

# 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
ConsListTest	2	1	0	0	0	3	0	0
ConsList	28	1	0	0	27	406	257	63
Triple	5	1	0	0	5	15	13	87
Pair	10	1	0	0	9	55	19	35
Empty	11	1	0	0	10	66	52	79
Utilities	8	1	0	0	7	36	22	61
CollectionMethods	7	1	0	0	6	28	6	21
Nonempty	13	1	0	0	12	91	69	76
Anonymous	10	1	0	0	9	55	45	82
PluralParseError	1	1	0	0	0	1	0	0
ImpossibleConstraint	3	1	0	0	2	6	2	33
ZeroFraction	4	1	0	0	3	10	5	50
AbstractFractionTermVisitor	7	1	0	0	6	28	20	71
VariableFraction	8	1	0	0	7	36	22	61
FractionConstraints	28	1	0	0	27	406	48	12
FractionConstraint	5	1	0	0	4	15	10	67
FractionAssignment	23	1	0	0	22	276	28	10
FractionRelation	8	1	0	0	3	36	6	17
Fraction	14	1	0	0	13	105	76	72
FractionSum	6	1	0	0	2	21	3	14
NamedFractionMapping	2	1	0	0	0	3	0	0
NamedFraction	9	1	0	0	8	45	14	31
VariableElimination	14	1	0	0	13	105	16	15
NormalizedFractionConstraint	15	1	0	0	6	120	21	18
Rational	22	1	0	0	17	253	153	60
GeneralizedSum	8	1	0	0	1	36	1	3
VariableRelativity	5	1	0	0	0	15	0	0
NormalizedFractionSum	2	1	0	0	1	3	1	33
SmtLibPrinter	4	1	0	0	4	10	7	70
SmtLibBenchmarkPrinter	10	1	0	0	1	55	1	2
Anonymous	5	1	1	0	4	15	10	67
OneFraction	4	1	0	0	3	10	5	50
FractionTerm	4	1	0	0	3	10	6	60
Anonymous	3	1	1	0	2	6	3	50

Anonymous	5	1	5	0	4	15	10	67
Anonymous	5	1	5	0	4	15	10	67
Relop	2	1	0	0	2	3	2	67
Anonymous	5	1	5	0	4	15	10	67
RelationFractionPair	8	1	0	0	5	36	15	42
Anonymous	9	1	3	0	8	45	36	80
SimpleFractionSum	8	1	0	0	7	36	13	36
Sumop	2	1	0	0	2	3	2	67
SimpleVariableRelativity	4	1	0	0	1	10	1	10
Anonymous	6	1	6	0	5	21	15	71
FractionElimination	8	1	0	0	7	36	8	22
FractionPair	5	1	0	0	0	15	0	0
Anonymous	9	1	9	0	8	45	36	80
Anonymous	3	1	1	0	2	6	3	50
SmtLibConstraintProcessor	10	1	0	0	9	55	25	45
Impossible	4	1	0	0	3	10	6	60
Total Classes=50	401	50	36	0	308	2747	1133	41

## Contents

<b>1</b>	<b>MethodFlowAnalysis</b>	<b>4</b>
<b>2</b>	<b>WorklistNodeOrderComparator</b>	<b>5</b>
<b>3</b>	<b>BranchInsensitiveWorklist</b>	<b>6</b>
<b>4</b>	<b>SingleResult</b>	<b>7</b>
<b>5</b>	<b>WorklistTemplate</b>	<b>8</b>
<b>6</b>	<b>AnalysisResult</b>	<b>9</b>
<b>7</b>	<b>NormalLabel</b>	<b>10</b>
<b>8</b>	<b>IncomingResult</b>	<b>11</b>
<b>9</b>	<b>BooleanLabel</b>	<b>12</b>
<b>10</b>	<b>BranchSensitiveWorklist</b>	<b>13</b>
<b>11</b>	<b>WorklistFactory</b>	<b>14</b>
<b>12</b>	<b>ConcurrentFlowAnalysis</b>	<b>15</b>
<b>13</b>	<b>FlowAnalysis</b>	<b>16</b>
<b>14</b>	<b>Anonymous</b>	<b>17</b>
<b>15</b>	<b>Utilities</b>	<b>18</b>
<b>16</b>	<b>AbstractWorklist</b>	<b>19</b>
<b>17</b>	<b>EclipseNodeFirstCFG</b>	<b>20</b>
<b>18</b>	<b>EclipseCFG</b>	<b>21</b>
<b>19</b>	<b>ExceptionMap</b>	<b>22</b>
<b>20</b>	<b>RunCrystalHandler</b>	<b>23</b>
<b>21</b>	<b>AbstractCrystalPlugin</b>	<b>24</b>
<b>22</b>	<b>Anonymous</b>	<b>25</b>
<b>23</b>	<b>WorkspaceUtilities</b>	<b>26</b>
<b>24</b>	<b>Crystal</b>	<b>27</b>
<b>25</b>	<b>Anonymous</b>	<b>28</b>
<b>26</b>	<b>ControlFlowGraph</b>	<b>29</b>
<b>27</b>	<b>ControlFlowNode</b>	<b>30</b>
<b>28</b>	<b>ControlFlowVisitor</b>	<b>32</b>
<b>29</b>	<b>Direction</b>	<b>33</b>

<b>30</b>	<b>CrystalRuntimeException</b>	<b>34</b>
<b>31</b>	<b>UserConsoleView</b>	<b>35</b>
<b>32</b>	<b>NullPrintWriter</b>	<b>36</b>
<b>33</b>	<b>Anonymous</b>	<b>37</b>
<b>34</b>	<b>ClearWarningHandler</b>	<b>38</b>
<b>35</b>	<b>Box</b>	<b>39</b>
<b>36</b>	<b>DisplayCrystalInfo</b>	<b>40</b>
<b>37</b>	<b>ShortFormatter</b>	<b>41</b>
<b>38</b>	<b>Utilities2</b>	<b>42</b>
<b>39</b>	<b>Option</b>	<b>43</b>
<b>40</b>	<b>Anonymous</b>	<b>44</b>
<b>41</b>	<b>AnalysisMenuPopulator</b>	<b>45</b>
<b>42</b>	<b>CrystalFileAction</b>	<b>46</b>
<b>43</b>	<b>Anonymous</b>	<b>47</b>
<b>44</b>	<b>Freezable</b>	<b>48</b>
<b>45</b>	<b>EnableAnalysisHandler</b>	<b>49</b>
<b>46</b>	<b>CrystalUIAction</b>	<b>50</b>
<b>47</b>	<b>Anonymous</b>	<b>51</b>
<b>48</b>	<b>MethodFindVisitor</b>	<b>52</b>
<b>49</b>	<b>Anonymous</b>	<b>53</b>
<b>50</b>	<b>BindingsCollectorVisitor</b>	<b>54</b>
<b>51</b>	<b>StudentRuntimeException</b>	<b>55</b>
<b>52</b>	<b>Abbreviation</b>	<b>56</b>
<b>53</b>	<b>Annotated Version of Sequential Java Program generated by Sip4j</b>	<b>57</b>

# 1 ConsListTest

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
ConsListTest	✓
testList	✓

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	ConsListTest	testList
ConsListTest	⌘	⌘
testList	⌘	⌘

## 2 ConsList

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
ConsList	✓
removeElement	✓
cons	✓
empty	✓
singleton	✓
list	✓
concat	✓
removeElementOnce	✓
map	✓
filter	✓
foldl	✓
listIterator	✓
subListSameTail	✓
get	✓
iterator	✓
subList	✓
contains	✓
toArray	✓
impossible	✓
add	✓
addAll	✓
clear	✓
remove	✓
removeOverload	✓
removeAll	✓
retainAll	✓
set	✓
main	✓

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	ConsList	removeElement	cons	empty	singleton	list	concat	removeElementOnce	map	filter	foldl	listIterator	subListSameTail	get	iterator	subList	contains	toArray	impossible	add	addAll	clear	remove
ConsList	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘



### 3 Triple

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
Triple	✓
fst	✓
snd	✓
thrd	✓
createTriple	✓

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	Triple	fst	snd	thrd	createTriple
Triple	✗				✗
fst					
snd					
thrd					
createTriple	✗				



## 4 Pair

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
Pair	✓
fst	✓
setComponent1	✓
snd	✓
setComponent2	✓
clone	✓
toString	✓
hashCode	✓
equals	✓
create	✓

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	Pair	fst	setComponent1	snd	setComponent2	clone	toString	hashCode	equals	create
Pair	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
fst	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
setComponent1	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
snd	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
setComponent2	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
clone	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
create	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 5 Empty

Table 14: Methods Requires Clause Satisfiability

Method	Satisfiability
Empty	✓
hd	✓
indexOf	✓
isEmpty	✓
lastIndexOf	✓
size	✓
tl	✓
indexOfHelper	✓
lastIndexOfHelper	✓
toString	✓
containsAll	✓

Table 15: State Transition Matrix

	alive
alive	↑

Table 16: Methods Concurrency Matrix

	Empty	hd	indexOf	isEmpty	lastIndexOf	size	tl	indexOfHelper	lastIndexOfHelper	toString	containsAll
Empty	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hd	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
indexOf	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isEmpty	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
lastIndexOf	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
size	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
tl	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
indexOfHelper	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
lastIndexOfHelper	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
containsAll	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 6 Utilities

Table 17: Methods Requires Clause Satisfiability

Method	Satisfiability
Utilities	✓
ASTNodeToString	✓
ModifierToString	✓
getMethodDeclaration	✓
methodDeclarationToString	✓
nyi	✓
nyiOverload	✓
main	✓

Table 18: State Transition Matrix

	alive
alive	↑

Table 19: Methods Concurrency Matrix

	Utilities	ASTNodeToString	ModifierToString	getMethodDeclaration	methodDeclarationToString	nyi	nyiOverload	main
Utilities	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
ASTNodeToString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
ModifierToString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getMethodDeclaration	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
methodDeclarationToString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
nyi	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
nyiOverload	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
main	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 7 CollectionMethods

Table 20: Methods Requires Clause Satisfiability

Method	Satisfiability
CollectionMethods	✓
map	✓
concat	✓
union	✓
addToMultiMap	✓
createSetWithoutElement	✓
mutableSet	✓

Table 21: State Transition Matrix

	alive
alive	↑

Table 22: Methods Concurrency Matrix

	CollectionMethods	map	concat	union	addToMultiMap	createSetWithoutElement	mutableSet
CollectionMethods	⌘	⌘	⌘	⌘	⌘	⌘	⌘
map	⌘	⌘	⌘	⌘	⌘		⌘
concat	⌘	⌘	⌘	⌘	⌘		⌘
union	⌘	⌘	⌘	⌘	⌘		⌘
addToMultiMap	⌘	⌘	⌘	⌘	⌘		⌘
createSetWithoutElement	⌘						
mutableSet	⌘	⌘	⌘	⌘	⌘		⌘

## 8 Nonempty

Table 23: Methods Requires Clause Satisfiability

Method	Satisfiability
Nonempty	✓
hd	✓
indexOfHelper	✓
indexOf	✓
isEmpty	✓
lastIndexOf	✓
lastIndexOfHelper	✓
size	✓
tl	✓
toString	✓
hashCode	✓
equals	✓
containsAll	✓

Table 24: State Transition Matrix

	alive
alive	↑

Table 25: Methods Concurrency Matrix

	Nonempty	hd	indexOfHelper	indexOf	isEmpty	lastIndexOf	lastIndexOfHelper	size	tl	toString	hashCode	equals	containsAll
Nonempty	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hd	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
indexOfHelper	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
indexOf	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isEmpty	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
lastIndexOf	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
lastIndexOfHelper	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
size	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
tl	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
containsAll	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 9 Anonymous

Table 26: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	✓
add	✓
hasNext	✓
hasPrevious	✓
next	✓
nextIndex	✓
previous	✓
previousIndex	✓
remove	✓
set	✓

Table 27: State Transition Matrix

	alive
alive	↑

Table 28: Methods Concurrency Matrix

	Anonymous	add	hasNext	hasPrevious	next	nextIndex	previous	previousIndex	remove	set
Anonymous	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
add	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
hasNext	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
hasPrevious	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
next	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
nextIndex	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
previous	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
previousIndex	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
remove	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
set	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈

## 10 PluralParseError

Table 29: Methods Requires Clause Satisfiability

Method	Satisfiability
PluralParseError	✓

Table 30: State Transition Matrix

	alive
alive	↑

## 11 ImpossibleConstraint

Table 31: Methods Requires Clause Satisfiability

Method	Satisfiability
ImpossibleConstraint	✓
dispatch	✓
toString	✓

Table 32: State Transition Matrix

	alive
alive	↑

Table 33: Methods Concurrency Matrix

	ImpossibleConstraint	dispatch	toString
ImpossibleConstraint	⌘	⌘	⌘
dispatch	⌘	⌘	
toString	⌘		



## 12 ZeroFraction

Table 34: Methods Requires Clause Satisfiability

Method	Satisfiability
ZeroFraction	✓
isZero	✓
dispatch	✓
toString	✓

Table 35: State Transition Matrix

	alive
alive	↑

Table 36: Methods Concurrency Matrix

	ZeroFraction	isZero	dispatch	toString
ZeroFraction	⧻	⧻	⧻	⧻
isZero	⧻			
dispatch	⧻		⧻	
toString	⧻			

