Année 2018-2019 Homework 1

Bordeaux INP ENSEIRB MATMECA

Logique et preuve de programmes
IF107

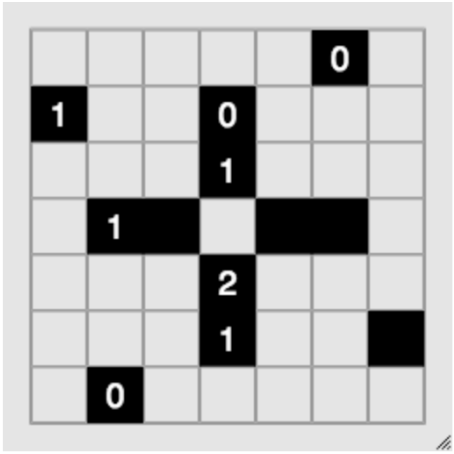
Filière : Informatique, Année : 1

Sujet de : F. Herbreteau

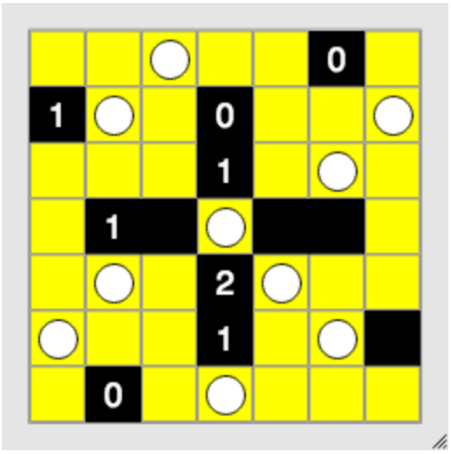
1 Présentation du jeu «Light Up»

«Light Up [1] » est un jeu qui consiste à placer des lampes (light bulb) sur les cases vides d'un plateau de jeu de manière à illuminer toutes les cases qui ne sont pas noires.

Le plateau de jeu est une grille de taille $N \times N$. Chaque case est initialement vide, ou elle contient un mur (case noire). Certains murs peuvent contenir un nombre entre 0 et 4 qui indique le nombre de lampes qui doivent être adjacentes à ce mur. La figure de gauche ci-dessous représente un plateau de jeu.



Un grille vide.



La même grille remplie.

L'objectif du jeu est de placer des lampes sur les cases vides du plateau, comme sur la figure de droite ci-dessus, en respectant les contraintes suivantes :

- toutes les cases vides doivent être illuminées
- une case est illuminée si et seulement si elle contient une lampe, ou elle est éclairée par une lampe située sur la même ligne ou la même colonne
- un mur bloque le passage de la lumière
- une lampe ne doit pas être éclairée par une autre lampe
- une lampe ne peut pas être placée sur un mur
- les cardinalités spécifiées sur les murs doivent être respectées. Par exemple, une case de valeur 2 doit avoir exactement 2 lampes adjacentes (au-dessus, à droite, en-dessous, à gauche).

Les lignes et colonnes de la grille sont numérotées de 1 à N. Le case en haut à gauche a les coordonnées $(1, 1)$. On ajoute, sur le pourtour de la grille, des cases «fictives » qui se comportent comme des murs sans cardinalité (ou plus simplement : sur lesquelles il est interdit de poser une lampe). Ceci permet dans la suite de traiter les cases du bord de la grille (réelle) simplement, sans introduire de cas particuliers. Nous transformons donc la grille de jeu en une grille $[0, N + 1] \times$

1. `https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/lightup.html`

$[0, N+1]$ contenant la grille réelle $[1, N] \times [1, N]$ et donc les cases du pourtour ne peuvent pas contenir de lampe.

L'objectif de ce devoir est de résoudre des grilles de ≪Light Up ≫ par réduction au problème de satisfaction d'une formule ψ de logique propositionnelle. Pour cela, on introduit les variables propositionnelles suivantes, où $0 \leq i, j \leq N+1$ sont des coordonnées sur la grille étendue :

— $bulb_{i,j}$ "il y a une lampe sur la case (i, j)"
— $wall_{i,j}$ "la case (i, j) contient un mur"
— $islit_{i,j}$ "la case (i, j) est éclairée"

Remarque : ces variables sont définies pour toutes les cases de la grille, y compris les cases fictives.

Les questions suivantes amène à l'écriture de ψ, puis à la programmation d'un script `bash` qui génère ψ au format `SMT2` permettant la résolution du jeu avec le solveur SAT `z3`.

