

# 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
Strength	7	1	0	0	6	28	21	75
UnaryConstraint	7	1	0	0	6	28	11	39
BinaryConstraint	9	1	0	0	8	45	21	47
Variable	5	1	0	0	3	15	3	20
Planner	8	1	0	0	7	36	6	17
Constraint	16	1	0	0	15	136	119	88
DeltaBlue	6	1	0	0	4	21	10	48
Plan	5	1	0	0	0	15	0	0
EditConstraint	2	1	0	0	1	3	1	33
StayConstraint	1	1	0	0	0	1	0	0
EqualityConstraint	1	1	0	0	0	1	0	0
ScaleConstraint	4	1	0	0	0	10	0	0
Total Classes=12	71	12	0	0	50	339	192	57

## Contents

<b>1</b>	<b>Strength</b>	<b>3</b>
<b>2</b>	<b>UnaryConstraint</b>	<b>4</b>
<b>3</b>	<b>BinaryConstraint</b>	<b>5</b>
<b>4</b>	<b>Variable</b>	<b>6</b>
<b>5</b>	<b>Planner</b>	<b>7</b>
<b>6</b>	<b>Constraint</b>	<b>8</b>
<b>7</b>	<b>DeltaBlue</b>	<b>10</b>
<b>8</b>	<b>Plan</b>	<b>11</b>
<b>9</b>	<b>EditConstraint</b>	<b>12</b>
<b>10</b>	<b>StayConstraint</b>	<b>13</b>
<b>11</b>	<b>EqualityConstraint</b>	<b>14</b>
<b>12</b>	<b>ScaleConstraint</b>	<b>15</b>
<b>13</b>	<b>Abbreviation</b>	<b>16</b>
<b>14</b>	<b>Annotated Version of Sequential Java Program generated by Sip4j</b>	<b>17</b>

# 1 Strength

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Strength	✓
print	✓
stronger	✓
strongest	✓
weaker	✓
weakestOf	✓
nextWeaker	✓

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	Strength	print	stronger	strongest	weaker	weakestOf	nextWeaker
Strength	⌘	⌘	⌘	⌘	⌘	⌘	⌘
print	⌘	⌘	⌘	⌘	⌘	⌘	⌘
stronger	⌘	⌘	⌘	⌘	⌘	⌘	⌘
strongest	⌘	⌘	⌘	⌘	⌘	⌘	⌘
weaker	⌘	⌘	⌘	⌘	⌘	⌘	⌘
weakestOf	⌘	⌘	⌘	⌘	⌘	⌘	⌘
nextWeaker	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 2 UnaryConstraint

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
UnaryConstraint	✓
chooseMethod	✓
markUnsatisfied	✓
output	✓
addToGraph	✓
inputsKnown	✓
recalculate	✓

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	UnaryConstraint	chooseMethod	markUnsatisfied	output	addToGraph	inputsKnown	recalculate
UnaryConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘
chooseMethod	⌘	⌘	⌘	⌘	⌘	⌘	⌘
markUnsatisfied	⌘	⌘	⌘	⌘	⌘	⌘	⌘
output	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addToGraph	⌘	⌘	⌘	⌘	⌘	⌘	⌘
inputsKnown	⌘	⌘	⌘	⌘	⌘	⌘	⌘
recalculate	⌘	⌘	⌘	⌘	⌘	⌘	⌘

### 3 BinaryConstraint

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
BinaryConstraint	✓
chooseMethod	✓
isSatisfied	✓
addToGraph	✓
markInputs	✓
input	✓
inputsKnown	✓
output	✓
recalculate	✓

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	BinaryConstraint	chooseMethod	isSatisfied	addToGraph	markInputs	input	inputsKnown	output	recalculate
BinaryConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
chooseMethod	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isSatisfied	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addToGraph	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
markInputs	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
input	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
inputsKnown	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
output	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
recalculate	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 4 Variable

