Università della Svizzera italiana

Faculty of Informatics

Software Testing and Analysis Research group

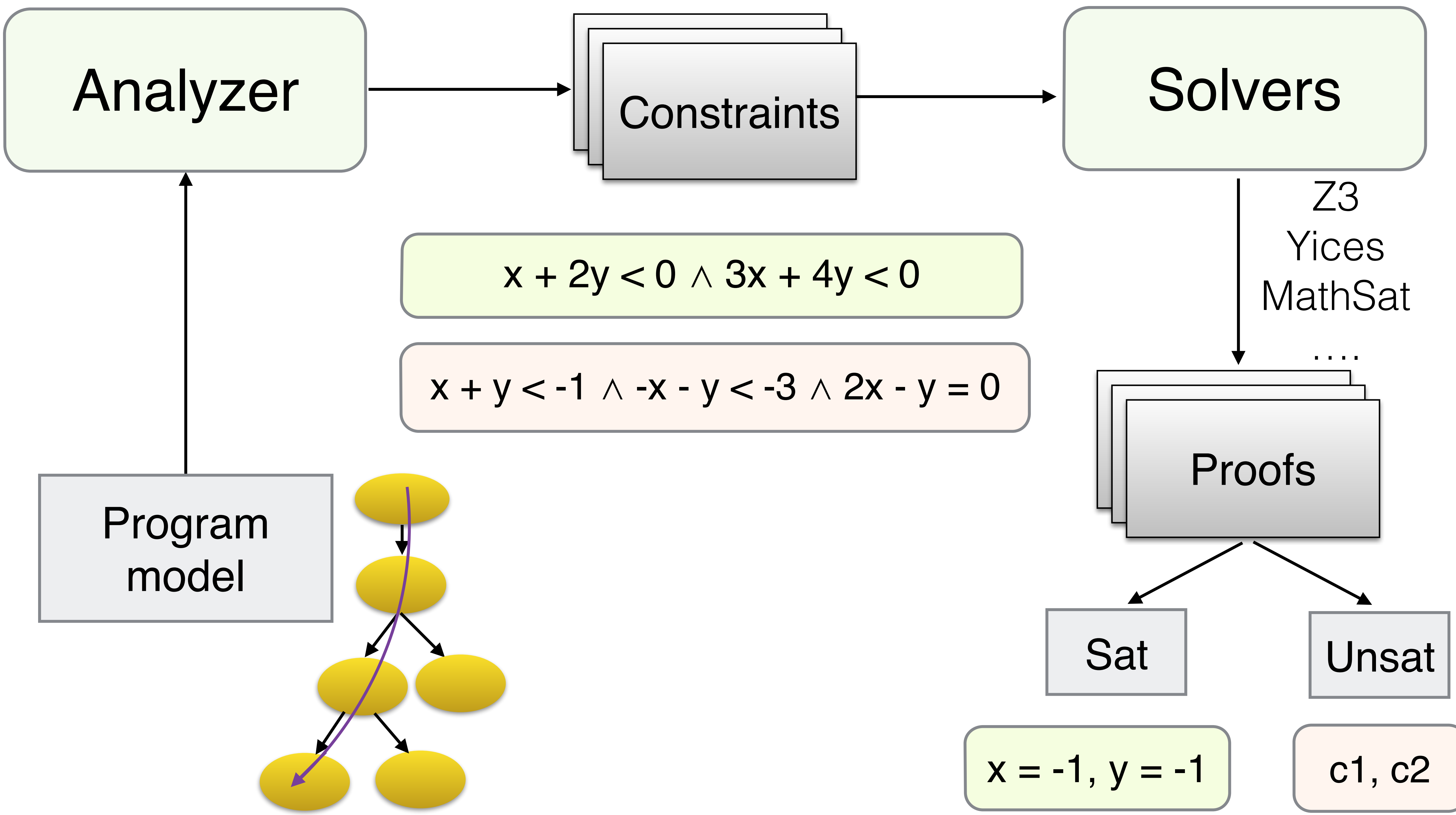
Reusing Constraint Proofs in Program Analysis

Andrea Aquino[*], Francesco A. Bianchi[*], **Meixian Chen**[*], Giovanni Denaro[+], Mauro Pezzè[*,+]

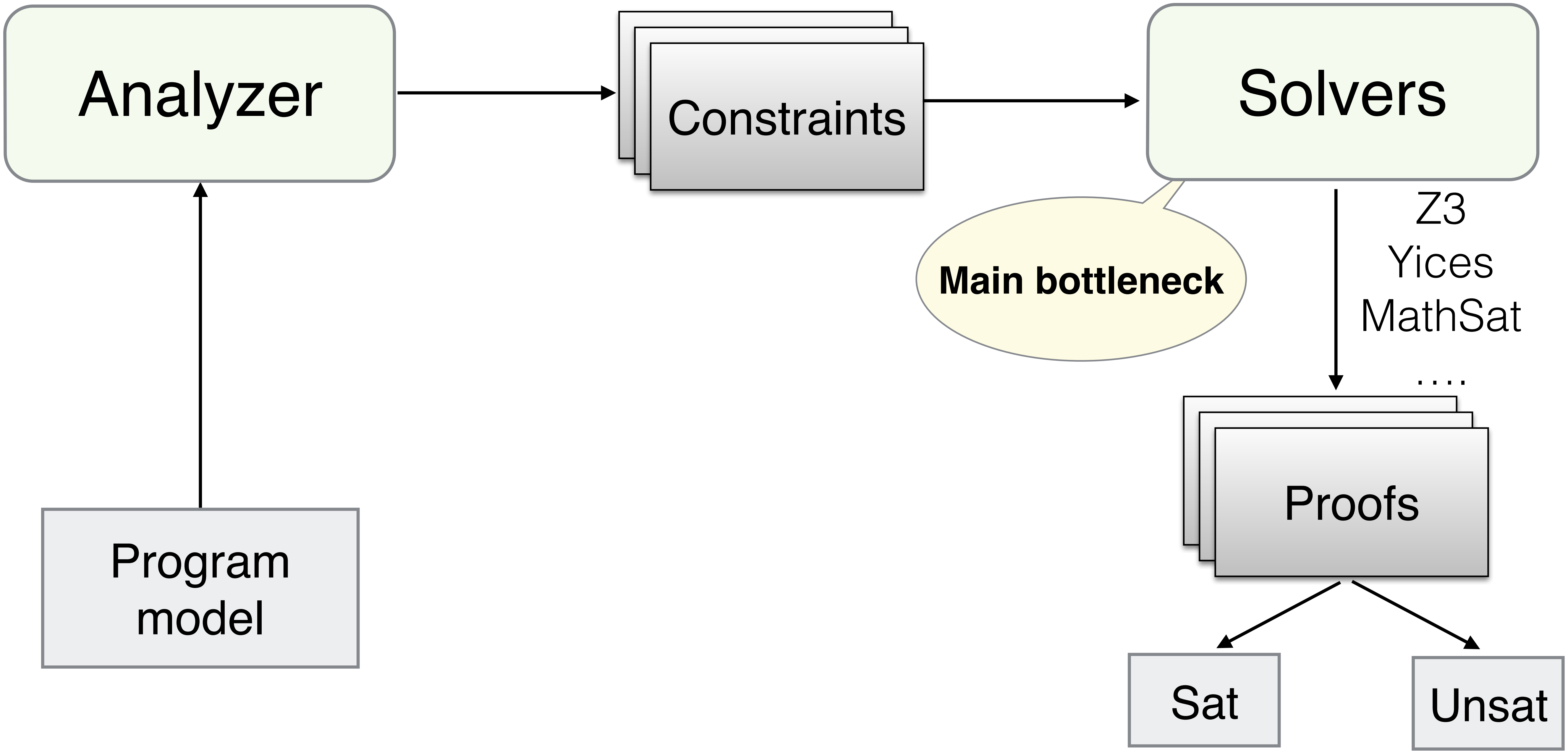
[*] *Università della Svizzera italiana (USI), Switzerland*

