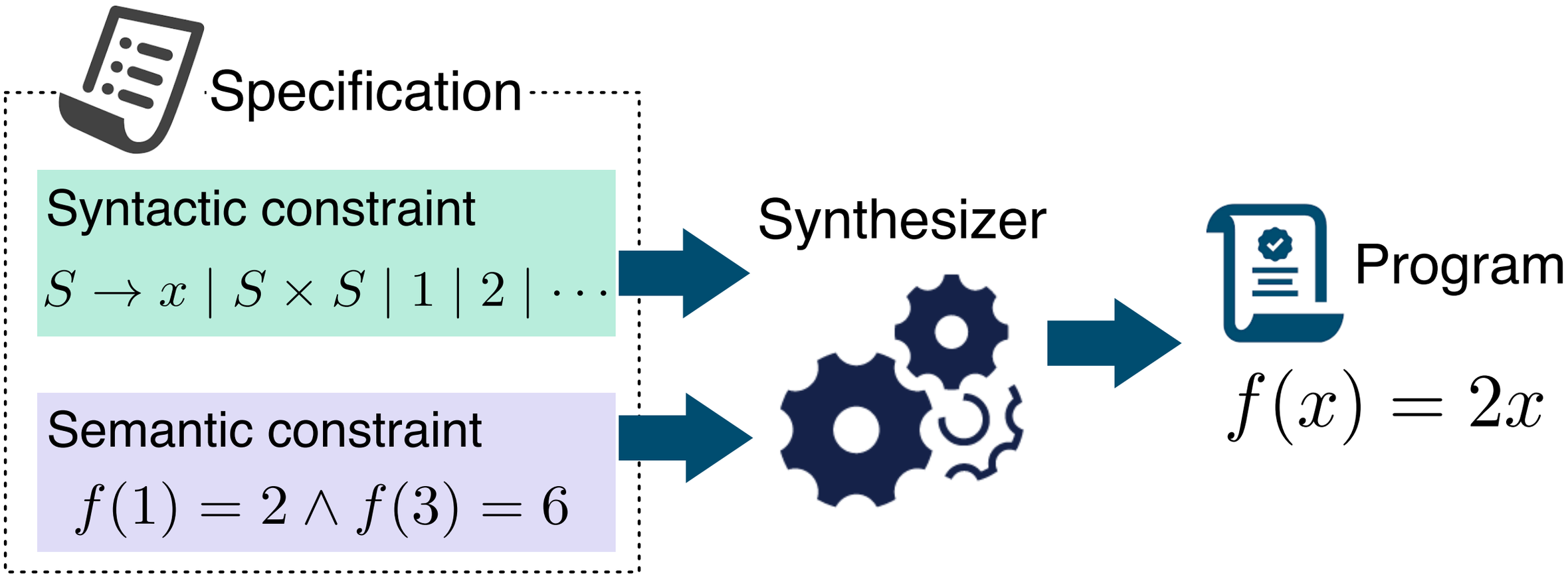
Accelerating Search-Based Synthesis Using Learned Probabilistic Models

Woosuk Lee Kihong Heo Rajeev Alur Mayur Naik

University of Pennsylvania

Syntax-Guided Program Synthesis (SyGuS)



Existing General-Purpose Strategies

- **Enumerative:** search with pruning
  - EUSolver: Udupa et al. (PLDI'13)
- **Symbolic:** constraint solving
  - CVC4: Reynolds et al. (CAV'15)
- **Stochastic:** probabilistic walk
  - STOKE: Schkufza et al. (ASPLOS'13)

Existing General-Purpose Strategies

- **Enumerative:** search with pruning
  - EUSolver: Udupa et al. (PLDI'13)
- **Symbolic:** constraint solving
  - CVC4: Reynolds et al. (CAV'15)
- **Stochastic:** probabilistic walk
  - STOKE: Schkufza et al. (ASPLOS'13)

Key limitation:
search not guided towards ***likely*** programs

Statistical Regularities in Programs

- Programs contain repetitive and predictable patterns.

```
for (i = 0; i < 100; ??)
```

- Statistical program models define a probability distribution over programs.

$Pr($`??` → `i++ | for (i = 0; i < 100; ??)`$) = 0.80$

$Pr($`??` → `i-- | for (i = 0; i < 100; ??)`$) = 0.01$

  - e.g., n-gram, probabilistic context-free grammar (PCFG), …

- Applications: code completion, deobfuscation, program repair…

Exploiting Statistical Regularities

Can we leverage statistical program models
to accelerate program synthesis?

Key Challenges:

1. How to guide the search given a statistical model?

2. How to learn a good statistical model?

Our Contributions

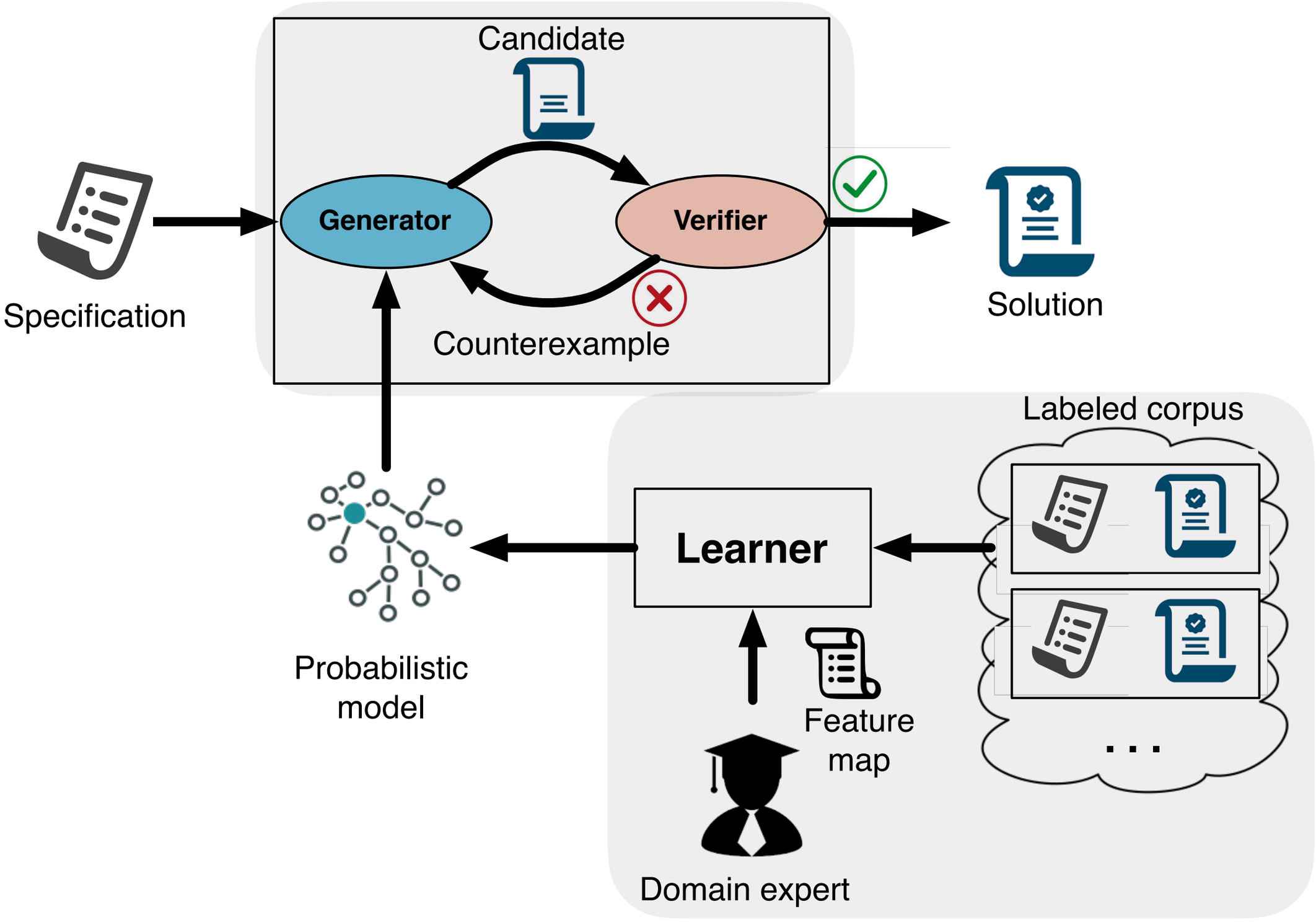
- A general approach to accelerate *CEGIS*-based program synthesis
  - by using a probabilistic model to guide the search
  - supports a wide range of models (e.g., *n*-gram, PCFG, PHOG, …)
- Transfer learning-based method to mitigate overfitting
- Tool (Euphony) and evaluation on widely applicable domains

