Model Checking of Multi-Applet JavaCards

Lars-Åke Fredlund
Swedish Institute of Computer Science

Joint work with

Gennady Chugunov
Swedish Institute of Computer Science

Dilian Gurov
Royal Institute of Technology
Problem Statement

- Analyse inter-applet method call patterns
- Motivating example due to Lanet et al:

```
logFullgetTrsgetTrsgetBalance
PurseAirFranceRentACar
```

```
VerifiCard Context

WP4: Analysis of Applet properties on the byte code level
INRIA Sophia-Antipolis & SICS

A common card model:

- A set of applets consisting of methods with program points
- Execution steps are:
  - Methods calls and returns
  - Intra method control flow
- Data is completely abstracted away
Barthe, Gurov and Huisman (FASE’02): a compositional program model

- Each applet has its own control stack of program points: \( \langle A, P_0 \cdot \ldots \cdot P_n \rangle \)

- A compositional operational semantics for deriving global transitions \( A_1 \parallel \ldots \parallel A_n \rightarrow \) from local ones \( A_i \rightarrow \)

- A compositional proof system (Gentzen style, logic the modal \( \mu \)-calculus)

\[
(1) \text{AirFrance} : \phi_A \\
(2) \text{Purse} : \phi_P \\
(3) \text{RentACar} : \phi_R \\
(4) X_A : \phi_A, X_P : \phi_P, X_R : \phi_R \vdash X_A \parallel X_P \parallel X_R : \phi
\]

\[\text{AirFrance} \parallel \text{Purse} \parallel \text{RentACar} : \phi\]
Our Verification Approach

- Application of model checking techniques by combining existing tools to achieve “push-button” verification
- Useful for checking individual applets
  - AirFrance: $\phi_A$
- Generally for checking closed systems
  - $\text{AirFrance} \parallel \text{Purse} \parallel \text{RentACar}: \phi$
  
  but not for open ones
Overview of Method

JavaCard byte code class → INRIA Rennes Call Graph Tool → Call-graph model → Translator to pushdown system → Pushdown system → LTL specification → Moped model checker (Schwoon)
Call Graph Example
Call Graph Construction

- Method call graphs produced by INRIA Rennes JVM analysis tool (Jenset et al) based on Soot
- Call graphs abstract away from data dependencies
  Branching constructs introduce graph nondeterminism
- Construction is class based
  Applet instances cannot be distinguished
- Class based (package based) analysis is a good fit with the JavaCard firewall mechanism
Generating Call-Graphs for JavaCard

The adaptation of the call-graph construction tool for JavaCard mostly concerns information collection.

- For each applet class (inherits from Applet class) the call graphs for methods install, select, deselect, process and getShareableInterfaceObject are generated.

- For each applet class the call-graphs for methods callable using sharable interfaces are generated.

```java
package purse.Loyalty;

...

public interface LoyaltyPurseInterface extends Shareable {
    void grantPoints (byte[] buffer);
}```
Pushdown System

- Pushdown systems are a natural execution model for programs with recursion
- A pushdown system (PDS) is a tuple

\[ P \triangleq \langle P, \Gamma, \Delta \rangle \]

(i) \( P \) is a finite set of control locations
(ii) \( \Gamma \) is a finite set of stack symbols
(iii) \( \Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*) \) is a finite set of rewrite rules
      of the shape \( \langle p, \gamma \rangle \rightarrow \langle q, \sigma \rangle \)

- A run of \( P \) is a sequence \( \rho = \langle p_0, \sigma_0 \rangle \langle p_1, \sigma_1 \rangle \langle p_2, \sigma_2 \rangle \cdots \) such that for all \( i \), there is a rule \( \langle p_i, \gamma \rangle \rightarrow \langle p_{i+1}, \sigma \rangle \) and \( \omega \in \Gamma^* \) such that \( \sigma_i \equiv \gamma \cdot \omega \) and \( \sigma_{i+1} \equiv \sigma \cdot \omega \)
Translation of Call-Graphs to PDSs

Translation of call-graphs to pushdown systems is easy. A single control location $c$ is used and the stack symbols encode JavaCard program points.

A common abstraction is to collapse API calls.

