Exercice I

1. **Quelques propriétés** :

(a) — Le seul mot de Lukasiewicz de longueur 1 est (-1)
— Il n'y a pas de mot de Lukasiewicz de longueur 2 car $w_1 + w_2 \in \{-2, 0, 2\}$.
— Le seul mot de Lukasiewicz de longueur 3 est $(1, -1, -1)$

Pour finir, il n'y a pas de mots de longueur paire. En effet si $w = w_1 w_2 \cdots w_{2p}$ est un mot de longueur $2p$, on a $\sum_{i=1}^{2p} w_i = \sum_{j=1}^{p} (w_{2j-1} + w_{2j})$. Or $w_{2j-1} + w_{2j} \in \{-2, 0, 2\}$ est pair donc $\sum_{i=1}^{2p} w_i$ est pair et ne peut donc pas valoir -1.

(b) On utilise une fonction auxiliaire `parcours : int -> int list -> bool` qui parcourt la liste, le paramètre `somme` est la somme des k premiers termes et `l1` le reste de la liste.

```
let lukasiewicz l =
  let rec parcours somme l1 = match l1 with
      | [] -> somme = (- 1)
      | t :: q -> (somme >= 0) && parcours (somme + t) q
  in aux 0 l;;
```

On ne parcourt qu'une fois la liste donc la fonction a une complexité (dans le pire des cas) linéaire en la longueur du mot.

(c) i. Soit $u \in L$ et $v \in L$. Notons $n_u = |u|$ la longueur de u et $n_v = |v|$ celle de v. Si on note $w = (+1).u.v$ alors :
— Pour $k = 1$, $w_1 = 1 \geq 0$
— pour $k \in [\![2, n_u + 1]\!]$, $\sum_{i=1}^{k} w_i = 1 + \sum_{i=1}^{k-1} u_i \geq 1 + (-1) \geq 0$
— pour $k \in [\![n_u + 2, n_u + n_v]\!]$, $\sum_{i=1}^{k} w_i = 1 + \sum_{i=1}^{n_u} u_i + \sum_{i=1}^{k-n_u-1} v_i \geq 0$.
— Pour $k = n_u + n_v + 1$, $\sum_{i=1}^{k} w_i = 1 + (-1) + (-1) = -1$.

On en déduit bien que $(+1).u.v \in L$.

ii. Soit w un mot de *Lukasiewicz* de longueur n supérieure ou égale à 3.
— Existence : On a $w_1 = +1$ car sinon (P_2) n'est pas vérifiée pour $k = 1$. Notons alors p le plus petit entier tel que $\sum_{i=1}^{p} w_i = 0$. Un tel entier existe car si on pose $S_k = \sum_{i=1}^{k} w_i$, on a $S_1 = 1$ et $S_n = -1$, et que pour tout k, $|S_{k+1} - S_k| = |w_k| = 1$. On pose alors $u = w_2 w_3 \cdots w_p$ et $v = w_{p+1} \cdots w_n$ de telle sorte que $w = (+1).u.v$

Maintenant, pour k compris entre 1 et $|u| - 1$, $\sum_{i=1}^{k} u_i = \sum_{j=2}^{k+1} w_j = -1 + \sum_{j=1}^{k+1} w_j$.

Comme, d'après la définition de p, pour $k \leq |u| - 1 = p - 2$, $S_{k+1} > 0$, on a bien $\sum_{i=1}^{k} u_i \geq 0$ et donc (P_2) est vérifiée.

Maintenant pour $k = |u| = p - 1$, la même égalité donne que $\sum_{i=1}^{p-1} u_i = -1 + \sum_{j=1}^{p} w_j = -1$, là encore par définition de p.

Finalement $u \in L$.