## 13 AbstractFractionTermVisitor

Table 37: Methods Requires Clause Satisfiability

Method	Satisfiability
AbstractFractionTermVisitor	✓
named	✓
one	✓
var	✓
zero	✓
literal	✓
sum	✓

Table 38: State Transition Matrix

	alive
alive	↑

Table 39: Methods Concurrency Matrix

	AbstractFractionTermVisitor	named	one	var	zero	literal	sum
AbstractFractionTermVisitor	⌘	⌘	⌘	⌘	⌘	⌘	⌘
named	⌘						
one	⌘						
var	⌘						
zero	⌘						
literal	⌘					⌘	
sum	⌘						

## 14 VariableFraction

Table 40: Methods Requires Clause Satisfiability

Method	Satisfiability
VariableFraction	✓
getVarName	✓
isVariable	✓
dispatch	✓
compareToVar	✓
toString	✓
hashCode	✓
equals	✓

Table 41: State Transition Matrix

	alive
alive	↑

Table 42: Methods Concurrency Matrix

	VariableFraction	getVarName	isVariable	dispatch	compareToVar	toString	hashCode	equals
VariableFraction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getVarName	⌘			⌘				
isVariable	⌘							
dispatch	⌘	⌘		⌘	⌘	⌘	⌘	⌘
compareToVar	⌘			⌘				
toString	⌘			⌘				
hashCode	⌘			⌘				
equals	⌘			⌘				

## 15 FractionConstraints

Table 43: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionConstraints	✓
createMutable	✓
addConstraint	✓
testConstraint	✓
isConsistent	✓
isConsistentInternal	✓
isImpossible	✓
simplify	✓
simplifyInternal	✓
toString	✓
getConstraints	✓
mutableCopy	✓
mutableCopyOverload	✓
testConstraints	✓
addAll	✓
getVariables	✓
getConstants	✓
getUniversalParameters	✓
registerFractions	✓
atLeastAsPrecise	✓
freeze	✓
concat	✓
seemsConsistent	✓
hashCode	✓
equals	✓
newVariableFraction	✓
newNamedFraction	✓
isKnown	✓

Table 44: State Transition Matrix

	alive
alive	↑

Table 45: Methods Concurrency Matrix

	FractionConstraints	createMutable	addConstraint	testConstraint	isConsistent	isConsistentInternal	isImpossible	simplify	simplifyInternal	toString	getConstraints	mutableCopy	mutableCopyOverload	testConstraints	addAll	getVariables	getConstants	getUniversalParameters	registerFractions	atLeastAsPrecise	freeze	concat
--	---------------------	---------------	---------------	----------------	--------------	----------------------	--------------	----------	------------------	----------	----------------	-------------	---------------------	-----------------	--------	--------------	--------------	------------------------	-------------------	------------------	--------	--------

FractionConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
createMutable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
testConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isConsistent	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isConsistentInternal	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isImpossible	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
simplify	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
simplifyInternal	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
mutableCopy	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
mutableCopyOverload	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
testConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addAll	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getVariables	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getConstants	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getUniversalParameters	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
registerFractions	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
atLeastAsPrecise	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
freeze	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
concat	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
seemsConsistent	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
newVariableFraction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
newNamedFraction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isKnown	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 16 FractionConstraint

Table 46: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionConstraint	✓
impossible	✓
createEquality	✓
createLessThan	✓
createLessThanOrEqual	✓

Table 47: State Transition Matrix

	alive
alive	↑

Table 48: Methods Concurrency Matrix

	FractionConstraint	impossible	createEquality	createLessThan	createLessThanOrEqual
FractionConstraint	⌘	⌘	⌘	⌘	⌘
impossible	⌘				
createEquality	⌘				
createLessThan	⌘				
createLessThanOrEqual	⌘				

## 17 FractionAssignment

Table 49: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionAssignment	✓
resetChangedFlag	✓
makeEquivalentOverload	✓
union	✓
mutableSet	✓
isZero	✓
getLiteral	✓
isOne	✓
areEquivalent	✓
areEquivalentOverload	✓
makeZero	✓
makeZeroOverload	✓
makeOne	✓
makeNonZero	✓
isNonZero	✓
isChanged	✓
isConsistent	✓
sumsToConstant	✓
equivalentLiteralValues	✓
makeEquivalent	✓
getConstant	✓
getRepresentative	✓
toString	✓

Table 50: State Transition Matrix

	alive
alive	↑

Table 51: Methods Concurrency Matrix

	FractionAssignment	resetChangedFlag	makeEquivalentOverload	union	mutableSet	isZero	getLiteral	isOne	areEquivalent	areEquivalentOverload	makeZero	makeZeroOverload	makeOne	makeNonZero	isNonZero	isChanged	isConsistent	sumsToConstant	equivalentLiteralValues	makeEquivalent	getConstant
FractionAssignment	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
resetChangedFlag	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
makeEquivalentOverload	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
union	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

mutableSet	⌘																					
isZero	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getLiteral	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isOne	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
areEquivalent	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
areEquivalentOverload	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
makeZero	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
makeZeroOverload	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
makeOne	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
makeNonZero	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isNonZero	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isChanged	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘		⌘			⌘	⌘
isConsistent	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
sumsToConstant	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘		⌘			⌘	⌘
equivalentLiteralValues	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘		⌘			⌘	⌘
makeEquivalent	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getConstant	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getRepresentative	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘		⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘



## 18 FractionRelation

Table 52: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionRelation	✓
getRelop	✓
getTerms	✓
dispatch	✓
toString	✓
hashCode	✓
equals	✓
compareTo	✓

Table 53: State Transition Matrix

	alive
alive	↑

Table 54: Methods Concurrency Matrix

	FractionRelation	getRelop	getTerms	dispatch	toString	hashCode	equals	compareTo
FractionRelation	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getRelop	⌘			⌘		⌘	⌘	⌘
getTerms	⌘			⌘		⌘	⌘	⌘
dispatch	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘			⌘		⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
compareTo	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 19 Fraction

Table 55: Methods Requires Clause Satisfiability

Method	Satisfiability
Fraction	✓
zero	✓
one	✓
isZero	✓
isOne	✓
isVariable	✓
isNamed	✓
createNamed	✓
dispatch	✓
createExplicit	✓
isFixed	✓
isNeitherZeroNorOne	✓
isGuaranteedGreaterThanOrEqualTo	✓
isPossiblyGreaterOrEqual	✓

Table 56: State Transition Matrix

	alive
alive	↑

Table 57: Methods Concurrency Matrix

	Fraction	zero	one	isZero	isOne	isVariable	isNamed	createNamed	dispatch	createExplicit	isFixed	isNeitherZeroNorOne	isGuaranteedGreaterThanOrEqualTo	isPossiblyGreaterOrEqual
Fraction	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
zero	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗
one	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗
isZero	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
isOne	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
isVariable	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
isNamed	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
createNamed	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
dispatch	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✓	✓	✗	✗
createExplicit	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗
isFixed	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
isNeitherZeroNorOne	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

isGuaranteedGreaterThanZero	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isPossiblyGreaterOrEqual	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 20 FractionSum

Table 58: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionSum	✓
getSummands	✓
dispatch	✓
toString	✓
hashCode	✓
equals	✓

Table 59: State Transition Matrix

	alive
alive	↑

Table 60: Methods Concurrency Matrix

	FractionSum	getSummands	dispatch	toString	hashCode	equals
FractionSum	⌘	⌘	⌘	⌘	⌘	⌘
getSummands	⌘	⌘	⌘	⌘	⌘	⌘
dispatch	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘

21    **NamedFractionMapping**

Table 61: Methods Requires Clause Satisfiability

Method	Satisfiability
NamedFractionMapping	✓
map	✓

Table 62: State Transition Matrix

	alive
alive	↑

Table 63: Methods Concurrency Matrix

	NamedFractionMapping	map
NamedFractionMapping	⌋	⌋
map	⌋	⌋

## 22 NamedFraction

Table 64: Methods Requires Clause Satisfiability

Method	Satisfiability
NamedFraction	✓
equals	✓
getVarName	✓
isVariable	✓
isJoinVariable	✓
isNamed	✓
dispatch	✓
toString	✓
hashCode	✓

Table 65: State Transition Matrix

	alive
alive	↑

Table 66: Methods Concurrency Matrix

	NamedFraction	equals	getVarName	isVariable	isJoinVariable	isNamed	dispatch	toString	hashCode
NamedFraction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getVarName	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isVariable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isJoinVariable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isNamed	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
dispatch	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 23 VariableElimination

Table 67: Methods Requires Clause Satisfiability

Method	Satisfiability
VariableElimination	✓
eliminateVariables	✓
getTimeout	✓
normalizeConstraints	✓
eliminationOrder	✓
addVariableConstraints	✓
eliminateFraction	✓
addConstConstraints	✓
isConsistent	✓
isSatisfiable	✓
isPrimitiveConstraintSatisfiable	✓
setTimeout	✓
collectVariables	✓
normalizeTerm	✓

Table 68: State Transition Matrix

	alive
alive	↑

Table 69: Methods Concurrency Matrix

	VariableElimination	eliminateVariables	getTimeout	normalizeConstraints	eliminationOrder	addVariableConstraints	eliminateFraction	addConstConstraints	isConsistent	isSatisfiable	isPrimitiveConstraintSatisfiable	setTimeout	collectVariables	normalizeTerm
VariableElimination	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
eliminateVariables	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getTimeout	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
normalizeConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
eliminationOrder	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addVariableConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
eliminateFraction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addConstConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isConsistent	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isSatisfiable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isPrimitiveConstraintSatisfiable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

setTimeout	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
collectVariables	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
normalizeTerm	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘



## 24 NormalizedFractionConstraint

Table 70: Methods Requires Clause Satisfiability

Method	Satisfiability
NormalizedFractionConstraint	✓
createConstraintOverload	✓
isolateFraction	✓
getRelop	✓
createConstraint	✓
isTrueWithAssumptions	✓
isTriviallyTrue	✓
dominates	✓
equals	✓
isRangeConstraint	✓
isPrimitive	✓
getRight	✓
getLeft	✓
toString	✓
hashCode	✓

Table 71: State Transition Matrix

	alive
alive	↑

Table 72: Methods Concurrency Matrix

	NormalizedFractionConstraint	createConstraintOverload	isolateFraction	getRelop	createConstraint	isTrueWithAssumptions	isTriviallyTrue	dominates	equals	isRangeConstraint	isPrimitive	getRight	getLeft	toString	hashCode
NormalizedFractionConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
createConstraintOverload	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isolateFraction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getRelop	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
createConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isTrueWithAssumptions	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isTriviallyTrue	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
dominates	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isRangeConstraint	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isPrimitive	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

getRight	⌈	⌋	⌈	⌋	⌈	⌈	⌋	⌈	⌈	⌈	⌋	⌈	⌋	⌋	⌈	⌈
getLeft	⌈	⌋	⌈	⌋	⌈	⌈	⌋	⌈	⌈	⌈	⌋	⌈	⌋	⌋	⌈	⌈
toString	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈
hashCode	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈	⌈

## 25 Rational

Table 73: Methods Requires Clause Satisfiability

Method	Satisfiability
Rational	✓
one	✓
minusOne	✓
isZero	✓
abs	✓
isPositive	✓
negation	✓
gcd	✓
div	✓
plus	✓
zero	✓
minus	✓
isNegative	✓
isSmallerThan	✓
times	✓
inverse	✓
getP	✓
getQ	✓
isOne	✓
toString	✓
hashCode	✓
equals	✓

Table 74: State Transition Matrix

	alive
alive	↑

Table 75: Methods Concurrency Matrix

	Rational	one	minusOne	isZero	abs	isPositive	negation	gcd	div	plus	zero	minus	isNegative	isSmallerThan	times	inverse	getP	getQ	isOne	toString	hashCode	equals
Rational	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
one	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
minusOne	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isZero	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
abs	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isPositive	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
negation	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
gcd	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
div	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

plus	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
zero	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
minus	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isNegative	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isSmallerThan	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
times	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
inverse	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getP	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getQ	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isOne	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 26 GeneralizedSum

Table 76: Methods Requires Clause Satisfiability

Method	Satisfiability
GeneralizedSum	✓
getFractions	✓
getCoefficient	✓
equals	✓
getConstant	✓
isGround	✓
toString	✓
hashCode	✓

Table 77: State Transition Matrix

	alive
alive	↑

Table 78: Methods Concurrency Matrix

	GeneralizedSum	getFractions	getCoefficient	equals	getConstant	isGround	toString	hashCode
GeneralizedSum	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getFractions	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getCoefficient	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getConstant	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isGround	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
toString	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 27 VariableRelativity

Table 79: Methods Requires Clause Satisfiability

Method	Satisfiability
VariableRelativity	✓
addRight	✓
addLeft	✓
dumpRelations	✓
dumpRelation	✓