[+] *University of Milano-Bicocca, Italy*

Program Analysis

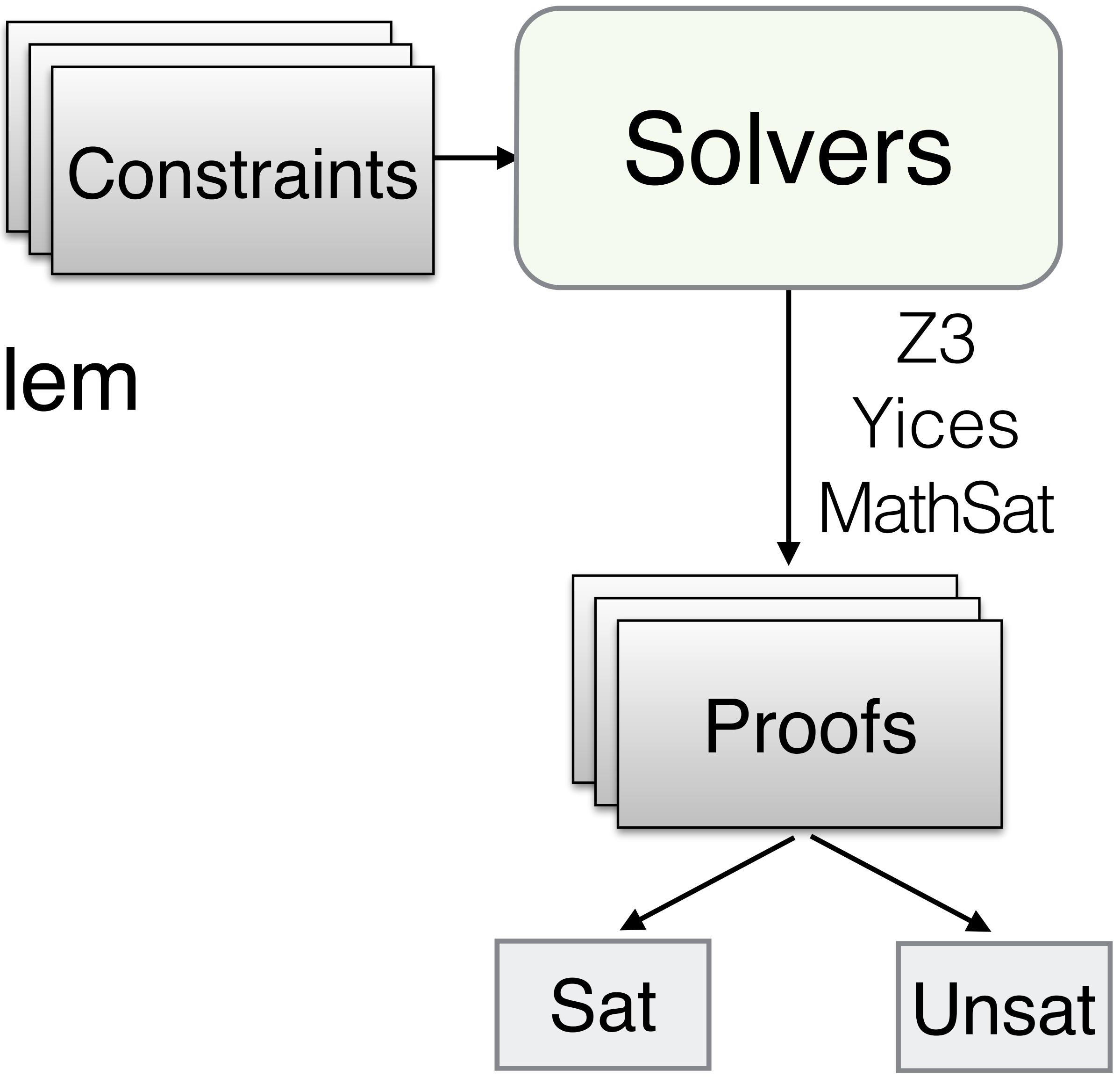


Main Bottleneck



Solving time accounts for 92% of overall execution time on average. (KLEE. Cadar et al. osdi'08)

Main Bottleneck



- High complexity of the SMT problem
- A large set of big constraints
- Solving time hard to predict

Solving time is hard to predict

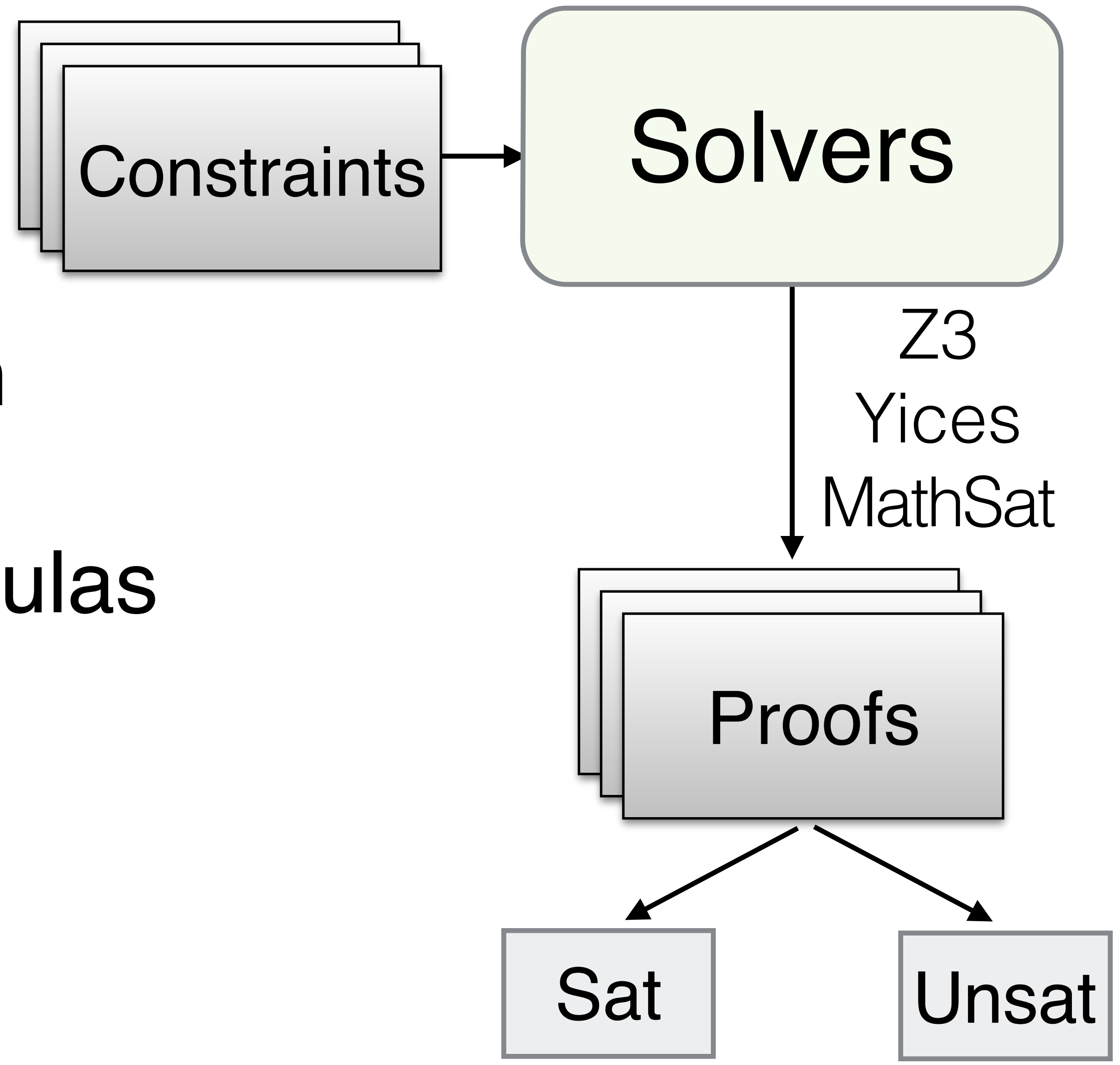
-2a + 85b - 90c - 44d + 39e + 96f - 76g - 88h - 72i - 79j ≤ 66
-100a - 19b + 60c - 96d - 42e - 30f + 82g + 75h + 73i - 41j ≤ 97
-56a + 96b - 15c - 45d - 33e - 42f + 50g … - 47i - 92j ≠ 64
41a + 79b + 9c - 96d - 35e + 24f - … i - 58j ≠ 41
-67a - 65b - 46c - 49d + 71e + 100f - … i + 64j ≤ 48
-80a + 59b + 95c - 4d + 32e + 39f + … i + 35j ≤ 32
68a + 70b + 66c - 43d + 32e - 69f + … g - 32h + 73i - 28j ≠ 12
-45a + 51b - 88c - 46d - 27e + 9f + 34g + 57h + 14i - 1j ≠ 60
-52a - 46b + 55c - 74d - 21e - 52f - 55g + 41h - 96i + 61j ≤ 9
53a + 68b + 3c + 15d + 50e - 38f + 25g - 82h - 96i + 11j ≤ 9

