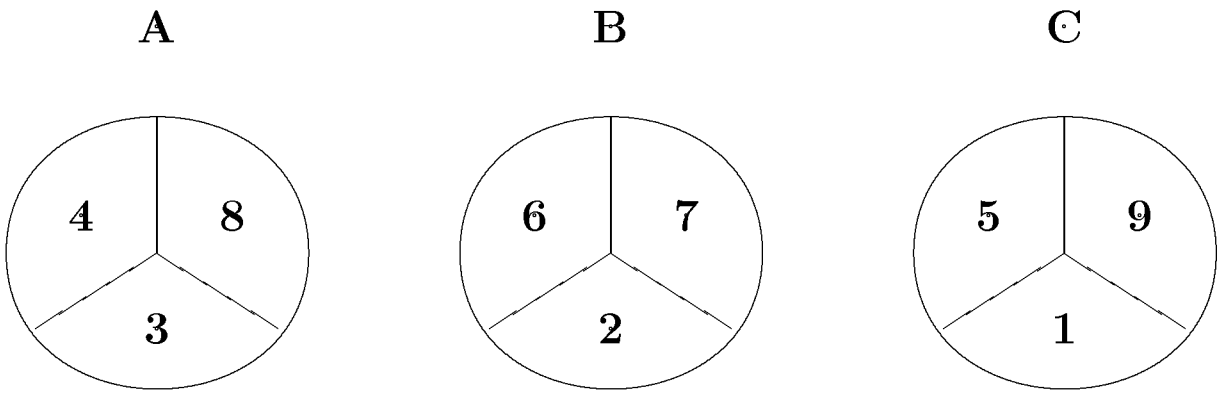
De nouveaux paradoxes de Condorcet

La première partie : présentation du problème avec de premiers exemples
La seconde : les automates d'avancement simultanés dans deux mots
La troisième : programmation Python
La quatrième : version matricielles des automates et usage de Maple
La dernière : ce qui resterait à faire

A les "paradoxes de Condorcet"

1 un exemple classique

Vous avez ci-dessous trois "roues de loterie" (ou dés-à-3-faces faciles à faire avec des autocollants doublés sur des dés à 6 faces)



Deux joueurs vont chacun choisir une roue et lancer sa roue : celui ayant le résultat le plus élevé gagne. De simples comptes montrent que :

1. la roue A est (strictement) meilleure que la roue B : elle gagne 5 fois sur 9
2. la roue B est (strictement) meilleure que la roue C : elle gagne 5 fois sur 9
3. la roue C est (strictement) meilleure que la roue A : elle gagne 5 fois sur 9

Ainsi, celui qui choisit en premier est "sûr de perdre"

2 des jeux de Pile ou Face

Chaque joueur va choisir un mot sur l'alphabet {P,F} (par exemple je les nomme x et y de longueur 5) puis on va aligner 5 lancers de pièces : si on a obtenu x il gagne $\boxed{\text{R}}$ (de même pour y), sinon on retire la première pièce, on la relance et on la met en bout de ligne, si on a obtenu x il gagne GOTO $\boxed{\text{R}}$

Pour éviter la routine PileOuFacique on peut prendre un textes aléatoire (écrit sur un alphabet de deux lettres), on cherchera (par un CTRL-F) quel est le mot qui apparait le premier

Une intuition fausse nous dit que si les deux mots sont de même longueur ils semblent bien être équiprobables

L'exemple de FF et PF montre le contraire : PF gagne 2 fois sur 3 :

- si la première lettre est un P .... 'PF' gagnera lors de l'apparition du premier F

- si la première lettre est un F, celui qui veut 'FF' gagnera 2 fois sur 3 (on détaille cela plus bas)

3 plus généralement

les "paradoxes de Condorcet" sont les instances de "ce qui semble être un ordre" et n'en est pas un

des références : un moteur de recherche + Condorcet ou Dogson = Lewis Caroll ou Arrow ou Jacques Attalli

en gros :

- Condorcet (voulant créer des lois électorales) a compris qu'il y avait un problème
- Dogson a compris que c'était un problème mathematiques et "qu'on prouverait cela un jour"
- Arrow (1930) l'a prouvé
- Jacques Attalli : l'a énormément généralisé

