



REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université d'Alger 1
Faculté des Sciences
Département informatique

Année 2021/2022



Rapport TP *THL*

Réalisé par :

Hammoudi Nasreddine (G1)

Belgueloul Hayder (G1)

Chekikene Hadil (G1)

Fezoui Yacine (G1)

Section : A

Semestre 2

SOMMAIRE

INTRODUCTION	3
L'ENVIRONNEMENT DE DEVELOPEMENT	3
PARTIE 1	5
PARTIE 2	6
PARTIE 3	10
EXEMPLAIRE D’AFFICHAGE CONSOLE	12
LE PROGRAMME SOURCE	13

INTRODUCTION :

En théorie des langages, l'ensemble des entités élémentaires est appelé l'alphabet. Une combinaison d'entités élémentaires est appelée un mot. Un ensemble de mots est appelé un langage et est décrit par une grammaire.

L'ENVIRONNEMENT DE DEVELOPPEMENT :

Dans ce TP, on a choisi de programmer avec le langage Python3 en utilisant l'IDLE de Python 3.7.

Comme bibliothèque externe, on a utilisé « PyQt5 » qui permet d'intégrer une interface graphique et de la gérer.

Python est un langage de programmation interprété, multiparadigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions.

Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques, des smartphones aux ordinateurs centraux5, de Windows à Unix avec notamment GNU/Linux en passant par MacOS, ou encore Android, iOS, et peut aussi être traduit en Java. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Python est un langage de programmation qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées.

LA CONCEPTION DE NOTRE TP :

Notre TP est composé de 8 fonctions assurant le fonctionnement des 3 parties demandées.

Partie 1 : Manipulation et opérations sur les mots

Dans cette partie on nous a demandé de réaliser deux programmes permettant de donner la puissance et le mot miroir d'un mot quelconque qui appartient à T^* (Avec $T = \{a, b, c\}$) qu'un utilisateur tape au clavier.

Rappel :

❖ **MIROIR :**

- **Définition :** on appelle mot miroir d'un mot w note $Mir(w)$ ou w^R le mot obtenu en inversant les lettres de « w ».

- **Formellement :**

$$Mir(\epsilon) = \epsilon$$

$$Mir(a) = a$$

$$Mir(aw) = Mir(w).a$$

$$Mir(abc) = Mir(bc).a = Mir(c).ba = cba$$

❖ **PUISSANCE :**

- **Définition :** la puissance d'un mot W est définie par récurrence par la manière suivante :

$$W^0 = \epsilon$$

$$W^1 = W$$

$$W^{n+1} = W^n.W \quad (abbc)^n =$$

Pour cela on a utilisé les fonctions suivantes :

- **Appartenance_a_b_c(word) :** c'est une fonction qui retourne vrai si le mot en entrée appartient à l'ensemble T^* sinon elle retourne faux.

Avant de passer à la conception du programme on a élaboré l'algorithme suivant :

```
FONCTION Appartenance_a_b_c (word :CHAINE DE CARACTERE ) : BOOLEAN
VAR i :ENTIER ; appartenance : BOOLEAN ;
DEBUT
    appartenance ← VRAI ;
    i ← 0 ;
    TANT QUE (i < Longueur(word)) FAIRE
        SI ( word [i] != 'a' ET word [i] != 'b' ET word [i] != 'c' ) ALORS
            appartenance ← FAUX ;
        FIN SI ;
    FINTANTQUE ;
    RETOURNER appartenance ;
FIN ;
```

Après ça on la traduit ont un programme (voir le code ci-dessous).

- `miroir_mot(word)` : c'est une fonction qui retourne l'inverse du mot en entrée.
Avant de passer à la conception du programme on a élaboré l'algorithme suivant :

```

FONCTION miroir_mot( word : CHAINE DE CARACTERE )
  VAR
    i : ENTIER ; miroir : CHAINE DE CARACTERE ;
  DEBUT
    SI (Appartenance_a_b_c(word) == VRAI ) ALORS
      miroir ← '' ; i ← 0 ;
      TANT QUE (i < Longueur(word) ET i >= 0) FAIRE
        miroir ← miroir . word [i] ;
        i ← i - 1 ;
      FIN TANTQUE ;
      RETOURNER miroir ;
    FIN SI ;
  FIN ;

```

Après ça on la traduit ont un programme (voir le code ci-dessous).

- `puissance_mot(word,n)` : c'est une fonction qui prend en entrée un mot word et un entier n et retourne $(word)^n$.

Avant de passer à la conception du programme on a élaboré l'algorithme suivant :

```

FONCTION puissance_mot (word : CHAINE DE CARACTERE, n : ENTIER)
  VAR
    i : ENTIER ;
    temporaire : CHAINE DE CARACTERE ;
  DEBUT
    SI ( Appartenance_a_b_c ( word ) == VRAI ) ALORS
      temp ← word ;
      POUR( i ← 1 jusqu'à n-1 ) FAIRE
        word = word.temp ;
      FINPOUR ;
      RETOURNER word ;
    FINSI ;
  FIN ;

```

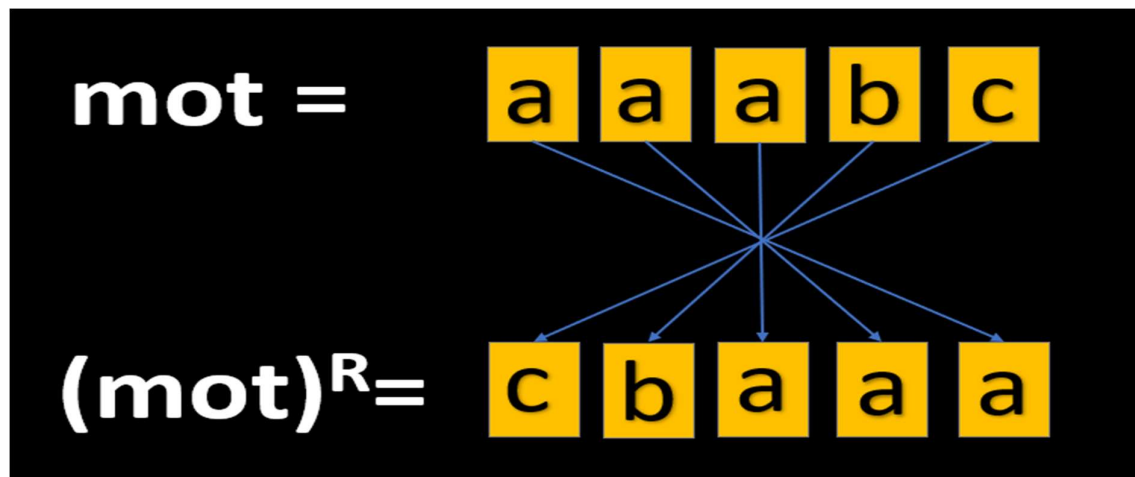
Après ça on la traduit ont un programme (voir le code ci-dessous).

