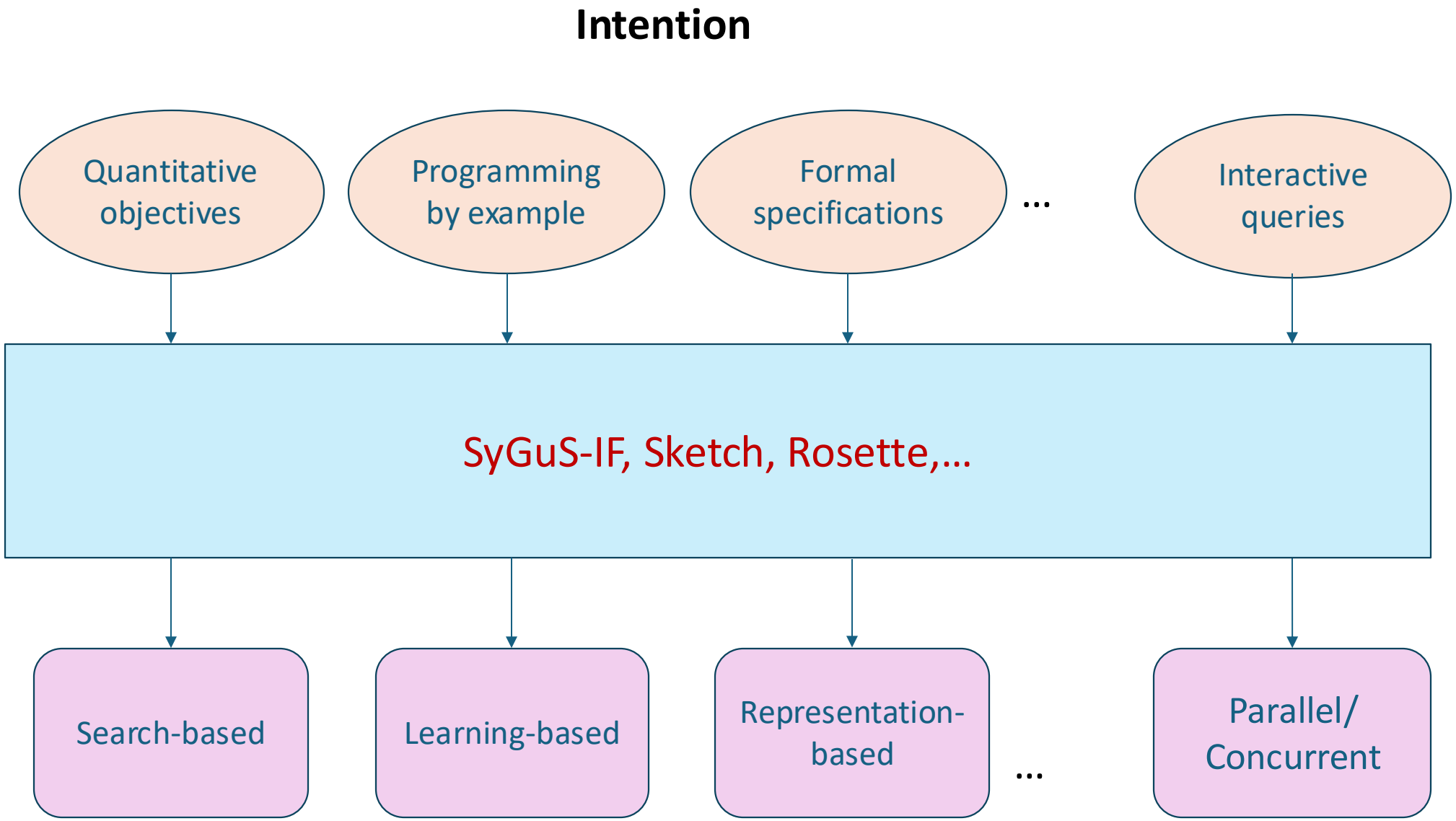
Search-Based Synthesis


Intention
Quantitative objectives
Programming by example
Formal specifications
...
Interactive queries
SyGuS-IF, Sketch, Rosette,...
Search-based
Learning-based
Representation-based
...
Parallel/ Concurrent
Invention

Max3 in SYGUS-IF

Semantic specification:

"Find a function of three integers"

$$\forall x, y, z.$$
$$f(x,y,z) \geq x \wedge f(x,y,z) \geq y \wedge$$
$$f(x,y,z) \geq z \wedge (f(x,y,z) = x \vee$$
$$f(x,y,z) = y \vee f(x,y,z) = z)$$

Syntax-guided Synthesizer

Syntactic constraint:

$G \equiv CLIA$

Aexpr := x | y | z | 0 | 1
| Aexpr + Aexpr
| Aexpr - Aexpr
| ite(Bexpr, Aexpr, Aexpr)

Bexpr := ¬ Bexpr
| Bexpr ∧ Bexpr
| Bexpr ∨ Bexpr
| Aexpr ≤ Aexpr
| Aexpr = Aexpr
| Aexpr ≥ Aexpr

$$f(x,y,z) \equiv ite(x \geq y \wedge x \geq z, x, ite(y \geq z, y, z))$$

SyGuS-IF: Example

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
        ((Start Int (x y 0 1
                (+ Start Start)
                (- Start Start)
                (ite StartBool Start Start)))
        (StartBool Bool (
                (and StartBool StartBool)
                (or StartBool StartBool)
                (not StartBool)
                (<= Start Start))))
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
(check-synth)
```

Theory: Linear integer arithmetic

Syntactic restriction:
Context-Free Grammar

Specification:

$$max2(x, y) \geq x$$
$$max2(x, y) \geq y$$
$$max2(x, y) = x \vee max2(x, y) = y$$

SyGuS Solvers

the ***DryadSynth*** solver

- Supports multiple theories
  - CLIA
  - INV
  - BV
  - String
- Publications
  - **Reconciling Enumerative and Deductive Program Synthesis (PLDI 2020, CLIA)**
  - **Enhanced Enumeration Techniques for Syntax-Guided Synthesis (POPL 2024, BV)**
  - **A Concurrent Approach to String Transformation Synthesis (PLDI 2025, String)**

- Other solvers: CVC4/CVC5, Duet, EUSolver, LoopInvGen, Probe, Simba, etc.

Try DryadSynth at: https://cap.ecn.purdue.edu/dryadsynth

Search-Based Synthesis

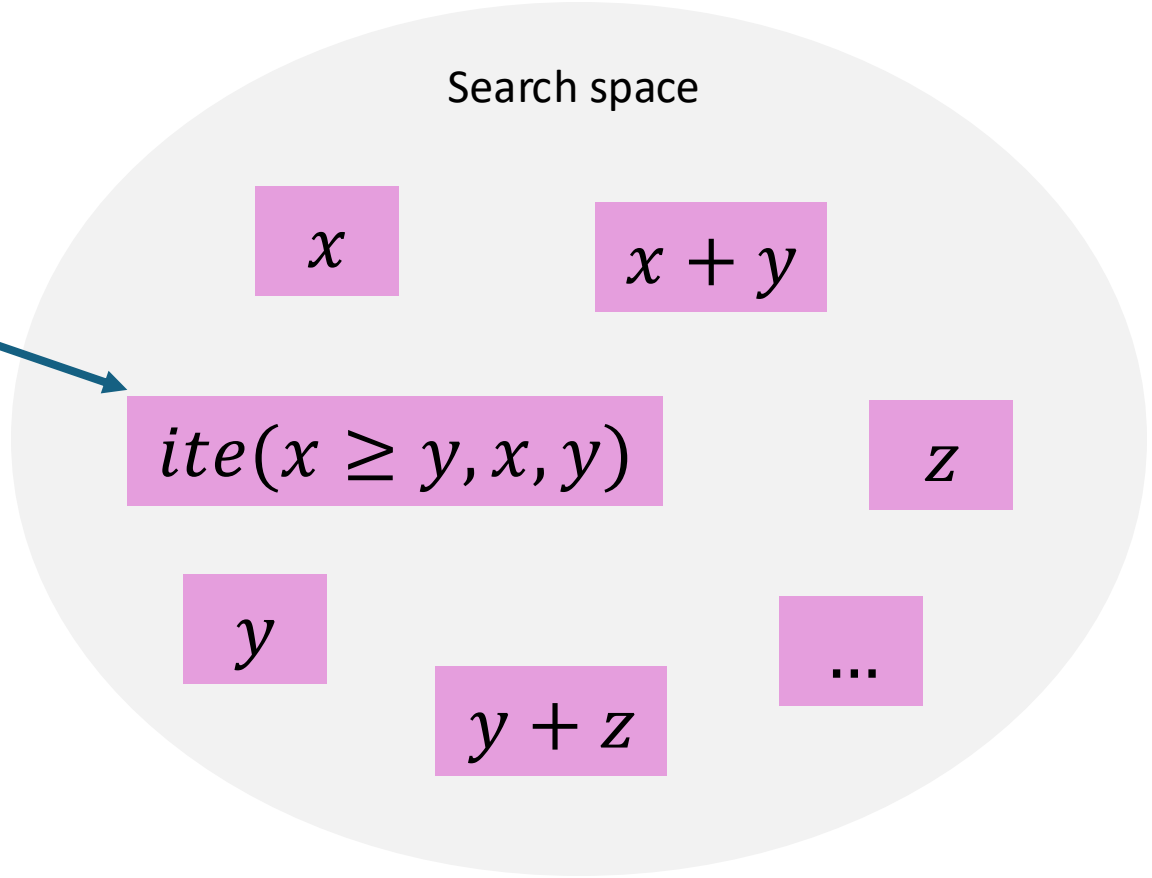
Semantic specification:

Find a function f where $\forall x, y, z.$
$f(x,y,z) \geq x \wedge f(x,y,z) \geq y \wedge$
$f(x,y,z) \geq z \wedge (f(x,y,z) = x \vee$
$f(x,y,z) = y \vee f(x,y,z) = z)$

Syntactic grammar:

Aexpr := x | y | z | 0 | 1
| Aexpr + Aexpr
| Aexpr - Aexpr
| ite(Bexpr, Aexpr, Aexpr)
Bexpr := ¬ Bexpr
| Bexpr ∧ Bexpr
| Bexpr ∨ Bexpr
| Aexpr ≤ Aexpr
| Aexpr = Aexpr
| Aexpr ≥ Aexpr

check



Size-ordered: EUSolver [Alur et al. TACAS'17]

Counterexample-guided: CEGIS [Solar-Lezama et al. ASPLOS'06]

Learning-based: LoopInvGen [Padhi et al. PLDI'16]

Size-Based Search

Aexpr := x | y | 0 | 1
| Aexpr + Aexpr
| Aexpr - Aexpr
| ite(Bexpr, Aexpr, Aexpr)
Bexpr := ¬ Bexpr
| Bexpr ∧ Bexpr
| Bexpr ∨ Bexpr
| Aexpr ≤ Aexpr
| Aexpr = Aexpr
| Aexpr ≥ Aexpr

Size	Candidates
1	$0, 1, x, y$
3	$0+0, 0+1, 0+x, 0+y, 1+1, 1+x, 1+y, x+x, x+y, y+0, y+1, y+x, y+y,$ $0-0, 0-1, 0-x, 0-y, 1-1, 1-x, 1-y, x-x, x-y, y-0, y-1, y-x, y-y$
5	$x+y+1, x+y-1, (1-x)+y, \ldots$
6	$ite(0 \leq 1,0,1), ite(0 \leq 1,0,x), ite(0 \leq 1,0,y), \ldots$
7	…

Symmetries

Multiple ways of representing the same problem

