Summary

Sink States:0(0×10^0)

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
StdRandom	25	1	0	0	1	325	1	0
MersenneTwisterFast	6	1	0	0	0	21	0	0
StdOut	5	1	0	0	0	15	0	0
Gaussian	8	1	0	0	7	36	23	64
SeqBlackScholes	5	1	0	0	1	15	1	7
BlackScholes	1	1	0	0	0	1	0	0
Total Classes=6	50	6	0	0	9	413	25	6

Contents

1 StdRandom

Table 2: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
StdRandom	√
setSeed	√
getSeed	√
uniformO1	√
uniformO2	√
random	√
uniformO3	√
uniform	√
bernoulliO1	√
bernoulliO2	√
gaussianO1	√
gaussianO2	√
geometric	√
poisson	√
pareto	√
cauchy	√
discrete	√
exp	√
shuffleO1	√
shuffleO2	√
shuffleO3	√
shuffleO4	√
shuffleO5	√
shuffleO6	√
main	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	StdRandom	setSeed	getSeed	uniformO1	uniformO2	random	uniformO3	uniform	bernoulliO1	bernoulliO2	gaussianO1	gaussianO2	geometric	poisson	pareto	cauchy	discrete	exp	shuffleO1	shuffleO2	shuffleO3	shuffleO4	shuffleO5	shuffleO6	main
StdRandom	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
setSeed	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
getSeed	∦	∦	‖	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
uniformO1	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
uniformO2	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
random	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
uniformO3	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦

uniform	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
bernoulliO1	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
bernoulliO2	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
gaussianO1	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
gaussianO2	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
geometric	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
poisson	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
pareto	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
cauchy	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
discrete	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
exp	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
shuffleO1	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
shuffleO2	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
shuffleO3	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
shuffleO4	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
shuffleO5	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
shuffleO6	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦
main	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦	∦

2 MersenneTwisterFast

Table 5: Method's Satisfiability(Code Reachabiity Analysis

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

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

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

3 StdOut

Table 8: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
StdOut	√
println	√
printf	√
print	√
close	√

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	StdOut	println	printf	print	close
StdOut	∦	∦	∦	∦	∦
println	∦	∦	∦	∦	∦
printf	∦	∦	∦	∦	∦
print	∦	∦	∦	∦	∦
close	∦	∦	∦	∦	∦

4 Gaussian

Table 11: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Gaussian	√
phi	√
phiOverload	√
PhiOverload	√
Phi	√
PhiInverse	√
PhiInverseOverload	√
main	√

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	Gaussian	phi	phiOverload	PhiOverload	Phi	PhiInverse	PhiInverseOverload	main
Gaussian	∦	∦	∦	∦	∦	∦	∦	∦
phi	∦	∥	∥	∥	∥	∥	∥	∦
phiOverload	∦	∥	∥	∥	∥	∥	∥	∦
PhiOverload	∦	∥	∥	∥	∥	∥	∥	∦
Phi	∦	∥	∥	∥	∥	∥	∥	∥
PhiInverse	∦	∥	∥	∥	∥	∥	∥	∥
PhiInverseOverload	∦	∥	∥	∥	∥	∥	∥	∦
main	∦	∦	∦	∦	∥	∥	∦	∦

5 SeqBlackScholes

Table 14: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
SeqBlackScholes	√
callPrice	√
call	√
call2	√
main	√

Table 15: State Transition Matrix

	alive
alive	↑

Table 16: Methods Concurrency Matrix

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

6 BlackScholes

Table 17: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
BlackScholes	√

Table 18: State Transition Matrix

	alive
alive	↑

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 the input program generated by Sip4J

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

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

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
  void setSeed(long s) {

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
  long getSeed() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double uniform01() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  int uniform02(int N) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double random() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  int uniform03(int a, int b) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double uniform(double a, double b) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  boolean bernoulli01(double p) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  boolean bernoulli02() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double gaussian01() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double gaussian02(double mean, double stddev) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  int geometric(double p) {
 return 0;
```

```
}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  int poisson(double lambda) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double pareto(double alpha) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double cauchy() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  int discrete(double[] a) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double exp(double lambda) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void shuffle01(Object[] a) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void shuffle02(double[] a) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void shuffle03(int[] a) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void shuffle04(Object[] a, int lo, int hi) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void shuffle05(double[] a, int lo, int hi) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void shuffle06(int[] a, int lo, int hi) {

}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
  void main(String[] args) {

}

}ENDOFCLASS

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

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

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
  void setSeed(final long seed) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double nextDouble() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  int nextInt(final int n) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  short nextShort() {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  boolean nextBoolean() {
 return 0;

}

}ENDOFCLASS

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

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

@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void println(Object x) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void printf(String format, Object... args) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void print(Object x) {

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

}

}ENDOFCLASS

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

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

@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
  double phi(double x) {
 return 0;

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
  double phiOverload(double x, double mu, double sigma) {
 return 0;

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
  double PhiOverload(double z) {
 return 0;

}

  double Phi(double z, double mu, double sigma) {
```

```
 return 0;

}

  double PhiInverse(double y, double delta, double lo, double hi) {
 return 0;

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
  double PhiInverseOverload(double y) {
 return 0;

}
@Perm(requires="unique(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="immutable(this) in alive",
ensures="immutable(this) in alive")
  double callPrice(double S, double X, double r, double sigma, double T) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double call(double S, double X, double r, double sigma, double T, long N) {
 return 0;

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  double call2(double S, double X, double r, double sigma, double T, long N) {
 return 0;

}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
  void main(String[] args) {

}

}ENDOFCLASS

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

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


}ENDOFCLASS
```