▶ **MPSI – Option Informatique**
Année 2002, Quatrième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Le jeu de Marienbad

Dans le film d'Alain Resnais « L'année dernière à Marienbad » *(1961), l'un des personnages, M, propose à ses interlocuteurs, au fil d'énigmatiques déambulations dans un château, de jouer avec lui à un jeu de société. Il dispose quelques jetons sur une table. La règle est simple et aucune habileté ne semble nécessaire pour triompher. Mais la partie tourne systématiquement à l'avantage de M.* « Je puis perdre, mais je gagne toujours... » *assure-t-il à son protagoniste inexorablement vaincu.*

Les deux premières sections du sujet sont prévues pour être traitées pendant la séance de travaux pratiques. La troisième partie est à rendre pour le samedi 4 mai (groupes A1, A2, B1 et B2), même si vous n'avez pas TP ce jour là.

1 Écriture binaire d'un entier

Vous savez que tout entier naturel n admet une écriture binaire définie par

$$n = \sum_{j=0}^{k-1} a_j 2^j = \overline{a_{k-1} \dots a_1 a_0}$$

Chaque a_j est un *booléen* appartenant à l'ensemble $\{0, 1\}$. Cette écriture est unique si on impose $a_{k-1} = 1$. Nous représenterons en *Caml* l'écriture $\overline{a_{k-1} \dots a_1 a_0}$ par une liste de type *bool list* : $[a_0; a_1; \dots; a_{n-1}]$ (notez que l'ordre des éléments a été inversé).

▶ **Question 1** *Écrivez un fonction* int_of_bits *qui prend pour argument une liste de booléens m. Cette fonction calculera l'entier n dont m est une écriture binaire.*

value int_of_bits : *bool list* → *int*

On peut calculer l'écriture binaire m d'un entier n de manière récursive :
– Si $n = 0$ alors $m = [\,]$.
– Si $n \geq 1$, on effectue la division euclidienne de n par 2. On obtient deux entiers naturels q et r tels que $n = 2q + r$ (avec $r \in \{0, 1\}$). Si m' est la représentation de q en base 2 (obtenue par récursion) alors $r :: m'$ est la représentation de n.

▶ **Question 2** *Implémentez maintenant cet algorithme pour obtenir une fonction* bits_of_int *donnant la représentation binaire d'un entier.*

value bits_of_int : *int* → *bool list*

Le « *ou exclusif* » est un opérateur binaire $\oplus$ défini sur les booléens par la table de vérité suivante :

$\oplus$	0	1
0	0	1
1	1	0

On étend généralement cette opération aux entiers naturels : si m et n admettent respectivement $\overline{a_{k-1} \dots a_0}$ et $\overline{b_{k-1} \dots b_0}$ comme représentations binaires, on pose pose $n \oplus m = \overline{c_{k-1} \dots c_0}$ avec $c_j = a_j \oplus b_j$ pour tout $j \in \{0, \dots, k-1\}$. On dit généralement qu'on effectue l'opération « *bit à bit* ».

▶ **Question 3** *Écrivez une fonction* xor *implémentant l'opération* $\oplus$ *sur les booléens. Déduisez-en une fonction* xor_int *pour les entiers.*

value xor : *bool* → *bool* → *bool*
value xor_int : *int* → *int* → *int*

2 Le jeu

Le jeu de Marienbad oppose deux candidats. On dispose sur une table un certain nombre de jetons répartis en quelques tas de manière « aléatoire ». Chaque joueur retire à son tour autant de jetons qu'il souhaite, mais au moins un et tous du même tas. La partie continue jusqu'à ce qu'il n'y ait plus de jetons sur la table. Le gagnant est le joueur qui retire le dernier jeton du plateau de jeu.
Nous notons par la suite n le nombre de tas. Nous représenterons ainsi une *configuration* du jeu par un tableau d'entiers c comportant n cases qui indiquent le nombre de jetons de chaque tas.

