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
SeqQuickSort	4	1	0	0	2	10	3	30
ArrayHelper	3	1	0	0	2	6	3	50
QuickSort	2	1	0	0	1	3	1	33
Total Classes=3	9	3	0	0	5	19	7	37

Contents

1 SeqQuickSort

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqQuickSort	√
main	√
qsort_seq	√
sort	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

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

2 ArrayHelper

Table 5: Methods Requires Clause Satisfiability

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: Methods Requires Clause Satisfiability

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 Sequential Java 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="none(this) in alive",
10 ensures="unique(this) in alive")
11  void main(String[] args) {
12 }
13 @Perm(requires="full(#0) in alive",
14 ensures="full(#0) in alive")
15  void qsort_seq(long[] data, int left, int right) {
16 }
17 @Perm(requires="full(#0) in alive",
18 ensures="full(#0) in alive")
19  void sort(long[] original_array) {
20 }

22 }ENDOFCLASS

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

26 class ArrayHelper {
27 @Perm(ensures="unique(this) in alive")
28 ArrayHelper() {   }

30 @Perm(requires="none(#0) in alive",
31 ensures="unique(#0) in alive")
32  long[] generateRandomArray(long[] ar, int size) {
33  return null;
34 }
35 @Perm(requires="pure(#0) in alive",
36 ensures="pure(#0) in alive")
37  boolean checkArray(long[] c) {
38  return 0;
39 }

41 }ENDOFCLASS

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

45 class QuickSort {
46 @Perm(ensures="unique(this) in alive")
47 QuickSort() {   }

49 @Perm(requires="full(#0) in alive",
50 ensures="full(#0) in alive")
51  int partition(long[] data, int left, int right) {
52  return 0;
53 }

55 }ENDOFCLASS
```