Presented virtually at OOPLSA 2020

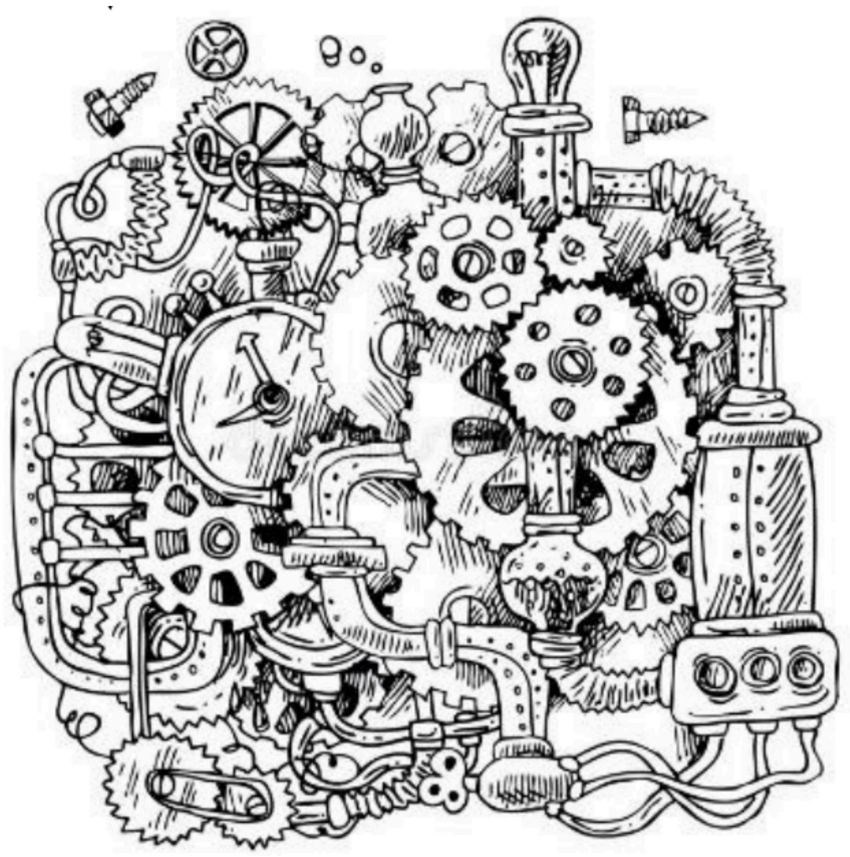
Just-in-time Learning for Bottom-Up Enumerative Synthesis

Shraddha Barke
Hila Peleg
Nadia Polikarpova

UC San Diego

Program Synthesis

Synthesizer



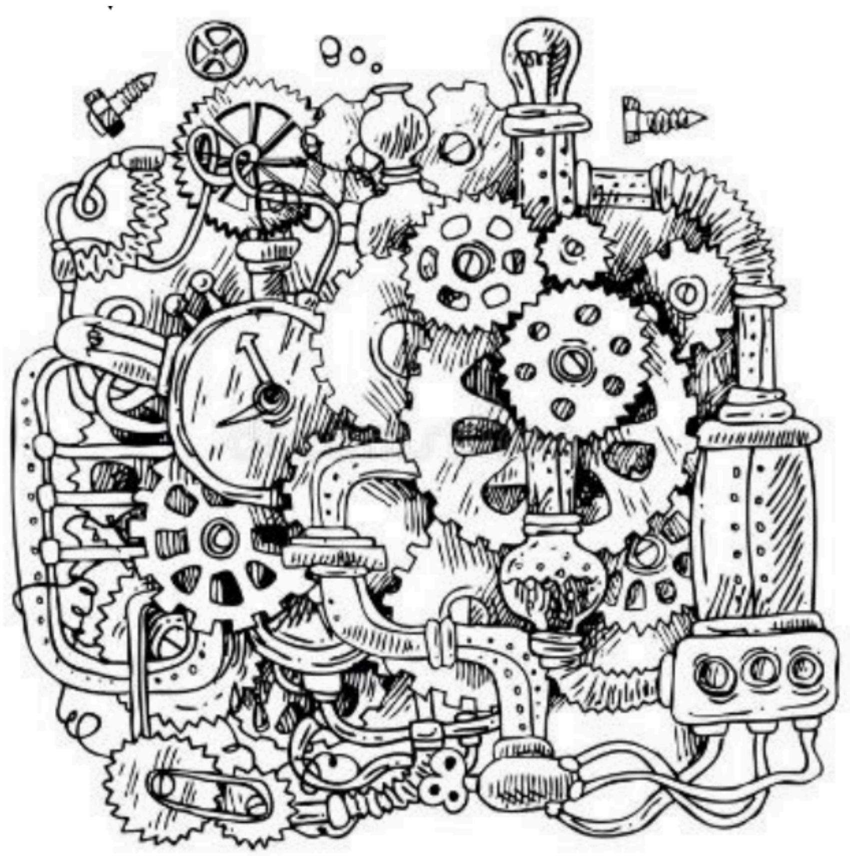
Specification → ← Program Space

↓

Program

Syntax-Guided Program Synthesis (SyGuS)

Synthesizer



Specification

Input-output examples

(or first-order formula)

Program Space

Context-free Grammar

Program

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

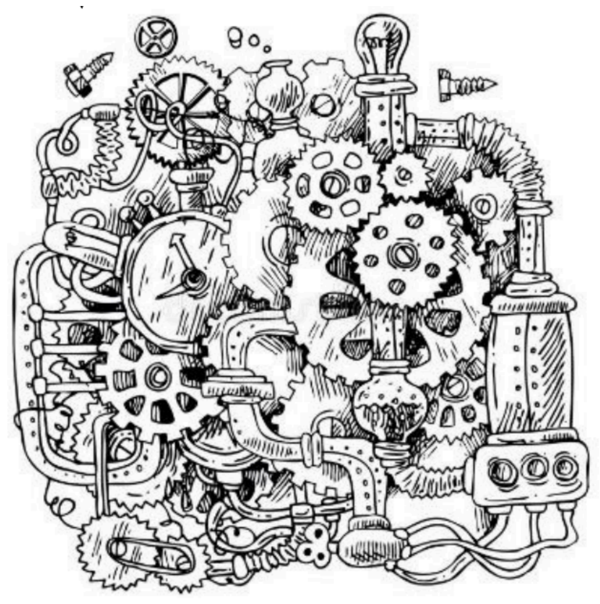
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

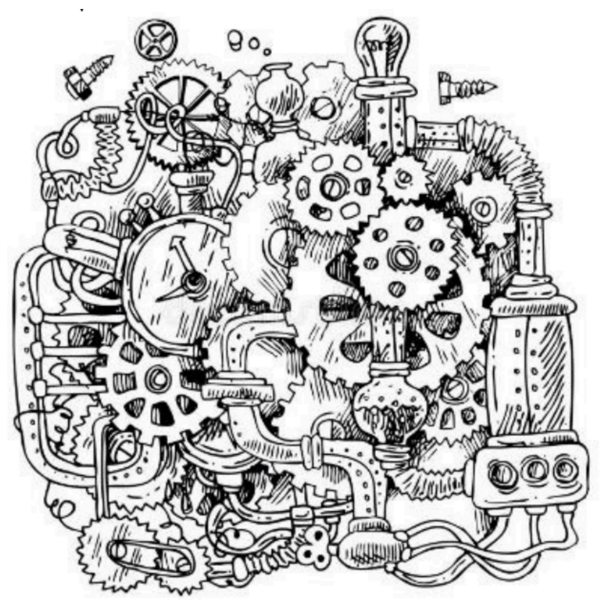
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



Context-free Grammar

S → x | ' ' | '<' | '>'

| rep S S S (**rep x y z** replaces first x in y by z)

| ++ S S (string concatenation)

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

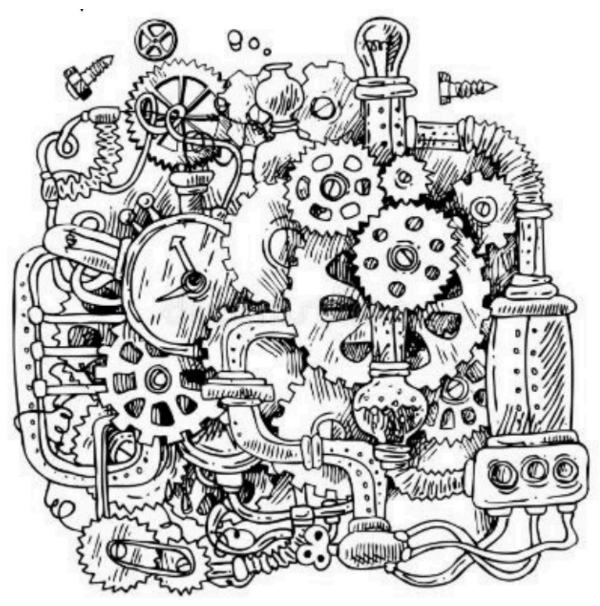
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



Context-free Grammar

S → x | '' | '<' | '>'

| rep S S S (**rep x y z** replaces first x in y by z)

| ++ S S (string concatenation)

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

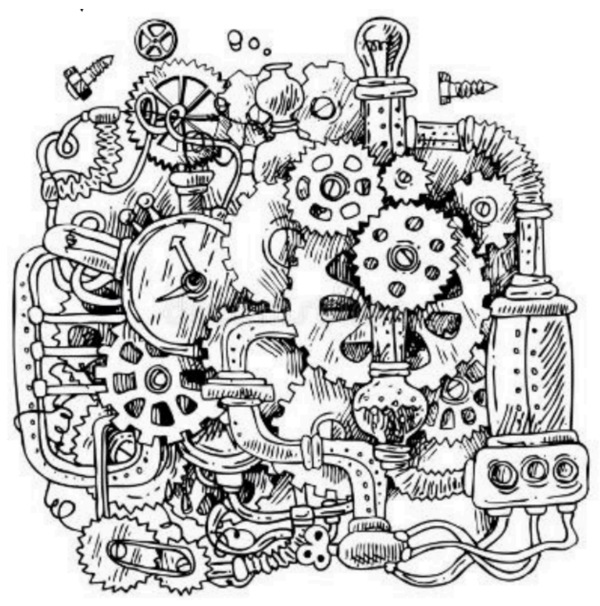
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



Context-free Grammar

S → x | ' ' | '<' | '>'

| rep S S S (**rep x y z** replaces first x in y by z)

| ++ S S (string concatenation)

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

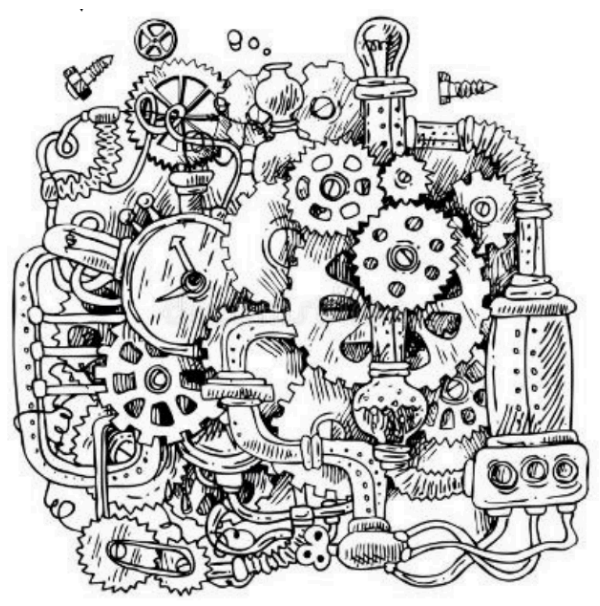
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



Context-free Grammar

S → x | ' ' | '<' | '>'

| rep S S S (**rep x y z** replaces first x in y by z)

| ++ S S (string concatenation)

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

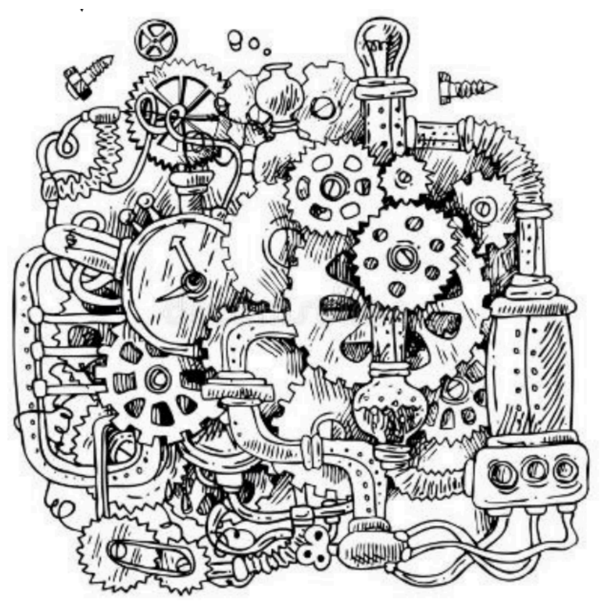
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



Context-free Grammar

S → x | ' ' | '<' | '>'

| rep S S S (**rep x y z** replaces first x in y by z)

| ++ S S (string concatenation)

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

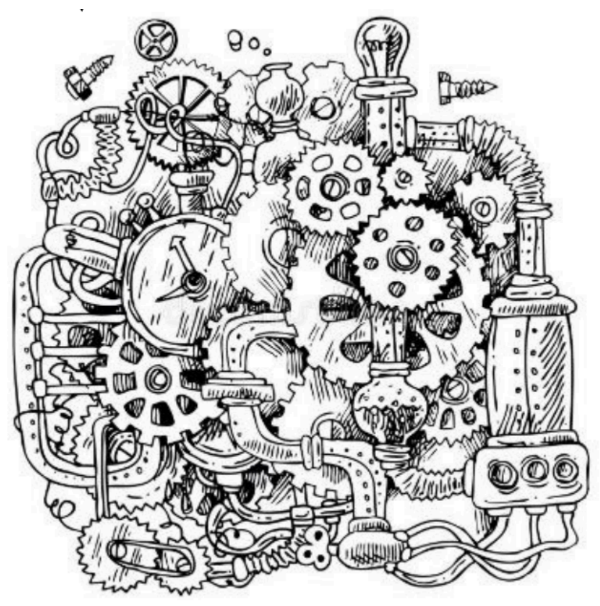
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Synthesizer



Context-free Grammar

S → x | ' ' | '<' | '>'

| rep S S S (**rep x y z** replaces first x in y by z)

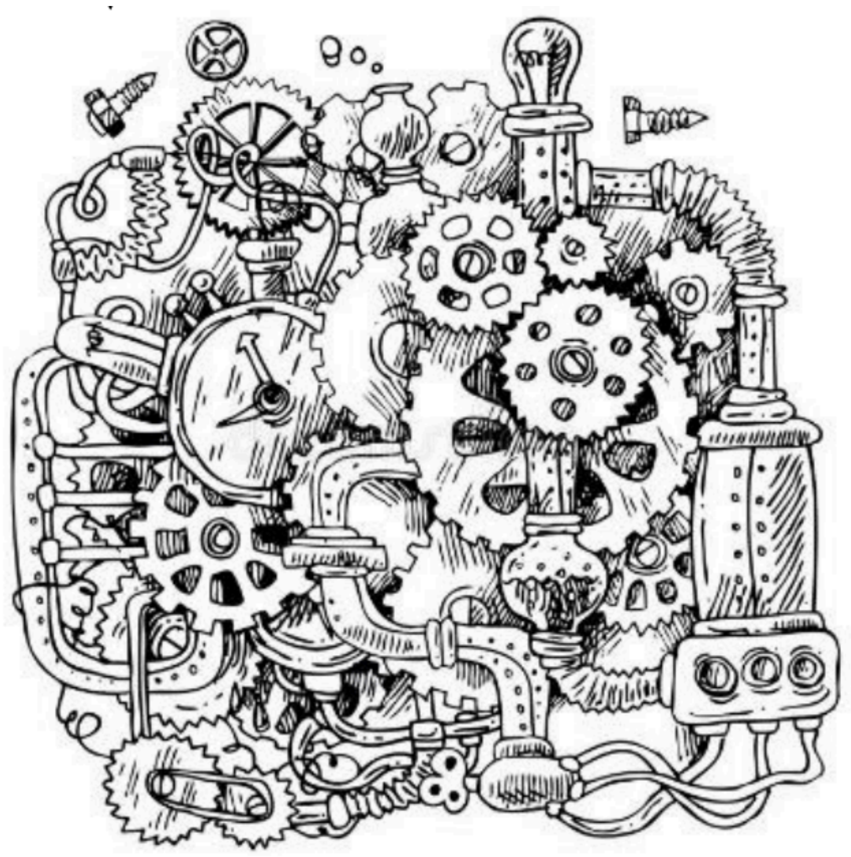
| ++ S S (string concatenation)

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Traditional Program Synthesis