Concept :

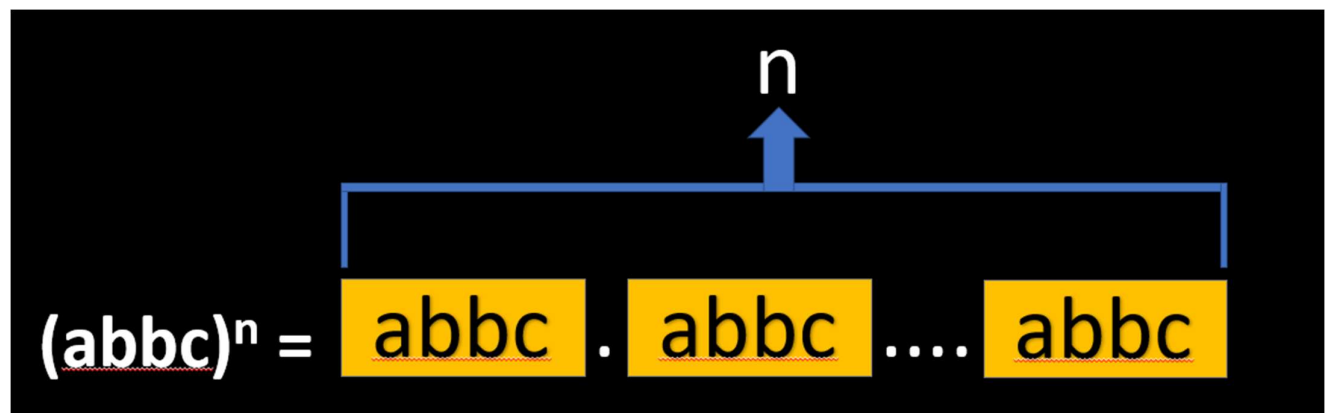
- ✚ Lorsqu'un utilisateur saisi un mot pour son mot miroir ou pour sa puissance, la fonction `Appartenance_a_b_c` vérifie d'abord si ce mot appartient à T^* autrement dit, elle vérifie si le mot entré est composé de « a », de « b » ou de « c ».
- ✚ Après avoir vérifié l'appartenance du mot entré par l'utilisateur, la fonction `miroir_mot` retourne son inverse en lisant les caractères de la fin jusqu'au début du mot.
- ✚ Après avoir vérifié l'appartenance du mot entré par l'utilisateur, la fonction `puissance_mot` prend en entrée un mot word et un entier « n » (l'exposant) et retourne $(word)^n$, en concaténant le mot word avec lui-même autant de fois que souhaite l'utilisateur (Avec l'exposant « n ») grâce à une simple boucle.

Des schémas explicatifs :

- Le miroir :



- La puissance :



Partie 2 : Langage et Grammaires

Dans cette partie on nous a demandé d'écrire un programme paramétré qui permet de générer tous les mots de $L(G)$ d'une longueur donnée n ($n \geq 0$). Les mots générés doivent appartenir à T^* (Avec $T = \{a, b, c\}$) qu'un utilisateur tape au clavier.

Soit $G = \langle T, N, S, P \rangle$ tel que :
 $T = \{a, b\}$. $N = \{S, A, B, C\}$.
 $P : S \rightarrow AB$
 $A \rightarrow aA / bA / ab$
 $B \rightarrow bC$
 $C \rightarrow aC / bC / \epsilon$

Quelques exemples de dérivation, on obtient les mots suivants :

$S \rightarrow AB \rightarrow abbC \rightarrow abb\epsilon \rightarrow abb$ (Le 1er mot est « abb »)
 $S \rightarrow AB \rightarrow aAbC \rightarrow abAbC \rightarrow ababb\epsilon \rightarrow ababb$ (Le 2ème mot est « ababb »)
 $S \rightarrow AB \rightarrow bAbC \rightarrow baAbaC \rightarrow baabba\epsilon \rightarrow baabba$ (Le 3ème mot est « baabba »)

Par déduction on obtient le langage généré par la grammaire G :

$L(G) = \{w_1abbw_2 / w_1, w_2 \in \{a, b\}^*\}$

Quelques exemples des mots appartenant au langage $L(G)$:

$L = \{abb, ababb, baabba, \dots\}$

Note :

Il Ya deux conditions à vérifier pour créer un mot qui appartient au langage $L(G)$:

- 1) L'expression régulière : $(a+b)^*abb(a+b)^*$ dénote tous les mots sur l'alphabet $\{a,b\}$ contenant le sous mot « abb ». Avec « abb » comme facteur propre.
- 2) Il est impossible de générer un mot de longueur inférieure à "3".

- [Liste_De_Combinaison\(ens, k\)](#) : c'est une fonction qui permet de générer toutes les combinaisons possible des mots de taille 'k' avec les caractères de l'ensemble des mots 'ens' $T^* = \{a,b\}$.

