TP OCaml n°1

Le jeu du solitaire

Préliminaires

Quelques opérations sur les entiers

Dans ce TP, nous supposerons les entiers OCaml (type `int`) codés sur 62 bits. Pour $k \in [\![0, 61]\!]$ et pour $n \in [\![0, 2^{62} - 1]\!]$, on notera $[n]_k$ le k-ième bit de poids le plus faible dans la représentation binaire de n. On notera le nombre associé à une représentation binaire par une barre au-dessus. Ainsi, nous pouvons donc écrire

$$n = \overline{[n]_{61}\,[n]_{60} \cdots [n]_0}.$$

On introduit plusieurs opérateurs sur les entiers. Ces opérateurs réalisent des opérations sur la représentation binaire de des deux nombres considérés. Ils s'utilisent avec des éléments de type `int`. On définit :

(1) le « et » logique, `land` en OCaml, tel que a `land` $b = \overline{\left([a]_{61} \wedge [b]_{61}\right) \cdots \left([a]_0 \wedge [b]_0\right)}$;

(2) le « ou » logique, `lor` en OCaml, tel que a `lor` $b = \overline{\left([a]_{61} \vee [b]_{61}\right) \cdots \left([a]_0 \vee [b]_0\right)}$;

(3) le « xor » logique,[1] `lxor` en OCaml, tel que a `lxor` $b = \overline{\left([a]_{61} \oplus [b]_{61}\right) \cdots \left([a]_0 \oplus [b]_0\right)}$;

(4) le « non » logique, `lnot` en OCaml, tel que `lnot` $a = \overline{\left(\neg[a]_{61}\right) \cdots \left(\neg[a]_0\right)}$;

(5) le « décalage vers la gauche », `lsl` en OCaml, tel que a `lsl` $b = \overline{[a]_{61-b} \cdots [a]_0\, 0 \cdots 0}$;

(6) le « décalage vers la droite », `lsr` en OCaml, tel que a `lsr` $b = \overline{0 \cdots 0\, [a]_{61} \cdots [a]_b}$.

L'exemple considéré dans le tableau ci-dessous (Tableau 1) montre l'effet des opérations définies ci-avant sur les entiers $a = 6$ et $b = 3$.

Expression	Représentation binaire	Résultat numérique
a	0000 ··· 0110	6
b	0000 ··· 0011	3
a `land` b	0000 ··· 0010	2
a `lor` b	0000 ··· 0111	7
a `lxor` b	0000 ··· 0101	5
a `lsl` b	0 ··· 0110000	48
a `lsr` b	0000000 ··· 0	0

Tableau 1. Exemple sur les opérations logiques sur les entiers OCaml

L'entier 2^n peut être obtenu aisément à l'aide de l'expression `1 lsl n`.

[1] Le « xor » vient de l'anglais *exclusive or* : il s'agit du « ou exclusif ».

Jeu du solitaire

Le *jeu du solitaire* se joue sur un tablier percé d'encoches disposées en quadrillage et remplissant une certaine forme géométrique. Parmi les tabliers les plus fréquents, le *tablier européen* est formés de 37 encoches organisées en octogone, comme le montre la figure, dont on peut décrire les coordonnées par l'ensemble

