1

Parcours de tableaux

« *Ordre, Permutations, Jeux.* »

2

Jouer avec les mots

« *Reconnaissance, Cosntruction, 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 :

$$\begin{array}{rcl} \text{aA} & \to & \varepsilon \\ \text{Aa} & \to & \varepsilon \\ \varepsilon & \to & \text{aA} \\ \varepsilon & \to & \text{Aa} \\ \text{bB} & \to & \varepsilon \\ \text{Bb} & \to & \varepsilon \\ \varepsilon & \to & \text{bB} \\ \varepsilon & \to & \text{Bb} \\ \text{aab} & \to & \text{baa} \\ \text{baa} & \to & \text{aab} \\ \text{bba} & \to & \text{abb} \\ \text{abb} & \to & \text{bba} \end{array}$$

Par exemple :

$$\begin{array}{rl} & \text{aababaabAAABBB} \\ \to & \text{aababbaaAAABBBB} \\ \to & \text{aababbABBB} \\ \to & \text{aababbABBB} \\ \to & \text{aabbbaABBB} \\ \to & \text{aa} \end{array}$$

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 $v = u$a.

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 $\to$ a $\to$ Aaa, donc aa commute avec A.
aab $\to$ baa est une règle, donc aa commute avec b.
aaB $\to$ BbaaB $\to$ BaabB $\to$ Baa, donc aa commute avec B.

Commutation de AA avec toutes les lettres : AAa $\to$ A $\to$ aAA, donc AA commute avec a.
AA commute évidemment avec A.
AAb $\to$ AAbaaAA $\to$ AAaabAA $\to$ bAA, donc AA commute avec b.
AAB $\to$ AABaaAA $\to$ AAaaBAA $\to$ 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 alternes, 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 difficicle 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 : $\texttt{ku} = 0$; 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.

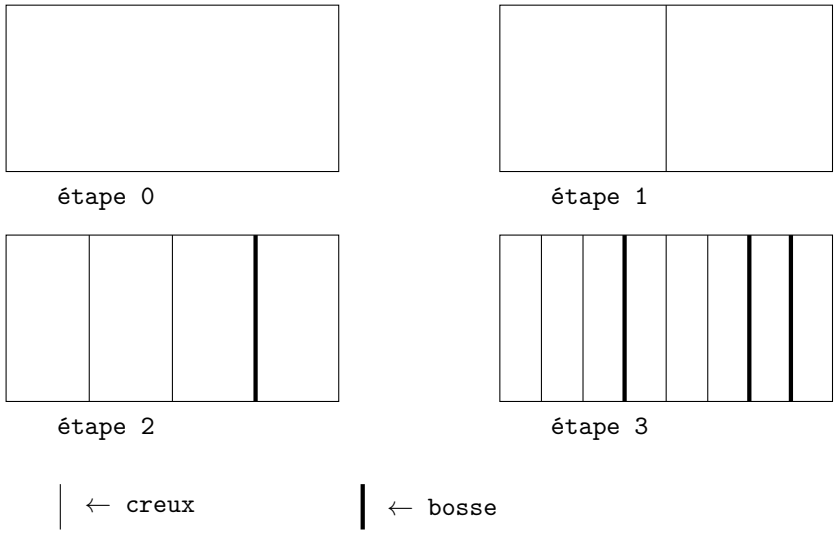
```
let representant_canonique (w:string) : (int * int * int) =
  let n = String.length w in
  let mot = ref (0, 0, 0) in
  for i=0 to n-1 do
    if w.[i] = 'a' then mot := ajouter_a !mot else
    if w.[i] = 'b' then mot := ajouter_b !mot else
    if w.[i] = 'A' then mot := ajouter_A !mot else
    mot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

⋆⋆⋆⋆⋆⋆⋆⋆

2.2 Pliage de papier

On prend une feuille de papier et on la plie n fois dans le sens vertical, en repliant à chaque fois la moitié droite sur la gauche. Les plis de la feuille, une fois redépliée, sont une suite de creux et de bosses. Voir figure :



Question 1

Combien y a-t-il de plis à la n-ième étape ?

Question 2

On représente chaque étape du pliage par un mot. Un creu est codé par un 0 et une bosse par un 1. Ainsi : $w_0 = \varepsilon$ (mot vide)
$w_1 = 0$
$w_2 = 001$
$w_3 = 0010011$
...
Montrer que w_i est toujours un préfixe de w_{i+1}, c'est à dire que le début de w_{i+1} coïncide avec w_i.

Question 3

Donner un algorithme de construction de w_n à partir de w_{n-1}. Écrire une fonction de calcul de w_n. On prendra comme entrée n et on renverra une liste `w` d'entiers remplie adéquatement.

Question 4

Écrire une fonction qui prend pour entrée un entier n et renvoie une liste contenant la représentation binaire de n, poids fort en tête.

Question 5

Les mots de la suite de pliage étant préfixes les uns des autres, on peut considérer le mot infini w dont ils sont tous préfixes. Proposer un algorithme qui prend pour entrée un entier n et renvoie le n-ième bit de w.

CORRIGÉ

Question 1

Si on considère les intervalles entre les plis (y compris les côtés de la feuille), on a 1 intervalle, puis 2, puis 4, etc. ; à chaque étape, chaque intervalle est coupé en deux et le nombre d'intervalles double. Après n étapes, il y a donc 2^n intervalles, et le nombre de plis est $2^n - 1$.

Question 2

Déchirons le papier le long du pli de la première étape et débarrassons-nous de la moitié de droite : la k-ième étape de pliage du papier initial est comme la $(k-1)$-ième étape de pliage du demi-papier, et donc w_{k-1} forme la moitié gauche de w_k.

Question 3

Le pli du milieu est un creux et donc donne un 0 au milieu de w_n. Le demi-papier droit est plié, après la première étape, comme le demi-papier gauche, sauf qu'il est tourné dans l'autre sens. Soit $\mathrm{r}(w)$ le mot obtenu à partir d'un mot w en lisant les chiffres de droite à gauche, et c(w) le mot obtenu à partir de w en remplaçant chaque 0 par un 1 et chaque 1 par un 0 : on a la relation de récurrence

$$w_n = w_{n-1}\, 0\, \mathrm{c}(\mathrm{r}(w_{n-1}))$$

De cette relation, se déduit facilement la fonction permettant de construire w_n.

