Summary

Sink States:$0(0 \times 10^0)$

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
JGFSparseMatmultBench	8	1	0	0	7	36	22	61
JGFInstrumentor	4	1	0	0	3	10	3	30
SparseMatmult	2	1	0	0	0	3	0	0
JGFTimer	2	1	0	0	0	3	0	0
Total Classes=4	16	4	0	0	10	52	25	48

Contents

1 JGFSparseMatmultBench 3

2 JGFInstrumentor 4

3 SparseMatmult 5

4 JGFTimer 6

5 Abbreviation 7

6 Annotated Version of Sequential Java Program generated by Sip4j 8

1 JGFSparseMatmultBench

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFSparseMatmultBench	$\surd$
main	$\surd$
JGFsetsize	$\surd$
JGFrun	$\surd$
JGFinitialise	$\surd$
RandomVector	$\surd$
JGFkernel	$\surd$
JGFtidyup	$\surd$

Table 3: State Transition Matrix

	alive
alive	$\uparrow$

Table 4: Methods Concurrency Matrix

	JGFSparseMatmultBench	main	JGFsetsize	JGFrun	JGFinitialise	RandomVector	JGFkernel	JGFtidyup
JGFSparseMatmultBench	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
main	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFsetsize	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFrun	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$
JGFinitialise	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$
RandomVector	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFkernel	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$
JGFtidyup	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

2 JGFInstrumentor

Table 5: Methods Requires Clause Satisfiability

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

Table 6: State Transition Matrix

	alive
alive	$\uparrow$

Table 7: Methods Concurrency Matrix

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

3 SparseMatmult

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
SparseMatmult	$\surd$
test	$\surd$

Table 9: State Transition Matrix

	alive
alive	$\uparrow$

Table 10: Methods Concurrency Matrix

	SparseMatmult	test
SparseMatmult	$\nparallel$	$\nparallel$
test	$\nparallel$	$\nparallel$

4 JGFTimer

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFTimer	$\surd$
stop	$\surd$

Table 12: State Transition Matrix

	alive
alive	$\uparrow$

Table 13: Methods Concurrency Matrix

	JGFTimer	stop
JGFTimer	$\nparallel$	$\nparallel$
stop	$\nparallel$	$\nparallel$

5 Abbreviation

Table 14: 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

6 Annotated Version of Sequential Java Program generated by Sip4j

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

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


 void main(String argv[]) {
}
@Perm(requires="full(#0) in alive",
ensures="full(#0) in alive")
public void JGFsetsize(int size) {
}
@Perm(requires="unique(this) * full(#0) in alive",
ensures="unique(this) * full(#0) in alive")
public void JGFrun(int size) {
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
public void JGFinitialise() {
}
@Perm(requires="pure(#0) * full(#1) in alive",
ensures="pure(#0) * full(#1) in alive")
 double[] RandomVector(int N, java.util.Random R) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void JGFkernel() {
}

public void JGFtidyup() {
}

}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 stopTimer(String name) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void printTimer(String name) {
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) * full(#0) * full(#1) * full(#2) * full(#3) * pure(#4) * pure(#5) in alive",
ensures="full(this) * full(#0) * full(#1) * full(#2) * full(#3) * pure(#4) * pure(#5) in alive")
 void test(double y[], double val[], int row[], int col[], double x[], int NUM_ITERATIONS) {
}

}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 stop() {
}

}ENDOFCLASS
```