$$\{(x, y) \in [\![0, 6]\!]^2 \mid |x - 3| + |y - 3| \leq 4\}.$$

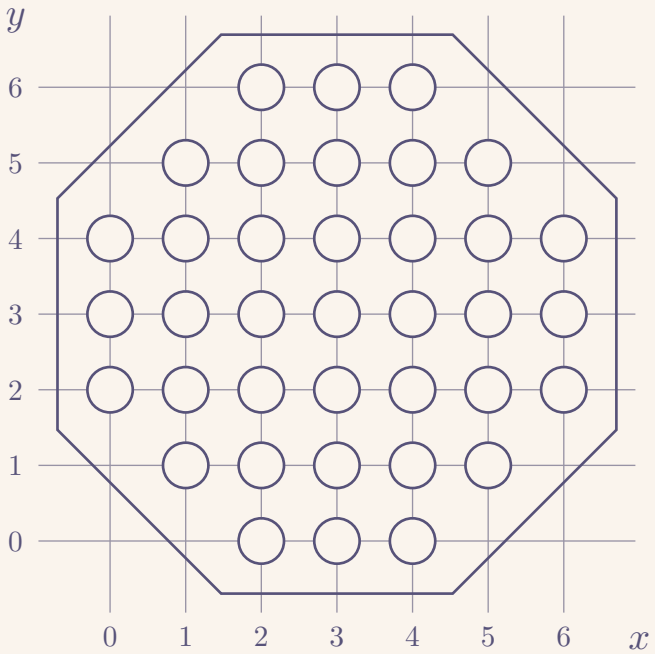


Fig. 1. Tablier européen

D'autres tabliers existent, comme le tablier rectangulaire de dimension $n \times 1$, où $n \in \mathbb{N}$. Ce tablier rectangulaire sera étudié dans le TD Notations n°1.

Sur un tablier, toute encoche de coordonnées (x, y) possède au plus quatre encoches voisines, de coordonnées $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$ et $(x, y - 1)$, sous reserve d'existence.

Un *motif* est un sous-ensemble d'encoches dans un tablier. Un motif est *ponctuel* s'il est de cardinal 1.

Au début du jeu, des billes (ou des fichets) viennent se loger dans les encoches d'un motif initial. En cours de jeu, un *coup simple* consiste à déplacer l'une des billes en sautant par-dessus l'une des encoches voisines, pour atteindre celle immédiatement après. Ce coup peut être joué à condition que l'encoche voisine soit occupée et que la bille déplacée retombe dans une encoche inoccupée. À l'issue d'un coup simple, la bille dans l'encoche intermédiaire est retirée. On donne un exemple de coup simple en Fig. 2.

On représente sur la Fig. 2, la bille déplacée en rouge. Les autres billes sont en noir.

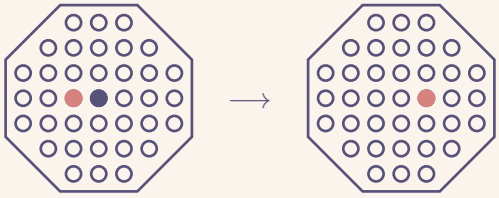


Fig. 2. Exemple de coup simple

En cours de jeu, un *coup composé* consiste à jouer zéro, un ou plusieurs coups simples en déplaçant la même bille. On en donne un exemple en Fig. 3.

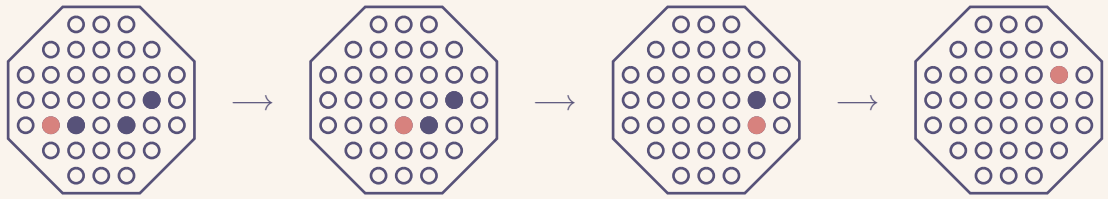


Fig. 3. Exemple de coup composé

Une suite de motifs obtenus en jouant une suite de coups est appelée une *partie*. L'objectif de la joueuse ou du joueur est de transformer un motif en un autre motif par une suite de coups.

Ce sujet de TP est extrait de l'épreuve d'informatique du concours Mines-Ponts de 2021 pour les MP en option. Dans ce sujet, on traitera la partie 1 de l'épreuve.