Avant de passer à la conception du programme on a élaboré l'algorithme suivant :

```
// ens=['a','b'] une liste qui contient les deux lettres 'a' et 'b'
FONCTION Liste_De_Combinaison(ens : LISTE, k : ENTIER) :
  VAR aList : Liste; n : ENTIER
  DEBUT
    aList[:] ← [];
    n ← taille(ens); // n=2
    All_combin(ens, "", n, k);
  RETOURNER aList;
FIN ;

FONCTION All_combin(ens, prefix, n, k):
  VAR newPrefix : CHAINE DE CARACTERE;
  DEBUT
    SI (k == 0) ALORS
```

```

    aList.append(prefix); // insérer Le mot dans la liste
    RETOURNER ;
FINSI;
POUR (i← 0 a n) FAIRE
    newPrefix ← prefix . ens[i];
    All_combin(ens, newPrefix, n, k - 1);
FINPOUR;
FIN;

```

Après ça on la traduit ont un programme (voir le code ci-dessous).

- **generer_L(n)** : fonction qui génère les mot de taille 'n' de la grammaire. Et utilise la fonction précédente.

Avant de passé à la conception du programme on a élaboré l'algorithme suivant :

```

FONCTION generer_L (n : ENTIER) :
VAR
    lg,lg2,ld,ld2,lmg,lmg2,lmd: LISTE;
DEBUT
    SI (n==3)ALORS RETOURNER ("abb"); FINSI;
    flist[:] = []; // initialisation de la liste
    SI (n>3)ALORS
        //partie gauche
        lg<-Liste_De_Combinaison(['a', 'b'],n-3);
        s<-lg;
        TANT QUE (s!=NULL) FAIRE
            s<- lg.info;
            lg2.append(s.info . "abb");// insérer dans une liste
            s<-s.suiv;
        FAIT;
        flist.extend(lg2) // inserer une liste dans une autre
        //partie droite
        ld<-Liste_De_Combinaison(['a', 'b'],n-3);
        s<-ld;
        TANT QUE (s!=NULL) FAIRE
            ld2.append("abb". s.info );// insérer dans une liste
            s<-s.suiv;
        FAIT;
        flist.extend(ld2) // inserer une liste dans une autre
    SI (n==4) ALORS RETOURNER flist; FINSI;
    //milieu
    SI (n-3>=2) ALORS
        i=n-3-1;
        TANTQUE (i>0)FAIRE
            lmg<-Liste_De_Combinaison(['a', 'b'],i);
            s<-lmg;
            TANT QUE (s!=NULL) FAIRE
                lmg2.append(s.info . "abb");// insérer dans une liste
                s<-s.suiv;
            FAIT;
            lmd<-Liste_De_Combinaison(['a', 'b'],n-3-i);

```



```

// CETTE PARTIE NE PEUT PAS ETRE TRANSFORMER EN LANGAGE ALGORITHMIQUE
lorg=[] // initialisation d'une table de tuples
//produit cartésien entre lmg et lmd
for element in itertools.product(lmg,lmd):
    lorg.append(element)
//transformer la table lorg de tuple en liste res
res = [''.join(o) for o in lorg]
flist.extend(res) // inserer une liste dans une autre
i=i-1;
    FAIT;
    RETOURNER flist;
    FINSI;
FINSI;
FIN;

```

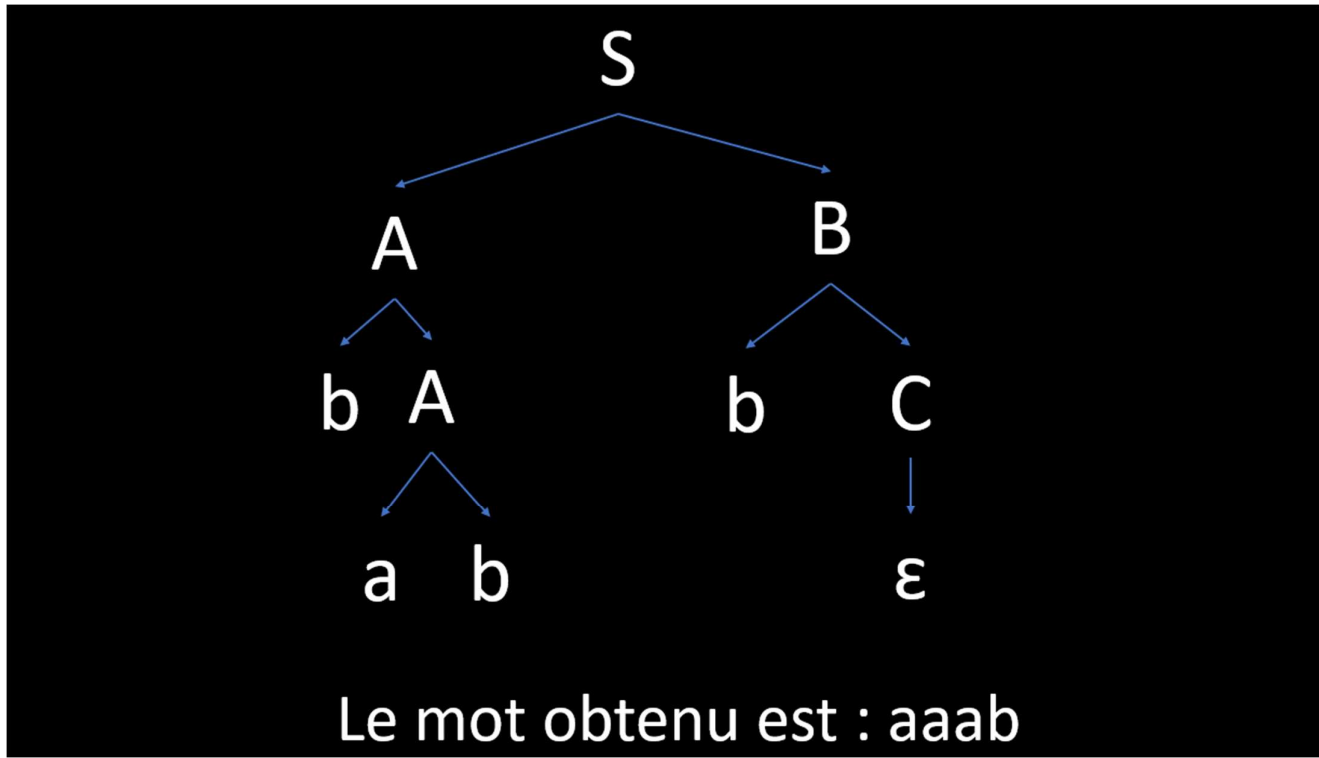
Après ça on la traduit ont un programme (voir le code ci-dessous).

