ISSTA 2021 Online
ACM SIGSOFT International Symposium on Software Testing and Analysis

Type and Interval Aware Array Constraint Solving for Symbolic Execution

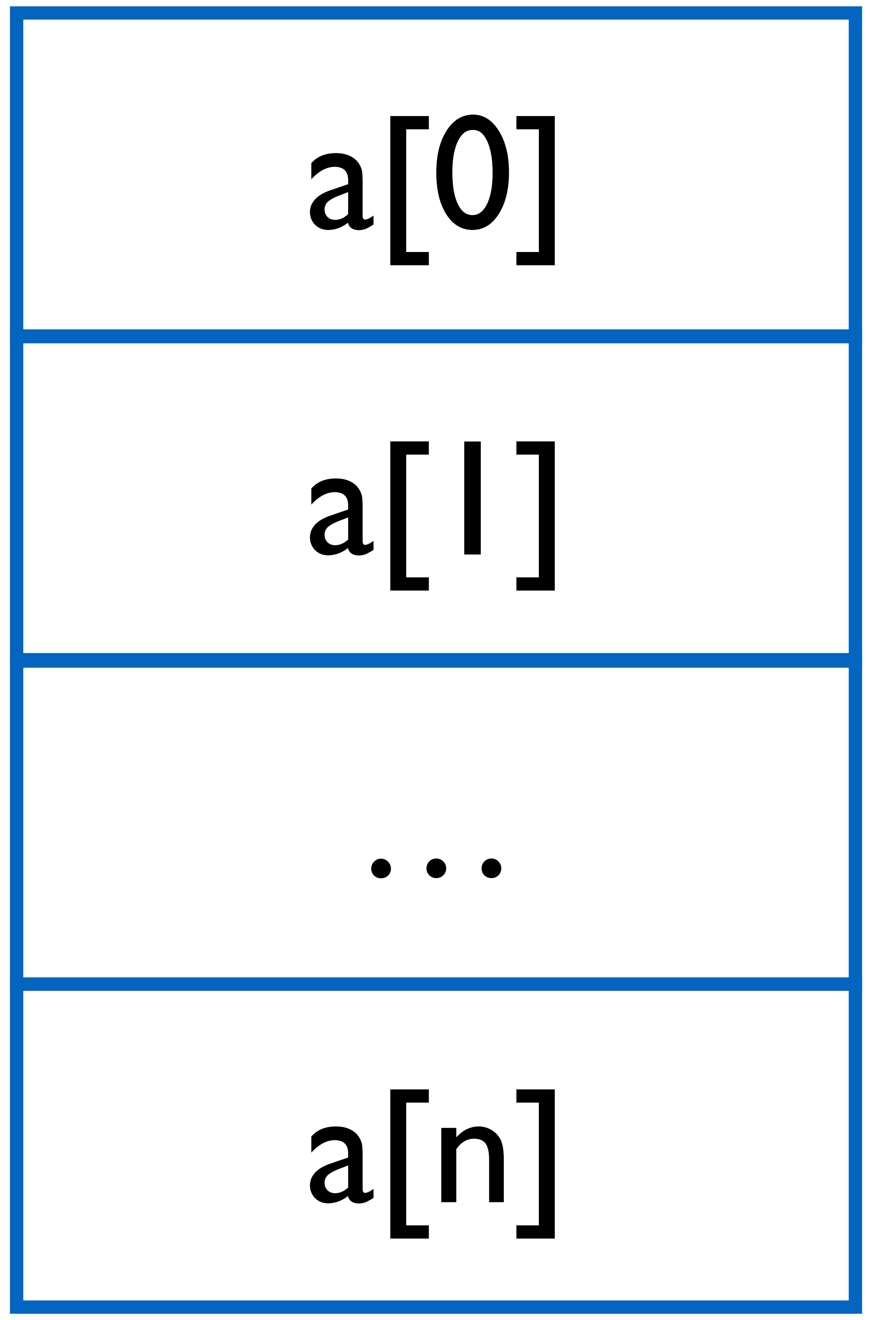
Ziqi Shuai and Zhenbang Chen
{szq, zbchen}@nudt.edu.cn

Joint work with Yufeng Zhang, Jun Sun and Ji Wang

Array Code Symbolic Execution

Arrays are ubiquitous in programs

int a[n];



Array Code Symbolic Execution

Arrays are ubiquitous in programs

int a[n];

a[0]
a[1]
…
a[n]

Array sorting

```
for(int i = 0; i < N-1; i++){
      int min = i;
      for(int j = i+1; j < N ; j++) {
        if (a[j] < a[min]) min = j;
      }
      int tmp = a[i];
      a[i] = a[min];
      a[min] = tmp;
}
```

Array Code Symbolic Execution

Arrays are ubiquitous in programs

The symbolic execution of array code is challenging

Array Code Symbolic Execution

Arrays are ubiquitous in programs

The symbolic execution of array code is challenging

a[0]	a[1]	a[2]	…	a[n]

Array Code Symbolic Execution

Arrays are ubiquitous in programs

The symbolic execution of array code is challenging

a[0]	a[1]	a[2]	…	a[n]

a[i]

Array Code Symbolic Execution

Arrays are ubiquitous in programs

The symbolic execution of array code is challenging

a[0]	a[1]	a[2]	…	a[n]

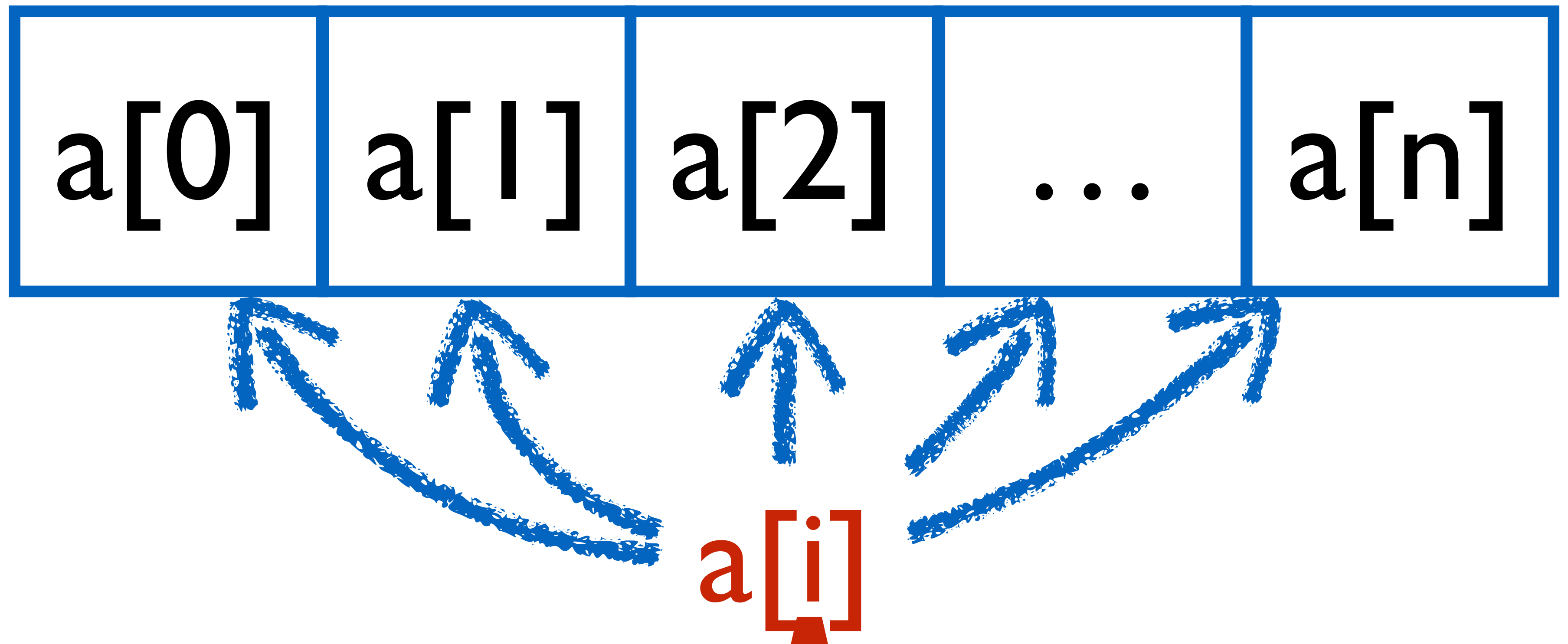
a[i]

i is a symbolic variable

Array Code Symbolic Execution

Arrays are ubiquitous in programs

The symbolic execution of array code is challenging

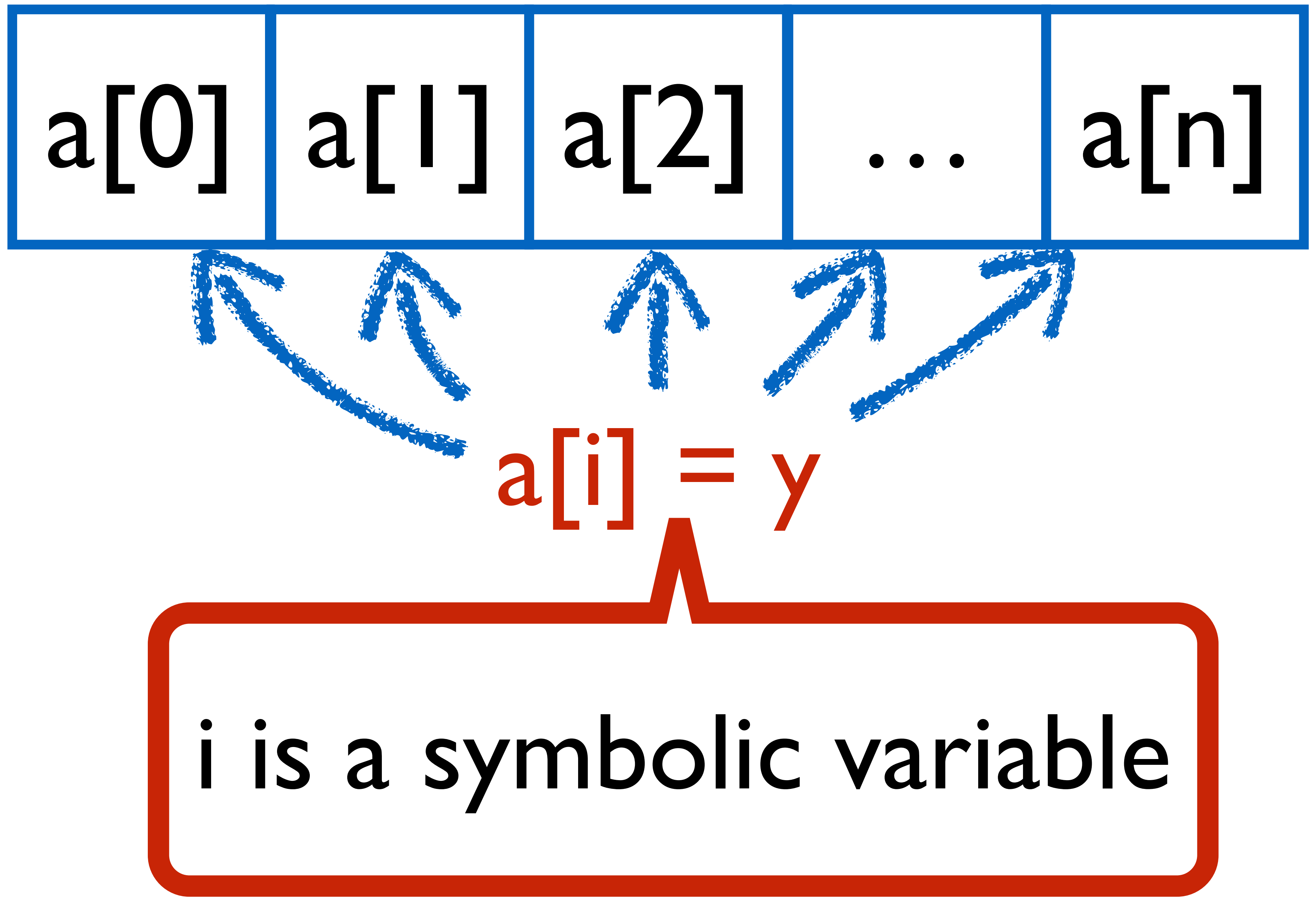


i is a symbolic variable

Array Code Symbolic Execution

Arrays are ubiquitous in programs

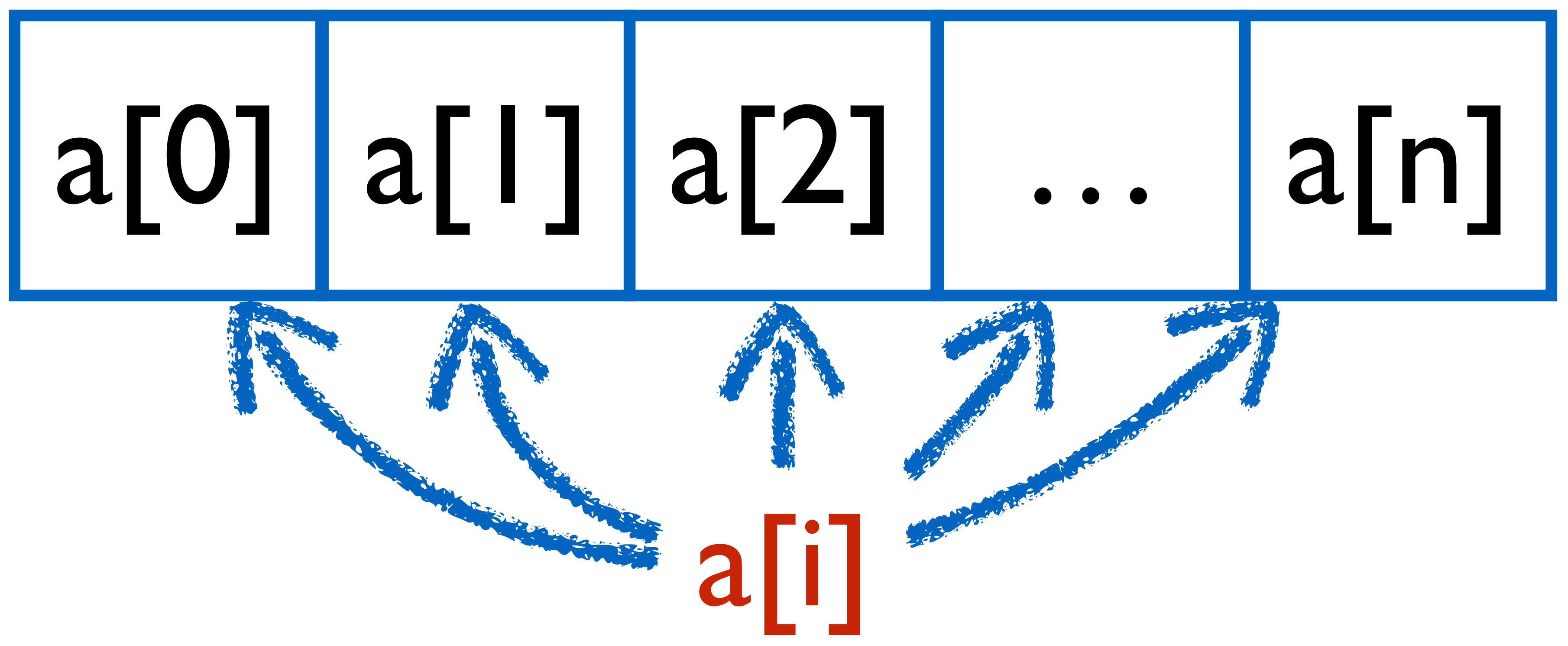
The symbolic execution of array code is challenging