```
Expr := var*const
      | Expr + Expr
```

```
Expr := var*const
      | var*const + Expr
```

w*c1+(x*c2+(y*c3+z*c4))

- Grammar on the right has fewer symmetries
- Grammar on the left can produce all possible ways to parenthesize
- Can completely eliminate symmetries from the right by enforcing a variable ordering
  - Can't be done with a grammar, but it can with a generative model

```
Expr(vmin) := let v = var() in v*const  (assert v > vmin)
            | let v=var() in v*const + Expr(v) (assert v > vmin)
```

Symmetries

Do symmetries matter?

- It depends

Some methods are very sensitive to symmetries

- E.g., search-based synthesis

Others are largely oblivious to them

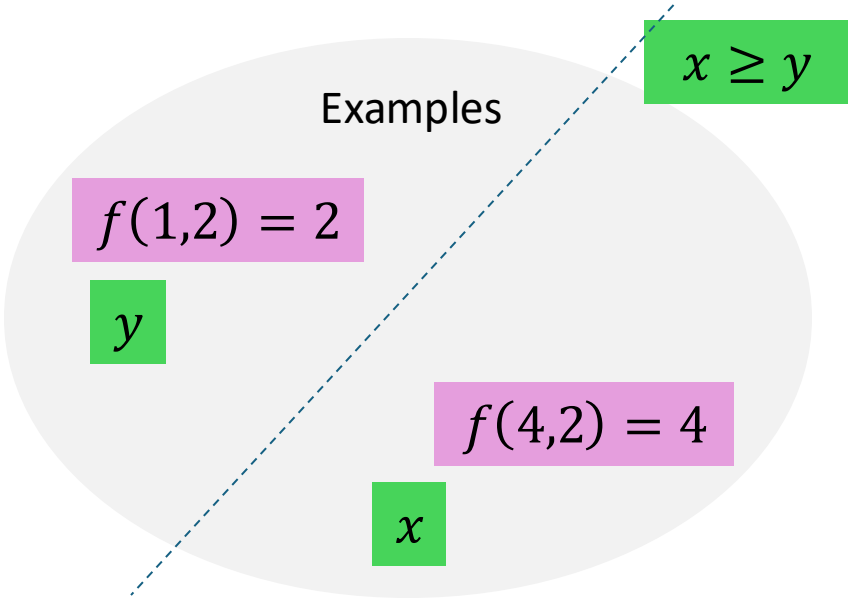
- E.g., random sampling

Decision Tree Representation

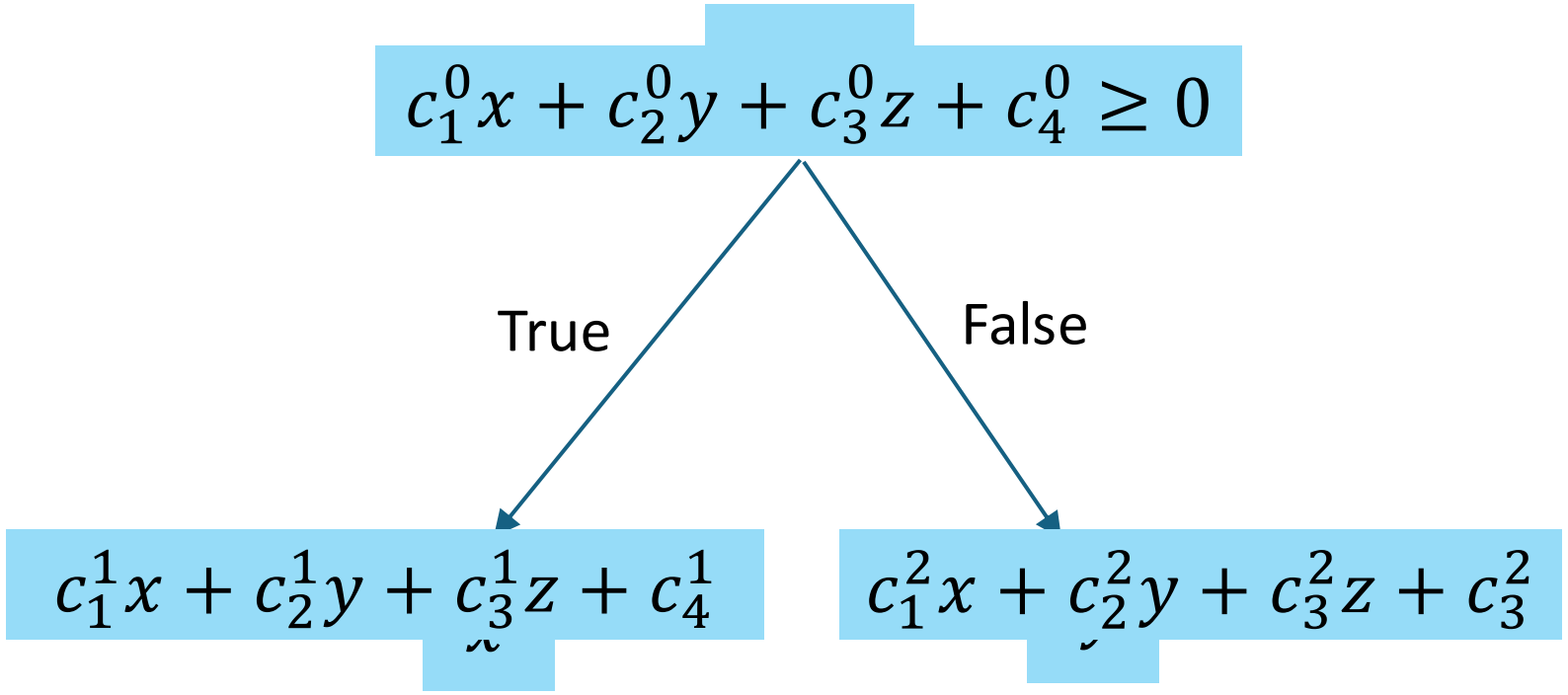
$$f(x,y) \equiv ite(\underbrace{x \geq y}_{\text{True}}, \underbrace{x}_{\text{False}}, y)$$

EUSolver [Alur et al. TACAS'17]

- Enumerate terms and predicates separately
- Assemble terms and predicates to form a decision tree (using information gain heuristics)



Height-Based Enumeration



Synthesis problem

$\exists f. \forall \vec{x}. Spec_f(\vec{x})$

Fixed height h

$\exists \vec{c}. \forall \vec{x}. Spec_{func(T)}(\vec{x})$

Kangjing Huang, Peiyuan Shen, Xiaokang Qiu, and Yanjun Wang. [PLDI'20]

Identifying equivalent programs

Program semantic equivalence is hard

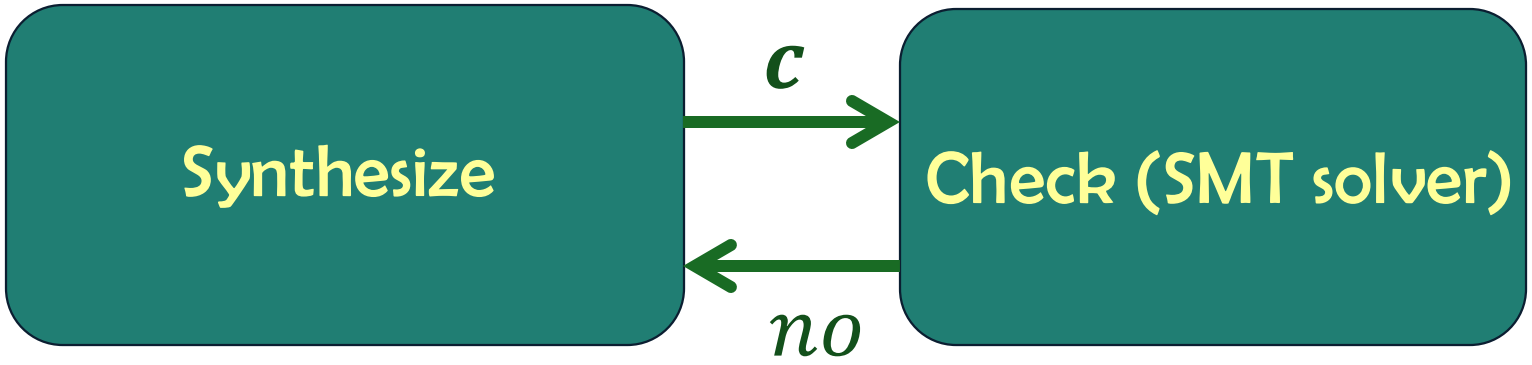
- It is also unnecessary!

Observational Equivalence

- Are they equivalent wrt the inputs
  - easy to check efficiently
  - sufficient for the purpose of PBE
- Keep only the simplest one

Better search strategies?

Enumerate-check loop



✓ guarantees to produce the smallest program

✗ number of potential solutions grows exponentially

