Advanced Software Security

5. Search Space Prioritization

Kihong Heo

KAIST

Naive Enumerative Search

- Explore the search space in increasing size of programs (i.e., Occam's razor)
- With search space pruning techniques
- But, is this enough?

iter 0	x	y				
iter 1	$x + x$	$x - x$	$x + y$	$\ldots$	$x \leq y$	$\ldots$
iter 2	$x + x + y$	$x + x - y$	$\ldots$	$\text{if } (x \leq y)\ y\ x$		$\ldots$
iter 3	$x + x + x + y$		$\ldots$	$\text{if } (x \leq y)\ (y + x)\ x$		

Problem of Enumerative Search

- Blindly search over the large search space without any guidance
- Two major problems:
  - Scalability: #programs grows exponentially in program size
  - Quality: may overfit the I/O examples
- For example, $f(-1,0) = 0 \ \wedge \ f(0,-1) = 0$

iter 0	x	y				
iter 1	$x + x$	$x - x$	$x + y$	…	$x \leq y$	…
iter 2	$x + x + y$	$x + x - y$	…	if $(x \leq y)\ y\ x$		…
iter 3	$x + x + x + y$		…	if $(x \leq y)\ (y + x)\ x$		

But which one is more likely to be a solution?

$x - x$ **vs.** `if` $(x \leq y)\ y\ x$

Statistical Regularities in Programs

- Programs often contain **repetitive** and **predictable** patterns

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

- **Statistical** program models: probabilistic distribution over programs
  - E.g., n-gram, probabilistic context-free gramma (PCFG), etc

$$Pr(\texttt{??} \rightarrow \texttt{i++} \mid \texttt{for (i = 0; i < 100; ??)}) = 0.85$$
$$Pr(\texttt{??} \rightarrow \texttt{i--} \mid \texttt{for (i = 0; i < 100; ??)}) = 0.01$$

- Applications: code completion, deobfuscation, program repair, etc

A Solution: Euphony*

- Enumerate programs by **likelihood**, not by program size
- Likelihood is provided by **probabilistic models** over programs
  - *"How likely is the candidate program?"*
- Try the **most likely** (highest probability) candidate first

*Lee et al., Accelerating Search-Based Synthesis Using Learned Probabilistic Models, PLDI, 2018

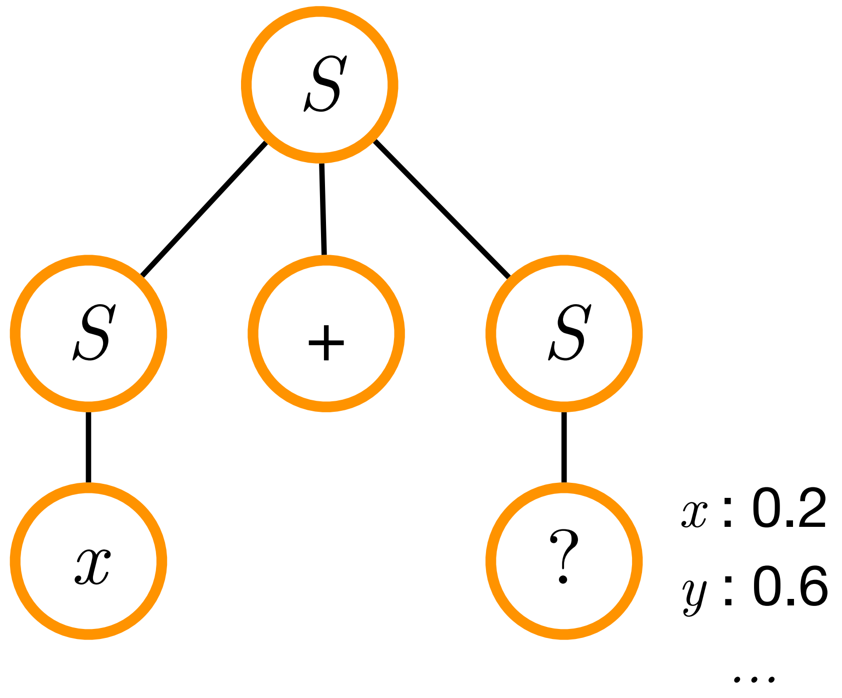
Probabilistic Language Model

- Learn a probabilistic model of programs from a corpus of programs
  - Human-written or auto-generated programs by other synthesizers
- A wide range of models is applicable