< 1 second

54a + 90b - 32c + 45d - 73e + 77f - 98g + 54h - 45i - 67j ≠ 4
52a + 22b + 71c + 40d + 21e - 75f - 75g + 13h + 33i - 18j ≤ 12
-17a - 100b + 56c - 94d + 79e + 19f + 39g - 53h - 78i + 98j ≤ 2
-38a + 72b - 86c - 8d + 54e - 68f + 44g … + 34i + 72j ≤ 81
66a - 73b + 86c - 44d - 66e + 22f + … - 91j ≤ 37
-51a - 64b - 19c + 80d - 74e + 37f - … - 30j ≠ 44
71a - 44b + 3c - 4d + 14e - 18f + 13… i - 60j ≠ 91
-89a + 4b - 73c + 5d + 39e + 4f + 8…g - 2h - 16i + 95j ≠ 37
13a + 56b + 87c - 39d - 60e - 36f + 35g + 74h - 3i + 5j ≤ 70
-37a + 51b - 30c + 24d + 34e + 63f + 84g - 34h + 91i + 39j ≠ 66

>> 10 minutes

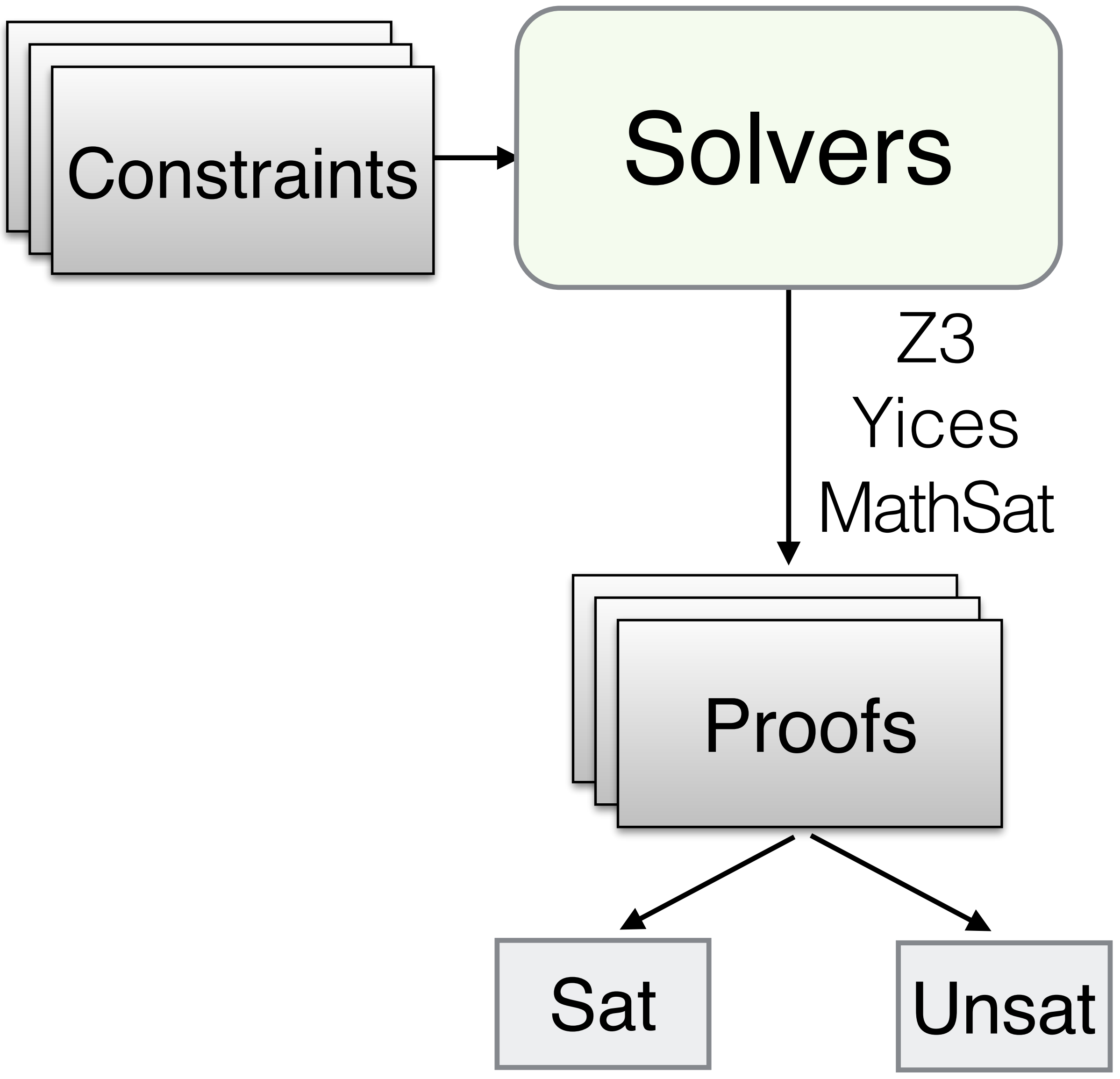
Main Bottleneck



- High complexity of SMT problem
- A large set of big constraint formulas
- Solving time hard to predict

