Enhancing Reuse of Constraint Solutions to Improve Symbolic Execution

Xiangyang Jia (Wuhan University)
Carlo Ghezzi (Politecnico di Milano)
Shi Ying (Wuhan University)
Outline

❖ Motivation
❖ Logical Basis of our Approach
❖ GreenTrie Framework
  ❖ Constraint Reduction
  ❖ Constraint Storing
  ❖ Constraint Querying
❖ Evaluation
❖ Conclusion and Future Work
Motivation

❖ Symbolic Execution (SE)
  ❖ A well-known program analysis technique, mainly used for test-case generation and bug finding.

❖ Constraint Solving
  ❖ The most time-consuming work in SE
  ❖ Optimization approaches:
    ❖ Irrelevent constraint elimination
    ❖ Caching and reuse
Motivation

Aggregated data over 73 applications

- Base
- Irrelevant Constraint Elimination
- Caching
- Irrelevant Constraint Elimination + Caching

Time (s)

Executed instructions (normalized)

[From Shauvik Roy Choudhary’s Slides]
Motivation

❖ Reuse of Constraint Solutions

Equivalence based approach (Green)

- $x > 0$ is equivalent to $y > 0$
- $x + 1 > 0 \land x \leq 1$ is equivalent to $y < 2 \land y \geq 0$ (if $x$, $y$ are integers)
Motivation

❖ Reuse of Constraint Solutions

If $A^B^C$ is satisfiable, then $A^B$ is satisfiable.
If $A^B^C$ is unsatisfiable, then $A^B^C^D$ is unsatisfiable.
Motivation

❖ Reuse of Constraint Solutions

- Equivalence based approach (Green)
- Subset/superset based approach (KLEE)

If \( x > 0 \) is satisfiable, can we prove \( x > -1 \) satisfiable?
If \( x < 0 \land x > 1 \) is unsatisfiable, can we prove \( x < -1 \land x > 2 \) unsatisfiable?
Motivation

❖ Reuse of Constraint Solutions

- Equivalence based approach (Green)
- Subset/superset based approach (KLEE)
- Implication based approach (Our approach)

If $x > 0$ is satisfiable, can we prove $x > -1$ satisfiable?
If $x < 0 \land x > 1$ is unsatisfiable, can we prove $x < -1 \land x > 2$ unsatisfiable?
Logical Basis of our Approach

Implication and Satisfiability

Providing $C_1 \rightarrow C_2$

- if $C_1$ is satisfiable, $C_2$ is satisfiable
- if $C_2$ is unsatisfiable, $C_1$ is unsatisfiable

It looks easy to apply it to constraint reuse!
However, there is a problem:
Implication checking with SAT/SMT solver is even more expensive than only solving the single constraint itself.
Logical Basis of our Approach

• The subset/superset (KLEE)
  • \( \{c_1,c_2\} \subseteq \{c_1,c_2,c_3\} \) means \( c_1 \land c_2 \land c_3 \rightarrow c_1 \land c_2 \)

• Logical subset/superset
  • Given two constraint sets \( X,Y \), if \( \forall a \in X \exists b \in Y \) (\( b \rightarrow a \)), then \( X \) is a logical subset of \( Y \), and \( Y \) is a logical superset of \( X \)
  • E.g.: \( X = \{m \neq 0, m > -1, m < 2\} \), \( Y = \{m > 1, m < 2\} \)
  • It is easy to prove that \( (m > 1 \land m < 2) \rightarrow (m \neq 0 \land m > -1 \land m < 2) \)

the subset/superset is a specific case of logical subset/superset

Logical subset/superset checks more implication cases!
❖ the two sets might have totally different atomic constraints
❖ the length of logical superset may be shorter than its subset
Logical Basis of our Approach

- Implication checking rules for atomic constraints

\[
\begin{align*}
(R1) & \quad C \rightarrow C \\
(R2) & \quad P + n = 0 \rightarrow P + n' \neq 0 \\
(R3) & \quad n \geq n' \quad \rightarrow \quad P + n = 0 \rightarrow P + n' \leq 0 \\
(R4) & \quad n \leq n' \quad \rightarrow \quad P + n = 0 \rightarrow P + n' \geq 0 \\
(R5) & \quad n > n' \quad \rightarrow \quad P + n \leq 0 \rightarrow P + n' \neq 0 \\
(R6) & \quad n \leq n' \quad \rightarrow \quad P + n \leq 0 \rightarrow P + n' \leq 0 \\
(R7) & \quad n < n' \quad \rightarrow \quad P + n \geq 0 \rightarrow P + n' \neq 0 \\
(R8) & \quad n \geq n' \quad \rightarrow \quad P + n \geq 0 \rightarrow P + n' \geq 0
\end{align*}
\]

