Rendu de projet:
Introduction à la vérification
Le berger et le solitaire

ABDEL WAHAB Ismail

17 Avril 2020

Sommaire

Chapitre 1

Le problème du berger

1.1 Énoncé du problème

La situation est la suivante: Sur la rive gauche d'une rivière, **un berger**(Shepherd), **un loup**(Wolf), **un chou**(Cabbage) et **un mouton**(Sheep), doivent traverser cette dernière.
Ils ont à leur disposition **une barque** qui:

1. Ne peut être dirigée seulement que par le berger.
2. Ne supporte au plus que deux individus en même temps.
3. Peut faire autant de voyages que nécessaire.

Néanmoins, il existe des tensions entre les individus de ce groupe, nous devrons donc trouver une succession de voyages dans laquelle:

1. Le mouton ne doit jamais être avec le loup si le berger n'est pas là.
2. Le chou ne doit pas être seul avec le mouton si le berger n'est pas là.

En effet le loup(resp. le mouton) mangerait le mouton(resp. le chou) sinon.

1.2 Caractéristique d'une solution valide

Une solution valide devra nous indiquer, une succession de déplacements utilisant la barque, tout en respectant les conflits entre individus, que l'on définira à partir de maintenant comme "les contraintes" du problème.

Ainsi notre objectif est de générer à l'aide de spin une trace où, à la fin de celle-ci tous les individus sont sur la rive droite **ET** durant le trajet les contraintes ont été respectées.

1.3 Apports fait au modèle promela fourni

Le modèle fourni n'a pas été modifié dans son fonctionnement, seuls des apports supplémentaires ont été fait dans le code.

1.3.1 Les positions des individus

Afin de repérer leurs positions, nous utilisons un tableau de booléen de taille 4. Extrait du code de /peg_solitaire/peg_solitaire.pml:

```
/* Position of characters:
  position[0] ==> Shepherd
  ________[1] ==> Sheep
  ________[2] ==> Cabbage
  ________[3] ==> Wolf */
bool positions[4];
```

Ainsi position[0]==0 signifie que le berger est sur la rive gauche, sinon il est sur la rive droite.

1.3.2 Les déplacements non-atomiques

Lors d'un déplacement en binôme, deux individus doivent simultanément changer de rive. Cela pose un petit problème à notre modèle:

Soit la situation où M, L et B sont sur la rive gauche:

1) **B souhaite aller à droite en companie de L**. On modifie donc la position de B qui est maintenant égale à 1.

2) A ce moment préci: nous avons **B à droite** et **M avec L à gauche**. (en effet nous n'avons pas encore modifié la position de L).

3) Et bien que le déplacement ne soit pas fini, notre système (si il est bien établi) doit reconnaître cette situation comme un non respect des contraintes du problème.

Pour pallier à cela, nous introduisons un booléen qui indiquera à notre système quand il doit vérifier si une des contraintes n'est pas respectée.

```
/* Flag used to verify if we need to
check the current state of the system in our LTL formula */
bool check_here=0;
```

Durant un déplacement, cette variable sera mise à 0 et pendant le reste de la simulation elle sera à 1. Elle sera utilisée au sein de notre formule LTL afin "d'activer" ou non les vérifications à faire.

1.4 Comment obtenir une solution au problème?

Notre but ici sera de volontairement produire une "erreur" à l'aide de spin.

En effet spin possède une fonctionnalité particulière : lorsqu'un processus never est atteint, l'outil nous informe que notre modèle respecte la formule LTL ayant générée le processus never, mais surtout nous fourni une trace de l'exécution ayant amenée cette erreur.

Pour que la trace soit la solution à notre problème, nous devons fournir à spin un processus never qui est l'énoncé de notre problème. Notre formule LTL doit donc etre:

```
[On respecte les contraintes] U [Tout le monde est à droite]
```

1.5 Production d'une trace qui est solution

Vous avez à votre disposition dans le dossier *shepherd/* un script *compile.sh*. Executer ce script, génèrera les fichiers *pan**, compilera le code C, et affichera la trace générée.

