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	2	15	3	20
Client	2	1	0	0	0	3	0	0
Total Classes=2	7	2	0	0	2	18	3	17

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
$\sqrt{}$	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

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 Integer[] InitializeColl(Integer[] data) {
  return null;
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 void displayArray(Integer[] data) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void Sort(Integer[] data, Integer[] gaps) {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 boolean isSorted(Integer[] data) {
  return 0;
}

}ENDOFCLASS

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

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

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

}ENDOFCLASS
```