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
SeqIntegral	5	1	0	0	4	15	10	67
Total Classes=1	5	1	0	0	4	15	10	67

Contents

1 SeqIntegral

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqIntegral	√
compute	√
main	√
f	√
display	√

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	SeqIntegral	compute	main	f	display
SeqIntegral	∦	∦	∦	∦	∦
compute	∦	∥	∥	∥	∥
main	∦	∥	∥	∥	∥
f	∦	∥	∥	∥	∥
display	∦	∥	∥	∥	∥

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

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

4  @ClassStates({@State(name = "alive")})
5  class SeqIntegral {
6  @Perm(ensures="unique(this) in alive")
7  SeqIntegral() {   }

9  @Perm(requires="full(this) * pure(#0) * pure(#1) in alive",
10 ensures="full(this) * pure(#0) * pure(#1) in alive")
11  Double compute(Double x1, Double x2) {
12  return null;
13 }
14 @Perm(requires="none(this) in alive",
15 ensures="unique(this) in alive")
16  void main(String[] args) {
17 }
18 @Perm(requires="pure(#0) in alive",
19 ensures="pure(#0) in alive")
20  Double f(final Double x1) {
21  return null;
22 }
23 @Perm(requires="pure(#0) in alive",
24 ensures="pure(#0) in alive")
25  void display(Double area) {
26 }

28 }ENDOFCLASS
```