



Program Synthesis-based Mixed Boolean Arithmetic (MBA) Deobfuscation

Woosuk Lee
(Joint work with my student Jaehyung Lee)

Hanyang University

2025. 10. 17 @ CheckMATE 2025

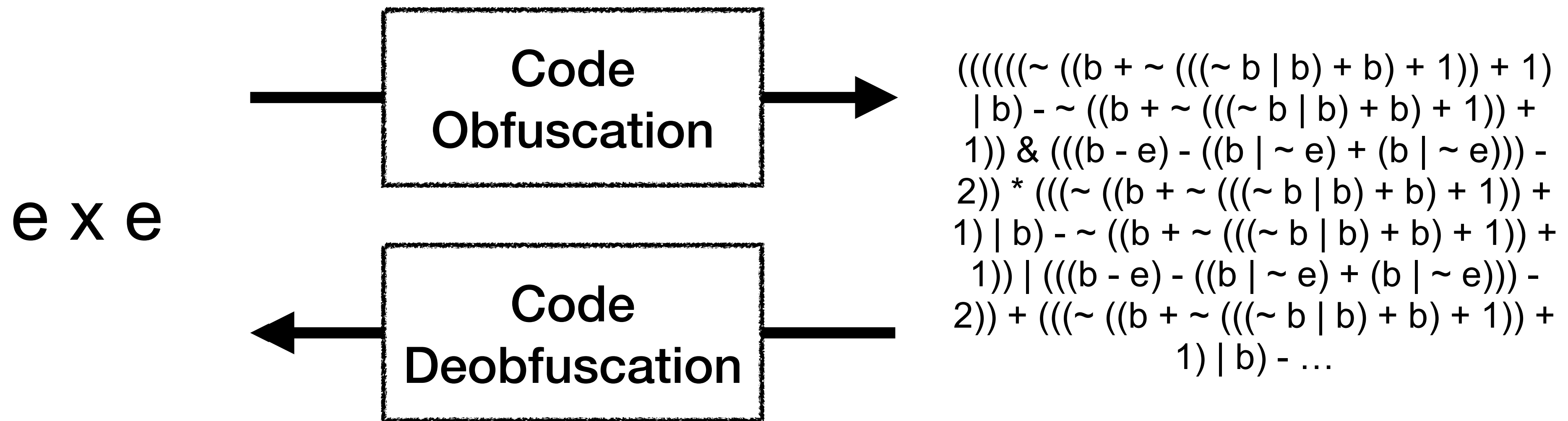
Who am I

- Associate Professor at Hanyang University in South Korea
- Research areas: programming languages (PL), software engineering (SE)
 - Automated program generation & repair
 - Static program analysis

<http://psl.hanyang.ac.kr>

- Vita
 - 2017-2018 Postdoctoral fellow, University of Pennsylvania (Advisor: Mayur Naik)
 - 2016-2017 Postdoctoral fellow, Georgia Tech (Advisor: Mayur Naik)
 - 2016 Ph.D, Seoul National University (Advisor: Kwangkeun Yi)
 - 2012 UC Berkeley visiting researcher (Advisor: Dawn Song)

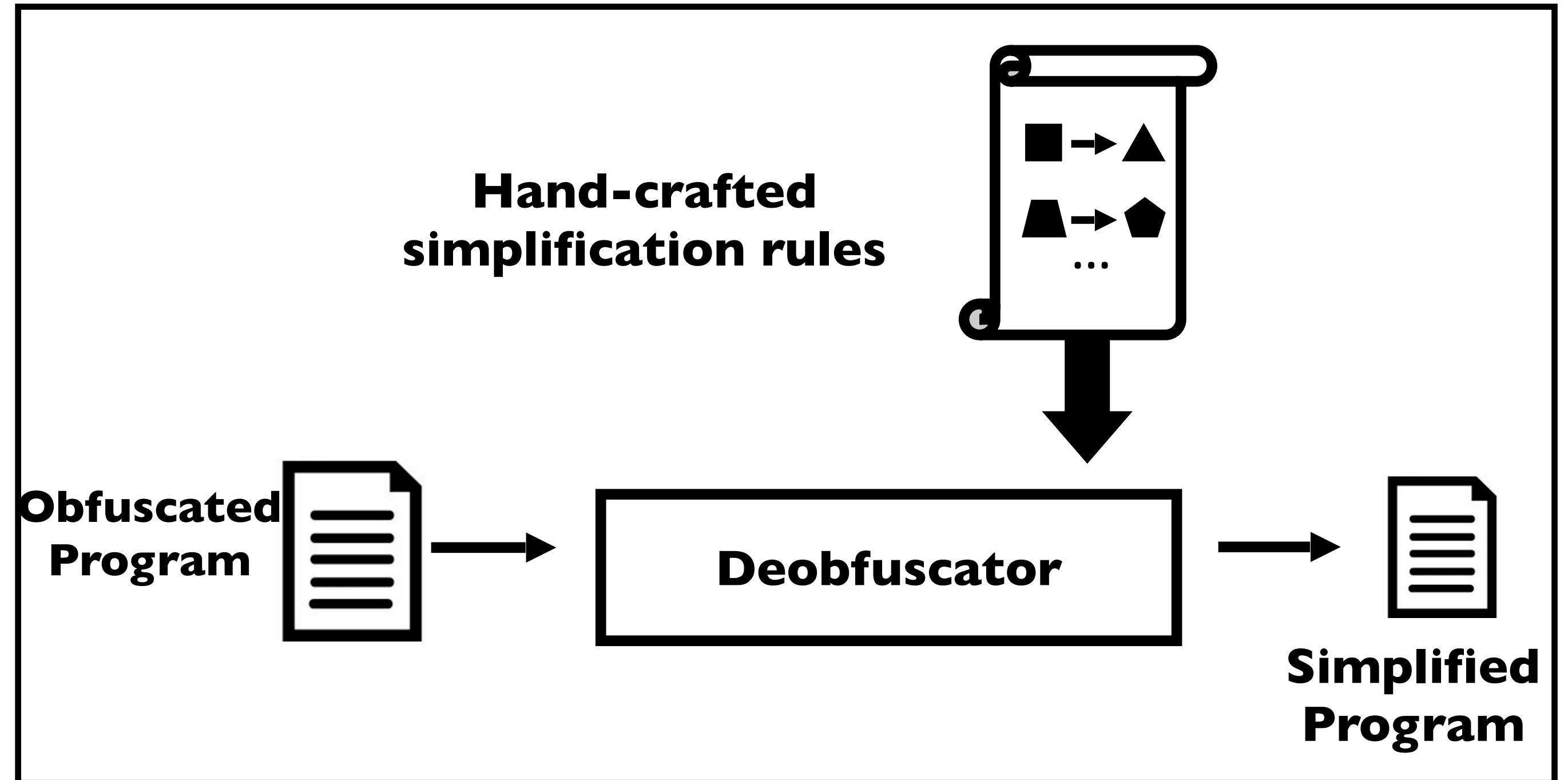
Code Obfuscation



- Produces Hard-to-Analyze Code
- Semantic-Preserving Transformation
- For: evasion of malware detection 🐈 copyright protection 😊

Deobfuscation by Rewriting

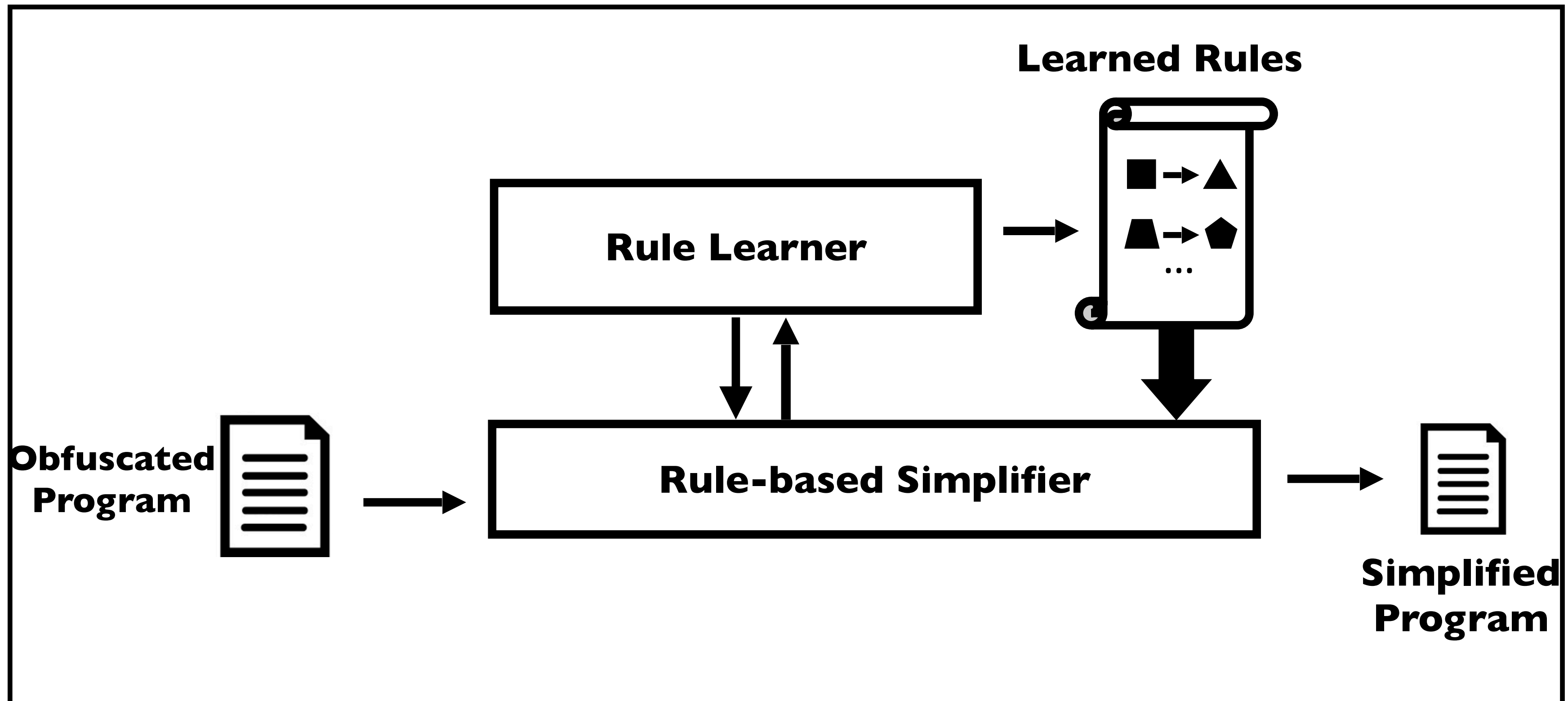
- Transforming into a simpler program
- Applying simplification rules (e.g., $x + 0 \rightarrow x$) (x can be anything)



- Rules in prior methods (e.g., SSPAM)
 - Hand-crafted by domain experts
 - Limited search space
- Application order in prior methods
 - Heuristics by domain experts
 - May miss the optimal solutions

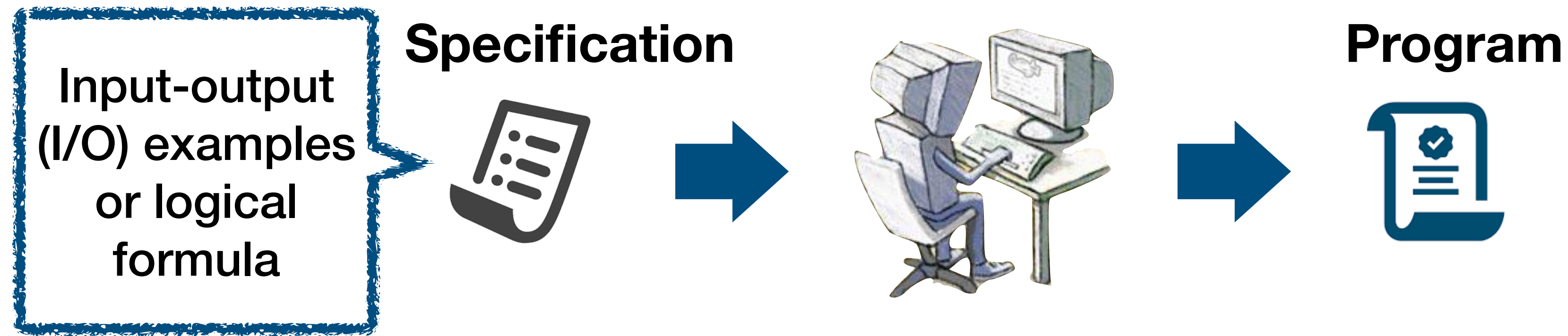
Our Solution

Discovering new rules (by *Program synthesis*) + Systematically applying the rules (by *Term rewriting*) + Finding optimal solutions by exhaustive search (by *Equality saturation*)



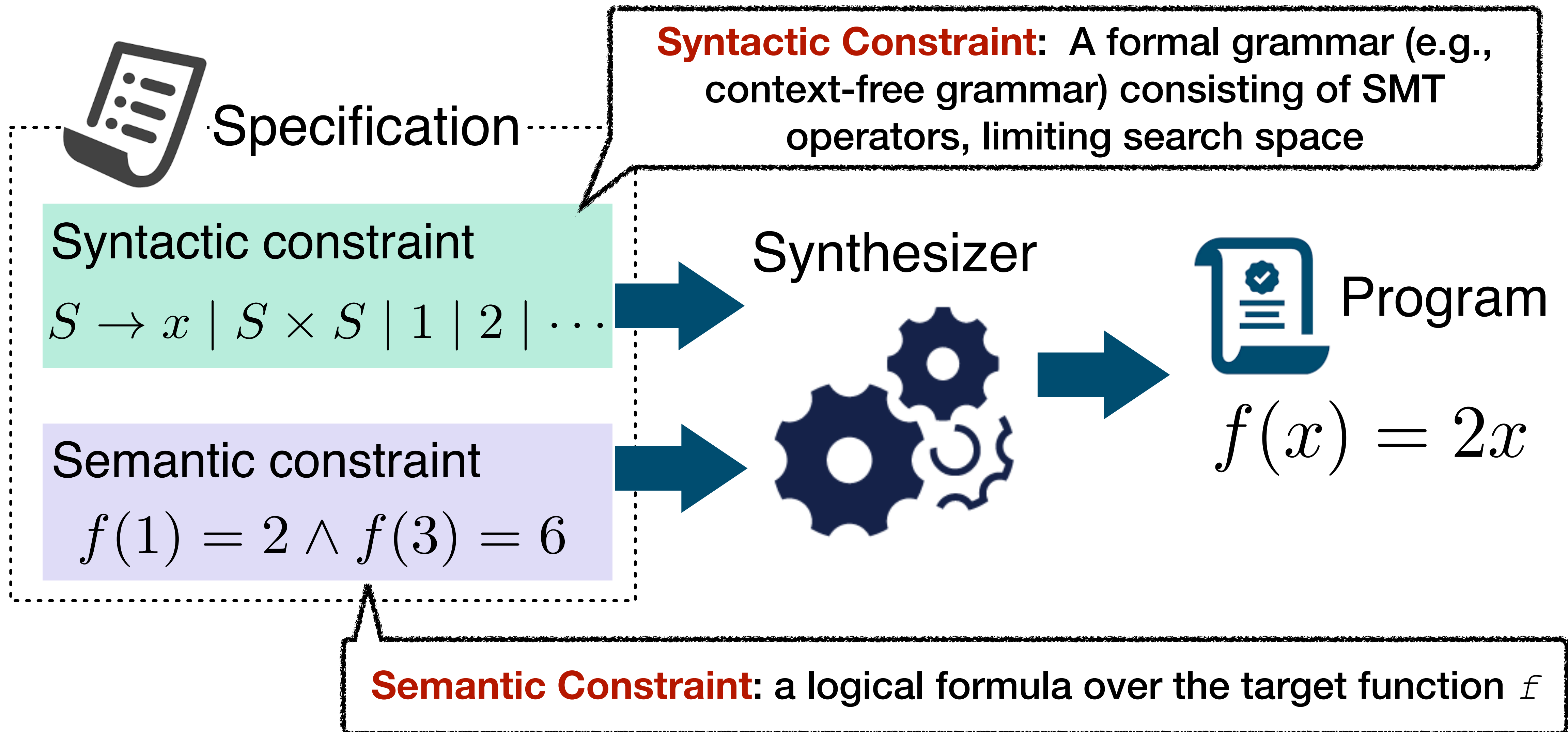
Enabled by *Program Synthesis*

- Automatically finding programs satisfying user intent



- Difference from LLMs?
 - Search algorithms based on programming languages theory
 - (If it finishes) 100% guarantee of correctness
 - Better at discovering new programs (e.g., compiler superoptimization)

Syntax-guided Synthesis (SyGuS) Standard



Which Obfuscation?

- **Mixed Boolean Arithmetic (MBA) expressions:** with bitwise logical operators (AND, OR, XOR...) and arithmetic operators (+, -, *, /, %, ...)
 - e.g., $8458(x \vee y \wedge z)^3 ((xy) \wedge x \vee t) + x + 9(x \vee y)yz^3$
- **MBA obfuscation:** transforming arbitrary expressions into highly complex MBA expressions while maintaining their meaning

MBA Obfuscation is Popular

- The cost of obfuscation and executing obfuscated code is low.
 - Only basic operations are added, and the execution flow remains unchanged (no additional calls to user/system functions, etc.)
- Theoretical foundation: any bitwise expressions can be obfuscated in infinitely many ways. Deobfuscation is NP-hard.
- Widely adopted by various tools
 - Code obfuscation (Tigress, VMProtect)
 - DRM (Irdeto)
 - **Being used for malware**



Previous Approaches for MBA Deobfuscation

- **Term Rewriting** : SSPAM [Eyrolles et al. 2016]
- **Program Synthesis** : Syntia [Blazytko et al. 2017], QSynth [David et al. 2020], Xyntia [Menguy et al. 2021]
- **Neural Network Inference** : NeuReduce [Feng et al. 2020]
- **Algebraic Methods** : MBA-Solver [Xu et al. 2021], SiMBA [Reichenwallner et al. 2022]

Previous Approaches for MBA Deobfuscation

- **Term Rewriting** : SSPAM [Eyre et al. 2019] Cannot handle a wide range of MBA obfuscation rules
- **Program Synthesis** : Syntia [Blazy et al. 2020], Xyntia [Menguy et al. 2021] No Guarantee of Correctness
- **Neural Network Inference** : NeuReduce [Feng et al. 2021] Unscalable
- **Algebraic Methods** : MBA-Solver [Xu et al. 2021], SiMBA [Reichenwallner et al. 2022] Limited to a specific class of MBA expressions

Our Goal

Cannot handle a wide range of MBA obfuscation rules

- Term Rewriting : SSPAM [Eyraud et al., 2015]

To overcome the limitations, we should achieve :

- **Soundness** : Guarantee of correctness
- **Generality** : Covers arbitrary MBA expressions
- **Flexibility** : Irrespective of obfuscation rules
- **Scalability** : Handles huge MBA expressions

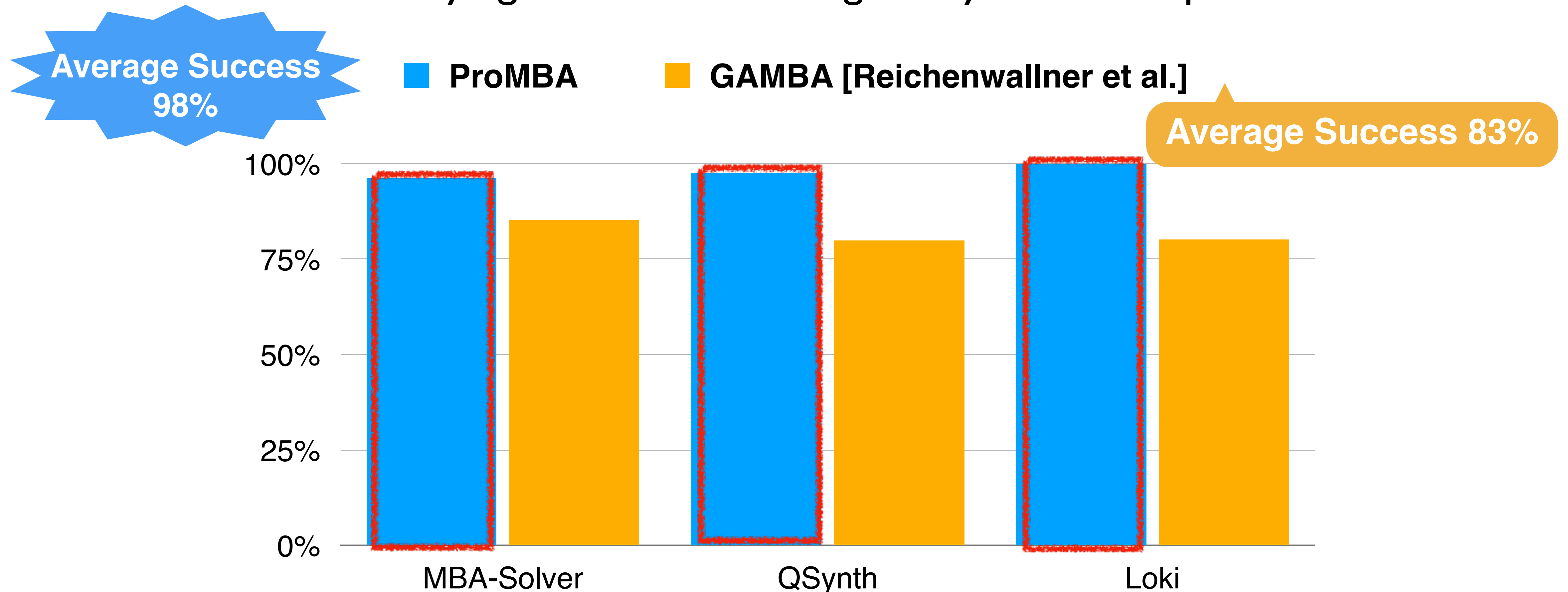
Our Contribution

- A novel and versatile method for deobfuscating arbitrary MBA expressions
 - Synergistic combination of program synthesis, term rewriting, and equality saturation
 - Two novel methods: *on-the-fly learning*, *scoped equality saturation*
- Tool (**ProMBA**)
<https://github.com/astean1001/ProMBA>



Result in a Nutshell

- **Success** : $| \text{deobfuscation result} | \leq | \text{original program} |$ & semantic equivalence
 - The size is measured by # of AST nodes
- Better than SOTA relying on heuristics designed by domain experts



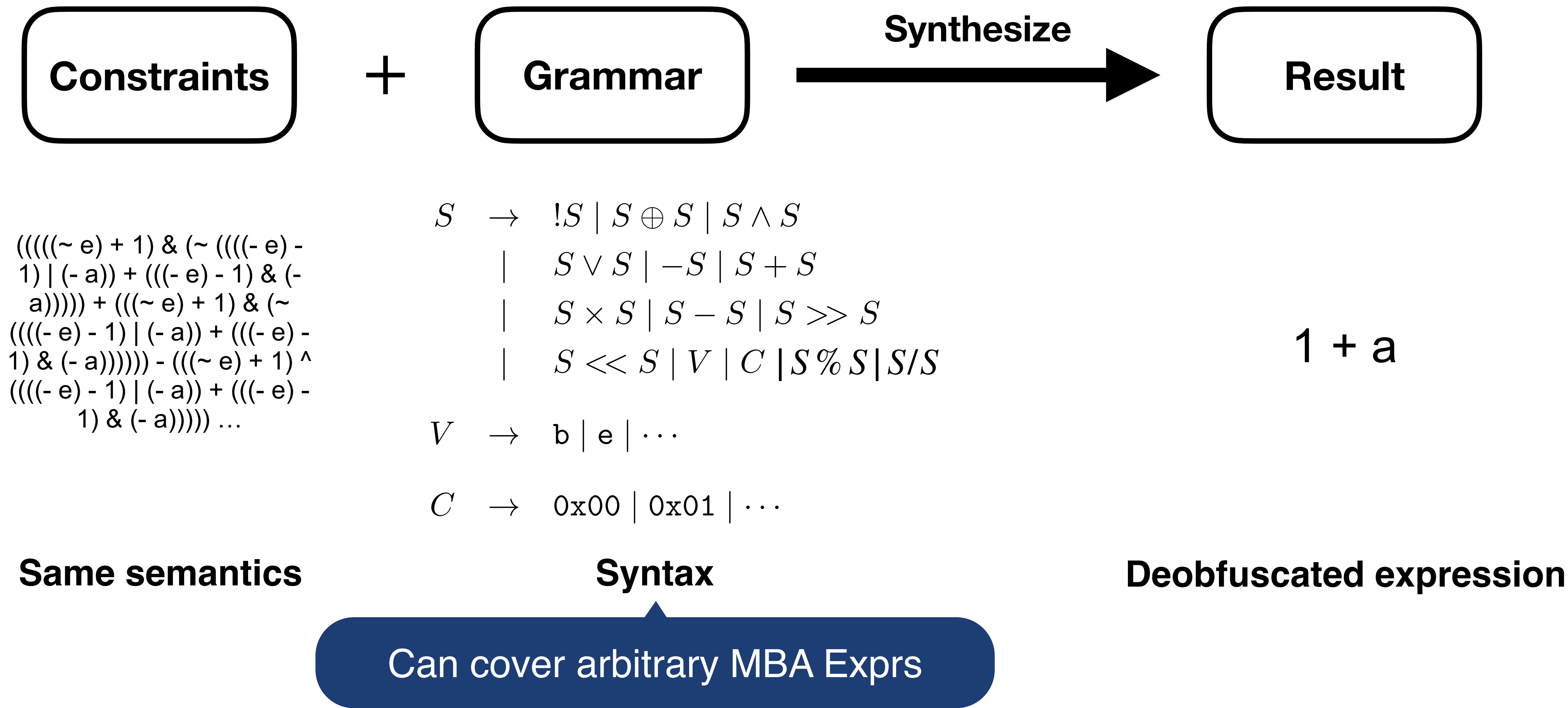
Papers

- MBA Deobfuscation with Program Synthesis and Term Rewriting
 - Jaehyung Lee and Woosuk Lee, *Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Term Rewriting*, **ACM CCS 2023**
- Advanced MBA Deobfuscation with Scoped Equality Saturation
 - Jaehyung Lee, Seoksu Lee, Eunsun Cho and Woosuk Lee, *Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Equality Saturation*, **ACM TOSEM (Under Review)**
- Enabling technology: high-performance program synthesis
 - Yongho Yoon, Woosuk Lee, and Kwangkeun Yi, *Inductive Program Synthesis via Iterative Forward-Backward Abstract Interpretation*. **ACM PLDI 2023**
 - Doyoon Lee, Woosuk Lee, and Kwangkeun Yi, *Inductive Program Synthesis by Meta-Analysis-Guided Hole Filling* **ACM POPL 2026 (conditionally accepted)**

Contents

- **MBA Deobfuscation with Program Synthesis and Term Rewriting**
 - Jaehyung Lee and Woosuk Lee, *Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Term Rewriting*, ACM CCS 2023
- Advanced MBA Deobfuscation with Scoped Equality Saturation
- Enabling technology: high-performance program synthesis

Program Synthesis-Based MBA Deobfuscation



Challenge : Scalability

$$\begin{aligned} &((((\sim e) + 1) \& (\sim (((- e) - \\ & 1) \mid (- a)) + (((- e) - 1) \& (- \\ & a)))) + (((\sim e) + 1) \& (\sim \\ & ((((- e) - 1) \mid (- a)) + (((- e) - \\ & 1) \& (- a)))) - (((\sim e) + 1) \wedge \\ & ((((- e) - 1) \mid (- a)) + (((- e) - \\ & 1) \& (- a)))) \dots \end{aligned}$$

Synthesize

$1 + a$

Too Slow! 🤨

size of obfuscated expression $\uparrow \rightarrow$ synthesis performance $\downarrow$

Solution 1 : Divide-and-Conquer

$$\begin{aligned} & (((((\sim e) + 1) \& (\sim (((- e) - \\ & 1) \mid (- a)) + (((- e) - 1) \& (- \\ & a)))))) + (((\sim e) + 1) \& (\sim \\ & ((((- e) - 1) \mid (- a)) + (((- e) - \\ & 1) \& (\sim e)))))) + (((\sim e) + 1) \& \end{aligned}$$

Synthesize 

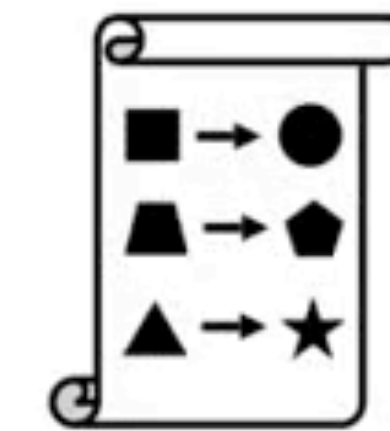
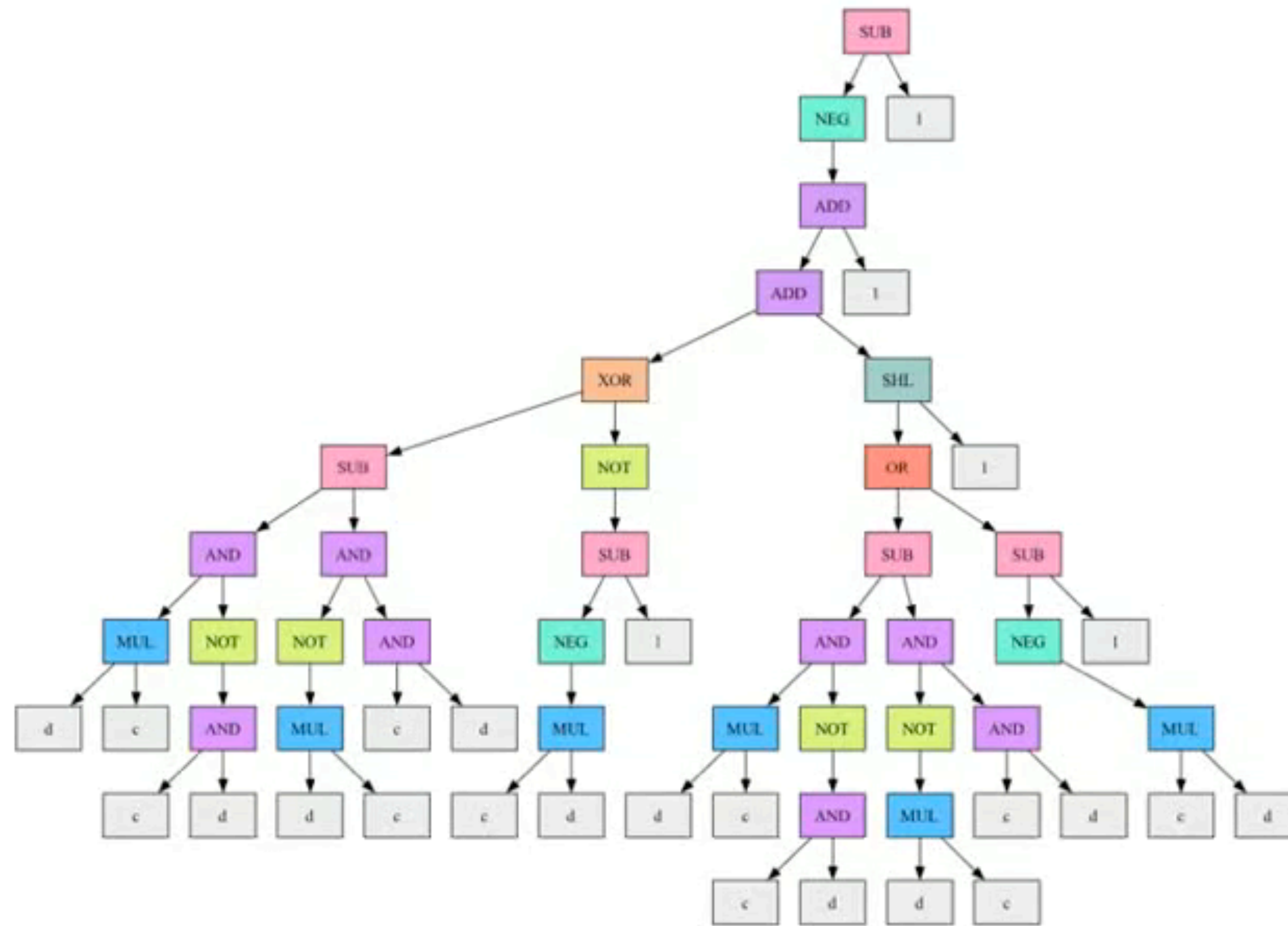
$1 + a$

Our Solution : Simplifies a given MBA expression by recursively simplifying its sub-expressions. (divide-and-conquer)

size of obfuscated expression $\uparrow$ $\rightarrow$ deobfuscation performance $\downarrow$

Solution 2 : Learning Successful Synthesis Patterns On-the-fly

- (1) **Synthesize a minimal equivalent expression** for a chosen subexpression
- (2) If a smaller expression found, **add the transformation as a rewrite rule** and apply to the other parts of the obfuscated expression.

