

Summary

Sink States:0(0×10^0)

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	3	10	6	60
Total Classes=1	4	1	0	0	3	10	6	60

Contents

1	Fibonacci	3
2	Abbreviation	4
3	Annotated Version of Sequential Java Program generated by Sip4j	5

1 Fibonacci

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Fibonacci	✓
computeFibonacci	✓
display	✓
main	✓

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	Fibonacci	computeFibonacci	display	main
Fibonacci	⌘	⌘	⌘	⌘
computeFibonacci	⌘			
display	⌘			
main	⌘			

2 Abbreviation

Table 5: Used Abbreviation

Symbol	Meaning
✓	requires clause of the method is satisfiable
✗	requires clause of the method is unsatisfiable
↑	The row-state can be transitioned to the column-state
✕	The row-state cannot be transitioned to the column-state
	The row-method can be possibly executed parallel with the column-method
⋈	The row-method cannot be executed parallel with the column-method

3 Annotated Version of Sequential Java Program generated by Sip4j

```
1 package outputs;
2 import edu.cmu.cs.plural.annot.*;
3
4 @ClassStates({@State(name = "alive")})
5 class Fibonacci {
6   @Perm(ensures="unique(this) in alive")
7   Fibonacci() { }
8
9   @Perm(requires="pure(#0) in alive",
10  ensures="pure(#0) in alive")
11   Integer computeFibonacci(Integer num) {
12     return null;
13   }
14   @Perm(requires="pure(#0) in alive",
15  ensures="pure(#0) in alive")
16   void display(Integer num) {
17   }
18   @Perm(requires="none(this) in alive",
19  ensures="unique(this) in alive")
20   void main(String[] args) {
21   }
22 }
23 }ENDOFCLASS
```