# Enhancing Reuse of Constraint Solutions to Improve Symbolic Execution


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

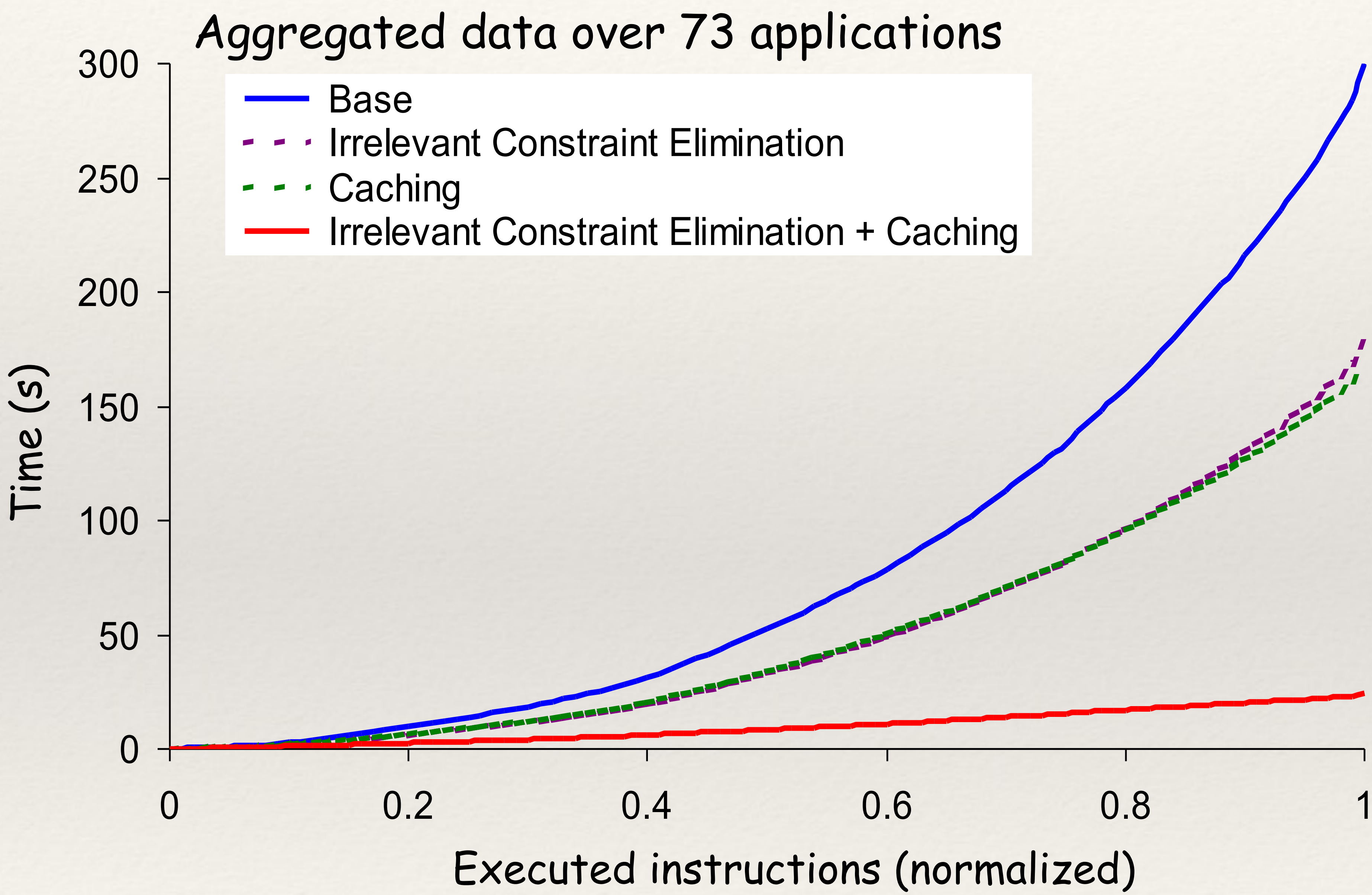
# Outline

- 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






[From Shauvik Roy Choudhary's Slides]

# Motivation

- **Reuse of Constraint Solutions**

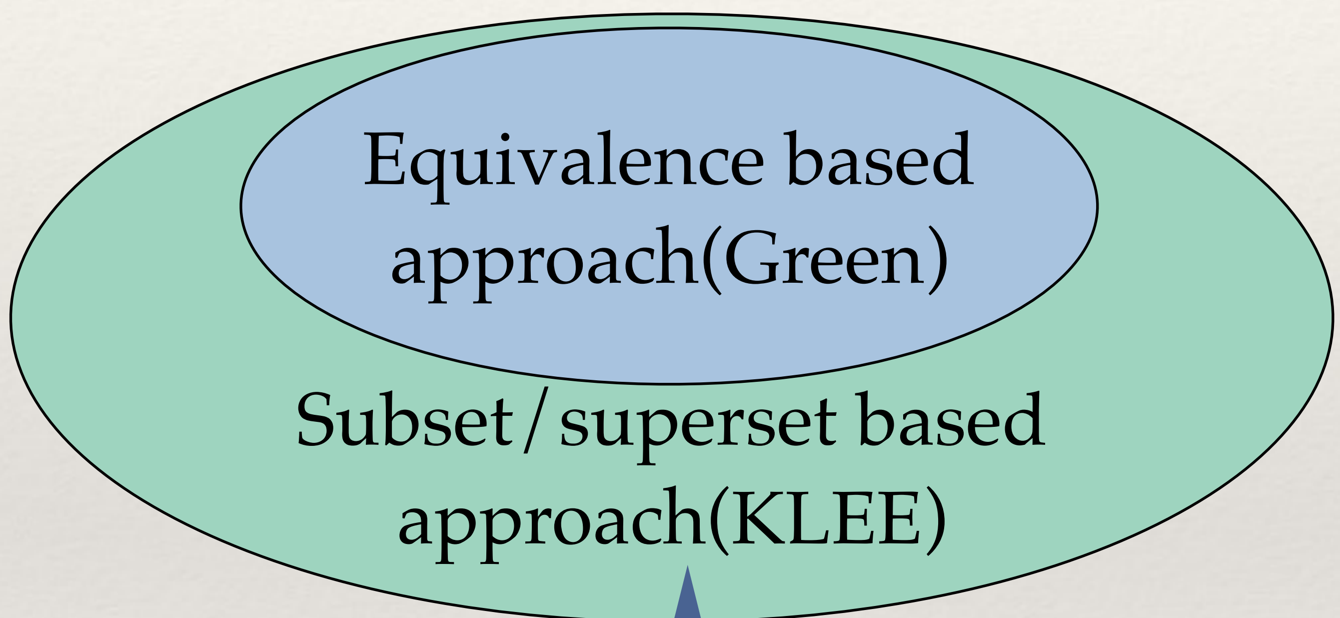
Equivalence based approach(Green)

$x>0$ is equivalent to $y>0$

$x+1>0 \wedge x<=1$ is equivalent to $y<2 \wedge y>=0$ (if x, y are integers)

