

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
SeqIntegral	5	1	0	0	2	15	3	20
Total Classes=1	5	1	0	0	2	15	3	20

Contents

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

1 SeqIntegral

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqIntegral	✓
compute	✓
main	✓
display	✓
f	✓

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	SeqIntegral	compute	main	display	f
SeqIntegral	⌘	⌘	⌘	⌘	⌘
compute	⌘	⌘	⌘	⌘	⌘
main	⌘	⌘	⌘	⌘	⌘
display	⌘	⌘	⌘		
f	⌘	⌘	⌘		

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 SeqIntegral {
6   @Perm(ensures="unique(this) in alive")
7   SeqIntegral() { }
8
9   @Perm(requires="full(this) in alive",
10  ensures="full(this) in alive")
11   Double compute(Double x1, Double x2) {
12     return null;
13   }
14   @Perm(requires="unique(this) in alive",
15  ensures="unique(this) in alive")
16   void main(String[] args) {
17   }
18   @Perm(requires="pure(this) in alive",
19  ensures="pure(this) in alive")
20   void display(Double area) {
21   }
22   @Perm(requires="pure(this) in alive",
23  ensures="pure(this) in alive")
24   Double f(final Double x1) {
25     return null;
26   }
27
28 }ENDOFCLASS
```