https://github.com/wslee/euphony

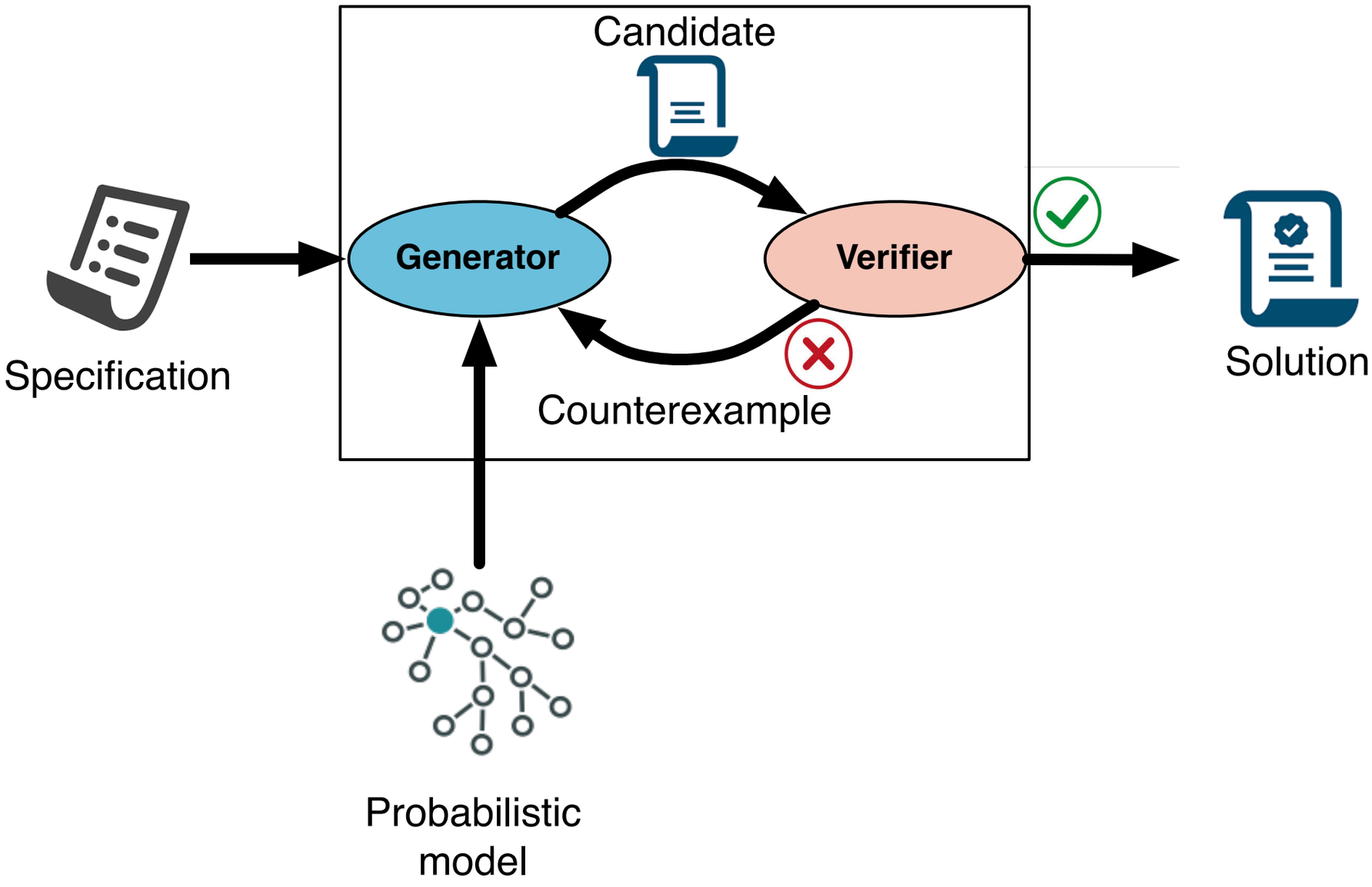
Talk Outline

- Overall Architecture
- Illustrative Example
- Empirical Evaluation

Overall Architecture

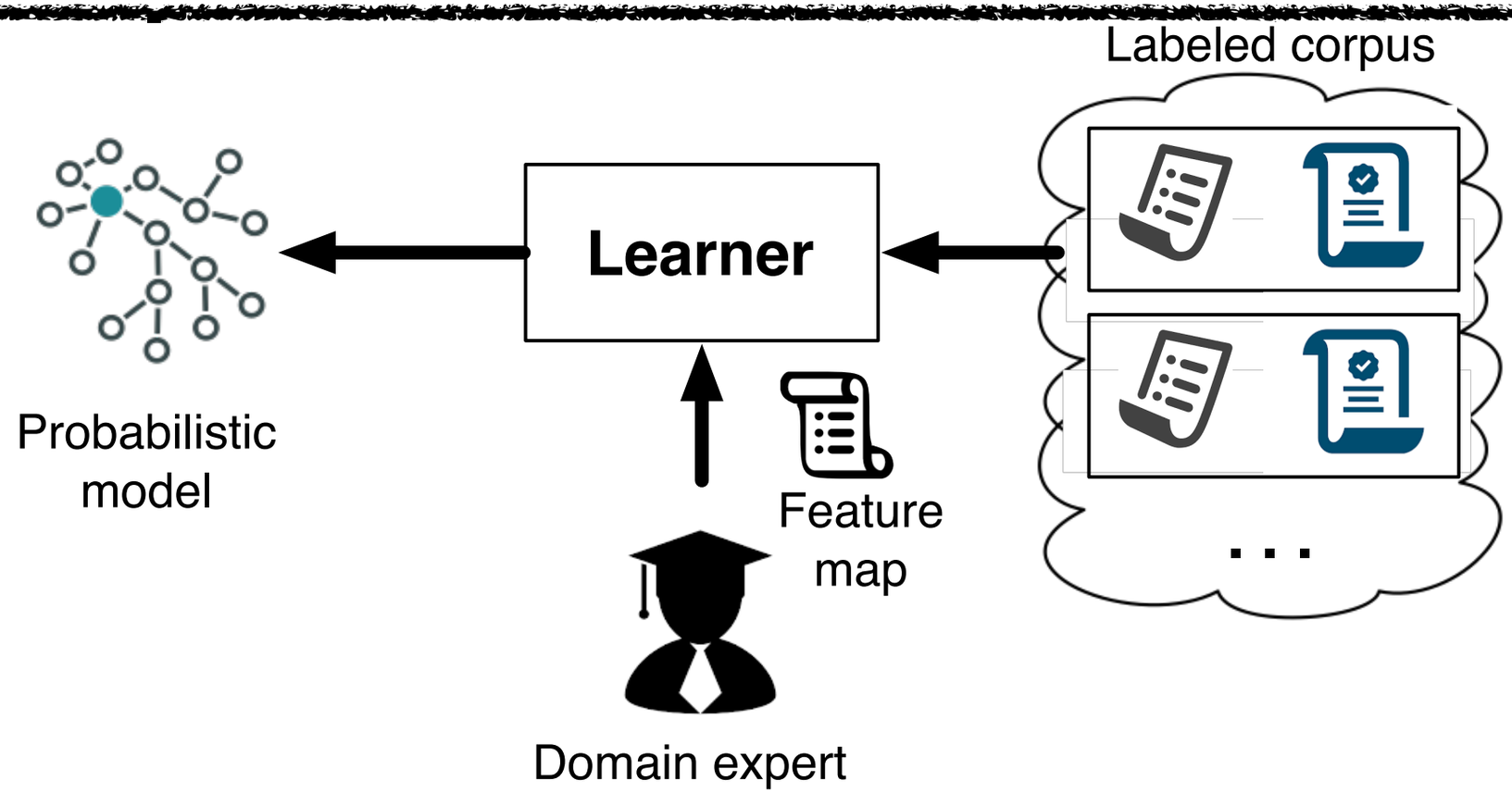


CEGIS with Guided Search



Transfer Learning

- Problem: **overfitting**
- Our solution: generalize to unseen programs better using a **feature map** designed by domain expert



Talk Outline

- Overall Architecture
- Illustrative Example
- Empirical Evaluation

Example

- **Goal:** Replacing a hyphen (-) by a dot (.) in a given string x
- **Specification**

Syntactic specification:

String concatenation

$$S \rightarrow x \mid \text{“-”} \mid \text{“.”} \mid S + S \mid \text{Rep}(S, S, S)$$