4 ce que l'on étudie ci-dessous

on va chercher trois mots sur l'alphabet {P,F} tels que

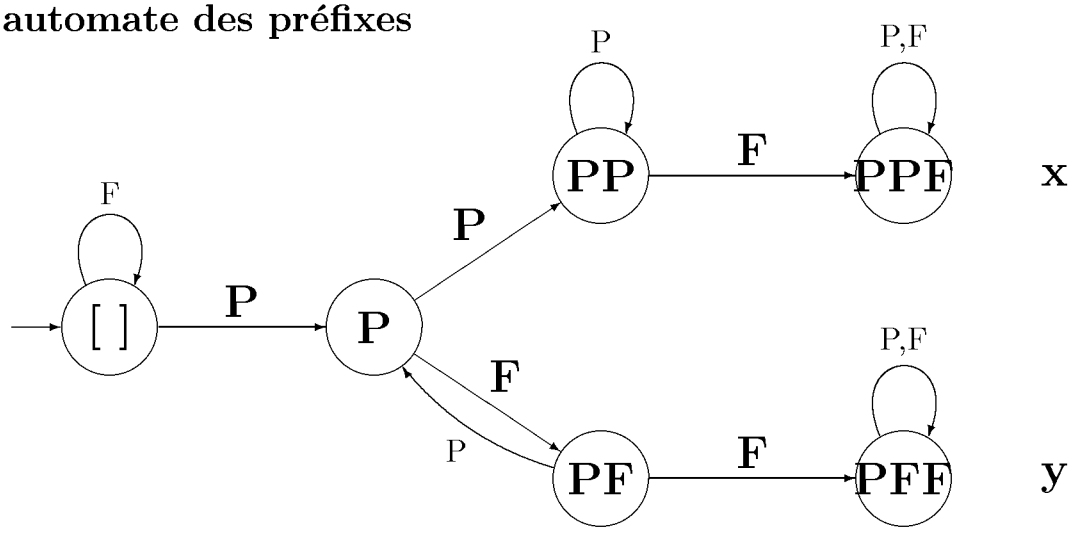
mot1 bat strictement mot2 bat strictement mot3 bat strictement mot1

B tout sur un exemple

on va comparer les premières occurences de PPF et PFF et sur cet exemple tout montrer : l'automate de comparaison, les langages reconnus, les comptes de mots de longueur n qui gagnent (pour chacun des mots) enfin un matrice qui résume tout

Si vous connaissez déjà tout cela (c'est de la routine pour l'Option Informatique) vous pouvez vous demander pourquoi ce premier exemple est si mal choisi

1 l'automate des préfixes



L'objet ci-dessus décrit à chaque instant l'état d'avancement dans la recherche de l'un des mots x=PPF ou y=PFF. Si par exemple on a le mot FFPFPPPPFFP, on part de l'état V=[ ]

1. le premier F ne sert à rien (on n'a avancé ni dans x ni dans y) : on suit la boucle qui nous dit de rester en V

2. le second F ne vaut pas mieux : on reste en V
3. le P qui est en troisième nous enmène en $\boxed{\text{P}}$
4. le F nous fait aller en $\boxed{\text{PF}}$
5. le P nous ramène en $\boxed{\text{P}}$: l'état de notre mot est PFP n'est ni x ni y, FP n'est pas non plus un préfixe de x ni de y, seul le denier P sert à quelque chose)
6. le P qui suit va en $\boxed{\text{PP}}$
7. on y reste avec les P qui suivent
8. enfin un F nous mêne en x=$\boxed{\text{PPF}}$
9. les dernières lettres ne servent à rien

2 les probabilités de gain

On va en dessous de chacun des états mettre les probabilités de (x-gagne,y-gagne). Sous $\boxed{\text{PPF}}$ on a donc (1,0) et sous $\boxed{\text{PFF}}$ on a (0,1) Sous $\boxed{\text{PP}}$ on met (x_{PP},y_{PP}) et comme de $\boxed{\text{PP}}$ on peut aller équiprobablement en $\boxed{\text{PP}}$ ou en $\boxed{\text{PPF}}$ on a (x_{PP},y_{PP})=$\frac{1}{2}$(x_{PP},y_{PP})+$\frac{1}{2}$(0,1), d'où : x_{PP}=1 et y_{PP}=0

De la même façon on met (x_{PF},y_{PF}) sous $\boxed{\text{PF}}$, (x_P,y_P) sous $\boxed{\text{P}}$, ($x_{[]}$,$y_{[]}$) sous $\boxed{[\,]}$.