▶ **Question 4** *Écrivez une fonction* create *qui crée un plan de jeu. Elle prendra deux entiers comme argument : n, le nombre de tas et m le nombre maximal de jetons à placer initialement dans chaque tas. La fonction retournera un tableau représentant le plan de jeu dans lequel un nombre « aléatoire » de jetons (compris entre* 0 *et m) aura été disposé dans chaque tas.*

value create : *int* → *int* → *int vect*

▶ **Question 5** *Écrivez une fonction* empty *qui teste si une configuration est vide (i.e. si toutes les cases du tableau c contiennent* 0*).*

value empty : *int vect* → *bool*

Nous allons maintenant écrire quelques fonctions permettant d'organiser une partie entre deux joueurs humains. Pour cela, nous aurons besoin de quelques fonctions de la bibliothèque standard permettant de lire des entrées saisies par les joueurs ou d'afficher des résultats. Ces fonctions sont rappelées dans la table suivante.

print_int	*int* → *unit* Imprime un entier
print_string	*string* → *unit* Imprime une chaîne de caractères
print_newline	*unit* → *unit* Imprime un retour à la ligne
read_line	*unit* → *string* Lit une chaîne saisie par l'utilisateur
int_of_string	*string* → *int* Convertit une chaîne en entier (lève l'exception Failure " int_of_string " si la chaîne ne représente pas un entier)

▶ **Question 6** *Écrivez une fonction* print *qui imprime une configuration sous une forme telle que*

0:4 1:3 2:0 3:4 4:6 5:1 6:1

value print : *int vect* → *unit*

▶ **Question 7** *Programmez ensuite une fonction* human_turn *prenant une configuration c (supposée non vide) en argument et qui :*

1. *Affiche le plan de jeu,*
2. *Demande au joueur à quel tas il souhaite supprimer des jetons, puis combien il souhaite en supprimer,*
3. *Vérifie que l'entrée donnée par le joueur est correcte (et, dans le cas contraire, demande au joueur de recommencer sa saisie),*
4. *Modifie le tableau c* en place *pour enregistrer le retrait de jetons,*
5. *Renvoie un booléen indiquant si le joueur a gagné.*

value human_turn : *int vect* → *bool*

▶ **Question 8** *Écrivez enfin une fonction* human_play *qui permette de jouer une partie à deux. Cette fonction prendra les mêmes arguments que la fonction* create.

value human_play : *int* → *int* → *unit*

3 Une stratégie gagnante

Le jeu de Marienbad possède une caractéristique remarquable : on connaît une stratégie gagnante, c'est-à-dire une manière de jouer qui conduit celui qui la suit de manière assurée à la victoire. Dans cette section, nous allons étudier cette stratégie puis écrire quelques fonctions en *Caml* permettant de faire jouer votre ordinateur en suivant cette stratégie.

3.1 Quelques propriétés de l'opération $\oplus$

Avant de présenter cette stratégie, nous avons besoin d'établir quelques propriétés de l'opération $\oplus$ introduite dans la première partie.

▶ **Question 9 (⋆)** *Vérifiez que l'opérateur* $\oplus$ *définit une loi de groupe commutatif sur les entiers. Quel est l'élément neutre ? Quel est l'inverse pour cette loi d'un entier n ?*

Rappel : $\oplus$ est une loi de groupe commutatif sur $\mathbb{N}$ si et seulement si les propriétés suivantes sont vérifiées :

1. $\oplus$ est commutative (i.e. $\forall\, x, y \in \mathbb{N},\ x \oplus y = y \oplus x$).
2. $\oplus$ est associative (i.e. $\forall\, x, y, z \in \mathbb{N},\ x \oplus (y \oplus z) = (x \oplus y) \oplus z$).
3. Il existe $e \in \mathbb{N}$ (appelé élément neutre) tel que pour tout $x \in \mathbb{N}$ on ait $e \oplus x = x$.
4. Pour tout $x \in \mathbb{N}$, il existe $x' \in \mathbb{N}$ (appelé inverse de x) tel que $x \oplus x' = e$.

▶ **Question 10 (⋆)** *Considérons* k *entiers naturels* $c_0, \ldots, c_{k-1}$. *Soit* $b = c_0 \oplus \cdots \oplus c_{k-1}$. *Démontrez qu'il existe* $i \in \{0, \ldots, k-1\}$ *tel que* $c_i \oplus b < c_i$.

3.2 Configurations paires

Notre stratégie de jeu repose sur la notion de *configuration paire*. Une configuration $c = [|c_0; c_1; \ldots; c_{n-1}|]$ est paire est *paire* si et seulement si la quantité $c_0 \oplus c_1 \oplus \cdots \oplus c_{n-1}$ est nulle. (Si une configuration n'est pas paire, on dit qu'elle est *impaire*.)