Table 80: State Transition Matrix

	alive
alive	↑

Table 81: Methods Concurrency Matrix

	VariableRelativity	addRight	addLeft	dumpRelations	dumpRelation
VariableRelativity	⌘	⌘	⌘	⌘	⌘
addRight	⌘	⌘	⌘	⌘	⌘
addLeft	⌘	⌘	⌘	⌘	⌘
dumpRelations	⌘	⌘	⌘	⌘	⌘
dumpRelation	⌘	⌘	⌘	⌘	⌘

28   **NormalizedFractionSum**

Table 82: Methods Requires Clause Satisfiability

Method	Satisfiability
NormalizedFractionSum	✓
zero	✓

Table 83: State Transition Matrix

	alive
alive	↑

Table 84: Methods Concurrency Matrix

	NormalizedFractionSum	zero
NormalizedFractionSum	⧻	⧻
zero	⧻	

## 29 SmtLibPrinter

Table 85: Methods Requires Clause Satisfiability

Method	Satisfiability
SmtLibPrinter	✓
toString	✓
printStatus	✓
getInverse	✓

Table 86: State Transition Matrix

	alive
alive	↑

Table 87: Methods Concurrency Matrix

	SmtLibPrinter	toString	printStatus	getInverse
SmtLibPrinter	⌘	⌘		
toString	⌘	⌘		
printStatus				
getInverse				



### 30 SmtLibBenchmarkPrinter

Table 88: Methods Requires Clause Satisfiability

Method	Satisfiability
SmtLibBenchmarkPrinter	✓
addLineComment	✓
addStatus	✓
addFormula	✓
addUnknown	✓
addAssumption	✓
appendConjunction	✓
addNegatedQuantifiedImplicationFormula	✓
appendExists	✓
getResult	✓

Table 89: State Transition Matrix

	alive
alive	↑

Table 90: Methods Concurrency Matrix

	SmtLibBenchmarkPrinter	addLineComment	addStatus	addFormula	addUnknown	addAssumption	appendConjunction	addNegatedQuantifiedImplicationFormula	appendExists	getResult
SmtLibBenchmarkPrinter	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addLineComment	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addStatus	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addFormula	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addUnknown	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addAssumption	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
appendConjunction	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
addNegatedQuantifiedImplicationFormula	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
appendExists	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
getResult	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 31 Anonymous

Table 91: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
named	✓
one	✓
var	✓
zero	✓

Table 92: State Transition Matrix

	alive
alive	↑

Table 93: Methods Concurrency Matrix

	Anonymous	named	one	var	zero
Anonymous	⧻	⧻	⧻	⧻	⧻
named	⧻				
one	⧻				
var	⧻				
zero	⧻				

## 32 OneFraction

Table 94: Methods Requires Clause Satisfiability

Method	Satisfiability
OneFraction	✓
isOne	✓
dispatch	✓
toString	✓

Table 95: State Transition Matrix

	alive
alive	↑

Table 96: Methods Concurrency Matrix

	OneFraction	isOne	dispatch	toString
OneFraction	⌘	⌘	⌘	⌘
isOne	⌘			
dispatch	⌘		⌘	
toString	⌘			

### 33 FractionTerm

Table 97: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionTerm	✓
createSum	✓
createSumOverload	✓
compareTo	✓

Table 98: State Transition Matrix

	alive
alive	↑

Table 99: Methods Concurrency Matrix

	FractionTerm	createSum	createSumOverload	compareTo
FractionTerm	⌘	⌘	⌘	⌘
createSum	⌘			
createSumOverload	⌘			
compareTo	⌘			

34 Anonymous

Table 100: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
literal	✓
sum	✓

Table 101: State Transition Matrix

	alive
alive	↑

Table 102: Methods Concurrency Matrix

	Anonymous	literal	sum
Anonymous	⌈	⌈	⌈
literal	⌈	⌈	⌈
sum	⌈	⌈	⌈

35 Anonymous

Table 103: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
named	×
one	×
var	×
zero	×

Table 104: State Transition Matrix

	alive
alive	↑

Table 105: Methods Concurrency Matrix

	Anonymous	named	one	var	zero
Anonymous	⧻	⧻	⧻	⧻	⧻
named	⧻	⧻	⧻	⧻	⧻
one	⧻	⧻	⧻	⧻	⧻
var	⧻	⧻	⧻	⧻	⧻
zero	⧻	⧻	⧻	⧻	⧻

36 Anonymous

Table 106: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
one	×
zero	×
named	×
var	×

Table 107: State Transition Matrix

	alive
alive	↑

Table 108: Methods Concurrency Matrix

	Anonymous	one	zero	named	var
Anonymous	⧻	⧻	⧻	⧻	⧻
one	⧻	⧻	⧻	⧻	⧻
zero	⧻	⧻	⧻	⧻	⧻
named	⧻	⧻	⧻	⧻	⧻
var	⧻	⧻	⧻	⧻	⧻

37 Relop

Table 109: Methods Requires Clause Satisfiability

Method	Satisfiability
Relop	✓
toString	✓

Table 110: State Transition Matrix

	alive
alive	↑

Table 111: Methods Concurrency Matrix

	Relop	toString
Relop	✗	
toString		



38 Anonymous

Table 112: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
named	×
one	×
var	×
zero	×

Table 113: State Transition Matrix

	alive
alive	↑

Table 114: Methods Concurrency Matrix

	Anonymous	named	one	var	zero
Anonymous	⧻	⧻	⧻	⧻	⧻
named	⧻	⧻	⧻	⧻	⧻
one	⧻	⧻	⧻	⧻	⧻
var	⧻	⧻	⧻	⧻	⧻
zero	⧻	⧻	⧻	⧻	⧻

### 39 RelationFractionPair

Table 115: Methods Requires Clause Satisfiability

Method	Satisfiability
RelationFractionPair	✓
createEqual	✓
createLeq	✓
createLess	✓
getRelop	✓
toString	✓
hashCode	✓
equals	✓

Table 116: State Transition Matrix

	alive
alive	↑

Table 117: Methods Concurrency Matrix

	RelationFractionPair	createEqual	createLeq	createLess	getRelop	toString	hashCode	equals
RelationFractionPair	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
createEqual	⌘						⌘	⌘
createLeq	⌘						⌘	⌘
createLess	⌘						⌘	⌘
getRelop	⌘						⌘	⌘
toString	⌘						⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘

## 40 Anonymous

Table 118: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
createRelation	✓
compare	✓
getRepresentative	✓
getRepresentatives	✓
impossible	✓
relation	✓
literal	×
sum	×

Table 119: State Transition Matrix

	alive
alive	↑

Table 120: Methods Concurrency Matrix

	Anonymous	createRelation	compare	getRepresentative	getRepresentatives	impossible	relation	literal	sum
Anonymous	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
createRelation	⊥								
compare	⊥								
getRepresentative	⊥								
getRepresentatives	⊥								
impossible	⊥								
relation	⊥								
literal	⊥								
sum	⊥								

## 41 SimpleFractionSum

Table 121: Methods Requires Clause Satisfiability

Method	Satisfiability
SimpleFractionSum	✓
createAdd	✓
createSub	✓
getSumop	✓
toString	✓
dispatch	✓
hashCode	✓
equals	✓

Table 122: State Transition Matrix

	alive
alive	↑

Table 123: Methods Concurrency Matrix

	SimpleFractionSum	createAdd	createSub	getSumop	toString	dispatch	hashCode	equals
SimpleFractionSum	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
createAdd	⌘		⌘				⌘	⌘
createSub	⌘	⌘	⌘	⌘	⌘		⌘	⌘
getSumop	⌘		⌘				⌘	⌘
toString	⌘		⌘				⌘	⌘
dispatch	⌘						⌘	
hashCode	⌘	⌘	⌘	⌘	⌘		⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘		⌘	⌘

42 Sumop

Table 124: Methods Requires Clause Satisfiability

Method	Satisfiability
Sumop	✓
toString	✓

Table 125: State Transition Matrix

	alive
alive	↑

Table 126: Methods Concurrency Matrix

	Sumop	toString
Sumop	✗	
toString		

43 SimpleVariableRelativity

Table 127: Methods Requires Clause Satisfiability

Method	Satisfiability
SimpleVariableRelativity	✓
addRight	✓
addLeft	✓
dumpRelations	✓

Table 128: State Transition Matrix

	alive
alive	↑

Table 129: Methods Concurrency Matrix

	Simple VariableRelativity			
		addRight	addLeft	dumpRelations
SimpleVariableRelativity	⌘	⌘	⌘	⌘
addRight	⌘	⌘	⌘	⌘
addLeft	⌘	⌘	⌘	⌘
dumpRelations	⌘	⌘	⌘	⌘

44 Anonymous

Table 130: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
named	×
one	×
sum	×
var	×
zero	×

Table 131: State Transition Matrix

	alive
alive	↑

Table 132: Methods Concurrency Matrix

	Anonymous	named	one	sum	var	zero
Anonymous	⌈	⌈	⌈	⌈	⌈	⌈
named	⌈	⌈	⌈	⌈	⌈	⌈
one	⌈	⌈	⌈	⌈	⌈	⌈
sum	⌈	⌈	⌈	⌈	⌈	⌈
var	⌈	⌈	⌈	⌈	⌈	⌈
zero	⌈	⌈	⌈	⌈	⌈	⌈

## 45 FractionElimination

Table 133: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionElimination	✓
eliminateVariableOverload	✓
containsVariable	✓
subtractVariable	✓
eliminateVariables	✓
normalizeConstraints	✓
collectVariables	✓
isConsistent	✓

Table 134: State Transition Matrix

	alive
alive	↑

Table 135: Methods Concurrency Matrix

	FractionElimination	eliminateVariableOverload	containsVariable	subtractVariable	eliminateVariables	normalizeConstraints	collectVariables	isConsistent
FractionElimination	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
eliminateVariableOverload	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
containsVariable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
subtractVariable	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
eliminateVariables	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
normalizeConstraints	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
collectVariables	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
isConsistent	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘



## 46 FractionPair

Table 136: Methods Requires Clause Satisfiability

Method	Satisfiability
FractionPair	✓
getComponent1	✓
getComponent2	✓
hashCode	✓
equals	✓

Table 137: State Transition Matrix

	alive
alive	↑

Table 138: Methods Concurrency Matrix

	FractionPair	getComponent1	getComponent2	hashCode	equals
FractionPair	⌘	⌘	⌘	⌘	⌘
getComponent1	⌘	⌘	⌘	⌘	⌘
getComponent2	⌘	⌘	⌘	⌘	⌘
hashCode	⌘	⌘	⌘	⌘	⌘
equals	⌘	⌘	⌘	⌘	⌘

## 47 Anonymous

Table 139: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
named	×
one	×
sum	×
var	×
zero	×
literal	×
impossible	×
relation	×

Table 140: State Transition Matrix

	alive
alive	↑

Table 141: Methods Concurrency Matrix

	Anonymous	named	one	sum	var	zero	literal	impossible	relation
Anonymous	⧻	⧻	⧻	⧻	⧻	⧻	⧻	⧻	⧻
named	⧻								
one	⧻								
sum	⧻								
var	⧻								
zero	⧻								
literal	⧻								
impossible	⧻								
relation	⧻								

48     **Anonymous**

Table 142: Methods Requires Clause Satisfiability

Method	Satisfiability
Anonymous	×
printStatus	✓
getInverse	✓

Table 143: State Transition Matrix

	alive
alive	↑

Table 144: Methods Concurrency Matrix

	Anonymous	printStatus	getInverse
Anonymous	⧻	⧻	⧻
printStatus	⧻		
getInverse	⧻		

## 49 SmtLibConstraintProcessor

Table 145: Methods Requires Clause Satisfiability

Method	Satisfiability
SmtLibConstraintProcessor	✓
impossible	✓
relation	✓
formatRelation	✓
literal	✓
sum	✓
named	✓
one	✓
var	✓
zero	✓

Table 146: State Transition Matrix

	alive
alive	↑

Table 147: Methods Concurrency Matrix

	SmtLibConstraintProcessor	impossible	relation	formatRelation	literal	sum	named	one	var	zero
SmtLibConstraintProcessor	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘
impossible	⌘									
relation	⌘		⌘	⌘	⌘	⌘	⌘		⌘	
formatRelation	⌘		⌘		⌘	⌘	⌘		⌘	
literal	⌘		⌘	⌘	⌘	⌘	⌘		⌘	
sum	⌘		⌘	⌘	⌘	⌘	⌘		⌘	
named	⌘		⌘	⌘	⌘	⌘	⌘		⌘	
one	⌘									
var	⌘		⌘	⌘	⌘	⌘	⌘		⌘	
zero	⌘									

## 50 Impossible

Table 148: Methods Requires Clause Satisfiability

Method	Satisfiability
Impossible	✓
getInstance	✓
equals	✓
hashCode	✓

Table 149: State Transition Matrix

	alive
alive	↑

Table 150: Methods Concurrency Matrix

	Impossible	getInstance	equals	hashCode
Impossible	✗	✗	✗	✗
getInstance	✗			
equals	✗			
hashCode	✗			

