Search Prioritization

Woosuk Lee

CSE9116 SPRING 2024

Hanyang University

Limitations of Enumerative Search
1. Limited Scalability

- Explore candidates in order of increasing size
  - Good for finding generalizable solutions (Occam's razor)
- What if desired solutions are large?
  - Search space exponentially grows
- Enumerative search wastes computation resources for exploring many "unlikely" candidates.

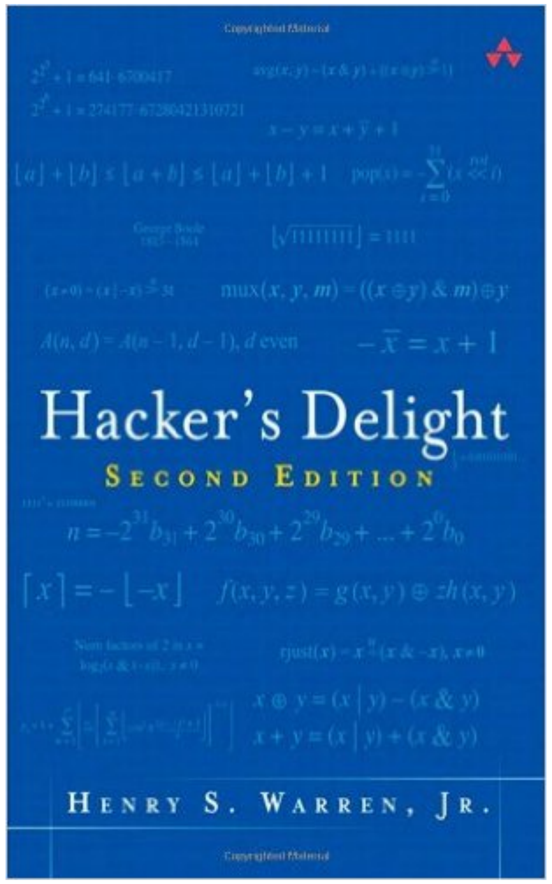
Example: Hacker's Delight

- Find a program transforming rightmost contiguous 1's into 0's (without a loop)

- Target : f(x: BitVec) : BitVec

- Syntactic constr.:

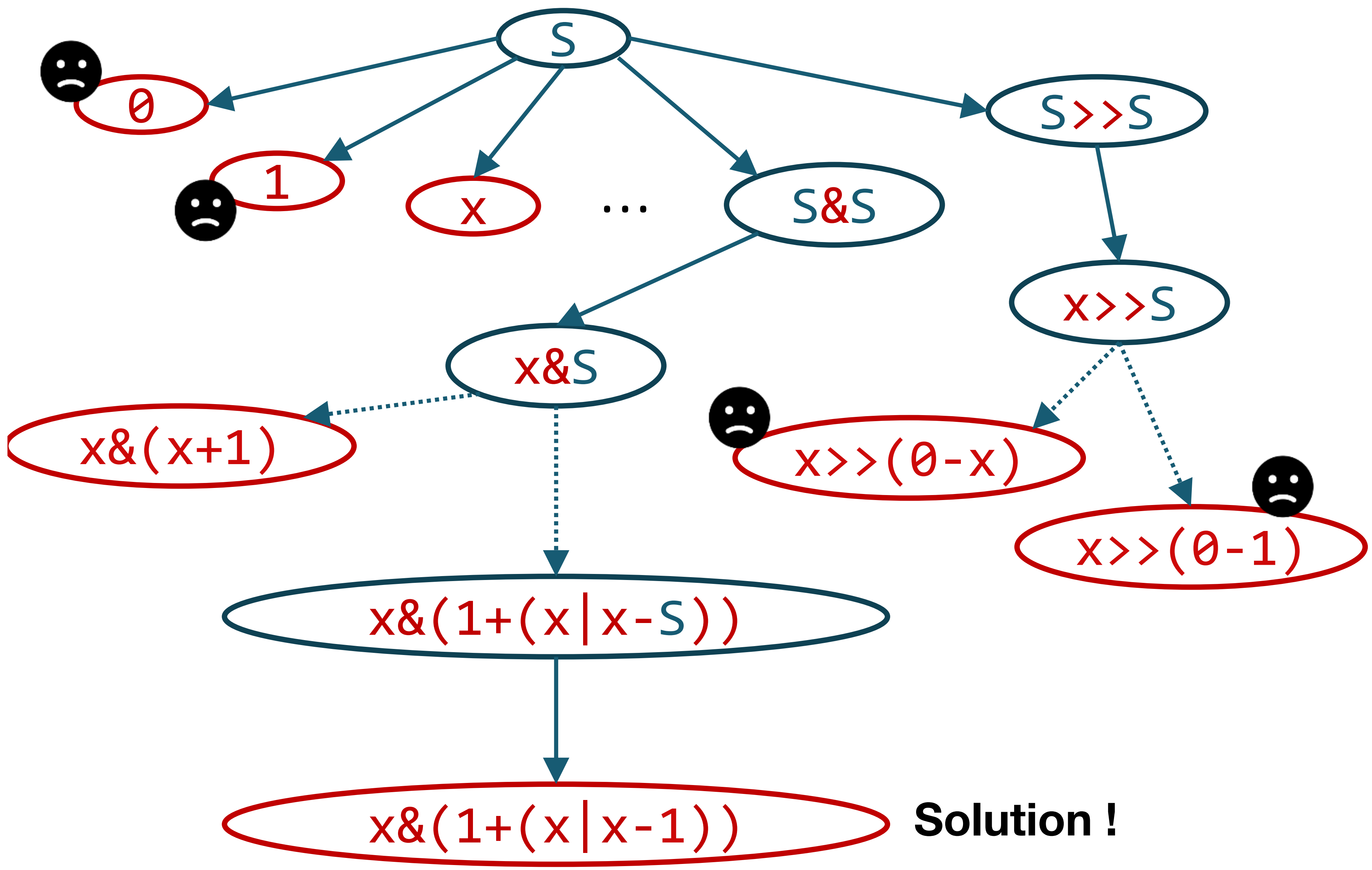
```
S ->   0 | 1 | x |
       S + S     |
       S - S     |
       S & S     |
       S | S     |
       S << S    |
       S >> S
```



- Semantic constr: f(00101) = 00100, f(10110) = 10000 …

Example: Hacker's Delight

Many "unlikely" candidates are explored by top-down search!



From https://github.com/nadia-polikarpova/cse291-program-synthesis

Limitations of Enumerative Search
2. Overfitting

- Despite Occam's razor, enumerative search does not guarantee most "likely" solutions.
  - "likely": generalizable beyond given I/O examples

Statistical Regularities in Programs

- Programs contain repetitive and predictable patterns [Hindle et al. ICSE'12]

```
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, neural network (e.g., LSTM), — Sequence-based

probabilistic context-free grammar (PCFG),

probabilistic higher-order grammar (PHOG)… — Grammar-based

- Many applications: code completion, deobfuscation, program repair, etc.

Applications of Statistical Program Models

Input: code snippet with holes

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  ? {smsMgr, msgList}  // (H1)
} else {
  ? {smsMgr, message}  // (H2)
}
```

SLANG

Output: holes completed with (sequences) of method calls

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  smsMgr.sendMultipartTextMessage(...msgList...);
} else {
  smsMgr.sendTextMessage(...message...);
}
```

Raychev et al., Code Completion with Statistical Language Models, PLDI'14

Applications of Statistical Program Models

- Fixing syntactic errors

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  for a in range(0, len(poly) - 1):
    f = poly[a]*x**a+f
    a += 1
  return f
```