Search strategy: explore programs in order of size

Synthesizer



Specification → [Synthesizer] ← Program Space

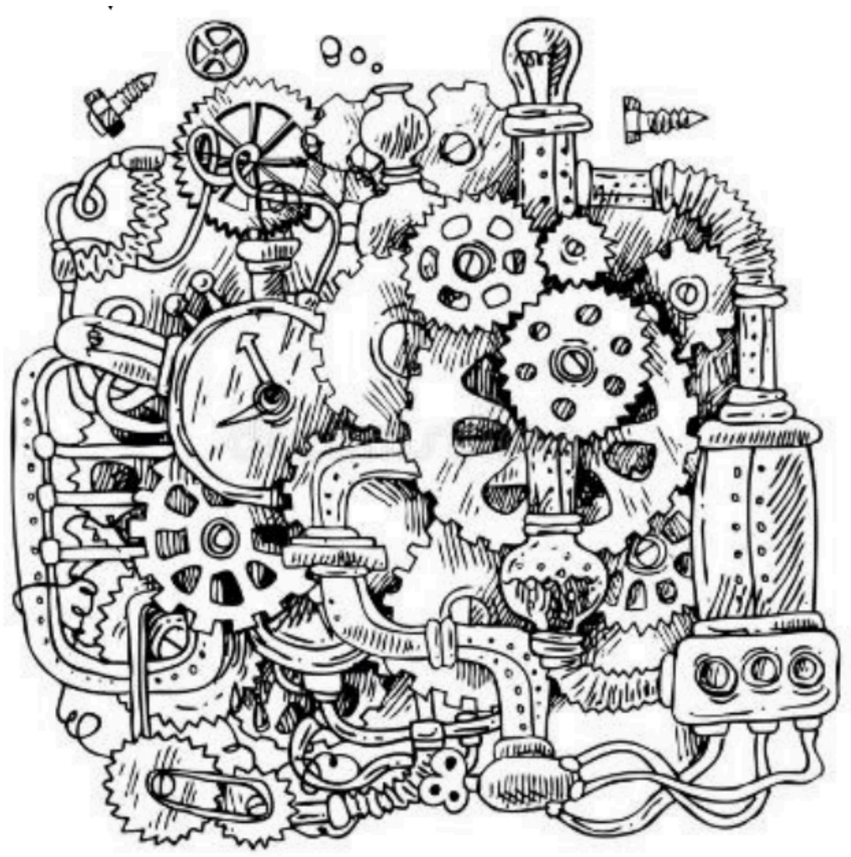
↓

Program

Traditional Program Synthesis

Search strategy: explore programs in order of size

Synthesizer

Specification →  ← Program Space

>30 million programs

↓

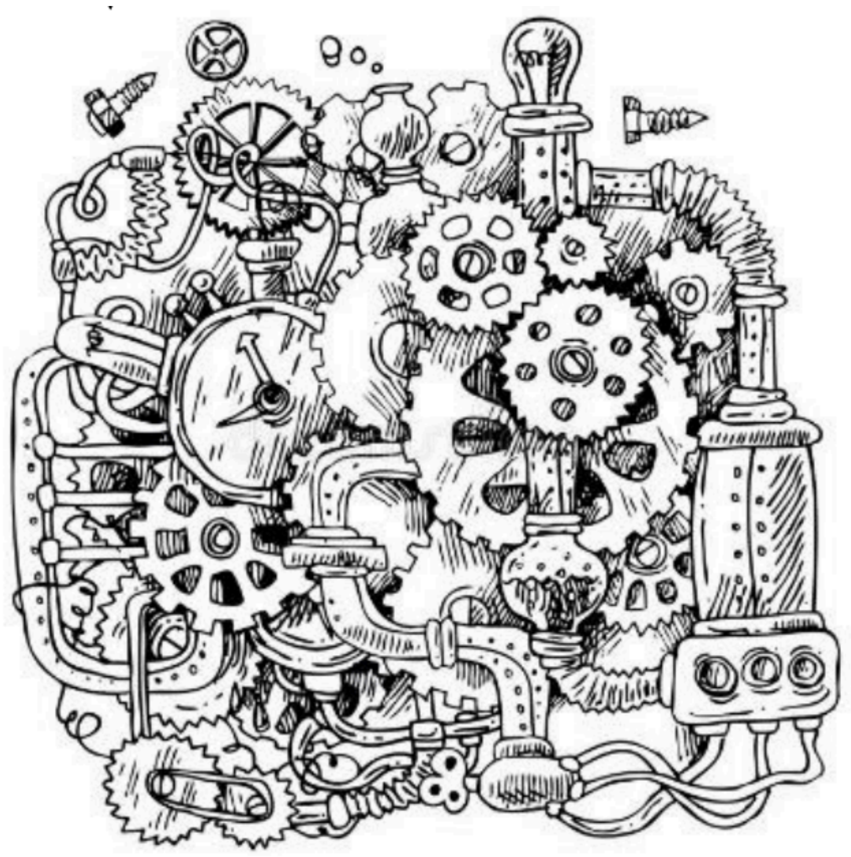
Program

Traditional Program Synthesis

Search strategy: explore programs in order of size

Timeout after 20 minutes

>30 million programs

Specification →  ← Program Space

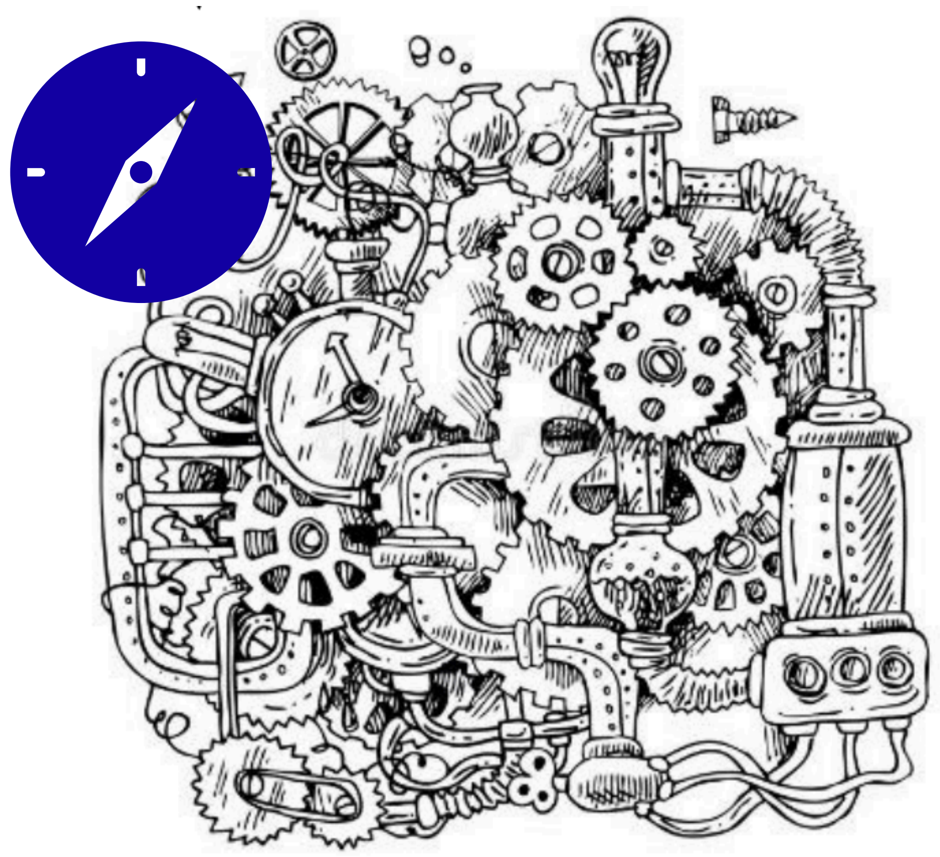
↓

Program

Guided Program Synthesis

Search strategy: explore programs in order of cost[1]

Guided Synthesizer

Specification →  ← Program Space

↓

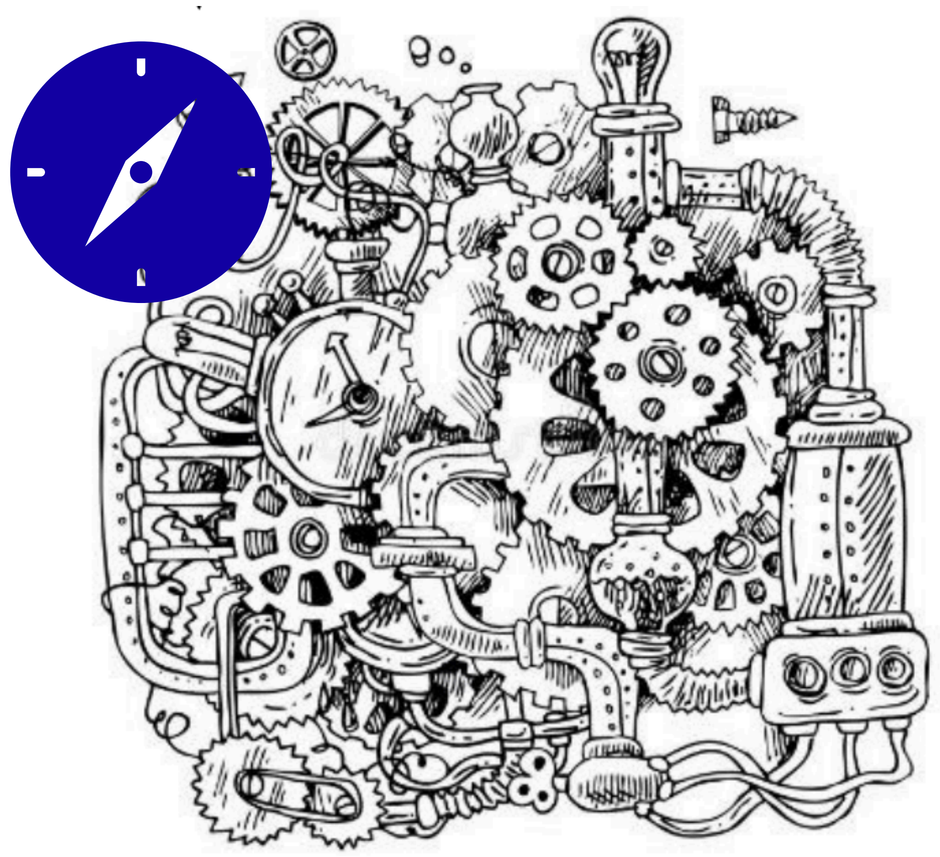
Program

[1] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. PLDI 2018

Guided Program Synthesis

Search strategy: explore programs in order of cost[1]

Guided Synthesizer



Specification → Guided Synthesizer ← Program Space

↓

Program

[1] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. PLDI 2018

Guided Program Synthesis

Search strategy: explore programs in order of cost

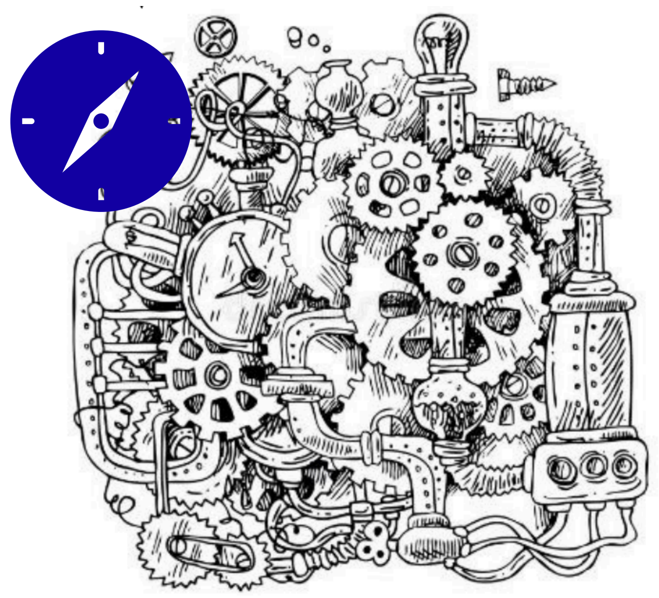
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Guided Synthesizer



Context-free Grammar

S → x | ' ' | '<' | '>'

  | rep S S S

  | ++ S S

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Guided Program Synthesis

Search strategy: explore programs in order of cost

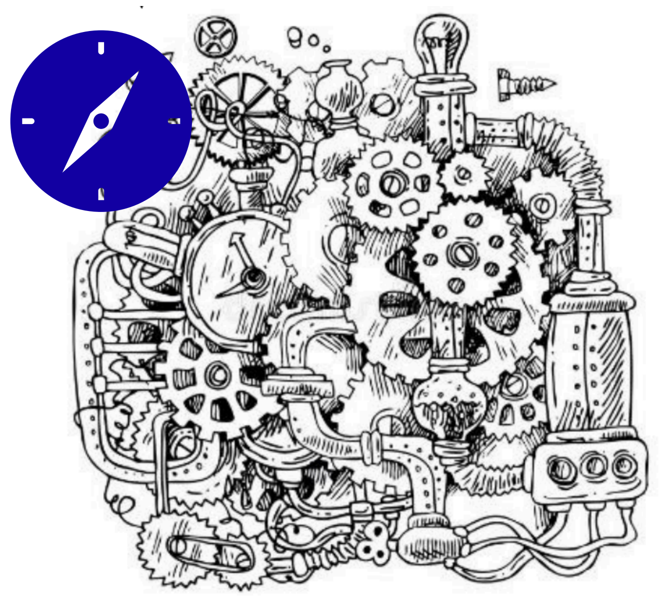
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Guided Synthesizer



PCFG

S → x [$2] | ' ' [$2] | '<' [$2] | '>' [$2]

| rep [$2] S S S

| ++ [$4] S S

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

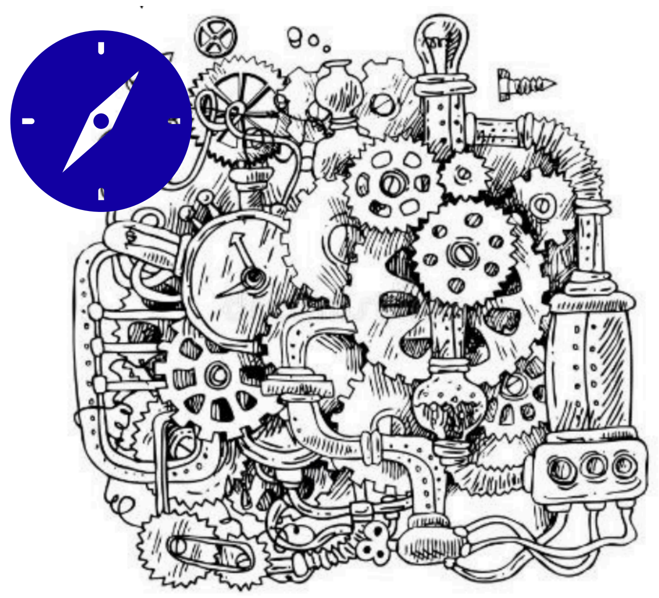
Guided Program Synthesis

Search strategy: explore programs in order of cost

Input-output examples

"<a>"	→	"a"
"<a> <b>"	→	"a b"
"<a> <b> <c>"	→	"a b c"

Guided Synthesizer



PCFG

S → x ($2) | ' ' ($2) | '<' ($2) | '>' ($2)

| rep S S S ($2)

| ++ S S ($4)

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Guided Program Synthesis

Search strategy: explore programs in order of cost

130K programs

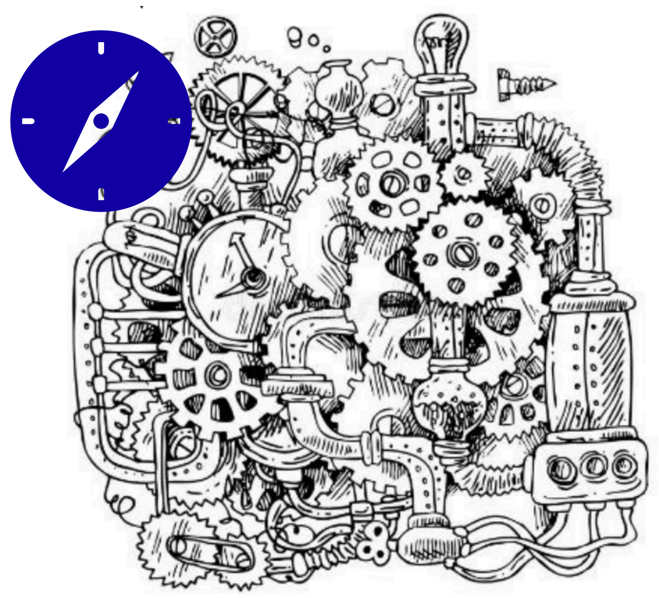
Input-output examples

"<a>" → "a"

"<a> <b>" → "a b"

"<a> <b> <c>" → "a b c"

Guided Synthesizer



PCFG

S → x ($2) | ' ' ($2) | '<' ($2) | '>' ($2)

| rep S S S ($2)

| ++ S S ($4)

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Guided Program Synthesis: Challenges

Search strategy: explore programs in order of cost

1. How to learn useful costs?

2. How to guide search given costs?

Guided Program Synthesis: Challenges

1. How to learn useful costs?

2. How to guide search given costs?

Guided Program Synthesis: Challenges

1. How to learn useful costs?

2. How to guide search given costs?

Prior Work	Our Technique
1. Data-driven learning	1. Just-in-time learning from partial solutions

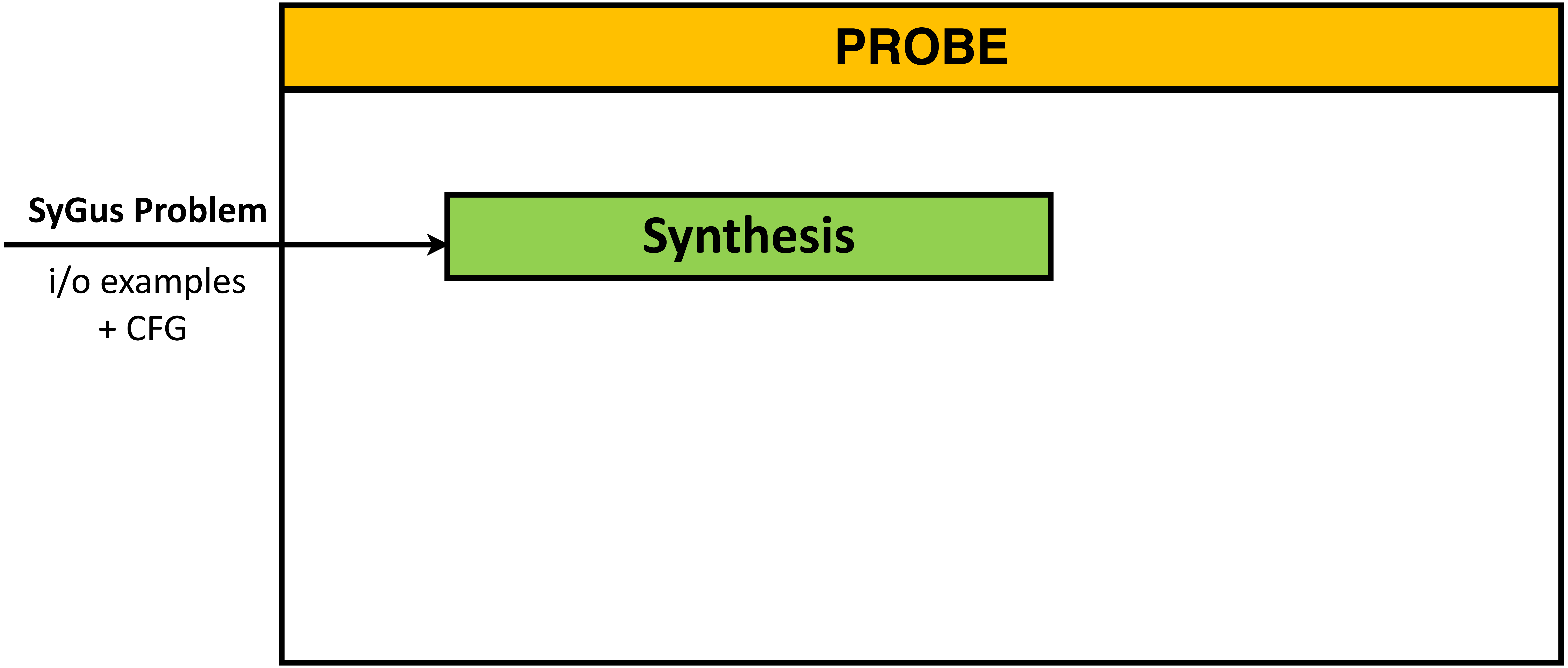
Guided Program Synthesis: Challenges

1. How to learn useful costs?

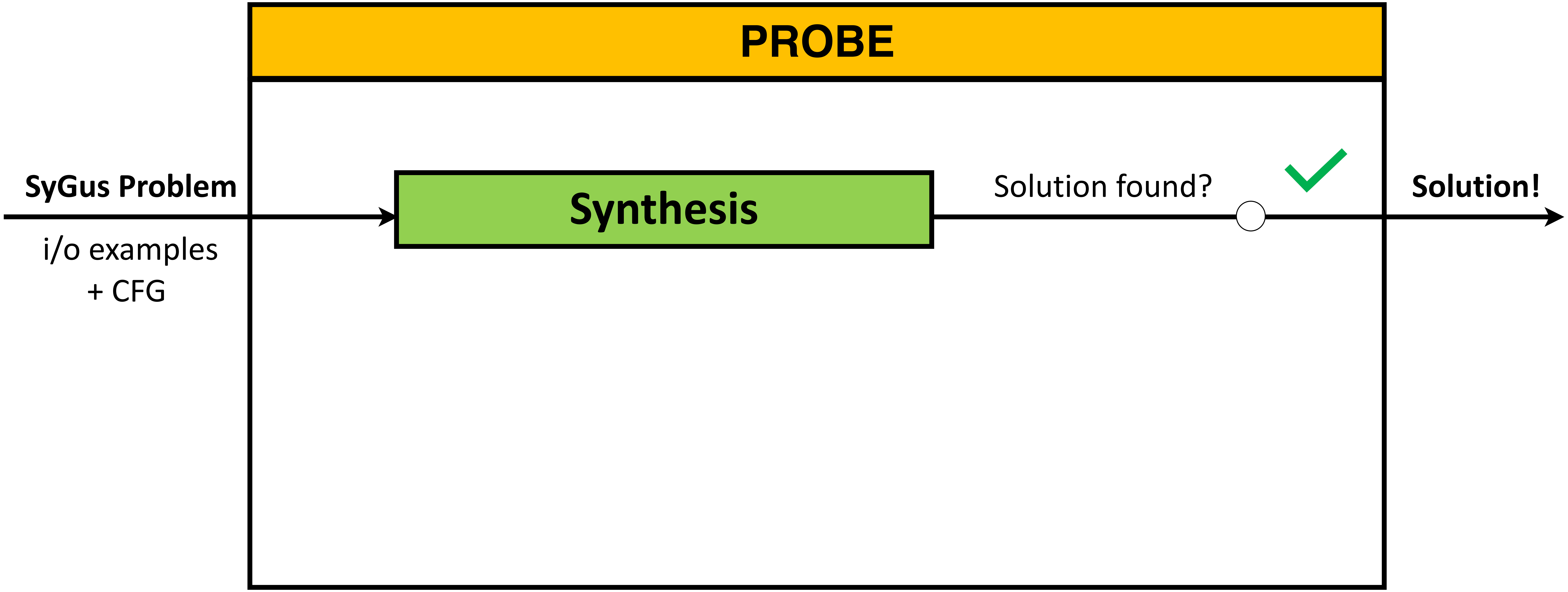
2. How to guide search given costs?