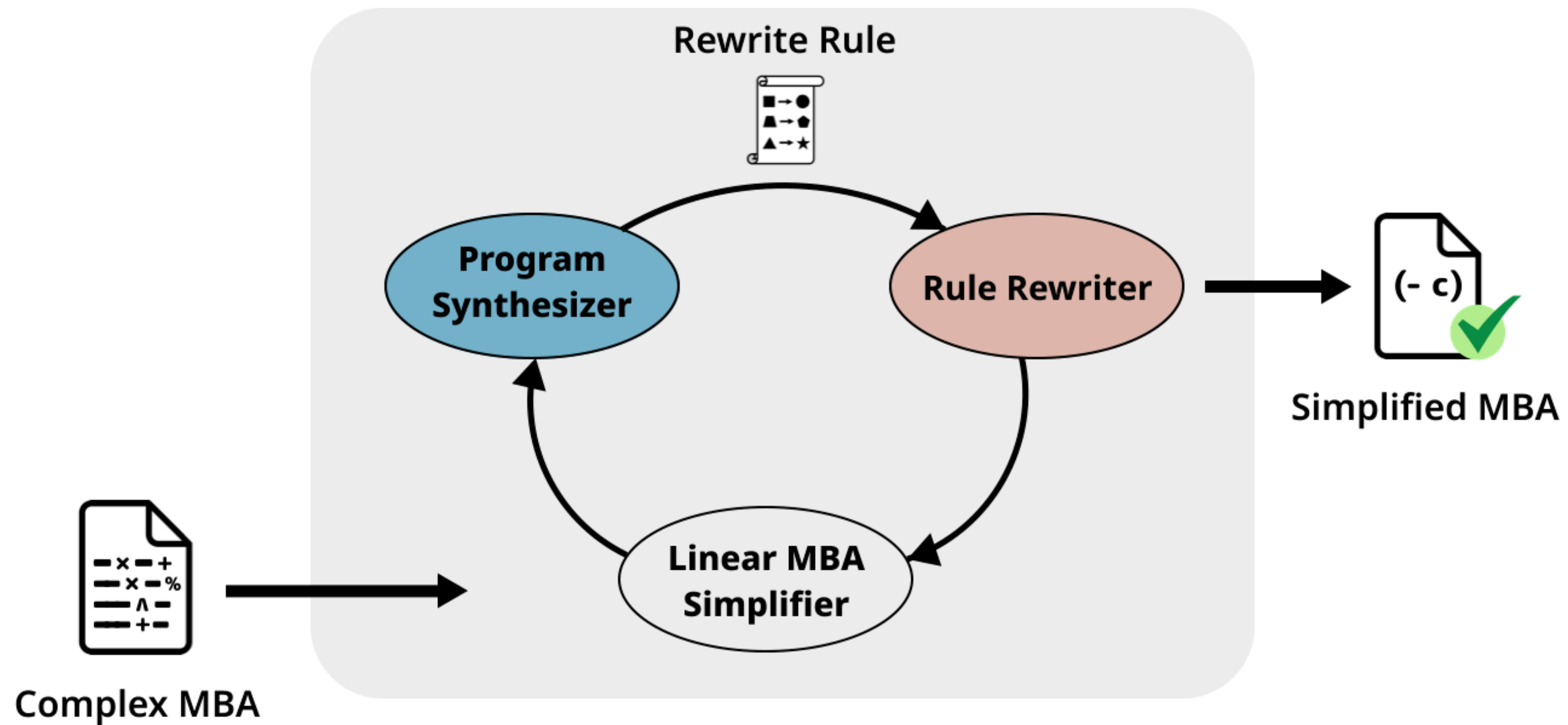


Rewrite Rules



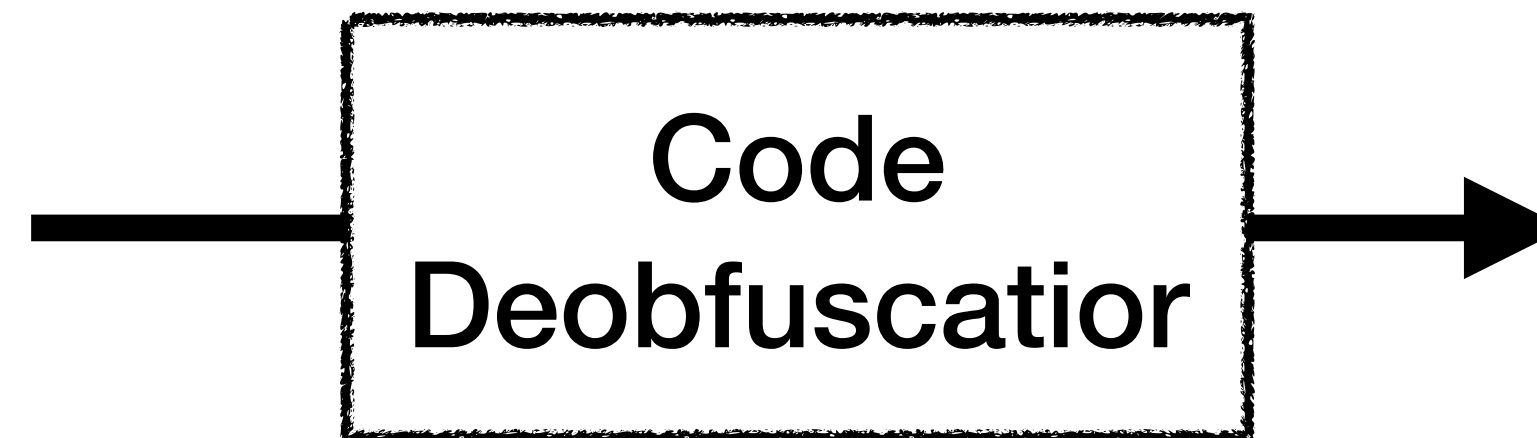
Program Synthesizer

Overall Architecture



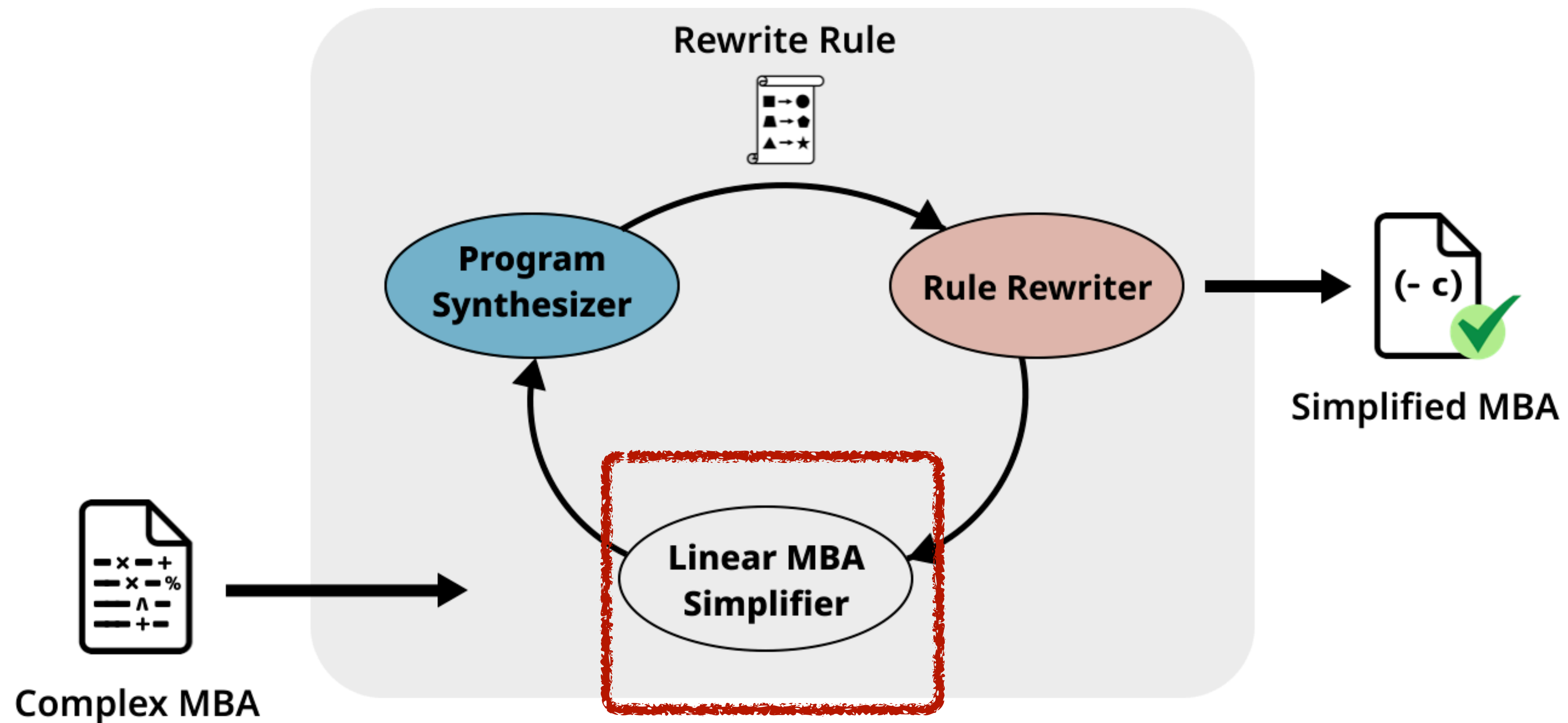
Illustrative Example

$$\begin{aligned} & (((((\sim e) + 1) \& (\sim (((- e) - 1) \\ & | (- a)) + (((- e) - 1) \& (- a)))))) \\ & + (((\sim e) + 1) \& (\sim (((- e) - 1) \\ & | (- a)) + (((- e) - 1) \& (- a)))))) \\ & - (((\sim e) + 1) ^ ((((- e) - 1) | (- \\ & a)) + (((- e) - 1) \& (- a)))) ... \end{aligned}$$



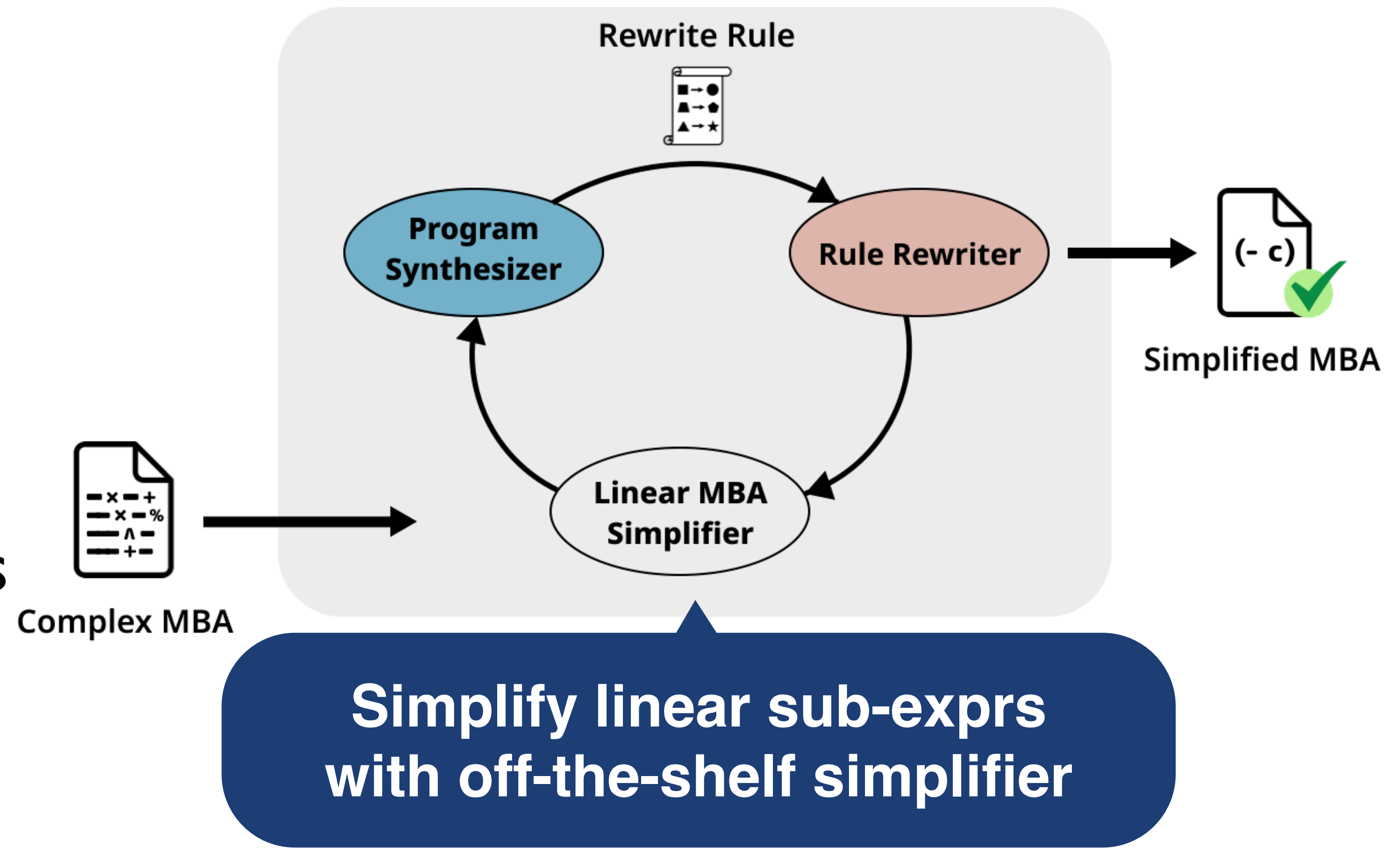
$1 + a$

Applying Linear Simplifier



Applying Linear Simplifier

- Linear combination of bitwise logical operations
 - e.g.,
$$(x \oplus (\neg y)) - 2(x \vee y) + 1$$
- Various off-the-shelf tools for linear MBA
 - We use SiMBA [Reichenwallner 2022].

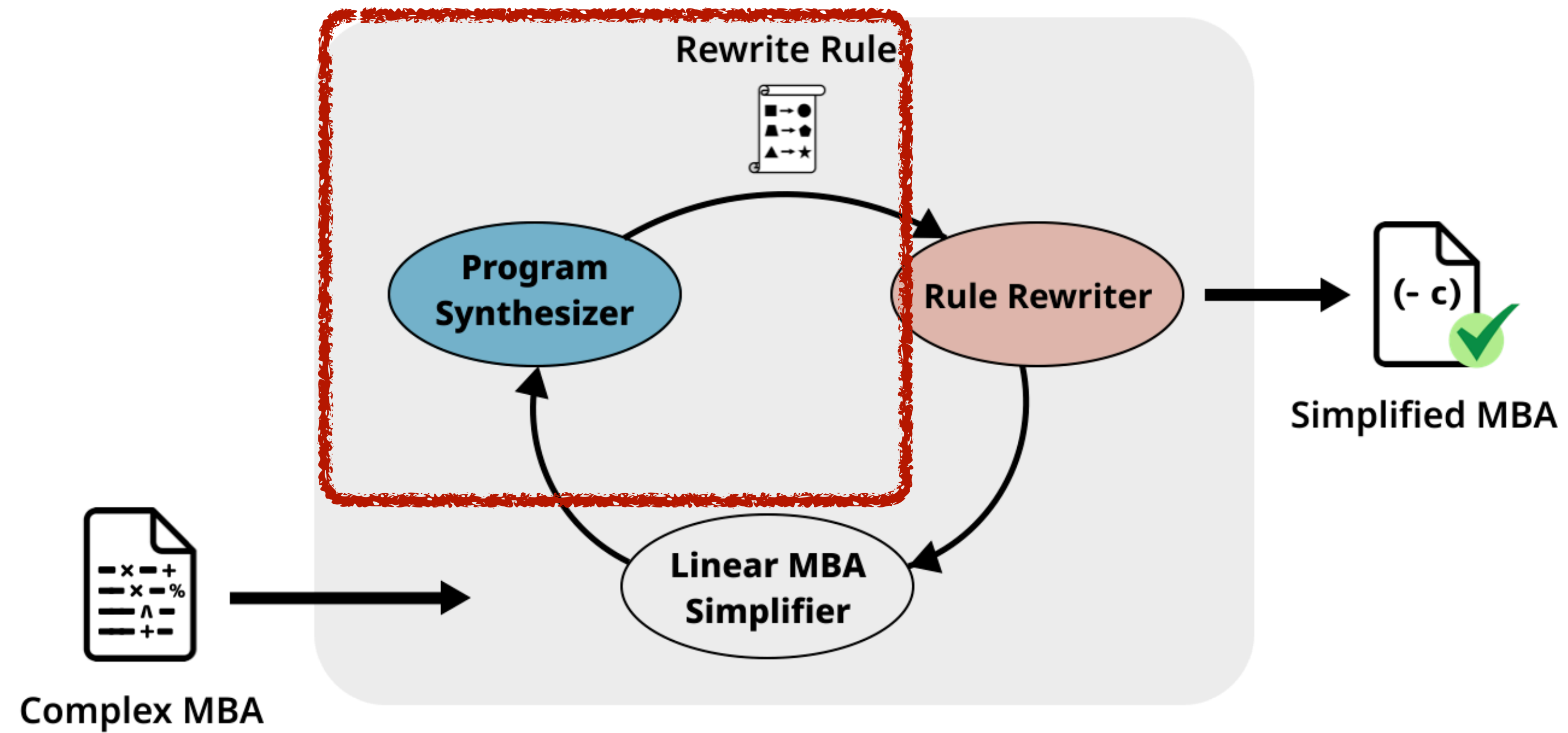


Apply Linear Simplifier

$(((((\sim e) + 1) \& (\sim (((- e) - 1) \mid (- a)) + (((- e) - 1) \& (- a)))))) + (((\sim e) + 1) \& (\sim (((- e) - 1) \mid (- a)) + (((- e) - 1) \& (- a)))))) -$
 $((((\sim e) + 1) ^ ((((- e) - 1) \mid (- a)) + (((- e) - 1) \& (- a)))))) \dots$

$((\sim e) + 1)$	Simplify 	$- e$
$((- e) - 1)$		$\sim e$

Synthesize Subexpression



Choose Subexpression

$((((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) +$
 $((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) \dots$

$(((((e + a) * (e \mid (-a))) \& (\sim (-e))) + (e + a)) -$
 $(((-e) \mid ((e + a) * (e \mid (-a)))) \& ((e + a) - (-e))))$

$\dots$

Challenge : Still Unscalable

$$\begin{aligned} & ((((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) + \\ & ((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) \dots \\ & (((((e+a) * (e \mid (-a))) \& (\sim (-e))) + (e+a)) - \\ & (((-e) \mid ((e+a) * (e \mid (-a)))) \& ((e+a) - (-e)))) \end{aligned}$$

...

$$\begin{aligned} & (((((e+a) * (e \mid (-a))) \& (\sim (-e))) + (e+a)) - \\ & (((-e) \mid ((e+a) * (e \mid (-a)))) \& ((e+a) - (-e)))) \end{aligned}$$

Synthesize

?

Still too big 😡

Solution : Changing Sub-Expressions to Variables

Substitution

$$\begin{aligned} v1 &\rightarrow (-e) \\ v2 &\rightarrow (e + a) \\ v3 &\rightarrow (e \mid (-a)) \end{aligned}$$

$$\begin{aligned} &((((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a)))))) + \\ &((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a)))))) \dots \\ &(((((e + a) * (e \mid (-a))) \& (\sim (-e))) + (e + a)) - \\ &((((-e) \mid ((e + a) * (e \mid (-a)))) \& ((e + a) - (-e)))) \\ &\dots \end{aligned}$$

$$\begin{aligned} &(((v2 * v3) \& (\sim v1)) + v2) \\ &- ((v1 \& (v2 * v3)) \& (v2 - v1)) \end{aligned}$$

Synthesize

$$(v1 \& ((v2 * v3) \& (v2 - v1)))$$

Scalable 😊

Learn Rewrite Rule

Rewrite Rule

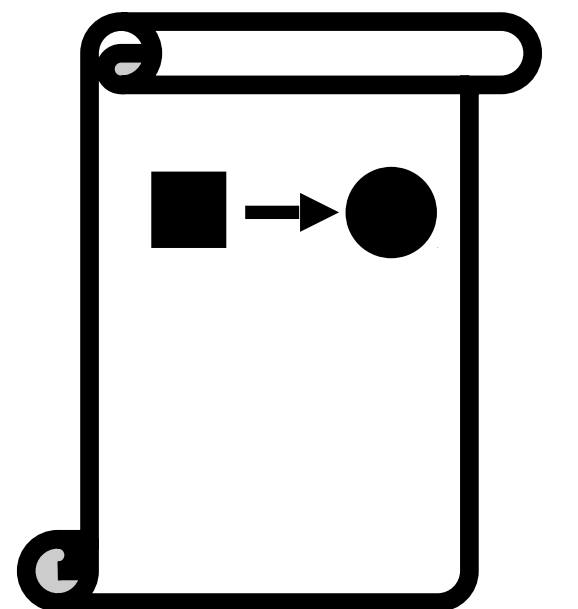
Substitution

$v1 \rightarrow (-e)$
 $v2 \rightarrow (e + a)$
 $v3 \rightarrow (e \mid (-a))$

$((((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) +$
 $((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) \dots$
 $(v1 \& ((v2 * v3) \& (v2 - v1)))$

...

Learn Rule



Synthesize

$((v2 * v3) \& (\sim v1)) + v2$
 $- ((v1 \& (v2 * v3)) \& (v2 - v1))$

$(v1 \& ((v2 * v3) \& (v2 - v1)))$

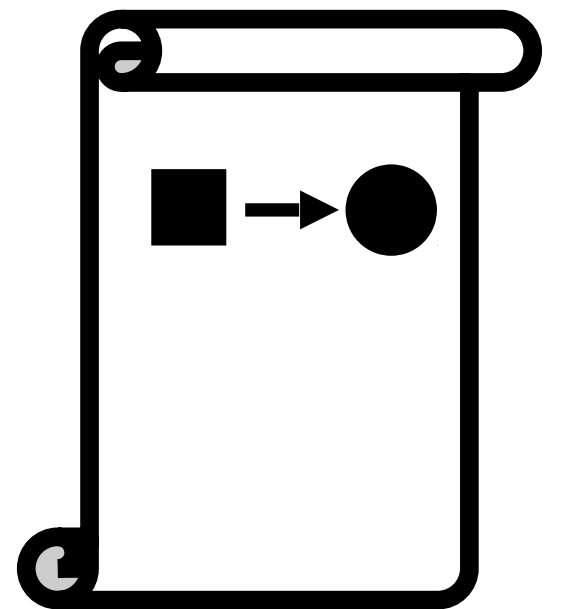
Substitute Variables

Substitution

$v1 \rightarrow (-e)$
 $v2 \rightarrow (e + a)$
 $v3 \rightarrow (e \mid (-a))$

$((((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a)))))$
 $+ (((-e) \& (\sim (((\sim e) \mid (-a)) + ((\sim e) \& (-a))))) \dots$
 $((e + a) \& (((e \mid (\sim a)) * (\sim e)) \& ((e \mid (\sim a)) - (\sim e))))$
 $\dots$

Rewrite Rule



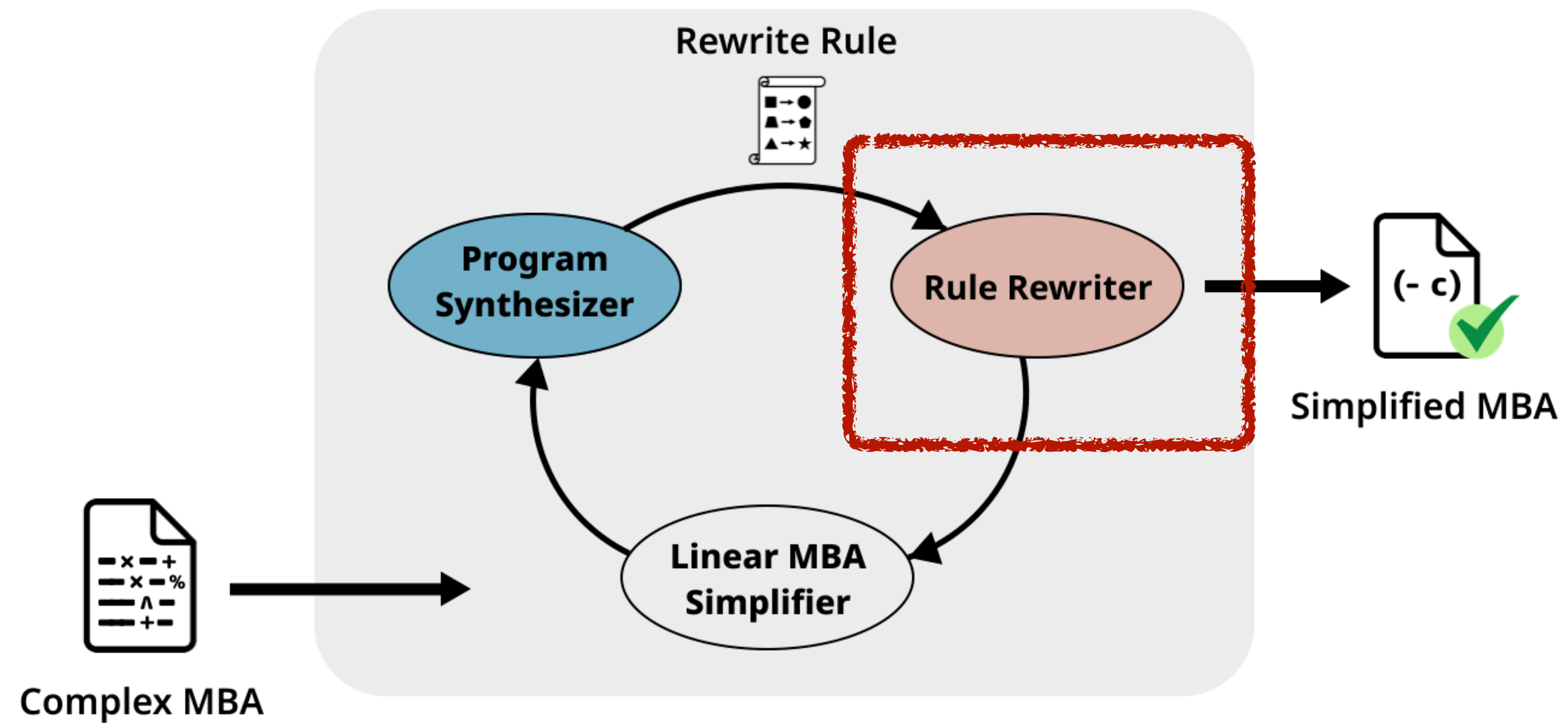
Learn Rule

$((v2 * v3) \& (\sim v1)) + v2$
 $- ((v1 \& (v2 * v3)) \& (v2 - v1))$

Synthesize

$(v1 \& ((v2 * v3) \&$
 $(v2 - v1)))$

Apply Rewrite Rule

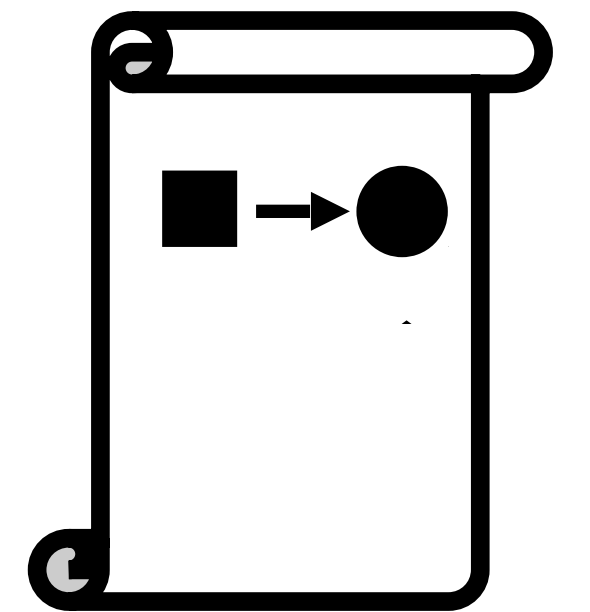


Problem : Rule seems too specific



$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))))$
 $- ((- e) \wedge (((\sim e) \mid (- a)) + ((\sim e) \& (- a)))) \dots$
 $((((\sim e) \& (\sim a)) + e) - ((a \& (\sim e)) \& (- a)) \dots$

Rewrite Rule



$((v2 * v3) \& (\sim v1)) + v2$
 $- ((v1 \& (v2 * v3)) \& (v2 - v1))$



$(v1 \& ((v2 * v3) \& (v2 - v1)))$

Solution : Generalize Rule

Rewrite Rule

$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))))$

Our Solution : Generalize existing rule

$((v2 * v3) \& (\sim v1)) + v2$
 $- ((v1 \& (v2 * v3)) \& (v2 - v1))$



$(v1 \& ((v2 * v3) \& (v2 - v1)))$

1. Identifies sub-expression that are common in both sides

Solution : Generalize Rule

Rewrite Rule

$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))))$

Our Solution : Generalize existing rule

$((y1 \& (\sim v1)) + v2)$
 $- ((v1 \& y1) \& y2)$



$(v1 \& (y1 \& y2))$

2. Replace common sub-expressions into fresh variable
3. Check correctness of the generalized rule with SMT Solver

Solution : Generalize Rule

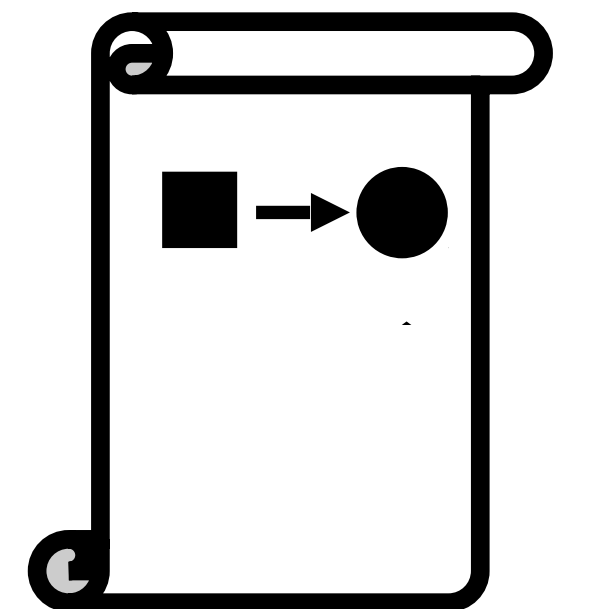
Substitution

$v1 \rightarrow a$
 $v2 \rightarrow e$
 $y1 \rightarrow (\sim e)$
 $y2 \rightarrow (- a)$

$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))))$
 $- ((- e) ^ (((\sim e) \mid (- a)) + ((\sim e) \& (- a)))) \dots$
 $((\sim e) \& (\sim a)) + e - (a \& (\sim e)) \& (- a) \dots$

Rule Match 😊

Rewrite Rule



$((y1 \& (\sim v1)) + v2)$
 $- ((v1 \& y1) \& y2)$



$(v1 \& (y1 \& y2))$

Apply Rewrite Rule

Substitution

$v1 \rightarrow a$

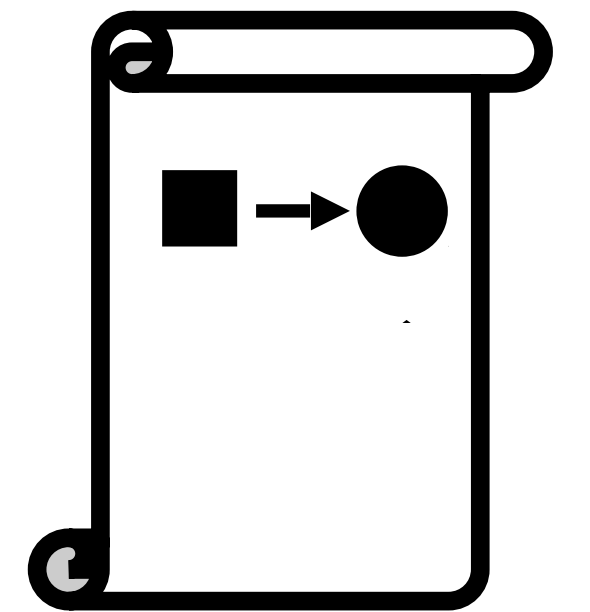
$v2 \rightarrow e$

$y1 \rightarrow (\sim e)$

$y2 \rightarrow (- a)$

$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a)))))$
 $- ((- e) ^ (((\sim e) \mid (- a)) + ((\sim e) \& (- a)))) \dots$
 $(a \& ((\sim e) \& (- a))) \dots$

Rewrite Rule



$((y1 \& (\sim v1)) + v2)$
 $- ((v1 \& y1) \& y2)$

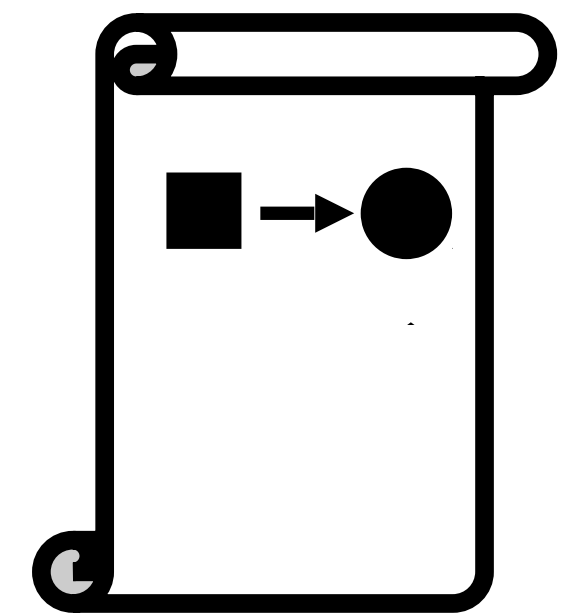


$(v1 \& (y1 \& y2))$

Choose Subexpression

$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))))$
 $- ((- e) ^ (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))$
...

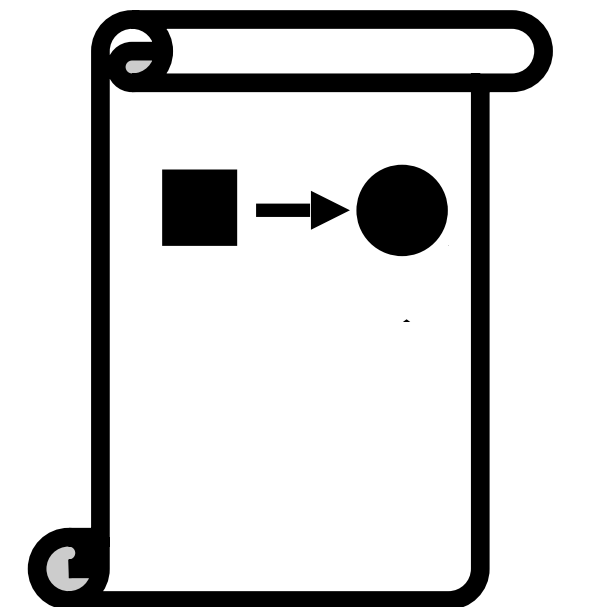
Rewrite Rule

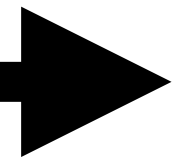


Synthesize

$((2 * ((- e) \& (\sim (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))))$
 $- ((- e) ^ (((\sim e) \mid (- a)) + ((\sim e) \& (- a))))$
...

Rewrite Rule

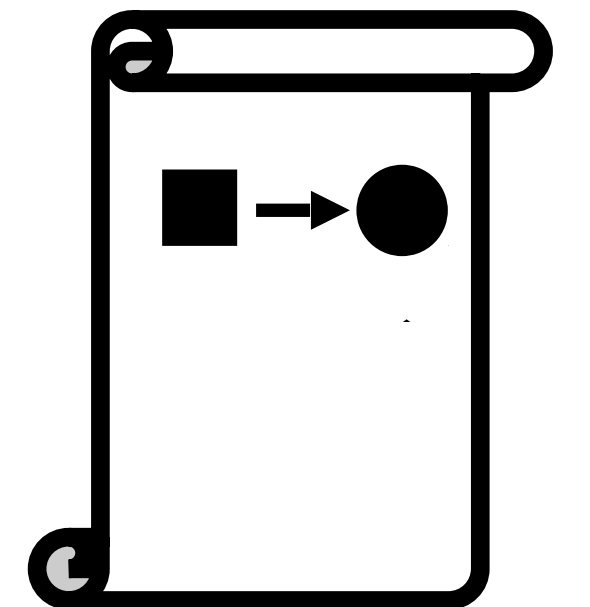


$((\sim e) \mid (- a)) + ((\sim e) \& (- a))$ **Synthesize** 

Synthesize

$((2 * ((- e) \& (\sim (((\sim e) | (- a)) + ((\sim e) \& (- a))))))$
 $- ((- e) ^ (((\sim e) | (- a)) + ((\sim e) \& (- a))))$
...