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  while a < len(poly):
    f = poly[a]*x**a+f
    a += 1
  return f
```

- Program = sequence of tokens

- Fix syntax errors using a skip-gram model

Pu et al., sk_p: a neural program corrector for MOOCs

Exploiting Statistical Regularities

- Can we leverage statistical program models to accelerate program synthesis?

- Key Challenges

  - ***Guided search*:** How to guide 1.the search given a statistical model?

  - **Learning models:** How to learn a good statistical model?

Euphony for Guiding Top-Down Enumerative Search

- Woosuk Lee, Kihong Heo, Rajeev Alur, Mayur Naik, Accelerating Search-Based Synthesis Using Learned Probabilistic Moels, PLDI'18
- **Guided search:** A general approach to accelerate *CEGIS*-based program synthesis
  - by using a probabilistic model to guide the search towards likely programs
  - supports a wide range of models (e.g., *n*-gram, PCFG, PHOG, neural nets, …)
- **Learning models**: Transfer learning-based method to mitigate overfi tting
- https://github.com/wslee/euphony

Example SyGuS Problem

- Goal: a function f that replaces a hyphen (-) by a dot (.) in a given string x
- Specification

  String concatenation

  Syntactic specification:

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

  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 $\boxed{\mathsf{Rep}(x, \text{“-”}, \text{“.”}).}$

Guided Search

- Existing unguided enumerative search

$$\underbrace{\text{“.”},\ \text{“-”},\ x}_{\textbf{Size 1}},\ \underbrace{\text{“-”} + \text{“-”}, \cdots, x + \text{“.”}}_{\textbf{Size 3}},\ \underbrace{\mathsf{Rep}(x, \text{“.”}, \text{“-”}),\ \boxed{\mathsf{Rep}(x, \text{“-”}, \text{“.”})}}_{\textbf{Size 4}}$$

  - Unlikely candidates (e.g.,“-“ + “-“) are explored.

- Guided enumerative search

$$\underbrace{x + \text{“.”}}_{\textbf{Prob 0.27}},\ \underbrace{x + \text{“-”}}_{\textbf{Prob 0.21}}, \cdots,\ \underbrace{\boxed{\mathsf{Rep}(x, \text{“-”}, \text{“.”})}}_{\textbf{Prob 0.11}}$$

  - Likely candidates are explored first, while preserving the existing pruning optimizations

Guided Search

- **Model learning**
- Guided enumerative search

(Grammar-based) Statistical Program Models

- Given CFG $\langle N, \Sigma, R, S \rangle$
- and an incomplete program: $(N \cup \Sigma)^*$ (i.e., *sentential form*),
- provides a probability for each production rule applicable next (usually on the leftmost nonterminal)
  - *Pr(production rule | sentential form)*

(Grammar-based) Statistical Program Models

- Determines a probability of a given program
- E.g., probability of $x + \text{``.''}$

$$\underline{S} \Longrightarrow \underline{S + S} \Longrightarrow \underline{x + S} \Longrightarrow x + \text{``.''}$$

$$Pr(S \rightarrow S + S \mid \underline{S})$$
$$\times\ Pr(S \rightarrow x \mid \underline{S + S})$$
$$\times\ Pr(S \rightarrow \text{``.''} \mid \underline{x + S})$$

Example

- P*robabilistic context-free grammar* (PCFG)

$$
\begin{array}{rclc}
 & A \rightarrow \beta & & P \\
S & \rightarrow & \text{``.''} & 0.2 \\
S & \rightarrow & \text{``-''} & 0.2 \\
S & \rightarrow & x & 0.1 \\
S & \rightarrow & S + S & 0.1 \\
S & \rightarrow & \texttt{Rep}(S, S, S) & 0.4
\end{array}
$$

- Limitation: context around the place where a rule is applied is not considered → imprecise

A Uniform Interface to Statistical Program Models

- A statistical program model is $G_q = \langle G, C, p, q \rangle$ for a given CFG $G = \langle N, \Sigma, R, S \rangle$