What if the checker can provide richer feedback?

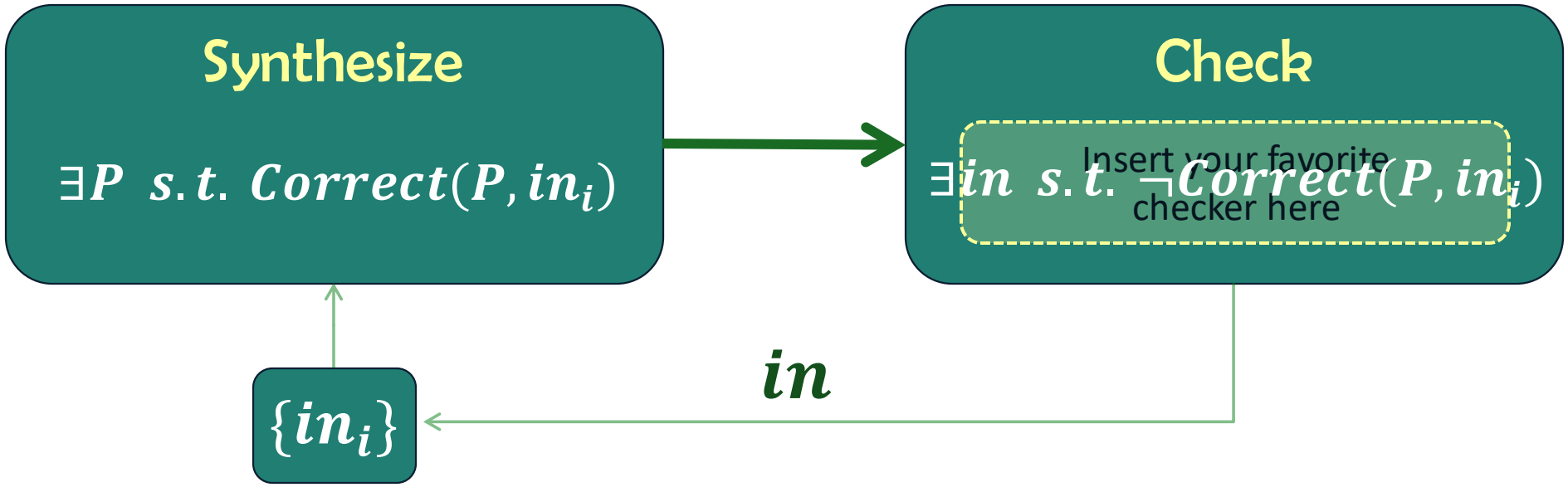
CounterExample-Guided Inductive Synthesis (CEGIS)

Counterexample guided inductive synthesis

Ideas

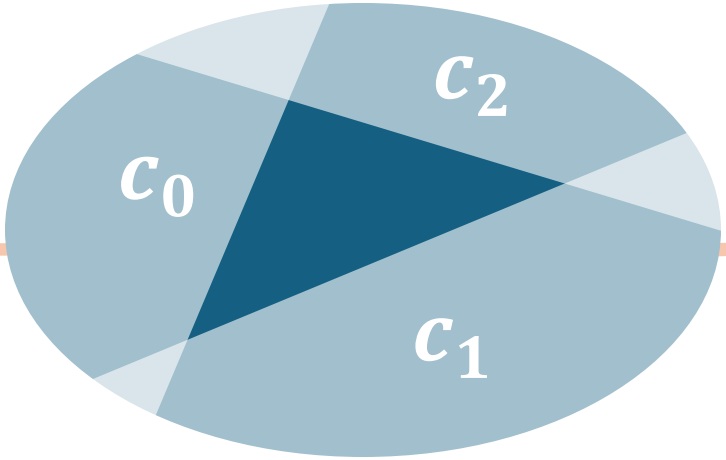
- Rely on an oracle to tell you if your program is correct
- If it is not, rely on oracle to generate counterexample inputs
- Reduce to an inductive synthesis problem

CounterExample-Guided Inductive Synthesis (CEGIS)

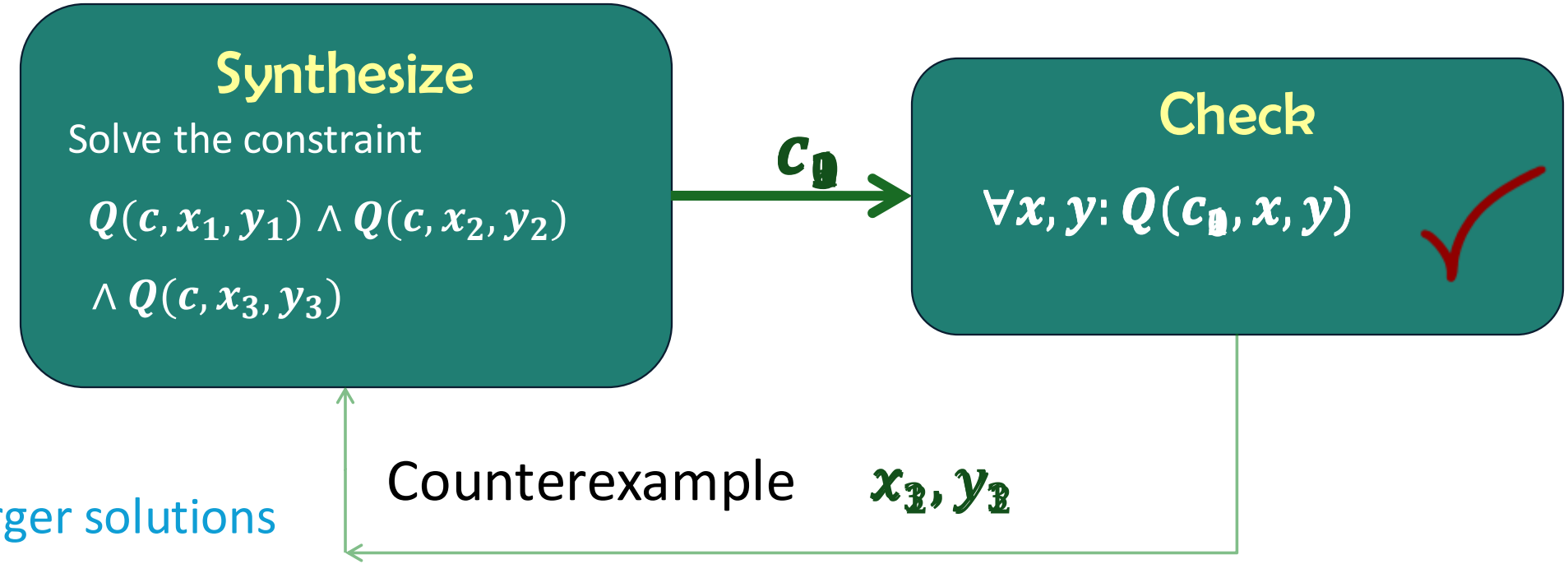


Sketch [Solar-Lezama et al. ASPLOS'06]

CEGIS



Goal: find a solution c such that $\forall x, y: Q(c, x, y)$



✓ scales to larger solutions

✗ constraint solving is ad hoc

Deductive Synthesis

(Top-Down Search)

Deductive Synthesis

max2_max3:

Semantic specification Φ:

Find a function foo where $\forall x, y, z.$

$f(x,y,z) \geq x \wedge f(x,y,z) \geq y \wedge f(x,y,z) \geq z \wedge (f(x,y,z) = x \vee f(x,y,z) = y \vee f(x,y,z) = z)$

Syntactic constraint $\mathcal{G}$:

Aexpr := x | y | z |

|max2(Aexpr, Aexpr)*

** $max2(a, b) \equiv ite(a \geq b, a, b)$*

$$f(x,y,z) = max2(max2(x,y),z)$$

$$f(x,y,z) = ite(ite(x \geq y, x, y) \geq z, ite(x \geq y, x, y), z)$$

$f(x,y,z) \geq ite(ite(x \geq$ ...

$\wedge f(x,y,z) \leq ite(ite($ ...

$f(x,$ ...

$\wedge (f($ ...

$\wedge (f($ ...

Match

$(f, f(y) = e, \mathcal{G}) \Rightarrow (f, f(y) = e', \mathcal{G})$ $if\ e \Rightarrow_{\mathcal{G}}^{*} e'\ and\ e' \in [\![\mathcal{G}(y)]\!]$

Eq

$f(e) \geq e_1 \wedge f(e) \leq e_2 \Rightarrow f(e) = e_1\ \ if \Gamma \Rightarrow e_1 = e_2$

IntEq

$f(y) = e \wedge \Psi \Rightarrow f(y) = e \wedge \Psi[\lambda y.\, e/f]$

GeMax

$f(e) \geq e_1 \wedge f(e) \geq e_2 \Rightarrow f(e) \geq ite(e_1 \geq e_2, e_1, e_2)$

LeMin

...

CNF

$(\Phi \vee \Psi_1) \wedge (\Phi \vee \Psi_2) \Rightarrow \Phi \vee (\Psi_1 \wedge \Psi_2)$