P: non-constant prefix, n: constant number
E.g. \(x+y+3\geq 0\) has a non-constant prefix \(x+y\) and a constant number 3
GreenTrie Framework

**Architecture of GreenTrie**

- Symbolic Executor

**GreenTrie**

- Known: $C_1 \land C_2 \land \ldots \land C_n$
- Unknown: $C_0$

- Slicing
- Canonization

- Sliced: $C_0 \land C_1 \land \ldots \land C_{n-1}$
- Canonized: $C'_1 \land C_0 \land \ldots \land C'_{n-1}$
- Reduced: $C'_1 \land C'_m$

- Reusing

- Translating

- Storing

**L-Trie**

- Satisfiable Constraint Store (SCS)
- Unsatisfiable Constraint Store (UCS)

- Reduction
- Querying

**Two separated stores for SAT and UNSAT constraints**

Constraint Solver: CVC3, Z3, Yices, Choco...
GreenTrie Framework

• Architecture of GreenTrie

Known: C_1 \land C_2 \land \ldots \land C_n

Unknown: C_0

Slicing

Canonization

Sliced: C_0 \land C_1 \land \ldots \land C_{n-1}

Canonicalized: C'_1 \land C_0 \land C'_n \land \ldots \land C'_{n-1}

Reduced: C'_1 \land C'_m

Get(C'_1 \land C'_m)

Result

If result is null

Put(C'_1 \land C'_m, SAT)

Translating

Reusing

Satisfiable Constraint Store (SCS)

Constraint Trie

Logical Index

Satisfiable

Unsatisfiable

Constraint Store (UCS)

Constraint Trie

Logical Index

Querying

Reduction

Storing

Constraint Solver: CVC3, Z3, Yices, Choco...
GreenTrie Framework

- **Architecture of GreenTrie**
  
  [Diagram showing the architecture of GreenTrie]

  An exceptional point set \( E = \{6, 0, 4\} \). After this, we go through the following steps:

  1. We discard all exceptional points that are outside the overlapping interval; in the example, the value of \( E \) becomes \( \{0, 4\} \).
  2. If one endpoint of the overlapping interval \( A \) (or \( B \)) belongs to \( E \), we (repeatedly) change its value and eliminate \( A \) (or \( B \)) from \( E \) at the same time. In the example after this step the interval becomes \( [3, 3] \) and the new value of \( E \) is \( \{0\} \).
  3. If the overlapping interval is empty then the constraint is unsatisfiable and we report a conflict; otherwise we translate \( [A, B] \) and \( E \) into a constraint in normal form. In the example, the final result of our reduction is \( x + y + 3 \leq 0 \).

  5. **CONSTRAINT STORING**

     L-Trie provides a different storage scheme that replaces the Redis store of Green:

     - Unlike Redis, which stores the strings representing constraints and solutions as key-value pairs, L-Trie splits constraints into sub-constraint sets, and stores them into tries, in order to support logical subset and superset queries based on Theorem 1.
     - L-Trie stores unsatisfiable and satisfiable constraints into separate areas: the **Unsatisfiable Constraint Store** (UCS) and the **Satisfiable Constraint Store** (SCS) respectively. The two areas are organized differently to efficiently support logical subset querying and logical superset querying, which pose different requirements.
     - L-Trie maintains a logical index for each of the two tries, to support efficient check of the implication relations. The logical index is represented as an implication partial order graph (IPOG), whose nodes contain references to nodes in the trie.