Rewrite Rule



$((\sim e) | (- a)) +$
 $((\sim e) \& (- a))$

Synthesize



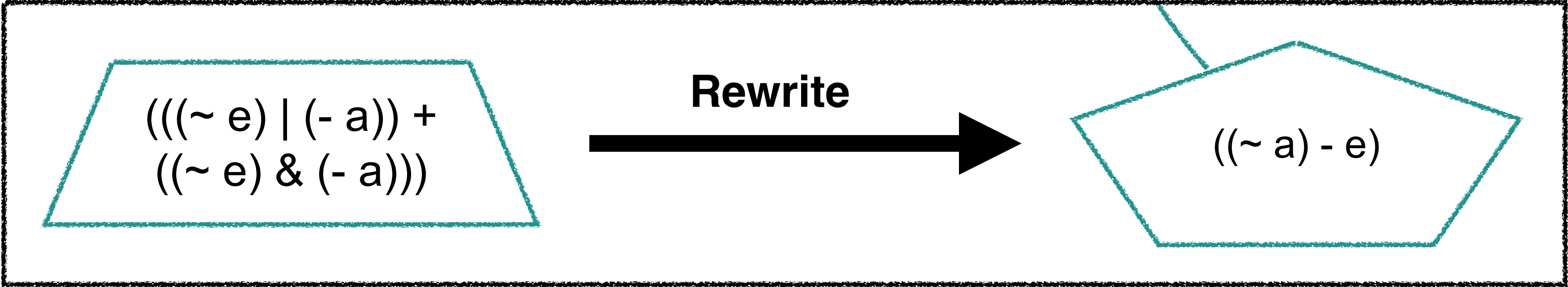
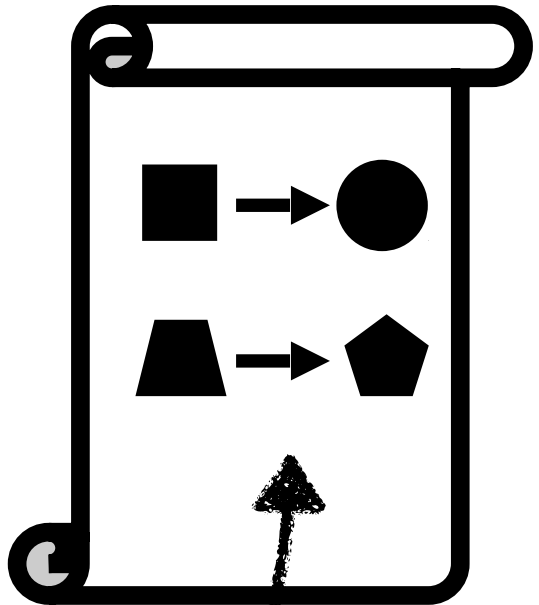
$((\sim a) - e)$

Synthesize Successful 😊

Add Rule

$$\begin{aligned} & ((2 * ((- e) \& (\sim ((\sim a) - e)))) \\ & - ((- e) ^ (((\sim e) | (- a)) + ((\sim e) \& (- a)))) \\ & \dots \end{aligned}$$

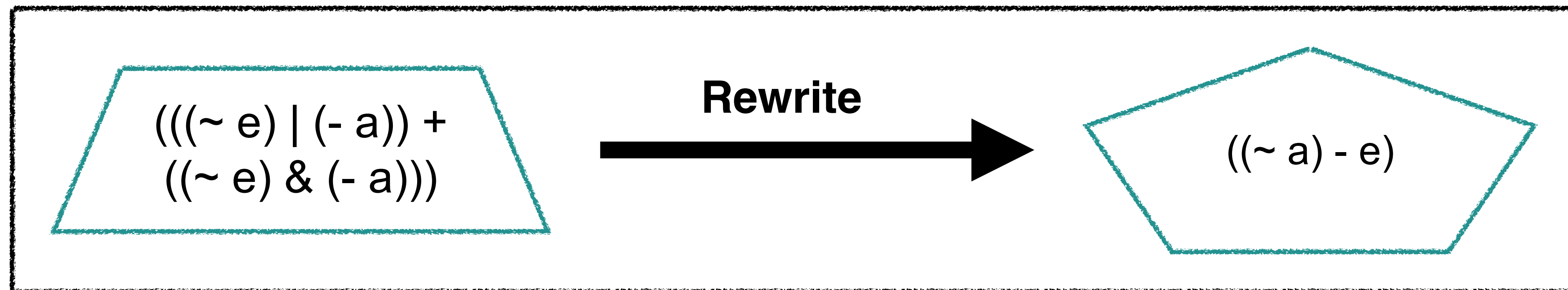
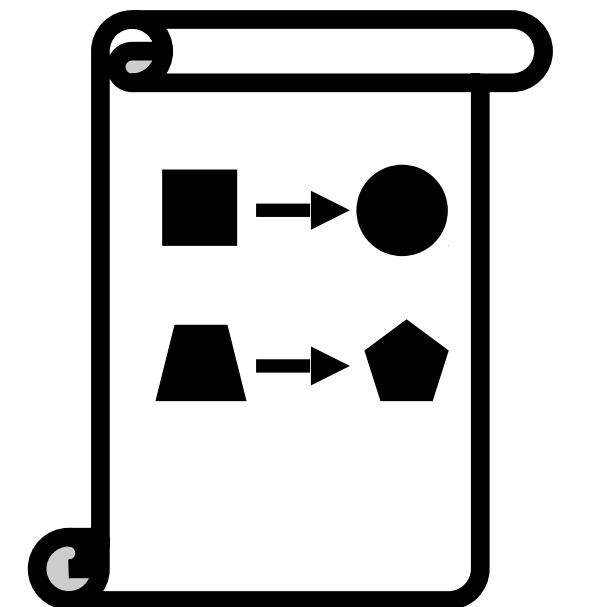
Rewrite Rule



Apply Rules

$((2 * ((- e) \& (\sim ((\sim a) - e))))$
 $- ((- e) ^ (((\sim e) | (- a)) + ((\sim e) \& (- a))))$
 $\dots (((\sim e) | (- a)) + ((\sim e) \& (- a))) \dots$
 $((\sim e) | (- a)) + ((\sim e) \& (- a)) \dots$

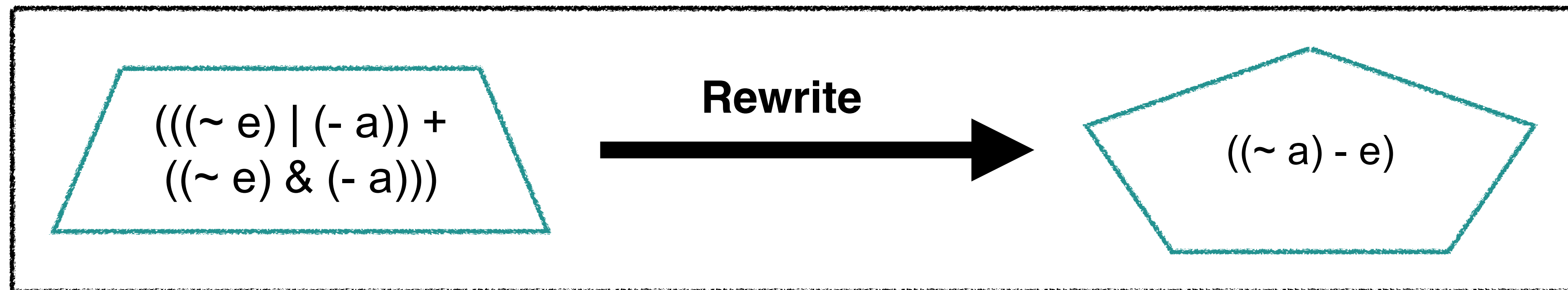
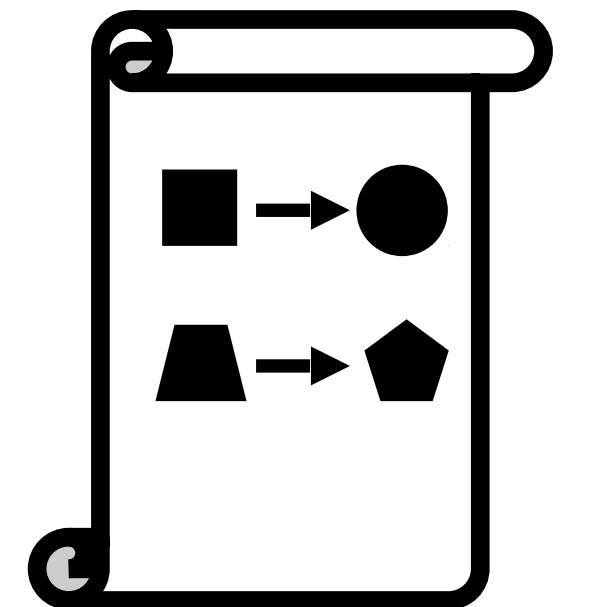
Rewrite Rule



Apply Rules

$$\begin{aligned} & ((2 * ((- e) \& (\sim ((\sim a) - e)))) \\ - & ((- e) ^ (((\sim e) | (- a)) + ((\sim e) \& (- a)))) \\ & \dots ((\sim a) - e) \dots \\ & ((\sim a) - e) \dots \end{aligned}$$

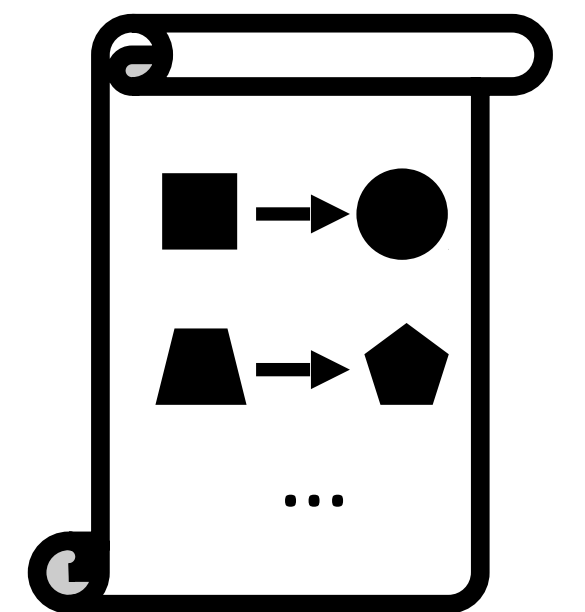
Rewrite Rule



Choose Subexpression

$((2 * ((- e) \& (\sim (((\sim a) + (- e)))))$
 $- ((- e) ^ (((\sim a) + (- e))))$

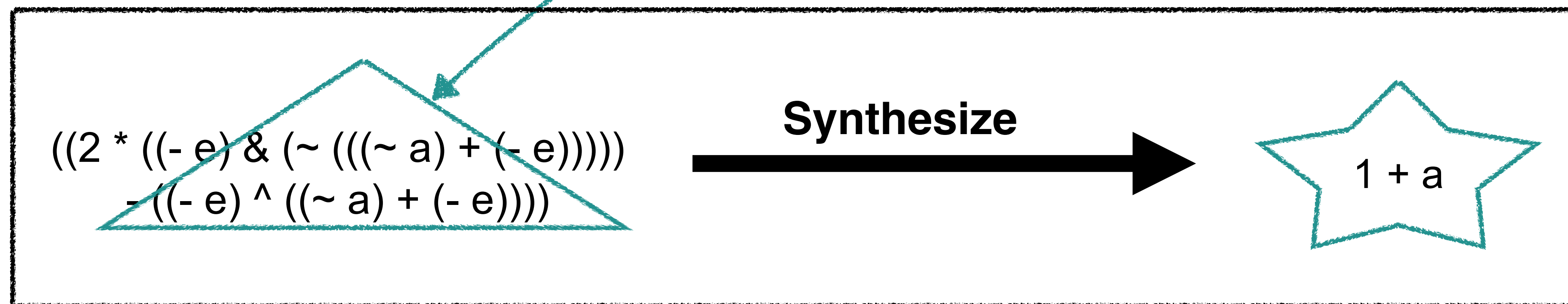
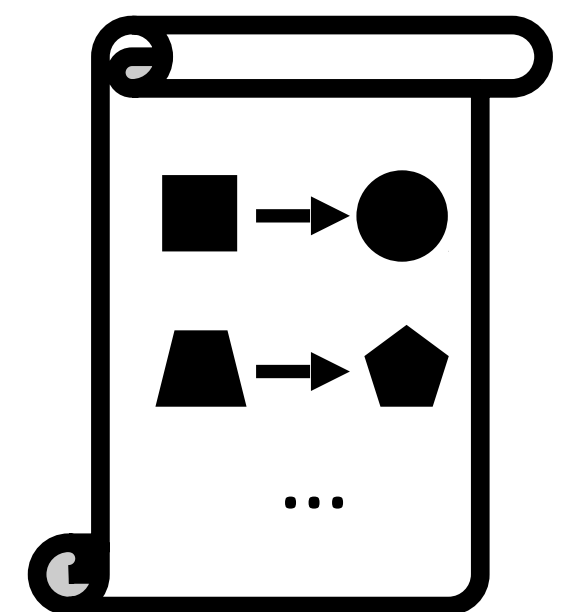
Rewrite Rule



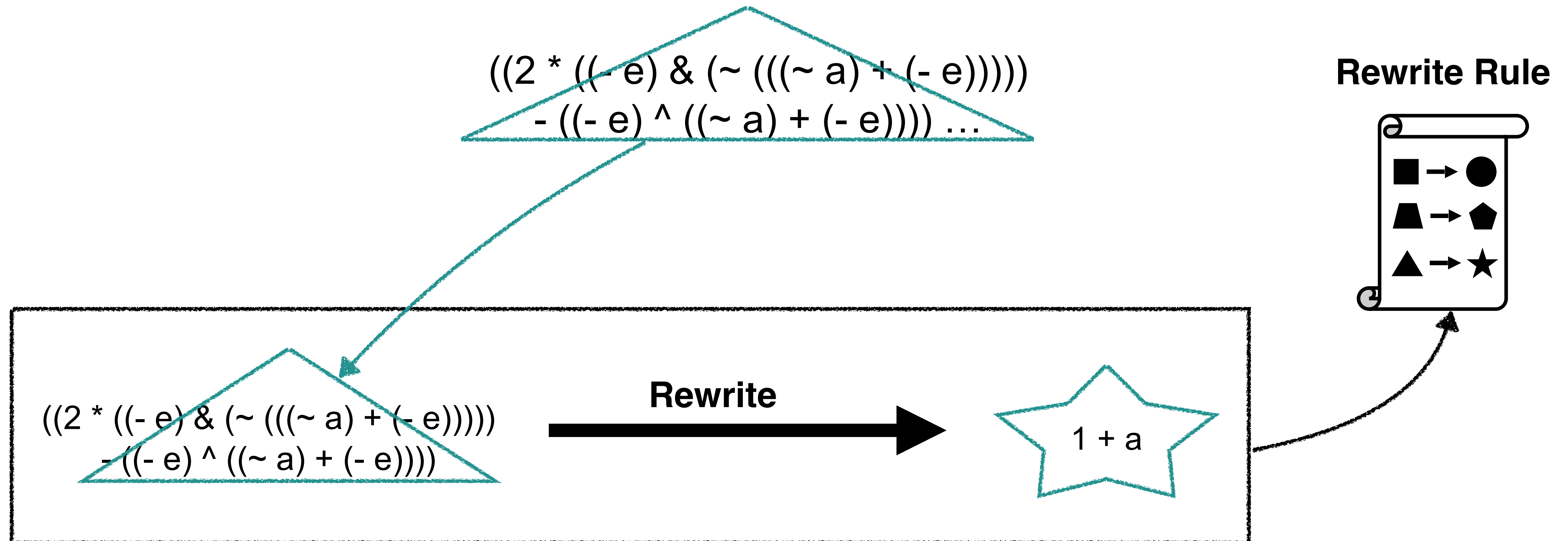
Synthesize

$$\begin{aligned} & ((2 * ((- e) \& (\sim (((\sim a) + (- e))))) \\ & - ((- e) ^ ((\sim a) + (- e)))) \end{aligned}$$

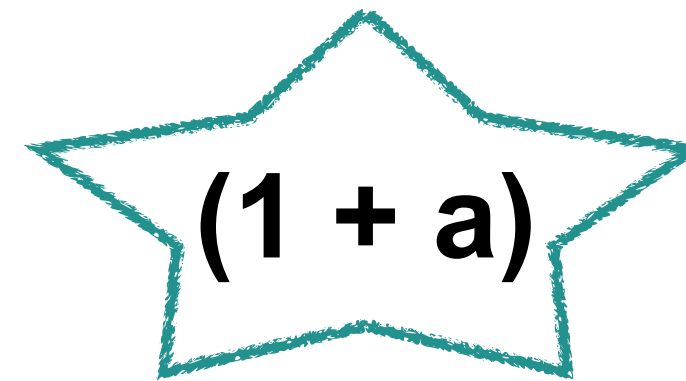
Rewrite Rule



Learn Rule

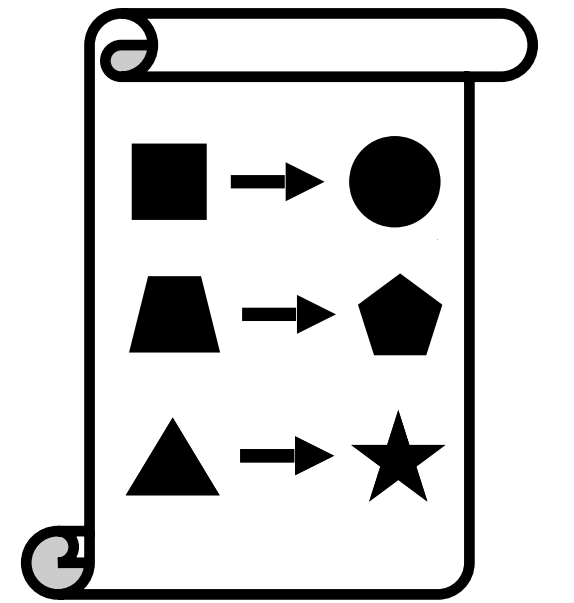


Apply Rules



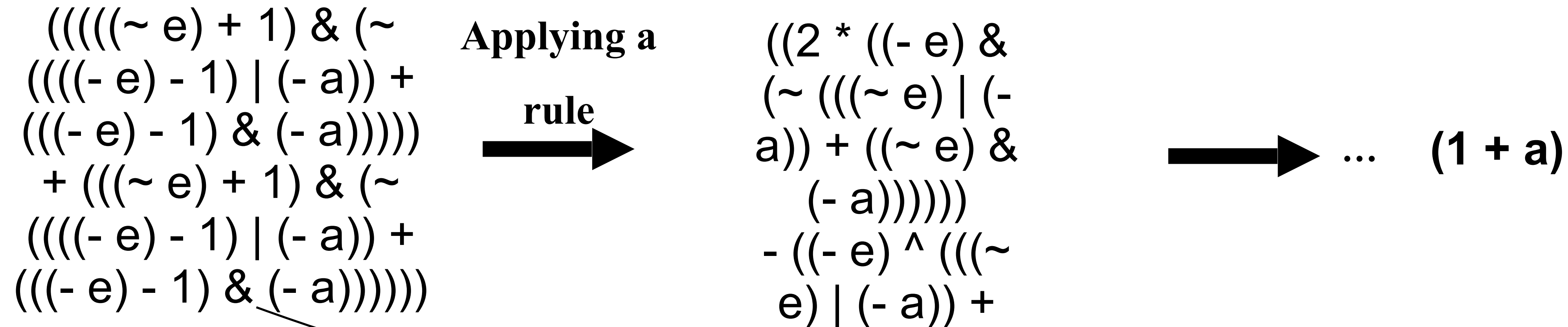
Deobfuscation Completed! 😊

Rewrite Rule



Formal Properties

(Soundness) semantics unchanged



(Termination) finitely many rule applications

Goals Achieved

Our technique achieves the following properties :

- **Sound** : By soundness of synthesis and term rewriting
- **General** : By synthesizing arbitrary MBA expression
- **Flexible** : By applying rules learned on-the-fly
- **Scalable** : By divide-and-conquer

Brief Experimental Result

- **Success Rate**

(1) size of result $\leq$ size of groundtruth (measured by by # of AST nodes)

(2) semantic equivalence

- For 4000 MBA obfuscated expression, success rate of **84%**

- **Can we do better?**

Contents

- MBA Deobfuscation with Program Synthesis and Term Rewriting
- **Advanced MBA Deobfuscation with Scoped Equality Saturation**
 - Jaehyung Lee, Seoksu Lee, Eunsun Cho and Woosuk Lee, *Simplifying Mixed Boolean-Arithmetic Obfuscation by Program Synthesis and Equality Saturation*, **ACM TOSEM (Under Review)**
- Enabling technology: high-performance program synthesis

Limitation: Phase-Ordering Problem



VS



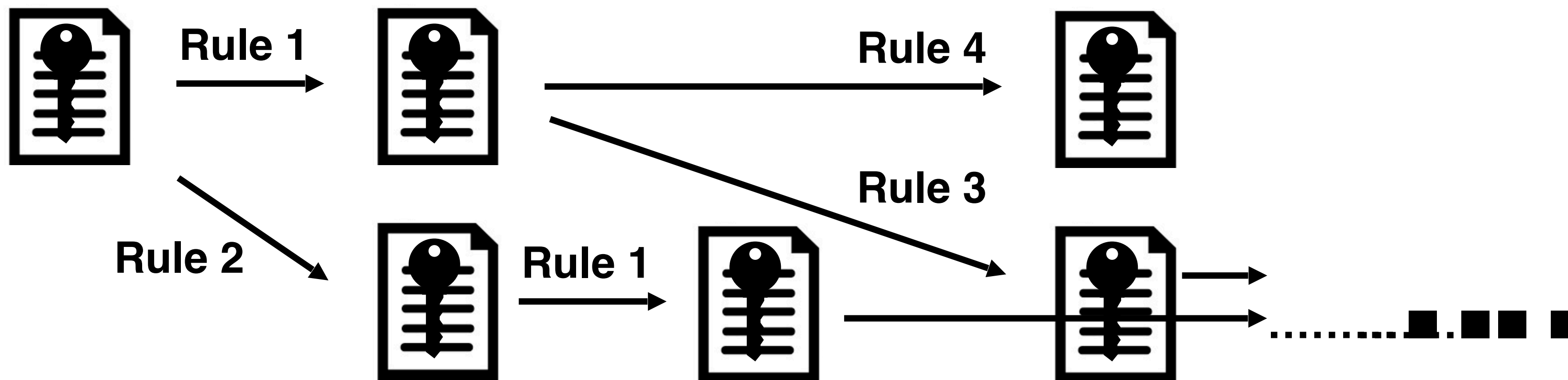
Different outcome
depending on the
application order

Naive Existing Solutions

- Using a pre-defined application order — **suboptimal!**



- Backtracking (i.e., maintaining top-k candidates) — **suboptimal and expensive**

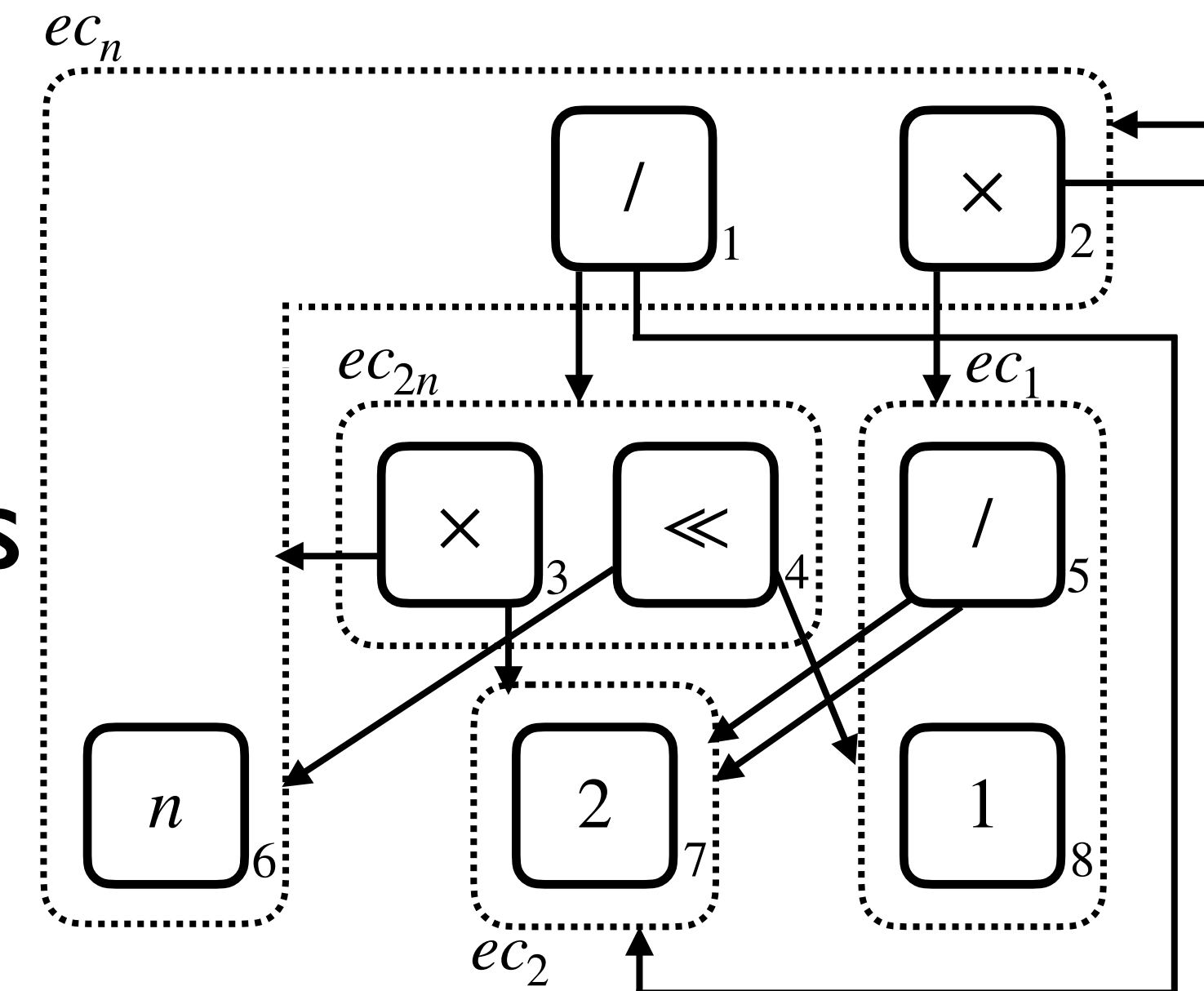


Equality Saturation

- A principled solution to the phase ordering problem
- Obtains results of all possible orderings and extract the best one among them
- Enabled by ***E-graph***, a very efficient data structure

E-Graph

- E-graph = e-nodes + e-classes
 - E-classes = set of e-nodes
 - E-node = a node whose children are e-classes
- Meaning
 - E-node (bold): expressions with sub-expressions represented by children e-classes
 - E-class (doted): semantically equivalent e-nodes



Example

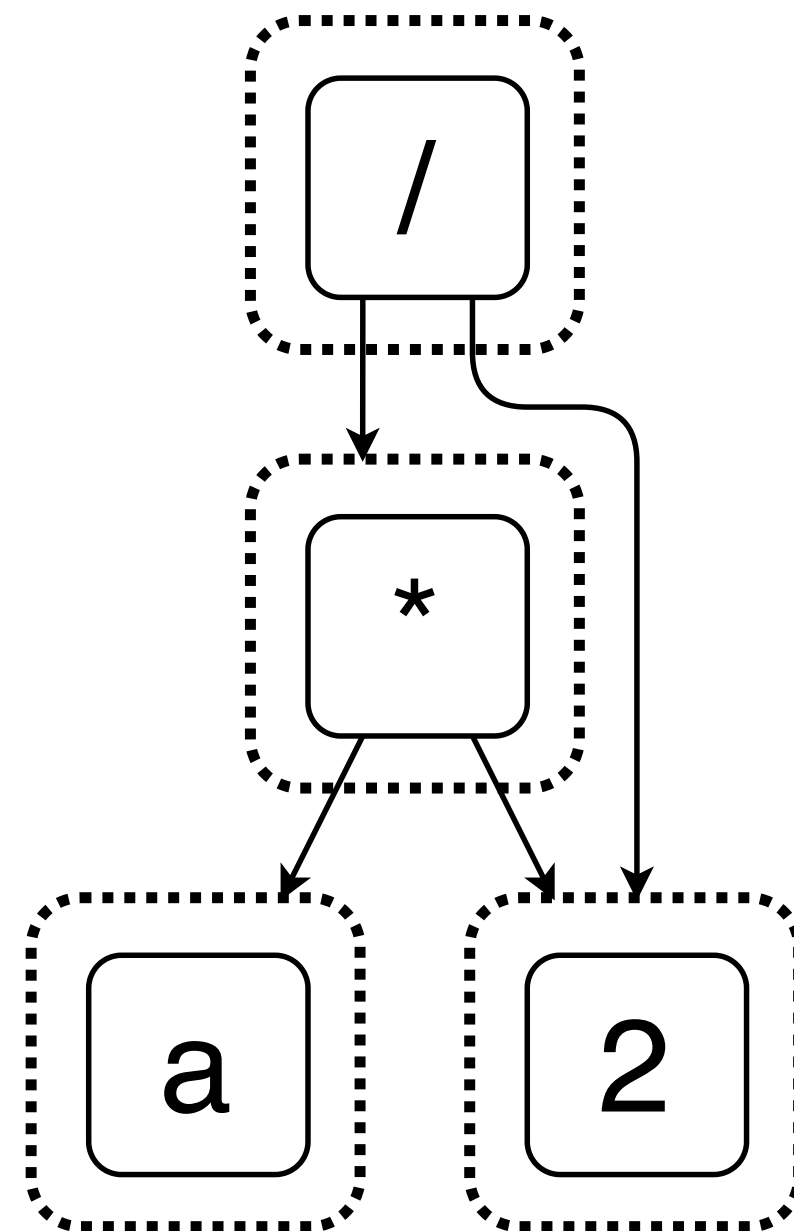
- Optimizing $(a \times 2)/2$ using the following rules:

(1) $x \times 2 \rightarrow x \ll 1$

(2) $(x \times y)/z \rightarrow x \times (y/z)$

(3) $x/x \rightarrow 1$

(4) $1 \times x \rightarrow x$



Example

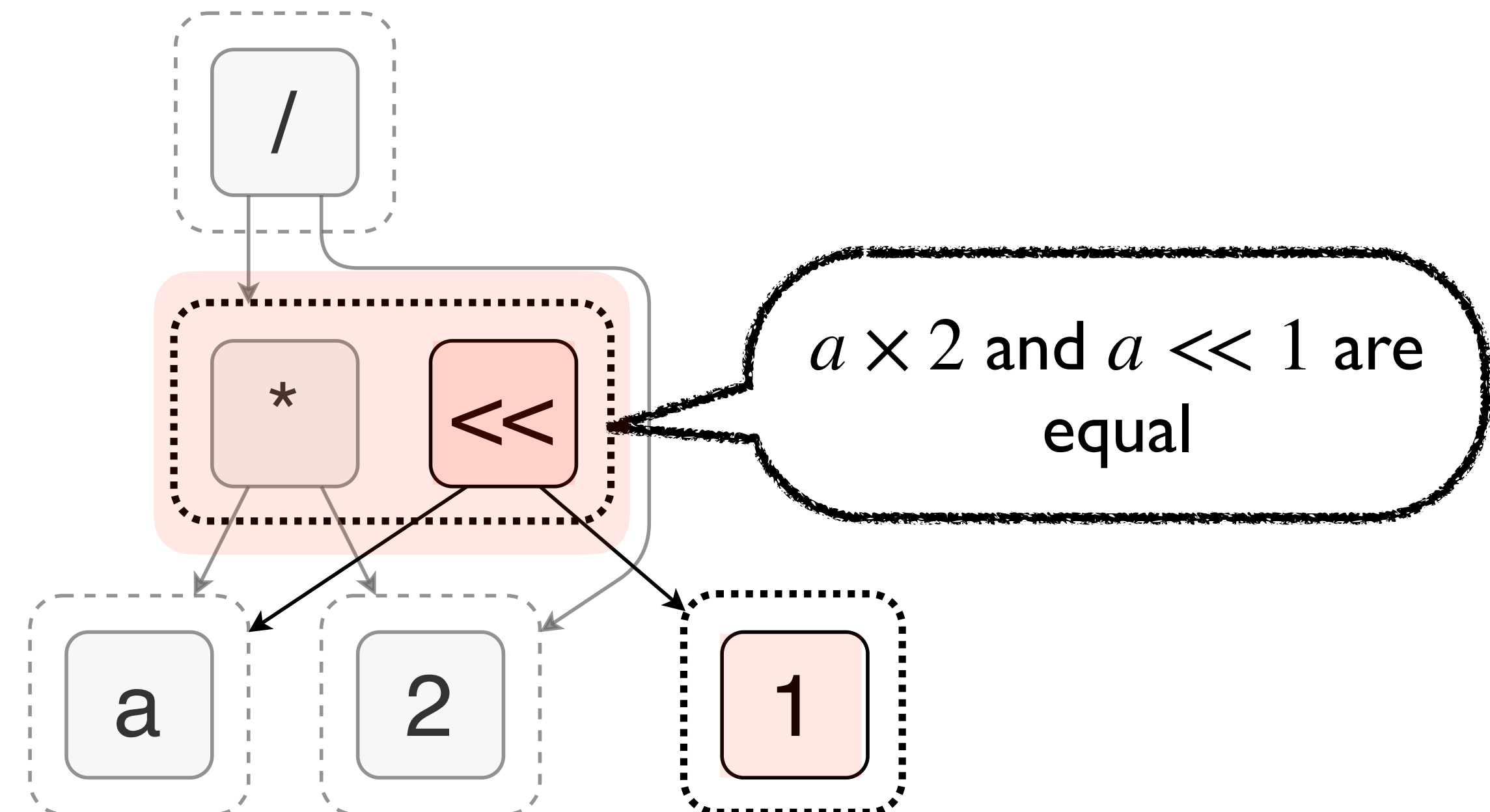
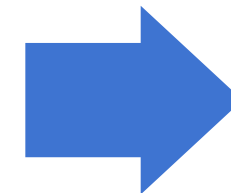
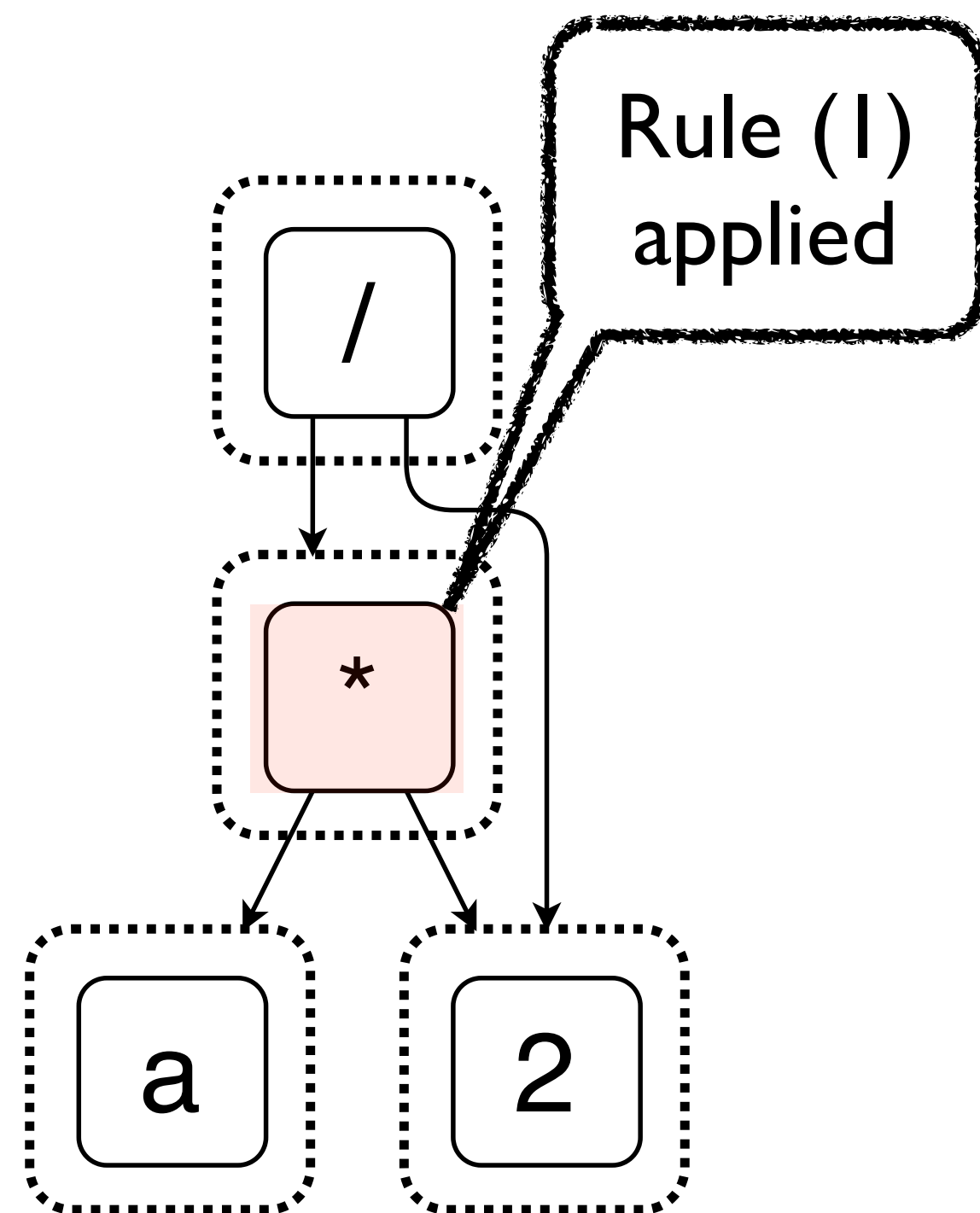
- Optimizing $(a \times 2)/2$ using the following rules:

(1) $x \times 2 \rightarrow x \ll 1$

(2) $(x \times y)/z \rightarrow x \times (y/z)$

(3) $x/x \rightarrow 1$

(4) $1 \times x \rightarrow x$



Example

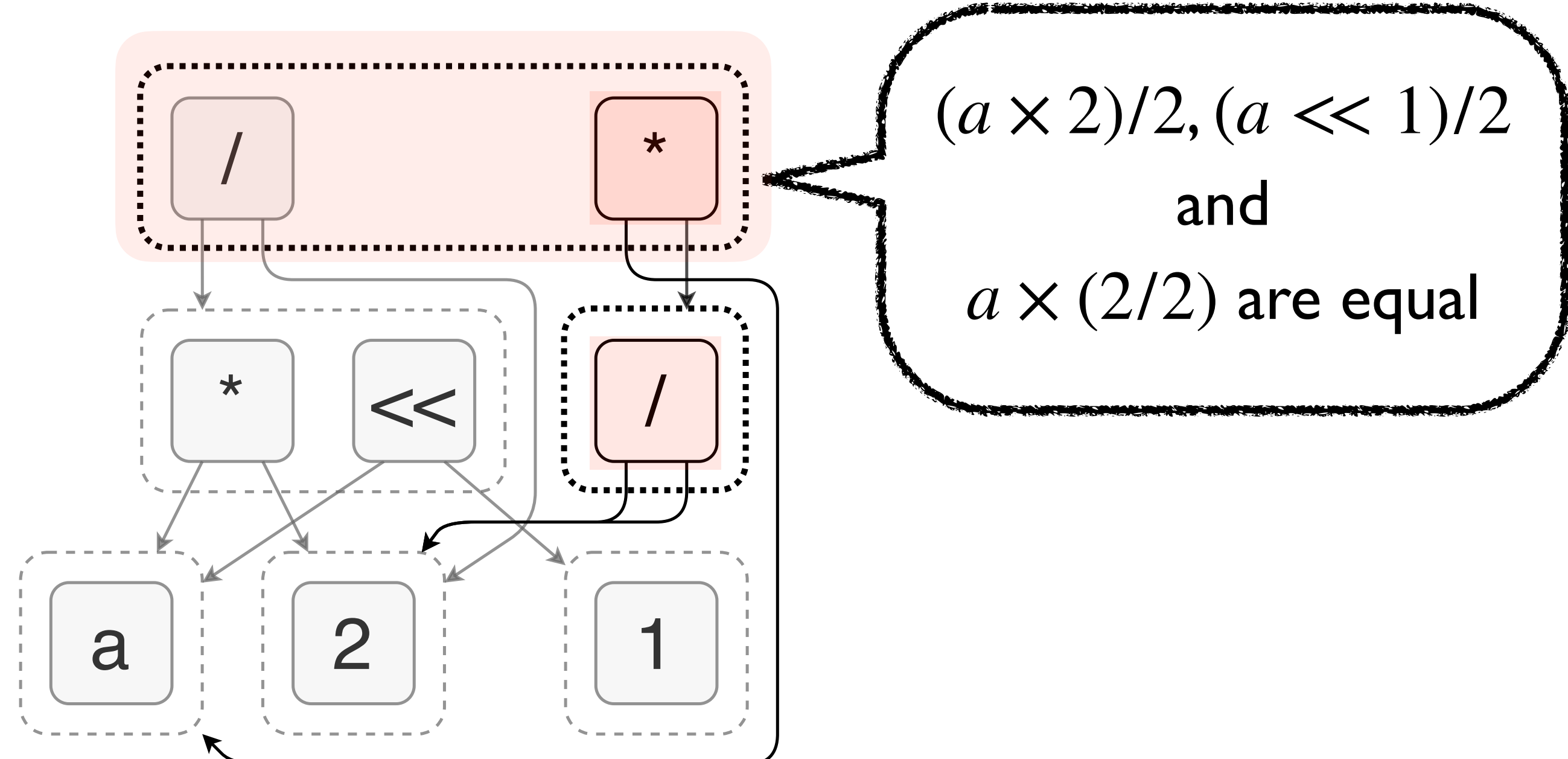
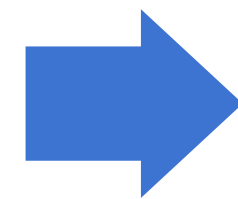
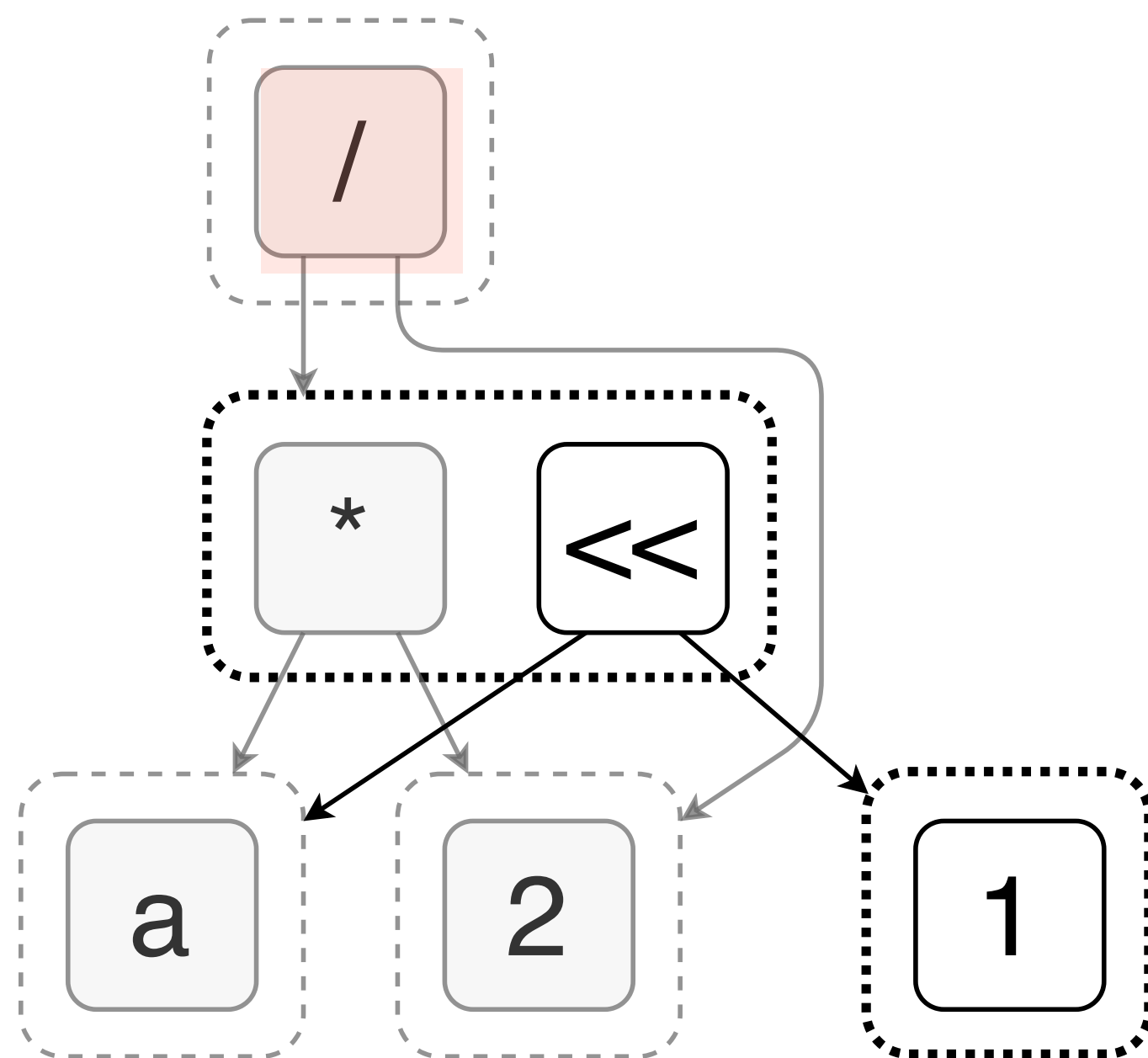
- Optimizing $(a \times 2)/2$ using the following rules:

(1) $x \times 2 \rightarrow x \ll 1$

(3) $x/x \rightarrow 1$

(2) $(x \times y)/z \rightarrow x \times (y/z)$

(4) $1 \times x \rightarrow x$



Example

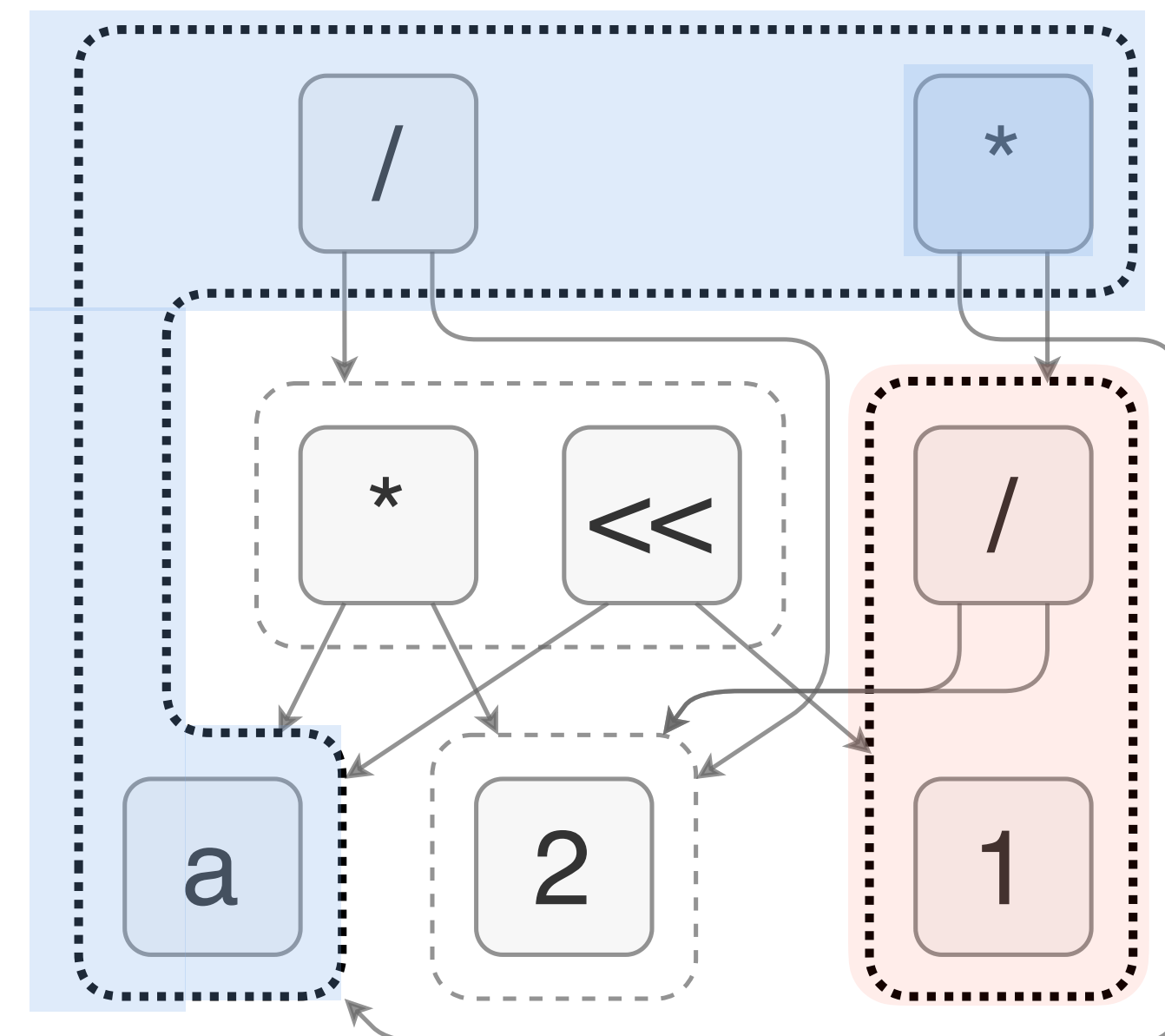
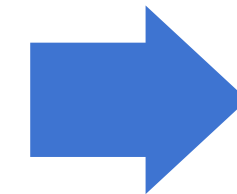
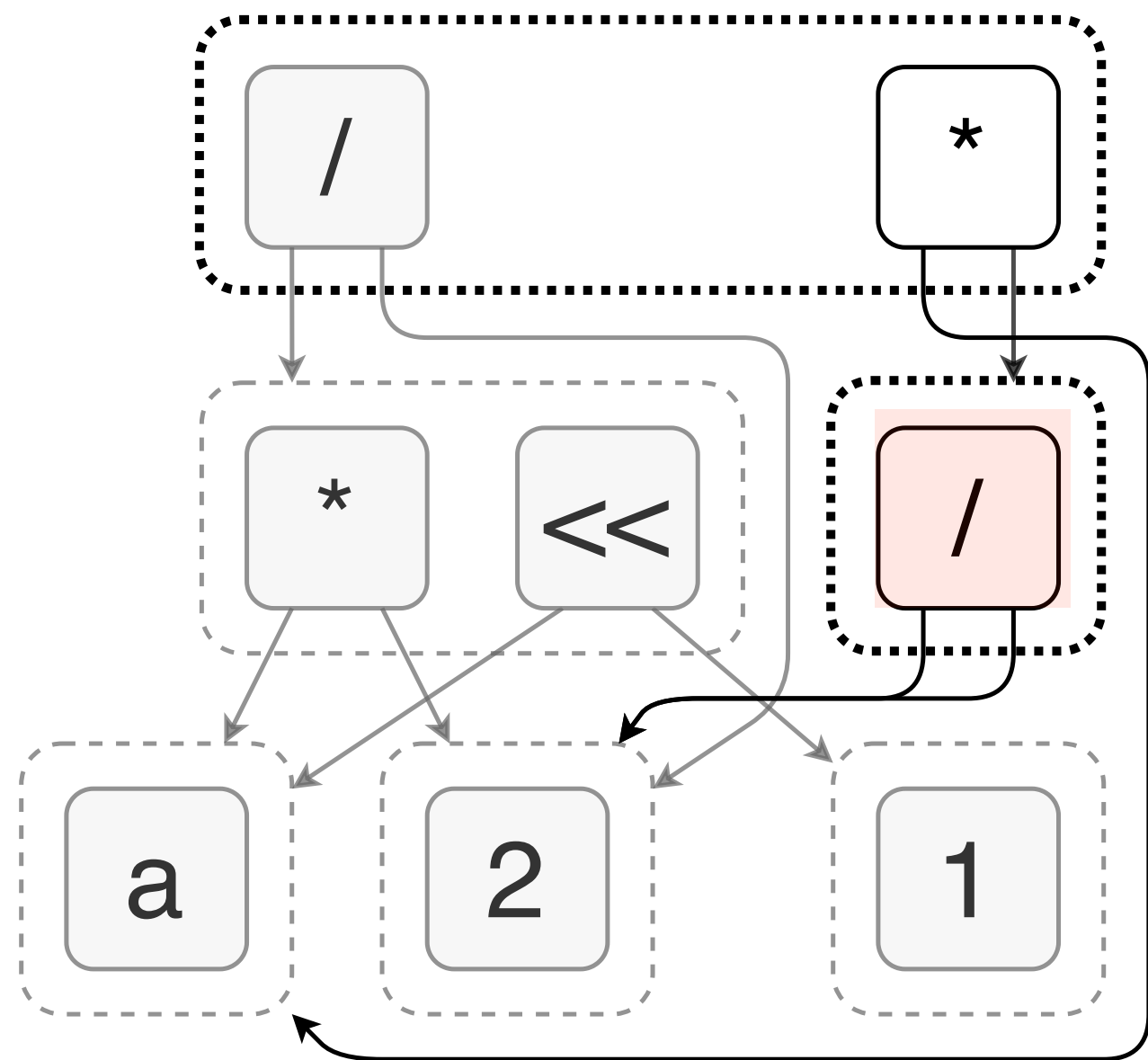
- Optimizing $(a \times 2)/2$ using the following rules:

$$(1) \quad x \times 2 \rightarrow x \ll 1$$

$$(3) \quad x/x \rightarrow 1$$

$$(2) \quad (x \times y)/z \rightarrow x \times (y/z)$$

$$(4) \quad 1 \times x \rightarrow x$$



Example

- Optimizing $(a \times 2)/2$ using the following rules:

$$(1) \quad x \times 2 \rightarrow x \ll 1$$

$$(2) \quad (x \times y)/z \rightarrow x \times (y/z)$$

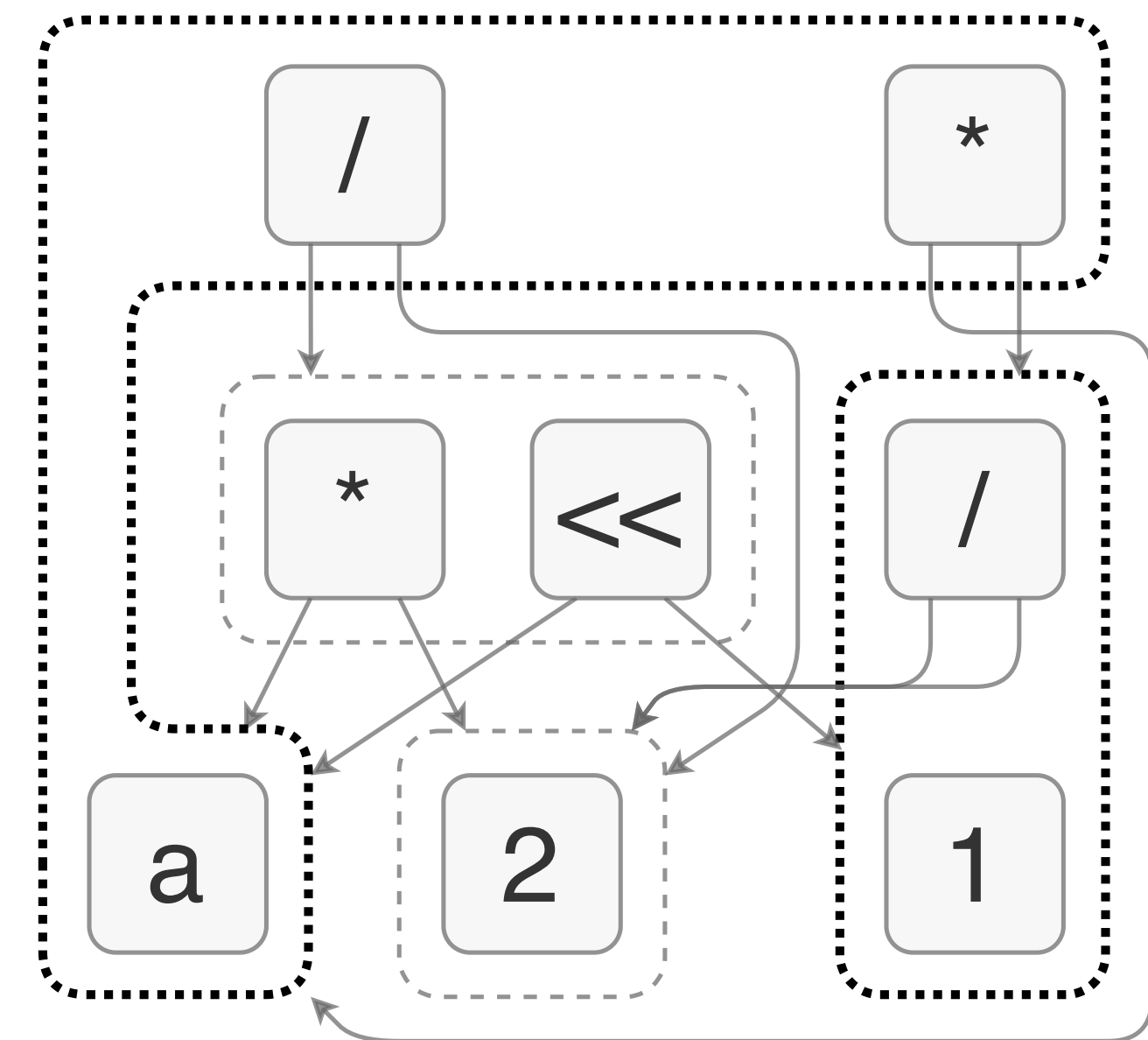
$$(3) \quad x/x \rightarrow 1$$

$$(4) \quad 1 \times x \rightarrow x$$

- More rule application can't change the graph

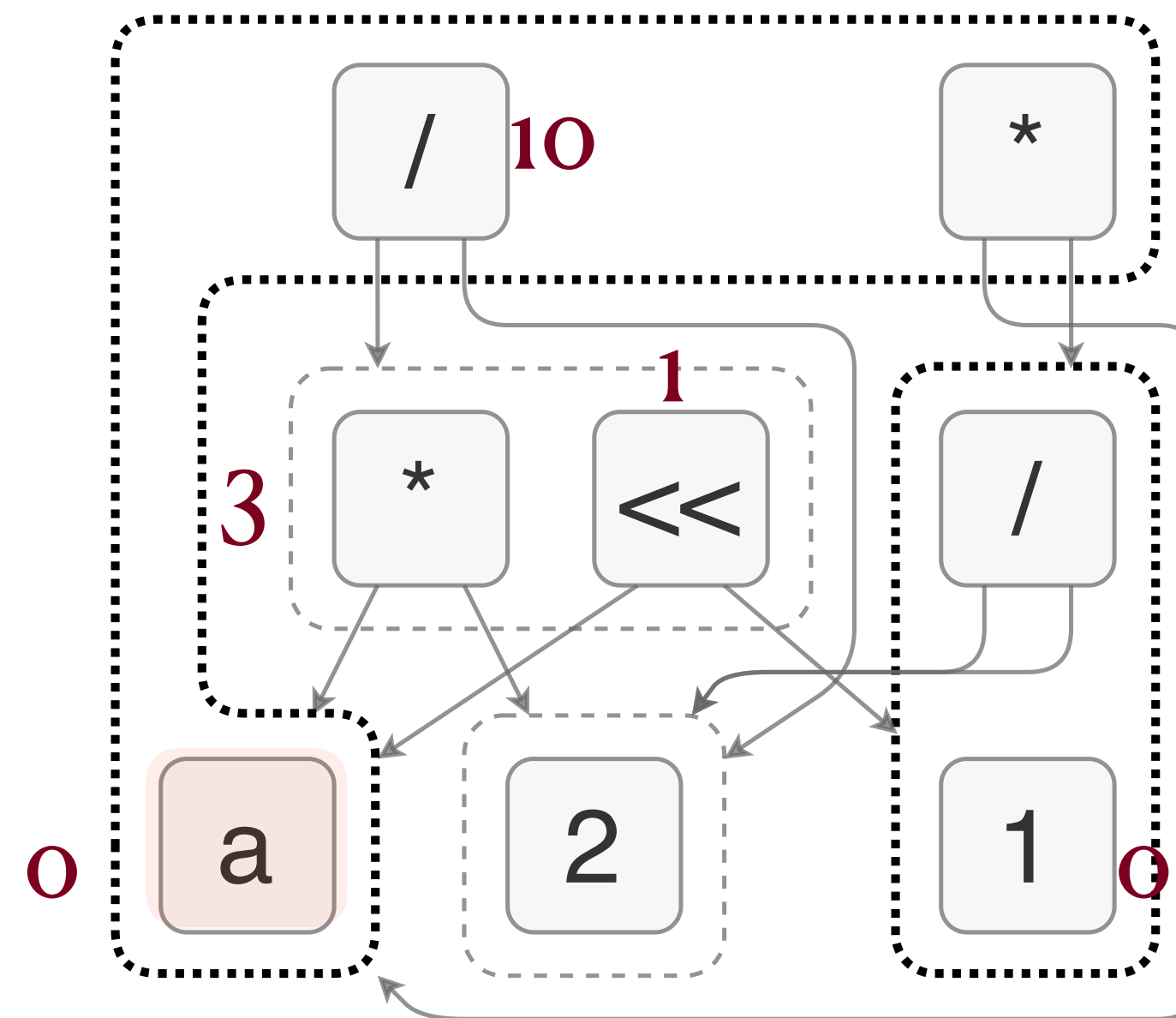
— *Saturation!*

- Exprs represented by the root node's e-class is *all* exprs obtainable by applying the rules in *all* possible orders



Extracting an Optimal Solution

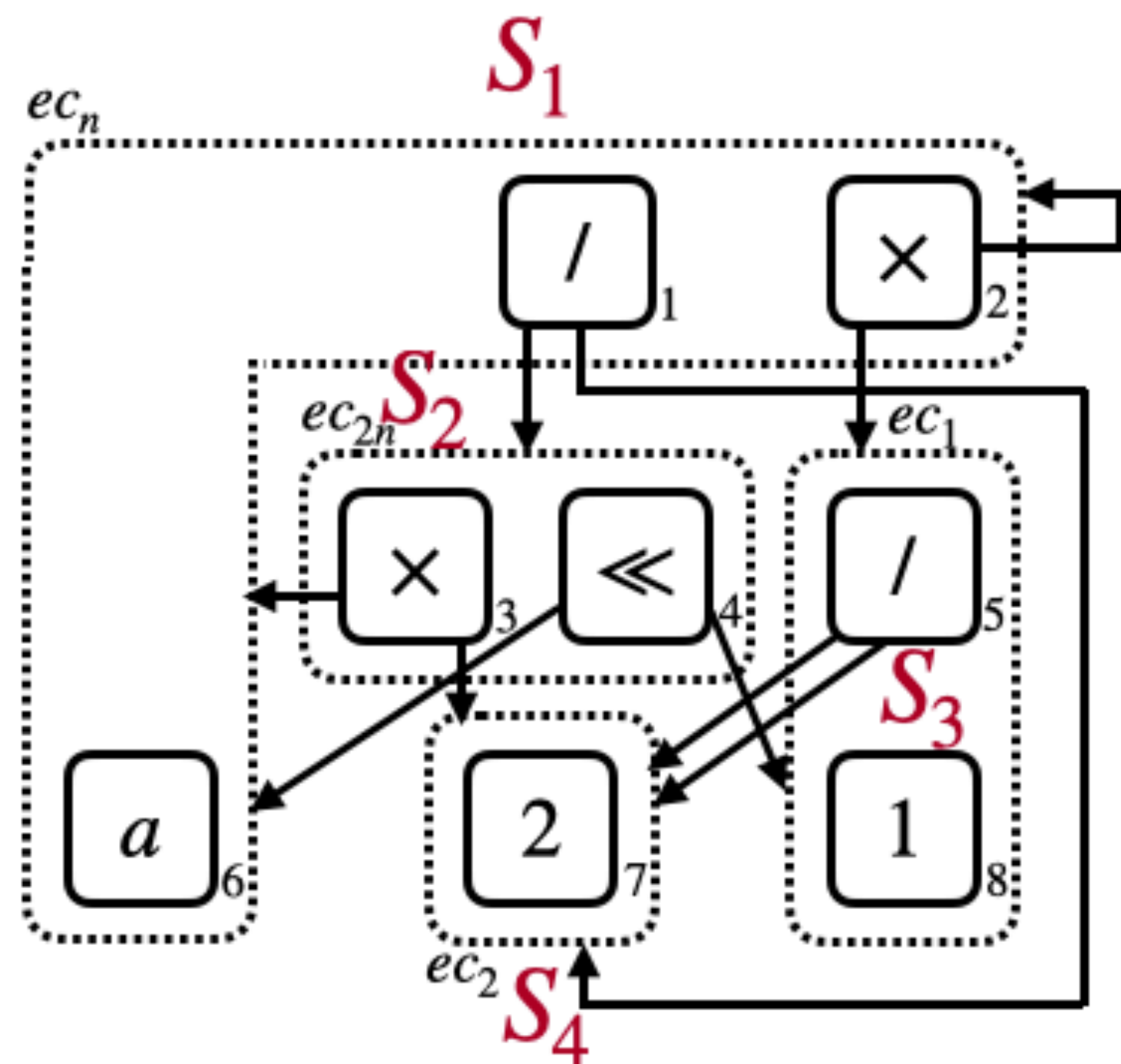
- Extract an expression of the best score after saturation†
 - e.g., greedy method using scores assigned for each kind of e-node



† Termination is not always guaranteed

Fundamental Meaning of Equality Saturation

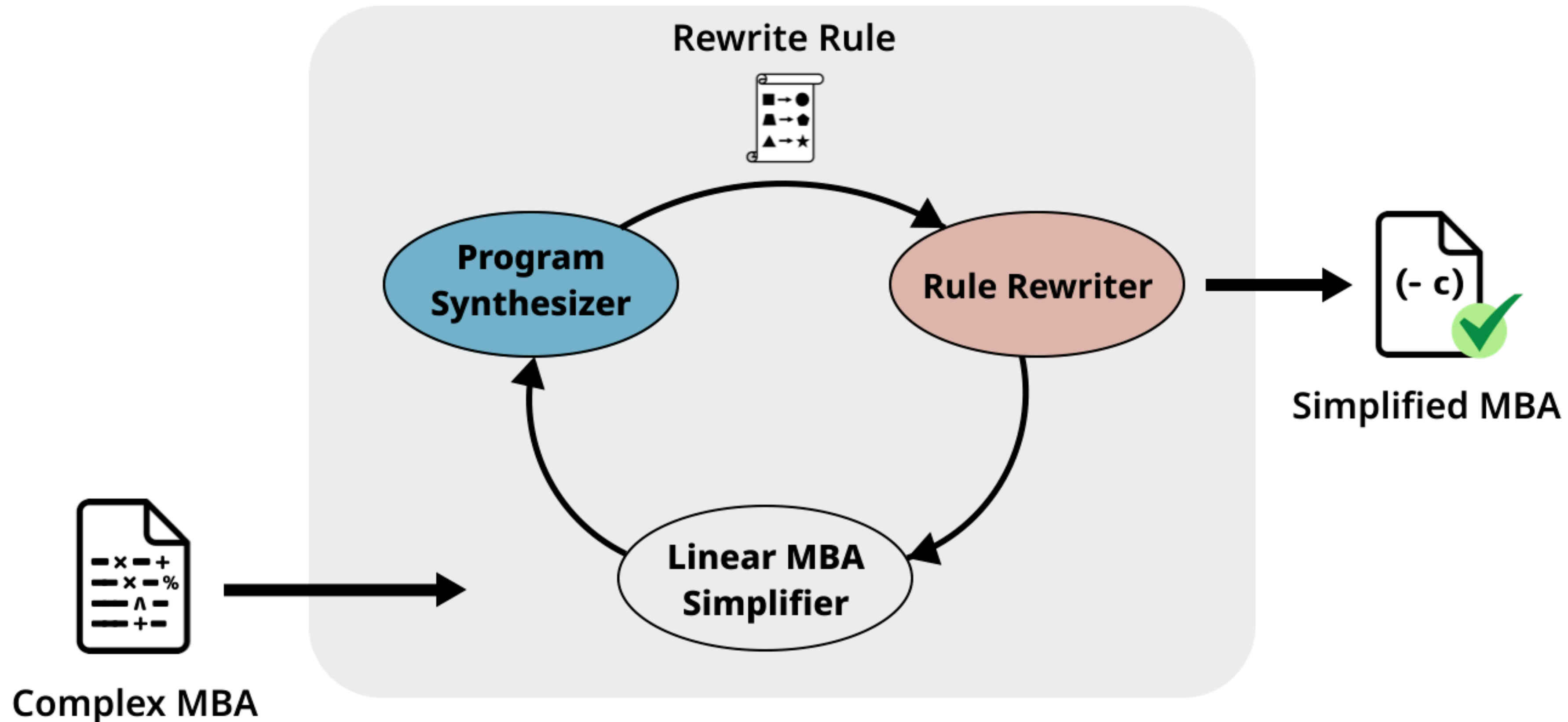
- E-graph $\cong$ Grammar representing semantically equivalent exprs
 - (E-class $\cong$ non-terminal, E-node $\cong$ production rule)
- Equality saturation = **grammar induction**



$$\begin{array}{lcl} S_1 & \rightarrow & S_2 / S_4 \\ & | & S_1 * S_3 \\ & | & a \\ S_2 & \rightarrow & S_1 \ll S_3 \\ & | & S_1 * S_4 \\ S_3 & \rightarrow & S_4 / S_4 \\ & | & 1 \\ S_4 & \rightarrow & 2 \end{array}$$

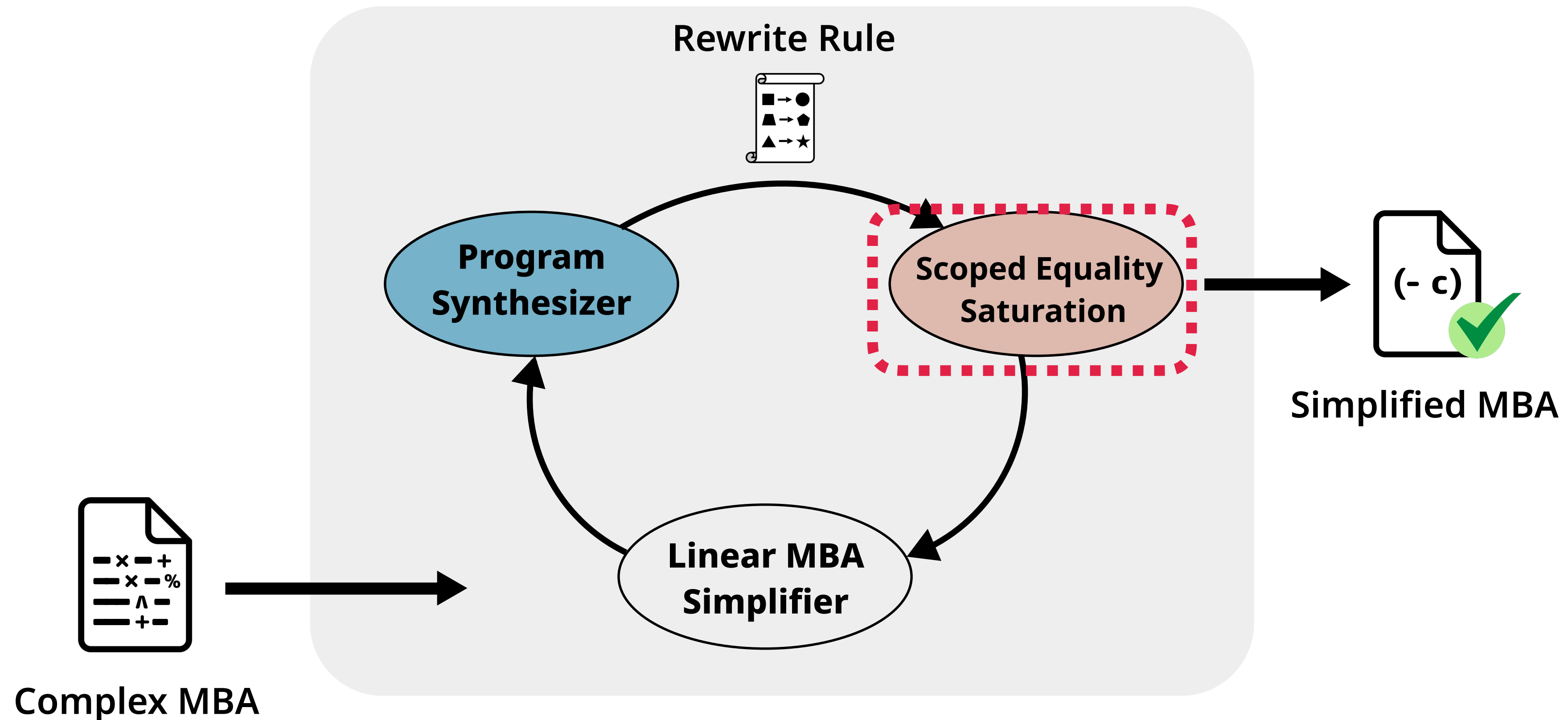
Major Change to the Overall Architecture

- Rule rewriter → Equality saturator



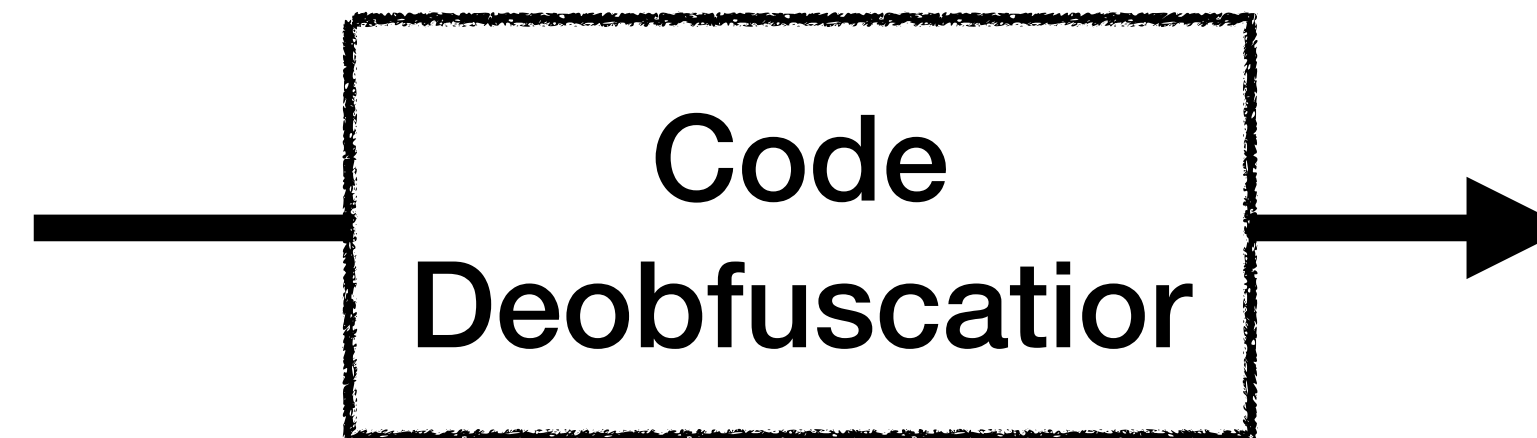
Major Change to the Overall Architecture

- Rule rewriter → Equality saturator



Illustrative Example

$(((((\neg(((d \wedge (\neg c)) -$
 $((\neg d) \wedge c)) \oplus (-d - 1)) +$
 $((((d \wedge (\neg c)) -$
 $((\neg d) \wedge c)) \wedge (-d - 1)) < < 1))))$
 $+ 1) \wedge (\neg(((\neg(e - 1)) \wedge (\neg((\neg((e \wedge$
 $(\neg d)) + (e \wedge (\neg d))) - (e \oplus d)))$
 $+ 1)))) + ((\neg(((e \wedge (\neg d)) +$
 $(e \wedge (\neg d)) - (e \oplus d))) + 1))))$
 $+ (((\neg(e - 1)) \wedge (\neg((\neg((e$
 $\wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d)))$
 $+ 1))) + ((\neg(((e \wedge (\neg d)) + (e$
 $\wedge (\neg d)) - (e \oplus d))) + 1)))$

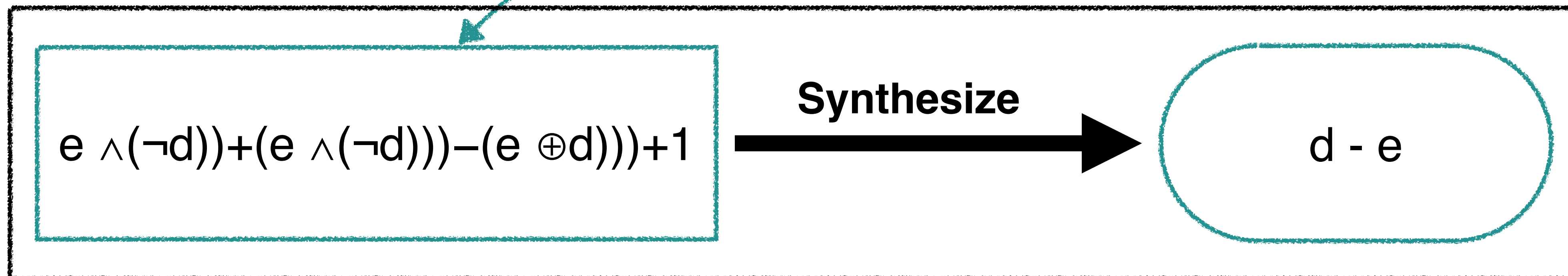
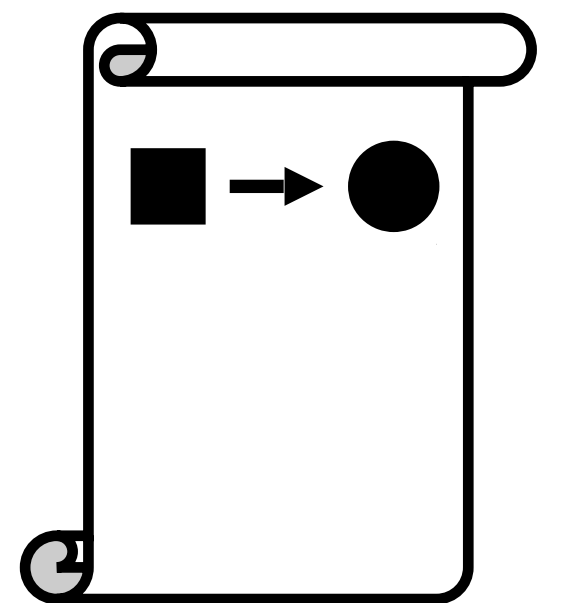