Dans ce sujet de TP, on considèrera une joueuse jouant sur un **tablier européen**. Elle souhaite transformer un motif initial en un motif final en un minimum de coup.

I. Partie jouée sur le tablier européen en un minimum de coups

Q1. À titre préliminaire, nommer le type de parcours de graphe le plus approprié pour déterminer un chemin entre deux sommets qui minimise le nombre d'arcs traversés. Quelle politique de mise en attente de sommets caractérise ce parcours ?

I.1. Numérotation du tablier

Nous numérotons l'encoche de coordonnées $(x, y) \in \mathbb{N}^2$ dans le tablier européen par l'entier $z = 8y + x \in \mathbb{N}$ (Fig. 4). Nous notons l'ensemble des numéros d'encoches du tablier européen par :

$$\mathscr{E} = \{8y + x \mid (x, y) \in [\![0, 6]\!]^2 \text{ et } |x - 3| + |y - 3| \leq 4\} \subseteq \mathbb{N}.$$

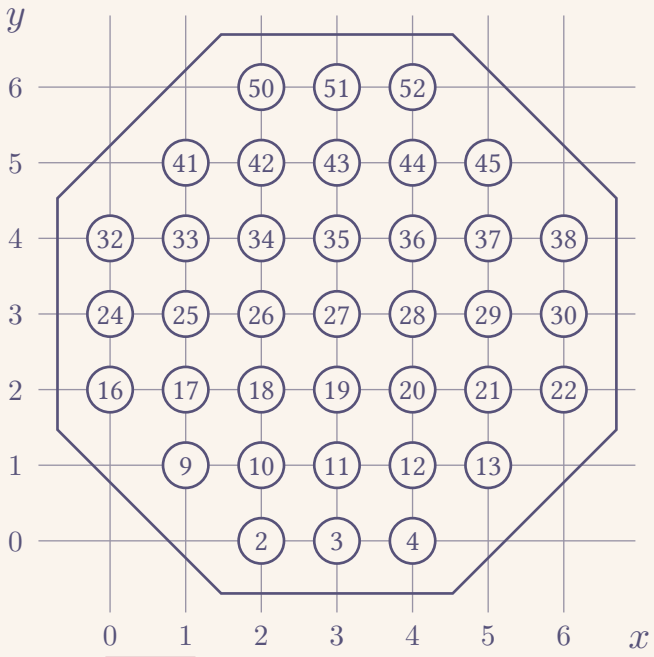


Fig. 4. Tablier européen numéroté

Indication OCaml. On peut retrouver les coordonnées (x, y) d'une encoche à partir de son numéro z en écrivant z `mod` 8 pour obtenir l'abscisse x et z `/` 8 pour obtenir l'ordonnée y. On peut calculer la valeur absolue d'un entier avec `abs`.

Q2. Écrire une fonction `numero_interieur (z: int): bool` qui teste si un entier `z` est le numéro d'une encoche du tablier européen (*i.e.*, est un élément de $\mathscr{E}$).

Q3. En énumérant les entiers naturels inférieurs à 52, et à l'aide de la fonction `numero_interieur` définie en Q2, définir une constante globale `numeros_europeens`, de type `int list`, égale à la liste des numéros d'encoches du tablier européen (*i.e.* une liste des éléments de $\mathscr{E}$).

Q4. En supposant qu'elles existent dans le tablier, donner, par une expression mathématique, les numéros des quatre encoches voisine de l'encoche de numéro z.

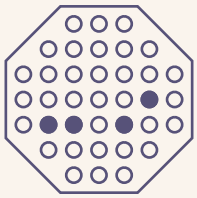
I.2. Motifs

Pour représenter un motif $M \subseteq \mathscr{E}$, nous utiliserons un entier naturel m dont nous exploitons une partie des bits de l'écriture binaire comme suit : pour tout numéro d'encoche $z \in \mathscr{E}$, nous fixons

$$[m]_z = \begin{cases} 1 & \text{si } z \in M \\ 0 & \text{si } z \notin M \text{ ou } z \notin \mathscr{E}. \end{cases}$$

Lorsqu'un motif est ponctuel, c'est à dire qu'il ne contient qu'une seule encoche, sa représentation est simplement une puissance de 2.

