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

1 Strength

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
Strength	$\surd$
print	$\surd$
stronger	$\surd$
strongest	$\surd$
weaker	$\surd$
weakestOf	$\surd$
nextWeaker	$\surd$

Table 3: State Transition Matrix

	alive
alive	$\uparrow$

Table 4: Methods Concurrency Matrix

	Strength	print	stronger	strongest	weaker	weakestOf	nextWeaker
Strength	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
print	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
stronger	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
strongest	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
weaker	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
weakestOf	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
nextWeaker	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

2 UnaryConstraint

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
UnaryConstraint	$\surd$
chooseMethod	$\surd$
markUnsatisfied	$\surd$
output	$\surd$
addToGraph	$\surd$
inputsKnown	$\surd$
recalculate	$\surd$

Table 6: State Transition Matrix

	alive
alive	$\uparrow$

Table 7: Methods Concurrency Matrix

	UnaryConstraint	chooseMethod	markUnsatisfied	output	addToGraph	inputsKnown	recalculate
UnaryConstraint	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
chooseMethod	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$
markUnsatisfied	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$
output	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
addToGraph	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$
inputsKnown	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
recalculate	$\nparallel$	$\nparallel$	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\nparallel$

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	$\surd$
print	$\surd$
removeConstraint	$\surd$
setValue	$\surd$
addConstraint	$\surd$

Table 12: State Transition Matrix

	alive
alive	$\uparrow$

Table 13: Methods Concurrency Matrix

	Variable	print	removeConstraint	setValue	addConstraint
Variable	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
print	$\nparallel$	$\parallel$	$\nparallel$	$\parallel$	$\parallel$
removeConstraint	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
setValue	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$
addConstraint	$\nparallel$	$\parallel$	$\nparallel$	$\nparallel$	$\nparallel$

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	$\surd$
satisfy	$\surd$
chooseMethod	$\surd$
isSatisfied	$\surd$
output	$\surd$
markUnsatisfied	$\surd$
destroyConstraint	$\surd$
recalculate	$\surd$
inputsKnown	$\surd$
execute	$\surd$
isInput	$\surd$
addConstraint	$\surd$
addToGraph	$\surd$
printOutput	$\surd$
print	$\surd$
printInputs	$\surd$

Table 18: State Transition Matrix

	alive
alive	$\uparrow$

Table 19: Methods Concurrency Matrix

	Constraint	satisfy	chooseMethod	isSatisfied	output	markUnsatisfied	destroyConstraint	recalculate	inputsKnown	execute	isInput	addConstraint	addToGraph	printOutput	print	printInputs
Constraint	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
satisfy	$\nparallel$	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
chooseMethod	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
isSatisfied	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
output	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
markUnsatisfied	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
destroyConstraint	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
recalculate	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
inputsKnown	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
execute	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
isInput	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
addConstraint	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
addToGraph	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
printOutput	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

print	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
printInputs	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

7 DeltaBlue

Table 20: Methods Requires Clause Satisfiability

Method	Satisfiability
DeltaBlue	$\surd$
error	$\surd$
chainTest	$\surd$
change	$\surd$
inst_main	$\surd$
main	$\surd$

Table 21: State Transition Matrix

	alive
alive	$\uparrow$

Table 22: Methods Concurrency Matrix

	DeltaBlue	error	chainTest	change	inst_main	main
DeltaBlue	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
error	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$
chainTest	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$
change	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$
inst_main	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
main	$\nparallel$	$\parallel$	$\parallel$	$\parallel$	$\nparallel$	$\parallel$

8 Plan

Table 23: Methods Requires Clause Satisfiability

Method	Satisfiability
Plan	$\surd$
addConstraint	$\surd$
size	$\surd$
constraintAt	$\surd$
execute	$\surd$

Table 24: State Transition Matrix

	alive
alive	$\uparrow$

Table 25: Methods Concurrency Matrix

	Plan	addConstraint	size	constraintAt	execute
Plan	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
addConstraint	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
size	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
constraintAt	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
execute	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$

9 EditConstraint

Table 26: Methods Requires Clause Satisfiability

Method	Satisfiability
EditConstraint	$\surd$
execute	$\surd$

Table 27: State Transition Matrix

	alive
alive	$\uparrow$

Table 28: Methods Concurrency Matrix

	EditConstraint	execute
EditConstraint	$\nparallel$	$\nparallel$
execute	$\nparallel$	$\parallel$

