Bidirectional Search-Based Synthesis

Woosuk Lee

CSE9116 SPRING 2024

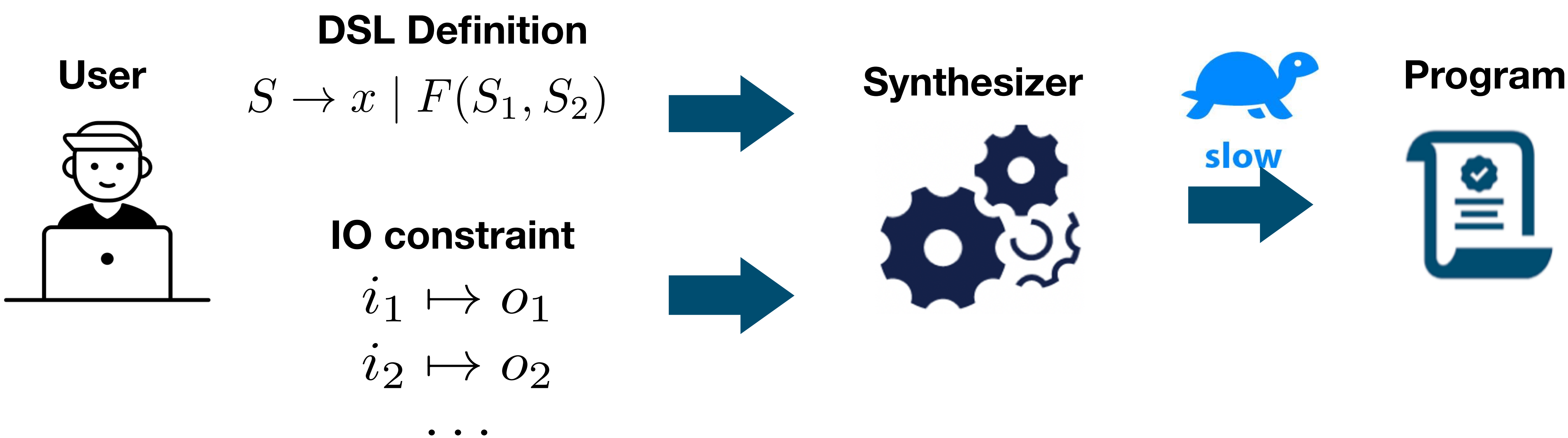
Hanyang University

Top-Down vs. Bottom-Up

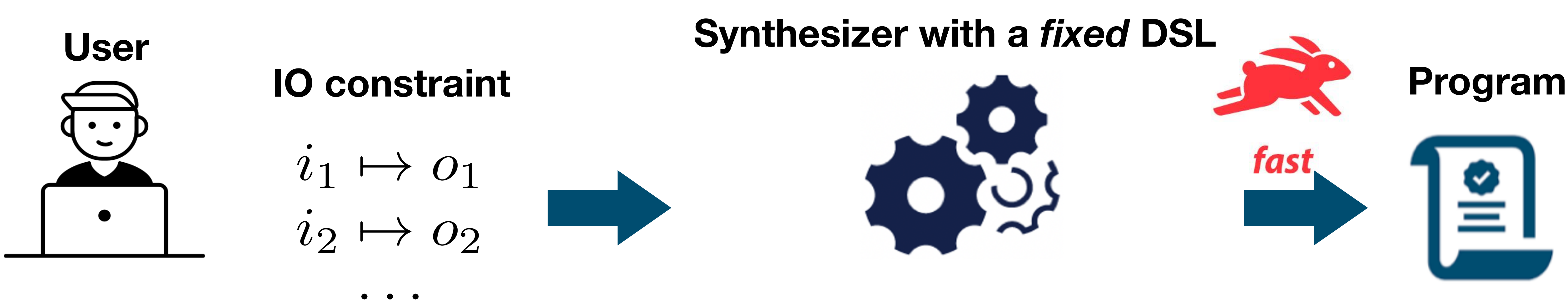
- **Top-down search**
  - Candidates are whole but might be incomplete (w/ holes); cannot always run on inputs but always relate to outputs
  - Optimization: *top-down propagation*
    - Efficient but only applicable for specific kinds of languages
- **Bottom-up search**
  - Candidates are complete programs but might not be whole; can always run on inputs but cannot always relate to outputs
  - Optimization: *observational equivalence reduction*
    - Generally applicable but inefficient

Dichotomy : "General-purpose" vs. "Domain-specific"

General-purpose synthesizer

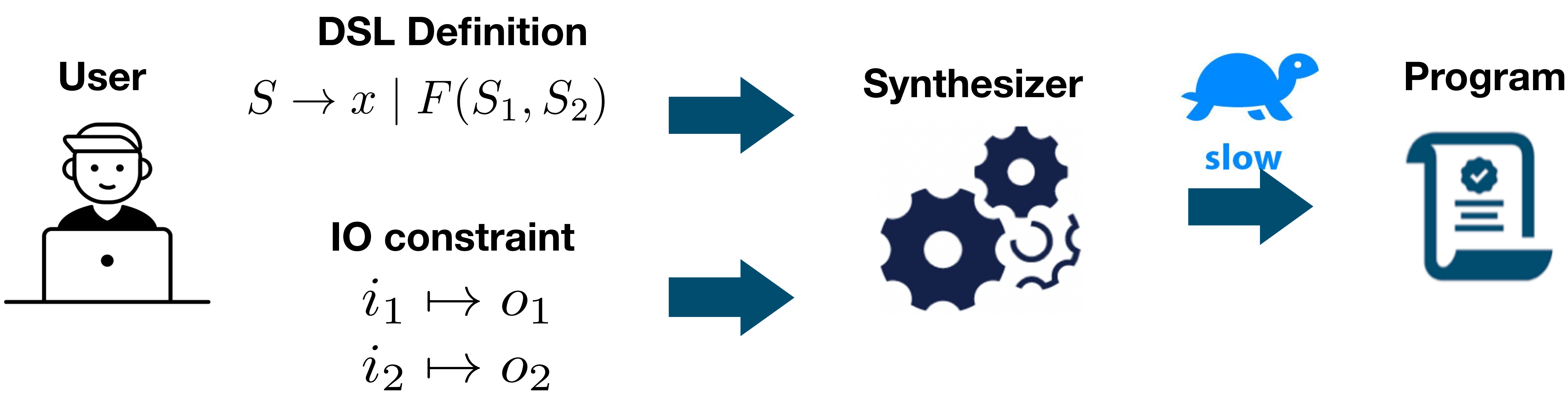


Domain-specific synthesizer



Dichotomy : "General-purpose" vs. "Domain-specific"

General-purpose synthesizer



- Encouraged by a standard formulation: Syntax-guided Synthesis (SyGuS)
- General search strategies (e.g., ***Bottom-up enumeration***)

+ Broad application domains − unscalable

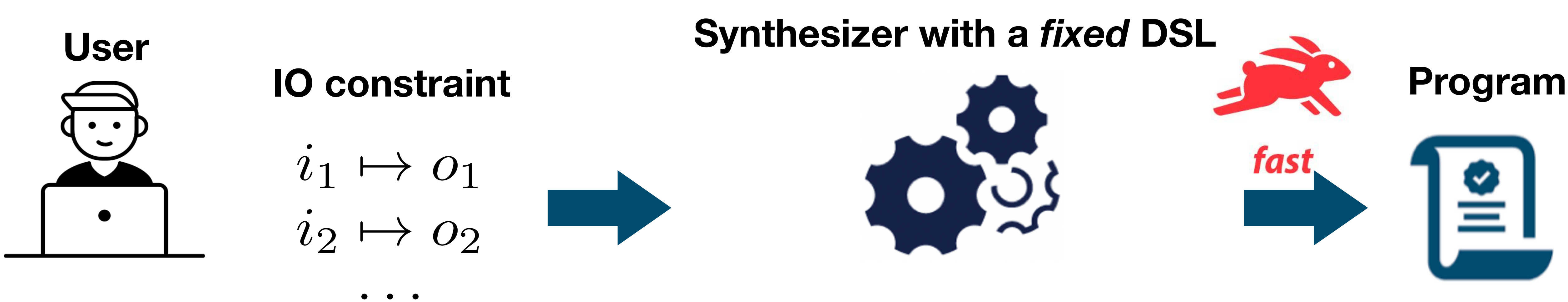
Dichotomy : "General-purpose" vs. "Domain-specific"

- Domain-specific search strategies (e.g., ***Top-down propagation***)
- Successful industrialization (e.g.,)

+ very efficient − only applicable for specific applications

. . .

Domain-specific synthesizer

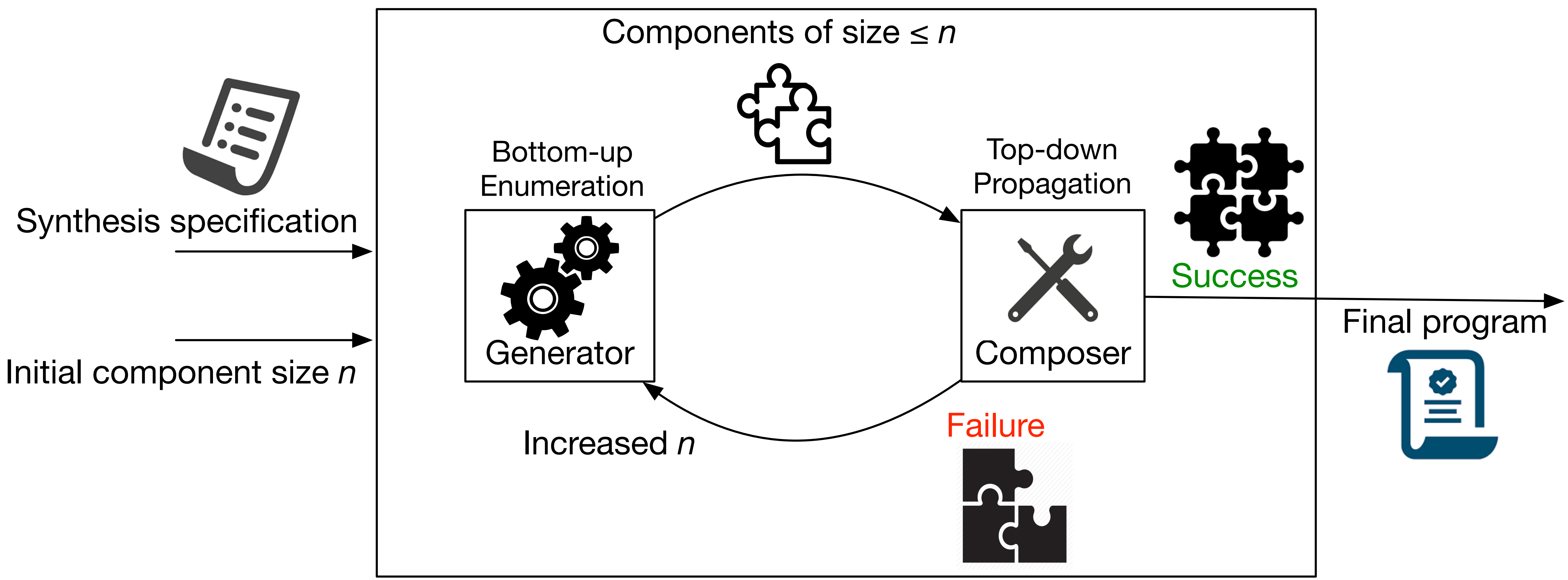


A Bidirectional Search Strategy

- Woosuk Lee, Combining the Top-down Propagation and Bottom-up Enumeration for Inductive Program Synthesis, POPL 2021
- Synergistically combine top-down and bottom-up search
- Generally applicable for a wide range of SyGuS instances
- https://github.com/wslee/duet

Overall Algorithm

Domain-**U**naware inductive synthesis via **E**numeration and **T**op-down propagation



Example

- Goal: function f converting a given phone number x into another format
- Specification

Syntactic:

$$
\begin{aligned}
S \;&\rightarrow\; x \mid \text{“+”} \mid \text{“␣”} \mid \text{“-”} \mid \text{“.”} \mid \mathrm{ConCat}(S, S) \\
&\mid\; \mathrm{SubStr}(S, I, I) \\
&\mid\; \mathrm{IntToStr}(I) \\
&\mid\; \mathrm{Replace}(S, S, S) \\
I \;&\rightarrow\; 1 \mid \cdots \mid 9 \mid I - I \mid \mathrm{Length}(S)
\end{aligned}
$$

String concatenation

Semantic:

$$f(\text{“␣+1␣2.3”}) = \text{“1␣2-3.”} \wedge f(\text{“␣+4␣56.78”}) = \text{“4␣56-78.”}$$

Example

- Goal: function f converting a given phone number x into another format
- S

$\mathrm{SubStr}(str, start\_pos, len)$

e.g., $\mathrm{SubStr}(\text{“}abc\text{”}, 0, 2) = \text{“}ab\text{”}$

Sy

$$\text{“-”} \mid \text{“.”} \mid \mathrm{ConCat}(S, S)$$
$$\mid \quad \mathrm{SubStr}(S, I, I)$$
$$\mid \quad \mathrm{IntToStr}(I)$$
$$\mid \quad \mathrm{Replace}(S, S, S)$$
$$I \quad \rightarrow \quad 1 \mid \cdots \mid 9 \mid I - I \mid \mathrm{Length}(S)$$

Semantic:

$$f(\text{“}\llcorner + 1\llcorner 2.3\text{”}) = \text{“}1\llcorner 2\text{-}3.\text{”} \wedge f(\text{“}\llcorner + 4\llcorner 56.78\text{”}) = \text{“}4\llcorner 56\text{-}78.\text{”}$$

Example

- Goal: function f converting a given phone number x into another format
- Specification

Syntactic: $S \rightarrow \quad \ldots S, S)$

$\quad |$

$\quad | \quad \mathrm{IntToStr}(I)$

$\quad | \quad \mathrm{Replace}(S, S, S)$

$I \rightarrow 1 \mid \cdots \mid 9 \mid I - I \mid \mathrm{Length}(S)$

> $\mathrm{Replace}(str, match, replacement)$
>
> e.g., $\mathrm{Replace}(\text{“}aba\text{”}, \text{“}a\text{”}, \text{“}b\text{”}) = \text{“}bba\text{”}$

Semantic:

$$f(\text{“}\_ + 1\_2.3\text{”}) = \text{“}1\_2\text{-}3.\text{”} \wedge f(\text{“}\_ + 4\_56.78\text{”}) = \text{“}4\_56\text{-}78.\text{”}$$

Example

- Solution: **Size : 12 AST nodes**

$$\underbrace{\text{Replace}(\underbrace{\text{SubStr}(\underbrace{\text{ConCat}(x, \text{``.''})}_{\text{outputs: } \langle \text{``\textvisiblespace+1\textvisiblespace 2.3.''}, \text{``\textvisiblespace+4\textvisiblespace 56.78.''} \rangle}, 2, \text{Length}(x) - 1)}_{\text{outputs: } \langle \text{``1\textvisiblespace 2.3.''}, \text{``4\textvisiblespace 56.78.''} \rangle}, \text{``.''}, \text{``-''})}_{\text{outputs: } \langle \text{``1\textvisiblespace 2}-\text{3.''}, \text{``4\textvisiblespace 56}-\text{78.''} \rangle}$$

Example

- Solution:

	Applicable?	Efficient?
Bottom-up Enumeration	O	X
Top-down Propagation	X	—
Duet	O	O

Existing Bottom-up Enumerative Strategy

- Enumerate expressions in order of increasing size
- Put smaller expressions together into larger ones

Size	**Expressions**
1	x "+" "␣" "-" "." $1 \cdots 9$
2	Length(x) Length("␣") Length("-") Length(".")
3	$1-2$ $2-1$ ConCat(x, ␣) $\cdots$ SubStr(x, 0, 1) $\cdots$
4	
…	

Existing Bottom-up Enumerative Strategy

Optimization: maintain only semantically unique expressions

Size	**Expressions**
1	x "+" "␣" "-" "." 1 ⋯ 9
2	Length(x) ~~Length("␣")~~ ~~Length("-")~~ ~~Length(".")~~
3	$1 - 2$ ~~$2 - 1$~~ ConCat(x, ␣) ⋯ ~~SubStr(x, 0, 1)~~ ⋯
4	
⋯	

+ Generally applicable − Limited scalability

Existing Bottom-up Enumerative Strategy

Optimization: maintain only semantically unique expressions

Size **Expressions**

The solution is too large to be quickly found by Bottom-up enumeration.

…

…

+ Generally applicable − Limited scalability

Top-Down Propagation

- **Divide-and-conquer**: if a program $F(e_1, \cdots, e_k)$ outputs o on some input, what should $e_1, \cdots, e_k$ output on the same input?

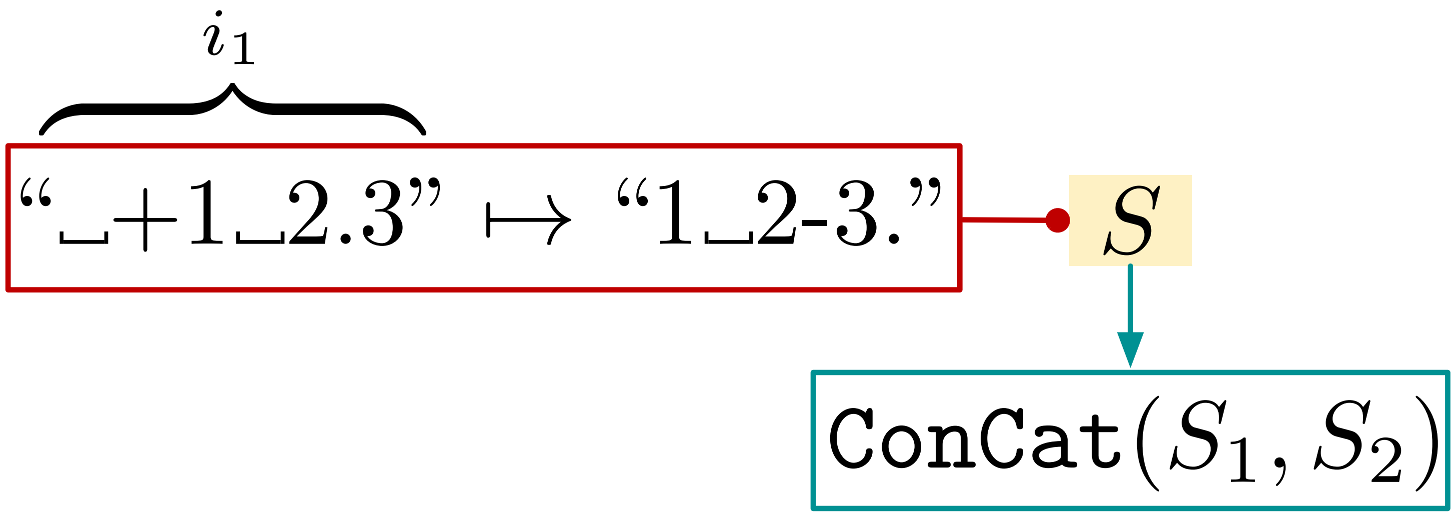
- infer specs for subexpressions using ***inverse functions***

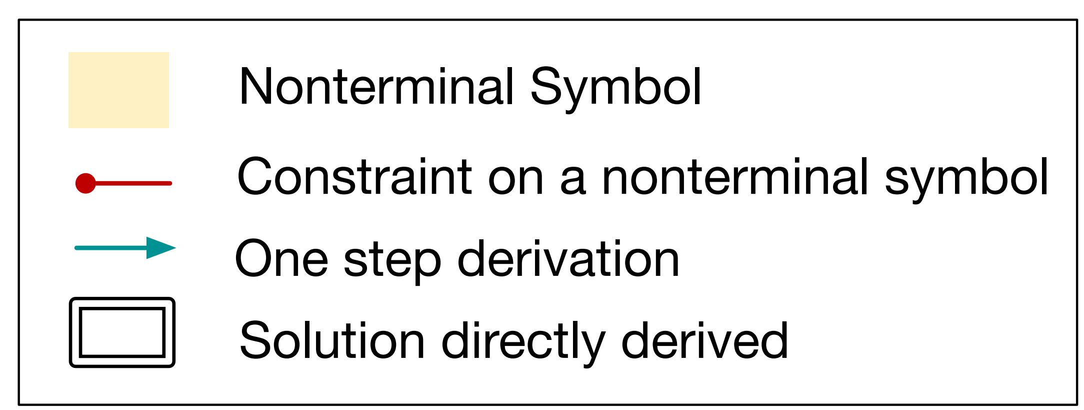
$$F^{-1}(o) = \{(a_1, \cdots, a_k) \mid F(a_1, \cdots, a_k) = o\}$$

