Summary

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

Table 1: Sip4J Analysis Summary

Classes	Methods	States	Unreachable clauses	Unreachable states	Possible concurrent methods	Total. no. of method pairs	No. of concurrent method pairs	Percentage of concurrent methods pairs
JGFInstrumentor	13	1	0	0	12	91	12	13
JGFTimer	9	1	0	0	3	45	6	13
JGFCryptBenchSizeA	2	1	0	0	0	3	0	0
JGFCryptBench	7	1	0	0	2	28	3	11
IDEATest	9	1	0	0	8	45	16	36
Total Classes=5	40	5	0	0	25	212	37	17

Contents

1 JGFInstrumentor

Table 2: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
JGFInstrumentor	√
addTimer	√
startTimer	√
stopTimer	√
addOpsToTimer	√
readTimer	√
resetTimer	√
printTimer	√
printperfTimer	√
storeData	√
retrieveData	√
printHeader	√
main	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	JGFInstrumentor	addTimer	startTimer	stopTimer	addOpsToTimer	readTimer	resetTimer	printTimer	printperfTimer	storeData	retrieveData	printHeader	main
JGFInstrumentor	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
addTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
startTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
stopTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
addOpsToTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
readTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
resetTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
printTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
printperfTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
storeData	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
retrieveData	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦
printHeader	∦	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖	‖
main	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	‖	∦

2 JGFTimer

Table 5: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
JGFTimer	√
start	√
stop	√
addops	√
reset	√
print	√
perf	√
printperf	√
longprint	√

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	JGFTimer	start	stop	addops	reset	print	perf	printperf	longprint
JGFTimer	∦	∦	∦	∦	∦	∦	∦	∦	∦
start	∦	∦	∦	∦	∦	∦	∦	∦	∦
stop	∦	∦	∦	∦	∦	∦	∦	∦	∦
addops	∦	∦	∦	∦	∦	∦	∦	∦	∦
reset	∦	∦	∦	∦	∦	∦	∦	∦	∦
print	∦	∦	∦	∦	∦	∦	∦	∦	∦
perf	∦	∦	∦	∦	∦	∦	‖	‖	‖
printperf	∦	∦	∦	∦	∦	∦	‖	‖	‖
longprint	∦	∦	∦	∦	∦	∦	‖	‖	‖

3 JGFCryptBenchSizeA

Table 8: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
JGFCryptBenchSizeA	√
main	√

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	JGFCryptBenchSizeA	main
JGFCryptBenchSizeA	∦	∦
main	∦	∦

4 JGFCryptBench

Table 11: Method's Satisfiability(Code Reachabiity Analysis

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

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	JGFCryptBench	JGFrun	JGFsetsize	JGFinitialise	JGFkernel	JGFvalidate	JGFtidyup
JGFCryptBench	∦	∦	∦	∦	∦	∦	∦
JGFrun	∦	∦	∦	∦	∦	∦	∦
JGFsetsize	∦	∦	∦	∦	∦	∦	∦
JGFinitialise	∦	∦	∦	∦	∦	∦	∦
JGFkernel	∦	∦	∦	∦	‖	‖	∦
JGFvalidate	∦	∦	∦	∦	‖	‖	∦
JGFtidyup	∦	∦	∦	∦	∦	∦	∦

5 IDEATest

Table 14: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
IDEATest	$\surd$
buildTestData	$\surd$
calcEncryptKey	$\surd$
calcDecryptKey	$\surd$
inv	$\surd$
Do	$\surd$
cipheridea	$\surd$
freeTestData	$\surd$
mul	$\surd$

Table 15: State Transition Matrix

	alive
alive	$\uparrow$

Table 16: Methods Concurrency Matrix

	IDEATest	buildTestData	calcEncryptKey	calcDecryptKey	inv	Do	cipheridea	freeTestData	mul
IDEATest	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
buildTestData	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$
calcEncryptKey	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$
calcDecryptKey	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$
inv	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
Do	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$	$\nparallel$	$\parallel$
cipheridea	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$
freeTestData	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$
mul	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