▶ **Question 11** *Les configurations* $[|11; 10; 10; 9; 7; 6; 10|]$ *et* $[|8; 10; 8; 9; 16; 8; 4|]$ *sont-elles paires ?*

▶ **Question 12** *Écrivez une fonction* even *qui teste si une configuration de jeu est paire.*

value even : *int vect* → *bool*

▶ **Question 13 (⋆)** *Démontrez que si, au début d'un tour de jeu la configuration est paire, alors, à la fin du tour (i.e. lorsque le joueur a retiré ses jetons), la configuration est impaire.*

3.3 La stratégie

Nous nous intéressons maintenant à la question suivante : si au début d'un tour, la configuration $c = [|c_0; \ldots; c_{n-1}|]$ du jeu est impaire, est-il toujours possible de jouer de manière à obtenir une configuration $c' = [|c'_0; \ldots; c'_{n-1}|]$ qui soit paire ?
Posons $b = c_0 \oplus \cdots \oplus c_{n-1}$. En utilisant le résultat de la question 10, on sait qu'il existe i tel que $c_i \oplus b < c_i$. Choisissons de retirer $c_i - (c_i \oplus b)$ jetons du i^e tas. On a alors $c'_i = c_i \oplus b$ et $c'_j = c_j$ pour $j \neq i$.

▶ **Question 14** *En utilisant le fait que* $\oplus$ *est une loi de groupe commutatif, démontrez que* $c'_1 \oplus \cdots \oplus c'_{n-1} = 0$*, i.e. que* c' *est paire.*

Grâce à ce résultat, nous obtenons une manière de jouer un tour qui permet à coup sûr de rendre paire une configuration impaire $c = [|c_0; \ldots; c_{n-1}|]$:

1. Calculer $b = c_0 \oplus \cdots \oplus c_{n-1}$.
2. Rechercher $i \in \{0, \ldots, n-1\}$ tel que $c_i \oplus b < c_i$.
3. Retirer $c_i - (c_i \oplus b)$ du i^e tas.

▶ **Question 15** *Écrivez une fonction* odd_to_even *implémentant cette méthode de jeu. La fonction prendra en argument une configuration c supposée impaire. Elle modifiera* en place *le tableau c de manière à effectuer un tour de jeu et rendre la configuration paire. Elle imprimera le nombre de jetons retirés et le numéro du tas auquel elles appartenaient.*

value odd_to_even : *int vect* → *unit*

▶ **Question 16 (⋆)** *En remarquant qu'une configuration où tous les tas sauf un sont vides n'est pas paire, expliquez pourquoi quelqu'un qui agit toujours de telle sorte qu'après son tour de jeu la configuration soit paire est certain de gagner.*

3.4 Caml joue

Nous allons maintenant écrire quelques fonctions permettant à *Caml* de jouer au jeu de Marienbad comme M, c'est-à-dire en appliquant s'il le peut cette stratégie. À chacun de ses tours, *Caml* procédera de la manière suivante :

- Si la configuration au début du tour est paire, il appellera la fonction play_even de façon à jouer son tour en rendant la configuration paire.
- Si la configuration au début du tour est impaire, il lui est impossible de suivre la stratégie gagnante. Il retirera alors un jeton d'un tas (non vide) choisi aléatoirement.

▶ **Question 17** *Écrivez une fonction* caml_turn *qui fait jouer à Caml un tour de jeu en appliquant la méthode que nous venons de décrire.*

value caml_turn : *int vect* → *unit*

▶ **Question 18** *Déduisez-en une fonction* caml_play *qui permette à un joueur humain de jouer au jeu de Nim face à Caml. Cette fonction prendra les mêmes arguments que la fonction* create*. Le joueur humain commencera la partie.*

value caml_play : *int* → *int* → *unit*

3.5 Qui gagne ?

Évidemment, si les deux joueurs d'une partie connaissent la stratégie gagnante, il ne leur est pas possible à tous les deux de la respecter : pour appliquer cette stratégie, il est nécessaire de disposer initialement d'une configuration impaire.

▶ **Question 19** *Comment déterminer à partir de la configuration initiale lequel de deux joueurs essayant de suivre la stratégie gagnante peut gagner une partie ?*

Pour terminer, on se propose de calculer la probabilité qu'une configuration initiale soit paire. On suppose pour cela fixés n (le nombre de tas) et $m = 2^k - 1$ (le nombre maximal de jetons par tas). On suppose équiprobables toutes les configurations à n tas comportant chacun entre 0 et m jetons.