Prior Work	Our Technique
1. Data-driven learning	1. Just-in-time learning from partial solutions
2. Guided Top-down search	2. Guided Bottom-up search

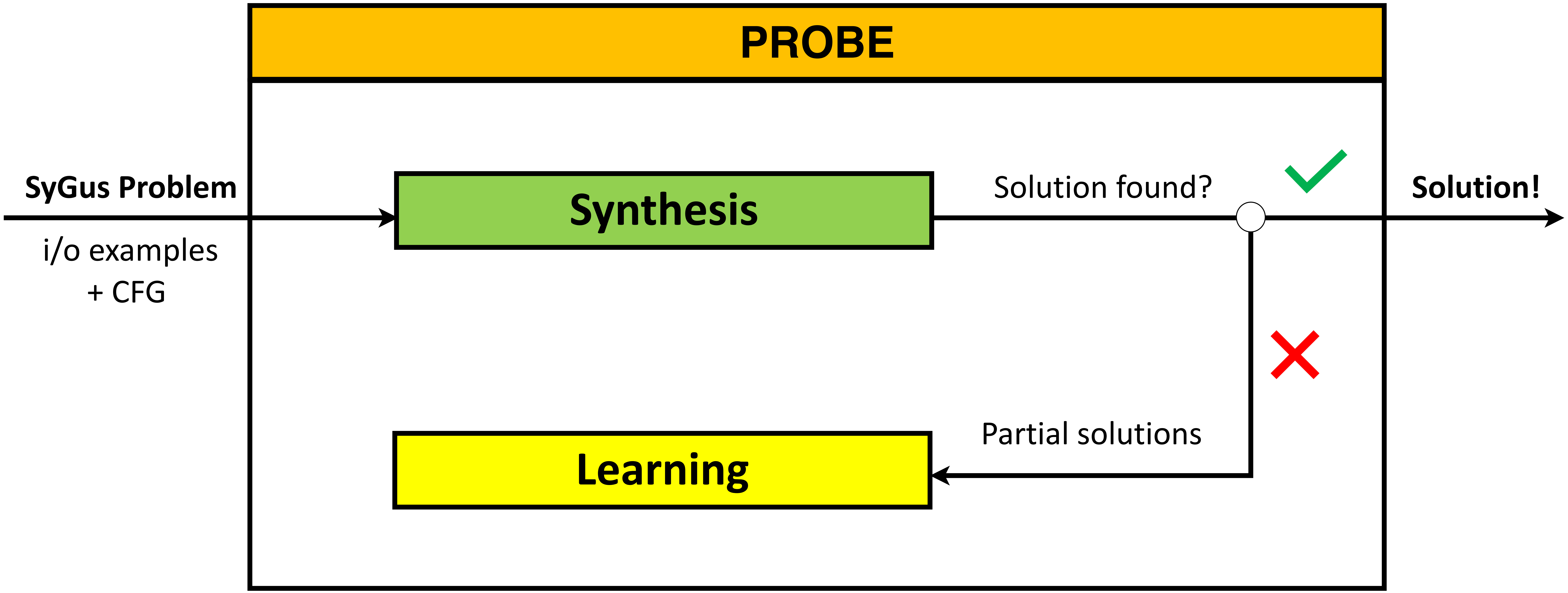
PROBE Overview



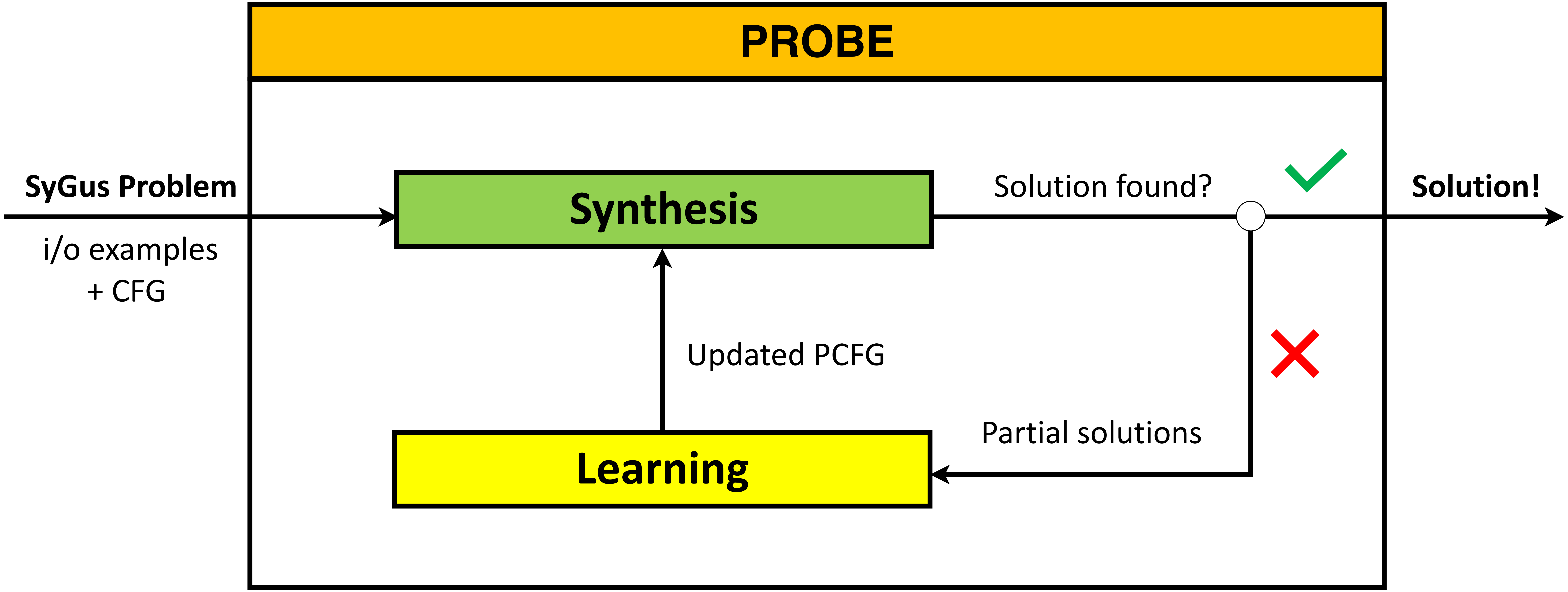
PROBE Overview



PROBE Overview



PROBE Overview



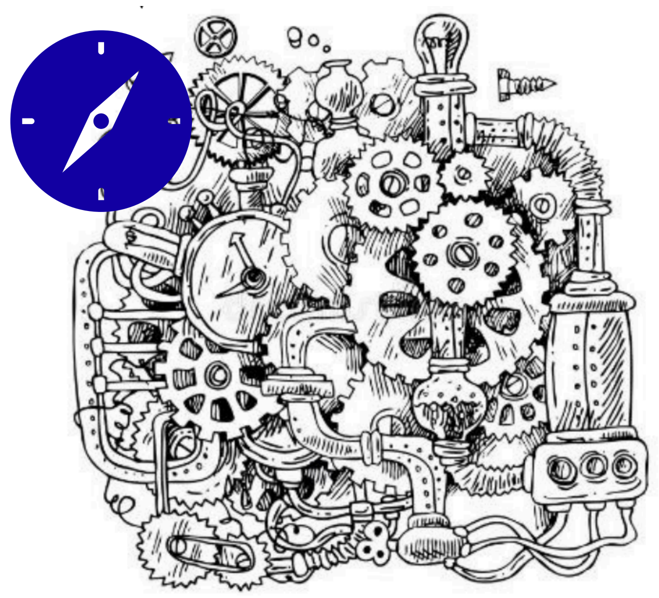
SyGuS Example (remove-angles)

Goal : remove angle brackets < and > from the input string x

Input-output examples

"<a>"	→	"a"
"<a> <b>"	→	"a b"
"<a> <b> <c>"	→	"a b c"

Guided Synthesizer



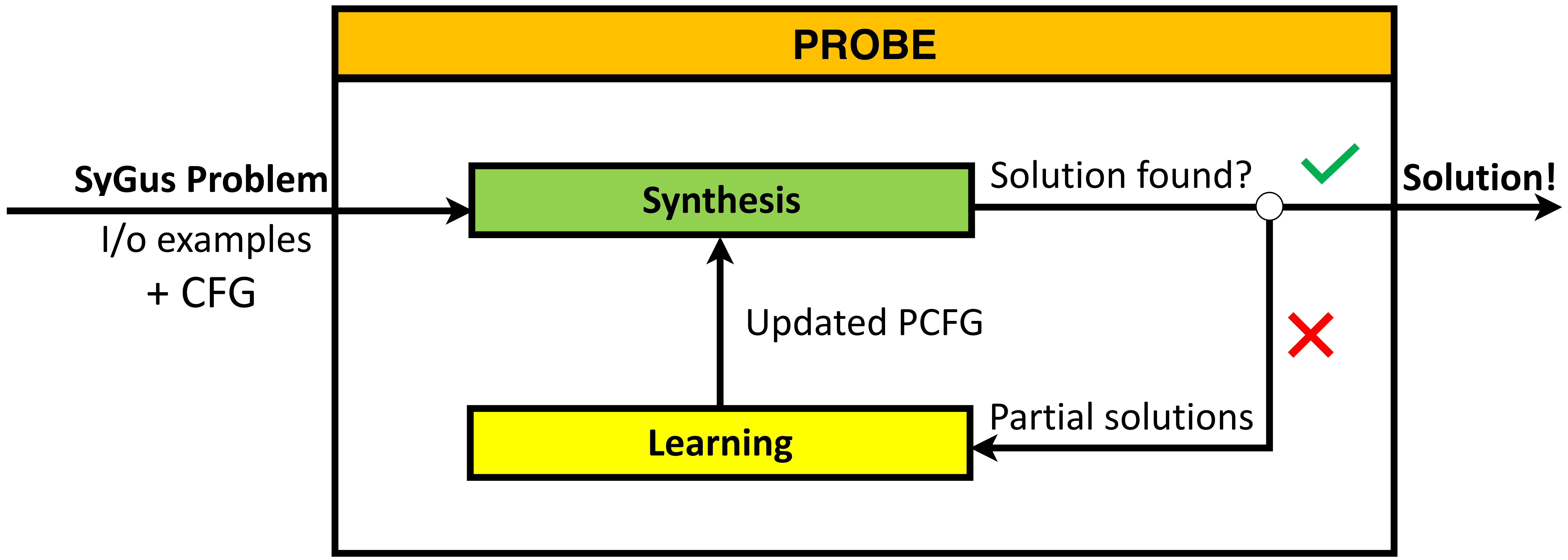
PCFG

S → x ($2) | ' ' ($2) | '<' ($2) | '>' ($2)
| rep S S S ($2)
| ++ S S ($4)

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

PROBE finds solution in 5 seconds!