  - C — set of *contexts*

  - $p : (N \cup \Sigma)^* \to C$ — Given a sentential form, for extracting contextual information around the next hole (i.e., nonterminal) to be filled

  - $q : R \times C \to \mathbb{R}^+$ — Considering contextual information, for determining a probability for production rule

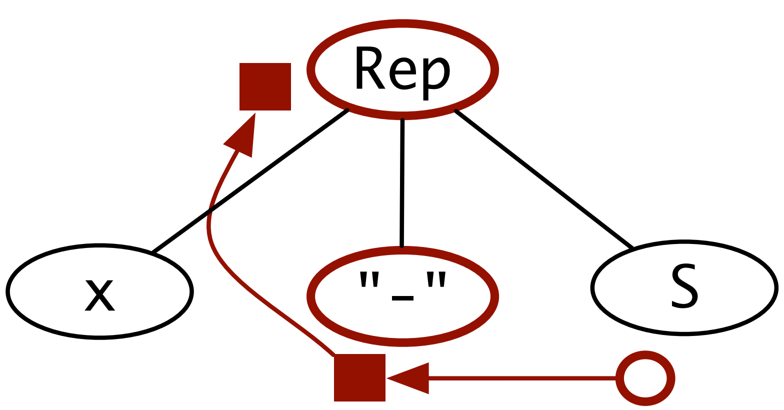
Contexts

- Sequence of terminal/nonterminal symbols

- E.g., 2-gram

$$\mathrm{p}(\mathrm{x}+\mathrm{S}) = [+,\ \mathrm{x}]$$

- E.g., Sibling and parent nodes



$$\mathrm{p}(\mathtt{Rep}(\mathtt{x},\ \text{“-”},\ \mathrm{S})) = [\text{“-”},\ \mathtt{Rep}]$$

Example

Probabilistic Higher-order Grammar (PHOG) — the model used by Euphony

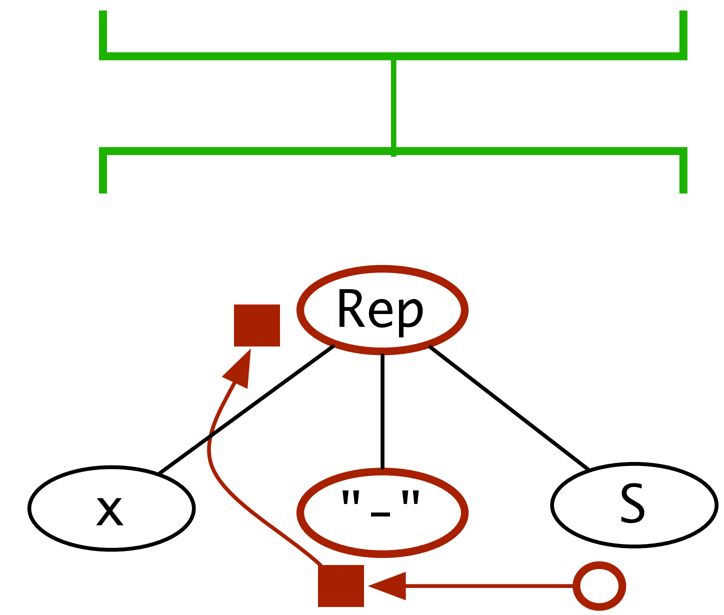
$A[\textcolor{darkred}{context}] \to \beta$

			P
$S[\text{“-”}, \texttt{Rep}]$	$\to$	“.”	**0.72**
$S[\text{“-”}, \texttt{Rep}]$	$\to$	“-”	**0.001**
$S[\text{“-”}, \texttt{Rep}]$	$\to$	x	0.12
$S[\text{“-”}, \texttt{Rep}]$	$\to$	$S + S$	0.02
$S[\text{“-”}, \texttt{Rep}]$	$\to$	$\texttt{Rep}(S, S, S)$	0.139

$\ldots$

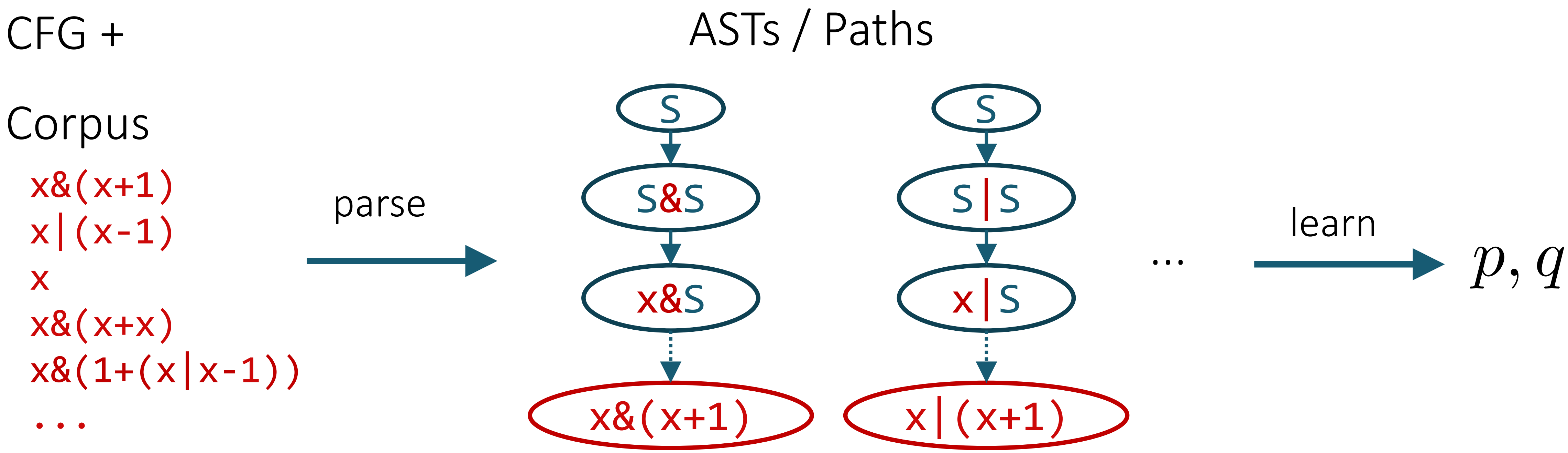
PHOG when $context$ is symbols at **left sibling** and **parent**

$$Pr(S \to \text{“.”} \mid \text{Rep}(\text{“}x\text{”}, \text{“-”}, S)) = 0.72$$



Bielik et al., Probabilistic Higher-Order Grammar, ICML’16

Learning a PHOG



- From *derivation sequences* of training programs, count occurrences of each rule application under certain contexts

  - E.g., From derivation sequence x & S $\Longrightarrow$ x & 1 , x | S $\Longrightarrow$ x | 1, production rule S → 1 is counted twice when sibling is x

- Prob. of $\alpha \rightarrow \beta$ under context γ : $q(\alpha[\gamma] \rightarrow \boldsymbol{\beta}) = \dfrac{Count(\alpha[\gamma] \rightarrow \boldsymbol{\beta})}{Count(\alpha[\gamma])}$

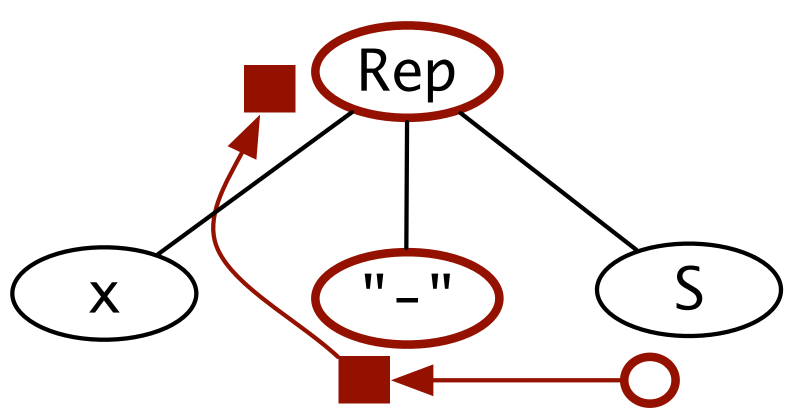
Learning a PHOG

- How to determine which contexts matter?

- The p function: program written in the TCond language

$$\begin{aligned} \texttt{TCond} &\rightarrow \epsilon \mid \texttt{Write TCond} \mid \texttt{MoveOp TCond} \\ \texttt{MoveOp} &\rightarrow \texttt{Up} \mid \texttt{Left} \mid \texttt{Right} \mid \texttt{DownFirst} \mid \texttt{DownLast} \end{aligned}$$

- E.g., collect contents of sibling and parent nodes



$\texttt{Left} \cdot \texttt{Write} \cdot \texttt{Up} \cdot \texttt{Write}$

Learning a PHOG

- Given training programs $\mathcal{D}$, Find TCond program s.t.

$$p_{best} = \underset{p' \in \textsc{TCond}}{\arg\min}\ cost(\mathcal{D}, p').$$

sum of negative log probabilities of training programs using p' for extracting contexts

- By using a genetic algorithm

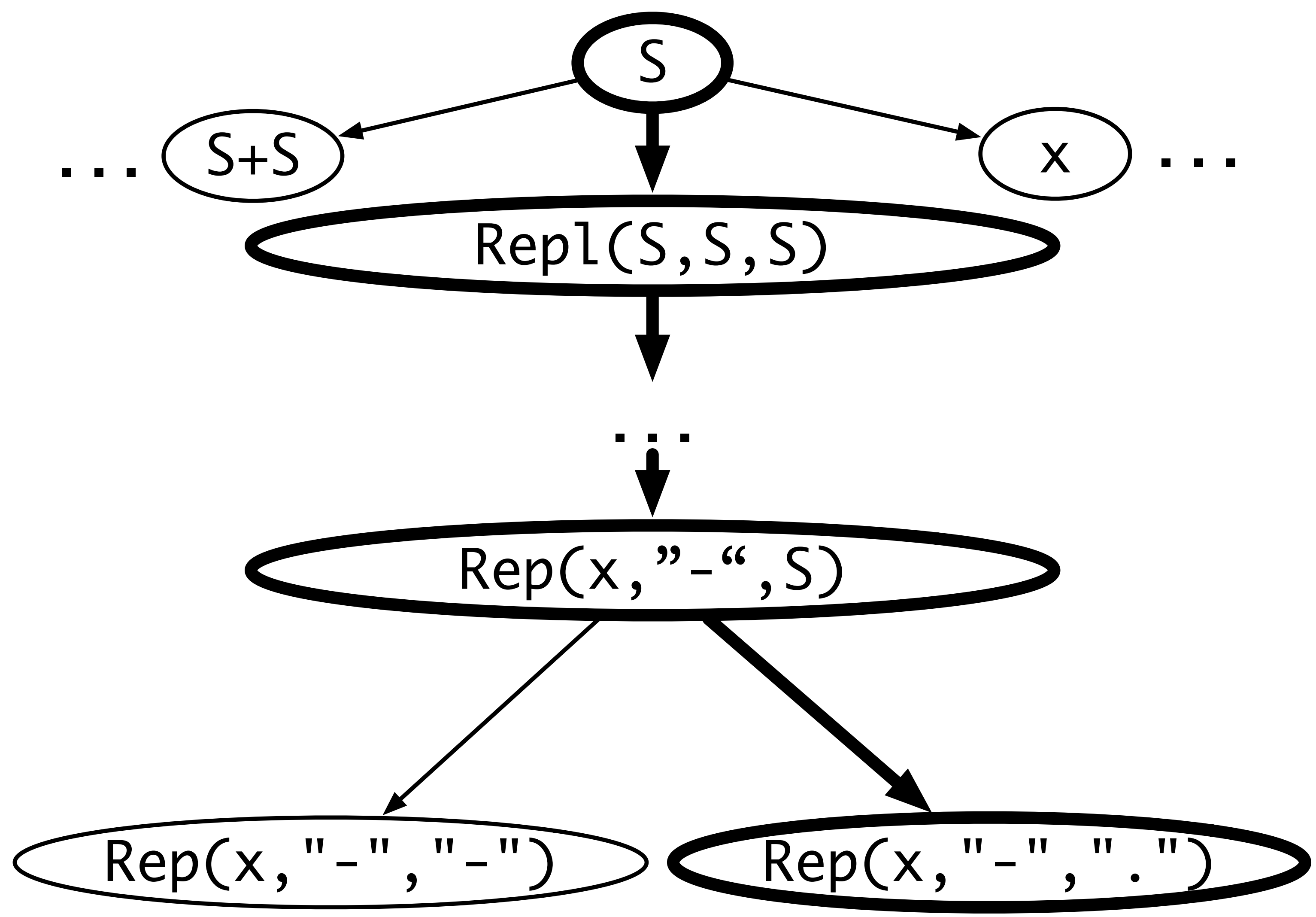
Guided Search

- Model Learning
- **Guided enumerative search**

Guided Search as Path Finding

Contruct a directed weighted graph from a given CFG

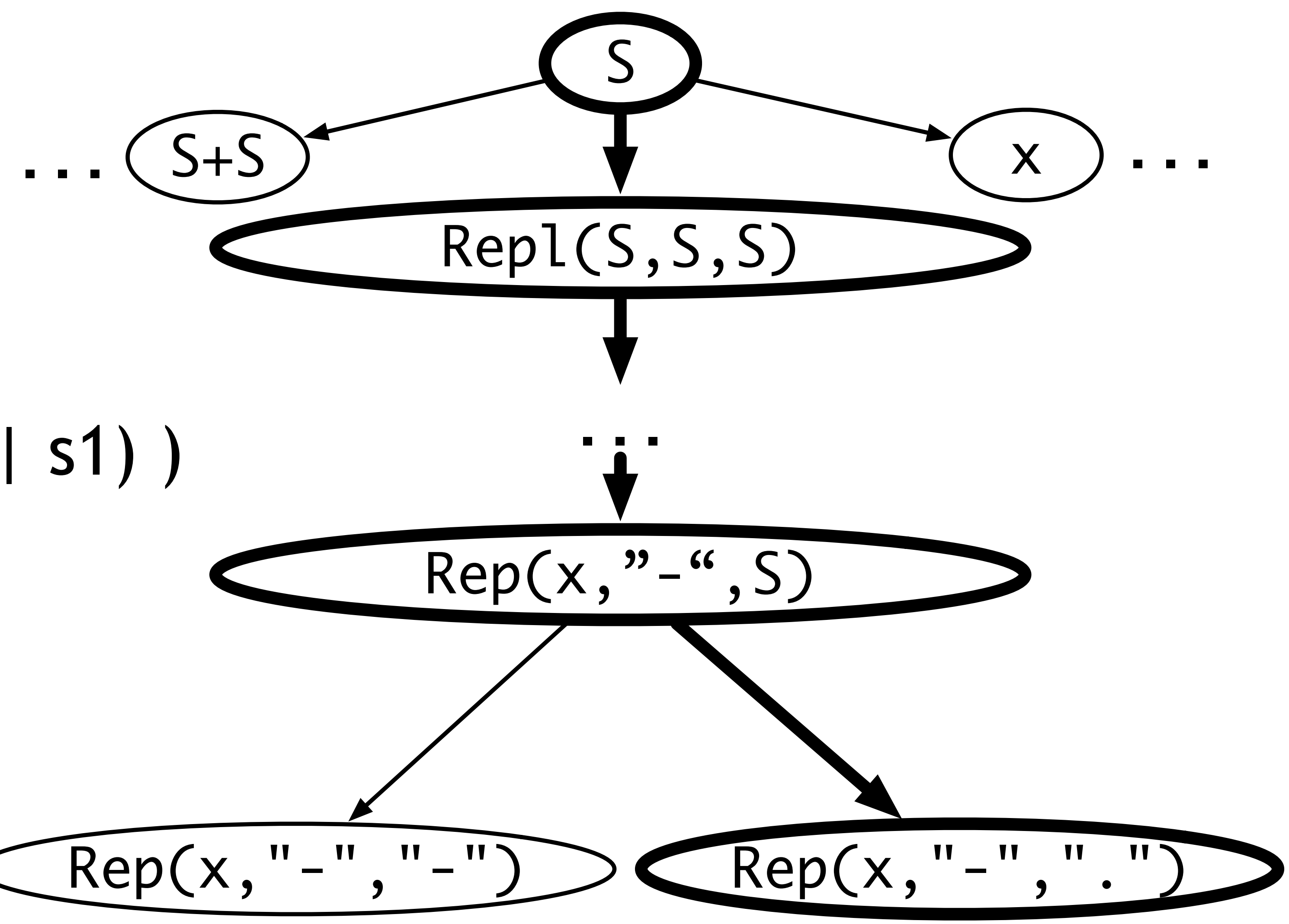
- Nodes: sentential forms
- Edges: derivations between sentential forms
- Terminal nodes: sentences



Guided Search as Path Finding

Contruct a directed weighted graph from a given CFG

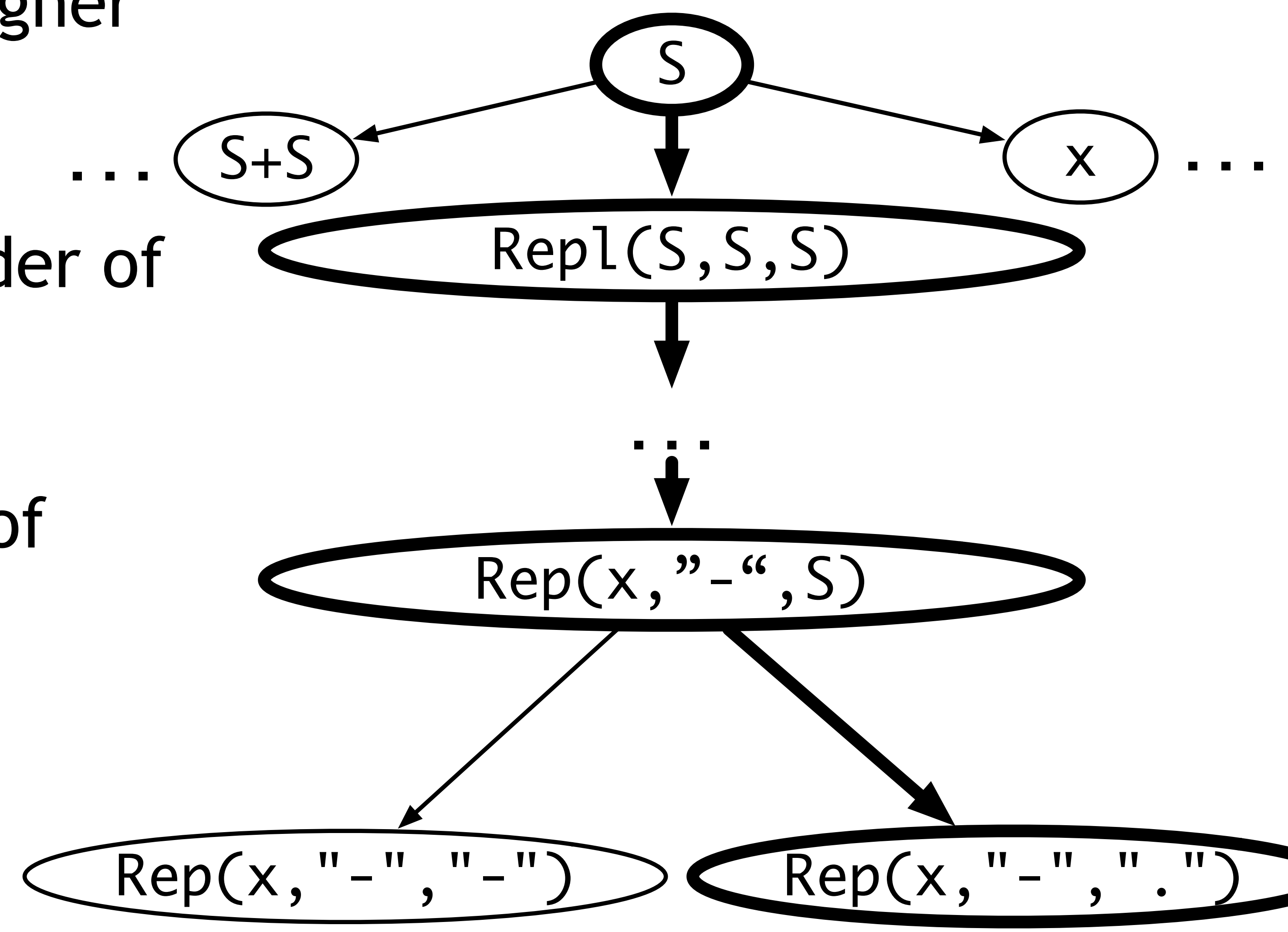
- Weights:
  - w(s1 → s2) = – log (Pr (r | s1))



Guided Search as Path Finding

Contruct a directed weighted graph from a given CFG

- Goal: explore candidates of higher probabilities first
- = enumerating programs in order of decreasing probability
- = enumerating paths in order of increasing distance



Unguided Top-Down Search