Concept :

- ✚ Lorsqu'un utilisateur saisi une taille inferieure a « 3 » un message d'erreur s'affiche.
- ✚ Lorsqu'un utilisateur saisi la taille « 3 » le programme affiche le mot « abb » car c'est l'unique mot de taille « 3 », il apparait dans tous les mots du langage, c'est le facteur propre.
- ✚ Lorsqu'un utilisateur saisi une taille supérieure à « 3 », la fonction `generer_L` prend en paramètre la taille « n » ($n > 3$) entré par l'utilisateur, puis cree une liste de combinaison de « a », « b » avec la fonction `Liste_De_Combinaison` qui prend en paramètre une liste qui contient « ['a','b'] » et une taille « n-3 » (enlevée la longueur du mot propre « abb ») puis une concaténation droite et une concaténation gauche de tous les éléments de la liste avec « abb ». Une concaténation droite et une concaténation a gauche puis vérifier si la taille de départ « n » est égale a « 4 » ($n=4$?)
 - Si ($n = 4$) alors c'est terminer on retourne la liste finale contenant tous les mots du langage de longueur « 4 ».
 - Si ($n > 4$) on va continuer la création des listes de combinaisons droites et gauches grâce a la fonction `Liste_De_Combinaison` mais qui prend en paramètre une taille qui diffèrent avec l'utilisation des boucles et des conditions adapter (voir l'algorithme ci-dessus) permettant de garder la taille finale des mots obtenus égale à la taille entrée par l'utilisateur.

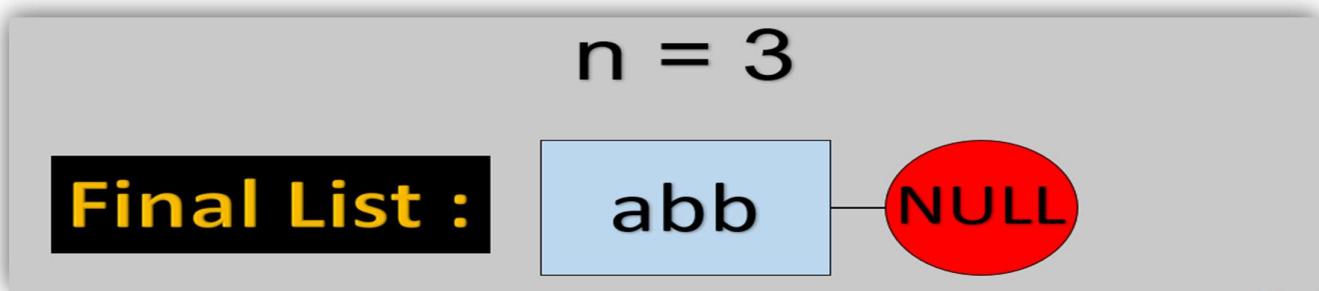
Des schémas explicatifs :