On montre de plus que v est aussi un mot de Lukasiewicz. Il suffit de voir que pour tout $k \in [\![1, |v|]\!]$,

$$\sum_{i=1}^{k} v_i = \sum_{j=p+1}^{k+p} w_j = \sum_{j=1}^{k+p} w_j \text{ car } \sum_{j=1}^{p} w_j = 0.$$

— Unicité : On suppose qu'il existe u et v dans L tels que $w = (+1).u.v$. En notant $p = |u| + 1$, on a que $\sum_{j=1}^{p} w_j = 1 + \sum_{j=1}^{|u|} u_j = 0$. Donc p vérifie bien que $\sum_{j=1}^{p} w_j = 0$ et s'il existe $p' < p$ vérifiant la même relation alors $\sum_{j=1}^{p'-1} u_j = -1$ ce qui est absurde car $u \in L$. Finalement, la décomposition proposée est bien la seule.

iii. Voici la fonction

```
let decompose l =
   let rec parcours somme li =
      match li with
         | [] -> failwith "impossible"
         | t1 :: q1 -> if (somme + t1) <> 0
                       then let (u, v) = parcours (somme + t1) q1 in
                            (t1 :: u, v)
                       else [t1], q1
   in match l with
       | [] -> failwith "impossible"
       | t :: q -> parcours t q;;
```

La fonction auxiliaire `parcours : int -> int list -> int list*int list` permet de parcourir le mot en gardant (dans la variable `somme`) la somme des valeurs précédentes afin de déterminer quand cette somme vaut 0 pour la première fois.

(d) Il semble nécessaire de stocker au fur et à mesure les mots de Lukasiewicz de longueur inférieure afin de ne pas les recalculer systématiquement.

(e) On va construire un tableau `t` tel que pour tout `i`, `t.(i)` contient la liste des mots de Lukasiewicz de longueur $2i+1$. Le tableau `t` est donc de type `int list list array`. On initialise, `t.(0)` à `[[-1]]` car il n'y a qu'un seul mot de Lukasiewicz de longueur 1.

Ensuite on remplit les cases du tableau en utilisant la propriété précédente. Précisément, pour remplir la ligne `t.(i)` on ajoute tous les mots de la forme $(+1).u.v$ où u est de longueur $k \in [\![1, 2i-1]\!]$ et v de longueur $i-k-1$.

```
let rec recomposeMotListe mot l =
  match l with
    | [] -> []
    | t :: q -> (1 :: (mot @ t)) :: (recomposeMotListe mot q);;

let rec recomposeListes l1 l2 =
  match l1 with
    | [] -> []
    | t :: q -> (recomposeMotListe t l2) @ (recomposeListes q l2);;
```

La fonction `recomposeMotListe : int list -> int list list -> int list list` construit la liste des mots obtenus en concaténant (+1), le mot `mot` et un mot de la liste `l`. On en déduit la fonction `recomposeListes : int list list -> int list list -> int list list` qui construit la liste des mots obtenus en concatenant (+1), un mot de la liste `l1` et un mot de la liste `l2`.

```
let obtenirLuka n =
  if n mod 2 = 0 then [] else
    (let p = n / 2 in
        let t = Array.make (p+1) [] in
          t.(0) <- [[- 1]];
          for i = 1 to (p) do
            for k = 0 to i - 1 do
              t.(i) <- (recomposeListes t.(k) t.(i - k - 1)) @ t.(i)
            done;
          done;
          t.(p));;
```

2. **Dénombrement** :

(a) i. Soit $w = w_1 w_2 \cdots w_n$ un mot de $A^\star$ vérifiant la condition (P_1). Pour tout i compris entre 1 et n, le mot $\tilde{w} = w_i w_{i+1} \cdots w_n w_1 w_2 \cdots w_{i-1}$ vérifie encore la condition (P_1) par commutativité de l'addition.

Notons $Z = \{i \in [\![1,n]\!], S_i \text{ minimal}\}$ et posons $i_0 = \min Z$. C'est-à dire que si on regarde les sommes partielles, S_{i_0} est le premier minimum. On en déduit que pour $k \in [\![i_0+1, n]\!]$, $\sum_{i=i_0}^{k} w_i \geq 0$ car $i_0 \in Z$ et pour $k \in [\![1, i_0-1]\!]$,

$$w_{i_0} + \cdots + w_n + w_1 + \cdots + w_k = \sum_{i=i_0}^{n} w_i + \sum_{i=1}^{k} w_i = -1 - \sum_{i=k+1}^{i_0-1} w_i$$

Mais, $\sum_{i=k+1}^{i_0-1} w_i$ est strictement positif car i_0 est **le premier** minimum donc finalement $w_{i_0}+\cdots+w_n+w_1+\cdots+w_k \geq 0$. La propriété (P_2) est vérifiée.