Le système qui rend compte des équiprobabilités en PF donne (x_{PF},y_{PF})=($\frac{x_P}{2}$,$\frac{y_P+1}{2}$), en P : $x_P = \frac{1+\frac{x_P}{2}}{2}$ d'où $x_P = \frac{2}{3}$, d'où y_P puis : $x_{[]}$=$\frac{2}{3}$, $y_{[]}$=$\frac{1}{3}$

$\boxed{\text{PPF a 2 fois plus de chances de gagner que PFF}}$

3 les langages reconnus

J'utilise sans détailler les notations (qui me semblent) universelles pour les langages réguliers : le langage des mots qui aboutissent en y est F*P(FP)*FF(P + F)*. Pour ceux qui aboutissent en x on a de même : F*P(FP)*PP*F(P + F)*

Quand vous cherchez un mot dans un texte "avec des jokers" c'est à peu près cette sintaxe là (détails pour linuxiens dans `man grep -e`)

4 les comptes de mots gagnants

Les mots de longueur N qui font gagner y sont (je recopie l'expression régulière qui décrit les mots de fin y) au nombre de $n_F + 1 + 2n_{FP} + 1 + 1 + n_{PouF}$ ce qui amène à résoudre (en entiers) : N =a+1+2b+1+1+c ou encore a+2b+c=N-3, pour éviter les formules avec des parties entières on peut discuter selon n pair ou impair

On peut faire de même pour ceux finissant en x

5 la matrice qui résume tout

Les vecteurs de base de mon espace vectoriel sont [] P PF PP PFF PPF, la matrice M

$$M=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 2 \end{pmatrix}$$

donne dans chaque colonne C le nombre des façon d'obtenir lesdits vecteurs de base en partant de l'état C : la colonne 4 me dit qu'en partant de C=PP je peux 1 fois aboutir à PP et 1 fois à PPF

Quand je vais partir de [] = colonne[1 0 0 0 0 0] = ColonnedeVide = CV

1. M va m'envoyer en [] ou P : M.CV c'est la première colonne de M
2. M^2 me fait arriver en [] P PF ou PP (somme des colonnes 1 et 2, première colonne de M^2)
3. M^3 me donnera colonne[1 2 1 2 1 0 1] j'ai 1 possibilité d'être resté en [] (avec le mot FFF) 2 manières d'aller en PP (avec FPP et avec PPP)
4. etc ..

ainsi M^n appliquée à CV (autrement dit la première colonne de M^n) me donne le nombre de façons d'aboutir à l'un de mes 6 vecteurs de base : ce sont bien sûr les 4-ième et 5-ième qui intéressent, ils traduisent "x gagne" et "y gagne"

pour calculer M^n on va la diagonaliser .... bing, justement elle n'est pas diagonalisable, et selon ce que j'ai trouvé dans dans scipy et sympy .... il n'y a pas de commande **jordan** dans Python

je suis donc revenu à mon vieux Maple, je lui ai dit `with(LinearAlgebra)` je lui ai donné M , demade JordanForm(M), il m'a répondu

$$\text{J}=\begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

c'est Maple qui a choisi l'ordre des vecteurs

6 pourquoi cet exemple est-il peu malin ...

... à cause du P commun au départ : tant qu'on lit des F on n'a pas démarré, quand on lit un P les deux avancent de la même façon, finalement on compare PF et FF ... c'est justement ce dont on avait parlé dans le § "des jeux de Pile ou Face"

C expérimentation avec Python

1 quelques commentaires

Je remercie par avance le Pythoneurs expérimentés pour les reproches qu'ils feront

`daniel.goffinet42@gmail.com`

sur mon "style de Python"

Chacune des fonctions est suivie d'un commentaire (ligne à #) donnant un exemple d'usage

J'ai écrit `fuse_ordre` et `fuse_ordreDec` alors que le premier suffisait (en utilisant not(ordre)) ... mais je m'étais trop trompé

Avec ce `fuse_ordre` on est à la distance de deux fonctions monolignes d'une fonction de tri-fusion

