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
BlackScholes	1	1	0	0	0	1	0	0
StdRandom	11	1	1	0	10	66	56	85
MersenneTwisterFast	7	1	1	0	0	28	0	0
Gaussian	5	1	0	0	4	15	10	67
StdOut	6	1	0	0	5	21	15	71
SeqBlackScholes	4	1	0	0	3	10	6	60
Total Classes=6	34	6	2	0	22	141	87	62

Contents

1 BlackScholes

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
BlackScholes	√

Table 3: State Transition Matrix

	alive
alive	↑

2 StdRandom

Table 4: Methods Requires Clause Satisfiability

Method	Satisfiability
StdRandom	√
setSeed	√
uniform	√
uniform	×
bernoulli	√
gaussian	√
geometric	√
pareto	√
discrete	√
shuffle	√
poisson	√

Table 5: State Transition Matrix

	alive
alive	↑

Table 6: Methods Concurrency Matrix

	StdRandom	setSeed	uniform	uniform	bernoulli	gaussian	geometric	pareto	discrete	shuffle	poisson
StdRandom	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
setSeed	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
uniform	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
uniform	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
bernoulli	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
gaussian	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
geometric	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
pareto	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
discrete	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
shuffle	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥
poisson	∦	∥	∥	∥	∥	∥	∥	∥	∥	∥	∥

3 MersenneTwisterFast

Table 7: Methods Requires Clause Satisfiability

Method	Satisfiability
MersenneTwisterFast	√
setSeed	√
setSeed	×
nextInt	√
nextShort	√
nextBoolean	√
nextDouble	√

Table 8: State Transition Matrix

	alive
alive	↑

Table 9: Methods Concurrency Matrix

	MersenneTwisterFast	setSeed	setSeed	nextInt	nextShort	nextBoolean	nextDouble
MersenneTwisterFast	∦	∦	∦	∦	∦	∦	∦
setSeed	∦	∦	∦	∦	∦	∦	∦
setSeed	∦	∦	∦	∦	∦	∦	∦
nextInt	∦	∦	∦	∦	∦	∦	∦
nextShort	∦	∦	∦	∦	∦	∦	∦
nextBoolean	∦	∦	∦	∦	∦	∦	∦
nextDouble	∦	∦	∦	∦	∦	∦	∦

4 Gaussian

Table 10: Methods Requires Clause Satisfiability

Method	Satisfiability
Gaussian	√
phi	√
Phi	√
PhiInverse	√
main	√

Table 11: State Transition Matrix

	alive
alive	↑

Table 12: Methods Concurrency Matrix

	Gaussian	phi	Phi	PhiInverse	main
Gaussian	∦	∦	∦	∦	∦
phi	∦	∥	∥	∥	∥
Phi	∦	∥	∥	∥	∥
PhiInverse	∦	∥	∥	∥	∥
main	∦	∥	∥	∥	∥

5 StdOut

Table 13: Methods Requires Clause Satisfiability

Method	Satisfiability
StdOut	$\surd$
println	$\surd$
close	$\surd$
print	$\surd$
printf	$\surd$
main	$\surd$

Table 14: State Transition Matrix

	alive
alive	$\uparrow$

Table 15: Methods Concurrency Matrix

	StdOut	println	close	print	printf	main
StdOut	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
println	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
close	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
print	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
printf	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
main	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

6 SeqBlackScholes

Table 16: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqBlackScholes	√
main	√
call	√
call2	√

Table 17: State Transition Matrix

	alive
alive	↑

Table 18: Methods Concurrency Matrix

	SeqBlackScholes	main	call	call2
SeqBlackScholes	∦	∦	∦	∦
main	∦	∥	∥	∥
call	∦	∥	∥	∥
call2	∦	∥	∥	∥

7 Abbreviation

Table 19: 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

8 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 BlackScholes {
6  @Perm(ensures="unique(this) in alive")
7  BlackScholes() {   }
8
9
10 }ENDOFCLASS
11
12 @ClassStates({@State(name = "alive")})
13
14 class StdRandom {
15 @Perm(ensures="unique(this) in alive")
16 StdRandom() {   }
17
18 @Perm(requires="none(this) in alive",
19 ensures="unique(this) in alive")
20  void setSeed(long s) {
21 }
22 @Perm(requires="pure(this) in alive",
23 ensures="pure(this) in alive")
24  double uniform() {
25  return 0;
26 }
27
28  int uniform(int N) {
29  return 0;
30 }
31 @Perm(requires="pure(this) in alive",
32 ensures="pure(this) in alive")
33  boolean bernoulli(double p) {
34  return 0;
35 }
36 @Perm(requires="pure(this) in alive",
37 ensures="pure(this) in alive")
38  double gaussian() {
39  return 0;
40 }
41 @Perm(requires="pure(this) in alive",
42 ensures="pure(this) in alive")
43  int geometric(double p) {
44  return 0;
45 }
46 @Perm(requires="pure(this) in alive",
47 ensures="pure(this) in alive")
48  double pareto(double alpha) {
49  return 0;
50 }
51 @Perm(requires="pure(this) in alive",
52 ensures="pure(this) in alive")
53  int discrete(double[] a) {
54  return 0;
55 }
56 @Perm(requires="pure(this) in alive",
57 ensures="pure(this) in alive")
58  void shuffle(Object[] a) {
59 }
60 @Perm(requires="pure(this) in alive",
61 ensures="pure(this) in alive")
62  int poisson(double lambda) {
63  return 0;
64 }
65
66 }ENDOFCLASS
67
68 @ClassStates({@State(name = "alive")})
69
70 class MersenneTwisterFast {
71 @Perm(ensures="unique(this) in alive")
72 MersenneTwisterFast() {   }
73
74 @Perm(requires="unique(this) in alive",
75 ensures="unique(this) in alive")
```

```
 void setSeed(final long seed) {
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 void setSeed(final long seed) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 int nextInt() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 short nextShort() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 boolean nextBoolean() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 double nextDouble() {
 return 0;
}

}ENDOFCLASS

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

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


 double phi(double x) {
 return 0;
}

 double Phi(double z) {
 return 0;
}

 double PhiInverse(double y) {
 return 0;
}

 void main(String[] args) {
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void println(double x) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void close() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void print() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void printf(String format, Object... args) {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
 void main(String[] args) {
}

}ENDOFCLASS
```

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

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

@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
 void main(String[] args) {
}
@Perm(requires="pure(this) * pure(#0) * pure(#1) * pure(#2) * pure(#3) * pure(#4) in alive",
ensures="pure(this) * pure(#0) * pure(#1) * pure(#2) * pure(#3) * pure(#4) in alive")
 double call(double S, double X, double r, double sigma, double T, long N) {
 return 0;
}
@Perm(requires="pure(this) * pure(#0) * pure(#1) * pure(#2) * pure(#3) * pure(#4) in alive",
ensures="pure(this) * pure(#0) * pure(#1) * pure(#2) * pure(#3) * pure(#4) in alive")
 double call2(double S, double X, double r, double sigma, double T, long N) {
 return 0;
}

}ENDOFCLASS
```