Overcome the Bottleneck

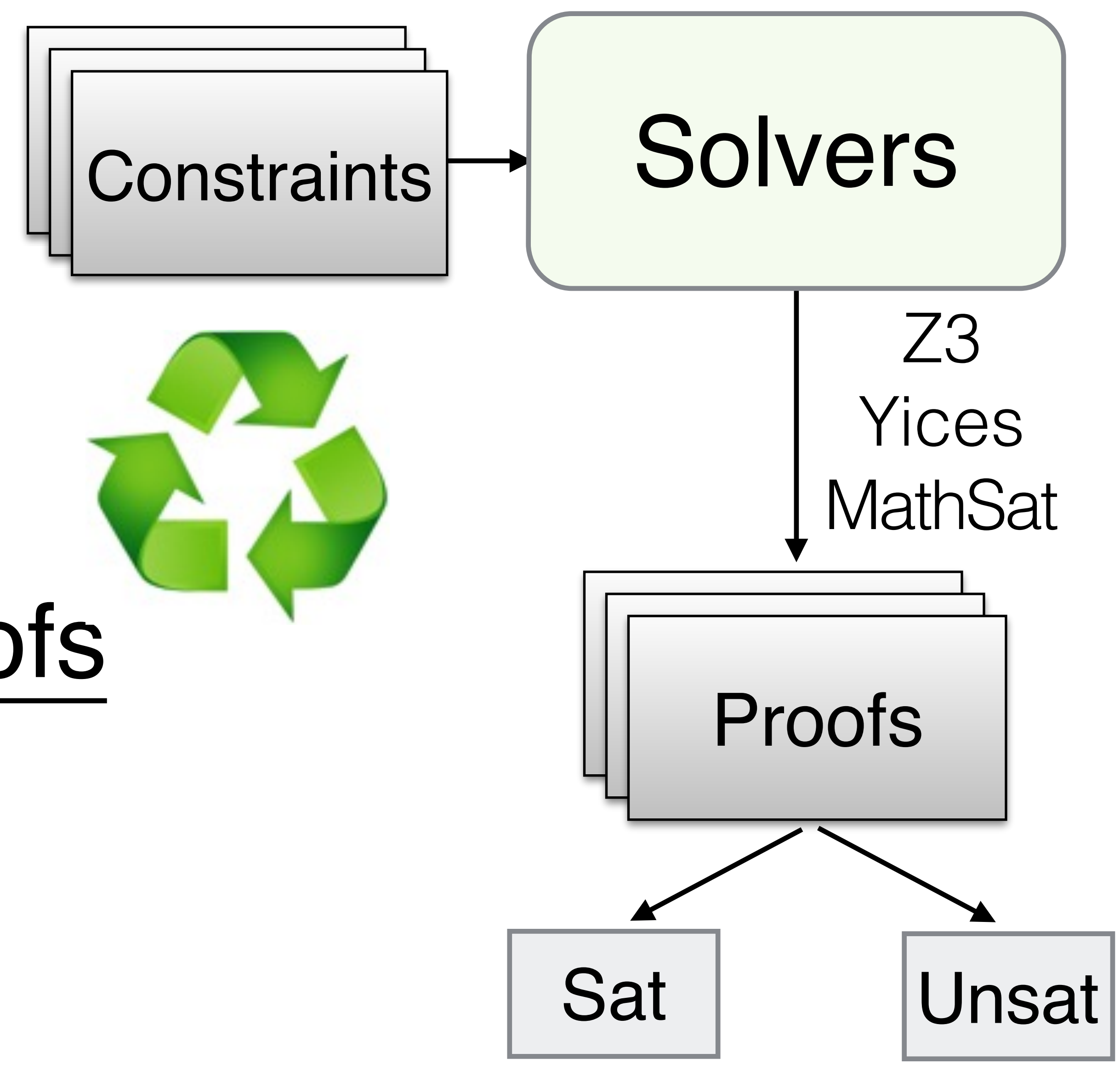
Improve solvers



Overcome the Bottleneck

Improve solvers

<u>Reuse constraint proofs</u>



Reuse Proofs

$x + y < 0 \wedge a + 2b \neq 9 \wedge x - y \neq 2 \wedge a - b > 10$

$x + y \geq 0 \wedge x - y = 2 \wedge a + 2b \neq 9 \wedge a - b > 10$

Reuse Proofs

$x + y < 0 \wedge \mathbf{a + 2b \neq 9} \wedge x - y \neq 2 \wedge \mathbf{a - b > 10}$

$x + y \geq 0 \wedge x - y = 2 \wedge \mathbf{a + 2b \neq 9} \wedge \mathbf{a - b > 10}$

Slicing

$x + y < 0 \wedge x - y \neq 2$	$x + y \geq 0 \wedge x - y = 2$
$\mathbf{a + 2b \neq 9} \wedge \mathbf{a - b > 10}$	$\mathbf{a + 2b \neq 9} \wedge \mathbf{a - b > 10}$

State of the Art

KLEE
(OSDI'08, Cadar et al.)

GREEN
(FSE'12, Visser et al.)

Slicing

Simplification

Variable renaming

Improve the State of the Art

KLEE
(OSDI’08, Cadar et al.)

GREEN
(FSE’12, Visser et al.)

Recognize More Reusable Constraints

(1) Equivalence by reordering terms and clauses

(2) Stricter constraints by containment and implication

(1) Equivalence by reordering terms and clauses

C_1: $x + 2y + 1 < 0 \wedge 3x + 4y - 1 < 0$

C_2: $4a + 3b - 1 < 0 \wedge 2a + b + 1 < 0$

$2y + x + 1 < 0 \wedge 4y + 3x - 1 < 0$ → $4y + 3x - 1 < 0 \wedge 2y + x + 1 < 0$ → $4V_1 + 3V_2 - 1 < 0 \wedge 2V_1 + V_2 + 1 < 0$

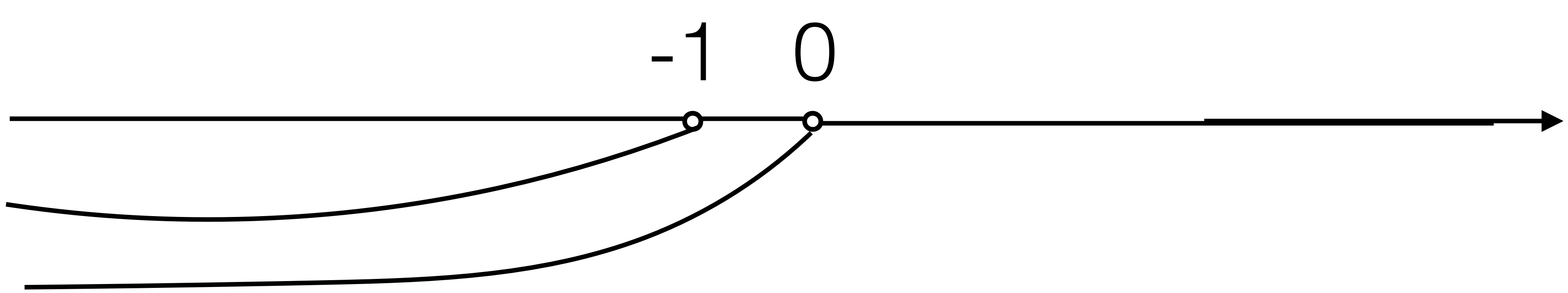
(2) Stricter constraints by containment and implication

C1:

X < -1

C2:

X < 0



Our Solution

(1) Equivalence by reordering terms and clauses

(2) Stricter constraints by containment and implication

(1) Equivalence by reordering terms and clauses

$$C_1 \equiv C_2 \text{ iff } C_1 \in \text{Permutation}(C_2)$$

Permutation-based Equivalence Problem = Graph Isomorphism Problem

Search for equivalent constraints?

Equivalent Constraints Search

via Canonical Form

$$C_1 \equiv C_2 \Leftrightarrow \text{canonical}(C_1) = \text{canonical}(C_2)$$

Equivalent Constraints Search

via Canonical Form

$C_1 \equiv C_2 \Leftrightarrow \text{canonical}(C_1) = \text{canonical}(C_2)$

C_1

$x + 2y + 1 \leq 0 \wedge$
$3x + 4y - 1 \leq 0$

1	2	1	≤
3	4	-1	≤

C_2

$4a + 3b - 1 \leq 0 \wedge$
$2a + b + 1 \leq 0$

4	3	-1	≤
2	1	1	≤

Canonical form

4	3	-1	≤
2	1	1	≤

The Canonicalization Algorithm

$$2a + b \leq 0$$
$$\wedge\ a + 2b \leq 0$$
$$\wedge\ a \neq 0$$
$$\wedge\ a + 3b \leq 0$$
$$\wedge\ a - 1 \leq 0$$

→

2	1	0	≤
1	2	0	≤
1	0	0	≠
1	3	0	≤
1	0	-1	≤

The Canonicalization Algorithm

sort rows by comparison and constant terms

2	1	0	≤
1	2	0	≤
1	0	0	≠
1	3	0	≤
1	0	-1	≤

The Canonicalization Algorithm

sort rows by comparison and constant terms

2	1	0	≤
1	2	0	≤
1	0	0	≠
1	3	0	≤
1	0	-1	≤

The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

2	1	**0**	**≤**
1	2	**0**	**≤**
1	3	**0**	**≤**
1	**0**	**-1**	**≤**
1	**0**	**0**	**≠**

0	initial
0	1-D locked
0	2-D locked

The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

1	**3**	**0**	**≤**
2	1	**0**	**≤**
1	2	**0**	**≤**
1	**0**	**-1**	**≤**
1	**0**	**0**	**≠**