# Motivation

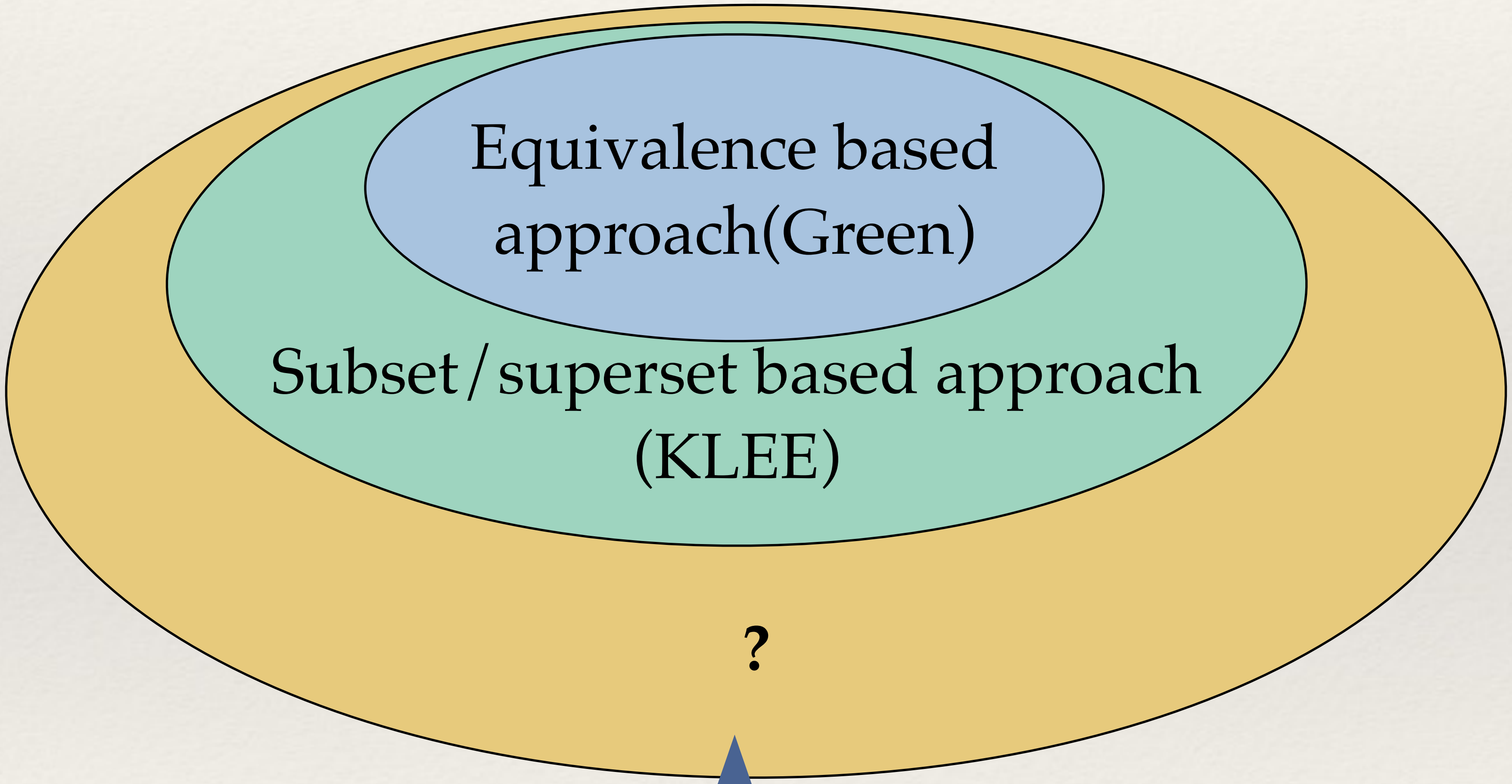
- **Reuse of Constraint Solutions**




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

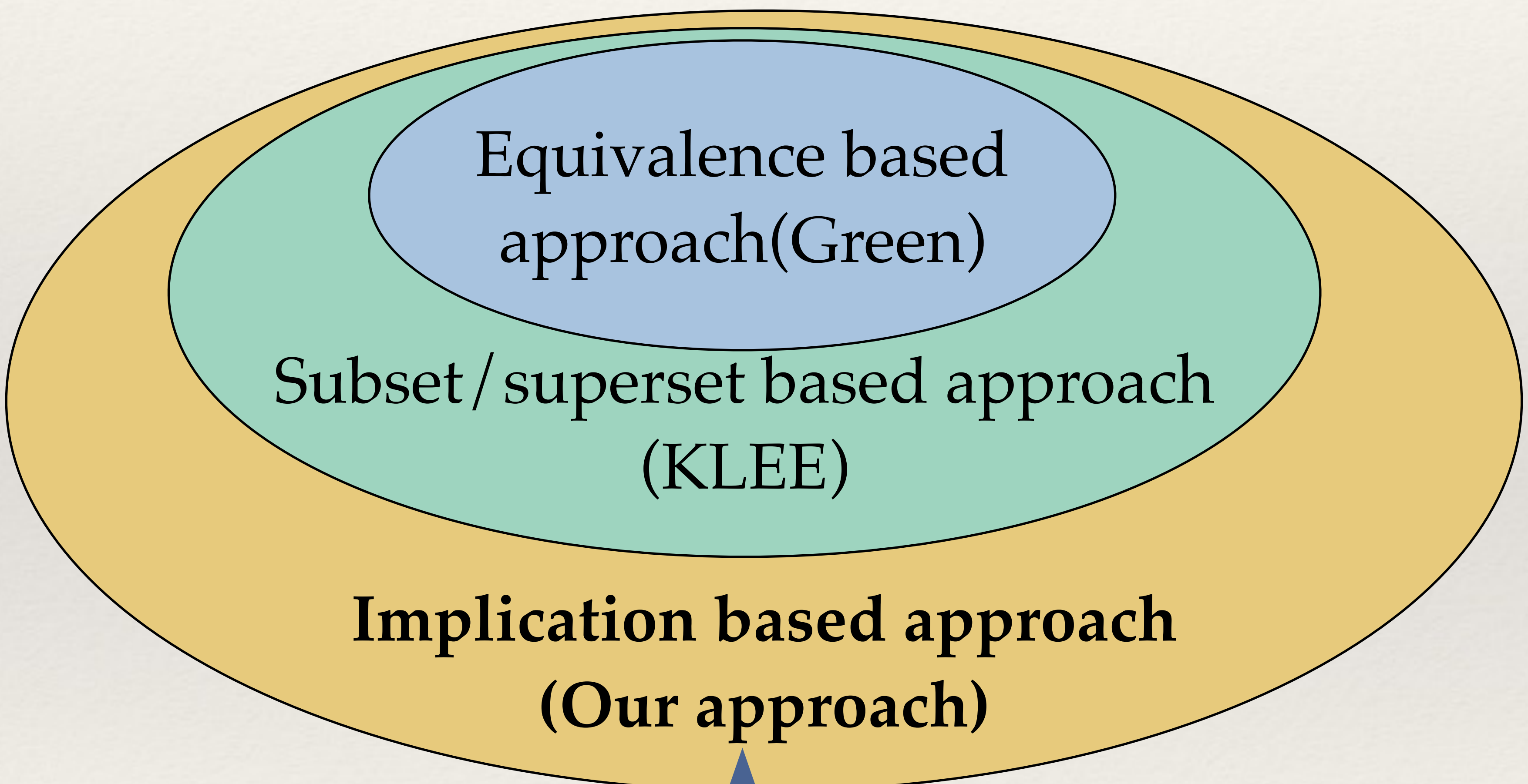
- **Reuse of Constraint Solutions**




If x>0 is satisfiable, can we prove x>-1 satisfiable?
If x<0^x>1 is unsatisfiable, can we prove x<-1^x>2 unsatisfiable?

# Motivation

- **Reuse of Constraint Solutions**




If x>0 is satisfiable, can we prove x>-1 satisfiable?
If x<0^x>1 is unsatisfiable, can we prove x<-1^x>2 unsatisfiable?

# Logical Basis of our Approach

## Implication and Satisfiability

Providing C1 → C2

- if C1 is satisfiable, C2 is satisfiable
- if C2 is unsatisfiable, C1 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）**
  - $\{c1,c2\} \subseteq \{c1,c2,c3\}$ means $c1 \wedge c2 \wedge c3 \rightarrow c1 \wedge c2$
- **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 $(m > 1 \wedge m < 2) \rightarrow (m \neq 0 \wedge m > -1 \wedge m < 2)$

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**

$$(R1)\frac{}{C \rightarrow C} \qquad (R2)\frac{n \neq n'}{P + n = 0 \rightarrow P + n' \neq 0}$$

$$(R3)\frac{n \geq n'}{P + n = 0 \rightarrow P + n' \leq 0} \qquad (R4)\frac{n \leq n'}{P + n = 0 \rightarrow P + n' \geq 0}$$

$$(R5)\frac{n > n'}{P + n \leq 0 \rightarrow P + n' \neq 0} \qquad (R6)\frac{n > n'}{P + n \leq 0 \rightarrow P + n' \leq 0}$$

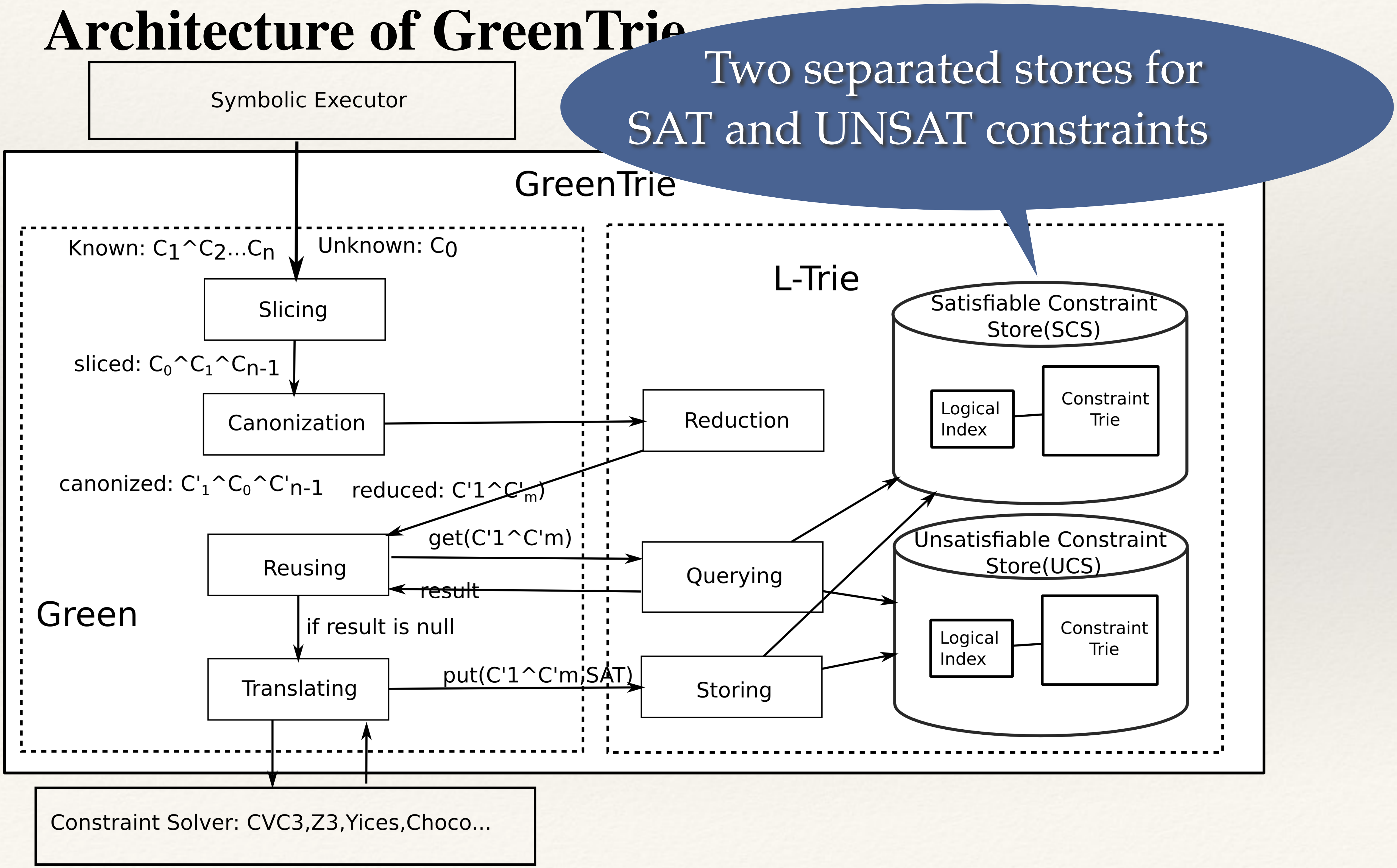
$$(R7)\frac{n < n'}{P + n \geq 0 \rightarrow P + n' \neq 0} \qquad (R8)\frac{n < n'}{P + n \geq 0 \rightarrow P + n' \geq 0}$$

