Opportunities and challenges for formal specification of Java programs

Joe Kiniry Erik Poll

LOOP group

University of Nijmegen

supported by NWO and the EU IST project VerifiCard

Definitions for this talk

Component

= Java object/class/package/API

Trusted

= says *something* it does

with JML specification (contract)

that we can have some confidence in

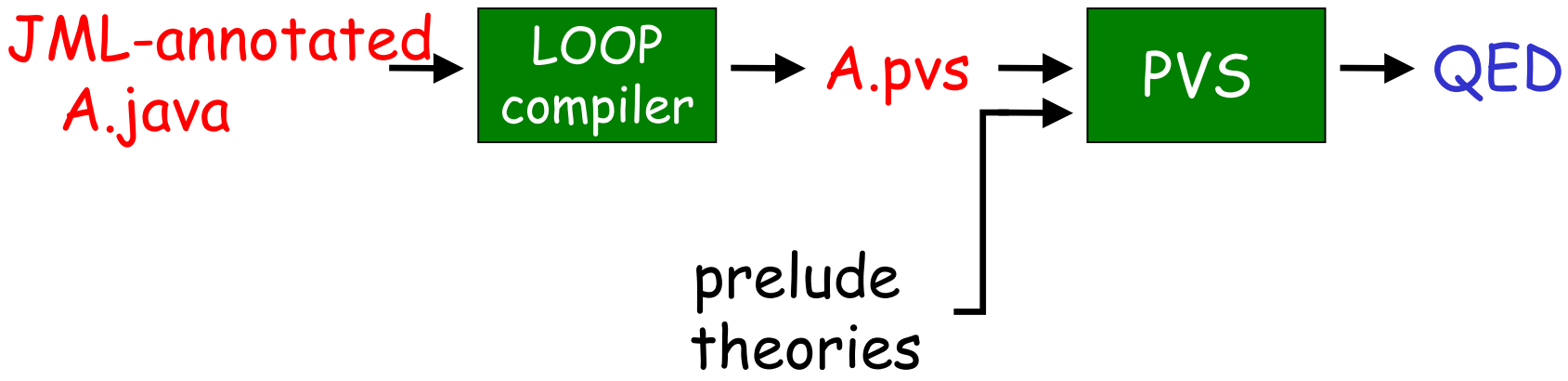
Overview

1. Java verification using the LOOP tool
2. JML
3. JavaCard
   - where the high road meets the low road ?
4. Conclusions

Java verification using the LOOP tool

The LOOP project

- Interactive verification of sequential Java programs using theorem prover PVS



Evolution of the LOOP project

1. Denotational semantics of sequential Java
   - (symbolically) executable in PVS
2. Hoare logic
   - set of proven lemmas
   - incl. abrupt termination, side-effects in expressions, ...
3. WP calculus
   - no need to supply PVS with intermediate predicates

Ongoing work

- covering more JML constructs
- more automation in proofs
  - PVS tactics
  - suitable collections of (generated) lemmas

  to reduce user-interaction
- but lack of good feedback when proofs fail may remain a fundamental limitation

JML
Java Modeling Language

JML (www.jmlspecs.org)

- Assertion language for Java - based on Eiffel, Larch... - by Gary Leavens et al.
  - pre- and (exceptional) postconditions
  - invariants
  - assignable aka modifies clauses
  - model variables and refinement
  - using Java boolean expressions extended with \forall, \exists, ==>, \old, \result, ...
  - pure, non_null, \nonnullelements, etc.