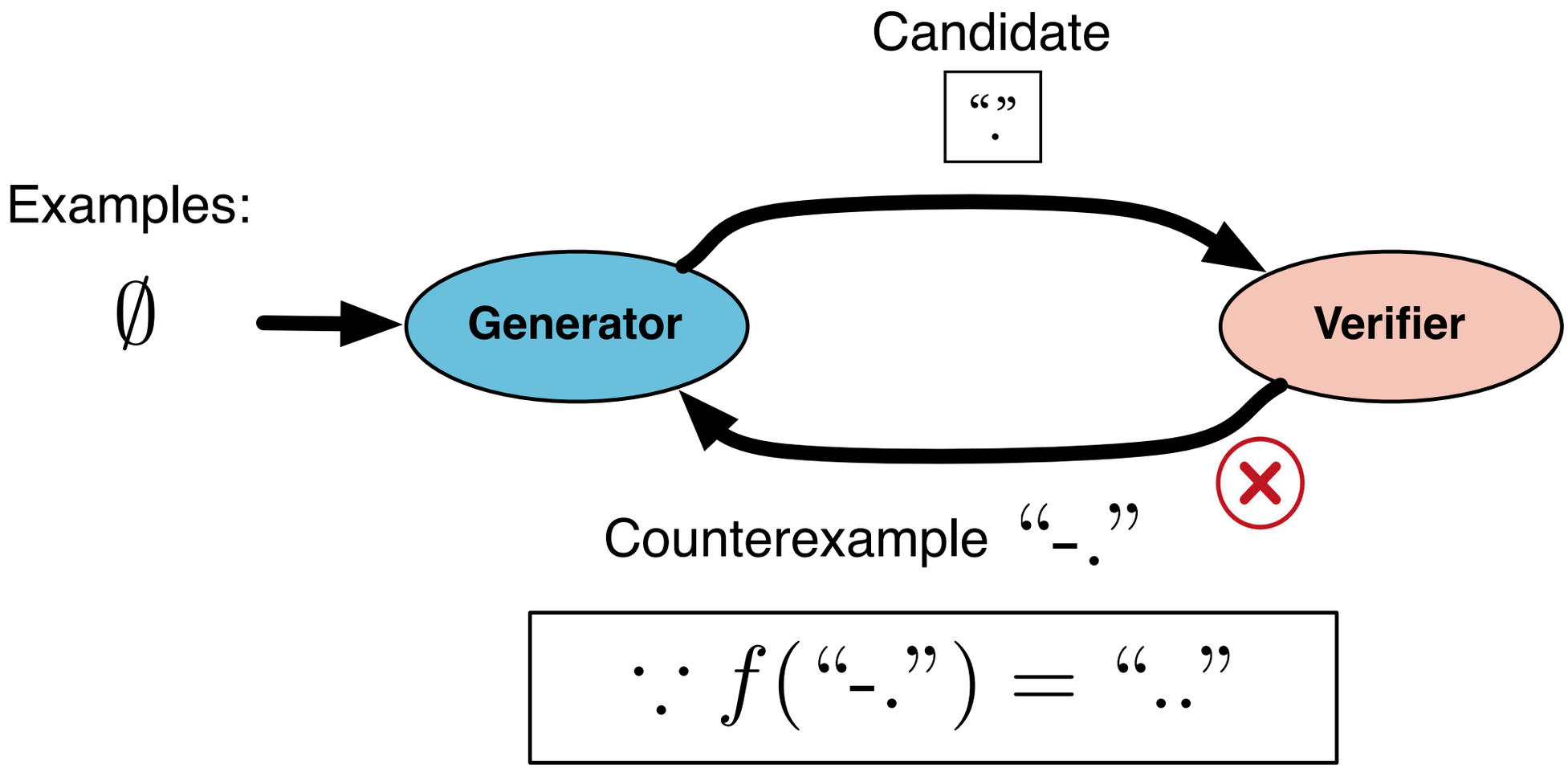
Rep(s, t1, t2): t1 in **s** is replaced by **t2**

Semantic specification:

$$f(\text{“-.”}) = \text{“..”} \wedge f(\text{“308-916”}) = \text{“308.916”} \wedge f(\text{“1”}) = \text{“1”}$$

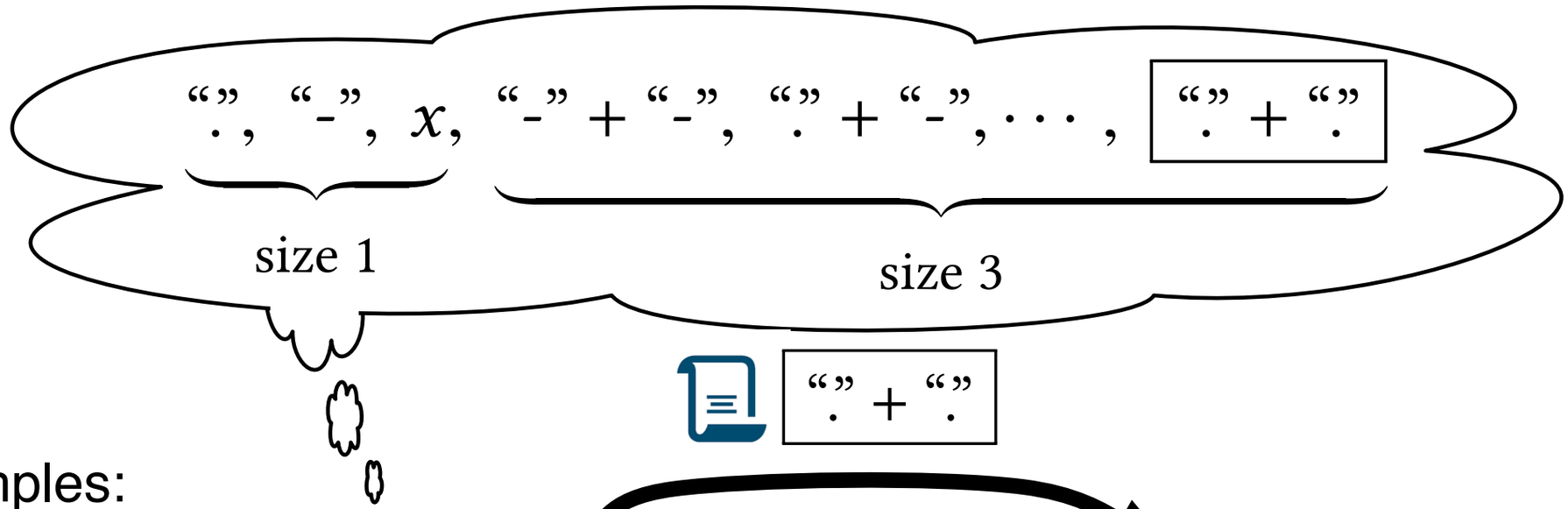
- **Solution:** $\text{Rep}(x, \text{“-”}, \text{“.”})$.