2 le programme

```
# Automates de Pile ou Face

Ess1='PPPFFF'
Ess5='FPPPPP'
Ess6='FFPPFP'

mot1=Ess5 # on choisit un des mots ci-dessus définis
mot2=Ess6 # ... et un autre

def ListeLettres(mot):
    if mot=="":return []
    else:return [mot[0]]+ListeLettres(mot[1:])

#    ListeLettres(Ess6) -> ['F', 'F', 'P', 'P', 'F', 'P']

def SuffixesDec(mot): # donne la liste des suffixes par longueurs décroissantes
    if mot=="":return [""]
    else: return [mot]+SuffixesDec(mot[1:])

#  SuffixesDec(Ess6) -> ['FFPPFP', 'FPPFP', 'PPFP', 'PFP', 'FP', 'P', '']

def Prefixes(mot):
    return [mot[0:x] for x in range(1+len(mot))]

#  Prefixes(Ess6)   ->  ['', 'F', 'FF', 'FFP', 'FFPP', 'FFPPF', 'FFPPFP']

def ordre_len(a,b): #ordre des longueurs sur les mots
    return (len(a)<=len(b))

def fuse_ordre(ordre,x,y): # fusion de deux listes avec l'ordre 'ordre'
    if x==[]: return y
    elif y==[]: return x
    elif ordre(x[0],y[0]): return [x[0]]+fuse_ordre(ordre,x[1:],y)
    elif ordre(y[0],x[0]): return [y[0]]+fuse_ordre(ordre,x,y[1:])

def fuse_ordreDec(ordre,x,y): # fusion de deux listes en Decroissant avec l'ordre 'ordre'
    if x==[]: return y
    elif y==[]: return x
    elif ordre(y[0],x[0]): return [x[0]]+fuse_ordre(ordre,x[1:],y)
    elif ordre(x[0],y[0]): return [y[0]]+fuse_ordre(ordre,x,y[1:])

#  fuse_ordre(ordre_len,Prefixes(Ess1),Prefixes(Ess6)) ->
# ['', '', 'P', 'F', 'PP', 'FF', 'PPP', 'FFP', 'PPPF', 'FFPP', 'PPPFF',
#                  'FFPPF', 'PPPFFF', 'FFPPFP']
```