- *Inverse set (or pre-image):*

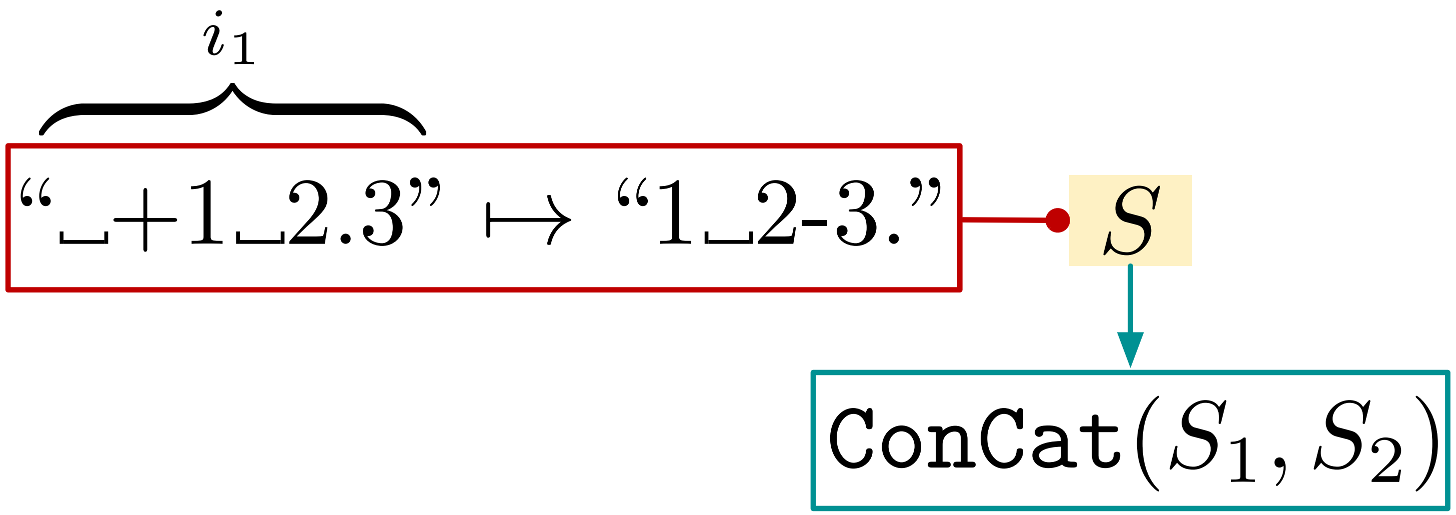
e.g., $\mathrm{ConCat}^{-1}(\text{“}USA\text{”}) = \{(\text{“}U\text{”}, \text{“}SA\text{”}), (\text{“}US\text{”}, \text{“}A\text{”})\}$

Top-Down Propagation

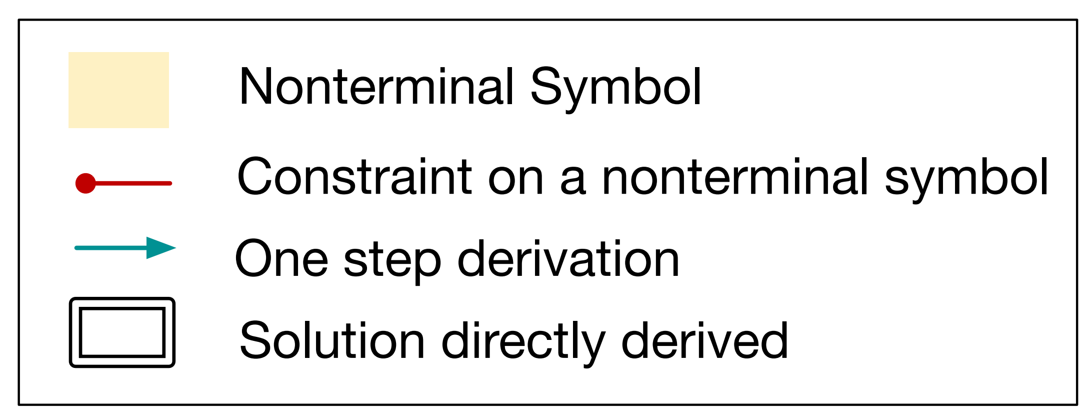




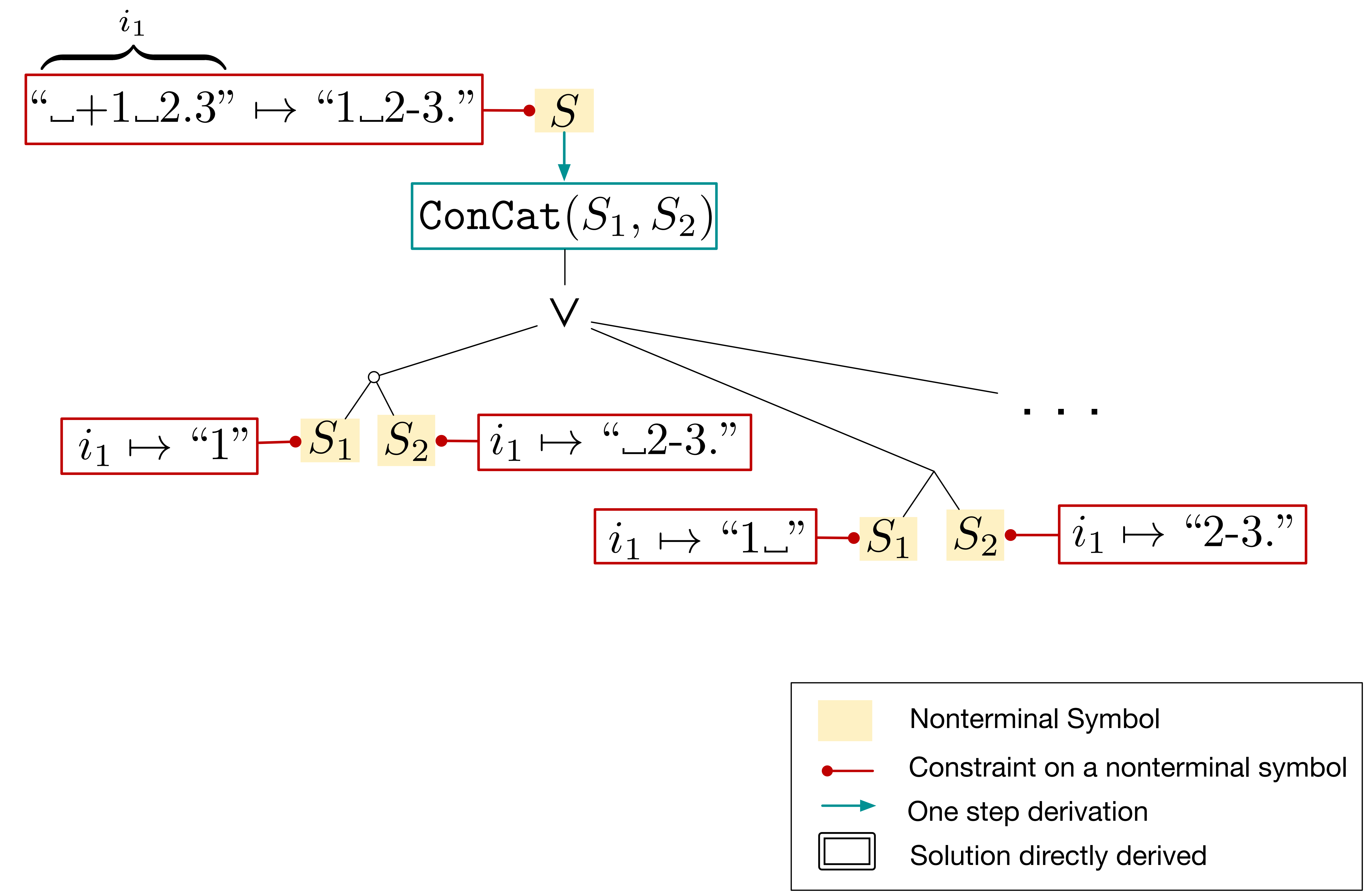
Top-Down Propagation



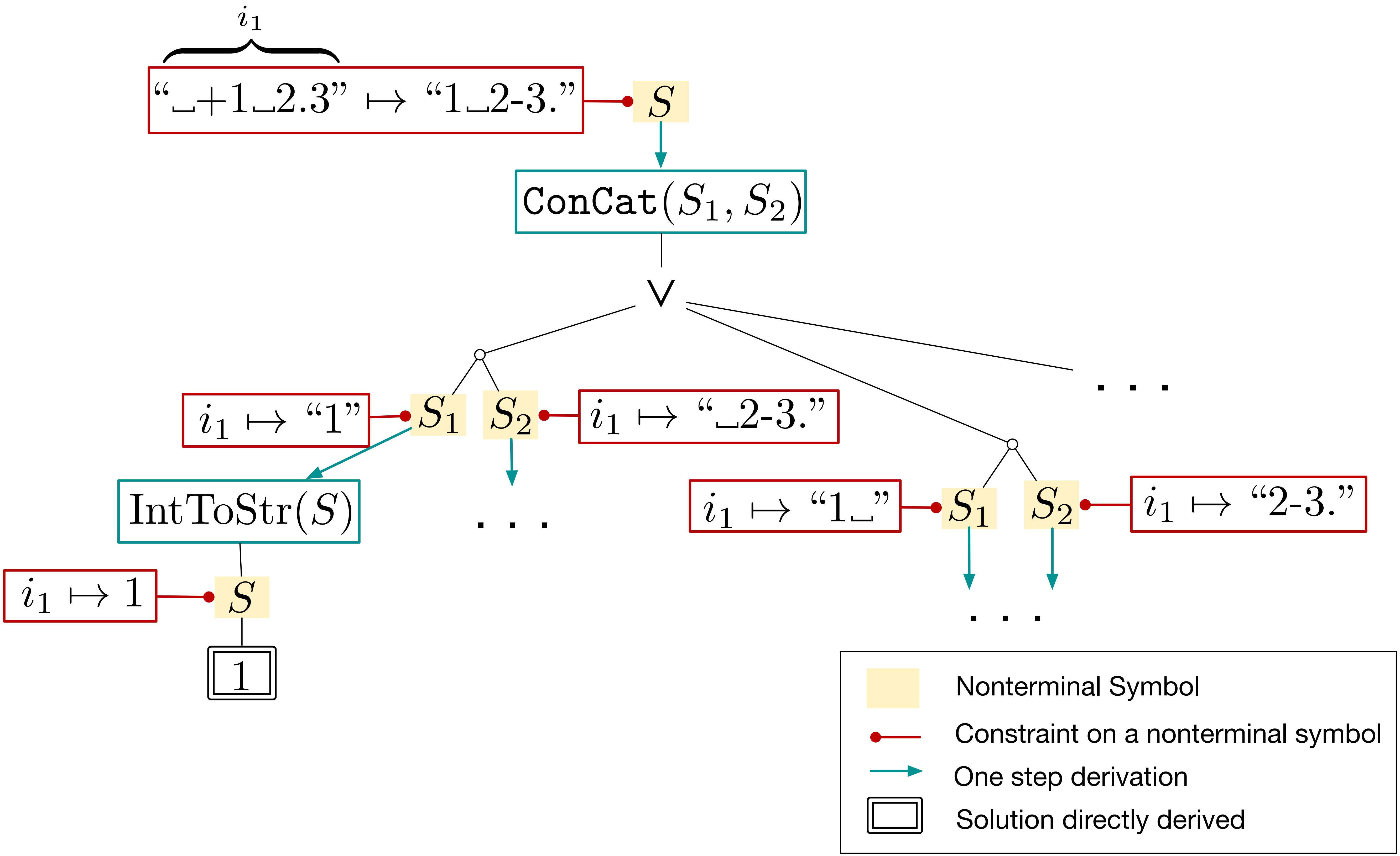
$$\text{ConCat}^{-1}(\text{“1␣2-3.”}) = \{(\text{“1”}, \text{“␣2-3.}), (\text{“1␣”}, \text{“2-3.”}), \cdots\}$$



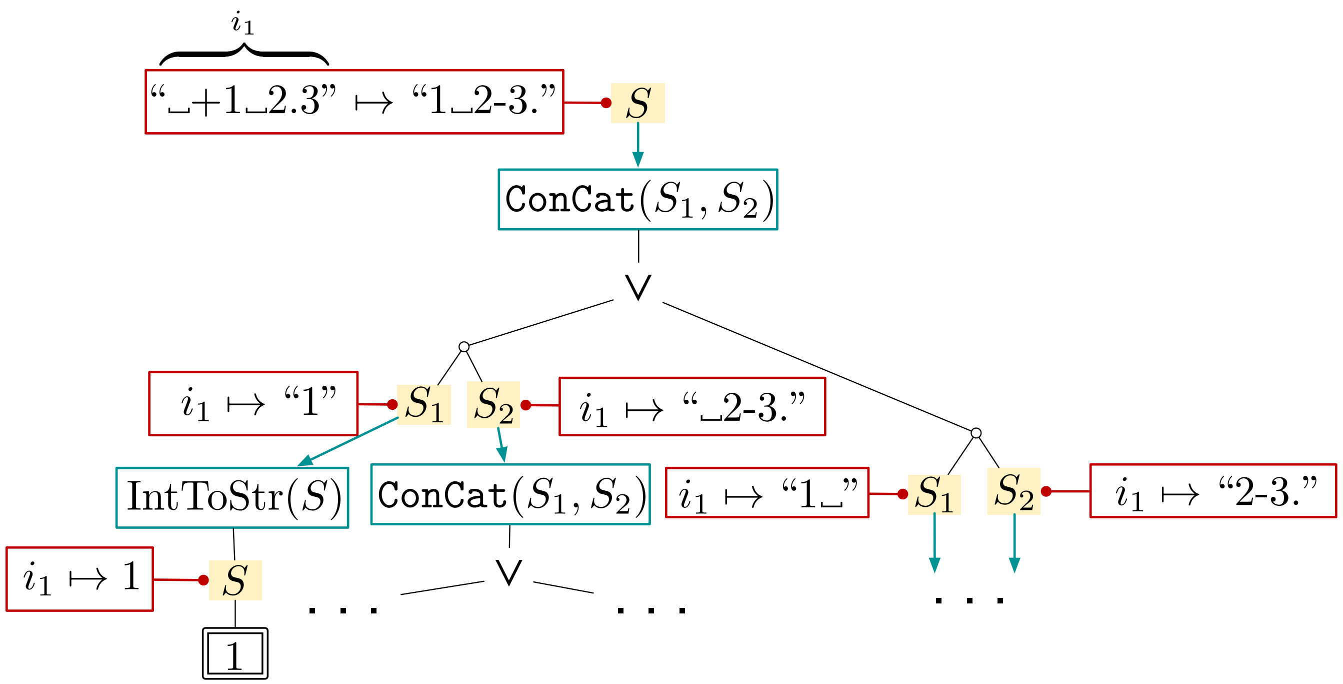
Top-Down Propagation

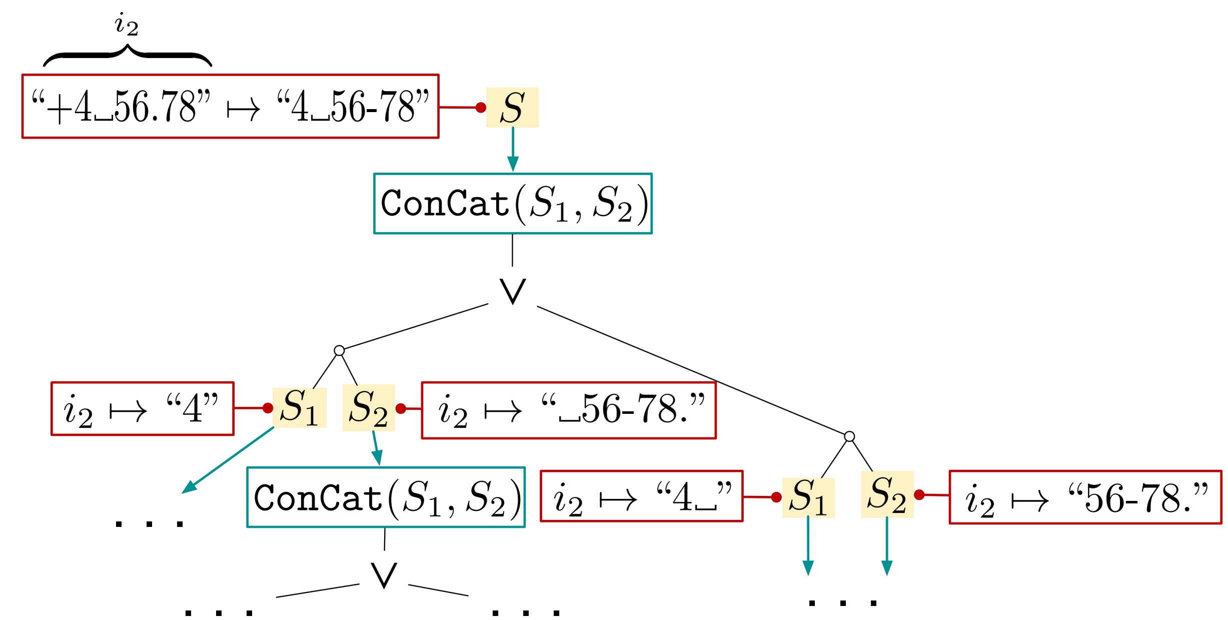


Top-Down Propagation



Top-Down Propagation





All possible solutions for IO example 1

All possible solutions for IO example 2

{ConCat(ConCat(IntToStr(1), "␣"), ⋯),
ConCat(ConCat(IntToStr(1), ConCat("␣" ⋯),
⋯ }

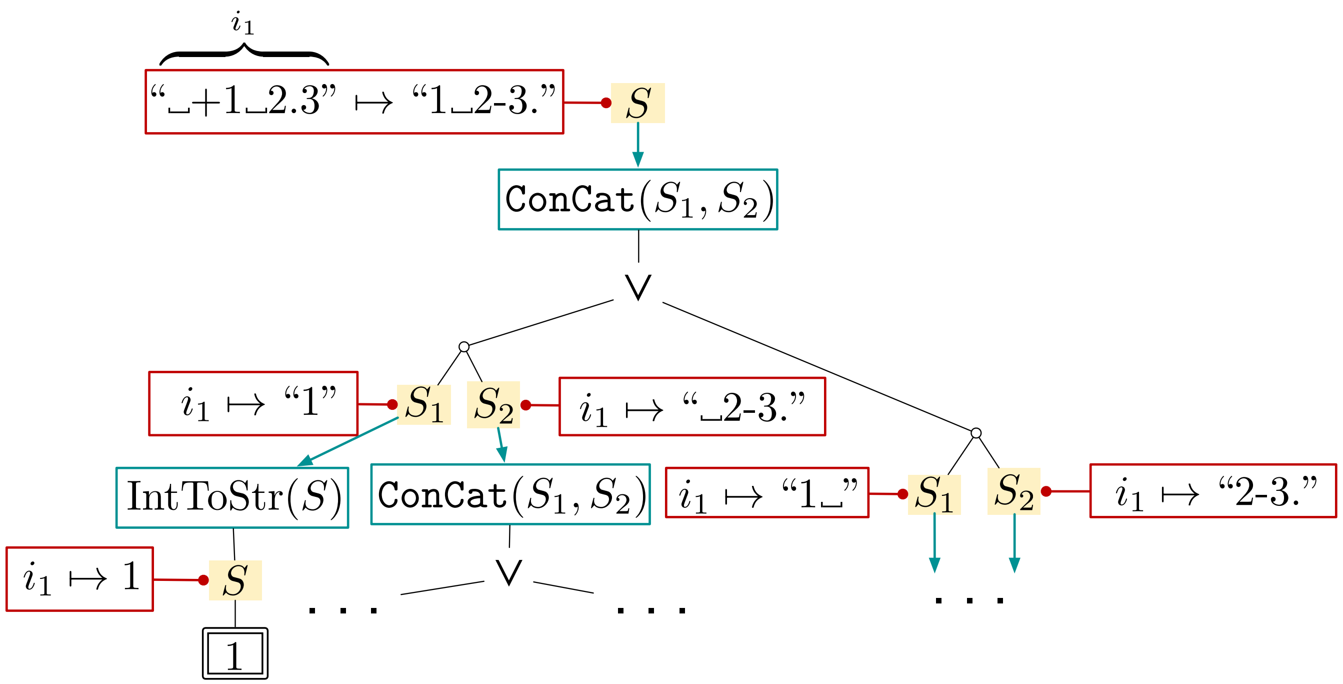
$\cap$

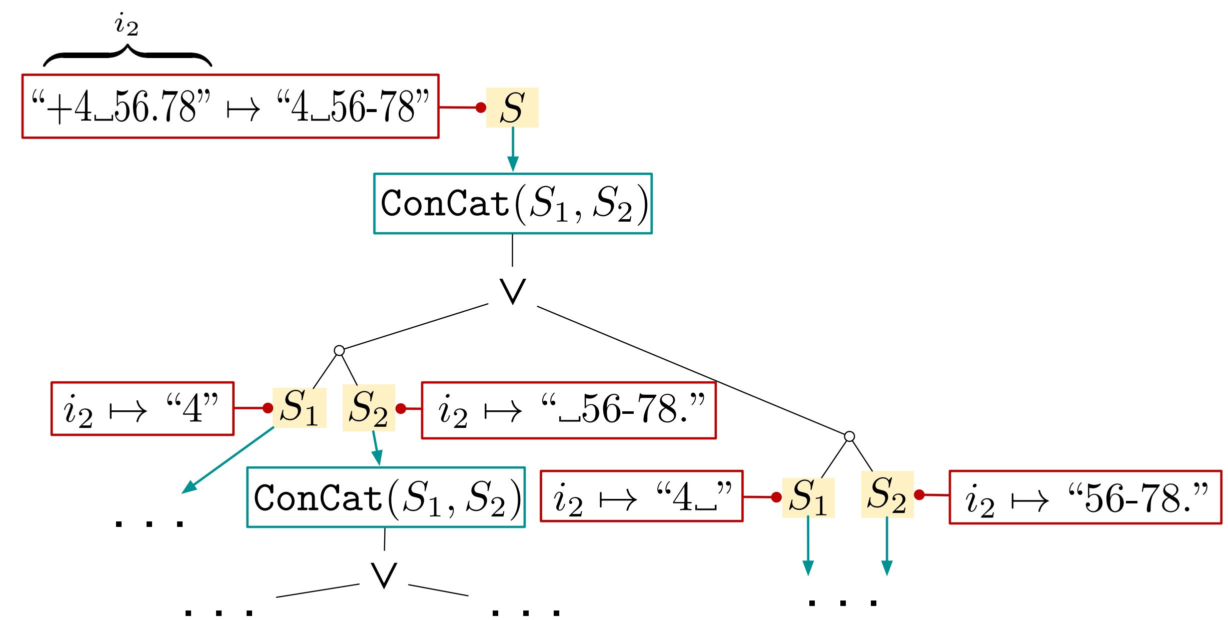
{ConCat(ConCat(IntToStr(4), "␣"), ⋯),
ConCat(ConCat(IntToStr(4), ConCat("␣" ⋯),
⋯ }

