q_2	0	1	$\rightarrow$	q_3
q_2	1	0	$\rightarrow$	q_3

1) Traduire le programme ci-contre pour pouvoir le tester.
2) Écrire plusieurs tests pour ce programme.
3) Tester le programme avec votre jeu de tests.

Pour aller plus loin

Exercice 8 : Écrire et tester les programmes solutions des exercices 3 à 8 de la feuille sur les machines de Turing.