```
TopDown (grammar G = ⟨N, Σ, R, S⟩, spec Φ):
Q := {S}
while Q != ∅:
  remove p from Q
  if Φ(p): return p
  P' := Unroll(G, p)
  forall p' ∈ P':
    Q := Q.Enqueue(p')

Unroll (grammar G, spec Φ):
P' := ∅
forall A ∈ p:
  forall A → B ∈ R:
    p' := p[B/A]
    P' := P' U {p'}
return P'
```

Guided Top-Down Search

```
TopDown (grammar G = ⟨N, Σ, R, S⟩, spec ɸ):
Q := {(S,0)}
while Q != ∅:
  remove (p, d) whose d is minimal from Q
  if ɸ(p): return p
  P' := Unroll(G, p, d)
  forall p'∈ P':
    Q := Q.Enqueue(p')
```

Candidates are with their distances from the root (S)

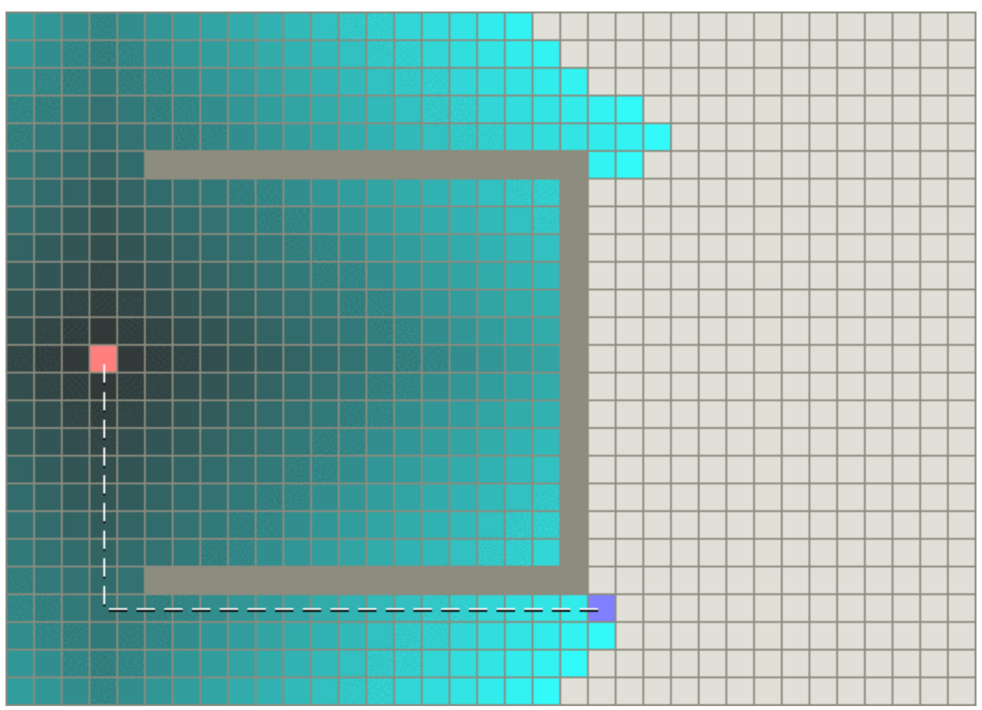
Pick one closest to S

```
Unroll (grammar G, program p, distance d):
P' := ∅
forall A ∈ p:
  forall A → B ∈ R:
    p' := p[B/A]
    P' := P' U {(p', d + w(p,p'))}
return P'
```

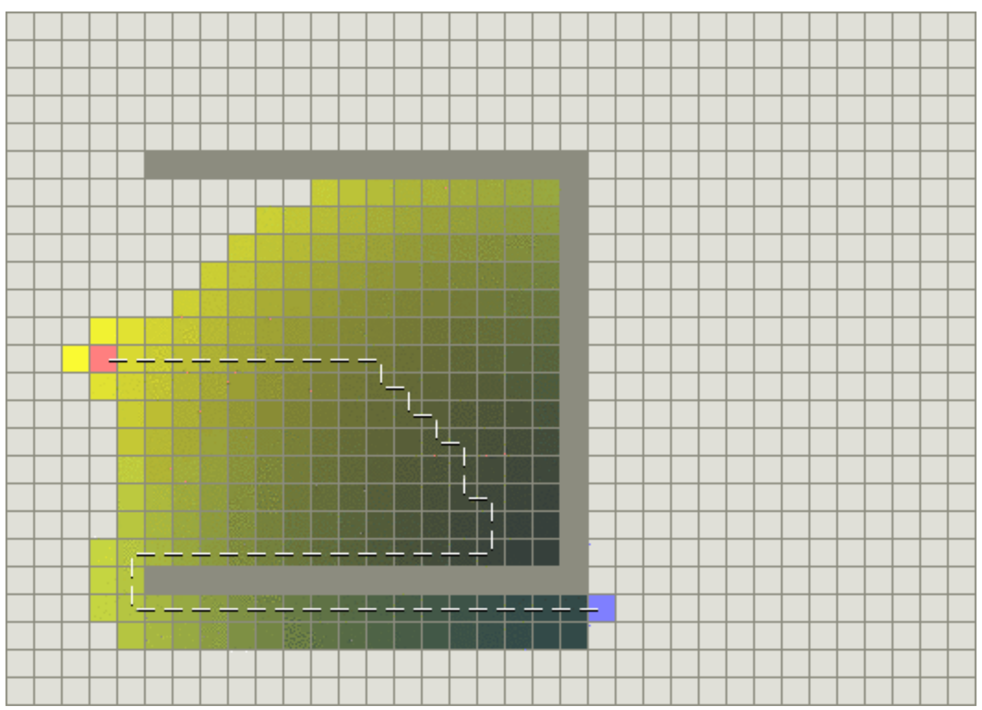
Add new candidates to the queue with their distances from S

Better Guided Top-Down Search

- The previous algorithm is based on Dijkstra algorithm
- We can use A*, which is better.



Dijkstra



A*

Red: start node
Blue: goal node
GreenYellow: explored nodes

A* Search

- Dijkstra: picks one closest to the root
- A*: picks one of the *estimated* shortest path (= distance from the root + <u>guessed future distance to the closest goal node</u>)
- Often infeasible to compute the exact future distance — an *under-approximation* is used.
- Heuristic function *g :* Node → Guessed Future distance
- A* finds the shortest paths if the heuristic function always underestimates future distances.

Guided Top-Down Search (improved)

```
TopDown (grammar G = ⟨N,Σ,R,S⟩, spec ϕ):
Q := {(S,0,g(S))}
while Q != ∅:
  remove (p, d, h) whose d + h is minimal from Q
  if ϕ(p): return p
  P' := Unroll(G, p, d)
  forall p'∈ P':
    Q := Q.Enqueue(p')
```

Guessed future distance
$g : (N \cup \Sigma)^* \to \mathbb{R}^+$

Pick one of the estimated shortest path
(distance so far + future distance)