## 51 Abbreviation

Table 151: 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

## 52 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 ConsListTest {
6   @Perm(ensures="unique(this) in alive")
7   ConsListTest() { }
8
9   @Perm(requires="unique(this) in alive",
10  ensures="unique(this) in alive")
11   public void testList() {
12
13   }
14
15 }ENDOFCLASS
16
17 @ClassStates({@State(name = "alive")})
18
19 class ConsList {
20   @Perm(ensures="unique(this) in alive")
21   ConsList() { }
22
23   @Perm(requires="immutable(this) in alive",
24  ensures="immutable(this) in alive")
25   ConsList<T> removeElement(T t) {
26     return null;
27
28   }
29   @Perm(requires="pure(this) in alive",
30  ensures="pure(this) in alive")
31   ConsList<T> cons(T hd, ConsList<T> tl) {
32     return null;
33
34   }
35   @Perm(requires="full(this) in alive",
36  ensures="full(this) in alive")
37   ConsList<T> empty() {
38     return null;
39
40   }
41   @Perm(requires="pure(this) in alive",
42  ensures="pure(this) in alive")
43   ConsList<T> singleton(T hd) {
44     return null;
45
46   }
47   @Perm(requires="full(this) in alive",
48  ensures="full(this) in alive")
49   ConsList<T> list(T... ts) {
50     return null;
51
52   }
53   @Perm(requires="full(this) in alive",
54  ensures="full(this) in alive")
55   ConsList<T> concat(ConsList<T> front, ConsList<T> back) {
56     return null;
57
58   }
59   @Perm(requires="immutable(this) in alive",
60  ensures="immutable(this) in alive")
61   ConsList<T> removeElementOnce(T t) {
62     return null;
63
64   }
65   @Perm(requires="unique(this) in alive",
66  ensures="unique(this) in alive")
67   ConsList<O> map(Lambda<? super T,? extends O> lam) {
68     return null;
69
70   }
71   @Perm(requires="unique(this) in alive",
72  ensures="unique(this) in alive")
73   ConsList<T> filter(Lambda<? super T,? extends Boolean> lam) {
74     return null;
```

```

76 }
77 @Perm(requires="unique(this) in alive",
78 ensures="unique(this) in alive")
79 0 foldl(Lambda2<? super T,? super 0,? extends 0> lam, 0 o) {
80   return null;
81 }
82 }
83 @Perm(requires="unique(this) in alive",
84 ensures="unique(this) in alive")
85 ListIterator<T> listIterator(final int index) {
86   return null;
87 }
88 }
89 @Perm(requires="immutable(this) in alive",
90 ensures="immutable(this) in alive")
91 private ConsList<T> subListSameTail(int fromIndex) {
92   return null;
93 }
94 }
95
96 T get(int index) {
97   return null;
98 }
99 }
100
101 Iterator<T> iterator() {
102   return null;
103 }
104 }
105 @Perm(requires="share(this) in alive",
106 ensures="share(this) in alive")
107 ConsList<T> subList(int fromIndex, int toIndex) {
108   return null;
109 }
110 }
111 @Perm(requires="share(this) in alive",
112 ensures="share(this) in alive")
113 boolean contains(Object o) {
114   return 0;
115 }
116 }
117 @Perm(requires="unique(this) in alive",
118 ensures="unique(this) in alive")
119 Object[] toArray() {
120   return null;
121 }
122 }
123
124 R impossible() {
125   return null;
126 }
127 }
128
129 void add(int index, T element) {
130 }
131 }
132
133 boolean addAll(Collection<? extends T> c) {
134   return 0;
135 }
136 }
137
138 void clear() {
139 }
140 }
141
142 T remove(int index) {
143   return null;
144 }
145 }
146
147 T removeOverload(int index) {
148   return null;
149 }
150 }
151
152 boolean removeAll(Collection<?> c) {
153   return 0;
154 }
155 }

```



```

157     boolean retainAll(Collection<?> c) {
158         return 0;
159     }
160 }
161
162     T set(int index, T element) {
163         return null;
164     }
165 }
166 @Perm(requires="unique(this) in alive",
167 ensures="unique(this) in alive")
168     void main(String[] args) {
169     }
170 }
171
172 }ENDOFCLASS
173
174 @ClassStates({@State(name = "alive")})
175
176 class Triple {
177     @Perm(ensures="unique(this) in alive")
178     Triple() { }
179
180     @Perm(ensures="none(this) in alive")
181     public F fst() {
182         return null;
183     }
184 }
185 @Perm(ensures="none(this) in alive")
186     public S snd() {
187         return null;
188     }
189 }
190 @Perm(ensures="none(this) in alive")
191     public T thrd() {
192         return null;
193     }
194 }
195
196     Triple<F,S,T> createTriple(F f, S s, T t) {
197         return null;
198     }
199 }
200
201 }ENDOFCLASS
202
203 @ClassStates({@State(name = "alive")})
204
205 class Pair {
206     @Perm(ensures="unique(this) in alive")
207     Pair() { }
208
209     @Perm(requires="pure(this) in alive",
210 ensures="pure(this) in alive")
211     public A fst() {
212         return null;
213     }
214 }
215 @Perm(requires="share(this) in alive",
216 ensures="share(this) in alive")
217     public void setComponent1(A component1) {
218     }
219 }
220 @Perm(requires="pure(this) in alive",
221 ensures="pure(this) in alive")
222     public B snd() {
223         return null;
224     }
225 }
226 @Perm(requires="share(this) in alive",
227 ensures="share(this) in alive")
228     public void setComponent2(B component2) {
229     }
230 }
231 @Perm(requires="pure(this) in alive",
232 ensures="pure(this) in alive")
233     protected Object clone() {
234         return null;
235     }
236 }
237 @Perm(requires="pure(this) in alive",

```

```

238 ensures="pure(this) in alive")
239 public String toString() {
240     return null;
241 }
242 }
243 @Perm(requires="share(this) in alive",
244 ensures="share(this) in alive")
245 public int hashCode() {
246     return 0;
247 }
248 }
249 @Perm(requires="unique(this) in alive",
250 ensures="unique(this) in alive")
251 public boolean equals(Object obj) {
252     return 0;
253 }
254 }
255
256 Pair<A,B> create(A component1, B component2) {
257     return null;
258 }
259 }
260
261 }ENDOFCLASS
262
263 @ClassStates({@State(name = "alive")})
264
265 class Empty {
266     @Perm(ensures="unique(this) in alive")
267     Empty() { }
268
269
270     public T hd() {
271         return null;
272     }
273 }
274
275     public int indexOf(Object o) {
276         return 0;
277     }
278 }
279
280     public boolean isEmpty() {
281         return 0;
282     }
283 }
284
285     public int lastIndexOf(Object o) {
286         return 0;
287     }
288 }
289
290     public int size() {
291         return 0;
292     }
293 }
294 @Perm(requires="share(this) in alive",
295 ensures="share(this) in alive")
296 public ConsList<T> tl() {
297     return null;
298 }
299 }
300
301     protected int indexOfHelper(int cur_index, Object o) {
302         return 0;
303     }
304 }
305
306     protected int lastIndexOfHelper(boolean found, int cur_index, int cur_last, Object o) {
307         return 0;
308     }
309 }
310
311     public String toString() {
312         return null;
313     }
314 }
315 @Perm(requires="unique(this) in alive",
316 ensures="unique(this) in alive")
317 public boolean containsAll(Collection<?> c) {
318     return 0;

```

```

320 }
322 }ENDOFCLASS
324 @ClassStates({@State(name = "alive")})
326 class Utilities {
327   @Perm(ensures="unique(this) in alive")
328   Utilities() { }
329
331   String ASTNodeToString(ASTNode node) {
332     return null;
333   }
334
336   String ModifierToString(int modifier) {
337     return null;
338   }
339
340   @Perm(requires="share(this) in alive",
341   ensures="share(this) in alive")
342   MethodDeclaration getMethodDeclaration(ASTNode node) {
343     return null;
344   }
345
346   @Perm(requires="share(this) in alive",
347   ensures="share(this) in alive")
348   String methodDeclarationToString(MethodDeclaration md) {
349     return null;
350   }
351
353   T nyi() {
354     return null;
355   }
356
358   T nyiOverload(String err_msg) {
359     return null;
360   }
361
362   @Perm(requires="unique(this) in alive",
363   ensures="unique(this) in alive")
364   void main(String[] args) {
365
366   }
367
368 }ENDOFCLASS
370 @ClassStates({@State(name = "alive")})
372 class CollectionMethods {
373   @Perm(ensures="unique(this) in alive")
374   CollectionMethods() { }
375
376   @Perm(requires="unique(this) in alive",
377   ensures="unique(this) in alive")
378   List<O> map(List<? extends I> list, Mapping<I,O> fun) {
379     return null;
380   }
381
382   @Perm(requires="unique(this) in alive",
383   ensures="unique(this) in alive")
384   List<T> concat(List<? extends T> l1, List<? extends T> l2) {
385     return null;
386   }
387
388   @Perm(requires="unique(this) in alive",
389   ensures="unique(this) in alive")
390   Map<K,V> union(Map<? extends K,? extends V> m1, Map<? extends K,? extends V> m2) {
391     return null;
392   }
393
394   @Perm(requires="unique(this) in alive",
395   ensures="unique(this) in alive")
396   void addToMultiMap(K key, V val, Map<K,List<V>> map) {
397
398   }

```

```

400     Set<T> createSetWithoutElement(Set<T> s, T element) {
401     return null;
402
403     }
404     @Perm(requires="unique(this) in alive",
405     ensures="unique(this) in alive")
406     Set<T> mutableSet(T... elements) {
407     return null;
408
409     }
410
411 }ENDOFCLASS
412
413 @ClassStates({@State(name = "alive")})
414
415 class Nonempty {
416 @Perm(ensures="unique(this) in alive")
417 Nonempty() {    }
418
419 @Perm(requires="pure(this) in alive",
420 ensures="pure(this) in alive")
421 public T hd() {
422     return null;
423
424 }
425
426 protected int indexOfHelper(int cur_index, Object o) {
427     return 0;
428
429 }
430
431 public int indexOf(Object o) {
432     return 0;
433
434 }
435
436 public boolean isEmpty() {
437     return 0;
438
439 }
440
441 public int lastIndexOf(Object o) {
442     return 0;
443
444 }
445
446 protected int lastIndexOfHelper(boolean found, int cur_index, int cur_last, Object o) {
447     return 0;
448
449 }
450 @Perm(requires="immutable(this) in alive",
451 ensures="immutable(this) in alive")
452 public int size() {
453     return 0;
454
455 }
456 @Perm(requires="pure(this) in alive",
457 ensures="pure(this) in alive")
458 public ConsList<T> tl() {
459     return null;
460
461 }
462
463 public String toString() {
464     return null;
465
466 }
467 @Perm(requires="share(this) in alive",
468 ensures="share(this) in alive")
469 public int hashCode() {
470     return 0;
471
472 }
473 @Perm(requires="unique(this) in alive",
474 ensures="unique(this) in alive")
475 public boolean equals(Object obj) {
476     return 0;
477
478 }
479
480 public boolean containsAll(Collection<?> c) {

```

```

481     return 0;
482 }
483 }ENDOFCLASS
484
485 @ClassStates({@State(name = "alive")})
486
487 class Anonymous {
488     @Perm(ensures="unique(this) in alive")
489     Anonymous() { }
490
491     public void add(T e) {
492     }
493     @Perm(requires="full(this) in alive",
494           ensures="full(this) in alive")
495     public boolean hasNext() {
496         return 0;
497     }
498     @Perm(requires="pure(this) in alive",
499           ensures="pure(this) in alive")
500     public boolean hasPrevious() {
501         return 0;
502     }
503     @Perm(requires="unique(this) in alive",
504           ensures="unique(this) in alive")
505     public T next() {
506         return null;
507     }
508     @Perm(requires="pure(this) in alive",
509           ensures="pure(this) in alive")
510     public int nextIndex() {
511         return 0;
512     }
513     @Perm(requires="share(this) in alive",
514           ensures="share(this) in alive")
515     public T previous() {
516         return null;
517     }
518     @Perm(requires="pure(this) in alive",
519           ensures="pure(this) in alive")
520     public int previousIndex() {
521         return 0;
522     }
523     public void remove() {
524     }
525     public void set(T e) {
526     }
527 }ENDOFCLASS
528
529 @ClassStates({@State(name = "alive")})
530
531 class PluralParseError {
532     @Perm(ensures="unique(this) in alive")
533     PluralParseError() { }
534 }ENDOFCLASS
535
536 @ClassStates({@State(name = "alive")})
537
538 class ImpossibleConstraint {
539     @Perm(ensures="unique(this) in alive")
540     ImpossibleConstraint() { }
541     @Perm(requires="unique(this) in alive",
542           ensures="unique(this) in alive")
543     public T dispatch(FractionConstraintVisitor<T> visitor) {

```