Array Code Symbolic Execution

Arrays are ubiquitous in programs

The symbolic execution of array code is challenging



Array SMT Theory

Array SMT Theory

Read $\mathcal{R}(a, i)$

Write $\mathcal{W}(a, i, v)$

Array SMT Theory

Read $\mathcal{R}(a, i)$

Read the *ith* element of the array *a*

Write $\mathcal{W}(a, i, v)$

Array SMT Theory

Read $\mathcal{R}(a, i)$

Read the *ith* element of the array *a*

Write $\mathcal{W}(a, i, v)$

Write value *v* to the *ith* element of the array *a*

Array SMT Theory

Read $\mathcal{R}(a, i)$

Write $\mathcal{W}(a, i, v)$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

Array SMT Theory

Read $\mathcal{R}(a, i)$

Write $\mathcal{W}(a, i, v)$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

$$a = \{0, 0, 0, 11\}$$

Array SMT Theory

Read $\mathcal{R}(a, i)$

Write $\mathcal{W}(a, i, v)$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

$$a = \{0, 0, 0, 11\}$$

Satisfiable

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a, i) = \mathcal{R}(a, j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a, j, v), i) = \begin{cases} v & i = j \\ \mathcal{R}(a, i) & \text{otherwise} \end{cases}$

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a, i) = \mathcal{R}(a, j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a, j, v), i) = \begin{cases} v & i = j \\ \mathcal{R}(a, i) & \text{otherwise} \end{cases}$

Use these two axioms to eliminate the array terms in the array constraint

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a, i) = \mathcal{R}(a, j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a, j, v), i) = \begin{cases} v & i = j \\ \mathcal{R}(a, i) & \text{otherwise} \end{cases}$

$$a = \{0, 0, 0, 11\}$$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a, i) = \mathcal{R}(a, j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a, j, v), i) = \begin{cases} v & i = j \\ \mathcal{R}(a, i) & \text{otherwise} \end{cases}$

$$a = \{0, 0, 0, 11\}$$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

$$u > 10 \wedge i \geq 0 \wedge i \leq 3 \wedge \Big(\bigwedge_{n \in \{0,1,2\}} i = n \Rightarrow u = 0 \Big) \wedge i = 3 \Rightarrow u = 11$$

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a, i) = \mathcal{R}(a, j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a, j, v), i) = \begin{cases} v & i = j \\ \mathcal{R}(a, i) & \text{otherwise} \end{cases}$

$$a = \{0, 0, 0, 11\}$$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

$$u > 10 \wedge i \geq 0 \wedge i \leq 3 \wedge \left(\bigwedge_{n \in \{0,1,2\}} i = n \Rightarrow u = 0 \right) \wedge i = 3 \Rightarrow u = 11$$

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a,i) = \mathcal{R}(a,j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a,j,v),i) = \begin{cases} v & i = j \\ \mathcal{R}(a,i) & \text{otherwise} \end{cases}$

$$a = \{0, 0, 0, 11\}$$

$$\mathcal{R}(a,i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

$$u > 10 \wedge i \geq 0 \wedge i \leq 3 \wedge \left(\bigwedge_{n \in \{0,1,2\}} i = n \Rightarrow u = 0 \right) \wedge i = 3 \Rightarrow u = 11$$

Array SMT Theory

Axiom 1 $i = j \Rightarrow \mathcal{R}(a, i) = \mathcal{R}(a, j)$

Axiom 2 $\mathcal{R}(\mathcal{W}(a, j, v), i) = \begin{cases} v & i = j \\ \mathcal{R}(a, i) & \text{otherwise} \end{cases}$

$$a = \{0, 0, 0, 11\}$$

$$\mathcal{R}(a, i) > 10 \wedge i \geq 0 \wedge i \leq 3$$

⇩

$$u > 10 \wedge i \geq 0 \wedge i \leq 3 \wedge \Big(\bigwedge_{n \in \{0,1,2\}} i = n \Rightarrow u = 0 \Big) \wedge i = 3 \Rightarrow u = 11$$

Memory modeling in SE

- Byte-level memory reasoning in symbolic execution
  - QF_ABV SMT theory
  - KLEE、S2E、…

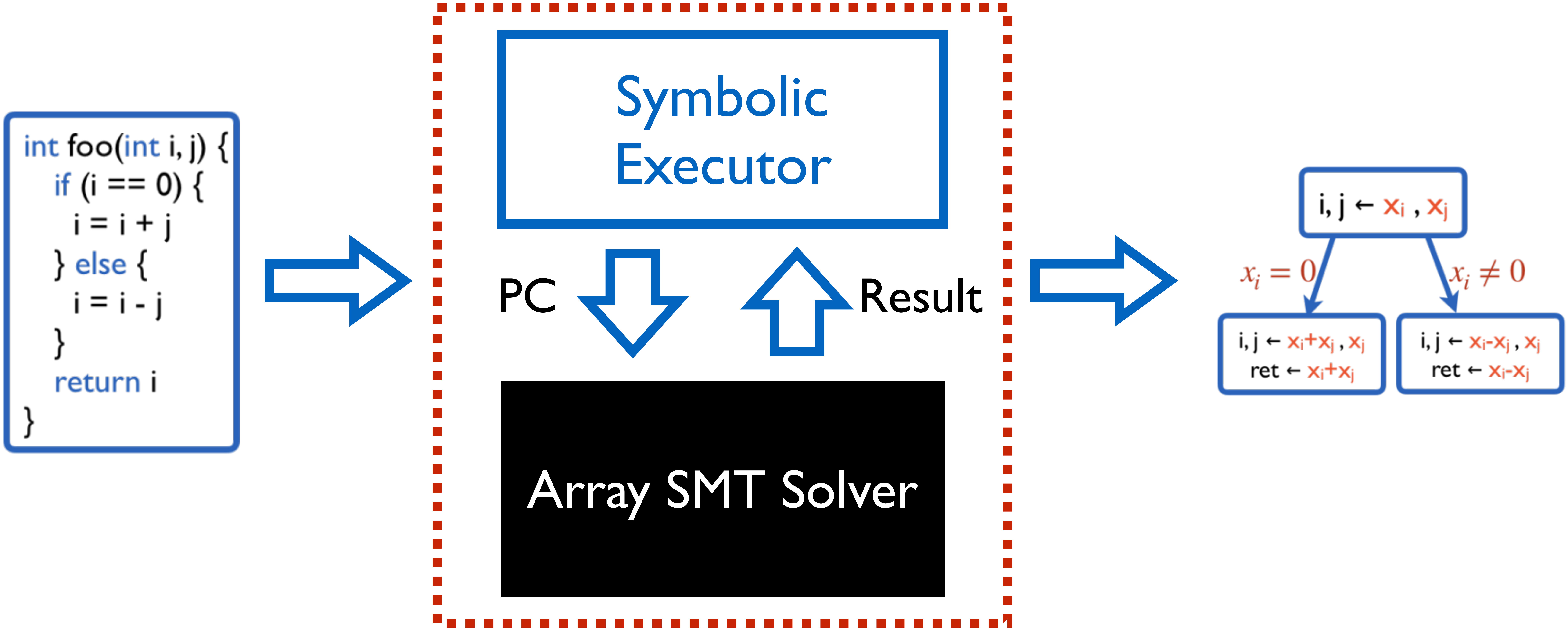
Memory modeling in SE

- Byte-level memory reasoning in symbolic execution
  - QF_ABV SMT theory
  - KLEE、S2E、…
- Every data is represented by a byte array
  - Many array variables in the path constraints
  - Large amount of axioms (O(n^2))

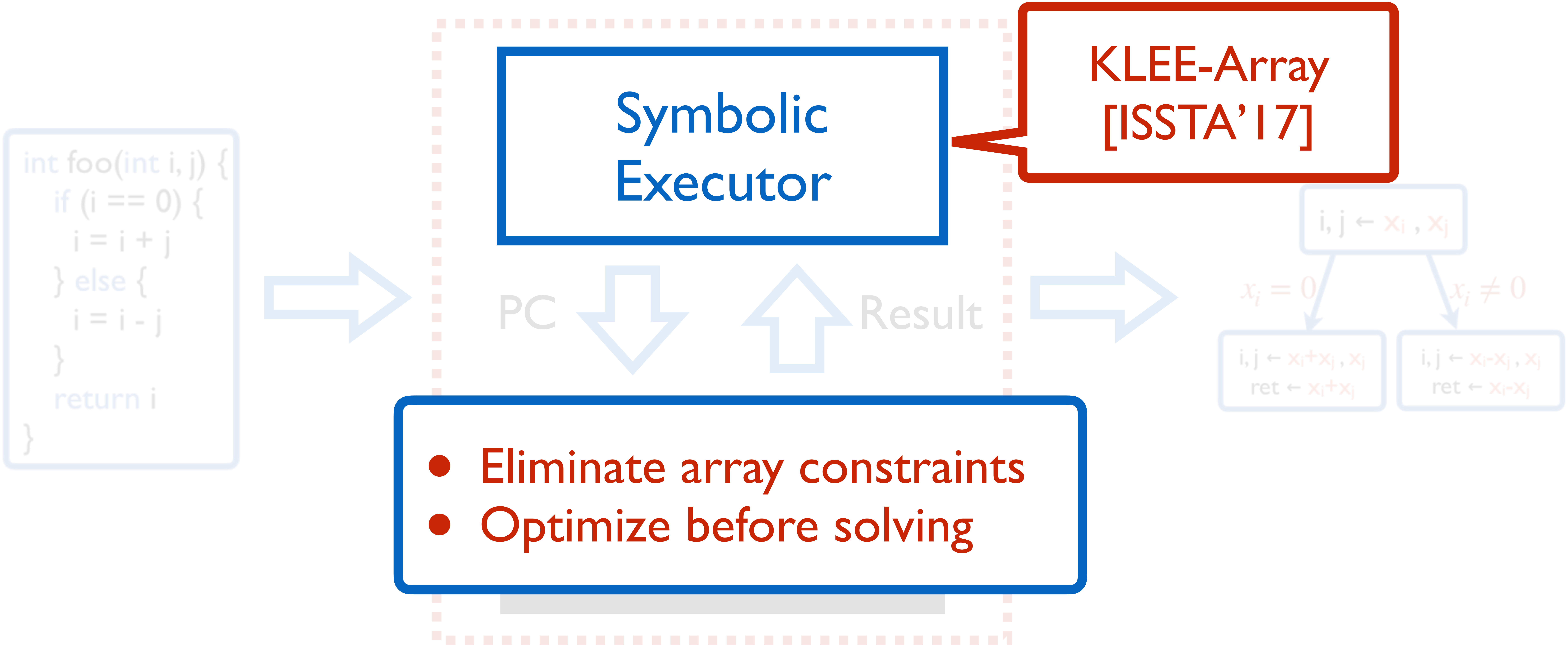
Problem

- Scalability of array constraint solving in symbolic execution
  - Byte-level array representation
  - Large number of axioms
  - …

