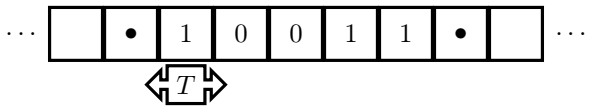
Calculabilité, décidabilté

Histoire de la calculabilité

1 Calculabilité

Une *machine de Turing* est un système comportant deux composants : un ruban infini muni d'une tête de lecture/écriture.

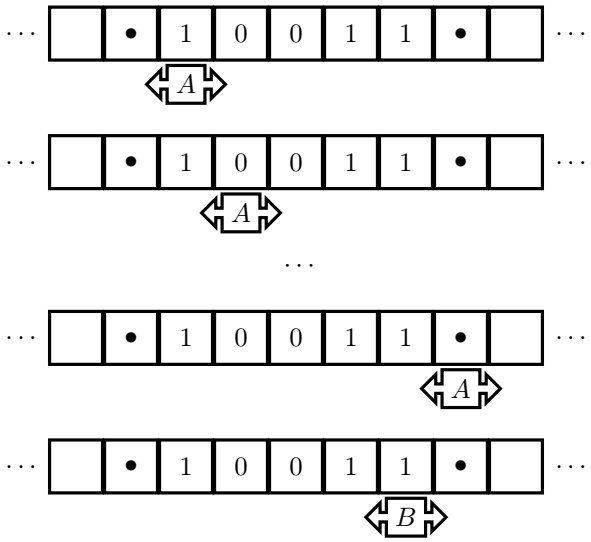


Le ruban est divisé en cases qui contiennent chacune un symbole d'un ensemble fini appelé *alphabet*. On prendra typiquement comme alphabet les symboles 0, 1 et • («blanc») où 0 et 1 représentent des bits et • indique une case vide. La tête peut se déplacer dune case vers la gauche ou la droite et peut lire ou écrire sur la case du ruban qui lui fait face. Le programme réalisé par une machine de Turing est décrit par un ensemble de *règles de transition*.

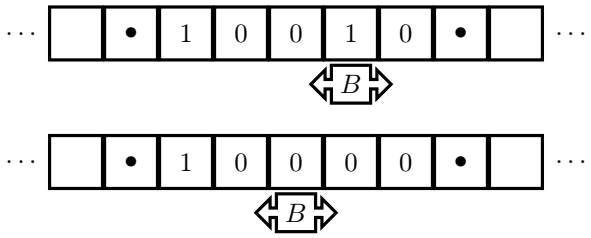
Considérons la table de transition suivante :

état	lu	écrit	dépl.	suiv.
A	0	0	→	A
A	1	1	→	A
A	•	•	←	B
B	0	1	↓	F
B	1	0	←	B
B	•	1	↓	F

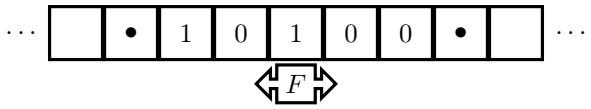
L'état A va déplacer la tête tout à droite puis passer à l'état B :



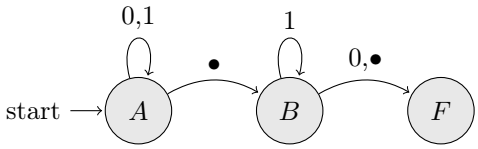
Dans cet état, tant que la machine observe es 1 elle les rempalces par des 0 et poursuit son retour vers la gauche :



Ce retur en arrière s'arrête dès que la machine lit un 0 ou un •, auquel cas ce symbole est remplacé par 1 et la machine passe à son état final F et s'arrête.



La valeur initial sur le ruban était 10011, c'est à dire 19, et le résultat est 10100, c'est à dire 20. La machine de Turong donnée par la table de transition incrémente de 1 l'entier écrit sur le ruban. Cette machine peut être représenté par un *automate* :



On peut voir une machine de Turing comme calculant le résultat d'une fonction f de paramètre e écrit sur le ruban. Une fonction calculable selon Turing est alors définie comme une fonction pour laquelle il existe une machine.

2 Décidabilité

En algorithmique, un *problème de décision* est une question à laquelle on répond par oui ou par non. On dit qu'un problème est *décidable* s'il existe un algorithme qui permet de répondre par oui ou par non à la question posée par le problème. Sinon on dit que le problème est *indécidable*.

Un exemple de problème de décision, est de déterminer si un nombre entier `n` est pair ou non. Le programme suivant répond à cette question :

```
def pair(n):
    while n>0:
        n = n-2
    return n==0
```

Le problème est donc décidable.

Le *problème de l'arrêt* est le premier problème indécidable exposé par Alan Turing en 1936. Il a démontré qu'il ne pouvait exister un algorithme `arret(f, e)` qui prend en paramètre un algorithme `f` et son entrée `e`, et déterminant si `f` se termine avec l'entrée `e` ou se poursuit indéfiniment.

Supposons qu'une telle fonction Python existe.

```
def arret(f, e):
    """True si la fonction f termine avec l'entrée e False sinon."""
    pass
```

Nous pouvons alors construire une nouvelle fonction `paradoxe(n)`, qui prend un entier `n` comme paramètre, à partir de `arret` de la manière suivante :

```
def paradoxe(n):
    if arret(paradoxe, n):
        while True:
            pass
    else:
        return n
```

Que ce passe-t-il alors lorsque l'on exécute `paradoxe(0)` ? De deux choses l'une :

- L'exécution se termine et dans ce cas `arret(paradoxe, 0)` va renvoyer `True` ce qui va déclencher une boucle infinie et donc `paradoxe(0)` ne s'arrête pas. On a une contradiction.
- L'exécution ne se termine pas et dans ce cas `arret(paradoxe, 0)` va renvoyer `False` et donc `paradoxe(0)` va renvoyer `0`. On a à nouveau une contradiction.

On en déduit qu'il ne peut exister un algorithme tel que `arret`.