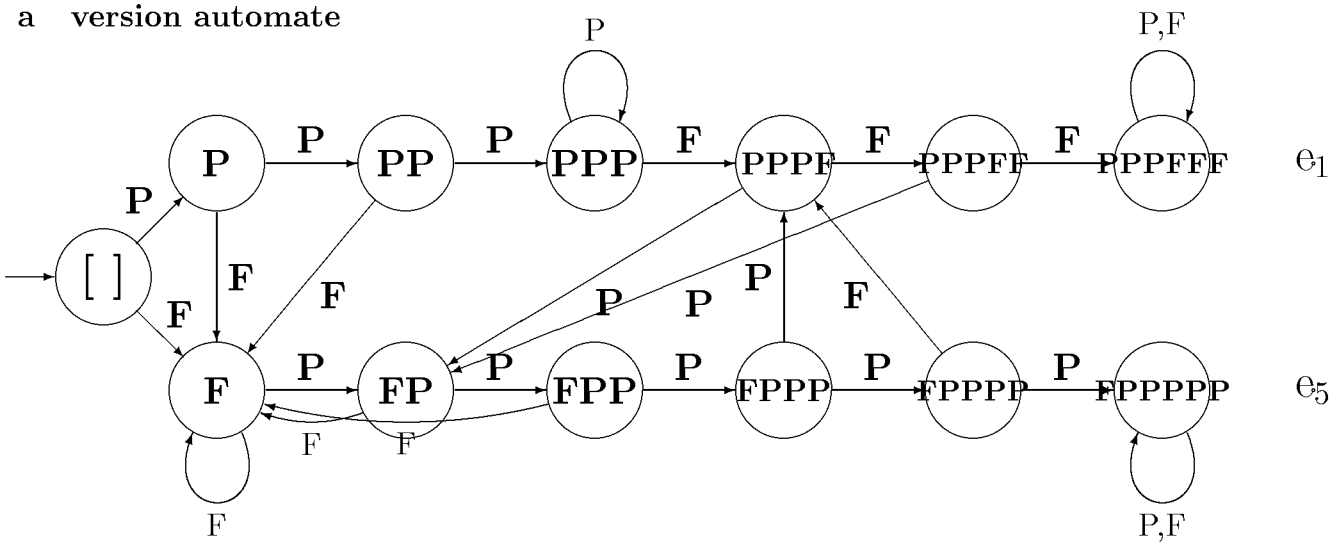
```
def VireDoublons(liste): # on sait que la liste est triée, on retire les doublons
    if len(liste)<=1:return liste
    if liste[0]==liste[1]:return VireDoublons(liste[1:])
    else: return [liste[0]]+VireDoublons(liste[1:])

def Suffixes2(m1,m2):
    S1=SuffixesDec(m1)
    S2=SuffixesDec(m2)
    return fuse_ordreDec(ordre_len,S1,S2)

# Suffixes2(Ess5,Ess6)  ->
# ['FPPPPP', 'PPPPP', 'PPPP', 'PPP', 'PP', 'P', '', 'FFPPFP',
# 'FPPFP', 'PPFP', 'PFP', 'FP', 'P', '']

def Prefixes2(m1,m2):
    P1=Prefixes(m1)
    P2=Prefixes(m2)
    return VireDoublons(fuse_ordre(ordre_len,P1,P2))

# Prefixes2(Ess5,Ess6)  ->
# ['', 'F', 'FP', 'FF', 'FPP', 'FFP', 'FPPP', 'FFPP', 'FPPPP',
# 'FFPPF', 'FPPPPP', 'FFPPFP']

def PremierTrouve(mot,lm): # cherche 'mot' dans une Liste de mots
    if lm==[]:return 'Rate'
    elif mot==lm[0]:return mot
    else: return PremierTrouve(mot,lm[1:])

# PremierTrouve('FPPP',Suffixes2(Ess5,Ess6)) -> 'Rate'
# PremierTrouve('FPPP',Prefixes2(Ess5,Ess6)) -> 'FPPP'

def PremierTrouveListe(ls,lm): # cherche le premier mot de la liste ls qui est dans lm
    res=PremierTrouve(ls[0],lm)
    if res=='Rate':return PremierTrouveListe(ls[1:],lm)
    else:return res

# PremierTrouveListe(SuffixesDec('FFPP'+'P'),Prefixes2(Ess5,Ess6)) -> 'FPPP'

#ci-dessous la définition de l'automate : (Q,A,I,F,T)

EtatsAuto=Prefixes2(mot1,mot2)

Q=EtatsAuto
A=['P','F']
I=''
Finaux=[mot1,mot2]
def T(q,a):
```

```
    if q in Finaux:return q
    else:return PremierTrouveListe(SuffixesDec(q+a),EtatsAuto)

# T('FFP','F') -> 'F'

def evalue(etat,mot):
    if mot=='':return etat
    else: return evalue(T(etat,mot[0]),mot[1:])

# evalue('FFP','PP') -> 'FPPP'

def reconnu(mot):
    return evalue(I,mot) in Finaux

# reconnu('PPPPP') -> False
# reconnu('FFFFPPPPP') -> True

comptex=0 # initialisation d'un compteur de succès pour x
comptey=0 # initialisation d'un compteur de succès pour y

def compte(mot):
        global comptex,comptey
        z=evalue(I,mot)
        if z==mot1:comptex=comptex+1
        if z==mot2:comptey=comptey+1

from random import * # importation de la librairie "random"

def choixPF(): # tirage au sort d'UN P ou F
if choice([0,1])==0:return 'P'
else:return 'F'

def tirePF(n): # tirage au sort d'une liste de longueur n de P ou F
    R=''
    for k in range(n):
        R=R+choixPF()
    return R

# On compare Ess1, Ess5 et Ess6 avec :
# for k in range(100000):
#        compte(tirePF(20))
#  puis en affichant comptex et comptey on constate que :
# E1 bat E6 qui bat E5 qui bat E1
#
```

D les détails des automates et des matrices

1 la lutte de e_1 contre e_5

a version automate



b version matrice

l'ordre des vecteurs de base est : [] P PP .....e_1 F FP ..... e_5

$$M=\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&2&0&0&0&0&0&0\\
1&0&0&0&0&0&0&1&1&1&0&0&0\\
0&0&0&0&1&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&2
\end{pmatrix}$$