0	initial
0	1-D locked
0	2-D locked

The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

sort 1-D-locked rows and columns lexicographically

3	1	0	≤
1	2	0	≤
2	1	0	≤
0	1	-1	≤
0	1	0	≠

0 initial

0 1-D locked

0 2-D locked

The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

sort 1-D-locked rows and columns lexicographically

3	**1**	**0**	≦
2	**1**	**0**	≦
1	**2**	**0**	≦
0	**1**	**-1**	≦
0	**1**	**0**	≠

0	initial
0	1-D locked
0	2-D locked

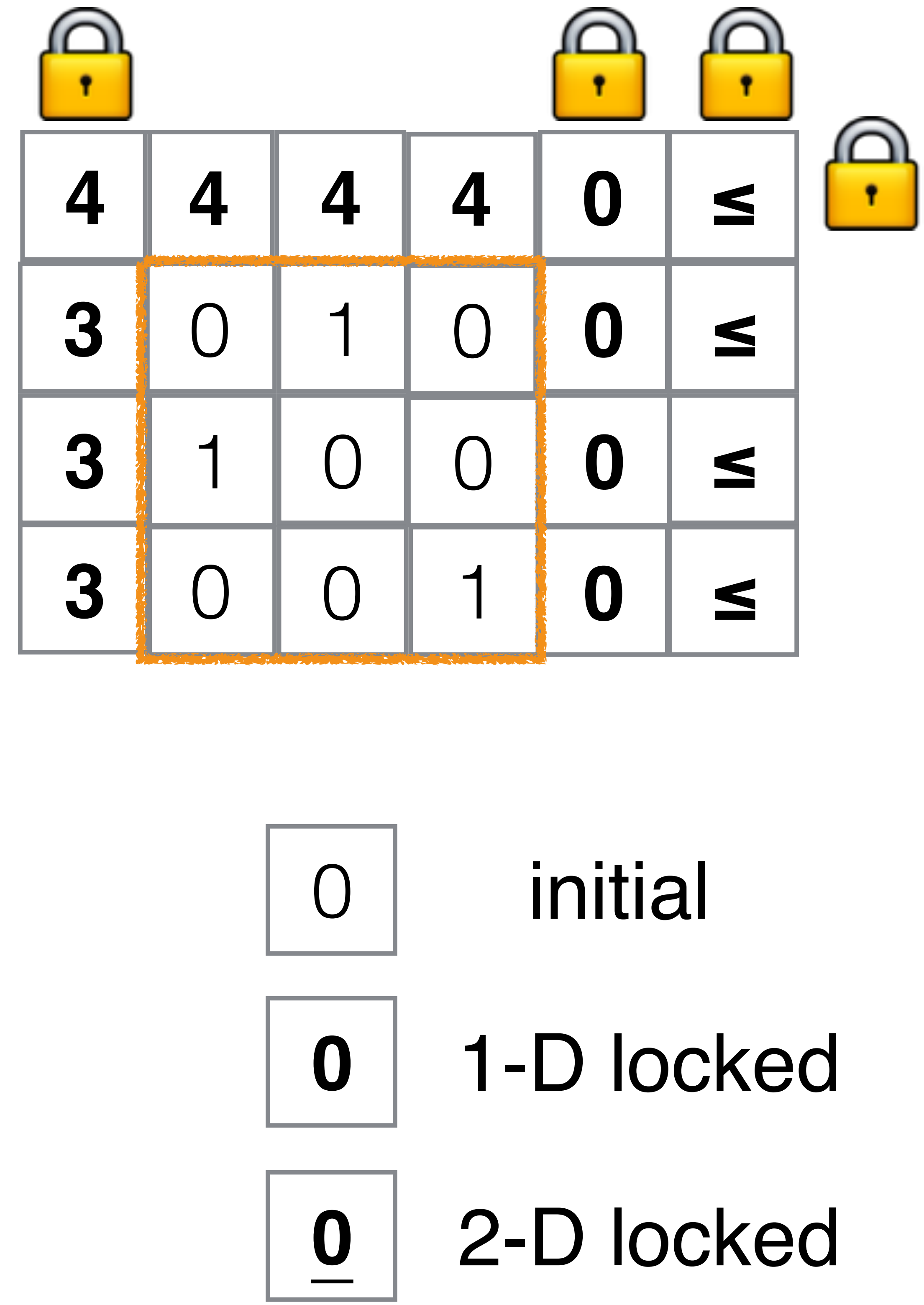
The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

sort 1-D-locked rows and columns lexicographically

sort the remaining rows and columns by brute-force



The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

sort 1-D-locked rows and columns lexicographically

sort the remaining rows and columns by brute-force

4	4	4	4	0	≦
3	1	0	0	0	≦
3	0	1	0	0	≦
3	0	0	1	0	≦

0	initial
0	1-D locked
0	2-D locked

The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values

sort 1-D-locked rows and columns lexicographically

Polynomial

sort the remaining rows and columns by brute-force

Exponential

93% of constraints converge up to the polynomial steps.

(2) Stricter constraints by containment and implication

What is a stricter constraint?

Search for stricter constraints?

Stricter Constraints

C_1

$3X < 0 \land$
$X + Y < 10$

Sat

$3X < 0 \land$
$X + Y < 10 \land$
$2X - Y = 0$

$3X < \underline{-1} \land$
$X + Y < 10$

Stricter Constraints

C_1

$3X < 0 \land$
$X + Y < 10$

Sat

$3X < 0 \land$
$X + Y < 10 \land$
$2X - Y = 0$

$3X < \underline{-1} \land$
$X + Y < 10$

C_2

$X + Y < -1 \land$
$-X - Y < -3 \land$
$2X - Y = 0$

UnSat

$X + Y < -1 \land$
$-X - Y < -3$

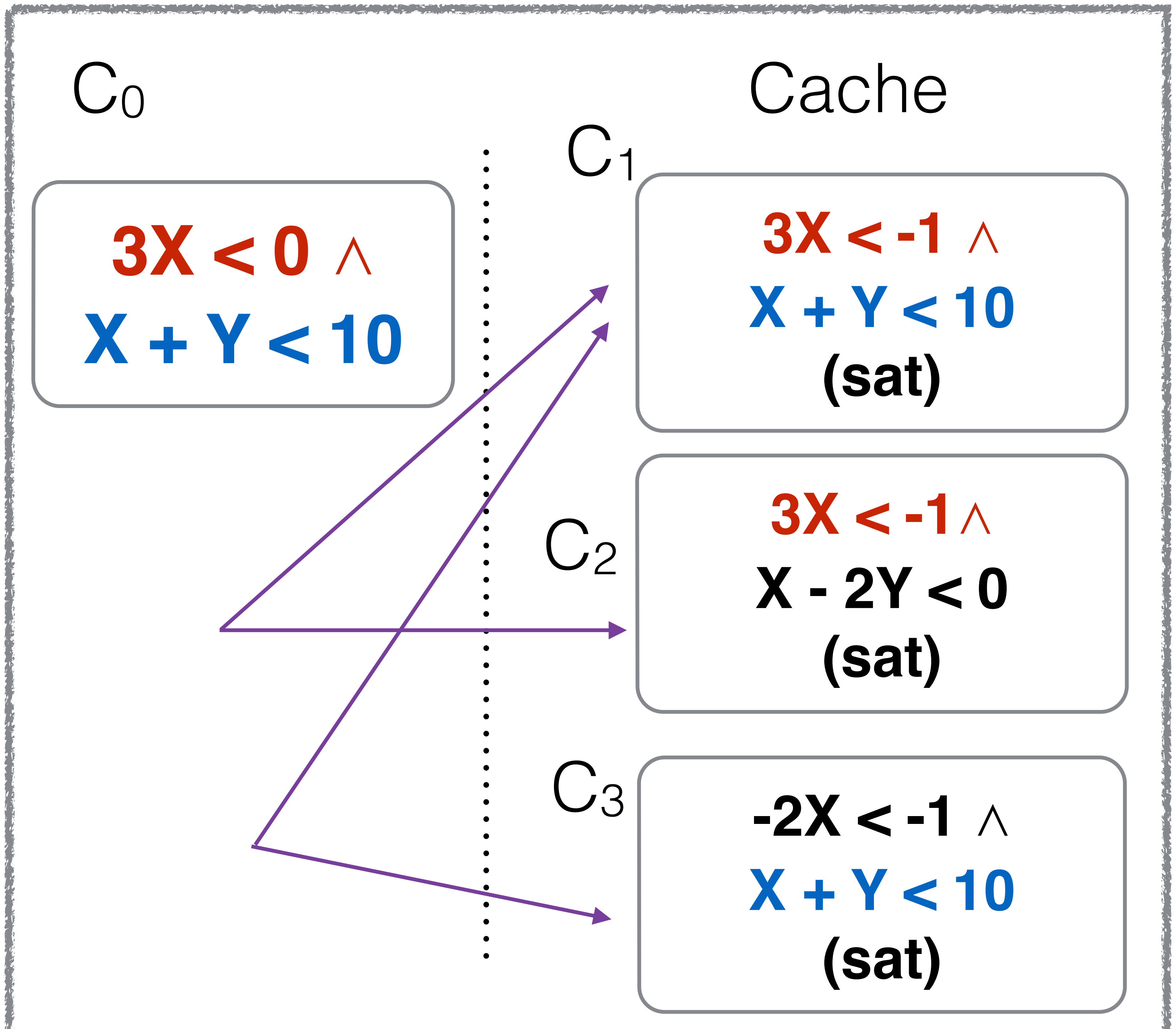
$X + Y < \underline{0} \land$
$-X - Y < -3$

