Notes de cours de Analyse de programmes

Yann Miguel

18 février 2021

Table des matières

1 Introduction

Trois parties pour le cours:

1. Concurrence
2. Protocole cryptologique
3. Reverse engineering/Fonction à l'exécution /Attaques

Chaque partie aura des logiciels associés:

1. Promela/Spin
2. Proverif
3. GDB/Radare2/Ghidra

La vérification dans le concurrent est encore plus compliqué que dans le séquentiel.

Un protocole est un algorithme.

Sur Promela, -> et ; sont équivalents, mais -> est utilisé plus souvent après une condition.
Le mot clé "active" permet de dire que le processus peux être lancé. Si on ne mets pas active, il faut le lancer manuellement depuis init. Il est conseillé, si on lance les processus dans init, de mettre les run dans un "atomic". On peux donner des arguments aux processus. Les crochets définissent les tableaux.
L'entrelacement entre P et Q est noté $P||Q$.
"atomic" et "d_step" sont des mots clés similaires.

Spin permet de tester des propriétés comme:

- Exclusion mutuelle: un seul à accès à la ressource à la fois.
- Deadlock: chaque processus attend une ressource qu'un autre utilise.
- Famine: un processus n'a **jamais** accès à la ressource.

2 Cours 1

2.1 Problème de concurrence

2.1.1 Exclusion mutuelle

Deux preuves de correction possible:

1. à la main: graphe
2. via Spin: variable critique + assert

Afin d'éviter les problèmes de Deadlock, on peux faire une synchronisation par blocage.
Cela veut dire mettre une condition qui fait attendre le processus jusqu'à ce qu'il puisse accéder à la ressource.

On peux aussi utiliser des "sémaphores", via une variable qui va incrémenter ou décrémenter selon les appels faits. La valeur de la variable de sémaphore doit être égale au nombre de processus qui peuvent prendre la ressource en parallèle.

Il existe aussi l'algorithme de Fisher, qui est faux si il n'y a pas de délais entre les instructions. C'est un algorithme très efficace, mais a un problème de contrainte de temps.

2.2 Model-Checking via logique LTL

Il s'agit de la logique permettant la vérification via Spin sans utiliser d'assert.
Avec cette technique, on va travailler sur les systèmes de transition.

La définition formelle de la logique LTL ressemble assez à celle d'un automate, sauf que les états sont étiquettés par les variables vraies à cet état.
Une exécution est donc une suite supposée infinie.

Les opérateurs de la LTL:

- $\neg$: négation
- $\wedge$: et logique
- $\vee$: ou logique ($\neg\wedge$)
- X_φ: φ est vrai à l'instant suivant
- $\varphi U \psi$: φ est vrai jusqu'à ce que ψ soit vrai.
- $\Box$: globalement
- $\Diamond$: un jour
- $\varphi_1 R \varphi_2$ φ_2 toujours vraie jusqu'à φ_1 vraie (les deux peuvent l'être en parallèle)

T est un modèle de φ ss toute exécution de T valide φ.
Propriétés classiques:
- Safety: $\Box_\varphi$
- Liveness: $\Diamond_\varphi$
- $\Diamond\Box_\varphi$: à partir d'un moment, φ sera toujours vraie.
- $\Box\Diamond_\varphi$: φ sera infiniment souvent vraie.

Une formule de LTL correspond à un **automate de Buchi**.

2.2.1 Automate de Buchi

Au lieu de montrer que le programme vérifie φ, on va montrer que le programme ne peux pas vérifier $\neg\varphi$, et on associe à $\neg\varphi$ un automate de Buchi.

On fait donc un produit carthésien entre le programme et l'automate de $\neg\varphi$, ce qui fait un autre automate de Buchi.

Comment on associe un automate de Buchi à une formule?

Il faut donner à l'automate un état final par lequel on passe infiniment souvent. L'automate n'est pas obligatoirement fini.
La lettre ω représente une suite infinie du symbole.

Les automates non-déterministes ont plus de puissance que les automates déterministes.
On sait si un automate est "vide" si on ne peux pas retourner vers son état final à partir du dit état final.

Chaque formule LTL a un automate de Buchi qui la reconnaît exactement.

Traduire un automate de Buchi alternant en automate de Buchi tranditionnel est de complexité exponentiel le en la taille de la formule.
L'algorithme de détermination du "vide" a une complexité linéaire en la taille de $T\times A_{\neg\varphi}$

2.3 Spin

Spin prends un programme Promela avec une propriété LTL et donne un graphe d'exécution et un automate de Buchi correspondant à la négation de la formule.
Cet automate de Buchi est un **never claim**.

3 Informations importantes

Groupe de tp de 4, 5 tps, dont 4 à rendre
1 partiel de mi parcours
1 exam final
NF=$\frac{EF}{2} + \frac{DS}{4} + \frac{TP}{4}$
Rendu tp: 1 semaine après tp en général, bonus si 5 rendus Livre:
Principle of the SPIN model checker