# Travaux Dirigés No9
# Modélisation de systèmes concurrents

Frédéric Peschanski

28 mars 2017

Dans ce TD nous utilisons le langage Promela (et en TME l'outil Spin) pour modéliser et analyser des systèmes concurrents.

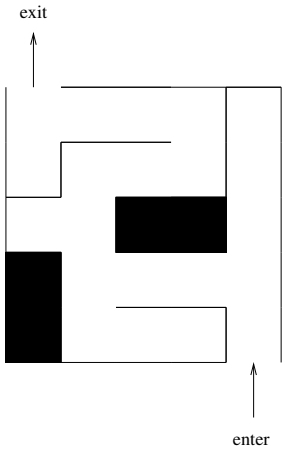
## Exercice 1 : Labyrinthes

On se propose de coder des labyrinthes de façon à ce que la découverte d'un chemin vers la sortie (si il existe) se ramène au problème de la détection de *deadlock* dans le système concurrent encodant le labyrinthe.

Dans un état donné, les actions possibles sont les suivantes :
- action `NORTH` : un déplacement vers le nord est possible
- actions `SOUTH`, `EAST`, `WEST` : idem sud, est ouest.
- action `EXIT` : la sortie vers `STOP` est possible
- action `ENTER` : représente l'entrée dans le labyrinthe

### Question 1

On souhaite décrire en Promela le système représentant le labyrinthe suivant :




Le labyrinthe sera défini par un processus `laby` dont le comportement sera observé par le processus suivant :

```
active proctype observateur() {
  mtype dir ;

  do
  ::go?(dir) -> if
                ::dir==EXIT -> printf("go EXIT") ; goto exit
                ::else -> printf("go %e\n", dir) // %e pour afficher un mtype
                fi
  od
```

```
exit:
}
```

### Question 2 (en TME)

Pour déterminer un chemin conduisant à la sortie, on peut lancer le mode simulation :

```
spin laby.pml
```

De façon plus intéressante, on souhaite déterminer un chemin dans le labyrinthe par analyse de *deadlock*. Pour cela il faut ajouter un *deadlock* explicite dans le labyrinthe (par exemple avec une garde `false`) puis construire un analyseur :

```
spin -a laby.pml
```

- Avec une recherche en profondeur (par défaut) :
  `gcc -DSAFETY -O2 pan.c -o pan`
- Avec une recherche en largeur :
  `gcc -DSAFETY -DBFS -O2 pan.c -o pan`

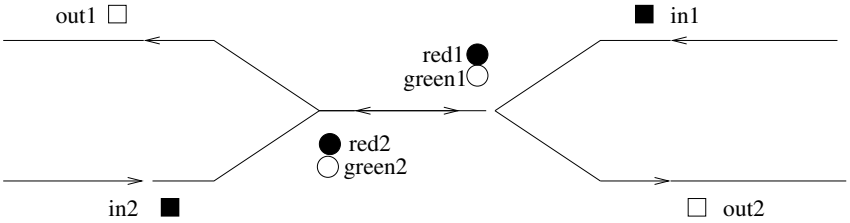
Si un *deadlock* est détecté, on peut afficher la séquence de messages par la commande suivante :

```
spin -t -M laby.pml
```

Quelle différence faites vous entre l'analyse en profondeur et celle en largeur ?

## Exercice 2 : Les petits trains

On considère un système ferroviaire composé de deux voies en sens opposées et d'un tronçon commun protégé par un système de signalisation.




Le système est composé des définitions de processus suivantes :

- une définition de processus `Sensor` pour les capteurs : `in_1`, `in_2`, `out_1` et `out_2`.
- une définition `Feu` pour les feux de signalisation (un par voie) qui peuvent être dans deux états : `RED` ou `GREEN`.
- une définition `Train` pour les trains arrivant sur la voie (1 train maximum par voie sur le tronçon considéré).
- une définition `Control` pour le processus de contrôle du tronçon modélisé.

Les contraintes de modélisation sont les suivantes :

- les différents processus du système communiquent uniquement par l'intermédiaire de canaux synchrones
- seul le processus de contrôle peut modifier l'état des feux
- on ne peut empêcher un train d'arriver
- un train ne peut s'engager sur la voie unique que si son feu est dans l'état `GREEN`
- un train sort forcément du tronçon après passage sur la voie.

On impose de plus les définitions suivantes :

```
mtype { RED, GREEN };

proctype Sensor(chan sense; chan signal) {
```

```
  bool b;

end: // pas de deadlock ici
  do
  ::sense?b -> signal!true
  od
}

proctype Feu(chan swtch; chan change) {
  bool b;
  mtype color;


end_red_color:
  color = RED ;
  change!color ;

end_wait_swtch:
  swtch?(b) -> if
               ::color == RED -> goto end_green_color
               ::color == GREEN -> goto end_red_color
               fi

end_green_color:
  color = GREEN ;
  change!color ;
  goto end_wait_swtch
}
```

### Question 1

Compléter le modèle du système ferroviaire.

Vérifier (en TME) que le système ne contient aucun *deadlock*.

### Question 2

Exprimer et vérifier (en TME) les propriétés suivantes :

- au plus un train se trouve sur la voie partagée
- les feux ne sont pas simultanément dans l'état `GREEN`

Modifier le modèle pour contredire ces propriétés. Peut-on décrire un modèle dans lequel une seule de ces propriétés est invalidée ?

### Question 3

Exprimer et vérifier (en TME) que les propriétés suivantes ne sont pas valides pour le système concerné :

- un train arrivant sur une voie donnée (1 ou 2) finira forcément par passer
- un feu finit forcément par passer au vert

Proposer des modifications du modèle pour garantir ces propriétés.