Stricter Constraints Search

Clause-to-constraint index

Stricter Constraints Search

Clause-to-constraint index



$3X \longrightarrow \{C_1, C_2\}$

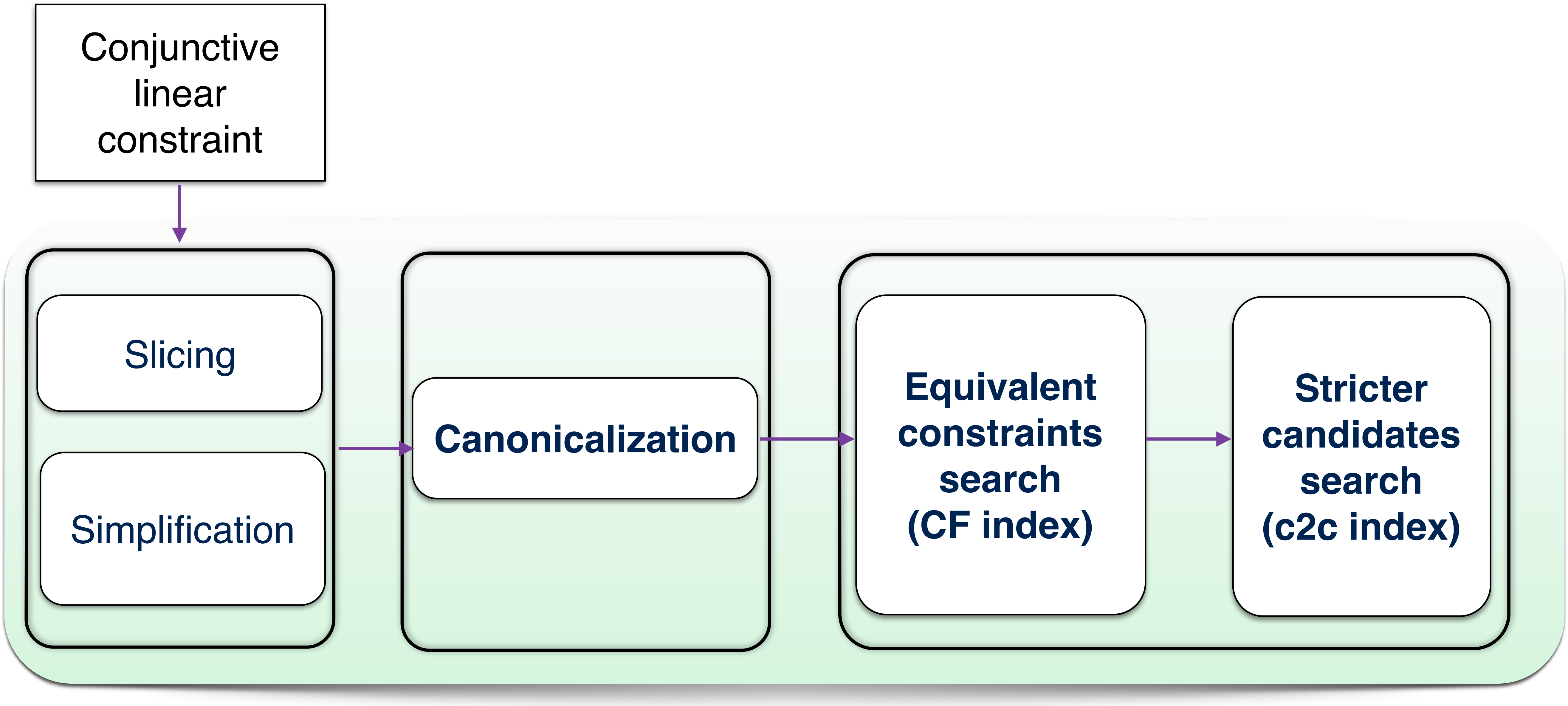
$X + Y \longrightarrow \{C_1, C_3\}$

intersection

$\{C_1, C_2\} \cap \{C_1, C_3\} = \{C_1\}$

The Recal Framework

The Recal Framework



Evaluation

Effectiveness: Can Recal effectively identify reusable constraints?

Efficiency: Is Recal more efficient than SMT solvers?

A large set of real-world constraints

Program	LOC	Language	#Constraints
old-tax	78	Java	27
new-tax	78	Java	35
afs	75	Java	48
dijkstra	142	Java	85
doubly-linked-list	806	Java	114
swapwords	30	Java	173
kbfiltr	599	C	188
wbs	297	Java	191
ball	15	Java	202
reverseword	32	Java	303
block	79	Java	336
division	87	Java	1,257
multiplication	50	Java	1,263
collision	21	Java	1,741
avl	519	Java	2,985
tcas	200	Java	9,780
treemap	806	Java	13,556
cdaudio	2171	C	55,329
floppy	1137	C	100,006
grep	10068	C	100,126
diskperf	1104	C	103,505

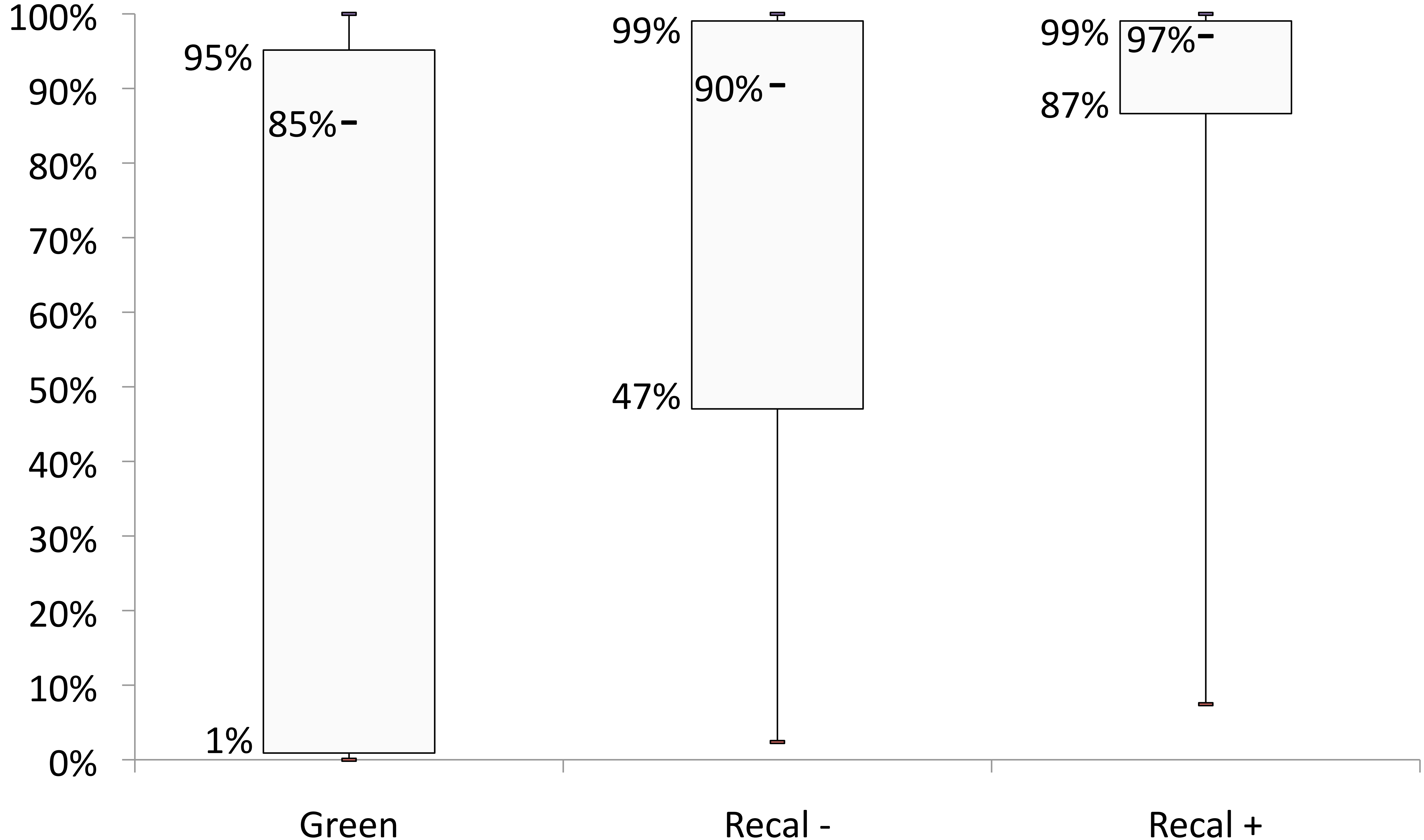
JBSE [Braione, et al., FSE'13]