stored in a space-efficient data structure

All possible final solutions

Top-Down Propagation





All possible solutions for IO example 1

All possible solutions for IO example 2

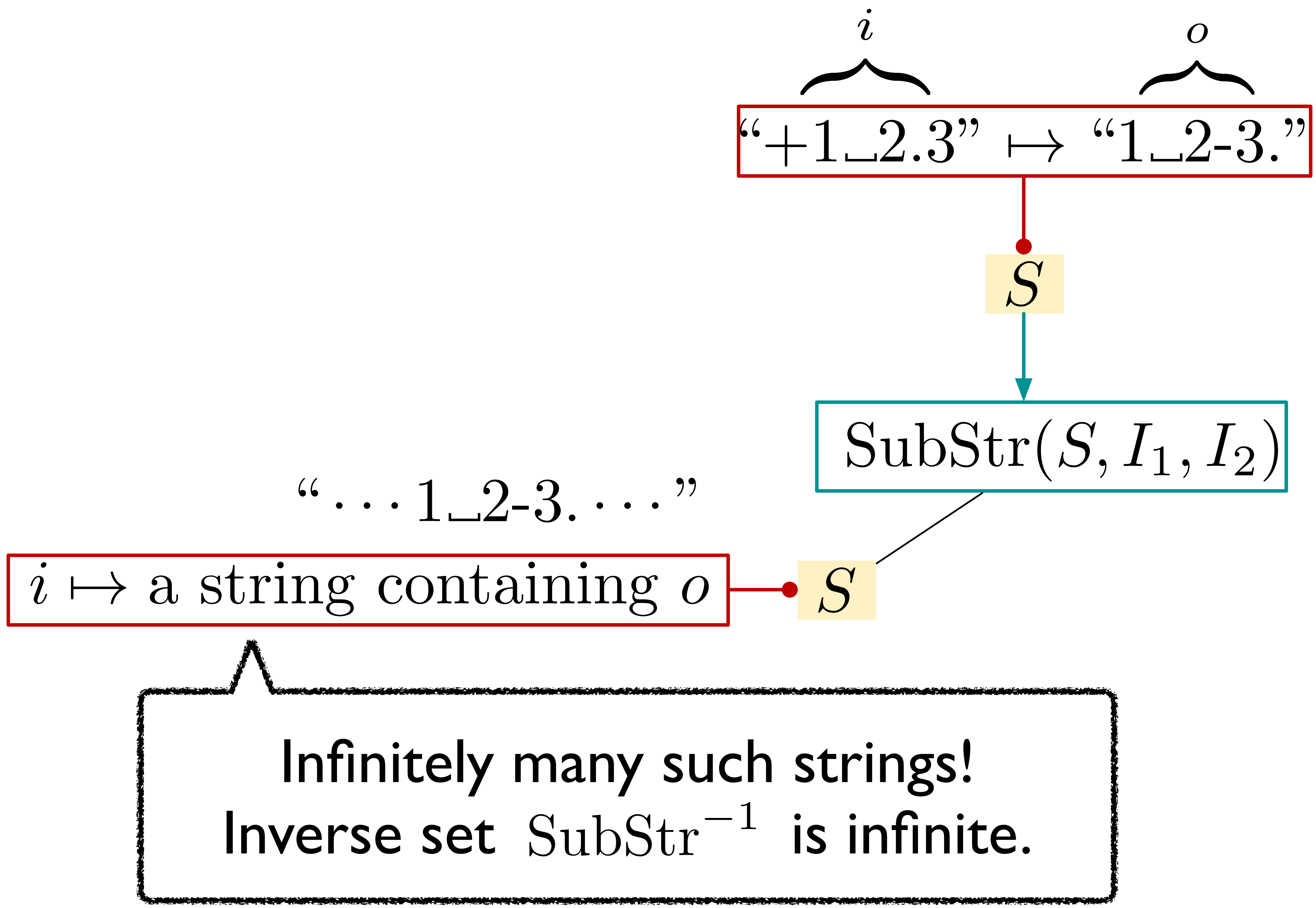
+ very efficient (divide-and-conquer, goal-directed)

− # of IO examples ↑ → performance ↓

− **Not always applicable**

Top-Down Propagation

Problem: ***unconstrained arguments***



Top-Down Propagation

Problem: ***unconstrained arguments***

$$\overbrace{\text{“+1␣2.3”}}^{i} \mapsto \overbrace{\text{“1␣2-3.”}}^{o}$$

Top-down propagation is not even applicable due to the general grammar.

Infinitely many such strings!
Inverse set SubStr^{-1} is infinite.

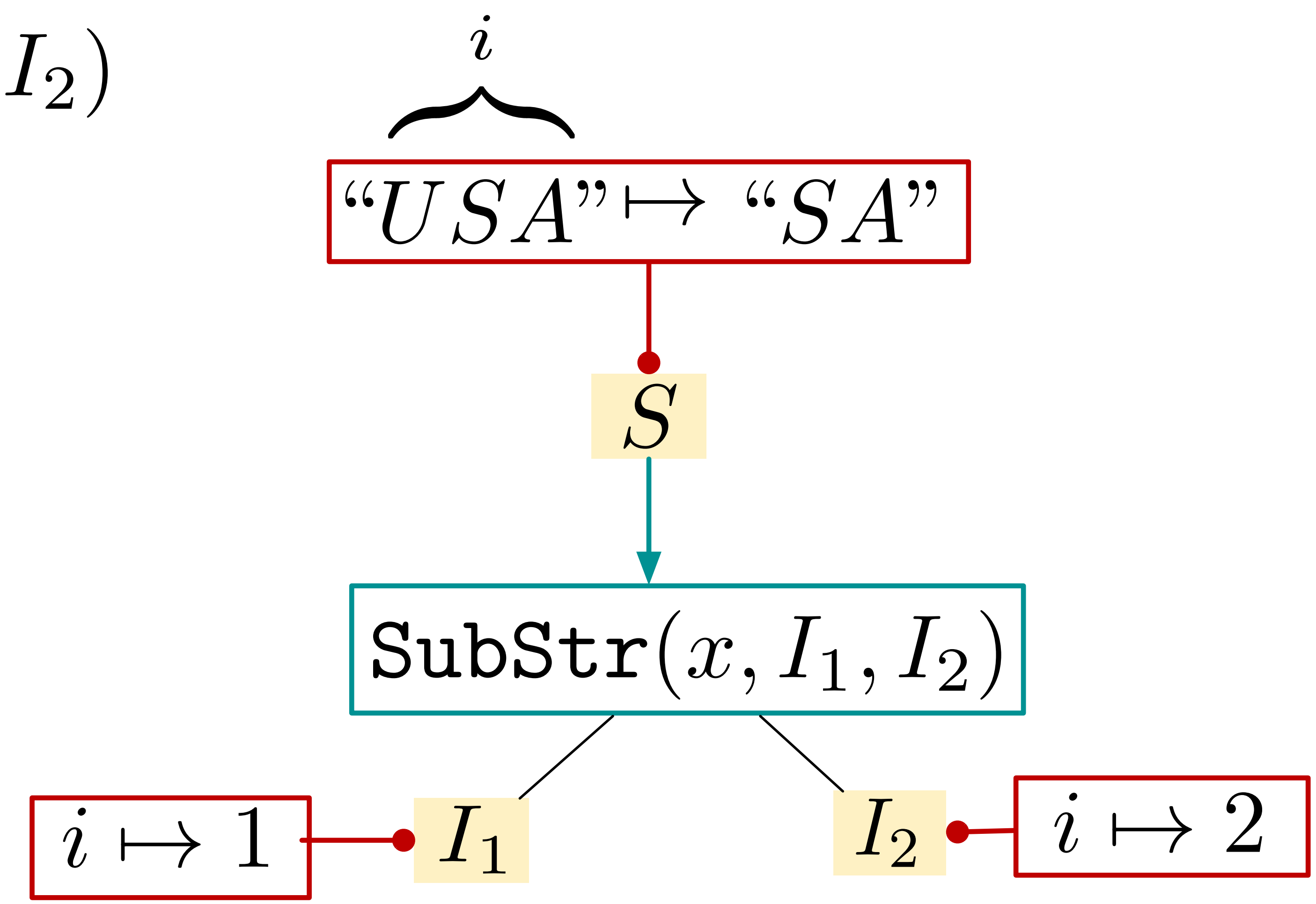
Top-Down Propagation

Previous incomplete workaround: *restricting* the grammar

(which limits the kinds of programs that can be synthesized)

$$S \rightarrow \cdots \mid \text{SubStr}(x, I_1, I_2)$$

Input example



Top-Down Propagation

Flash Fill DSL

$$Tuple(String\ x_1, \ldots, String\ x_n) \rightarrow String$$

top-level expr T := C | $ifThenElse(B, C, T)$

condition-free expr C := A | $Concat(A, C)$

atomic expression A := $SubStr(X, P, P)$ | $ConstantString$

input string X := x_1 | x_2 | …

position expression P := K | $Pos(X, R_1, R_2, K)$

Slide from "Programming by Examples: PL meets ML"

Only for DSLs with *Balanced Expressivity*

"DSL design = Art + *Lots* of iterations"

"The DSL should be *expressive enough* to represent various tasks… and *restricted enough* to allow efficient search."

Gulwani et al., Program synthesis
Polozov et al., FlashMeta: a framework for inductive program synthesis
Gulwani et al., Programming by example (and its application to data wrangling)

Key Idea

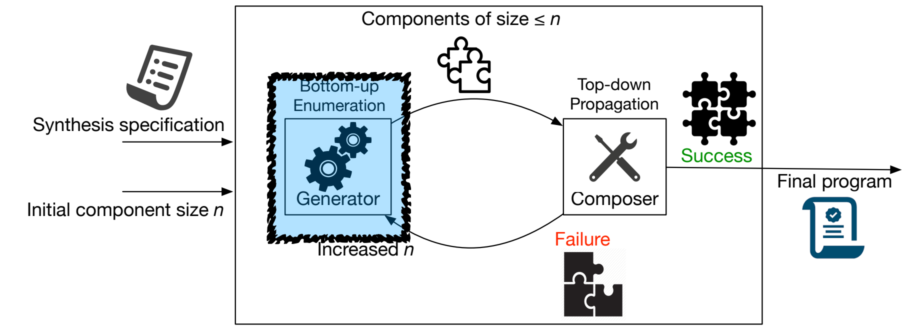
- Bottom-up enumeration can quickly identify *small* but *important* subexpressions of the solution.

$$\text{Replace}(\text{SubStr}(\boxed{\text{ConCat}(x, \text{“.”})}, 2, \boxed{\text{Length}(x)} - 1), \text{“.”}, \text{“-”})$$

In places of unconstrained arguments

- Our Top-down propagation: if an inverse set is infinite,
  - inverse set ⊆ {output of e | e is a component}, or
  - inverse set ⊆ {value similar to output of e | e is a component}

Component Generation

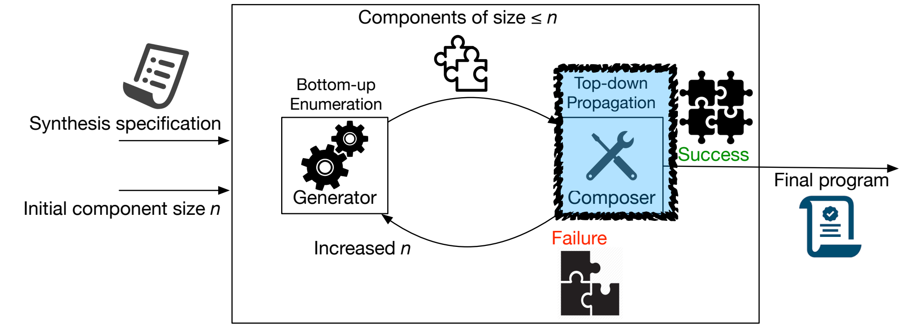


Suppose the initial component size is given **1**.

Bottom-up enumeration generates

$$C = \left\{ \begin{array}{c} x, \text{``+''}, \text{``.''}, \text{``\textvisiblespace''}, \text{``-''}, \\ 1, \cdots, 9 \end{array} \right\}$$

Inverse Set $\subseteq$ Outputs of Components



Component Pool

$$C = \left\{ \begin{array}{c} x, \text{“+”}, \text{“.”}, \text{“␣”}, \text{“-”}, \\ 1, \cdots, 9 \end{array} \right\}$$

S

i_1, o_1

“␣+1␣2.3” $\mapsto$ “1␣2-3.”

“␣+4␣56.78” $\mapsto$ “4␣56-78.”

i_2, o_2

$\mathrm{SubStr}(S, I_1, I_2)$

S

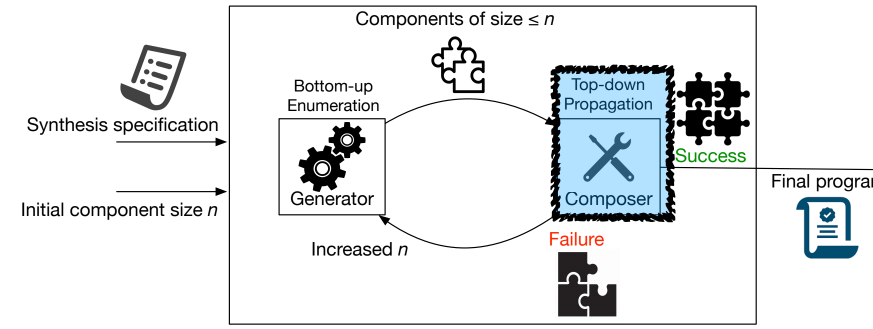
$i_1 \mapsto [\![e]\!](i_1) \quad i_2 \mapsto [\![e]\!](i_2)$

where $e \in C$, $o_1 \prec [\![e]\!](i_1)$,

$o_2 \prec [\![e]\!](i_2)$

Search for a component that produces strings containing the outputs

Inverse Set $\subseteq$ Outputs of Components



Component Pool

$$C = \left\{ \begin{array}{c} x, \text{“+”}, \text{“.”}, \text{“␣”}, \text{“-”}, \\ 1, \cdots, 9 \end{array} \right\}$$

S

$\text{SubStr}(S, I_1, I_2)$

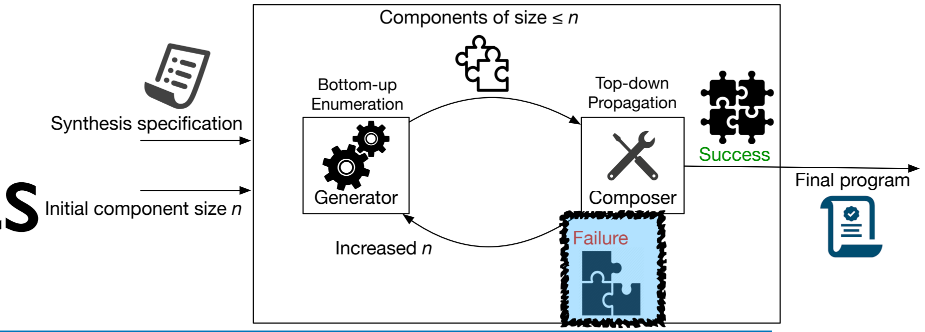
$$\overbrace{\text{“␣+1␣2.3”}}^{i_1} \mapsto \overbrace{\text{“1␣2-3.”}}^{o_1}$$
$$\underbrace{\text{“␣+4␣56.78”}}_{i_2} \mapsto \underbrace{\text{“4␣56-78.”}}_{o_2}$$

$$i_1 \mapsto [\![e]\!](i_1) \quad i_2 \mapsto [\![e]\!](i_2)$$
$$\text{where } e \in C, \; o_1 \prec [\![e]\!](i_1), \; o_2 \prec [\![e]\!](i_2)$$

Simultaneous decomposition:

- handle multiple IO examples simultaneously
- no need for set intersection
- our method scales well on the # of IO examples.

Inverse Set $\subseteq$ Outputs of Components



Component Pool

$$C = \{\, x,\ \text{“+”},\ \text{“.”},\ \text{“␣”},\ \text{“-”},\ 1, \cdots, 9 \,\}$$

S

$i_1 \qquad\qquad o_1$

“␣+1␣2.3” $\mapsto$ “1␣2-3.”

“␣+4␣56.78” $\mapsto$ “4␣56-78.”

$i_2 \qquad\qquad o_2$

$\text{SubStr}(S, I_1, I_2)$

S

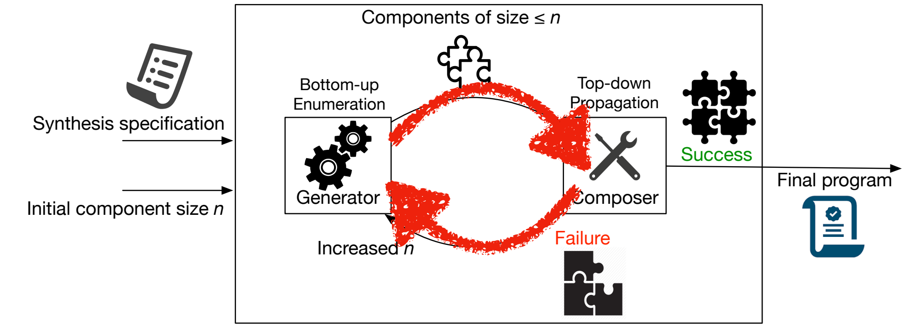
$i_1 \mapsto [\![e]\!](i_1) \quad i_2 \mapsto [\![e]\!](i_2)$

where $e \in C,\ o_1 \prec [\![e]\!](i_1),\ o_2 \prec [\![e]\!](i_2)$

$\nexists e \in C.\ o_1 \prec [\![e]\!](i_1) \wedge o_2 \prec [\![e]\!](i_2)$

No such a component.
Reject the hypothesis.

We don’t miss a solution



- We may miss the following solution of form $\mathrm{SubStr}(\ldots)$ at this iteration because we used size 1 components.

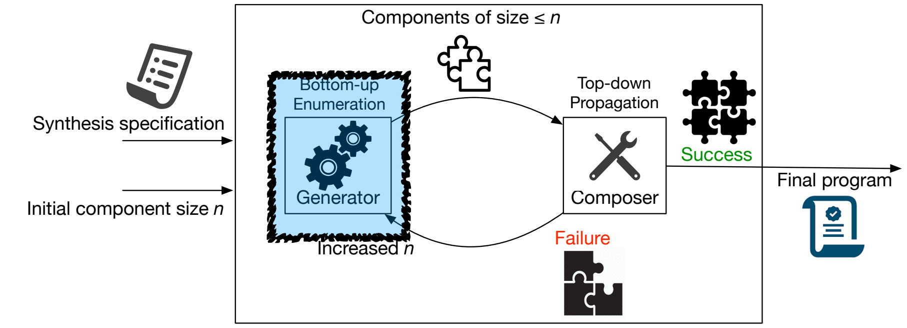
$$\mathrm{SubStr}(\mathrm{Replace}(\mathrm{ConCat}(x, \text{“.”}), \text{“.”}, \text{“-”}), 2, \mathrm{Length}(x) - 1)$$

We should’ve used this component (size 6)

- But we will find it by increasing the component size.

- ***Search completeness***: if a solution exists, we find it.

Component Generation

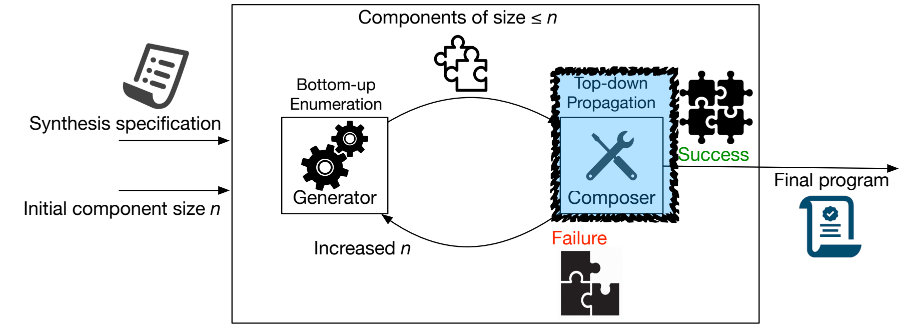


Now suppose the component size is increased to 3.