Kangjing Huang, Peiyuan Shen, Xiaokang Qiu, and Yanjun Wang. [PLDI'20]

Deductive rules for $\mathcal{G}_{CLIA}$

GeMax

$$f(\boldsymbol{e}) \geq e_1 \wedge f(\boldsymbol{e}) \geq e_2 \implies f(\boldsymbol{e}) \geq \texttt{ite}(e_1 \geq e_2, e_1, e_2)$$

LeMin

$$f(\boldsymbol{e}) \leq e_1 \wedge f(\boldsymbol{e}) \leq e_2 \implies f(\boldsymbol{e}) \leq \texttt{ite}(e_1 \geq e_2, e_2, e_1)$$

GeMin

$$f(\boldsymbol{e}) \geq e_1 \vee f(\boldsymbol{e}) \geq e_2 \implies f(\boldsymbol{e}) \geq \texttt{ite}(e_1 \geq e_2, e_2, e_1)$$

LeMax

$$f(\boldsymbol{e}) \leq e_1 \vee f(\boldsymbol{e}) \leq e_2 \implies f(\boldsymbol{e}) \leq \texttt{ite}(e_1 \geq e_2, e_1, e_2)$$

Eq

$$f(\boldsymbol{e}) \geq e_1 \wedge f(\boldsymbol{e}) \leq e_2 \implies f(\boldsymbol{e}) = e_1 \quad \text{if } \mathcal{T} \models e_1 = e_2$$

NotEq

$$f(\boldsymbol{e}) \geq e_1 \vee f(\boldsymbol{e}) \leq e_2 \implies f(\boldsymbol{e}) \neq e_1 - 1 \quad \text{if } \mathcal{T} \models e_1 = e_2 + 2$$

CNF

$$(\Phi \vee \Psi_1) \wedge (\Phi \vee \Psi_2) \implies \Phi \vee (\Psi_1 \wedge \Psi_2) \quad \text{if } f \text{ does not occur in } \Psi_1 \text{ or } \Psi_2$$

Efficient

Specific to some grammars or applications

SyGuS problem: qm_max3

Semantic specification:

"Find a function f where

"Find the max of three integers"

$\forall x, y, z. f(x, y, z) =$

$ite(x \geq y \wedge x \geq z, x, ite(y \geq z, y, z))$

New syntactic grammar:

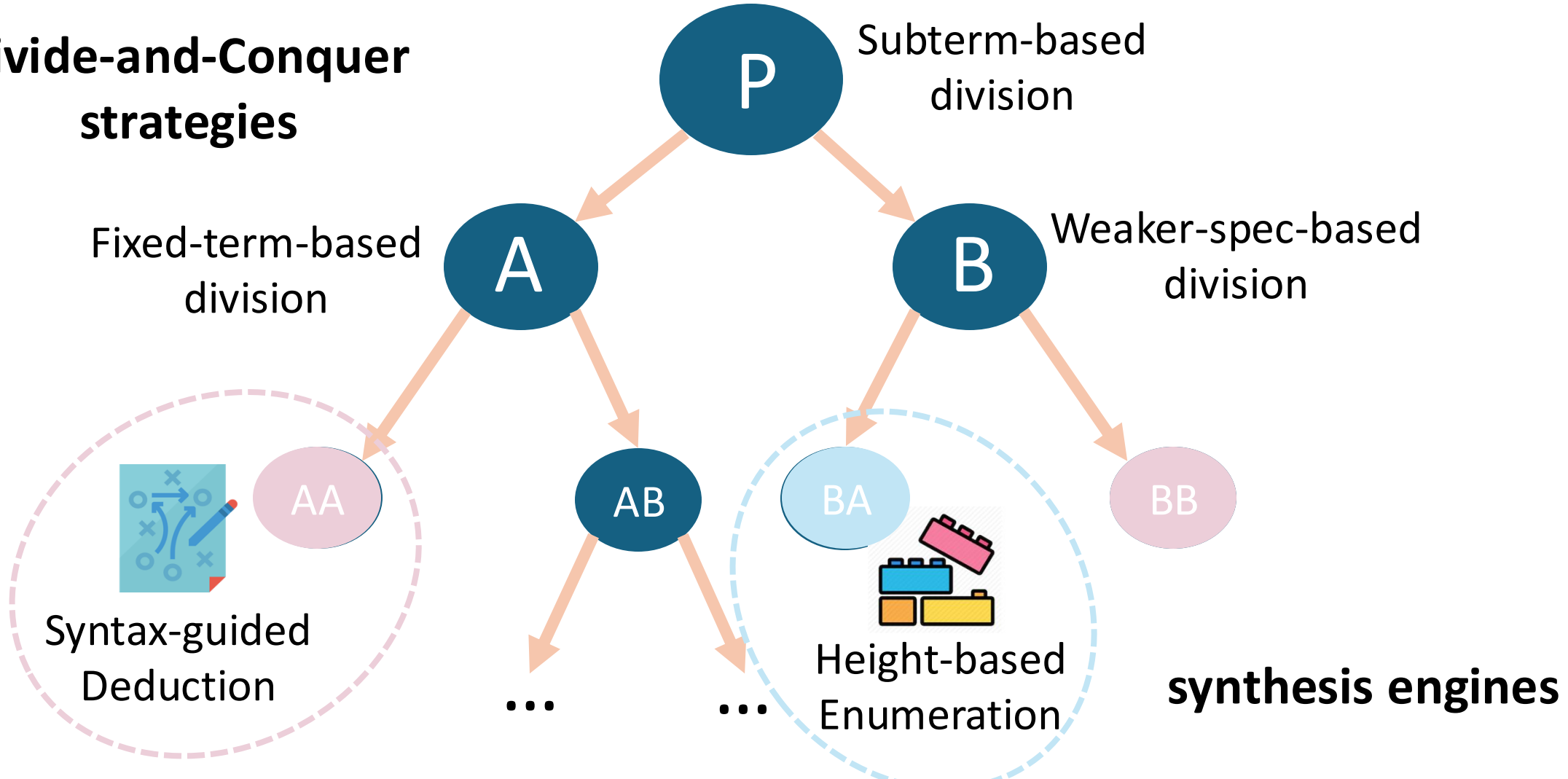
Aexpr := x | y | z | 0 | 1
| Aexpr + Aexpr
| Aexpr - Aexpr
| qm(Aexpr, Aexpr) *

$* \; qm(a, b) \equiv ite(a < 0, b, a)$

Syntax-guided Synthesizer

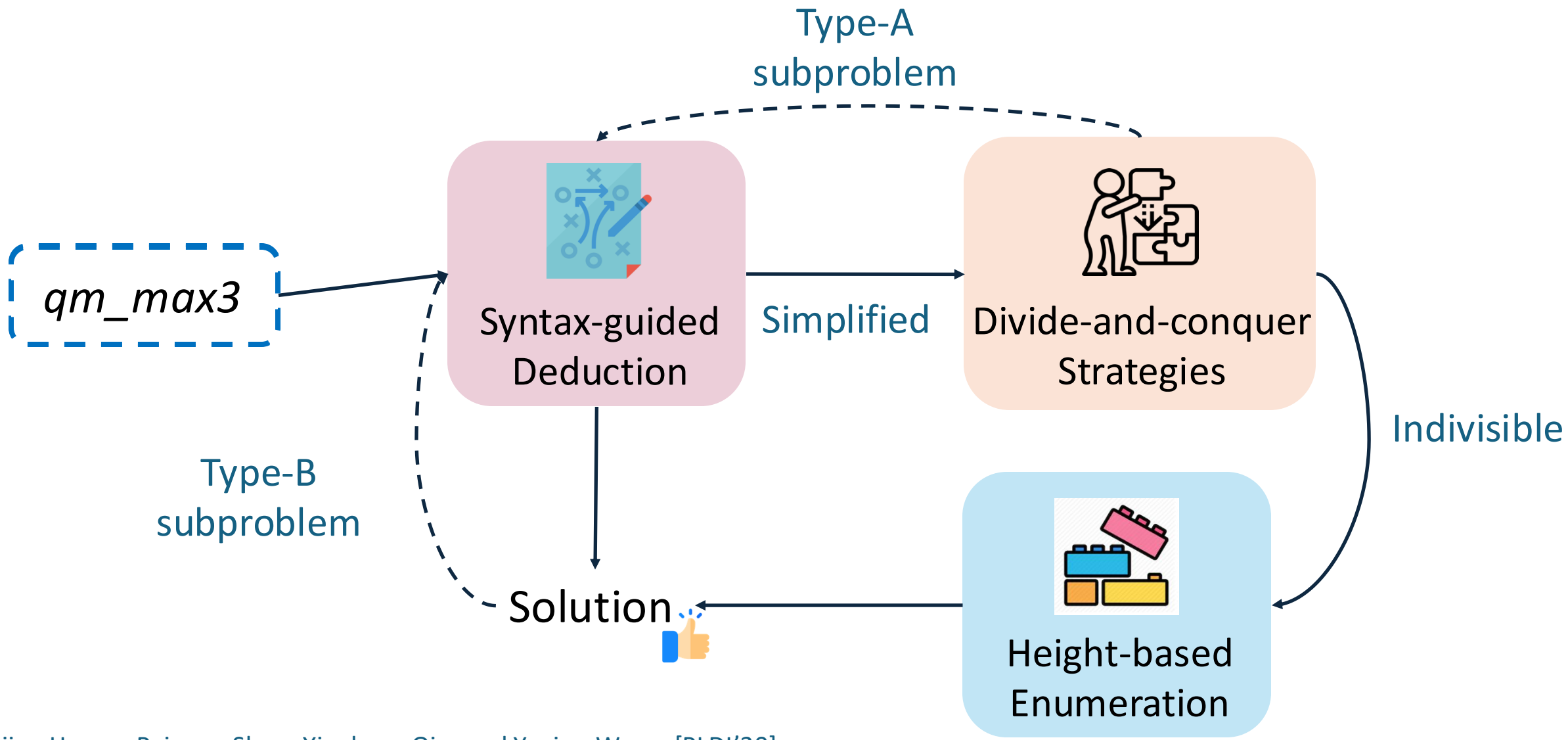
Reconciling Enumeration and Deduction

Divide-and-Conquer strategies



synthesis engines

Cooperative Synthesis



