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
SeqQuickSort	4	1	0	0	1	10	1	10
ArrayHelper	3	1	0	0	1	6	1	17
QuickSort	2	1	0	0	0	3	0	0
Total Classes=3	9	3	0	0	2	19	2	11

Contents

1 SeqQuickSort

Table 2: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
SeqQuickSort	√
main	√
sort	√
qsort_seq	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	SeqQuickSort	main	sort	qsort_seq
SeqQuickSort	∦	∦	∦	∦
main	∦	∦	∦	∦
sort	∦	∦	∥	∦
qsort_seq	∦	∦	∦	∦

2 ArrayHelper

Table 5: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
ArrayHelper	√
generateRandomArray	√
checkArray	√

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	ArrayHelper	generateRandomArray	checkArray
ArrayHelper	∦	∦	∦
generateRandomArray	∦	∦	∦
checkArray	∦	∦	‖

3 QuickSort

Table 8: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
QuickSort	√
partition	√

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	QuickSort	partition
QuickSort	∦	∦
partition	∦	∦

4 Abbreviation

Table 11: 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

5 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 SeqQuickSort {
6  @Perm(ensures="unique(this) in alive")
7  SeqQuickSort() {   }

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

13 }
14 @Perm(requires="pure(this) in alive",
15 ensures="pure(this) in alive")
16   void sort(long[] original_array) {

18 }
19 @Perm(requires="pure(this) in alive",
20 ensures="pure(this) in alive")
21   void qsort_seq(long[] data, int left, int right) {

23 }

25 }ENDOFCLASS

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

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

33 @Perm(requires="unique(this) in alive",
34 ensures="unique(this) in alive")
35   long[] generateRandomArray(long[] ar, int size) {
36  return null;

38 }
39 @Perm(requires="pure(this) in alive",
40 ensures="pure(this) in alive")
41   boolean checkArray(long[] c) {
42  return 0;

44 }

46 }ENDOFCLASS

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

50 class QuickSort {
51 @Perm(ensures="unique(this) in alive")
52 QuickSort() {   }

54 @Perm(requires="share(this) in alive",
55 ensures="share(this) in alive")
56   int partition(long[] data, int left, int right) {
57  return 0;

59 }

61 }ENDOFCLASS
```