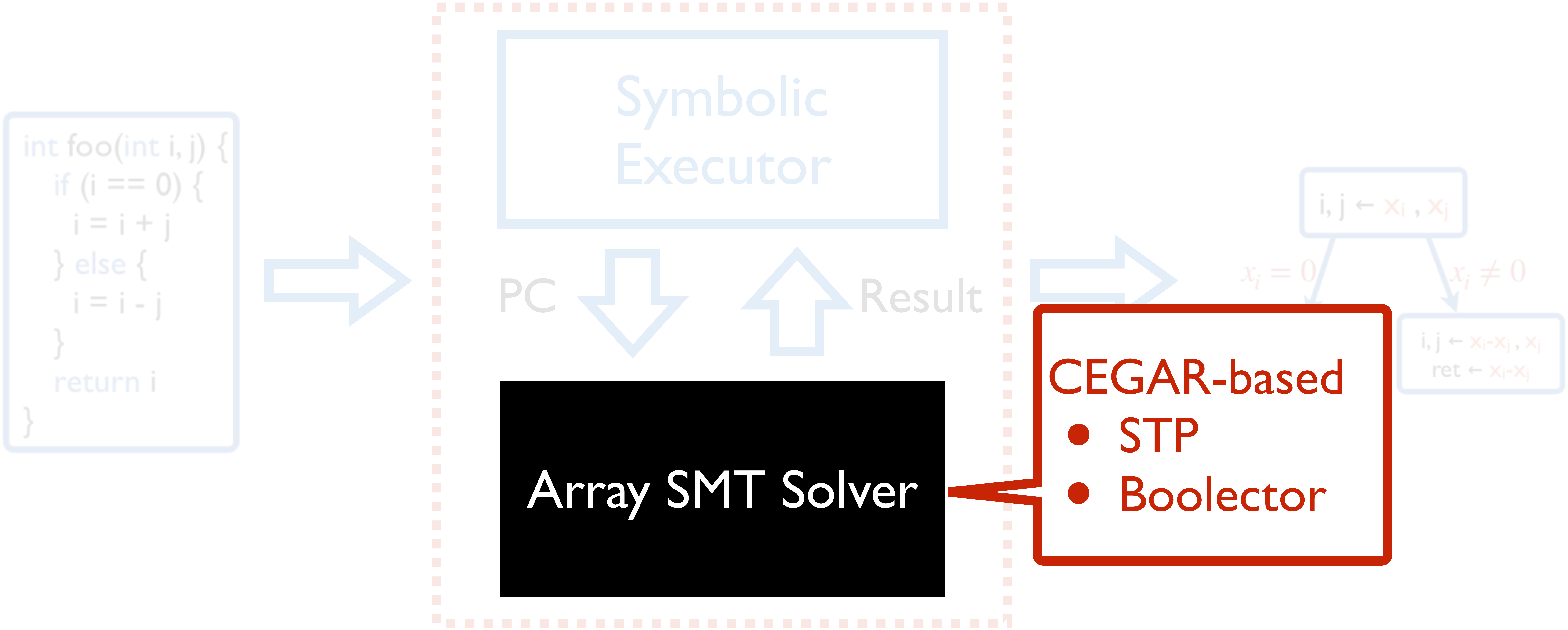
Related Work



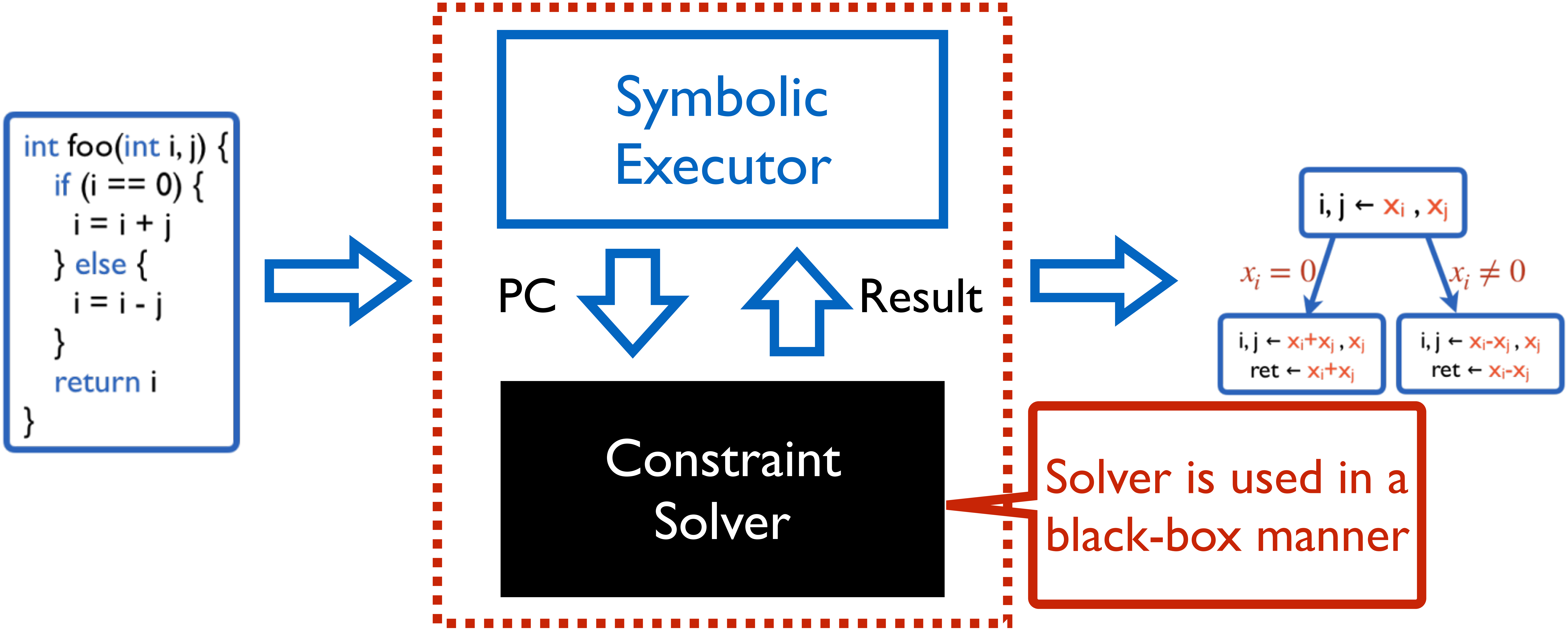
Related Work (1/2)



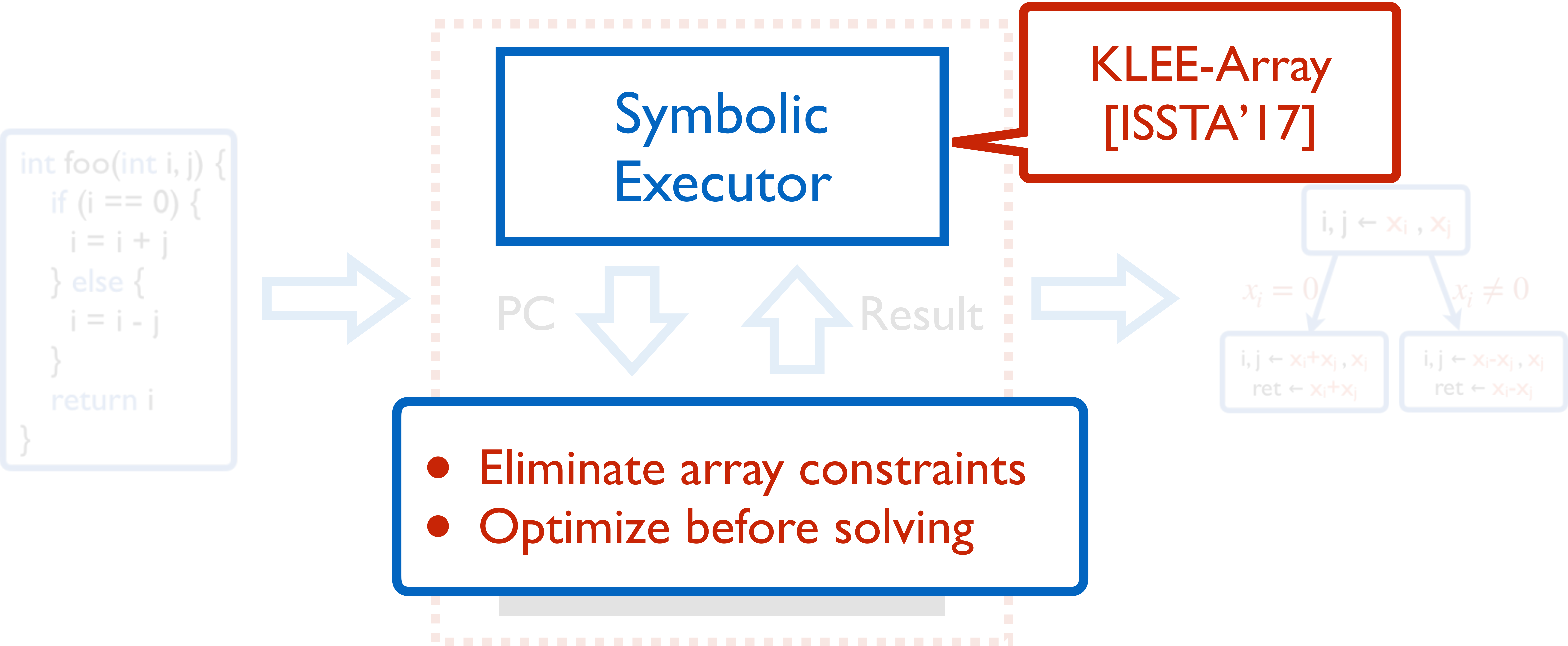
Related Work (2/2)



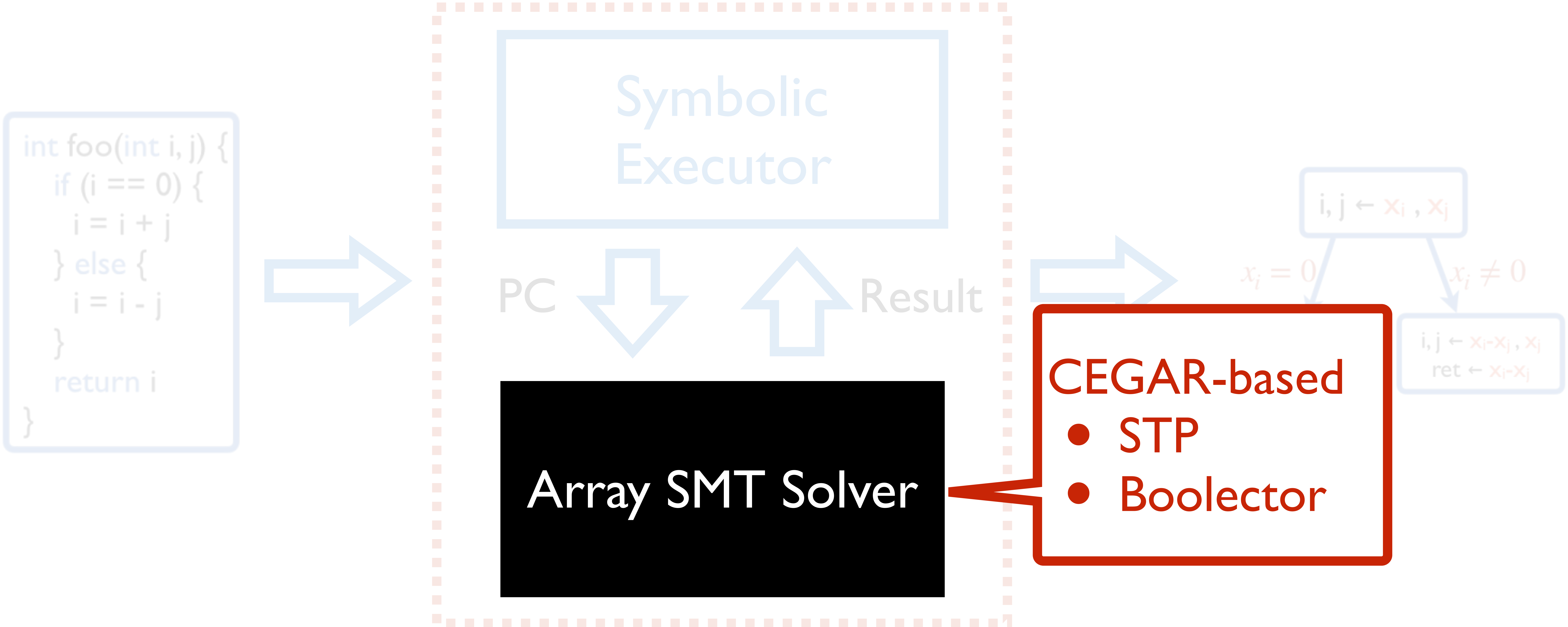
Related Work



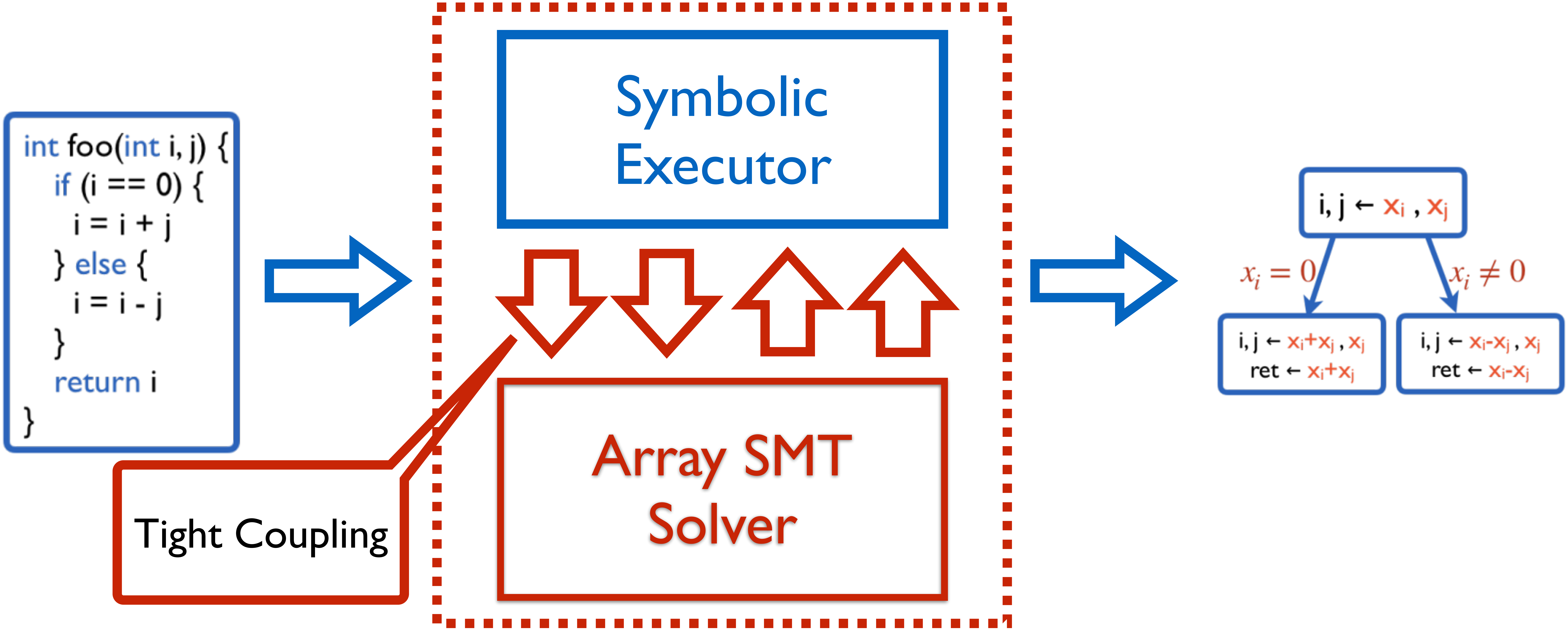
Related Work (1/2)



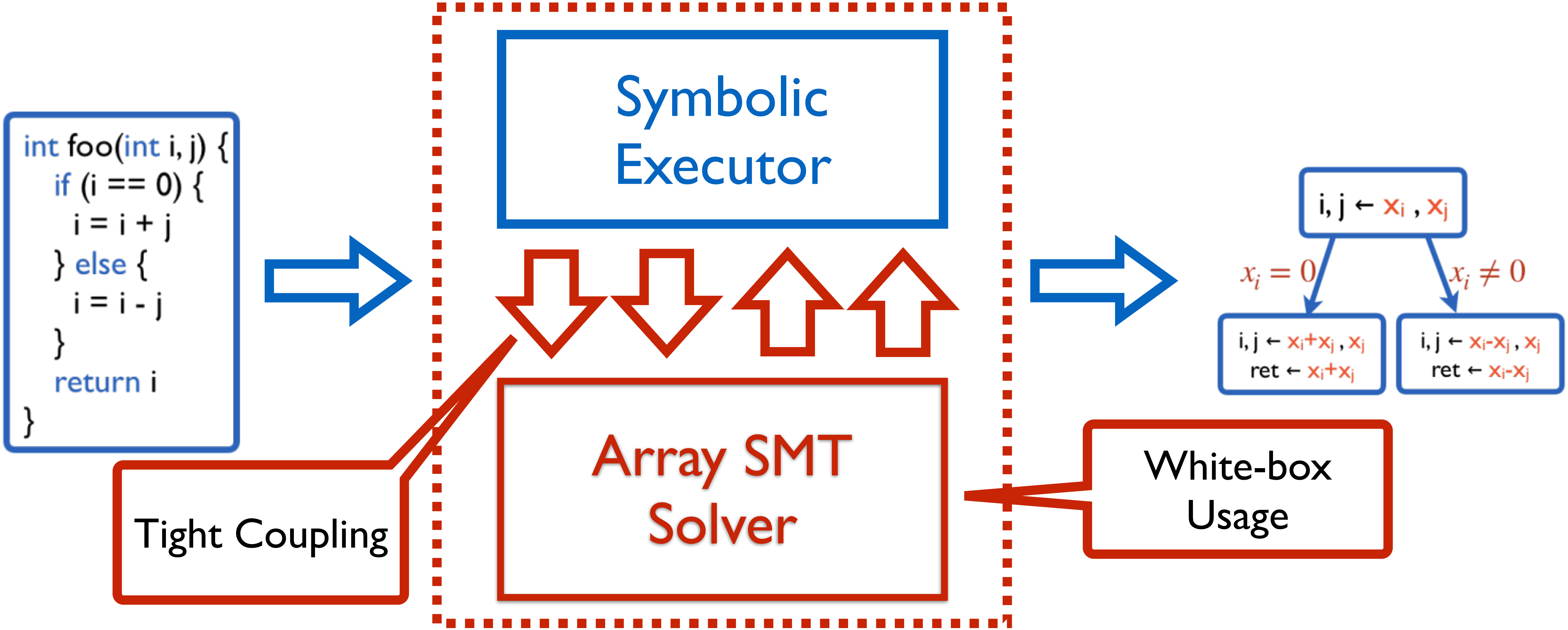
Related Work (2/2)