▶ **Question 20** *Calculez le nombre de configurations possibles pour des entiers* n *et* $m = 2^k - 1$ *fixés. En utilisant la représentation binaire sur* k *bits des entiers du tableau c, déterminez ensuite le nombre de ces configurations qui sont paires. Déduisez-en la probabilité que la configuration initiale soit paire.*

Épilogue

Nous savons gagner à coup sûr au jeu de Marienbad. Plus personne ne voudra jouer avec nous... L'existence d'une stratégie gagnante n'est pas un cas isolé et peut être démontrée pour tout un éventail de jeux plus ou moins célèbres. Ce qui est remarquable ici, c'est que l'on *connaît* la stratégie et que l'on sait l'appliquer de manière algorithmique.
Dans beaucoup de cas, l'existence d'une stratégie gagnante est établie de manière *non constructive* : son existence est démontrée sans pour autant que l'on sache en quoi elle consiste. Un théorème appelé théorème de *Sprague et Grundy* permet de ramener un certain nombre de jeux dits impartiaux au jeu de Marienbad et de trouver ainsi une stratégie gagnante. En voici quelques exemples :

- *Partage d'une tablette de chocolat.* A tour de rôle chaque joueur coupe la tablette en deux (sans couper au milieu des carrés...) et écarte l'une des deux parties. Le jeu se termine lorsqu'il ne reste plus qu'un carré et c'est le dernier à avoir découpé la tablette qui gagne.
- *Dominos.* Les deux joueurs placent à tour de rôle des dominos sur un échiquier. Le dernier joueur à placer un domino gagne.
- *Jeu de Grundy.* On joue avec des piles d'allumettes. Le seul coup possible consiste à diviser une pile en deux parties inégales. Le jeu se termine quand toutes les piles contiennent une ou deux allumettes. Le joueur jouant en dernier gagne.

▶ **MPSI – Option Informatique**
Année 2002, Quatrième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Le jeu de Marienbad

Un corrigé

▶ Question 1

```
let rec int_of_bits = function
    [] -> 0
  | true :: q -> 1 + 2 * (int_of_bits q)
  | false :: q -> 2 * (int_of_bits q)
;;
```

▶ Question 2

```
let rec bits_of_int = function
    0 -> []
  | n ->
      let q = n / 2
      and r = n mod 2 in
      (r = 1) :: (bits_of_int q)
;;
```

▶ Question 3

```
let xor = fun
    true true -> false
  | false false -> false
  | true false -> true
  | false true -> true
;;

let xor_int x y =
  let rec xor_list = fun
      m [] -> m
    | [] m -> m
    | (t1 :: q1) (t2 :: q2) ->
        (xor t1 t2) :: (xor_list q1 q2)
  in
  int_of_bits
    (xor_list (bits_of_int x) (bits_of_int y))
;;
```

▶ Question 4

```
let create n m =
  let c = make_vect n 0 in
  for i = 0 to n - 1 do
    c.(i) <- random__int (m + 1)
  done;
  c
;;
```

▶ Question 5

```
let empty c =
  let rec aux = function
      0 -> true
    | i -> c.(i - 1) = 0 && aux (i - 1)
  in
  aux (vect_length c)
;;
```

▶ Question 6

```
let print c =
  for i = 0 to vect_length c - 1 do
    print_int i;
    print_string ":";
    print_int c.(i);
    print_string " "
  done;
  print_newline ()
;;
```

▶ Question 7

```
let human_turn c =
  let n = vect_length c in

  let rec aux () =
    print c;
    try
      print_string "Tas : ";
      let i = int_of_string (read_line ()) in
      print_string "Jetons : ";
      let j = int_of_string (read_line ()) in

      if i < 0 or i >= n or j <= 0 or c.(i) < j
      then raise (Failure "");

      c.(i) <- c.(i) - j

    with Failure _ ->
      print_newline ();
      print_string "Saisie incorrecte";
      print_newline ();
      aux ()
  in

  aux ();
  empty c
  ;;
```

▶ Question 8

```
let human_play n m =
  let c = create n m in

  let rec loop j =
    print_newline ();
    print_string "C'est au joueur ";
    print_int j;
    print_newline ();

    if not (human_turn c) then loop (3 - j)
  in

  loop 1;

  print_string "Vous avez gagn^^e9 !";
  print_newline ()

;;
```

