# *Vérification d'un composant Java*

Julien Charles

Jeudi 12 mai 2005

# Introduction

- But : trouver des moyens de vérifier un composant système Java
- D'habitude on vérifie plutôt un modèle.
- On vérifie l'implémentation en annotant le composant et en générant des obligations de preuves à partir de ces annotations.
- Le cas choisi : le vérificateur de bytecode Java. $\longrightarrow$ Un des composants les plus importants en Java.

# Le vérificateur de bytecode

Exécuté sur chaque méthode pendant leur chargement.
Un algorithme non trivial :

- des thèses ont été écrites sur la preuve des vérificateurs de bytecode.
- terminaison : quand on a trouvé le point fixe.
  $\longrightarrow$ Utilisation d'un prouveur interactif.

# Java Modelling Language (JML)

Un langage de spécifications de Java :

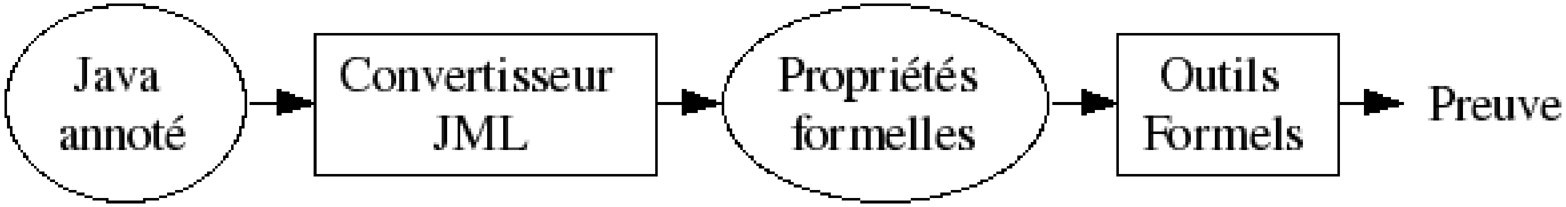
- Vérifications dynamiques (assertions)
- Vérifications statiques (logique de Hoare)

On peut exprimer les préconditions, les postconditions et les invariants.
Plusieurs outils :

- ESC/Java
- Krakatoa
- Jack

# Jack

Jack génère des obligations de preuves à partir d'un programme Java annoté en JML.




Il permet d'utiliser différents prouveurs.
$\longrightarrow$ Il n'avait pas de sortie vers un prouveur interactif.

# Un plugin Coq

Coq est un langage de preuve interactif fondé sur le calcul des constructions inductives.

- des scripts de preuves (avec les tactiques)
- langage de macros de tactiques (Ltac)

Le plugin traduit l'AST de Jack vers Coq :

- Correction d'erreurs

Automatisations

# Conclusion

- Jack a maintenant un plugin Coq.
- J'ai effectué une grande partie de la preuve du vérificateur.
- J'ai developpé des techniques pour automatiser les preuves en Coq.
- Perspectives : les automatisations, la JVM modulaire, le multi-threading.