```
Unroll (grammar G, program p, distance d):
P' := ∅
forall A ∈ p:
  forall A → B ∈ R:
    p' := p[B/A]
    P' := P' U {(p', d + w(p,p'), g(p')}
return P'
```

Add new candidates to the queue
with their gussed future distances

How to compute g?

- $n \overset{\mathbf{r}}{\rightsquigarrow} s$: path from n to s
- $w(n \overset{\mathbf{r}}{\rightsquigarrow} s)$: distance of the path from n to s
- Ideal heuristic function:

$$g^*(n) = \min_{s \in \Sigma^*,\, n \overset{\mathbf{r}}{\rightsquigarrow} s} w(n \overset{\mathbf{r}}{\rightsquigarrow} s)$$

- which is infeasible ($\because$ possibly infinitely many goal nodes reachable from n)

How to compute g?

- Ues an underapproximation

- Compute the *h* function satisfying the following condition

$$\forall A \in N.\ h(A) = \max_{A \to \beta \in R, c \in C} \left(q(A \to \beta \mid c) \times \prod_{\beta_i \in N} h(\beta_i) \right)$$

1) start with *h(A) = 0* for all *A*

2) repeatedly update h according to the above equation until saturation

- E.g., Consider the following PCFG

$$S \quad \longrightarrow \quad aSb \quad (0.9) \qquad\qquad S \quad \longrightarrow \quad c \quad (0.1)$$

1st iteration: $h(S) = \max(0.9 \times 0,\ 0.1) = 0.1.$

2nd iteration: $h(S) = \max(0.9 \times 0.1,\ 0.1) = 0.1.$

The highest probability of program derivable from S is 0.1

How to compute g?

- Define the following function using the *h* function

$$g(n) = \begin{cases} 0 & (n \in \Sigma^*) \\ -\sum_{n_i \in N} \log_2 h(n_i) & (\text{otherwise}) \end{cases}$$

n_i : i-th symbol in n

- This heuristic function is correct (why?):

$$\forall n \in (N \cup \Sigma)^*.\ g(n) \le g^*(n).$$

Overfitting

- The guided search quickly finds the solution $\boxed{\text{Rep}(x, \text{“-”}, \text{“.”})}$.

- What if a similar problem of the following semantic constraint is given?

$$f(\text{“12.31”}) = \text{“12-31”} \wedge f(\text{“01.07”}) = \text{“01-07”}.$$

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

Overfitting

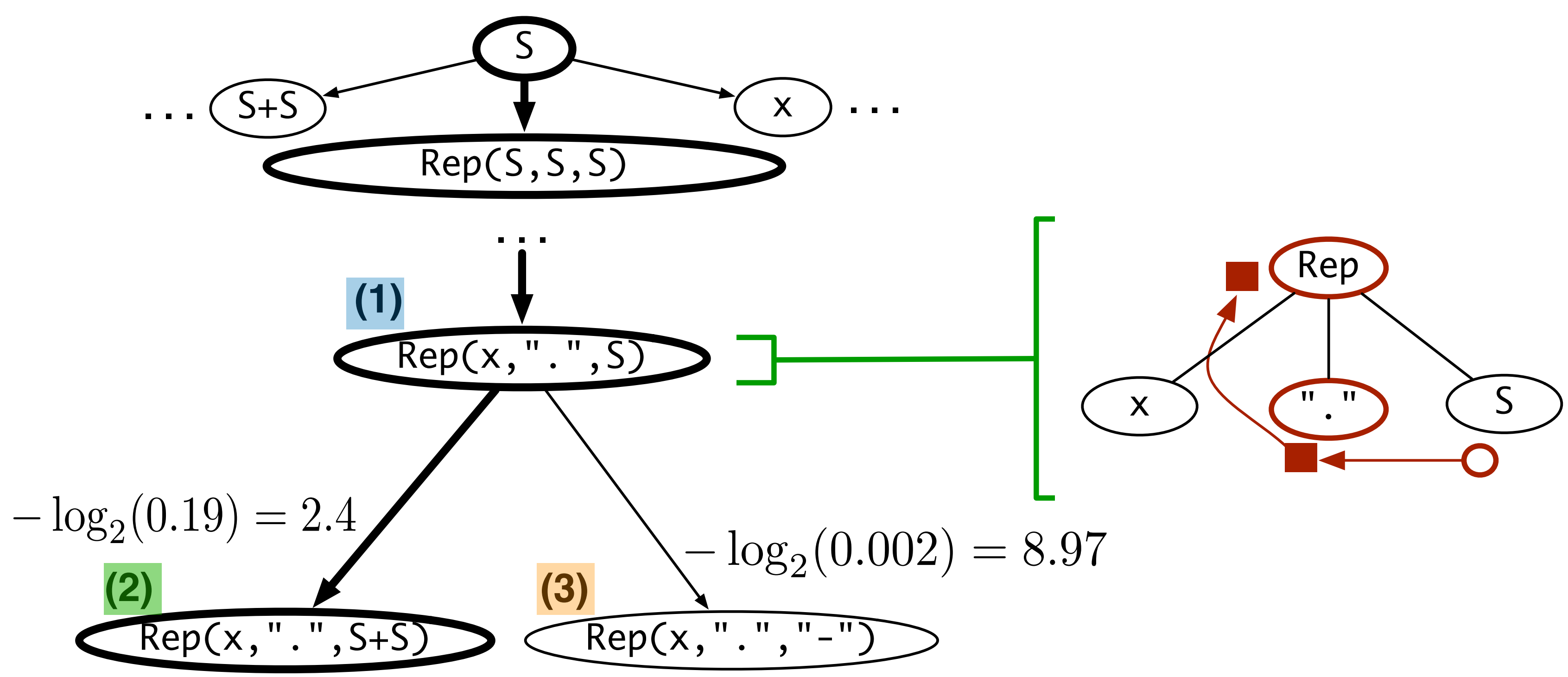
- Suppose $\texttt{Rep}(x, \text{“.”}, S)$ (node **(1)**) is currently explored.

- Using the PHOG we have, node **(2)** is preferred above **(3)** as the next candidate

$A[\textcolor{darkred}{context}] \to \beta$

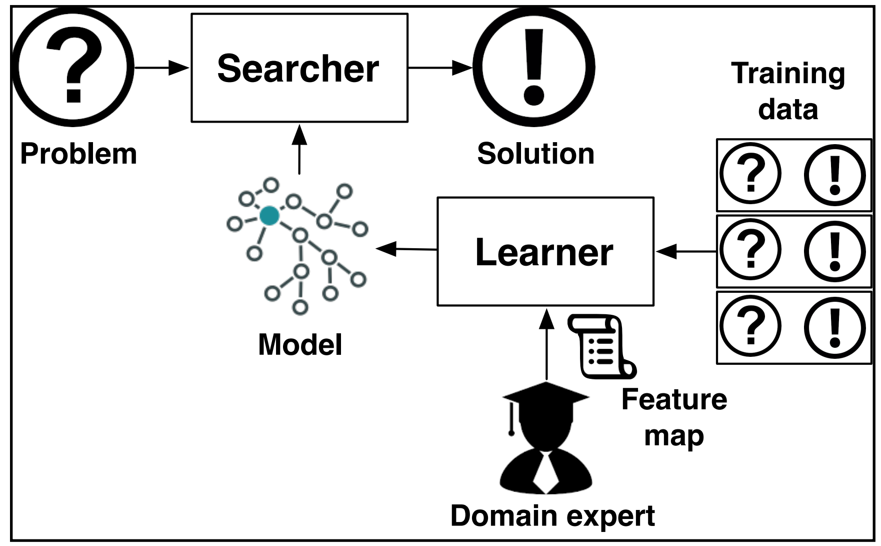
			P
$S[\text{“-”}, \texttt{Rep}]$	$\to$	“.”	**0.72**
$S[\text{“-”}, \texttt{Rep}]$	$\to$	“-”	**0.001**
$S[\text{“-”}, \texttt{Rep}]$	$\to$	x	0.12
$S[\text{“-”}, \texttt{Rep}]$	$\to$	$S+S$	0.02
	$\dots$		

			P
$S[\text{“.”}, \texttt{Rep}]$	$\to$	“.”	0.001
$S[\text{“.”}, \texttt{Rep}]$	$\to$	“-”	**0.002**
$S[\text{“.”}, \texttt{Rep}]$	$\to$	x	0.01
$S[\text{“.”}, \texttt{Rep}]$	$\to$	$S+S$	0.19
	$\dots$		



- Because statistical models like PHOG only consider *syntactic information.*

Transfer Learning



- Training data: solutions of existing synthesis problems
- Testing data: solutions of unseen synthesis problems
- They may follow different probability distributions because of diverse semantic specifications.
- Transfer learning reduces discrepancy between the probability distributions of training and testing data

Transfer Learning

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

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

Constant string that appear in the inputs

Constant string that appear in the outputs

- Spec: $f(\text{“12.31”}) = \text{“12-31”} \wedge f(\text{“01.07”}) = \text{“01-07”}.$

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

Constant string that appear in the inputs

Constant string that appear in the outputs

Transfer Learning

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

$$\texttt{Rep}(x, \text{“-”}, \text{“.”}) \longrightarrow \texttt{Rep}(x, \mathrm{const}_I, \mathrm{const}_O)$$

Constant string that appear in the inputs

Constant string that appear in the outputs

- Spec: $f(\text{“12.31”}) = \text{“12-31”} \wedge f(\text{“01.07”}) = \text{“01-07”}.$

$$\texttt{Rep}(x, \text{“.”}, \text{“-”}) \longrightarrow \texttt{Rep}(x, \mathrm{const}_I, \mathrm{const}_O)$$

<u>**Now the solutions of the two problems become equal**</u>

Types of Constants

- I : Input examples O : Output examples
- $const_{IO}$: constants that appear in I and O
- $const_{I}$: constants that appear in I
- $const_{O}$: constants that appear in O
- $const_{\perp}$: constants that appear in neither I nor O

Pivot PHOG

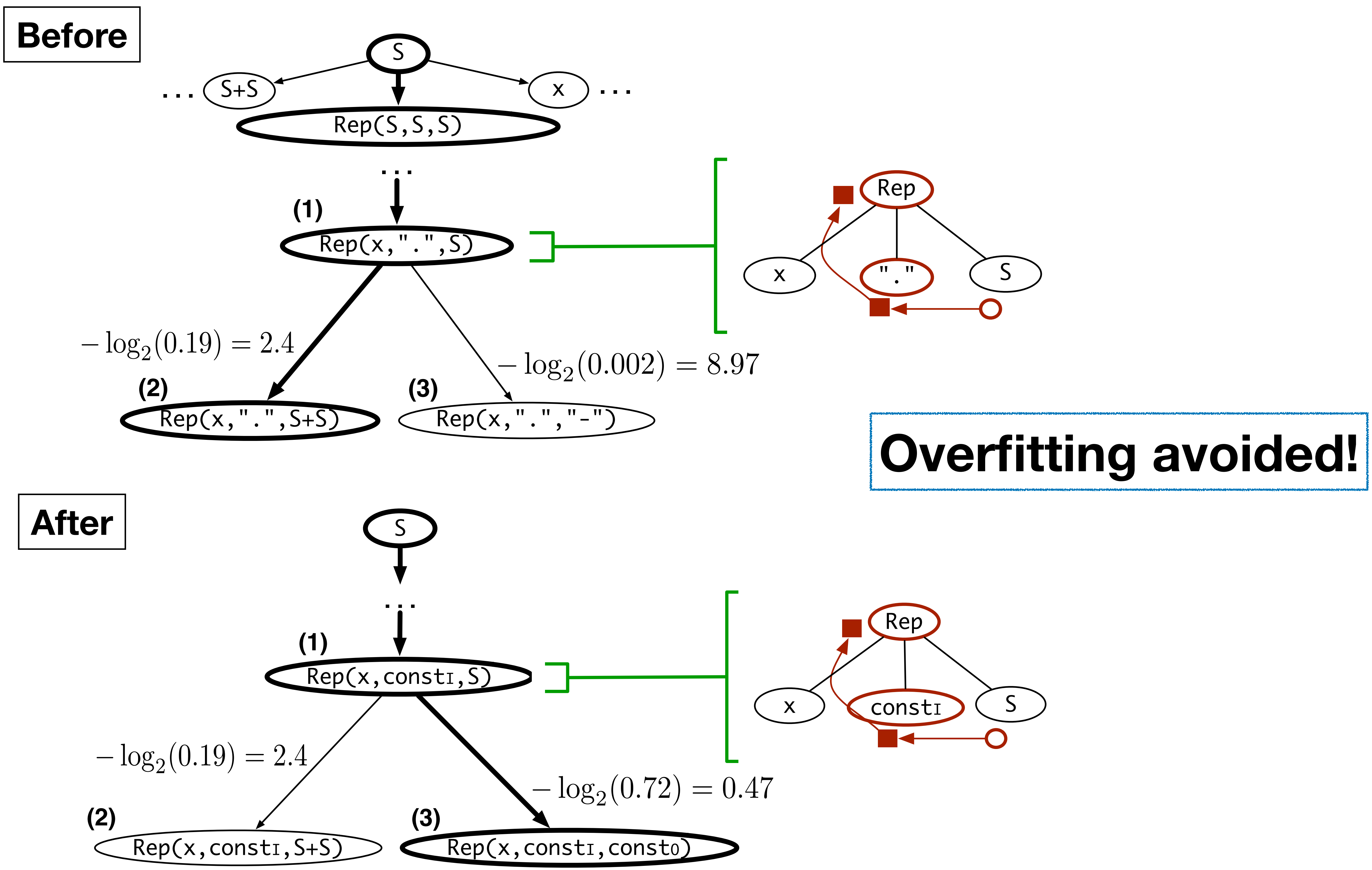
$$
\begin{array}{rcl}
S & \rightarrow & x \mid S + S \\
& \mid & \mathtt{Rep}(S, S, S) \\
& \mid & \mathtt{const}_{IO} \mid \mathtt{const}_{I} \\
& \mid & \mathtt{const}_{O} \mid \mathtt{const}_{\perp}
\end{array}
$$

(a) A pivot grammar for string manipulation tasks

$$
\begin{array}{lcll}
\multicolumn{3}{c}{A[\textcolor{darkred}{context^{\#}}] \rightarrow \beta^{\#}} & P \\
S[\mathtt{const}_I, \mathtt{Rep}] & \rightarrow & \mathtt{const}_O & \mathbf{0.72} \\
S[\mathtt{const}_I, \mathtt{Rep}] & \rightarrow & \mathtt{const}_I & \mathbf{0.001} \\
S[\mathtt{const}_I, \mathtt{Rep}] & \rightarrow & x & 0.12 \\
S[\mathtt{const}_I, \mathtt{Rep}] & \rightarrow & S + S & 0.02 \\
& \cdots & & P \\
S[\mathtt{const}_O, \mathtt{Rep}] & \rightarrow & \mathtt{const}_O & 0.001 \\
S[\mathtt{const}_O, \mathtt{Rep}] & \rightarrow & \mathtt{const}_I & 0.002 \\
S[\mathtt{const}_O, \mathtt{Rep}] & \rightarrow & x & 0.01 \\
S[\mathtt{const}_O, \mathtt{Rep}] & \rightarrow & S + S & 0.19 \\
& \cdots & &
\end{array}
$$

(b) A pivot PHOG learned using the pivot grammar

Guided Search with a Pivot PHOG



Other Examples of Exploiting Spec

An input-output example:
Input:
`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`
Output:
`[-12, -20, -32, -36, -68]`

$$\downarrow$$

a ← [int]
b ← FILTER (<0) a
c ← MAP (*4) b
d ← SORT c
e ← REVERSE d

Balog et al., DEEPCODER: Learning To Write Programs

Other Examples of Exploiting Spec

An input-output example:

Input:

`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`

Output:

`[-12, -20, -32, -36, -68]`

Neural net inference

(+1)	(-1)	(*2)	(/2)	(*-1)	(**2)	(*3)	(/3)	(*4)	(/4)	(>0)	(>0)	(%2==1)	(%2==0)	HEAD	LAST
.0	.0	.1	.0	.0	.0	.0	.0	**1.0**	.0	.0	**1.0**	.0	.2	.0	.0

MAP	FILTER	SORT	REVERSE	TAKE	DROP	ACCESS	ZIPWITH	SCANL1	+	-	*	MIN	MAX	COUNT	MINIMUM	MAXIMUM	SUM
1.0	**1.0**	**1.0**	.7	.0	.1	.0	.4	.0	.0	.1	.0	.2	.1	.0	.0	.0	.0

DFS + Sort and add + …

$\mathtt{a} \leftarrow \mathtt{[int]}$

$\mathtt{b} \leftarrow \textsc{Filter}\ \mathtt{(<0)\ a}$

$\mathtt{c} \leftarrow \textsc{Map}\ \mathtt{(*4)\ b}$

$\mathtt{d} \leftarrow \textsc{Sort}\ \mathtt{c}$

$\mathtt{e} \leftarrow \textsc{Reverse}\ \mathtt{d}$

Other Examples of Exploiting Spec