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
Variable	✓
print	✓
removeConstraint	✓
setValue	✓
addConstraint	✓

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	Variable	print	removeConstraint	setValue	addConstraint
Variable	✗	✗	✗	✗	✗
print	✗		✗		
removeConstraint	✗	✗	✗	✗	✗
setValue	✗		✗	✗	✗
addConstraint	✗		✗	✗	✗

## 5 Planner

Table 14: Methods Requires Clause Satisfiability

Method	Satisfiability
Planner	✓
propagateFrom	✓
addConstraintsConsumingTo	✓
addPropagate	✓
makePlan	✓
newMark	✓
incrementalAdd	✓
extractPlanFromConstraints	✓

Table 15: State Transition Matrix

	alive
alive	↑

Table 16: Methods Concurrency Matrix

	Planner	propagateFrom	addConstraintsConsumingTo	addPropagate	makePlan	newMark	incrementalAdd	extractPlanFromConstraints
Planner	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
propagateFrom	⌘	⌘	⌘		⌘	⌘	⌘	⌘
addConstraintsConsumingTo	⌘	⌘	⌘		⌘	⌘	⌘	⌘
addPropagate	⌘			⌘				
makePlan	⌘	⌘	⌘		⌘	⌘	⌘	⌘
newMark	⌘	⌘	⌘		⌘	⌘	⌘	⌘
incrementalAdd	⌘	⌘	⌘		⌘	⌘	⌘	⌘
extractPlanFromConstraints	⌘	⌘	⌘		⌘	⌘	⌘	⌘

## 6 Constraint

Table 17: Methods Requires Clause Satisfiability

Method	Satisfiability
Constraint	✓
satisfy	✓
chooseMethod	✓
isSatisfied	✓
output	✓
markUnsatisfied	✓
destroyConstraint	✓
recalculate	✓
inputsKnown	✓
execute	✓
isInput	✓
addConstraint	✓
addToGraph	✓
printOutput	✓
print	✓
printInputs	✓

Table 18: State Transition Matrix

	alive
alive	↑

Table 19: Methods Concurrency Matrix

	Constraint	satisfy	chooseMethod	isSatisfied	output	markUnsatisfied	destroyConstraint	recalculate	inputsKnown	execute	isInput	addConstraint	addToGraph	printOutput	print	printInputs
Constraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
satisfy	⌘	⌘														
chooseMethod	⌘															
isSatisfied	⌘															
output	⌘															
markUnsatisfied	⌘															
destroyConstraint	⌘															
recalculate	⌘															
inputsKnown	⌘															
execute	⌘															
isInput	⌘															
addConstraint	⌘															
addToGraph	⌘															
printOutput	⌘															



print	<del>  </del>																
printInputs	<del>  </del>																

## 7 DeltaBlue

Table 20: Methods Requires Clause Satisfiability

Method	Satisfiability
DeltaBlue	✓
error	✓
chainTest	✓
change	✓
inst_main	✓
main	✓

Table 21: State Transition Matrix

	alive
alive	↑

Table 22: Methods Concurrency Matrix

	DeltaBlue	error	chainTest	change	inst_main	main
DeltaBlue	⌘	⌘	⌘	⌘	⌘	⌘
error	⌘				⌘	
chainTest	⌘				⌘	
change	⌘				⌘	
inst_main	⌘	⌘	⌘	⌘	⌘	⌘
main	⌘				⌘	

## 8 Plan

Table 23: Methods Requires Clause Satisfiability

Method	Satisfiability
Plan	✓
addConstraint	✓
size	✓
constraintAt	✓
execute	✓

Table 24: State Transition Matrix

	alive
alive	↑

Table 25: Methods Concurrency Matrix

	Plan	addConstraint	size	constraintAt	execute
Plan	⌘	⌘	⌘	⌘	⌘
addConstraint	⌘	⌘	⌘	⌘	⌘
size	⌘	⌘	⌘	⌘	⌘
constraintAt	⌘	⌘	⌘	⌘	⌘
execute	⌘	⌘	⌘	⌘	⌘