**Constraint Trie**

The constraint trie is designed to store a sub-constraint set of solved constraints. The sub-constraint set is sorted in lexicographic order based on string comparison, to guarantee that sub-constraints with same non-constant prefix are kept close to each other. The labels of the constraint trie record the sub-constraints. The leaf nodes indicate the end of the constraint and are annotated with the solution (the solution is null for the leaves of the UCS trie). As shown in Fig. 2, the leaf node \( C2 \) corresponds to a constraint \( v_0 + 5 > 0 \) and \( v_0 + v_1 < 0 \), which has a solution \( \{v_0: 0, v_1: 1\} \), and its sub-constraints \( v_0 + 5 > 0 \) and \( v_0 + v_1 < 0 \), are annotated as edge labels in the path.

**Implication Partial Order Graph (IPOG)**

IPOG is a graph that contains all the atomic sub-constraints appearing in its associated constraint trie, and arranges them as a graph based on the partial order defined by the implication relation. With this graph, given a constraint \( C \), we can query the sub-constraints which imply \( C \), as well as the sub-constraints which \( C \) implies, as we will see later. This is useful to improve the efficiency of implication checking in logical subset and superset querying. IPOG nodes are labeled by a sub-constraint and have references to all trie nodes whose input edge is labeled with exactly this sub-constraint. Through these references, it is possible to trace all the occurrences of a given sub-constraint.

**Storing the constraints.**

Everytime a constraint is solved (or it is proved to be unsatisfiable), SCS (respectively UCS) removes redundant sub-constraints for better matching.
GreenTrie Framework

• Architecture of GreenTrie

```
\begin{align*}
\text{Known: } & C_1 \land C_2 \ldots \land C_n \\
\text{Unknown: } & C_0 \\
\text{sliced: } & C_0 \land C_1 \land C_{n-1} \\
\text{canonized: } & C'_1 \land C_0 \land C'_n \land C'_{n-1} \\
\text{reduced: } & C'_1 \land C'_m \\
\text{get(C'} & C'_m) \\
\text{result} & \\
\text{if result is null} & \\
\text{put(C'} & C'_m, \text{SAT}) \\
\text{Translating} & \\
\text{Querying} & \\
\text{Reduction} & \\
\text{Constraint Trie} & \\
\text{Unsatisfiable Constraint Store (UCS)} & \\
\text{Logical Index} & \\
\text{Constraint Trie} & \\
\text{Logical Index} & \\
\text{Constraint Trie} & \\
\text{Query reusable constraints through logical subset/superset checking} & \\
\end{align*}
```

Constraint Solver: CVC3, Z3, Yices, Choco...
GreenTrie Framework

• Architecture of GreenTrie

![Diagram of GreenTrie framework]

If no reusable constraint is found, solve it, and then puts the solving result into stores.
Constraint Reduction

- Constraint Reduction
  - target: remove redundant sub-constraints
  - idea: interval computation-based constraint reduction

Example

\[ x + y + 3 \geq 0 \land x + y + 5 \geq 0 \land x + y - 4 \leq 0 \land x + y \neq 0 \land x + y + 6 \neq 0 \land x + y - 4 \neq 0 \]

compute: \([-3, \infty) \cap [-5, \infty) \cap (-\infty, 4] - \{0, -6, 4\} = [-3, 4) - \{0\}\]

reduced: \(x + y + 3 \geq 0 \land x + y - 4 < 0 \land x + y \neq 0\)
Constraint Storing

C3 represents a constraint $V_0 + 5 \geq 0 \land V_1 + (-1) \leq 0$, which has a solution \{\(v_0:0, v_1:-5\}\}.
Constraint Storing

- $v_0 + 5 \geq 0$ is implied by $v_0 + (-3) = 0$ and $v_0 + (-4) = 0$
- $v_0 + 5 \geq 0$ has one occurrence in the trie, therefore it has a reference to the successive trie node.
Example

Constraint: $v_0 \geq 0$

$\text{IS}_{v_0 \geq 0}: \{v_0 + 5 \geq 0\}$

$\text{RIS}_{v_0 \geq 0}: \{v_0 + (-3) = 0, \ v_0 + (-4) = 0\}$
Constraint Querying

Logical Superset Checking Algorithm

Find a path in trie, so that every sub-constraint in target constraint is implied by at least one constraint on this path

Example

Target: $v_0 \neq 0 \land v_0 + (-1) \neq 0 \land v_1 + (-2) \leq 0$

RIS$_{v_1 + (-2) \leq 0} \{v_1 + (-1) \leq 0\}$

So, we got two candidate paths to check!