Our Argument



Our Argument



Our Key Insights

- Many redundant axioms exist for byte array constraints
  - Array access type information
  - Array index constraint

Our Key Insights

- Many redundant axioms exist for byte array constraints
  - Array access type information
  - Array index constraint
- Unsatisfiability can be decided earlier

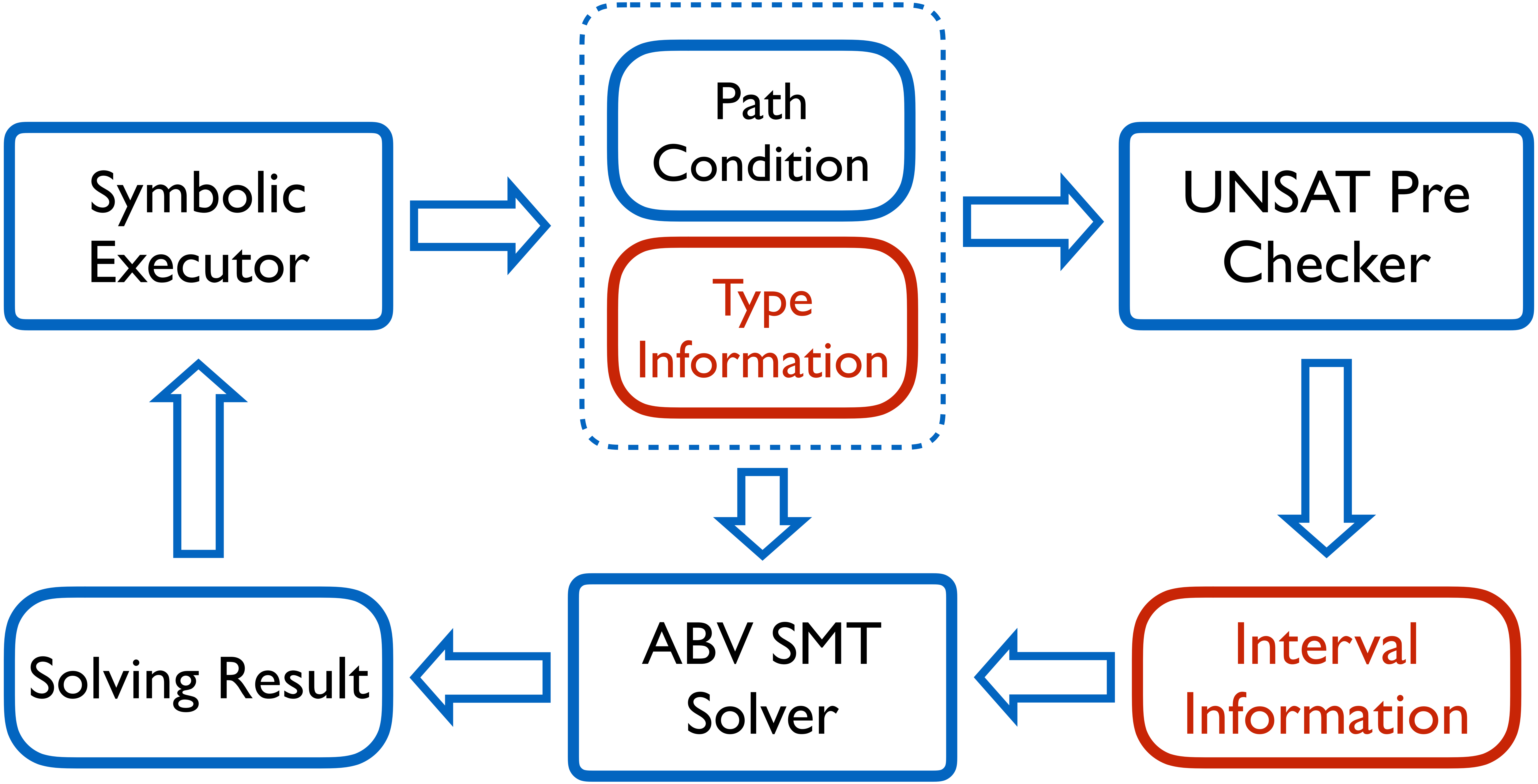
Our Key Idea

- Utilize the information calculated during symbolic execution
  - Type information of array accesses
  - Interval information of array index variables

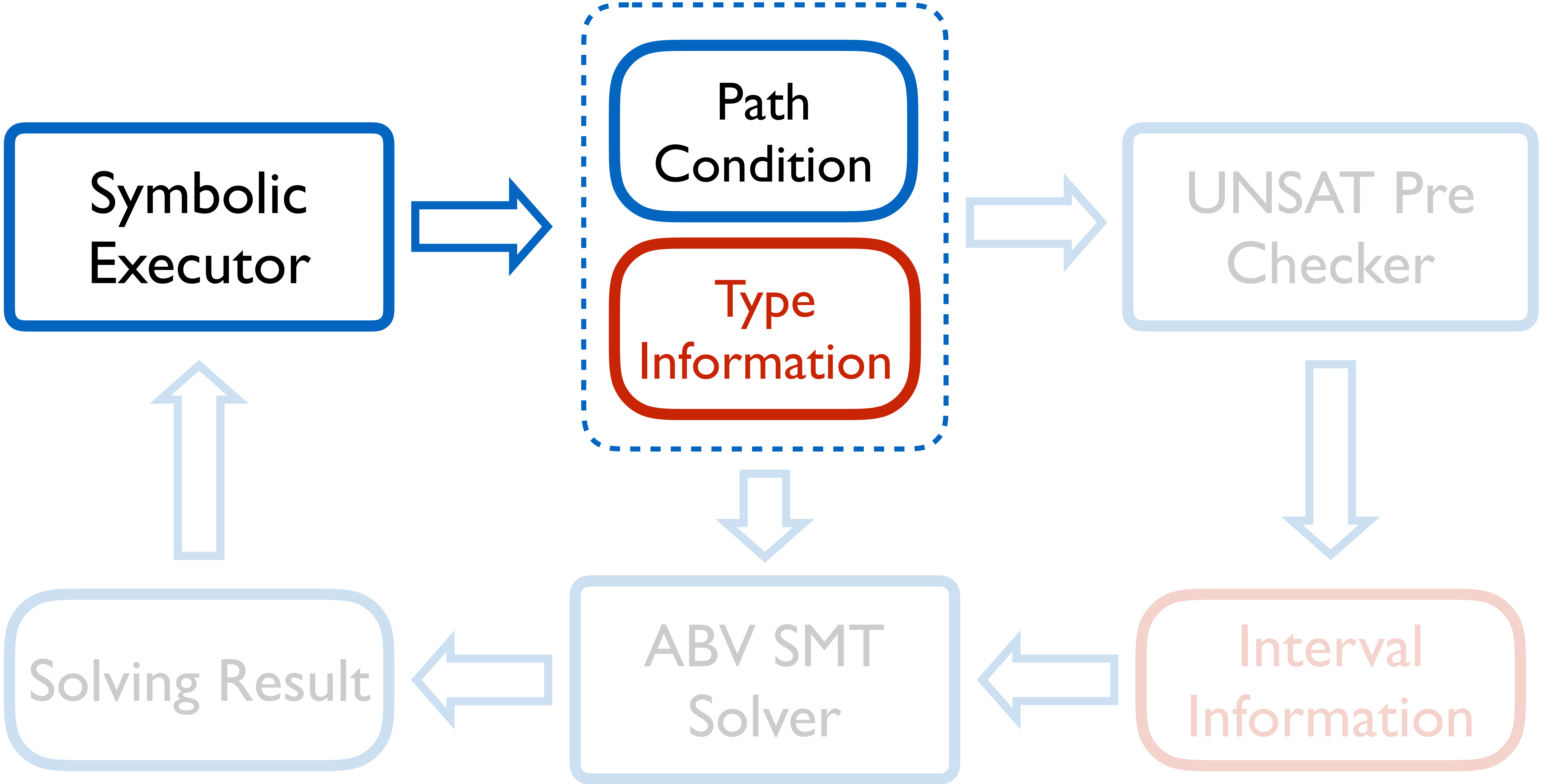
Our Key Idea

- Utilize the information calculated during symbolic execution
  - Type information of array accesses
  - Interval information of array index variables
- Check the unsatisfiability earlier
- Remove redundant axioms during solving