Kangjing Huang, Peiyuan Shen, Xiaokang Qiu, and Yanjun Wang. [PLDI'20]

Subterm-Based Division

qm_max3:

Semantic specification Φ:

Find a function foo where

$\forall x, y, z. foo(x, y, z) =$
$ite(x \geq y \wedge x \geq z, x, ite(y \geq z, y, z))$

Syntactic constraint $\mathcal{G}$:

Aexpr := x | y | z | 0 | 1
| Aexpr + Aexpr
| Aexpr - Aexpr
| qm(Aexpr, Aexpr) *

$* qm(a, b) \equiv ite(a < 0, b, a)$

Subproblem A:

Semantic specification Φ_A:

Find an ***auxiliary*** function

$aux(y, z) = ite(y \geq z, y, z)$

Syntactic constraint $\mathcal{G}_A$: Unchanged

Subproblem B:

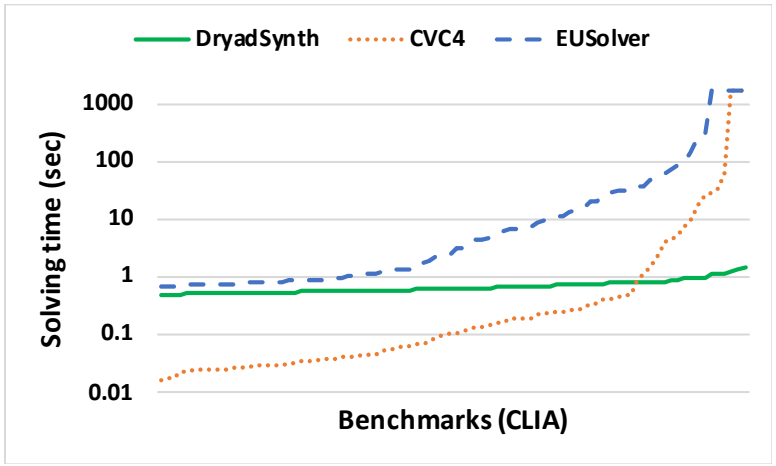
Semantic specification Φ: Unchanged

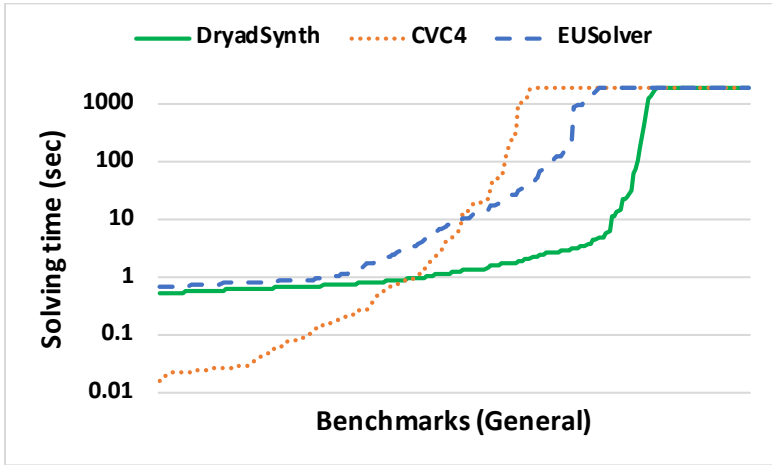
Syntactic constraint $\mathcal{G}$:

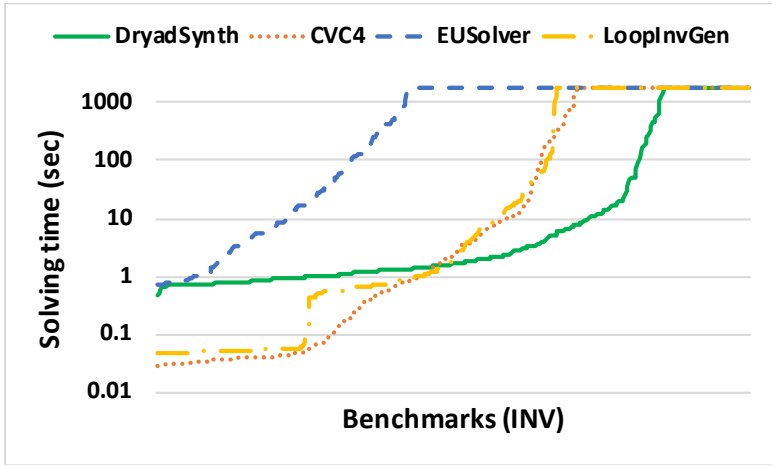
Aexpr := x | y | z | 0 | 1
| Aexpr + Aexpr
| Aexpr - Aexpr
| qm(Aexpr, Aexpr) *
| aux(Aexpr, Aexpr) ‘

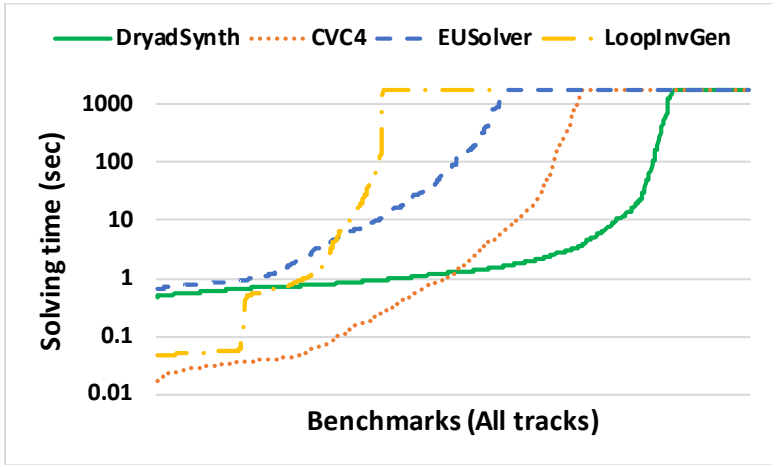
$*qm(a, b) \equiv ite(a < 0, b, a)$
$'aux(a, b) \equiv ite(a \geq b, a, b)$

Experimental Results









Solving time per benchmark in ascending order

How about bit-vectors?

Hacker's delight 25 in SyGuS-IF

Semantic specification:

Find a function $f(x, y)$ where

$$f\left(\begin{pmatrix}\#x8000\\ \#x01FF\\ \dots\end{pmatrix}, \begin{pmatrix}\#x0044\\ \#x00F5\\ \dots\end{pmatrix}\right) = \begin{pmatrix}\#x0055\\ \#x0001\\ \dots\end{pmatrix}$$

Compute higher order half of product of bitvectors x and y.

Syntax-guided Synthesizer

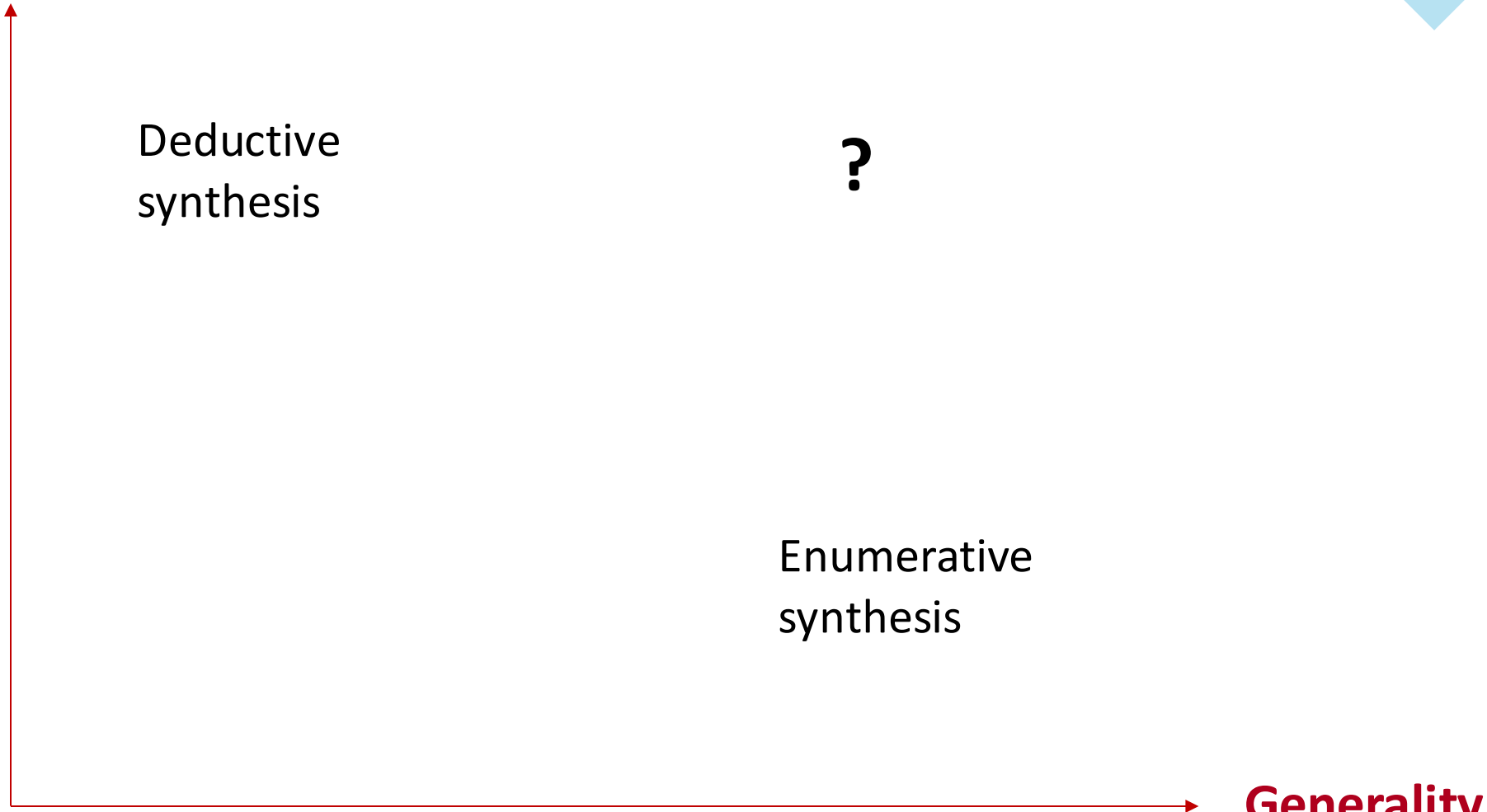
Syntactic constraints:

$G :=$ *Many Bitvec operations*

$x \mid y \mid$ 0x0000 | 0x00ff | 0x0008 | ...
| not E | E and E | E or E | E xor E
| neg E | E add E | E mul E
| E sub E | E udiv E | E sdiv E |
| E urem E | E srem E | E ashr E
| E << E | E >> E | ITE (E = 0, E, E)

$$f(x, y) \stackrel{\text{def}}{=}$$