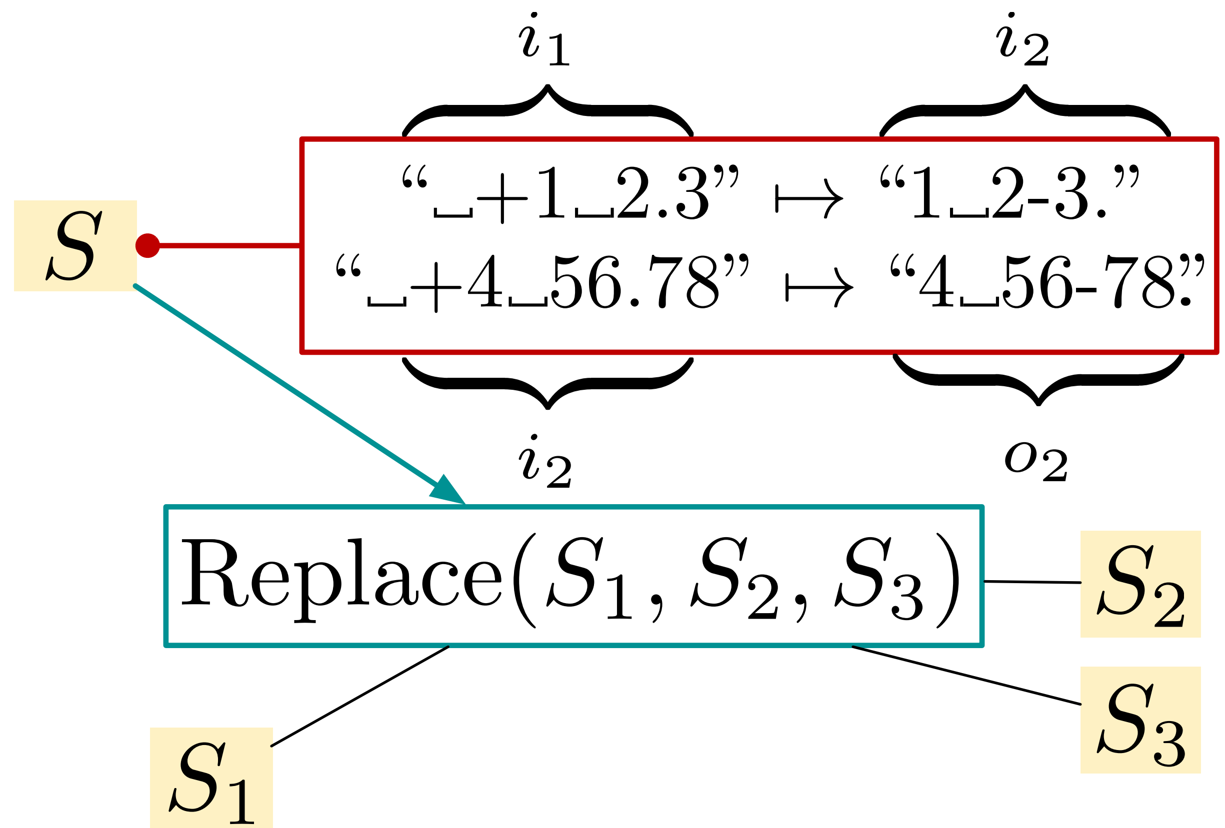
Bottom-up enumeration generates

$$C = \left\{ \begin{array}{c} x, \text{“.”}, \text{“␣”}, \text{“-”}, \\ 1, \cdots, \mathrm{Length}(x), \\ \mathrm{ConCat}(x, \text{“.”}), \\ \ldots \end{array} \right\}$$

Problem of "Inverse Set $\subseteq$ Outputs of Components"



- Suppose now we consider the hypothesis $\text{Replace}(S_1, S_2, S_3)$



- Simple Replace^{-1} similar to SubStr^{-1} needs the following component

$$\text{Replace}(\boxed{\text{SubStr}(\text{ConCat}(x, \text{“.”}), 2, \text{Length}(x) - 1)}, \text{“.”}, \text{“-”})$$

which is large (size 9) and cannot be efficiently generated.

Component-guided Inverse Function

- We generate subproblems that can be eventually solved by the current small components.

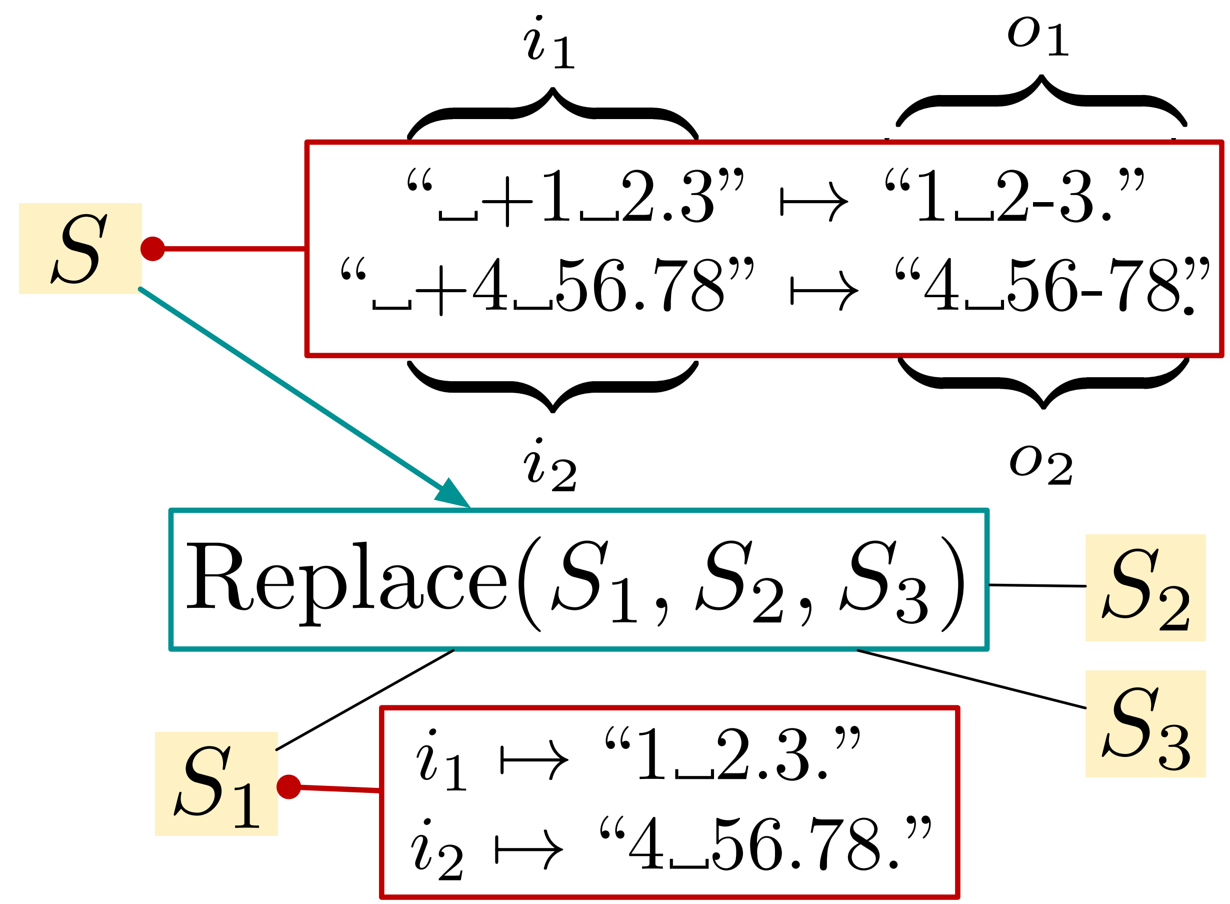
- Our Replace^{-1} : find a component producing strings most similar to the desired outputs → $\text{ConCat}(x, \text{“.”})$

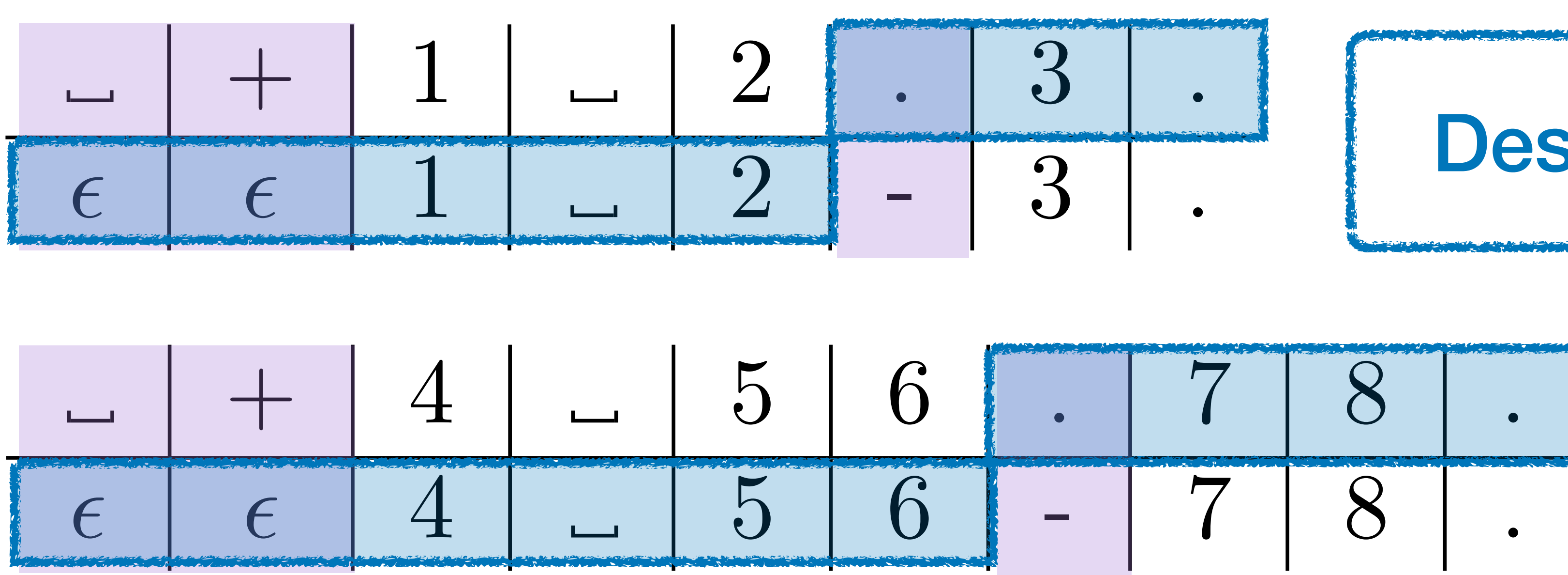
- Compute *alignments* and generate specs for arguments

$\llbracket \text{ConCat}(x, \text{“.”}) \rrbracket (i_1)$	␣	+	1	␣	2	.	3	.
o_1 (desired output 1)	ϵ	ϵ	1	␣	2	-	3	.

$\llbracket \text{ConCat}(x, \text{“.”}) \rrbracket (i_2)$	␣	+	4	␣	5	6	.	7	8	.
o_2 (desired output 2)	ϵ	ϵ	4	␣	5	6	-	7	8	.