$(d - e) \vee ((-e) \vee (c + 1))$

Synthesize

$$\begin{aligned}
 & (((\neg(((d \wedge (\neg c)) - ((\neg d) \wedge c)) \oplus (-d - 1)) + (((d \wedge (\neg c)) - ((\neg d) \wedge c)) \wedge (-d - 1)) < 1))) \\
 & + 1) \wedge (\neg(((\neg(e - 1)) \wedge (\neg((\neg((e \wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d))) + 1))) + ((\neg(((e \wedge (\neg d)) + \\
 & (e \wedge (\neg d))) - (e \oplus d))) + 1)))) \\
 & + (((\neg(e - 1)) \wedge (\neg((\neg((e \wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d))) + 1)))) + ((\neg(((e \wedge (\neg d)) + (e \\
 & \wedge (\neg d))) - (e \oplus d))) + 1)))
 \end{aligned}$$

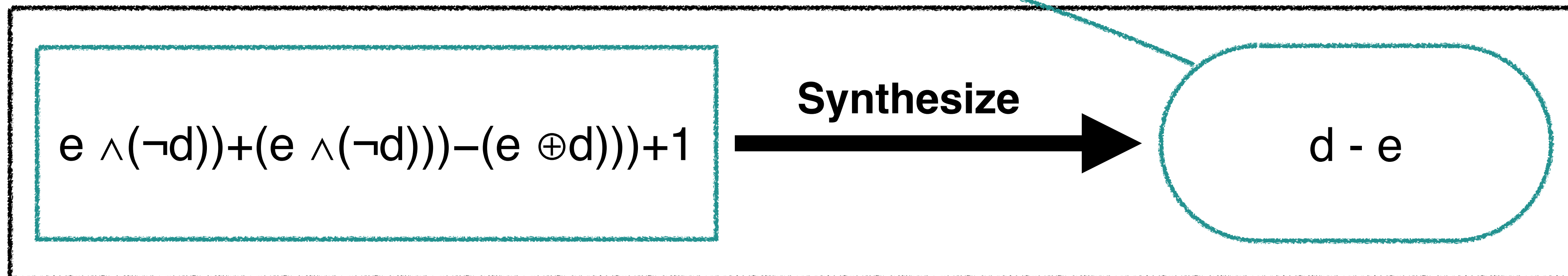
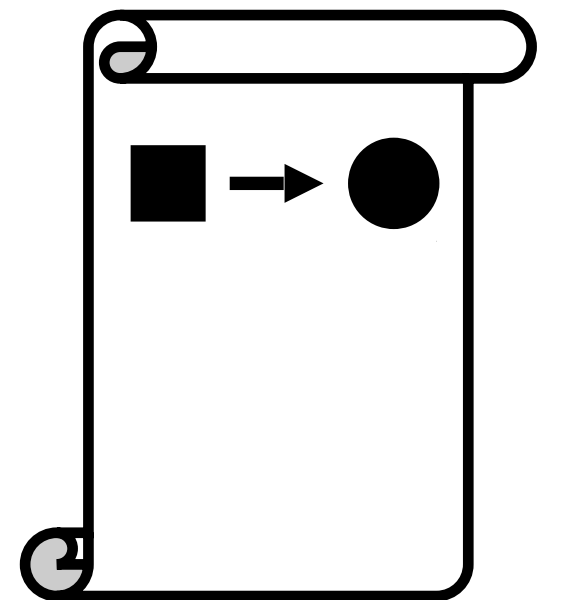
Rewrite Rule



Synthesize

$$\begin{aligned}
 & (((\neg(((d \wedge (\neg c)) - ((\neg d) \wedge c)) \oplus (-d - 1)) + (((d \wedge (\neg c)) - ((\neg d) \wedge c)) \wedge (-d - 1)) << 1))) \\
 & + 1) \wedge (\neg(((\neg(e - 1)) \wedge (\neg((\neg((e \wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d))) + 1))) + ((\neg((e \wedge (\neg d)) + \\
 & (e \wedge (\neg d))) - (e \oplus d))) + 1)))) \\
 & + (((\neg(e - 1)) \wedge (\neg((\neg((d - e)))) + ((\neg((e \wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d))) + 1)))
 \end{aligned}$$

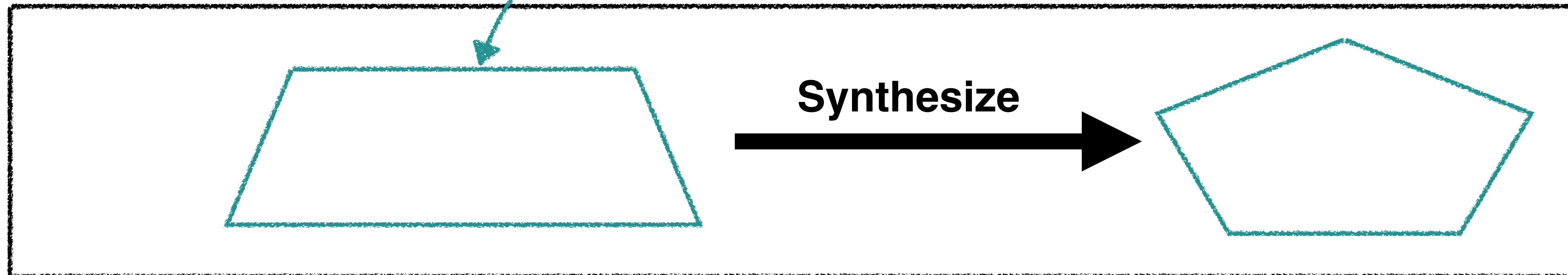
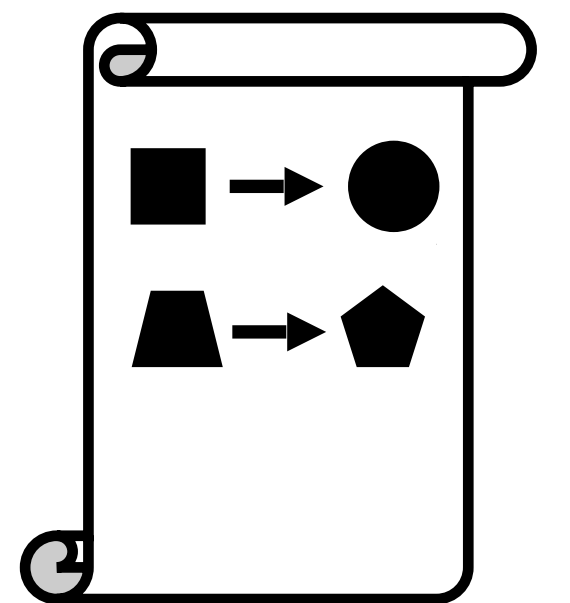
Rewrite Rule



Synthesize

$$\begin{aligned}
 & (((\neg(((d \wedge (\neg c)) - ((\neg d) \wedge c)) \oplus (-d - 1)) + (((d \wedge (\neg c)) - ((\neg d) \wedge c)) \wedge (-d - 1)) << 1))) \\
 & + 1) \wedge (\neg(((\neg(e - 1)) \wedge (\neg((\neg((e \wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d))) + 1))) + ((\neg(((e \wedge (\neg d)) + \\
 & (e \wedge (\neg d))) - (e \oplus d))) + 1)))) \\
 & + (((\neg(e - 1)) \wedge (\neg((\neg((\mathbf{d} - \mathbf{e}))) + ((\neg(((e \wedge (\neg d)) + (e \wedge (\neg d))) - (e \oplus d))) + 1))))
 \end{aligned}$$

Rewrite Rule



...

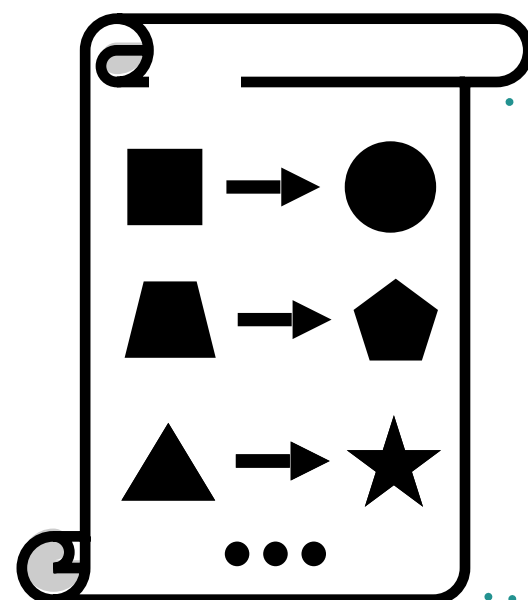
After Rule Synthesis

- After repeated synthesis, we have

$$(-(((d - c - 2) \oplus (-d - 1)) + (((d - c - 2) \wedge (-d - 1)) \ll 1))) \vee (((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d))))$$

- with the following learned rules

Rewrite Rule

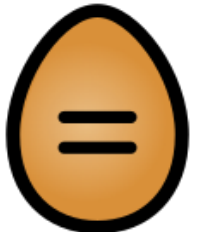


- $a \ll 1 \rightarrow 2 \times a$
- $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
- $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
- $(a + 1) \rightarrow (\neg(\neg a))$
- $(\neg(\neg a)) \rightarrow a$
- ...

Equality Saturation

- We may apply equality saturation to the current expression with the learned rules

$$\begin{aligned} &(-(((d - c - 2) \oplus (-d - 1)) + (((d - c - 2) \wedge (-d - 1)) \ll 1))) \vee \\ &(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d))) \end{aligned}$$

- Because of its large size, even the state-of-the-art equality saturation library (egg ) times out.

Scoped Equality Saturation

- “**Scoped**” in both target region and rules
- **Step 1** : split the large expr into smaller regions
 - **Region (1)** and **Region (2)**

$$\begin{aligned} & (-(((d - c - 2) \oplus (-d - 1)) + (((d - c - 2) \wedge (-d - 1)) \ll 1))) \quad \vee \\ & (((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) + (1 + (\neg(e - d)))) \end{aligned}$$

Each part does not exceed a certain size limit (e.g., height 6 in AST)

Scoped Equality Saturation

Step 2 : for each target region, get “relevant” rules that may be useful

$$\begin{aligned} & (-(((d - c - 2) \oplus (-d - 1)) + (((d - c - 2) \wedge (-d - 1)) \ll 1))) \quad \vee \\ & (((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) + (1 + (\neg(e - d)))) \end{aligned}$$

Scoped Equality Saturation

Step 2 : for each target region, get “relevant” rules that may be useful

$$\begin{aligned} &(-(((d - c - 2) \oplus (-d - 1)) + \\ &(((d - c - 2) \wedge (-d - 1)) \ll 1))) \end{aligned}$$

- ✓ $a \ll 1 \rightarrow 2 \times a$
- ✓ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
- ✗ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
- ✗ $(1 + a) \rightarrow (-(\neg a))$
- ✗ $(\neg(\neg a)) \rightarrow a$

$$\begin{aligned} &(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) \\ &+(1 + (\neg(e - d)))) \end{aligned}$$

- ✗ $a \ll 1 \rightarrow 2 \times a$
- ✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
- ✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
- ✓ $(1 + a) \rightarrow (-(\neg a))$
- ✓ $(\neg(\neg a)) \rightarrow a$

Scoped Equality Saturation

Step 3 : pick the region of more“relevant” rules and do Eqsat with the rules

$$\begin{aligned} &(-(((d - c - 2) \oplus (-d - 1)) + \\ &(((d - c - 2) \wedge (-d - 1)) \ll 1))) \end{aligned}$$

$$\begin{aligned} &(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) \\ &+(1 + (\neg(e - d)))) \end{aligned}$$

2 relevant rules < 3 relevant rules

Pick the **region (2)**

✓ $a \ll 1 \rightarrow 2 \times$

✓ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

✗ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

✗ $(1 + a) \rightarrow (\neg(\neg a))$

✗ $(\neg(\neg a)) \rightarrow a$

$\rightarrow 2 \times a$

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

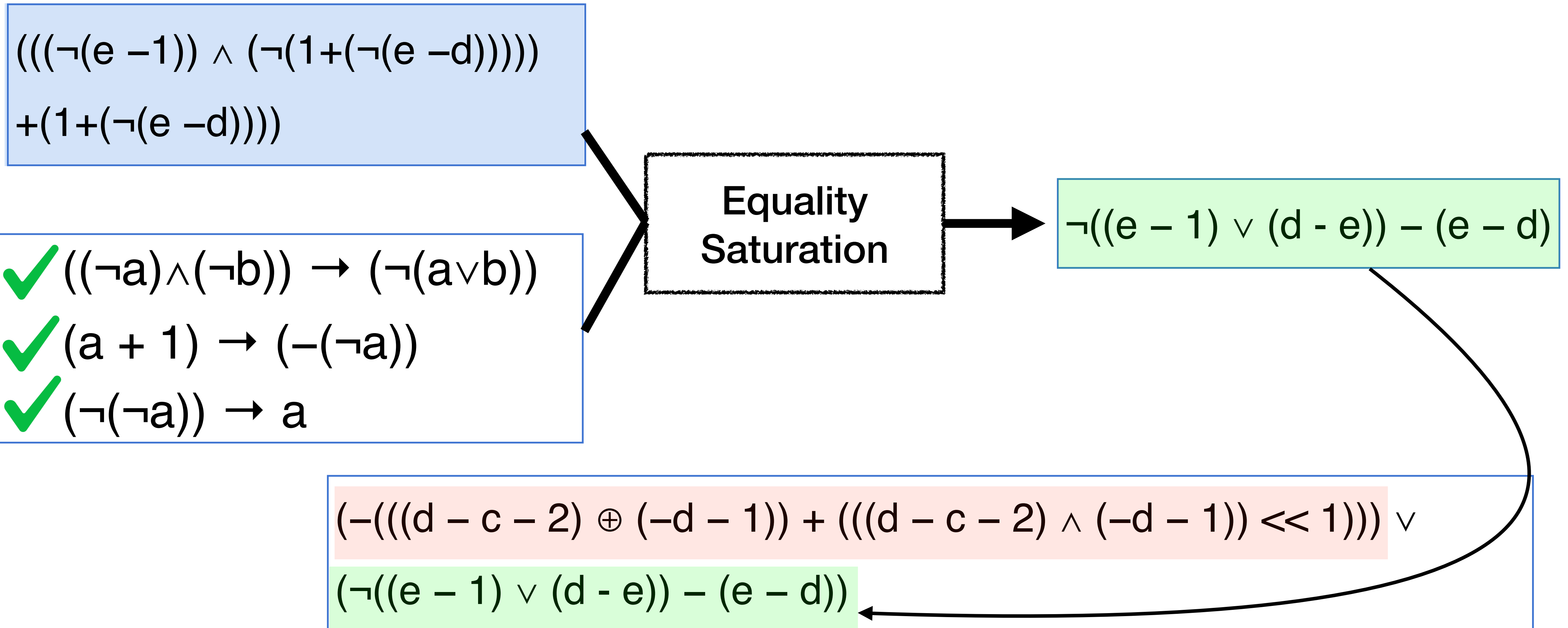
✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

✓ $(1 + a) \rightarrow (\neg(\neg a))$

✓ $(\neg(\neg a)) \rightarrow a$

Scoped Equality Saturation

Step 3 : pick the region of more“relevant” rules and do Eqsat with the rules



How to Find “Relevant” Rules?

- A simple way to determine the relevance of a rule is to check if the left-hand side of the rule matches some sub-expressions.

$((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) + (1 + (\neg(e - d)))$

No subexprs of
form

$a \ll 1$ and
 $a \oplus b + 2 \times a \wedge b$

✗ $a \ll 1 \rightarrow 2 \times a$

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

$((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

$(1 + a) \rightarrow (\neg(\neg a))$

$(\neg(\neg a)) \rightarrow a$

How to Find “Relevant” Rules?

- A simple way to determine the relevance of a rule is to check if the left-hand side of the rule matches some sub-expressions.

$((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) + (1 + (\neg(e - d)))$

✗ $a \ll 1 \rightarrow 2 \times a$

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

$(1 + a) \rightarrow (\neg(\neg a))$

$(\neg(\neg a)) \rightarrow a$

How to Find “Relevant” Rules?

- A simple way to determine the relevance of a rule is to check if the left-hand side of the rule matches some sub-expressions.

$((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))) + (1 + (\neg(e - d))))$

✗ $a \ll 1 \rightarrow 2 \times a$

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

✓ $(1 + a) \rightarrow (\neg(\neg a))$

$(\neg(\neg a)) \rightarrow a$

How to Find “Relevant” Rules?

- A simple way to determine the relevance of a rule is to check if the left-hand side of the rule matches some sub-expressions.

$((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))) + (1 + (\neg(e - d))))$

No subexprs of
form
 $\neg(\neg a)$

- ✗ $a \ll 1 \rightarrow 2 \times a$
- ✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
- ✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
- ✓ $(1 + a) \rightarrow (\neg(\neg a))$
- ✗ $(\neg(\neg a)) \rightarrow a$

May Miss Useful Rules

- Unfortunately, this simple method may miss useful rules.

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d))))$$

No subexprs of
form
 $\neg(\neg a)$

- ✗ $a \ll 1 \rightarrow 2 \times a$
- ✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
- ✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
- ✓ $(1 + a) \rightarrow (-(\neg a))$
- ✗ $(\neg(\neg a)) \rightarrow a$

This rule should
NOT be excluded
— essential for
simplification!

May Miss Useful Rules

- Why do we need $(\neg(\neg a)) \rightarrow a$?

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d))))$$



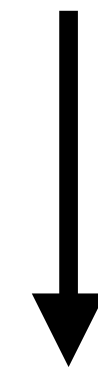
$$(1 + a) \rightarrow (\neg(\neg a))$$

$$(((\neg(e - 1)) \wedge (\neg(\neg \neg(\neg(e - d)))))) + (\neg \neg(\neg(e - d))))$$

May Miss Useful Rules

- Why do we need $(\neg(\neg a)) \rightarrow a$?

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d))))$$



$$(1 + a) \rightarrow (\neg(\neg a))$$

$$(((\neg(e - 1)) \wedge (\neg(\neg \neg(\neg(e - d)))))) + (\neg \neg(\neg(e - d))))$$



$$(\neg(\neg a)) \rightarrow a$$

We need this

$$(((\neg(e - 1)) \wedge (\neg(\neg(e - d)))) + (\neg(e - d)))$$



...

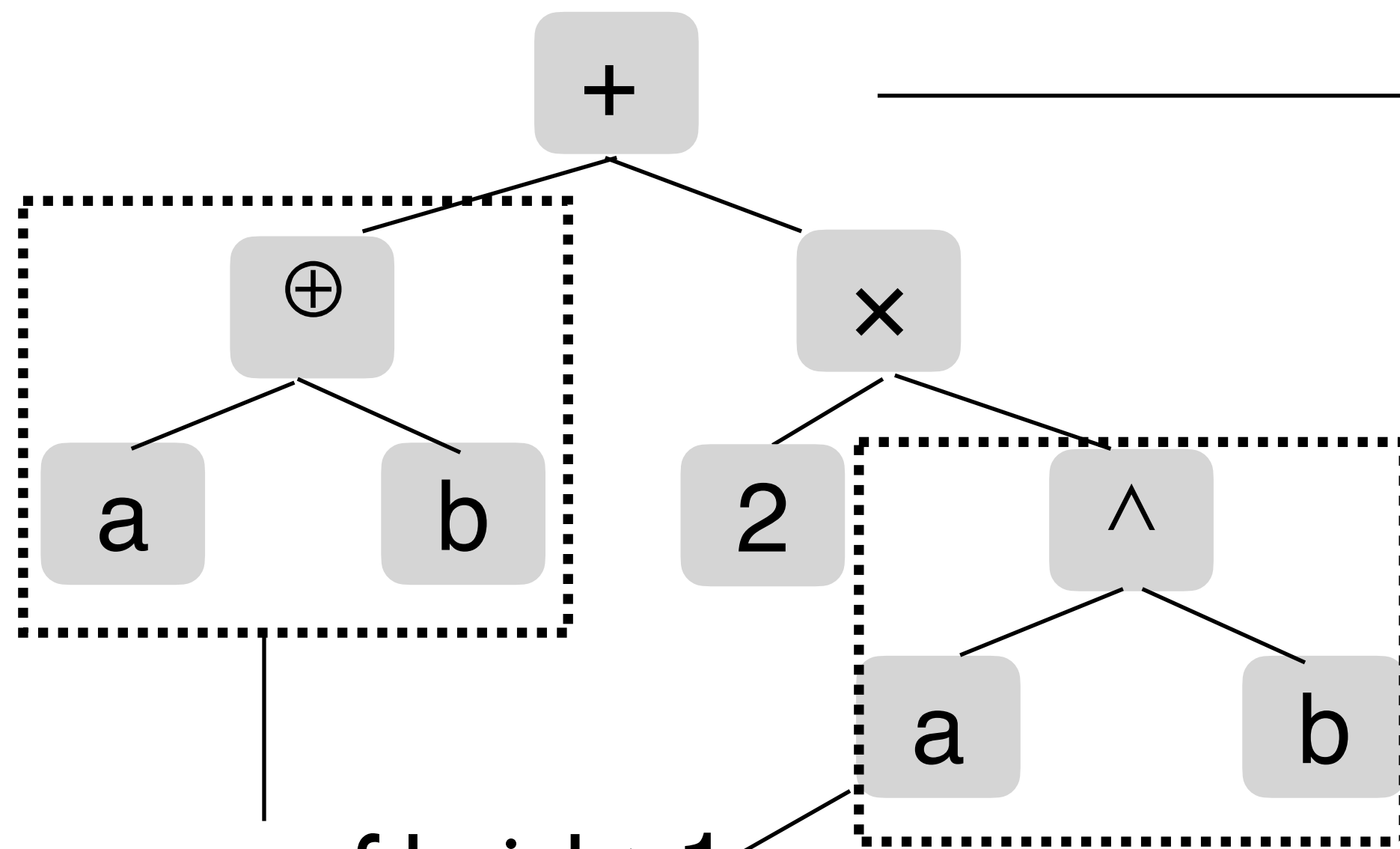
Heuristic for Finding Relevant Rules

- The simple syntactic matching doesn't work
 - Identifying rules useful for equality saturation is hard because of **the complex interactions among the rules.**
- We use the following heuristic.
 - A rule $L \rightarrow R$ is relevant for equality saturation over e if all sub-expressions of a certain size (e.g., height 1 in AST) in L match some sub-expressions of e

Heuristic for Finding Relevant Rules

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d)))$$

AST of the LHS



Sub-exprs of height 1



$$a \oplus b + 2 \times a \wedge b \rightarrow a + b$$

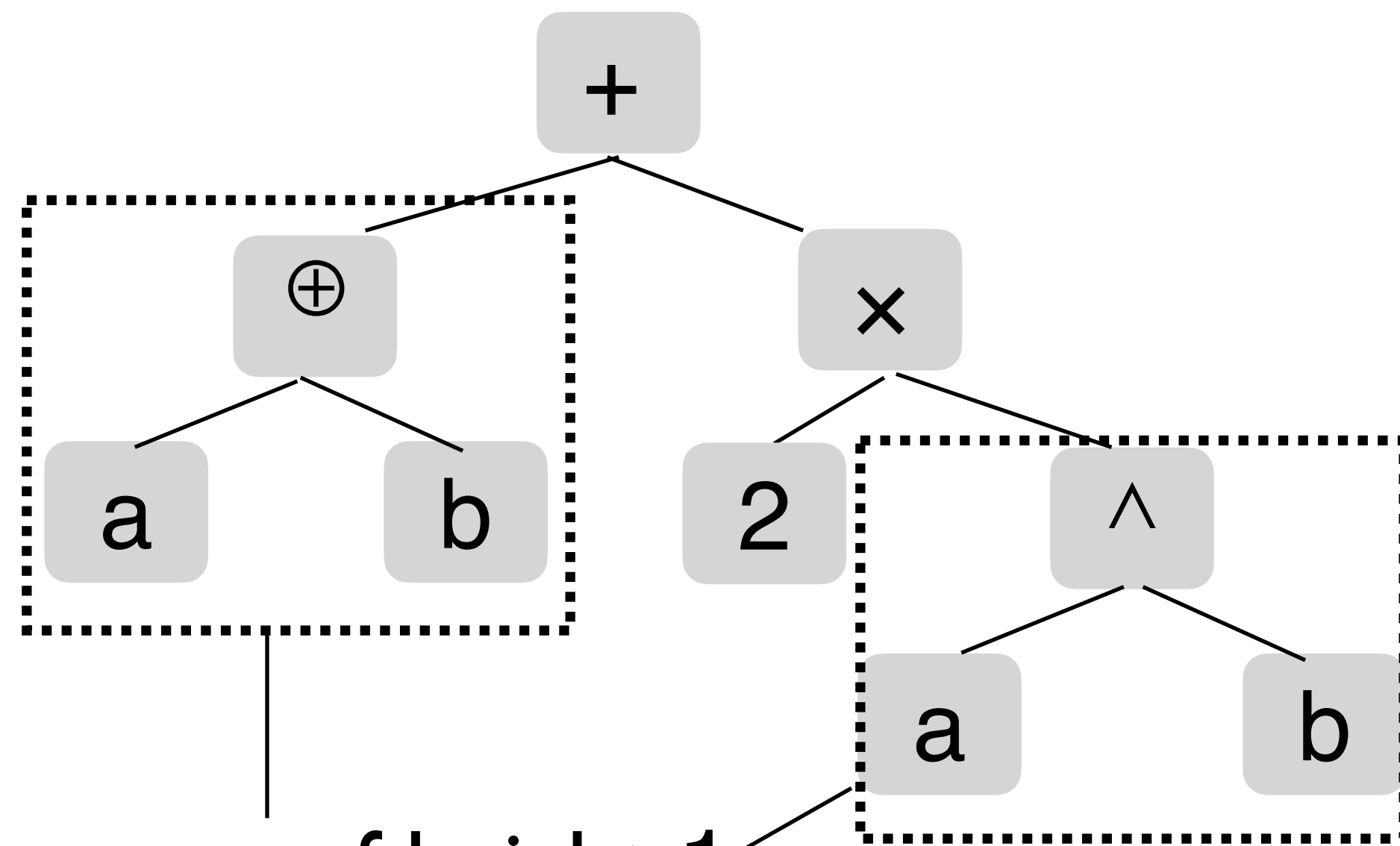
$$((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$$

$$(\neg(\neg a)) \rightarrow a$$

Heuristic for Finding Relevant Rules

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) + (1 + (\neg(e - d))))$$

AST of the LHS



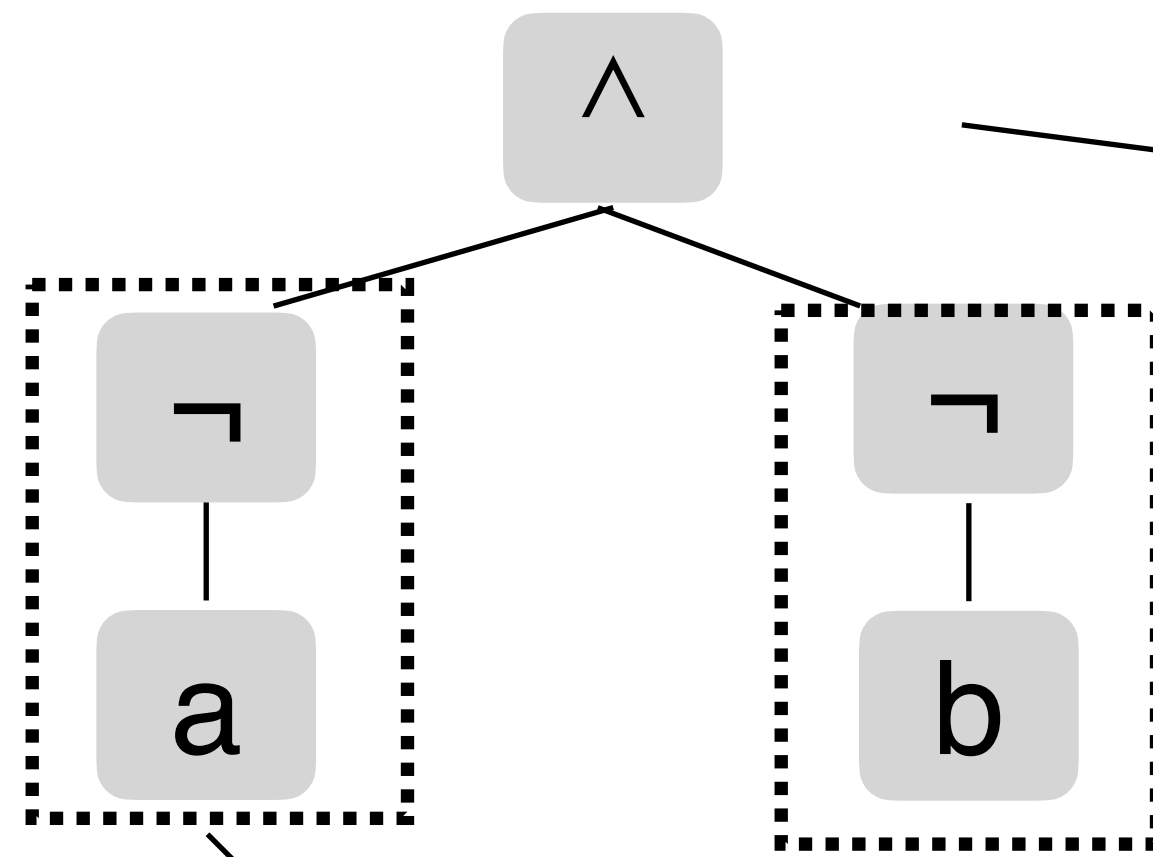
Sub-exprs of height 1

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
 $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
 $(\neg(\neg a)) \rightarrow a$

Heuristic for Finding Relevant Rules

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d)))$$

AST of the LHS



Sub-exprs of height 1

$\times a \oplus b + 2 \times a \wedge b \rightarrow a + b$

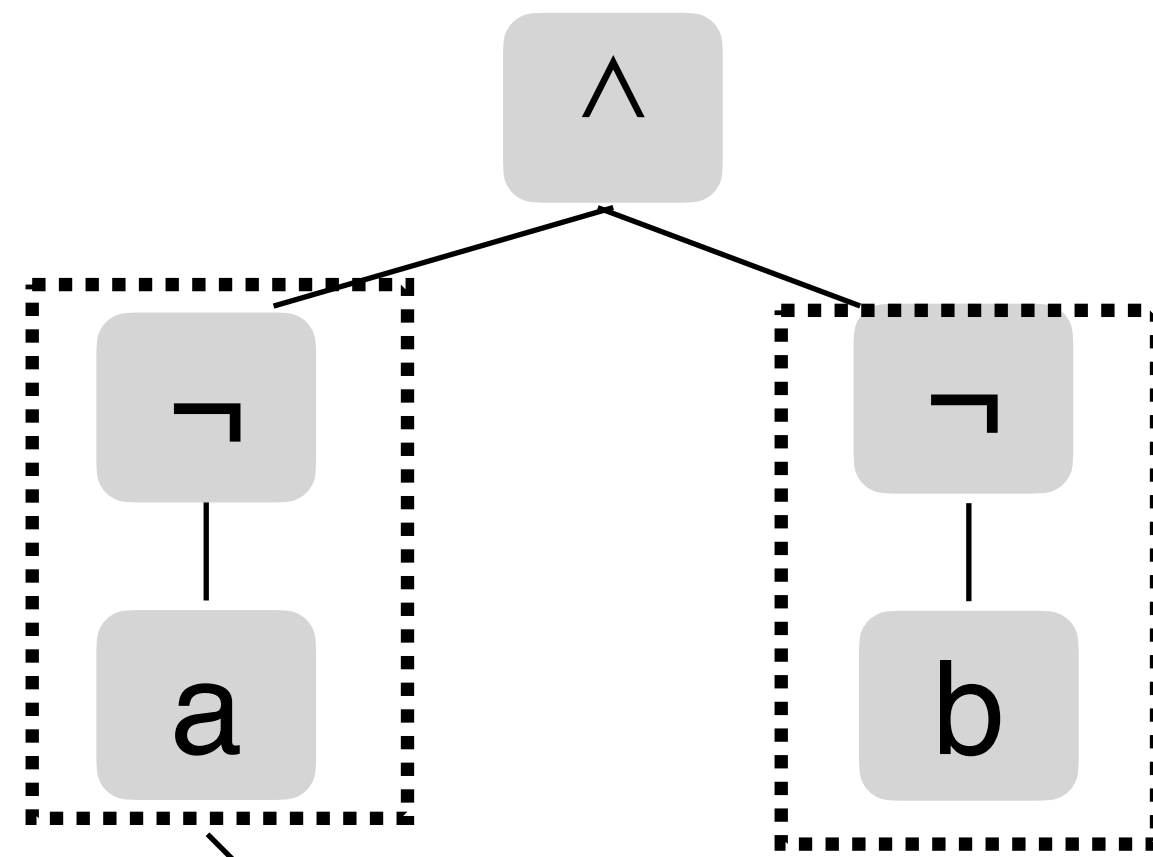
$((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

$(\neg(\neg a)) \rightarrow a$

Heuristic for Finding Relevant Rules

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d)))$$

AST of the LHS



Sub-exprs of height 1

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

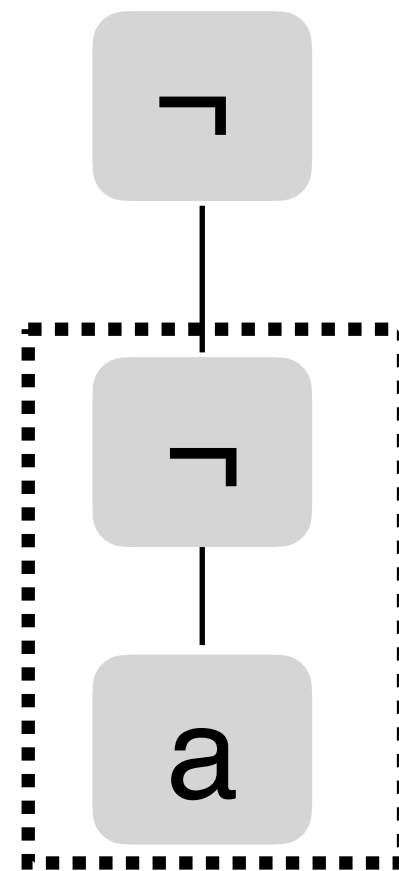
✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

$(\neg(\neg a)) \rightarrow a$

Heuristic for Finding Relevant Rules

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d)))))) + (1 + (\neg(e - d)))$$

AST of the LHS



Sub-exprs of height 1

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

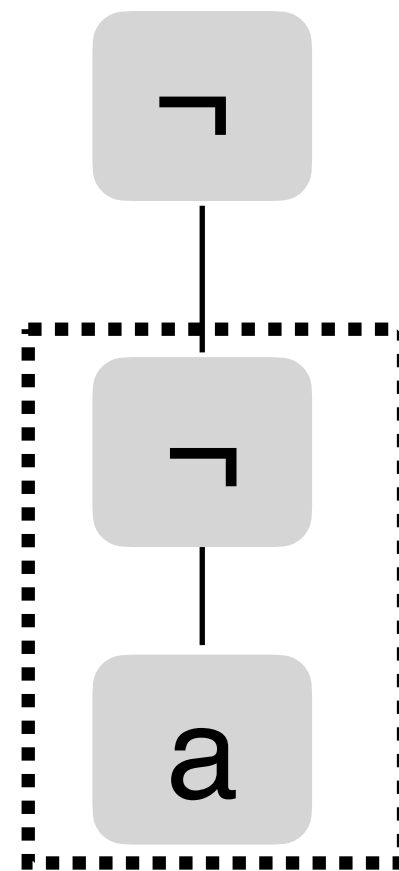
✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

✓ $(\neg(\neg a)) \rightarrow a$

Heuristic for Finding Relevant Rules

$$(((\neg(e - 1)) \wedge (\neg(1 + (\neg(e - d))))) + (1 + (\neg(e - d))))$$

AST of the LHS



Sub-exprs of height 1

✗ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$

✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$

✓ $(\neg(\neg a)) \rightarrow a$

Scoped Equality Saturation

- After reducing **Region (2)**, now we have

$$\begin{aligned} & (-(((d - c - 2) \oplus (-d - 1)) + (((d - c - 2) \wedge (-d - 1)) \ll 1))) \vee \\ & (\neg((e - 1) \vee (d - e)) - (e - d)) \end{aligned}$$

which is small enough for equality saturation.