## 9 EditConstraint

Table 26: Methods Requires Clause Satisfiability

Method	Satisfiability
EditConstraint	✓
execute	✓

Table 27: State Transition Matrix

	alive
alive	↑

Table 28: Methods Concurrency Matrix

	EditConstraint	execute
EditConstraint	⦿	⦿
execute	⦿	⦿

## 10 StayConstraint

Table 29: Methods Requires Clause Satisfiability

Method	Satisfiability
StayConstraint	✓

Table 30: State Transition Matrix

	alive
alive	↑

## 11 EqualityConstraint

Table 31: Methods Requires Clause Satisfiability

Method	Satisfiability
EqualityConstraint	✓

Table 32: State Transition Matrix

	alive
alive	↑

## 12 ScaleConstraint

Table 33: Methods Requires Clause Satisfiability

Method	Satisfiability
ScaleConstraint	✓
removeFromGraph	✓
execute	✓
recalculate	✓

Table 34: State Transition Matrix

	alive
alive	↑

Table 35: Methods Concurrency Matrix

	ScaleConstraint	removeFromGraph	execute	recalculate
ScaleConstraint	⌘	⌘	⌘	⌘
removeFromGraph	⌘	⌘	⌘	⌘
execute	⌘	⌘	⌘	⌘
recalculate	⌘	⌘	⌘	⌘