Start from these two nodes!
Constraint Querying

Logical Superset Checking Algorithm

Example
Target: \(v_0 \neq 0 \land v_0 + (-1) \neq 0 \land v_1 + (-2) \leq 0\)

\(\text{RIS}_{v_0 \neq 1}: \{v_0 + (-3) = 0, v_0 + (-4) = 0\}\)

\(v_0 + 5 \geq 0\) is not in the RIS, the trie root is reached, so this path doesn’t match!
**Constraint Querying**

- **Logical Superset Checking Algorithm**

  **Example**

  Target: $v_0 
eq 0 \land v_0 + (-1) \neq 0 \land v_1 + (-2) \leq 0 \
  \text{RIS}_{v_0 \neq 1} : \{v_0 + (-3) = 0, v_0 + (-4) = 0\}$

  $v_0 + (-3) \geq 0$ is in the RIS, go on to check next sub-constraint of target!
**Logical Superset Checking Algorithm**

**Example**

Target: \( v_0 \neq 0 \land v_0 + (-1) \neq 0 \land v_1 + (-2) \leq 0 \)

\( RIS_{v_0 \neq 0} : \{v_0 + (-3) = 0, v_0 + (-4) = 0\} \)

\( v_0 + (-3) \geq 0 \) is also in the RIS of \( v_0 \neq 0 \), now, every sub-constraint in target is implied by one constraint on this path. 
C4 is the reusable constraint!
Constraint Querying

Logical Subset Checking Algorithm

Target: \( v_0 + (-1) \geq 0 \) ^ \( v_0 + 3 \neq 0 \) ^ \( v_0 + 4 \leq 0 \)

Union of ISs of the sub-constraints: \( \{v_0 \geq 0\} \cup \{\} \cup \{v_0 + 2 \leq 0, v_0 + 1 \leq 0\} \)

\( IS_{union} = \{v_0 \geq 0, v_0 + 2 \leq 0, v_0 + 1 \leq 0\} \)

We will find a trie path, so that all its sub-constraints on the path exists in \( IS_{union} \)
Constraint Querying

❖ Logical Subset Checking Algorithm

Target: \( v_0 + (-1) \geq 0 \land v_0 + 3 \neq 0 \land v_0 + 4 \leq 0 \)

\( IS_{\text{union}} = \{ v_0 \geq 0, \, v_0 + 2 \leq 0, \, v_0 + 1 \leq 0 \} \)
Constraint Querying

Logical Subset Checking Algorithm

Target: $v_0 + (-1) \geq 0 \land v_0 + 3 \neq 0 \land v_0 + 4 \leq 0$

$\text{IS}_{\text{union}} = \{ v_0 \geq 0, v_0 + 2 \leq 0, v_0 + 1 \leq 0 \}$

We found two paths, so the target constraint is unsatisfiable.
Evaluation

❖ Research Question
  ❖ Does GreenTrie achieve better reuse and save more time than other approaches (Green, KLEE)?

❖ Benchmarks
  ❖ 6 programs from Green (Willem Visser’s FSE’12 paper)
  ❖ 1 program from Guowei Yang’s ISSTA 2012 paper.

❖ Experiment scenarios
  ❖ (1) reuse in a single run of the program
  ❖ (2) reuse across runs of different versions of the same program
  ❖ (3) reuse across different programs
Evaluation

❖ Experiment setup
  ❖ PC with a 2.5GHz Intel processor with 4 cores and 4Gb of memory
  ❖ We implemented GreenTrie by extending Green
  ❖ We implemented KLEE’s subset/superset checking approach, and also integrated it into Green as an extension.
  ❖ Symbolic executor: Symbolic Pathfinder (SPF)
  ❖ Constraint Solver: Z3
Evaluation