Talk Outline

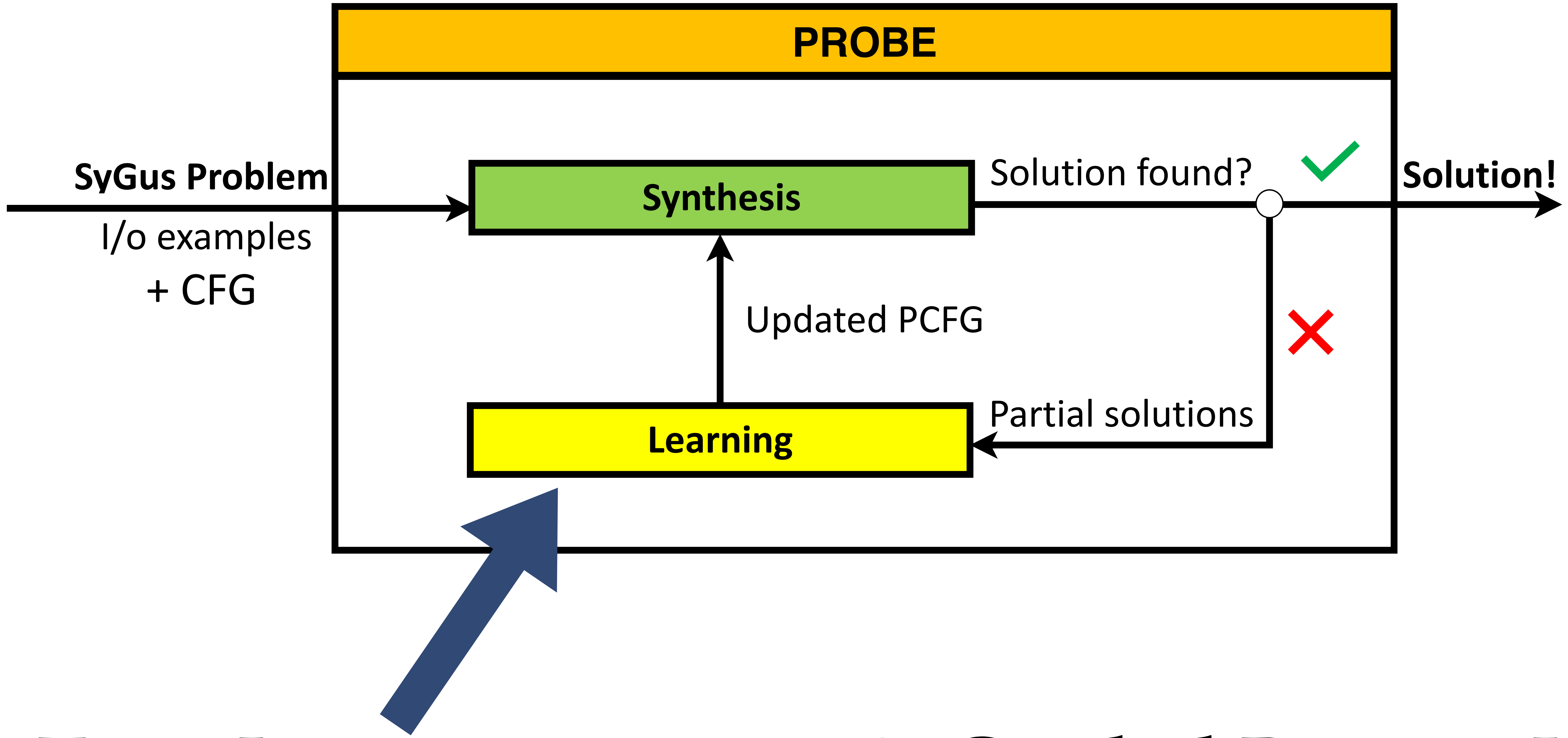


1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

Talk Outline

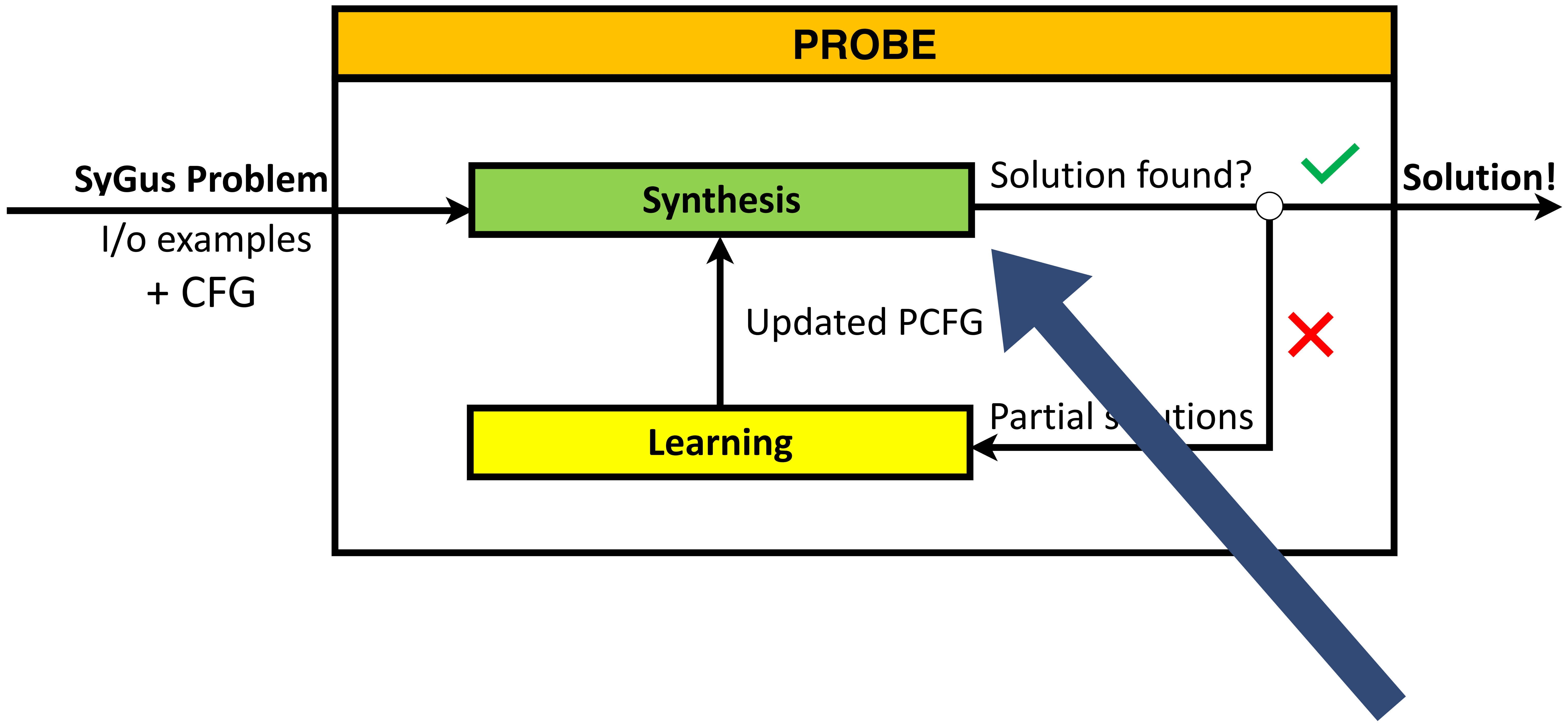


1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

Talk Outline

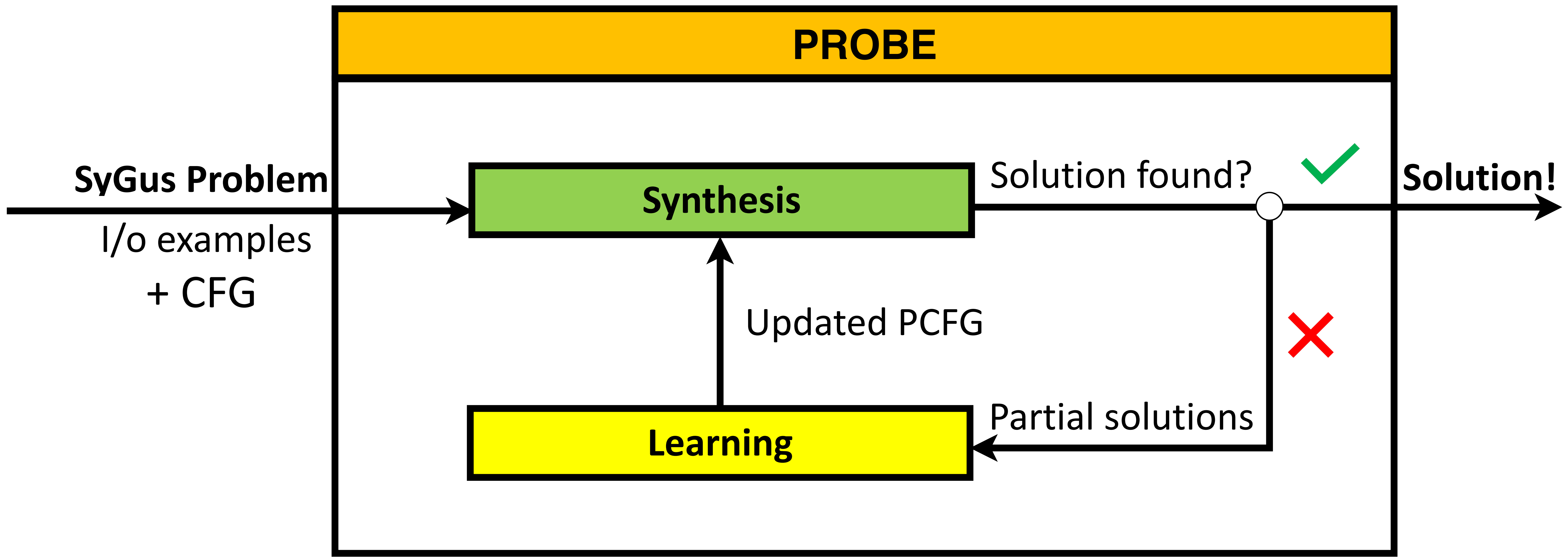


1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

Talk Outline

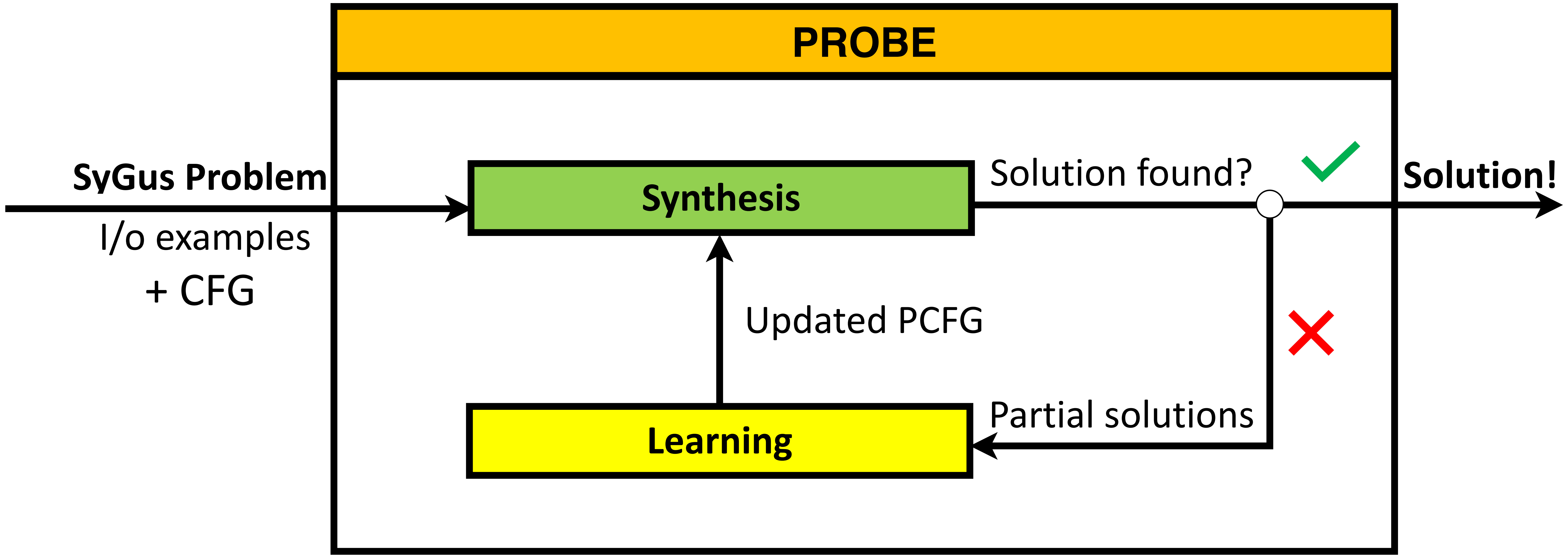


1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

Talk Outline



1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

Our Solution: Just-in-Time Learning

Idea: partial solutions are similar in structure to the solution

Our Solution: Just-in-Time Learning

Idea: partial solutions are similar in structure to solution

Input-output examples

e1	"<a>"	→	"a"
e2	"<a> <b>"	→	"a b"
e3	"<a> <b> <c>"	→	"a b c"

Uniform PCFG

S → x ($3) | ' ' ($3) | '<' ($3) | '>' ($3)

rep ($3) S S S

++ ($3) S S

Solution: (rep (rep (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: partial solutions are similar in structure to solution

Input-output examples

e1	"<a>"	→	"a"
e2	"<a> <b>"	→	"a b"
e3	"<a> <b> <c>"	→	"a b c"

Uniform PCFG

	$3		$3		$3		$3
S →	x	|	' '	|	'<'	|	'>'
	rep S S S ($3)						
	++ S S ($3)						

replace-2: (rep (rep x '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: reward productions that appear in partial solutions

Uniform PCFG

		$3		$3		$3		$3
S	→	x	|	' '	|	'<'	|	'>'
		$3						
		rep S S S						
		$3						
		++ S S						

replace-2: (rep (rep x '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: reward productions that appear in partial solutions

Uniform PCFG

S → x ($3) | ' ' ($3) | '<' ($3) | '>' ($3)

rep ($3) S S S

++ ($3) S S

Updated PCFG

S → x ($2) | ' ' ($2) | '<' ($2) | '>' ($2)

rep ($2) S S S

++ ($3) S S

replace-2: (rep (rep x '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: reward productions that appear in partial solutions

Input-output examples

e1	"<a>"	→	"a"
e2	"<a> <b>"	→	"a b"
e3	"<a> <b> <c>"	→	"a b c"

Updated PCFG

S → x ($2) | ' ' ($2) | '<' ($2) | '>' ($2)

rep ($2) S S S

++ ($3) S S

replace-2: (rep (rep x '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: reward productions that appear in partial solutions

Input-output examples

e1	"<a>"	→	"a"
e2	"<a> <b>"	→	"a b"
e3	"<a> <b> <c>"	→	"a b c"

Updated PCFG

S → x ($2) | ' ' ($2) | '<' ($2) | '>' ($2)

rep S S S ($2)

++ S S ($3)

replace-2: (rep (rep x '<' ' ') '>' ' ')

replace-4: (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: reward productions that appear in partial solutions

Updated PCFG

```
       $2    $2    $2    $2
S  →   x  |  ' '  | '<' | '>'

       $2
      rep  S S S

       $3
      ++  S S
```

replace-4: (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Our Solution: Just-in-Time Learning

Idea: reward productions that appear in partial solutions

Updated PCFG

S → x [$2] | ' ' [$2] | '<' [$2] | '>' [$2]

rep [$2] S S S

++ [$3] S S

Biased PCFG

S → x [$2] | ' ' [$2] | '<' [$2] | '>' [$2]

rep [$2] S S S

++ [$4] S S

replace-4: (rep (rep (rep (rep x '<' ' ') '>' ' ') '<' ' ') '>' ' ')

Partial Solution Selection

Challenge: Too many redundant partial solutions

3500 even for the tiny grammar!

Partial Solution Selection

Challenge: Too many redundant partial solutions

3500 even for the tiny grammar!

Redundant Partial Solution: (rep (rep (rep (++ x '<') '<' ' ') '<' ' ') '>' ' ')

replace-2: (rep (rep x '<' ' ') '>' ' ')

Partial Solution Selection

Challenge: Too many redundant partial solutions

3500 even for the tiny grammar!

Redundant Partial Solution: (rep (rep (rep (++ x '<') '<' ' ') '<' ' ') '>' ' ')

replace-2: (rep (rep x '<' ' ') '>' ' ')

Partial Solution Selection

Challenge: Too many redundant partial solutions

3500 even for the tiny grammar!

Redundant Partial Solution: (rep (rep (rep (++ x '<') '<' ' ') '<' ' ') '>' ' ')

replace-2: (rep (rep x '<' ' ') '>' ' ')

Partial Solution Selection

Challenge: Too many redundant partial solutions

3500 even for the tiny grammar!

Redundant Partial Solution: (rep (rep (rep (++ x '<') '<' ' ') '<' ' ') '>' ' ')

Observation: Avoid rewarding irrelevant partial solutions

Partial Solution Selection

Challenge: Too many redundant partial solutions

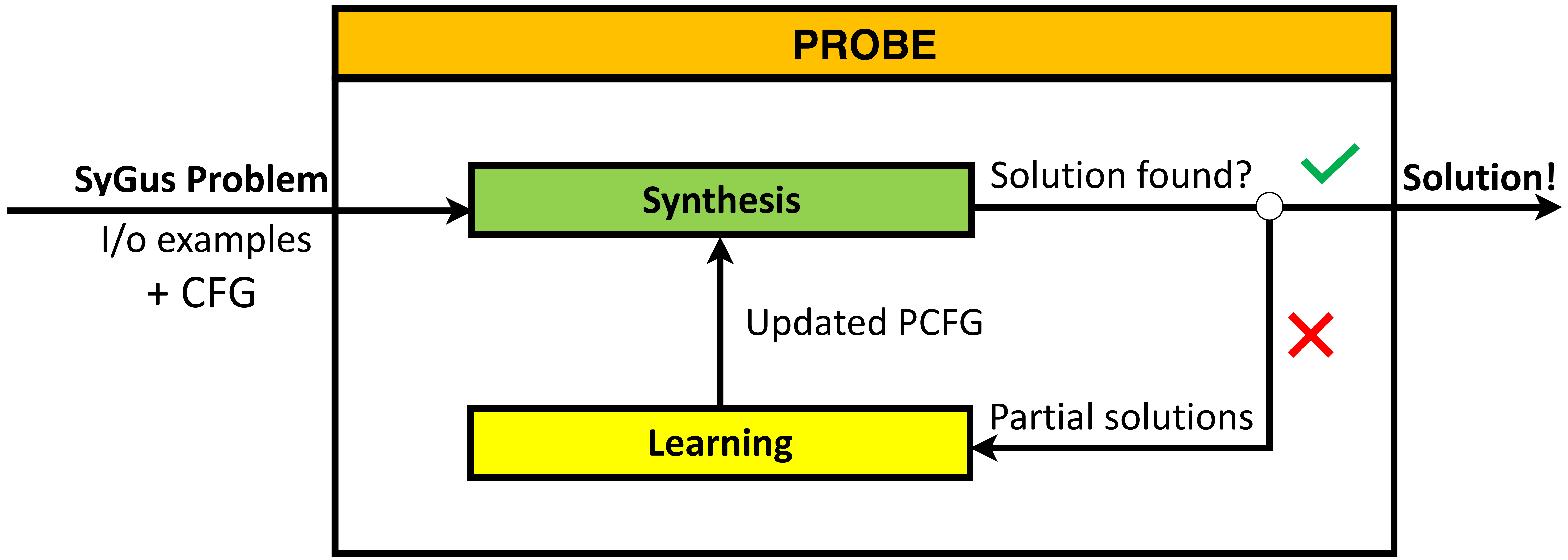
3500 even for the tiny grammar!

Redundant Partial Solution: (rep (rep (rep (++ x '<') '<' ' ') '<' ' ') '>' ' ')

Observation: Avoid rewarding irrelevant partial solutions

Idea: cheapest partial solutions that satisfy new subset of examples

Talk Outline



1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

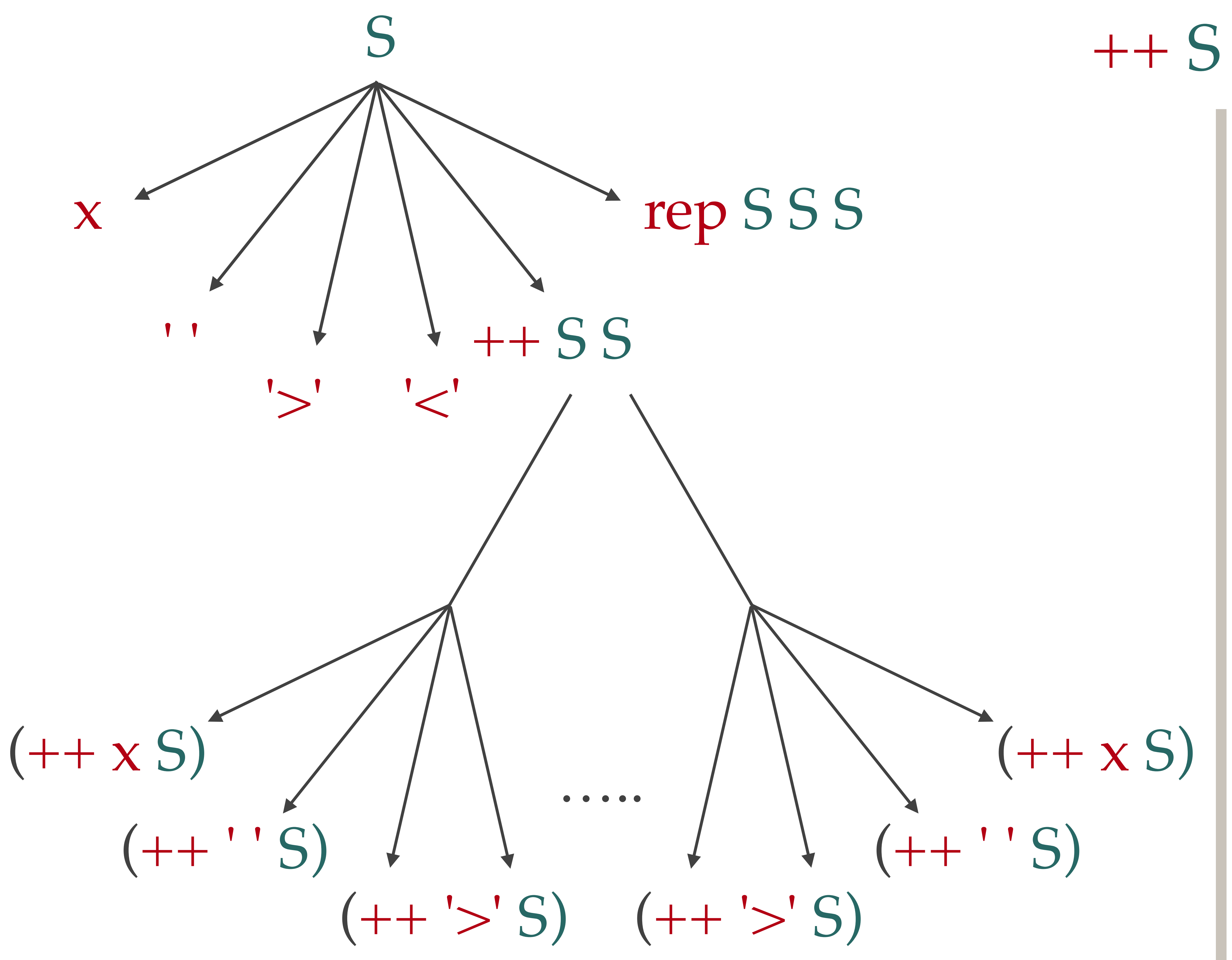
Unguided Search Techniques

S → x | ' ' | '<' | '>'