## 13 Abbreviation

Table 36: 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



## 14 Annotated Version of Sequential Java Program generated by Sip4j

```
1 package outputs;
2 import edu.cmu.cs.plural.annot.*;
3
4 @ClassStates({@State(name = "alive")})
5 class Strength {
6   @Perm(ensures="unique(this) in alive")
7   Strength() { }
8
9   @Perm(requires="pure(this) in alive",
10  ensures="pure(this) in alive")
11   public void print() {
12   }
13   @Perm(requires="pure(this) in alive",
14  ensures="pure(this) in alive")
15   boolean stronger(Strength s1, Strength s2) {
16   return 0;
17   }
18   @Perm(requires="pure(this) in alive",
19  ensures="pure(this) in alive")
20   Strength strongest(Strength s1, Strength s2) {
21   return null;
22   }
23   @Perm(requires="pure(this) * pure(#0) * pure(#1) in alive",
24  ensures="pure(this) * pure(#0) * pure(#1) in alive")
25   boolean weaker(Strength s1, Strength s2) {
26   return 0;
27   }
28   @Perm(requires="pure(this) * pure(#0) * pure(#1) in alive",
29  ensures="pure(this) * pure(#0) * pure(#1) in alive")
30   Strength weakestOf(Strength s1, Strength s2) {
31   return null;
32   }
33   @Perm(requires="pure(this) in alive",
34  ensures="pure(this) in alive")
35   public Strength nextWeaker() {
36   return null;
37   }
38 }
39 }ENDOFCLASS
40
41 @ClassStates({@State(name = "alive")})
42
43 class UnaryConstraint {
44   @Perm(ensures="unique(this) in alive")
45   UnaryConstraint() { }
46
47   @Perm(requires="full(this) in alive",
48  ensures="full(this) in alive")
49   protected void chooseMethod(int mark) {
50   }
51   @Perm(requires="full(this) in alive",
52  ensures="full(this) in alive")
53   public void markUnsatisfied() {
54   }
55   @Perm(requires="pure(this) in alive",
56  ensures="pure(this) in alive")
57   public Variable output() {
58   return null;
59   }
60   @Perm(requires="full(this) in alive",
61  ensures="full(this) in alive")
62   public void addToGraph() {
63   }
64
65   public boolean inputsKnown(int mark) {
66   return 0;
67   }
68   @Perm(requires="full(this) in alive",
69  ensures="full(this) in alive")
70   public void recalculate() {
71   }
72 }
73 }ENDOFCLASS
74
75 @ClassStates({@State(name = "alive")})
```

```

77 class BinaryConstraint {
78   @Perm(ensures="unique(this) in alive")
79   BinaryConstraint() { }

81   @Perm(requires="full(this) in alive",
82     ensures="full(this) in alive")
83   protected void chooseMethod(int mark) {
84   }

85   @Perm(requires="pure(this) in alive",
86     ensures="pure(this) in alive")
87   public boolean isSatisfied() {
88     return 0;
89   }

90   @Perm(requires="full(this) in alive",
91     ensures="full(this) in alive")
92   public void addToGraph() {
93   }

94   @Perm(requires="full(this) in alive",
95     ensures="full(this) in alive")
96   protected void markInputs(int mark) {
97   }

98   @Perm(requires="pure(this) in alive",
99     ensures="pure(this) in alive")
100   public Variable input() {
101     return null;
102   }

103   @Perm(requires="pure(this) in alive",
104     ensures="pure(this) in alive")
105   public boolean inputsKnown(int mark) {
106     return 0;
107   }

108   @Perm(requires="full(this) in alive",
109     ensures="full(this) in alive")
110   public Variable output() {
111     return null;
112   }

113   @Perm(requires="full(this) in alive",
114     ensures="full(this) in alive")
115   public void recalculate() {
116   }

118 }ENDOFCLASS

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

122 class Variable {
123   @Perm(ensures="unique(this) in alive")
124   Variable() { }

126   @Perm(requires="pure(this) in alive",
127     ensures="pure(this) in alive")
128   public void print() {
129   }

130   @Perm(requires="unique(this) in alive",
131     ensures="unique(this) in alive")
132   public void removeConstraint(Constraint c) {
133   }

134   @Perm(requires="full(this) in alive",
135     ensures="full(this) in alive")
136   public void setValue(int value, Strength strength) {
137   }

138   @Perm(requires="full(this) in alive",
139     ensures="full(this) in alive")
140   public void addConstraint(Constraint c) {
141   }

143 }ENDOFCLASS

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

147 class Planner {
148   @Perm(ensures="unique(this) in alive")
149   Planner() { }

151   @Perm(requires="full(this) in alive",
152     ensures="full(this) in alive")
153   public void propagateFrom(Variable v) {
154   }

155   @Perm(requires="full(this) in alive",
156     ensures="full(this) in alive")

```

```

157 protected void addConstraintsConsumingTo(Variable v, ArrayList<Constraint> coll) {
158 }
159 @Perm(requires="pure(this) * full(#0) in alive",
160 ensures="pure(this) * full(#0) in alive")
161 public boolean addPropagate(Constraint c, int mark) {
162     return 0;
163 }
164 @Perm(requires="full(this) in alive",
165 ensures="full(this) in alive")
166 protected Plan makePlan(ArrayList<Constraint> sources) {
167     return null;
168 }
169 @Perm(requires="full(this) in alive",
170 ensures="full(this) in alive")
171 private int newMark() {
172     return 0;
173 }
174 @Perm(requires="full(this) in alive",
175 ensures="full(this) in alive")
176 public void incrementalAdd(Constraint c) {
177 }
178 @Perm(requires="full(this) in alive",
179 ensures="full(this) in alive")
180 protected Plan extractPlanFromConstraints(ArrayList<Constraint> constraints) {
181     return null;
182 }
183 }ENDOFCLASS
184
185 @ClassStates({@State(name = "alive")})
186
187 class Constraint {
188     @Perm(ensures="unique(this) in alive")
189     Constraint() { }
190
191     @Perm(requires="full(this) in alive",
192     ensures="full(this) in alive")
193     public Constraint satisfy(int mark) {
194         return null;
195     }
196 }
197
198 void chooseMethod(int mark) {
199 }
200
201 boolean isSatisfied() {
202     return 0;
203 }
204
205 Variable output() {
206     return null;
207 }
208
209 void markUnsatisfied() {
210 }
211 @Perm(requires="pure(this) in alive",
212 ensures="pure(this) in alive")
213 public void destroyConstraint() {
214 }
215
216 void recalculate() {
217 }
218
219 boolean inputsKnown(int mark) {
220     return 0;
221 }
222
223 void execute() {
224 }
225
226 public boolean isInput() {
227     return 0;
228 }
229 @Perm(requires="pure(this) in alive",
230 ensures="pure(this) in alive")
231 protected void addConstraint() {
232 }
233
234 void addToGraph() {
235 }
236 @Perm(requires="pure(this) in alive",
237 ensures="pure(this) in alive")