Maple réduit (facilement cette fois-ci), mais je trouve qu'il est plus parlant de regarder les valeurs de $M^{100}[7,1]$ (qui est le nombre de mots de longueur 100 ayant abouti à e_1) et de $M^{100}[13,1]$ (le nombre de mots de longueur 100 ayant abouti à e_5)

$$M^{100}[7,1] = 1240813567807329274525569364864120$$

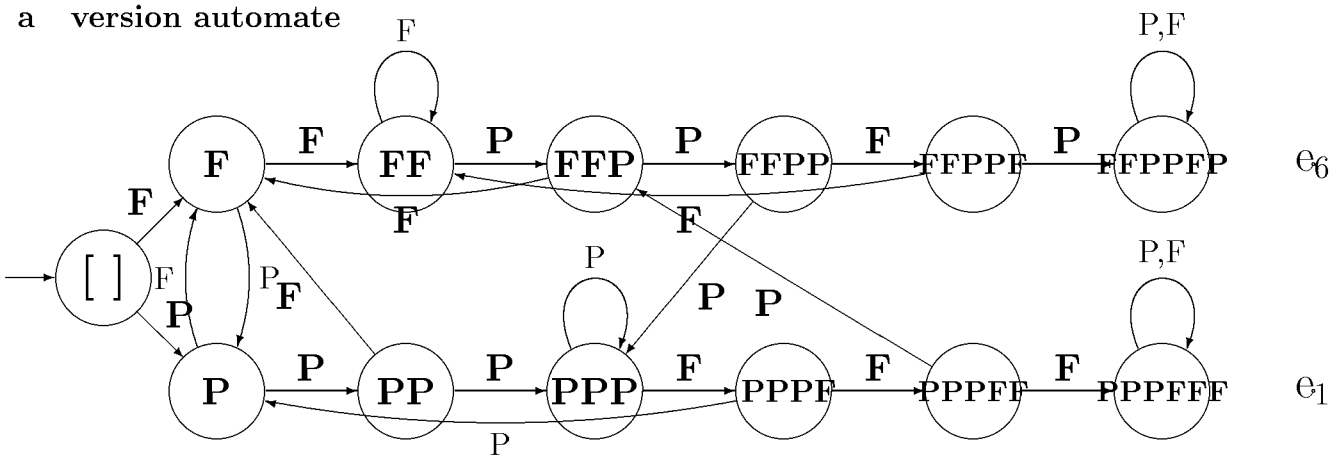
$$M^{100}[13,1] = 1892255244959733256212629338029161$$

c la conclusion

e_5 a battu e_1 d'un facteur 1.525011729 pour l'ensemble des 2^{100} mots de longueur au plus 100

2 la lutte de e_6 contre e_1

a version automate



b version matrice

l'ordre des vecteurs de base est : [] F FF ... e_6 P PP ...e_1

$$M=\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2
\end{pmatrix}$$

$$M^{100}[7,1] = 591253824509439906038468786991$$

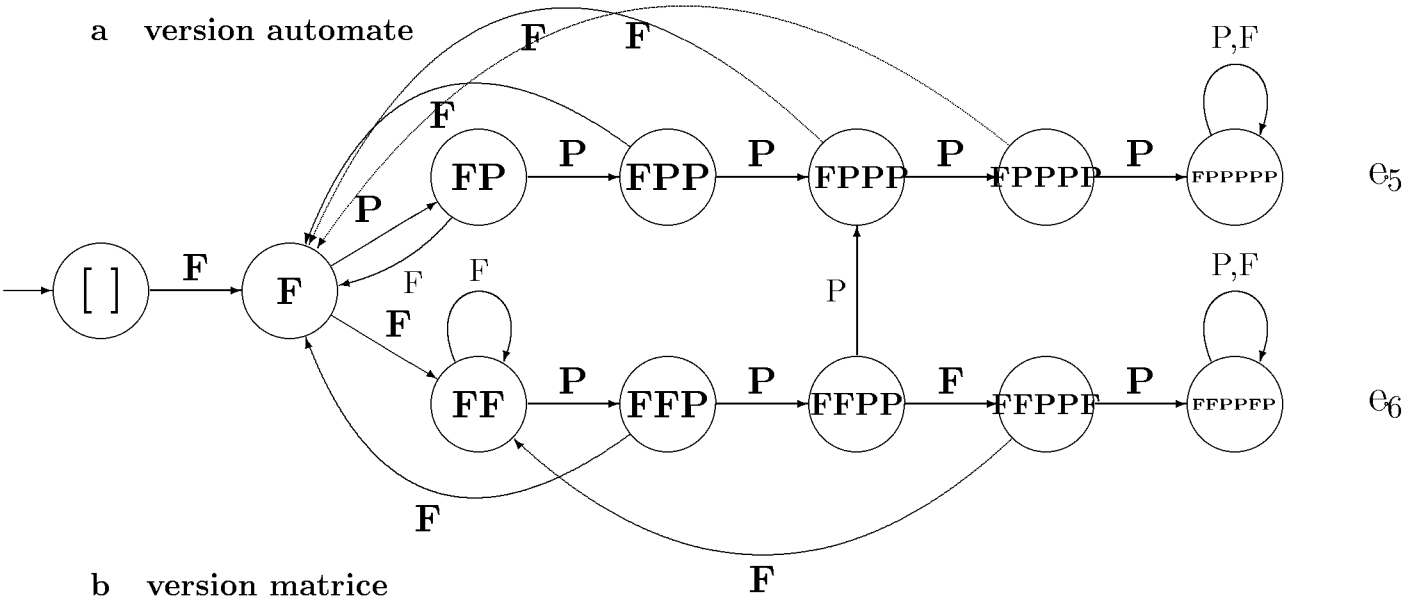
$$M^{100}[13,1] = 631775935492282389438923129480$$