Enumerative Search: Unguided



$$\because f(\text{“–.”}) = \text{“..”}$$

Enumerative Search: Unguided

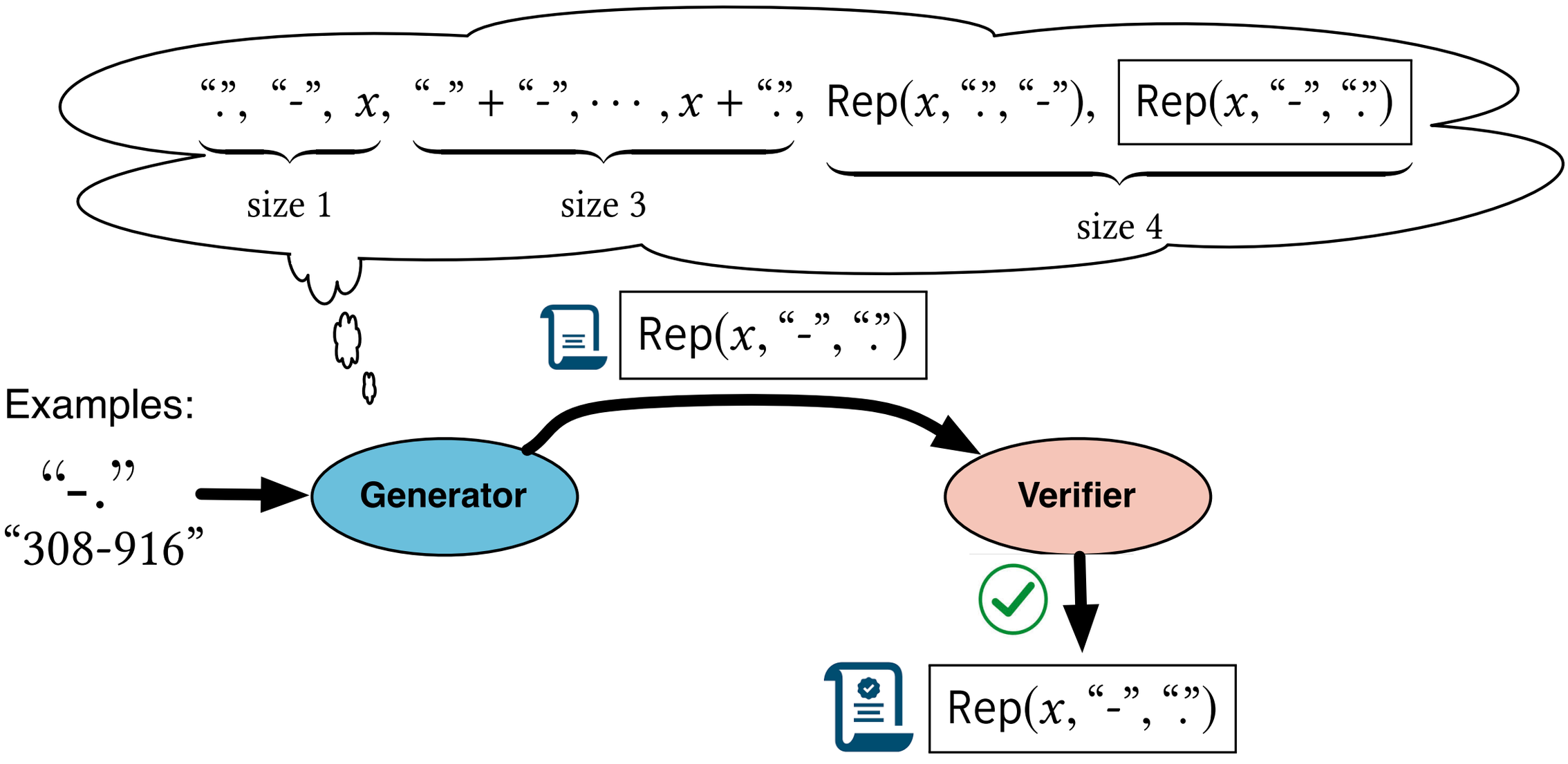


Counterexample “308-916”

$$\because f(\text{“308-916”}) = \text{“308.916”}$$

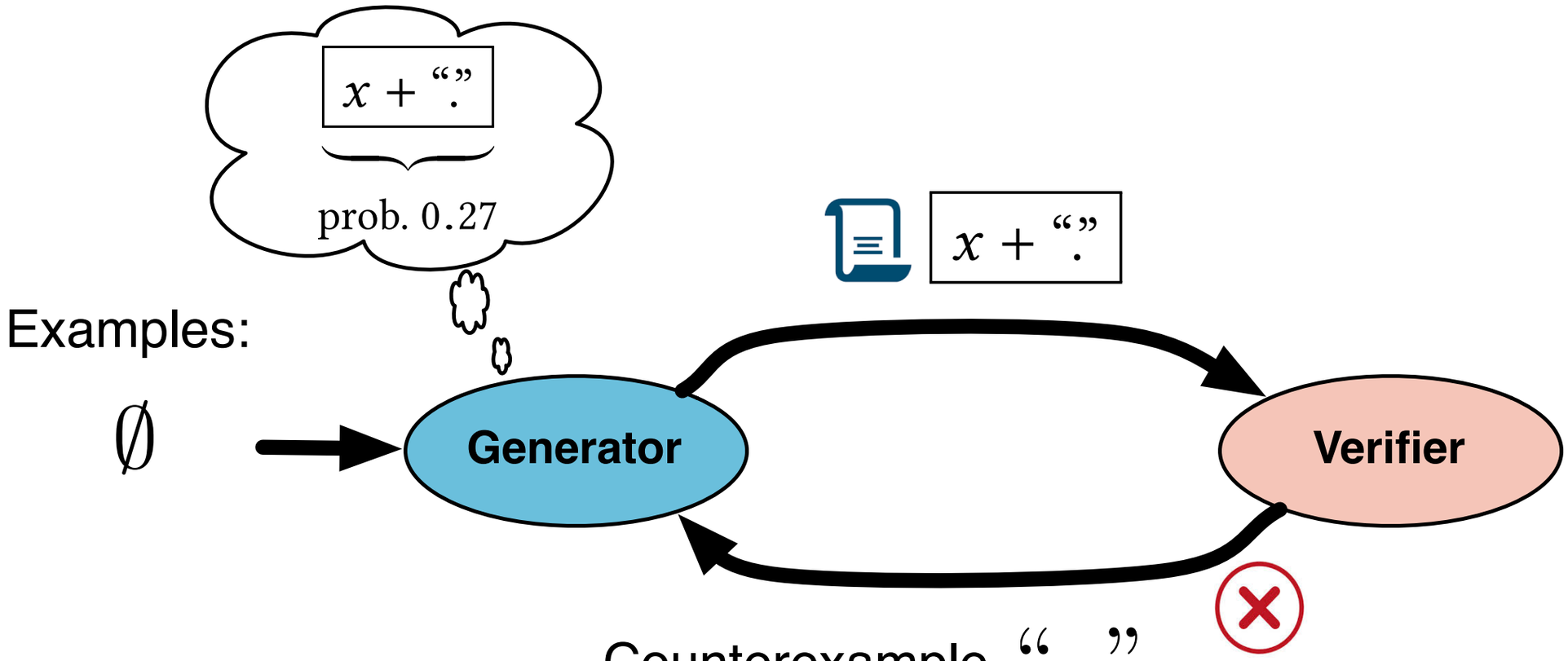
Pruning optimization: “–” + “.” is not explored.

Enumerative Search: Unguided



Many unlikely programs (e.g., “ .” + “ .”) are explored.

