PLAN DE COURS

UQAC

8INF958 • Spécification, test et vérification

Contenu général

On réalise à peine que nos téléphones mobiles, ou même de simples jouets électroniques, sont maintenant plus puissants que l'ordinateur utilisé par Neil Armstrong pour se rendre sur la Lune. En parallèle à de rapides avancées techniques, les ordinateurs se sont progressivement faufilés, sous des formes et des tailles variées, dans presque toutes les tâches du quotidien. La confiance que l'on porte en tous ces systèmes informatiques les ont rendus indispensables et en ont fait, en quelque sorte, nos *gardiens*.

Cependant, la croissance des capacités des ordinateurs s'est accompagnée d'une croissance encore plus grande de la complexité de leurs programmes —et de la difficulté à prévoir toutes leurs exécutions possibles et à retracer toutes les erreurs imaginables. Dès le départ, le développement d'applications informatiques s'est largement appuyé sur la notion de *test* : on imagine des séquences d'actions ou d'entrées à soumettre au programme visant à révéler des potentielles erreurs de programmation. Cette technique, qui ne fournit aucune garantie formelle de qualité, commence à montrer ses limites à mesure que les systèmes informatiques gagnent en complexité.

Ce cours se veut une introduction aux concepts fondamentaux de la spécification, du test et de la vérification automatiques, qui se veulent une alternative aux tests classiques. On y verra comment spécifier le fonctionnement attendu d'un système au moyen de langages formels comme les automates, la logique propositionnelle et la logique temporelle. On étudiera ensuite les méthodes permettant de vérifier, par différents moyens, qu'un système donné satisfait bien la spécification fournie. Finalement, le cours sera l'occasion d'utiliser plusieurs logiciels de test et de vérification automatique.

Objectifs du cours

Au terme de ce cours, l'étudiant aura acquis les compétences suivantes :

- Identifier les limites de l'approche par tests dans le développement de logiciels
- Modéliser des systèmes simples au moyen de différents langages formels
- Comprendre et appliquer des algorithmes de vérification automatique (satisfaisabilité, runtime monitoring, model checking)
- Utiliser une variété d'outils automatiques pour vérifier qu'un système satisfait une spécification

Sujets abordés

Un aperçu du contenu de chaque leçon est donné ci-dessous. L'ordre de présentation peut être modifié selon les besoins.

1. **Introduction** Générations de langages de programmation. Notion de spécification, exemples, conséquences pour l'informaticien. Présentation du travail pratique 1.

2. **Les bugs** Définition. Exemples de bugs, méthodes de classification. Rapport de bugs, outils de

suivi des bugs (*bug trackers*). Présentation de la machine virtuelle du cours ; modalités du travail de session.

3. **Les tests** Notions de base de test. Types de tests classiques : tests de régression, tests unitaires, *fuzzing*, etc. Norme IEEE pour les tests ; certification ISTQB. Métriques de couverture. Limites de l'approche par tests. Automatisation des tests.

4. **Exprimer des contraintes** Notions de complexité : problème de décision, classes P et NP. Exemples de problèmes NP-complets en informatique. Logique propositionnelle : le problème SAT, les algorithmes DPLL et CDLL, les solveurs SAT. Application aux tests : assertions, tests concoliques. Les théories SATmodulo.

5. **Les annotations** Le typage en tant que spécification. Notion de pré/postcondition. Le design par contrat. Le *typestate checking* et la norme JSR 308. Le langage D. Rappels sur UML. La logique du premier ordre et les solveurs. Application : le langage OCL. Présentation du travail pratique 2.

6. **Les événements** Produire des événements système : l'instrumentation et les outils de journalisation. Programmation orientée aspect. Exemples de scénarios : tests manuels, appels de méthodes, contrats d'interface, analyse du comportement d'une application, détection d'intrusions. Notion de trace ; formats standard d'événements : XES, *protocol buffers*. Langages de spécification sur les traces : automates, expressions régulières, requêtes de bases de données. Patrons courants de propriétés.

7. **Le runtime monitoring** Algorithme paramétrique pour les automates. La logique temporelle LTL ; algorithme de monitoring. Les outils BeepBeep, Java-MOP et RV-Droid. Présentation du travail pratique 3.

8. **Vérification statique** Recherche de bugs par analyse syntaxique. Système de transition. Le problème de l'accessibilité. La logique temporelle CTL. Le model checking : différences avec énumération de traces, algorithme pour CTL. Bounded model checking. Retour sur les pré/postconditions : la logique de Hoare. Démonstration automatique. Application aux protocoles de communication : notion de réalisabilité.

9. **Encore plus d'automatisation** Apprentissage automatique de spécifications. Programmation par l'exemple. Synthèse de programmes à partir de spécifications. Application aux tests : génération de *stubs*.

10. **Pot-pourri de spécification, test et vérification** Notions de modèle, de satisfaisabilité et d'indécidabilité ; conséquences pour la vérification. Programmation lettrée (*literate programming*). Code porteur de preuve (*proof carrying code*.