rep S S S

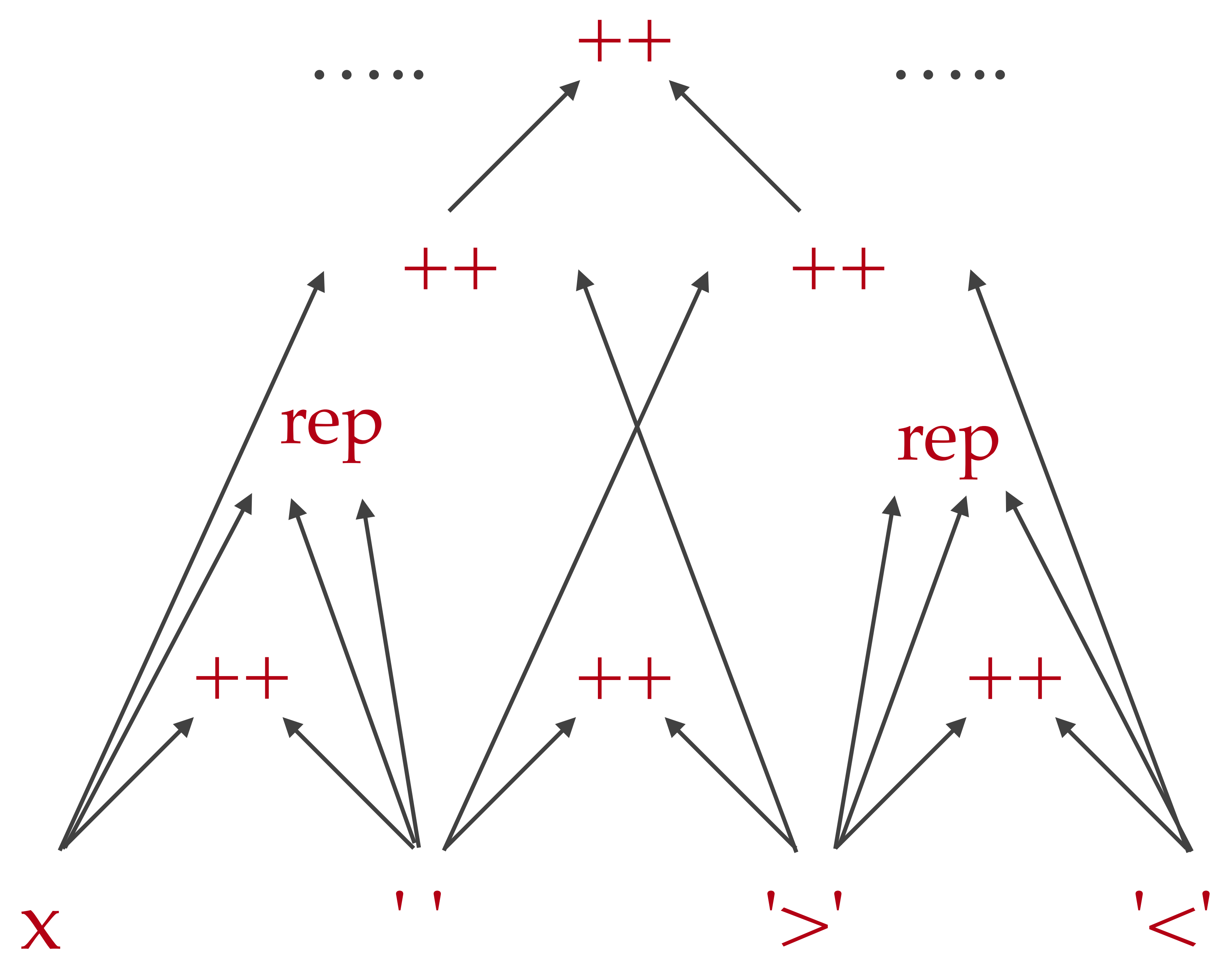
++ S S

Top-down search



Bottom-up search

Observational Equivalence Reduction



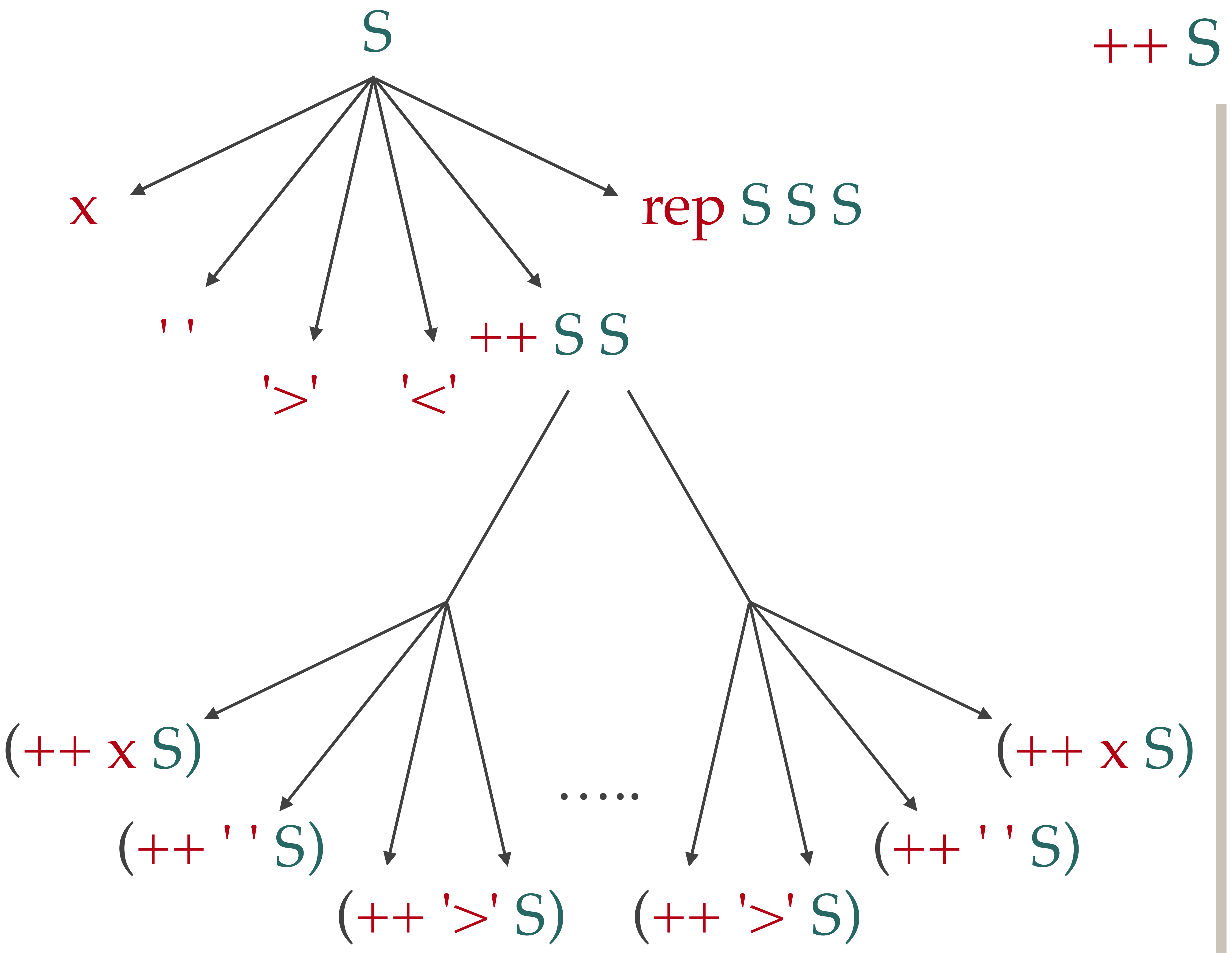
Unguided Search Techniques

S → x | ' ' | '<' | '>'

rep S S S

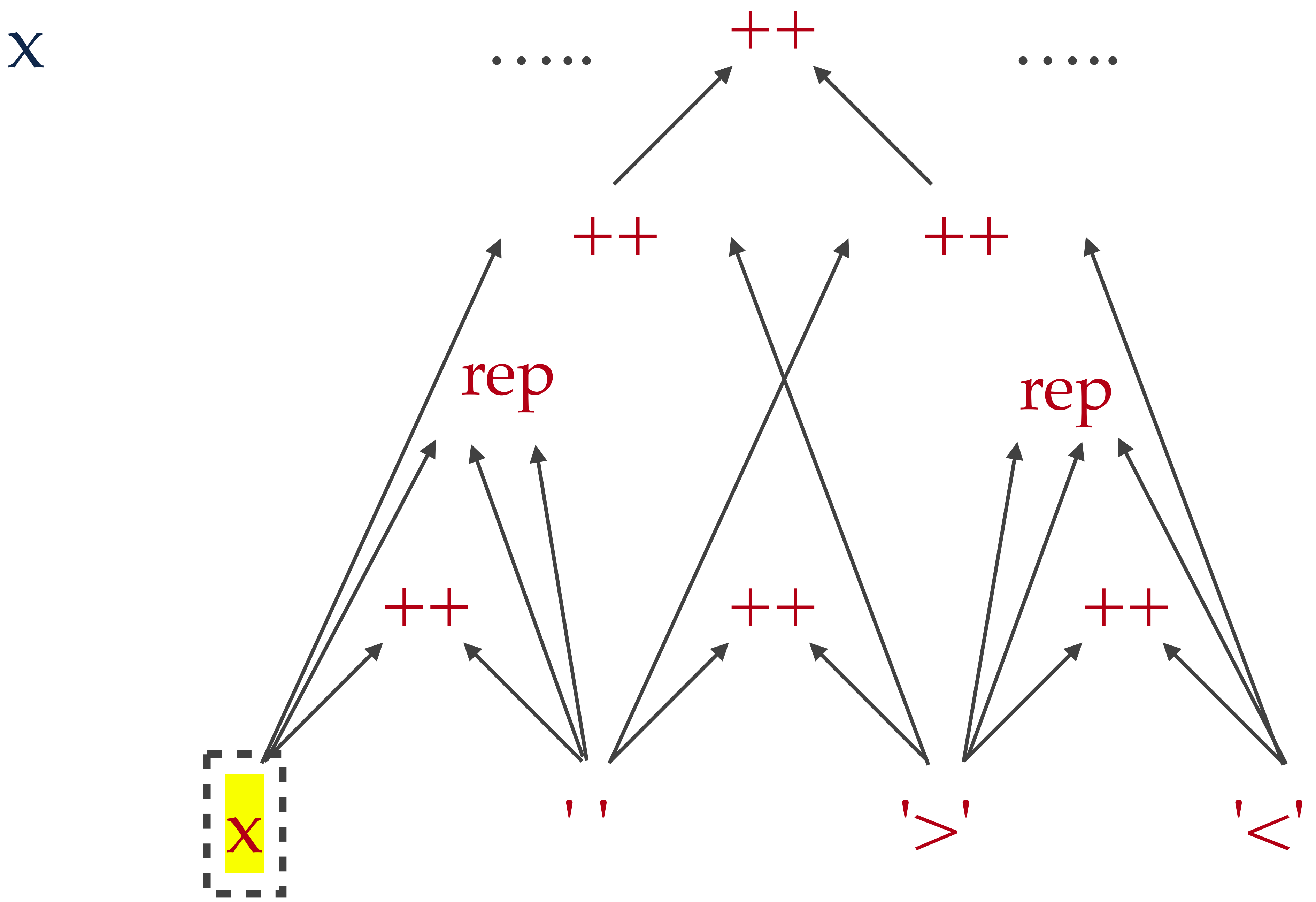
++ S S

Top-down search



Bottom-up search

Observational Equivalence Reduction



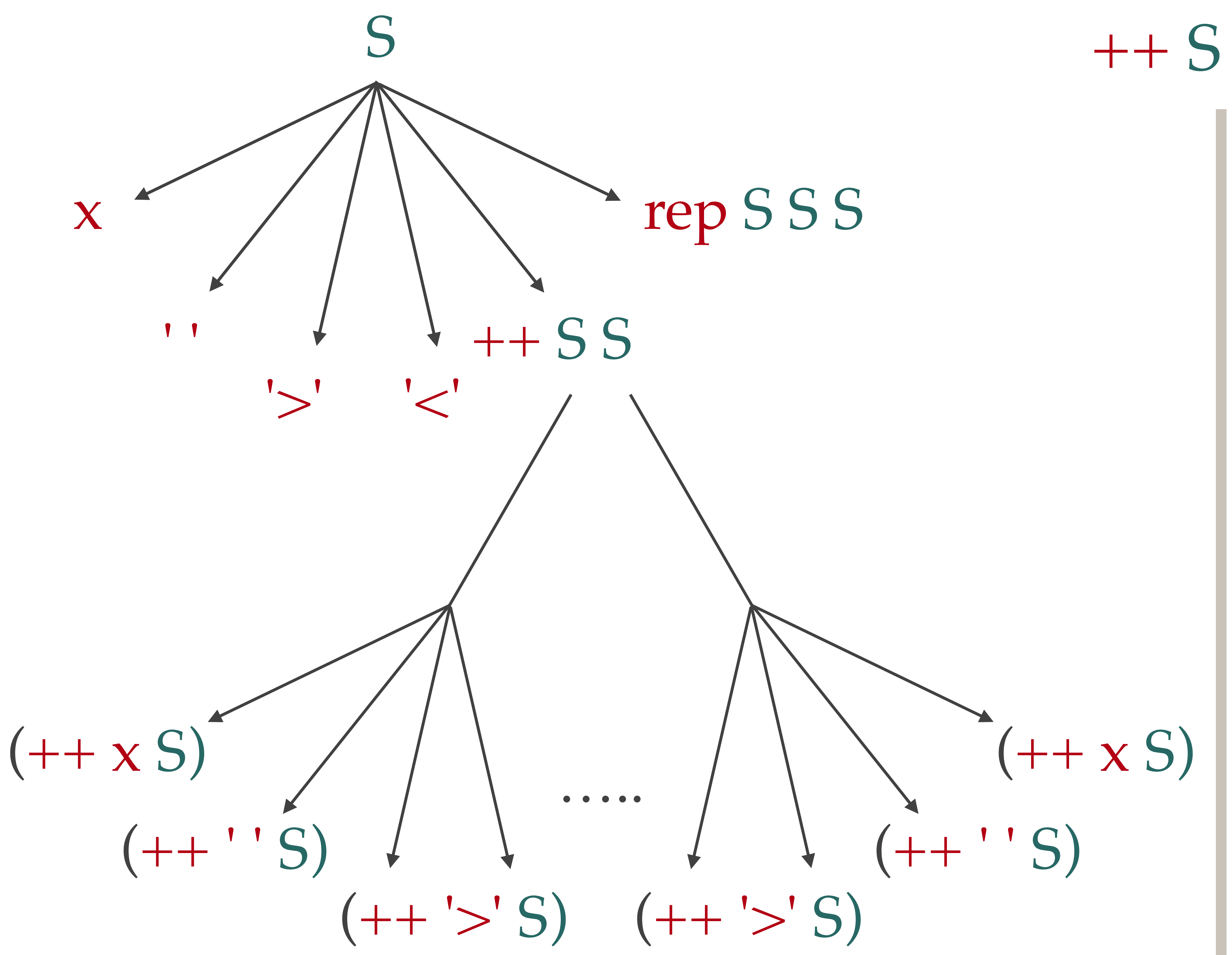
Unguided Search Techniques

S → x | ' ' | '<' | '>'

rep S S S

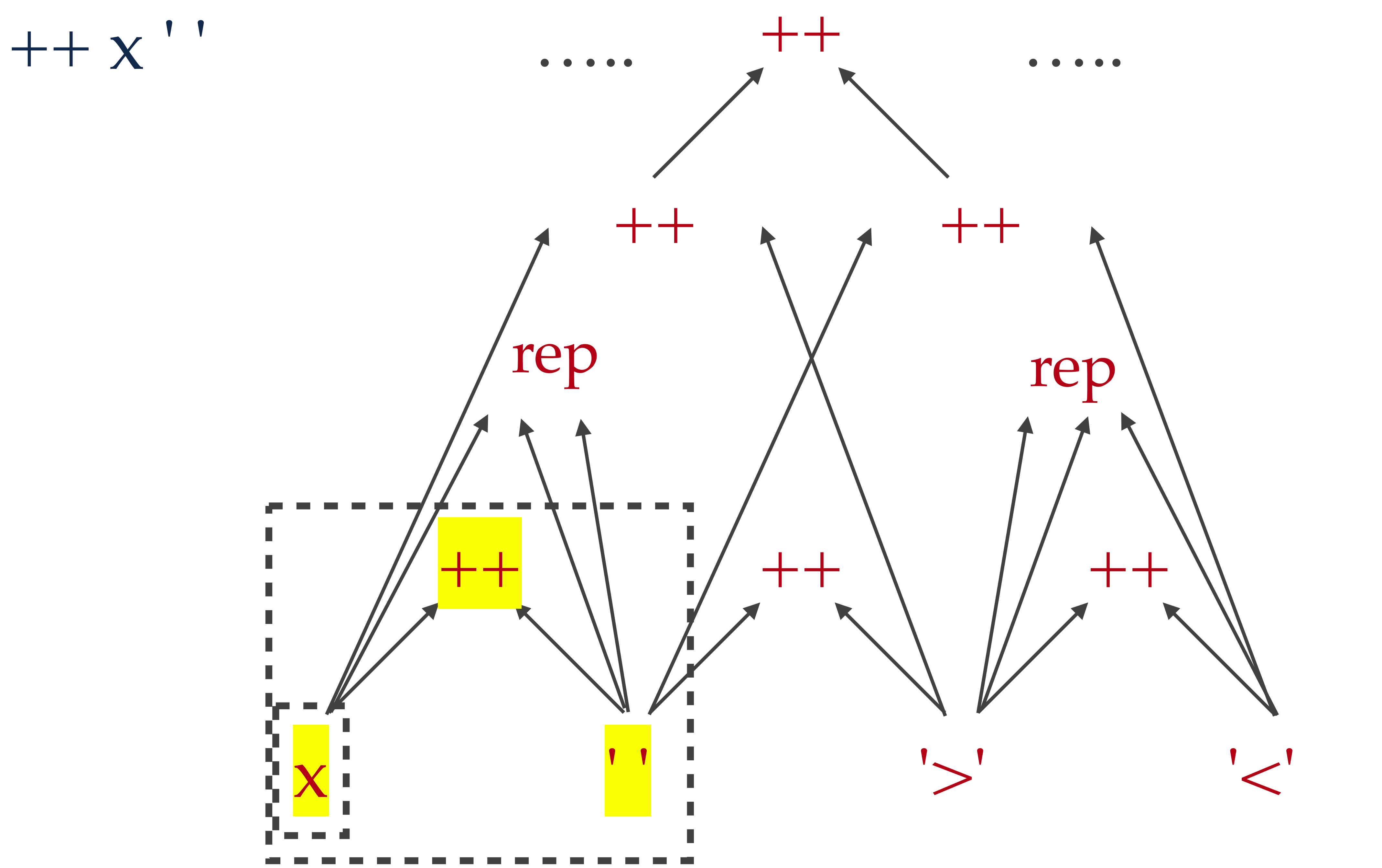
++ S S

Top-down search



Bottom-up search

Observational Equivalence Reduction



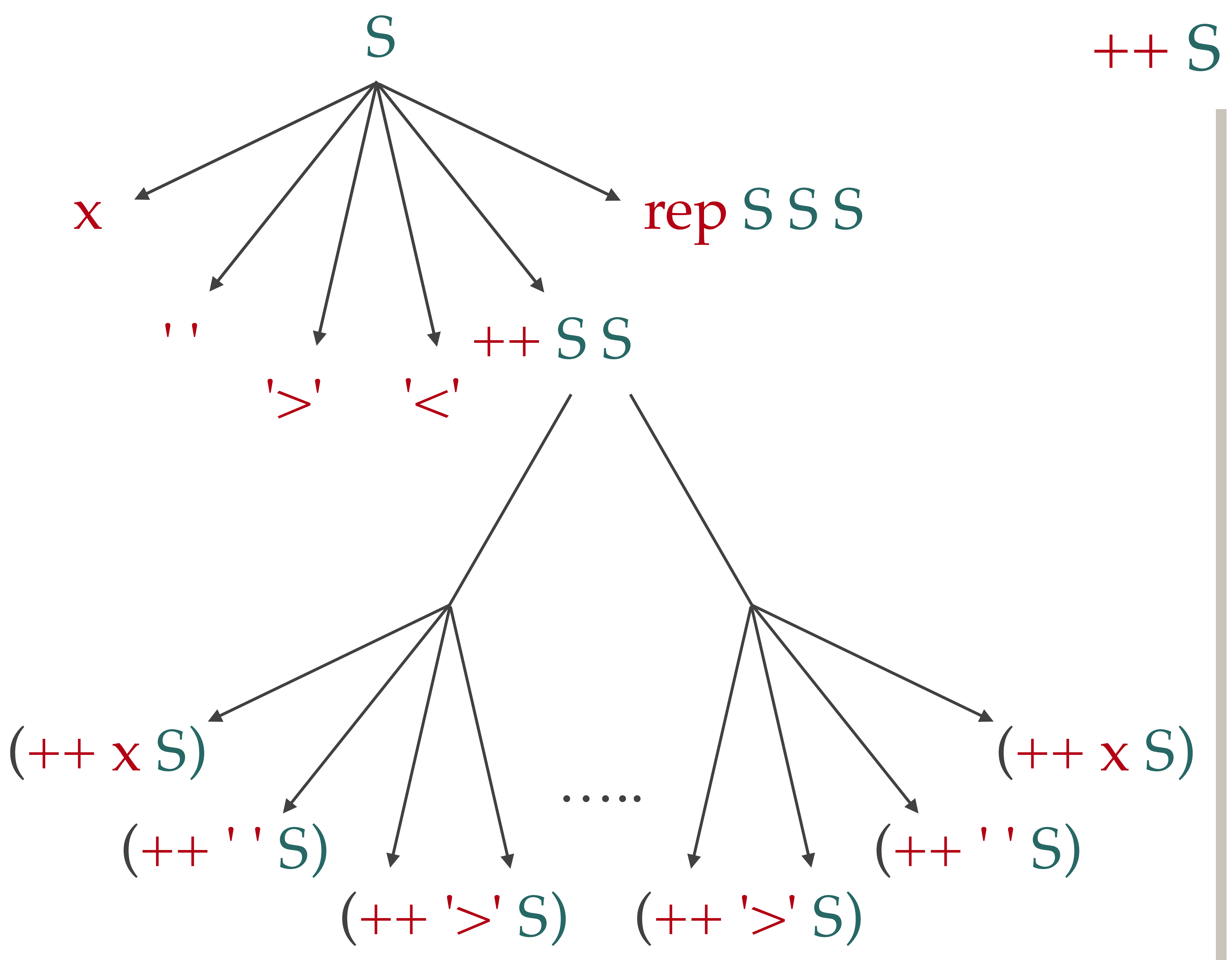
Unguided Search Techniques

S → x | ' ' | '<' | '>'

