Année 2016-2017 Homework 1

Bordeaux INP ENSEIRB MATMECA

Logique et preuve de programmes
IF107

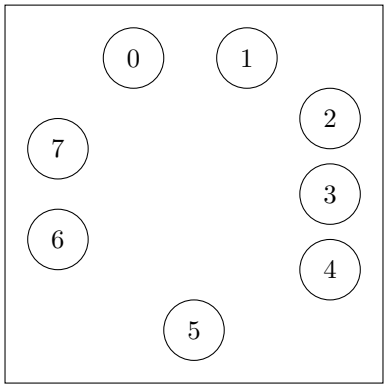
Filière : Informatique, Année : 1

Sujet de : F. Herbreteau

Comments

ULURU (ou ≪Pelican Cove ≫) est un jeu qui consiste à placer chacun des pions sur un plateau de jeu en tenant compte de contraintes de positionnement. La règle du jeu en français est notamment disponible ici : `http://regle.jeuxsoc.fr/uluru_rg.pdf`. Le but de ce devoir est de modéliser le jeu en logique propositionnelle et de le résoudre avec un solveur SAT.

Dans la suite, on numérote les 8 pions de 0 à 7. Le joueur dispose du plateau de jeu représenté ci-dessous, où les positions sont également numérotées de 0 à 7. Un *placement* correspond donc à une bijection $p : [0, 7] \to [0, 7]$ qui associe un pion à chaque position.



À chaque partie, une contrainte de placement est donnée pour chaque pion i :

(00) le pion i peut être placé à n'importe quelle position.

(NS) le pion i est au nord (positions 0 et 1) ou au sud (position 5).

(EW) le pion i est à l'est (positions 2, 3 et 4) ou à l'ouest (positions 6 et 7).

(SW) le pion i est au sud ou à l'ouest.

(NE) le pion i est au nord ou à l'est.

(Nj) le pion i est à côté du pion j (sur la même face, et sur une case adjacente).

(Fj) le pion i est face au pion j (sur la face opposée).

(Cj) le pion i partage un coin avec le pion j (positions 1-2, 4-5, 5-6 et 7-0)

(Sj) le pion i a la même contrainte de placement que le pion j.

Une partie consiste donc en 8 contraintes, une par pion. Un placement est *valide* s'il satifait toutes les contraintes. Certaines parties n'admettent pas de placement valide, par exemple une partie où le pion i a la contrainte Fi.

1 Modélisation en logique propositionnelle

On modélise le placement des pions par l'ensemble des variables propositionnelles $\mathbf{p_{i,j}}$ avec $i, j \in [0, 7]$. Une variable $p_{i,j}$ représente la proposition ≪le pion i est en position j ≫.

Dans ce devoir, on cherche à associer à chaque partie une formule Ψ de la logique propositionnelle telle qu'une valuation satisfait Ψ si et seulement si elle décrit un placement valide.

Exercice 1 (3 points)

Toutes les valuations des variables propositionnelles ne représentent pas un placement. Par exemple, une valuation telle que $p_{0,0}$ et $p_{0,1}$ sont vraies placerait le pion 0 à la fois sur les positions 0 et 1. Modéliser en logique propositionnelle :

($\mathbf{\Psi_1}$) Chaque pion est sur au moins une position

($\mathbf{\Psi_2}$) Chaque position est occupée par au plus un pion

Toute valuation qui satisfait la formule $\Psi_1 \wedge \Psi_2$ est un placement. ♦

Commentaires.

($\mathbf{\Psi_1}$) Chaque pion est sur au moins une position

$$\bigwedge_{i\in[0,7]} \left(\bigvee_{j\in[0,7]} p_{i,j} \right)$$

($\mathbf{\Psi_2}$) Chaque position est occupée par au plus un pion

$$\bigwedge_{j\in[0,7]} \bigwedge_{i\in[0,7]} \left(p_{i,j} \implies \bigwedge_{k\in[0,7],k\neq i} \neg p_{k,j} \right)$$

Exercice 2 (3 points)

Modéliser en logique propositionnelle les contraintes de positionnement suivantes pour le pion i :

$\mathbf{\Psi_{i,(00)}}$ le pion i peut être placé à n'importe quelle position.