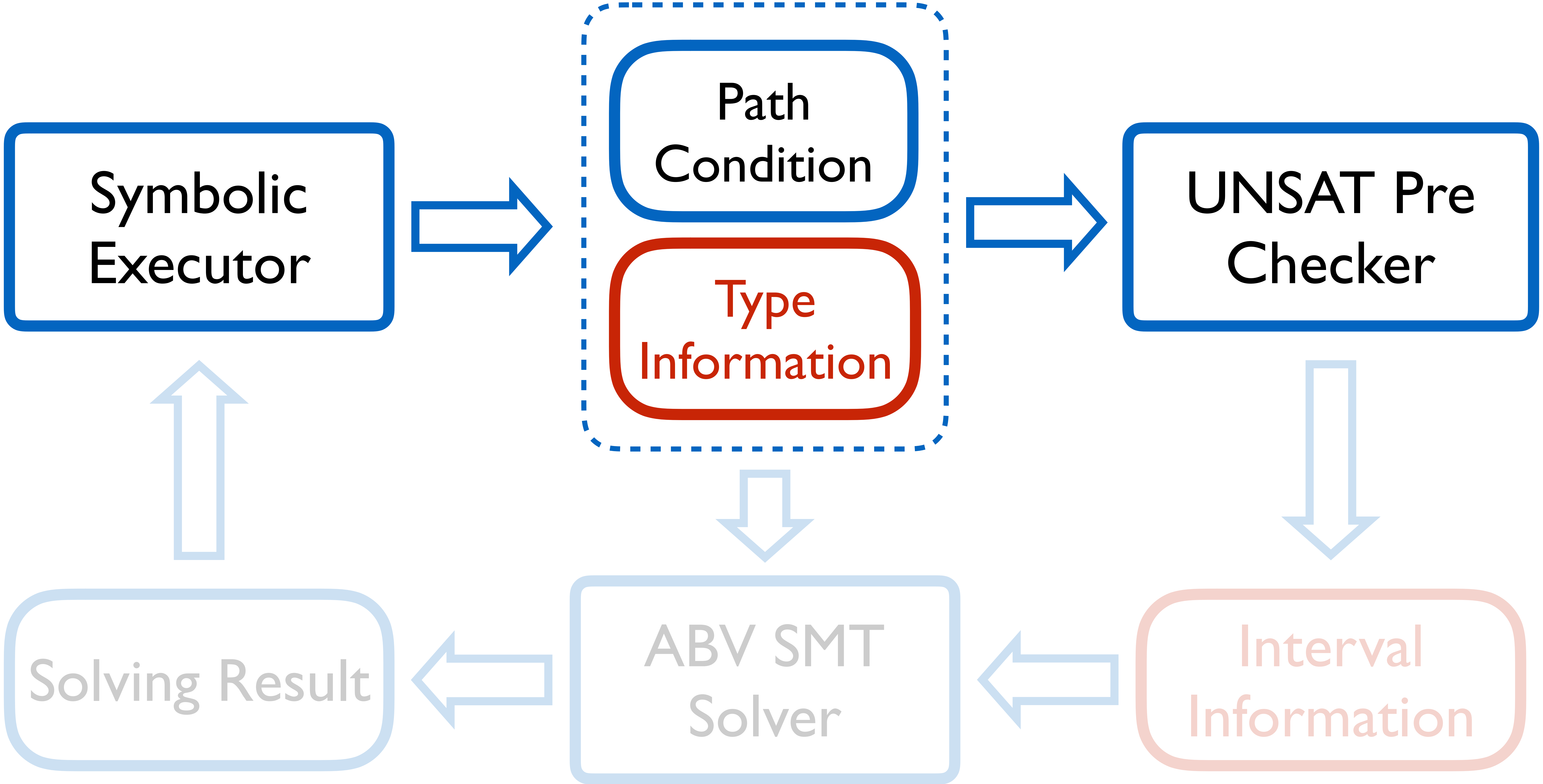
High-Level Procedure



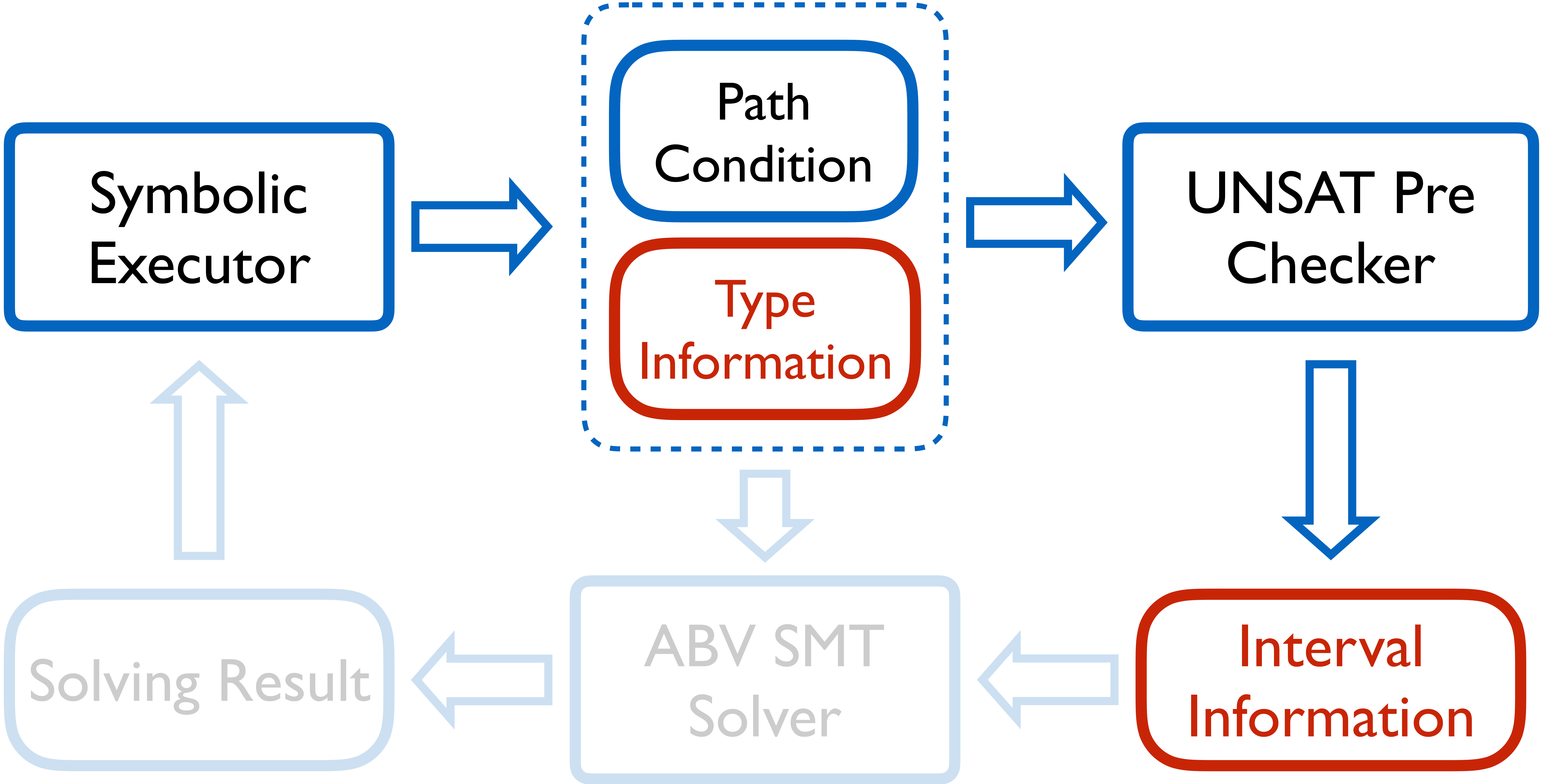
High-Level Procedure



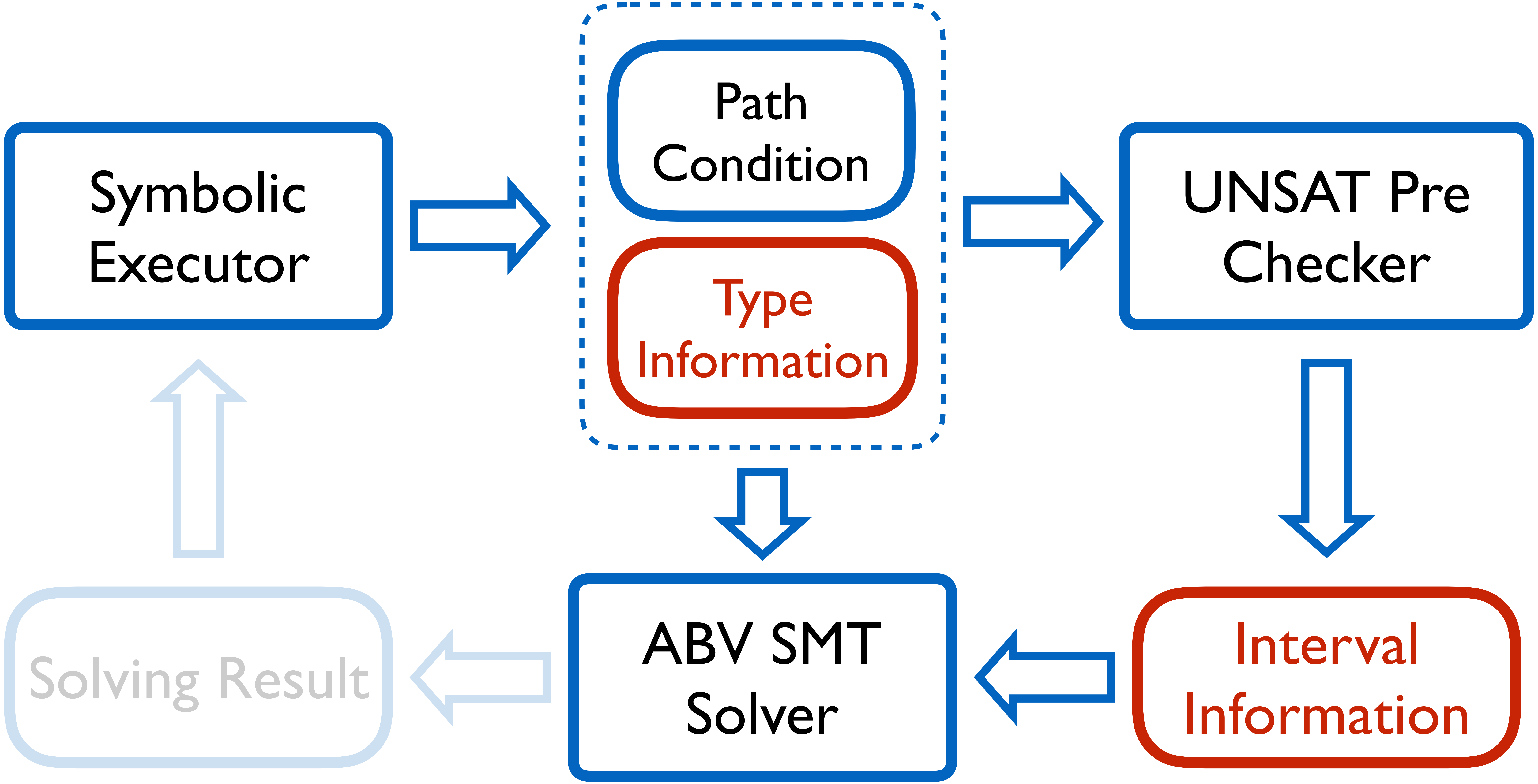
High-Level Procedure



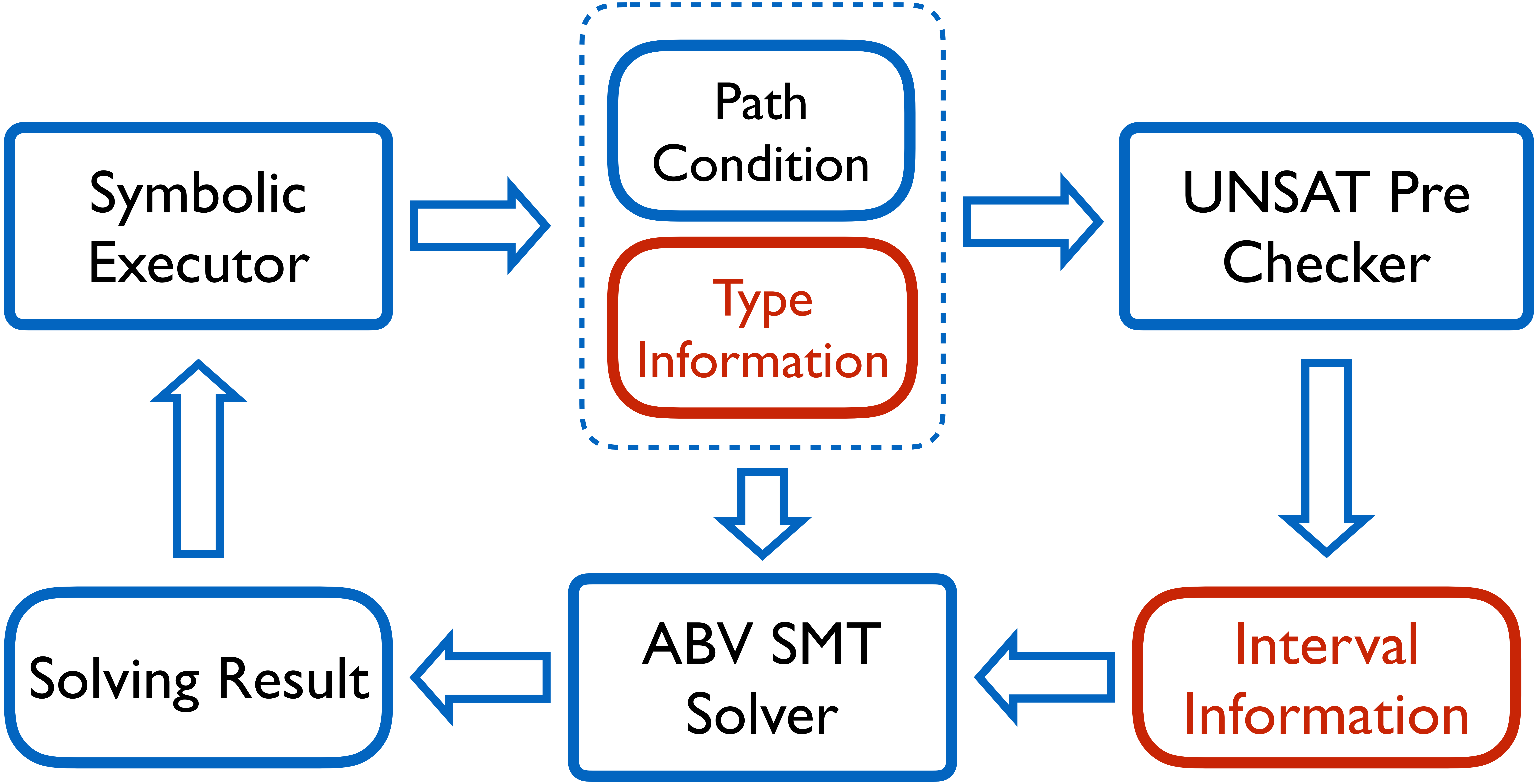
High-Level Procedure



High-Level Procedure



High-Level Procedure



Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
   int a[4] = {0, 0, 0, 5}
   if (i + j > 4) {
      if (a[i] + a[j] > 10) {
         printf("Bug!!!\n")
         return 1
      }
   }
   return 0
}
```

Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
  int a[4] = {0, 0, 0, 5}
  if (i + j > 4) {
    if (a[i] + a[j] > 10) {
      printf("Bug!!!\n")
      return 1
    }
  }
  return 0
}
```

$$0 \leq i \leq 3 \land 0 \leq j \leq 3 \land i + j > 4$$

$$\bigwedge$$

$$R_I(a, i) + R_I(a, j) > 10$$

$$a[4] = \{0, 0, 0, 5\}$$

Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
  int a[4] = {0, 0, 0, 5}
  if (i + j > 4) {
    if (a[i] + a[j] > 10) {
      printf("Bug!!!\n")
      return 1
    }
  }
  return 0
}
```

UNSAT Pre-check

Index Constraints

$$0 \leq i \leq 3 \wedge 0 \leq j \leq 3 \wedge i + j > 4$$

$$\bigwedge$$

Array Constraint

$$R_I(a, i) + R_I(a, j) > 10$$

$$a[4] = \{0, 0, 0, 5\}$$

Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
  int a[4] = {0, 0, 0, 5}
  if (i + j > 4) {
    if (a[i] + a[j] > 10) {
      printf("Bug!!!\n")
      return 1
    }
  }
  return 0
}
```

UNSAT Pre-check

$0 \leq i \leq 3 \wedge 0 \leq j \leq 3 \wedge i + j > 4$

Index Constraints

ILP

$2 \leq i \leq 3 \wedge 2 \leq j \leq 3$

Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
  int a[4] = {0, 0, 0, 5}
  if (i + j > 4) {
    if (a[i] + a[j] > 10) {
      printf("Bug!!!\n")
      return 1
    }
  }
  return 0
}
```

UNSAT Pre-check

Index Constraints

$$0 \leq i \leq 3 \land 0 \leq j \leq 3 \land i + j > 4$$

ILP

$$2 \leq i \leq 3 \land 2 \leq j \leq 3$$

$a[4] = \{0, 0, 0, 5\}$

$$0 \leq R_I(a, i) \leq 5 \land 0 \leq R_I(a, j) \leq 5$$

Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
   int a[4] = {0, 0, 0, 5}
   if (i + j > 4) {
      if (a[i] + a[j] > 10) {
         printf("Bug!!!\n")
         return 1
      }
   }
   return 0
}
```

UNSAT Pre-check

- An over-approximation

$$0 \leq R_I(a, i) \leq 5 \wedge 0 \leq R_I(a, j) \leq 5$$

$$\bigwedge$$

$$R_I(a, i) + R_I(a, j) > 10$$

ILP

Unsatisfiable!!!

Motivation Example

$i, j \in [0, 3]$

```
int foo(int i, j) {
   int a[4] = {0, 0, 0, 9}
   if (i + j > 4) {
      if (a[i] + a[j] > 10) {
         printf("Bug!!!\n")
         return 1
      }
   }
   return 0
}
```

UNSAT Pre-check

- An over-approximation

$$0 \leq R_I(a, i) \leq 9 \wedge 0 \leq R_I(a, j) \leq 9$$

$$\bigwedge$$

$$R_I(a, i) + R_I(a, j) > 10$$

ILP

Satisfiable??? NO

Motivation Example

i, j ∈ [0, 3]

```
int foo(int i, j) {
  int a[4] = {0, 0, 0, 9}
  if (i + j > 4) {
    if (a[i] + a[j] > 10) {
      printf("Bug!!!\n")
      return 1
    }
  }
  return 0
}
```

Axiom Elimination

- Interval info computed in pre-check

$$0 \leq i \leq 3 \land 0 \leq j \leq 3 \land i + j > 4$$

ILP

$$2 \leq i \leq 3 \land 2 \leq j \leq 3$$

- Type info collected in SE (int)

