

1

Parcours de tableaux

« Ordre, Permutations, Jeux. »

2

Jouer avec les mots

« *Reconnaissance, Construction, Codage.* »

2.1 Réécriture de mots

On considère les mots écrits sur l'alphabet $\{a, b, A, B\}$, tel que $w = abbaBabAA$ par exemple. Soit ε le mot vide. On dit que deux mots u et v sont en relation si on peut réécrire des parties de u de façon à obtenir v après une suite de transformations effectuées en suivant les règles suivantes :

aA	→	ε
Aa	→	ε
ε	→	aA
ε	→	Aa
bB	→	ε
Bb	→	ε
ε	→	bB
ε	→	Bb
aab	→	baa
baa	→	aab
bba	→	abb
abb	→	bba

Par exemple :

	aababaabAAABBB
→	aababbbaaAAABBBB
→	aababbABBB
→	aababbABBB
→	aabbbaABBB
→	aa

Question 1

Montrer que cette relation est une relation d'équivalence. Montrer que aa et AA commutent avec toutes les lettres.

Question 2

Montrer que tout mot w est équivalent à un mot xyz (formé de trois mots x , y et z mis bout à bout), où : x est de la forme $aa...aa$ ou $AA...AA$ et de longueur paire, y est de la forme $bb...bb$ ou $BB...BB$ et de longueur paire, z de la forme $ababa...bab$, ou $ba...bab$, ou $ab...aba$, ou $ba...ba$, c'est-à-dire alterne les lettres a et b , commençant par a ou b et finissant par a ou b . Un mot de la forme xyz est dit canonique.

Question 3

Proposer un codage des mots canoniques sous forme de triplets d'entiers.

Question 4

Écrire une fonction `forme3 (w:string) : bool` qui prend en entrée un mot w et teste si w est un mot alternant les lettres a et b , c'est-à-dire de la forme de z .

Question 5

Écrire une fonction `ajouter_a (iu, ju, ku: int * int * int) : (int * int * int)` qui prend en entrée un mot canonique u , codé par un triplet (iu, ju, ku) , et donne en sortie le codage (iv, jv, kv) du mot $v = ua$.

Question 6

Écrire une fonction `representant_canonique (w:string) : (int * int * int)` qui prend pour entrée un mot quelconque w et donne en sortie un triplet (i, j, k) codant un mot canonique équivalent à w .

————— CORRIGÉ —————

Question 1

Réflexivité : il suffit de prendre une suite de transformations réduite à l'ensemble vide.
Symétrie : pour chaque règle, la transformation inverse est dans l'ensemble des règles donc toute suite de transformations peut être inversée.

Transitivité : évidente.

On a donc une relation d'équivalence.

Commutation de aa avec toutes les lettres : aa commute évidemment avec a .

$aaA \rightarrow a \rightarrow Aaa$, donc aa commute avec A .

$aab \rightarrow baa$ est une règle, donc aa commute avec b .

$aaB \rightarrow BbaaB \rightarrow BaabB \rightarrow Baa$, donc aa commute avec B .

Commutation de AA avec toutes les lettres : $AAa \rightarrow A \rightarrow aAA$, donc AA commute avec a.

AA commute évidemment avec A.

$AAb \rightarrow AAbaaAA \rightarrow AAaabAA \rightarrow bAA$, donc AA commute avec b.

$AAB \rightarrow AABaaAA \rightarrow AAaABAA \rightarrow BAA$, donc AA commute avec B.

Remarquons que, de façon symétrique, bb et BB commutent également avec toutes les lettres.

Question 2

Pour transformer w , on commence par bouger toutes les suites de deux lettres consécutives identiques et les mettre au début du mot, les aa et AA précédant les bb et BB, et par simplifier toutes les occurrences de aA, Aa, bB ou Bb. On se retrouve avec un mot commençant par des aa et AA, continuant avec des bb et BB, et se terminant avec un mot qui alterne des a ou A avec des b ou B. On remplace alors chaque A par AAa et chaque B par BBb, puis on ramène les AA et les BB plus au début : on se retrouve avec une suite de aa et de AA, suivie d'une suite de bb et de BB, suivie d'un mot de la forme de z . En simplifiant les aaAA, les AAaa, les bbBB et les BBbb, on obtient mot xyz de la forme requise. Notons qu'il n'est pas demandé ici de montrer l'unicité de cette écriture.

