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
Item	1	1	0	0	0	1	0	0
SeqGA	2	1	0	0	0	3	0	0
Knapsack	7	1	0	0	1	28	1	4
MersenneTwisterFast	6	1	0	0	0	21	0	0
Indiv	3	1	0	0	1	6	1	17
ComparatorOnFitness	2	1	0	0	1	3	1	33
Total Classes=6	21	6	0	0	3	62	3	5

Contents

1 Item

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Item	$\surd$

Table 3: State Transition Matrix

	alive
alive	$\uparrow$

2 SeqGA

Table 4: Methods Requires Clause Satisfiability

Method	Satisfiability
SeqGA	$\surd$
main	$\surd$

Table 5: State Transition Matrix

	alive
alive	$\uparrow$

Table 6: Methods Concurrency Matrix

	SeqGA	main
SeqGA	$\nparallel$	$\nparallel$
main	$\nparallel$	$\nparallel$

3 Knapsack

Table 7: Methods Requires Clause Satisfiability

Method	Satisfiability
Knapsack	$\surd$
resetSeed	$\surd$
createRandomIndiv	$\surd$
evaluate	$\surd$
phenotype	$\surd$
recombine	$\surd$
mutate	$\surd$

Table 8: State Transition Matrix

	alive
alive	$\uparrow$

Table 9: Methods Concurrency Matrix

	Knapsack	resetSeed	createRandomIndiv	evaluate	phenotype	recombine	mutate
Knapsack	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
resetSeed	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
createRandomIndiv	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
evaluate	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
phenotype	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$
recombine	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
mutate	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$

4 MersenneTwisterFast

Table 10: Methods Requires Clause Satisfiability

Method	Satisfiability
MersenneTwisterFast	√
nextDouble	√
nextInt	√
nextFloat	√
setSeed	√
nextBoolean	√

Table 11: State Transition Matrix

	alive
alive	↑

Table 12: Methods Concurrency Matrix

	MersenneTwisterFast	nextDouble	nextInt	nextFloat	setSeed	nextBoolean
MersenneTwisterFast	∦	∦	∦	∦	∦	∦
nextDouble	∦	∦	∦	∦	∦	∦
nextInt	∦	∦	∦	∦	∦	∦
nextFloat	∦	∦	∦	∦	∦	∦
setSeed	∦	∦	∦	∦	∦	∦
nextBoolean	∦	∦	∦	∦	∦	∦

5 Indiv

Table 13: Methods Requires Clause Satisfiability

Method	Satisfiability
Indiv	$\surd$
set	$\surd$
compareTo	$\surd$

Table 14: State Transition Matrix

	alive
alive	$\uparrow$

Table 15: Methods Concurrency Matrix

	Indiv	set	compareTo
Indiv	$\nparallel$	$\nparallel$	$\nparallel$
set	$\nparallel$	$\nparallel$	$\nparallel$
compareTo	$\nparallel$	$\nparallel$	$\parallel$

6 ComparatorOnFitness

Table 16: Methods Requires Clause Satisfiability

Method	Satisfiability
ComparatorOnFitness	$\surd$
compare	$\surd$

Table 17: State Transition Matrix

	alive
alive	$\uparrow$

Table 18: Methods Concurrency Matrix

	ComparatorOnFitness	compare
ComparatorOnFitness	$\nparallel$	$\nparallel$
compare	$\nparallel$	$\parallel$

7 Abbreviation

Table 19: 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 Sequential Java Program generated by Sip4j

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

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


}ENDOFCLASS

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

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

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

}ENDOFCLASS

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

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

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 void resetSeed() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 Indiv createRandomIndiv(Indiv ind) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void evaluate(Indiv indiv) {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 int[] phenotype(Indiv indiv) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 Indiv recombine(Indiv ind, Indiv p1, Indiv p2) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 void mutate(Indiv indiv) {
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 double nextDouble() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 int nextInt(final int n) {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 float nextFloat() {
 return 0;
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
 void setSeed(final long seed) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
 boolean nextBoolean() {
 return 0;
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void set(int w, boolean h) {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public int compareTo(Indiv other) {
 return 0;
}

}ENDOFCLASS

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

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

@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public int compare(Integer a, Integer b) {
 return 0;
}

}ENDOFCLASS
```