

# 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
Patient	1	1	0	0	0	1	0	0
Village	10	1	0	0	1	55	1	2
Health	2	1	0	0	0	3	0	0
SeqHealth	3	1	0	0	0	6	0	0
Results	1	1	0	0	0	1	0	0
Hosp	1	1	0	0	0	1	0	0
Total Classes=6	18	6	0	0	1	67	1	1

## Contents

<b>1</b>	<b>Patient</b>	<b>3</b>
<b>2</b>	<b>Village</b>	<b>4</b>
<b>3</b>	<b>Health</b>	<b>5</b>
<b>4</b>	<b>SeqHealth</b>	<b>6</b>
<b>5</b>	<b>Results</b>	<b>7</b>
<b>6</b>	<b>Hosp</b>	<b>8</b>
<b>7</b>	<b>Abbreviation</b>	<b>9</b>
<b>8</b>	<b>Annotated version of the input program generated by Sip4J</b>	<b>10</b>

# 1 Patient

Table 2: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Patient	✓

Table 3: State Transition Matrix

	alive
alive	↑

## 2 Village

Table 4: Method's Satisfiability(Code Reachability Analysis)

Method	Satisfiability
Village	✓
tick	✓
checkPatientsInside	✓
checkPatientsAssess	✓
checkPatientsWaiting	✓
checkPatientsRealloc	✓
putInHosp	✓
checkPatientsPopulation	✓
displayVillageData	✓
DisplayVillagePatients	✓

Table 5: State Transition Matrix

	alive
alive	↑

Table 6: Methods Concurrency Matrix

	Village	tick	checkPatientsInside	checkPatientsAssess	checkPatientsWaiting	checkPatientsRealloc	putInHosp	checkPatientsPopulation	displayVillageData	DisplayVillagePatients
Village	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
tick	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
checkPatientsInside	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
checkPatientsAssess	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
checkPatientsWaiting	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
checkPatientsRealloc	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
putInHosp	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
checkPatientsPopulation	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
displayVillageData	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
DisplayVillagePatients	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈

### 3 Health

Table 7: Method's Satisfiability(Code Reachability Analysis)

Method	Satisfiability
Health	✓
allocateVillage	✓

Table 8: State Transition Matrix

	alive
alive	↑

Table 9: Methods Concurrency Matrix

	Health	allocateVillage
Health	⌋	⌋
allocateVillage	⌋	⌋

## 4 SeqHealth

Table 10: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
SeqHealth	✓
main	✓
simVillage	✓

Table 11: State Transition Matrix

	alive
alive	↑

Table 12: Methods Concurrency Matrix

	SeqHealth	main	simVillage
SeqHealth	⌈	⌈	⌈
main	⌈	⌈	⌈
simVillage	⌈	⌈	⌈

5 Results

Table 13: Method’s Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Results	✓

Table 14: State Transition Matrix

	alive
alive	↑

## 6 Hosp

Table 15: Method's Satisfiability(Code Reachabiity Analysis

Method	Satisfiability
Hosp	✓

Table 16: State Transition Matrix

	alive
alive	↑



## 7 Abbreviation

Table 17: 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

## 8 Annotated version of the input program generated by Sip4J

```
1 package outputs;
2 import edu.cmu.cs.plural.annot.*;
3
4 @ClassStates({@State(name = "alive")})
5 class Patient {
6   @Perm(ensures="unique(this) in alive")
7   Patient() { }
8
9 }
10 }ENDOFCLASS
11
12 @ClassStates({@State(name = "alive")})
13
14 class Village {
15   @Perm(ensures="unique(this) in alive")
16   Village() { }
17
18   @Perm(requires="share(this) in alive",
19   ensures="share(this) in alive")
20   public void tick() {
21
22   }
23   @Perm(requires="share(this) in alive",
24   ensures="share(this) in alive")
25   public void checkPatientsInside() {
26
27   }
28   @Perm(requires="share(this) in alive",
29   ensures="share(this) in alive")
30   public void checkPatientsAssess() {
31
32   }
33   @Perm(requires="share(this) in alive",
34   ensures="share(this) in alive")
35   public void checkPatientsWaiting() {
36
37   }
38   @Perm(requires="share(this) in alive",
39   ensures="share(this) in alive")
40   public void checkPatientsRealloc() {
41
42   }
43   @Perm(requires="share(this) in alive",
44   ensures="share(this) in alive")
45   public void putInHosp(Patient p) {
46
47   }
48   @Perm(requires="share(this) in alive",
49   ensures="share(this) in alive")
50   public void checkPatientsPopulation() {
51
52   }
53   @Perm(requires="share(this) in alive",
54   ensures="share(this) in alive")
55   void displayVillageData(Village v) {
56
57   }
58   @Perm(requires="pure(this) in alive",
59   ensures="pure(this) in alive")
60   static void DisplayVillagePatients(Village v) {
61
62   }
63 }
64 }ENDOFCLASS
65
66 @ClassStates({@State(name = "alive")})
67
68 class Health {
69   @Perm(ensures="unique(this) in alive")
70   Health() { }
71
72   @Perm(requires="unique(this) in alive",
73   ensures="unique(this) in alive")
74   Village allocateVillage(int level, int vid, Village back) {
75     return null;
76   }
77 }
```

```

79 }ENDOFCLASS
81 @ClassStates({@State(name = "alive")})
83 class SeqHealth {
84   @Perm(ensures="unique(this) in alive")
85   SeqHealth() { }
87   @Perm(requires="unique(this) in alive",
88     ensures="unique(this) in alive")
89   void main(String[] args) {
91   }
92   @Perm(requires="share(this) in alive",
93     ensures="share(this) in alive")
94   void simVillage(Village village) {
96   }
98 }ENDOFCLASS
100 @ClassStates({@State(name = "alive")})
102 class Results {
103   @Perm(ensures="unique(this) in alive")
104   Results() { }
107 }ENDOFCLASS
109 @ClassStates({@State(name = "alive")})
111 class Hosp {
112   @Perm(ensures="unique(this) in alive")
113   Hosp() { }
116 }ENDOFCLASS

```