other plugins

Annotation generation

Annotation propagation

WP at bytecode level

Annotation generation 1/2

- Based on two facts:
  - One JML drawbacks is the amount of specification that has to be written
  - Some annotations can be automatically generated
- After a first study, annotations are generated preserving from some NullPointerException et IndexArrayOutOfBoundsException, some modifies are also generated.

Annotation generation 2/2

- The WP is used to calculated the annotation. Rather to generate lemmas, generated lemmas are inspected to retreive some potential requires and modifies clause.
- The first results are quite good and we aim to complete the process to generate more annotations letting to developper the core work.

Annotation propagation

- Annotation generation works at method level
- What we call annotation propagation works at application level.
- The aim is to propagate through method call some precondition and postcondition

Annotation propagation

- This allow to add automatically some annotations to improve the development process:
  - If a method requires a precondition that is ensured by one of its ancestor in the call graph, the precondition can be propagated through the call graph to the method that ensure it.

Annotation propagation

- This allows also to prove some properties and moreover some security properties :
  - If a property can be expressed with some precondition and postcondition on some method. Then one can propagate those annotations through the application and check that they are fulfill.

Annotation propagation

- In practice:
  - From a prop file with some core annotations
  - Propagate the core annotation in the application
  - Run Jack to check if the propagated annotations are correct

Annotation propagation

- Once again, the propagation is done by a *simplified* WP calculus.
- The technic is not complete: wrong annotations can be generated due to the *simplified*, but sound since the annotations are verified by the tool after propagation.

Annotation propagation

- A future step to verify security properties with Jack is to allow to describe the property with an automata with potentially some annotations and then to translate this automata into core annotations to be propagated
- Current trainee on this subject

Bytecode

- Compiler from JML to BCSL
- WP on Java bytecode annotated with BCSL
  - Generating JPOL verification condition
  - Benefits from prover plugins
- Integrated in last Jack release
- Mariela Pavlova Phd Thesis

Why validate at bycode level ?

- To give a client a way to check properties on code produced by a producer
-

Framework

- Client expresses properties at source level
- Producer writes the application and validates it in regards of the client requirements
- Producer compiles its application and add the BCSL annotations
- Then the code is transfered to the client that can check that it fullfils its requirements

EVEREST

Framework 2

- Client can express requirements at two levels:
  - With an annotated interface that has to be implemented by the code producer
  - With some restricted access to an API that the code producer has to use
- In most cases, code producer has to insert some annotations to prove that it fullfils the requirement

EVEREST

JML Compiler 1/2

- We furnish a JML compiler that allows to insert the annotations in a class file
  - Annotations are added as special attributes (do not interfer with the JVM)
    - Class invariant
    - Method specifications
  - Annotations are linked with the Constant Pool created by the Java compiler

JML Compiler 2/2

- For in-method annotations (assert, loop specification), one use the *debbuger* attributes to retreive the index in the bytecode corresponding to the source line
  - Special difficulty with imbriqued loop starting at the same line. Solution : loop are ordered in the source.

Bytecode WPC

- *Similar* to Java WPC but
  - One has to make the control graph acyclic using loop invariant
    - Restriction : no re-entrante loop
  - For the rest, the instruction are treated as they are at source level: method call, field assignment, exception throw
  - Jsr are *inlined*
    - Restriction: Mariela ?

Bytecode WPC

- Deal with a stack and a stack pointer but the frame is modelized by the set of fields
-

- The formula resulting from the WP is translated into JPOL
  - Allows to edit the lemma using the Jack lemma viewer
  - Allows to prove the lemmas
    - With the automatic provers
    - With the interactive provers

Conclusion

With the same framework, one can work at source or a bytecode level