Pour cela placez vous à l'aide d'un terminal dans *shepherd/* et éxécutez:

```
./compile.sh [-v|-h]
```

Pour plus d'informations, merci d'utiliser "*-h*" afin de voir les modalités d'usage.

1.6 Résultat de notre vérification

1.6.1 Les voyages effectués

Voici une copie du texte produit par notre programme à l'aide des *printf*, cela résume la solution trouvée par spin:

```
The {shepherd} if going [RIGHT].
    The [sheep] is coming with him to the [RIGHT].
The {shepherd} if going [LEFT] but he is [ALONE].
The {shepherd} if going [RIGHT].
        The [wolf] followed him to the [RIGHT].
The {shepherd} if going [LEFT].
    The [sheep] is coming with him to the [LEFT].
The {shepherd} if going [RIGHT].
        He takes the [cabbage] with him.
The {shepherd} if going [LEFT] but he is [ALONE].
The {shepherd} if going [RIGHT].
    The [sheep] is coming with him to the [RIGHT].
```

1.6.2 MSC associé à notre solution

A l'aide des *printf* nous avons une formalisation de la solution, Néanmoins spin peut nous fournir un MSC qui vous est copié ci-dessous:

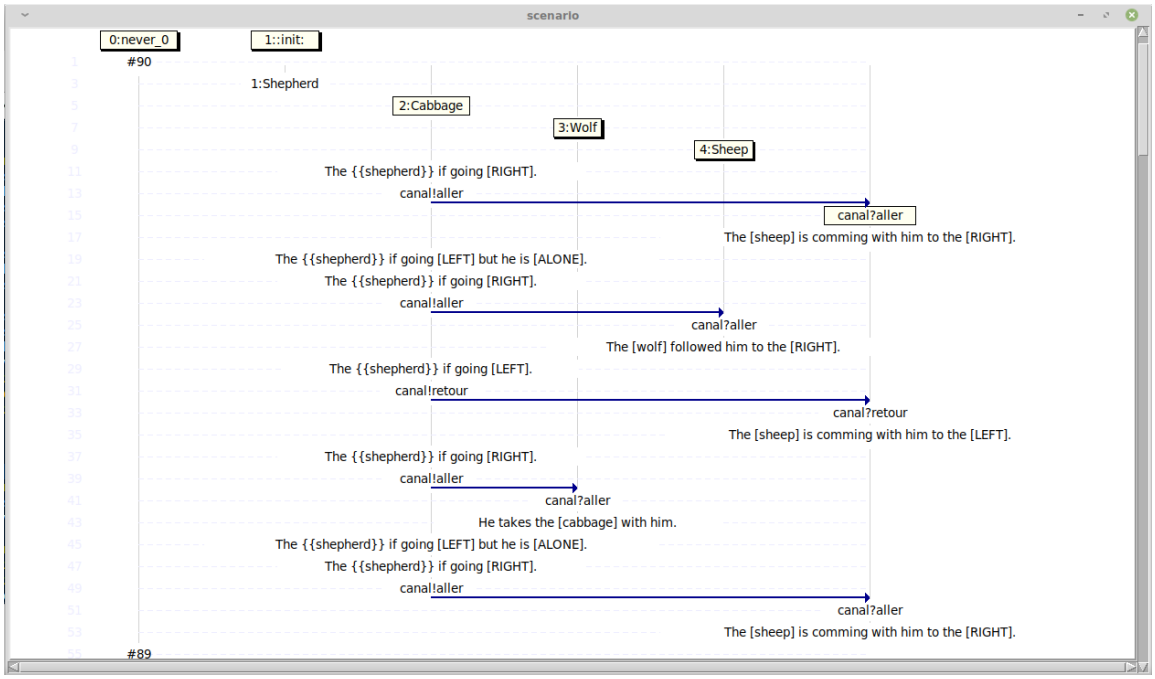

Figure 1.1: Message sequence chart of our solution.

À noter que: **pour une raison inconnue, les flèches en bleu sont décalées d'une colonne vers la droite**. Bien évidement c'ést le B(colonne Shepherd) qui envoie les messages et non pas C.