```
Anthony Hopkins
Al Pacino
Tom Hanks
Tom Hanks
Nicolas Cage
```

$\rightarrow$

```
Anthony Hopkins (1)
Al Pacino (1)
Tom Hanks (2)
Nicolas Cage (1)
```

Syntactic features of I/O examples

Feature	Answ
Duplicated lines in input but not output?	Y
Parentheses in output but not input?	Y
Numbers on each line in output but not input?	Y
…	

Infer PCFG

Production	Probability	Production	Probability
P→join(LIST,DELIM)	1	CAT→LIST	0.7
LIST→split(x,DELIM)	0.3	CAT→DELIM	0.3
LIST→concatList(CAT,CAT,CAT)	0.1	DELIM→"\n"	0.5
LIST→concatList("(",CAT,")")	0.2	DELIM→" "	0.3
LIST→dedup(LIST)	0.2	DELIM→"("	0.1
LIST→count(LIST,LIST)	0.2	DELIM→")"	0.1

Enumeration in order of decreasing probability

Solution

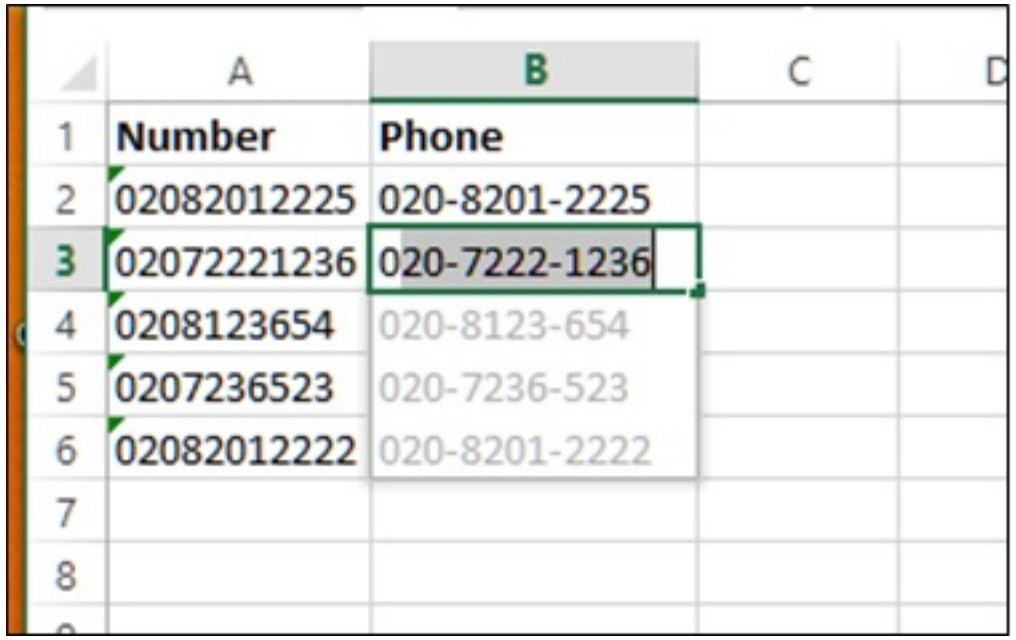
$$f(x) = \texttt{dedup}(\texttt{concatLists}(x, \text{“ ”}, \texttt{concatLists}(\text{“(”}, \texttt{count}(x, x), \text{“)”}))).$$

Menon et al., A Machine Learning Framework for Programming by Example

Evaluation

- Benchmarks
  - **1,167** problems used in SyGuS annual competitions
- Baselines
  - **EUSolver** (general-purpose): winner of 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
```

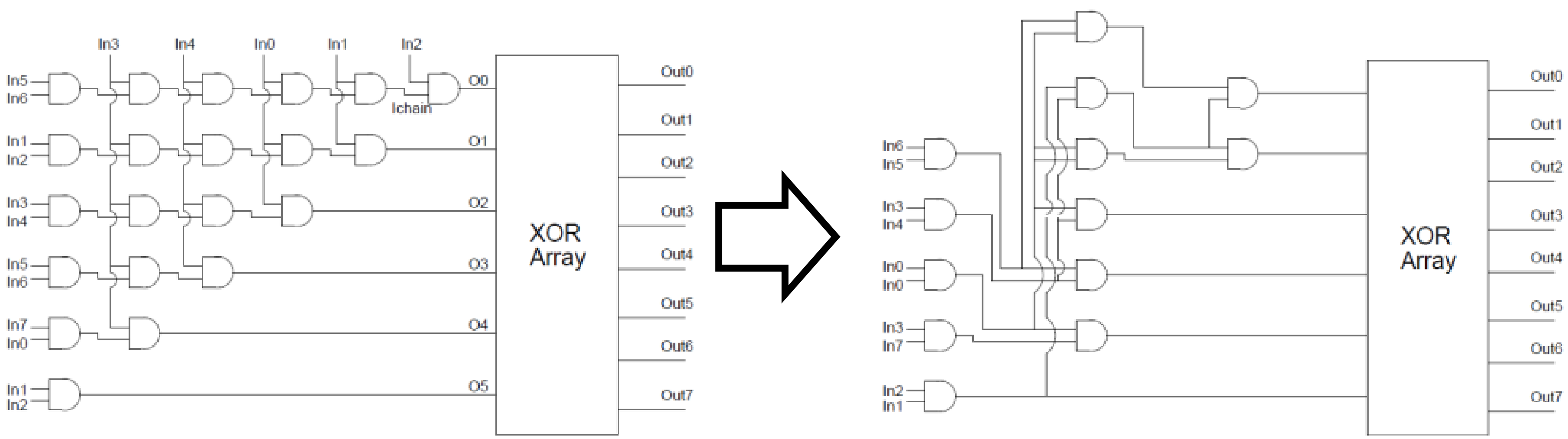
bitwise or

```
  01010001110101110000000000001111
| 00110001011011100011000101101110
  01110001111111110011000101101111
```

bitwise xor

```
  01010001110101110000000000001111
^ 00110001011011100011000101101110
  01100000101110010011000101100001
```