Le jeu de Marienbad

Suite et fin du corrigé

► **Question 9** Montrons tout d'abord que $\oplus$ définit une loi de groupe sur $\mathbb{B} = \{0, 1\}$. Soient $x, y, z \in \mathbb{B}$. $x \oplus y$ est non nul si et seulement si exactement un parmi x et y est non nul. On en déduit la commutativité de $\oplus$. De même $x \oplus (y \oplus z)$ est non nul si et seulement si exactement un ou trois parmi x, y et z sont non nuls. On déduit alors l'associativité de la commutativité. 0 est élément neutre et l'inverse d'un booléen est lui-même.
« Étendons » maintenant ce résultat à $\mathbb{N}$. Soient x et y deux entiers d'écriture binaire sur k bits $\overline{x_{k-1} \cdots x_0}$ et $\overline{y_{k-1} \cdots y_0}$. On a

$$\begin{aligned} x \oplus y &= \overline{x_{k-1} \cdots x_0} \oplus \overline{y_{k-1} \cdots y_0} \\ &= \overline{(x_{k-1} \oplus y_{k-1}) \cdots (x_0 \oplus y_0)} \\ &= \overline{(y_{k-1} \oplus y_{k-1}) \cdots (y_0 \oplus z_0)} \\ &= \overline{y_{k-1} \cdots y_0} \oplus \overline{x_{k-1} \cdots x_0} \\ &= y \oplus x \end{aligned}$$

On en déduit que $\oplus$ est commutative sur $\mathbb{N}$. On montre de même l'associativité. On vérifie également que pour tout entier x, $x \oplus 0 = 0$ et $x \oplus x = 0$, i.e. que 0 est élément neutre et que chaque entier est son propre inverse pour $\oplus$.

► **Question 10** Soit $b = \overline{b_{n-1} \cdots b_0}$ l'écriture de b sur n bits et $c_i = \overline{c_{i,n-1} \cdots c_{i,0}}$ (resp. $c'_i = \overline{c'_{i,n-1} \cdots c'_{i,0}}$) l'écriture de c_i (resp. $c'_i = c_i \oplus b$) sur n bits (pour $0 \leq i \leq k-1$).
Si $b = 0$ on a, pour tout i, $c_i \oplus b = c_i$. Supposons maintenant que $b \neq 0$. Soit $m = \max\{j \mid b_j = 1\}$. Il existe i tel que $c_{i,m} = 1$ (sinon on aurait $b_m = 0$). On a $c_{i,m} = 1$, $c'_{i,m} = 0$ et pour $j > m$, $c'_{i,j} = c_{i,j}$. On en déduit que $c'_i < c_i$, i.e. $c_i \oplus b < c_i$.

► **Question 11** Ces deux configurations comportent un nombre impair de tas contenant un nombre impair de jetons. C'est une condition suffisante pour qu'elles soient impaires.

► **Question 12** On définit tout d'abord une fonction xor_conf prenant en argument une configuration $c = [|c_0; \ldots; c_{n-1}|]$ et calculant $c_0 \oplus \cdots \oplus c_{n-1}$.

```
let xor_conf c =
  let n = vect_length c in
  let x = ref 0 in
  for i = 0 to n - 1 do
    x := xor_int c.(i) !x
  done;
  !x
;;
```

On en déduit la fonction even :

```
let even c =
  (xor_conf c) = 0
;;
```

► **Question 13** Soit $c = [|c_0; c_1; \ldots; c_{n-1}|]$ (resp. $c' = [|c'_0; c'_1; \ldots; c'_{n-1}|]$ la configuration initiale (resp. finale). Il existe $0 \leq j \leq n-1$ tel que $c'_j < c_j$ et pour tout $i \neq j$, $c_i = c'_j$. La configuration c est paire donc $c_0 \oplus \cdots \oplus c_{n-1} = 0$. On a alors

$$\begin{aligned} c'_0 \oplus \cdots \oplus c'_{n-1} &= (c_0 \oplus \cdots \oplus c_{n-1}) \oplus c_j \oplus c'_j \\ &= c_j \oplus c'_j \end{aligned}$$

Or $c'_j \neq c_j$ donc $c_j \oplus c'_j \neq 0$. On en déduit que c' est une configuration impaire.

► **Question 14** Soit $b' = c'_1 \oplus \cdots \oplus c'_{n-1}$. On a

$$\begin{aligned} b' &= c'_1 \oplus \cdots \oplus c'_{i-1} \oplus c'_i \oplus c'_{i+1} \oplus \cdots \oplus c'_{n-1} \\ &= c_1 \oplus \cdots \oplus c'_{i-1} \oplus (c_i \oplus b) \oplus c_{i+1} \oplus c_{n-1} \\ &= (c_1 \oplus \cdots \oplus c_{n-1}) \oplus b \\ &= b \oplus b = 0 \end{aligned}$$

On en déduit que la configuration obtenue est paire.