Chapitre 2

Le solitaire

2.1 Présentation du jeu

2.1.1 Environnement de jeu

Le solitaire (aussi appelé peg solitaire), est un jeu de plateau comportant des emplacements, sur lesquels on retrouve des pions (appelés **peg(s)**).

Dans la position initiale, tous les emplacements sont remplis de pegs sauf une case, le trou initial.

2.1.2 Mécanique de jeu

La mécanique de jeu est simple, on peut faire sauter un peg par dessus un autre si à la réception, il arrive sur un emplacement libre.
Le peg ayant été sauté est également retiré du jeu.

2.1.3 Position gagnante

Pour gagner il suffit simplement d'atteindre une position où il ne reste qu'un seul peg sur le plateau.

On utilisera dans ce devoir, la version où le peg restant peut se trouver n'importe où. Sur certains plateaux comme la version anglaise, le but est que le dernier peg soit sur l'emplacement libre de la position initiale. Malheureusement sur d'autres (comme l'européen) cela n'est pas possible.

2.2 Modélisation du problème

2.2.1 Le modèle du jeu

Mon modèle opère sur deux phases:

1. On initialise tout d'abord le plateau.
   (cf code du processus **init()**)
2. Tant que le nombre de pegs est supérieur à 1:
   (a) On choisit aléatoirement "i" et "j".
   (b) Si un déplacement est possible sur la case (j,i) on peut le faire.
   (Ici le modèle peut aussi choisir de ne rien faire, il n'est pas obligé d'effectuer l'action.)
   (cf code du processus **play_game()**)

2.2.2 Processus never

Notre objectif étant de démontrer si il existe une succession de coups, amenant à une position gagnante(ie. un seul peg restant), sachant que les contraintes du jeu sont spécifiées dans notre modèle, il nous suffit de vérifier si l'on peut bien atteindre une telle position.
Il est ainsi naturel que notre formule LTL soit: F(p).

Avec p un booléen de type: #define p (nombre_de_pegs_restants == 1)

2.3 Génération automatique basée sur un plateau

2.3.1 Représentation d'un plateau à fournir

Differents types de plateaux existent pour ce jeu, j'ai pris soin d'en modéliser certains qui sont mis à votre disposition et sont répertoriés dans:

`/peg_solitaire/boards/*.board`

C'est une approche semi-graphique qui a été employée pour qu'un utilisateur puisse soumettre un plateau de jeu et que notre script génère pour lui le fichier promela correspondant.

C'est dans un fichier texte où l'utilisateur "dessinera" son plateau.

Les règles d'écriture sont simples:

1. Le plateau est inscrit dans un rectangle de caractères où:
   (a) "o" est un peg
   (b) "X" est le trou initial
   (c) " " représente les cases qui n'appartiennent pas au plateau.
2. Les commentaires sont acceptés.
   Ils commencent sur une nouvelle ligne avec le caractère "#".
3. Les sauts de lignes ou lignes vides sont interdits.

Exemple d'un plateau: */peg_solitaire/boards/european.board* :

```
  Xoo
#This line is a comment and will not be taken in consideration
 ooooo
ooooooo
ooooooo
ooooooo
 ooooo
  ooo
```

2.3.2 Script python

La génération automatique du fichier promela est faite à l'aide d'un script python. Ce script est consultable:

`/peg_solitaire/generate_pml_file.py`

Le rôle de ce script est de prendre en paramètre un fichier "some_board.board", de le parser et de produire le fichier promela.

Pour cela notre script va composer le fichier PML à l'aide de différents bouts de codes dans */peg_solitaire/src/* ainsi que des données du plateau de jeu donné en paramètre.