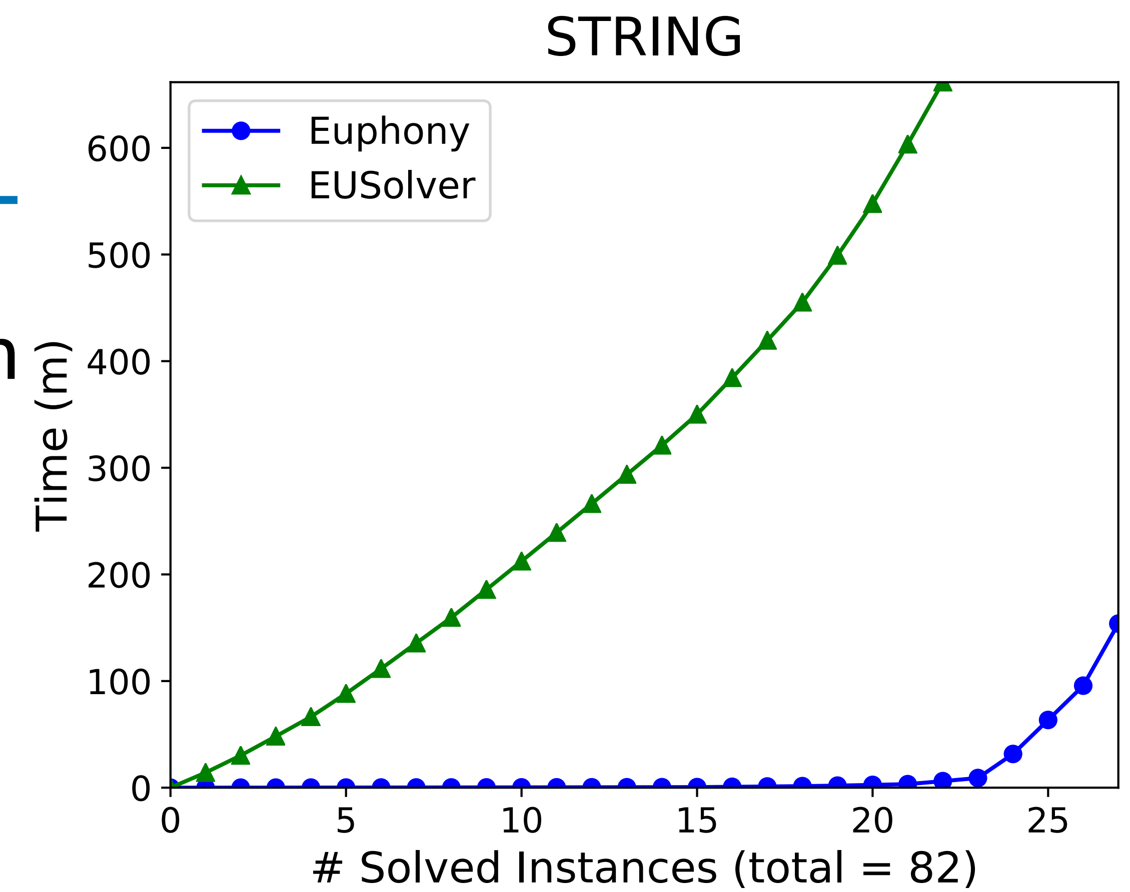
BITVEC: Efficient low-level algorithm
750 problems

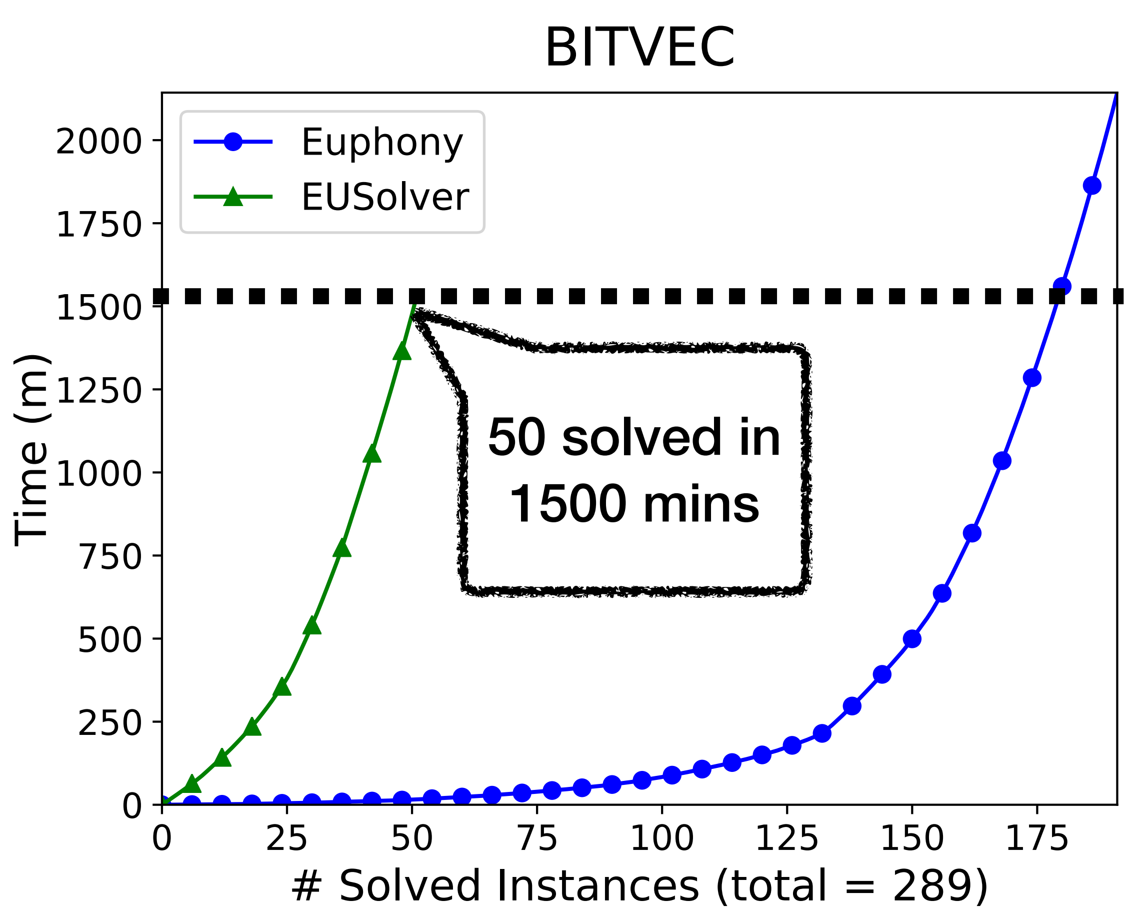


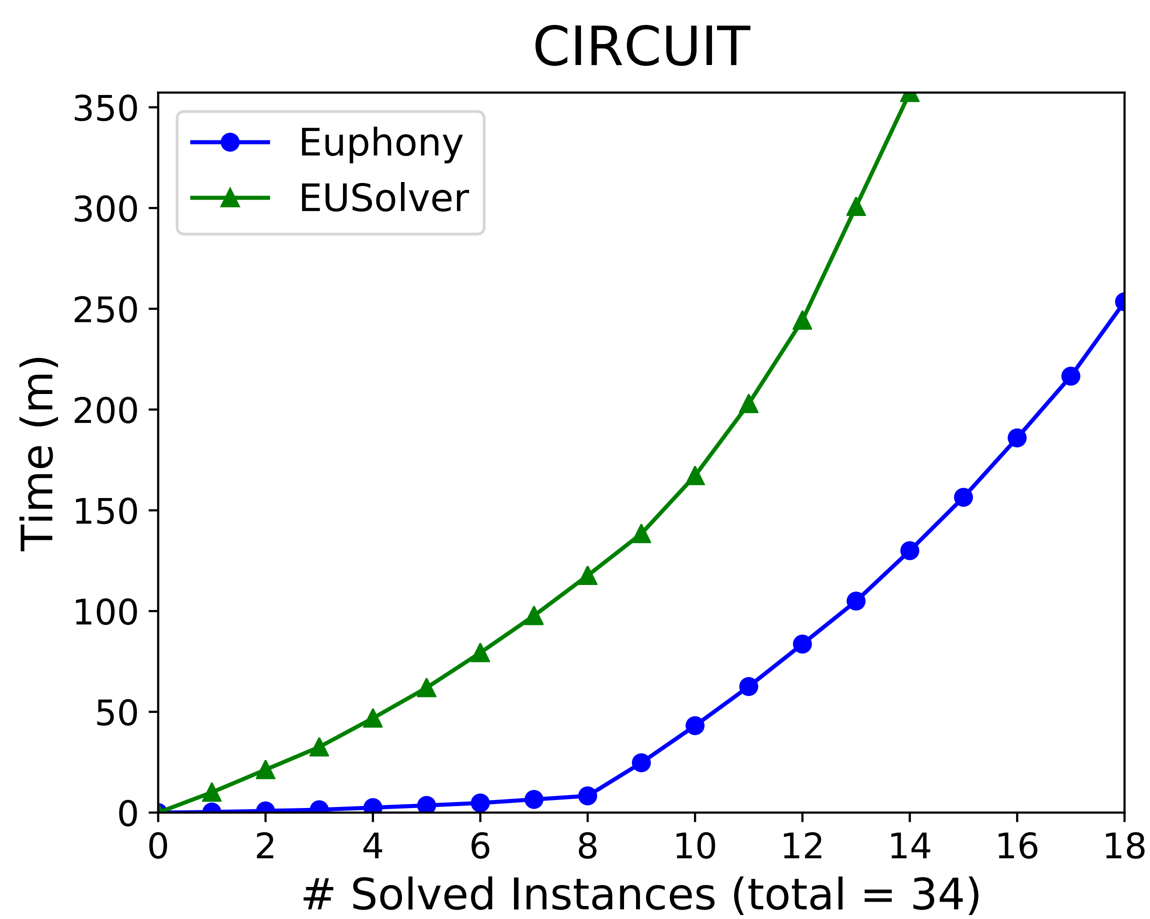
CIRCUIT: Attack-resistant crypto circuits
212 problems

