HAI908I - Conférences Génie Logiciel - 3

Cours: 3
Date: 27/09/2022
LIRMM - Simon ROBILLARD

Notes :

VÉRIFICATION DÉDUCTIVE DE PROGRAMMES JAVA AVEC L'OUTIL KEY

Les bugs c'est grave ?

- OUI
- Dans certains cas, cela peut mener à la perte de données, de temps, d'argent et même de vie

Les tests -> Approche traditionnelle pour garantir la qualité du logiciel

- conception facile
- utiles à différentes étapes du logiciel :
  - développement
  - livraison
  - mises à jour
  -

Pour pouvoir tout tester, il faut être exhaustif
Exemple : Tester tous les inputs de la fct abs(n: int) :

- Si int est un entier 64bits : 2^64
- Avec des entiers multiprécision(Python), nb infini de possibilités

peux pas être exhaustif ? -> Couverture de code

- différent niveaux possibles : exécuter toutes les fct/instr/branches/conditions du prog
- exemple: critère MC/DC (critère de condition) pour les logiciels critiques
- impossible de décrire la couverture sans écrire le code / si le code change alors les tests changent

Les méthodes formelles :

On abstrait notre programme

On prend la spec du programme (cahier des charges en langage semi formel) on formalise la spec et on modélise le programme
On obtient alors des propriétés formelles et un modèle sémantique
On vérifie sans exécuter le programme. On peut utiliser des techniques mathématiques, car il s'agit d'une représentation mathématique

Il y a différents types de méthodes formelles :

- Différents modèles :
  - abstrait ou détaillé
  - impératif ou déclaratif
  - séquentiel ou distribué
- Différents types de propriétés :
  - propriété fonctionnelle(ce que fait le prog donc tests unitaires), temporelle(quand le logiciel fait telle chose), sécurité(est ce que mon prog transmet des données qu'il ne devrait pas)....
  - spec du domaine d'appli
  - spec du paradigme de programmation(ex: concurrence, calcul sur les flottants...)

Technique de méthodes formelles :

- Le Typage : le typage permet de savoir exactement ce que le code doit faire / le langage de spec et de prog sont les mêmes / langage Coq par exemple -> Limite : Long et demande un haut niveau en logique
- Model-Checking : Représenter les états/transitions de notre système sous la forme d'un graphe et utiliser un algo de parcours pour vérifier les propriétés / Permet de spécifier le comportement temporel / Le processus est automatique / Niveau d'abstraction très haut -> Limite : Le graphe est immense
- Raffinement : Partir de la spec (abstrait) et la transformer pour obtenir un code exécutable / Étapes successives permet de limiter la difficulté -> Limite : chaque transformation doit être prouvée
- **Vérification déductive**

Vérification déductive

Système = programme
propriété à vérifier = relations entre les états d'entrées et ses sorties
Le but est de prouver qu'un programme avec certaines pré conditions, donne bien certaines post conditions

Spécifier des programmes Java :

- JML (Java Modeling Language)
  - commentaires java avec une syntaxe spéciale
  - permet de définir des contrats pour chaque méthode
    - Pré conditions (arguments passés en entrée)
    - Post conditions (valeurs retournées)

-> Caractéristiques :

Les conditions sont des formules de logique du premier ordre
quantificateurs comme \forall, \exists ...
requires : mot clé pour les pré conditions
ensures : mot clé pour les post conditions

La logique dynamique :

- modalité box [p]P. -> le prog p termine dans un état qui satisfait P
- modalité diamond. <p>P. -> si p termine, alors il le fait dans un état qui satisfait P

On peut créer une formule qui caractérise le programme grâce à la spec et notre prog
La logique dynamique est une extension de la logique du premier ordre

La spec ne suffit pas si on a des boucles : il faut rajouter des invariants de boucle
Stratégie pour trouver des invariants :

- utiliser des motifs récurrents
- partir de la post condition et la généraliser jusqu'à ce que ce soit un invariant