P： non-constant prefix， n： constant number

E.g. x+y+3>=0 has a non-constant prefix x+y and a constant number 3

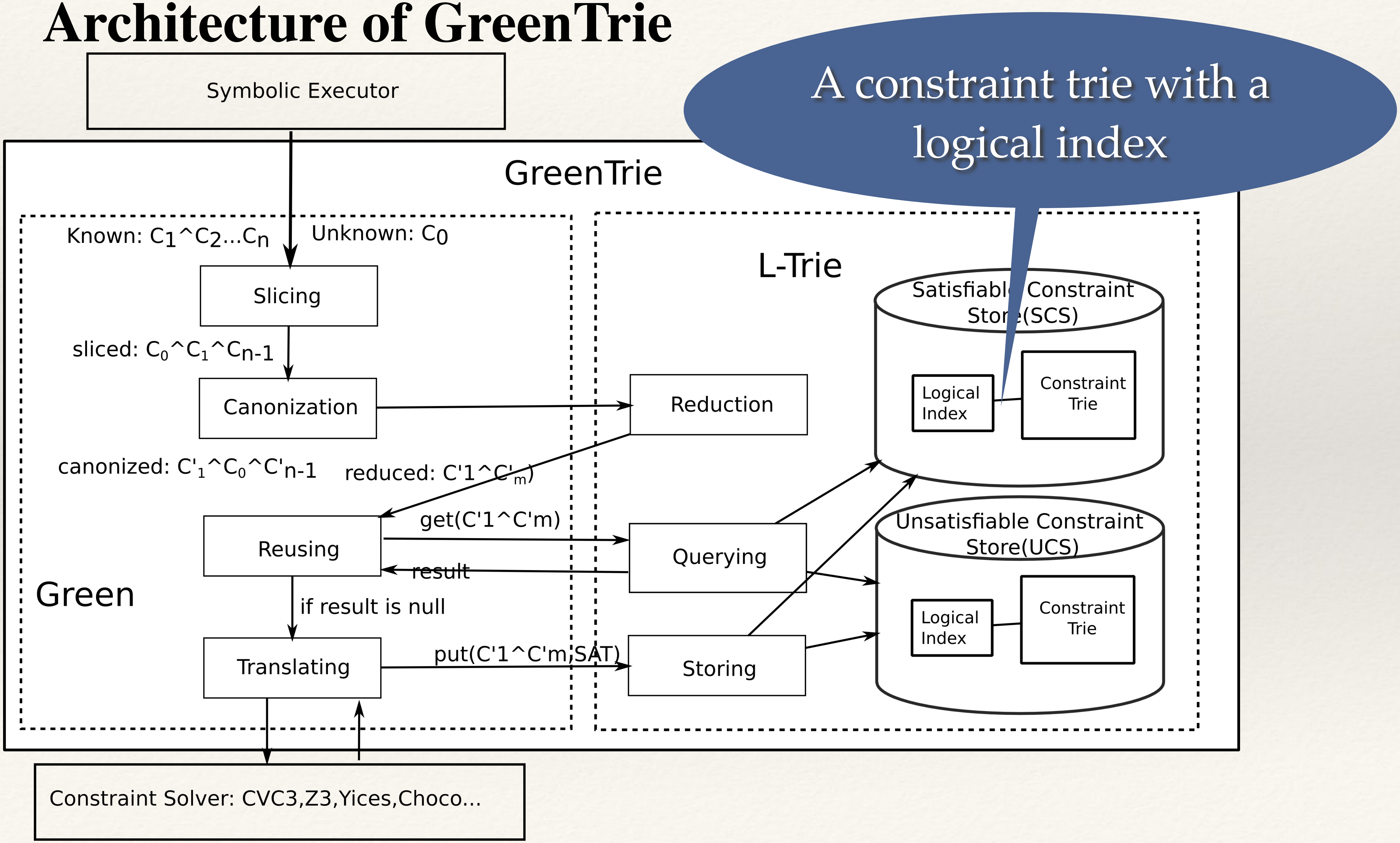
# GreenTrie Framework

- **Architecture of GreenTrie**



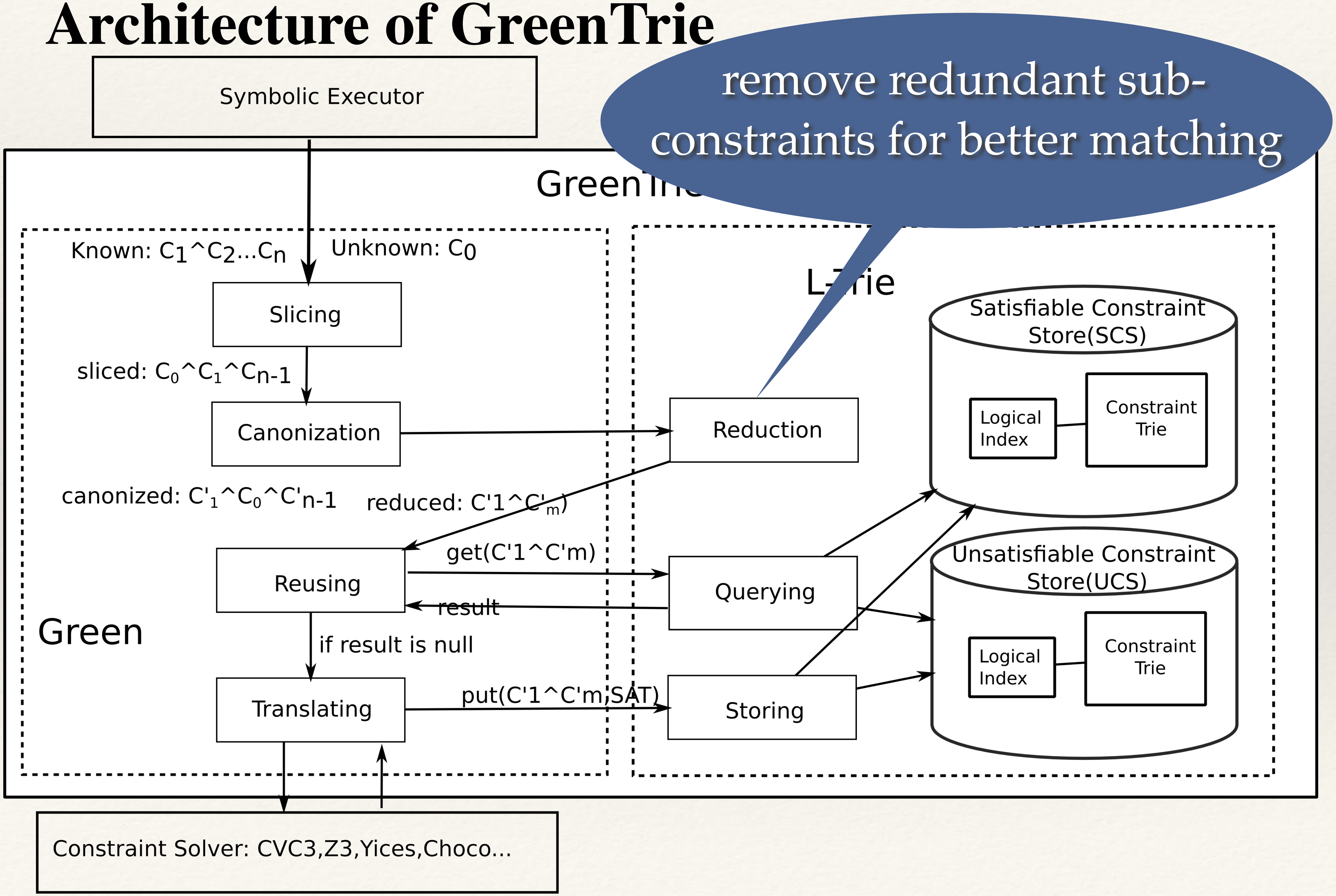
# GreenTrie Framework

- **Architecture of GreenTrie**



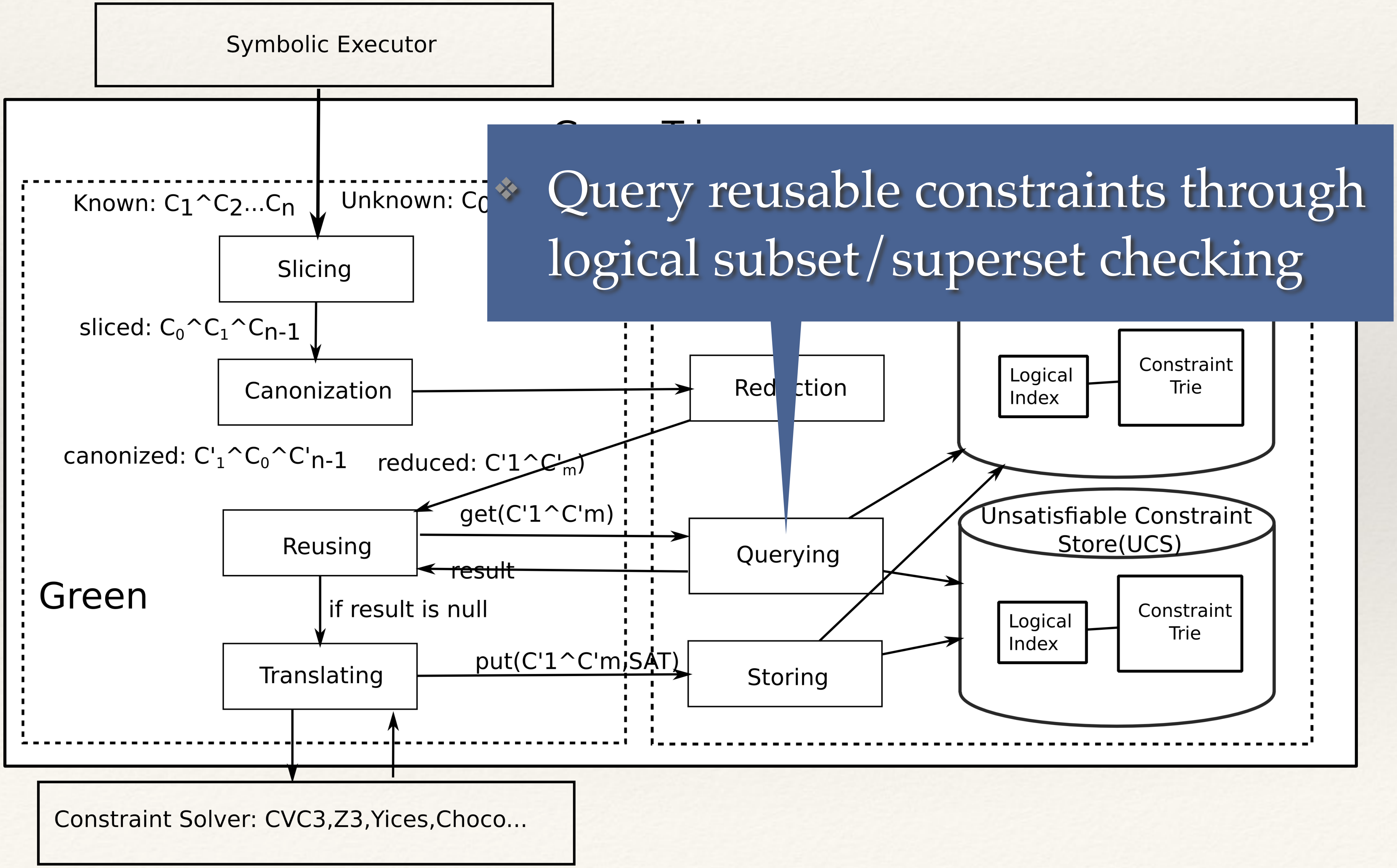
# GreenTrie Framework

- **Architecture of GreenTrie**



# GreenTrie Framework

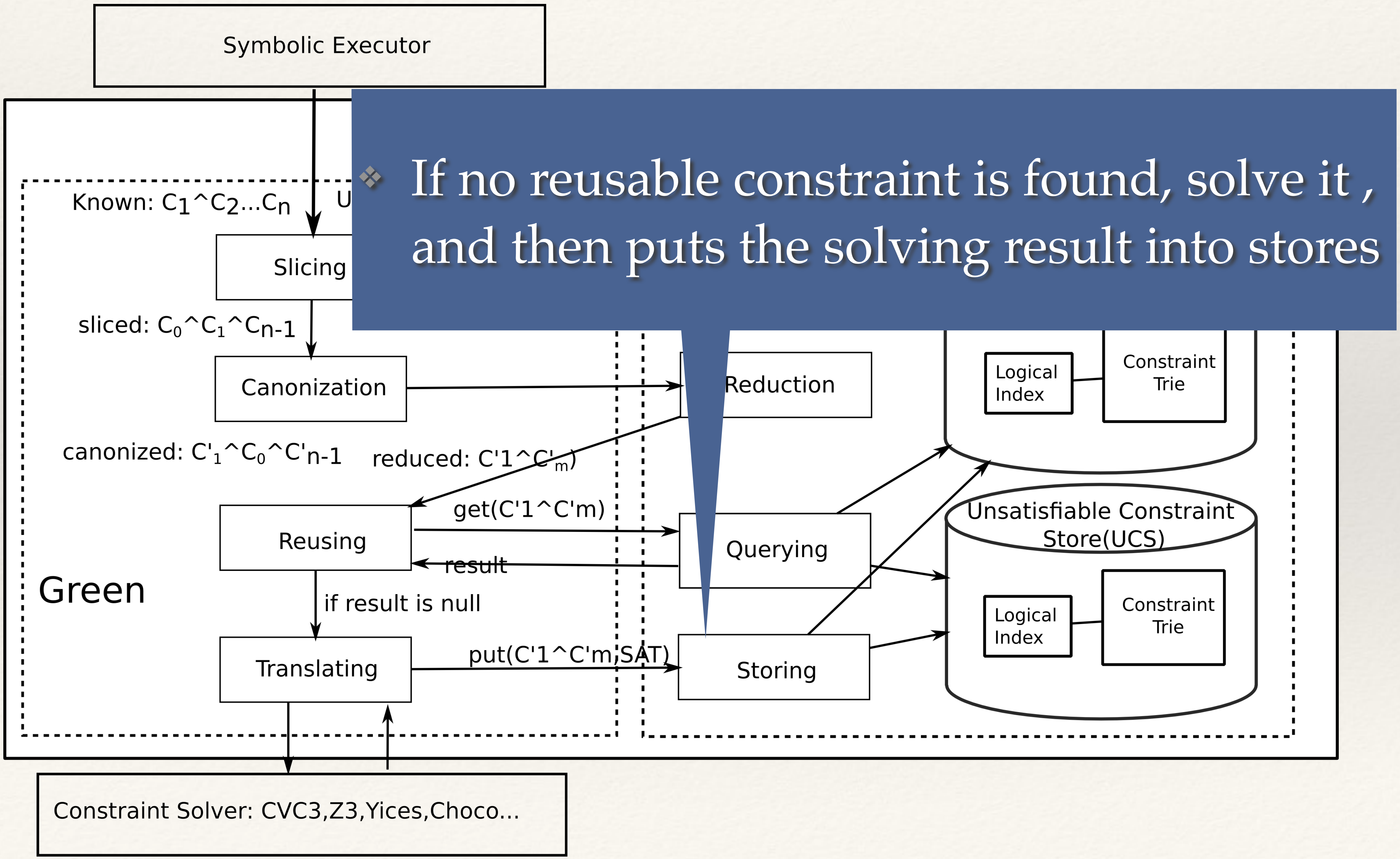
- **Architecture of GreenTrie**