rep S S S

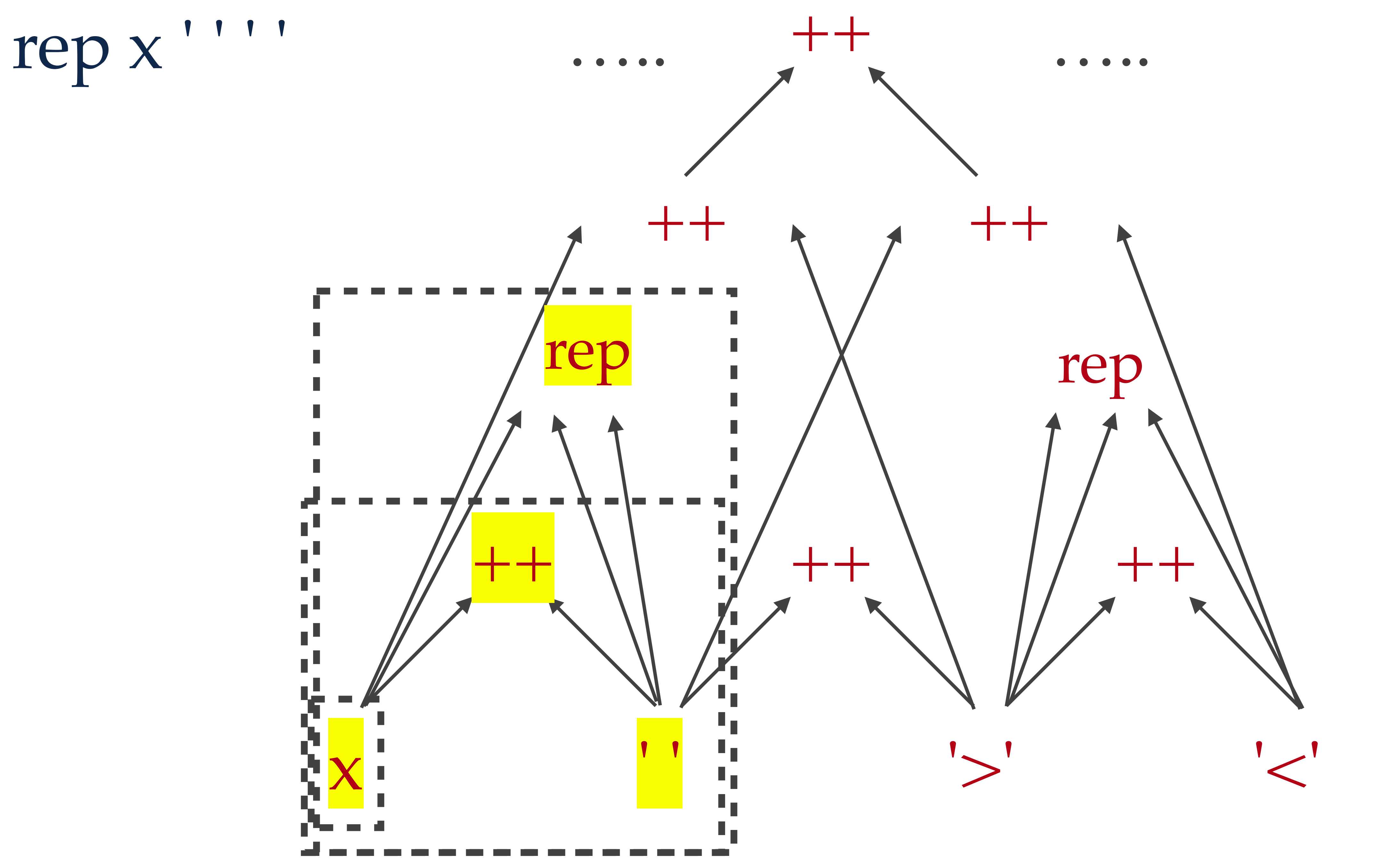
++ S S

Top-down search



Bottom-up search

Observational Equivalence Reduction



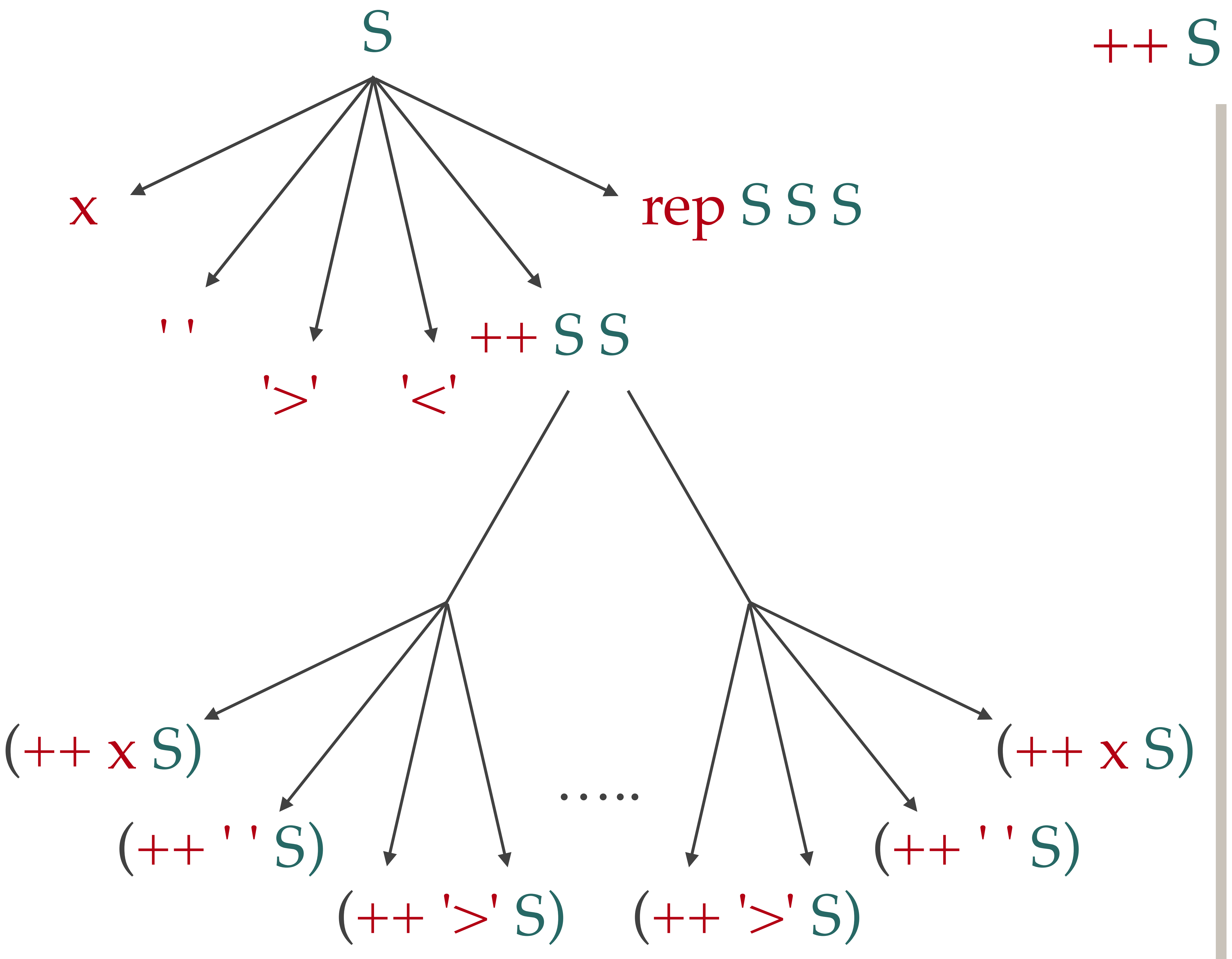
Unguided Search Techniques

S → x | ' ' | '<' | '>'