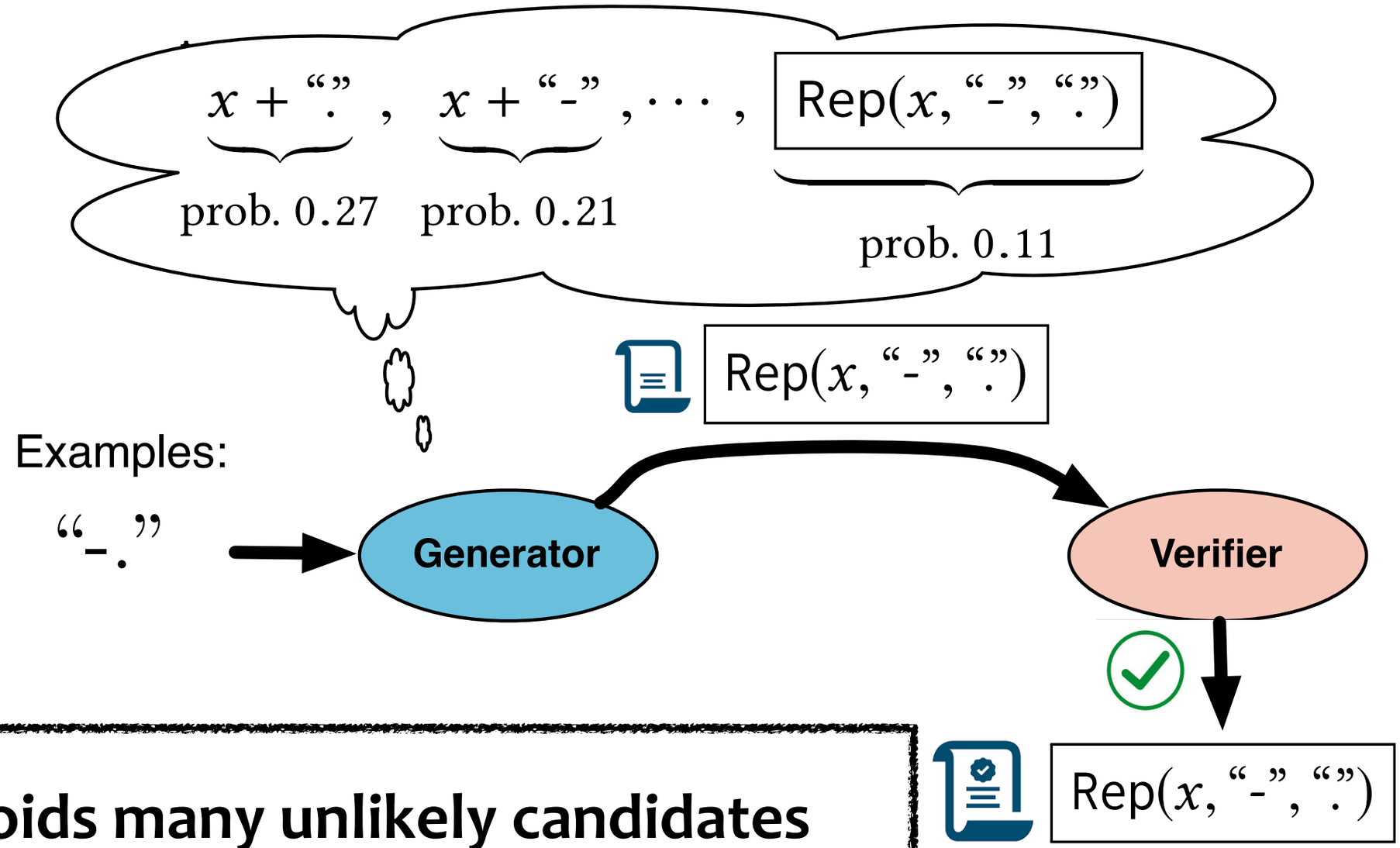
Enumerative Search: Guided



$$\because f(\text{“–.”}) = \text{“..”}$$

Enumerates in order of likelihood instead of size

Enumerative Search: Guided



- **Avoids many unlikely candidates**
- **Preserves the pruning optimization**

A Uniform Interface to Statistical Program Models

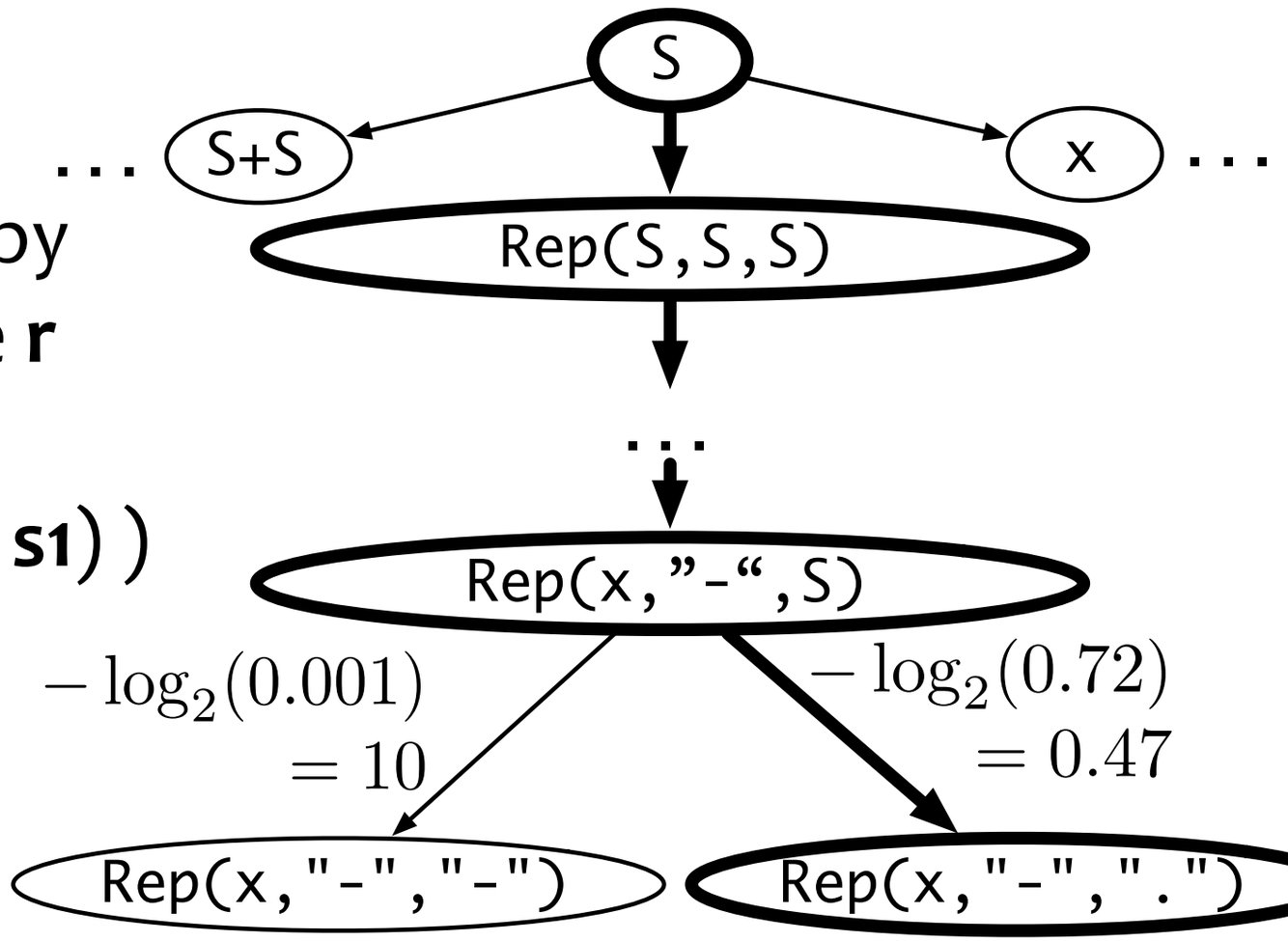
- Given a sequence of terminal/nonterminal symbols (i.e., sentential form), provide a probability for each production rule

$$Pr(S \rightarrow \text{“.”} \mid \texttt{Rep}(x, \text{“-”}, S)) = 0.72$$
$$Pr(S \rightarrow \text{“-”} \mid \texttt{Rep}(x, \text{“-”}, S)) = 0.001$$
$$\ldots$$

Guided Enumeration via Path Finding

Given a model, we construct a directed graph.

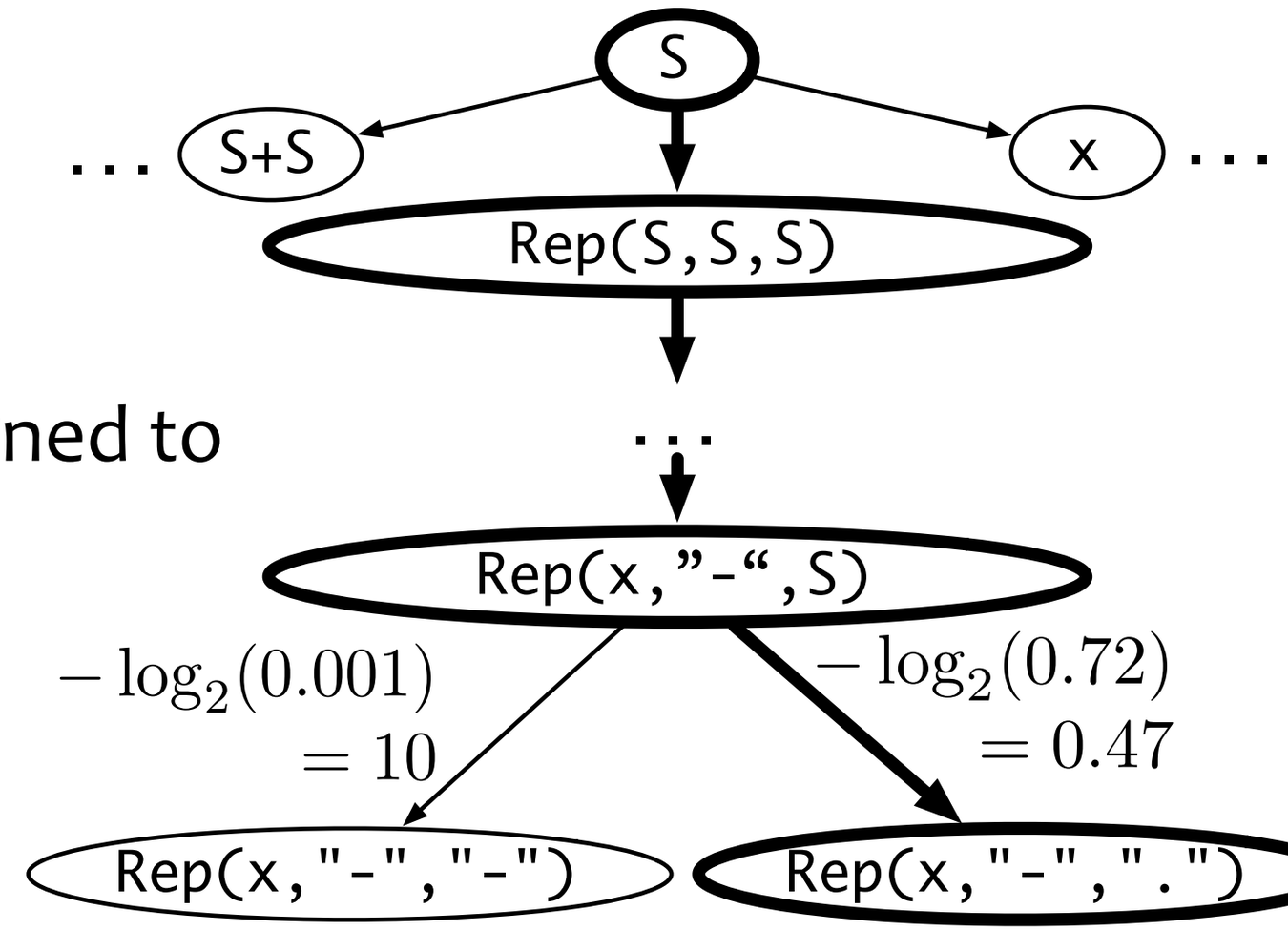
- Nodes: sentential forms
- **s1** $\xrightarrow{\mathbf{r}}$ **s2**: **s1** expands to **s2** by applying a production rule **r**
- w(**s1** $\xrightarrow{\mathbf{r}}$ **s2**) = − log (Pr (**r** | **s1**))