CREST [Burnim, et al., EECS'08]

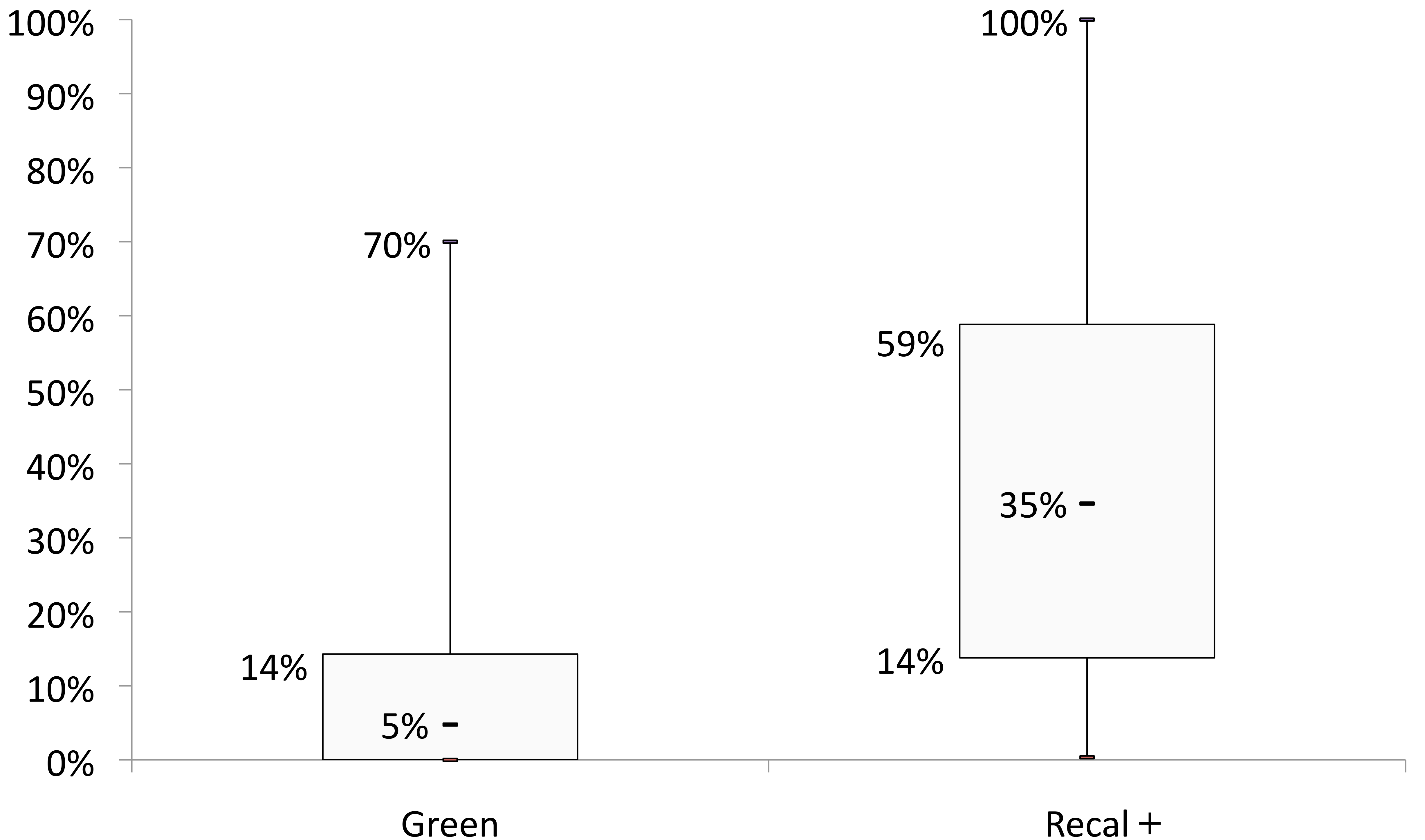
Constraints

391,250

Intra-program Reuse Rates



Inter-program Reuse Rates



High Reuse Rates

Program	LOC	Language	#Constraints
old-tax	78	Java	27
new-tax	78	Java	35
afs	75	Java	48
dijkstra	142	Java	85
doubly-linked-list	806	Java	114
swapwords	30	Java	173
kbf[illegible]	[illegible]	[illegible]	[illegible]88
wb[illegible]	[illegible]	[illegible]	[illegible]91
ba[illegible]	[illegible]	[illegible]	[illegible]02
rev[illegible]	[illegible]	[illegible]	[illegible]03
blo[illegible]	[illegible]	[illegible]	[illegible]36
div[illegible]	[illegible]	[illegible]	[illegible]57
multiplication	50	Java	1,263
collision	21	Java	1,741
avl	519	Java	2,985
tcas	200	Java	9,780
treemap	806	Java	13,556
cdaudio	2171	C	55,329
floppy	1137	C	100,006
grep	10068	C	100,126
diskperf	1104	C	103,505