Question 3

On peut coder x par un entier i , positif si x contient des aa et négatif s'il contient des AA, et de valeur absolue égale au nombre de lettres de x . De même, y peut être codé par un entier j . On peut coder z par un entier k , positif si z commence par un a et négatif si z commence par un b, et de valeur absolue égale au nombre de lettres de z .

Question 4

Supposons la longueur de w supérieure ou égale à 1. On regarde la première lettre de w , puis on teste si les suivantes alternent, en s'arrêtant dès qu'on trouve une erreur ou qu'on a parcouru tout le mot.

```
let forme3 (w:string) : bool =
  if w.[0] <> 'a' && w.[0] <> 'b' then false else

  let last = ref w.[0] and res = ref true in
  let n = String.length w in
  for i=1 to n-1 do
    if !last = 'a' && w.[i] <> 'b' || !last = 'b' && w.[i] <> 'a'
      then res := false;
    last := w.[i]
  done;
  !res
```

Question 5

La fonction n'est pas difficile mais demande un peu d'attention pour ne pas faire d'étourderie : il est particulièrement recommandé d'écrire d'abord l'algorithme en détail. Si z est non vide et se termine par un b : ku strictement positif et pair ou strictement négatif et impair ; alors z s'allonge d'une lettre : ku augmente de 1 dans le premier cas et diminue de 1 dans le deuxième cas. Si z est non vide et se termine par un a : ku strictement positif et impair ou strictement négatif et pair ; alors z raccourcit d'une lettre et x change : ku diminue de 1 dans le premier cas et augmente de 1 dans le second cas, et iu augmente de 2 s'il était positif et diminue de 2 s'il était strictement négatif. Si z est vide : $ku = 0$; alors ku devient égal à 1. D'où la fonction suivante :

```
let ajouter_a (iu, ju, ku: int * int * int) : (int * int * int) =
  if ku = 0 then (iu, ju, 1) else (
    if ku > 0 && ku mod 2 = 0 then (iu, ju, ku+1)
    else if ku < 0 && ku mod 2 = 1 then (iu, ju, ku-1)
    else if ku > 0 && ku mod 2 = 1 then (
      if iu = 0 then (iu+2, ju, ku-1) else (iu-2, ju, ku-1)
    )
    else ( (* ku < 0 && ku mod 2 = 0 *)
      if iu = 0 then (iu+2, ju, ku+1) else (iu-2, ju, ku+1)
    )
  )
)
```

Question 6

On peut supposer qu'on dispose des fonctions `ajouter_b`, `ajouter_A` et `ajouter_B` similaires à celle de la question 5. Il est alors facile d'ajouter des lettres une par une pour construire un mot canonique pour chaque préfixe de w et finalement pour w .

```
let representant_canonique (w:string) : (int * int * int) =
  let n = String.length w in
  let mot = ref (0, 0, 0) in
  for i=0 to n-1 do
    if w.[i] = 'a' then mot := ajouter_a !mot else
    if w.[i] = 'b' then mot := ajouter_b !mot else
    if w.[i] = 'A' then mot := ajouter_A !mot else
    mot := ajouter_B !mot
  done;
  !mot
```

La relation d'équivalence étudiée dans cet exercice correspond à un groupe donné par une permutation finie : les éléments sont les classes d'équivalence de la relation définie à la question 1, et on peut facilement vérifier que cette relation est compatible avec la concaténation. L'étude faite ici consiste en la construction d'une « structure automatique » pour le groupe, permettant de calculer un représentant distingué de la classe du produit de deux éléments donnés. L'étude de groupes automatiques est un domaine de recherche récent et actif en mathématiques.