Guided Enumeration via Path Finding

Idea: solving a shortest pathfinding problem via A* search

- Start node: **S**
- Goal nodes: all programs
- A **heuristic function** designed to work with any model



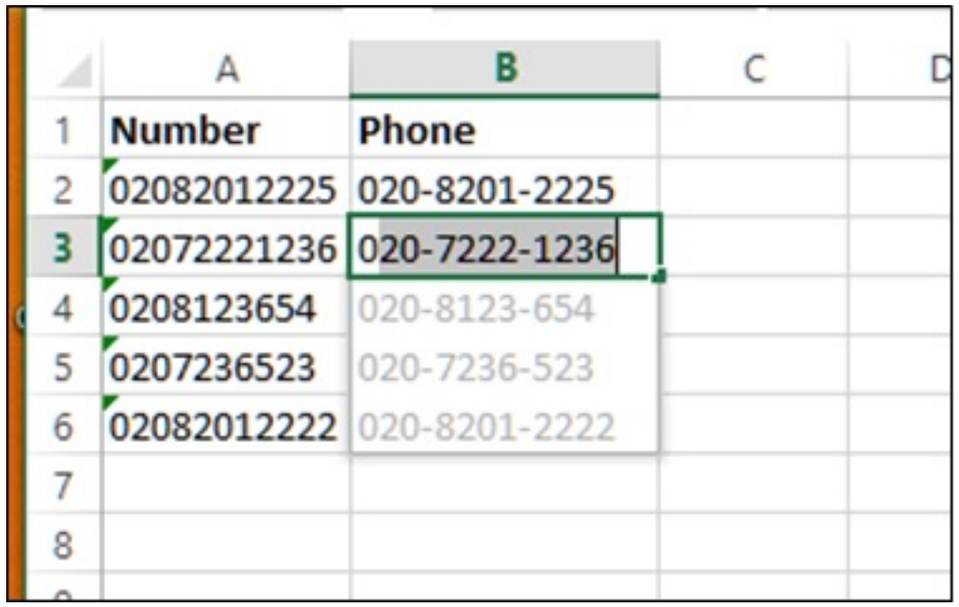
Talk Outline

- Overall Architecture
- Illustrative Example
- Empirical Evaluation

Evaluation Setup

- Benchmarks:
  - **1,167** problems from 2017 SyGuS competition and online forums
- Comparison to two baselines:
  - **EUSolver** (general-purpose): winner of 2017 SyGuS competition
  - **FlashFill** (domain-specific): string processing in spreadsheets

Benchmarks



	A	B	C	D
1	Number	Phone		
2	02082012225	020-8201-2225		
3	02072221236	020-7222-1236		
4	0208123654	020-8123-654		
5	0207236523	020-7236-523		
6	02082012222	020-8201-2222		
7				
8				

STRING: End-user Programming

205 problems

complement

```
~ 01010001110101110000000000001111
  10101110001010001111111111110000
```

bitwise and

```
  01010001110101110000000000001111
& 00110001011011100011000101101110
  00010001010001100000000000001110
```

bitwise or

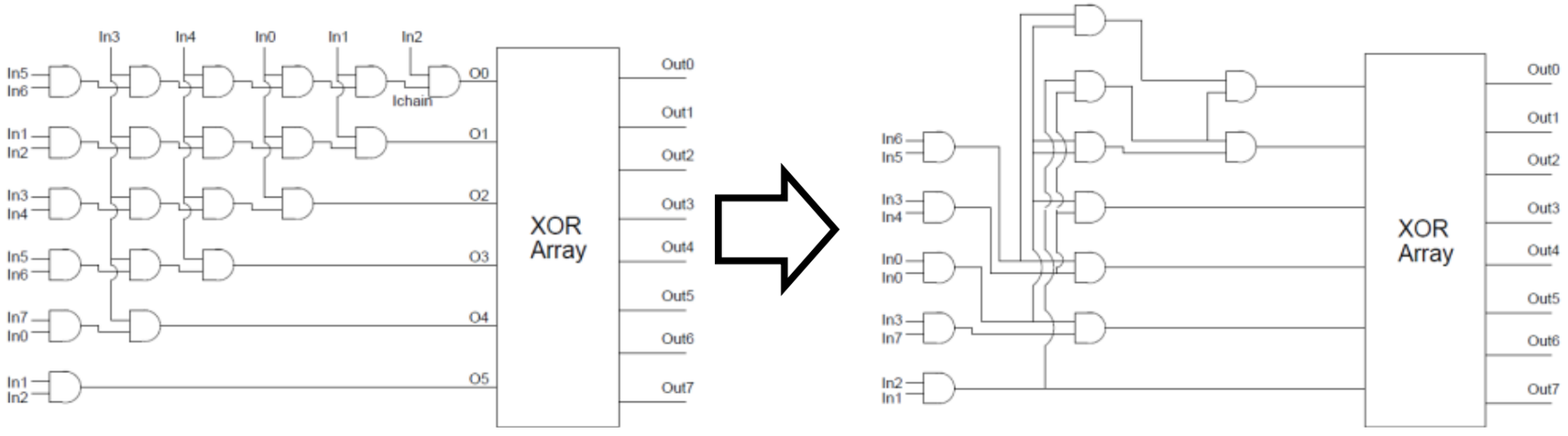
```
  01010001110101110000000000001111
| 00110001011011100011000101101110
  01110001111111110011000101101111
```

bitwise xor

```
  01010001110101110000000000001111
^ 00110001011011100011000101101110
  01100001011100100011000101100001
```

