Laboratoire 7

Model Checking

Source : Image faite par ChatGPT

Introduction au problème

Le modèle checking est une méthode permettant de vérifier formellement le comportement de systèmes concurrents ou complexes en explorant toutes les exécutions possibles. L'objectif est de détecter des états non désirés comme des blocages (deadlocks) ou des violations de propriétés.

Dans ce laboratoire, nous désirons réaliser une classe gérant l'accès à un sas de sécurité. Cette classe est exploitée par des agents 0 et des agents 1. Les agents des deux types ne peuvent pas être présents en même temps dans le sas, et celui-ci a une capacité de N > 1 agents au maximum. Les fonctions permettant d'entrer dans le sas (access()) et d'en sortir (leave()) prennent en argument le type d'agent, soit un entier valant 0 ou 1 (on ne vérifie pas que ça soit bien 0 ou 1, on assume qu'on nous utilise correctement). Elles prennent aussi un ObservableThread afin de pouvoir effectuer les séparations des sections au sein de nos fonctions. La fonction access() peut être bloquante. La priorité est donnée aux types d'agents actuellement présents dans le sas. S'il n'y a pas d'agents dans le sas, et que des agents (du même type que ceux dernièrement en attente) sont en attente, ils peuvent rentrer. Sinon, on laisse la place aux agents en attente de l'autre type. Si personne n'est en attente, alors c'est le premier agent qui arrive qui rentre.

Notre implémentation du Sas a purement pour but d'être compatible avec nos tests, et de la sorte ses fonctions incorporent déjà la mise en place des sections. Si on voulait utiliser le Sas ailleurs, il faudrait le « nettoyer », pour ainsi dire.

L'objectif est d'assurer que le comportement du sas respecte les règles établies et d'éviter tout blocage ou comportement inattendu.

Choix d'implémentation

L'implémentation repose sur plusieurs classes principales, chacune jouant un rôle essentiel dans la modélisation et la vérification du système :

SasAccess

Cette classe gère l'accès au sas et contient les mécanismes nécessaires pour :

- Contrôler l'entrée des agents selon leur type.
- Gérer les files d'attente pour les agents en attente d'accès.
- Assurer que le sas ne dépasse jamais sa capacité maximale (**N agents**).
- Maintenir une priorité pour les agents du type déjà présent dans le sas.

Pour cela, des **sémaphores** et un **mutex** sont utilisés pour synchroniser les différents threads. Les méthodes principales sont :

- *access(int id, ThreadParent overseer)** : Permet à un agent de demander l'accès au sas. Si l'accès est impossible, l'agent est placé en attente sur un sémaphore.
- *leave(int id, ThreadParent overseer)** : Permet à un agent de quitter le sas et de libérer les agents en attente, tout en respectant les priorités.

ThreadParent (et ses dérivés ThreadZero et ThreadOne)

Cette classe permet de gérer un agent entrant et sortant du sas, et donc de tester le sas. Elle utilise un graphe de scénario pour modéliser les différentes étapes de leur exécution, en incluant :

- L'entrée dans une section critique pour demander l'accès au sas.
- L'attente (si nécessaire) et l'accès effectif au sas.
- La sortie du sas après avoir accompli leur tâche.

ModelSas

Cette classe regroupe les threads et les scénarios de test pour modéliser les différentes combinaisons possibles d'exécution. Les principales étapes incluent :

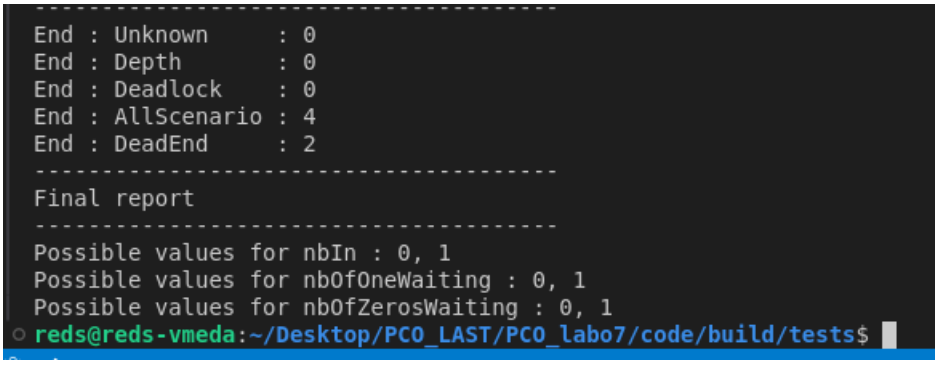
- **build()** : Création des threads et des scénarios de test. Les scénarios sont générés avec la classe **ScenarioCreator**, qui construit toutes les interleavings possibles des threads.
- **preRun()** : Initialisation du sas avant chaque scénario.
- **postRun()** : Affichage des résultats après l'exécution d'un scénario (nombre d'agents dans le sas, agents en attente, etc.).
- **finalReport()** : Rapport global des exécutions, montrant les différents états possibles du système.

Tests effectués

Les tests ont été réalisés pour différents scénarios afin de valider le comportement du sas. Voici les résultats des scénarios testés :

TEST 1 : Taille = 2, 2 Threads

- Un agent de type 0 et un agent de type 1 tentent d'accéder au sas.
- Résultat attendu :



TEST 2 : Taille = 2, 3 Threads

- 3 agents, 2 de type 0 et 1 agents de type 1 tentent d'accéder au sas.
- Résultat attendu :

```
---------------------------------------
End : Unknown     : 0
End : Depth       : 0
End : Deadlock    : 0
End : AllScenario : 52
End : DeadEnd     : 38
---------------------------------------
Final report
---------------------------------------
Possible values for nbIn : 0, 1, 2
Possible values for nbOfOneWaiting : 0, 1
Possible values for nbOfZerosWaiting : 0, 1, 2
reds@reds-vmeda:~/Desktop/PCO_LAST/PCO_labo7/code/build/tests$
```

TEST 3 : Taille = 3, 4 Threads

- 4 agents, 2 de chaque type tentent d'accéder au sas.
- Résultat attendu :

```
---------------------------------------
End : Unknown     : 0
End : Depth       : 0
End : Deadlock    : 0
End : AllScenario : 1216
End : DeadEnd     : 1304
---------------------------------------
Final report
---------------------------------------
Possible values for nbIn : 0, 1, 2
Possible values for nbOfOneWaiting : 0, 1, 2
Possible values for nbOfZerosWaiting : 0, 1, 2
reds@reds-vmeda:~/Desktop/PCO_LAST/PCO_labo7/code/build/tests$
```

Ce que l'on peut constater à travers les différents tests est que le nombre de scénarios possibles et le nombre de *DeadEnd* augmentent de façon exponentielle avec le nombre de threads et la taille du sas. Cela est dû à la croissance combinatoire des interleavings possibles dans la génération des scénarios.

Les résultats montrent que :

- **Synchronisation correcte** : Le sas gère efficacement les accès simultanés et les files d'attente, sans dépasser la capacité configurée.
- **Gestion des threads équilibrée** : Les threads respectent les règles de priorité et les états attendus pour chaque test.
- **Complexité accrue** : Avec des configurations plus grandes, la génération de scénarios devient plus coûteuse en termes de temps de calcul.

Ces observations valident le fonctionnement du sas dans des scénarios simples et complexes.

Merci de votre lecture !