Tle

Serge Bays

PRÉPAS SCIENCES

COLLECTION DIRIGÉE PAR BERTRAND HAUCHECORNE

SPÉCIALITÉ

NUMÉRIQUE ET SCIENCES INFORMATIQUES

2e édition

- Cours complet et détaillé
- Nombreux exemples
- Vrai/Faux
- Exercices avec indications
- Corrections et commentaires

ellipses

Chapitre 1
Notion de programme

Alan Turing a eu l'intuition géniale de concevoir une machine capable par la pensée de préciser la notion de procédé calculable. Quelques années plus tard, apparaissaient les premiers ordinateurs dont l'objet essentiel est de calculer, de trier, d'ordonner. Contrairement aux appareils construits jusque-là, les utilisations des ordinateurs se multiplient grâce à la conception de programmes de plus en plus variés.

Un scientifique

Stephen Kleene (1909-1994) passe une thèse de logique mathématique en 1934 à l'université de Princeton sous la direction d'Alonzo Church. Il enseigne alors à Madison dans le Wisconsin de 1935 jusqu'à sa retraite en 1979. Ses travaux concernent la logique mathématique et il participe avec Church et Von Neumann à l'éclosion de la science informatique.

LE SAVIEZ-VOUS ?

Durant la Seconde Guerre mondiale, les Allemands utilisaient *Enigma*, une machine prétendue inviolable, pour envoyer des messages secrets. Pourtant les services secrets britanniques, parmi lesquels Alan Turing joua un rôle primordial, sont parvenus à déchiffrer de nombreux messages ce qui donna un avantage considérable aux Alliés dans le déroulement de la guerre.

Les notions présentées dans ce chapitre sont à la base de l'informatique, en particulier des programmes.

OBJECTIFS

- Comprendre ce qu'est un programme :
  - distinguer un fichier programme et un fichier de données ;
  - savoir quelle est l'utilité d'un interpréteur et d'un compilateur ;
  - comprendre le principe d'une machine de Turing ;
  - établir une correspondance entre un programme très simple et une machine de Turing.
- Appréhender la notion de calculabilité :
  - établir la correspondance avec les machines de Turing ;
  - faire le lien avec la notion de décidabilité ;
  - connaître la question soulevée par le problème de l'arrêt.
- Assimiler le principe de la récursivité :
  - comprendre la notion ;
  - parvenir à formuler une solution récursive à un problème ;
  - être capable de transformer un programme itératif en un programme récursif et réciproquement ;
  - reconnaître un programme récursif terminal.

Cours

On peut considérer Alan Turing (1912-1954) comme le père de l'informatique. Cette science en effet est née à partir de ses nombreux travaux dans les années 1930 et 1940. Mais il n'est pas seul dans cette aventure. Les recherches de Church et Kleene, Mauchly, Eckert et von Neumann, ont aussi grandement contribué à l'éclosion de la science informatique.

Turing s'est penché en particulier sur le problème de la *calculabilité* et a fait le lien avec celui de la *décidabilité* en arithmétique, problème posé en 1928 par le mathématicien David Hilbert. En 1936, il présente la machine de Turing, une expérience de pensée qui permet de préciser la notion de procédé calculable, et à partir de cette notion, de définir clairement ce qu'il appelle un programme. Il démontre l'indécidabilité du problème de l'arrêt en prouvant qu'on ne peut pas répondre à cette question avec un algorithme. Sa démonstration est liée à la notion de calculabilité qu'il définit par ce qui peut se calculer mécaniquement avec un algorithme. La même année mais de manière différente, Alonzo Church démontre aussi ce résultat dans sa thèse. À cette fin, il utilise principalement des notions de logique développées par Gödel.

Plus tard, Alan Turing participe au débat sur l'intelligence artificielle et présente ce qu'on appelle le *test de Turing*.

L'objet central de la science informatique est le calcul. Les additions sont bien sûr beaucoup plus anciennes mais déjà en ces temps, on utilisait un algorithme qui était exécuté mentalement. Puis on l'a écrit sur papier et ensuite programmé dans une calculette. Un algorithme ne traite pas que des nombres. Il est aussi appliqué à toutes sortes d'objets. Dans tous les cas, un algorithme, et en particulier un calcul sur des nombres, est une succession de tâches simples appelées opérations, à exécuter dans un ordre précis et qui va produire un résultat après un nombre fini d'étapes. Un programme utilisable sur machine est alors une description de l'algorithme dans un langage (de programmation) compréhensible par la machine.