Par exemple, le motif à 4 encoches suivant



a pour représentation le nombre entier

$$m_0 = 2^{29} + 2^{20} + 2^{18} + 2^{17} = 538\,312\,704 \in \mathbb{N}.$$

Indication OCaml. Pour représenter les motifs et les motifs ponctuels, nous définissons les types `motif` et `ponctuel` par la déclaration suivante :

```
type motif = int
type ponctuel = motif
```

L'exemple ci-dessus se déclare à l'aide des opérateurs logiques en OCaml par :

```
let m0 = (1 lsl 29) lor (1 lsl 20) lor (1 lsl 18) lor (1 lsl 17)
```

Q5. Écrire une fonction `numero_vers_ponctuel (z: int): ponctuel` qui renvoie le motif ponctuel $\{z\} \subseteq \mathscr{E}$ formé d'une seule encoche de numéro z.

Q6. Écrire une fonction `numeros_vers_motif (l: int list): motif` qui revoie le motif formé des encoches de numéro appartenant à la liste `l`.

Q7. En s'appuyant sur la constante `numeros_europeens` introduite en Q3, définir une constante globale `motif_europeen`, de type `motif`, égale au motif formé de toutes les encoches du tablier européen.

Q8. Démontrer que la fonction `est_ponctuel` définie par :

```
let est_ponctuel (m: motif): bool =
  m > 0 && (m land (m-1)) = 0
```

renvoie `true` si le motif `m` est un motif ponctuel, et `false` sinon.

Q9. Écrire, à l'aide des opérateurs logiques sur les entiers, une fonction ayant pour signature `inclus (m: motif) (p: ponctuel): bool` qui renvoie `true` si le motif ponctuel `p` est inclus dans le motif `m`, ou `false` dans le cas opposé. Donner une preuve mathématique du résultat.

Q10. Écrire, à l'aide des opérateurs logiques sur les entiers, une fonction ayant pour signature `voisin_g (p: ponctuel): ponctuel` qui calcule la translation par une encoche vers la gauche du motif ponctuel `p`. Si le motif ponctuel `p` sort du tablier, la valeur de retour est le motif vide, 0.

Dans la suite du problème, nous supposerons avoir codé des fonctions similaires `voisin_d`, `voisin_h`, `voisin_b` qui calculent les décalages d'un motif ponctuel respectivement vers la droite, vers le haut, et vers le bas.[2] Nous déclarons également une constante globale :

```
let voisins = [voisin_g; voisin_d; voisin_h; voisin_b]
```

I.3. Coups simples et composés

On définit le type `coup` comme ci-dessous :

```
type coup = motif * ponctuel
```

Q11. Écrire une fonction `coup_simple ((m,p): coup): coup list` qui prend en entrée un motif quelconque `m` et un motif ponctuel `p` contenu dans `m`, et qui construit la liste des couples `(m', p')`, où `m'` est le motif obtenu par un coup simple de la bille initialement placée dans `p`, et où `p'` est le motif ponctuel repérant la même bille après son déplacement.

Q12. Écrire une fonction `coup_compose ((m,p): coup): coup list` qui prend en entrée un motif quelconque `m`, et un motif ponctuel `p` contenu dans `m`, et qui construit la liste des couples `(m', p')`, où `m'` est le motif obtenu à partir de `m` à la suite du déplacement par un coup composé de la bille initialement placée dans `p`, et où `p'` est le motif ponctuel repérant la même bille après son déplacement.