```
(bvadd
    (bvlshr
        (bvadd
            (bvmul (bvand v0 #x00000000ffffffff) (bvlshr v1 #x0000000000000020))
            (bvlshr
                (bvmul (bvand v1 #x00000000ffffffff) v0)
            #x0000000000000020))
    #x0000000000000020)
    (bvadd
        (bvmul (bvlshr v0 #x0000000000000020) (bvlshr v1 #x0000000000000020))
        (bvlshr
            (bvadd
                (bvmul (bvlshr v0 #x0000000000000020) (bvand v1 #x00000000ffffffff))
                (bvlshr
                    (bvmul (bvand v0 #x00000000ffffffff) (bvand v1 #x00000000ffffffff))
                #x0000000000000020))
        #x0000000000000020)))
```


Efficiency
Deductive synthesis
?
Enumerative synthesis
Generality

Top-down deduction: too many ways to decompose!

$f(11,\ 10) = 01$

$f(x, y) \stackrel{\text{def}}{=} x \ \textbf{xor}\ y$

$x - y$

$\textbf{not}\ y$

$\text{and}(x, y) \gg 1$

…

$01\ \textbf{xor}\ 00$

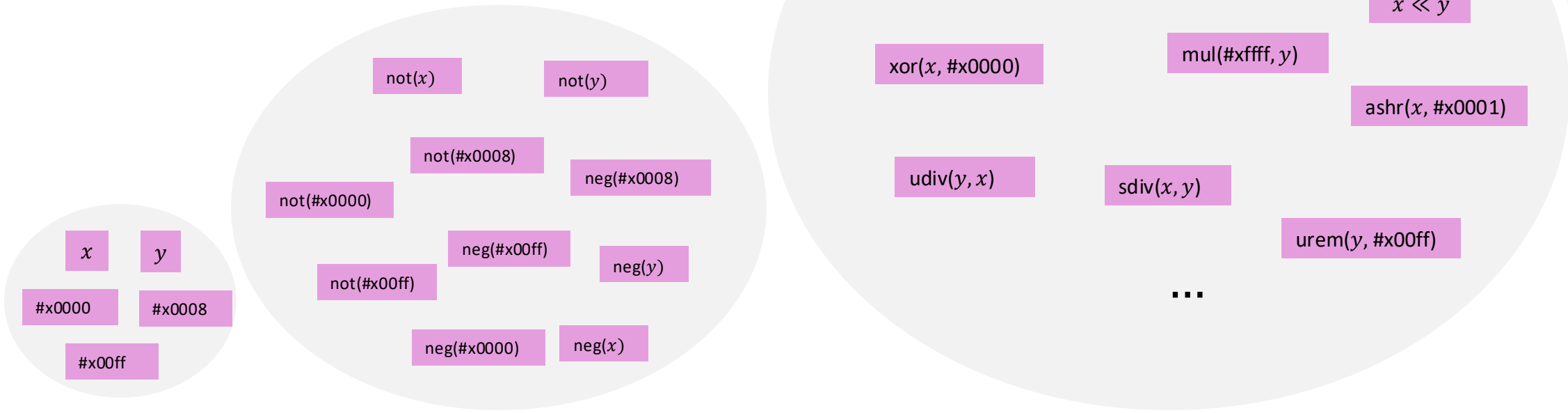
$00\ \textbf{xor}\ 01$

$11\ \textbf{xor}\ 10$

$10\ \textbf{xor}\ 11$

Bottom-up enumeration: exponential blowup!

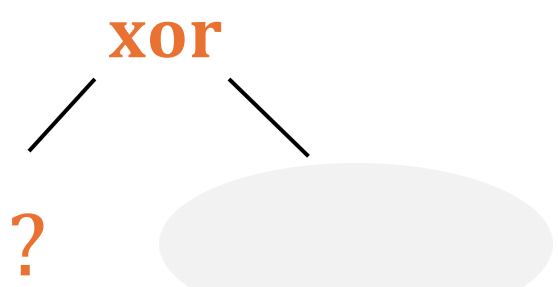
E :=
x | y | 0x0000 | 0x00ff | 0x0008 | ...
| not E | E and E | E or E | E xor E
| neg E | E add E | E mul E
| E sub E | E udiv E | E sdiv E |
| E urem E | E srem E | E ashr E
| E << E | E >> E | ITE (E = 0, E, E)

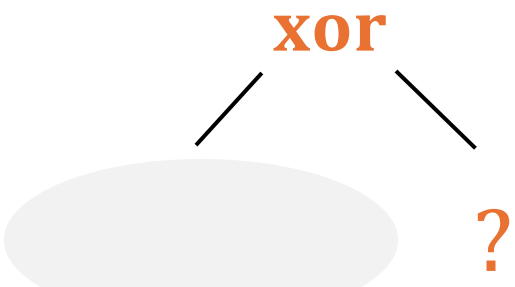


Example (Hacker's delight 19, difficulty 5): Given a register x with two fields A (indicated by mask m) and B (indicated by distance k from A), exchange A and B.

- E.g., $f(\#\text{xAFB1}, \#\text{xFF}, \#\text{x08}) = \$\text{xB1AF}$

$\mathbf{xor} f(x, m, k) \vDash spec$?





#1: Bottom-up deduction

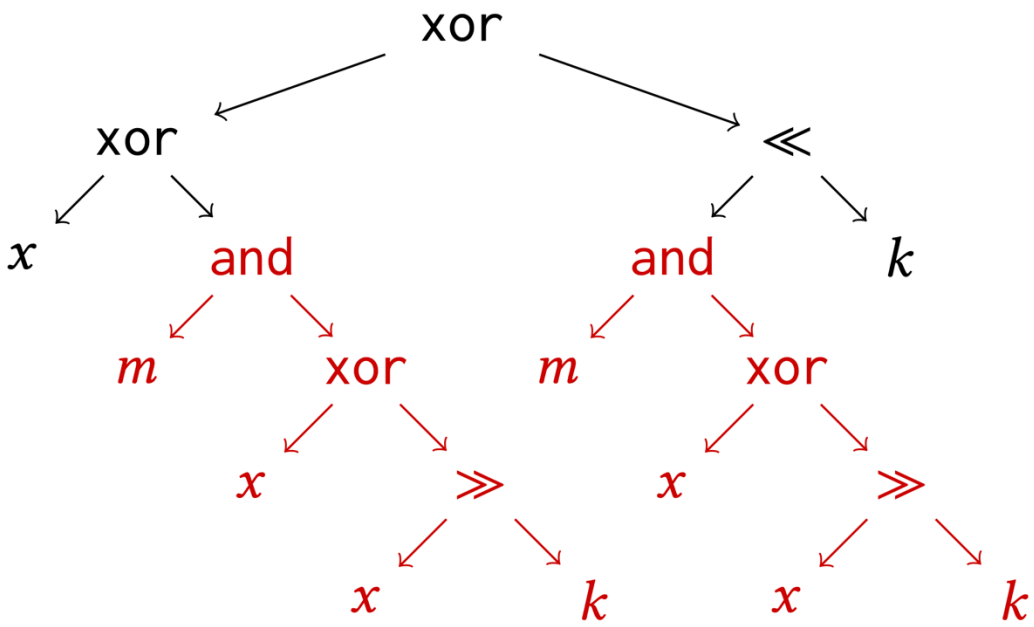
$\boldsymbol{E} =$ 1 x $(\xi \ll k)$ $(x \mathbf{xor} \xi)$ $(x \mathbf{xor} \xi) \mathbf{xor} (\xi \ll k)$ Larger

1264524th 3828868th $\approx 10^{14}$ th

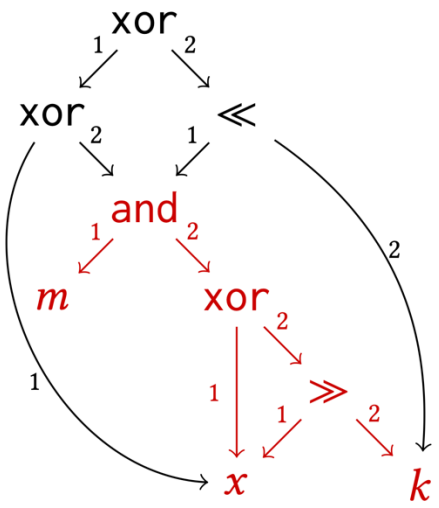
$\xi \stackrel{\text{def}}{=} (m \mathbf{and} (x \mathbf{xor} (x \gg k)))$

#2: Term-graph-based enumeration

$$(x \ \mathbf{xor}\ \xi)\ \mathbf{xor}\ (\xi \ll k)$$



(19 nodes)

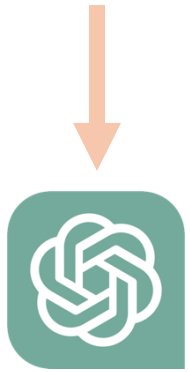


(9 nodes)

Yuantian Ding and Xiaokang Qiu: Enhanced enumeration techniques for syntax-guided synthesis of bit-vector manipulations. [POPL'24]

#3: Large language model guidance

Ask: Give me some useful subexpressions in each step to compute higher order half of product of x and y. Here are some of the examples: .....



Solution 1:

```
xy = x * y
xyl = xy & #xFFFFFFFF
xyh = xy >> 64
high = xyh + ((x >> 64) *
  (y & #xFFFFFFFF)) + ((y >> 64) *
  (x & #xFFFFFFFF))
res = (high << 64) | xyl
```

Solution 2:

```
xu = x >> 32
xl = x & #xFFFFFFFF
yu = y >> 32
yl = y & #xFFFFFFFF
p1 = xu * yu, p2 = xu * yl
p3 = xl * yu, p4 = xl * yl
res = ((p1 << 64) + (p2 << 32)
    + (p3 << 32) + p4) >> 64
```

...

Solution 20:

```
xu = x >> 32
xl = x & #xFFFFFFFF
yu = y >> 32
y = y & #xFFFFFFFF
p1 = xu * yu, p2 = xu * yl
p3 = xl * yu, p4 = xl * yl
res = p1 + (p2 >> 32) + (p3 >> 32)
+ (p4 >> 32)
```

Common Subexpressions:

`xu = x >> 32` [7]	`xl = x & #xFFFFFFFF` [7]
`yu = y >> 32` [7]	`yl = y & #xFFFFFFFF` [7]
`p1 = xu * yu` [6]	`p2 = xu * yl` [6]
`p3 = xl * yu` [6]	`p4 = xl * yl` [6]