- Query reusable constraints through logical subset/superset checking

# GreenTrie Framework

- **Architecture of GreenTrie**




- 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**

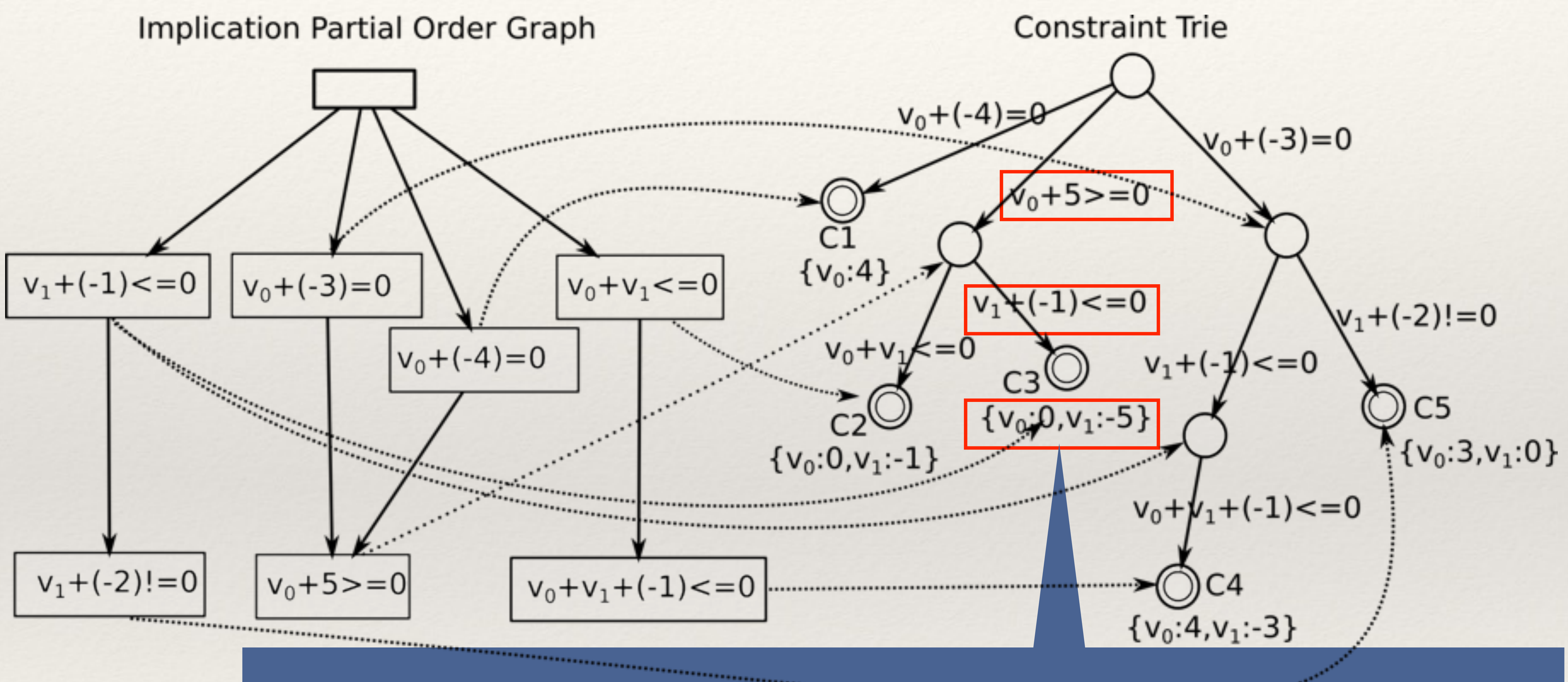
**Example**

$x+y+3 \geq 0 \wedge x+y+5 \geq 0 \wedge x+y-4 \leq 0 \wedge x+y \neq 0 \wedge x+y+6 \neq 0 \wedge x+y-4 \neq 0$

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

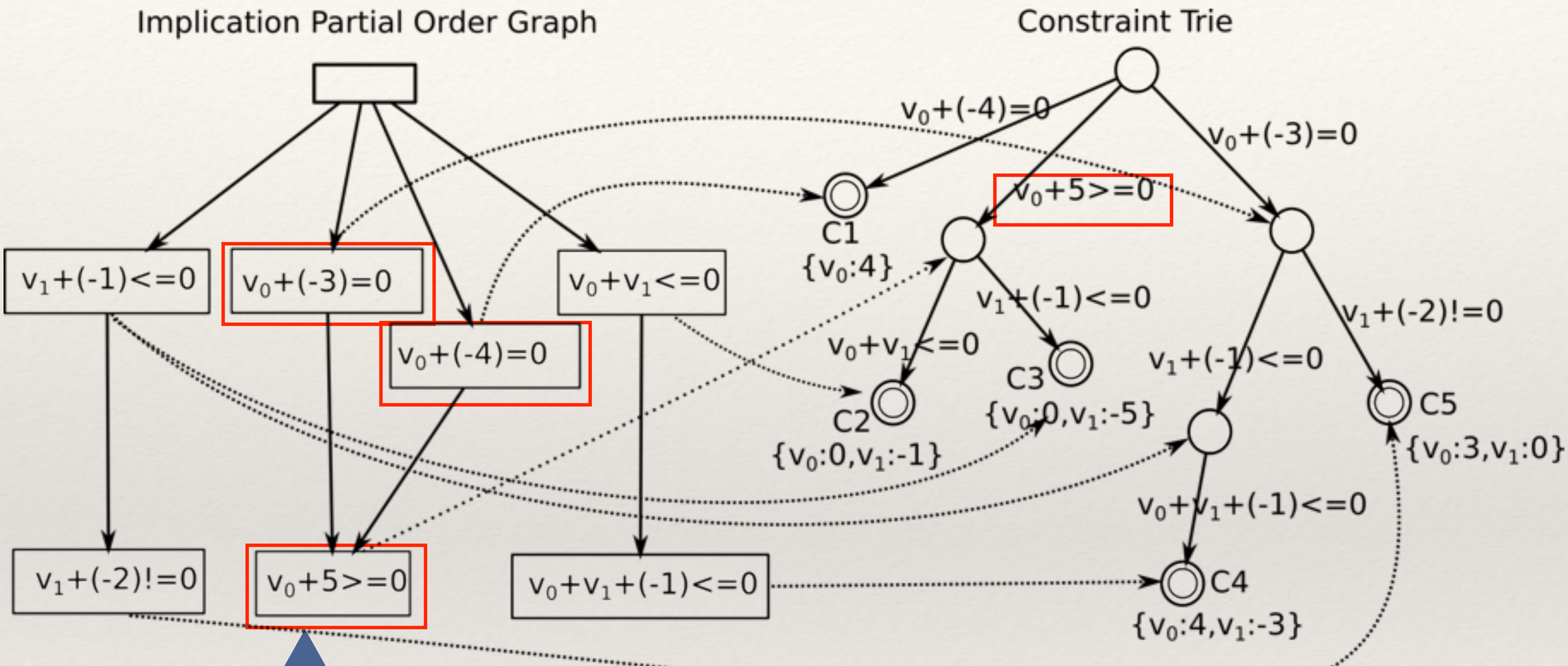
reduced: $x+y+3 \geq 0 \wedge x+y-4<0 \wedge x+y \neq 0$