Program Corpus

$$
\begin{aligned}
&Pr(S \rightarrow S + S) = 0.3 \\
&Pr(S \rightarrow x \mid S + S) = 0.8 \\
&\quad\cdots \\
&Pr(S \rightarrow x \mid x + S) = 0.2 \\
&Pr(S \rightarrow y \mid x + S) = 0.6 \\
&\quad\cdots
\end{aligned}
$$

Learned Probabilistic Model



Probability of Programs

Probabilistic Language Model

- For a CFG $\langle N, \Sigma, R, S \rangle$,
- Given a context, provide the prob. of each production rule: Pr(rule | context)
  - Context: sentential form $\in (N \cup \Sigma)^*$
- Ultimately assign a probability to each program
- Example:

CFG $S \rightarrow x \mid 1 \mid S + S$

Probability of "x + 1"

$$S \rightarrow S + S \rightarrow x + S \rightarrow x + 1$$

$$\Pr(x+1) = \Pr(S \rightarrow S + S \mid S) \times \Pr(S \rightarrow x + S \mid S + S) \times \Pr(S \rightarrow x + 1 \mid x + S)$$

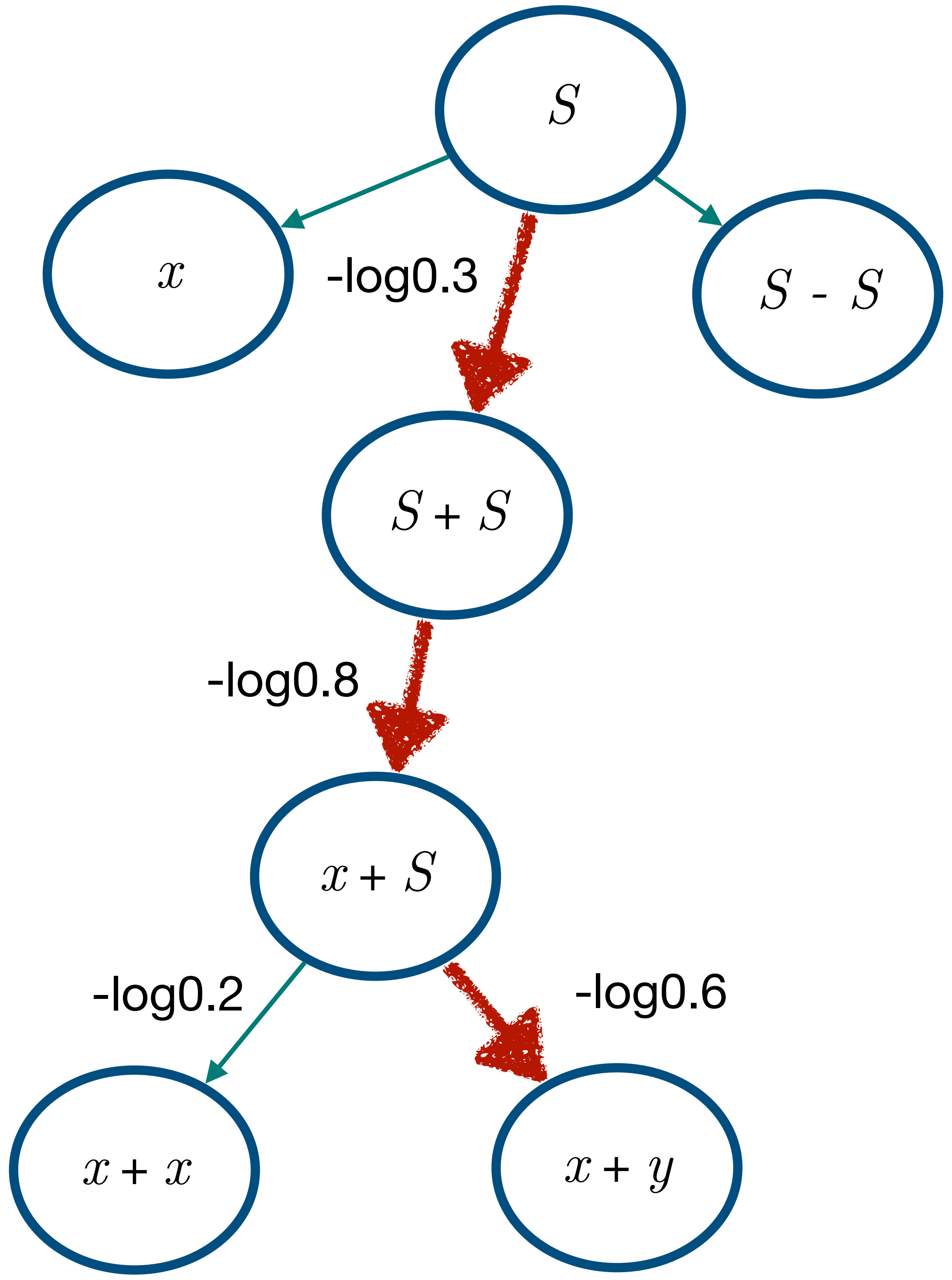
Example: PCFG

- Probabilistic Context Free Grammar (PCFG)
- One of the simplest form of probabilistic language model: ignore context
- Provide a probability to each production rule

$A \to \beta$	P
$S \to 0$	0.2
$S \to 1$	0.3
$S \to x$	0.1
$S \to S + S$	0.4
$S \to S - S$	0.3

Guided Enumeration by Probabilistic Model

- Given a model, construct a directed graph
  - Node: sentential forms
  - Weight: negative log probability of a production rule
- Compute the shortest path
  - starting from the start symbol to the program
  - E.g., Dijkstra's, A*, etc



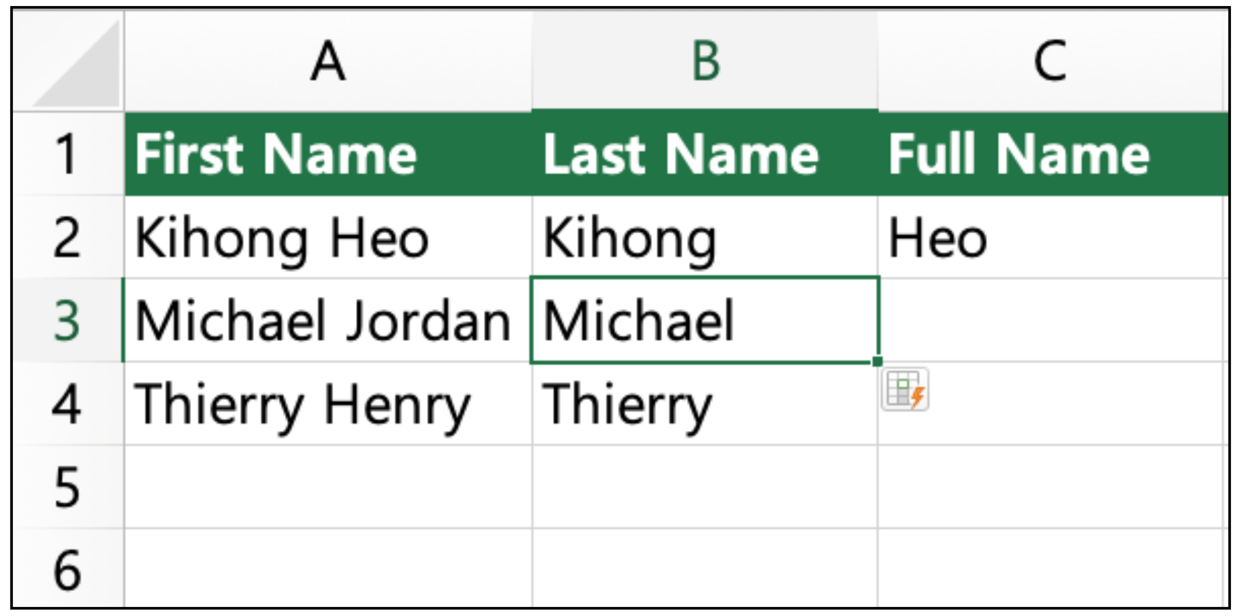
$Pr(S \to S + S) = 0.3$
$Pr(S \to x \mid S + S) = 0.8$
…
$Pr(S \to x \mid x + S) = 0.2$
$Pr(S \to y \mid x + S) = 0.6$
…

