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

Analyzer

Constraints

Solvers

Program model

x + 2y < 0 \land 3x + 4y < 0

x + y < -1 \land -x - y < -3 \land 2x - y = 0

Proofs

Sat

Unsat

x = -1, y = -1

c1, c2
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

Constraints → Solvers

Z3 Yices MathSat

Proofs

Sat Unsat
Solving time is hard to predict

\[
\begin{align*}
-2a + 85b - 90c - 44d + 39e + 96f - 76g - 88h - 72i - 79j & \leq 66 \\
-100a - 19b + 60c - 96d - 42e - 30f + 82g + 75h + 73i - 41j & \leq 97 \\
-56a + 96b - 15c - 45d - 33e - 42f + 50g + 92h - 47i - 92j & \neq 64 \\
41a + 79b + 9c - 96d - 35e + 24f - 61g + 21h - 84i - 58j & \neq 41 \\
-67a - 65b - 46c - 49d + 71e + 100f & \leq 64 \\
-80a + 59b + 95c - 96d - 35e + 24f - 61g + 21h - 84i - 58j & \neq 41 \\
-67a - 65b - 46c - 49d + 71e + 100f & \leq 64 \\
-80a + 59b + 95c - 96d - 35e + 24f - 61g + 21h - 84i - 58j & \neq 41 \\
-67a - 65b - 46c - 49d + 71e + 100f & \leq 64 \\
-80a + 59b + 95c - 96d - 35e + 24f - 61g + 21h - 84i - 58j & \neq 41
\end{align*}
\]

\[
\begin{align*}
54a + 90b - 32c + 45d - 73e + 77f - 98g + 54h & - 45i - 67j \neq 4 \\
52a + 22b + 71c + 40d + 21e & - 75f - 75g + 13h + 33i - 18j \leq 12 \\
-17a - 100b + 56c - 94d + 79e + 19f & + 39g - 53h - 78i + 98j \leq 2 \\
-38a + 72b - 86c - 8d + 54e - 68f + 44g & + 47h - 34i + 72j \leq 81 \\
66a - 73b + 86c - 44d & - 66e + 22f + 96g - 14i - 91j \leq 37 \\
-51a - 64b - 19c + 80d & - 74e + 37f - 15g - 30j \neq 44 \\
71a - 44b + 3c - 4d & + 14e - 18f + 13g + 15i - 60j \neq 91 \\
-89a + 4b - 73c + 5d & + 39e + 4f + 85g - 2h - 16i + 95j \neq 37 \\
13a + 56b + 87c - 39d & - 60e - 36f + 35g + 74h - 3i + 5j \leq 70 \\
-37a + 51b - 30c + 24d & + 34e + 63f + 84g - 34h + 91i + 39j \neq 66
\end{align*}
\]

\[
\begin{align*}
&< 1 \text{ second} \\
&\ggg 10 \text{ minutes}
\end{align*}
\]
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

Reuse constraint proofs
Reuse Proofs

\[ x + y < 0 \land a + 2b \neq 9 \land x - y \neq 2 \land a - b > 10 \]

\[ x + y \geq 0 \land x - y = 2 \land a + 2b \neq 9 \land a - b > 10 \]
Reuse Proofs

\[ x + y < 0 \land a + 2b \neq 9 \land x - y \neq 2 \land a - b > 10 \]

\[ x + y \geq 0 \land x - y = 2 \land a + 2b \neq 9 \land a - b > 10 \]

Slicing

\[ x + y < 0 \land x - y \neq 2 \land a + 2b \neq 9 \land a - b > 10 \]

\[ x + y \geq 0 \land x - y = 2 \land a + 2b \neq 9 \land a - b > 10 \]
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

$\mathbf{C}_1$  
\[ x + 2y + 1 < 0 \land 3x + 4y - 1 < 0 \]
\[ 2y + x + 1 < 0 \land 4y + 3x - 1 < 0 \]