# Constraint Storing




- C3 represents a constraint $V_0+5>=0 \wedge V_1+(-1)<=0$, which has a solution {v0:0, v1:-5}

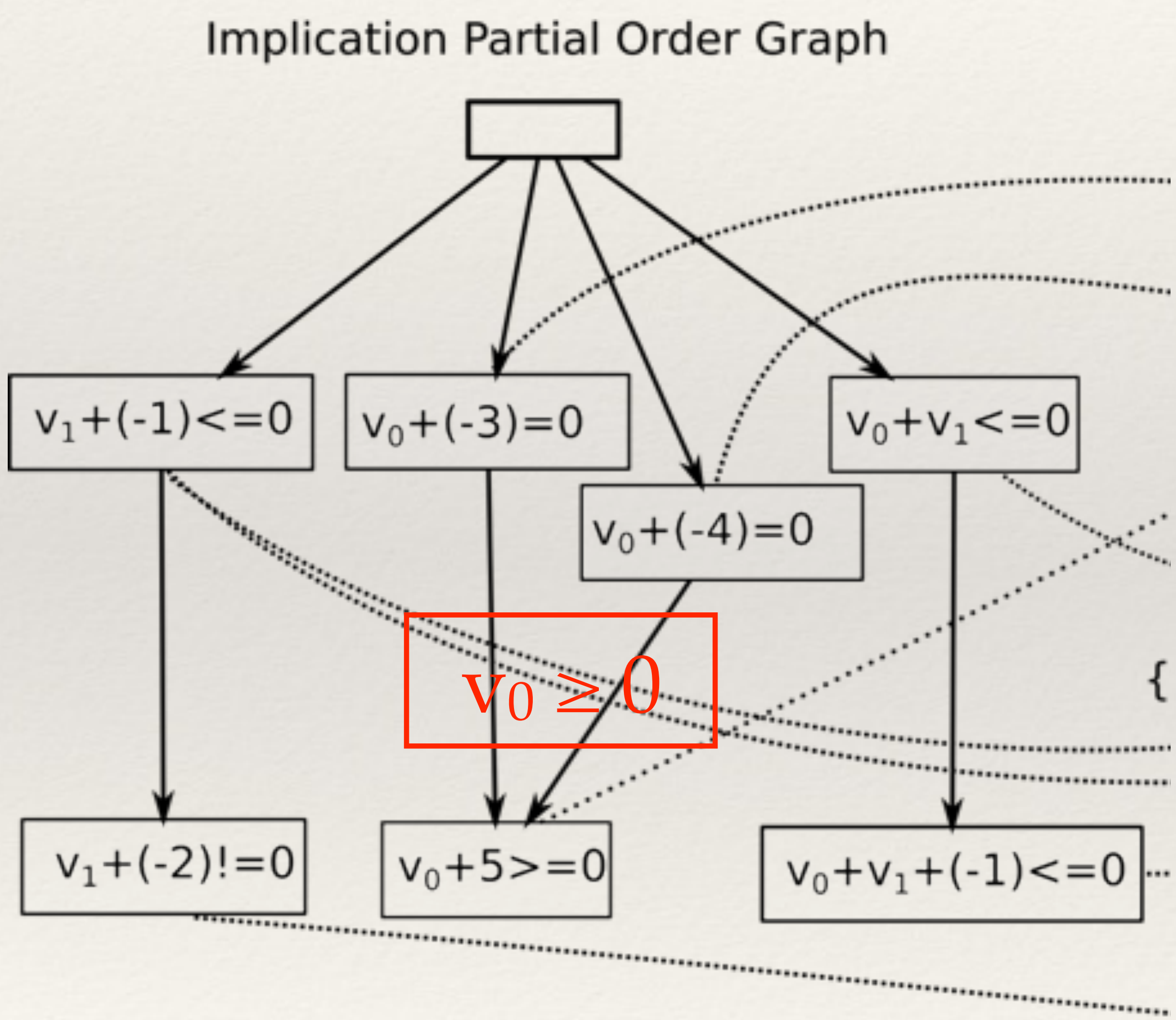
# Constraint Storing




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

# Constraint Querying

- **Implication Set(IS) and Reverse Implication Set(RIS)**




Example

Constraint: $v_0 \geq 0$

$IS_{v0 \geq 0}$: $\{v_0 + 5 >= 0\}$

$RIS_{v0 \geq 0}$: $\{v_0 + (-3) = 0,\ v_0 + (-4) = 0\}$

# Constraint Querying

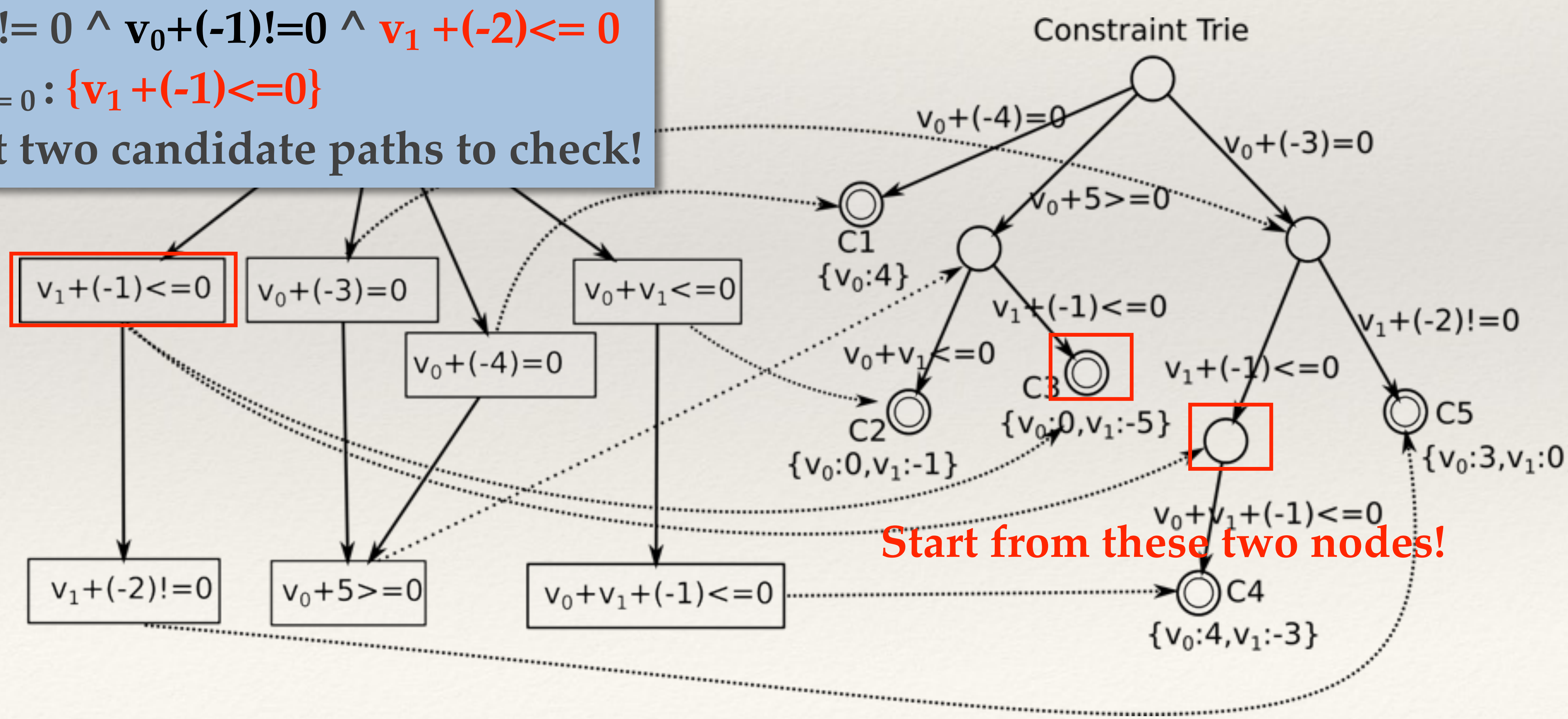
- **Logical Superset Checking Algorithm**
  - **Find a path in trie, so that every sub-constraint in target constraint is implied by at least one constraint on this path**

**Example**

Target: $v_0$ != 0 ^ $v_0$+(-1)!=0 ^ $v_1$ +(-2)<= 0

$RIS_{v1\ +(-2)<=\ 0}$ : {$v_1$ +(-1)<=0}

So, we got two candidate paths to check!



