Summary

Sink States:$0(0 \times 10^0)$

Table 1: Sip4J Analysis Summary

Classes	Methods	States	Unreachable clauses	Unreachable states	Possible concurrent methods	Total. no. of method pairs	No. of concurrent method pairs	Percentage of concurrent methods pairs
SeqIntegral	5	1	0	0	2	15	3	20
Total Classes=1	5	1	0	0	2	15	3	20

Contents

1 SeqIntegral

Table 2: Method's Satisfiability(Code Reachabiity Analysis

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
$\surd$	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 the input 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="share(this) in alive",
10 ensures="share(this) in alive")
11   Double compute(Double x1, Double x2) {
12  return null;

14 }
15 @Perm(requires="unique(this) in alive",
16 ensures="unique(this) in alive")
17   void main(String[] args) {

19 }
20 @Perm(requires="pure(this) in alive",
21 ensures="pure(this) in alive")
22   void display(Double area) {

24 }
25 @Perm(requires="immutable(this) in alive",
26 ensures="immutable(this) in alive")
27   Double f(final Double x1) {
28  return null;

30 }

32 }ENDOFCLASS
```