BITVEC: Efficient low-level algorithm

750 problems

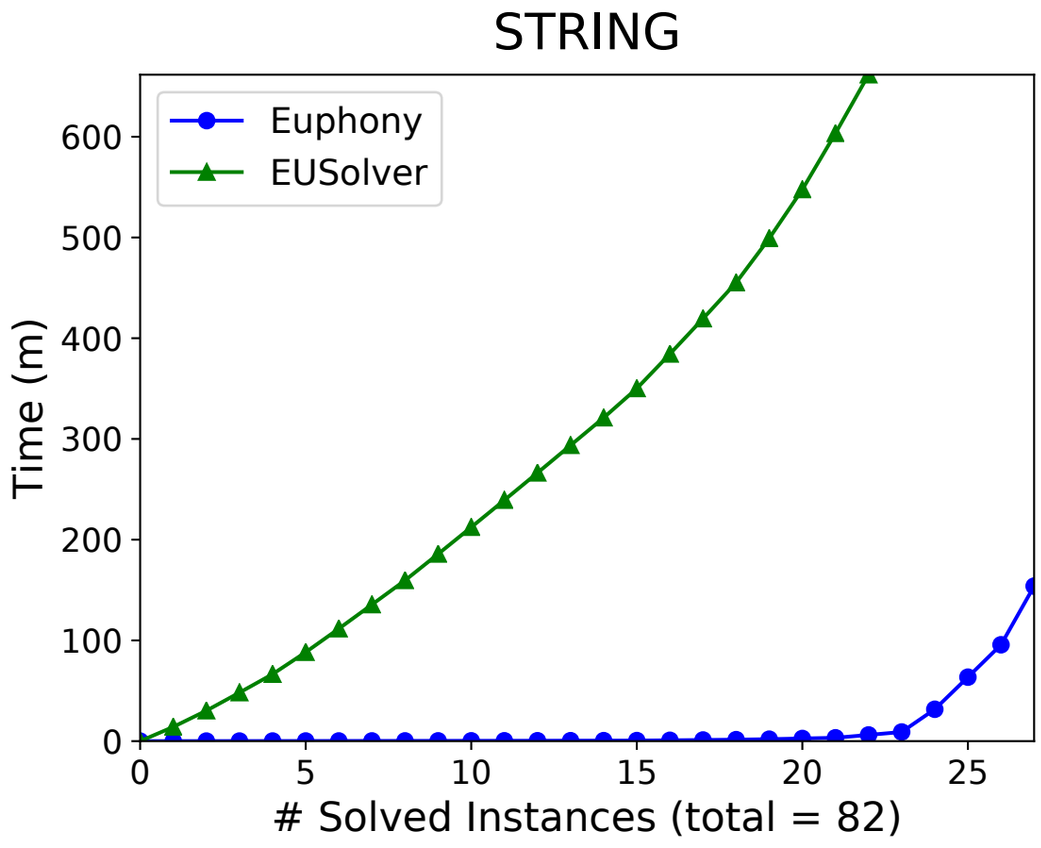


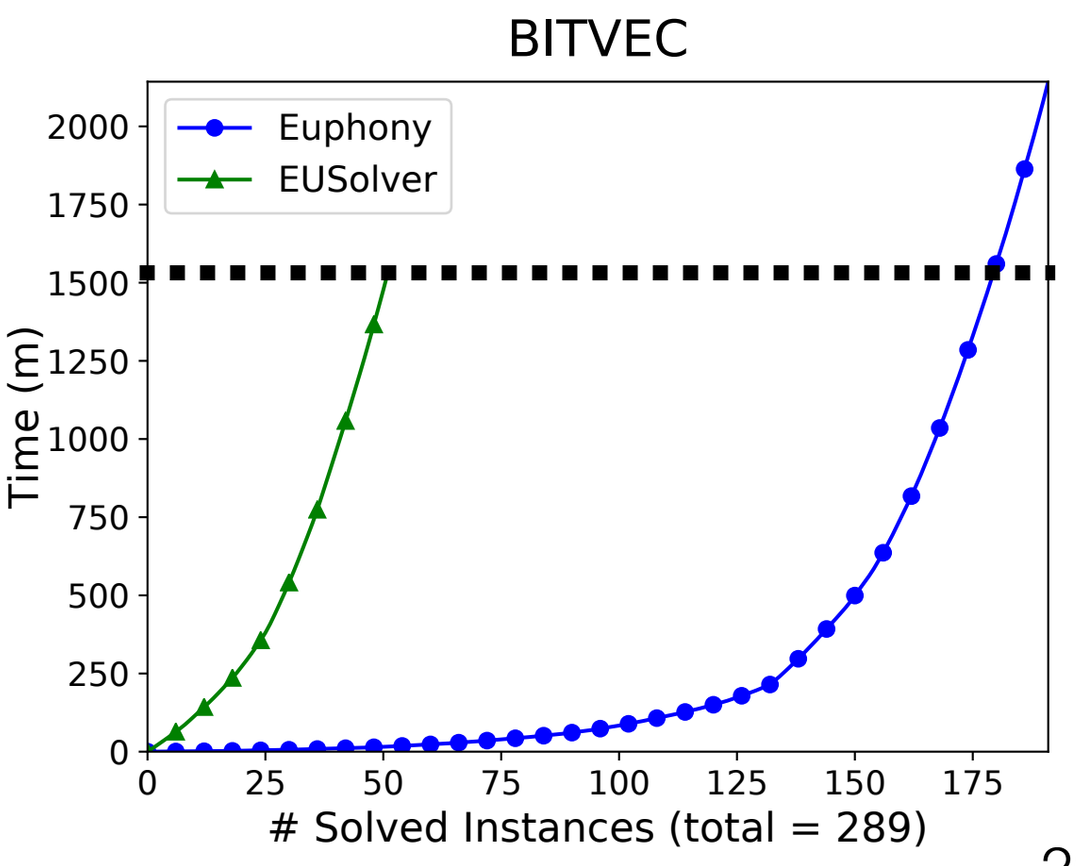
CIRCUIT: Attack-resistant crypto circuits

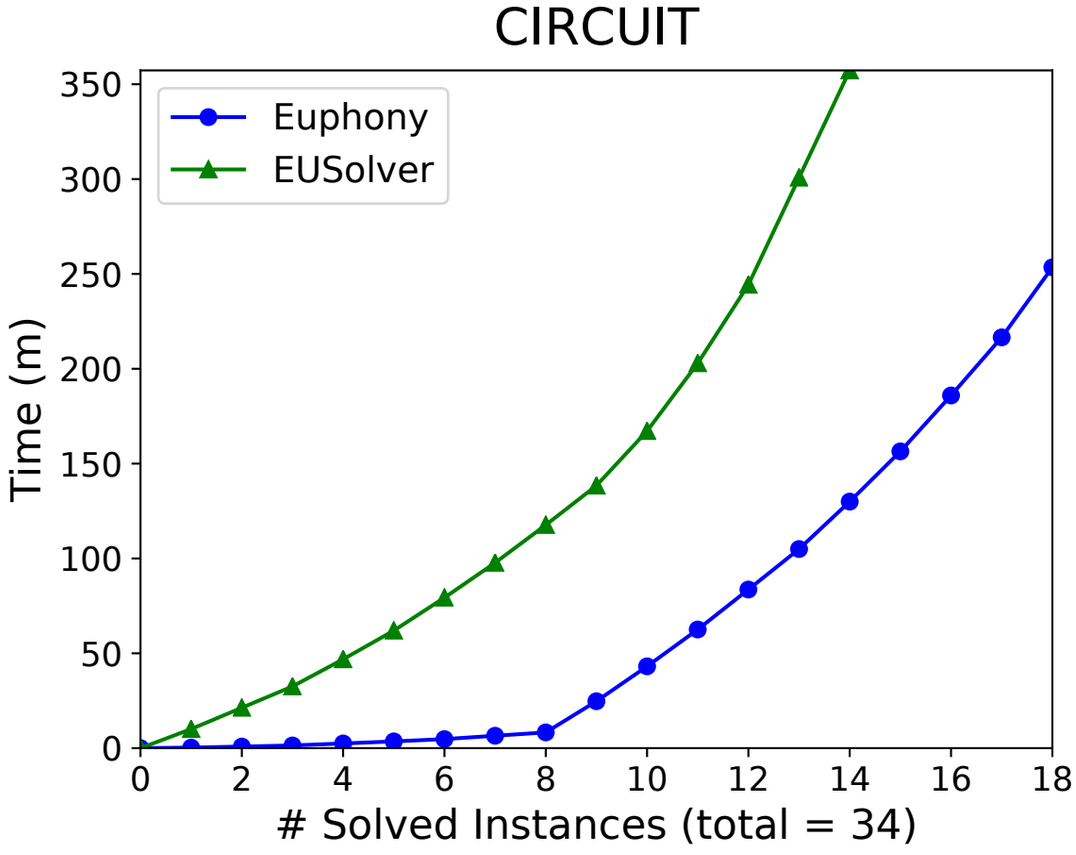
212 problems

Comparison with EUSolver

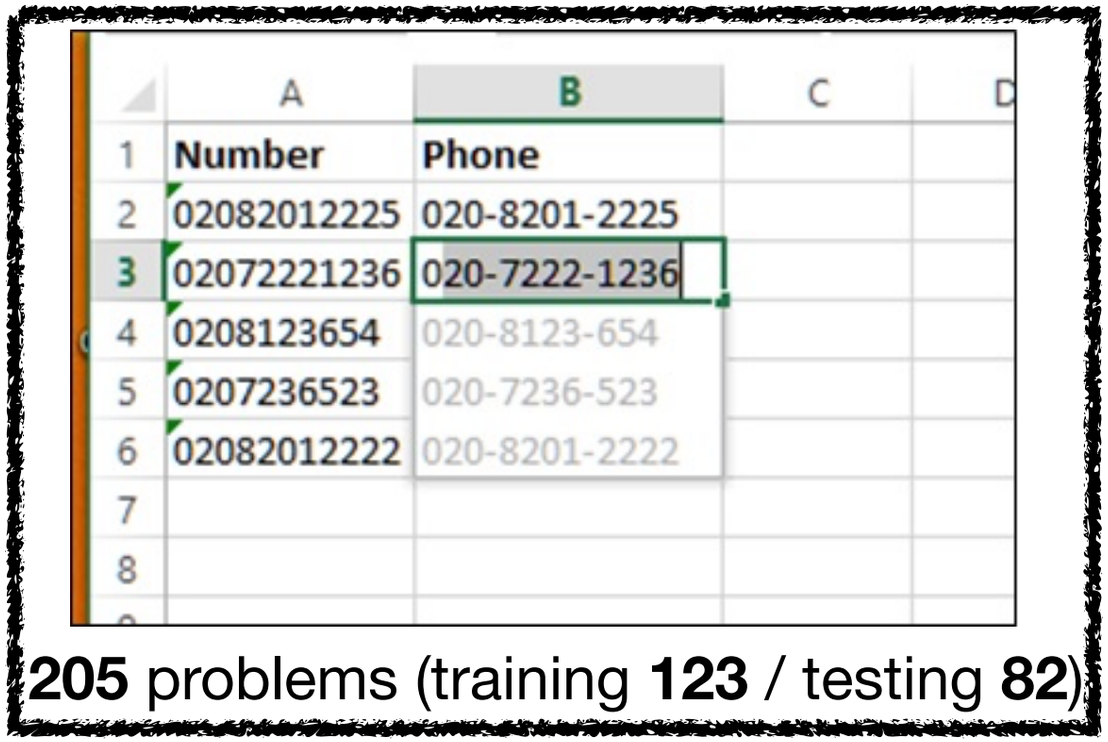
- Training: **762** solved by EUSolver in 10 m
- Testing: **405** (timeout: 1 hour)
- # solved: Euphony **236**, EUSolver **87**