c la conclusion

e_1 a battu e_6 d'un facteur 1.068535897 pour l'ensemble des 2^{100} mots de longueur au plus 100

3 la lutte de e_5 contre e_6

a version automate



b version matrice

l'ordre des vecteurs de base est : [] F FP ... e_5 FF FFP ... e_6

$$M=\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2
\end{pmatrix}$$

$$M^{100}[7,1] = 594734857996916564604450886255$$

$$M^{100}[12,1] = 6341707719246225738360075623 84$$

c la conclusion

e_6 a battu e_5 d'un facteur 1.066308395 pour l'ensemble des 2^{100} mots de longueur au plus 100

E divers

1 concernant ce qui précède : techniquement

j'avais d'abord trouvé le cas de PPF et de PFF (pour en faire un T.D.-caml-option-info)

j'ai deviné qu'il y avait un paradoxe de Condorcet la dessous, ça m'a donné l'occasion de me mettre à Python

j'ai essayé avec "tous les mots de 4 lettres" ... rien

j'ai essayé avec "tous les mots de 5 lettres" ... rien

j'ai essayé avec "tous les mots de 6 lettres" ... rien au début

puis j'ai pensé à dessiner les automates et ...

pour trouver mes trois mots, j'ai "bricolé" avec les automates : voyez celui de e_1 contre e_5, en dehors de l'avancement naïf dans les mots j'ai 3 liens qui vont vers e_1 (je compte la boucle sur PPP et pas celle du puits) tandis que j'en ai 7 qui vont vers e_5

une erreur bizarre : dans la lutte e_5 contre e_6 j'avais par erreur mis M(2,11)=1 au lieu de M(8,11)=1 (le successeur de FFPPF par F est FF et non F)

il en résultait une matrice dont les termes $M^n(7,1)$ (x gagne) et $M^n(12,1)$ (y gagne) étaient toujours égaux (?????)

mes essais m'ont fait réduire "de nombreuses matrices" : je dirais que 50% étaient non-diagonalisables : ???

2 concernant ce qui précède : philosophiquement

en écrivant les automates je me suis trompé de nombreuses fois, en les traduisant en matrices je me suis encore trompé, entre Python - LaTeX - Maple j'ai recopié faux de partout. Comment finalement être sûr du résultat?

les parties de programmes Python sont individuellement testables, quand on teste avec des mots autres que e_1 e_5 et e_6 on n'a "jamais le paradoxe". Finalement l'expérimentation semble (?) plus fiable (?) que la pensée (?)

3 la généralisation qu'il faudrait

on doit bien pouvoir trouver 3 mots A, B, C de la langue française tels que : en testant sur l'ensemble des textes numérisés A apparaît en moyenne avant B, B apparaît en moyenne avant C, C apparaît en moyenne avant A

on doit facilement y arriver par "jeu de mots" : A ayant deux sens, dans un des contextes il est "naturellement avant B, dans l'autre après C

mais le but du paradoxe est d'obtenir cela sans jeu de mots