2 Jeu sans les murs

Dans un premier temps, on considère uniquement les grilles sans murs. Formaliser chacune des contraintes par une formule de la logique propositionnelle utilisant les variables propositionnelles $bulb_{i,j}$, $wall_{i,j}$ et $islit_{i,j}$ (en indiquant clairement à quelle contrainte correspond chaque formule).

Remarque : on souhaite que la conjonction de vos formules soit satisfaite par une valuation v si et seulement v est une solution de la grille considérée.

On écrira $\bigwedge_{i,j \in [1,N]} \phi$ pour indiquer que ϕ est vraie pour toutes les cases de la grille, et $\bigvee_{i,j \in [1,N]} \phi$ lorsque ϕ est vraie pour au moins l'une des cases de la grille. On peut bien entendu utiliser ces notations pour n'importe quels ensembles d'indices.

3 Ajout des murs sans cardinalité

On ajoute au jeu des murs sans cardinalité. Réécrire les formules de la question précédente qui nécessitent une modification. Justifier.

Pour cela, on commencera par écrire les formules :

— $nowall_{(i,j),(k,l)}$ qui exprime qu'il n'y a pas de mur dans le rectangle, supposé non vide (i.e. $i \leq k$ et $j \leq l$), de coin supérieur gauche (i, j) et de coin inférieur bas (k, l).
— $haswall_{(i,j),(k,l)}$ qui exprime qu'il y a un mur dans le rectangle, supposé non vide (i.e. $i \leq k$ et $j \leq l$), de coin supérieur gauche (i, j) et de coin inférieur bas (k, l).

4 Ajout des cardinalités

On ajoute finalement les cardinalités sur les murs. Écrire les formules à ajouter pour tenir compte des contraintes spécifiées par les cardinalités.

On commencera par écrire les formules :

— $card_{(i,j),0}$ qui indique qu'il n'y a pas de lampe en (i, j)
— $card_{(i,j),1}$ qui indique qu'il y a une lampe en (i, j)
— $card_{(i,j),n,e,s,w}$ qui étant donnés quatre booléens $n, e, s, w \in \{0, 1\}$ indique si les cases adjacentes à la case (i, j) contiennent une lampe ($= 1$) ou n'en contiennent pas ($= 0$). On nomme **n**orth, **e**ast, **s**outh, **w**est les cases situées respectivement au-dessus, à droite, au-dessous et à gauche de la case (i, j). Par exemple, $card_{(i,j),1,0,0,1}$ est vraie si les cases au-dessus (n) et à gauche (w) de (i, j) contiennent une lampe, alors que les cases à droite (e) et au sud (s) de (i, j) n'en contiennent pas.

On pourra écrire une formule $card0_{(i,j)}$, …, $card4_{(i,j)}$ pour chaque cardinalité en (i, j).

5 Résolution avec Z3

Le solveur `Z3` est présenté en annexe A. Écrire un script `bash` qui lit la grille de jeu depuis un fichier, et qui génère sur sa sortie standard un fichier `Z3` définissant une formule de la logique propositionnelle qui est satisfaisable si et seulement si la grille admet une solution. La formule générée devra être conforme à vos réponses aux questions précédentes. Vous vous baserez sur le fichier `light-up-skeleton.sh` fournit avec l'énoncé.

Quelques conseils de mise en œuvre :
- préférez à `(assert (and p q))` deux assertions `(assert p)` et `(assert q)` qui sont plus lisibles (surtout quand `p` et `q` sont de longues formules) ;
- découpez votre code en fonctions pour éviter la répétition et accroître la lisibilité ;
- implémentez les contraintes incrémentalement : d'abord le jeu sans murs, puis le jeu avec murs sans cardinalité, puis les cardinalités ;
- vérifiez systématiquement les formules généréees à la main sur de petits exemples (une grille 3×3) puis sur des exemples plus conséquents (une grille 7×7) ;
- souvenez-vous que nous ajoutons des cases fictives autour de la grille afin de traiter les cases du bord sans devoir introduire de cas particuliers ;
- votre script sera évalué par des tests automatiques. Il est nécessaire de nommer vos variables propositionnelles `bulb_i_j`, `wall_i_j` et `islit_i_j` où $i, j \in [0, N+1]$ et il n'est pas permis d'ajouter d'autres variables.

A Brève présentation du solveur Z3

`Z3` [2] 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 ...)`. Le solveur `Z3` résout la formule formée par la conjonction des assertions.
- 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)
```

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

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 :

```
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`.