Génération de code certifié pour des applications orientées objet

Spécification, raffinement, preuve et détection d'erreurs

http://geccoo.lri.fr

Christine PAULIN

Journées ACI Sécurité Rennes, 12-12-2003

Objectifs

Outils de développement de code orienté objet certifié

- Cartes à puce (JavaCard)
- Terminaux
- Code critique

Couvrir la chaîne de développement

- Spécification des propriétés de sécurité
- Traduction en terme de propriétés logiques
- Développement modulaire : raffinement et composition
- Résolution automatique des obligations de preuve
- Détection d'erreurs : simulation et test

Les équipes et savoir-faire (1/2)

Équipe TFC LIFC, Besançon, F. Bellegarde

- Génération de tests à partir de spécification (BZ-Testing-Tools)
- Système de contraintes

Projet CASSIS LORIA-LIFC, Nancy-Besançon, S. Ranise

- Preuve de programmes (HaRVey)
- Propriétés de sécurité

Projet Everest INRIA-Sophia Antipolis, M. Huisman

- Plateforme JavaCard
- Programmes Java annotés en JML (CHASE, Jack,...)

Les équipes et savoir-faire (2/2)

Projet LogiCal PCRI (INRIA-LRI-LIX), Saclay, C. Paulin

- Assistant de Preuve Coq
- Outils de génération d'obligations de preuve (Why, Krakatoa)

Équipe VaSCo LSR, Grenoble, M.-L. Potet

- Raffinement, composition (B)
- Génération de code sûr pour cartes à puce

Les bases : Programmes Java annotés en JML

JML: Java Modeling Language (Gary Leavens, Iowa)

- Commentaires spéciaux
- Langage logique : sous-ensemble pur de Java + extensions logiques `\forall`, mémoire `\old` ...
- Traitement possible à différents niveaux
  - documentation `jmldoc`
  - génération de code défensif `jmlc`
  - génération de pilotes de test `jmlunit`
  - vérification automatique `ESC/Java`
  - vérification interactive `LOOP, JIVE, Jack, Krakatoa`
- Langage ouvert à des extensions

Exemple de programme Java annoté en JML

```
class Purse {
    //@ public invariant balance >= 0;
    int balance;

    /*@ public behavior
      @   requires s >= 0;
      @   modifiable balance;
      @   ensures s<=\old(balance) && balance==\old(balance)-s;
      @   signals (NoCreditException)
      @           s>\old(balance) && balance == \old(balance);
      @*/
   public void withdraw(int s) throws NoCreditException
        if (balance >= s) balance -= s;
        else throw new NoCreditException();
```

Les outils Jack et Krakatoa

Jack (J.-L. Lanet, L. Burdy, Gemplus-INRIA)
Krakatoa (C. Marché, LogiCal)

Entrée : Programme Java annoté en JML
Génération automatique : Modélisation (B ou Coq) du programme (classes, mémoire...)
Analyse statique Interprétation du programme et de sa spécification comme un programme impératif avec assertions
Objet = adresse
Champ d'objet : tableau indicé par les adresses
$v.t = e \mapsto t[v] = e$
Génération d'obligations de preuve à prouver interactivement ou automatiquement Coq, B, Simplify, haRVey...

Les enjeux

Expressions de propriétés de sécurité

- absence d'erreurs de programmation
  (`NullPointerException`, `ArrayOutofBoundException` ...)
- respect de cycles de vies (applets, transactions, allocation)
- contrôle d'accès

Faciliter l'écriture des spécifications et du code

- découpage modulaire, raffinement
- génération de pré-conditions, invariants
- test et simulation de spécifications

Automatiser les preuves

- Choix d'une théorie appropriée
- Stratégie adaptée

A propos de spécification

Exceptions Choix de spécification : comportement normal
contraint par précondition ou spécification générale

Cycles de vie M. Pavlova (DEA, Sophia-Antipolis)
Utilisation de variables ghost de JML pour spécifier des états du système et contraindre l'enchainement des méthodes.
Spécification défensive.
Annotations automatiques.

Automatisation versus Interaction

Besoins

- Obligations : formules du premier ordre, théorie des tableaux, arithmétique (Simplify, haRVey)
- Modèles d'ordre supérieur (B, Coq)

Coopération

- Génération de contre-exemples
- Simplifications
- Recherches d'invariants
- Méthodes sûres vs méthodes approchées
- Exploiter les échecs de la démonstration automatique
  code défensif, objectifs de test

Organisation du projet

- Des équipes de compétences complémentaires
- Développement d'outils avancés
- Phase de mise en place :
  communication entre outils, définition d'études de cas pertinentes