2.3.3 Exemples de génération de fichiers PML

Placez vous dans */peg_solitaire/* et exécutez:

```
./generate_pml_file.py boards/line_boards/horizontal_line.board
./generate_pml_file.py boards/square_boards/square_4x4.board
./generate_pml_file.py boards/european.board
./generate_pml_file.py boards/UNSOLVABLE.board
```

2.4 Trace du plateau anglais

Ci-dessous, la trace du plateau anglais qui permet de resoudre le jeu avec l'emplacement initialement vide au centre.

```
Never claim moves to line 93 [(1)]
          GO {direction} : [ROW number;Column number] {arrow direction}
              Go down : [1;3]
              Go right: [2;1]
              Go down : [0;2]
              Go left : [0;4]
              Go left : [2;3]
              Go right: [2;0]
              Go up   : [2;4]
              Go left : [2;6]
              Go up   : [3;2]
              Go down : [0;2]
              Go right: [3;0]
              Go up   : [3;2]
              Go up   : [3;4]
              Go down : [0;4]
              Go left : [3;6]
              Go up   : [3;4]
              Go up   : [5;2]
              Go right: [4;0]
              Go up   : [4;2]
              Go down : [1;2]
              Go right: [3;2]
              Go up   : [4;4]
              Go down : [1;4]
              Go left : [4;6]
              Go right: [4;3]
              Go up   : [6;4]
              Go down : [3;4]
              Go right: [6;2]
              Go up   : [6;4]
              Go left : [4;5]
              Go up   : [5;3]
spin: peg_solitaire.pml:92, Error: assertion violated
spin: text of failed assertion: assert(!((nbPegs==1)))
Never claim moves to line 92 [assert(!((nbPegs==1)))]
```

Remarque: les déplacements se réfèrent aux numéros de lignes et de colones, ainsi:

```
Go left  : [2][4]
```

Veut dire: à la deuxième ligne et quatrième colone sauter vers la gauche. Également, lors d'une simulation avec spin, les printf vous afficheront, à la suite de chaque ligne, une flèche directionnelle afin de faciliter la lecture de la trace. (L'indexation des lignes commence à 0)

2.5 Résultats obtenus

Notre modèle semble bien fonctionner, mais supporte assez mal la montée en charge en terme de taille de plateau.
Afin d'obtenir ces résultats le script compile.sh utilise les paramètres suivants pour compiler pan.c:

```
gcc -DBITSTATE \
-DNOBOUNDCHECK -DNOCOMP -DNOFAIR -DNOSTUTTER -DSAFETY\
-DNIBIS -o pan pan.c
```

Et par la suite cherche la plus courte trace possible:

```
./pan -i
```

Résultats obtenus: Temps moyen de calcul selon les plateaux.

Nom du fichier	Taille du plateau	Nombre de pegs	Temps de calcul
horizontal_line.board	1x6	5	<1 s
vertical_line.board	6x1	5	<1 s
square_4x4.board	4x4	15	1 à 2 s
rectangle_6x4.board	6x4	23	<2 minutes
square_5x5.board	5x5	24	2 minutes
rectangle_8x4.board	8x4	31	<3 minutes
square_6x6.board	6x6	35	3 minutes 30 s
english.board	inclut dans 7x7	32	4 minutes
european.board	inclut dans 7x7	36	Solution non atteinte
diamond.board	inclut dans 9x7	41	Solution non atteinte
rectangle_8x6.board	8x6	47	Solution non atteinte
square_8x8.board	8x8	63	Solution non atteinte

A noter que nous trouvons bien une trace permettant de gagner sur le plateau anglais du jeu, mais que le plateau européen (bien que résolvable avec l'emplacement libre initial en haut à gauche), ne provoque pas de trace.

Je pense que cela est du à mon modèle, en effet il comporte une grande quantité d'états ce qui peut être un problème pour spin lors de l'exploration de l'automate qu'il génère. Spin ne semble pas accepter de faire un calcul de plus de cinq minutes avec les options de compilation choisies. Malheureusement je n'ai pas trouvé de combinaison d'options plus efficace que celle-ci.

De plus j'ai réalisé, à l'aide de mon modèle spin, une partie "à la main" avec le plateau européen et j'ai bien réussi à gagner. Montrant que mon modèle est bien fonctionnel mais que je dois certainement manquer de ressources face à une verification de spin.

La video utilisé pour cette démarche: https://www.youtube.com/watch?v=fUVukxJtGi8

D'après les resultats que nous avons recueillis, il semble donc que le temps de calcul croît exponentiellement, quand la taille et le nombre de pegs du plateau augmentent.