# Constraint Querying

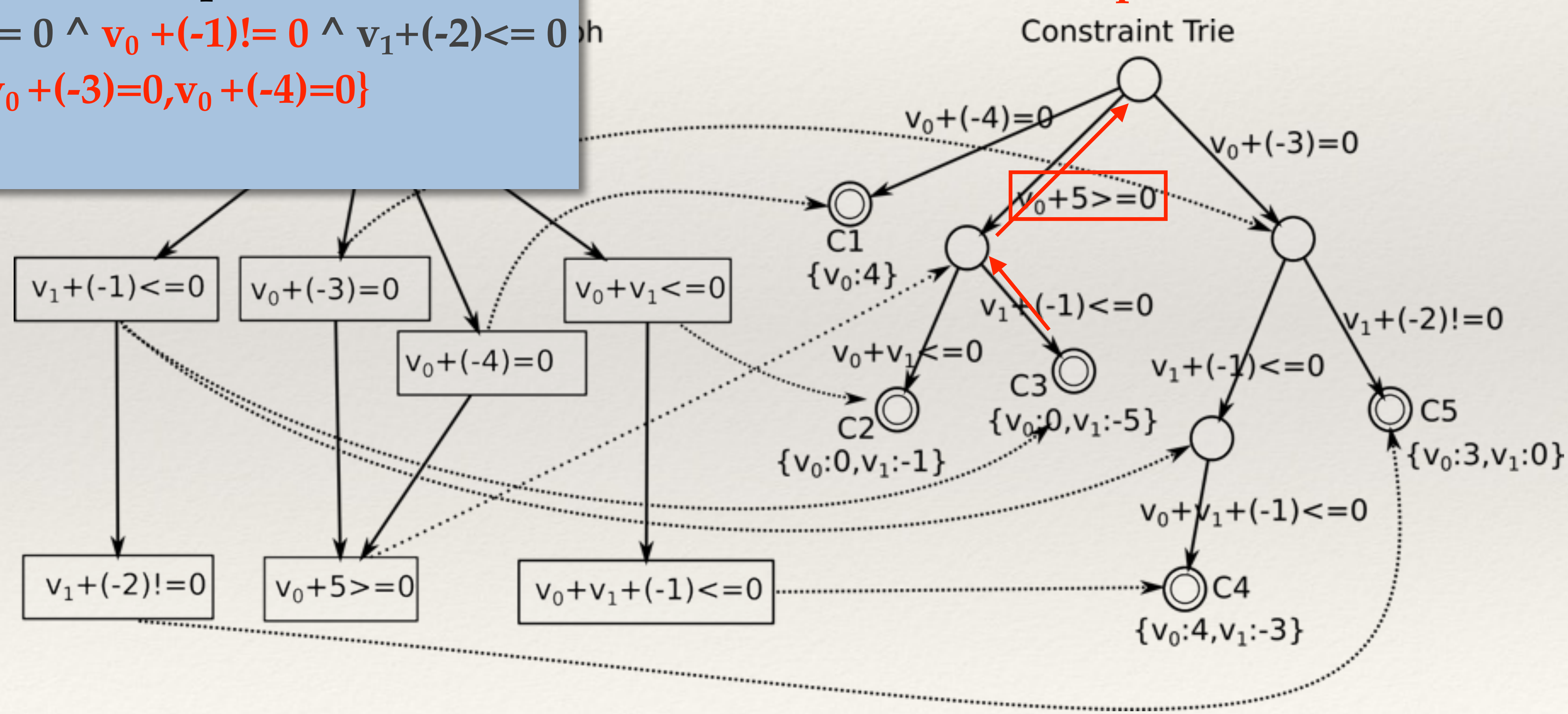
- **Logical Superset Checking Algorithm**

**Example**

Target : $v_0$ != 0 ^ $v_0$ +(-1)!= 0 ^ $v_1$+(-2)<= 0

$RIS_{v0\ !=\ 1}$ : {$v_0$ +(-3)=0,$v_0$ +(-4)=0}

**v0+5>=0 is not in the RIS,**
**the trie root is reached,**
**so this path doesn't match!**



# Constraint Querying

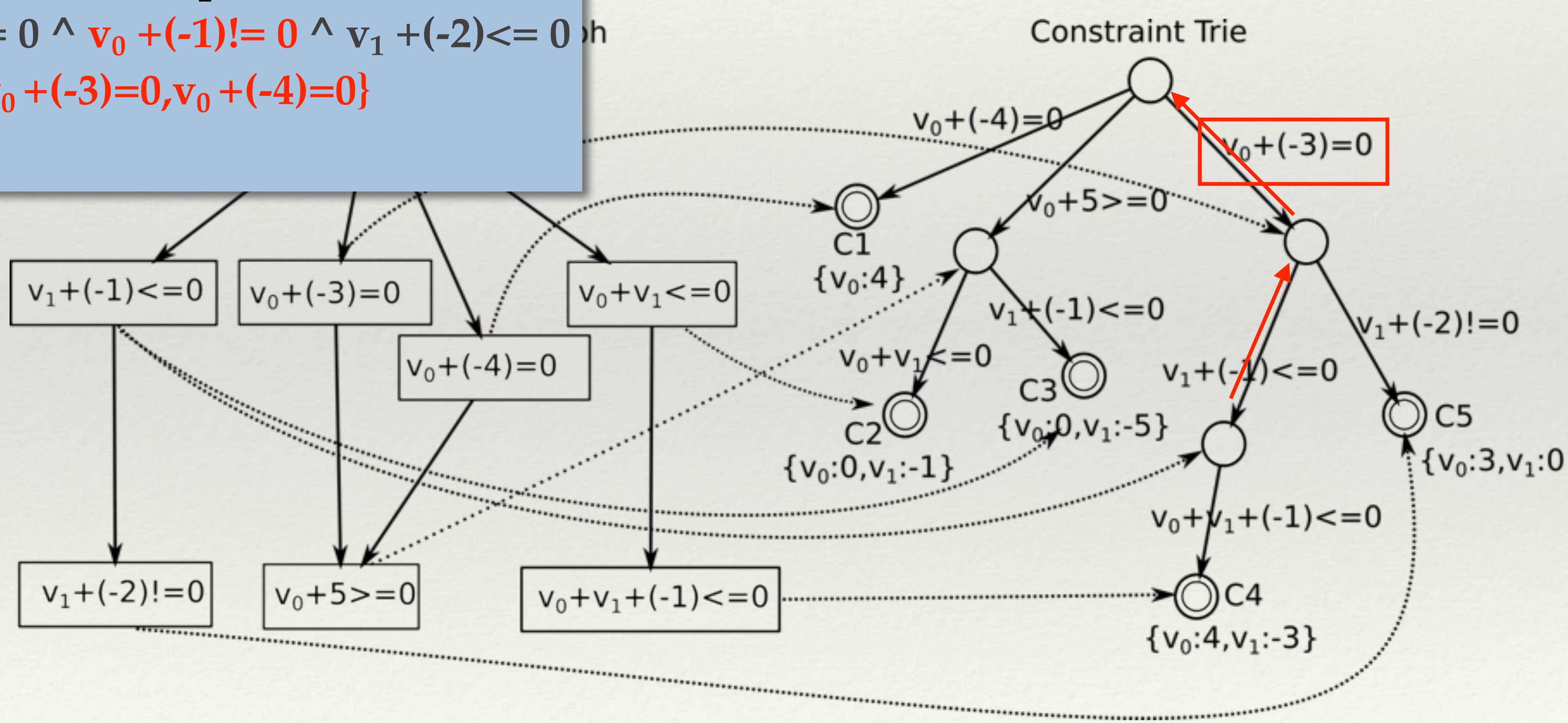
- **Logical Superset Checking Algorithm**

**Example**

Target: $v_0 \,!{=}\, 0$ ^ $v_0 +(-1)!= 0$ ^ $v_1 +(-2)<= 0$

$RIS_{v0\,!=\,1}$: $\{v_0 +(-3)=0, v_0 +(-4)=0\}$

v0+(-3)>=0 is in the RIS,
go on to check next sub-constraint
of target!



# Constraint Querying

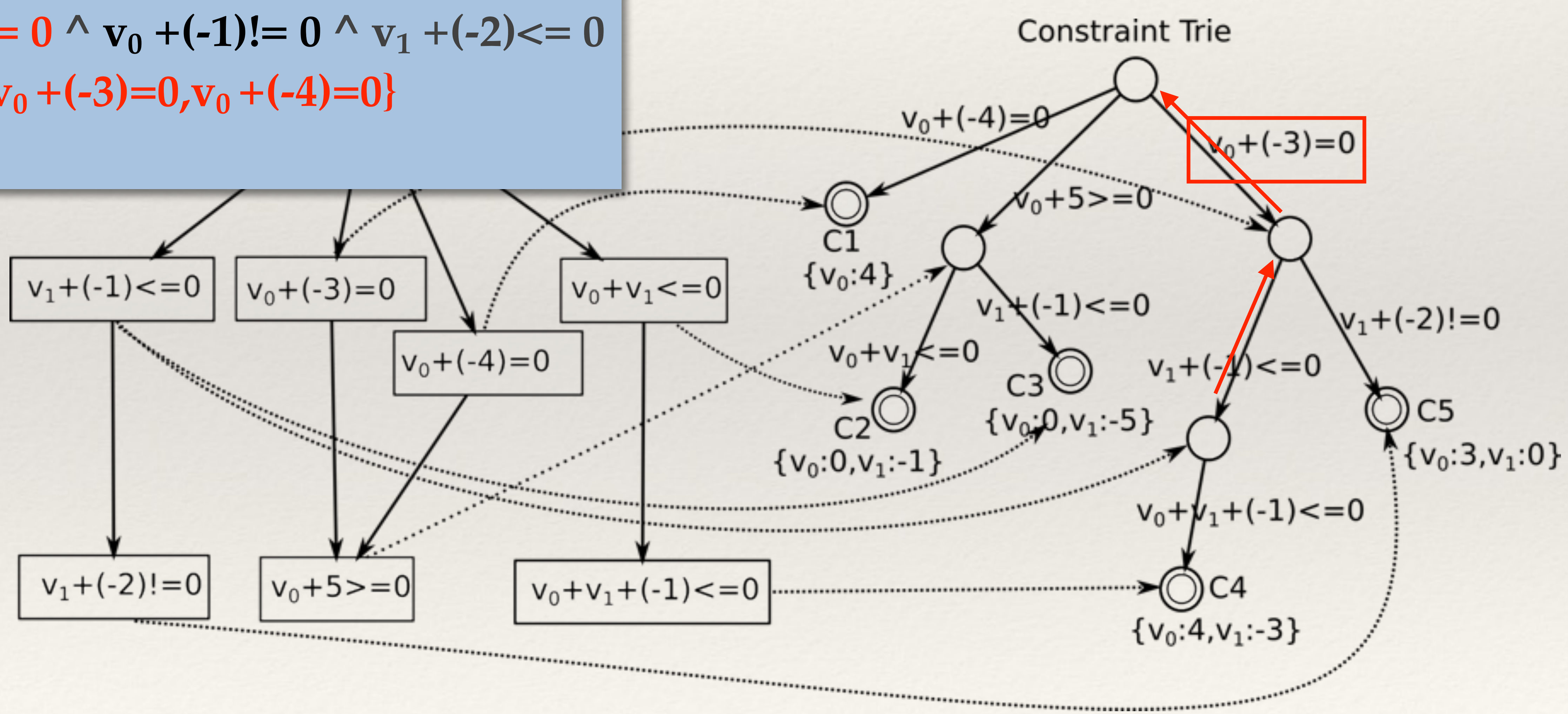
- **Logical Superset Checking Algorithm**

**$v_0+(-3)>=0$ is also in the RIS of $v_0$ != 0, now, every sub-constraint in target is implied by one constraint on this path. C4 is the reusable constraint!**