Guided Top-down Enumeration

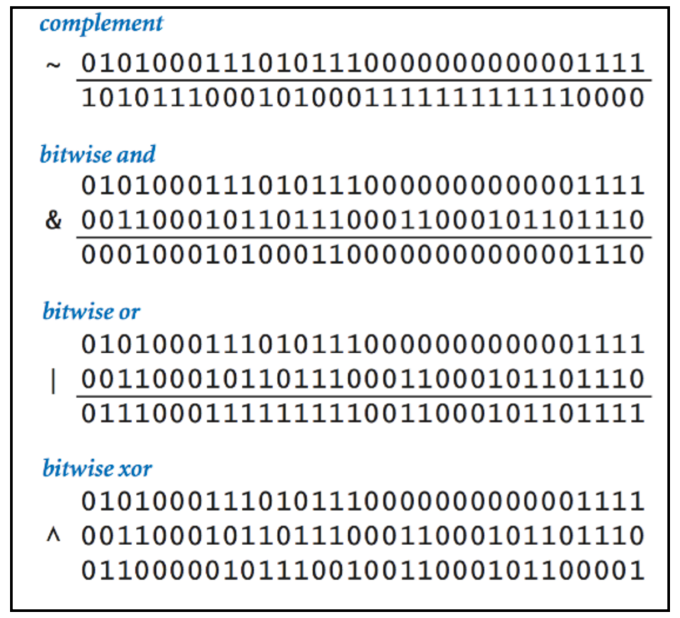
```
top-down(G = <Σ, N, R, S>, ϕ):
  Q := {(S, 0)}
  while Q != {}:
    (p, d) := dequeue_min(Q)
    if ground(p) ∧ ϕ(p): return p
    P’ := unroll(G, p, d)
    forall p’ ∈ P’:
      if not equiv(p, p’):
        enqueue(Q, p’)


unroll(G = <Σ, N, R, S>, p, d):
  Q’ := {}
  A := left-most non-terminal in p
  forall (A → B) in R:
    p’ := p[B/A]
    Q’ := Q’ ∪ {(p’, d + w(p, p’)}
  return Q’
```