Reuse in a Single Run

Table 1: Experimental results of reuse in single run

| Program   | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $R'$      | $R''$     | $t_0$(ms) | $t_1$(ms) | $t_2$(ms) | $t_3$(ms) | $T'$     | $T''$   |
|-----------|-------|-------|-------|-------|-----------|-----------|-----------|-----------|-----------|-----------|----------|---------|
| Trityp    | 32    | 28    | 28    | 28    | 0.00%     | 0.00%     | 1040      | 915       | 922       | 995       | -8.74%   | -7.92%  |
| Euclid    | 642   | 552   | 464   | 464   | 15.94%    | 0.00%     | 5105      | 6503      | 7274      | 6311      | 2.95%    | 13.24%  |
| TCAS      | 680   | 41    | 20    | 14    | 65.85%    | 30.00%    | 12742     | 3356      | 2182      | 2165      | 35.49%   | 0.78%   |
| TreeMap1  | 24    | 24    | 24    | 24    | 0.00%     | 0.00%     | 871       | 942       | 947       | 882       | 6.37%    | 6.86%   |
| TreeMap2  | 148   | 148   | 140   | 140   | 5.41%     | 0.00%     | 2918      | 2542      | 2851      | 2606      | -2.52%   | 8.59%   |
| TreeMap3  | 1080  | 956   | 833   | 806   | 15.69%    | 3.24%     | 21849     | 10729     | 11809     | 9871      | 8.00%    | 16.41%  |
| BinTree1  | 84    | 41    | 25    | 25    | 39.02%    | 0.00%     | 1476      | 1103      | 1092      | 1027      | 6.89%    | 5.95%   |
| BinTree2  | 472   | 238   | 133   | 118   | 50.42%    | 11.28%    | 4322      | 3648      | 3156      | 2872      | 21.27%   | 9.00%   |
| BinTree3  | 3252  | 1654  | 939   | 873   | 47.22%    | 7.03%     | 36581     | 17197     | 14764     | 12041     | 29.98%   | 18.44%  |
| BinomialHeap1 | 448 | 32    | 23    | 19    | 40.63%    | 17.39%    | 3637      | 2137      | 2046      | 2017      | 5.62%    | 1.42%   |
| BinomialHeap2 | 3184 | 190   | 85    | 68    | 64.21%    | 20.00%    | 27165     | 7653      | 6442      | 6071      | 20.67%   | 5.76%   |
| BinomialHeap3 | 23320 | 988   | 337   | 288   | 70.85%    | 14.54%    | 249224    | 28549     | 31892     | 21392     | 25.07%   | 32.92%  |
| MerArbiter | 60648 | 21    | 15    | 13    | 38.10%    | 13.33%    | >10min    | 304726    | 290854    | 272813    | 10.47%   | 6.20%   |

$n_i$: the number of invocations to solver

$t_i$: running time for symbolic execution

$i=0$: SE without reuse  
$i=1$: SE with Green  
$i=2$: SE with KLEE’s approach  
$i=3$: SE with GreenTrie

Reuse improvement ratio: $R' = \frac{n_1 - n_3}{n_1}$  
$R'' = \frac{n_2 - n_3}{n_2}$

Time improvement ratio: $T' = \frac{t_1 - t_3}{t_1}$  
$T'' = \frac{t_2 - t_3}{t_2}$
Evaluation

❖ Reuse in a Single Run

Table 1: Experimental results of reuse in single run

| Program     | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $R'$  | $R''$ | $t_0$ (ms) | $t_1$ (ms) | $t_2$ (ms) | $t_3$ (ms) | $T'$  | $T''$ |
|-------------|-------|-------|-------|-------|-------|-------|------------|------------|------------|------------|-------|-------|
| Trityp      | 32    | 28    | 28    | 28    | 0.00% | 0.00% | 1040       | 915        | 922        | 995        | -8.74%| -7.92%|
| Euclid      | 642   | 552   | 464   | 464   | 15.94%| 0.00% | 5105       | 6503       | 7274       | 6311       | 2.95% | 13.24%|
| TCAS        | 680   | 41    | 20    | 14    | 65.85%| 30.00%| 12742      | 3356       | 2182       | 2165       | 35.49%| 0.78% |
| TreeMap1     | 24    | 24    | 24    | 24    | 0.00% | 0.00% | 871        | 942        | 947        | 882        | 6.37% | 6.86% |
| TreeMap2     | 148   | 148   | 140   | 140   | 5.41% | 0.00% | 2918       | 2542       | 2851       | 2606       | -2.52%| 8.59% |
| TreeMap3     | 1080  | 956   | 833   | 806   | 15.69%| 3.24% | 21849      | 10729      | 11809      | 9871       | 8.00% | 16.41%|
| BinTree1     | 84    | 41    | 25    | 25    | 39.02%| 0.00% | 1476       | 1103       | 1092       | 1027       | 6.89% | 5.95% |
| BinTree2     | 472   | 238   | 133   | 118   | 50.42%| 11.28%| 4322       | 3648       | 3156       | 2872       | 21.27%| 9.00% |
| BinTree3     | 3252  | 1654  | 939   | 873   | 47.22%| 7.03% | 36581      | 17197      | 14764      | 12041      | 29.98%| 18.44%|
| BinomialHeap1| 448   | 32    | 23    | 19    | 40.63%| 17.39%| 3637       | 2137       | 2046       | 2017       | 5.62% | 1.42% |
| BinomialHeap2| 3184  | 190   | 85    | 68    | 64.21%| 20.00%| 27165      | 7653       | 6442       | 6071       | 20.67%| 5.76% |
| BinomialHeap3| 23320 | 988   | 337   | 288   | 70.85%| 14.54%| 249224     | 28549      | 31892      | 21392      | 25.07%| 32.92%|
| MerArbiter   | 60648 | 21    | 15    | 13    | 38.10%| 13.33%| >10min     | 304726     | 290854     | 272813     | 10.47%| 6.20% |
| total/average| 94014 | 4913  | 3066  | 2880  | 41.38%| 6.07% | /          | 390000     | 374012     | 341063     | 12.55%| 9.35% |