rep S S S

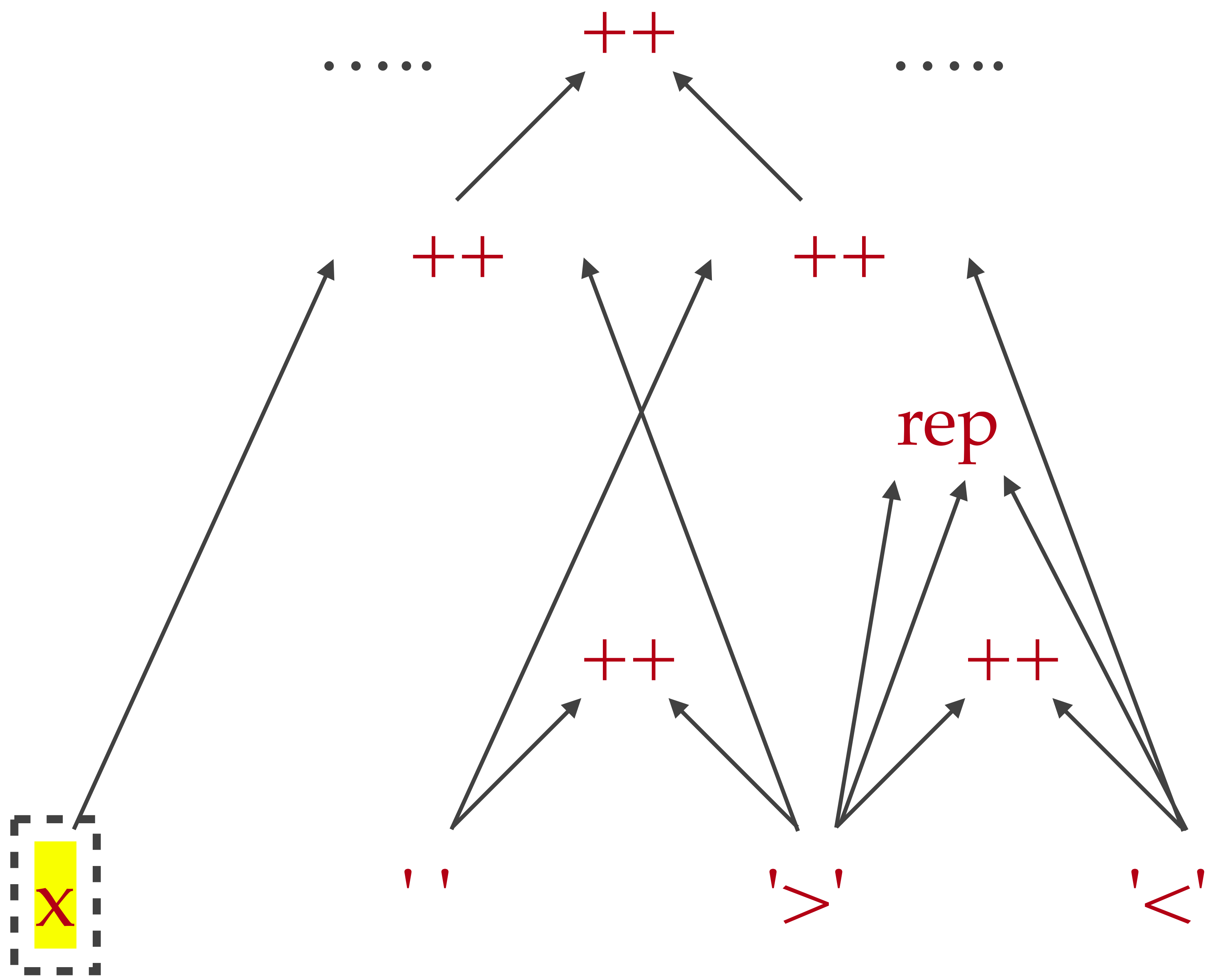
++ S S

Top-down search



Bottom-up search

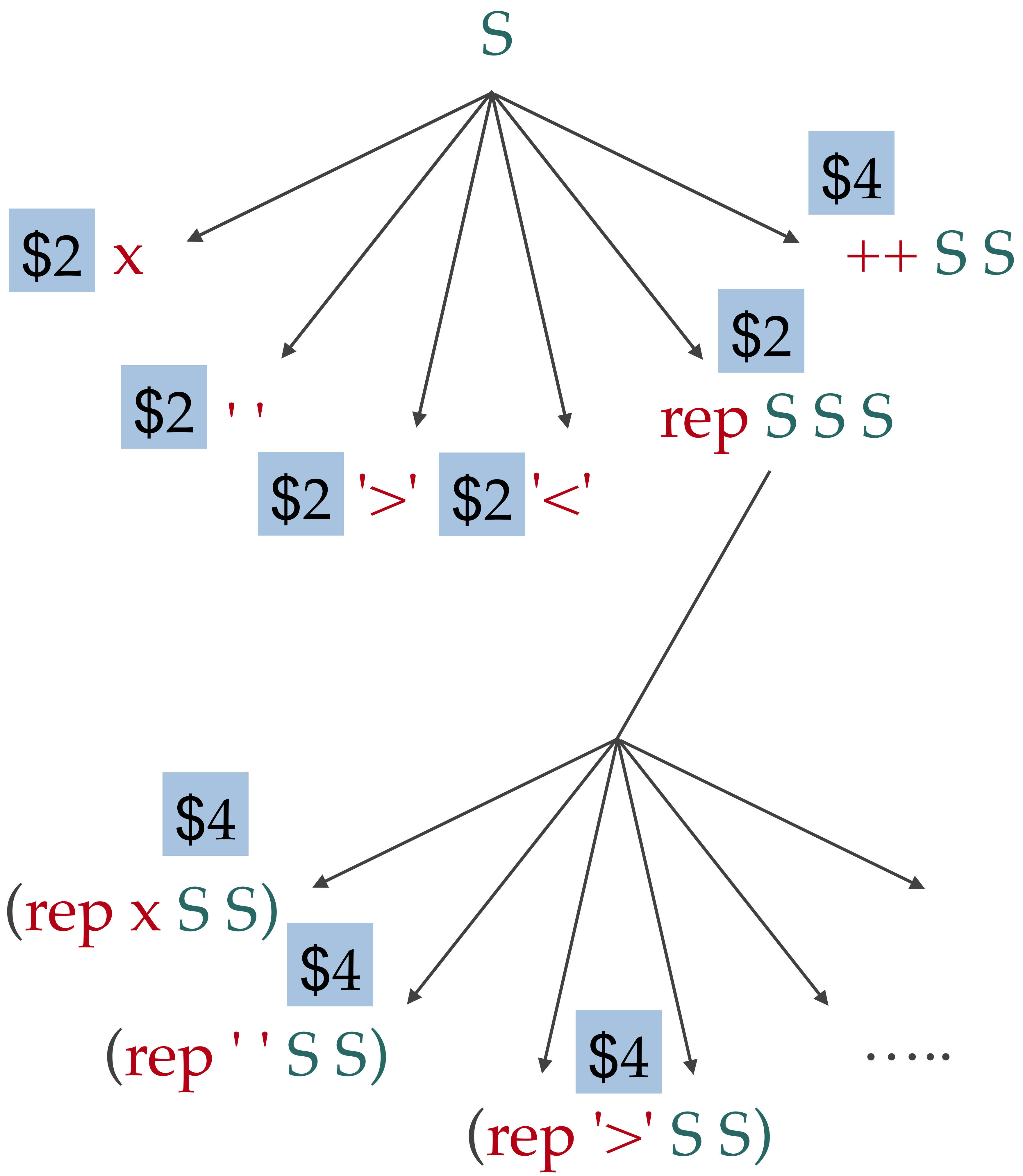
Observational Equivalence Reduction



Guided Search Techniques

Prior Work

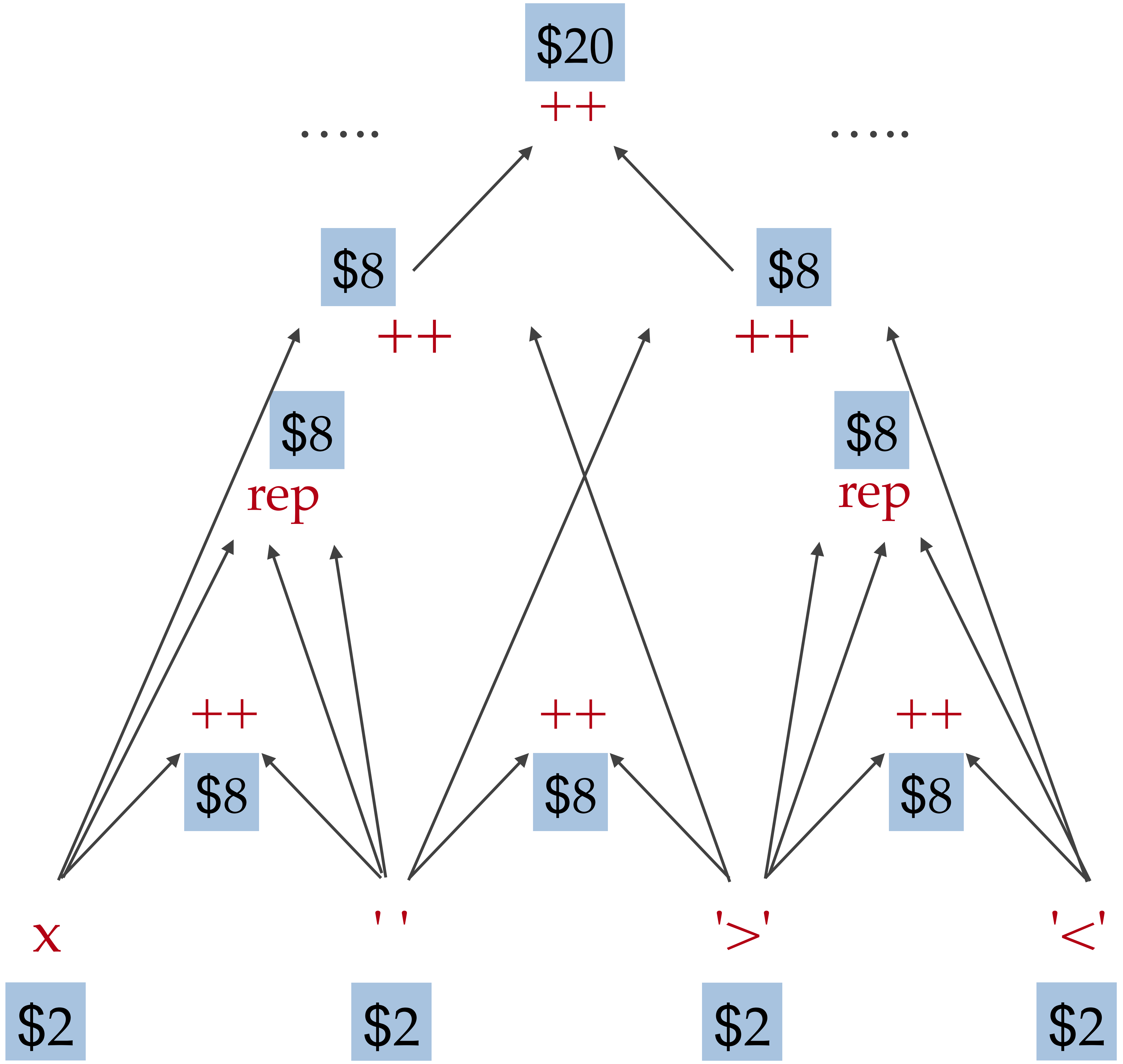
Guided Top-down search



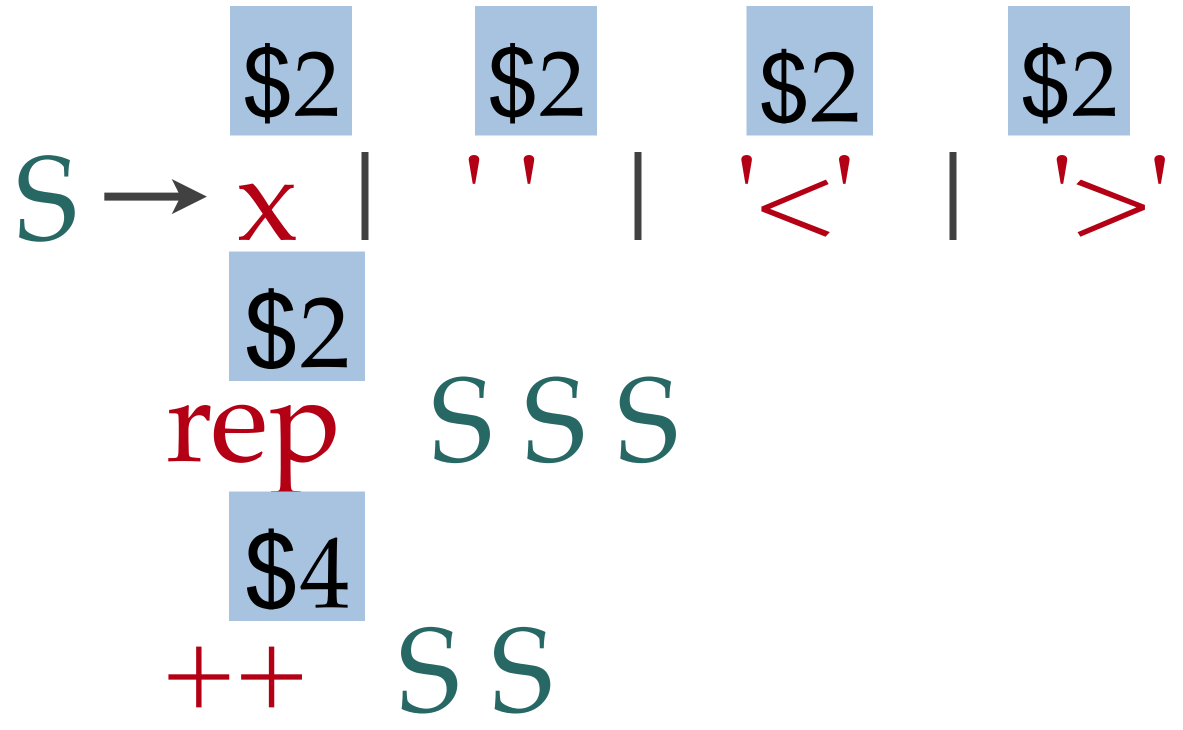
S → x | ' ' | '<' | '>'   ($2 each)
rep S S S   ($2)
++ S S   ($4)

Our Technique

Guided Bottom-up search

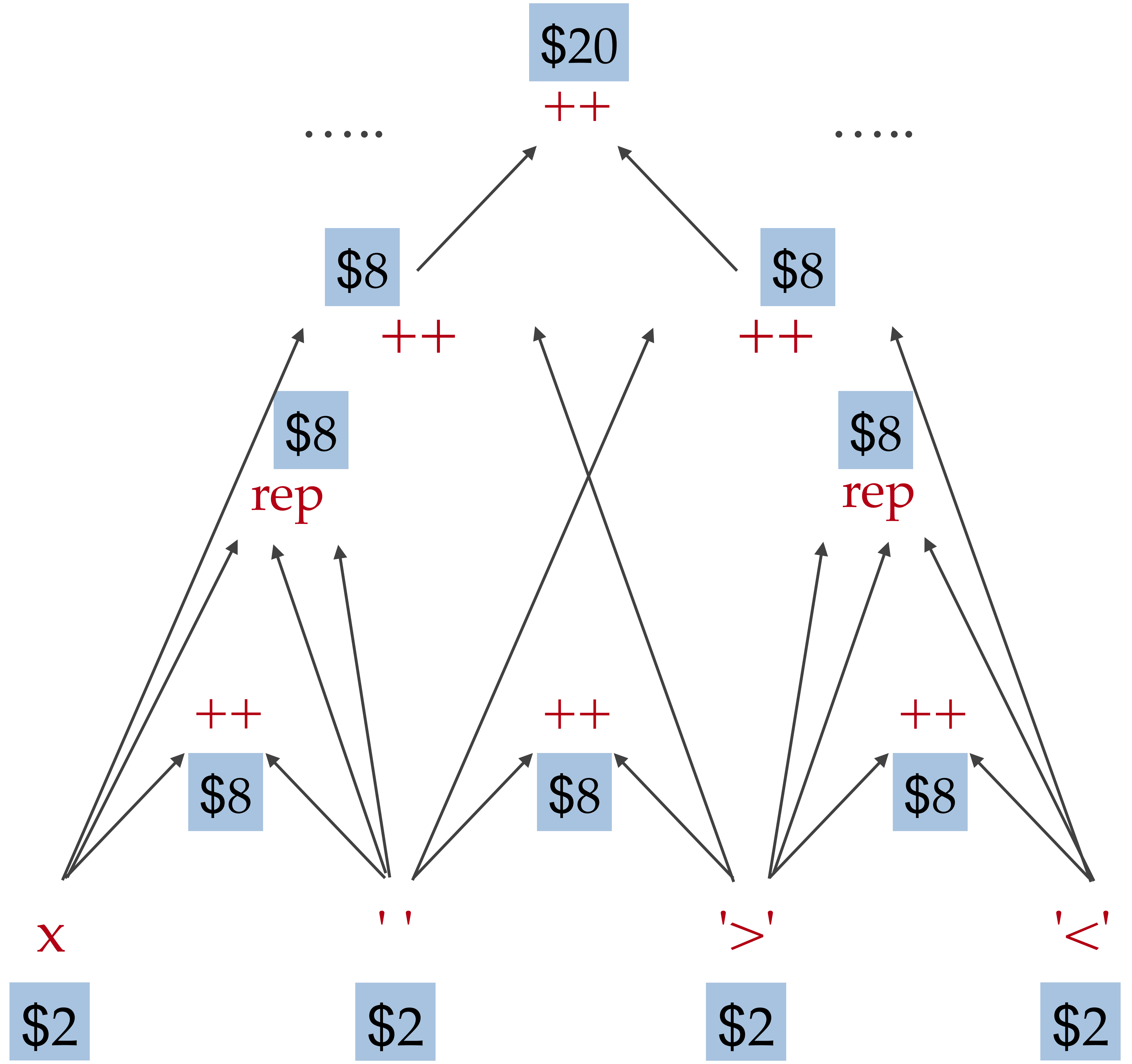


Guided Search Techniques



Our Technique

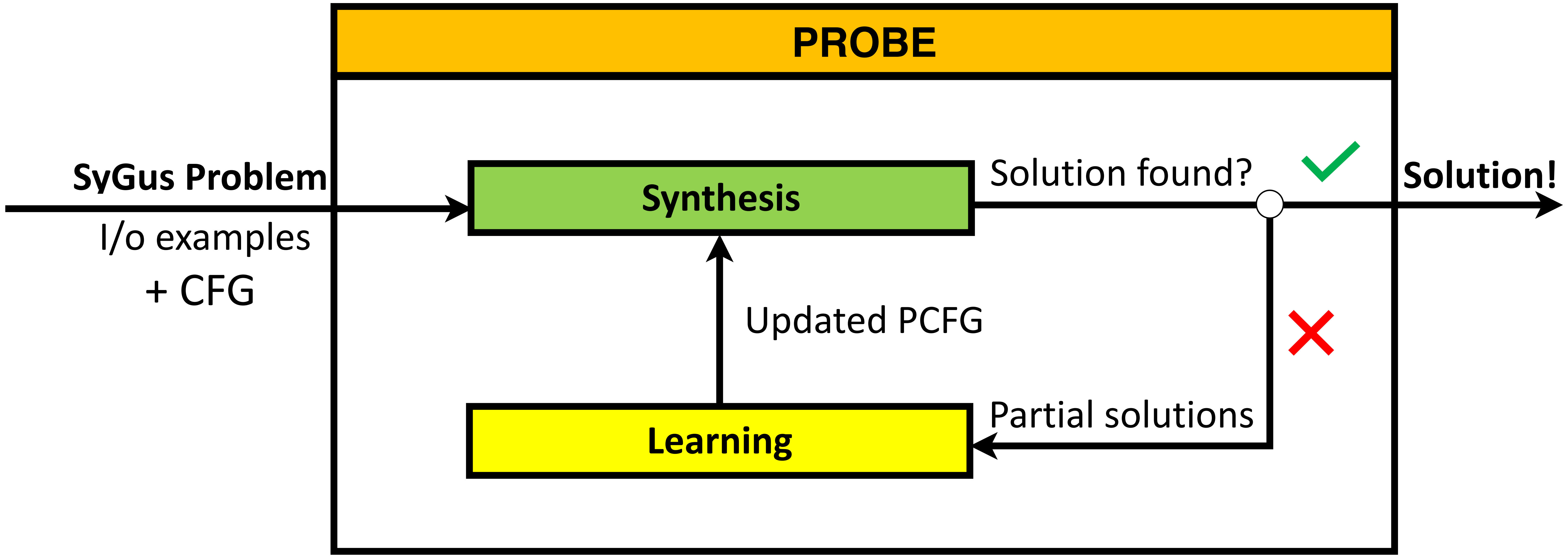
Guided Bottom-up search



Enables Equivalence Reduction

Enables Just-in-Time Learning!

Talk Outline

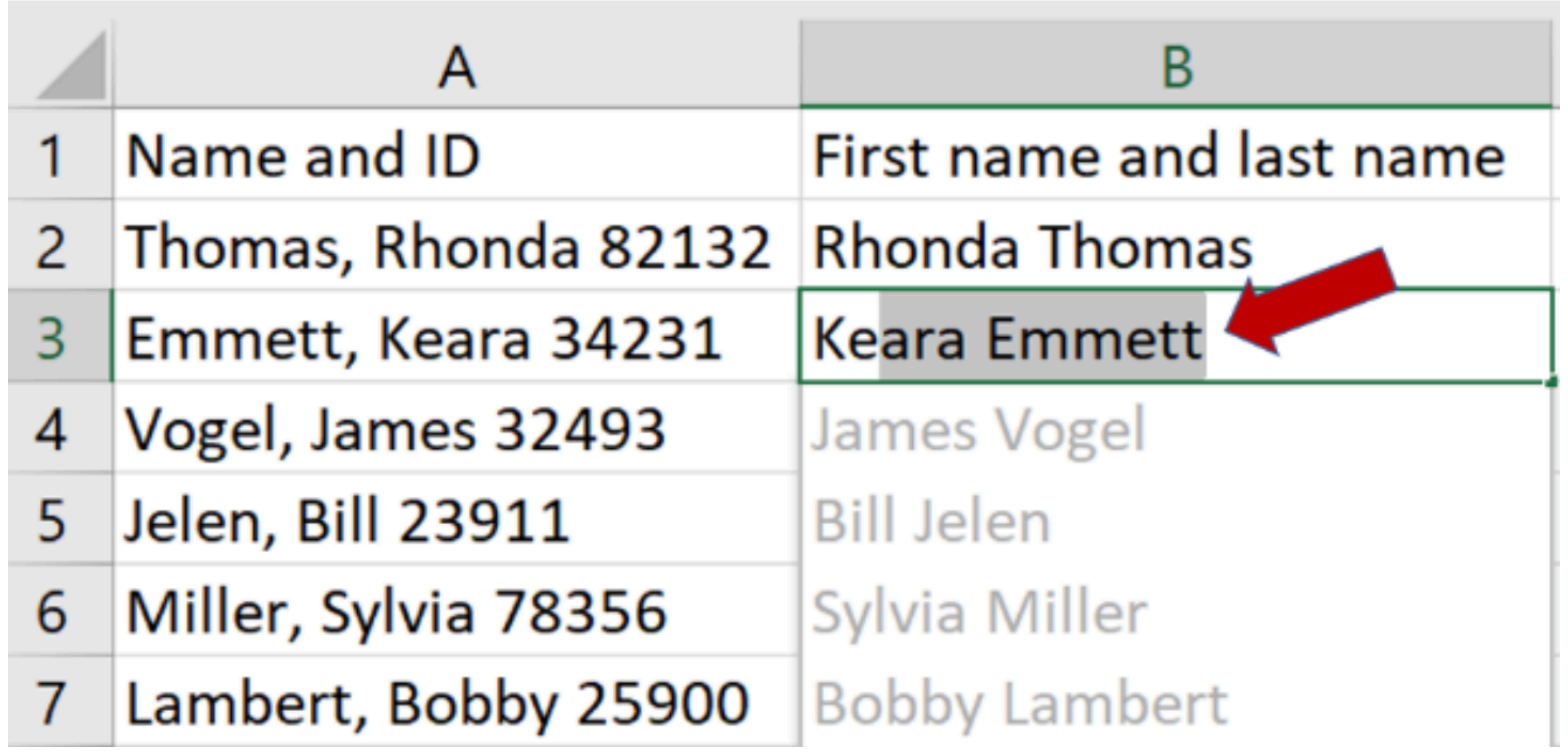


1. Just-in-Time Learning

2. Guided Bottom-Up Search

3. Evaluation Results

Experimental Set-up: Benchmarks



	A	B
1	Name and ID	First name and last name
2	Thomas, Rhonda 82132	Rhonda Thomas
3	Emmett, Keara 34231	Keara Emmett
4	Vogel, James 32493	James Vogel
5	Jelen, Bill 23911	Bill Jelen
6	Miller, Sylvia 78356	Sylvia Miller
7	Lambert, Bobby 25900	Bobby Lambert

