Summary

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

Table 1: Pulse Analysis Summary

Classes	Methods	States	Unsatisfiable Clauses	Unreachable States	Possible Concuurent Methods
JGFEulerBenchSizeA	2	1	0	0	2
JGFEulerBench	6	1	0	0	6
JGFInstrumentor	4	1	0	0	0
Tunnel	11	1	0	0	10
JGFTimer	2	1	0	0	0
Statevector	6	1	0	0	5
Vector2	3	1	0	0	2
Total Classes=7	34	7	0	0	25

Contents

1 JGFEulerBenchSizeA

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFEulerBenchSizeA	√
main	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	JGFEulerBenchSizeA	main
JGFEulerBenchSizeA	∦	‖
main	‖	‖

2 JGFEulerBench

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFEulerBench	$\surd$
JGFrun	$\surd$
JGFapplication	$\surd$
JGFtidyup	$\surd$
JGFsetsize	$\surd$
JGFinitialise	$\surd$

Table 6: State Transition Matrix

	alive
alive	$\uparrow$

Table 7: Methods Concurrency Matrix

	JGFEulerBench	JGFrun	JGFapplication	JGFtidyup	JGFsetsize	JGFinitialise
JGFEulerBench	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$
JGFrun	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFapplication	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$
JGFtidyup	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JGFsetsize	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$
JGFinitialise	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

3 JGFInstrumentor

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFInstrumentor	√
addTimer	√
addOpsToTimer	√
printTimer	√

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

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

4 Tunnel

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
Tunnel	√
runiters	√
doIteration	√
calculateDeltaT	√
calculateR	√
initialise	√
calculateStateVar	√
calculateG	√
calculateF	√
calculateDamping	√
calculateDummyCells	√

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	Tunnel	runiters	doIteration	calculateDeltaT	calculateR	initialise	calculateStateVar	calculateG	calculateF	calculateDamping	calculateDummyCells
Tunnel	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
runiters	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
doIteration	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateDeltaT	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateR	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
initialise	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateStateVar	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateG	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateF	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateDamping	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
calculateDummyCells	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖

5 JGFTimer

Table 14: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFTimer	√
addops	√

Table 15: State Transition Matrix

	alive
alive	↑

Table 16: Methods Concurrency Matrix

	JGFTimer	addops
JGFTimer	∦	∦
addops	∦	∦

6 Statevector

Table 17: Methods Requires Clause Satisfiability

Method	Satisfiability
Statevector	√
svect	√
amvect	√
avect	√
mvect	√
smvect	√

Table 18: State Transition Matrix

	alive
alive	↑

Table 19: Methods Concurrency Matrix

	Statevector	svect	amvect	avect	mvect	smvect
Statevector	∦	∦	∦	∦	∦	∦
svect	∦	∥	∥	∥	∥	∥
amvect	∦	∥	∥	∥	∥	∥
avect	∦	∥	∥	∥	∥	∥
mvect	∦	∥	∥	∥	∥	∥
smvect	∦	∥	∥	∥	∥	∥

7 Vector2

Table 20: Methods Requires Clause Satisfiability

Method	Satisfiability
Vector2	√
magnitude	√
dot	√

Table 21: State Transition Matrix

	alive
alive	↑

Table 22: Methods Concurrency Matrix

	Vector2	magnitude	dot
Vector2	∦	∦	∦
magnitude	∦	∥	∥
dot	∦	∥	∥

8 Abbreviation

Table 23: 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

9 Annotated Version of Sequential Java Program generated by Sip4j

```
1  package outputs;
2  import edu.cmu.cs.plural.annot.*;
3
4  @ClassStates({@State(name = "alive")})
5  class JGFEulerBenchSizeA {
6  @Perm(ensures="unique(this) in alive")
7  JGFEulerBenchSizeA() {   }
8
9  @Perm(requires="none(this) in alive",
10 ensures="none(this) in alive")
11  void main(String argv[]) {
12 }
13
14 }ENDOFCLASS
15
16 @ClassStates({@State(name = "alive")})
17
18 class JGFEulerBench {
19 @Perm(ensures="unique(this) in alive")
20 JGFEulerBench() {   }
21
22 @Perm(requires="unique(this) in alive",
23 ensures="none(this) in alive")
24 public void JGFrun(int size) {
25 }
26 @Perm(requires="full(this) in alive",
27 ensures="full(this) in alive")
28 public void JGFapplication() {
29 }
30 @Perm(requires="unique(this) in alive",
31 ensures="none(this) in alive")
32 public void JGFtidyup() {
33 }
34 @Perm(requires="full(this) in alive",
35 ensures="full(this) in alive")
36 public void JGFsetsize(int size) {
37 }
38
39 public void JGFinitialise() {
40 }
41
42 }ENDOFCLASS
43
44 @ClassStates({@State(name = "alive")})
45
46 class JGFInstrumentor {
47 @Perm(ensures="unique(this) in alive")
48 JGFInstrumentor() {   }
49
50 @Perm(requires="full(this) in alive",
51 ensures="full(this) in alive")
52  void addTimer(String name, String opname, int size) {
53 }
54 @Perm(requires="full(this) in alive",
55 ensures="full(this) in alive")
56  void addOpsToTimer(String name, double count) {
57 }
58 @Perm(requires="full(this) in alive",
59 ensures="full(this) in alive")
60  void printTimer(String name) {
61 }
62
63 }ENDOFCLASS
64
65 @ClassStates({@State(name = "alive")})
66
67 class Tunnel {
68 @Perm(ensures="unique(this) in alive")
69 Tunnel() {   }
70
71 @Perm(requires="full(this) in alive",
72 ensures="full(this) in alive")
73 public void runiters() {
74 }
75 @Perm(requires="full(this) in alive",
```

```
ensures="full(this) in alive")
 void doIteration() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private void calculateDeltaT() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private void calculateR() {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
public void initialise() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
private void calculateStateVar(double localpg[][], double localtg[][], Statevector localug[][]) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private void calculateG(double localpg[][], double localtg[][], Statevector localug[][]) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private void calculateF(double localpg[][], double localtg[][], Statevector localug[][]) {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
private void calculateDamping(double localpg[][], Statevector localug[][]) {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
private void calculateDummyCells(double localpg[][], double localtg[][], Statevector localug[][]) {
}

}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 addops(double count) {
}

}ENDOFCLASS

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

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

@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
public Statevector svect(Statevector that) {
 return null;
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
public Statevector amvect(double m, Statevector that) {
 return null;
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
public Statevector avect(Statevector that) {
 return null;
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
public Statevector mvect(double m) {
 return null;
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
public Statevector smvect(double m, Statevector that) {
 return null;
```

```
157 }

159 }ENDOFCLASS

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

163 class Vector2 {
164 @Perm(ensures="unique(this) in alive")
165 Vector2() {   }

167 @Perm(requires="pure(this) in alive",
168 ensures="pure(this) in alive")
169 public double magnitude() {
170  return 0;
171 }
172 @Perm(requires="pure(this) in alive",
173 ensures="pure(this) in alive")
174 public double dot(Vector2 that) {
175  return 0;
176 }

178 }ENDOFCLASS
```