$\mathbf{\Psi_{i,(NS)}}$ le pion i est au nord (positions 0 et 1) ou au sud (position 5).

$\mathbf{\Psi_{i,(EW)}}$ le pion i est à l'est (positions 2, 3 et 4) ou à l'ouest (positions 6 et 7).

$\mathbf{\Psi_{i,(SW)}}$ le pion i est au sud ou à l'ouest.

$\mathbf{\Psi_{i,(NE)}}$ le pion i est au nord ou à l'est.

♦

Commentaires.

$\mathbf{\Psi_{i,(00)}}$ le pion i peut être placé à n'importe quelle position.

$$true$$

$\mathbf{\Psi_{i,(NS)}}$ le pion i est au nord (positions 0 et 1) ou au sud (position 5).

$$(p_{i,0} \vee p_{i,1} \vee p_{i,5})$$

$\mathbf{\Psi_{i,(EW)}}$ le pion i est à l'est (positions 2, 3 et 4) ou à l'ouest (positions 6 et 7).

$$(p_{i,2} \vee p_{i,3} \vee p_{i,4} \vee p_{i,6} \vee p_{i,7})$$

$\mathbf{\Psi_{i,(SW)}}$ le pion i est au sud ou à l'ouest.

$$(p_{i,5} \vee p_{i,6} \vee p_{i,7})$$

$\mathbf{\Psi_{i,(NE)}}$ le pion i est au nord ou à l'est.

$$(p_{i,0} \vee p_{i,1} \vee p_{i,2} \vee p_{i,3} \vee p_{i,4})$$

Exercice 3 (3 points)

On s'intéresse maintenant aux contraintes de positionnement relatif Nj, Fj et Cj. On notera N la relation telle que $(j_1, j_2) \in N$ si et seulement si j_1 et j_2 sont deux cases voisines. Similairement, on note F et C les relations qui définissent les cases opposées et les cases formant un coin respectivement.

Modéliser en logique propositionnelle les contraintes suivantes pour le pion i :

$\mathbf{\Psi_{i,(Nj)}}$ le pion i est à côté du pion j (sur des cases voisines appartenant à la même face)

$\mathbf{\Psi_{i,(Fj)}}$ le pion i est face au pion j (sur la face opposée).

$\mathbf{\Psi_{i,(Cj)}}$ le pion i partage un coin avec le pion j.

♦

Commentaires.
Toutes ces formules ont la même structure paramétrée par la relation R qui peut être N, F ou C suivant la formule souhaitée.

$$\bigwedge_{k\in[0,7]} \left(p_{j,k} \implies \bigvee_{l\in[0,7],(k,l)\in R} p_{i,l} \right)$$

Exercice 4 (3 points)

Pour terminer, on considère la contrainte :

(Sj) le pion i a la même contrainte de placement que le pion j.

Donner un algorithme qui calcule la contrainte pour le pion i à partir de la contrainte du pion j.

♦

Commentaires.
La difficulté vient des cycles créés par les contraintes S. Un cycle de contraintes S se réduit à l'absence de contrainte de placement, donc une contrainte 00.

```
tant que (la contrainte sur i est de la forme Sj) faire
  si (i == j) alors
    remplacer la contrainte sur i par 00
  sinon
    remplacer la contrainte sur i par celle sur j
  finsi
fin tant que
```

2 Résolution avec Z3

Z3 [1] est un solveur de contraintes capable de résoudre les problèmes de satisfaisabilité en logique propositionnelle (entre autres). Sa syntaxe est la suivante :

Logique propositionnelle	*Z3*
$\neg\phi$	(`not` ϕ)
$\phi \wedge \psi$	(`and` ϕ ψ)
$\phi \vee \psi$	(`or` ϕ ψ)
$\phi \implies \psi$	(`implies` ϕ ψ)
$\phi \iff \psi$	(`iff` ϕ ψ)

Les opérateurs `and` et `or` sont associatifs à gauche, ainsi $p \wedge q \wedge r$ s'écrit (`and p q r`). Les opérateurs `implies` and `iff` sont associatifs à droite, donc $p \implies q \implies r$ s'écrit (`implies p q r`).