- Equality saturation with relevant rules

gives us $(d - e) \vee ((-e) \vee (c + 1))$

which is the ground truth.

✓	$a \ll 1 \rightarrow 2 \times a$
✓	$a \oplus b + 2 \times a \wedge b \rightarrow a + b$
✓	$((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
✗	$(1 + a) \rightarrow (\neg(\neg a))$
✓	$(\neg(\neg a)) \rightarrow a$

Scoped Equality Saturation

- After reducing **Region (2)**, now we have

$$\begin{aligned} & (-(((d - c - 2) \oplus (-d - 1))) + (((d - c - 2) \wedge (-d - 1)) \ll 1))) \vee \\ & (\neg((e - 1) \vee (d - e)) - (e - d)) \end{aligned}$$

which is small enough for equality saturation.

- Equality saturation with relevant rules

gives us $(d - e) \vee ((-e) \vee (c + 1))$

which is the ground truth.

- ✓ $a \ll 1 \rightarrow 2 \times a$
- ✓ $a \oplus b + 2 \times a \wedge b \rightarrow a + b$
- ✓ $((\neg a) \wedge (\neg b)) \rightarrow (\neg(a \vee b))$
- ✗ $(1 + a) \rightarrow (\neg(\neg a))$
- ✓ $(\neg(\neg a)) \rightarrow a$

Heuristic for Finding Relevant Rules

- Our rule choice heuristic is neither sound nor complete.
 - i.e., we may select irrelevant rules or miss relevant rules.
- To mitigate the hazard of missing useful rules, we always include base rules that are useful in many cases.
- Seems to work well in practice
 - Reduce time for Eqsat by up to 50%
 - Disabling the rule choice heuristic (i.e., including all the available rules) increases runtime without further simplification.

Experimental Setup

- Implementation : ProMBA
- Major Components
 - Synthesizer : SimbaS [Yoon et al. 2023]
 - Linear MBA Simplifier : SimbaD [Reichenwallner and Meerwald-Stadler. 2022]
 - Equality saturation : egg [Willsey et al. 2021]
 - Correctness Verifier: Z3

Experimental Setup (Dataset)

- 4011 **non-linear** MBA expressions

= Original Expr

- MBA-Solver [PLDI '21] : 2011 MBA Expr (Complex Groundtruth)

- **Small** Polynomial + Non-polynomial MBA with limited operators

- QSynth [BAR '20] : 500 MBA Expr (Complex Groundtruth)

- **Medium-sized** Non-polynomial MBA with limited operators

- Loki [USENIX '22] : 1000 MBA Expr (Simple Groundtruth)

- **Huge** Non-polynomial MBA with **all of the arithmetic and logical operators**

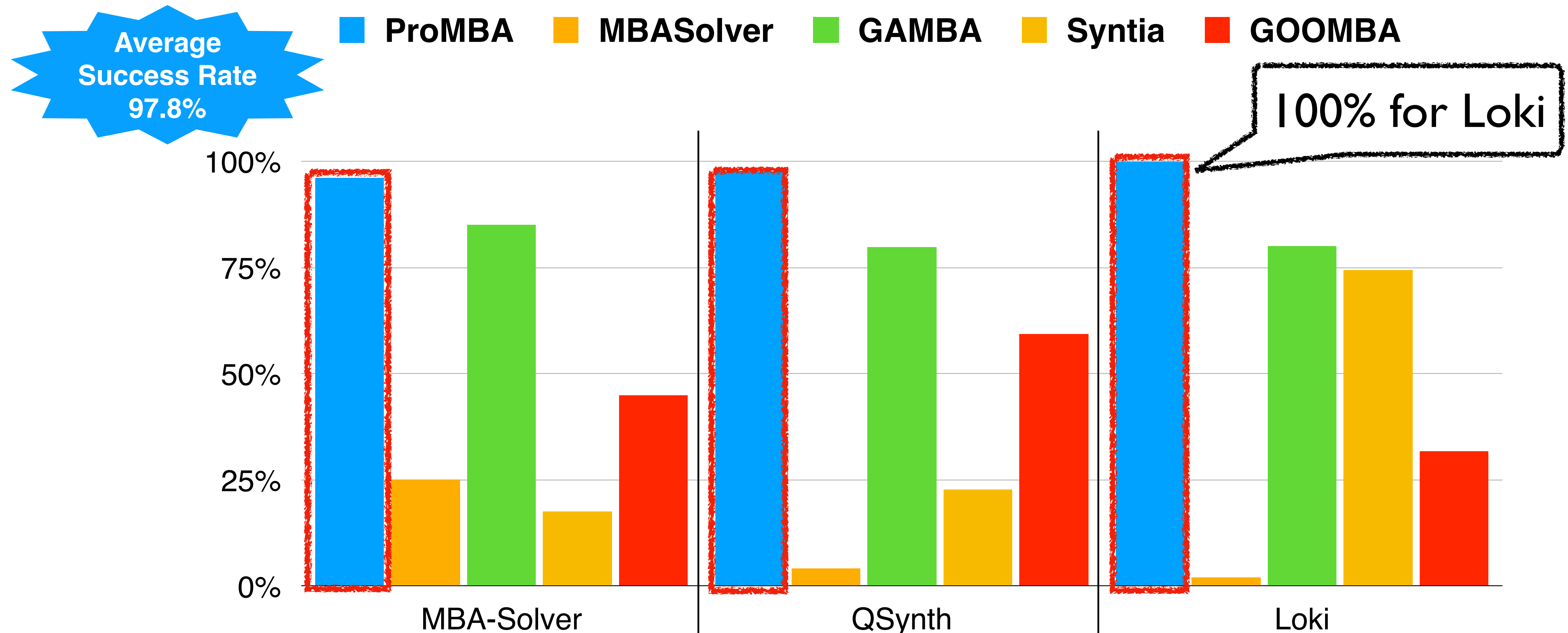
- 1 hr timeout

Experimental Setup (Baselines)

- MBASolver [PLDI '22]
 - Sound, algebraic method, limited to polynomial MBA expressions
- Syntia [USENIX '17]
 - Unsound, synthesis-based blackbox MBA Deobfuscator
 - Synthesis from 10 random I/O Examples, Use best outcome in 30 trial
- GAMBA [WORMA' 23]
 - Sound, algebraic method based, state-of-the-art MBA Deobfuscator
- GOOMBA [2023]
 - Sound, algebraic method + table lookup, plugin for the Hex-Rays decompiler

Success Rate

- Average deobfuscation success rate of 97.8%, significantly outperforming all the other tools (13%, 82.5%, 39.4%, 41.8%)

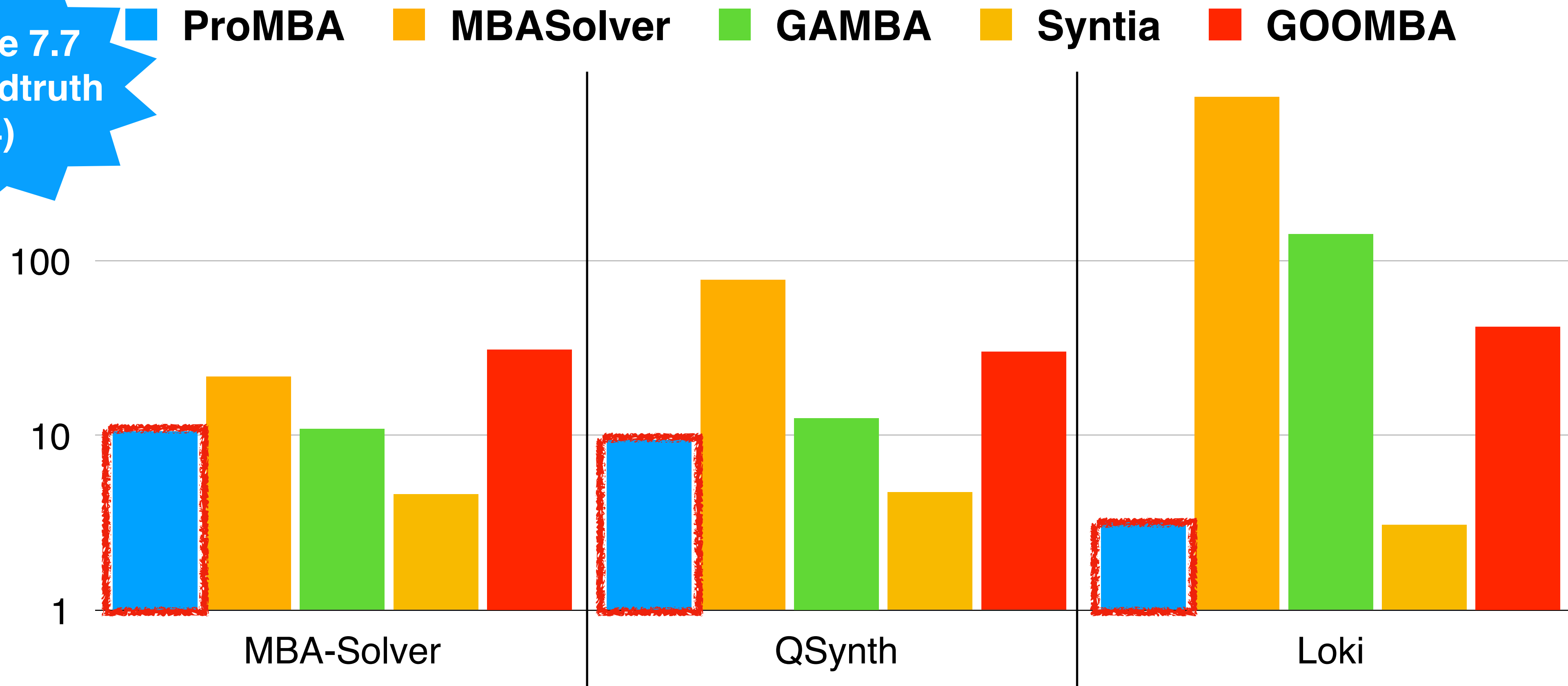


Size of Results

- Generates the smallest results among sound tools

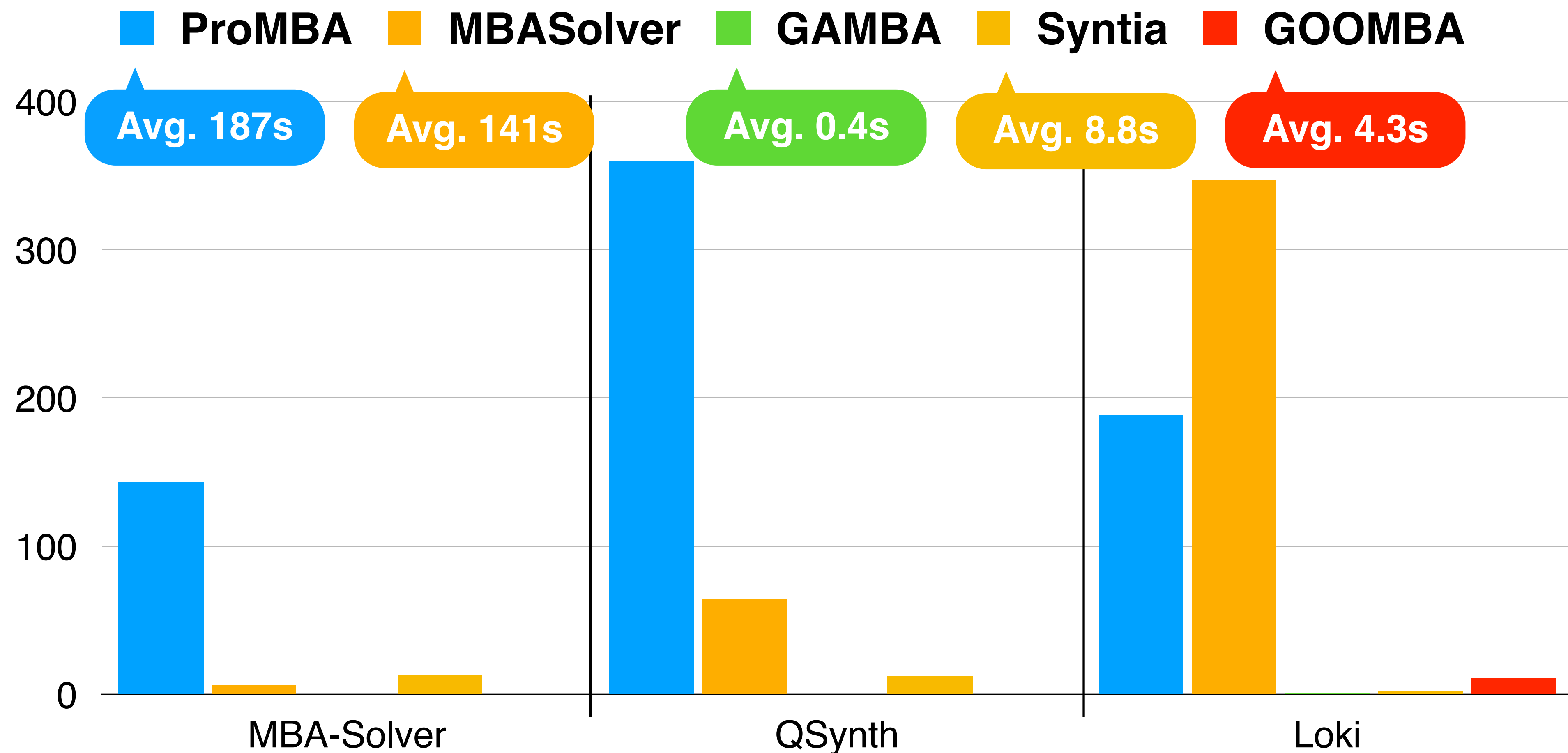
Avg Size 3.6
But Can Handle Only
Small Groundtruth

Avg Size 7.7
($<$ Groundtruth
9.64)

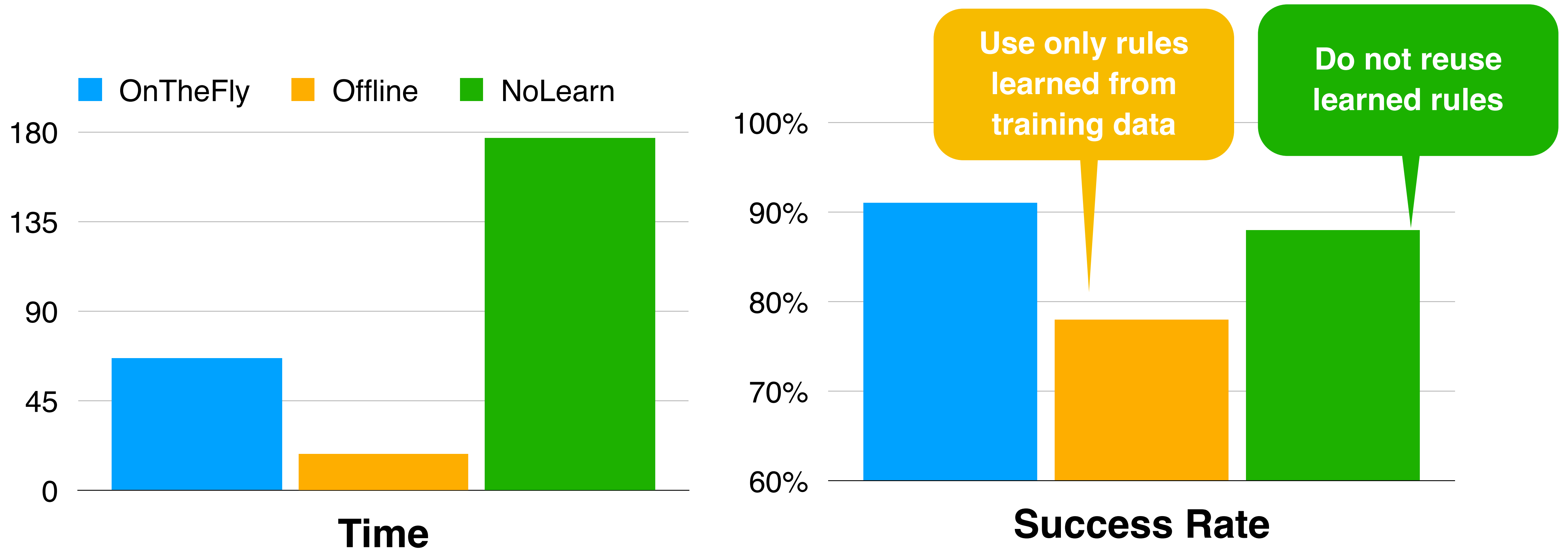


Deobfuscation Time

- Slower than the other tools 😓

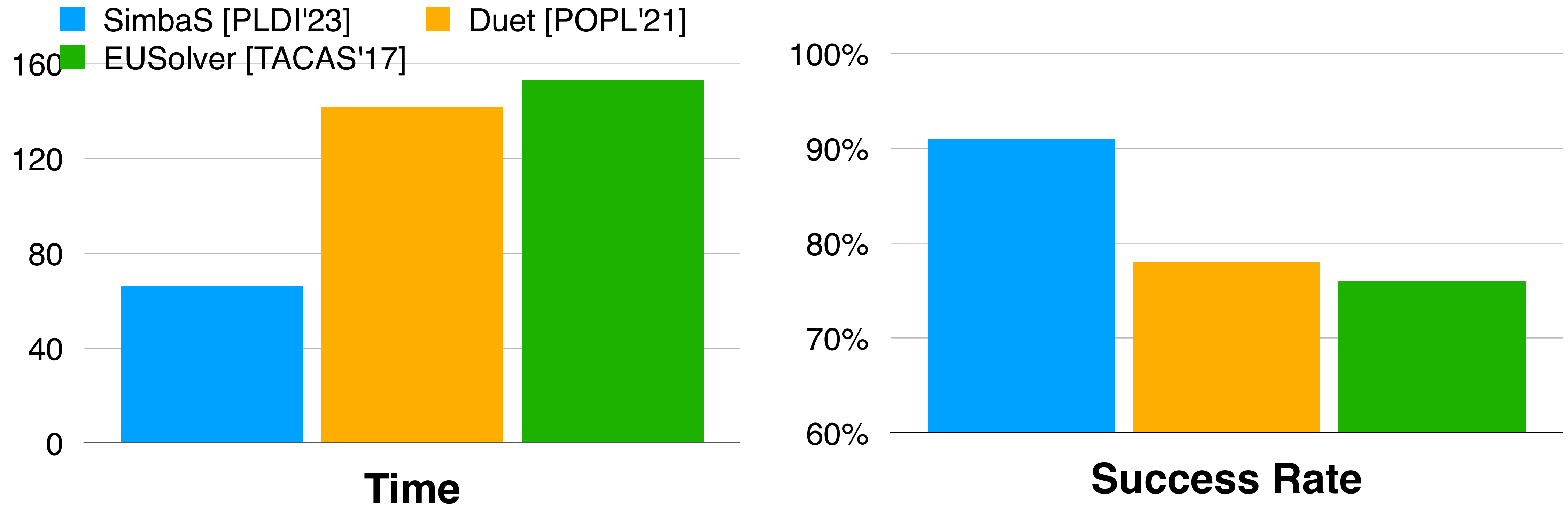


Efficacy of on-the-fly learning



- On the fly learning is better than offline learning or no learning

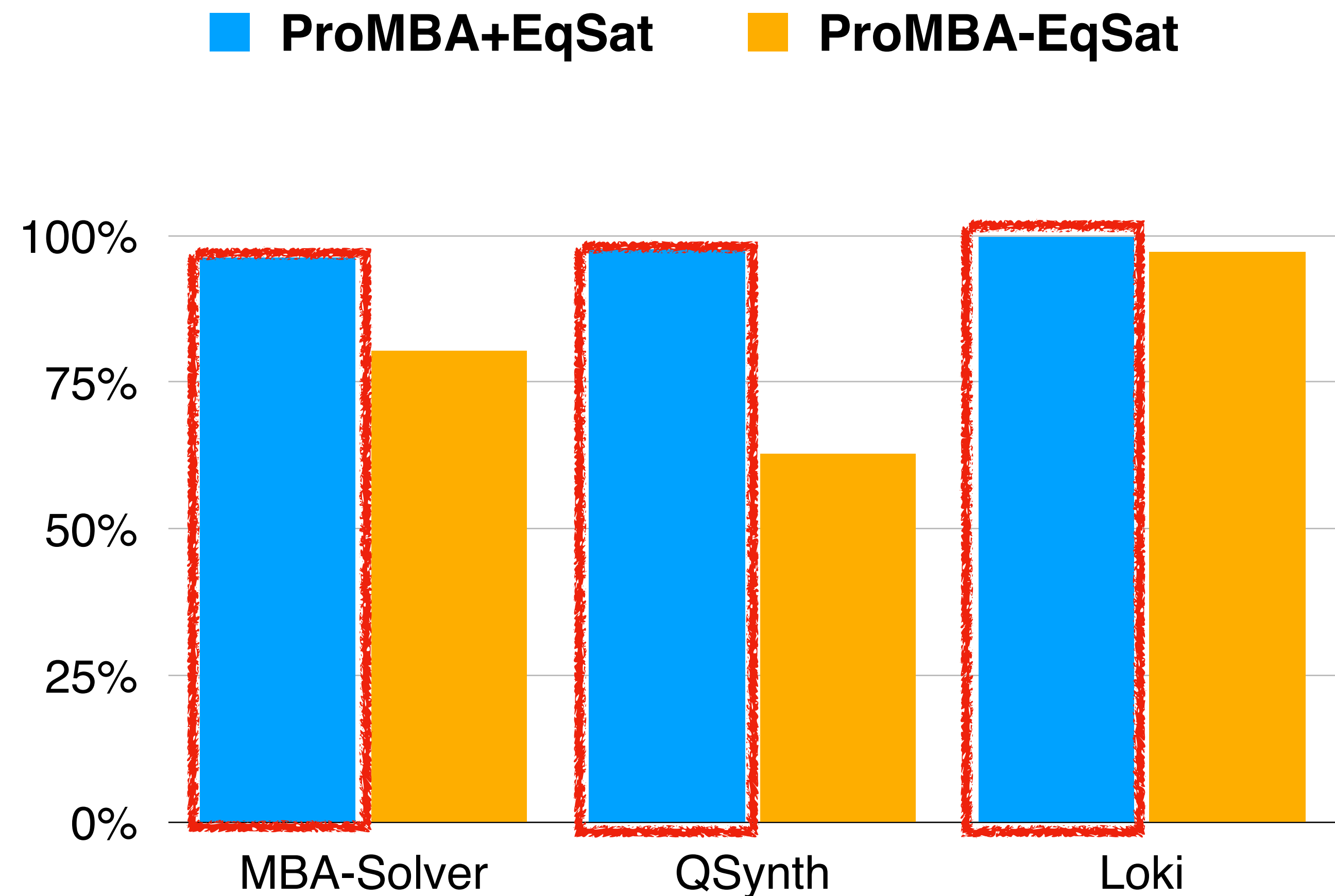
Impact of Underlying Synthesizer



- Synthesis performance : $\text{SimbaS} > \text{Duet} > \text{EUSolver}$
- Rules discovered by a better synthesizer lead to better simplification.

Efficacy of Scoped Equality Saturation

- Increased success rate:
84.4% → 97.8%
- Reduced average size
of deobfuscated results
: 9.4 → 7.7 (in AST
nodes)
- Increased runtime:
100s → 187s 🥵



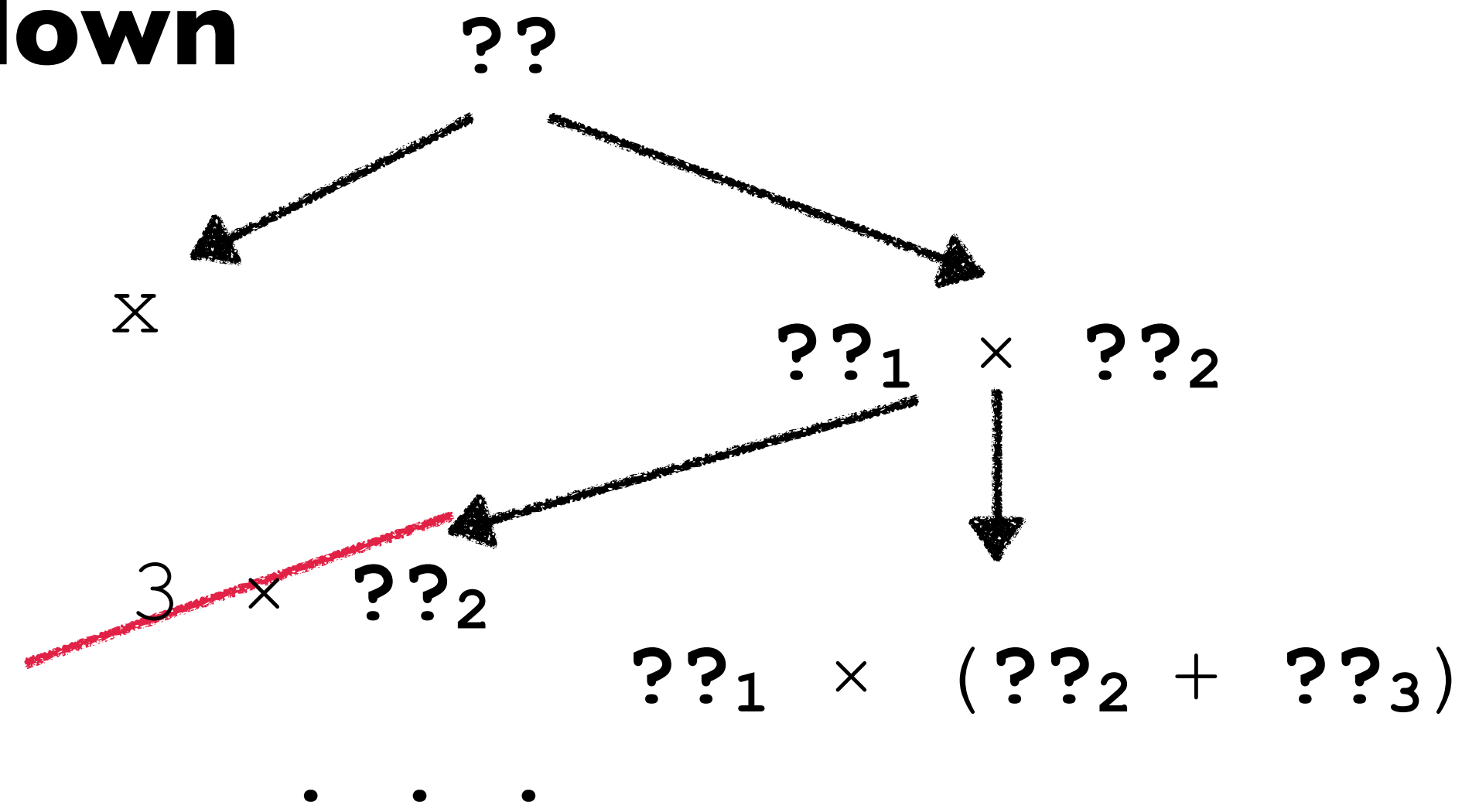
Contents

- MBA Deobfuscation with Program Synthesis and Term Rewriting
- Advanced MBA Deobfuscation with Scoped Equality Saturation
- **Enabling technology: high-performance program synthesis**
 - Yongho Yoon, Woosuk Lee, and Kwangkeun Yi, Inductive Program Synthesis via Iterative Forward-Backward Abstract Interpretation. **ACM PLDI 2023**
 - Doyoon Lee, Woosuk Lee, and Kwangkeun Yi, Inductive Program Synthesis by Meta-Analysis-Guided Hole Filling **ACM POPL 2026 (conditionally accepted)**

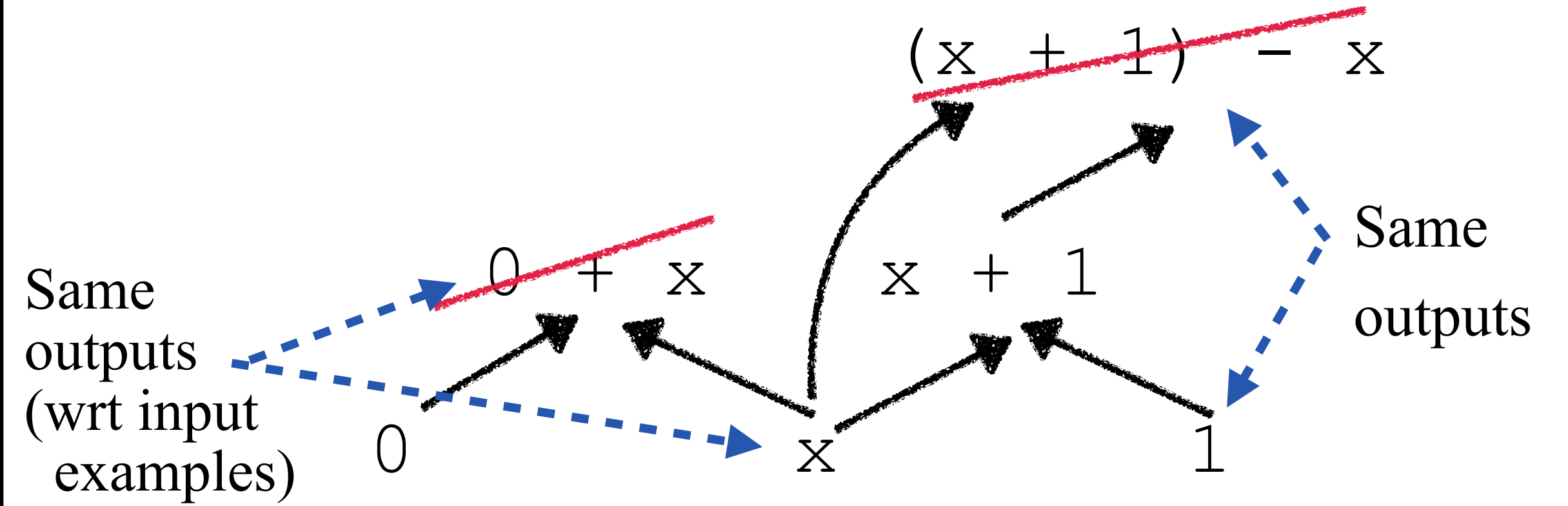
Background: Two Strategies for Synthesis

function $f\ x = ??$ (spec: $0 \mapsto 0, 1 \mapsto 2$)

Top-down



Bottom-up



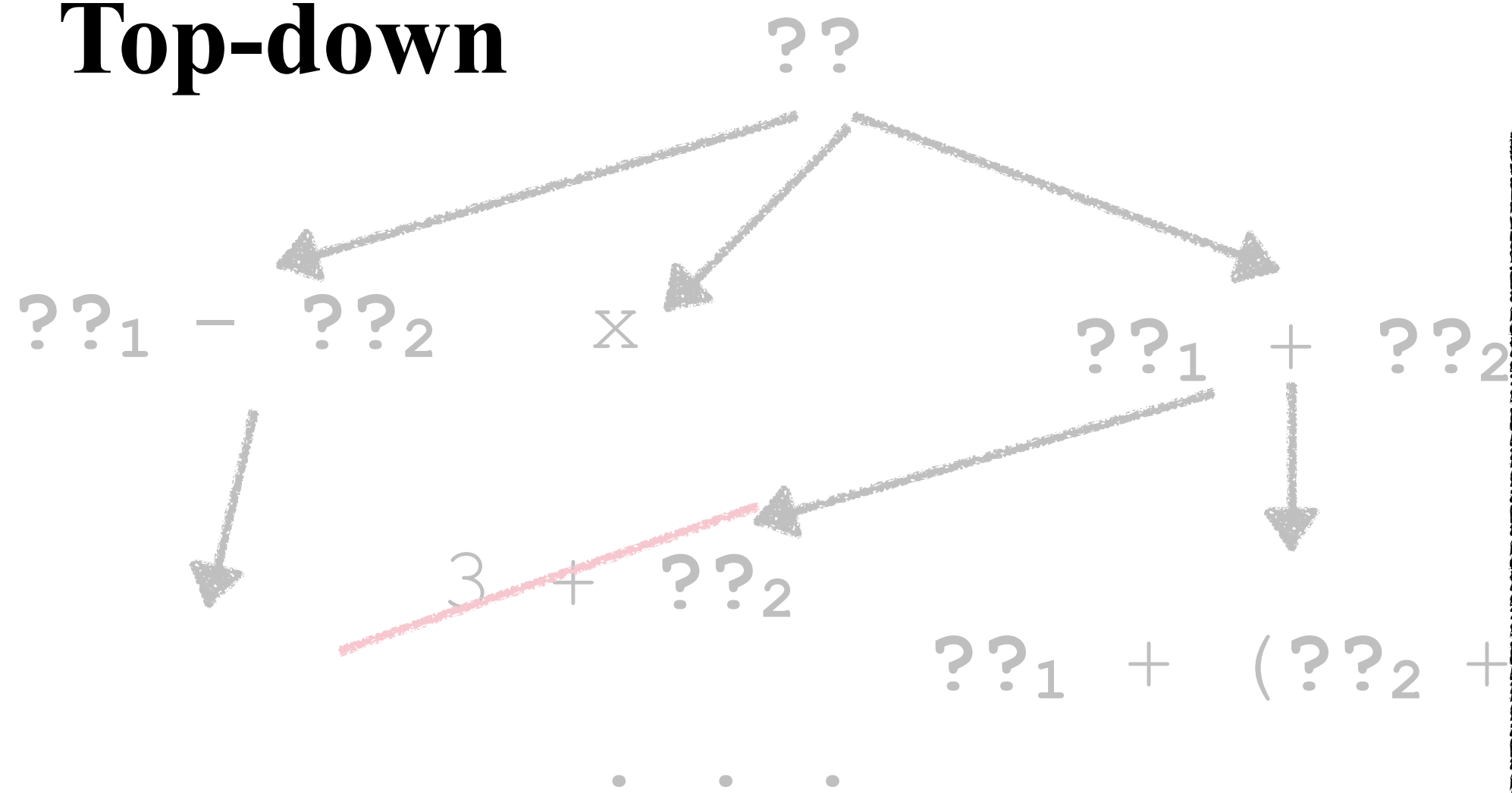
- Starts from empty program, fills in holes
- Prune *infeasible* partial programs by domain-specific reasoning

- Builds larger programs from smaller ones
- Prune *redundant* expressions by evaluation (*observational equivalence* reduction)

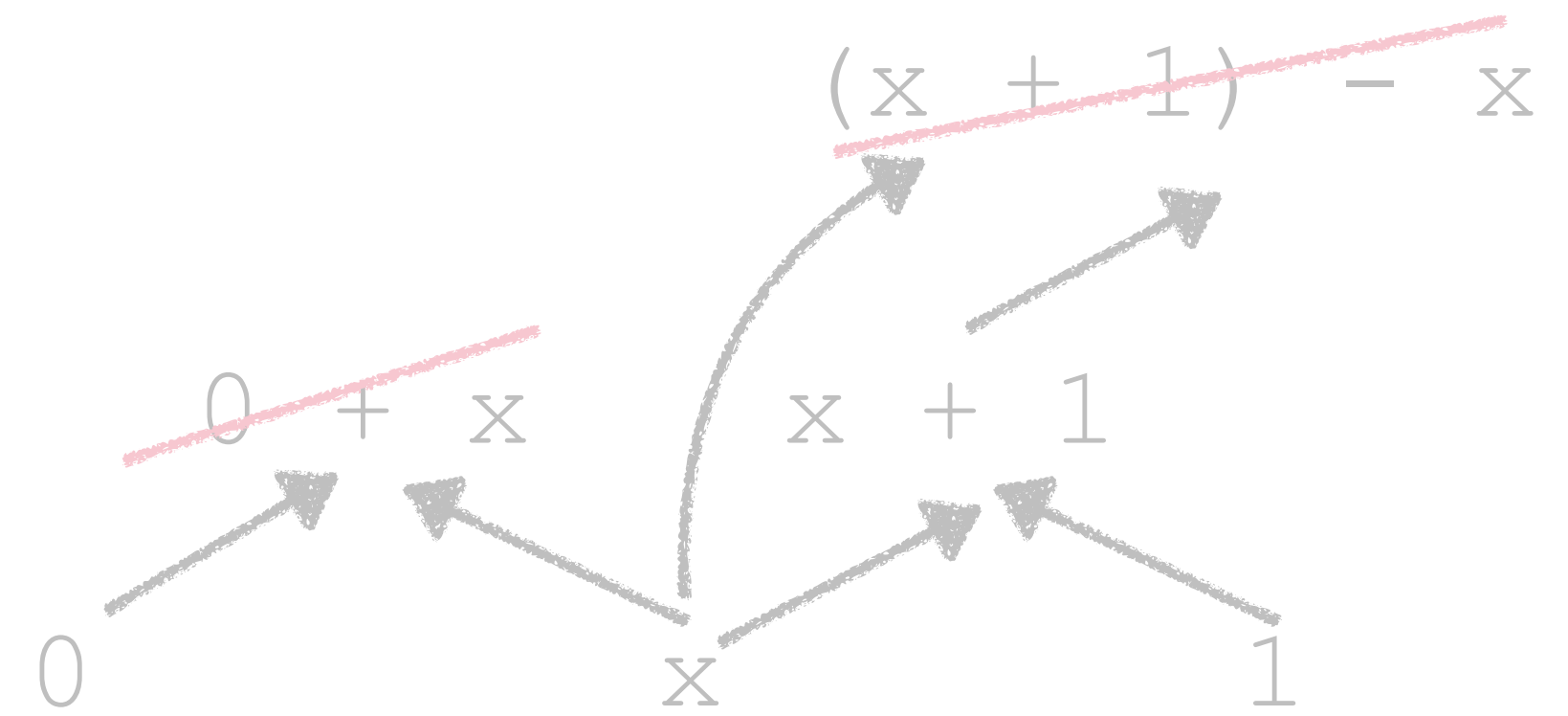
Bidirectional Synthesis†

- Partial programs with holes by top-down, component expressions by bottom-up
- Missing holes are filled with the component expressions.
- Infeasible partial programs and redundant component expressions are pruned.