Experimental Setup

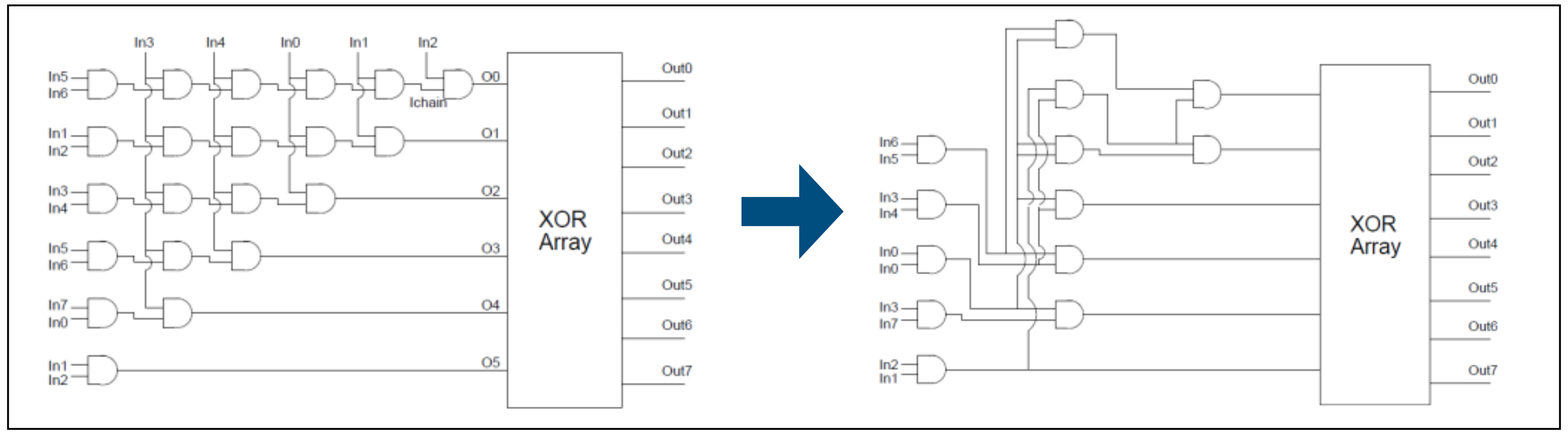
- 1167 tasks from 3 different domains



STRING: End-user programming for string manipulations
(205 tasks)



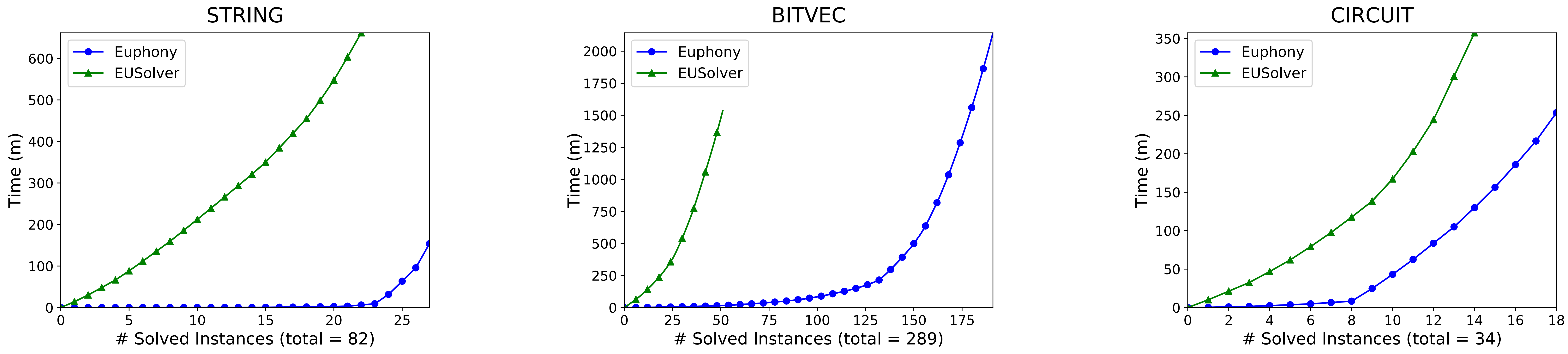
BITVEC: Efficient low-level algorithms
(750 tasks)



CIRCUIT: Attack-resistant crypto circuits generations
(212 tasks)

Effectiveness

- Comparison to EUSolver (a program synthesizer without prob. guidance)
  - Training: 762 tasks solved by EUSolver in 10 minutes
  - Testing: 405 (timeout: 1 hour)



Summary

- Problem: scalability and quality
- Euphony: a program synthesizer guided by a **learned probabilistic model**
  - E.g., probabilistic program model + shortest pathfinding
- Need a lot more research on efficient search
  - E.g., advanced learning techniques, static analysis, constraint solving, etc