Maintenant, pour montrer l'unicité, il suffit de montrer qu'une rotation d'un mot de Lukasiewicz n'est pas un mot de Lukasiewicz. Si $w \in L$ et $i \in [\![2,n]\!]$,

$$\sum_{k=i}^{n} w_i = \sum_{k=1}^{n} w_i - \sum_{k=1}^{i-1} w_i \leq \sum_{k=1}^{n} w_i = -1$$

Cela contredit la propriété (P_2).

ii. On veut écrire une fonction `conjugue : int list -> int list` qui associe à tout mot vérifiant (P_1) son conjugué. On parcourt le mot en calculant les sommes partielles afin de déterminer i_0. Il reste alors à effectuer la permutation.

Commençons par écrire la fonction de décalage telle que `decal i l []` renvoie la liste $(\ell_{i+1}, \ldots, \ell_n, \ell_1, \ldots, \ell_i)$ si $\ell = (\ell_1, \ldots, \ell_n)$

```
let rec decal i l acc =
   if i = 0 then l @ acc
   else match l with
           | [] -> []
           | t :: q -> decal (i - 1) q  (t::acc);;
```

Il ne reste plus qu'à trouver i_0 par un parcours (le fait que l'on ait une inégalité stricte permet de retourner le premier minimum) :

```
let conjugue l =
   let rec parcours li imin vmin somme i =
      match li with
         | [] -> imin
         | t :: q -> if somme + t < vmin
                     then parcours q i (somme + t)  (somme +t) (i + 1)
                     else parcours q imin vmin (somme+t) (i + 1)
   in let i = parcours l 0 0 0 1 in
   decal i l [];;
```

(b) Soit $n \in \mathbb{N}$. On considère M_n l'ensemble des mots sur A de longueur $2n+1$ vérifiant la propriété (P_1) et L_n l'ensemble des mots de Lukasiewicz de longueur $2n+1$. Comme la somme des chiffres d'un mot de M_n vaut -1, il y a n fois la lettre $(+1)$ et $n+1$ fois la lettre (-1). Pour déterminer uniquement un tel mot, il suffit de positionner les n lettres $(+1)$. Le cardinal de M_n est donc $|M_n| = \binom{2n+1}{n}$.

On considère sur M_n la relation $\sim$ telle que $w \sim w'$ s'il existe une rotation de w qui est égale à w'. Il est clair que $\sim$ est une relation d'équivalence. On a vu en 2.a.i) que tout mot de M_n est en relation avec un et un seul mot de Lukasiewicz ce qui revient à dire qu'il y a autant de classes d'équivalences pour $\sim$ que de mots de Lukasiewicz.

Maintenant soit $u \in L$, on note C_u la classe d'équivalence de u. On a que $|C_u| = 2n+1$ car, en effectuant les $2n+1$ rotations on ne retombe pas sur deux mots identiques. En effet si pour $i < j$,

$$u_i \ldots u_{2n+1} u_1 \ldots u_{i-1} = u_j \ldots u_{2n+1} u_1 \ldots u_{j-1}$$

alors en posant $x = u_i \dots_{j-1}$ et $y = u_j u_j \dots u_{2n+1} u_1 \dots u_{i-1}$ on a $xy = yx$. De ce fait, x et y sont d'après le lemme de Levi des puissances (non triviale) d'un même mot. Notons $x = v^p$ et $y = v^q$ où $p \geq 1$ et $q \geq 1$. Si on note d la somme des chiffres qui composent v on a que la somme des chiffres de xy (qui vaut -1) vaut $(p+q)d$. Ceci n'est pas possible car $p + q \geq 2$ ne peut pas diviser -1.

On en déduit finalement que M_n se partitionne en $|L_n|$ ensembles chacun de cardinal $2n+1$. De ce fait,

$$|L_n| = \frac{1}{2n+1}|M_n| = \frac{1}{2n+1}\binom{2n+1}{n} = \frac{1}{n+1}\binom{2n}{n}$$

Exercice II

1. Soit $A \in \mathcal{V}$ et f une tri-valuation définie sur $\mathcal{V}$ telle que $f(A) =?$. On a alors $\hat{f}(\neg A) =?$ et par suite $\hat{f}(A \vee \neg A) =? \neq \top$. On en déduit que $\hat{f}$ ne satisfait pas $A \vee \neg A$; cette dernière n'est donc pas une tautologie.
2. Soit $A \in \mathcal{V}$. On peut proposer : $\phi = (A \Rightarrow A)$. On peut établir la table de vérité de ϕ en ne faisant apparaitre que la valuation de A.