$\mathbf{C}_2$  
\[ 4a + 3b - 1 < 0 \land 2a + b + 1 < 0 \]
\[ 4V_1 + 3V_2 - 1 < 0 \land 2V_1 + V_2 + 1 < 0 \]
(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

\[ C_1 \equiv C_2 \iff \text{canonical}(C_1) = \text{canonical}(C_2) \]
## Equivalent Constraints Search

via Canonical Form

\[ C_1 \equiv C_2 \iff \text{canonical}(C_1) = \text{canonical}(C_2) \]

<table>
<thead>
<tr>
<th>( C_1 )</th>
<th>( x + 2y +1 \leq 0 \land 3x + 4y -1 \leq 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>≤</td>
</tr>
<tr>
<td>1</td>
<td>≤</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( C_2 )</th>
<th>( 4a + 3b -1 \leq 0 \land 2a + b +1 \leq 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>≤</td>
</tr>
<tr>
<td>1</td>
<td>≤</td>
</tr>
</tbody>
</table>

Canonical form

\[
\begin{align*}
4 & \leq \\
3 & \leq \\
1 & \leq \\
1 & \leq
\end{align*}
\]
The Canonicalization Algorithm

\[2a + b \leq 0 \land a + 2b \leq 0 \land a \neq 0 \land a + 3b \leq 0 \land a - 1 \leq 0\]
The Canonicalization Algorithm

sort rows by comparison and constant terms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>≤</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>≤</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>≤</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>≠</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>≤</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>≤</td>
</tr>
</tbody>
</table>
sort rows by comparison and constant terms

\[
\begin{array}{cccc}
2 & 1 & 0 & \leq \\
1 & 2 & 0 & \leq \\
1 & 0 & 0 & \neq \\
1 & 3 & 0 & \leq \\
1 & 0 & -1 & \leq \\
\end{array}
\]
The Canonicalization Algorithm

sort rows by comparison and constant terms

sort rows and columns by biggest values
The Canonicalization Algorithm

- Sort rows by comparison and constant terms
- Sort rows and columns by biggest values

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>≠</td>
</tr>
</tbody>
</table>

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
# 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.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Initial:
- 1-D locked:
- 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 table shows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>v</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>v</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>v</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>v</td>
</tr>
</tbody>
</table>

Initial, 1-D locked, 2-D locked.
The Canonicalization Algorithm

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Polynomial

93% of constraints converge up to the polynomial steps.

Exponential
(2) Stricter constraints by containment and implication

What is a stricter constraint?

Search for stricter constraints?
Stricter Constraints

\[ C_1 \]
\[ 3X < 0 \land X + Y < 10 \]

\[ \text{Sat} \]
\[ 3X < 0 \land X + Y < 10 \land 2X - Y = 0 \]
\[ 3X < -1 \land X + Y < 10 \]
Stricter Constraints

C₁

\[
3X < 0 \land X + Y < 10
\]

Sat

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

UnSat

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

C₂

\[
X + Y < -1 \land -X - Y < -3
\]

UnSat

\[
X + Y < 0 \land -X - Y < -3
\]
Stricter Constraints Search

Clause-to-constraint index
Stricter Constraints Search

Clause-to-constraint index

\[ 3X < 0 \land X + Y < 10 \]

\[ 3X < -1 \land X + Y < 10 \quad (\text{sat}) \]

\[ 3X < -1 \land X - 2Y < 0 \quad (\text{sat}) \]

\[ -2X < -1 \land X + Y < 10 \quad (\text{sat}) \]

\[ 3X \rightarrow \{C_1, C_2\} \]

\[ X + Y \rightarrow \{C_1, C_3\} \]

intersection

\[ \{C_1, C_2\} \cap \{C_1, C_3\} = \{C_1\} \]
The Recal Framework
The Recal Framework

Conjunctive linear constraint

Slicing

Simplification

Canonicalization

Equivalent constraints search (CF index)

Stricter candidates search (c2c index)
Evaluation

Effectiveness: Can Recal *effectively* identify reusable constraints?