Motivation Example

Array Memory Layout

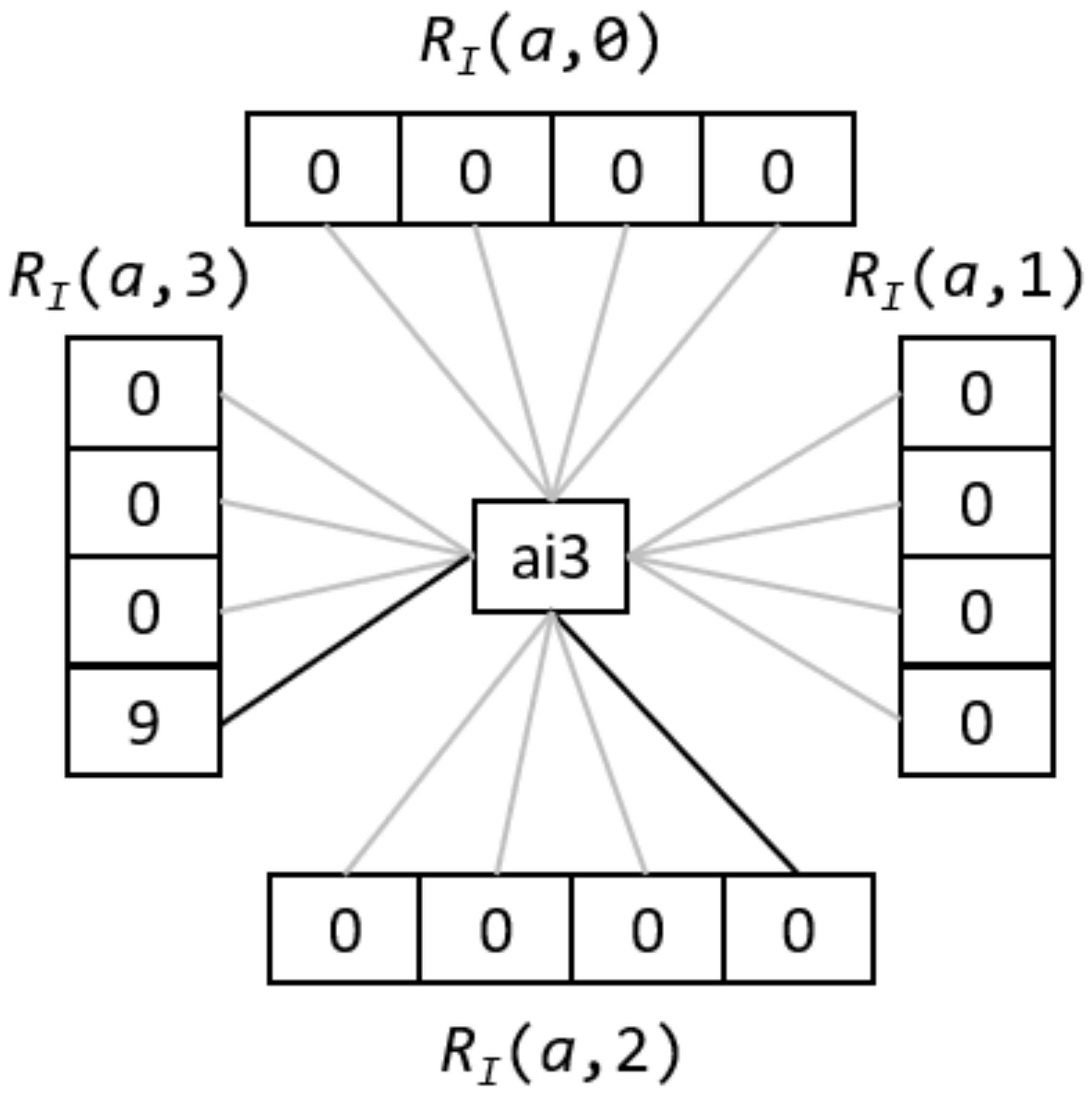
offset:	0	1	2	3
$R_I(a,i)$	ai0	ai1	ai2	ai3
$R_I(a,j)$	aj0	aj1	aj2	aj3
$R_I(a,0)$	0	0	0	0
$R_I(a,1)$	0	0	0	0
$R_I(a,2)$	0	0	0	0
$R_I(a,3)$	0	0	0	9

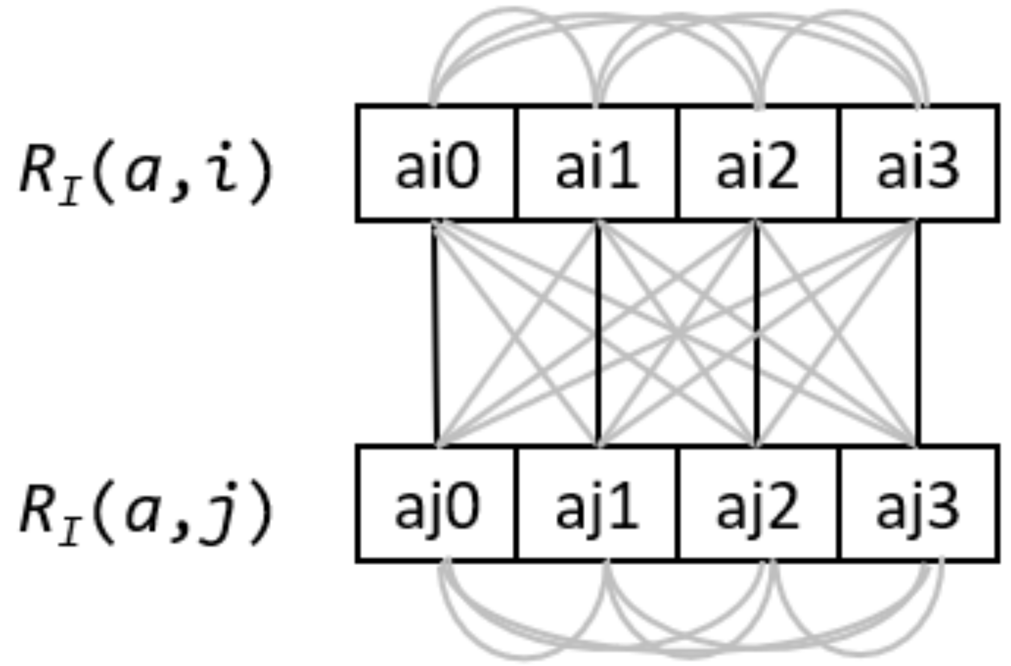
156 axioms

Motivation Example

Array Memory Layout

offset:	0	1	2	3
$R_I(a,i)$	ai0	ai1	ai2	ai3
$R_I(a,j)$	aj0	aj1	aj2	aj3
$R_I(a,0)$	0	0	0	0
$R_I(a,1)$	0	0	0	0
$R_I(a,2)$	0	0	0	0
$R_I(a,3)$	0	0	0	9





156
↓
20
axioms

Use type & interval info to remove axioms (gray lines)

- Bytes that have different offsets in the type (int)
- Bytes within the interval and any byte outside of the interval

Type Inference

$$1:\ \dfrac{\begin{array}{c} s = \textbf{var}\ v[e] : \mathrm{T} \\ (\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M}',\mathcal{G}) \quad u = \sigma(v) + \sigma(e) \times \mathcal{S}(\mathrm{T}) - 1 \end{array}}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,s} (\mathcal{M}'[v \leftarrow \mathcal{S}(\mathrm{T})], \mathcal{G}[v \leftarrow [\sigma(v), u]])}$$

$$2:\ \dfrac{s = v := e \quad (\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M}',\mathcal{G})}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,s} (\mathcal{M}',\mathcal{G})}$$

$$3:\ \dfrac{s = *v := e \quad (\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M}',\mathcal{G})}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,s} (\mathcal{M}',\mathcal{G})}$$

$$1:\ \dfrac{e = c}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M},\mathcal{G})} \qquad 2:\ \dfrac{e = v}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M},\mathcal{G})}$$

$$3:\ \dfrac{\begin{array}{c} e = (\mathrm{T}*)e_1 \quad \sigma(e_1) \in \mathcal{G}(v) \\ (\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e_1} (\mathcal{M}',\mathcal{G}) \quad s_1 = \min(\mathcal{S}(\mathrm{T}), \mathcal{M}'(v)) \end{array}}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M}'[v \leftarrow s_1], \mathcal{G})}$$

$$4:\ \dfrac{e = *e_1 \quad (\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e_1} (\mathcal{M}',\mathcal{G})}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M}',\mathcal{G})}$$

$$5:\ \dfrac{e = e_1 \oplus e_2 \quad (\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e_1} (\mathcal{M}_1,\mathcal{G}) \quad (\mathcal{M}_1,\mathcal{G}) \xrightarrow{\sigma,e_2} (\mathcal{M}_2,\mathcal{G})}{(\mathcal{M},\mathcal{G}) \xrightarrow{\sigma,e} (\mathcal{M}_2,\mathcal{G})}$$

Reserve minimum type size of array accesses

Index Constraint Abstraction

- Index constraint is a conjunction $\bigwedge_{i=1}^{n} c_i$

Abstraction

- Discard c_i that cannot be linearized
- Linearize c_i with complex operator

- Translate bit-vector index constraint to ILP problem

The abstraction rules ensure over-approximation

Other Internals

- Two simplifications to reduce cost of ILP solving
  - Simple interval computation before linearization
  - Caching ILP solutions

Evaluation

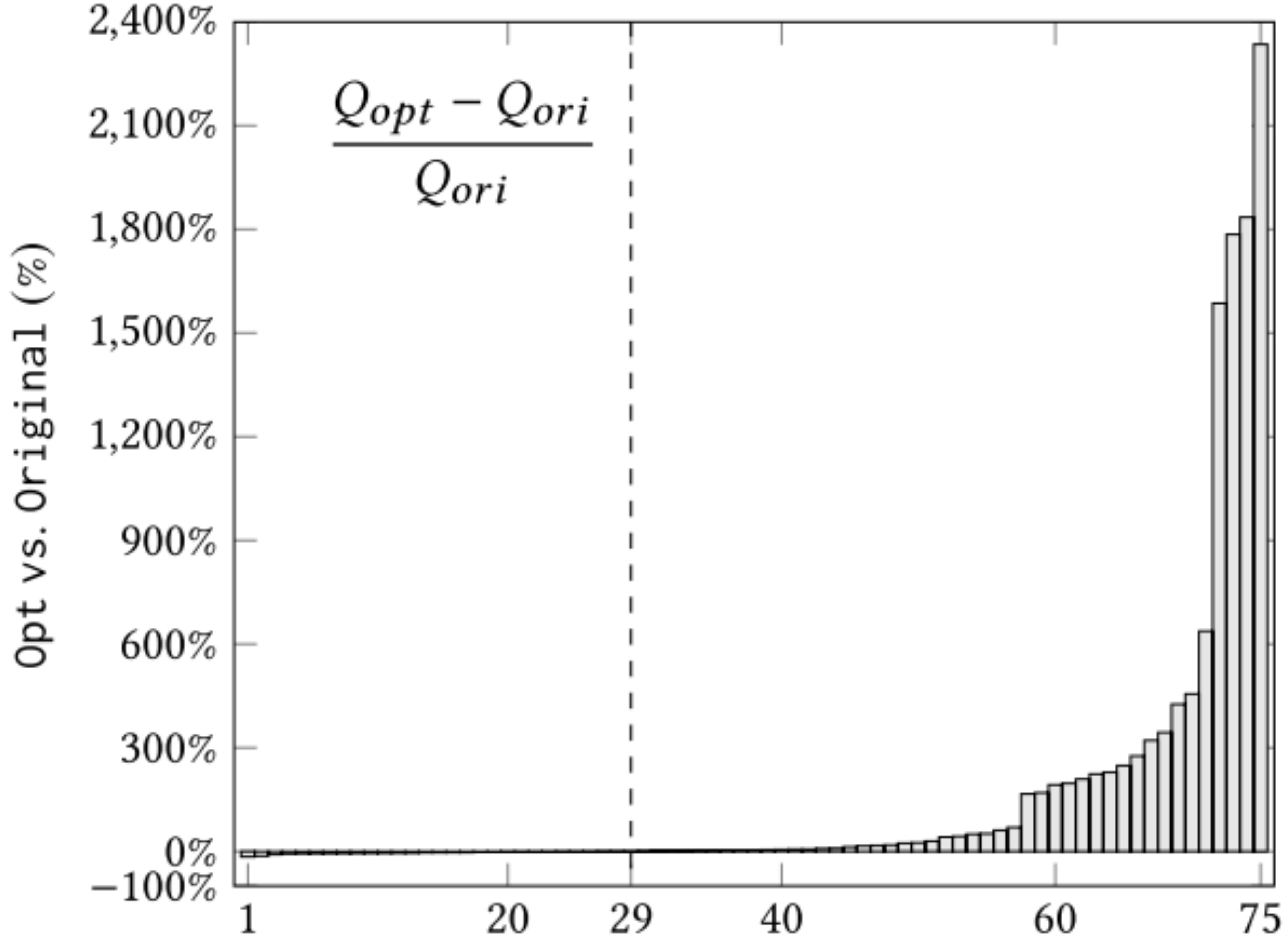
- Research Questions
  - Effectiveness
  - Relevance of either optimization
  - Comparison with KLEE-Array

Evaluation

- Implementation
  - KLEE with STP
  - PPL solver for ILP solving
- Real-world programs as benchmark
  - Coreutils programs (62)
  - Lexer programs of various grammars (13)