vs. EUSolver

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



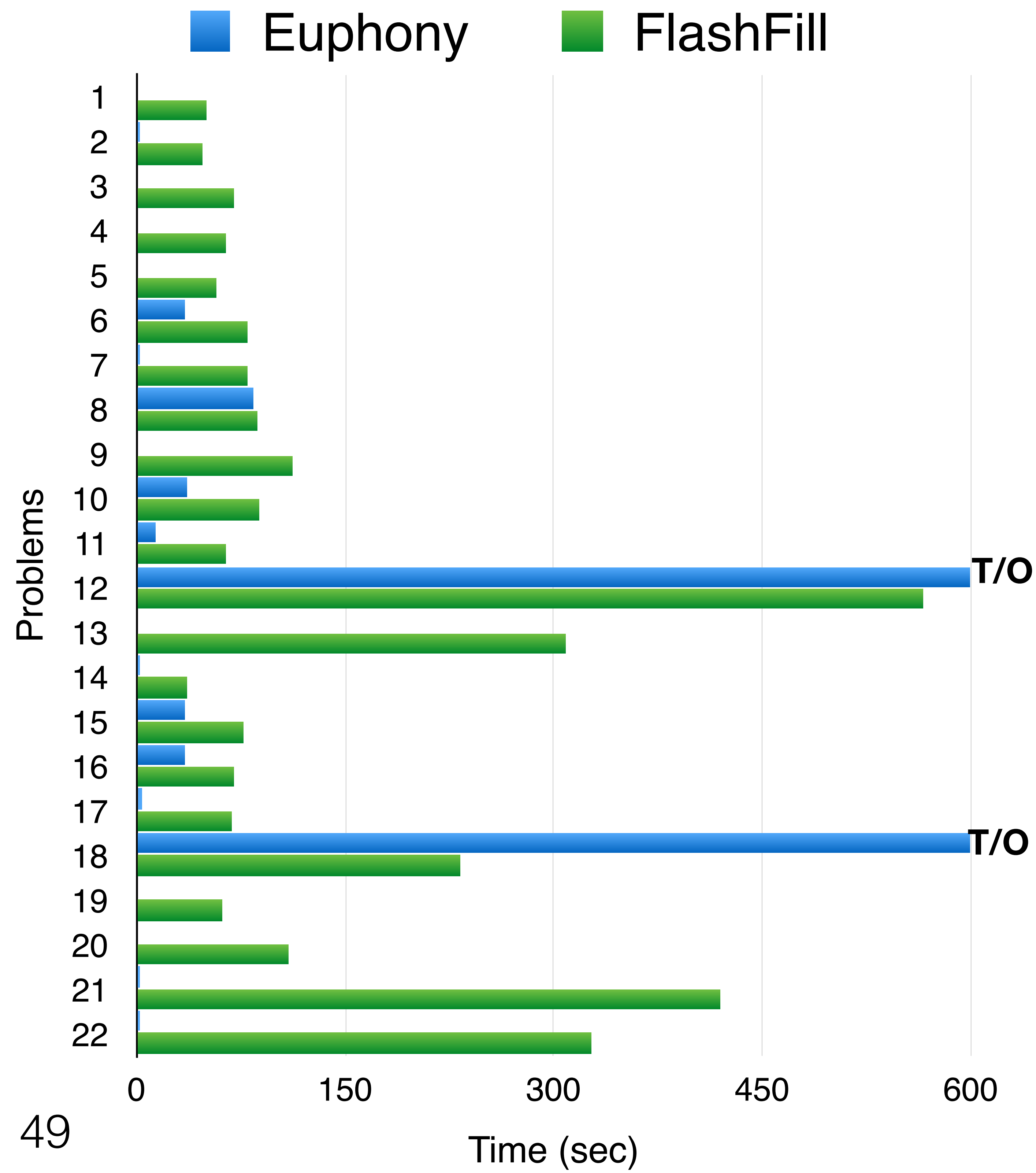




vs. FlashFill

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

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



Efficacy of A* and PHOG

- Using PCFG and PHOG [Bielik et al. ICML'16]

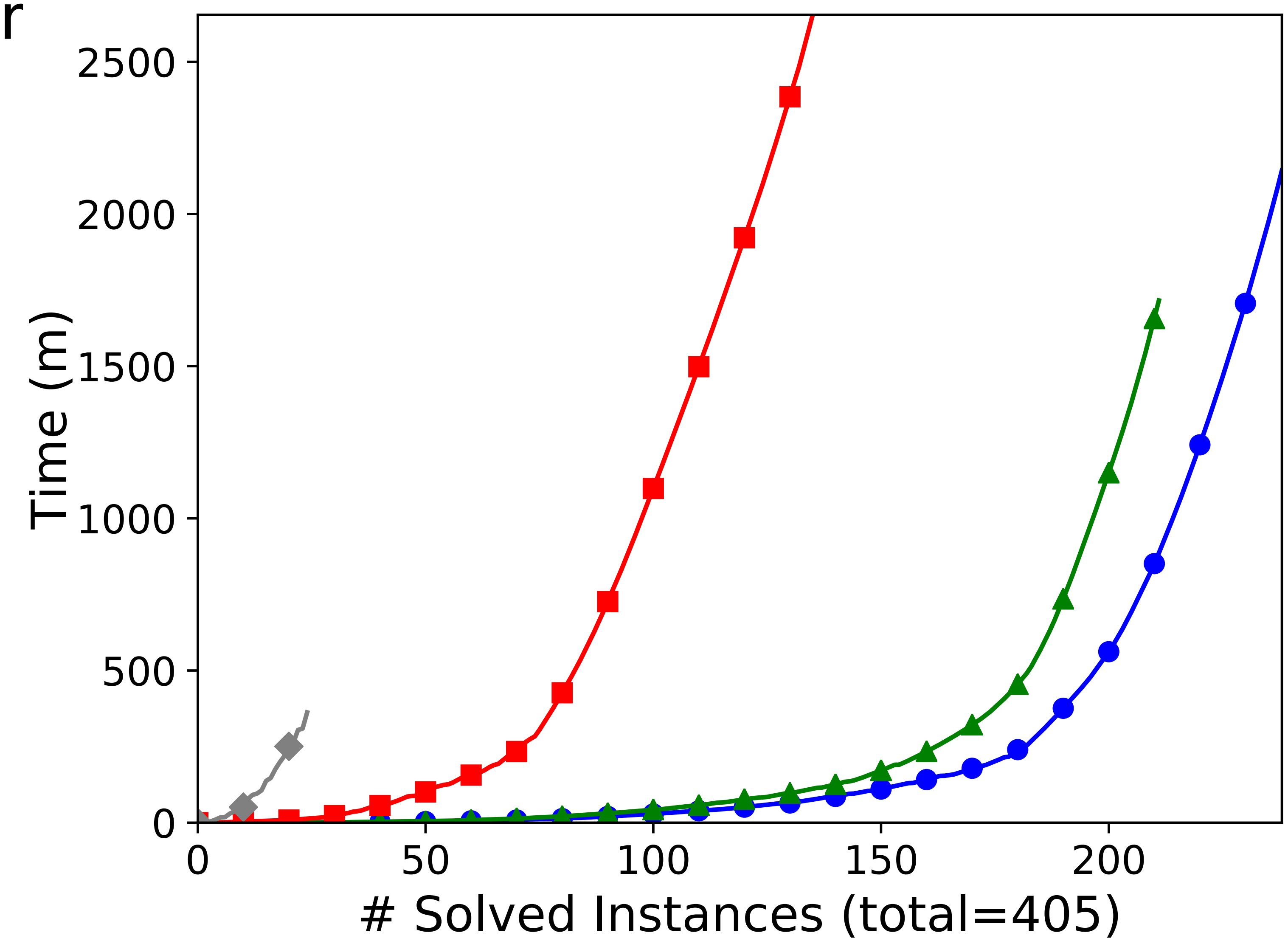
- # Solved (timout: 1 hour

  A* + PHOG: **236**

  Dijkstra + PHOG: **209**

  A* + PCFG: **133**

  Dijkstra + PCFG: **22**



What about Bottom-Up Search?

- Bottom-up enumeration in order of decreasing probability instead of increasing size

- Cannot consider contexts (why?)

- Use PCFGs, which are statistical models which do not consider contexts

    - Such PCFGs can be obtained from existing code, or just-in-time learning (probabilities of production rules keep changed during search).

Barke et al., Just-in-Time Learning for Bottom-Up Enumerative Synthesis, OOPSLA 2020

Guided Bottom-Up Search

$BottomUp(\text{PCFG } G_P, \text{specification } \phi)$

$P \leftarrow$ set of all terminals in G

$\text{cost} \leftarrow 0$

cost = -log (probability of program according to PCFG)

while True **do**

$P \leftarrow \mathsf{EnumerateExprs}(G_P, P, \text{cost})$

Generate candidate programs of target cost

$P \leftarrow \{p' \in P \mid \forall p \in P.\ \neg\textsc{Equiv}(\phi, p, p')\}$

foreach $p \in P$

if $\phi(p)$ **then return** p

$\text{cost} \leftarrow \text{cost} + 1$

Target cost is increased

Guided Bottom-Up Search (cont.)

$$\text{EnumerateExprs}(\mathrm{PCFG}\ G_P, P, \mathrm{cost})$$

$$P' := P$$

$$\textbf{for}\ N \to f(N_1, \cdots, N_k) \in R$$

For each production rule

$$P' := P \cup \{f(p_1, \cdots, p_k) \mid \forall i.\ N_i \Rightarrow^* p_i, -\log Pr(f(p_1, \cdots, p_k)) = \mathrm{cost}\}$$

p_i is derivable from N_i

$$\textbf{return}\ P'$$