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 the input program generated by Sip4J

```
1  package outputs;
2  import edu.cmu.cs.plural.annot.*;
3
4  @ClassStates({@State(name = "alive")})
5  class JGFInstrumentor {
6  @Perm(ensures="unique(this) in alive")
7  JGFInstrumentor() {   }
8
9  @Perm(requires="share(this) in alive",
10 ensures="share(this) in alive")
11   void addTimer(String name) {
12
13 }
14 @Perm(requires="share(this) in alive",
15 ensures="share(this) in alive")
16   void startTimer(String name) {
17
18 }
19 @Perm(requires="share(this) in alive",
20 ensures="share(this) in alive")
21   void stopTimer(String name) {
22
23 }
24 @Perm(requires="share(this) in alive",
25 ensures="share(this) in alive")
26   void addOpsToTimer(String name, double count) {
27
28 }
29 @Perm(requires="share(this) in alive",
30 ensures="share(this) in alive")
31   double readTimer(String name) {
32  return 0;
33
34 }
35 @Perm(requires="share(this) in alive",
36 ensures="share(this) in alive")
37   void resetTimer(String name) {
38
39 }
40 @Perm(requires="share(this) in alive",
41 ensures="share(this) in alive")
42   void printTimer(String name) {
43
44 }
45 @Perm(requires="share(this) in alive",
46 ensures="share(this) in alive")
47   void printperfTimer(String name) {
48
49 }
50 @Perm(requires="share(this) in alive",
51 ensures="share(this) in alive")
52   void storeData(String name, Object obj) {
53
54 }
55 @Perm(requires="share(this) in alive",
56 ensures="share(this) in alive")
57   void retrieveData(String name, Object obj) {
58
59 }
60
61   void printHeader(int section, int size) {
62
63 }
64 @Perm(requires="unique(this) in alive",
65 ensures="unique(this) in alive")
66   void main(String argv[]) {
67
68 }
69
70 }ENDOFCLASS
71
72 @ClassStates({@State(name = "alive")})
73
74 class JGFTimer {
75 @Perm(ensures="unique(this) in alive")
76 JGFTimer() {   }
77
78 @Perm(requires="share(this) in alive",
```

```
79  ensures="share(this) in alive")
80   public void start() {

82  }
83  @Perm(requires="share(this) in alive",
84  ensures="share(this) in alive")
85   public void stop() {

87  }
88  @Perm(requires="share(this) in alive",
89  ensures="share(this) in alive")
90   public void addops(double count) {

92  }
93  @Perm(requires="share(this) in alive",
94  ensures="share(this) in alive")
95   public void reset() {

97  }
98  @Perm(requires="share(this) in alive",
99  ensures="share(this) in alive")
100  public void print() {

102 }
103 @Perm(requires="pure(this) in alive",
104 ensures="pure(this) in alive")
105  public double perf() {
106  return 0;

108 }
109 @Perm(requires="pure(this) in alive",
110 ensures="pure(this) in alive")
111  public void printperf() {

113 }
114 @Perm(requires="pure(this) in alive",
115 ensures="pure(this) in alive")
116  public void longprint() {

118 }

120 }ENDOFCLASS

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

124 class JGFCryptBenchSizeA {
125 @Perm(ensures="unique(this) in alive")
126 JGFCryptBenchSizeA() {   }

128 @Perm(requires="unique(this) in alive",
129 ensures="unique(this) in alive")
130   void main(String argv[]) {

132 }

134 }ENDOFCLASS

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

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

142 @Perm(requires="unique(this) in alive",
143 ensures="unique(this) in alive")
144  public void JGFrun(int size) {

146 }
147 @Perm(requires="share(this) in alive",
148 ensures="share(this) in alive")
149  public void JGFsetsize(int size) {

151 }
152 @Perm(requires="unique(this) in alive",
153 ensures="unique(this) in alive")
154  public void JGFinitialise() {

156 }
157 @Perm(requires="pure(this) in alive",
158 ensures="pure(this) in alive")
159  public void JGFkernel() {
```

```
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 public void JGFvalidate() {

}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 public void JGFtidyup() {

}

}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="share(this) in alive",
ensures="share(this) in alive")
 private void calcEncryptKey() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 private void calcDecryptKey() {

}

 private int inv(int x) {
 return 0;

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 public void Do() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 private void cipheridea(byte[] text1, byte[] text2, int[] key) {

}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
  void freeTestData() {

}

 private int mul(int a, int b) {
 return 0;

}

}ENDOFCLASS
```