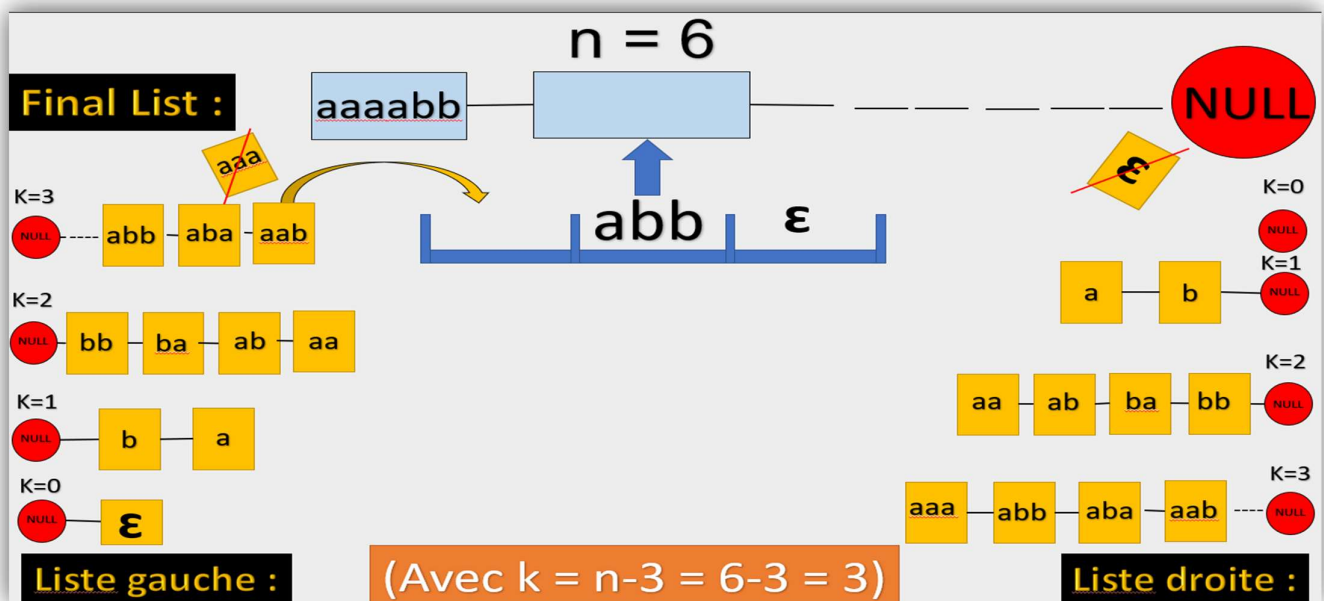
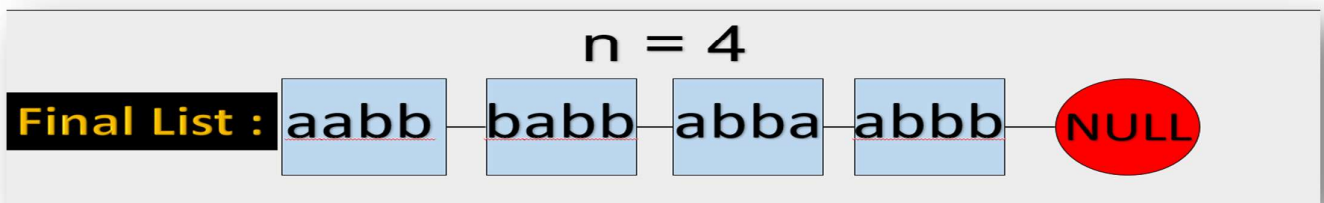
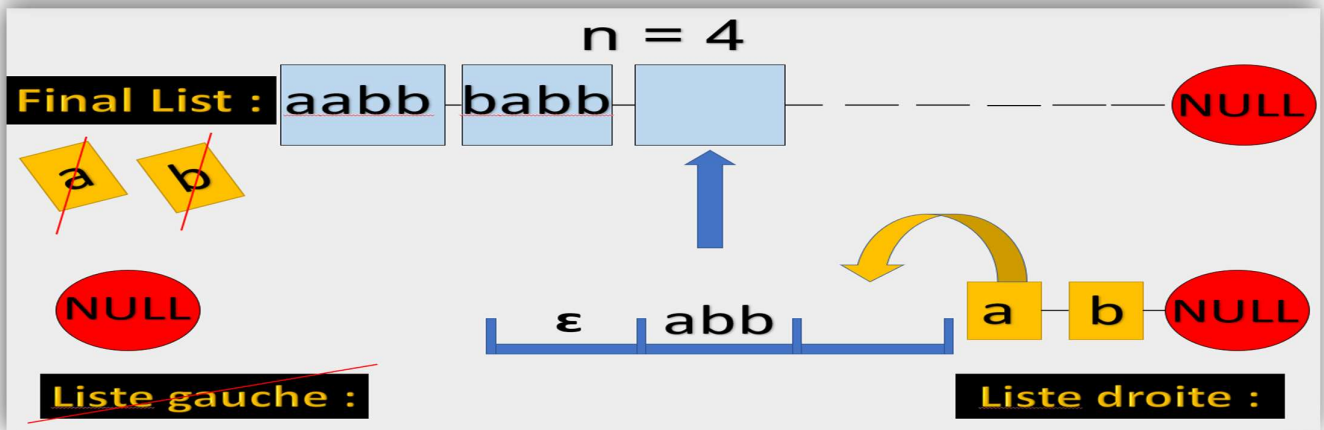
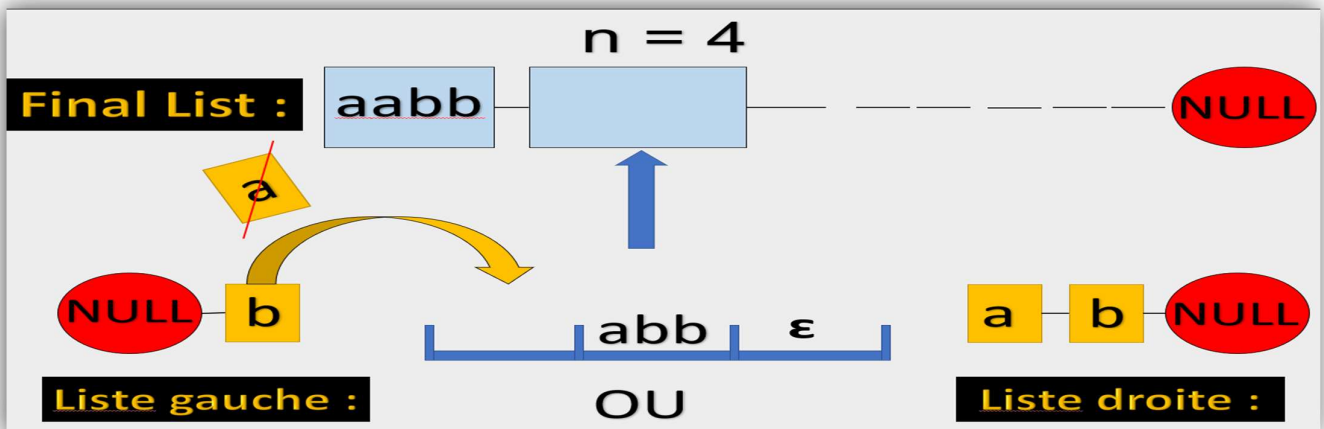
- Derivation :



- Générer les mots du langage $L(G)$:

On a illustré la conception avec 3 exemples de taille ($n=3$, $n=4$, $n=6$)





Partie 3 : Analyseur Syntaxique

Dans cette partie on nous a demandé d'écrire un programme paramétré qui, étant donné un mot quelconque qui appartient à T^* (Avec $T = \{a, b, c\}$) qu'un utilisateur tape au clavier. Vérifie si ce mot appartient au langage $L(G)$ généré par la grammaire G .

Soit $G = \langle T, N, S, P \rangle$ tel que :

$T = \{a, b\}$. $N = \{S, A, B, C\}$.

$P = \{ S \rightarrow aaSb / Sa / \epsilon \}$

Quelques exemples de dérivation, on obtient les mots suivants :

$S \rightarrow \epsilon$ (le premier mot c'est epsilon)

$S \rightarrow Sa \rightarrow a$ (le 2^{ème} mot est « a »)

$S \rightarrow aaSb \rightarrow aaSab \rightarrow aaab$ (le 3^{ème} mot est « aaab »)

$S \rightarrow aaSb \rightarrow aaaaSbb \rightarrow aaaaSabb \rightarrow aaaaabb$ (le 4^{ème} mot est « aaaaabb »)

Par déduction on obtient le langage généré par la grammaire G :

$L(G) = \{a^i.a^n.b^j.a^k / n, k \geq 0 \text{ ET } i \geq 2*j\}$.

Quelques exemples des mots appartenant au langage $L(G)$:

$L = \{\epsilon, a, aab, aaab, aaaaabb, \dots\}$

Note :

Il Ya deux conditions à vérifier pour qu'un mot appartient au langage $L(G)$:

- Le mot entré par l'utilisateur doit être composé que de « a » ou de « b ».
- Le nombre d'occurrence de « a » avant l'apparition du premier « b » du mot doit être supérieure ou égale au double du le nombre d'occurrence « b ».

