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
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: Method's Satisfiability(Code Reachabiity Analysis

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: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Client	$\surd$
main	$\surd$

Table 6: State Transition Matrix

	alive
alive	$\uparrow$

Table 7: Methods Concurrency Matrix

	Client	main
Client	$\nparallel$	$\nparallel$
main	$\nparallel$	$\nparallel$

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 the input program generated by Sip4J

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

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

@Perm(requires="share(this) in alive",
ensures="share(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="share(this) in alive",
ensures="share(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
```