Results of Effectiveness

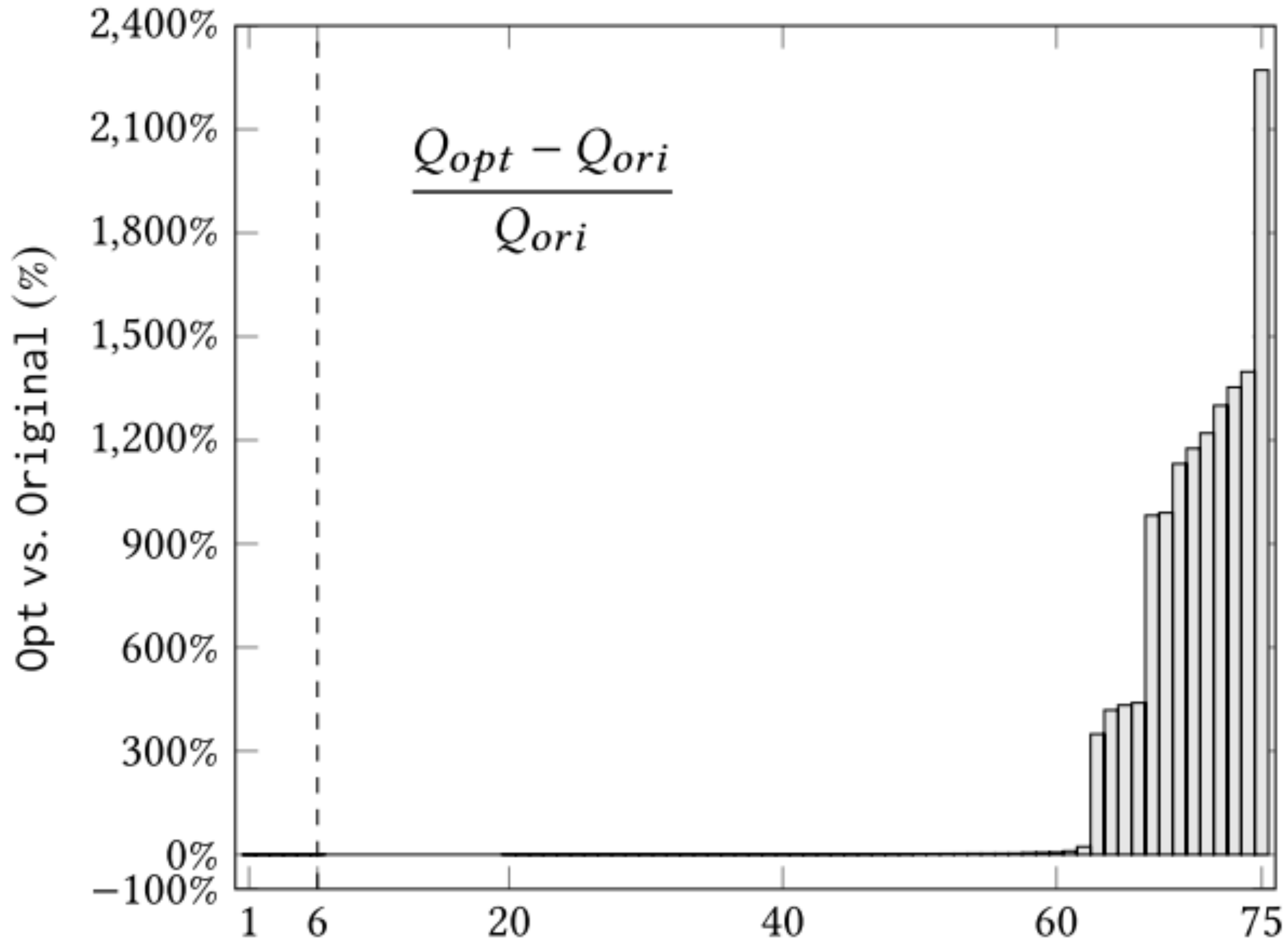
Queries without KLEE opt



Improves the queries for 46 programs, 160.52% on average

Results of Effectiveness

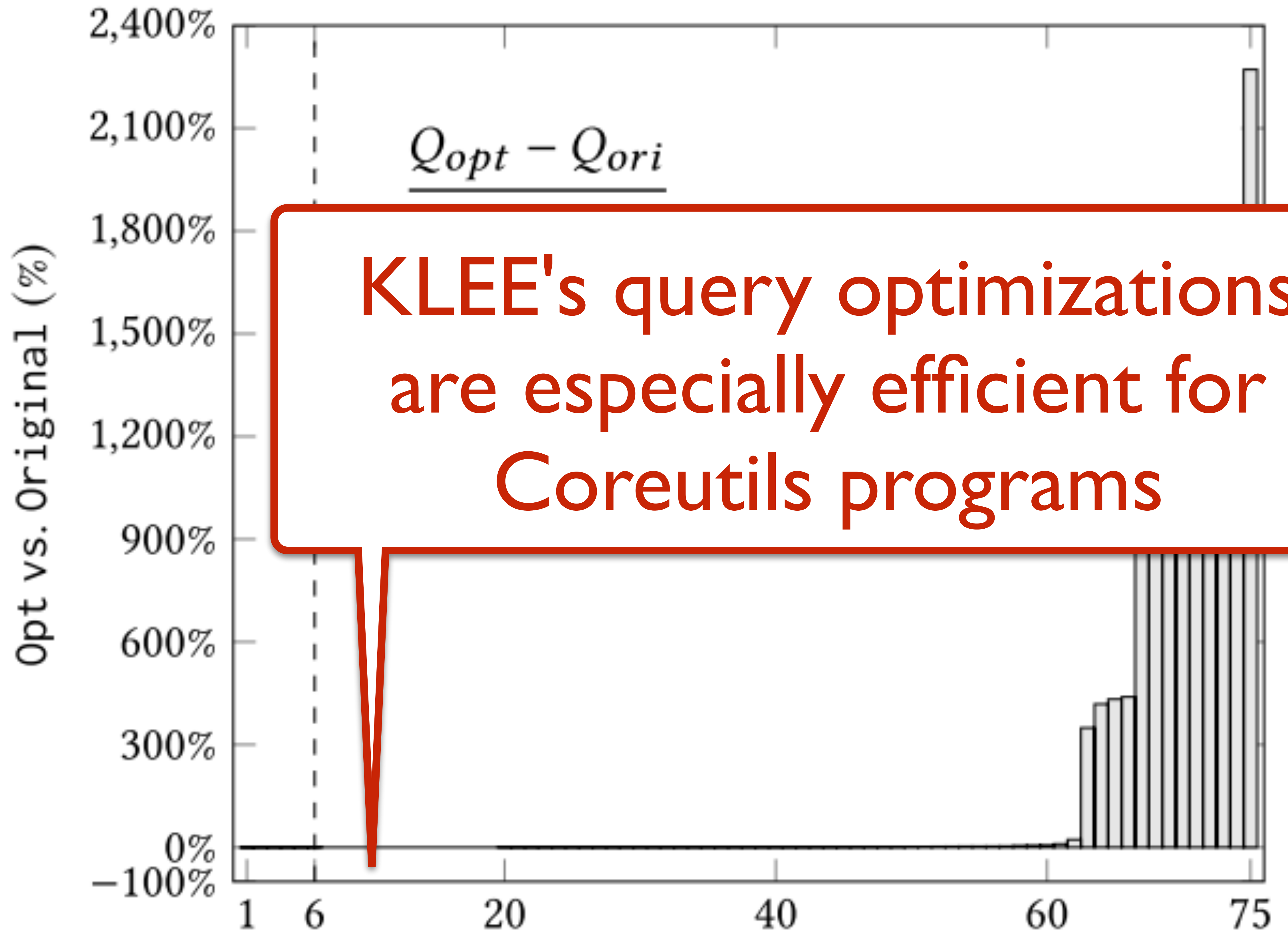
Queries with KLEE opt



Improves the queries for 56 programs, 182.56% on average

Results of Effectiveness

Queries with KLEE opt

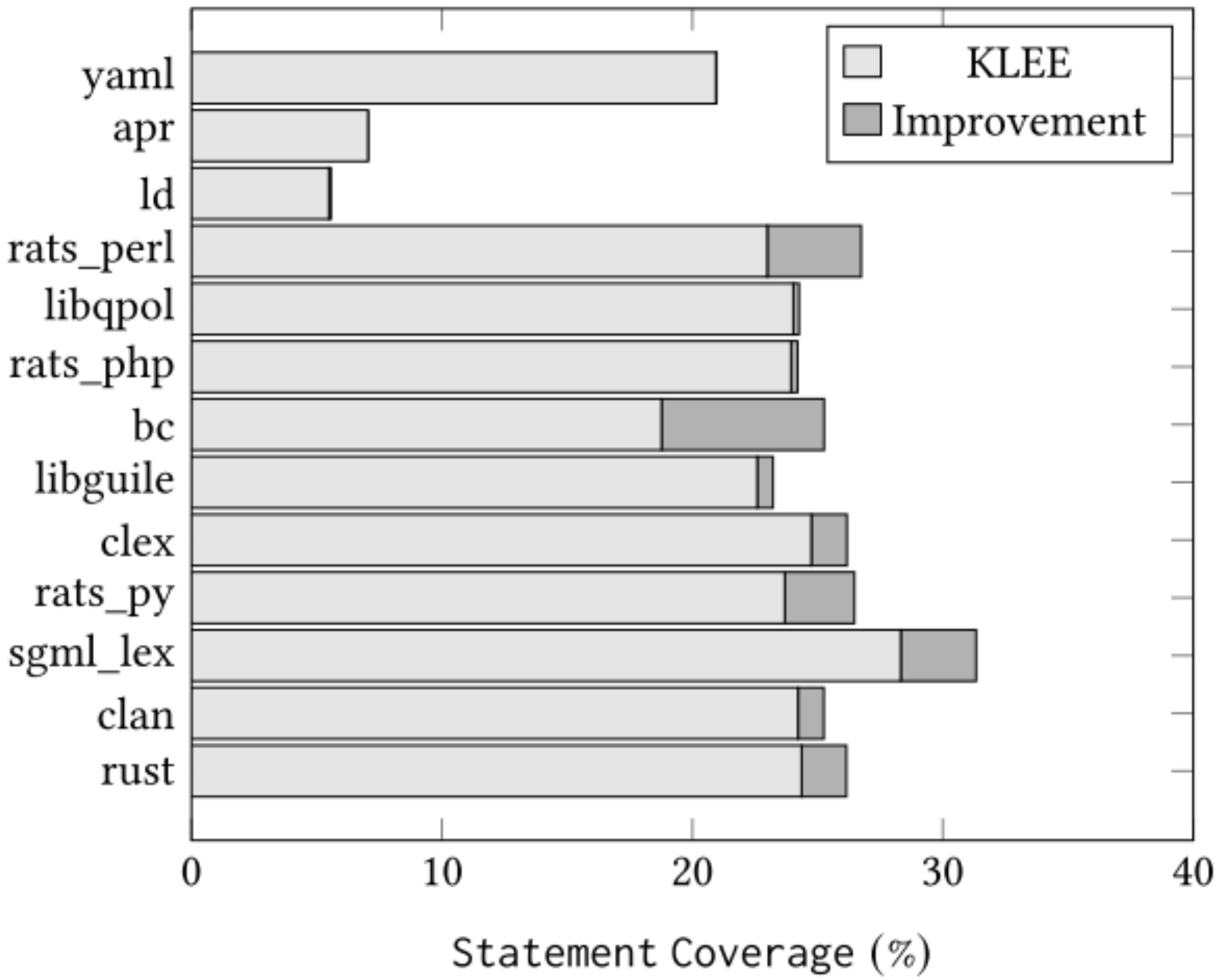


KLEE's query optimizations are especially efficient for Coreutils programs

Improves the queries for 56 programs, 182.56% on average

Results of Effectiveness

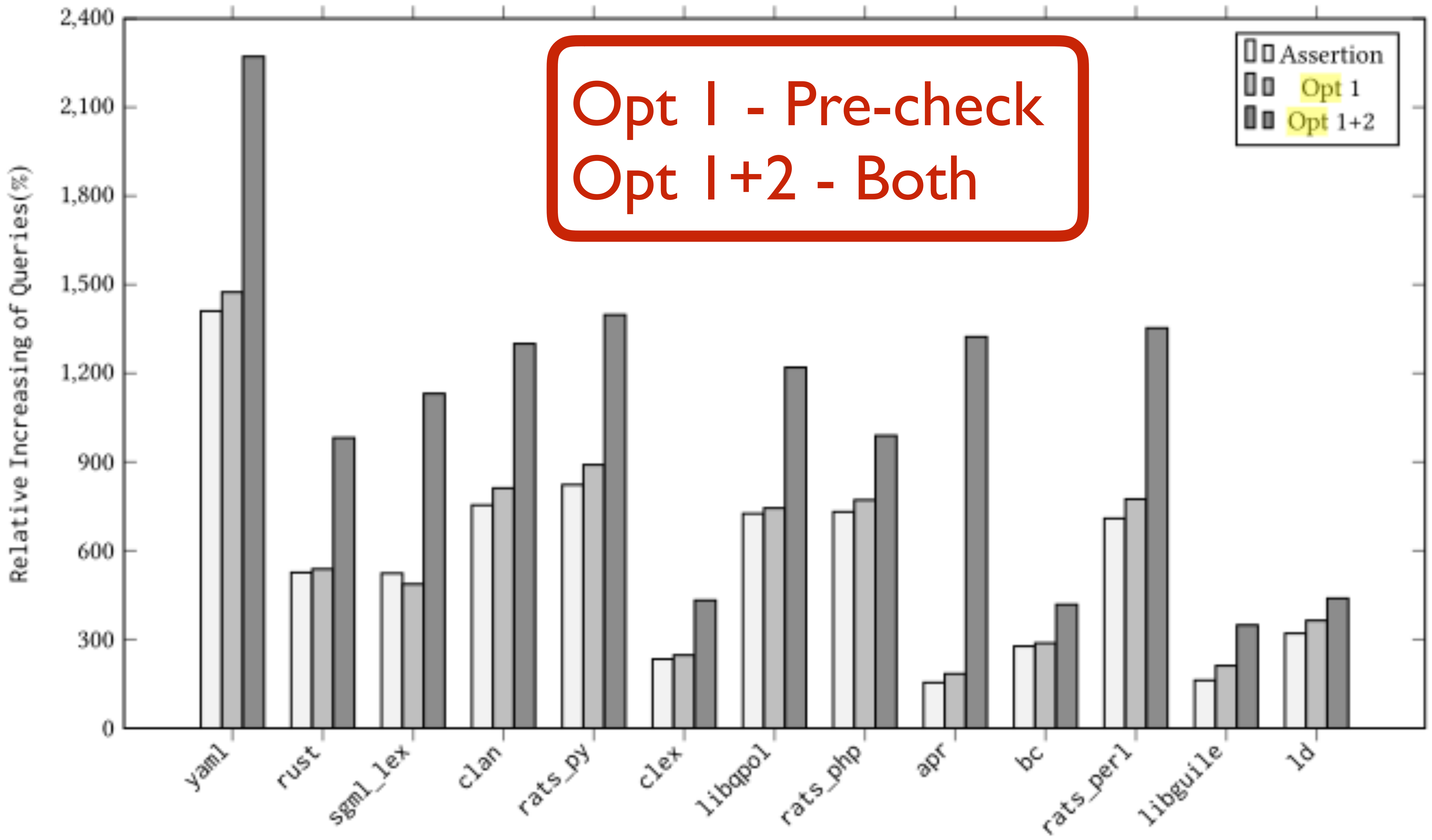
Coverage with KLEE opt



The advancement in constraint solving can directly benefit SE

Results of Relevance

Queries with KLEE opt



Opt 1 - Pre-check
Opt 1+2 - Both

Opt 2 is more significant, while Opt 1 can generate useful information for Opt 2

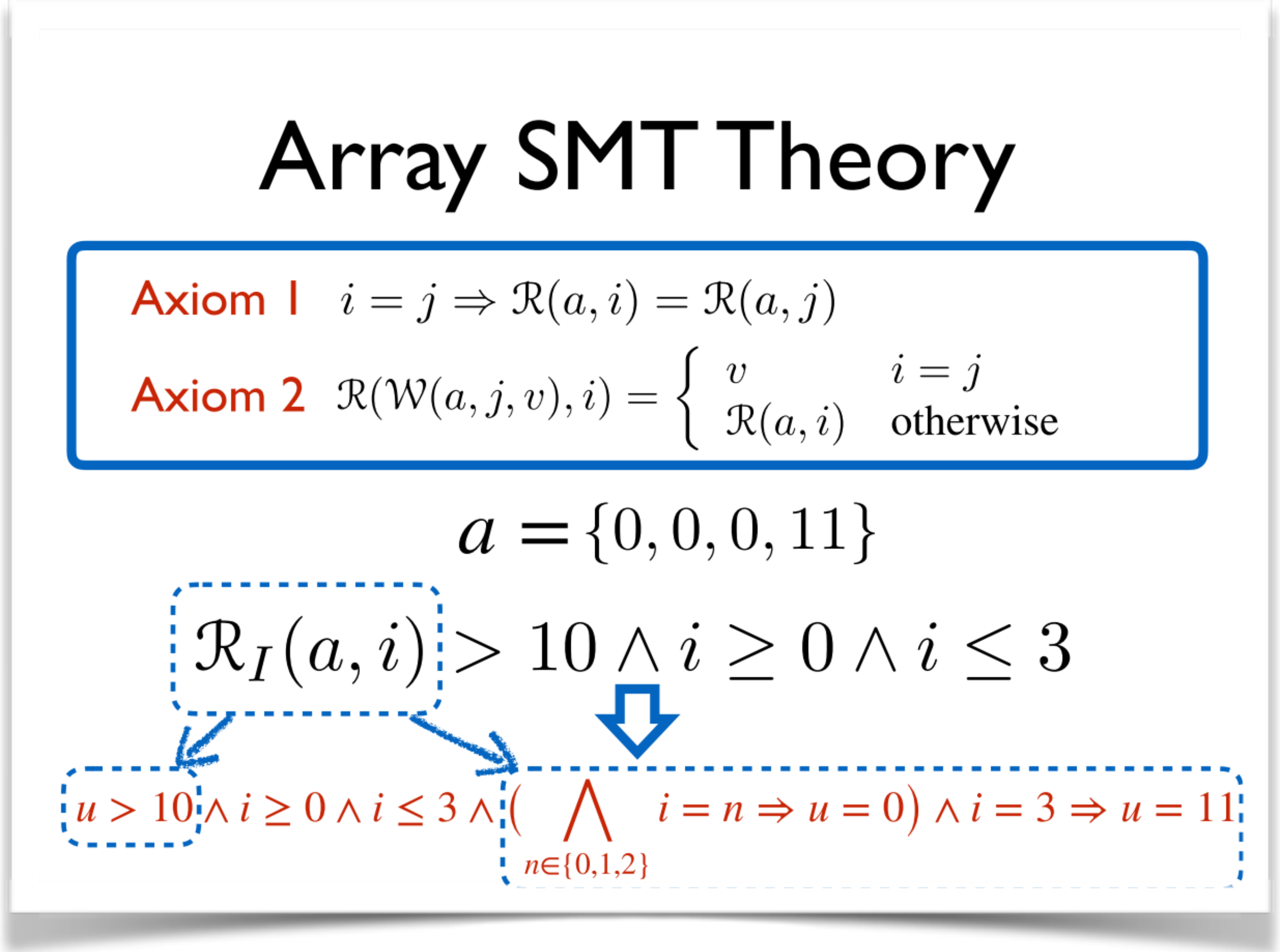
Comparison with KLEE-Array

With KLEE opt

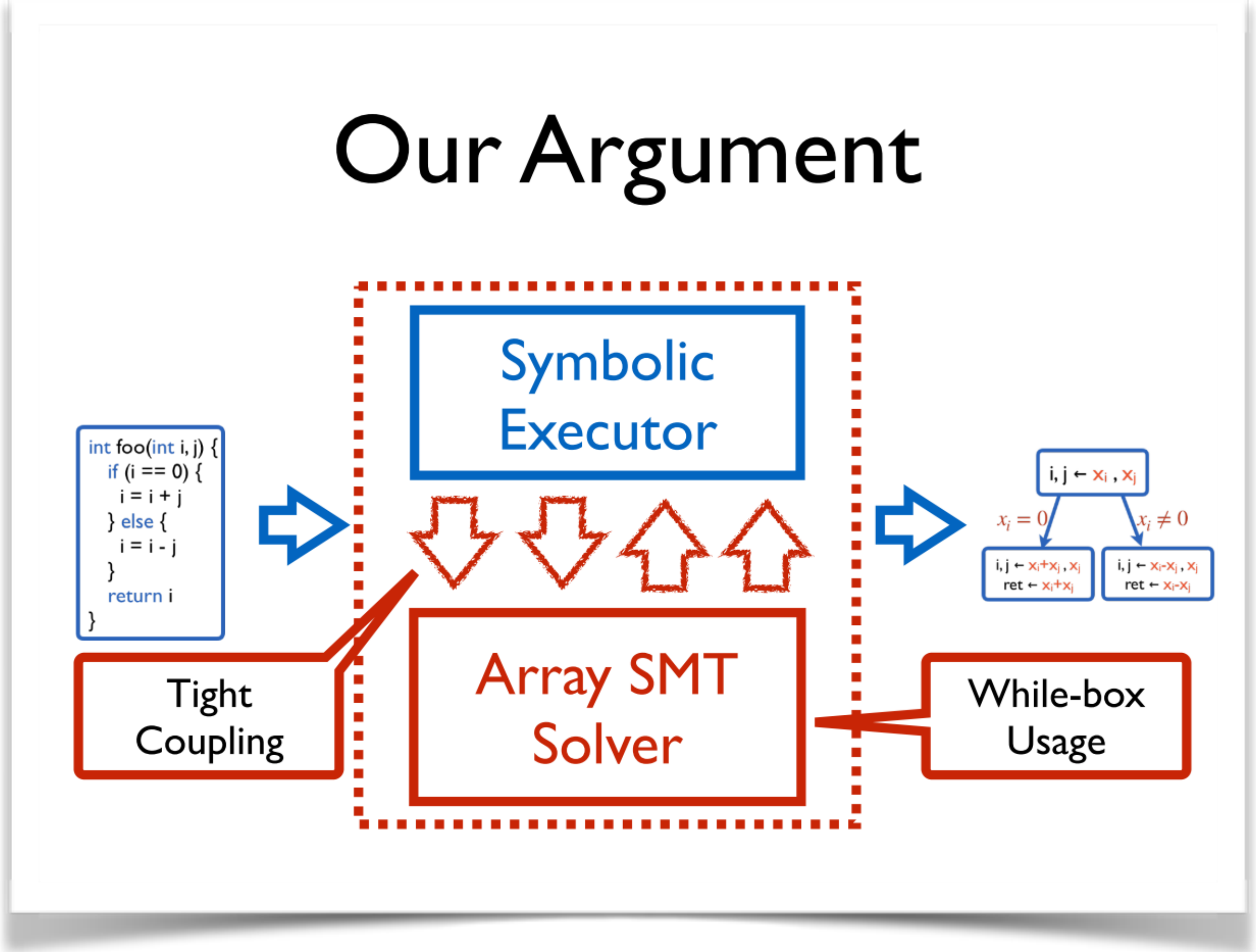
Programs	KLEE-Array		Our Method	
	#Instrs	#Paths	#Instrs	#Paths
yaml	71687	29	63864	28
rust	38892	24	53921	38
sgml_lex	599397	184	523956	165
clan	69777	66	89288	86
rats_py	353230	342	417394	401
clex	87322	87	115455	124
libqpol	35871	22	45190	35