```
let rec motdepliage (n:int) : (int list) =
  if n = 0 then [0] else
  let pred = motdepliage (n-1) in
  let pred' = List.map (fun e -> 1-e) (List.rev pred) in
  pred @ (0::pred')
```

Question 4

Question classique et accessible à tous.

```
let binaire (n:int) : (int list) =
  let rec aux n acc =
    if n = 0 then acc
    else aux (n / 2) ((n mod 2) :: acc)
  in
  if n = 0 then [0] else aux n []
```

Question 5

On peut trouver un autre point de vue pour construire w_k par récurrence à partir de w_{k-1} : on intercale un nouveau pli entre chaque paire de plis consécutifs de w_{k-1}, et ce pli est alternativement 0 ou 1, le premier étant 0. Ainsi : $w_k = 0x1x0x...x1$, si $w_{k-1} = xxx...xx$. Donc le n-ième bit de w est facile à trouver si n est impair : c'est 0 si n est congru à 1 modulo 4 et c'est 1 si n est congru à 3 modulo 4. Si n est pair, on divise n par deux et on remarque que le n-ième bit de w_k est le $(n/2)$-ième bit de w_{k-1}, donc si $n/2$ est impair, il suffit encore une fois de tester si $n/2$ est congru à 1 ou à 3 modulo 4. Si $n/2$ est est pair, on redivise par deux, et ainsi de suite. Lorsque n est écrit en binaire, on regarde son bit le plus à droite (celui de poids le plus faible), puis le bit immédiatement à sa gauche, et ainsi de suite jusqu'à trouver un bit égal à 1. Soit x le n-ième bit de w. On a :
Si $n = 1\ 0\ 0\ ...\ 0$, alors $x = 0$.
Si $n = *\ *\ ...\ *\ 0\ 1\ 0\ 0\ ...\ 0$, alors $x = 0$.
Si $n = *\ *\ ...\ *\ 1\ 1\ 0\ 0\ ...\ 0$, alors $x = 1$.
Ceci donne un programme extrêmement simple.
TODO : Vérifier cet algorithme

```
let bit (n:int) : int =
  assert(n > 0);
  let n = ref n in
  while !n mod 2 = 0 do
    n := Int.shift_right !n 1
  done;
  if !n mod 4 = 1 then 0 else 1
```

Des études du genre de celle faite dans cet exercice peuvent servir à montrer des propriétés d'algébricité ou de transcendance du nombre étudié. C'est actuellement un domaine actif de recherche en France. On obtient une suite de courbes (C_n) approximant une courbe de Peano (c'est-à-dire une courbe qui remplit le plan), définie à partir de (w_n), en dessinant un segment de longueur $1/2^{n/2}$ pour chaque bit lu, et en tournant à droite (d $+\pi/2$) ou à gauche (de $-\pi/2$) à chaque pas, selon que le bit lu est 0 ou 1. Le premier segment est dessiné dans la direction $\pi/4$. La figure suivante illustre les premiers pas :
TODO : Figures des courbes

⋆⋆⋆⋆⋆⋆⋆⋆