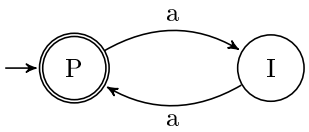
Calculabilité

Automates finis

Question 1 Ce sont les mots sur l'alphabet $\{a, b\}$ qui se terminent par un a.

Question 2



Question 3

```
def simule_automate(transitions, initial, mot):
    état = initial
    for symbole in mot:
        état = transitions[état][symbole]
    return état
```

Question 4 On n'a pas trop le choix: on applique une construction qui fonctionne pour n'importe quel langage fini. On se donne un état pour chaque préfixe des mots qu'on veux accepter:

- ϵ (le mot vide),
- a^i pour $1 \leq i \leq 10$,
- et $a^n b^i$ pour $1 \leq i \leq n \leq 10$.

Et on ajoute un état E qui sert de poubelle.

L'état initial est ϵ, et il est acceptant (c'est $a^0 b^0$). Chaque état $a^n b^n$ pour $1 \leq i \leq n$ est acceptant.

On pose $\delta(w, c) = wc$ si c'est un état valide (un préfixe d'un mot qu'on veut accepter) et $\delta(w, c) = E$ sinon. Autrement dit:

$$\delta : \begin{cases} (a^i, a) \mapsto a^{i+1} & \text{si } i < 10 \\ (a^n b^i, b) \mapsto a^n b^{i+1} & \text{si } i < n \leq 10 \\ (w, c) \mapsto E & \text{dans tous les autres cas} \end{cases}.$$

Question 5 Notons $\Delta(e, w)$ l'état obtenu en lisant le mot w à partir de l'état e: par définition, si $w = uv$ alors $\Delta(e, w) = \Delta(\Delta(e, u), v)$.

Fixons $n > 0$ et supposons que le mot $a^n b^n$ est accepté par un automate avec exactement n états. On sait que $\Delta(I, a^n b^n)$ est acceptant. Comme la suite d'états $\Delta(I, \epsilon), \Delta(I, a), \Delta(I, a^2), \ldots, \Delta(I, a^n)$ est de longueur $n + 1$ et qu'il y a n états possibles, on a forcément une répétition: disons $\Delta(I, a^i) = \Delta(I, a^j)$ avec $0 \leq i < j \leq n$.

Alors $\Delta(I, a^{n-j+i} b^n) = \Delta(\Delta(a^i), a^{n-j} b^n) = \Delta(\Delta(a^j), a^{n-j} b^n) = \Delta(I, a^n b^n)$ et donc l'automate accepte a^k avec $k = n - j + i < n$.

Donc, pour tout $n \in \mathbf{N}$, un automate à n états qui accepte $a^n b^n$ doit aussi accepter un mot de la forme a^k. Et donc le langage des mots $a^n b^n$ ne peut pas être reconnu par un automate fini.

Calculer avec une machine de Turing

Question 1

```
# On doit d'abord amener la tête sur le bit de poids faible.
0 1 1 r 0
0 0 0 r 0
0 _ _ l I
# À partir de ce point c'est la traduction directe du graphique
I 1 0 l I
I 0 1 r R
I _ 1 r R
R 0 0 r R
R 1 1 r R
R _ _ l F
```

Question 2 Une idée est d'enlever alternativement un a à gauche et un b à droite : le mot est de la bonne forme si et seulement si on parvient au mot vide après avoir enlevé un b.

```
mange_un_a _ _ * oui
mange_un_a a _ r a_droite
mange_un_a b _ * non
a_droite a a r a_droite
a_droite b b r a_droite
a_droite _ _ l mange_un_b
mange_un_b b _ l a_gauche
mange_un_b a _ * non
mange_un_b _ _ * non
a_gauche a a l a_gauche
a_gauche b b l a_gauche
a_gauche _ _ r mange_un_a
```

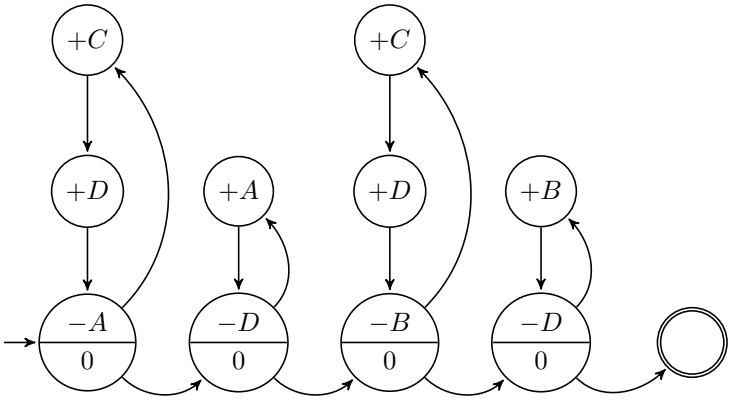
Implémentation des machines de Turing

Questions 1 et 2 Voir les fichiers joints.

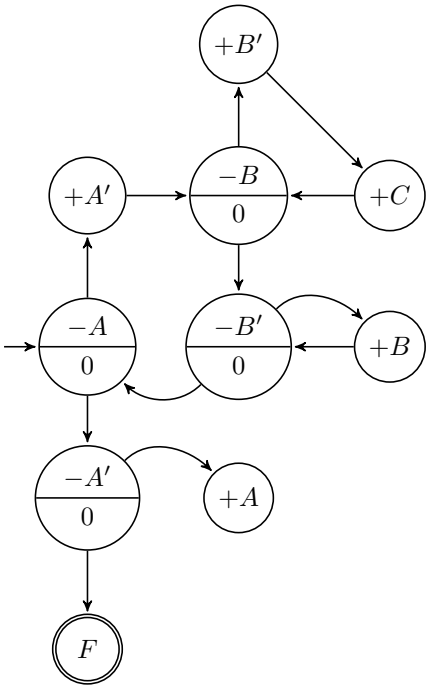
Machines à compteurs

Question 1 La somme, évidemment.

Question 2



Question 3



Questions 4 et 5 Voir les fichiers joints.

Problèmes indécidables: réduction

Question 1 Il suffit de voir que `f(x)` boucle sans rendre de valeur si et seulement si `f` est la fonction qui ne rend jamais de valeur coïncident en `x`.

On peut donc écrire:

```
def arret(f,x):
  def boucle(x):
    while True:
      pass
  return not cestpareil(f,boucle,x)
```

Question 2

Il suffit de voir que `f(x)` renvoie une valeur si et seulement si la fonction constante qui renvoie `f(x)` est totale.

On peut donc écrire:

```
def arret(f,x):
  def constante_f_x(y):
    return f(x)
  return totale(constante_f_x)
```

Question 3 Il suffit de voir que `f(x)` renvoie une valeur si et seulement si la fonction suivante calcule la fonction nulle:

```
def f_puis_zero(y):
  f(x)
  return 0
```

On peut donc écrire:

```
def arret(f,x):
  def f_puis_zero(y):
    f(x)
    return 0
  return implemente(f_puis_zero, lambda x: 0)
```