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
SOR	2	1	0	0	0	3	0	0
JGFSORBenchSizeB	2	1	0	0	1	3	1	33
JGFSORBench	8	1	0	0	7	36	22	61
JGFInstrumentor	3	1	0	0	0	6	0	0
JGFTimer	3	1	0	0	2	6	2	33
Total Classes=5	18	5	0	0	10	54	25	46

Contents

1 SOR

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
SOR	√
SORrun	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	SOR	SORrun
SOR	∦	∦
SORrun	∦	∦

2 JGFSORBenchSizeB

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFSORBenchSizeB	√
main	√

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	JGFSORBenchSizeB	main
JGFSORBenchSizeB	∦	∦
main	∦	‖

3 JGFSORBench

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFSORBench	$\surd$
JGFrun	$\surd$
JGFinitialise	$\surd$
JGFkernel	$\surd$
RandomMatrix	$\surd$
JGFtidyup	$\surd$
JGFsetsize	$\surd$
JGFvalidate	$\surd$

Table 9: State Transition Matrix

	alive
alive	$\uparrow$

Table 10: Methods Concurrency Matrix

	JGFSORBench	JGFrun	JGFinitialise	JGFkernel	RandomMatrix	JGFtidyup	JGFsetsize	JGFvalidate
JGFSORBench	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
JGFrun	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$
JGFinitialise	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFkernel	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$
RandomMatrix	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFtidyup	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFsetsize	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$
JGFvalidate	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

4 JGFInstrumentor

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFInstrumentor	√
addTimer	√
printTimer	√

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	JGFInstrumentor	addTimer	printTimer
JGFInstrumentor	∦	∦	∦
addTimer	∦	∦	∦
printTimer	∦	∦	∦

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

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

4  @ClassStates({@State(name = "alive")})
5  class SOR {
6  @Perm(ensures="unique(this) in alive")
7  SOR() {   }

9  @Perm(requires="full(this) * pure(#0) in alive",
10 ensures="full(this) * pure(#0) in alive")
11  void SORrun(double omega, double G[][], int num_iterations) {
12 }

14 }ENDOFCLASS

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

18 class JGFSORBenchSizeB {
19 @Perm(ensures="unique(this) in alive")
20 JGFSORBenchSizeB() {   }

22 @Perm(requires="none(this) in alive",
23 ensures="unique(this) in alive")
24  void main(String argv[]) {
25 }

27 }ENDOFCLASS

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

31 class JGFSORBench {
32 @Perm(ensures="unique(this) in alive")
33 JGFSORBench() {   }

35 @Perm(requires="full(this) in alive",
36 ensures="full(this) in alive")
37 public void JGFrun(int size) {
38 }

40 public void JGFinitialise() {
41 }
42 @Perm(requires="full(this) in alive",
43 ensures="full(this) in alive")
44 public void JGFkernel() {
45 }
46 @Perm(requires="pure(#0) * pure(#1) * full(#2) in alive",
47 ensures="pure(#0) * pure(#1) * full(#2) in alive")
48  double[][] RandomMatrix(int M, int N, java.util.Random R) {
49  return null;
50 }

52 public void JGFtidyup() {
53 }
54 @Perm(requires="full(this) in alive",
55 ensures="full(this) in alive")
56 public void JGFsetsize(int size) {
57 }
58 @Perm(requires="pure(this) in alive",
59 ensures="pure(this) in alive")
60 public void JGFvalidate() {
61 }

63 }ENDOFCLASS

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

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

71 @Perm(requires="full(this) in alive",
72 ensures="full(this) in alive")
73  void addTimer(String name, String opname, int size) {
74 }
75 @Perm(requires="full(this) in alive",
```

```
76 ensures="full(this) in alive")
77  void printTimer(String name) {
78 }

80 }ENDOFCLASS

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

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

88 @Perm(requires="full(this) in alive",
89 ensures="full(this) in alive")
90 public void print() {
91 }
92 @Perm(requires="pure(this) in alive",
93 ensures="pure(this) in alive")
94 public double perf() {
95  return 0;
96 }

98 }ENDOFCLASS
```