**Example**

**Target: $v_0$ != 0 ^ $v_0$ +(-1)!= 0 ^ $v_1$ +(-2)<= 0**

**$RIS_{v0\ != 0}$: {$v_0$ +(-3)=0, $v_0$ +(-4)=0}**



# Constraint Querying

- **Logical Subset Checking Algorithm**

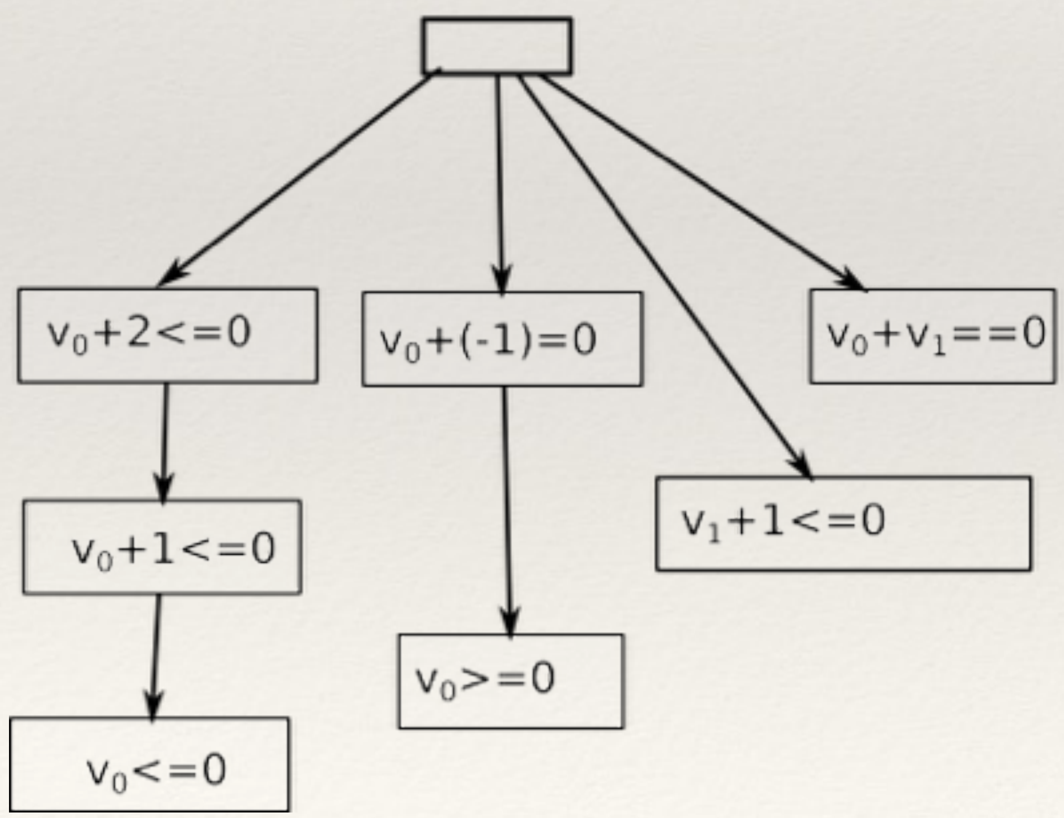
Target: $v_0 + (-1) >= 0 \ \wedge\ v_0 + 3 != 0 \ \wedge\ v_0 + 4 <= 0$

Union of ISs of the sub-constraints : $\{v_0 >= 0\} \cup \{\} \cup \{v_0 + 2 <= 0,\ v_0 + 1 <= 0\}$

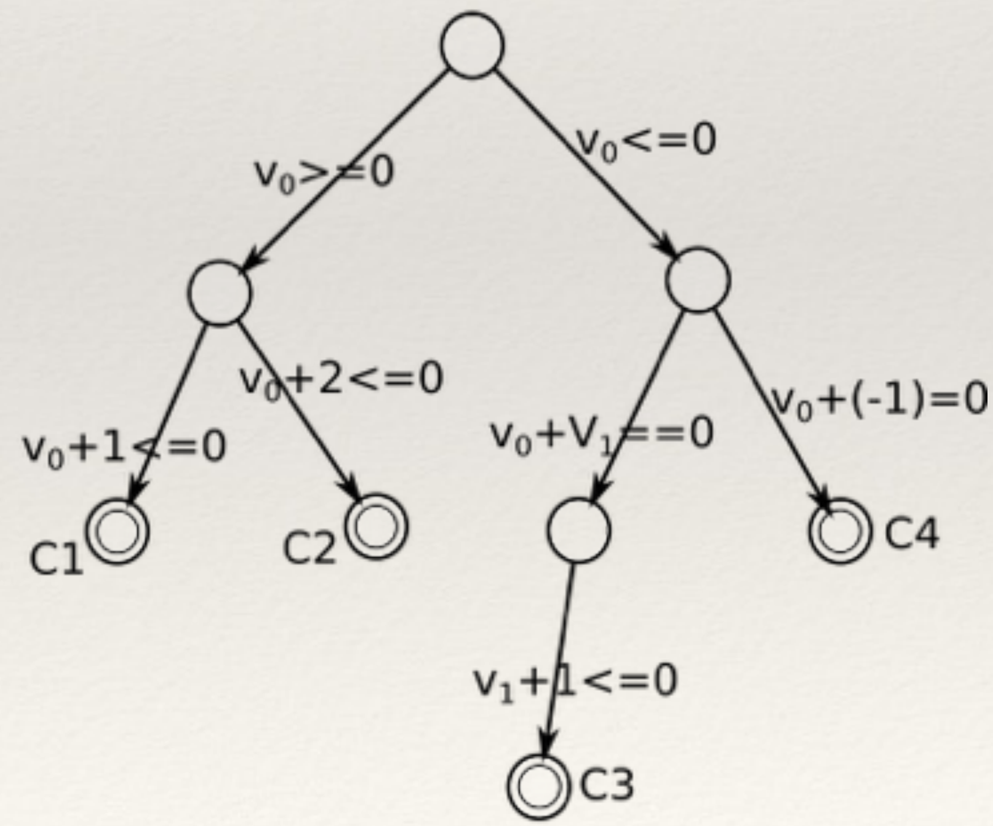
$IS_{union} = \{v_0 >= 0,\ v_0 + 2 <= 0,\ v_0 + 1 <= 0\}$

We will find a trie path, so that all its sub-constraints on the path exists in $IS_{union}$

