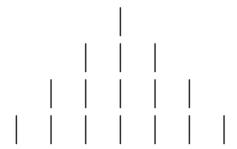
Université Paris Diderot - M1 Informatique

TD de *Programmation logique et par contraintes* n° 5

Le jeu de Marienbad

Ce jeu appartient à la famille des jeux de Nim, qui sont des jeux de duel à information parfaite et somme nulle (deux joueurs, pas de match nul). Cette variante a été rendue célèbre par le film d'Alain Resnais *L'année dernière à Marienbad*. Configuration initiale :



Règle du jeu : les deux joueurs, chacun à son tour, prélèvent une quantité (non nulle) de pions d'une (et une seule) rangée. Fin du jeu : le joueur qui ne peut plus jouer (car il n'y a plus de pions) perd.

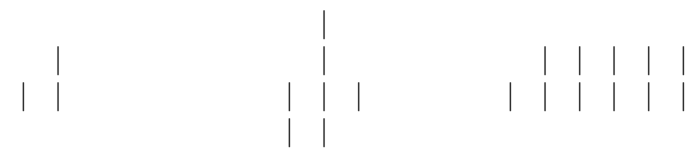
Exercice 1 Choisir une représentation des configurations du jeu, et définir un prédicat

```
move(+Config_Source,-Config_Destination)
```

qui implémente la règle du jeu.

Le prédicat `move` doit produire, pour une configuration donnée, toutes les configurations atteignables par un coup à partir de celle la.

Une configuration `c` est gagnante si à partir `c` il existe une stratgie qui permet au joueur de gagner contre toute defense de l'opposant. Voici par exemple 3 configurations gagnantes (pour chacune d'entre elles, vous pourrez trouver la strategie qui permet de gagner) :



Exercice 2 Définir un prédicat `gagne(+Configuration)`, tel que le but `gagne(c).` réussit si et seulement s'il existe une stratégie gagnante à partir de la configuration `c`, en utilisant "force brute" (jouer et tester que la configuration atteinte n'est pas gagnante).

Exercice 3 Il est possible de tester si un configuration est gagnante sans visiter l'arbre de jeu, comme décrit ci-dessous (extrait de l'examen partiel de 2013).

1. Écrire un prédicat `dec2bin_inv(+n,-res)` dont le premier argument est un entier n (qu'on suppose non négatif) et qui calcule la représentation de n en binaire **inversée** (c.à.d. chiffre moins significatif en tête), sous forme de liste de bits. Par exemple :

   ```
   ?- dec2bin_inv(32,L).
   L = [0,0,0,0,0,1]
   yes
   ```

Rappel : si `N` est un entier, `N mod 2` est le reste et `N//2` le quotient de la division entière de `N` par 2.

2. Écrire un prédicat `somme_xor(+l1,+l2,-res)` qui prends deux listes de bits (non nécessairement de même longueur) comme premier et deuxième arguments, et calcule la liste obtenue en appliquant la fonction *xor* ("ou exclusif") chiffre par chiffre aux deux arguments. Rappel : a *xor* $b = 1$ si et seulement si $a \neq b$. Par exemple :

```
| ?- dec2bin_inv(16,X),dec2bin_inv(32,Y),somme_xor(X,Y,Z).
X = [0,0,0,0,1]
Y = [0,0,0,0,0,1]
Z = [0,0,0,0,1,1] ?
yes
```

3. Écrire un prédicat `somme_xor_iteree(+l,-res)` qui prends une liste de listes de bits comme premier argument, et calcule la `somme_xor` de ces listes. Par exemple :

```
| ?- dec2bin_inv(16,W),dec2bin_inv(48,X),dec2bin_inv(19,Y),
somme_xor_iteree([W,X,Y],Z).
W = [0,0,0,0,1]
X = [0,0,0,0,1,1]
Y = [1,1,0,0,1]
Z = [1,1,0,0,1,1] ?
yes
```

4. Écrire un prédicat `bin_inv2dec(+l,-res)` qui calcule l'inverse de `dec2bin_inv`.
   Par exemple :

```
| ?- bin_inv2dec([0,0,0,0,1],R).
R = 16
yes
```

5. La *somme nim* d'une liste d'entiers est l'entier correspondant à la somme xor itérée des représentations binaires des entiers de la liste. Par exemple, la somme nim de $1,2,4,8$ est 15, celle de $2,3,8$ est 9 et celle de $5,3,6$ est 0. Écrire un prédicat `somme_nim(+l,-res)` qui prend une liste `l` d'entiers non négatifs et calcule leur somme nim, de la manière suivante :
   — on construit la liste des représentations binaires inversées des éléments de `l`, en mappant sur `l` le prédicat `dec2bin_inv`.
   — on applique le prédicat `somme_xor_iteree` à la liste ainsi obtenue.
   — on converti le résultat en utilisant `bin_inv2dec`

6. Un théorème dû au mathématicien americain Charles Bouton dit qu'une position $n_1,...,n_k$ du jeu de Mariembad est gagnante si et seulement si la somme nim de $n_1,...,n_k$ est strictement positive.
   Vérifier en utilisant le théorème de Bouton que $1,3,5,7$ est perdante.
   Écrire un prédicat `gagne_bis(+l)` qui prends une liste d'entiers non négatifs représentant une position du jeu de Nim et réussit si cette position est gagnante, en utilisant le théorème de Bouton. Comparer ce prédicat au prédicat `gagne` de l'exercice 2.

Exercice 4 Définir un prédicat `jeu(+Configuration)` qui permet à l'utilisateur de jouer contre le programme, comme premier joueur et à partir de la configuration donnée. Le programme joue une stratégie optimale (c'est à dir que, dès que le programme joue à partir d'une configuration gagnante, il gagne).