Efficiency: Is Recal more *efficient* than SMT solvers?
A large set of real-world constraints

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

JBSE [Braione, et al., FSE’13]
CREST [Burnim, et al., EECS’08]

# Constraints
391,250
Intra-program Reuse Rates

- **Green**: 1% (85% confidence interval)
- **Recal -**: 47% (90% confidence interval)
- **Recal +**: 87% (97% confidence interval)
Inter-program Reuse Rates

Green
- 14%
- 70%
- 5%

Recal +
- 100%
- 59%
- 35%
- 14%
## High Reuse Rates

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Language</th>
<th>#Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>old-tax</td>
<td>78</td>
<td>Java</td>
<td>27</td>
</tr>
<tr>
<td>new-tax</td>
<td>78</td>
<td>Java</td>
<td>35</td>
</tr>
<tr>
<td>afs</td>
<td>75</td>
<td>Java</td>
<td>48</td>
</tr>
<tr>
<td>dijkstra</td>
<td>142</td>
<td>Java</td>
<td>85</td>
</tr>
<tr>
<td>doubly-linked-list</td>
<td>806</td>
<td>Java</td>
<td>114</td>
</tr>
<tr>
<td>swapwords</td>
<td>30</td>
<td>Java</td>
<td>173</td>
</tr>
<tr>
<td>kb</td>
<td></td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>wb</td>
<td></td>
<td></td>
<td>291</td>
</tr>
<tr>
<td>ba</td>
<td></td>
<td></td>
<td>292</td>
</tr>
<tr>
<td>rev</td>
<td></td>
<td></td>
<td>293</td>
</tr>
<tr>
<td>blk</td>
<td></td>
<td></td>
<td>386</td>
</tr>
<tr>
<td>div</td>
<td></td>
<td></td>
<td>757</td>
</tr>
<tr>
<td>multiplication</td>
<td>50</td>
<td>Java</td>
<td>1,263</td>
</tr>
<tr>
<td>collision</td>
<td>21</td>
<td>Java</td>
<td>1,741</td>
</tr>
<tr>
<td>avl</td>
<td>519</td>
<td>Java</td>
<td>2,985</td>
</tr>
<tr>
<td>tcas</td>
<td>200</td>
<td>Java</td>
<td>9,780</td>
</tr>
<tr>
<td>treemap</td>
<td>806</td>
<td>Java</td>
<td>13,556</td>
</tr>
<tr>
<td>cdaudio</td>
<td>2171</td>
<td>C</td>
<td>55,329</td>
</tr>
<tr>
<td>floppy</td>
<td>1137</td>
<td>C</td>
<td>100,006</td>
</tr>
<tr>
<td>grep</td>
<td>10068</td>
<td>C</td>
<td>100,126</td>
</tr>
<tr>
<td>diskperf</td>
<td>1104</td>
<td>C</td>
<td>103,505</td>
</tr>
</tbody>
</table>

# 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

Program dataset (391,250 constraints):
- Time (sec) to solve with Z3: 2,735
- Time (sec) to solve with MathSat: 1,932
- Time (sec) to find with Recal: 1,996

SMT dataset (289 constraints):
- Time (sec) to solve with Z3: 8,329
- Time (sec) to solve with MathSat: 739
- Time (sec) to find with Recal: 3
Main Bottleneck

Analyzer -> Constraints -> Solvers

Program model

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

Main bottleneck

Sat
Unsat

Our Solution

(1) Equivalence by reordering terms and clauses

(2) Stricter constraints by containment and implication

The Recal Framework

Conjunctive linear constraint

Slicing
Canonicalization
Equivalent constraints search (CF index)
Stricter candidates search (c2c index)

Searching vs. Solving

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Time (sec) to solve with Z3</th>
<th>Time (sec) to solve with MathSat</th>
<th>Time (sec) to find with Recal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program dataset</td>
<td>2,735</td>
<td>1,932</td>
<td>1,996</td>
</tr>
<tr>
<td>SMT dataset</td>
<td>8,329</td>
<td>739</td>
<td>3</td>
</tr>
</tbody>
</table>

Program dataset (391,250 constraints)  SMT dataset (289 constraints)