rats_php	5221268	1554	14514660	4479
apr	637629	3456	880674	5542
bc	340874	36	440008	43
rats_perl	325398	338	379466	402
libguile	665723	337	750713	421
ld	373181619	489	373304921	584

Our method increases the number of paths and instructions by 30.31% and 40.39%, respectively

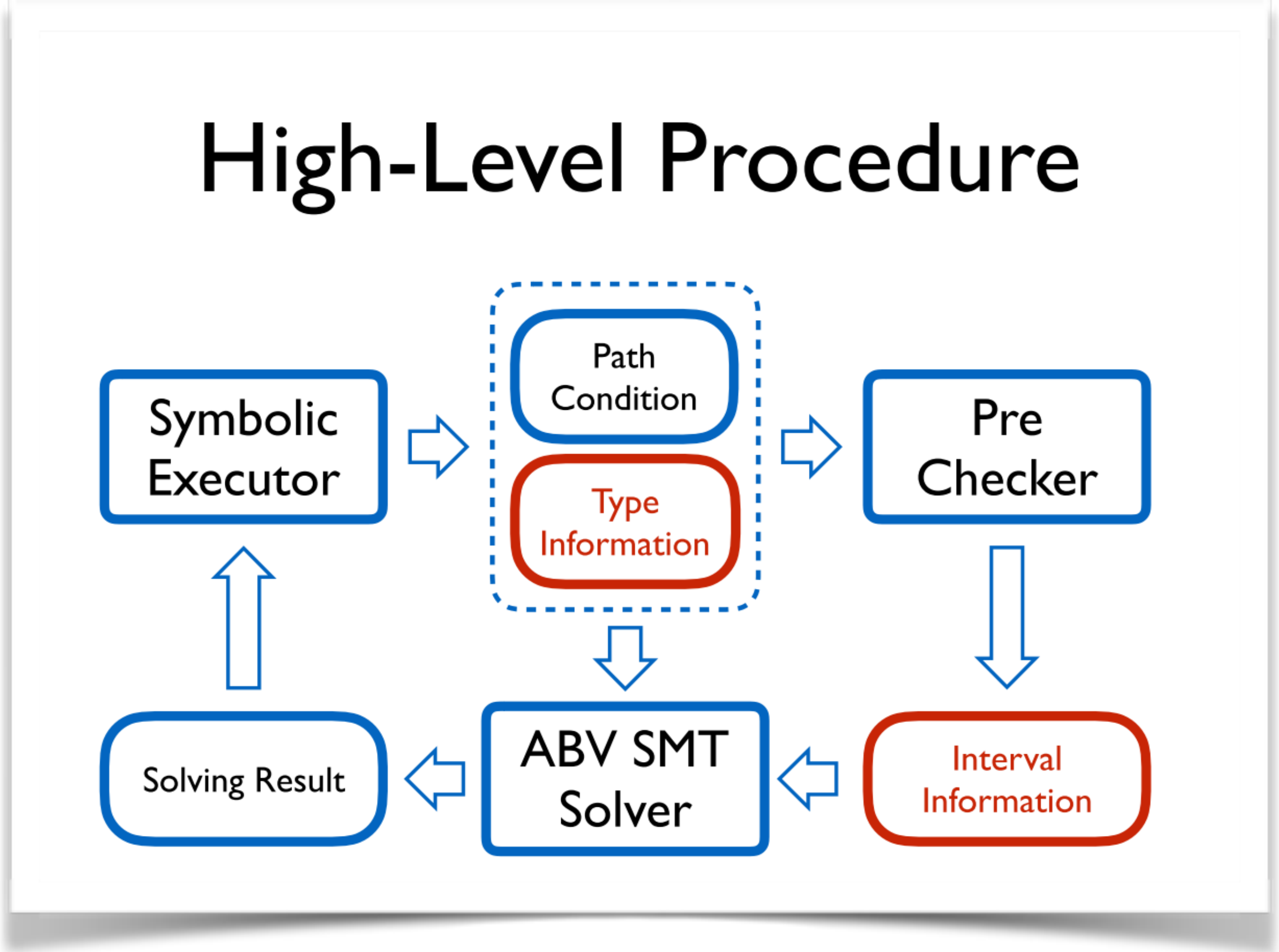
Conclusion



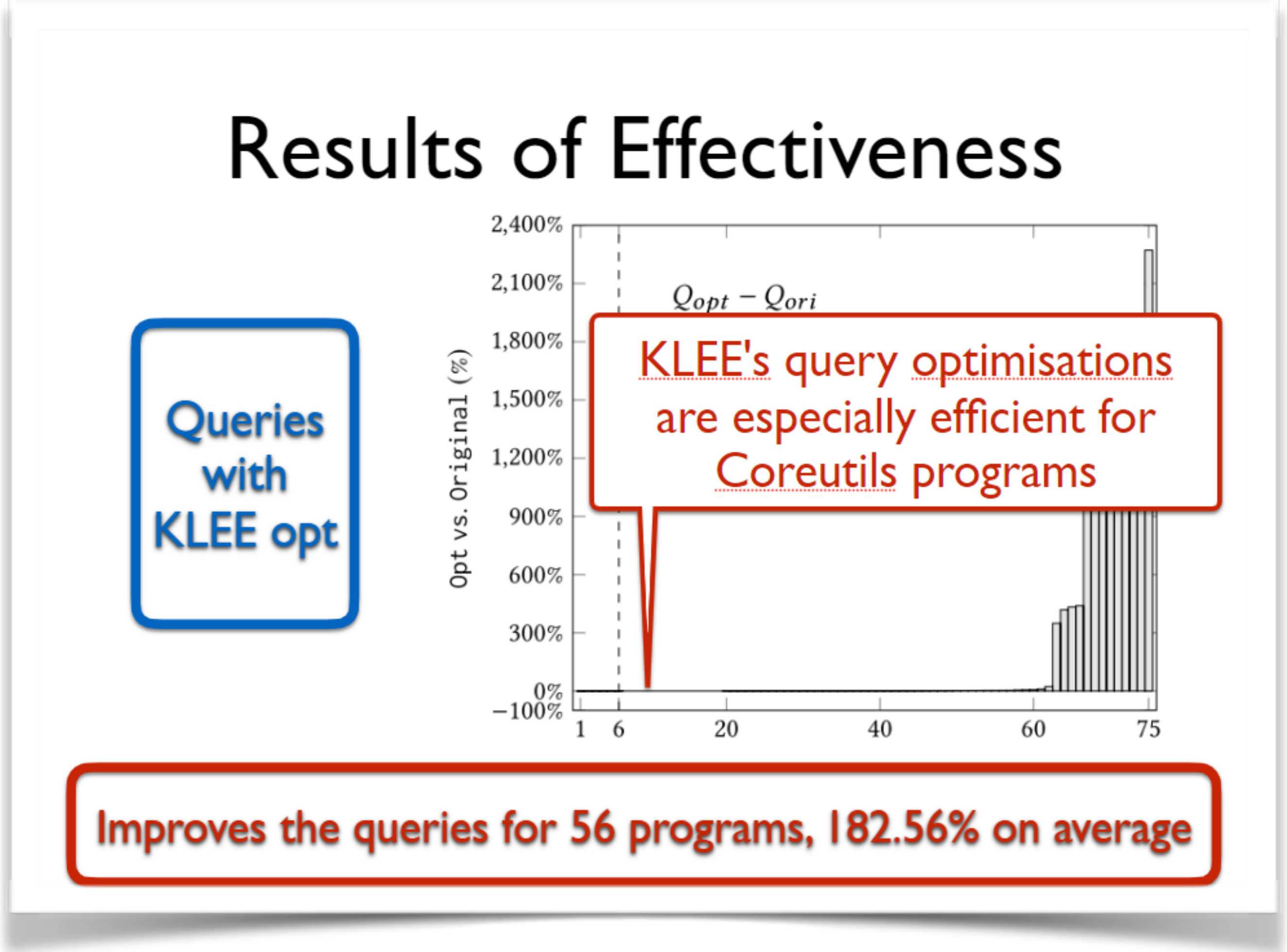
Array SMT Theory
Axiom 1 $i = j \Rightarrow \mathcal{R}(a,i) = \mathcal{R}(a,j)$
Axiom 2 $\mathcal{R}(\mathcal{W}(a,j,v),i) = \begin{cases} v & i = j \\ \mathcal{R}(a,i) & \text{otherwise} \end{cases}$
$a = \{0,0,0,11\}$
$\mathcal{R}_I(a,i) > 10 \wedge i \geq 0 \wedge i \leq 3$
$u > 10 \wedge i \geq 0 \wedge i \leq 3 \wedge (\bigwedge_{n \in \{0,1,2\}} i = n \Rightarrow u = 0) \wedge i = 3 \Rightarrow u = 11$



Our Argument
Symbolic Executor
Array SMT Solver
Tight Coupling
While-box Usage



High-Level Procedure
Symbolic Executor
Path Condition
Type Information
Pre Checker
Solving Result
ABV SMT Solver
Interval Information



Results of Effectiveness
Queries with KLEE opt
KLEE's query optimisations are especially efficient for Coreutils programs
Improves the queries for 56 programs, 182.56% on average

Thank you!
Q&A