GreenTrie gets better reuse ratio and saves more time when the scale of execution increases.
## Evaluation

### Reuse across Runs

#### Table 2: Experimental results of reuse across runs (program Euclid)

| Changes | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $R'$ | $R''$ | $t_1$ (ms) | $t_2$ (ms) | $t_3$ (ms) | $T'$ | $T''$ |
|---------|-------|-------|-------|-------|------|------|----------|----------|----------|------|------|
| ADD#1   | 492   | 432   | 5     | 3     | 99.54% | 60.00% | 3896     | 1375     | 1329     | 65.89% | 3.35% |
| ADD#2   | 438   | 331   | 216   | 216   | 34.74% | 0.00%  | 2830     | 3275     | 2284     | 19.29% | 30.26% |
| ADD#3   | 220   | 170   | 32    | 2     | 98.82% | 93.75% | 1382     | 972      | 552      | 60.06% | 43.21% |
| DEL#1   | 438   | 322   | 156   | 126   | 60.87% | 19.23% | 3428     | 2670     | 2171     | 36.67% | 18.69% |
| DEL#2   | 492   | 426   | 350   | 134   | 68.54% | 61.71% | 3777     | 4483     | 2046     | 45.83% | 54.36% |
| DEL#3   | 642   | 552   | 112   | 111   | 79.89% | 0.89%  | 4649     | 2560     | 2049     | 55.93% | 19.96% |
| MOD#1   | 642   | 552   | 464   | 463   | 16.12% | 0.22%  | 4851     | 6899     | 4400     | 9.30%  | 36.22% |
| MOD#2   | 642   | 552   | 464   | 462   | 16.30% | 0.43%  | 4765     | 7094     | 4351     | 8.69%  | 38.67% |
| MOD#3   | 642   | 551   | 442   | 433   | 21.42% | 2.04%  | 4505     | 7481     | 4240     | 5.88%  | 43.32% |

#### Table 4: Experimental results of reuse across runs (program BinTree)

| Changes | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $R'$ | $R''$ | $t_1$ (ms) | $t_2$ (ms) | $t_3$ (ms) | $T'$ | $T''$ |
|---------|-------|-------|-------|-------|------|------|----------|----------|----------|------|------|
| ADD#1   | 5930  | 1689  | 803   | 746   | 55.83% | 7.10%  | 17978    | 20355    | 11889    | 33.87% | 41.59% |
| ADD#2   | 13358 | 3938  | 2618  | 2556  | 35.09% | 2.37%  | 35382    | 105190   | 32465    | 8.24%  | 69.14% |
| ADD#3   | 15602 | 540   | 0     | 0     | 100.00% | 0/0    | 18106    | 61568    | 17180    | 5.11%  | 72.10% |
| DEL#1   | 13358 | 3149  | 2216  | 2185  | 30.61% | 1.40%  | 32134    | 126488   | 31002    | 3.52%  | 75.49% |
| DEL#2   | 5930  | 1154  | 599   | 0     | 100.00% | 100.00% | 13565    | 44789    | 10932    | 19.41% | 75.59% |
| DEL#3   | 3252  | 1682  | 0     | 0     | 100.00% | 0/0    | 12945    | 11482    | 4505     | 65.20% | 60.76% |
| MOD#1   | 3252  | 1682  | 1080  | 1002  | 40.43% | 7.22%  | 14553    | 16297    | 10628    | 26.97% | 34.79% |
| MOD#2   | 3252  | 1680  | 716   | 632   | 62.38% | 11.73% | 14147    | 13784    | 7953     | 43.78% | 42.30% |
| MOD#3   | 8310  | 2377  | 1068  | 964   | 59.44% | 9.74%  | 22772    | 32889    | 14593    | 35.92% | 55.63% |