► **Question 15**

```
let odd_to_even c =
  let b = xor_conf c in
  let i = ref 0 in
  while xor_int b c.(!i) >= c.(!i) do
    i := !i + 1
  done;
  let c'_i = xor_int b c.(!i) in
  print_int (c.(!i) - c'_i);
  print_string " jeton(s) retire(s) du tas ";
  print_int !i;
  print_newline ();
  c.(!i) <- c'_i
;;
```

► **Question 16** On remarque tout d'abord que :
- Une configuration dont tous les tas sont vides sauf exactement un est impaire. Il est donc nécessaire de disposer d'une configuration impaire à un moment du jeu pour pouvoir gagner.
- Le nombre de jetons disponible décroit strictement à chaque tour de jeu. On en déduit que toutes les parties finissent et qu'il y a systématiquement un gagnant.

Si un joueur agit de telle sorte qu'après un tour de jeu la configuration soit paire, son adversaire lui rendra une configuration impaire (question 13) qu'il pourra à nouveau rendre paire (questions 10 et 14). On vérifie alors par itération qu'il pourra toujours laisser une configuration paire à son adversaire. D'après notre première remarque, on en déduit que ce dernier ne pourra gagner. Grâce à notre seconde remarque, on conclut que le premier joueur l'emportera nécessairement.

► **Question 17** Nous écrivons tout d'abord une fonction implantant un tour de jeu "aléatoire".

```
let rec random_turn c =
  let i = random__int (vect_length c) in
  if c.(i) = 0 then random_turn c else begin
    c.(i) <- c.(i) - 1;
    print_string "1 jeton(s) retire(s) du tas ";
    print_int i;
    print_newline ()
  end
;;
```

On en déduit la fonction caml_turn :

```
let caml_turn c =
  if even c then random_turn c
  else odd_to_even c
;;
```

▶ Question 18

```
let caml_play n m=
  let c = create n m in
  let rec loop = function
    true ->
      if empty c then print_string "Vous avez perdu"
      else if human_turn c
           then print_string "Vous avez gagne"
           else loop false
  | false ->
      caml_turn c;
      loop true
  in
  loop true;
  print_newline ()
;;
```

▶ Question 19 Pour qu'un joueur puisse appliquer la stratégie gagnante, il est nécessaire et suffisant qu'il dispose à un moment du jeu d'une configuration impaire avant de jouer son tour. Si la configuration initiale est impaire et le premier joueur connaît la stratégie gagnante, il pourra l'appliquer et remporter la partie. Si la configuration initiale est paire, le premier joueur va la rendre impaire lors de son premier tour (question 13). Le second joueur va ainsi pouvoir appliquer la stratégie gagnante — s'il la connaît — jusqu'à la victoire.

▶ Question 21 *Une configuration à n tas est définie par la donnée d'un tableau de n entiers compris entre 0 et m. On en déduit qu'il existe $(m+1)^n$ configurations initiales possibles.*
Supposons $n \geq 2$. Nous allons montrer qu'il existe exactement autant de configurations paires à n tas que de configurations à $n-1$ tas. Définissons pour cela une application f par

$$f : [|c_0, \dots c_{n-2}|] \mapsto [|c_0, \dots c_{n-2}, c_0 \oplus \cdots \oplus c_{n-2}|]$$

On vérifie facilement que f est une bijection de l'ensemble des configurations à $n-1$ tas dans l'ensemble des configurations paires à n tas. Soient c est c' deux configurations à $n-1$ tas. Si $f(c) = f(c')$ alors pour tout $0 \leq i \leq n-2$ on a $c_i = c'_i$. On en déduit que $c = c'$. f est donc injective. Soit $c = [|c_0, \dots, c_{n-2}, c_{n-1}|]$ une configuration paire à n tas. On a $c_0 \oplus \cdots \oplus c_{n-2} \oplus c_{n-1} = 0$ donc $c_0 \oplus \cdots \oplus c_{n-2} = c_{n-1}$. On en déduit que $c = f([|c_0, \dots, c_{n-2}|])$. f est donc surjective.

Il existe ainsi $(m+1)^{n-1}$ configurations paires à n tas (ce résultat s'étend au cas $n = 1$ en remarquant qu'une configuration à un tas est paire si ce dernier est vide). La probabilité que la configuration initiale soit paire est donc $1/(m+1)$. Elle ne dépend pas du nombre de tas.