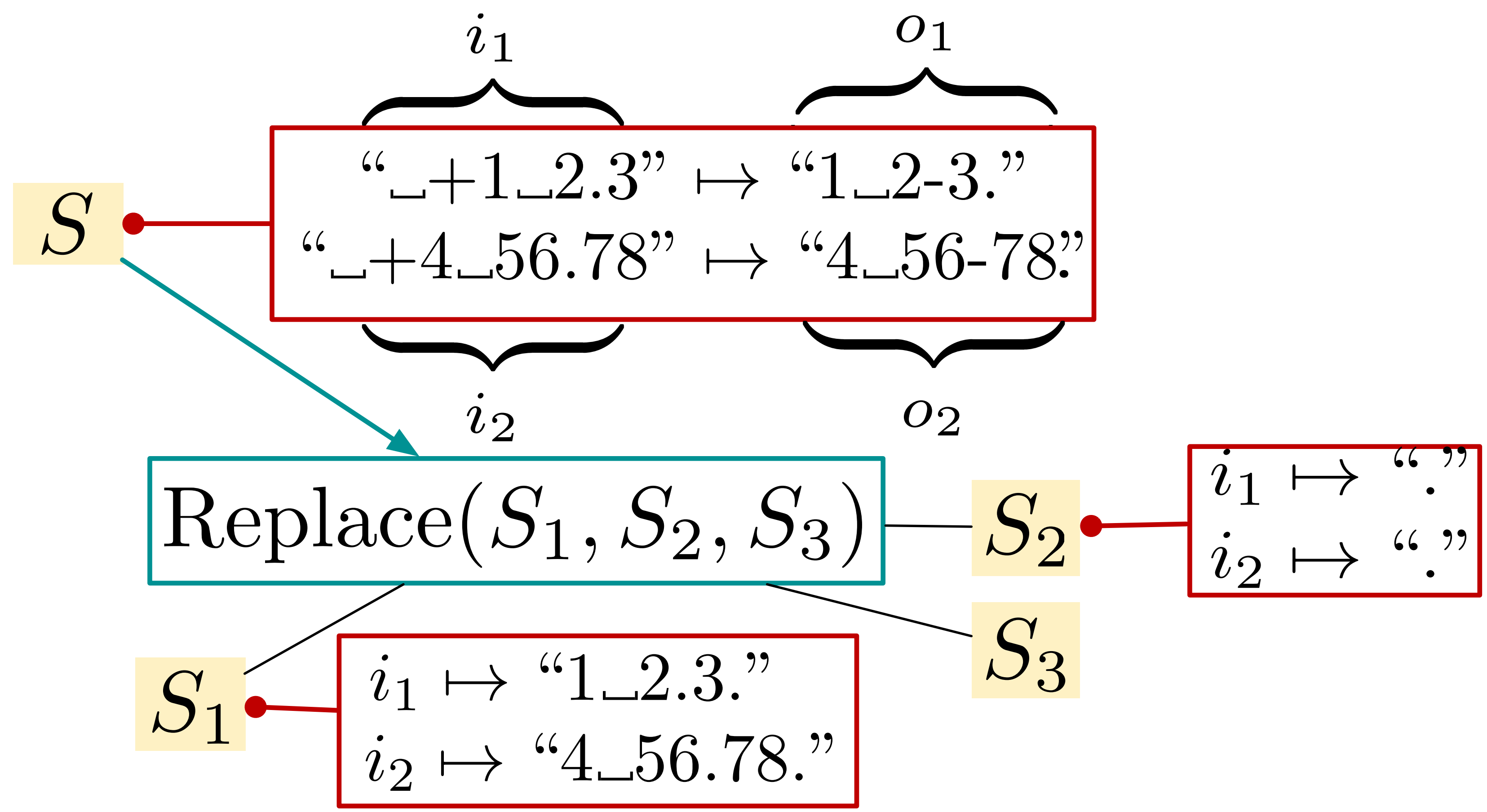
Component-guided Inverse Function

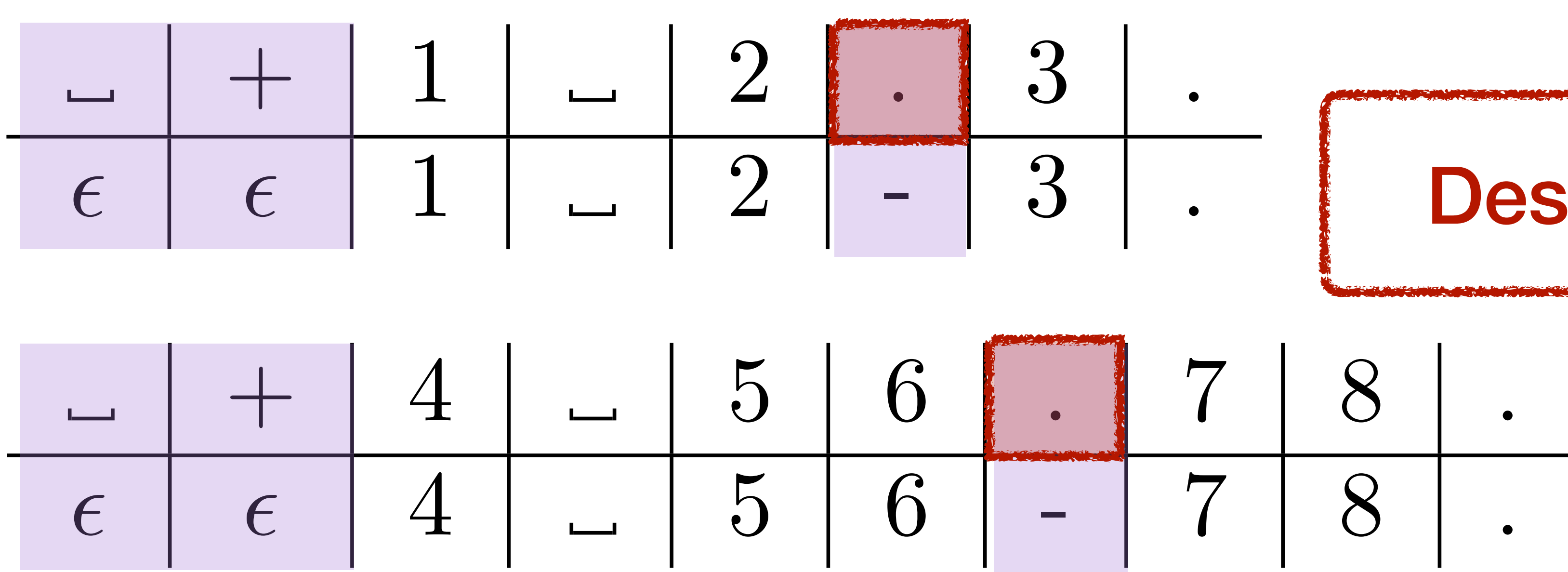




Desired outputs for S_1

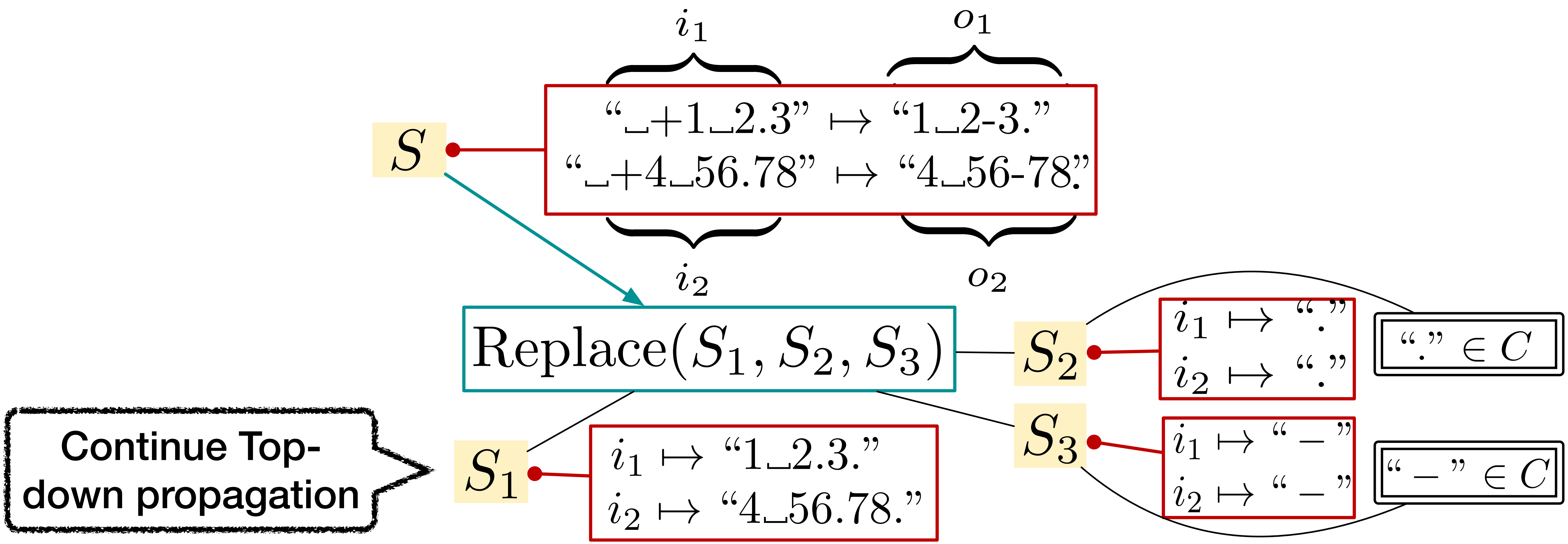
Component-guided Inverse Function





Desired output for S_2

Component-guided Inverse Function

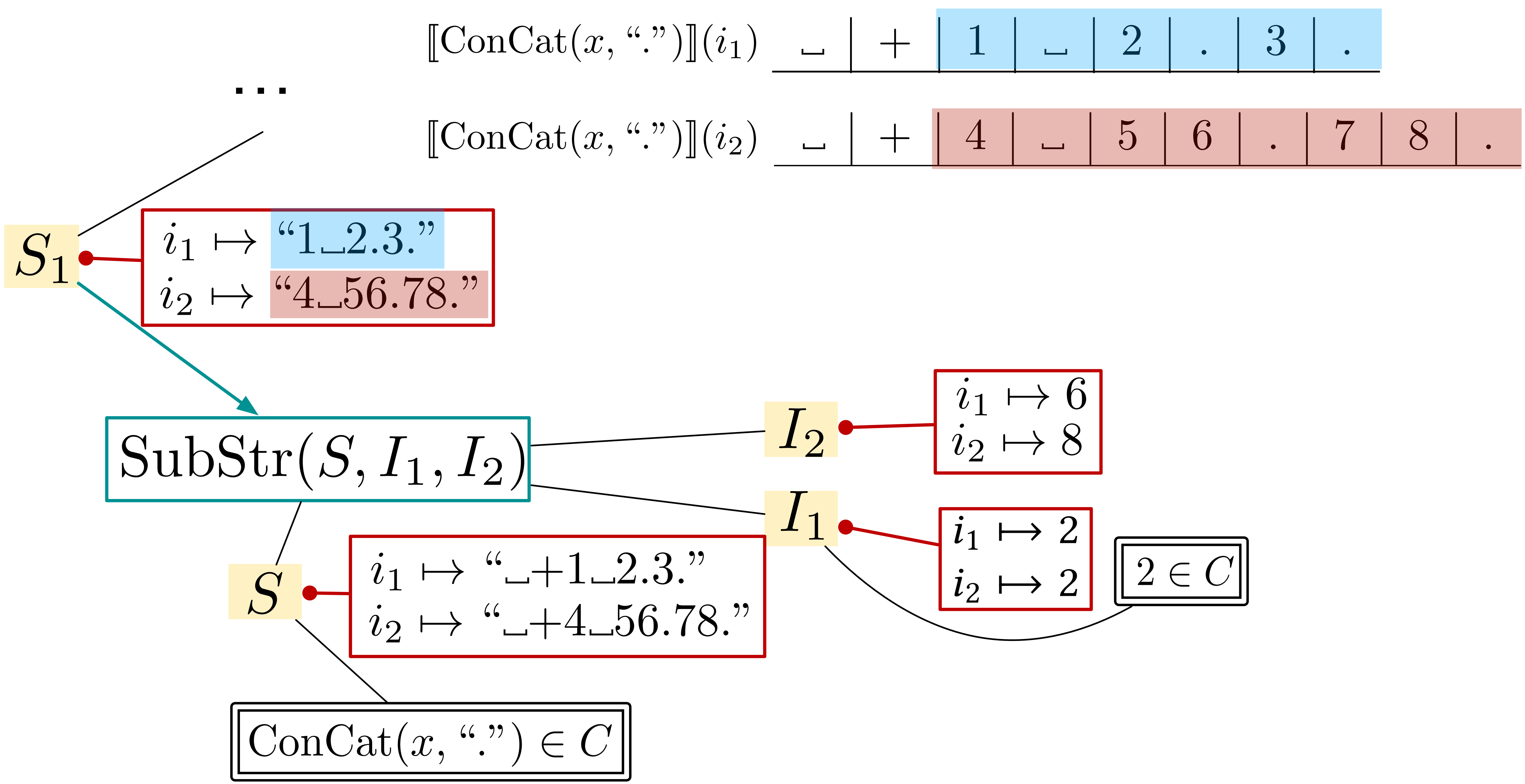


␣	+	1	␣	2	.	3	.
ϵ	ϵ	1	␣	2	-	3	.

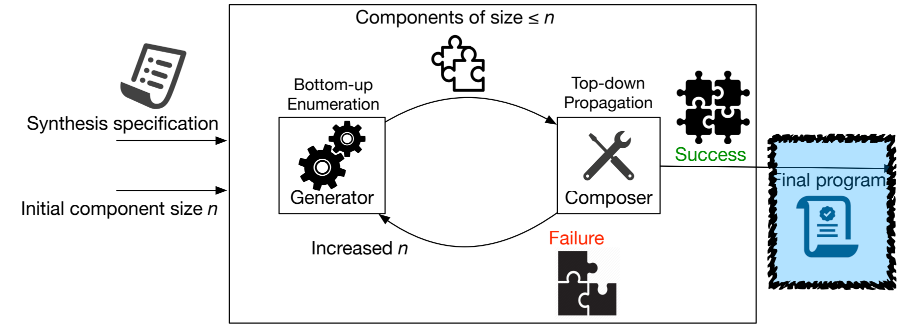
Desired output for S_3

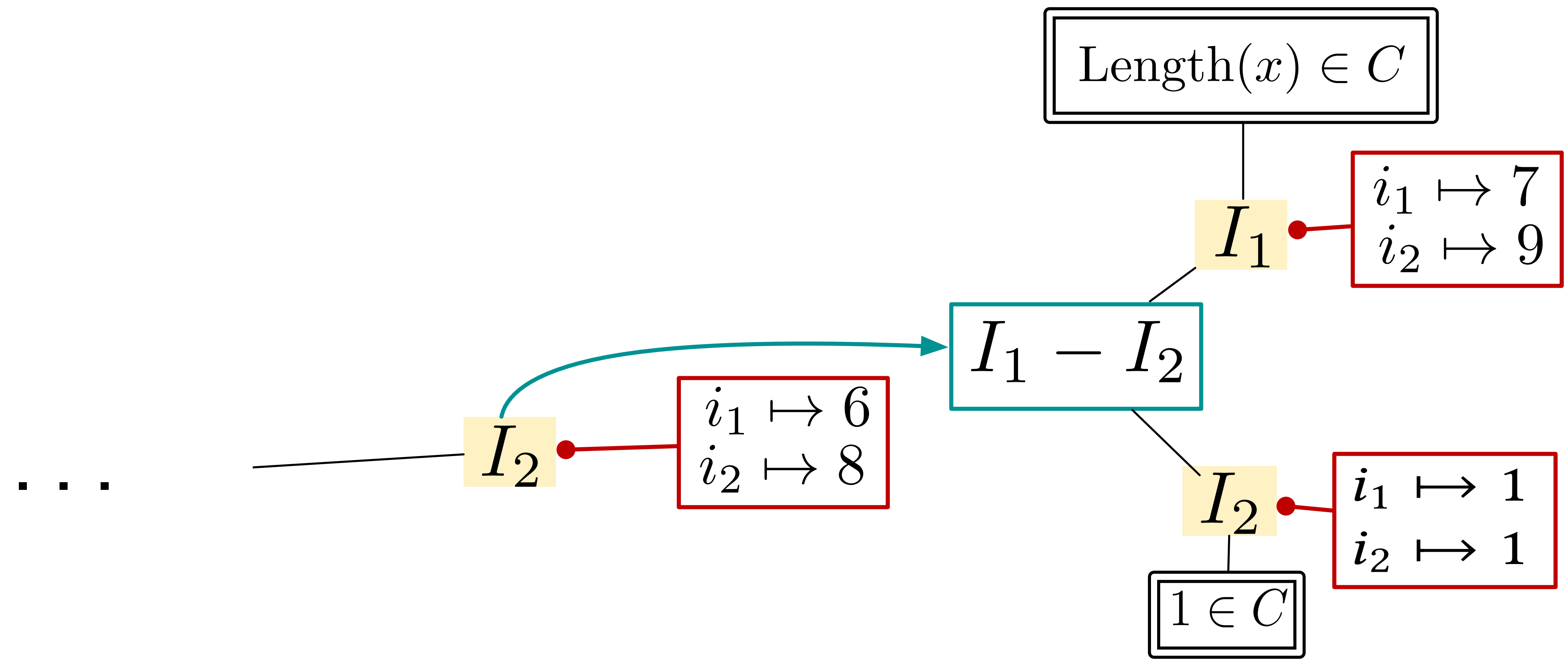
␣	+	4	␣	5	6	.	7	8	.
ϵ	ϵ	4	␣	5	6	-	7	8	.

Component-guided Inverse Function



Component-guided Inverse Function





No subproblems unsolved. The solution is found.

Our Inverse Functions for SyGuS

- Specialized for the operators supported in the SyGuS specification language (strings, bit vectors, LIA, and SAT)
- Type 1: deducing specs **not directly solvable** with components (e.g., Replace^{-1}) → consecutive propagations
- Type 2: deducing specs **directly solvable** with components (e.g., SubStr^{-1}) → solved in a single step

Universal Inverse Function

- In cases where (1) no particular domain knowledge of an underlying operator is exploitable, or (2) computing pre-images is expensive, we use the *universal inverse function.*

- Suppose a spec $i \mapsto o$ is given on N and $N \to F(N_1, \cdots, N_k)$

- Then the universal inverse function computes the following pre-images:

enumerate all possible combinations of components

$$F^{-1}(o) = \{(\llbracket e_1 \rrbracket(i), \cdots, \llbracket e_k \rrbracket(i)) \mid e_1, \ldots, e_k \in C, \\ \llbracket F(e_1, \cdots, e_k) \rrbracket(i) = o\}$$

Evaluation Setup

- Benchmarks: **1,536** SyGuS problems
  - **1,167** from the SyGuS annual competitions + **369** from optimization tasks for homomorphic evaluation [Lee et al. PLDI'20]
- Comparison to three baselines (Timeout 1 hour):
  - EUSolver: winner of 2016 SyGuS competition
  - CVC4: winner of 2017 - 2019 SyGuS competition
  - Euphony [Lee et al. PLDI'18]: statistical model-guided synthesizer

Benchmarks

	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl
11	Alexander.David@contoso.com	alexander david
12	Kim.Shane@northwindtraders.com	kim shane
13	Manish.Chopra@northwindtraders.com	manish chopra
14	Gerwald.Oberleitner@northwindtraders.com	gerwald oberleitner
15	Amr.Zaki@northwindtraders.com	amr zaki
16	Yvonne.McKay@northwindtraders.com	yvonne mckay
17	Amanda.Pinto@northwindtraders.com	amanda pinto

STRING: End-user programming

205 problems

complement

```
~ 01010001110101110000000000001111
  10101110001010001111111111110000
```

bitwise and

```
  01010001110101110000000000001111
& 00110001011011100011000101101110
  00010001010001100000000000001110
```

bitwise or

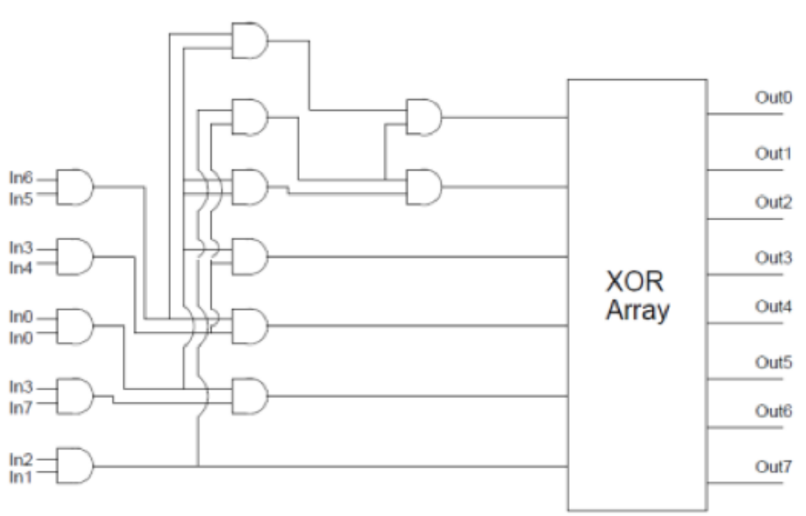
```
  01010001110101110000000000001111
| 00110001011011100011000101101110
  01110001111111110011000101101111
```

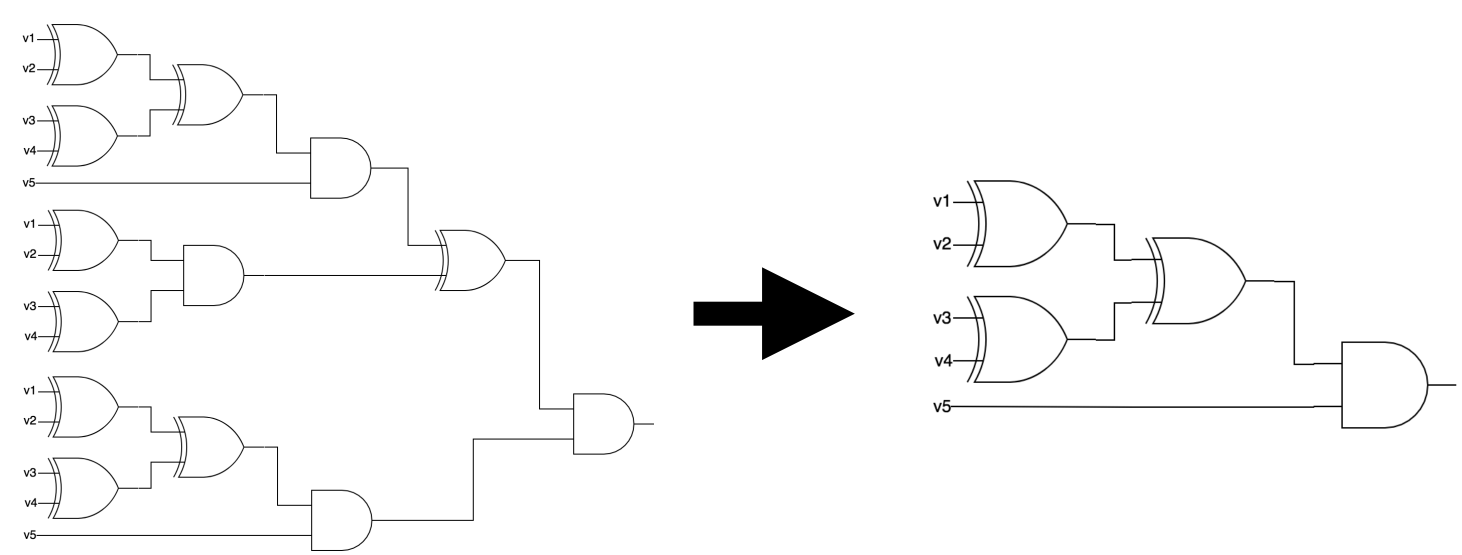
bitwise xor

```
  01010001110101110000000000001111
^ 00110001011011100011000101101110
  01100000101110010011000101100001
```

BITVEC: Efficient low-level algorithm

750 problems

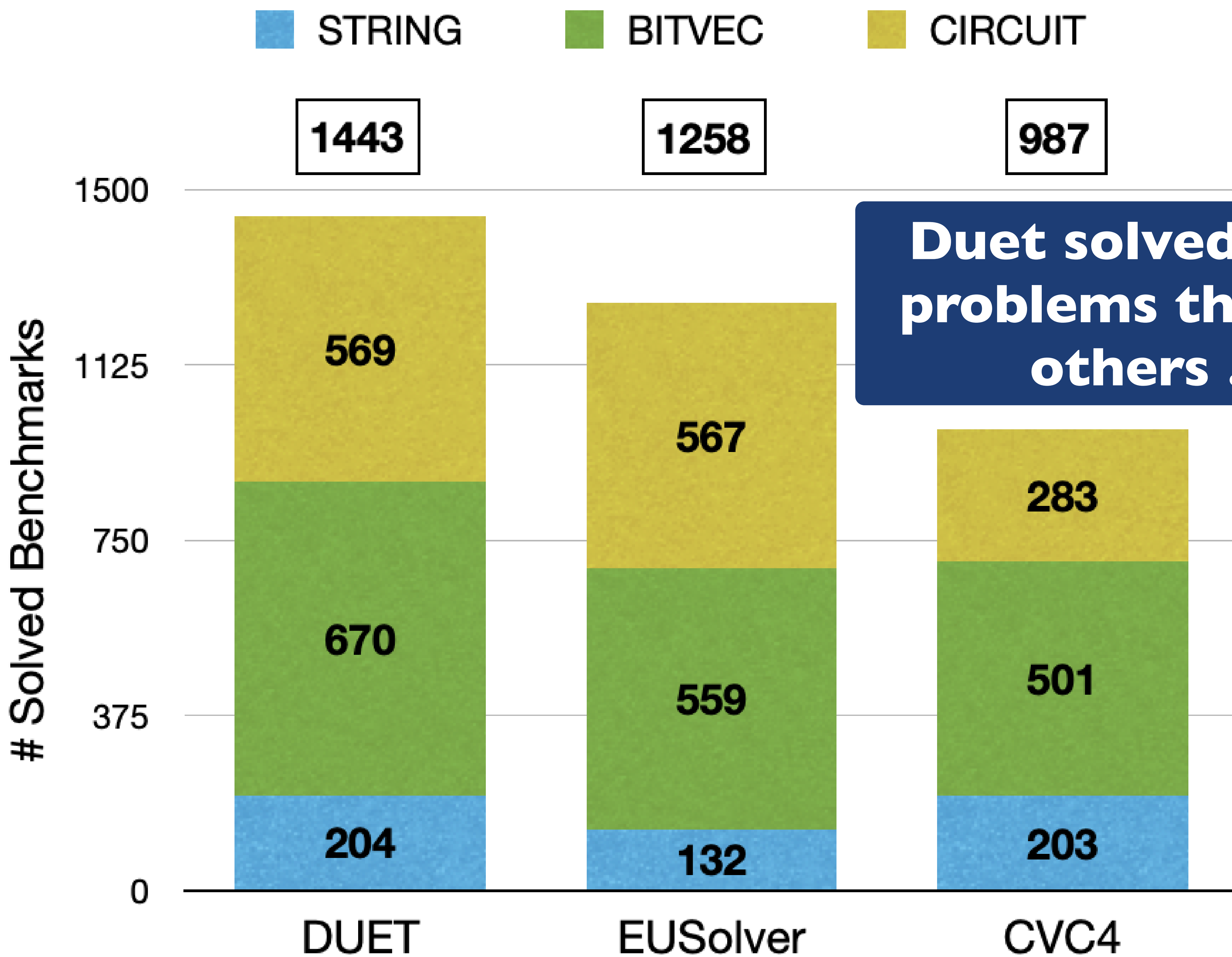




CIRCUIT: Attack-resilient crypto circuits + Optimized homomorphic evaluation circuits