```

562     return null;
563 }
564
565 public String toString() {
566     return null;
567 }
568
569 }
570
571 }ENDOFCLASS
572
573 @ClassStates({@State(name = "alive")})
574
575 class ZeroFraction {
576     @Perm(ensures="unique(this) in alive")
577     ZeroFraction() { }
578
579
580     public boolean isZero() {
581         return 0;
582     }
583
584     @Perm(requires="unique(this) in alive",
585           ensures="unique(this) in alive")
586     public T dispatch(FractionVisitor<T> visitor) {
587         return null;
588     }
589 }
590
591 public String toString() {
592     return null;
593 }
594
595 }ENDOFCLASS
596
597 @ClassStates({@State(name = "alive")})
598
599 class AbstractFractionTermVisitor {
600     @Perm(ensures="unique(this) in alive")
601     AbstractFractionTermVisitor() { }
602
603
604     public T named(NamedFraction fract) {
605         return null;
606     }
607
608 }
609
610     public T one(OneFraction fract) {
611         return null;
612     }
613 }
614
615     public T var(VariableFraction fract) {
616         return null;
617     }
618 }
619
620     public T zero(ZeroFraction fract) {
621         return null;
622     }
623 }
624     @Perm(requires="unique(this) in alive",
625           ensures="unique(this) in alive")
626     public T literal(Fraction fract) {
627         return null;
628     }
629 }
630
631     public T sum(FractionSum fract) {
632         return null;
633     }
634 }
635
636 }ENDOFCLASS
637
638 @ClassStates({@State(name = "alive")})
639
640 class VariableFraction {
641     @Perm(ensures="unique(this) in alive")
642     VariableFraction() { }

```

```

644 @Perm(requires="immutable(this) in alive",
645 ensures="immutable(this) in alive")
646 public String getVarName() {
647     return null;
648 }
649
650
651 public boolean isVariable() {
652     return 0;
653 }
654
655 @Perm(requires="unique(this) in alive",
656 ensures="unique(this) in alive")
657 public T dispatch(FractionVisitor<T> visitor) {
658     return null;
659 }
660
661 @Perm(requires="immutable(this) in alive",
662 ensures="immutable(this) in alive")
663 public int compareToVar(VariableFraction other) {
664     return 0;
665 }
666
667 @Perm(requires="immutable(this) in alive",
668 ensures="immutable(this) in alive")
669 public String toString() {
670     return null;
671 }
672
673 @Perm(requires="immutable(this) in alive",
674 ensures="immutable(this) in alive")
675 public int hashCode() {
676     return 0;
677 }
678
679 @Perm(requires="immutable(this) in alive",
680 ensures="immutable(this) in alive")
681 public boolean equals(Object obj) {
682     return 0;
683 }
684
685 }ENDOFCLASS
686
687 @ClassStates({@State(name = "alive")})
688
689 class FractionConstraints {
690     @Perm(ensures="unique(this) in alive")
691     FractionConstraints() { }
692
693     FractionConstraints createMutable() {
694         return null;
695     }
696
697     @Perm(requires="unique(this) in alive",
698     ensures="unique(this) in alive")
699     public FractionConstraints addConstraint(FractionConstraint newConstraint) {
700         return null;
701     }
702
703     @Perm(requires="unique(this) in alive",
704     ensures="unique(this) in alive")
705     public boolean testConstraint(FractionConstraint test) {
706         return 0;
707     }
708
709     @Perm(requires="unique(this) in alive",
710     ensures="unique(this) in alive")
711     public boolean isConsistent() {
712         return 0;
713     }
714
715     @Perm(requires="unique(this) in alive",
716     ensures="unique(this) in alive")
717     private boolean isConsistentInternal() {
718         return 0;
719     }
720
721     }
722
723 @Perm(requires="pure(this) in alive",

```

```

724 ensures="pure(this) in alive")
725 public boolean isImpossible() {
726     return 0;
727 }
728
729 @Perm(requires="unique(this) in alive",
730 ensures="unique(this) in alive")
731 public FractionAssignment simplify() {
732     return null;
733 }
734
735 @Perm(requires="unique(this) in alive",
736 ensures="unique(this) in alive")
737 private FractionAssignment simplifyInternal() {
738     return null;
739 }
740
741 @Perm(requires="unique(this) in alive",
742 ensures="unique(this) in alive")
743 public String toString() {
744     return null;
745 }
746
747 @Perm(requires="pure(this) in alive",
748 ensures="pure(this) in alive")
749 public Collection<FractionConstraint> getConstraints() {
750     return null;
751 }
752
753 @Perm(requires="share(this) in alive",
754 ensures="share(this) in alive")
755 public FractionConstraints mutableCopy() {
756     return null;
757 }
758
759 @Perm(requires="share(this) in alive",
760 ensures="share(this) in alive")
761 public FractionConstraints mutableCopyOverload(Set<NamedFraction> universals) {
762     return null;
763 }
764
765 @Perm(requires="unique(this) in alive",
766 ensures="unique(this) in alive")
767 public boolean testConstraints(FractionConstraint... test) {
768     return 0;
769 }
770
771 @Perm(requires="unique(this) in alive",
772 ensures="unique(this) in alive")
773 public void addAll(FractionConstraints moreConstraints) {
774 }
775
776 @Perm(requires="pure(this) in alive",
777 ensures="pure(this) in alive")
778 public Set<VariableFraction> getVariables() {
779     return null;
780 }
781
782 @Perm(requires="pure(this) in alive",
783 ensures="pure(this) in alive")
784 public Set<NamedFraction> getConstants() {
785     return null;
786 }
787
788 @Perm(requires="immutable(this) in alive",
789 ensures="immutable(this) in alive")
790 public Set<NamedFraction> getUniversalParameters() {
791     return null;
792 }
793
794 @Perm(requires="unique(this) in alive",
795 ensures="unique(this) in alive")
796 public void registerFractions(Set<Fraction> fractions) {
797 }
798
799 @Perm(requires="pure(this) in alive",
800 ensures="pure(this) in alive")
801 public boolean atLeastAsPrecise(FractionConstraints other) {
802     return 0;
803 }
804 }

```



```

805 @Perm(requires="share(this) in alive",
806 ensures="share(this) in alive")
807 public FractionConstraints freeze() {
808     return null;
809 }
810 }
811 @Perm(requires="share(this) in alive",
812 ensures="share(this) in alive")
813 public FractionConstraints concat(FractionConstraints other) {
814     return null;
815 }
816 }
817 @Perm(requires="unique(this) in alive",
818 ensures="unique(this) in alive")
819 public boolean seemsConsistent() {
820     return 0;
821 }
822 }
823 @Perm(requires="share(this) in alive",
824 ensures="share(this) in alive")
825 public int hashCode() {
826     return 0;
827 }
828 }
829 @Perm(requires="share(this) in alive",
830 ensures="share(this) in alive")
831 public boolean equals(Object obj) {
832     return 0;
833 }
834 }
835 @Perm(requires="share(this) in alive",
836 ensures="share(this) in alive")
837 public VariableFraction newVariableFraction() {
838     return null;
839 }
840 }
841 @Perm(requires="share(this) in alive",
842 ensures="share(this) in alive")
843 public NamedFraction newNamedFraction() {
844     return null;
845 }
846 }
847 @Perm(requires="unique(this) in alive",
848 ensures="unique(this) in alive")
849 public boolean isKnown(Fraction f) {
850     return 0;
851 }
852 }
853 }
854 }ENDOFCLASS
855
856 @ClassStates({@State(name = "alive")})
857
858 class FractionConstraint {
859     @Perm(ensures="unique(this) in alive")
860     FractionConstraint() { }
861
862     @Perm(requires="pure(this) in alive",
863     ensures="pure(this) in alive")
864     FractionConstraint impossible() {
865         return null;
866     }
867 }
868 @Perm(requires="pure(this) in alive",
869 ensures="pure(this) in alive")
870 FractionConstraint createEquality(FractionTerm... terms) {
871     return null;
872 }
873 }
874 @Perm(requires="pure(this) in alive",
875 ensures="pure(this) in alive")
876 FractionConstraint createLessThan(FractionTerm... terms) {
877     return null;
878 }
879 }
880 @Perm(requires="immutable(this) in alive",
881 ensures="immutable(this) in alive")
882 FractionConstraint createLessThanOrEqual(FractionTerm... terms) {
883     return null;
884 }
885 }

```

```

887 }ENDOFCLASS
889 @ClassStates({@State(name = "alive")})
891 class FractionAssignment {
892   @Perm(ensures="unique(this) in alive")
893   FractionAssignment() { }
895   @Perm(requires="share(this) in alive",
896   ensures="share(this) in alive")
897   void resetChangedFlag() {
899   }
900   @Perm(requires="share(this) in alive",
901   ensures="share(this) in alive")
902   void makeEquivalentOverload(Iterable<FractionTerm> terms) {
904   }
905   @Perm(requires="share(this) in alive",
906   ensures="share(this) in alive")
907   private void union(FractionTerm t1, FractionTerm t2) {
909   }
911   SortedSet<FractionTerm> mutableSet(FractionTerm... initialElements) {
912   return null;
914   }
915   @Perm(requires="share(this) in alive",
916   ensures="share(this) in alive")
917   public boolean isZero(FractionTerm f) {
918   return 0;
920   }
921   @Perm(requires="share(this) in alive",
922   ensures="share(this) in alive")
923   public Fraction getLiteral(FractionTerm f) {
924   return null;
926   }
927   @Perm(requires="share(this) in alive",
928   ensures="share(this) in alive")
929   public boolean isOne(FractionTerm f) {
930   return 0;
932   }
933   @Perm(requires="share(this) in alive",
934   ensures="share(this) in alive")
935   public boolean areEquivalent(FractionTerm t1, FractionTerm t2) {
936   return 0;
938   }
939   @Perm(requires="share(this) in alive",
940   ensures="share(this) in alive")
941   public boolean areEquivalentOverload(FractionTerm t1, FractionTerm t2, NamedFractionMapping mapping) {
942   return 0;
944   }
945   @Perm(requires="share(this) in alive",
946   ensures="share(this) in alive")
947   void makeZero(FractionTerm f) {
949   }
950   @Perm(requires="share(this) in alive",
951   ensures="share(this) in alive")
952   void makeZeroOverload(List<FractionTerm> terms) {
954   }
955   @Perm(requires="share(this) in alive",
956   ensures="share(this) in alive")
957   void makeOne(FractionTerm f) {
959   }
960   @Perm(requires="share(this) in alive",
961   ensures="share(this) in alive")
962   void makeNonZero(FractionTerm t) {
964   }
965   @Perm(requires="unique(this) in alive",
966   ensures="unique(this) in alive")

```

```

967 public boolean isNonZero(FractionTerm f) {
968     return 0;
969 }
970
971 @Perm(requires="pure(this) in alive",
972 ensures="pure(this) in alive")
973 boolean isChanged() {
974     return 0;
975 }
976
977 @Perm(requires="share(this) in alive",
978 ensures="share(this) in alive")
979 public boolean isConsistent() {
980     return 0;
981 }
982
983 @Perm(requires="pure(this) in alive",
984 ensures="pure(this) in alive")
985 boolean sumsToConstant(FractionSum sum) {
986     return 0;
987 }
988
989 @Perm(requires="pure(this) in alive",
990 ensures="pure(this) in alive")
991 boolean equivalentLiteralValues(FractionTerm t1, FractionTerm t2) {
992     return 0;
993 }
994
995 @Perm(requires="share(this) in alive",
996 ensures="share(this) in alive")
997 void makeEquivalent(FractionTerm... terms) {
998 }
999
1000 @Perm(requires="share(this) in alive",
1001 ensures="share(this) in alive")
1002 public Fraction getConstant(FractionTerm f) {
1003     return null;
1004 }
1005
1006 @Perm(requires="share(this) in alive",
1007 ensures="share(this) in alive")
1008 public Fraction getRepresentative(Fraction f) {
1009     return null;
1010 }
1011
1012 @Perm(requires="share(this) in alive",
1013 ensures="share(this) in alive")
1014 public String toString() {
1015     return null;
1016 }
1017 }
1018 }ENDOFCLASS
1019
1020 @ClassStates({@State(name = "alive")})
1021
1022 class FractionRelation {
1023     @Perm(ensures="unique(this) in alive")
1024     FractionRelation() { }
1025
1026     @Perm(requires="pure(this) in alive",
1027     ensures="pure(this) in alive")
1028     public Relop getRelop() {
1029         return null;
1030     }
1031 }
1032
1033 @Perm(requires="pure(this) in alive",
1034 ensures="pure(this) in alive")
1035 public List<FractionTerm> getTerms() {
1036     return null;
1037 }
1038
1039 @Perm(requires="unique(this) in alive",
1040 ensures="unique(this) in alive")
1041 public T dispatch(FractionConstraintVisitor<T> visitor) {
1042     return null;
1043 }
1044
1045 @Perm(requires="pure(this) in alive",
1046 ensures="pure(this) in alive")
1047 public String toString() {

```

