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
Fibonacci	4	1	0	0	2	10	3	30
Total Classes=1	4	1	0	0	2	10	3	30

Contents

1 Fibonacci

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Fibonacci	$\surd$
computeFibonacci	$\surd$
display	$\surd$
main	$\surd$

Table 3: State Transition Matrix

	alive
alive	$\uparrow$

Table 4: Methods Concurrency Matrix

	Fibonacci	computeFibonacci	display	main
Fibonacci	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
computeFibonacci	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$
display	$\nparallel$	$\parallel$	$\parallel$	$\nparallel$
main	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$

2 Abbreviation

Table 5: 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

3 Annotated Version of Sequential Java Program generated by Sip4j

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

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

@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 Integer computeFibonacci(Integer num) {
 return null;
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 void display(Integer num) {
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 void main(String[] args) {
}

}ENDOFCLASS
```