Implication Partial Order Graph




Constraint Trie

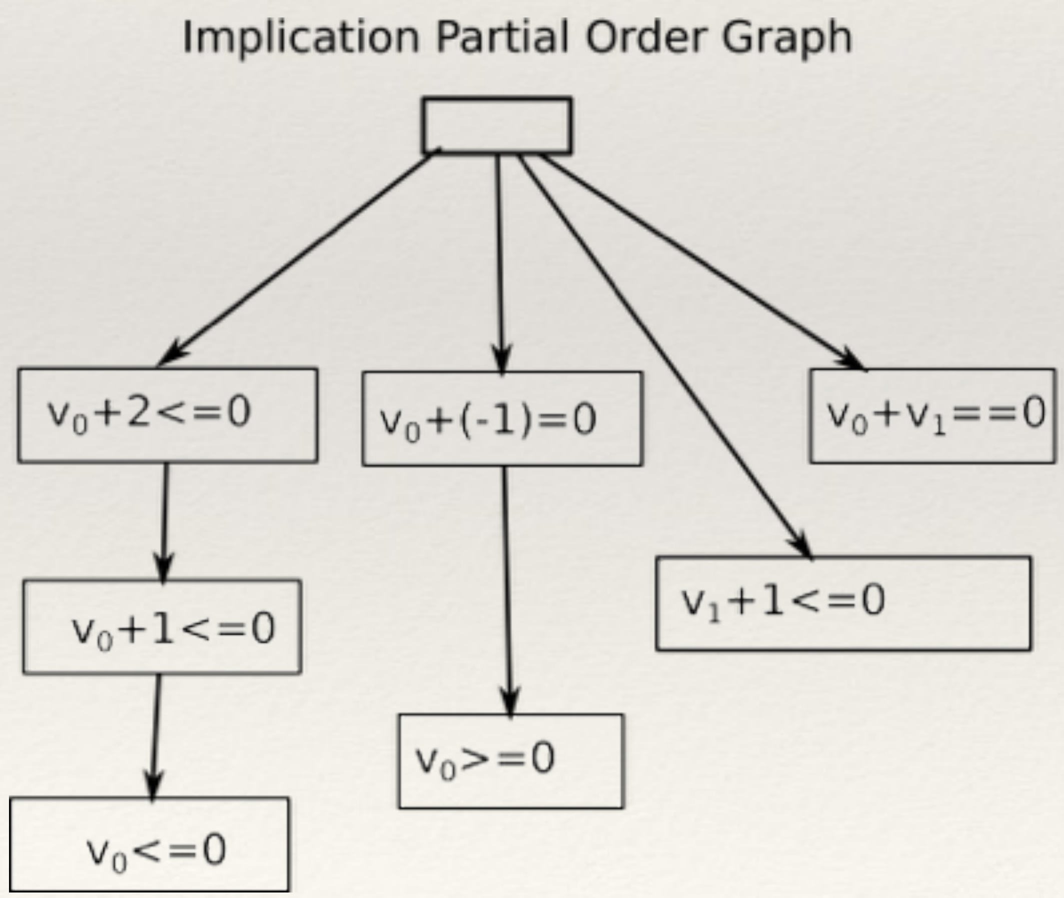


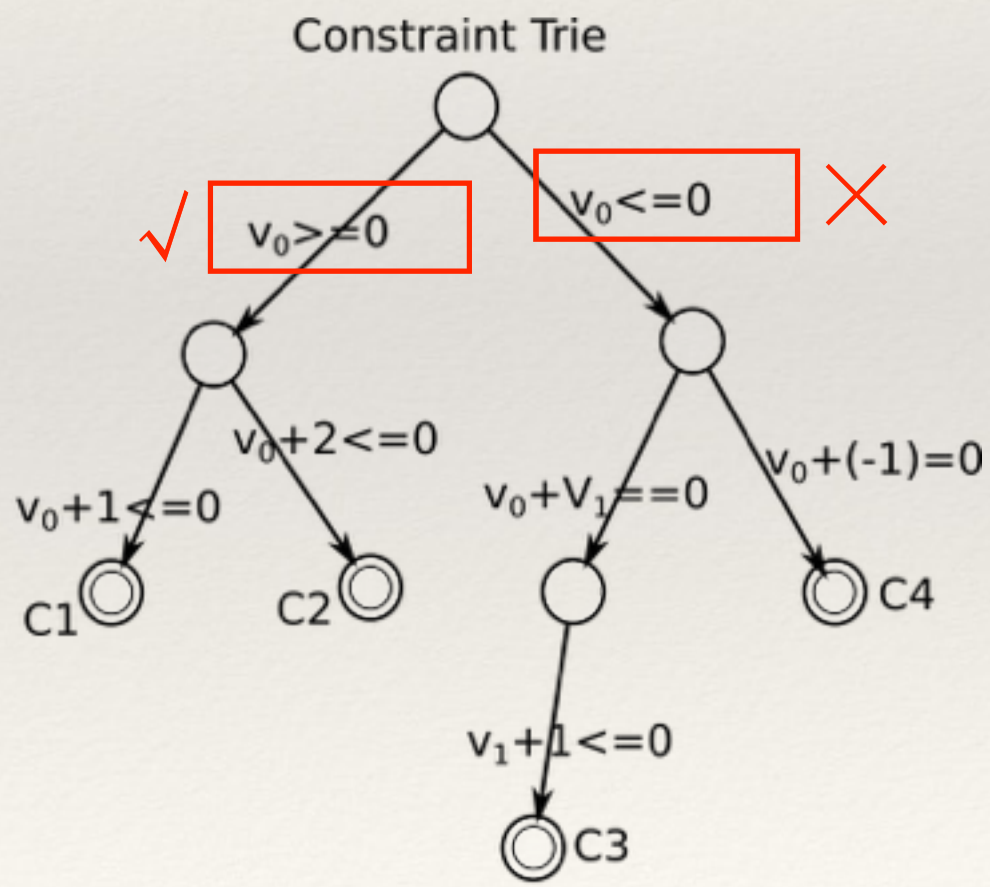
# Constraint Querying

- **Logical Subset Checking Algorithm**

**Target:** $v_0 + (-1) >= 0 \ \wedge \ v_0 + 3 != 0 \ \wedge \ v_0 + 4 <= 0$

$IS_{union} = \{v_0 >= 0, v_0 + 2 <= 0, v_0 + 1 <= 0\}$



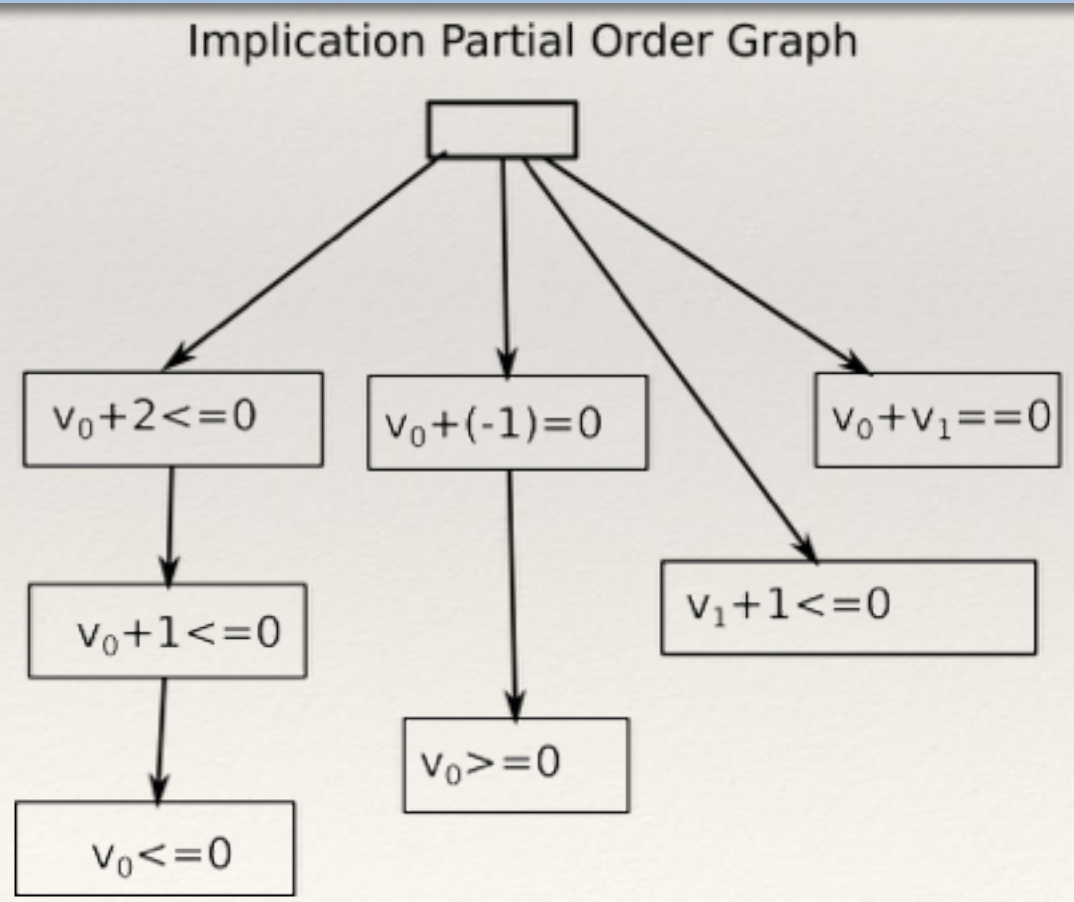
# Constraint Querying

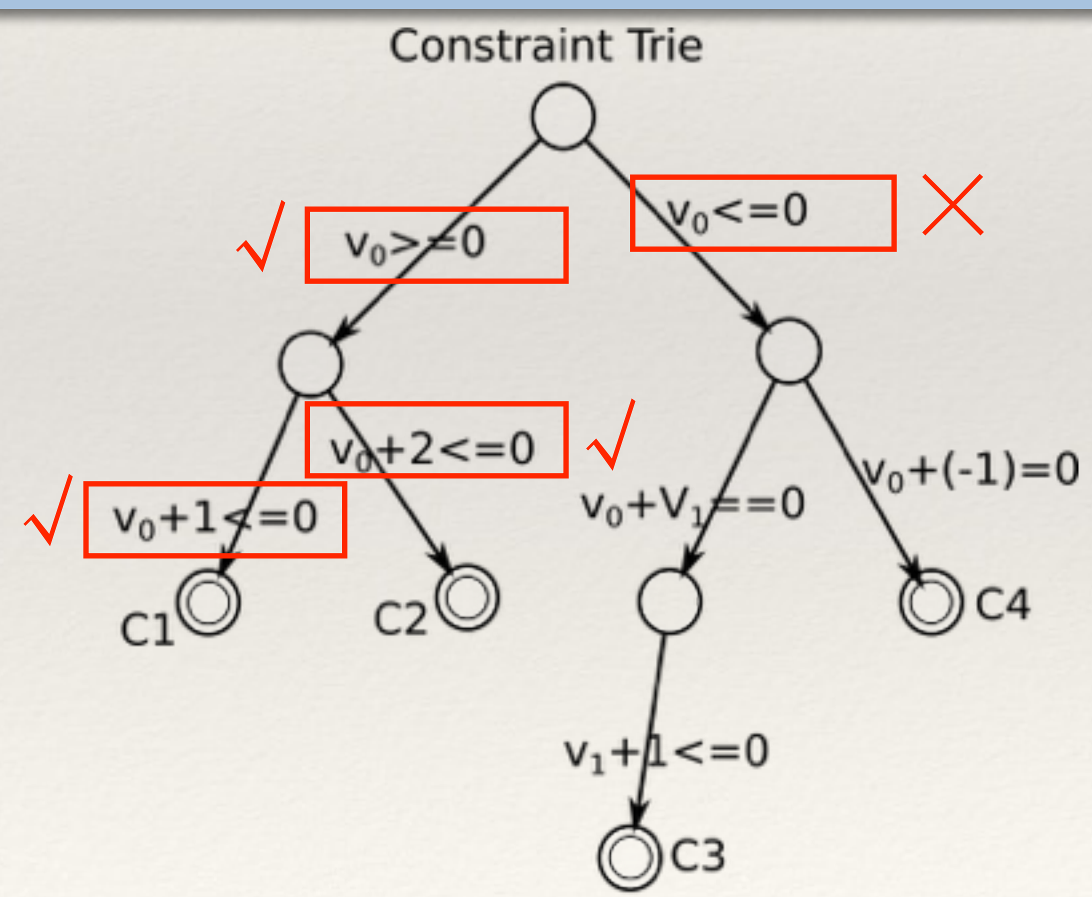
- **Logical Subset Checking Algorithm**

**Target:** $v_0 + (-1) >= 0 \ \wedge \ v_0 + 3 != 0 \ \wedge \ v_0 + 4 <= 0$

**$IS_{union}$ = {$v_0 >= 0$, $v_0 + 2 <= 0$, $v_0 + 1 <= 0$}**

**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: $R'=(n_1-n_3)/n_1$ $R''=(n_2-n_3)/n_2$

Time improvement ratio: $T'=(t_1-t_3)/t_1$ $T''=(t_2-t_3)/t_2$

# 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% |
| total/average | 4648 | 3888 | 2241 | 1949 | 49.87% | 13.03% | 34083 | 36809 | 23422 | 31.28% | 36.37% |

**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 gets better reuse ratio and saves more time than both Green and KLEE's approach.

# Evaluation

- **Reuse across Runs**

3421 constraints in store

Running time increases dramatically in KLEE's approach

**Table 4: Experimen** ... **s**

| 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

# Evaluation

- **Reuse across Programs**

Numbers of reused constraints for Green, KLEE approach and GreenTrie

**Table 5: Experiment[illegible] results of reuse across programs**

| Program | Trityp | Euclid | TCAS | TreeMap | BinTree | BinomialHeap | MerArbiter |
|---|---|---|---|---|---|---|---|
| Trityp | / | 0, 0, 3 | 0, 0, 3 | [illegible], 4, 4 | 0, 2, 2 | 0, 6, 7 | 0, 0, 1 |
| Euclid | 0, 0, 1 | / | 2, 5, 5 | [illegible], 0, 0 | 0, 3, 4 | 0, 2, 2 | 0, 0, 2 |
| TCAS | 0, 0, 2 | 2, 2, 2 | / | [illegible], 0, 0 | 0, 2, 3 | 0, 3, 4 | 0, 3, 4 |
| TreeMap | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | / | $\boxed{256,\ 326,\ 323}$ | 0, 0, 0 | 0, 0, 0 |
| BinTree | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | $\boxed{256,\ 449,\ 470}$ | / | 0, 1, 1 | 0, 0, 0 |
| BinomialHeap | 2, 2, 5 | 2, 2, 5 | 2, 8, 6 | 0, 2, 3 | 1, 11, 10 | / | 0, 0, 0 |
| MerArbiter | 0, 1, 2 | 0, 2 | 0, 3 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | / |

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