La définition d'un problème de satisfaisabilité consiste en :
- la déclaration des variables propositionnelles avec `(declare-const ...)` ;
- la déclaration d'assertions avec `(assert ...)`. La *conjonction* des assertions définit le problème de satisfaisabilité ;
- la résolution du problème de satisfaisabilité avec `(check-sat)` ;
- et l'obtention d'une solution avec `(get-model)` lorsque la formule est satisfaisable.

Par exemple :

```
(declare-const p Bool)
(declare-const q Bool)
(declare-const r Bool)
(assert (implies (and p q (implies p q r)) r))
(assert (or p (not p)))
(check-sat)
(get-model)
```

déclare trois variables propositionnelles p, q et r, ainsi que deux assertions $p \wedge q \wedge (p \implies q \implies r) \implies r$ et $p \vee \neg p$. Z3 considère la *conjonction* des assertions déclarées, c'est à dire la formule $[p \wedge q \wedge (p \implies q \implies r) \implies r] \wedge [p \vee \neg p]$.

La commande `(check-sat)` lance la résolution du problème de satisfaisabilité. Z3 donne deux réponses possibles : `sat` lorsque la formule est satisfaisable, et `unsat` lorsqu'elle ne l'est pas. Dans le premier cas, la commande `(get-model)` retourne une valuation qui satisfait la formule.

Z3 est invoqué par la commande `z3 -smt2 fichier.smt2` où `fichier.smt2` contient le problème à résoudre (par exemple, les 7 lignes ci-dessus).

Il produit la sortie :

1. https ://github.com/Z3Prover/z3/wiki

```
sat
(model
  (define-fun q () Bool
    true)
  (define-fun p () Bool
    true)
)
```

L'assertion est satisfaite (`sat`) par la valuation qui associe `true` à p et à q.

Le solveur Z3 est disponible sur les machines de l'ENSEIRB-MATMECA dans le répertoire `~herbrete/public/z3/bin`.

Exercice 5 (8 points)

Écrire un script shell qui prend 8 paramètres de ligne de commande, chacun donnant la contrainte de placement des pions 0 à 7. Les valeurs des paramètres d'entrée peuvent être : 00, NS, EW, SW, NE, Nj, Fj et Cj avec $j \in [0, 7]$. On ne demande pas d'implémenter la gestion des contraintes Sj.

Votre script génèrera, sur sa sortie standard, une formule au format Z3 telle que toute valuation qui la satisfait représente un placement valide selon les contraintes données en ligne de commande.

Quelques conseils de mise en œuvre :

- écrivez une fonction pour chaque type de contrainte qui affiche la formule correspondante ;
- implémentez les contraintes incrémentalement : d'abord 00, puis les contraintes absolues : NS, EW, ..., et enfin les autres contraintes. Testez chaque contrainte individuellement après son ajout ;
- l'opérateur `=~` de bash permet de tester si une chaîne de caractères satisfait une expression régulière. Par exemple, `if [[ $X =~ ^C[0-7]$ ]]; then ... fi` teste si la variable `X` contient un mot de la forme Cj avec $j \in [0, 7]$;
- utilisez la documentation bash : `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html` et `http://tldp.org/LDP/abs/html/` ;
- votre script sera testé automatiquement. Il *ne doit pas* ajouter les commandes `(check-sat)` ni `(get-model)` après la formule générée, sous peine d'invalider tous les tests.

♦