Pour cela on a utilisé les fonctions suivantes :

- [Appartenance_a_b\(word\)](#) : c'est une fonction qui retourne vrai si le mot en entrée appartient à l'ensemble T^* sinon elle retourne faux.

Avant de passé à la conception du programme on a élaboré l'algorithme suivant :

```
FONCTION Appartenance_a_b (word :CHAINE DE CARACTERE ) : BOOLEAN
VAR i :ENTIER ; appartenance : BOOLEAN ;
DEBUT
    appartenance ← VRAI ;
    i ← 0 ;
    TANT QUE (i < Longueur(word)) FAIRE
        SI ( word [i] != 'a' ET word [i] != 'b' ) ALORS appartenance ← FAUX ;
        FIN SI ;
    FINTANTQUE ;
    RETOURNER appartenance ;
FIN ;
```

Après ça on la traduit ont un programme (voir le code ci-dessous).

- **Nombre_De_a(word)** : calculer le nombre de 'a' depuis le début de la chaîne en entrée jusqu'à tomber sur un autre caractère.

Avant de passer à la conception du programme on a élaboré l'algorithme suivant :

```

FUNCTION Nombre_De_a (word :CHAINE DE CARACTERE ) : ENTIER
  VAR i , cpt :ENTIER ;
  DEBUT
    cpt ← 0 ;
    POUR( i ← 0 à Longueur(word)) FAIRE
      SI ( word [i] == 'a' ) ALORS      cpt ← cpt + 1 ;
      SINON RETOURNER cpt;
      FIN SI ;
    FINPOUR ;
  RETOURNER cpt ;
FIN ;

```

Après ça on la traduit ont un programme (voir le code ci-dessous).

- **Nombre_De_b(word)** : Fonction qui calcule le nombre de 'b' dans le mot en entrée.

Avant de passer à la conception du programme on a élaboré l'algorithme suivant :

```

FUNCTION Nombre_De_b (word :CHAINE DE CARACTERE ) : ENTIER
  VAR i , cpt :ENTIER ;
  DEBUT
    cpt ← 0 ;
    POUR( i ← 0 à Longueur(word)) FAIRE
      SI ( word [i] == 'b' ) ALORS cpt ← cpt + 1 ; FIN SI ;
    FINPOUR;
  RETOURNER cpt ;
FIN ;

```

Après ça on la traduit ont un programme (voir le code ci-dessous).

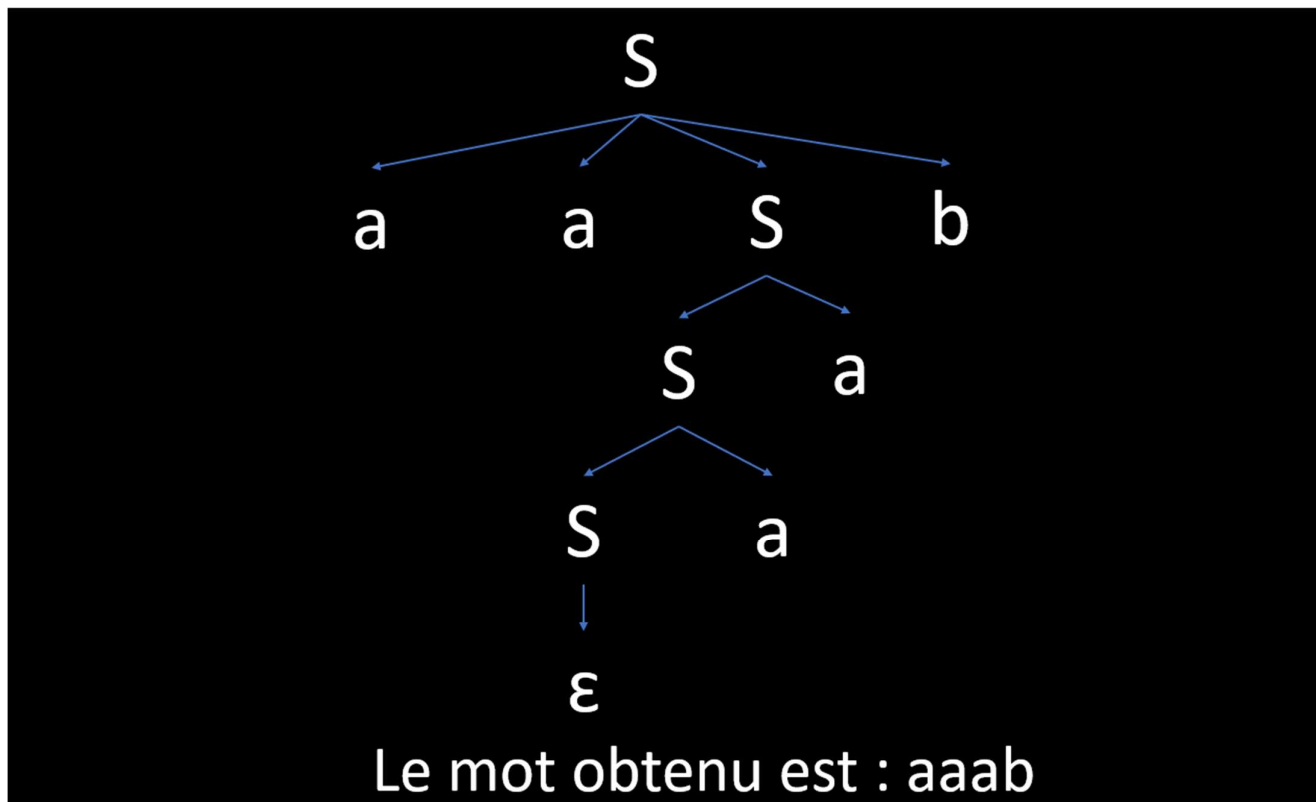
Concept :