hints

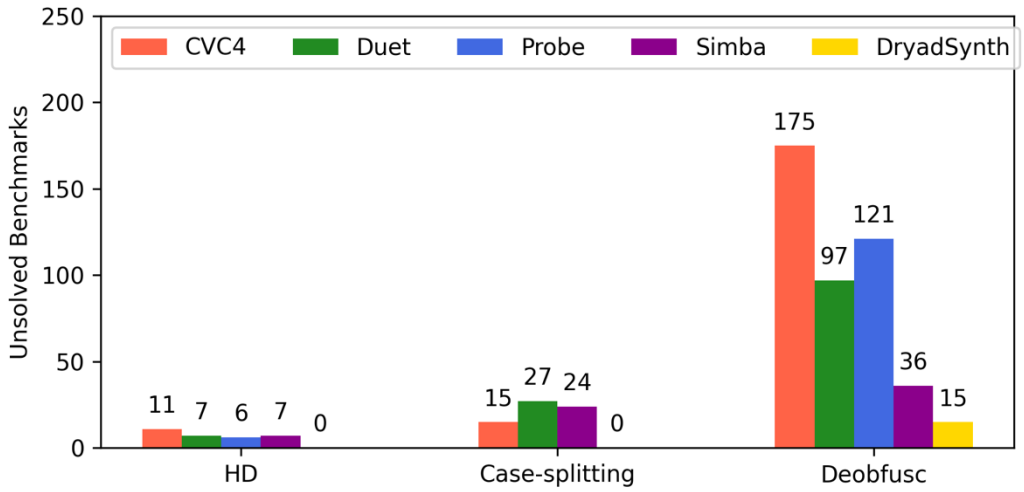
Grammar $\mathscr{G}$:

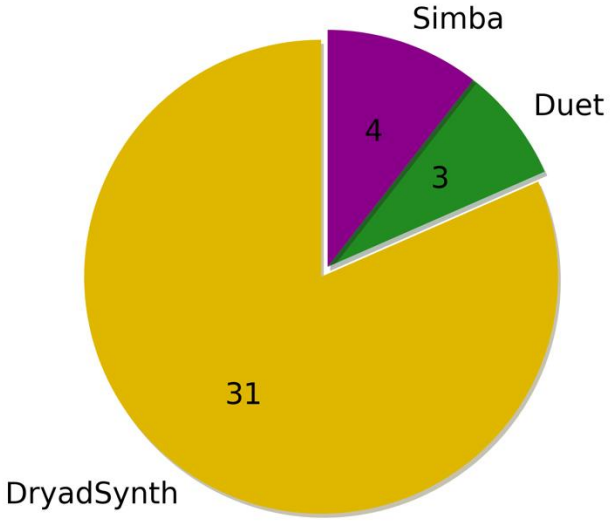
$E \to \text{not } E \mid E \text{ and } E \mid \cdots$

$E \to f_u(E) \mid f_l(E)$

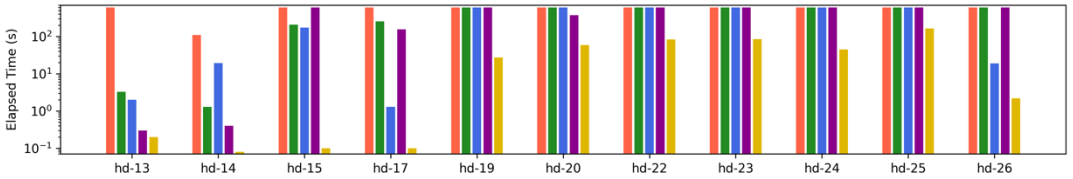
$E \to f_{p1}(E,E) \mid f_{p2}(E,E) \mid f_{p4}(E,E)$

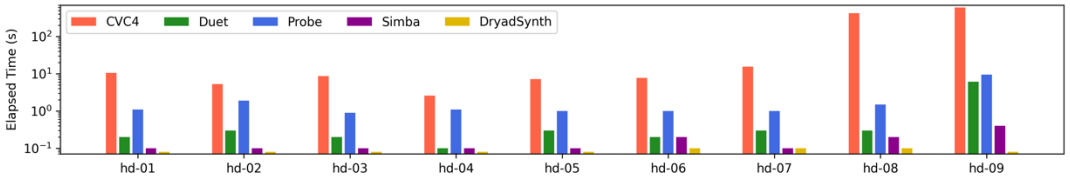
Yuantian Ding and Xiaokang Qiu: Enhanced enumeration techniques for syntax-guided synthesis of bit-vector manipulations. [POPL'24]



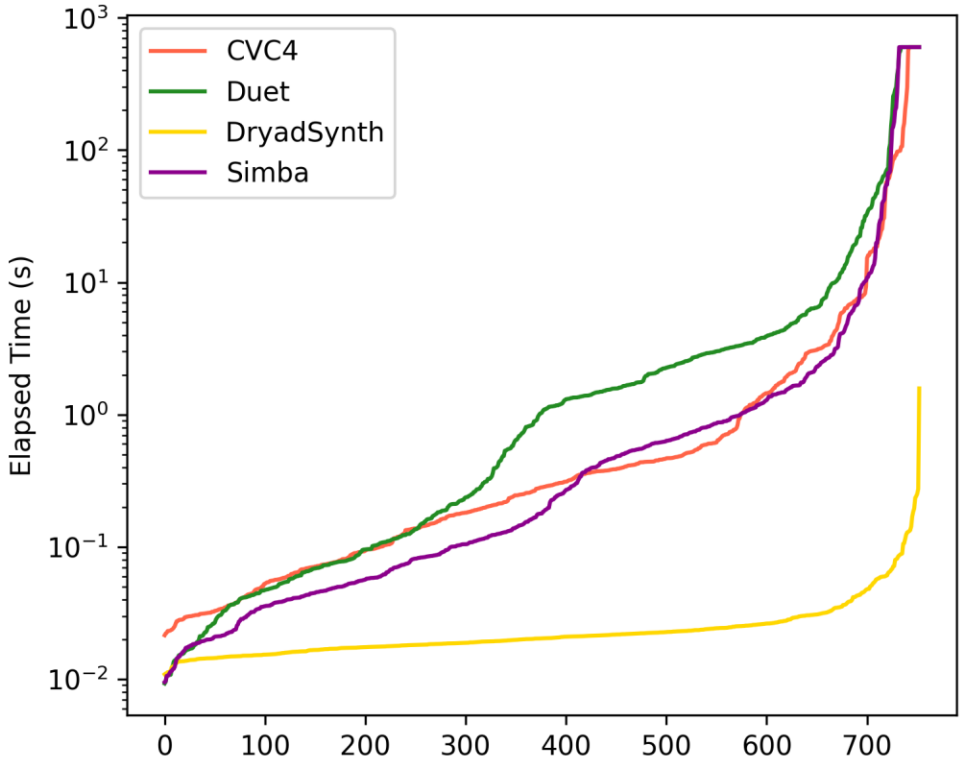


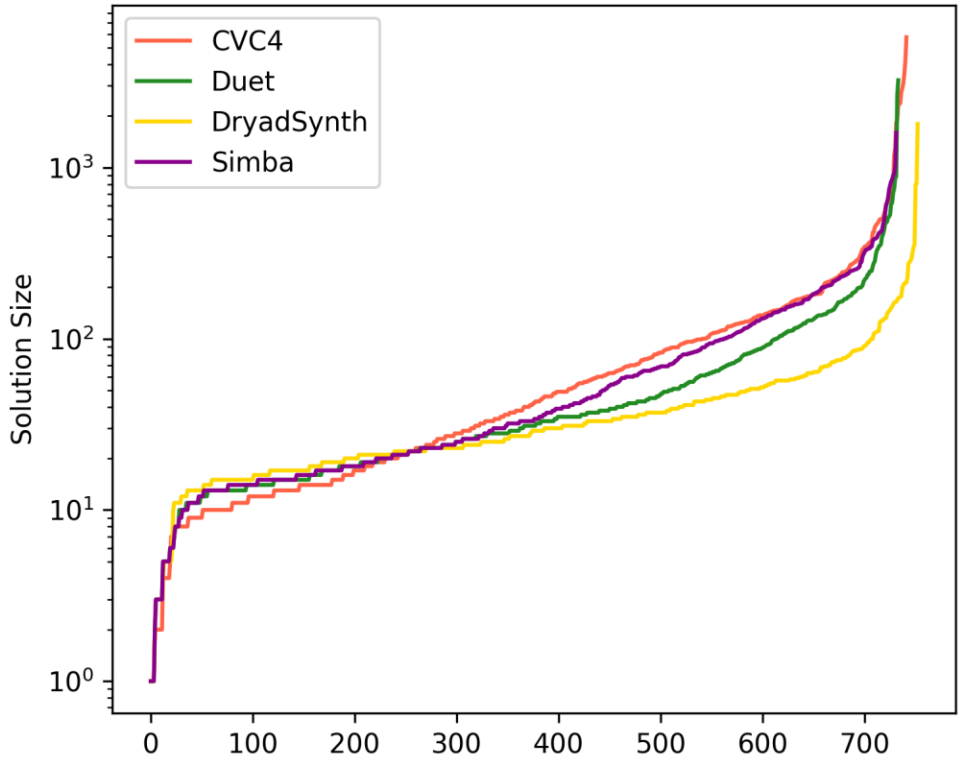
Unsolved and uniquely solved Benchmarks



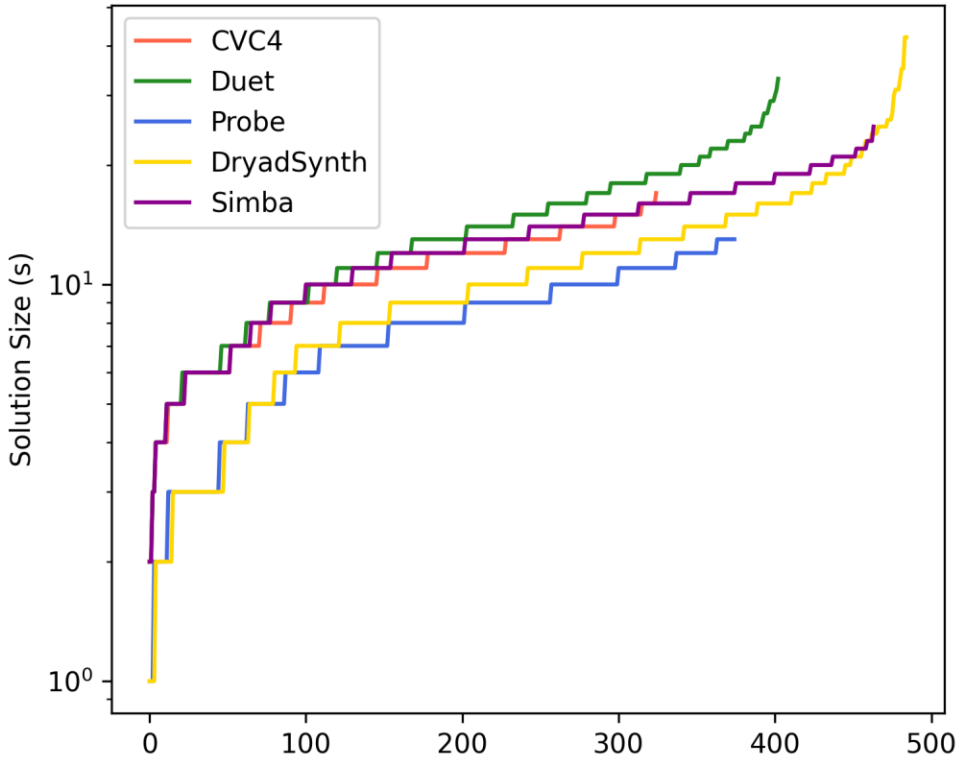


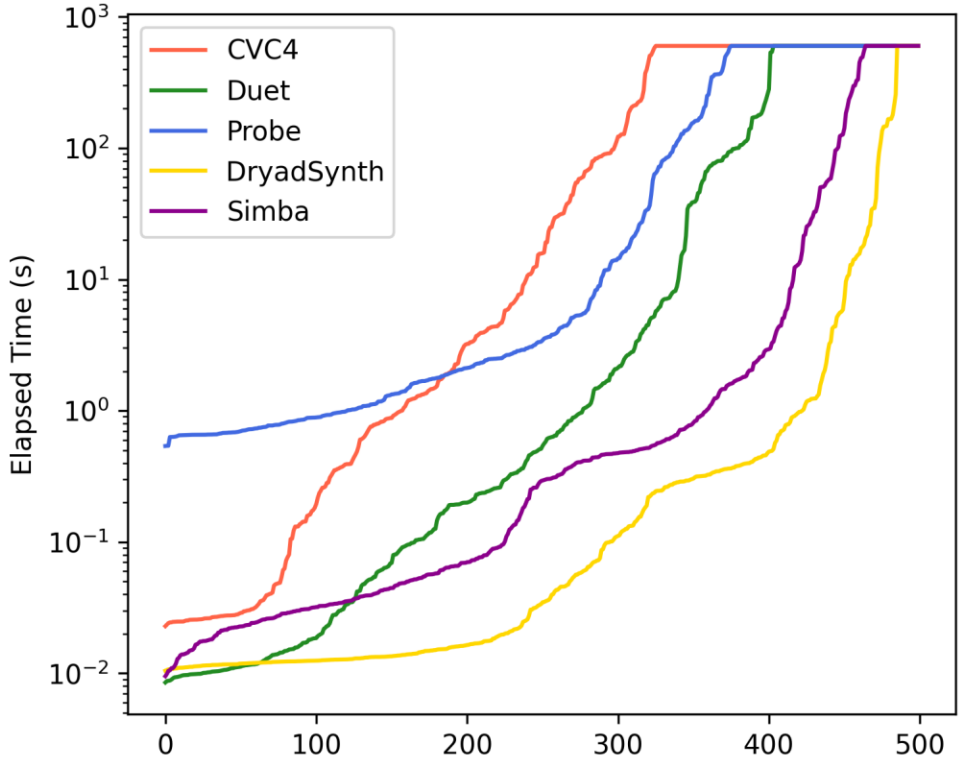
Hacker's Delight





Case-splitting





Deobfuscation