Example: exceptional postconditions

```
/*@ requires   buffer != null &&
               offset >= 0 &&
               length >= 0 &&
               offset + length <= buffer.length;
    assignable triesRemaining;
    ensures    ...
  @*/
public boolean checkPIN(byte[] buffer,
                        short  offset,
                        short  len)
```

Example: exceptional postconditions

```
/*@ requires    buffer != null;
    assignable triesRemaining;
    ensures     ...;
    signals   (ArrayBoundsException)
                   offset < 0 || length < 0
                 || offset + length > buffer.length;
    ...
  @*/
public boolean checkPIN(byte[] buffer,
                        short  offset,
                        short  len)
```

Example: more detailed spec

```
/*@ requires   buffer != null;
    assignable triesRemaining;
    ensures    ...;
    signals  (ArrayBoundsException)
                  offset < 0 || length < 0
                || offset + length > buffer.length;
    signals  (PINException)
                  \old(getTriesRemaining()) <= 0;
  @*/
public boolean checkPIN(byte[] buffer,
                        short  offset,
                        short  len)
```

Tools for JML

- typechecker, runtime assertion checker, jmlunit
  (Gary Leavens et al, Iowa State)
- jmldoc
  (Gary Leavens et al and David Cok, Kodak)
- ESC/Java – automatic "extended" static checking
  (Rustan Leino et al, Compaq)
- Chase – automatic static checking of modifies clauses
  (Nestor Cantaño, INRIA Sophia-Antipolis)
- LOOP – interactive verification
  (Bart Jacobs et al, Nijmegen)
- Daikon invariant generator
  (Michael Ernst, MIT)

Call for cooperation

JML is an open, cooperative project.

People interested in developing or using assertion languages and tools are welcome to join.

Having compatible specs & tools will greatly benefit everyone!

JavaCard

JavaCard

- Superset of a subset of Java for programming smart cards
  - subset: no reals, no threads, no strings, no gc, very restricted API
  - superset: API for communication, persistent & transient objects (EEPROM & RAM), transactions
- Ideal target for formal methods: small programs whose correctness is critical

JavaCard case studies

- JML specs for JavaCard API (48 classes)
  [www.verificard.org]
- using ESC/Java on electronic purse (9000 lines)
  [Nestor Cataño & Marieke Huisman, FME'02]
- using LOOP tool on Decimal implementation
  [Cees-Bart Breunesse, Joachim vd Berg, Bart Jacobs, AMAST'02]
- using ESC/Java on file system applet (3000 lines)
  [under NDA]
- using LOOP tool on EMV case study (700 lines)
  [under NDA]

Potential impact

- Some transfer to industry is starting...
  - One smartcard manufacturer has developed own "extended static checker" integrated with the IDE that their developers use
  - VLSI design company using for specification and validation of CAD tools
- Common Criteria – new ISO standard for IT security – requires formal methods for highest levels of evaluation

Some remarks...

Embarrassing open problem: pointers and call semantics

- Java, C, C++, ... offer no form of encapsulation or information hiding
- Some mechanism for specifying and proving/enforcing alias confinement and preventing representation exposure is needed!
- Upcalls and callbacks
- Specification and reasoning is terrifically difficulty – still no good answers

Who cares about trusted components ?

- In some niche markets – eg JavaCard – people may be interested in "trusted components"
- But getting people really interested will always require a lot of effort
- Telling them it is important is not enough, showing them the benefits and results is necessary
- It must be an economic necessity

Problems with specification (and solutions)

- Functional specification - saying what a program does - is hard
  - Most specs will have to be incomplete
  - Training and methodological changes necessary
  - Social and psychology issues are most critical
- Starting with basic safety properties ("lightweight specs"), eg invariants, and/or

```
requires ...
ensures  true;
signals  (Exception) false;
```

can be a good way to get started

Problems with specification (and solutions)

- Extra complexity, maintenance, and training must have payoffs
- Quality tool support is essential

- Specification language size and complexity continues to grow (even experts do not know corners)
- Introduce methodology and tool support to help developer write and sanity check specs

Problems with specification (and solutions)

- Public-only specifications is insufficient
- Language visibility notions are improvement, but semantics are unclear and they are often too coarse-grained
- Organizational-centric visibility (exposure between teams, Q/A, clients, etc.) is being demanded

Key opportunity

- Assertion languages - such as JML - are the most promising way of getting formal methods (thus trusted components) used and developed in industry
  - easy to learn:
    - little new syntax
    - no model other than the source code needed
  - can be introduced incrementally
  - runtime assertion checking, static checking, and unit test generation is mature and usable technology