Top-down



Bottom-up



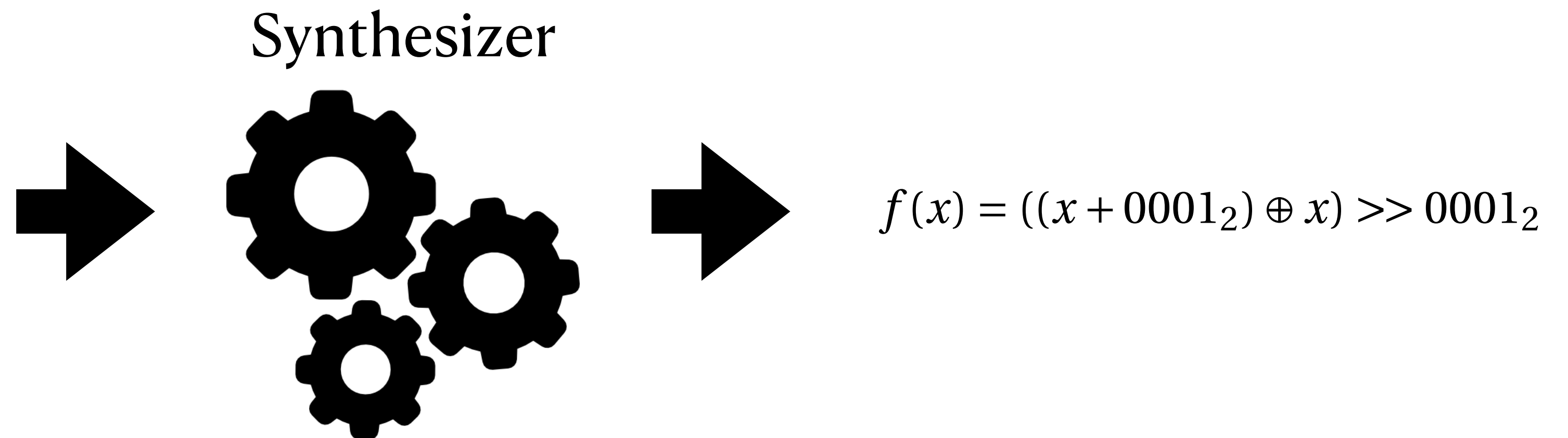
Inductive Program Synthesis Example

Syntactic Spec

$S \rightarrow x \mid 0001_2$
| $S \wedge S \mid S \vee S \mid S \oplus S$
| $S + S \mid S \times S \mid S / S \mid S \gg S$

Semantic Spec

$f(1011_2) = 0011_2$

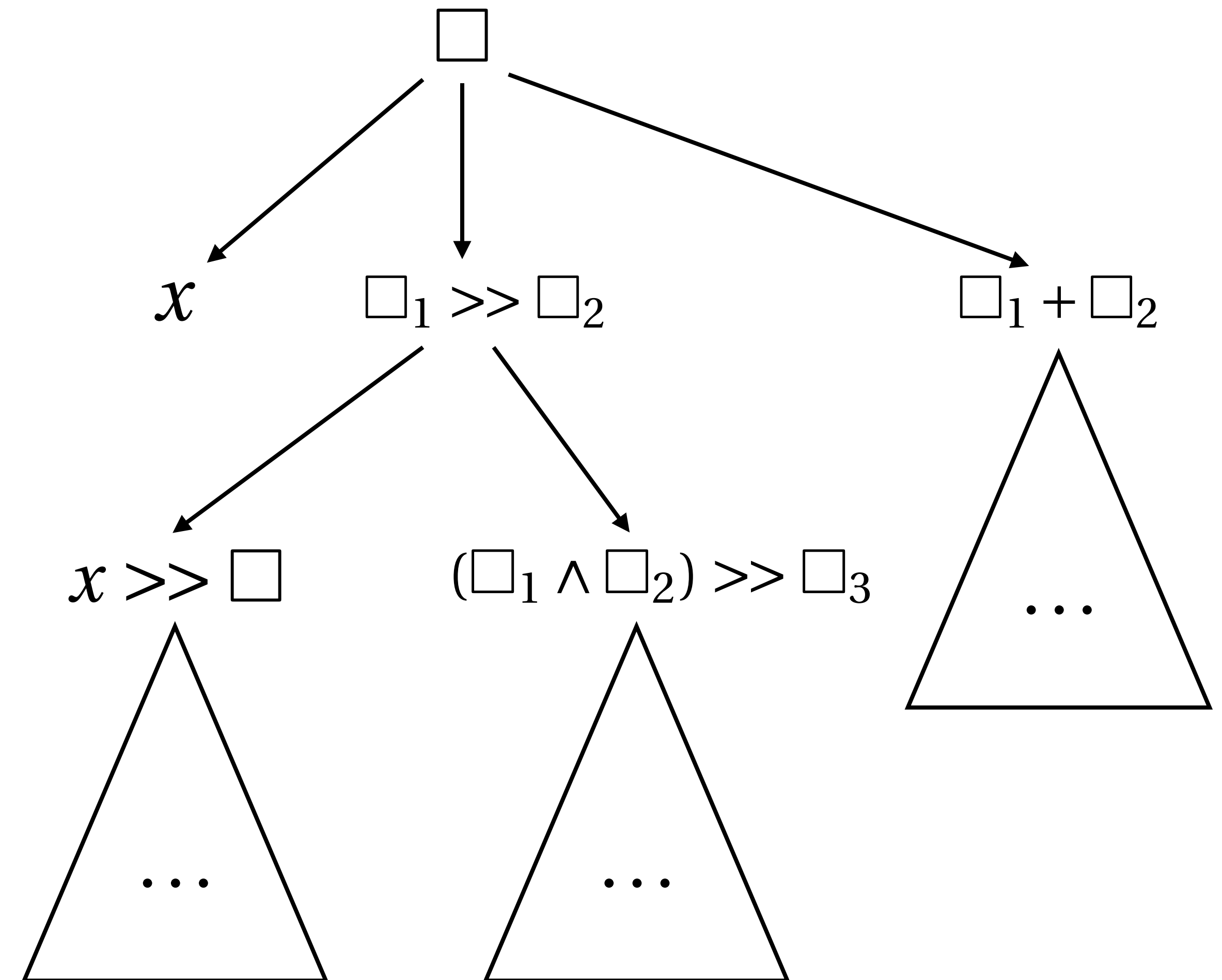


Pruning by Feasibility

Syntactic Spec

$$\begin{array}{lcl} S & \rightarrow & x \mid 0001_2 \\ & | & S \wedge S \mid S \vee S \mid S \oplus S \\ & | & S + S \mid S \times S \mid S / S \mid S \gg S \end{array}$$

Semantic Spec

$$f(1011_2) = 0011_2$$


Pruning by Feasibility

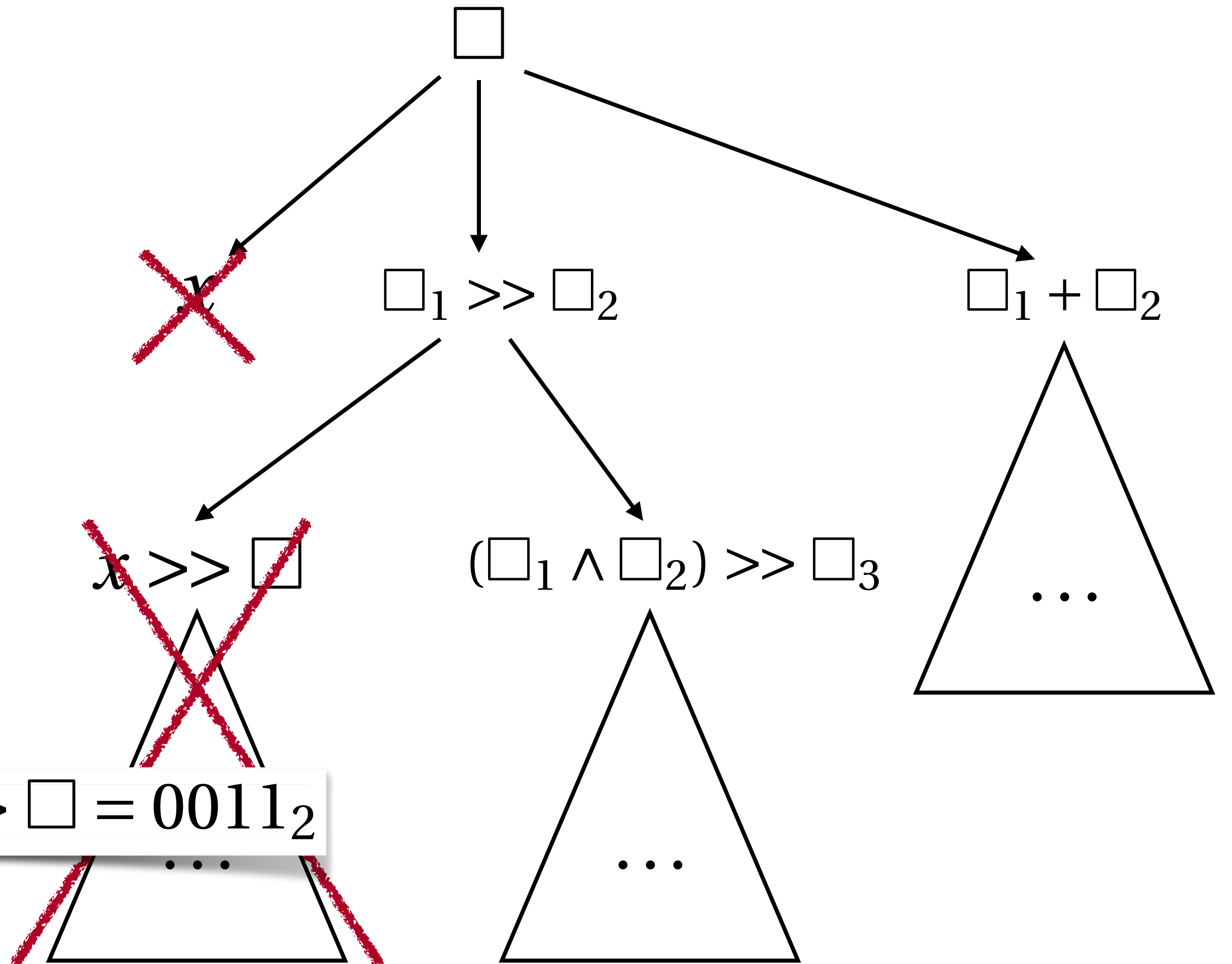
Syntactic Spec

$S \rightarrow x | 0001_2$
| $S \wedge S | S \vee S | S \oplus S$
| $S + S | S \times S | S / S | S \gg S$

Semantic Spec

$f(1011_2) = 0011_2$

$\nexists \square. f(1011_2) = 1011_2 \gg \square = 0011_2$



Pruning by Feasibility

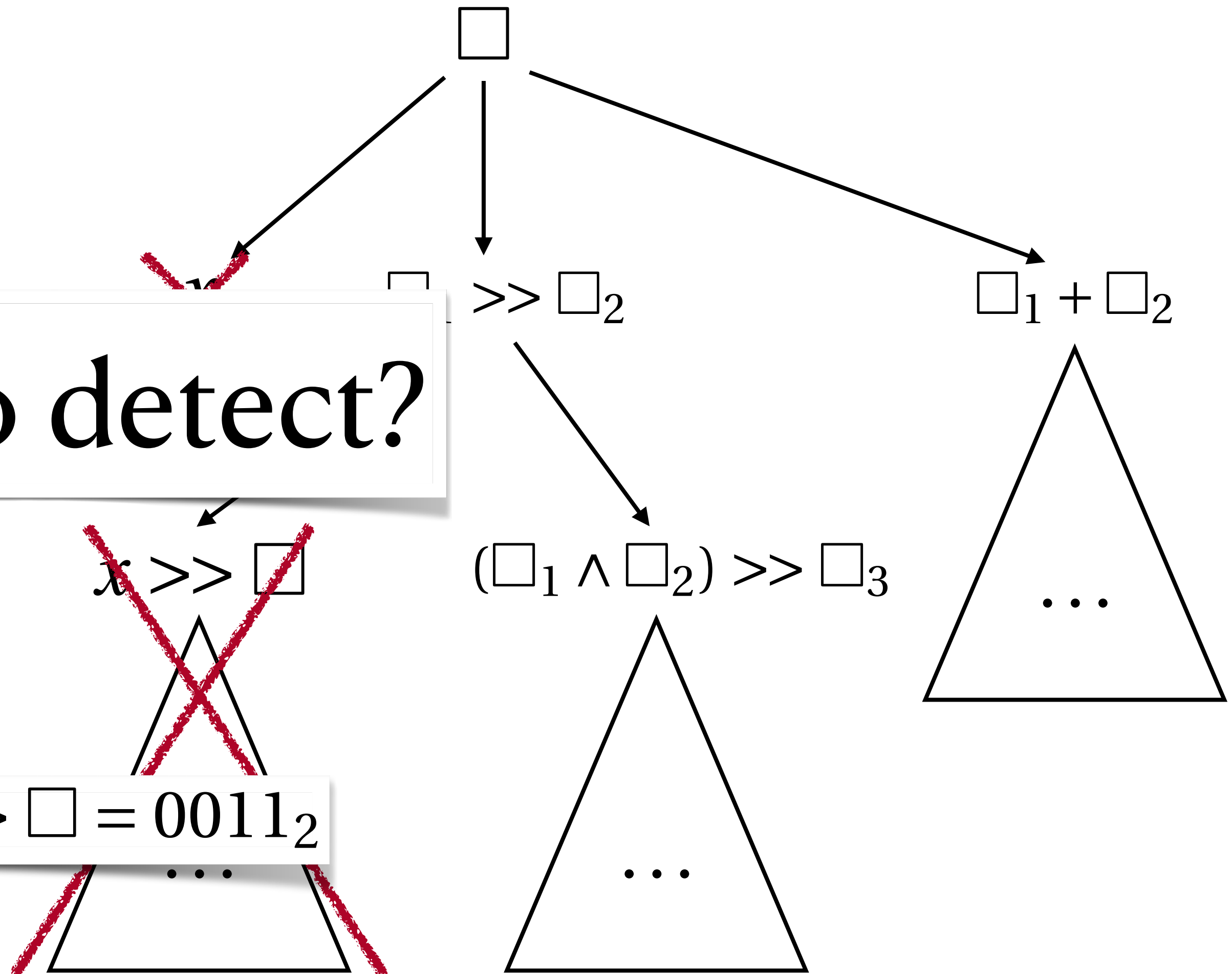
Syntactic Spec

$$\begin{array}{lcl} S & \rightarrow & x \mid 0001_2 \\ & | & S \wedge S \mid S \vee S \mid S \oplus S \\ & | & S + S \mid S \times S \mid S / S \mid S \gg S \end{array}$$

Semantic Spec

$$f(1011_2) = 0011_2$$

How to detect?

$$\nexists \square. f(1011_2) = 1011_2 \gg \square = 0011_2$$


Static Analysis

- To over-approximate all possible behaviors of programs
- Perform simulation of program execution with “abstract” values instead of concrete ones
 - Abstraction = over-approximation (e.g., concrete : $\{0, 2, 6\} \rightarrow$ abstract : even)
- E.g., bitfield abstraction
 - Each bit is represented by $\{0, 1, \perp, \top\}$ where $\top$: unknown, $\perp$: no value
 - e.g., $\top 0 \mid \top$ represents a set $\{00 \mid 0_2, 00 \mid 1_2, 10 \mid 0_2, 10 \mid 1_2\}$
- Abstract operators (denoted with #)
 - e.g., $1 \top 10 \wedge^\# 00 \top \top = 00 \top 0$
- **Forward analysis** computes possible outputs of a program given inputs.
- **Backward analysis** computes possible inputs of a program given outputs.

Using Forward Analysis

Checking only output feasibility

Forward



x	$\mapsto$	1011
$\square$	$\mapsto$	TTTT

Candidate Partial Program

$$f(x) = x \vee \square$$

Semantic Spec

$$f(1011_2) = 0011_2$$

Using Forward Analysis

Checking only output feasibility

Semantic Spec

$$f(1011_2) = 0011_2$$

Candidate Partial Program

$$f(x) = x \vee \square$$

Forward

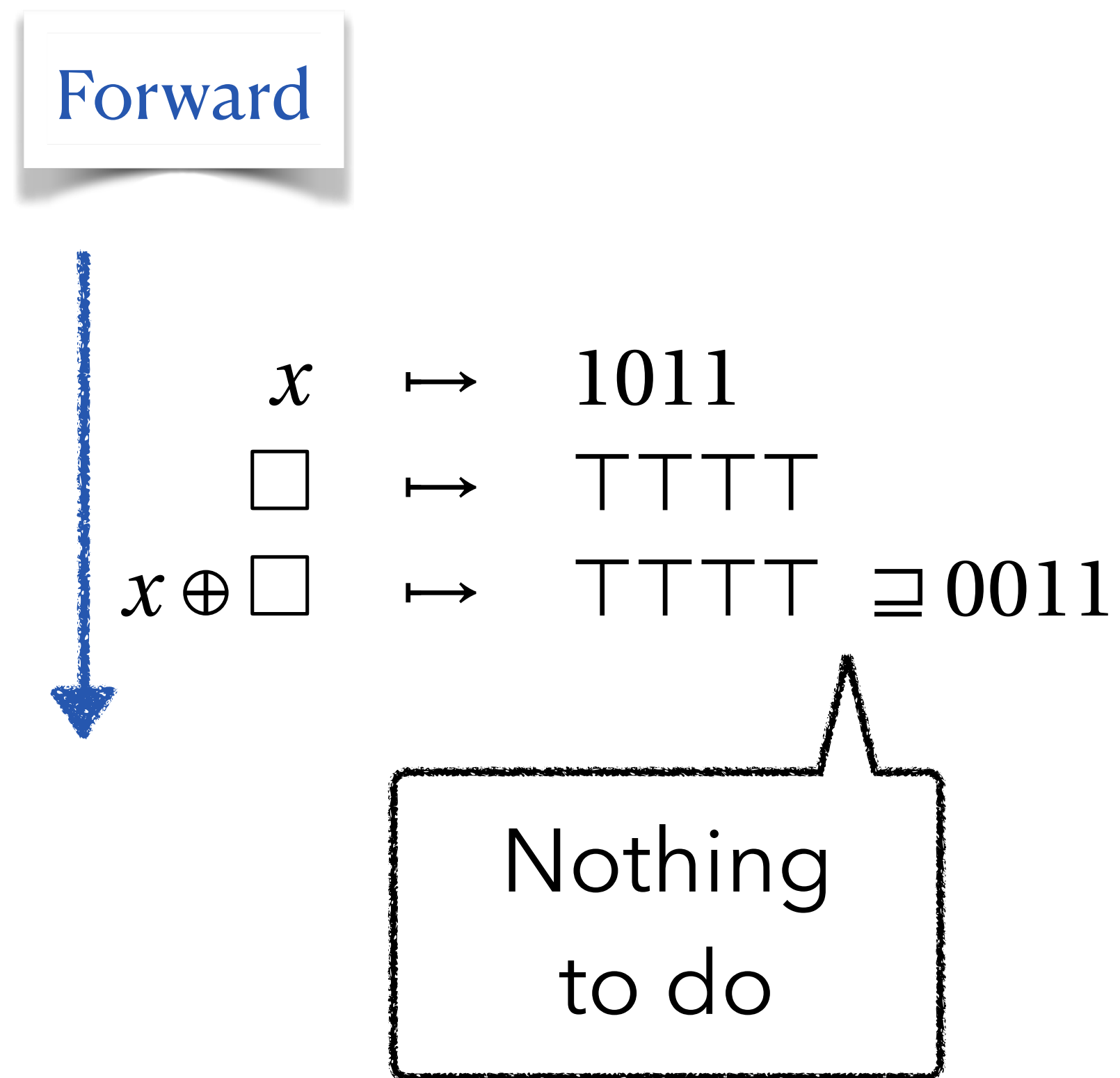


$$\begin{array}{lll} x & \mapsto & 1011 \\ \square & \mapsto & \top\top\top\top \\ x \vee \square & \mapsto & 1\top11 \not\equiv 0011 \end{array}$$

Infeasible
output

Limitation of Forward Analysis

Checking only output feasibility

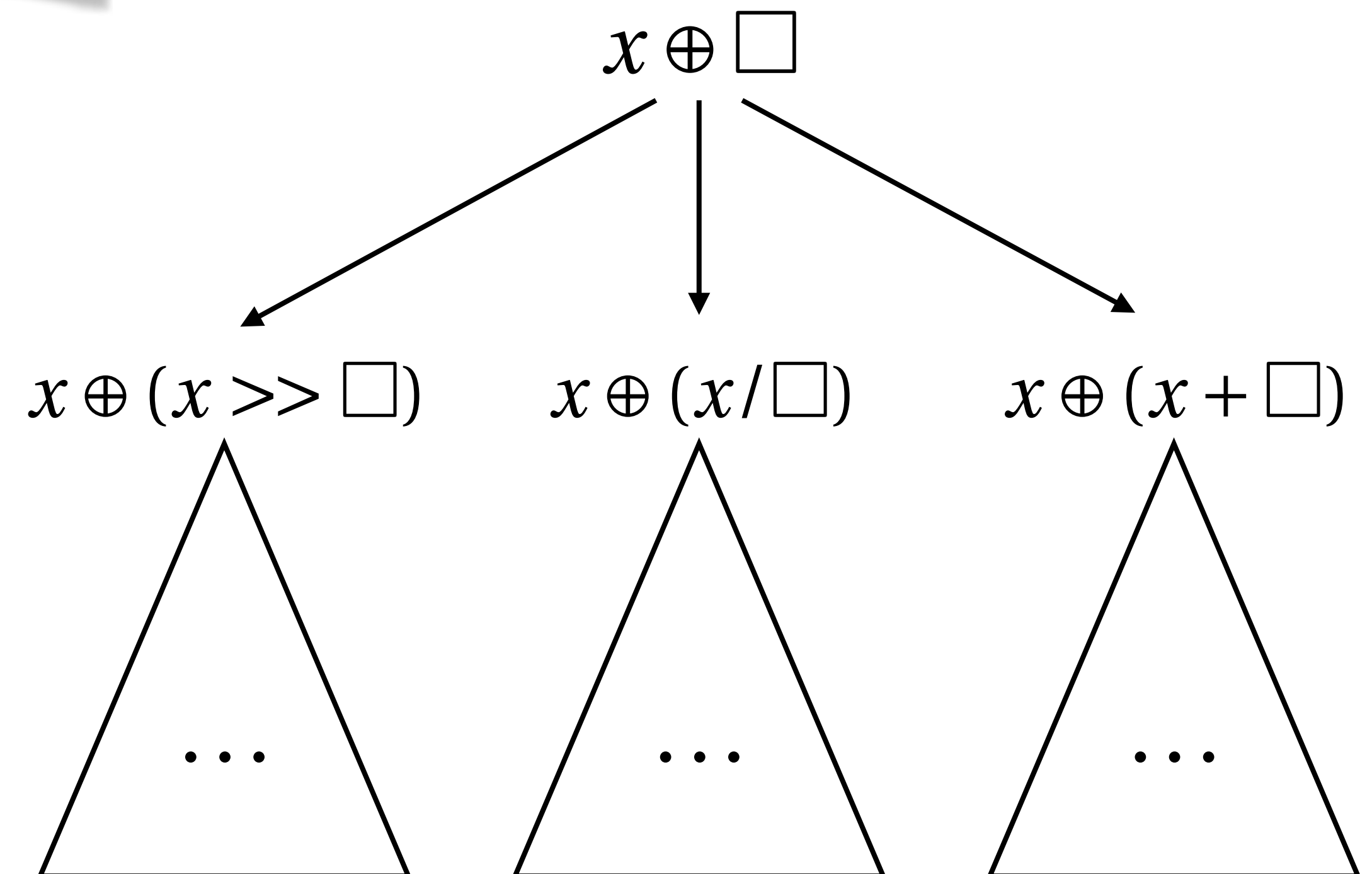


Candidate Partial Program

$$f(x) = x \oplus \square$$

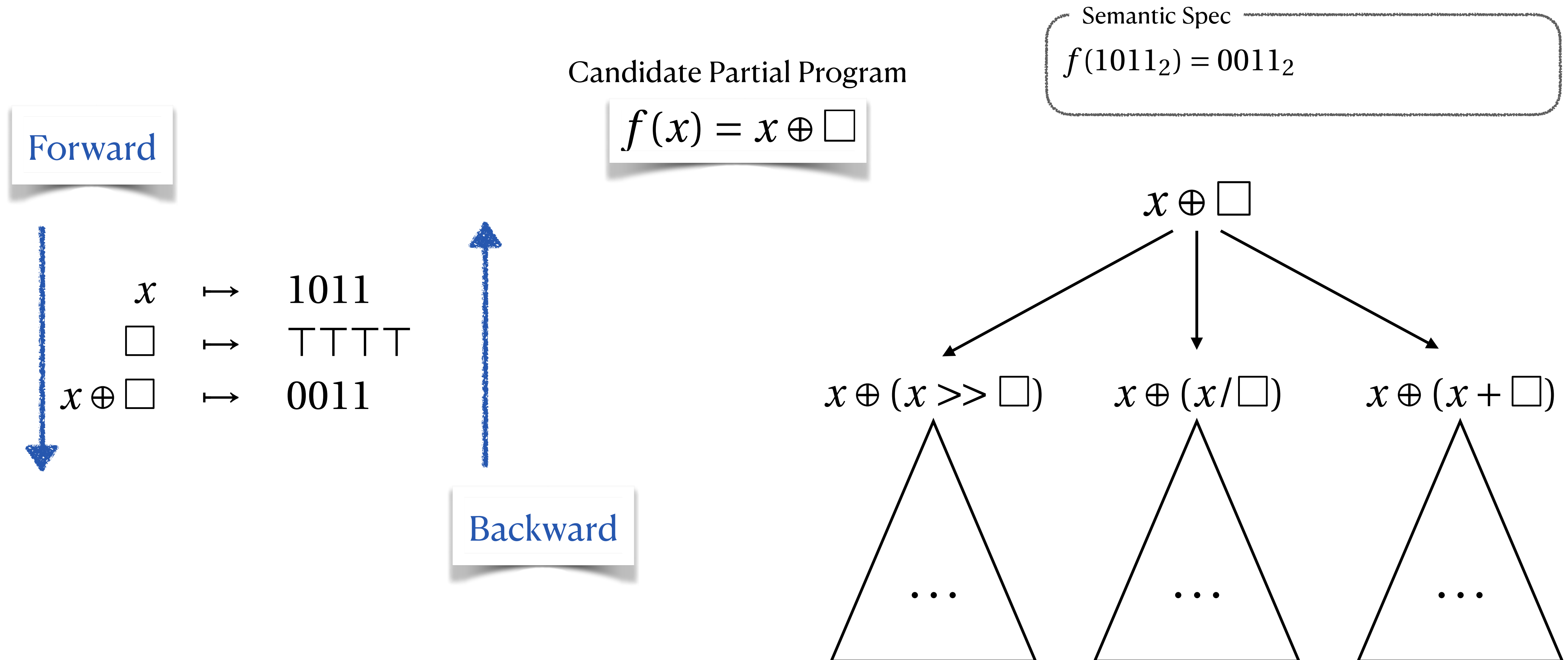
Semantic Spec

$$f(1011_2) = 0011_2$$



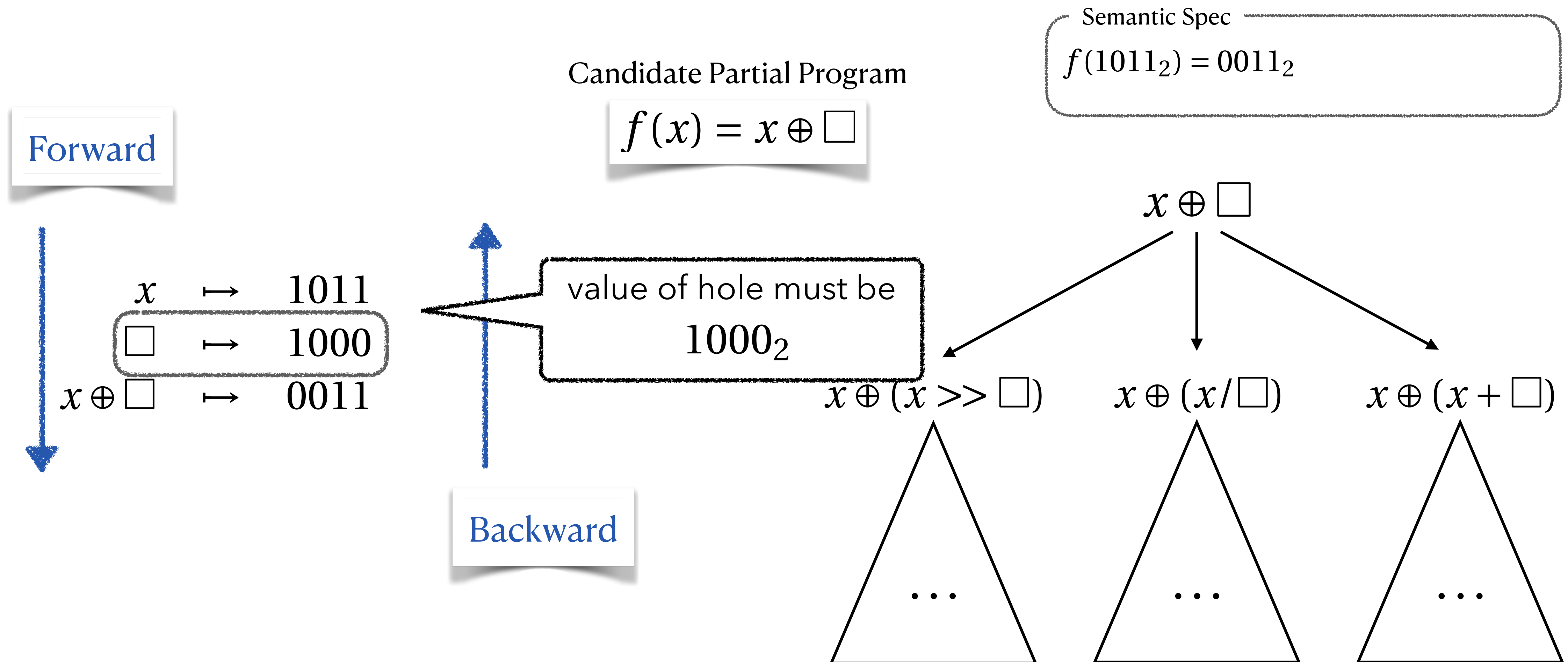
Need: Backward Analysis Too

Output feasibility + Hole Precondition



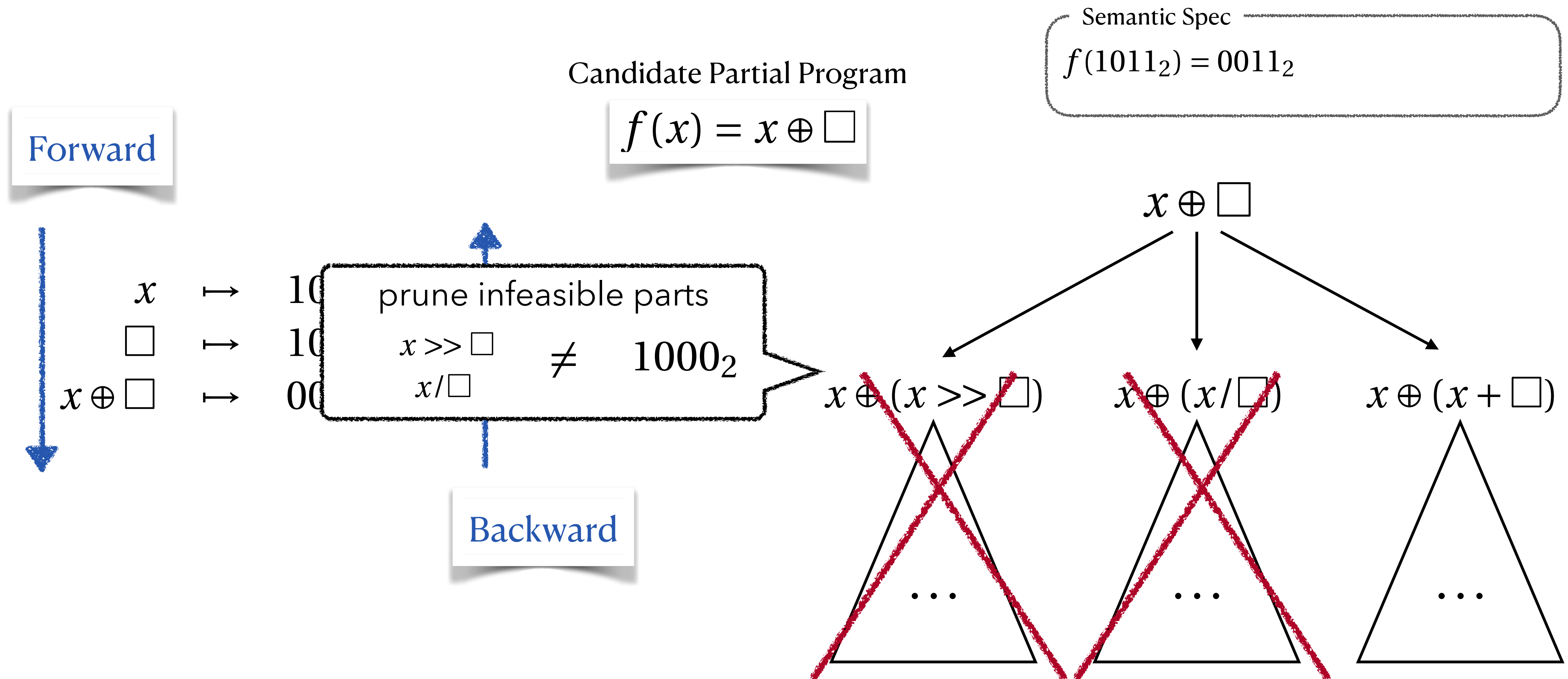
Need: Backward Analysis Too

Output feasibility + Hole Precondition



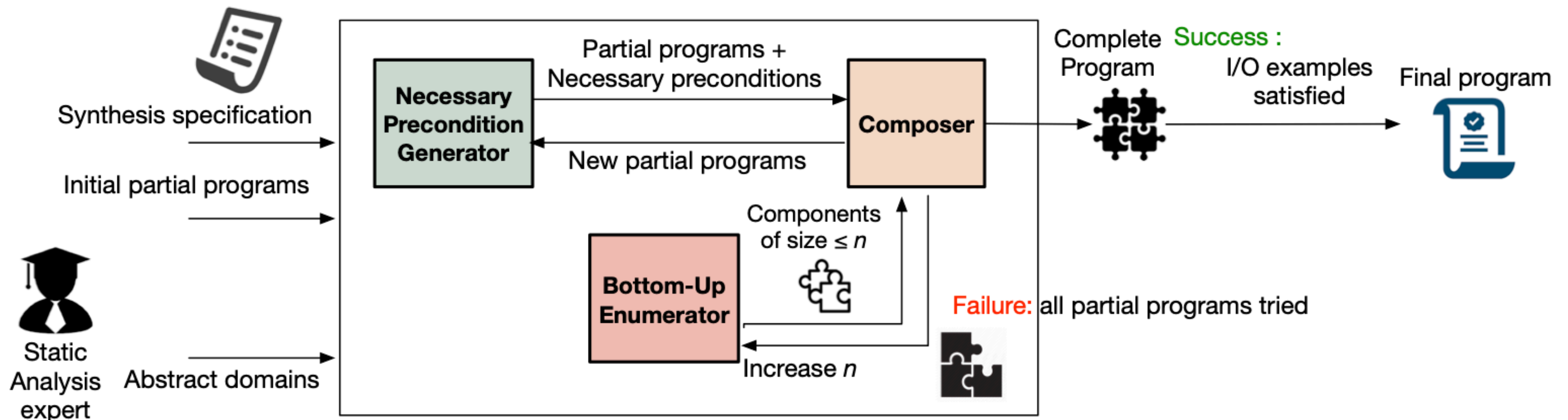
Need: Backward Analysis Too

Output feasibility + Hole Precondition

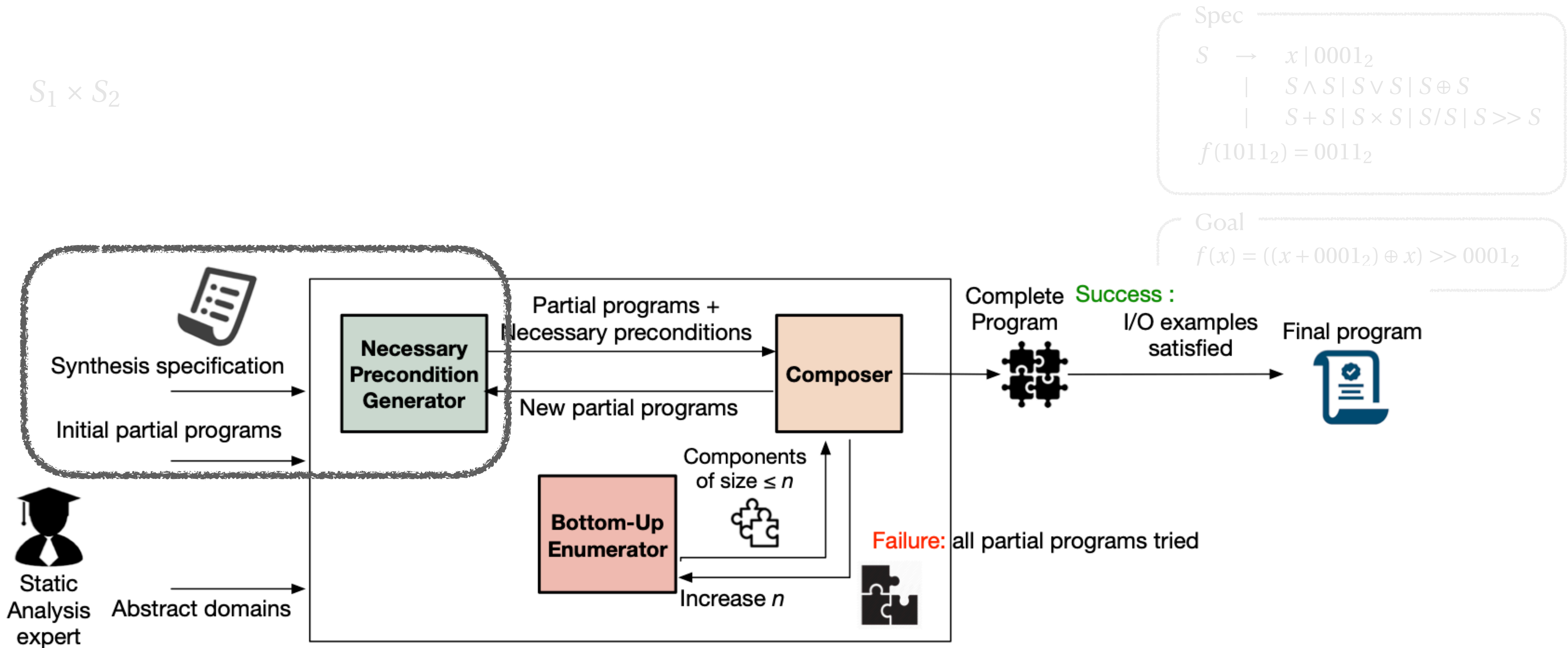


Realized: SIMBA

- 10K+ lines of OCaml code
- Available at <https://github.com/yhyoon/simba>
- Currently being used by ProMBA



