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

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 Reachabiity 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 Reachabiity 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
$\surd$	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

8 Annotated version of the input program generated by Sip4J

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

@ClassStates({@State(name = "alive")})
class Patient {
@Perm(ensures="unique(this) in alive")
Patient() {   }


}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class Village {
@Perm(ensures="unique(this) in alive")
Village() {   }

@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void tick() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void checkPatientsInside() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void checkPatientsAssess() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void checkPatientsWaiting() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void checkPatientsRealloc() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void putInHosp(Patient p) {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
 public void checkPatientsPopulation() {

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void displayVillageData(Village v) {

}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 static void DisplayVillagePatients(Village v) {

}

}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class Health {
@Perm(ensures="unique(this) in alive")
Health() {   }

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
  Village allocateVillage(int level, int vid, Village back) {
 return null;

}
```

```
}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class SeqHealth {
@Perm(ensures="unique(this) in alive")
SeqHealth() {   }

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

}
@Perm(requires="share(this) in alive",
ensures="share(this) in alive")
  void simVillage(Village village) {

}

}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class Results {
@Perm(ensures="unique(this) in alive")
Results() {   }


}ENDOFCLASS

@ClassStates({@State(name = "alive")})

class Hosp {
@Perm(ensures="unique(this) in alive")
Hosp() {   }


}ENDOFCLASS
```