

# Notes de cours de Analyse de programmes

Yann Miguel

25 mars 2021

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Cours 1</b>	<b>3</b>
2.1	Problème de concurrence . . . . .	3
2.1.1	Exclusion mutuelle . . . . .	3
2.2	Model-Checking via logique LTL . . . . .	3
2.2.1	Automate de Buchi . . . . .	4
2.3	Spin . . . . .	4
<b>3</b>	<b>Cours 2</b>	<b>5</b>
3.1	Introduction . . . . .	5
3.2	Cadre . . . . .	5
3.3	Modélisation . . . . .	6
3.3.1	Modèle Formel . . . . .	6
<b>4</b>	<b>Cours 3</b>	<b>8</b>
<b>5</b>	<b>Cours 4</b>	<b>9</b>
<b>6</b>	<b>Informations importantes</b>	<b>10</b>

# 1 Introduction

Trois parties pour le cours:

1. Concurrency
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 peut être lancé.

Si on ne met 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 peut 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 peut 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 peut 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 étiquetés 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 ( $\neg\wedge$ )
- $X\varphi$ :  $\varphi$  est vrai à l'instant suivant
- $\varphi U \psi$ :  $\varphi$  est vrai jusqu'à ce que  $\psi$  soit vrai.
- $\Box$ : globalement
- $\Diamond$ : un jour
- $\varphi_1 R \varphi_2$   $\varphi_2$  toujours vraie jusqu'à  $\varphi_1$  vraie (les deux peuvent l'être en parallèle)

T est un modèle de  $\varphi$  ss toute exécution de T valide  $\varphi$ .

Propriétés classiques:

- Safety:  $\Box_{\varphi}$
- Liveness:  $\Diamond_{\varphi}$
- $\Diamond\Box_{\varphi}$ : à partir d'un moment,  $\varphi$  sera toujours vraie.
- $\Box\Diamond_{\varphi}$ :  $\varphi$  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  $\varphi$ , on va montrer que le programme ne peut 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  $\omega$  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 peut 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 traditionnel est de complexité exponentielle 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 Cours 2

### 3.1 Introduction

Protocoles cryptographiques:

- Programmation concurrente
- Utilisation de primitives cryptographiques
- Description courte

Exemple de description de protocole cryptographique:  $A \rightarrow B: \text{aenc}(\langle A, K_{AB} \rangle, K_B^{Pub})$   
tel que:

- $A \rightarrow B$ : A envoie à B
- Après le :, message envoyé tel que:
  - aenc: chiffrement assymétrique
  - $K_{AB}$ : clé symétrique entre A et B
  - $K_B^{Pub}$ : clé publique de B

Ces protocoles sont utilisés dans divers domaines, tels que:

- Les connections sécurisées
- Le commerce en ligne, où ils assurent la non-répudation, c'est à dire l'impossibilité de nier l'implication dans l'échange
- Votes électroniques et administration électronique

Des exemples de protocoles utilisés sont TLS ou Kerberos.

Le vote électronique est contesté pour des raisons de sécurité, mais un système est utilisé dans certaines organisation, il s'agit de Belenios. Il s'agit d'un logiciel de vote électronique ou par correspondance, dont le protocole cryptographique a des propriétés de sécurités prouvées.

Analyse de protocoles:

- Analyse de failles logiques, qui sera le centre d'intérêt du cours
- analyse d'erreurs d'implémentations
  - Souvent implémenté en C pour des raisons d'efficacité
  - un protocole de quelques lignes peut être utilisé par des milliers d'individus

Il est important que la vérification puisse s'automatiser.

Pur ce cours, on supposera les protocoles incassable, et donc parfaits.

### 3.2 Cadre

Tout les protocoles sont publiés et n'ont rien de secret, pour des raisons de sécurité.

Il y a deux types d'ataquants possibles:

1. passif, qui est un observateur qui ne fait que écouter l'échange
2. actif, qui, en plus d'écouter l'échange, va intervenir dedans et intercepter voir altérer les messages

Vocabulaire:

- enc=chiffrement
- dec=déchiffrement
- a=assymétrique
- s=symétrique

Le chiffrement assymétrique est beaucoup plus lent( 100×), mais plus sécurisé que le chiffrement symétrique.

Il se peut que les fonctions de hachage soient utilisées hors de la cryptographie, pour pouvoir gérer plus facilement de gros objets. Par exemple, les graphes générés par Spin.

### 3.3 Modélisation

Il y a deux types de modèles:

1. Le modèle formel, qui sera étudié durant ce cours
  - Les messages sont des termes formels
  - L'attaquant est un ensemble de règles de calculs algébriques
  - La sécurité est donnée à partir des règles, et de leur capacité à déduire le secret
2. Le modèle calculatoire
  - Les messages sont des suites de bits
  - L'attaquant est une machine de Turing de capacité polynomiale
  - La sécurité est vérifié si il y a très peu de chances de trouver la faiblesse

Chaque modèle a des avantages et des inconvénients:

- Le modèle formel est simple et automatisable
- Le modèle calculatoire est difficile et à faire à la main

Dans certains cas, prouver la sécurité dans le modèle formel la prouve aussi dans le modèle calculatoire.

#### 3.3.1 Modèle Formel

Description d'un protocole:

- Session protocolaire
- Rôles A,B,...
- Règles d'envoi et de réception de messages
- Messages constantes, paires, chiffrés

Comment il est utilise:

- Les rôles instanciés par des agents, aussi appelés principaux
- Les sessions jouées en parallèle, de manière asynchrone
- L'attaquant est modélisé par l'environnement

Les constantes sont:

- Des clés  $K$
- Des nonces  $N$
- Des identités  $Id$
- Des chaines de caractères représentant le texte

Les opérateurs de ce modèle sont:

- Chiffrement symétrique  $senc(x,y)$ :  $x$  chiffré par  $y$
- Chiffrement assymétrique  $aenc(x,y)$ :  $x$  chiffré par  $y$
- $inv(K)$ : clé inverse de la clé  $K$
- $pk(K)$ : clé publique associée à la clé privée  $K$

Un agent du modèle peut être malhonnête.

On peut créer un protocole correcte dont l'implémentation n'est pas sûre.

## 4 Cours 3

Objectif: écrire un protocole, et avoir un algo qui dit si il marche bien ou pas.

Décrire la propriété de secret d'un protocole cryptographique pour un attaquant passif est un problème NP-complet si le nombre de sessions  $k$  est fixé à l'avance.



## 5 Cours 4

Les attaques montrées dans ce cours ne fonctionnent plus.  
En informatique, le fait que les données et les instructions partagent un modèle unique est problématique. Si le processeur voit l'octet 10010000, il le lit comme le chiffre 144, ou comme l'instruction nop?

La réponse à cette question change selon l'endroit où il est.

Petits programmes en C, sur des processeurs x86 32 bits, sur un linux ubuntu.

Syntax Intel: nom\_d'opération destination source

## 6 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