10 StayConstraint

Table 29: Methods Requires Clause Satisfiability

Method	Satisfiability
StayConstraint	$\surd$

Table 30: State Transition Matrix

	alive
alive	$\uparrow$

11 EqualityConstraint

Table 31: Methods Requires Clause Satisfiability

Method	Satisfiability
EqualityConstraint	$\surd$

Table 32: State Transition Matrix

	alive
alive	$\uparrow$

12 ScaleConstraint

Table 33: Methods Requires Clause Satisfiability

Method	Satisfiability
ScaleConstraint	$\surd$
removeFromGraph	$\surd$
execute	$\surd$
recalculate	$\surd$

Table 34: State Transition Matrix

	alive
alive	$\uparrow$

Table 35: Methods Concurrency Matrix

	ScaleConstraint	removeFromGraph	execute	recalculate
ScaleConstraint	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
removeFromGraph	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
execute	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$
recalculate	$\nparallel$	$\nparallel$	$\nparallel$	$\nparallel$

13 Abbreviation

Table 36: 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

14 Annotated Version of Sequential Java Program generated by Sip4j

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

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

@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public void print() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 boolean stronger(Strength s1, Strength s2) {
 return 0;
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
 Strength strongest(Strength s1, Strength s2) {
 return null;
}
@Perm(requires="pure(this) * pure(#0) * pure(#1) in alive",
ensures="pure(this) * pure(#0) * pure(#1) in alive")
 boolean weaker(Strength s1, Strength s2) {
 return 0;
}
@Perm(requires="pure(this) * pure(#0) * pure(#1) in alive",
ensures="pure(this) * pure(#0) * pure(#1) in alive")
 Strength weakestOf(Strength s1, Strength s2) {
 return null;
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public Strength nextWeaker() {
 return null;
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
protected void chooseMethod(int mark) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void markUnsatisfied() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public Variable output() {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void addToGraph() {
}

public boolean inputsKnown(int mark) {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void recalculate() {
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
protected void chooseMethod(int mark) {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public boolean isSatisfied() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void addToGraph() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
protected void markInputs(int mark) {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public Variable input() {
 return null;
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public boolean inputsKnown(int mark) {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public Variable output() {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void recalculate() {
}

}ENDOFCLASS

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

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

@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public void print() {
}
@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
public void removeConstraint(Constraint c) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void setValue(int value, Strength strength) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void addConstraint(Constraint c) {
}

}ENDOFCLASS

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

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

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

```
protected void addConstraintsConsumingTo(Variable v, ArrayList<Constraint> coll) {
}
@Perm(requires="pure(this) * full(#0) in alive",
ensures="pure(this) * full(#0) in alive")
public boolean addPropagate(Constraint c, int mark) {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
protected Plan makePlan(ArrayList<Constraint> sources) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
private int newMark() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void incrementalAdd(Constraint c) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
protected Plan extractPlanFromConstraints(ArrayList<Constraint> constraints) {
 return null;
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public Constraint satisfy(int mark) {
 return null;
}

 void chooseMethod(int mark) {
}

 boolean isSatisfied() {
 return 0;
}

 Variable output() {
 return null;
}

 void markUnsatisfied() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public void destroyConstraint() {
}

 void recalculate() {
}

 boolean inputsKnown(int mark) {
 return 0;
}

 void execute() {
}

public boolean isInput() {
 return 0;
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
protected void addConstraint() {
}

 void addToGraph() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
```

```
protected void printOutput() {
}
@Perm(requires="pure(this) in alive",
ensures="pure(this) in alive")
public void print() {
}

 void printInputs() {
}

}ENDOFCLASS

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

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


 void error(String s) {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
private void chainTest() {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
private void change(Variable var, int newValue) {
}
@Perm(requires="none(this) * pure(#0) in alive",
ensures="unique(this) * pure(#0) in alive")
public void inst_main(int n) {
}
@Perm(requires="none(this) in alive",
ensures="unique(this) in alive")
 void main(String[] args) {
}

}ENDOFCLASS

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

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

@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void addConstraint(Constraint c) {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public int size() {
 return 0;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public Constraint constraintAt(int index) {
 return null;
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void execute() {
}

}ENDOFCLASS

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

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


public void execute() {
}

}ENDOFCLASS

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

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


}ENDOFCLASS

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

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


}ENDOFCLASS

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

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

@Perm(requires="unique(this) in alive",
ensures="unique(this) in alive")
public void removeFromGraph() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void execute() {
}
@Perm(requires="full(this) in alive",
ensures="full(this) in alive")
public void recalculate() {
}

}ENDOFCLASS
```