Un programme

Caractéristiques d'un programme

- Un programme est écrit dans un fichier. Ce fichier peut être édité et lu. Le contenu peut être du texte plus ou moins compréhensible, le code source, qui est une suite d'instructions écrites dans un langage de programmation. Avec un langage comme Python, ces instructions sont interprétées. Elles sont traduites en des instructions en binaire, en langage machine. Dans le cas d'un langage nécessitant une compilation, un fichier exécutable est créé. Dans la suite, aucune distinction n'est faite entre un fichier code source et un fichier exécutable.

- Un programme peut être exécuté seul ou utilisé par un autre programme. Il est alors considéré comme une donnée par cet autre programme. Ainsi un interpréteur ou un compilateur prend un fichier, le code source d'un programme, en argument.

- On peut distinguer un algorithme écrit dans un pseudo-langage d'un programme écrit dans un certain langage de programmation. Les pseudo-langages qui sont utilisés dans la description d'algorithmes ont souvent de fortes ressemblances avec le langage de programmation utilisé ensuite pour l'implémentation. Ainsi, le langage Python peut être utilisé pour décrire des algorithmes et donc être considéré comme un pseudo-langage de description d'algorithme.

- Un fichier représentant un programme, comme un fichier représentant une image, peut-être

transmis, téléchargé, enregistré sur un support.

On peut considérer plusieurs programmes qui interagissent entre eux et agissent sur des données dont certaines sont d'autres programmes. C'est le cas d'un système d'exploitation installé sur une machine.

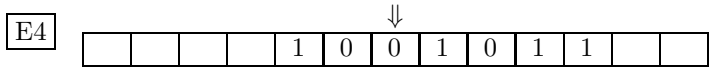
Remarque : un programme est exécuté par une machine. Le temps d'exécution est lié à la complexité du programme, son coût. La calculabilité permet de savoir si une fonction peut être calculée à l'aide d'un algorithme et donc si un problème peut être résolu par une machine ou pas.

■ Machine de Turing

Dans un papier publié en 1936, titré « On computable numbers, with an application to the Entscheidungsproblem », (« Sur les nombres calculables, avec une application au problème de la décision »), Turing décrit une machine qui effectue des calculs de manière mécanique, sans intervention de l'homme à part pour l'entrée des données et la lecture des résultats. Il décrit ensuite une machine universelle capable de simuler n'importe quelle machine particulière.

Cette machine abstraite imaginée par Turing comporte un ruban infini, une tête de lecture et écriture et une table de transition. Le ruban est divisé en cases qui contiennent chacune un symbole d'un alphabet fini. Par défaut les cases contiennent le symbole « blanc ». Le ruban peut se déplacer d'une case vers la gauche ou vers la droite et la tête peut lire ou écrire un symbole sur la case du ruban qui lui fait face. L'état de la machine décide du déplacement. Le nombre d'états possibles est fini et il y a un état de départ. La table de transition indique l'action à exécuter lorsque la tête lit une case du ruban en fonction de l'état courant de la machine : quel symbole écrire, comment se déplacer, et quel est le nouvel état. Cette machine représente d'une certaine manière un programme.

On peut la représenter comme ci-dessous avec une double flèche qui symbolise la tête de lecture et d'écriture au dessus du ruban, face à une case. Les symboles utilisés sont les nombres 0 et 1. La machine est par exemple dans l'état E4.



Remarque : une machine de Turing ressemble aux anciennes machines à écrire mécaniques. Ces machines avaient un ruban avec de l'encre. L'appui sur une touche du clavier mécanique comprimait le ruban contre le papier avec le caractère en relief choisi, et imprimait ainsi la forme du caractère sur le papier. Les différences principales avec une machine de Turing : le ruban n'était pas infini et la machine ne lisait pas ce qui était écrit, c'était le rôle de l'humain.

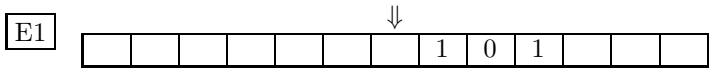
Nous allons voir deux exemples. Les tableaux qui suivent présentent des tables de transition. Dans la première colonne, nous trouvons les différents états et dans la deuxième colonne, ce que lit la machine. Nous trouvons ensuite ce qu'écrit la machine dans la case courante, puis le déplacement et le nouvel état. Le contenu des trois dernières colonnes dépend du contenu des deux premières.