```

```

238 protected void printOutput() {
239 }
240 @Perm(requires="pure(this) in alive",
241 ensures="pure(this) in alive")
242 public void print() {
243 }
244
245 void printInputs() {
246 }
247
248 }ENDOFCLASS
249
250 @ClassStates({@State(name = "alive")})
251
252 class DeltaBlue {
253 @Perm(ensures="unique(this) in alive")
254 DeltaBlue() { }
255
256
257 void error(String s) {
258 }
259 @Perm(requires="none(this) in alive",
260 ensures="unique(this) in alive")
261 private void chainTest() {
262 }
263 @Perm(requires="none(this) in alive",
264 ensures="unique(this) in alive")
265 private void change(Variable var, int newValue) {
266 }
267 @Perm(requires="none(this) * pure(#0) in alive",
268 ensures="unique(this) * pure(#0) in alive")
269 public void inst_main(int n) {
270 }
271 @Perm(requires="none(this) in alive",
272 ensures="unique(this) in alive")
273 void main(String[] args) {
274 }
275
276 }ENDOFCLASS
277
278 @ClassStates({@State(name = "alive")})
279
280 class Plan {
281 @Perm(ensures="unique(this) in alive")
282 Plan() { }
283
284 @Perm(requires="full(this) in alive",
285 ensures="full(this) in alive")
286 public void addConstraint(Constraint c) {
287 }
288 @Perm(requires="full(this) in alive",
289 ensures="full(this) in alive")
290 public int size() {
291 return 0;
292 }
293 @Perm(requires="full(this) in alive",
294 ensures="full(this) in alive")
295 public Constraint constraintAt(int index) {
296 return null;
297 }
298 @Perm(requires="full(this) in alive",
299 ensures="full(this) in alive")
300 public void execute() {
301 }
302
303 }ENDOFCLASS
304
305 @ClassStates({@State(name = "alive")})
306
307 class EditConstraint {
308 @Perm(ensures="unique(this) in alive")
309 EditConstraint() { }
310
311
312 public void execute() {
313 }
314
315 }ENDOFCLASS
316
317 @ClassStates({@State(name = "alive")})

```

```

319 class StayConstraint {
320   @Perm(ensures="unique(this) in alive")
321   StayConstraint() { }
322
323 }ENDOFCLASS
324
325 @ClassStates({@State(name = "alive")})
326
327 class EqualityConstraint {
328   @Perm(ensures="unique(this) in alive")
329   EqualityConstraint() { }
330
331 }ENDOFCLASS
332
333 @ClassStates({@State(name = "alive")})
334
335 class ScaleConstraint {
336   @Perm(ensures="unique(this) in alive")
337   ScaleConstraint() { }
338
339   @Perm(requires="unique(this) in alive",
340         ensures="unique(this) in alive")
341   public void removeFromGraph() {
342   }
343   @Perm(requires="full(this) in alive",
344         ensures="full(this) in alive")
345   public void execute() {
346   }
347   @Perm(requires="full(this) in alive",
348         ensures="full(this) in alive")
349   public void recalculate() {
350   }
351
352 }ENDOFCLASS
353

```