- ✚ Lorsqu'un utilisateur saisit un mot pour son mot miroir ou pour sa puissance, la fonction **Appartenance_a_b** vérifie d'abord si ce mot appartient à T^* autrement dit, elle vérifie si le mot entré est composé de « a » ou de « b ».
- ✚ Après avoir vérifié l'appartenance du mot entré par l'utilisateur, la fonction **Nombre_De_a** calcule le nombre d'occurrence de « a » avant l'apparition du premier « b » du mot passé en entrée.
- ✚ Après avoir vérifié l'appartenance du mot entré par l'utilisateur, la fonction **Nombre_De_b** calcule le nombre d'occurrence « b » qui apparaît dans tout le mot et qui est passé en entrée.
- ✚ Après avoir fait les calculs avec les deux fonctions précédentes il nous reste qu'à vérifier la condition obtenue en étudiant le langage obtenu $L(g)$, et qui stipule que le nombre d'occurrence de « a » avant l'apparition du premier « b » du mot doit être supérieure ou égale au double du nombre d'occurrence « b » qui apparaît dans tout le mot. Alors on obtient la formule suivante :

$$\text{Nombre_De_a(mot)} \geq 2 * \text{Nombre_De_b(mot)}$$

Schémas explicatifs :

- Dérivation :



EXEMPLE D’AFFICHAGE CONSOLE DE NOTRE TP :

 TP THL

TP THL

Partie 1 : Calcul du miroir et la puissance d'un mot

Miroir

Puissance


Partie 2 : Introduire la taille des mots à générer puis cliquer sur générer

Lancer La partie 2 du tp

Partie 3 : Introduire le mot et cliquer sur Ok pour vérifier si le mot appartient au langage G

Ok

REALISER PAR :
HAMMOUDI NASREDDINE & CHEKIKENE HADIL
FEZOUI YACINE & BELGUELOUL HAYDER

 TP THL

TP THL

Partie 2 : Introduire la taille des mots à générer puis cliquer sur générer

Generer

LE PROGRAMME :

```
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QWidget, QPushButton, QAction,
QLineEdit, QMessageBox, QSpinBox, QLabel
from PyQt5.QtGui import QIcon
import PyQt5.QtGui as QtGui #pour les icones
from PyQt5.QtCore import pyqtSlot
import itertools
from collections import OrderedDict
import TP_p2 as p2
import os
alist = []
flist = []

#PARTIE 1 :

#appartenance T={a,b,c}
def Appartenance_a_b_c(word):
    cpt=1
    for i in range(len(word)):
        if(word[i] != 'a' and word[i]!='b' and word[i]!='c' ):
            return -1
    return cpt

#elle prend en parametre un mot et renvoie son inverse
def miroir_mot(word):

    return(word[::-1])

#elle prend en parametre un mot et un entier (exposant) et renvoi la puissance du mot
def puissance_mot(word,n):
    d=""
    for i in range(n):
        d=d+word
    return d

#PARTIE 2 :

# elle permet de generer toutes les combinaisons possible des mots de taille k avec les
caractere de l'ensemble des motes (ens) T*={a,b}
def Liste_De_Combinaison(ens, k):
    alist[:] = []
    n = len(ens)
    All_combin(ens, "", n, k)
    return alist

def All_combin(ens, prefix, n, k):
    if (k == 0) :
        alist.append(prefix)
        return
```



```

    for i in range(n):
        newPrefix = prefix + ens[i]
        All_combin(ens, newPrefix, n, k - 1)

#fonction qui genere les mot de taille N de La grammaire donnee
def generer_L(n):

    if(n==3):
        return ("abb")
    flist[:] = []
    if (n>3):
        #partie gauche
        lg=Liste_De_Combinaison(['a', 'b'],n-3)
        lg=[s + "abb" for s in lg]

        flist.extend(lg)
        #partie droite
        ld=Liste_De_Combinaison(['a', 'b'],n-3)
        ld=["abb" + s for s in ld]

        flist.extend(ld)

    if(n==4):
        return flist
    #milieu
    if(n-3>=2):
        i=n-3-1
        while i>0:
            lmg=Liste_De_Combinaison(['a', 'b'],i)
            lmg=[s + "abb" for s in lmg]

            lmd=Liste_De_Combinaison(['a', 'b'],n-3-i)

            long=[]

            for element in itertools.product(lmg,lmd):#produit cartesien lmg lmd
                long.append(element)
            res = [''.join(o) for o in long] #transformer long de tuple en liste res
            flist.extend(res)
            i=i-1
        return flist

#PARTIE 3 :

#appartenance T={a,b}
def Appartenance_a_b(word):
    cpt=1
    for i in range(len(word)):
        if(word[i] != 'a' and word[i]!='b'):
            return -1
    return cpt
#calculer le nombre de a de debut de la chaine jusqu'a tombee sur un autre caractere

```

```

def Nombre_De_a(word):
    cpt=0
    for i in range(len(word)):
        if(word[i]=='a'):
            cpt=cpt+1
        else:
            return cpt
    return cpt
# nombre de b dans le mot entier
def Nombre_De_b(word):
    cpt=0
    for i in range(len(word)):
        if(word[i]=='b'):
            cpt=cpt+1
    return cpt

class App(QMainWindow):

    def __init__(self):
        super().__init__()
        self.title = 'TP THL'
        self.left = 650
        self.top = 250
        self.width = 1080
        self.height = 720
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setFixedSize(self.width, self.height)
        self.setGeometry(self.left, self.top, self.width, self.height)
        #self.setStyleSheet("background-color: #900C3F;")
        self.setWindowIcon(QtGui.QIcon('didine.ico'))
        #-----

--

        self.label = QLabel("TP THL",self)
        self.label.setGeometry(460, 10, 1080, 50)
        self.label.setFont(QtGui.QFont ("Times New Roman", 30))

        self.label = QLabel("Partie 1 : Calcul du miroir et la puissance d'un mot",self)
        self.label.setGeometry(20, 75, 1080, 50)
        self.label.setFont(QtGui.QFont ("Helvetica", 15))

        # Create a button for part 2 in the window
        self.button1 = QPushButton('Miroir', self)