Exemple 1

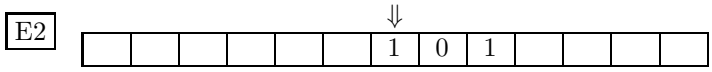
État	Lecture	Écriture	Déplacement	État suivant
E1	blanc	blanc	gauche	E2
E2	0	1	gauche	E2
	1	0	gauche	E2
	blanc	blanc	gauche	Fin

La machine peut être dans deux états E1 et E2. Si la machine est dans l'état E1 et contient un blanc, elle écrit un blanc. Le ruban se déplace vers la gauche et la machine passe dans l'état E2. Si la machine est dans l'état E2 et qu'elle lit un 0, elle écrit un 1 et si elle lit un 1, elle écrit un 0. Dans les deux cas le ruban se déplace vers la gauche et la machine reste dans l'état E2. Enfin si elle lit un blanc elle écrit un blanc, le ruban se déplace vers la gauche et la machine s'arrête.

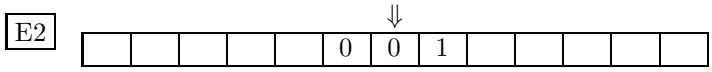
Les étapes successives sont représentées ci-dessous. Initialement la machine est dans l'état E1.



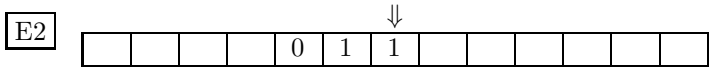
La machine lit un blanc, donc écrit un blanc, passe dans l'état E2 et le ruban se déplace vers la gauche.



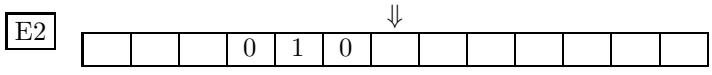
La machine lit un 1, donc écrit un 0, reste dans l'état E2 et le ruban se déplace vers la gauche.



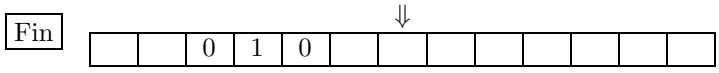
La machine lit un 0, donc écrit un 1, reste dans l'état E2 et le ruban se déplace vers la gauche.



La machine lit un 1, donc écrit un 0, reste dans l'état E2 et le ruban se déplace vers la gauche.



La machine lit un blanc, donc écrit un blanc, le ruban se déplace vers la gauche et la machine s'arrête.



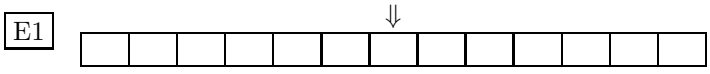
Exemple 2

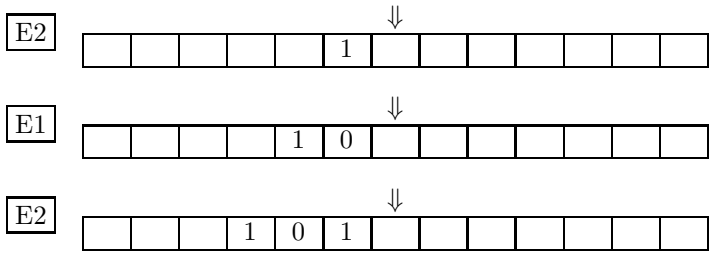
Dans cet exemple la machine n'a que deux états.

État	Lecture	Écriture	Déplacement	État suivant
E1	blanc	1	gauche	E2
E2	blanc	0	gauche	E1

Si la machine est dans un état E1, elle écrit un 1, le ruban se déplace vers la gauche et la machine passe dans un état E2. Si la machine est dans l'état E2, elle écrit un 0, le ruban se déplace vers la gauche et la machine passe dans l'état E1.

Les étapes successives sont représentées ci-dessous.





Le ruban étant infini, la machine écrit 10101010..., avec autant de chiffres que l'on souhaite. En ajoutant un point devant ce nombre, on obtient le nombre 0.101010... écrit en binaire qui représente la somme $1 \times 1/2 + 0 \times 1/4 + 1 \times 1/8 + 0 \times 1/16 + \ldots$ On montre que cette somme vaut $2/3$.

D'une certaine manière, une machine de Turing est un programme. Nous pouvons donc écrire en langage Python un programme correspondant à une machine de Turing particulière. Les paramètres en entrées sont un ruban représenté par une liste, le numéro de la case lue et un état. Chaque case du ruban est un élément de la liste. Le numéro de la case est l'indice de l'élément. La machine commence par lire la case d'indice 0 pour les deux exemples. Bien sûr, la liste n'est pas infinie comme le ruban.

Pour les exemples des deux machines précédentes, nous utilisons une boucle `while`. Tant que la machine n'est pas dans un état final, on applique les règles données par sa table de transition. Le symbole « blanc » est représenté par la valeur `None`. On pourrait aussi le représenter par le caractère `"b"`.

