Summary

Sink States:0(0×10^0)

Table 1: Pulse Analysis Summary

Classes	Methods	States	Unsatisfiable Clauses	Unreachable States	Possible concurrent Methods	Total. no. of pairs	No. of concurrent pairs	Percentage of concurrent Methods
JGFCryptBenchSizeA	2	1	0	0	1	3	1	33
JGFInstrumentor	3	1	0	0	2	6	2	33
JGFCryptBench	6	1	0	0	5	21	7	33
IDEATest	9	1	0	0	8	45	21	47
JGFTimer	3	1	0	0	2	6	2	33
Total Classes=5	23	5	0	0	18	81	33	41

Contents

1 JGFCryptBenchSizeA

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFCryptBenchSizeA	$\surd$
main	$\surd$

Table 3: State Transition Matrix

	alive
alive	$\uparrow$

Table 4: Methods Concurrency Matrix

	JGFCryptBenchSizeA	main
JGFCryptBenchSizeA	$\nparallel$	$\nparallel$
main	$\nparallel$	$\parallel$

2 JGFInstrumentor

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFInstrumentor	$\surd$
printHeader	$\surd$
printTimer	$\surd$

Table 6: State Transition Matrix

	alive
alive	$\uparrow$

Table 7: Methods Concurrency Matrix

	JGFInstrumentor	printHeader	printTimer
JGFInstrumentor	$\nparallel$	$\nparallel$	$\nparallel$
printHeader	$\nparallel$	$\parallel$	$\parallel$
printTimer	$\nparallel$	$\parallel$	$\nparallel$

3 JGFCryptBench

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFCryptBench	√
JGFrun	√
JGFinitialise	√
JGFvalidate	√
JGFtidyup	√
JGFsetsize	√

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	JGFCryptBench	JGFrun	JGFinitialise	JGFvalidate	JGFtidyup	JGFsetsize
JGFCryptBench	∦	∦	∦	∦	∦	∦
JGFrun	∦	∦	∦	∦	∥	∦
JGFinitialise	∦	∦	∦	∦	∥	∦
JGFvalidate	∦	∦	∦	∥	∥	∥
JGFtidyup	∦	∥	∥	∥	∥	∥
JGFsetsize	∦	∦	∦	∥	∥	∦

4 IDEATest

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
IDEATest	√
buildTestData	√
calcEncryptKey	√
calcDecryptKey	√
inv	√
Do	√
cipheridea	√
mul	√
freeTestData	√

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	IDEATest	buildTestData	calcEncryptKey	calcDecryptKey	inv	Do	cipheridea	mul	freeTestData
IDEATest	∦	∦	∦	∦	∦	∦	∦	∦	∦
buildTestData	∦	∦	∦	∦	‖	∦	‖	‖	∦
calcEncryptKey	∦	∦	∦	∦	‖	∦	‖	‖	∦
calcDecryptKey	∦	∦	∦	∦	‖	∦	‖	‖	∦
inv	∦	‖	‖	‖	‖	‖	‖	‖	‖
Do	∦	∦	∦	∦	‖	∦	‖	‖	∦
cipheridea	∦	‖	‖	‖	‖	‖	‖	‖	‖
mul	∦	‖	‖	‖	‖	‖	‖	‖	‖
freeTestData	∦	∦	∦	∦	‖	∦	‖	‖	∦

5 JGFTimer

Table 14: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFTimer	$\surd$
print	$\surd$
perf	$\surd$

Table 15: State Transition Matrix

	alive
alive	$\uparrow$

Table 16: Methods Concurrency Matrix

	JGFTimer	print	perf
JGFTimer	$\nparallel$	$\nparallel$	$\nparallel$
print	$\nparallel$	$\nparallel$	$\parallel$
perf	$\nparallel$	$\parallel$	$\parallel$

6 Abbreviation

Table 17: Used Abbreviation

Symbol	Meaning
$\surd$	requires clause of the method is satisfiable
$\times$	requires clause of the method is unsatisfiable
$\uparrow$	The row-state can be transitioned to the column-state
$\times$	The row-state cannot be transitioned to the column-state
$\parallel$	The row-method can be possibly executed parallel with the column-method
$\nparallel$	The row-method cannot be executed parallel with the column-method

7 Annotated Version of Sequential Java Program generated by Sip4j

```
package outputs;
import edu.cmu.cs.plural.annot.*;

@ClassStates({@State(name = "alive")})
class JGFCryptBenchSizeA {
@Perm(ensures="unique(this) in alive")
JGFCryptBenchSizeA() {   }

@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
 void main(String argv[]) {
}

}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class JGFInstrumentor {
@Perm(ensures="unique(this) in alive")
JGFInstrumentor() {   }


 void printHeader(int section, int size) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void printTimer(String name) {
}

}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class JGFCryptBench {
@Perm(ensures="unique(this) in alive")
JGFCryptBench() {   }

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
public void JGFrun(int size) {
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
public void JGFinitialise() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public void JGFvalidate() {
}

public void JGFtidyup() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void JGFsetsize(int size) {
}

}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class IDEATest {
@Perm(ensures="unique(this) in alive")
IDEATest() {   }

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 void buildTestData() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private void calcEncryptKey() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private void calcDecryptKey() {
}

private int inv(int x) {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void Do() {
}
@Perm(requires="full(#0) * full(#1) * full(#2) in alive",
ensures="full(#0) * full(#1) * full(#2) in alive")
private void cipheridea(byte[] text1, byte[] text2, int[] key) {
}

private int mul(int a, int b) {
 return 0;
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 void freeTestData() {
}

}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class JGFTimer {
@Perm(ensures="unique(this) in alive")
JGFTimer() {   }

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void print() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public double perf() {
 return 0;
}

}ENDOFCLASS
```