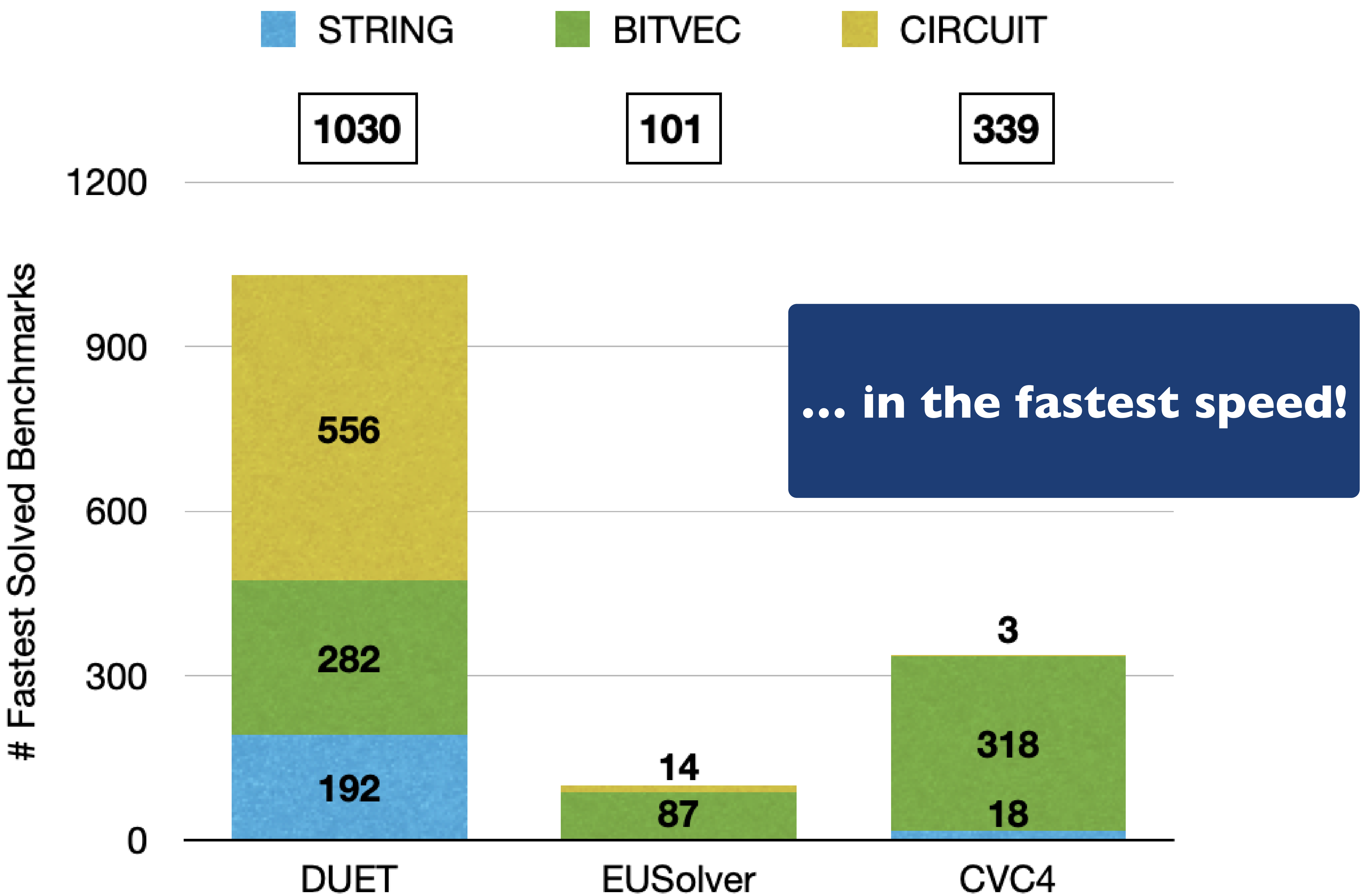
581 problems

Comparison to the Winners of SyGuS Competitions



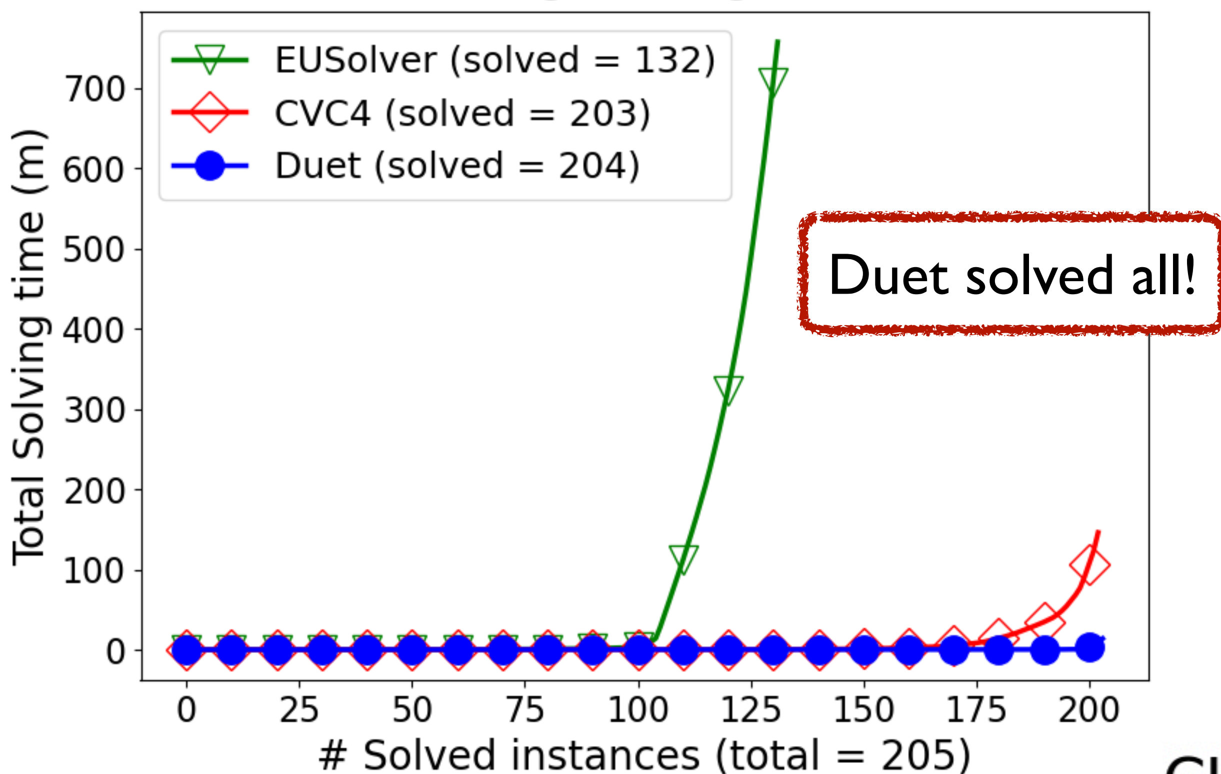
Duet solved more problems than the others …

Comparison to the Winners of SyGuS Competitions

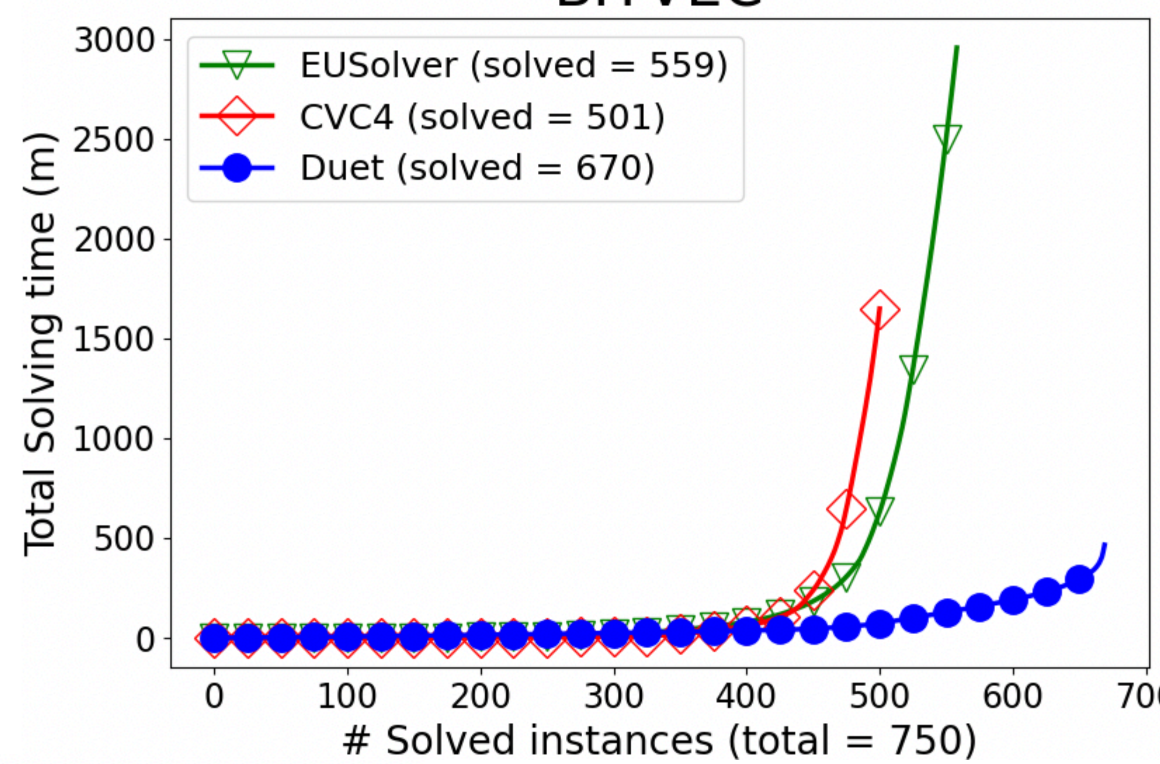


Comparison to the Winners of SyGuS Competitions

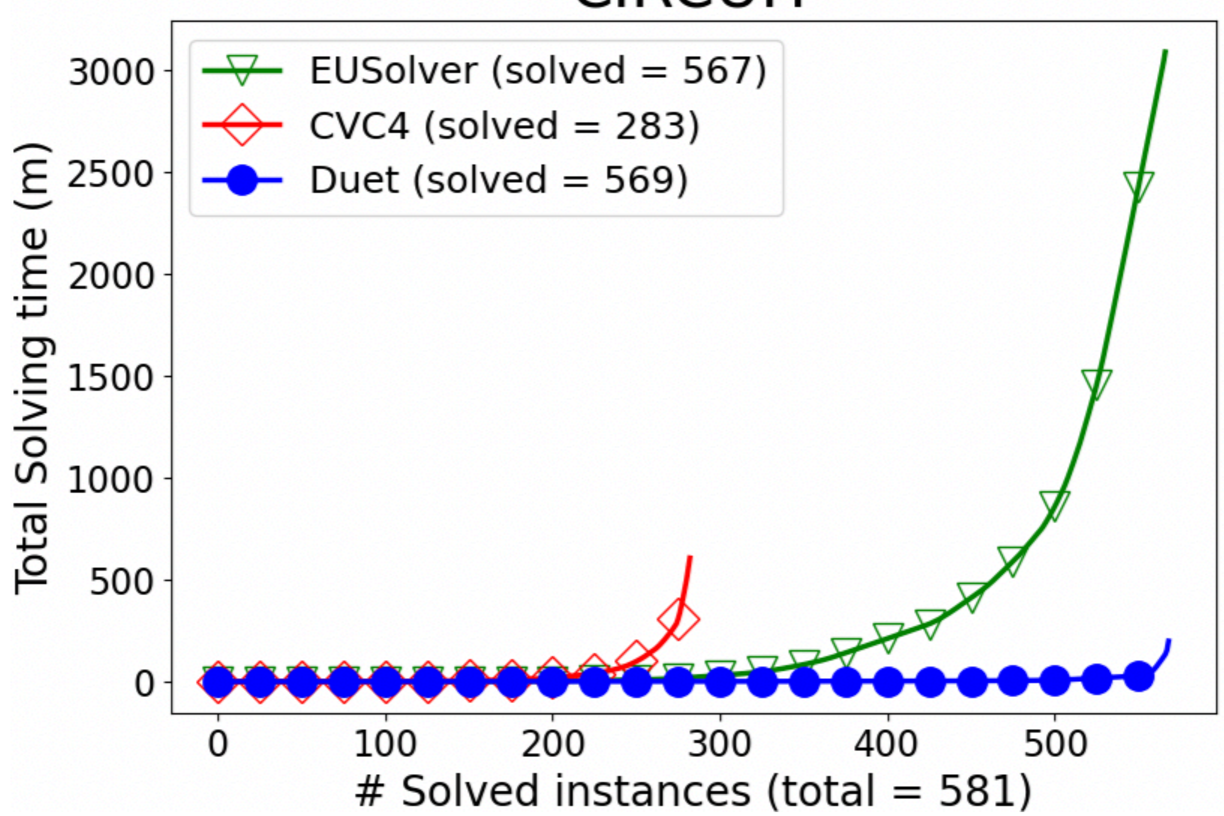
STRING



BITVEC



CIRCUIT

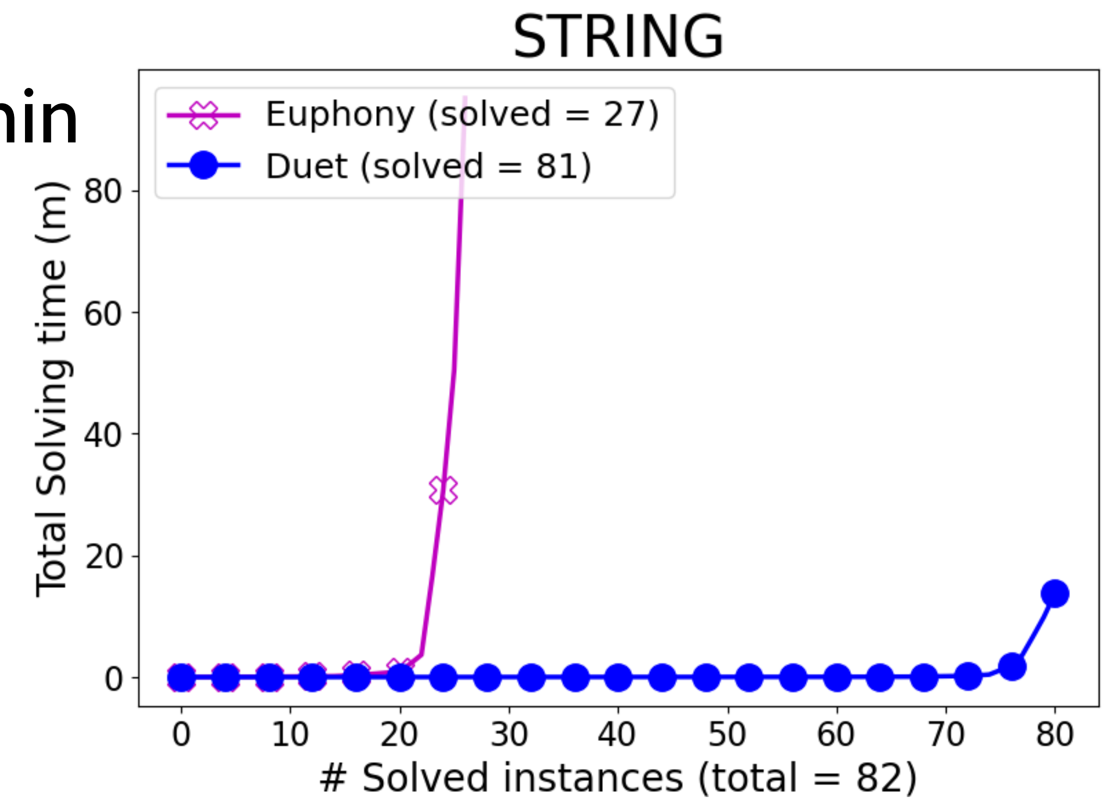


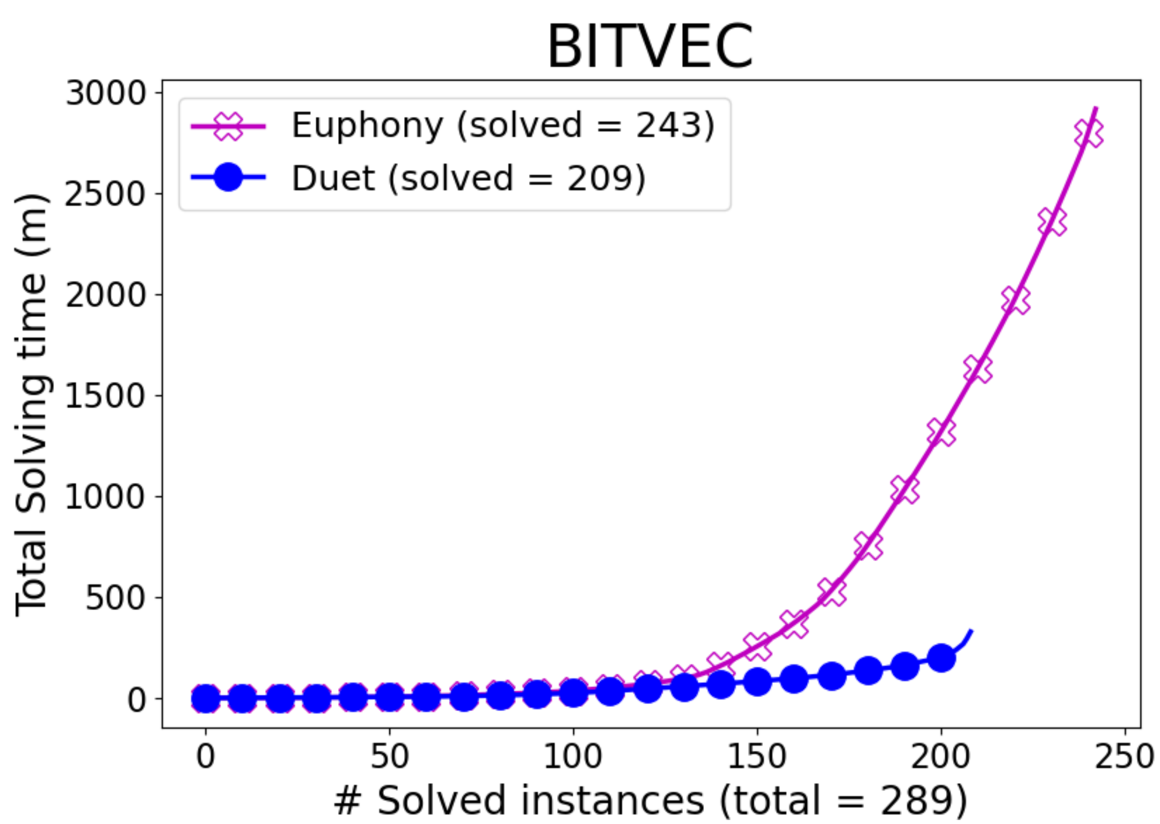
Duet solved

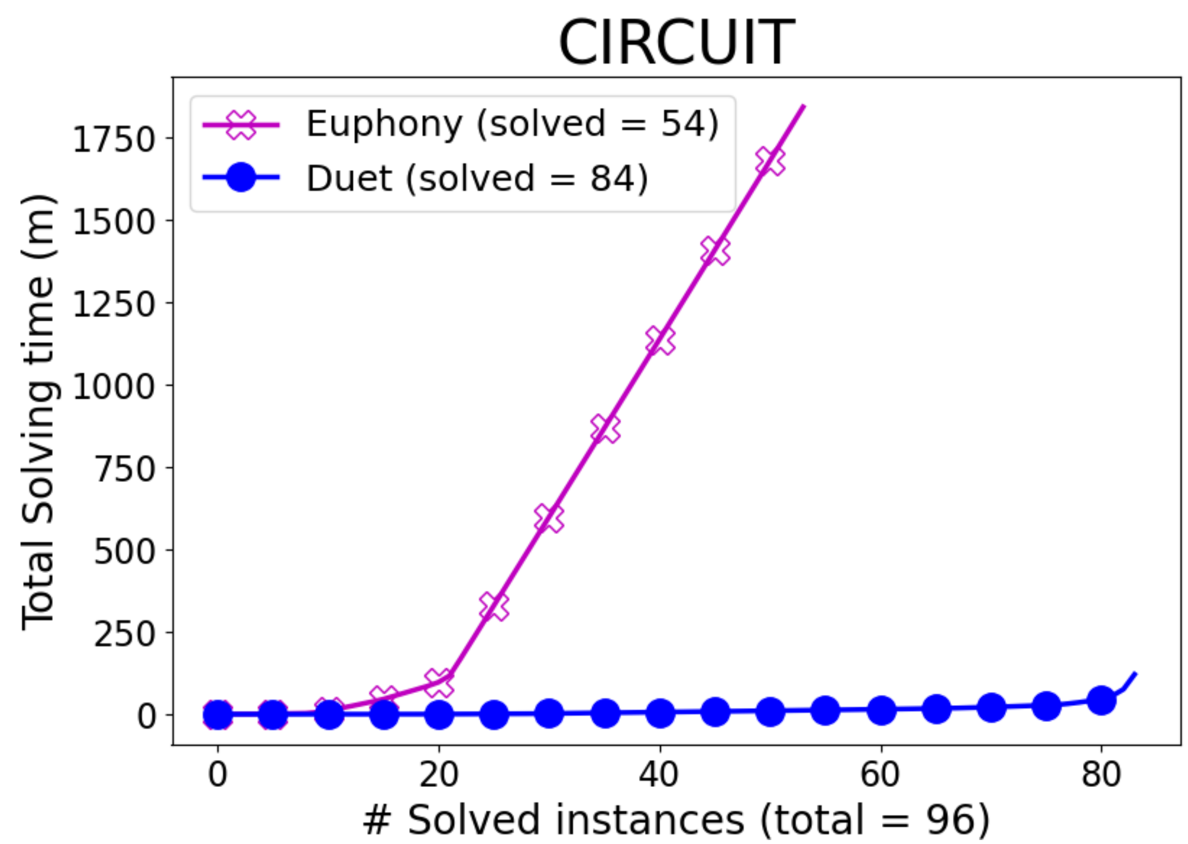
40% within **1 sec**
77% within **10 secs**
90% within **1 min.**

Comparison to Euphony

- Training: 1,069 solved EUSolver in 10 min
- Testing: 467
- # solved: Duet: 374, Euphony 324

