

Summary

Sink States:0(0×10^0)

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
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	1	3	1	33
Total Classes=6	21	6	0	0	3	62	3	5

Contents

1	Webserver	3
2	Server	4
3	Client	5
4	Abbreviation	6
5	Annotated Version of Sequential Java Program generated by Sip4j	7

1 Item

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Item	✓

Table 3: State Transition Matrix

	alive
alive	↑

2 SeqGA

Table 4: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqGA	✓
main	✓

Table 5: State Transition Matrix

	alive
alive	↑

Table 6: Methods Concurrency Matrix

	SeqGA	main
SeqGA	⧻	⧻
main	⧻	⧻

3 Knapsack

Table 7: Methods Requires Clause Satisfiability

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: Methods Requires Clause Satisfiability

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: Methods Requires Clause Satisfiability

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: Methods Requires Clause Satisfiability

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
✓	requires clause of the method is satisfiable
✗	requires clause of the method is unsatisfiable
↑	The row-state can be transitioned to the column-state
✕	The row-state cannot be transitioned to the column-state
	The row-method can be possibly executed parallel with the column-method
⋈	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 Item {
6   @Perm(ensures="unique(this) in alive")
7   Item() { }
8
9 }
10 }ENDOFCLASS
11
12 @ClassStates({@State(name = "alive")})
13
14 class SeqGA {
15   @Perm(ensures="unique(this) in alive")
16   SeqGA() { }
17
18   @Perm(requires="unique(this) in alive",
19   ensures="unique(this) in alive")
20   void main(String[] args) {
21   }
22
23 }ENDOFCLASS
24
25 @ClassStates({@State(name = "alive")})
26
27 class Knapsack {
28   @Perm(ensures="unique(this) in alive")
29   Knapsack() { }
30
31   @Perm(requires="unique(this) in alive",
32   ensures="unique(this) in alive")
33   void resetSeed() {
34   }
35   @Perm(requires="full(this) in alive",
36   ensures="full(this) in alive")
37   Indiv createRandomIndiv(Indiv ind) {
38     return null;
39   }
40   @Perm(requires="full(this) in alive",
41   ensures="full(this) in alive")
42   void evaluate(Indiv indiv) {
43   }
44   @Perm(requires="pure(this) in alive",
45   ensures="pure(this) in alive")
46   int[] phenotype(Indiv indiv) {
47     return null;
48   }
49   @Perm(requires="full(this) in alive",
50   ensures="full(this) in alive")
51   Indiv recombine(Indiv ind, Indiv p1, Indiv p2) {
52     return null;
53   }
54   @Perm(requires="full(this) in alive",
55   ensures="full(this) in alive")
56   void mutate(Indiv indiv) {
57   }
58
59 }ENDOFCLASS
60
61 @ClassStates({@State(name = "alive")})
62
63 class MersenneTwisterFast {
64   @Perm(ensures="unique(this) in alive")
65   MersenneTwisterFast() { }
66
67   @Perm(requires="full(this) in alive",
68   ensures="full(this) in alive")
69   double nextDouble() {
70     return 0;
71   }
72   @Perm(requires="full(this) in alive",
73   ensures="full(this) in alive")
74   int nextInt(final int n) {
75     return 0;
76   }
77 }
```

```

76 }
77 @Perm(requires="full(this) in alive",
78 ensures="full(this) in alive")
79 float nextFloat() {
80     return 0;
81 }
82 @Perm(requires="unique(this) in alive",
83 ensures="unique(this) in alive")
84 void setSeed(final long seed) {
85 }
86 @Perm(requires="full(this) in alive",
87 ensures="full(this) in alive")
88 boolean nextBoolean() {
89     return 0;
90 }
91
92 }ENDOFCLASS
93
94 @ClassStates({@State(name = "alive")})
95
96 class Indiv {
97     @Perm(ensures="unique(this) in alive")
98     Indiv() { }
99
100     @Perm(requires="full(this) in alive",
101     ensures="full(this) in alive")
102     public void set(int w, boolean h) {
103     }
104     @Perm(requires="pure(this) in alive",
105     ensures="pure(this) in alive")
106     public int compareTo(Indiv other) {
107         return 0;
108     }
109
110 }ENDOFCLASS
111
112 @ClassStates({@State(name = "alive")})
113
114 class ComparatorOnFitness {
115     @Perm(ensures="unique(this) in alive")
116     ComparatorOnFitness() { }
117
118     @Perm(requires="pure(this) in alive",
119     ensures="pure(this) in alive")
120     public int compare(Integer a, Integer b) {
121         return 0;
122     }
123
124 }ENDOFCLASS

```