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
SeqShellSort	5	1	0	0	4	15	10	67
Client	2	1	0	0	1	3	1	33
Total Classes=2	7	2	0	0	5	18	11	61

Contents

1 SeqShellSort

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqShellSort	√
InitializeColl	√
displayArray	√
Sort	√
isSorted	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	SeqShellSort	InitializeColl	displayArray	Sort	isSorted
SeqShellSort	∦	∦	∦	∦	∦
InitializeColl	∦	∥	∥	∥	∥
displayArray	∦	∥	∥	∥	∥
Sort	∦	∥	∥	∥	∥
isSorted	∦	∥	∥	∥	∥

2 Client

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
Client	√
main	√

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	Client	main
Client	#	#
main	#	‖

3 Abbreviation

Table 8: 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

4 Annotated Version of Sequential Java Program generated by Sip4j

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

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

9  @Perm(requires="full(#0) in alive",
10 ensures="full(#0) in alive")
11  Integer[] InitializeColl(Integer[] data) {
12  return null;
13 }
14 @Perm(requires="pure(#0) in alive",
15 ensures="pure(#0) in alive")
16  void displayArray(Integer[] data) {
17 }
18 @Perm(requires="full(#0) * pure(#1) in alive",
19 ensures="full(#0) * pure(#1) in alive")
20  void Sort(Integer[] data, Integer[] gaps) {
21 }
22 @Perm(requires="pure(#0) in alive",
23 ensures="pure(#0) in alive")
24  boolean isSorted(Integer[] data) {
25  return 0;
26 }

28 }ENDOFCLASS

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

32 class Client {
33 @Perm(ensures="unique(this) in alive")
34 Client() {   }

36 @Perm(requires="none(this) in alive",
37 ensures="unique(this) in alive")
38  void main(String[] args) {
39 }

41 }ENDOFCLASS
```