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
Item	1	1	0	0	0	1	0	0
SeqGA	2	1	0	0	0	3	0	0
Knapsack	7	1	0	0	1	28	1	4
MersenneTwisterFast	6	1	0	0	0	21	0	0
Indiv	3	1	0	0	1	6	1	17
ComparatorOnFitness	2	1	0	0	2	3	2	67
Total Classes=6	21	6	0	0	4	62	4	6

Contents

1 Item

Table 2: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Item	√

Table 3: State Transition Matrix

	alive
alive	↑

2 SeqGA

Table 4: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
SeqGA	√
main	√

Table 5: State Transition Matrix

	alive
alive	↑

Table 6: Methods Concurrency Matrix

	SeqGA	main
SeqGA	∦	∦
main	∦	∦

3 Knapsack

Table 7: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Knapsack	√
resetSeed	√
createRandomIndiv	√
evaluate	√
phenotype	√
recombine	√
mutate	√

Table 8: State Transition Matrix

	alive
alive	↑

Table 9: Methods Concurrency Matrix

	Knapsack	resetSeed	createRandomIndiv	evaluate	phenotype	recombine	mutate
Knapsack	∦	∦	∦	∦	∦	∦	∦
resetSeed	∦	∦	∦	∦	∦	∦	∦
createRandomIndiv	∦	∦	∦	∦	∦	∦	∦
evaluate	∦	∦	∦	∦	∦	∦	∦
phenotype	∦	∦	∦	∦	‖	∦	∦
recombine	∦	∦	∦	∦	∦	∦	∦
mutate	∦	∦	∦	∦	∦	∦	∦

4 MersenneTwisterFast

Table 10: Method's Satisfiability(Code Reachabiity Analysis

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

Table 11: State Transition Matrix

	alive
alive	↑

Table 12: Methods Concurrency Matrix

	MersenneTwisterFast	nextDouble	nextInt	nextFloat	setSeed	nextBoolean
MersenneTwisterFast	∦	∦	∦	∦	∦	∦
nextDouble	∦	∦	∦	∦	∦	∦
nextInt	∦	∦	∦	∦	∦	∦
nextFloat	∦	∦	∦	∦	∦	∦
setSeed	∦	∦	∦	∦	∦	∦
nextBoolean	∦	∦	∦	∦	∦	∦

5 Indiv

Table 13: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Indiv	√
set	√
compareTo	√

Table 14: State Transition Matrix

	alive
alive	↑

Table 15: Methods Concurrency Matrix

	Indiv	set	compareTo
Indiv	∦	∦	∦
set	∦	∦	∦
compareTo	∦	∦	∥

6 ComparatorOnFitness

Table 16: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
ComparatorOnFitness	√
compare	√

Table 17: State Transition Matrix

	alive
alive	↑

Table 18: Methods Concurrency Matrix

	ComparatorOnFitness	compare
ComparatorOnFitness	∦	‖
compare	‖	‖

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

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

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


10 }ENDOFCLASS

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

14 class SeqGA {
15 @Perm(ensures="unique(this) in alive")
16 SeqGA() {   }

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

22 }

24 }ENDOFCLASS

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

28 class Knapsack {
29 @Perm(ensures="unique(this) in alive")
30 Knapsack() {   }

32 @Perm(requires="unique(this) in alive",
33 ensures="unique(this) in alive")
34   void resetSeed() {

36 }
37 @Perm(requires="share(this) in alive",
38 ensures="share(this) in alive")
39   Indiv createRandomIndiv(Indiv ind) {
40  return null;

42 }
43 @Perm(requires="share(this) in alive",
44 ensures="share(this) in alive")
45   void evaluate(Indiv indiv) {

47 }
48 @Perm(requires="pure(this) in alive",
49 ensures="pure(this) in alive")
50   int[] phenotype(Indiv indiv) {
51  return null;

53 }
54 @Perm(requires="share(this) in alive",
55 ensures="share(this) in alive")
56   Indiv recombine(Indiv ind, Indiv p1, Indiv p2) {
57  return null;

59 }
60 @Perm(requires="share(this) in alive",
61 ensures="share(this) in alive")
62   void mutate(Indiv indiv) {

64 }

66 }ENDOFCLASS

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

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

74 @Perm(requires="share(this) in alive",
75 ensures="share(this) in alive")
76   double nextDouble() {
77  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")
  float nextFloat() {
 return 0;

}
@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")
  boolean nextBoolean() {
 return 0;

}

}ENDOFCLASS

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

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

@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void set(int w, boolean h) {

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 public int compareTo(Indiv other) {
 return 0;

}

}ENDOFCLASS

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

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

@Perm(ensures="none(this) in alive")
 public int compare(Integer a, Integer b) {
 return 0;

}

}ENDOFCLASS
```