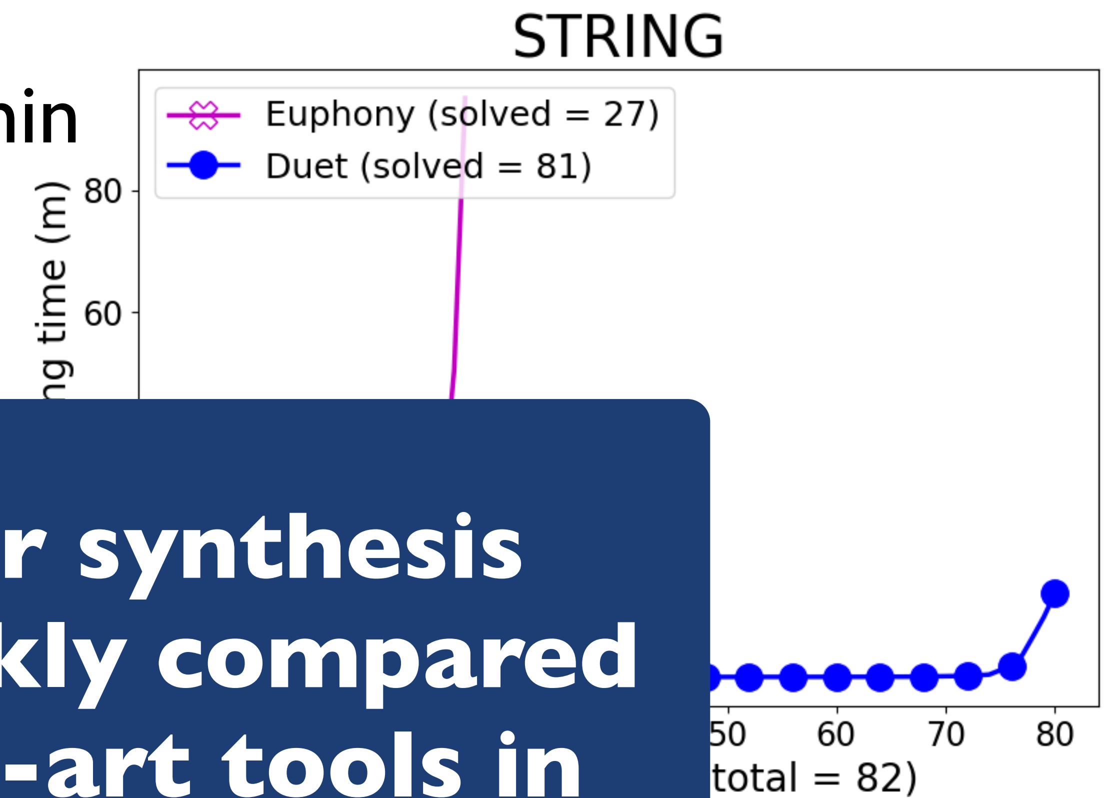




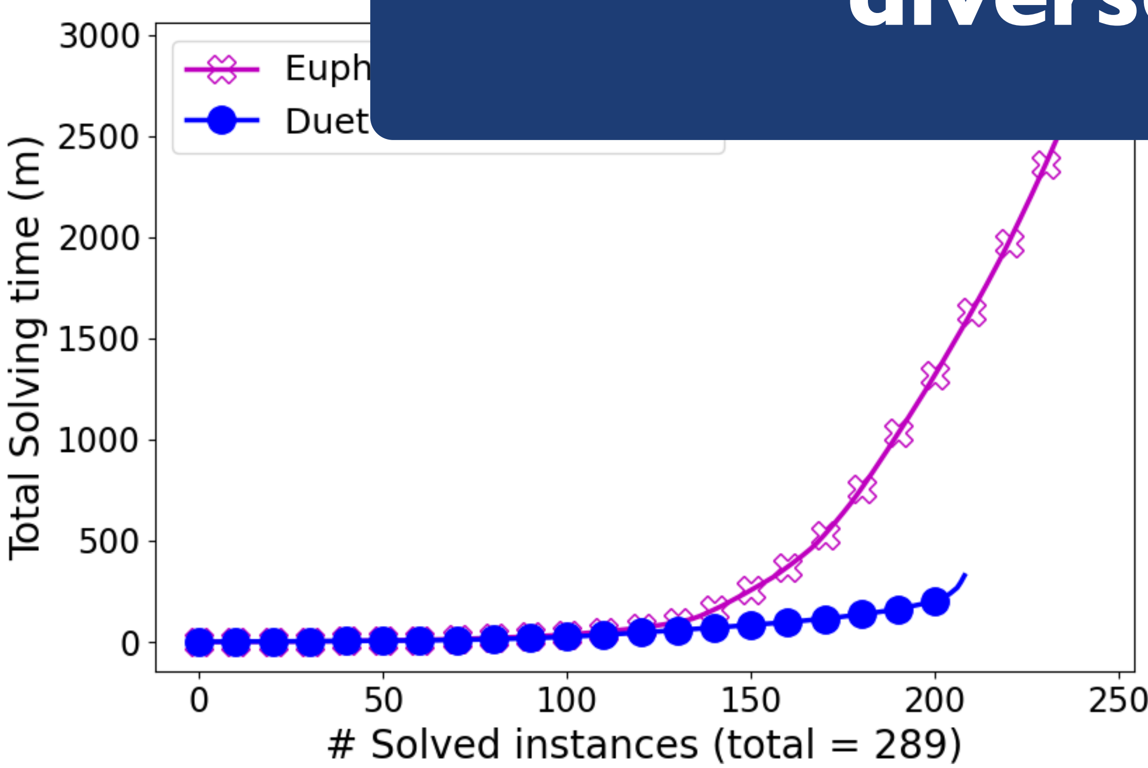


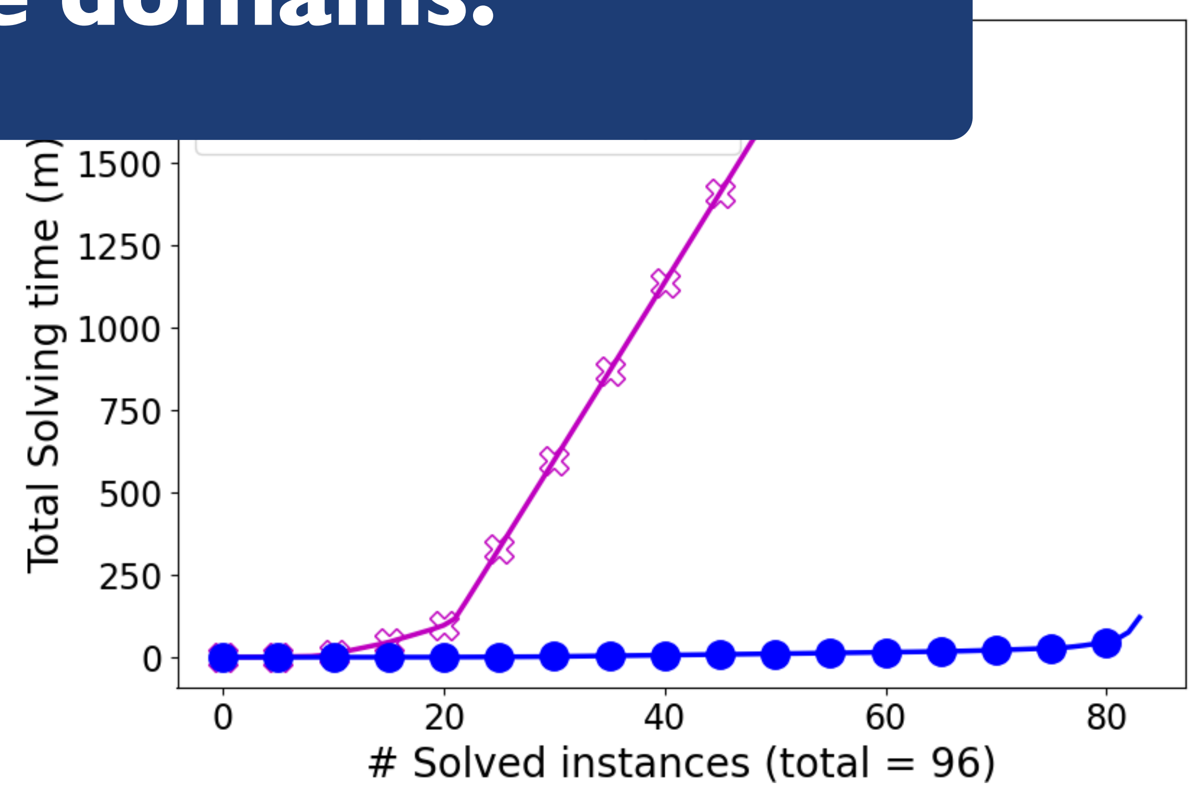
Comparison to Euphony

- Training: 1,069 solved EUSolver in 10 min
- Testing: 467
- # solved



Duet solves harder synthesis problems more quickly compared to the state-of-the-art tools in diverse domains.

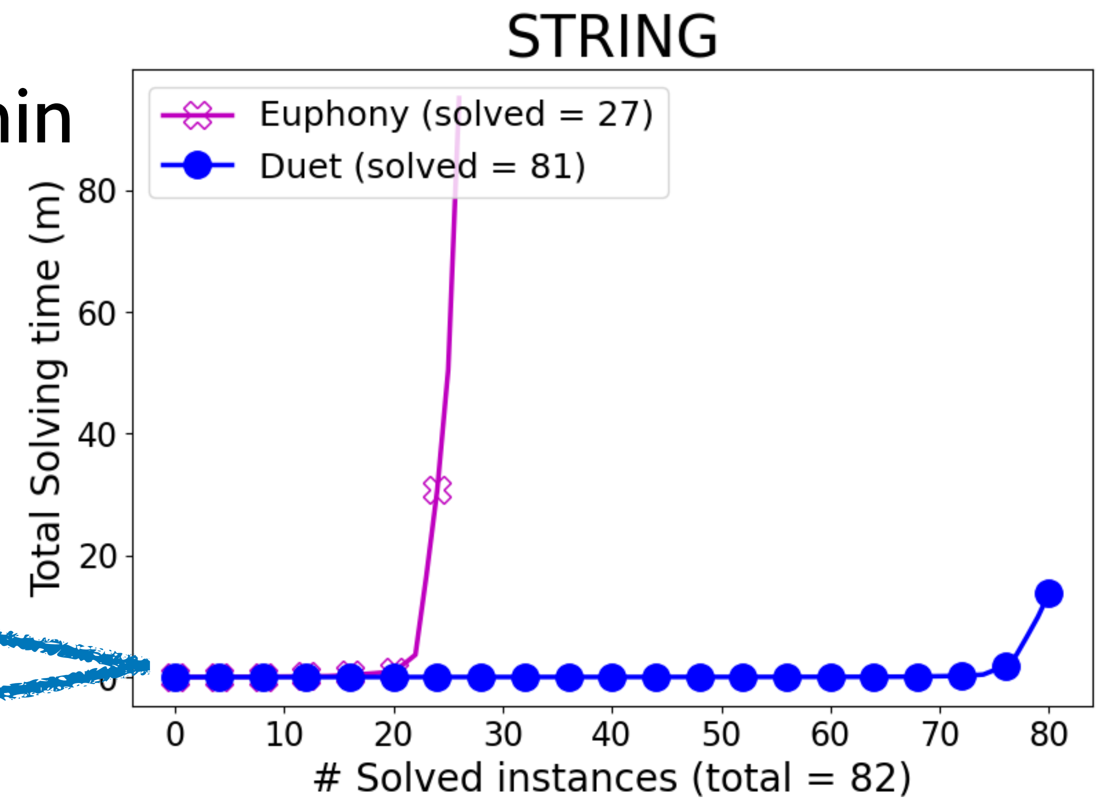


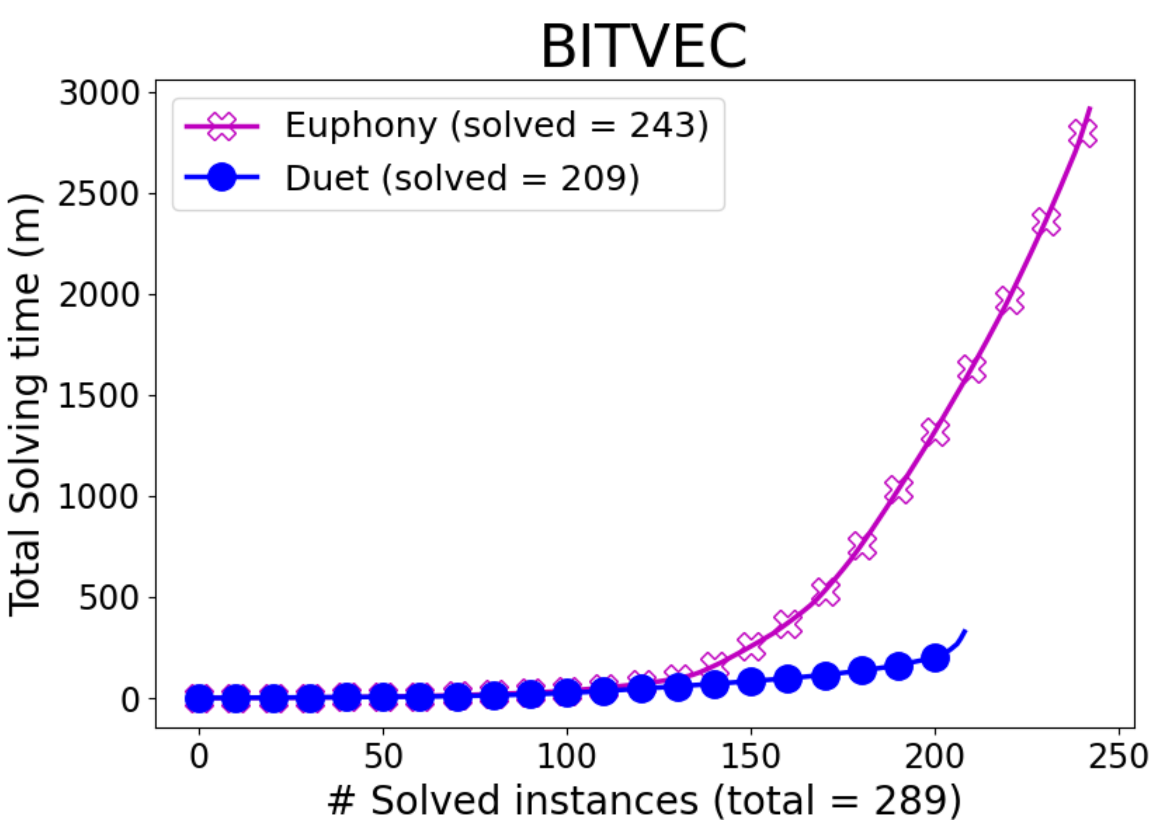


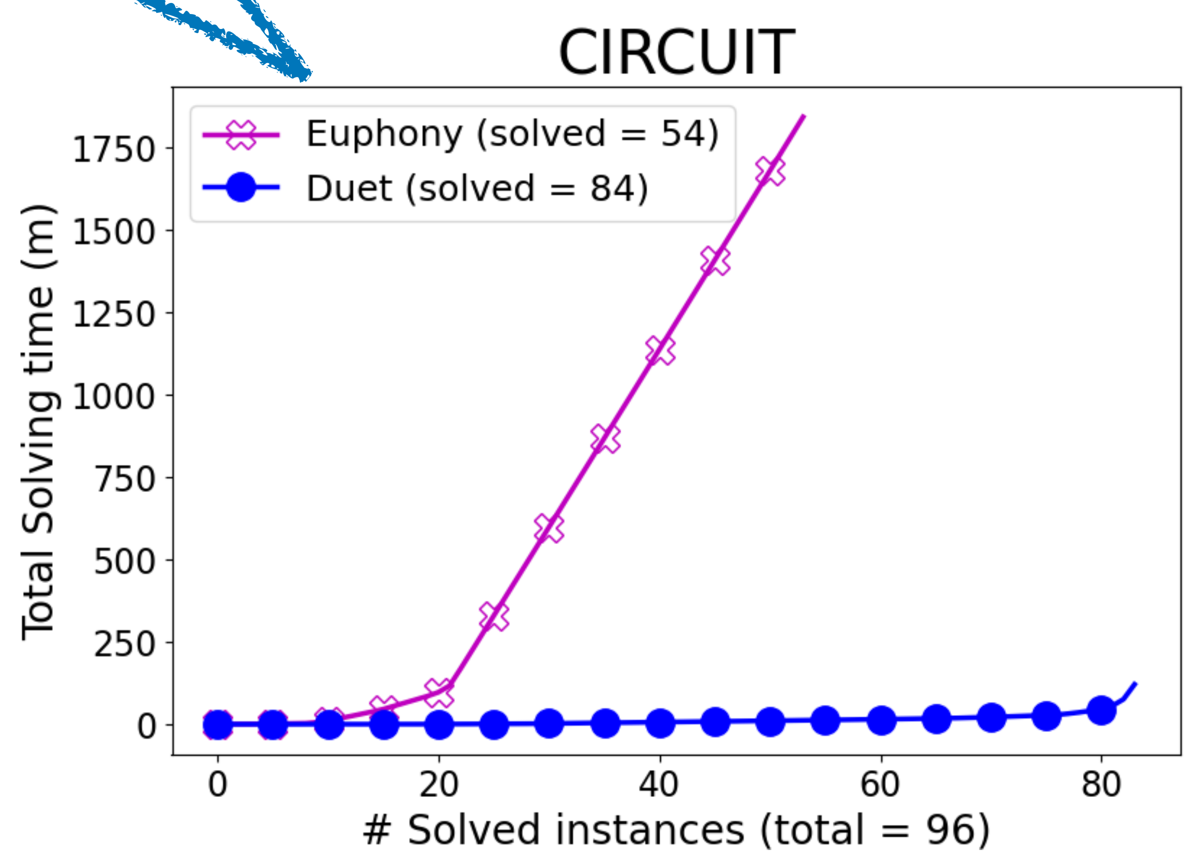
Comparison to Euphony

- Training: 1,069 solved EUSolver in 10 min
- Testing: 4
- # solved

Euphony is outperformed by Duet despite the use of statistical models.





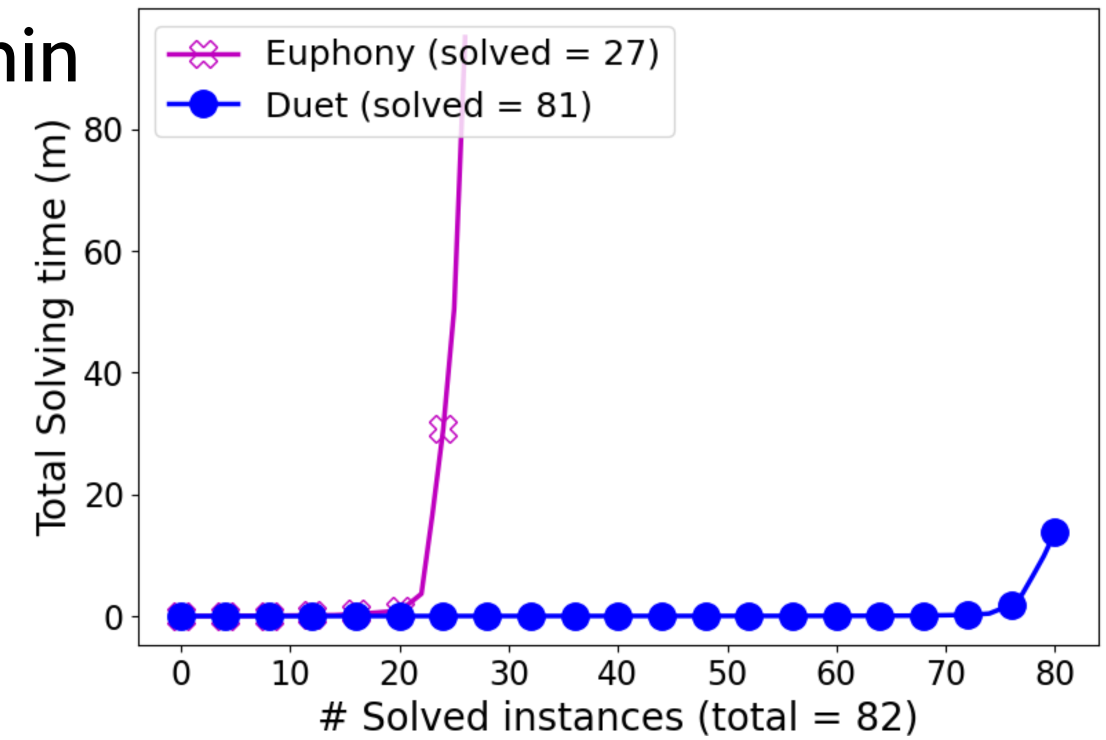


Comparison to Euphony

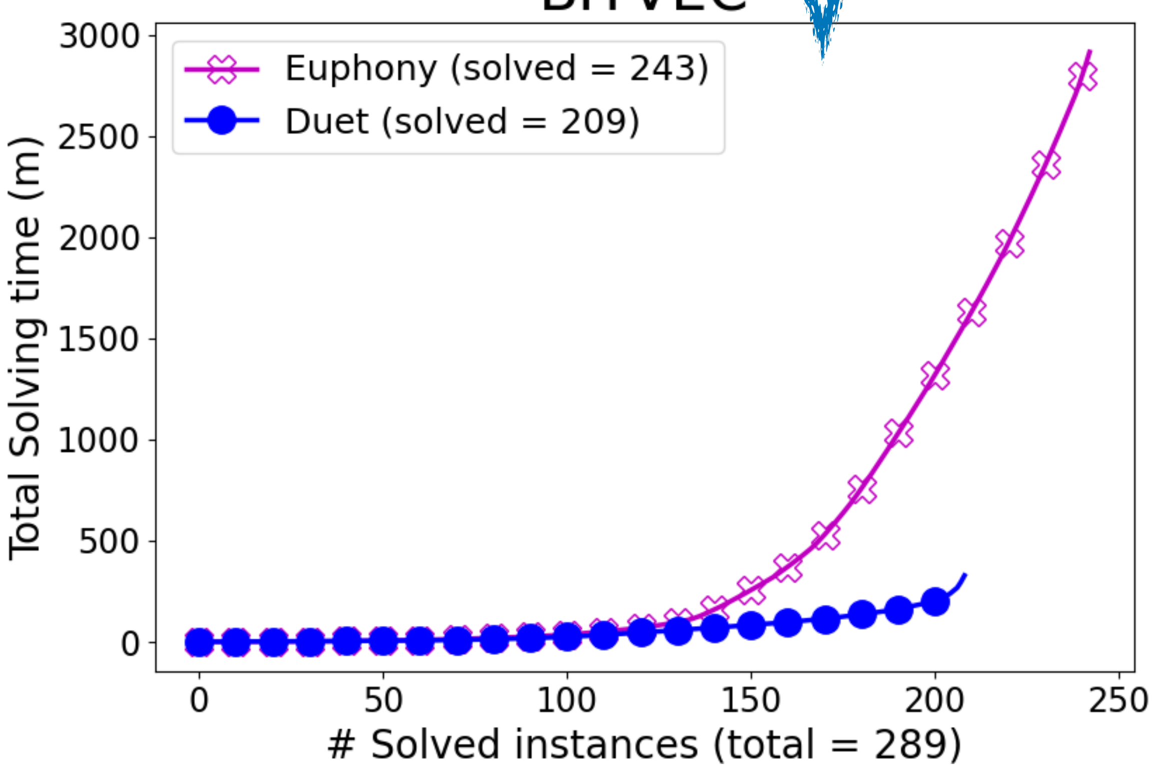
- Training: 1,069 solved EUSolver in 10 min
- Testing: 467
- # solved

Euphony outperforms Duet by exploiting statistical regularity.