A	$A \Rightarrow A$
$\top$	$\top$
$\bot$	$\top$
$?$	$\top$

3. On voit que pour tout couple $(A, B) \in \mathcal{V}^2$ et toute tri-valuation f :

$$\hat{f}(A \wedge B) = \min\left(f(A), f(B)\right) \quad \text{et} \quad \hat{f}(A \vee B) = \max\left(f(A), f(B)\right)$$

4. Il suffit d'écrire les tables de vérités :

A	B	$A \Rightarrow B$	$\neg A \vee B$
$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$
$\top$	$?$	$?$	$?$
$\bot$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\top$
$\bot$	$?$	$\top$	$\top$
$?$	$\top$	$\top$	$\top$
$?$	$\bot$	$?$	$?$
$?$	$?$	$\top$	$?$

On voit que les deux formules ne sont pas équivalentes car la dernière ligne n'est pas la même.

5. Comme demandé, on construit la table de vérité conjointe des deux propositions pour les comparer :

A	B	$A \Rightarrow B$	$\neg B$	$\neg A$	$\neg B \Rightarrow \neg A$
$\top$	$\top$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$?$	$?$	$?$	$\bot$	$?$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$
$\bot$	$?$	$\top$	$?$	$\top$	$\top$
$?$	$\top$	$\top$	$\bot$	$?$	$\top$
$?$	$\bot$	$?$	$\top$	$?$	$?$
$?$	$?$	$\top$	$?$	$?$	$\top$

On en déduit que : Les propositions $\neg B \Rightarrow \neg A$ et $A \Rightarrow B$ sont équivalentes

6. Soit $\psi = ((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)) \Rightarrow B$. On en construit la table de vérité :

A	B	$A \Rightarrow B$	$\neg A \Rightarrow B$	$((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)$	B	ψ
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\top$	?	?	$\top$	?	?	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	?	$\top$	?	?	?	$\top$
?	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
?	$\bot$	?	?	?	$\bot$	?
?	?	$\top$	$\top$	$\top$	?	?

On constate en observant les deux dernières lignes que ψ n'est pas une tautologie.

7. (a) La dernière ligne de la table de vérité montre f est une tri-valuation telle que $f(A) =?$ alors

$$\hat{f}(A \to A) =? \neq \top$$

Cela implique que $A \to A$ n'est pas une tautologie

(b) Notons q la tri-valuation de $\mathcal{V}$ dans $\{\top, \bot, ?\}$ définie par : $\forall A \in \mathcal{V}, q(A) =?$. Pour montrer qu'il n'existe pas de tautologie dans la logique tri-valuée associée aux opérateurs $\{\neg, \wedge, \vee, \to\}$, il suffit d'établir que : $\forall \phi \in \mathcal{F}, \hat{q}(\phi) =?$. Montrons cela par induction structurelle. Soit $\phi \in \mathcal{F}$.

- si $\phi \in \mathcal{V}$ alors $\hat{q}(\phi) = q(\phi) =?$
- supposons $\phi = \neg\phi_0$ et $\hat{q}(\phi_0) =?$. La table de vérité de l'opérateur $\neg$ montre que dans ce cas $\hat{q}(\phi) = \hat{q}(\neg\phi_0) =?$
- supposons $\phi = \phi_1 \diamond \phi_2$, où ϕ_1, ϕ_2 vérifient l'hypothèse d'induction et $\diamond$ représente l'un quelconque des opérateurs binaires $\wedge$, $\vee$ ou $\to$. En remarquant qu'une tri-valuation indéterminée à la fois de ϕ_1 et ϕ_2 correspond à la dernière ligne du tableau de vérité de tous ces opérateurs binaires, et que le résultat est dans les trois cas égal à ?, on en déduit que $\hat{q}(\phi) = \hat{q}\,(\phi_1 \diamond \phi_2) =?$

Il n'existe pas de tautologie dans la logique tri-valuée associée aux opérateurs $\{\neg, \wedge, \vee, \to\}$.