Synthesis Algorithm



Synthesis Algorithm

$$(\square \oplus x) \gg 0001_2$$

Initial partial programs
chosen for illustration purpose

$$(\square / x) \gg 0001_2$$

Spec

$$\begin{aligned} S &\rightarrow x | 0001_2 \\ &| S \wedge S | S \vee S | S \oplus S \\ &| S + S | S \times S | S / S | S \gg S \\ f(1011_2) &= 0011_2 \end{aligned}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Synthesis Algorithm

$$(\square \oplus x) \gg 0001_2$$

Forward Analysis

$$\begin{array}{ll} \square & \mapsto \text{TTTT} \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto \text{TTTT} \\ (\square \oplus x) \gg 0001_2 & \mapsto \text{TTTT} \end{array}$$

$$(\square / x) \gg 0001_2$$

Spec

$$\begin{array}{l} S \rightarrow x | 0001_2 \\ \quad | \quad S \wedge S | S \vee S | S \oplus S \\ \quad | \quad S + S | S \times S | S / S | S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Synthesis Algorithm

$$(\square \oplus x) \gg 0001_2$$

$$(\square / x) \gg 0001_2$$

Backward Analysis

$$\begin{array}{ll} \square & \mapsto 110\top \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto 011\top \\ (\square \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Spec

$$\begin{array}{l} S \rightarrow x | 0001_2 \\ \quad | S \wedge S | S \vee S | S \oplus S \\ \quad | S + S | S \times S | S / S | S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Backward BV xor

$$\begin{array}{l} \overleftarrow{\square \oplus^\# x = 011\top} \\ = \overleftarrow{\square \oplus^\# 1011 = 011\top} \\ = (110\top, 1011) \end{array}$$

Backward BV shift-right

$$\begin{array}{l} \overleftarrow{[\square \oplus x] \gg^\# 0001 = 0011} \\ = (011\top, 0011) \end{array}$$

Synthesis Algorithm

$$(\Box \oplus x) \gg 0001_2 \rightarrow \begin{array}{ll} \Box & \mapsto 110\top \\ x & \mapsto 1011 \\ \Box \oplus x & \mapsto 011\top \\ (\Box \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Spec

$$\begin{array}{l} S \rightarrow x \mid 0001_2 \\ \quad \mid S \wedge S \mid S \vee S \mid S \oplus S \\ \quad \mid S + S \mid S \times S \mid S / S \mid S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

$(\Box / x) \gg 0001_2$

Forward Analysis

$$\begin{array}{ll} \Box & \mapsto \top\top\top\top \\ x & \mapsto 1011 \\ \Box / x & \mapsto 000\top \end{array}$$

$$\begin{array}{ll} \Box & = [0, 15] \\ x & = [11, 11] \\ \xrightarrow{\quad} \Box /^\# x & = [l_S / h_x, h_S / l_x] \\ & = [0 / 11, 15 / 11] \\ & = [0, 1] \end{array}$$

Synthesis Algorithm

$$(\square \oplus x) \gg 0001_2 \rightarrow \begin{array}{ll} \square & \mapsto 110\top \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto 011\top \\ (\square \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Spec

$$\begin{array}{l} S \rightarrow x \mid 0001_2 \\ \quad \mid S \wedge S \mid S \vee S \mid S \oplus S \\ \quad \mid S + S \mid S \times S \mid S / S \mid S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

$$(\square / x) \gg 0001_2$$

Forward Analysis

$$\begin{array}{ll} \square & \mapsto \top\top\top\top \\ x & \mapsto 1011 \\ \square / x & \mapsto 000\top \\ (\square / x) \gg 0001_2 & \mapsto 0000 \neq 0011 \end{array}$$

Synthesis Algorithm

$$(\square \oplus x) \gg 0001_2 \rightarrow$$

$\square$	$\mapsto$	110T
x	$\mapsto$	1011
$\square \oplus x$	$\mapsto$	011T
$(\square \oplus x) \gg 0001_2$	$\mapsto$	0011

Discard

$$(\square / x) \gg 0001_2$$

$\square$	$\mapsto$	TTTT
x	$\mapsto$	1011
$\square / x$	$\mapsto$	000T
$(\square / x) \gg 0001_2$	$\mapsto$	0000

Spec

$$\begin{aligned} S &\rightarrow x \mid 0001_2 \\ &\mid S \wedge S \mid S \vee S \mid S \oplus S \\ &\mid S + S \mid S \times S \mid S / S \mid S \gg S \\ f(1011_2) &= 0011_2 \end{aligned}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Synthesis Algorithm

Partial Programs

$(\square \oplus x) \gg 0001_2 \rightarrow$

$\square$	$\mapsto$	$101\top$
x	$\mapsto$	1011
$\square \oplus x$	$\mapsto$	$011\top$
$(\square \oplus x) \gg 0001_2$	$\mapsto$	0011

Components

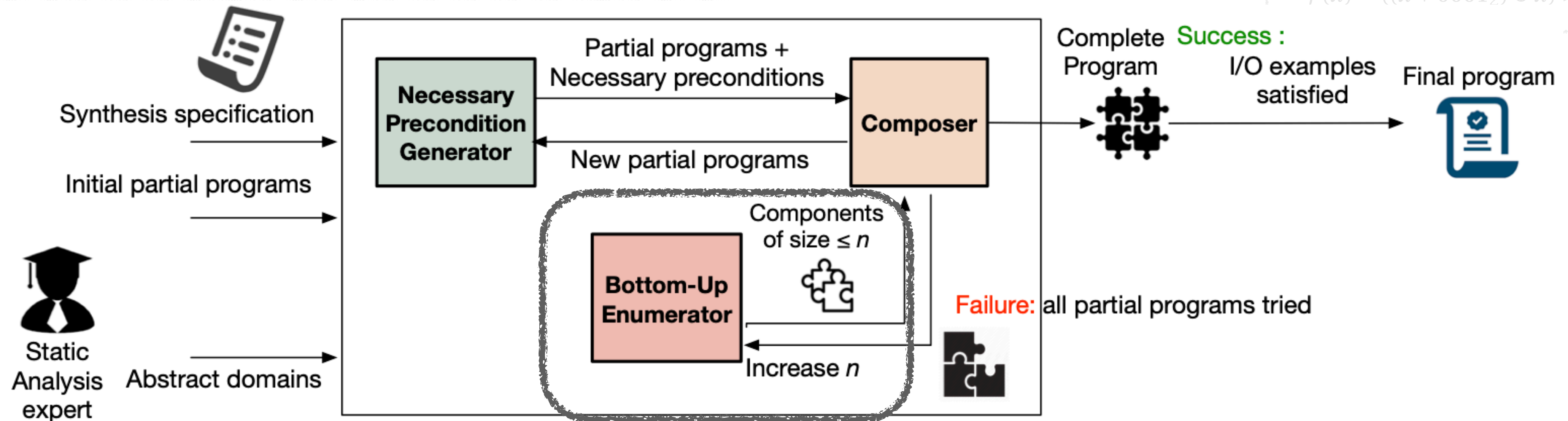
$x (= 1011_2)$
 0001_2

Spec

$S \rightarrow x | 0001_2$
 $| S \wedge S | S \vee S | S \oplus S$
 $| S + S | S \times S | S / S | S \gg S$
 $f(1011_2) = 0011_2$

Goal

$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$



Synthesis Algorithm

Partial Programs

$$(\square \oplus x) \gg 0001_2 \rightarrow \begin{array}{ll} \square & \mapsto 110\top \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto 011\top \\ (\square \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Components

$$x (= 1011_2) \\ 0001_2$$

Spec

$$\begin{array}{l} S \rightarrow x \mid 0001_2 \\ \quad \mid S \wedge S \mid S \vee S \mid S \oplus S \\ \quad \mid S + S \mid S \times S \mid S / S \mid S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Synthesis Algorithm

Partial Programs

$(\square \oplus x) \gg 0001_2 \rightarrow$

$\square$	$\mapsto$	$101\top$
x	$\mapsto$	1011
$\square \oplus x$	$\mapsto$	$011\top$
$(\square \oplus x) \gg 0001_2$	$\mapsto$	0011

Components

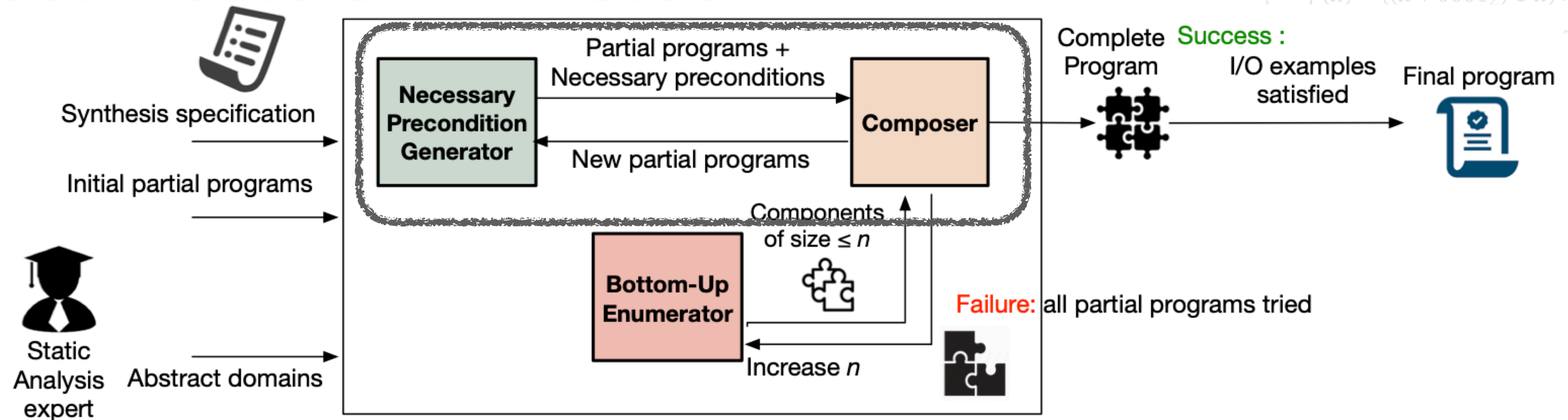
$x (= 1011_2)$
 0001_2

Spec

$S \rightarrow x | 0001_2$
 $| S \wedge S | S \vee S | S \oplus S$
 $| S + S | S \times S | S / S | S \gg S$
 $f(1011_2) = 0011_2$

Goal

$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$



Synthesis Algorithm

Partial Programs

$(\square \oplus x) \gg 0001_2$

→

$\square$	↦	110T
x	↦	1011
$\square \oplus x$	↦	011T
$(\square \oplus x) \gg 0001_2$	↦	0011

Components

$x (= 1011_2)$
 0001_2

No component
satisfies precondition
-> skip

Spec

$S \rightarrow x | 0001_2$
| $S \wedge S | S \vee S | S \oplus S$
| $S + S | S \times S | S / S | S \gg S$
 $f(1011_2) = 0011_2$

Goal

$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$

Synthesis Algorithm

Partial Programs

$(\square \oplus x) \gg 0001_2 \rightarrow$

$\square$	$\mapsto$	101_2
x	$\mapsto$	1011
$\square \oplus x$	$\mapsto$	011_2
$(\square \oplus x) \gg 0001_2$	$\mapsto$	0011

Components

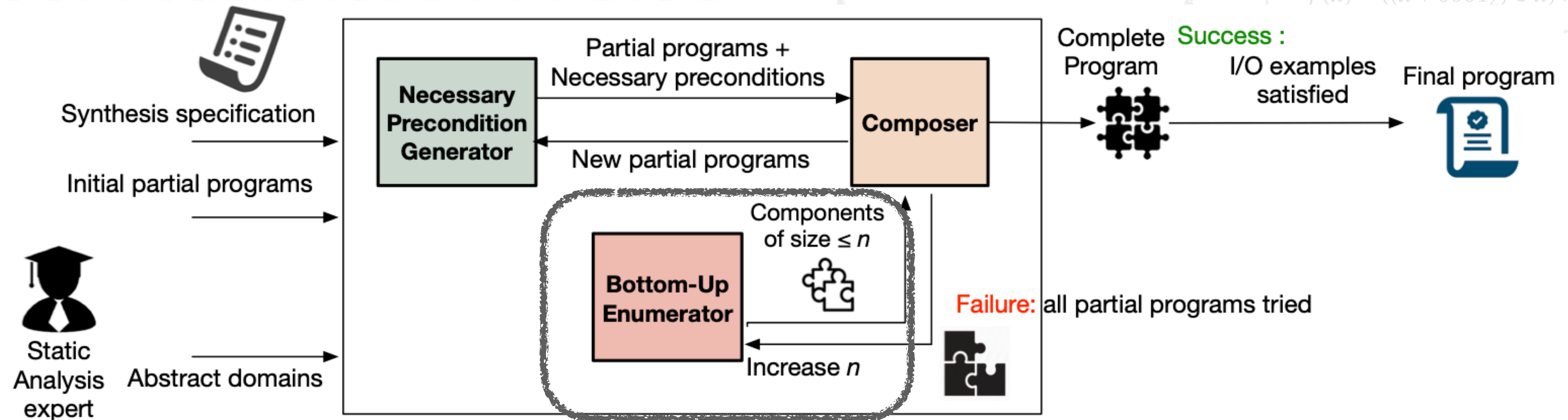
$x (= 1011_2)$
 0001_2
 $x + 0001_2 (= 1100_2)$
 $x + x (= 0110_2)$
 $x \oplus 0001_2 (= 1010_2)$

Spec

$S \rightarrow x | 0001_2$
 $| S \wedge S | S \vee S | S \oplus S$
 $| S + S | S \times S | S / S | S \gg S$
 $f(1011_2) = 0011_2$

Goal

$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$



Synthesis Algorithm

Partial Programs

$$(\square \oplus x) \gg 0001_2 \rightarrow \begin{array}{ll} \square & \mapsto 110\top \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto 011\top \\ (\square \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Components

$$\begin{array}{l} x (= 1011_2) \\ 0001_2 \\ x + 0001_2 (= 1100_2) \\ x + x (= 0110_2) \\ x \oplus 0001_2 (= 1010_2) \\ \dots \end{array}$$

Spec

$$\begin{array}{l} S \rightarrow x | 0001_2 \\ | S \wedge S | S \vee S | S \oplus S \\ | S + S | S \times S | S / S | S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Size increased

Synthesis Algorithm

Partial Programs

$$(\square \oplus x) \gg 0001_2 \rightarrow$$

$\square$	$\mapsto$	$110\top$
x	$\mapsto$	1011
$\square \oplus x$	$\mapsto$	$011\top$
$(\square \oplus x) \gg 0001_2$	$\mapsto$	0011

Components

$$x (= \boxed{1011_2})$$

$$\boxed{0001_2}$$

$$x + 0001_2 (= 1100_2)$$

$$x + x (= 0110_2)$$

$$x \oplus 0001_2 (= 1010_2)$$

$$\dots$$

Spec

$$S \rightarrow x | 0001_2$$

$$| S \wedge S | S \vee S | S \oplus S$$

$$| S + S | S \times S | S / S | S \gg S$$

$$f(1011_2) = 0011_2$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

~~$$x / 1 (= 1011_2)$$

$$x \wedge x (= 1011_2)$$

$$x \wedge 0001_2 (= 1011_2)$$

$$x / x (= 0001_2)$$

$$0001_2 \vee 0001_2 (= 0001_2)$$~~

Ignore
Observational Equivalent
Components

Synthesis Algorithm

Partial Programs

$$(\square \oplus x) \gg 0001_2$$

→

$\square$	$\mapsto$	$110\top$
x	$\mapsto$	1011
$\square \oplus x$	$\mapsto$	$011\top$
$(\square \oplus x) \gg 0001_2$	$\mapsto$	0011

Components

$$\begin{aligned} &x (= 1011_2) \\ &0001_2 \\ &x + 0001_2 (= 1100_2) \\ &x + x (= 0110_2) \\ &x \oplus 0001_2 (= 1010_2) \\ &\dots \end{aligned}$$

Spec

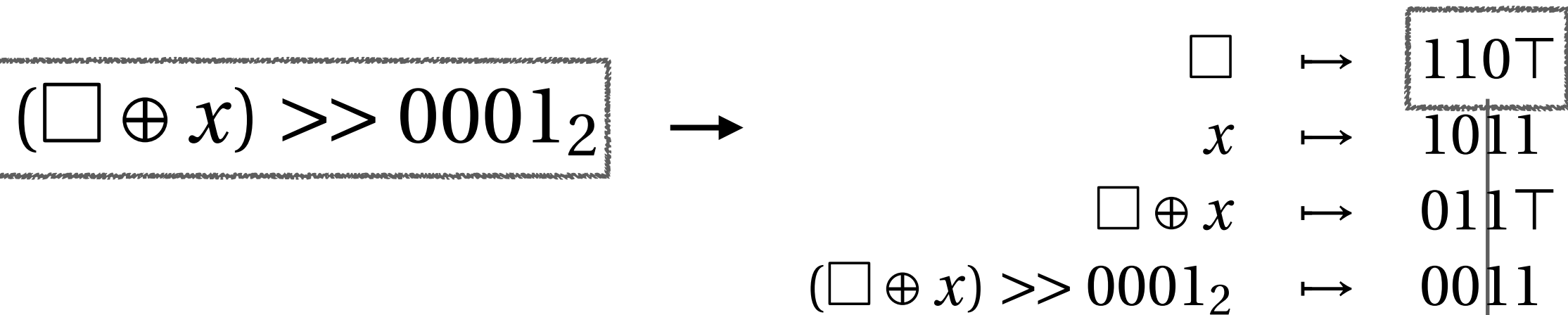
$$\begin{aligned} S &\rightarrow x | 0001_2 \\ &| S \wedge S | S \vee S | S \oplus S \\ &| S + S | S \times S | S / S | S \gg S \\ f(1011_2) &= 0011_2 \end{aligned}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Synthesis Algorithm

Partial Programs



Components

$x (= 1011_2)$
 0001_2
 $x + 0001_2 (= 1100_2)$
 $x + x (= 0110_2)$
 $x \oplus 0001_2 (= 1010_2)$
...

Spec

$S \rightarrow x | 0001_2$
 $| S \wedge S | S \vee S | S \oplus S$
 $| S + S | S \times S | S / S | S \gg S$
 $f(1011_2) = 0011_2$

Goal

$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$

$\{1100, 1101\} \rightarrow$

Lookup Table

0001	:	0001_2
0110	:	$x + x$
1010	:	$x \oplus 0001_2$
1011	:	x
1100	:	$x + 0001_2$

Synthesis Algorithm

Partial Programs

$$\boxed{(\square \oplus x) \gg 0001_2} \rightarrow \begin{array}{ll} \square & \mapsto 110\top \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto 011\top \\ (\square \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Components

$$\begin{array}{l} x (= 1011_2) \\ 0001_2 \\ x + 0001_2 (= 1100_2) \\ x + x (= 0110_2) \\ x \oplus 0001_2 (= 1010_2) \\ \dots \end{array}$$

Spec

$$\begin{array}{l} S \rightarrow x | 0001_2 \\ \quad | S \wedge S | S \vee S | S \oplus S \\ \quad | S + S | S \times S | S / S | S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

$\{1100, 1101\}$

$$110\top \sqsupseteq 1100$$

Lookup Table

0001	:	0001 ₂
0110	:	$x + x$
1010	:	$x \oplus 0001_2$
1011	:	x
1100	:	$x + 0001_2$

Synthesis Algorithm

Partial Programs

$$(\square \oplus x) \gg 0001_2 \rightarrow \begin{array}{ll} \square & \mapsto 110\top \\ x & \mapsto 1011 \\ \square \oplus x & \mapsto 011\top \\ (\square \oplus x) \gg 0001_2 & \mapsto 0011 \end{array}$$

Components

$$\begin{array}{l} x (= 1011_2) \\ 0001_2 \\ x + 0001_2 (= 1100_2) \\ x + x (= 0110_2) \\ x \oplus 0001_2 (= 1010_2) \\ \dots \end{array}$$

Spec

$$\begin{array}{l} S \rightarrow x | 0001_2 \\ \quad | S \wedge S | S \vee S | S \oplus S \\ \quad | S + S | S \times S | S / S | S \gg S \\ f(1011_2) = 0011_2 \end{array}$$

Goal

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

New Candidate Solution

$$f(x) = ((x + 0001_2) \oplus x) \gg 0001_2$$

Solution Found!

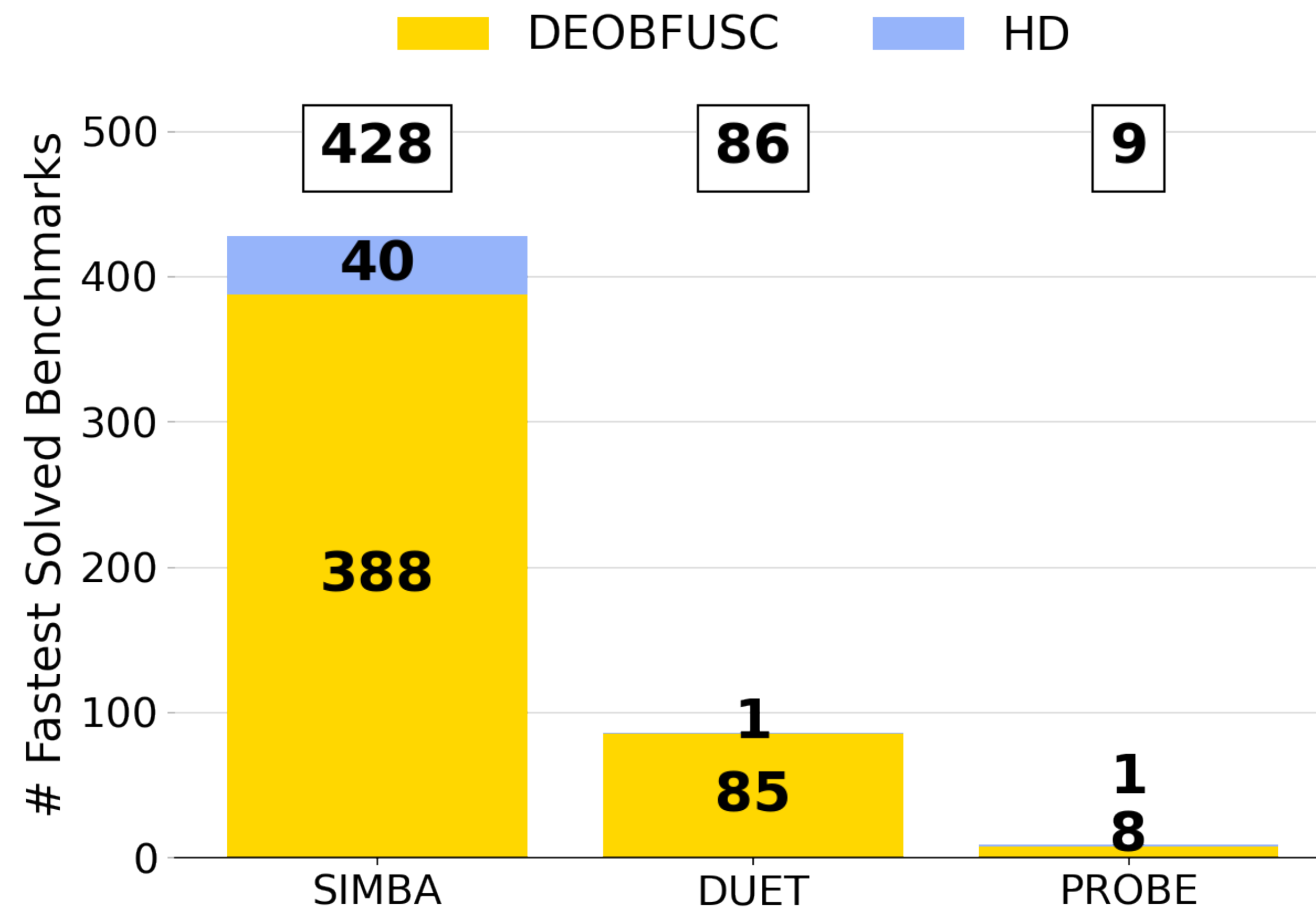
$$\begin{array}{lcl} f(1011_2) & = & ((1011_2 + 0001_2) \oplus 1011_2) \gg 0001_2 \\ & = & (1100_2 \oplus 1011_2) \gg 0001_2 \\ & = & 0111_2 \gg 0001_2 \\ & = & 0011_2 \end{array}$$

Evaluation

- Benchmark: 544 synthesis tasks from 2 sources
 - **HD**: 44 from hacker's delight
 - **Deobfsc**: 500 deobfuscation tasks from QSynth [David et al. 2020]
- Baseline tools
 - **Duet**: Woosuk Lee, “Combining the Top-Down Propagation and Bottom-Up Enumeration for Inductive Program Synthesis”, POPL’21
 - **Probe**: Barke et al., Just-in-Time Learning for Bottom-Up Enumerative Synthesis, OOPSLA’20

Results

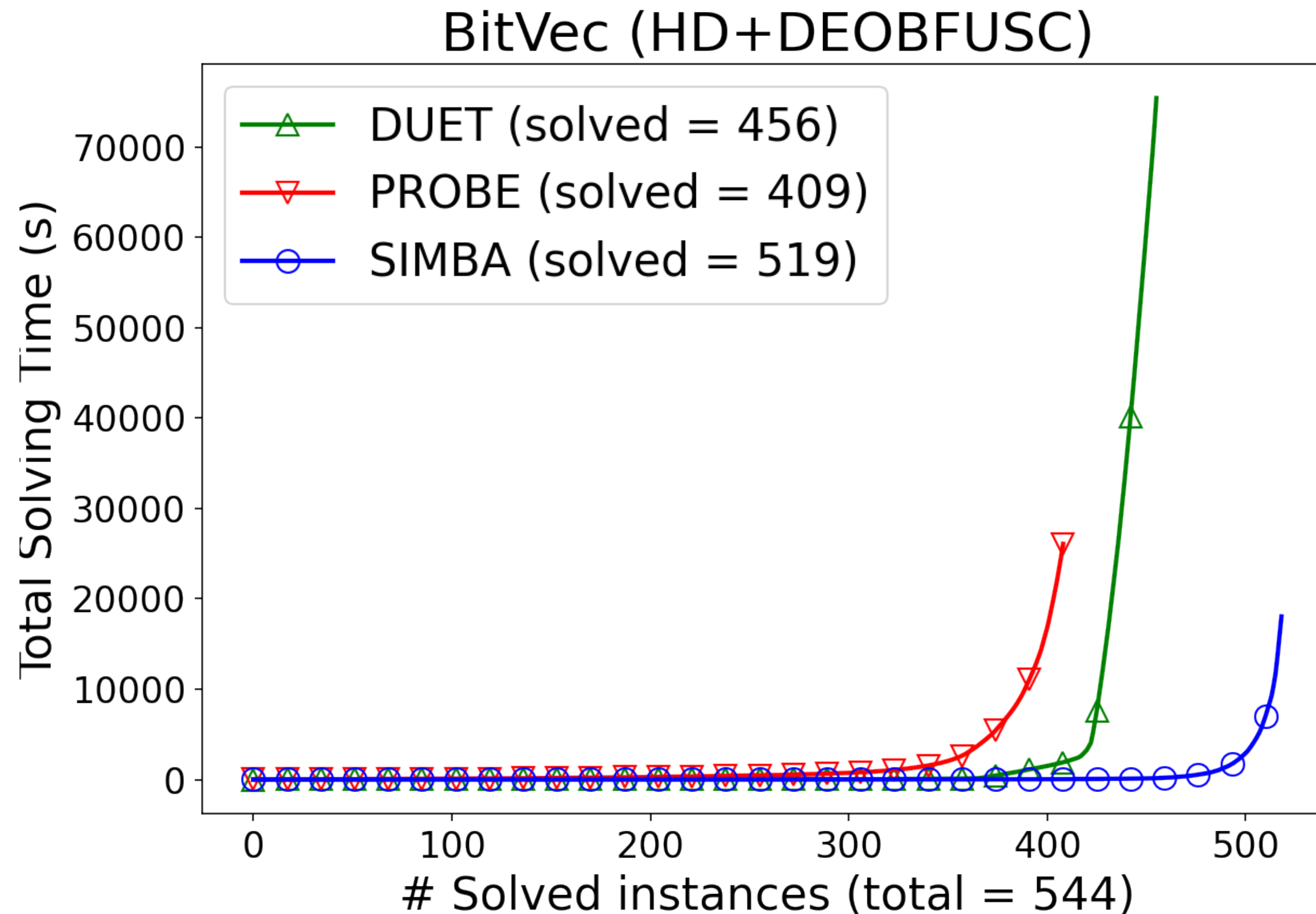
- Significantly outperforms the baseline tools



of the fastest-solved problems for each domain

Results

- SIMBA(plot ○) shows the best performance



Room for Improvement

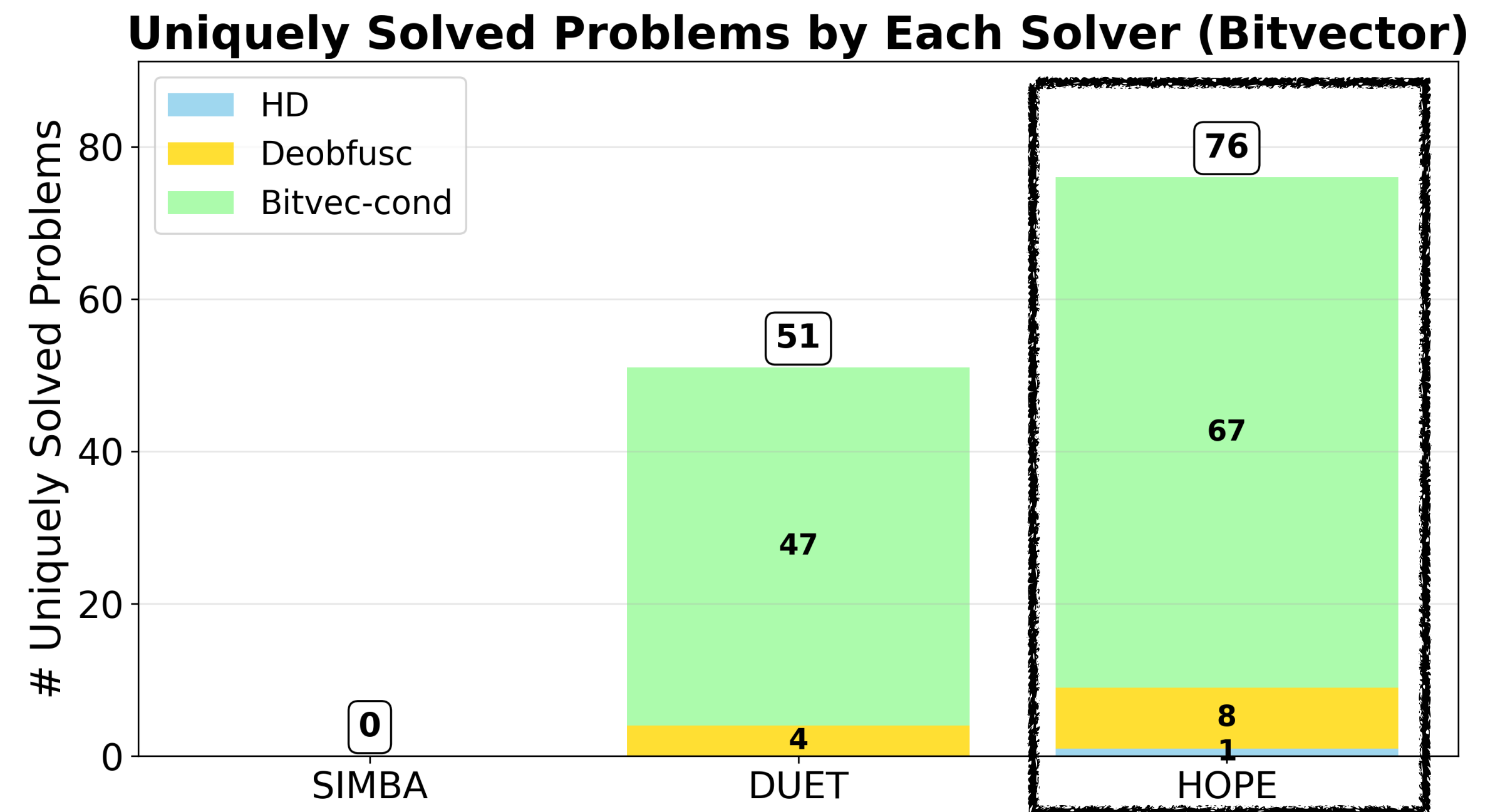
- The order in which holes are filled during the search can drastically affect search efficiency (but largely unattained).
- E.g., synthesizing a program with I/O example $\{x=4, y=1\} \rightarrow 9$
the solution is $(x - y) * 3$
- Suppose we have a partial program $(\square - \square) * \square$
- check the feasibility by performing the parity analysis (even/odd)

Room for Improvement

- If we always fill the leftmost hole first, (e.g., $(\times - \square) * \square$)
 - the parity analysis cannot determine the feasibility, since the rightmost hole (which is yet to be filled) can be either odd or even
- If we fill the rightmost hole first, (e.g., $(\square - \square) * 2$)
 - the parity analysis can determine it is infeasible ($\square * 2$ cannot be the desired output 9 (odd)), allowing to prune the search space early

Improvement

- We devise a static analysis that identifies the best hole-filling order (rightmost first in the previous example).
- By following the best hole-filling order, we can outperform SOTA
- Conditionally accepted to POPL'26
- We are integrating it into ProMBA.



Conclusion

- In MBA deobfuscation,
- By using advanced search algorithms
 - Search-based program synthesis and equality saturation
- We can achieve better performance
 - By discovering new simplification rules
 - and sophisticated rule application orders that yield (nearly) optimal results