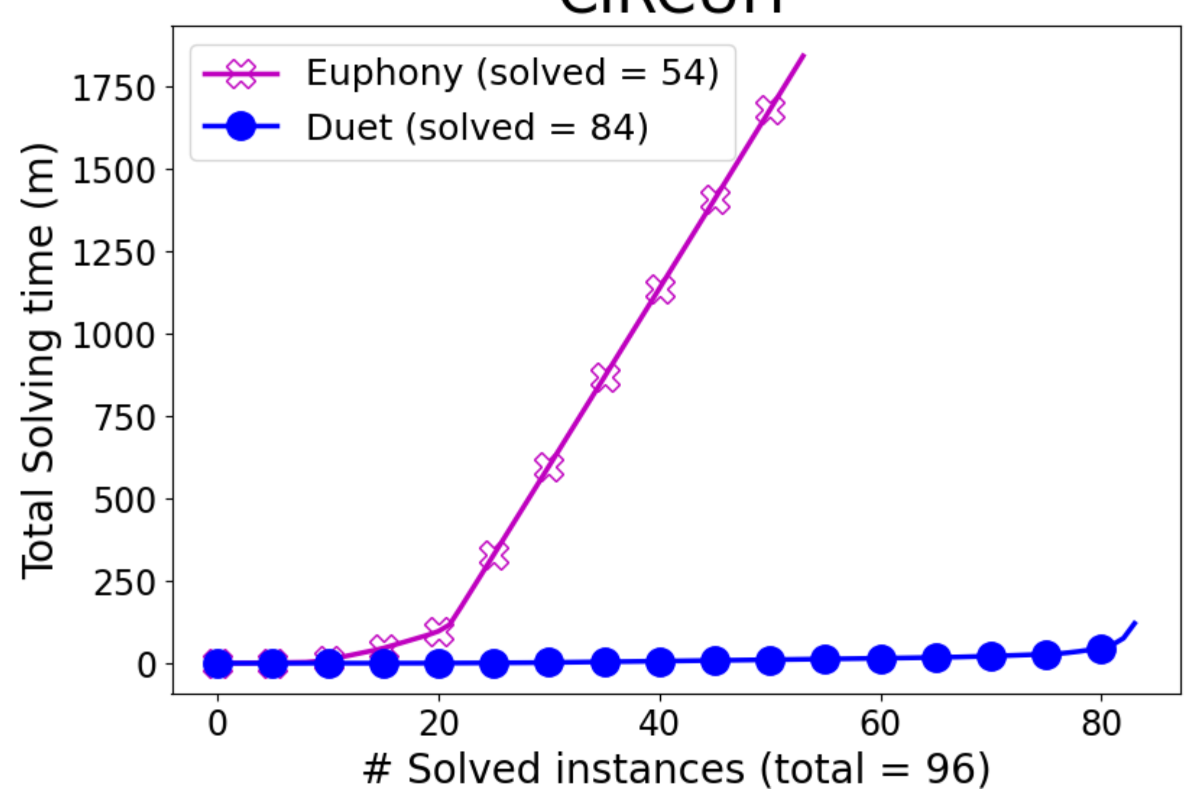
STRING



BITVEC

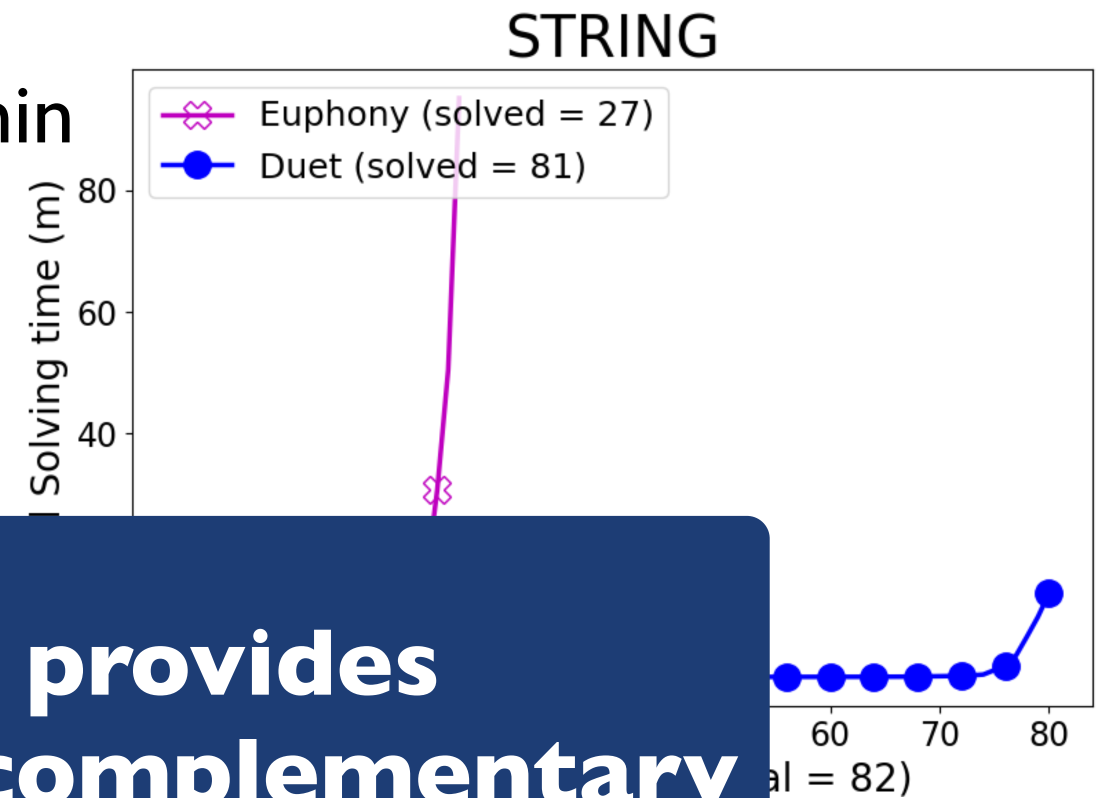


CIRCUIT

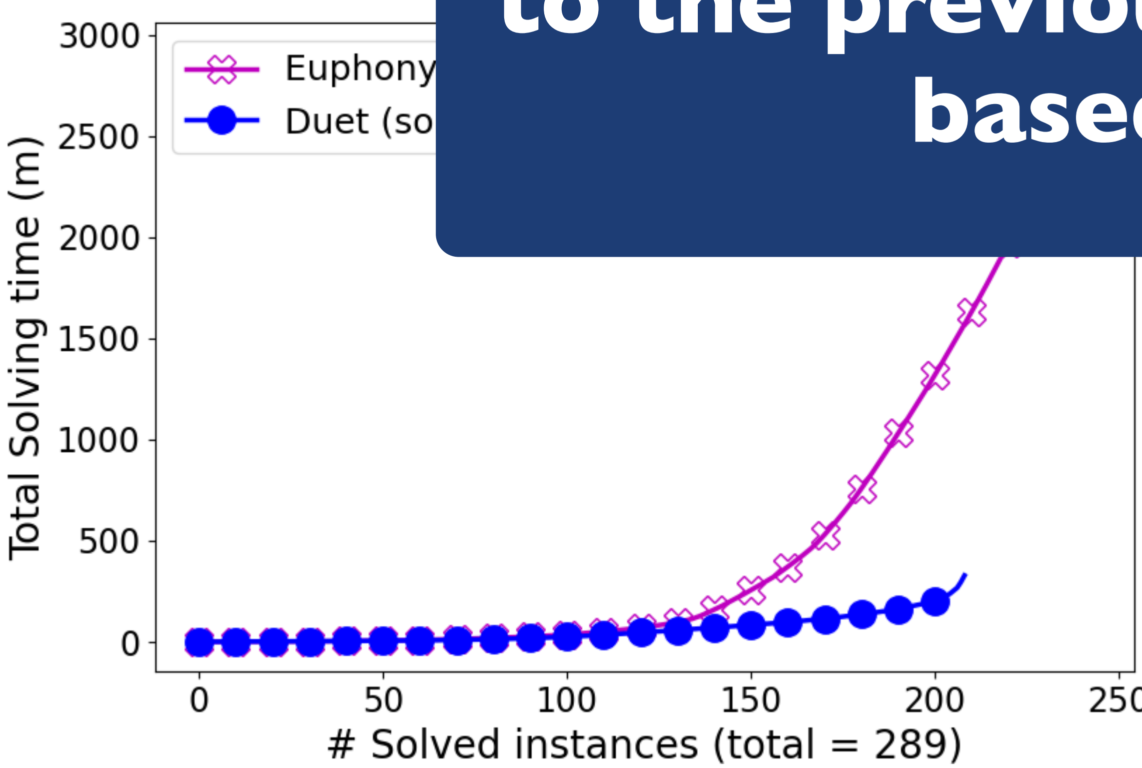


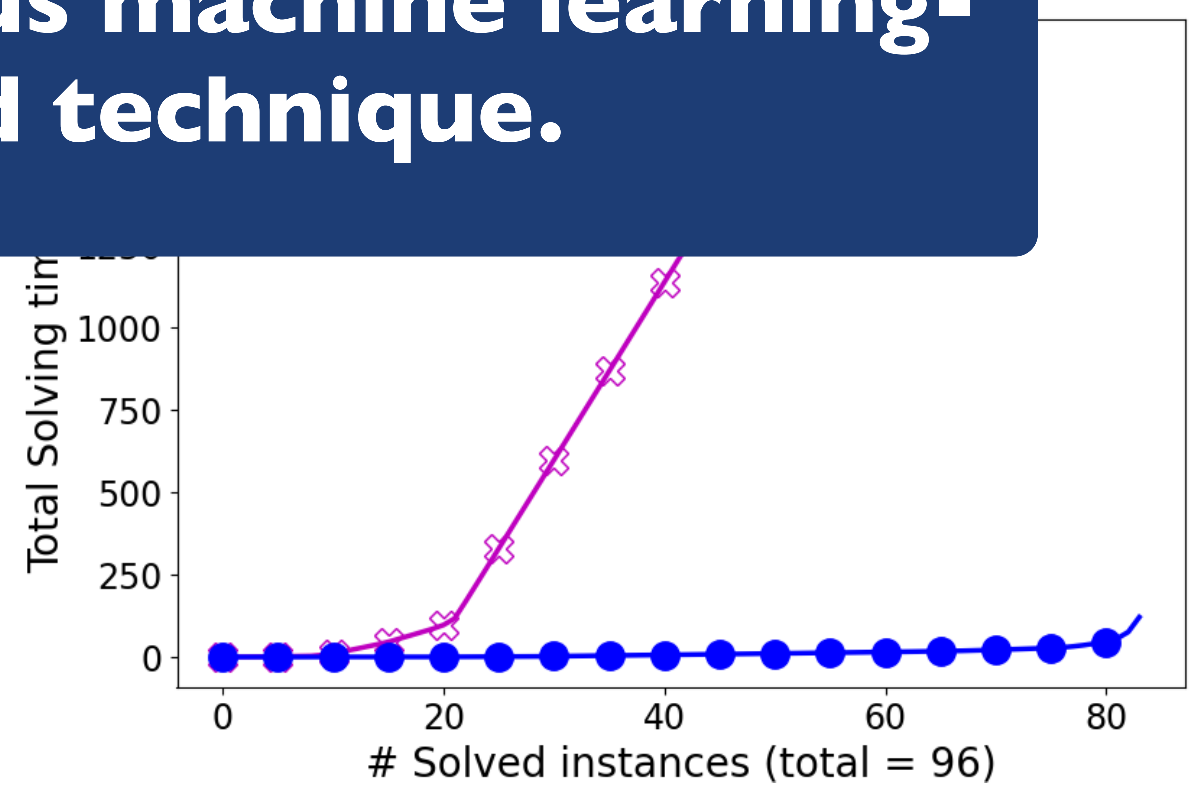
Comparison to Euphony

- Training: 1,069 solved EUSolver in 10 min
- Testing: 467
- # solved: D



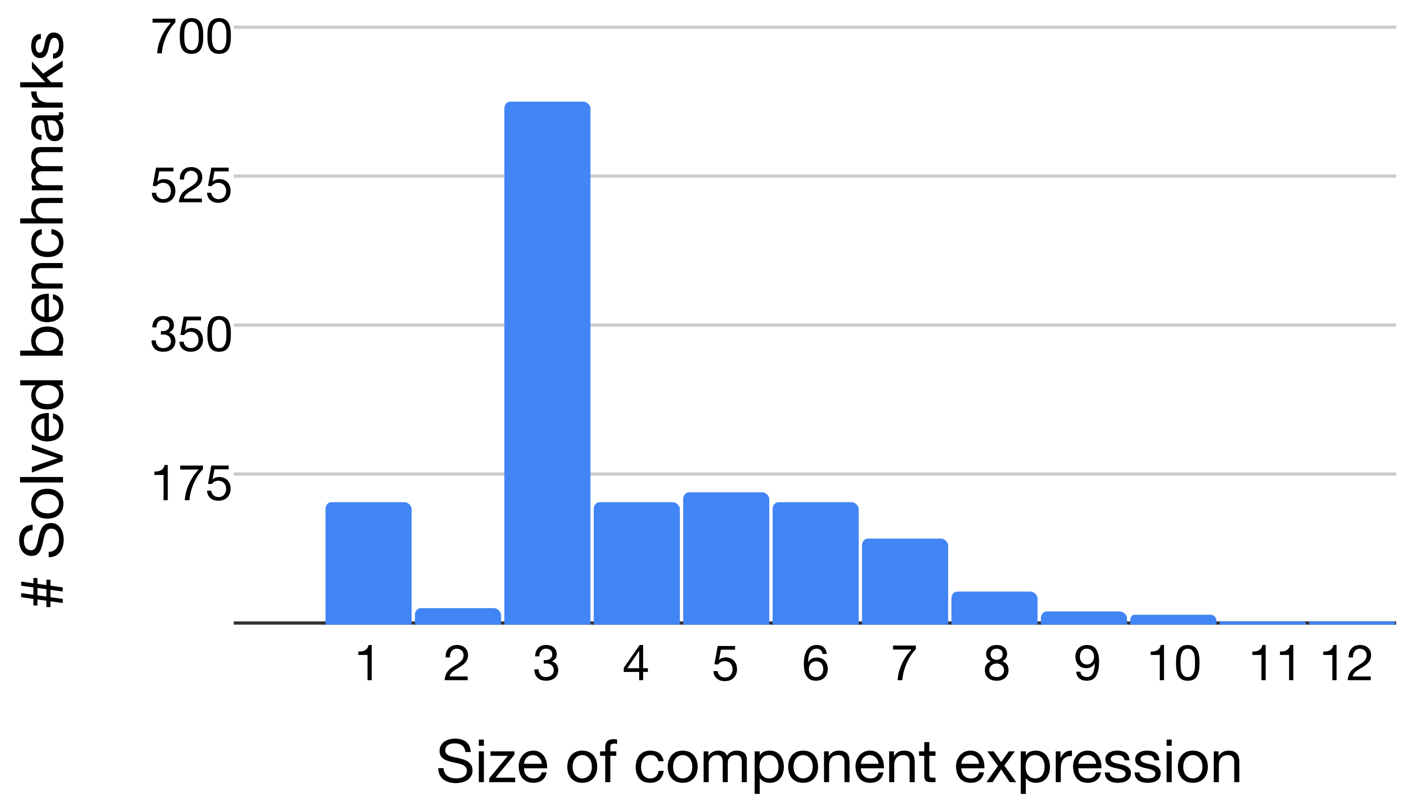
Our approach provides performance gains complementary to the previous machine learning-based technique.





Analysis of Component Sizes

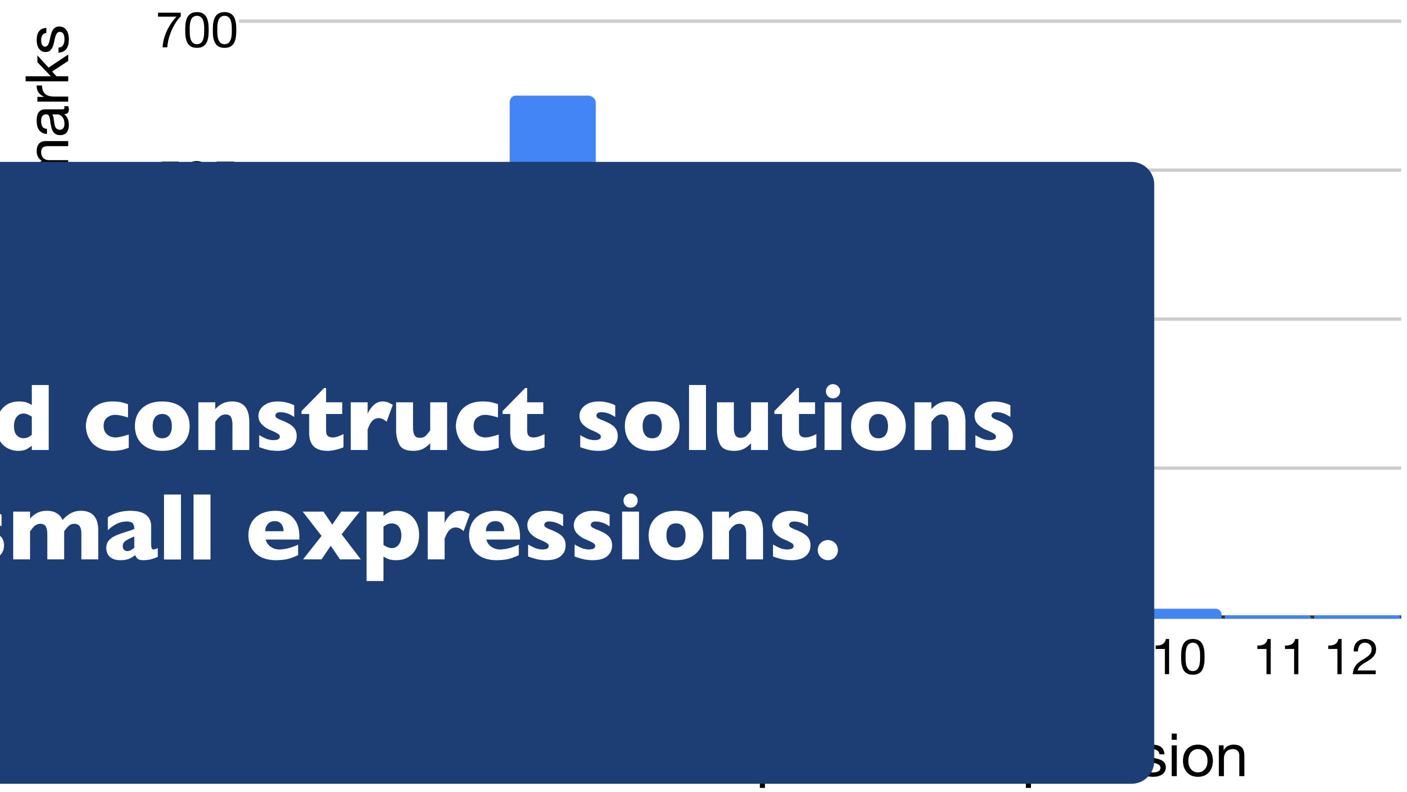
- Sizes of components needed to construct a solution
  - STRING: 1 ~ 6 (avg: 1.8)
  - BITVEC: 3 ~ 5 (avg: 3.5)
  - CIRCUIT: 3 ~ 12 (avg: 5.9)



- 92% of problems could be solved with components of size ≤ 7.

Analysis of Component Sizes

- Sizes of components needed to construct a solution
  - STRING: 1 ~ 6 (avg: 1.8)
  - BITVEC 2 ~ 5 (avg: 3.5)
  - CIRCU



Duet could construct solutions from small expressions.

- 92% of problems could be solved with components of size ≤ 7.