Le jeu de Nim

Tahina Ramananandro
ramanana@clipper.ens.fr

Lycée Louis-le-Grand – MPSI CAML
18 juin 2008

Pour cette dernière séance, on va étudier quelques jeux à deux joueurs fondés sur le jeu de Nim[1] pour lequel il existe une *stratégie gagnante* simple.

1 Le Nim proprement dit

On dispose des allumettes en plusieurs rangées. Chaque joueur à tour de rôle enlève un nombre quelconque (mais non nul...) d'allumettes dans une *seule* rangée de son choix. Celui qui prend la dernière allumette a *gagné*.

1.1 Implémentation d'une interface

Un jeu est représenté par un tableau d'entiers, chaque case du tableau indiquant le nombre d'allumettes présentes dans la rangée qu'elle représente.

Tout d'abord, écrire une fonction :

```
creer : int -> int vect
```

tel que `creer` *allumettes* crée un tableau de jeu aléatoire comportant exactement le nombre d'allumettes spécifié.

En utilisant la fonction prédéfinie :

```
read_int : unit -> int
```

qui lit un entier à partir de l'entrée de l'utilisateur, écrire une fonction :

```
coup_humain : int vect -> unit
```

qui affiche la situation en cours (le nombre d'allumettes dans chaque rangée) et demande à l'utilisateur d'entrer un coup (numéro de rangée et nombre d'allumettes à y enlever), et l'exécute en modifiant le tableau passé en entrée. Si le coup est non valide, ne pas toucher au tableau, mais afficher un message d'erreur.

Puis, écrire une fonction :

```
jeu_humain : unit -> unit
```

qui demande à l'utilisateur de taper un nombre d'allumettes, puis fait jouer deux êtres humains à tour de rôle, tapant sur le même clavier.

1.2 Stratégie gagnante

1.2.1 XOR bit à bit

Étant donnés deux booléens b et b', on note $b \oplus b'$ le XOR de ces deux booléens. On définit alors l'opération « XOR bit à bit » sur les entiers naturels :

$$\left(\sum_i 2^i b_i\right) \oplus \left(\sum_i 2^i b'_i\right) \equiv \sum_i 2^i \left(b_i \oplus b'_i\right)$$

Montrer que cette loi est associative, commutative, admet le neutre 0, et que tout entier naturel est son propre inverse.

Écrire d'abord deux fonctions récursives :

[1] D'après Vincent Simonet.

`bits_of_int : int -> bool list`

`int_of_bits : bool list -> int`

qui permettent de passer d'un entier à sa représentation binaire, le premier bit (coefficient de 2^0) étant le premier élément de la liste.

Puis écrire une fonction :

`xor_list : bool list -> bool list -> bool list`

qui exécute le XOR sur chacun des éléments de chaque liste. Les deux listes n'ayant pas nécessairement la même longueur, les booléens « absents » seront considérés comme faux.

Combiner toutes ces fonctions pour obtenir le XOR bit à bit :

`xor_int : int -> int -> int`

1.2.2 Configurations perdantes

Notons (r_i) la situation de jeu en cours : pour chaque rangée numéro i, r_i est le nombre d'allumettes qu'elle contient.

Montrer que si $\bigoplus_i r_i = 0$, et s'il y a au moins une allumette, alors *quel que soit* le coup joué par le joueur, on obtiendra une nouvelle configuration (r'_i) telle que $\bigoplus_i r'_i \neq 0$.

À l'inverse, montrer que si $\bigoplus_i r_i \neq 0$, alors *il existe* toujours un coup pour le joueur, tel que l'on obtienne une nouvelle configuration (r'_i) telle que $\bigoplus_i r'_i = 0$. Ce coup sera noté $CG((r_i))$. *Indication :* on pourra montrer qu'il existe un indice j tel que $r_j \oplus \bigoplus_i r_i < r_j$.

En déduire que $\bigoplus_i r_i = 0$ est une condition nécessaire et suffisante pour que celui qui doit jouer ait « forcément » perdu la partie (i.e. pour tout coup joué, son adversaire ait toujours une riposte qui lui assure la victoire).

Implémenter cette stratégie, sous la forme de deux fonctions :

`coup_ordi : int vect -> unit` qui étant donné une situation (r_i), joue (et affiche) le coup $CG((r_i))$ s'il existe ;

`jeu_ordi : unit -> unit` qui permet à un être humain de jouer contre l'ordinateur.

2 Variantes et jeux dérivés

2.1 Les tablettes de chocolat

2.1.1 Version à un coup

On dispose d'une tablette de chocolat organisée en carrés. Chaque joueur, à tour de rôle, casse la tablette suivant une des lignes séparant des rangées ou colonnes de carrés, et mange un des deux morceaux obtenus. Celui qui mange le dernier carré a perdu.

Montrer que ce jeu est un cas particulier du Nim.

2.1.2 Version à deux coups (*chomp*)

Maintenant, on autorise chaque joueur à casser « en un seul coup » la tablette suivant une des lignes séparant deux rangées, puis suivant une des lignes séparant deux colonnes, les deux séparations n'étant pas obligatoires (mais l'une au moins est exigée). Ceci revient à choisir un carré tel que tous les carrés strictement situés en-dessous et à droite seront mangés. Cette fois, celui qui mange le dernier carré a gagné.

Ce jeu est-il aussi un cas particulier du Nim ?

2.2 Condition de victoire « misère »

Que faut-il modifier dans la stratégie si on décrète que celui qui prend la dernière allumette a *perdu* ? Implémenter ce nouveau jeu avec cette nouvelle stratégie.

2.3 Limitation du nombre d'allumettes prises

Que faut-il modifier dans la stratégie si on décrète qu'un joueur a le droit d'enlever au plus l allumettes à chaque tour ? Implémenter ce nouveau jeu avec cette nouvelle stratégie (en modifiant les fonctions `coup_ordi` et `jeu_ordi`).

Imaginer d'autres contraintes (un nombre premier, pas plus de deux fois le nombre enlevé au coup précédent, etc.) et observer leur incidence sur la stratégie.