```

1048     return null;
1050 }
1051 @Perm(requires="share(this) in alive",
1052 ensures="share(this) in alive")
1053 public int hashCode() {
1054     return 0;
1056 }
1057 @Perm(requires="share(this) in alive",
1058 ensures="share(this) in alive")
1059 public boolean equals(Object obj) {
1060     return 0;
1062 }
1063 @Perm(requires="unique(this) in alive",
1064 ensures="unique(this) in alive")
1065 public int compareTo(FractionRelation o) {
1066     return 0;
1068 }
1070 }ENDOFCLASS
1072 @ClassStates({@State(name = "alive")})
1074 class Fraction {
1075     @Perm(ensures="unique(this) in alive")
1076     Fraction() { }
1078     @Perm(requires="immutable(this) in alive",
1079 ensures="immutable(this) in alive")
1080     Fraction zero() {
1081         return null;
1083     }
1084     @Perm(requires="pure(this) in alive",
1085 ensures="pure(this) in alive")
1086     Fraction one() {
1087         return null;
1089     }
1091     public boolean isZero() {
1092         return 0;
1094     }
1096     public boolean isOne() {
1097         return 0;
1099     }
1101     public boolean isVariable() {
1102         return 0;
1104     }
1106     public boolean isNamed() {
1107         return 0;
1109     }
1111     Fraction createNamed(String name) {
1112         return null;
1114     }
1115     @Perm(requires="unique(this) in alive",
1116 ensures="unique(this) in alive")
1117     T dispatch(FractionTermVisitor<T> visitor) {
1118         return null;
1120     }
1121     @Perm(requires="pure(this) in alive",
1122 ensures="pure(this) in alive")
1123     Fraction createExplicit(int p, int q) {
1124         return null;
1126     }
1128     boolean isFixed() {

```

```

1129     return 0;
1131 }
1133     boolean isNeitherZeroNorOne() {
1134     return 0;
1136 }
1137     @Perm(requires="unique(this) in alive",
1138     ensures="unique(this) in alive")
1139     boolean isGuaranteedGreaterThanZero() {
1140     return 0;
1142 }
1143     @Perm(requires="unique(this) in alive",
1144     ensures="unique(this) in alive")
1145     boolean isPossiblyGreaterOrEqual(final Fraction other) {
1146     return 0;
1148 }
1150 }ENDOFCLASS
1152 @ClassStates({@State(name = "alive")})
1154 class FractionSum {
1155     @Perm(ensures="unique(this) in alive")
1156     FractionSum() { }
1158     @Perm(requires="pure(this) in alive",
1159     ensures="pure(this) in alive")
1160     public List<Fraction> getSummands() {
1161     return null;
1163 }
1164     @Perm(requires="unique(this) in alive",
1165     ensures="unique(this) in alive")
1166     public T dispatch(FractionTermVisitor<T> visitor) {
1167     return null;
1169 }
1170     @Perm(requires="pure(this) in alive",
1171     ensures="pure(this) in alive")
1172     public String toString() {
1173     return null;
1175 }
1176     @Perm(requires="share(this) in alive",
1177     ensures="share(this) in alive")
1178     public int hashCode() {
1179     return 0;
1181 }
1182     @Perm(requires="share(this) in alive",
1183     ensures="share(this) in alive")
1184     public boolean equals(Object obj) {
1185     return 0;
1187 }
1189 }ENDOFCLASS
1191 @ClassStates({@State(name = "alive")})
1193 class NamedFractionMapping {
1194     @Perm(ensures="unique(this) in alive")
1195     NamedFractionMapping() { }
1197     @Perm(requires="share(this) in alive",
1198     ensures="share(this) in alive")
1199     public boolean map(NamedFraction f1, NamedFraction f2) {
1200     return 0;
1202 }
1204 }ENDOFCLASS
1206 @ClassStates({@State(name = "alive")})
1208 class NamedFraction {
1209     @Perm(ensures="unique(this) in alive")

```

```

1210 NamedFraction() { }

1212 @Perm(requires="share(this) in alive",
1213 ensures="share(this) in alive")
1214 public boolean equals(Object obj) {
1215     return 0;
1217 }
1218 @Perm(requires="pure(this) in alive",
1219 ensures="pure(this) in alive")
1220 public String getVarName() {
1221     return null;
1223 }
1224 @Perm(requires="pure(this) in alive",
1225 ensures="pure(this) in alive")
1226     boolean isVariable(ASTNode node) {
1227     return 0;
1229 }
1230 @Perm(requires="pure(this) in alive",
1231 ensures="pure(this) in alive")
1232     boolean isJoinVariable() {
1233     return 0;
1235 }

1237 public boolean isNamed() {
1238     return 0;
1240 }
1241 @Perm(requires="unique(this) in alive",
1242 ensures="unique(this) in alive")
1243     public T dispatch(FractionVisitor<T> visitor) {
1244     return null;
1246 }
1247 @Perm(requires="full(this) in alive",
1248 ensures="full(this) in alive")
1249     public String toString() {
1250     return null;
1252 }
1253 @Perm(requires="share(this) in alive",
1254 ensures="share(this) in alive")
1255     public int hashCode() {
1256     return 0;
1258 }

1260 }ENDOFCLASS

1262 @ClassStates({@State(name = "alive")})

1264 class VariableElimination {
1265     @Perm(ensures="unique(this) in alive")
1266     VariableElimination() { }

1268     @Perm(requires="unique(this) in alive",
1269     ensures="unique(this) in alive")
1270     public Set<NormalizedFractionConstraint> eliminateVariables(Collection<FractionConstraint> constraints,
1271     FractionAssignment a) {
1272     return null;
1273 }
1274     @Perm(requires="pure(this) in alive",
1275     ensures="pure(this) in alive")
1276     public long getTimeout() {
1277     return 0;
1279 }
1280     @Perm(requires="unique(this) in alive",
1281     ensures="unique(this) in alive")
1282     private Set<NormalizedFractionConstraint> normalizeConstraints(Collection<FractionConstraint>
1283     constraints) {
1284     return null;
1285 }
1286     @Perm(requires="unique(this) in alive",
1287     ensures="unique(this) in alive")
1288     private List<VariableFraction> eliminationOrder(Set<VariableFraction> vars) {

```

```

1289     return null;
1290 }
1291 @Perm(requires="pure(this) in alive",
1292 ensures="pure(this) in alive")
1293 private Set<NormalizedFractionConstraint> addVariableConstraints(Set<NormalizedFractionConstraint> rels
1294     , Iterable<VariableFraction> vars) {
1295     return null;
1296 }
1297 @Perm(requires="unique(this) in alive",
1298 ensures="unique(this) in alive")
1299 private Set<NormalizedFractionConstraint> eliminateFraction(Set<NormalizedFractionConstraint> rels,
1300     Fraction x, boolean populateGroundRels) {
1301     return null;
1302 }
1303 @Perm(requires="share(this) in alive",
1304 ensures="share(this) in alive")
1305 private Set<NormalizedFractionConstraint> addConstConstraints(Set<NormalizedFractionConstraint> rels,
1306     Iterable<NamedFraction> vars) {
1307     return null;
1308 }
1309 @Perm(requires="unique(this) in alive",
1310 ensures="unique(this) in alive")
1311 public boolean isConsistent() {
1312     return 0;
1313 }
1314 @Perm(requires="unique(this) in alive",
1315 ensures="unique(this) in alive")
1316 public boolean isSatisfiable(Set<NormalizedFractionConstraint> rels, Set<? extends Fraction> vars) {
1317     return 0;
1318 }
1319 @Perm(requires="share(this) in alive",
1320 ensures="share(this) in alive")
1321 private boolean isPrimitiveConstraintSatisfiable(NormalizedFractionConstraint c) {
1322     return 0;
1323 }
1324 @Perm(requires="full(this) in alive",
1325 ensures="full(this) in alive")
1326 public void setTimeout(long timeout) {
1327 }
1328 @Perm(requires="share(this) in alive",
1329 ensures="share(this) in alive")
1330 private SortedSet<T> collectVariables(Set<NormalizedFractionConstraint> rels, Class<T> variableType) {
1331     return null;
1332 }
1333 private NormalizedFractionSum normalizeTerm(final FractionTerm term) {
1334     return null;
1335 }
1336 }
1337 }ENDOFCLASS
1338 @ClassStates({@State(name = "alive")})
1339 class NormalizedFractionConstraint {
1340     @Perm(ensures="unique(this) in alive")
1341     NormalizedFractionConstraint() { }
1342     @Perm(requires="immutable(this) in alive",
1343     ensures="immutable(this) in alive")
1344     NormalizedFractionConstraint createConstraintOverload(Fraction left, Relop relop, Fraction right) {
1345         return null;
1346     }
1347     @Perm(requires="share(this) in alive",
1348     ensures="share(this) in alive")
1349     public Pair<NormalizedFractionSum, Boolean> isolateFraction(Fraction x) {
1350         return null;
1351     }
1352     @Perm(requires="pure(this) in alive",
1353     ensures="pure(this) in alive")

```

```

1367 public Relop getRelop() {
1368     return null;
1369 }
1370 @Perm(requires="share(this) in alive",
1371 ensures="share(this) in alive")
1372 NormalizedFractionConstraint createConstraint(GeneralizedSum left, Relop relop, GeneralizedSum right)
1373 {
1374     return null;
1375 }
1376 @Perm(requires="share(this) in alive",
1377 ensures="share(this) in alive")
1378 public boolean isTrueWithAssumptions(Map<NamedFraction, NamedFraction> upperBounds) {
1379     return 0;
1380 }
1381 @Perm(requires="pure(this) in alive",
1382 ensures="pure(this) in alive")
1383 public boolean isTriviallyTrue() {
1384     return 0;
1385 }
1386 @Perm(requires="share(this) in alive",
1387 ensures="share(this) in alive")
1388 public boolean dominates(NormalizedFractionConstraint other) {
1389     return 0;
1390 }
1391 @Perm(requires="share(this) in alive",
1392 ensures="share(this) in alive")
1393 public boolean equals(Object obj) {
1394     return 0;
1395 }
1396 @Perm(requires="pure(this) in alive",
1397 ensures="pure(this) in alive")
1398 private boolean isRangeConstraint() {
1399     return 0;
1400 }
1401 @Perm(requires="share(this) in alive",
1402 ensures="share(this) in alive")
1403 public boolean isPrimitive() {
1404     return 0;
1405 }
1406 @Perm(requires="immutable(this) in alive",
1407 ensures="immutable(this) in alive")
1408 public GeneralizedSum getRight() {
1409     return null;
1410 }
1411 @Perm(requires="immutable(this) in alive",
1412 ensures="immutable(this) in alive")
1413 public GeneralizedSum getLeft() {
1414     return null;
1415 }
1416 @Perm(requires="share(this) in alive",
1417 ensures="share(this) in alive")
1418 public String toString() {
1419     return null;
1420 }
1421 @Perm(requires="share(this) in alive",
1422 ensures="share(this) in alive")
1423 public int hashCode() {
1424     return 0;
1425 }
1426 }
1427 }ENDOFCLASS
1428
1429 @ClassStates({@State(name = "alive")})
1430
1431 class Rational {
1432     @Perm(ensures="unique(this) in alive")
1433     Rational() { }
1434
1435     @Perm(requires="immutable(this) in alive",

```



```

1447 ensures="immutable(this) in alive")
1448     Rational one() {
1449         return null;
1450     }
1451 }
1452 @Perm(requires="immutable(this) in alive",
1453 ensures="immutable(this) in alive")
1454     Rational minusOne() {
1455         return null;
1456     }
1457 }
1458 @Perm(requires="immutable(this) in alive",
1459 ensures="immutable(this) in alive")
1460     public boolean isZero() {
1461         return 0;
1462     }
1463 }
1464 @Perm(requires="share(this) in alive",
1465 ensures="share(this) in alive")
1466     public Rational abs() {
1467         return null;
1468     }
1469 }
1470 @Perm(requires="pure(this) in alive",
1471 ensures="pure(this) in alive")
1472     public boolean isPositive() {
1473         return 0;
1474     }
1475 }
1476 @Perm(requires="share(this) in alive",
1477 ensures="share(this) in alive")
1478     public Rational negation() {
1479         return null;
1480     }
1481 }
1482 @Perm(requires="share(this) in alive",
1483 ensures="share(this) in alive")
1484     int gcd(int a, int b) {
1485         return 0;
1486     }
1487 }
1488 @Perm(requires="pure(this) in alive",
1489 ensures="pure(this) in alive")
1490     public Rational div(Rational r) {
1491         return null;
1492     }
1493 }
1494 @Perm(requires="pure(this) in alive",
1495 ensures="pure(this) in alive")
1496     public Rational plus(Rational r) {
1497         return null;
1498     }
1499 }
1500 @Perm(requires="pure(this) in alive",
1501 ensures="pure(this) in alive")
1502     Rational zero() {
1503         return null;
1504     }
1505 }
1506 @Perm(requires="share(this) in alive",
1507 ensures="share(this) in alive")
1508     public Rational minus(Rational r) {
1509         return null;
1510     }
1511 }
1512 @Perm(requires="pure(this) in alive",
1513 ensures="pure(this) in alive")
1514     public boolean isNegative() {
1515         return 0;
1516     }
1517 }
1518 @Perm(requires="pure(this) in alive",
1519 ensures="pure(this) in alive")
1520     public boolean isSmallerThan(Rational other) {
1521         return 0;
1522     }
1523 }
1524 @Perm(requires="pure(this) in alive",
1525 ensures="pure(this) in alive")
1526     public Rational times(int i) {
1527         return null;

```