Formulas: 391,250

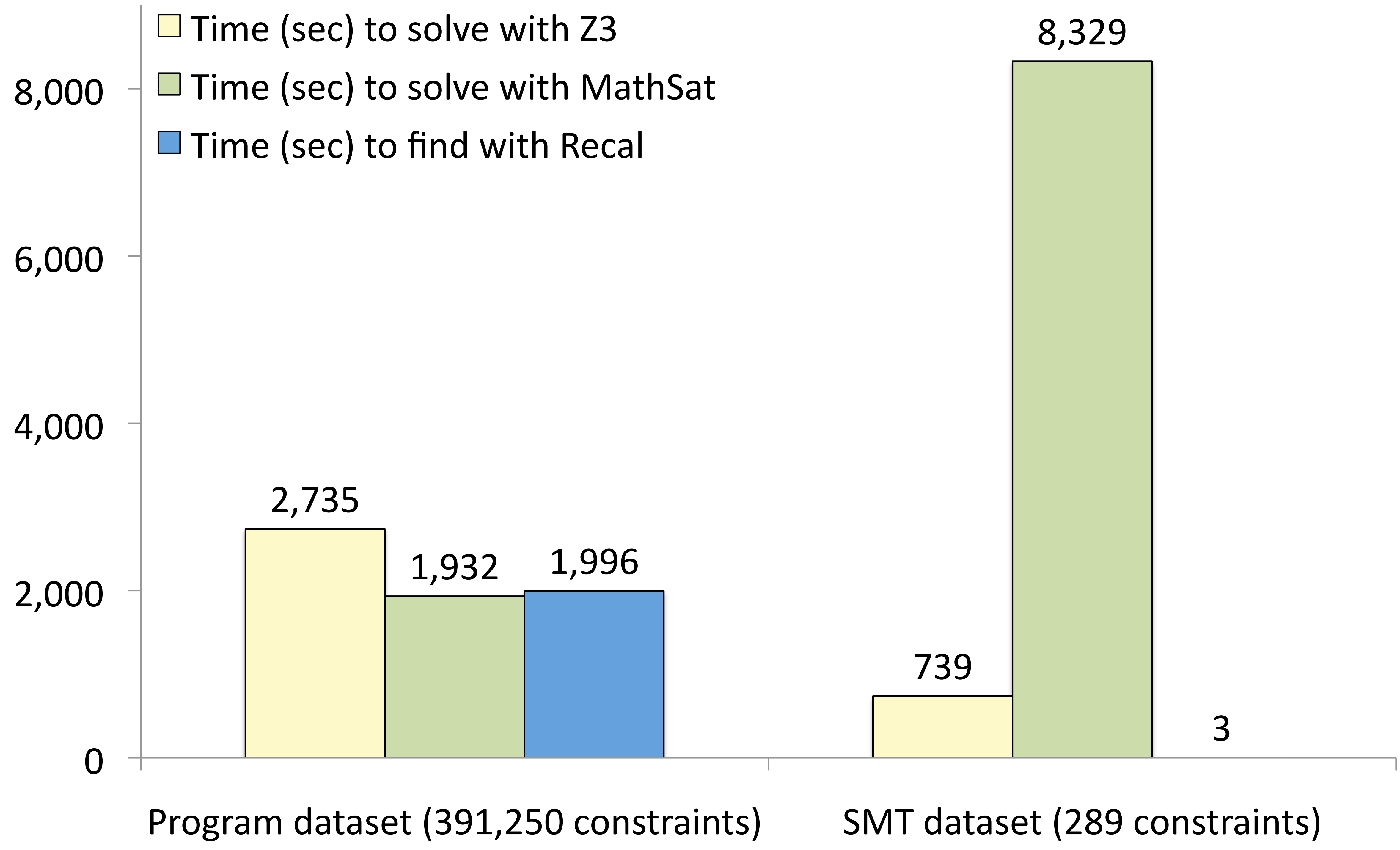
Queries to Solver: ~1,010

Evaluation

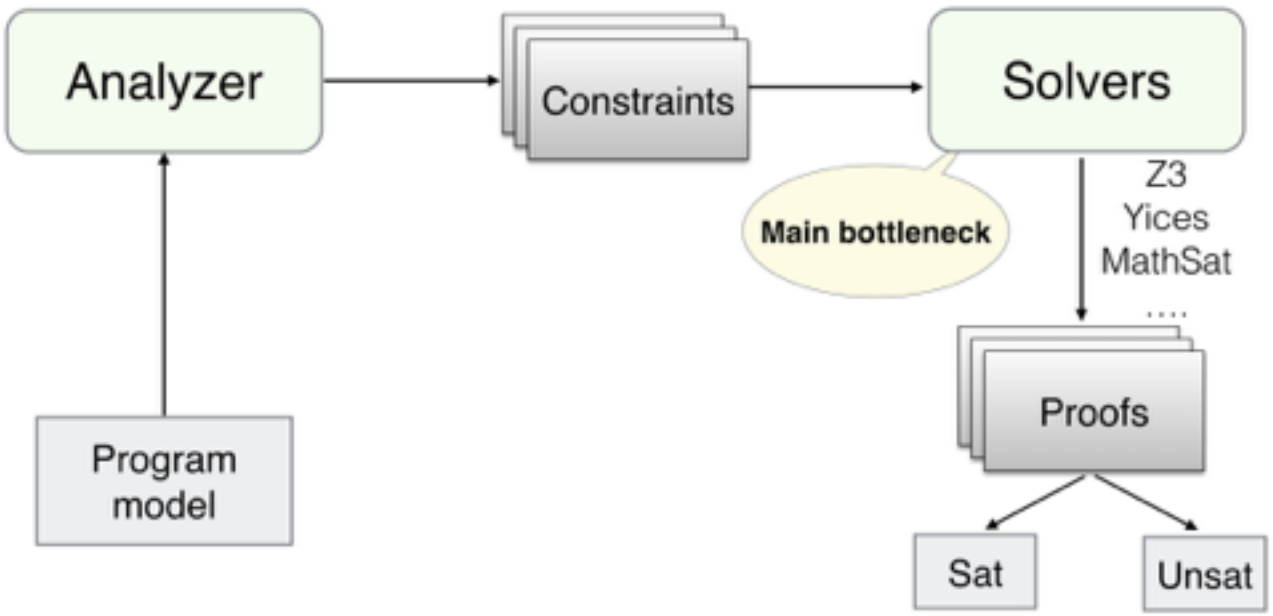
Effectiveness: Can Recal effectively identify reusable constraints?

Efficiency: Is Recal more efficient than SMT solvers?

Searching vs. Solving



Main Bottleneck



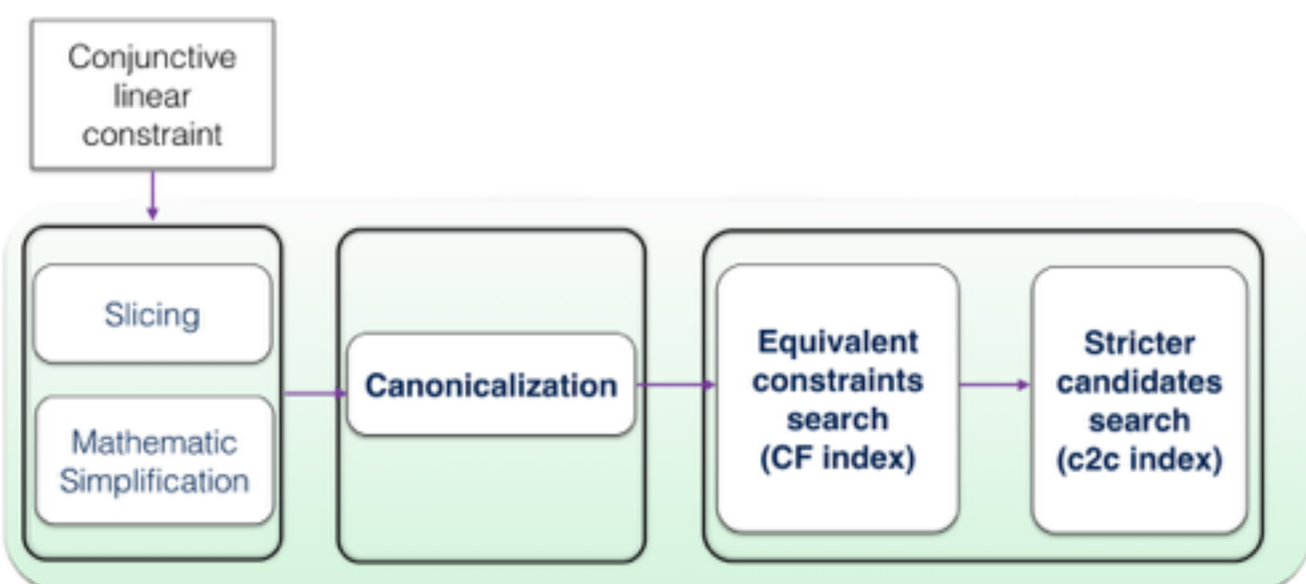
Solving time accounts for 92% of overall execution time on average. (KLEE. Cadar et al. osdi'08)

Our Solution

(1) Equivalence by reordering terms and clauses

(2) Stricter constraints by containment and implication

The Recal Framework



Searching vs. Solving