| total/average | 72244 | 17891 | 9100  | 8085  | 54.81% | 11.15% | 181582   | 432860   | 141147   | 22.27% | 67.39% |

GreenTrie gets better reuse ratio and saves more time than both Green and KLEE’s approach.
Evaluation

- Reuse across Runs

Table 4: Experimental results of reuse across runs (program BinTree)

| Changes    | \(n_0\) | \(n_1\) | \(n_2\) | \(n_3\) | \(R'\) | \(R''\) | \(t_1\) (ms) | \(t_2\) (ms) | \(t_3\) (ms) | \(T'\) | \(T''\) |
|------------|---------|---------|---------|---------|--------|--------|-------------|-------------|-------------|--------|--------|
| ADD#1      | 5930    | 1689    | 803     | 746     | 55.83% | 7.10%  | 17978       | 20355       | 11889       | 33.87% | 41.59% |
| ADD#2      | 13358   | 3938    | 2618    | 2556    | 35.09% | 2.37%  | 35382       | 105190      | 32465       | 8.24%  | 69.14% |
| ADD#3      | 15602   | 540     | 0       | 0       | 100.00% | 0/0    | 18106       | 61586       | 17180       | 5.11%  | 72.10% |
| DEL#1      | 13358   | 3149    | 2216    | 2185    | 30.61% | 1.40%  | 32134       | 126488      | 31002       | 3.52%  | 75.49% |
| DEL#2      | 5930    | 1154    | 599     | 0       | 100.00% | 100.00%| 13565       | 44789       | 10932       | 19.41% | 75.59% |
| DEL#3      | 3252    | 1682    | 0       | 0       | 100.00% | 0/0    | 12945       | 11482       | 4505        | 65.20% | 60.76% |
| MOD#1      | 3252    | 1682    | 1080    | 1002    | 40.43% | 7.22%  | 14553       | 16297       | 10628       | 26.97% | 34.79% |
| MOD#2      | 3252    | 1680    | 716     | 632     | 62.38% | 11.73% | 14147       | 13784       | 7953        | 43.78% | 42.30% |
| MOD#3      | 8310    | 2377    | 1068    | 964     | 59.44% | 9.74%  | 22772       | 32889       | 14593       | 35.92% | 55.63% |
| total/average | 72244  | 17891   | 9100    | 8085    | 54.81% | 11.15% | 181582      | 432860      | 141147      | 22.27% | 67.39% |

GreenTrie gains better scalability than KLEE’s approach

3421 constraints in store

Running time increases dramatically in KLEE’s approach
Evaluation

Reuse across Programs

GreenTrie achieves more inter-programs reuse than Green.
In some cases, GreenTrie has a little less reuse than KLEE’s approach.
The reason is that some constraints, which reuse the solution both across programs and in same program in GreenTrie, can only reuse constraints across programs in KLEE. Such constraints are counted for KLEE but not counted for GreenTrie.
Conclusion and Future Work

❖ **Contributions**
  ❖ Logical basis of implication-based reuse
  ❖ Trie-based store indexed with implication partial order graph
  ❖ Efficient logical subset/superset checking algorithms

❖ **Future works**
  ❖ Support more kinds of constraints other than linear integer constraints
  ❖ Reuse constraints which contains summaries
  ❖ Improve scalability for large-scale programs
Apologize

❖ I am sorry that I cannot answer your question face to face.
❖ If you have any question, please contact me with this email:

jxy@whu.edu.cn