```

1529 }
1530 @Perm(requires="pure(this) in alive",
1531 ensures="pure(this) in alive")
1532 public Rational inverse() {
1533     return null;
1534 }
1535 }
1536 @Perm(requires="immutable(this) in alive",
1537 ensures="immutable(this) in alive")
1538 public int getP() {
1539     return 0;
1540 }
1541 }
1542 @Perm(requires="pure(this) in alive",
1543 ensures="pure(this) in alive")
1544 public int getQ() {
1545     return 0;
1546 }
1547 }
1548 @Perm(requires="pure(this) in alive",
1549 ensures="pure(this) in alive")
1550 public boolean isOne() {
1551     return 0;
1552 }
1553 }
1554 @Perm(requires="pure(this) in alive",
1555 ensures="pure(this) in alive")
1556 public String toString() {
1557     return null;
1558 }
1559 }
1560 @Perm(requires="pure(this) in alive",
1561 ensures="pure(this) in alive")
1562 public int hashCode() {
1563     return 0;
1564 }
1565 }
1566 @Perm(requires="pure(this) in alive",
1567 ensures="pure(this) in alive")
1568 public boolean equals(Object obj) {
1569     return 0;
1570 }
1571 }
1572 }ENDOFCLASS
1573
1574 @ClassStates({@State(name = "alive")})
1575
1576 class GeneralizedSum {
1577     @Perm(ensures="unique(this) in alive")
1578     GeneralizedSum() { }
1579
1580     @Perm(requires="share(this) in alive",
1581     ensures="share(this) in alive")
1582     public Set<Fraction> getFractions() {
1583         return null;
1584     }
1585 }
1586 }
1587 @Perm(requires="share(this) in alive",
1588 ensures="share(this) in alive")
1589 public Rational getCoefficient(Fraction f) {
1590     return null;
1591 }
1592 }
1593 @Perm(requires="share(this) in alive",
1594 ensures="share(this) in alive")
1595 public boolean equals(Object obj) {
1596     return 0;
1597 }
1598 }
1599 @Perm(requires="share(this) in alive",
1600 ensures="share(this) in alive")
1601 public Rational getConstant() {
1602     return null;
1603 }
1604 }
1605 @Perm(requires="pure(this) in alive",
1606 ensures="pure(this) in alive")
1607 public boolean isGround() {
1608     return 0;

```

```

1610 }
1611 @Perm(requires="share(this) in alive",
1612 ensures="share(this) in alive")
1613 public String toString() {
1614     return null;
1615 }
1616 }
1617 @Perm(requires="share(this) in alive",
1618 ensures="share(this) in alive")
1619 public int hashCode() {
1620     return 0;
1621 }
1622 }
1623 }ENDOFCLASS
1624
1625 @ClassStates({@State(name = "alive")})
1626
1627 class VariableRelativity {
1628     @Perm(ensures="unique(this) in alive")
1629     VariableRelativity() { }
1630
1631     @Perm(requires="unique(this) in alive",
1632     ensures="unique(this) in alive")
1633     public boolean addRight(Relop relop, NormalizedFractionSum term) {
1634         return 0;
1635     }
1636 }
1637 }
1638 @Perm(requires="unique(this) in alive",
1639 ensures="unique(this) in alive")
1640 public boolean addLeft(NormalizedFractionSum term, Relop relop) {
1641     return 0;
1642 }
1643 }
1644 @Perm(requires="unique(this) in alive",
1645 ensures="unique(this) in alive")
1646 public Set<NormalizedFractionConstraint> dumpRelations() {
1647     return null;
1648 }
1649 }
1650 @Perm(requires="share(this) in alive",
1651 ensures="share(this) in alive")
1652 private void dumpRelation(NormalizedFractionSum less, Relop relop, NormalizedFractionSum more) {
1653 }
1654 }
1655 }ENDOFCLASS
1656
1657 @ClassStates({@State(name = "alive")})
1658
1659 class NormalizedFractionSum {
1660     @Perm(ensures="unique(this) in alive")
1661     NormalizedFractionSum() { }
1662
1663     @Perm(requires="immutable(this) in alive",
1664     ensures="immutable(this) in alive")
1665     NormalizedFractionSum zero() {
1666         return null;
1667     }
1668 }
1669 }
1670 }ENDOFCLASS
1671
1672 @ClassStates({@State(name = "alive")})
1673
1674 class SmtLibPrinter {
1675     @Perm(ensures="unique(this) in alive")
1676     SmtLibPrinter() { }
1677
1678     @Perm(requires="unique(this) in alive",
1679     ensures="unique(this) in alive")
1680     public String toString(FractionConstraints constraints, Boolean satisfiable) {
1681         return null;
1682     }
1683 }
1684 }
1685 @Perm(ensures="none(this) in alive")
1686 public String printStatus() {
1687     return null;
1688 }
1689 }

```

```

1690 @Perm(ensures="none(this) in alive")
1691 public SmtBenchmarkStatus getInverse() {
1692     return null;
1693 }
1694
1695 }ENDOFCLASS
1696
1697 @ClassStates({@State(name = "alive")})
1698
1699 class SmtLibBenchmarkPrinter {
1700     @Perm(ensures="unique(this) in alive")
1701     SmtLibBenchmarkPrinter() { }
1702
1703     @Perm(requires="share(this) in alive",
1704           ensures="share(this) in alive")
1705     public void addLineComment(String commentLine) {
1706
1707     }
1708     @Perm(requires="share(this) in alive",
1709           ensures="share(this) in alive")
1710     public void addStatus(SmtBenchmarkStatus status) {
1711
1712     }
1713     @Perm(requires="unique(this) in alive",
1714           ensures="unique(this) in alive")
1715     public void addFormula(String formula) {
1716
1717     }
1718     @Perm(requires="share(this) in alive",
1719           ensures="share(this) in alive")
1720     public void addUnknown(String name, String sort) {
1721
1722     }
1723     @Perm(requires="share(this) in alive",
1724           ensures="share(this) in alive")
1725     public void addAssumption(Set<String> formulae) {
1726
1727     }
1728     @Perm(requires="share(this) in alive",
1729           ensures="share(this) in alive")
1730     private void appendConjunction(Set<String> preds) {
1731
1732     }
1733     @Perm(requires="unique(this) in alive",
1734           ensures="unique(this) in alive")
1735     public void addNegatedQuantifiedImplicationFormula(Set<String> assumptions, Set<String> exists, Set<
1736               String> conclusion) {
1737
1738     }
1739     @Perm(requires="share(this) in alive",
1740           ensures="share(this) in alive")
1741     private void appendExists(Set<String> exists, Set<String> preds) {
1742
1743     }
1744     @Perm(requires="pure(this) in alive",
1745           ensures="pure(this) in alive")
1746     public String getResult() {
1747         return null;
1748     }
1749 }
1750
1751 }ENDOFCLASS
1752
1753 @ClassStates({@State(name = "alive")})
1754
1755 class Anonymous {
1756     @Perm(ensures="unique(this) in alive")
1757     Anonymous() { }
1758
1759     @Perm(requires="unique(this) in alive",
1760           ensures="unique(this) in alive")
1761     public Boolean named(NamedFraction fract) {
1762         return null;
1763     }
1764
1765     @Perm(requires="unique(this) in alive",
1766           ensures="unique(this) in alive")
1767     public Boolean one(OneFraction fract) {
1768         return null;

```

```

1770 }
1771 @Perm(requires="unique(this) in alive",
1772 ensures="unique(this) in alive")
1773 public Boolean var(VariableFraction fract) {
1774     return null;
1775 }
1776 }
1777
1778 public Boolean zero(ZeroFraction fract) {
1779     return null;
1780 }
1781 }
1782 }ENDOFCLASS
1783
1784 @ClassStates({@State(name = "alive")})
1785
1786 class OneFraction {
1787     @Perm(ensures="unique(this) in alive")
1788     OneFraction() { }
1789
1790
1791     public boolean isOne() {
1792         return 0;
1793     }
1794
1795     @Perm(requires="unique(this) in alive",
1796     ensures="unique(this) in alive")
1797     public T dispatch(FractionVisitor<T> visitor) {
1798         return null;
1799     }
1800 }
1801
1802
1803 public String toString() {
1804     return null;
1805 }
1806 }
1807 }ENDOFCLASS
1808
1809 @ClassStates({@State(name = "alive")})
1810
1811 class FractionTerm {
1812     @Perm(ensures="unique(this) in alive")
1813     FractionTerm() { }
1814
1815
1816     @Perm(requires="immutable(this) in alive",
1817     ensures="immutable(this) in alive")
1818     FractionTerm createSum(Fraction... summands) {
1819         return null;
1820     }
1821
1822     @Perm(requires="immutable(this) in alive",
1823     ensures="immutable(this) in alive")
1824     FractionTerm createSumOverload(List<Fraction> summands) {
1825         return null;
1826     }
1827 }
1828
1829 public int compareTo(final FractionTerm o) {
1830     return 0;
1831 }
1832 }
1833 }ENDOFCLASS
1834
1835 @ClassStates({@State(name = "alive")})
1836
1837 class Anonymous {
1838     @Perm(ensures="unique(this) in alive")
1839     Anonymous() { }
1840
1841
1842     @Perm(requires="unique(this) in alive",
1843     ensures="unique(this) in alive")
1844     public Integer literal(Fraction fract) {
1845         return null;
1846     }
1847 }
1848
1849 @Perm(requires="share(this) in alive",
1850 ensures="share(this) in alive")
1851 public Integer sum(FractionSum fract) {

```

```

1851     return null;
1852 }
1853 }ENDOFCLASS
1854
1855 @ClassStates({@State(name = "alive")})
1856
1857 class Anonymous {
1858   @Perm(ensures="unique(this) in alive")
1859   Anonymous() { }
1860
1861   @Perm(requires="unique(this) in alive",
1862         ensures="unique(this) in alive")
1863   public Boolean named(NamedFraction fract) {
1864     return null;
1865   }
1866
1867   @Perm(requires="unique(this) in alive",
1868         ensures="unique(this) in alive")
1869   public Boolean one(OneFraction fract) {
1870     return null;
1871   }
1872
1873   @Perm(requires="unique(this) in alive",
1874         ensures="unique(this) in alive")
1875   public Boolean var(VariableFraction fract) {
1876     return null;
1877   }
1878 }
1879
1880 public Boolean zero(ZeroFraction fract) {
1881   return null;
1882 }
1883
1884 }ENDOFCLASS
1885
1886 @ClassStates({@State(name = "alive")})
1887
1888 class Anonymous {
1889   @Perm(ensures="unique(this) in alive")
1890   Anonymous() { }
1891
1892   @Perm(requires="unique(this) in alive",
1893         ensures="unique(this) in alive")
1894   public Boolean one(OneFraction fract) {
1895     return null;
1896   }
1897
1898   public Boolean zero(ZeroFraction fract) {
1899     return null;
1900   }
1901
1902   @Perm(requires="unique(this) in alive",
1903         ensures="unique(this) in alive")
1904   public Boolean named(NamedFraction fract) {
1905     return null;
1906   }
1907
1908   @Perm(requires="unique(this) in alive",
1909         ensures="unique(this) in alive")
1910   public Boolean var(VariableFraction fract) {
1911     return null;
1912   }
1913 }
1914
1915 }ENDOFCLASS
1916
1917 @ClassStates({@State(name = "alive")})
1918
1919 class Relop {
1920   @Perm(ensures="unique(this) in alive")
1921   Relop() { }
1922
1923   @Perm(ensures="none(this) in alive")
1924   public String toString() {
1925     return null;
1926   }
1927 }
1928
1929 }

```

```

1933 }ENDOFCLASS
1935 @ClassStates({@State(name = "alive")})
1937 class Anonymous {
1938   @Perm(ensures="unique(this) in alive")
1939   Anonymous() { }
1941   @Perm(requires="unique(this) in alive",
1942     ensures="unique(this) in alive")
1943   public Boolean named(NamedFraction fract) {
1944     return null;
1946   }
1947   @Perm(requires="unique(this) in alive",
1948     ensures="unique(this) in alive")
1949   public Boolean one(OneFraction fract) {
1950     return null;
1952   }
1953   @Perm(requires="unique(this) in alive",
1954     ensures="unique(this) in alive")
1955   public Boolean var(VariableFraction fract) {
1956     return null;
1958   }
1960   public Boolean zero(ZeroFraction fract) {
1961     return null;
1963   }
1965 }ENDOFCLASS
1967 @ClassStates({@State(name = "alive")})
1969 class RelationFractionPair {
1970   @Perm(ensures="unique(this) in alive")
1971   RelationFractionPair() { }
1973   @Perm(requires="pure(this) in alive",
1974     ensures="pure(this) in alive")
1975   RelationFractionPair createEqual(NormalizedFractionTerm c1, NormalizedFractionTerm c2) {
1976     return null;
1978   }
1979   @Perm(requires="immutable(this) in alive",
1980     ensures="immutable(this) in alive")
1981   RelationFractionPair createLeq(NormalizedFractionTerm c1, NormalizedFractionTerm c2) {
1982     return null;
1984   }
1985   @Perm(requires="pure(this) in alive",
1986     ensures="pure(this) in alive")
1987   RelationFractionPair createLess(NormalizedFractionTerm c1, NormalizedFractionTerm c2) {
1988     return null;
1990   }
1991   @Perm(requires="pure(this) in alive",
1992     ensures="pure(this) in alive")
1993   public Relop getRelop() {
1994     return null;
1996   }
1997   @Perm(requires="pure(this) in alive",
1998     ensures="pure(this) in alive")
1999   public String toString() {
2000     return null;
2002   }
2003   @Perm(requires="share(this) in alive",
2004     ensures="share(this) in alive")
2005   public int hashCode() {
2006     return 0;
2008   }
2009   @Perm(requires="share(this) in alive",
2010     ensures="share(this) in alive")
2011   public boolean equals(Object obj) {
2012     return 0;

```

