Rapport du projet Logique

Aubian Guillaume & Hilaire Matthieu

March 5, 2015

Abstract

Voici le rapport de notre projet Logique trouvable ici:
`https://github.com/gaubian/Projet_logique-L3-ENS-Cachan`

Part I
Comment utiliser notre programme:

Le fichier à compiler est linké avec le mail, ou au pire situé ici:
`https://github.com/gaubian/Projet_logique-L3-ENS-Cachan`

Pour le compiler, ne pas oublier d'ajouter le module `unix.cma`. Par exemple:
`ocamlc unix.cma -o projet.out projet_logique.ml`

Quand à l'exécutable ainsi créé, on l'utilise en lui donnant deux arguments, qui sont deux noms de fichiers. Le premier correspond à l'endroit où est stockée l'instance du problème que l'on souhaite résoudre. Le second correspond au nom du fichier que l'on va créer, et dans lequel sera stocké la solution. Par exemple:

```
./projet.out "/gaubian/Documents/entree" "/gaubian/Documents/sortie"
```

Le fichier d'entrée est de la forme telle que définie dans l'énoncé. Par exemple:

```
%%%%
SM %
%MDE
%%%%
```

Alors que la sortie sera de la forme:

```
%%%%
S\ %
%\DE
%%%%
```

Part II
Idée Générale:

1 La technique sans les cycles:

Tout d'abord, voici comment nous nous sommes débrouillé pour résoudre une instance sans se soucier des éventuels problèmes de cycles externes.

Dans un premier temps, on parse le fichier d'entrée, pour en faire une matrice de caractères, ce qui ne pose pas de problème.

Ensuite, on va associer à chaque case un certain nombre de variables propositionnelles de la manière suivante:

- A chaque case, quelqu'elle soit, on associe quatre variables propositionnelles correspondant aux quatre directions sortantes de la case (i.e. par exemple la variable codant la direction droite est assignée à `true` si et seulement si le laser sort de la-dite case par la droite)
- A chaque miroir, on associe une variable supplémentaire, codant la position du dit-miroir: `true` si il est dans la position \ , `false` sinon

De plus pour chaque case, on a un certain nombre de formules propositionnelles qui doivent être vérifiées:

- Pour la sortie et pour les diamants, il faut que le laser passe par la case en question, i.e. qu'au moins une des variables directionnelles soit à `true`
- Pour les cases vides et les diamants, il y a une règle de propagation, qui désigne le fait que le laser passe par une case dans une certaine direction si et seulement si il est passé dans la même direction par la case située directement dans la direction opposée
- Pour les miroirs, la règle de propagation est modifiée, puisqu'elle change selon l'orientation du dit-miroir
- Enfin, les murs ont des variables directionnelles qui doivent être toujours fausses

En mettant chacune des formules propositionnelles correspondant à chaque case sous forme de conjonction de clauses, on obtient une grosse conjonction de clauses qui correspond à la formule générale. Une fois qu'on l'a faite passer à `z3`, on en déduit soit que l'instance considerée n'est pas résoluble, soit qu'elle est résoluble, et dans ce cas-là on a une orientation possible des miroirs qui nous est donnée par l'affectation proposée par `z3`.

Le problème de cette méthode, c'est que ce genre de configuration est considérée comme correcte:

```
%%%%
S  E
%/\%
%D %
%\/%
%%%%
```

2 La méthode finale:

Pour résoudre ce problème, on rajoute à chaque case une variable entière que l'on pourrait interpréter comme étant le temps au bout duquel le laser arrive sur cette case. Ainsi, on a une règle évidente qui est que si le laser passe d'une case à une suivante, alors l'entier associé à la première est inférieur à celui associé à la seconde.

Formellement, cela veut dire que pour chaque variable directionnelle `P` associée à une case `prem` en direction de `sec`, on a:
$P \Rightarrow (t_{prem} > t_{sec})$

Ainsi, on a aucun problème de cycle, puisque $>$ étant transitive, on aurait des incohérences.

Une fois une telle formule propositionnelle donnée à `SMT`, on agit comme précédemment.

Part III
Difficultés

La principale difficulté rencontrée est venue de la manière dont `SMT` renvoie les résultats, qui est assez compliquée à parser. A part ça, nous n'avons pas eu de grosse difficulté: une fois que l'on avait fait la première moitié (relative à `z3`, et dont la principale difficulté était d'avoir une idée de la manière dont on voulait transformer en formule logique une instance du problème), la seconde partie n'était pas très compliquée, modulo ce problème avec `SMT`.