Q13. Écrire une fonction `mouvements (m: motif): motif list` dont la valeur de retour est la liste de tous les motifs que l'on peut obtenir en jouant un coup composé depuis le motif `m`, n'importe quelle bille pouvant être déplacée.

I.4. Partie de longueur minimale

[2] Cette supposition est réalisée pour éviter de devoir écrire quatre fonctions très similaires. Ceci est un problème en écris de concours, on n'a pas de copié-collé. Mais, en TP, nous avons cette technologie, vous devrez donc également les coder.

Indication OCaml. Dans cette sous-section, nous utiliserons une structure de donnée de dictionnaire, avec modification en place, permettant d'associer des clés de type `motif` et des valeurs de type `int`. Nous notons le type de cette structure `dico`. Pour utiliser ces dictionnaires, nous disposons des fonctions suivantes :

- la fonction `create`, de type `unit -> dico`, qui permet de créer un dictionnaire vide ;
- la fonction `add`, de type `dico -> motif -> int -> unit`, telle que `add d m n` permette d'ajouter au dictionnaire `d` une association entre la clé `m` et l'entier `n` (si une association de clé `m` existe déjà, elle est remplacée) ;
- la fonction `mem`, de type `dico -> motif -> bool`, telle que `mem d m` renvoie le booléen `true` si la clé `m` est membre du dictionnaire `d`, et `false` sinon ;
- la fonction `find`, de type `dico -> motif -> int`, telle que `find d m` renvoie la valeur associée à la clé `m` si la clé est membre du dictionnaire `d`, et une erreur sinon.

En pratique, on utilisera le module OCaml `Hashtbl`. Les fonctions sont nommées `Hashtbl.` suivi du nom de la fonction : par exemple, `add` devient `Hashtbl.add`. Le type `dico` est défini comme ci-dessous.

```
type dico = (motif, int) Hashtbl.t
```

Q14. Écrire une fonction `add_and_mem (d: dico) (s: int) (m: motif): bool` qui ajoute l'association entre le motif `m` et l'entier `s` dans le dictionnaire `d` si le motif `m` est initialement absent du dictionnaire `d`. La valeur de retour de la fonction est `true` si le dictionnaire a été modifié, et `false` sinon.

Q15. Écrire une fonction `strate (d: dico) (s: int) (l: motif list): motif list` qui, pour tout motif `m` que l'on peut obtenir après un coup composé à partir d'un motif de la liste `l`, ajoute l'association entre le motif `m` et l'entier `s` au dictionnaire `d`, si le motif `m` n'est pas présent dans le dictionnaire `d`. La valeur de retour de la fonction est la liste des motifs que la fonction ajoute.

Q16. Dans cette question, nous supposerons qu'il est possible de passer d'un motif initial m_{i} à un motif final m_{f} par une suite de coups. Écrire une fonction ayant pour signature `partie_minimale (mi: motif) (mf: motif): int` qui calcule le nombre minimal de coups composés à jouer pour passer du motif initial `mi` au motif final `mf`. On pourra utiliser un dictionnaire qui associe des motifs au nombre de coups minimal pour les atteindre depuis le motif `mi`.

Q17. Peut-on considérer que la réponse donner en Q16 observe le parcours nommé en Q1 ? Introduire un graphe et argumenter.

Fin de la partie 1.

— Compétences —

- [ ] Connaitre les points positifs des différents parcours
- [ ] Connaitre les méthodes utilisée pour la réalisation de différents parcours
- [ ] Maitriser les manipulations « bit-à-bit » en OCaml (`lor`, `land`, `lsl`, `lsr` et `lnot`)
- [ ] Maitriser les listes et le module `List` (`List.map`, `List.fold_left`, `List.iter`, `List.filter`, `List.init`, et `List.flatten`)
- [ ] Utiliser des modules externes (`Hashtbl`), et les fonctions mises à disposition
- [ ] Maitriser les concepts de récursivité