```

2014 }
2016 }ENDOFCLASS
2018 @ClassStates({@State(name = "alive")})
2020 class Anonymous {
2021   @Perm(ensures="unique(this) in alive")
2022   Anonymous() { }
2024   @Perm(requires="share(this) in alive",
2025         ensures="share(this) in alive")
2026   private NormalizedFractionConstraint createRelation(NormalizedFractionSum left, Relop relop,
2027                                                       NormalizedFractionSum right) {
2028     return null;
2029   }
2030   @Perm(requires="share(this) in alive",
2031         ensures="share(this) in alive")
2032   public int compare(VariableFraction o1, VariableFraction o2) {
2033     return 0;
2034   }
2035   @Perm(requires="share(this) in alive",
2036         ensures="share(this) in alive")
2037   private Fraction getRepresentative(Fraction fract) {
2038     return null;
2039   }
2040   @Perm(requires="share(this) in alive",
2041         ensures="share(this) in alive")
2042   private Fraction[] getRepresentatives(List<Fraction> summands) {
2043     return null;
2044   }
2045   }
2046   }
2047   }
2048   public Boolean impossible(ImpossibleConstraint fract) {
2049     return null;
2050   }
2051   }
2052   @Perm(requires="share(this) in alive",
2053         ensures="share(this) in alive")
2054   public Boolean relation(FractionRelation fract) {
2055     return null;
2056   }
2057   }
2058   @Perm(requires="unique(this) in alive",
2059         ensures="unique(this) in alive")
2060   public Integer literal(Fraction fract) {
2061     return null;
2062   }
2063   }
2064   @Perm(requires="share(this) in alive",
2065         ensures="share(this) in alive")
2066   public Integer sum(FractionSum fract) {
2067     return null;
2068   }
2069   }
2070   }
2071   }ENDOFCLASS
2072   @ClassStates({@State(name = "alive")})
2073   class SimpleFractionSum {
2074     @Perm(ensures="unique(this) in alive")
2075     SimpleFractionSum() { }
2076     @Perm(requires="immutable(this) in alive",
2077           ensures="immutable(this) in alive")
2078     SimpleFractionSum createAdd(Fraction c1, Fraction c2) {
2079       return null;
2080     }
2081     @Perm(requires="share(this) in alive",
2082           ensures="share(this) in alive")
2083     SimpleFractionSum createSub(Fraction c1, Fraction c2) {
2084       return null;
2085     }
2086     }
2087     @Perm(requires="pure(this) in alive",

```



```

2093 ensures="pure(this) in alive")
2094 public Sumop getSumop() {
2095     return null;
2096 }
2097 }
2098 @Perm(requires="pure(this) in alive",
2099 ensures="pure(this) in alive")
2100 public String toString() {
2101     return null;
2102 }
2103 }
2104
2105 public T dispatch(NormalizedFractionVisitor<T> visitor) {
2106     return null;
2107 }
2108 }
2109 @Perm(requires="share(this) in alive",
2110 ensures="share(this) in alive")
2111 public int hashCode() {
2112     return 0;
2113 }
2114 }
2115 @Perm(requires="share(this) in alive",
2116 ensures="share(this) in alive")
2117 public boolean equals(Object obj) {
2118     return 0;
2119 }
2120 }
2121 }ENDOFCLASS
2122
2123 @ClassStates({@State(name = "alive")})
2124
2125 class Sumop {
2126     @Perm(ensures="unique(this) in alive")
2127     Sumop() { }
2128 }
2129
2130 @Perm(ensures="none(this) in alive")
2131 public String toString() {
2132     return null;
2133 }
2134 }
2135 }ENDOFCLASS
2136
2137 @ClassStates({@State(name = "alive")})
2138
2139 class SimpleVariableRelativity {
2140     @Perm(ensures="unique(this) in alive")
2141     SimpleVariableRelativity() { }
2142 }
2143
2144 @Perm(requires="unique(this) in alive",
2145 ensures="unique(this) in alive")
2146 public boolean addRight(Relop relop, NormalizedFractionTerm term) {
2147     return 0;
2148 }
2149 }
2150 @Perm(requires="unique(this) in alive",
2151 ensures="unique(this) in alive")
2152 public boolean addLeft(NormalizedFractionTerm term, Relop relop) {
2153     return 0;
2154 }
2155 }
2156 @Perm(requires="pure(this) in alive",
2157 ensures="pure(this) in alive")
2158 public Set<RelationFractionPair> dumpRelations(Set<RelationFractionPair> result) {
2159     return null;
2160 }
2161 }
2162 }ENDOFCLASS
2163
2164 @ClassStates({@State(name = "alive")})
2165
2166 class Anonymous {
2167     @Perm(ensures="unique(this) in alive")
2168     Anonymous() { }
2169 }
2170
2171 @Perm(requires="unique(this) in alive",
2172 ensures="unique(this) in alive")
2173 public Boolean named(NamedFraction fract) {

```

```

2174     return null;
2175 }
2176 @Perm(requires="unique(this) in alive",
2177 ensures="unique(this) in alive")
2178 public Boolean one(OneFraction fract) {
2179     return null;
2180 }
2181
2182 }
2183 @Perm(requires="share(this) in alive",
2184 ensures="share(this) in alive")
2185 public Integer sum(FractionSum fract) {
2186     return null;
2187 }
2188
2189 @Perm(requires="unique(this) in alive",
2190 ensures="unique(this) in alive")
2191 public Boolean var(VariableFraction fract) {
2192     return null;
2193 }
2194 }
2195
2196 public Boolean zero(ZeroFraction fract) {
2197     return null;
2198 }
2199 }
2200 }ENDOFCLASS
2201
2202 @ClassStates({@State(name = "alive")})
2203
2204 class FractionElimination {
2205     @Perm(ensures="unique(this) in alive")
2206     FractionElimination() { }
2207
2208     @Perm(requires="unique(this) in alive",
2209     ensures="unique(this) in alive")
2210     private Set<RelationFractionPair> eliminateVariableOverload(Set<RelationFractionPair> rels,
2211     VariableFraction x) {
2212         return null;
2213     }
2214 }
2215 @Perm(requires="share(this) in alive",
2216 ensures="share(this) in alive")
2217     int containsVariable(final NormalizedFractionTerm t, final VariableFraction x) {
2218         return 0;
2219     }
2220 }
2221 @Perm(requires="share(this) in alive",
2222 ensures="share(this) in alive")
2223     RelationFractionPair subtractVariable(final RelationFractionPair rel, final VariableFraction x, final
2224     int sign) {
2225         return null;
2226     }
2227 }
2228 @Perm(requires="unique(this) in alive",
2229 ensures="unique(this) in alive")
2230     public Set<RelationFractionPair> eliminateVariables(Set<FractionConstraint> constraints) {
2231         return null;
2232     }
2233 }
2234 @Perm(requires="unique(this) in alive",
2235 ensures="unique(this) in alive")
2236     private Set<RelationFractionPair> normalizeConstraints(Set<FractionConstraint> constraints) {
2237         return null;
2238     }
2239 }
2240 @Perm(requires="immutable(this) in alive",
2241 ensures="immutable(this) in alive")
2242     private Set<VariableFraction> collectVariables(Set<RelationFractionPair> rels) {
2243         return null;
2244     }
2245 }
2246
2247 public boolean isConsistent() {
2248     return 0;
2249 }
2250 }
2251 }ENDOFCLASS

```

```

2253 @ClassStates({@State(name = "alive")})
2254
2255 class FractionPair {
2256   @Perm(ensures="unique(this) in alive")
2257   FractionPair() { }
2258
2259   @Perm(requires="share(this) in alive",
2260         ensures="share(this) in alive")
2261   public T getComponent1() {
2262     return null;
2263   }
2264
2265   @Perm(requires="share(this) in alive",
2266         ensures="share(this) in alive")
2267   public T getComponent2() {
2268     return null;
2269   }
2270
2271   @Perm(requires="share(this) in alive",
2272         ensures="share(this) in alive")
2273   public int hashCode() {
2274     return 0;
2275   }
2276
2277   @Perm(requires="unique(this) in alive",
2278         ensures="unique(this) in alive")
2279   public boolean equals(Object obj) {
2280     return 0;
2281   }
2282 }
2283
2284 }ENDOFCLASS
2285
2286 @ClassStates({@State(name = "alive")})
2287
2288 class Anonymous {
2289   @Perm(ensures="unique(this) in alive")
2290   Anonymous() { }
2291
2292   @Perm(requires="unique(this) in alive",
2293         ensures="unique(this) in alive")
2294   public Boolean named(NamedFraction fract) {
2295     return null;
2296   }
2297
2298   @Perm(requires="unique(this) in alive",
2299         ensures="unique(this) in alive")
2300   public Boolean one(OneFraction fract) {
2301     return null;
2302   }
2303
2304   @Perm(requires="share(this) in alive",
2305         ensures="share(this) in alive")
2306   public Integer sum(FractionSum fract) {
2307     return null;
2308   }
2309
2310   @Perm(requires="unique(this) in alive",
2311         ensures="unique(this) in alive")
2312   public Boolean var(VariableFraction fract) {
2313     return null;
2314   }
2315 }
2316
2317 public Boolean zero(ZeroFraction fract) {
2318   return null;
2319 }
2320
2321 @Perm(requires="unique(this) in alive",
2322       ensures="unique(this) in alive")
2323 public Integer literal(Fraction fract) {
2324   return null;
2325 }
2326
2327
2328 public Boolean impossible(ImpossibleConstraint fract) {
2329   return null;
2330 }
2331
2332 @Perm(requires="share(this) in alive",
2333       ensures="share(this) in alive")

```

```

2334 public Boolean relation(FractionRelation fract) {
2335     return null;
2337 }
2339 }ENDOFCLASS
2341 @ClassStates({@State(name = "alive")})
2343 class Anonymous {
2344     @Perm(ensures="unique(this) in alive")
2345     Anonymous() { }
2347 }
2348 public String printStatus() {
2349     return null;
2351 }
2352 @Perm(requires="pure(this) in alive",
2353     ensures="pure(this) in alive")
2354     public SmtBenchmarkStatus getInverse() {
2355         return null;
2357 }
2359 }ENDOFCLASS
2361 @ClassStates({@State(name = "alive")})
2363 class SmtLibConstraintProcessor {
2364     @Perm(ensures="unique(this) in alive")
2365     SmtLibConstraintProcessor() { }
2367 }
2368 public Boolean impossible(ImpossibleConstraint fract) {
2369     return null;
2371 }
2372 @Perm(requires="share(this) in alive",
2373     ensures="share(this) in alive")
2374     public Boolean relation(FractionRelation fract) {
2375         return null;
2377 }
2378 @Perm(requires="pure(this) in alive",
2379     ensures="pure(this) in alive")
2380     private String formatRelation(String term1, Relop relop, String term2) {
2381         return null;
2383 }
2384 @Perm(requires="unique(this) in alive",
2385     ensures="unique(this) in alive")
2386     public Pair<String,Boolean> literal(Fraction fract) {
2387         return null;
2389 }
2390 @Perm(requires="unique(this) in alive",
2391     ensures="unique(this) in alive")
2392     public Pair<String,Boolean> sum(FractionSum fract) {
2393         return null;
2395 }
2396 @Perm(requires="full(this) in alive",
2397     ensures="full(this) in alive")
2398     public Pair<String,Boolean> named(NamedFraction fract) {
2399         return null;
2401 }
2403 public Pair<String,Boolean> one(OneFraction fract) {
2404     return null;
2406 }
2407 @Perm(requires="share(this) in alive",
2408     ensures="share(this) in alive")
2409     public Pair<String,Boolean> var(VariableFraction fract) {
2410         return null;
2412 }
2414 public Pair<String,Boolean> zero(ZeroFraction fract) {

```

```

2415     return null;
2417 }
2419 }ENDOFCLASS
2421 @ClassStates({@State(name = "alive")})
2423 class Impossible {
2424     @Perm(ensures="unique(this) in alive")
2425     Impossible() { }
2427     @Perm(requires="immutable(this) in alive",
2428     ensures="immutable(this) in alive")
2429     Impossible getInstance() {
2430         return null;
2432     }
2433     @Perm(requires="immutable(this) in alive",
2434     ensures="immutable(this) in alive")
2435     public boolean equals(Object obj) {
2436         return 0;
2438     }
2439     @Perm(requires="immutable(this) in alive",
2440     ensures="immutable(this) in alive")
2441     public int hashCode() {
2442         return 0;
2444     }
2446 }ENDOFCLASS

```