Unpredictable Behavior of Concurrent Unit Tests

Current Frameworks JUnit, TestNG Are Broken!

JUnit, TestNG, etc. were not designed with concurrency in mind
-- The criteria for success and failure are incorrect for concurrent programs
-- The design promotes writing tests that succeed by default even when they should fail

Writing good concurrent unit tests is as hard as writing good concurrent programs
Problem 1: Exceptions are not automatically detected and considered a test failure.
-- The program fails in an auxiliary thread, but the unit test nonetheless succeeds, even long after the other thread has failed.
Solution: Use a default exception handler

Problem 2: The test does not have to wait for all auxiliary threads to terminate.
-- The program fails in an auxiliary thread, but the primary thread completes the unit test so quickly, that failure goes unnoticed.
Solution: Wait for all spawned auxiliary threads

Concurrency-Aware Unit Testing Framework

Inserting Delays and Scheduled Replay

Custom class loader rewrites class files as they are loaded
-- Modified class files will run on any Java VM
-- Portable and open-source
-- Bytecode can be dynamically compiled by an embedded JIT compiler
-- Faster than interpretation
-- When an atomic block ends, a delay may be inserted, forcing the program into a different schedule

Lightweight Checking of Concurrency Invariants Using Java Annotations

Java Annotations to Specify Invariants
-- Declare which classes and methods should be or may only be executed by certain threads
-- Simple to introduce into existing large projects
-- Just annotate parts of the project, no need for a complete rewrite
-- Dynamic checks to maintain invariants are automatically inserted
-- Some checks are also performed statically

A Way to Specify API Threading Discipline

Augmenting Type Systems Using Annotations

By introducing annotations describing permitted threading behavior, we have in effect augmented Java’s type system.
-- Syntax problems: C <: D according to Java, but not by our checking
-- Formulation of the utility’s subtyping relation based on Featherweight Java
-- Annotations can be used to augment Java’s type system to get additional guarantees; material for future research.

Practical Tools for Testing Concurrent Programs

Practical Tools for Concurrency

(1) A concurrency-aware extension of JUnit
(2) Lightweight dynamic and static checking of concurrency invariants using Java annotations
(3) Execution with short delays inserted at critical places to test different execution schedules
(4) Recording of program execution and replaying according to a predetermined schedule

Concurrent Unit Tests

Current tools: JUnit, TestNG, Ant
Prevent old bugs from reappearing
Tests exhibiting fixed bugs are added to test suite
Tests must pass before code changes are committed

Some checks are also performed statically

Exception!

Java Annotations to Specify Invariants

-- Simple to introduce into existing large projects
-- Just annotate parts of the project, no need for a complete rewrite
-- Some checks are also performed statically

A Way to Specify API Threading Discipline

Augmenting Type Systems Using Annotations

By introducing annotations describing permitted threading behavior, we have in effect augmented Java’s type system.
-- Syntax problems: C <: D according to Java, but not by our checking
-- Formulation of the utility’s subtyping relation based on Featherweight Java
-- Annotations can be used to augment Java’s type system to get additional guarantees; material for future research.

Practical Tools for Testing Concurrent Programs

Practical Tools for Concurrency

(1) A concurrency-aware extension of JUnit
(2) Lightweight dynamic and static checking of concurrency invariants using Java annotations
(3) Execution with short delays inserted at critical places to test different execution schedules
(4) Recording of program execution and replaying according to a predetermined schedule

Concurrent Unit Tests

Current tools: JUnit, TestNG, Ant
Prevent old bugs from reappearing
Tests exhibiting fixed bugs are added to test suite
Tests must pass before code changes are committed

Some checks are also performed statically

Exception!

Java Annotations to Specify Invariants

-- Simple to introduce into existing large projects
-- Just annotate parts of the project, no need for a complete rewrite
-- Some checks are also performed statically

A Way to Specify API Threading Discipline

Augmenting Type Systems Using Annotations

By introducing annotations describing permitted threading behavior, we have in effect augmented Java’s type system.
-- Syntax problems: C <: D according to Java, but not by our checking
-- Formulation of the utility’s subtyping relation based on Featherweight Java
-- Annotations can be used to augment Java’s type system to get additional guarantees; material for future research.