A method call from program point $p$ to method $m$ becomes the PDS rule:

$$\langle c, p \rangle \rightarrow \langle c, m \cdot p \rangle$$

A method return from program point $p$ becomes:

$$\langle c, p \rangle \rightarrow \langle c, \epsilon \rangle$$
Correctness Properties

- Linear Temporal Logic used to specify properties for model checking:
  \( \neg \phi, \phi \land \psi, \phi \lor \psi, \text{true, } \text{false} \)
  \( X \phi \) (\( \phi \) holds in the next configuration)
  \( \phi U \psi \) (\( \phi \) holds until \( \psi \) eventually holds)
  \( \phi W \psi \) (\( \phi \) holds until \( \psi \) holds)

- The basic predicates are program points (\( p \)), classes (class \( c \)) and packages (package \( p \))

- The satisfaction relation of a formula \( \phi \) is defined relative to a run, \( r \models \phi \)
  Example: \( \langle c_0, p \cdot \sigma \rangle \langle c_1, \sigma_1 \rangle \ldots \models p' \) iff \( p \equiv p' \)

- The judgment \( m \vdash \phi \) expresses the claim that every run \( r \) of the PDS from the configuration \( \langle c, m \rangle \) satisfies \( \phi \)
Specification Patterns

- Specification patterns are used to write more readable properties and to provide the link to compositional verification ($\mu$-calculus)

- Examples:

  - Eventually $\phi$ $\triangleq$ $\text{true } U \phi$
  - Always $\phi$ $\triangleq$ $\neg (\text{true } U \neg \phi)$
  - Never $\phi$ $\triangleq$ Always $\neg \phi$

  - Within $m\phi$ $\triangleq$ $m \vdash \phi$
  - $a$ CannotCall $m$ $\triangleq$ Always (package $a \Rightarrow \neg (X m))$
  - $m_1$ NeverTriggers $m_2$ $\triangleq$ Within $m_1$ (Never $m_2$)
  - $m_2$ After $m_1$ $\triangleq$ (Never $m_2$) $\forall m_1$
  - $m_1$ Excludes $m_2$ $\triangleq$ Eventually $m_1 \Rightarrow$ Never $m_2$
Example Revisited

With these specification patterns the example

\[\text{Purse} \rightarrow \text{AirFrance} \rightarrow \text{RentACar}\]

violates the correctness property

\text{Within AirFrance.logFull CannotCall RentACar Purse.getTrs}\]
Could not find an efficient $\mu$-calculus based model checker

Instead: Moped for LTL (Schwoon)

Approach: a Büchi automaton is built for the negation of the formula and combined with the original PDS into a “Büchi” PDS; check if there is an accepting run

Time complexity $O(p^2b^3)$ where $p$ is the size of the pushdown system and $b$ is the size of the Büchi automaton; space complexity is $O(p^2b^2)$.

Diagnostics: reduced PDS exhibiting the error

Encoding of basic properties via regular expressions
In Practice

A concrete example (a modified purse from the SUN JavaCard development toolkit):

```
package purse.LoyaltyA
package purse.Purse
class LoyaltyA
class Purse
interface PurseLoyalty
method bonusPointsToPurse

package purse.Loyalty
package purse.LoyaltyB
class LoyaltyB
class Loyalty
interface LoyaltyPurse
interface LoyaltyLoyalty
method grantPoints
method grantLoyaltyPoints
extends
extends

package purse.LoyaltyA
class LoyaltyA

package purse.LoyaltyB
class LoyaltyB
```
Example Properties

- Property A: **Calls to grantPoints are not transitive**
  
  For all loyalty applets $L$ and $L'$, a call to $L$.grantPoints never triggers a call to $L'$.grantPoints

  \[
  \text{loyaltyA}\text{.grantPoints NeverTriggers loyaltyB}\text{.grantPoints}
  \]

- Property B: **An object constructor is not called from the process method**
  
  Any constructor invocation is recognized by the regular expression
  \[
  \text{Constructor} \triangleq .*\backslash..\backslash.<\text{init}>_.\star
  \]
  
  Checking:

  \[
  \text{Within purse.Purse.process Never Constructor}
  \]
Example Results

- Example size approx. 1400 lines of Java code
- About the same number of rewrite rules with API abstracted away
- Call graph generation approx. 15 seconds
- Model checking each property takes less than a second
Conclusions

- Automatic and light-weight verification techniques attractive to end users
- Possible to implement on-card in the near future?
- Is abstracting away all data dependencies too coarse an abstraction?
- Work in progress; first polished prototype to be delivered during autumn
- Paper describing initial experiments and results will be presented at CARDIS’02