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
Complex	4	1	0	0	0	10	0	0
SeqFFT	2	1	0	0	0	3	0	0
Client	2	1	0	0	1	3	1	33
FFTUtility	3	1	0	0	2	6	2	33
Total Classes=4	11	4	0	0	3	22	3	14

Contents

1 Complex

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Complex	√
plus	√
minus	√
times	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	Complex	plus	minus	times
Complex	∦	∦	∦	∦
plus	∦	∦	∦	∦
minus	∦	∦	∦	∦
times	∦	∦	∦	∦

2 SeqFFT

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqFFT	$\surd$
sequentialFFT	$\surd$

Table 6: State Transition Matrix

	alive
alive	$\uparrow$

Table 7: Methods Concurrency Matrix

	SeqFFT	sequentialFFT
SeqFFT	$\nparallel$	$\nparallel$
sequentialFFT	$\nparallel$	$\nparallel$

3 Client

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
Client	√
main	√

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	Client	main
Client	∦	∦
main	∦	‖

4 FFTUtility

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
FFTUtility	√
createRandomComplexArray	√
show	√

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	FFTUtility	createRandomComplexArray	show
FFTUtility	∦	∦	∦
createRandomComplexArray	∦	∦	‖
show	∦	‖	‖

5 Abbreviation

Table 14: 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

6 Annotated Version of Sequential Java Program generated by Sip4j

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public Complex plus(Complex b) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public Complex minus(Complex b) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public Complex times(Complex b) {
 return null;
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive * pure(#0) in alive",
ensures="full(this) in alive * pure(#0) in alive")
 Complex[] sequentialFFT(Complex[] x) {
 return null;
}

}ENDOFCLASS

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

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

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

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive * unique(#0) in alive * pure(#1) in alive",
ensures="full(this) in alive * unique(#0) in alive * pure(#1) in alive")
 Complex[] createRandomComplexArray(Complex[] x, int n) {
 return null;
}
@Perm(requires="pure(#0) in alive",
ensures="pure(#0) in alive")
 void show(Complex[] x, String title) {
}

}ENDOFCLASS
```