Commençons par la machine 2, dont la table de transition décrite dans l'exemple 2 est la plus simple, avec un ruban où il n'y a rien d'écrit.

```
def machine2(ruban, i, etat):
    while i < len(ruban):
        if etat == 1:
            if ruban[i] == None:
                ruban[i] = 1
                i = i + 1
                etat = 2
        elif etat == 2:
            if ruban[i] == None:
                ruban[i] = 0
                i = i + 1
                etat = 1
        else:
            i = len(ruban) # utile s'il y a une erreur sur l'état initial

# Pour tester
r = 20 * [None] # le ruban est initialisé (il ne contient que des blancs)
machine2(r, 0, 1) # conditions initiales: case 0 du ruban et état E1
print(r)
```

L'affichage donne : `[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]`.

Une autre manière de procéder est d'écrire une fonction décrivant la table de transition en imaginant qu'on appuie ensuite sur un bouton à chaque étape. Autrement dit, s'il y a n passages dans la boucle `while`, cela revient à appuyer n fois sur le bouton, soit à appeler n fois la machine.

```
def transition2(ruban, i, etat):
    if etat == 1:
        if ruban[i] == None:
            ruban[i] = 1
    elif etat == 2:
        if ruban[i] == None:
            ruban[i] = 0

def machine2(r):
    i, e = 0, 1
    etats = [2, 1]
    while i < len(r):
        transition2(r, i, e)
        i = i + 1
        e = etats[e-1]

r = 20 * [None]
machine2(r)
print(r)
```

L'affichage obtenu est le même que précédemment.

Finalement, on peut imaginer qu'une machine n'appelle pas la fonction de transition externe à chaque étape, mais que la fonction de transition est pratiquement la machine et donc que la machine s'appelle elle-même. C'est le principe de récursivité qui est exposé dans la section suivante.

```
def machine2(ruban, i, etat):
    if i < len(ruban):
        if etat == 1:
            if ruban[i] == None:
                ruban[i] = 1
                machine2(ruban, i+1, 2)
        elif etat == 2:
            if ruban[i] == None:
                ruban[i] = 0
                machine2(ruban, i+1, 1)

r = 20 * [None]
machine2(r, 0, 1)
print(r)
```

Encore une fois, l'affichage est identique.

Considérons maintenant la machine 1 dont la table de transition est décrite dans l'exemple 1.

Le programme simulant la machine 1, ci-dessous, est écrit en suivant le modèle de la dernière version du programme simulant la machine 2. Ce programme est ensuite exécuté après l'exécution du programme précédent, donc avec le ruban sorti de la machine 2.

```
def machine1(ruban, i, etat):
    if i < len(ruban):
        if etat == 1:
            if ruban[i] == None:
                machine1(ruban, i+1, 2)
            else:
                machine1(ruban, i, 2)
        elif etat == 2:
            if ruban[i] == 0:
                ruban[i] = 1
                machine1(ruban, i+1, 2)
            elif ruban[i] == 1:
                ruban[i] = 0
                machine1(ruban, i+1, 2)
            else:
                i = len(ruban) # pour arrêter la machine

machine1(r, 0, 1)
print(r)
```

L'affichage donne : `[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]`. La machine a donc inversé les bits qui étaient écrits sur le ruban.

Nous convenons pour la suite qu'une machine de Turing est un modèle abstrait de programme.

■ Machine de Turing universelle

Turing a montré qu'il existe une machine de Turing universelle qui peut simuler n'importe quelle autre machine de Turing et donc exécuter n'importe quel programme. Pour cela, la table de transition, le programme d'une machine particulière, est donnée à la machine universelle. Ce programme devient donc une donnée de la machine universelle, codée sur le ruban qu'elle va lire. Nous avons déjà dans les idées de Turing la notion de programme enregistré qui sera reprise par von Neumann : les données et les programmes sont représentés de la même manière (sur le ruban) et un programme peut être une donnée d'un autre programme, ce qui sera le cas avec les interpréteurs et compilateurs. La machine universelle est donc un modèle abstrait d'ordinateur, autrement dit elle décrit le fonctionnement d'un ordinateur.

Comme cela a été fait pour des machines particulières, nous pouvons écrire un programme en Python qui « simule » une machine de Turing universelle. La machine prend en argument une machine particulière `m`, (une fonction en Python), et un ruban `r` (représenté par une liste). La correspondance est loin d'être parfaite. Il s'agit juste d'aider à se faire une idée.