```

```

self.button1.move(400,148)
self.button1.resize(200,40)
self.button1.setFont(QtGui.QFont ("Arial", 15))
self.button1.setStyleSheet (" background-color: #e63946 ")

self.textbox1 = QLineEdit(self)
self.textbox1.move(20, 130)
self.textbox1.resize(350,80)
self.textbox1.setFont(QtGui.QFont ("Arial", 15))
# connect button to function on_click
self.button1.clicked.connect(self.mirroi)

#-----
--

# Create a button for part 2 in the window
self.button1 = QPushButton('Puissance', self)
self.button1.move(640,260)
self.button1.resize(200,40)
self.button1.setFont(QtGui.QFont ("Arial", 15))
self.button1.setStyleSheet (" background-color: #e63946 ")

# Create SpinBox
self.spin1 = QSpinBox(self)
self.spin1.move(400,235)
self.spin1.resize(200,80)
self.spin1.setFont(QtGui.QFont ("Arial", 15))

self.textbox2 = QLineEdit(self)
self.textbox2.move(20, 235)
self.textbox2.resize(350,80)
self.textbox2.setFont(QtGui.QFont ("Arial", 15))

# connect button to function on_click
self.button1.clicked.connect(self.puissance)

#-----
--

self.label = QLabel("Partie 2 : Introduire la taille des mots à générer puis
cliquer sur générer",self)
self.label.setGeometry(20, 320, 1080, 40)
self.label.setFont(QtGui.QFont ("Helvetica", 15))

# Create a button for part 2 in the window
self.button2 = QPushButton('Lancer La partie 2 du tp', self)
self.button2.move(280,390)
self.button2.resize(300,40)
self.button2.setFont(QtGui.QFont ("Arial", 15))
self.button2.setStyleSheet (" background-color: #e63946 ")

```

```

# connect button to function on_click
self.button2.clicked.connect(self.generer)

#-----
--
self.label3 = QLabel("Partie 3 : Introduire le mot et cliquer sur Ok pour vérifier
si le mot appartient au langage G",self)

# setting geometry
self.label3.setGeometry(20,460, 1080, 40)
self.label3.setFont(QtGui.QFont ("Helvetica", 15))

# Create textbox
self.textbox = QLineEdit(self)
self.textbox.move(20, 520)
self.textbox.resize(700,80)
self.textbox.setFont(QtGui.QFont ("Arial", 15))

# Create a button for part 3 in the window
self.button = QPushButton('Ok ', self)
self.button.resize(200,40)
self.button.move(780,540)
self.button.setFont(QtGui.QFont ("Arial", 15))
self.button.setStyleSheet (" background-color: #e63946 ")

self.label = QLabel("REALISER PAR :",self)
self.label.setGeometry(500, 650, 380, 40)
self.label.setFont(QtGui.QFont ("Arial", 10))

self.label = QLabel(" \nHAMMOUDI NASREDDINE & CHEKIKENE HADIL ",self)
self.label.setGeometry(390, 660, 380, 40)
self.label.setFont(QtGui.QFont ("Arial", 10))

self.label = QLabel(" \nFEZOUYI YACINE & BELGUELOUL HAYDER ",self)
self.label.setGeometry(400, 680, 380, 40)
self.label.setFont(QtGui.QFont ("Arial", 10))

# connect button to function on_click
self.button.clicked.connect(self.on_click)
self.show()

@pyqtSlot()
def on_click(self):
    textboxValue = self.textbox.text()
    if(Appartenance_a_b(textboxValue)<0):
        QMessageBox.question(self, 'TP THL', "La chaine ne doit contenir que des 'a' ou
des 'b'.", QMessageBox.Ok, QMessageBox.Ok)
    else:
        if(Nombre_De_a(textboxValue)>= 2*Nombre_De_b(textboxValue) ):

```

```

        if(textboxValue==""):
            QMessageBox.question(self, 'TP THL', "Le mot vide (epsilon) appartient
au langage L(G).", QMessageBox.Ok, QMessageBox.Ok)
        else:
            QMessageBox.question(self, 'TP THL', "La chaine : " + textboxValue + "
existe dans le langage L(G).", QMessageBox.Ok, QMessageBox.Ok)
        else:
            QMessageBox.question(self, 'TP THL', "La chaine : " + textboxValue + "
n'existe pas dans le langage L(G).", QMessageBox.Ok, QMessageBox.Ok)

        self.textbox.setText("")

    @pyqtSlot()
    def miroir(self):
        textboxValue = self.textbox1.text()
        if(Appartenance_a_b_c(textboxValue)<0):
            QMessageBox.question(self, 'TP THL', "La chaine ne doit contenir que des 'a',
des 'b' ou des 'c'. ", QMessageBox.Ok, QMessageBox.Ok)
        else:
            if (textboxValue==""):
                QMessageBox.question(self, 'TP THL', "Le miroir du mot est : Le mot vide
(epsilon) " , QMessageBox.Ok, QMessageBox.Ok)
            else:
                QMessageBox.question(self, 'TP THL', "Le miroir du mot est : " +
miroir_mot(textboxValue) , QMessageBox.Ok, QMessageBox.Ok)
            self.textbox1.setText("")

    @pyqtSlot()
    def puissance(self):
        textboxValue = self.textbox2.text()
        n = self.spin1.value()
        if(Appartenance_a_b_c(textboxValue)<0):
            QMessageBox.question(self, 'TP THL', "La chaine ne doit contenir que des 'a',
des 'b' ou des 'c'. ", QMessageBox.Ok, QMessageBox.Ok)
        else:
            if (textboxValue==" " or n==0):
                QMessageBox.question(self, 'TP THL', "La puissance du mot est : Le mot
vide (epsilon) " , QMessageBox.Ok, QMessageBox.Ok)
            else:
                QMessageBox.question(self, 'TP THL', "La puissance du mot est : " +
puissance_mot(textboxValue,n) , QMessageBox.Ok, QMessageBox.Ok)
            self.textbox2.setText("")
            self.spin1.clear()

    @pyqtSlot()
    def generer(self):
        os.system("TP_p2.py")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = App()
    sys.exit(app.exec_())

```