DD2460 Software Safety and Security

Introduction to VeriFast for Java

Dilian Gurov

1

The VeriFast Tool

- VeriFast is a modular, sound program verifier for sequential and concurrent C and Java programs
- If VeriFast reports that a program is correct:
  1. does not raise `NullPointerException` or `ArrayIndexOutOfBoundsException`
  2. does not contain **data races** (memory safety)
  3. **assertions** and **method contracts** (pre- and postconditions) are respected in every program execution

2

Object Oriented Programs

Verification of OO programs is tricky because of a number of reasons, most notably:

- **aliasing**: problematic for modular reasoning
- **inheritance**: dynamic call resolution

Concurrency adds to the complexity:

- **race conditions**, **deadlocks**

VeriFast is based on **modular verification**, **symbolic execution** and **separation logic**

3

Modular Verification

VeriFast performs **modular verification**, which is crucial for achieving **scalability** of verification:

- every method is specified with a method **contract** (a pre- and a postcondition) and is verified separately
- when verifying a method, method calls are replaced by the respective method contracts:
  - the precondition is asserted, and then
  - the postcondition is assumed

4

Symbolic Execution

- VeriFast symbolically executes the method body, starting in a symbolic state that represents an arbitrary concrete state that satisfies the precondition
- A symbolic state consists of:
  - the symbolic store: maps variable names to symbolic values
  - the symbolic heap: a multiset of heap chunks
  - the path condition: a list of formulas

5

Symbolic Execution

- VeriFast explores all (feasible) paths from the method's entry point to a an exit point of the method (return statement or uncaught exceptions)
- At method exit points VeriFast verifies that the respective symbolic state satisfies the postcondition to the method

6

Separation Logic

- To handle the problems of aliasing and data races due to shared memory concurrency, VeriFast employs **separation logic**
- the program memory (heap) is conceptually broken down into separate chunks (or more precisely, **permissions** to access chunks) that are passed during method calls and returns, or distributed between concurrent threads

7

Permissions

- Memory safety is guaranteed by explicitly specifying the heap chunks that are used (required/ consumed and ensured/produced) at a given place of the program
- Permissions themselves can be broken down into so-called **fractional permissions** to allow multiple read access to memory
- Permissions are represented by fractions between 0 and 1, only 1 giving write access

8

Data Races

- A data race occurs when two threads concurrently access the same memory location and at least one of these accesses is a write access
- VeriFast prevents data races by enforcing the system invariant that for each memory location, the total sum of the fractions of the permissions is at most 1

9

Data Abstraction

- One can define own **predicates** to be used in specifications
  - to make specifications more concise and abstract
  - to capture object invariants
  - to **encapsulate** parts of the heap, for example the private fields of objects (to get access to its parts one has to ”open” the predicate)
- Produced/ensured predicates create symbolic values during symbolic execution

10

Inductive Data Types

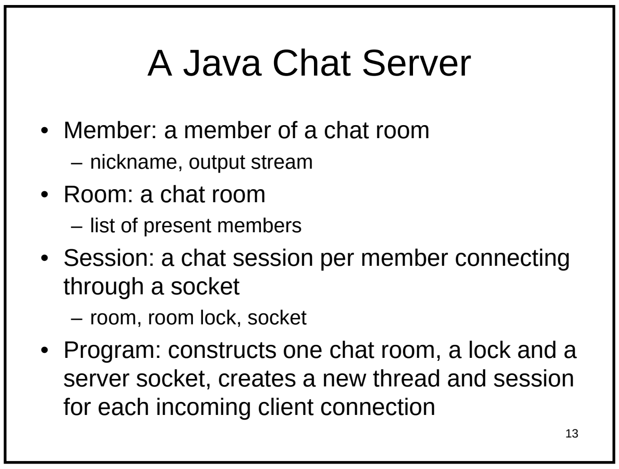
- Can be used to present abstract views of programmer-defined data types
- Specify such data types by relating (with a predicate) the data type to its abstract view
- for example, lists can be used as abstract views of stacks

11

More Advanced Features

- Fixpoint Functions
  - definitions of functions over inductive data types
  - follow the principle of structural induction
- Lemmas
  - contracts for pure functions (in specifications)
- Inheritance
  - contracts for Java interfaces
  - matched against implementation contracts

12



A Java Chat Server

- Member: a member of a chat room
  - nickname, output stream
- Room: a chat room
  - list of present members
- Session: a chat session per member connecting through a socket
  - room, room lock, socket
- Program: constructs one chat room, a lock and a server socket, creates a new thread and session for each incoming client connection

13