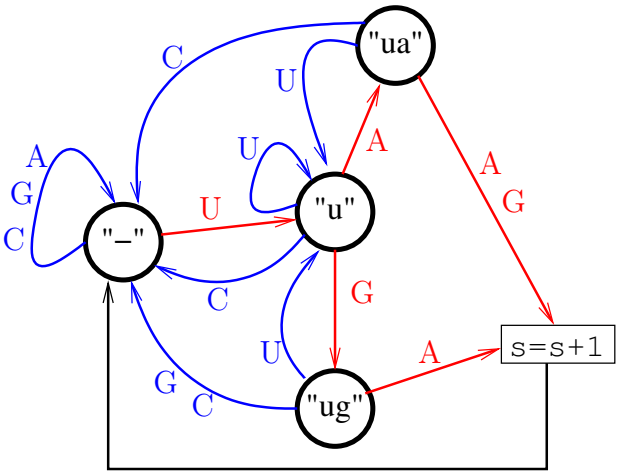
Cours n°9

1 Les automates (et l'instruction manquante en Python)

Les « automates » en informatique désignent en fait une « façon de penser » pour écrire des programmes qui doivent traiter des données qui arrivent les unes après les autres, et desquelles on doit extraire des informations. Cette technique permet de ne lire les données qu'une seule fois.
Par exemple, lorsqu'on programme la recherche de codons STOP dans un brin d'ARN `b`, la technique qui consiste à comparer, pour chaque indice `i`, la tranche de 3 nucléotides `b[i:i+3]` avec les codons STOP (`UAA`, `UAG` ou `UGA`) revient à lire 3 fois chaque nucléotide puisqu'un codon `b[j]` est lu dans les trois tranches `b[j-2:j+1]`, `b[j-1:j+2]` et `b[j:j+3]`.
Si l'on veut n'utiliser qu'une seule fois chaque nucléodite successivement, par conséquent en utilisant un `for` sur le brin `b`, c'est en faisant un dessin que l'on comprend ce qu'il faut faire (figure de droite).



- Au départ, n'ayant encore « lu » aucun nucléotide, on n'a « reconnu » aucun début de codon STOP. On se place alors dans l'état `"-"`qui indique ce fait.
- Depuis cet état où l'on n'a rien reconnu encore, si on lit un `A`, un `G` ou un `C` alors on n'a toujours rien reconnu, puisque tous les codons STOP commencent par `U`.
- En revanche, si l'on a lu un `U`, alors on passe dans un état où l'on a reconnu le premier nucléotide d'un codon STOP. On passe donc de l'état `"-"` à l'état appelé `"u"` sur le dessin, qui signifie conventionnellement qu'on a reconnu le premier nucléotide `U`.
- Depuis cet état `"u"`, si on lit un `C` alors ce n'était pas le début d'un codon STOP et l'on retourne à l'état de départ `"-"` où l'on n'a rien reconnu. En revanche, si on lit un `G` (ou un `A`) alors on passe dans un état où l'on a reconnu 2 nucléotides `"ug"` (ou respectivement `"ua"`). Enfin si on lit un second `U`, alors le premier n'était pas le premier nucléotide d'un codon STOP, mais celui qu'on vient de lire, peut-être... donc on reste dans l'état `"u"`.
- Depuis l'état "ug", si on lit un `G` ou un `C` alors tout est à refaire, on repart de l'état `"-"` où aucun nucléotide n'a été reconnu. Si on lit un `U` alors `UGU` n'est pas un codon STOP mais le `U` qu'on vient de lire est peut-être le premier nucléotide d'un codon STOP. Donc, on retourne dans l'état "u" où l'on a reconnu un seul nucléotide. Enfin, si on lit un `A`, alors on a reconnu le condon STOP `UGA`, donc on ajoute 1 au nombre de codons STOP rencontrés et on repart pour la suite dans l'état `"-"`.
- De même, depuis l'état "ua", si on lit `A` ou `G` alors on a reconnu un codon STOP et l'on ajoute 1 au nombre de codons STOP rencontrés. En revanche si on lit un `C`, il faut rapartie à zéro (état `"-"`). Enfin si on lit un `U`, on repart vers l'état où l'on n'a reconnu qu'un nucléotide (état "u").

À la fin de la boucle `for`, la somme `s` contiendra exactement le nombre de codonc STOP reconnus :

```
>>> def nbSTOP(b):
...     s=0
...     etat="-"
...     for c in b :
...       if etat == "-" :
...         if c == "U" :
...           etat = "u"
...       elif etat == "u" :
...         if c == "C" :
...           etat = "-"
...         elif c == "A" :
...           etat = "ua"
...         elif c == "G" :
...           etat = "ug"
...       elif etat == "ua" :
...         if c == "C" :
```

```
...         etat = "-"
...       elif c == "U" :
...         etat = "u"
...       else :
...         s = s + 1
...         etat = "-"
...     elif etat == "ug" :
...       if c == "U" :
...         etat = "u"
...       elif c == "A" :
...         s = s + 1
...         etat = "-"
...       else :
...         etat = "-"
...     else :
...       print("PROBLEME !")
...   return(s)
```

De ce point de vue, Python est l'un des seuls langages de programmation qui n'offrent pas une instruction de « distribution des calculs en fonction des cas ». On en arrive donc à cette accumulation de `elif` assez inélégante. Les autres langages de programmation impérative permettraient d'écrire l'automate de manière légèrement plus lisible qui pourrait resembler à :

```
def nbSTOP(b):
  s=0
  etat="-"
  for c in b :
    switch etat :
      case "-" :
          switch c :
            case "U" :
              etat = "u"
      case "u" :
          switch c :
            case "C" :
              etat = "-"
            case "A" :
              etat = "ua"
            case "G" :
              etat = "ug"
      case "ua" :
          switch c :
            case "C" :
              etat = "-"
            case "U" :
              etat = "u"
            default :
              s = s + 1
              etat = "-"
      case "ug" :
          switch c :
            case "U" :
              etat = "u"
            case "A" :
              s = s + 1
              etat = "-"
            default :
              etat = "-"
  return(s)
```

Malheureusement ceci n'est pas un programme Python par manque de l'instruction `switch`. Les concepteurs de

Python préconisent d'utiliser les dictionnaires pour encoder l'automate. Ceci reporte essentiellement la lourdeur de programmation dans la définition de ce dictionnaire : on est obligé d'y lister *tous* les cas puisqu'il n'y a pas de « `default` » et il faut tout de même gérer spécifiquement les états qui demandent une actions (comme la mise à jour de `s`) :

```
d = { ("-","A"):"-" , ("-","U"):"u" , ("-","G"):"-" , ("-","C"):"-" ,
      ("u","A"):"ua" , ("u","U"):"u" , ("u","G"):"ug" , ("u","C"):"-" ,
      ("ua","A"):"-" , ("ua","U"):"u" , ("ua","G"):"-" , ("ua","C"):"-" ,
      ("ug","A"):"-" , ("ug","U"):"u" , ("ug","G"):"-" , ("ug","C"):"-" }

def nbSTOP(b):
  s=0
  etat="-"
  for c in b :
    if ( etat == "ug" and c == "A" ) or ( etat == "ua" and c in "AG" ) :
      s = s + 1
    etat = d[(etat,c)]
  return(s)
```