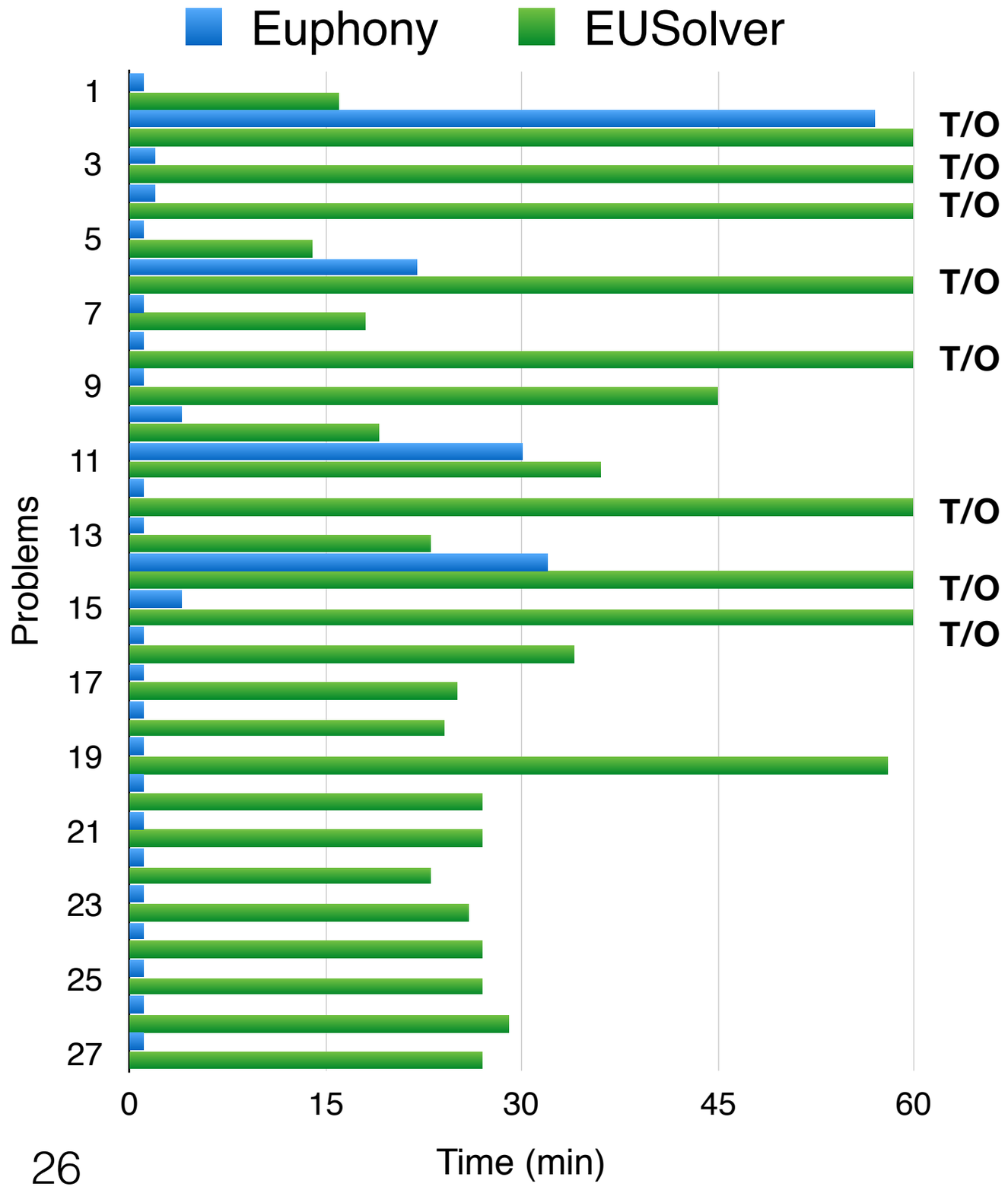


Result for STRING benchmarks



205 problems (training **123** / testing **82**)

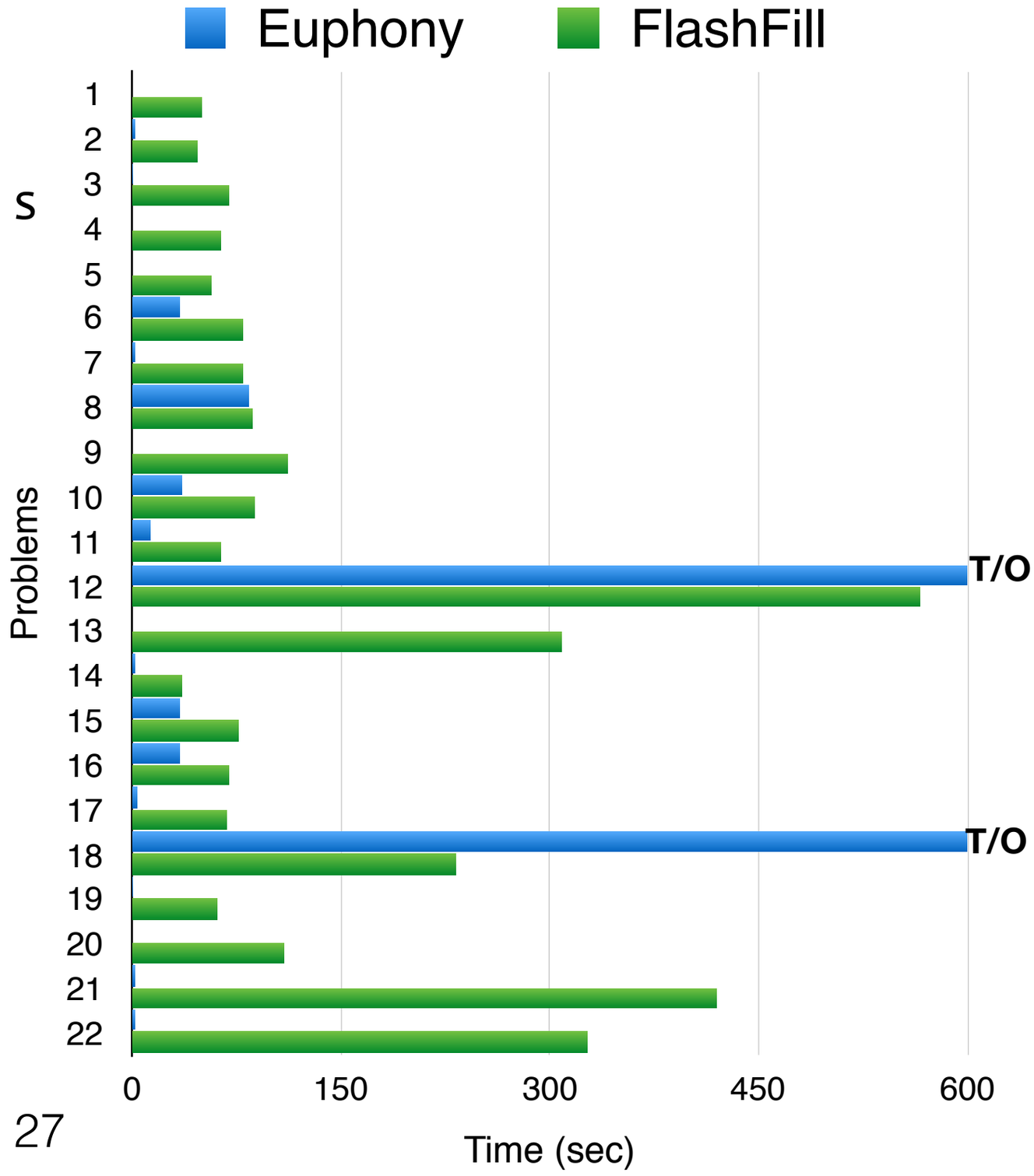
- Euphony solved **78**% within 1 min
- solved **8** on which EUSolver timed out
- outperformed EUSolver on all



Comparison with FlashFill (STRING)

- **113** problems handled by FlashFill
- Training: **91** solved by FlashFill in 10 s
- Testing: **22** (timeout: 10 m)
- Euphony outperforms in **20** / **22**

	Average	Median
Euphony	13 s	3 s
Flashfill	140 s	78 s



In the paper …

- General heuristic function for A* search
- How to preserve orthogonal search optimizations
- Feature maps for the three application domains
- Effectiveness of different models

Thank you!