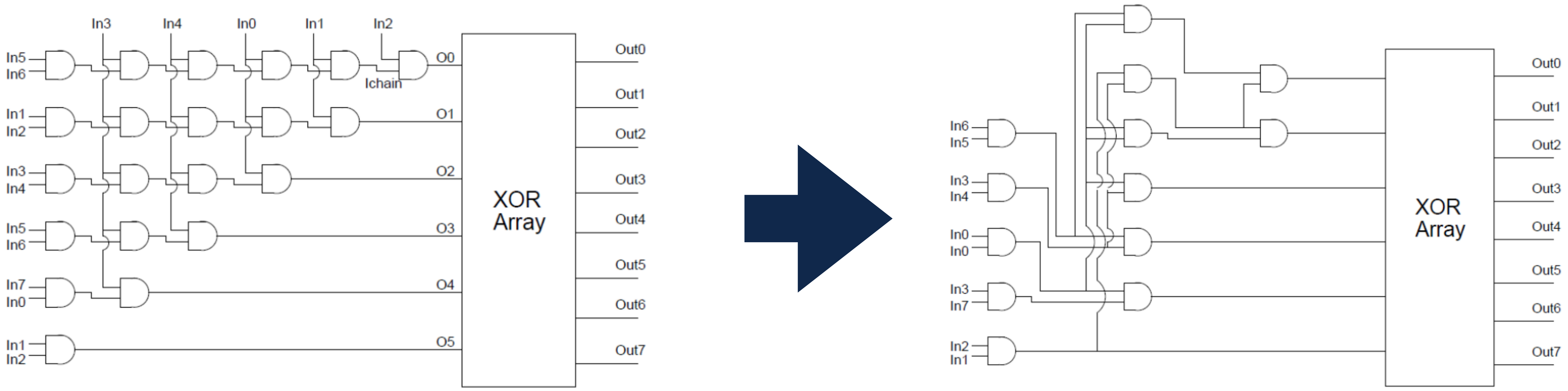
String Manipulation Tasks

Turn off the rightmost sequence of **1**s:

```
00101 → 00100
01010 → 01000
10110 → 10000

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

BitVector Manipulation Tasks



Circuit transformation tasks

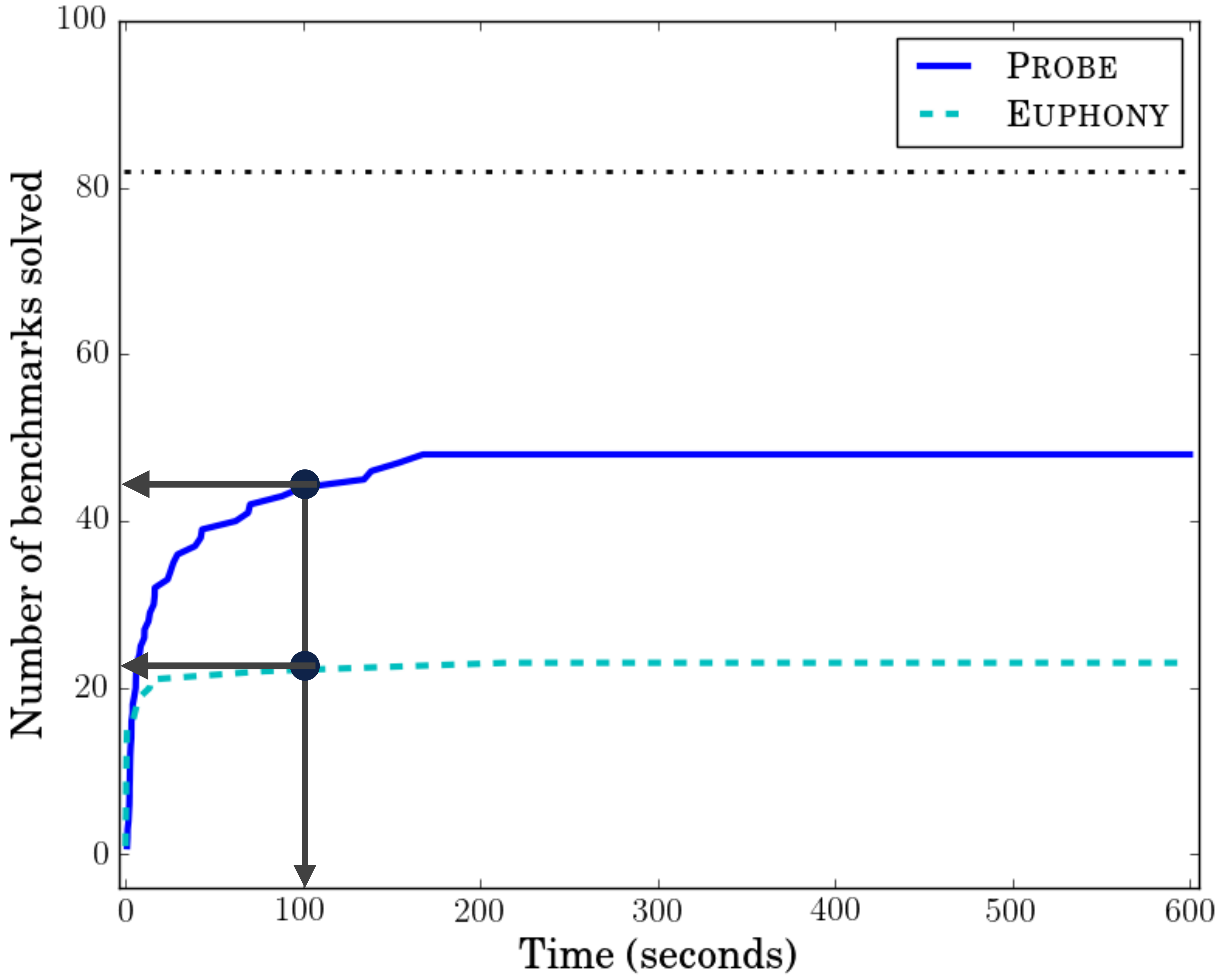
Evaluation Metrics

1. Synthesis Time (Time required to find a solution)
2. Quality of solutions

Experimental Setup: Baseline

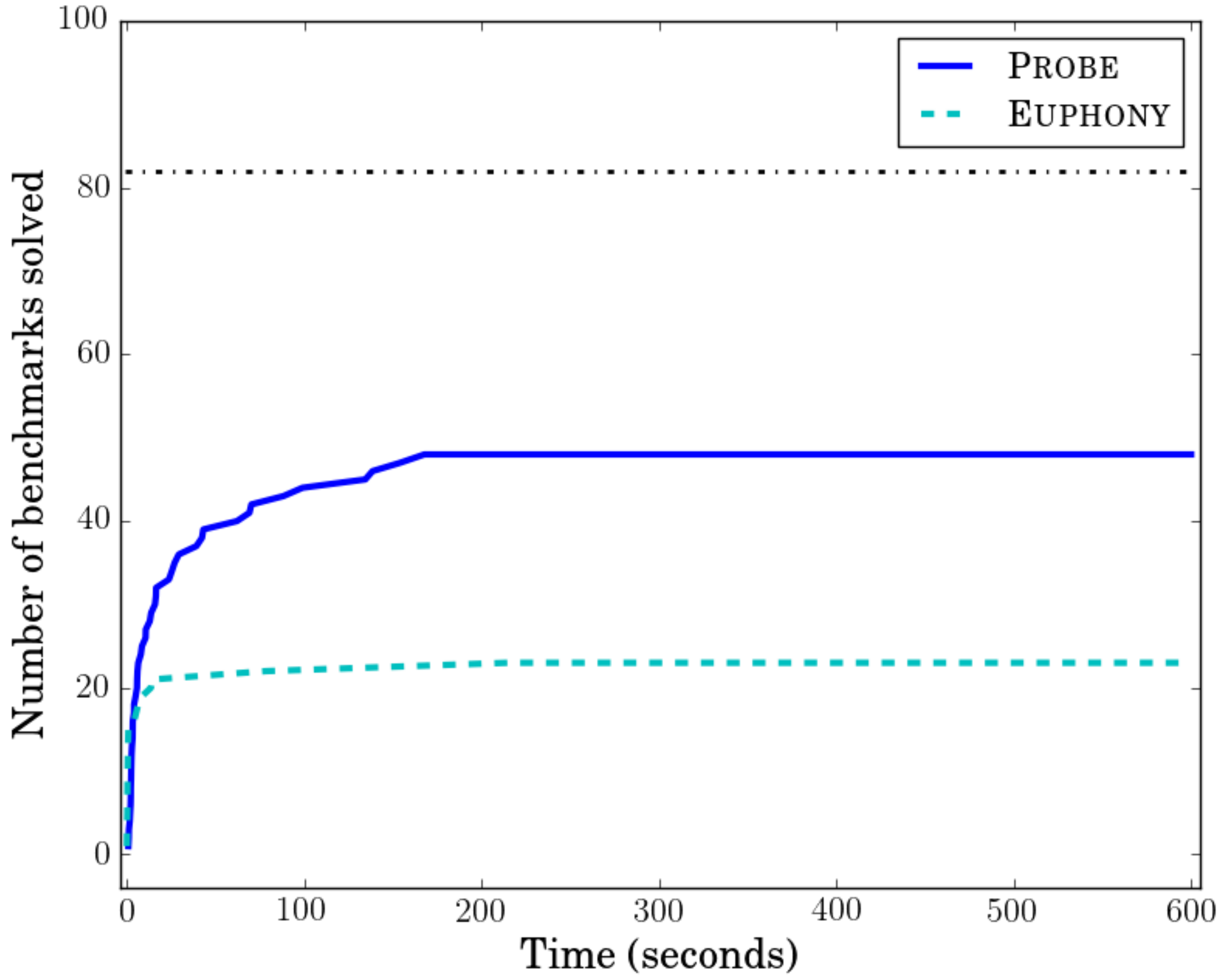
1. **Euphony (top-down enumeration + pre-trained costs)**

Synthesis Time (Probe VS Euphony)

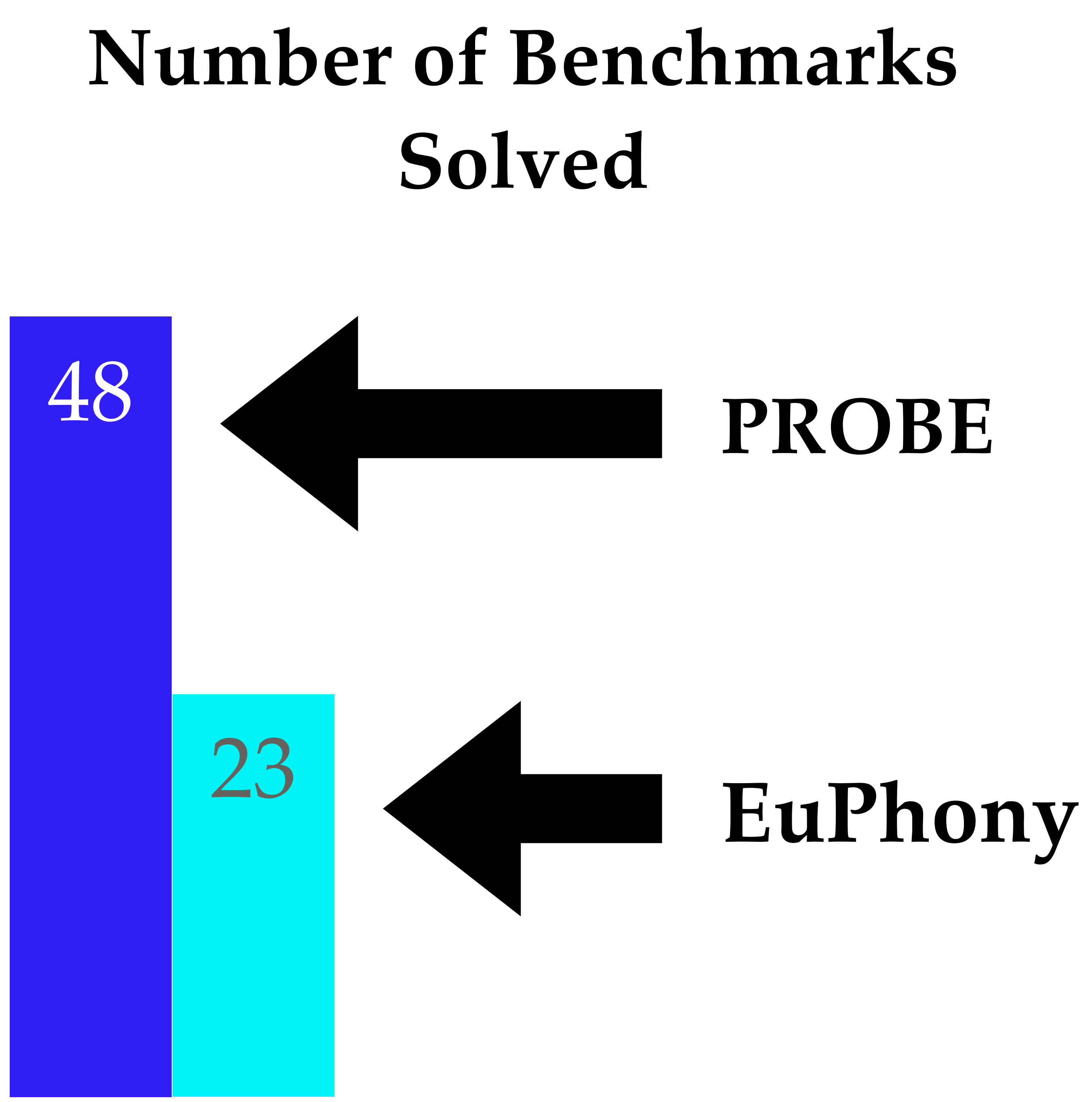


String Domain

Synthesis Time (Probe VS Euphony)

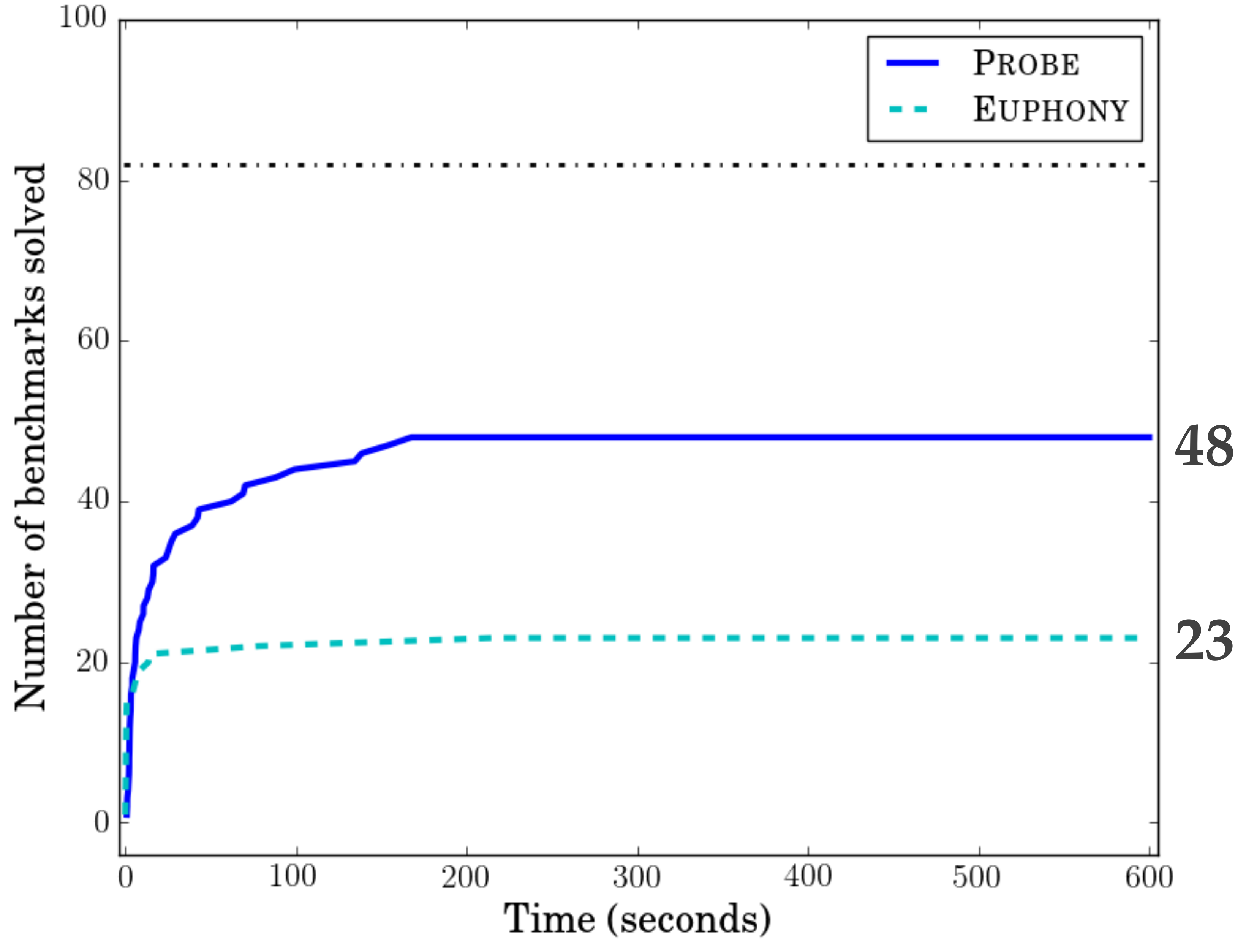


String Domain

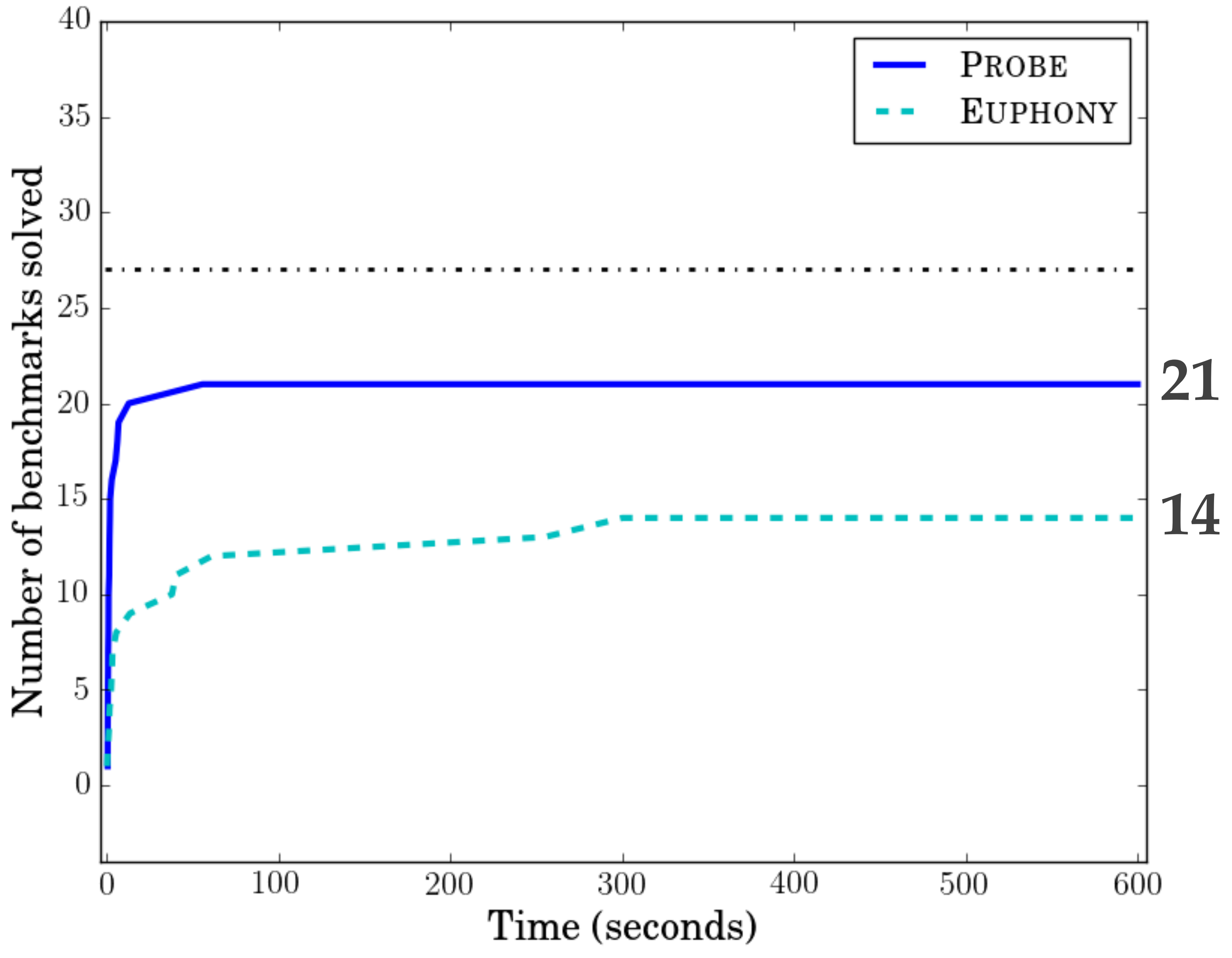


Synthesis Time (Probe VS Euphony)

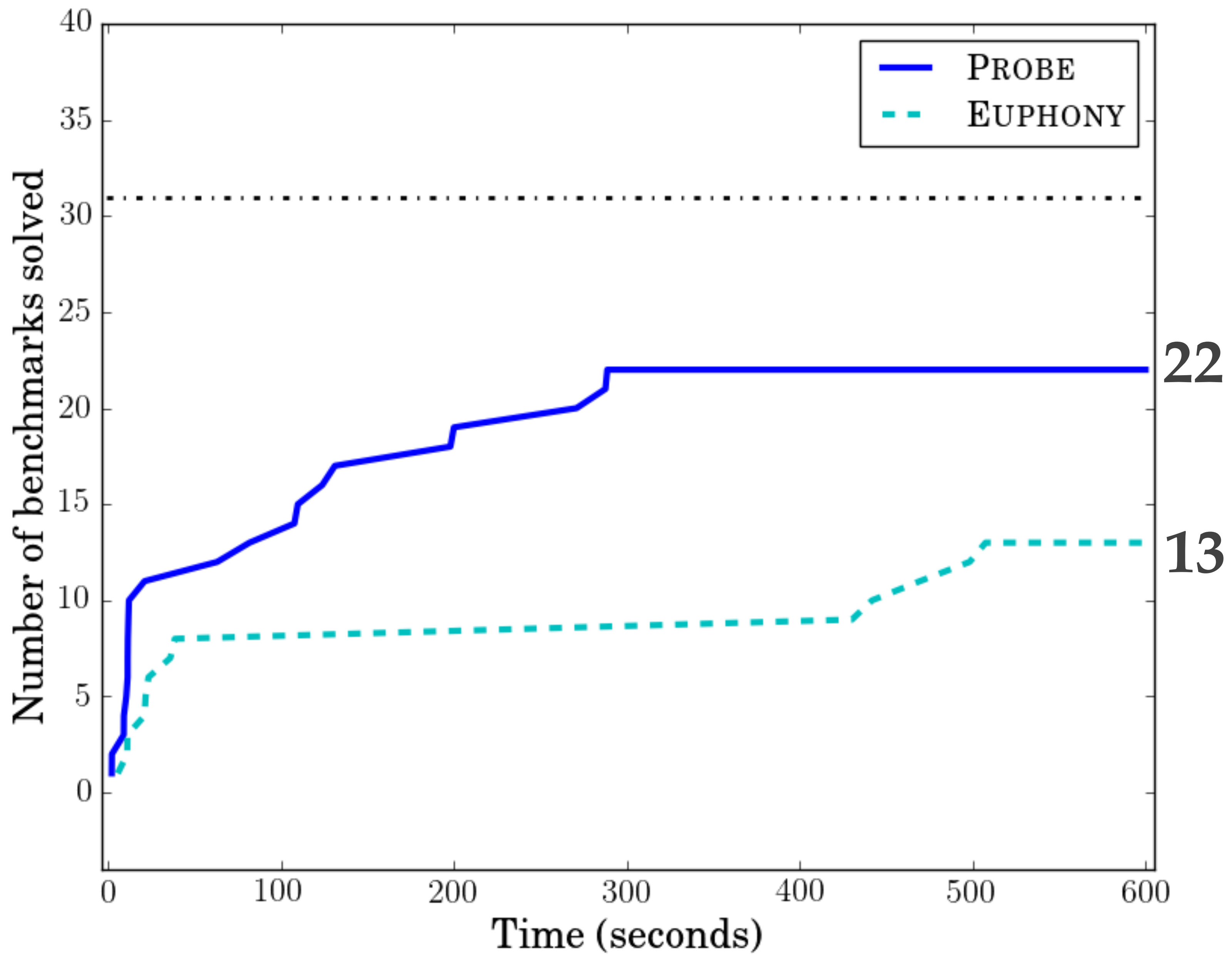
Probe is faster than Euphony on all 3 domains



String Domain



BitVector Domain



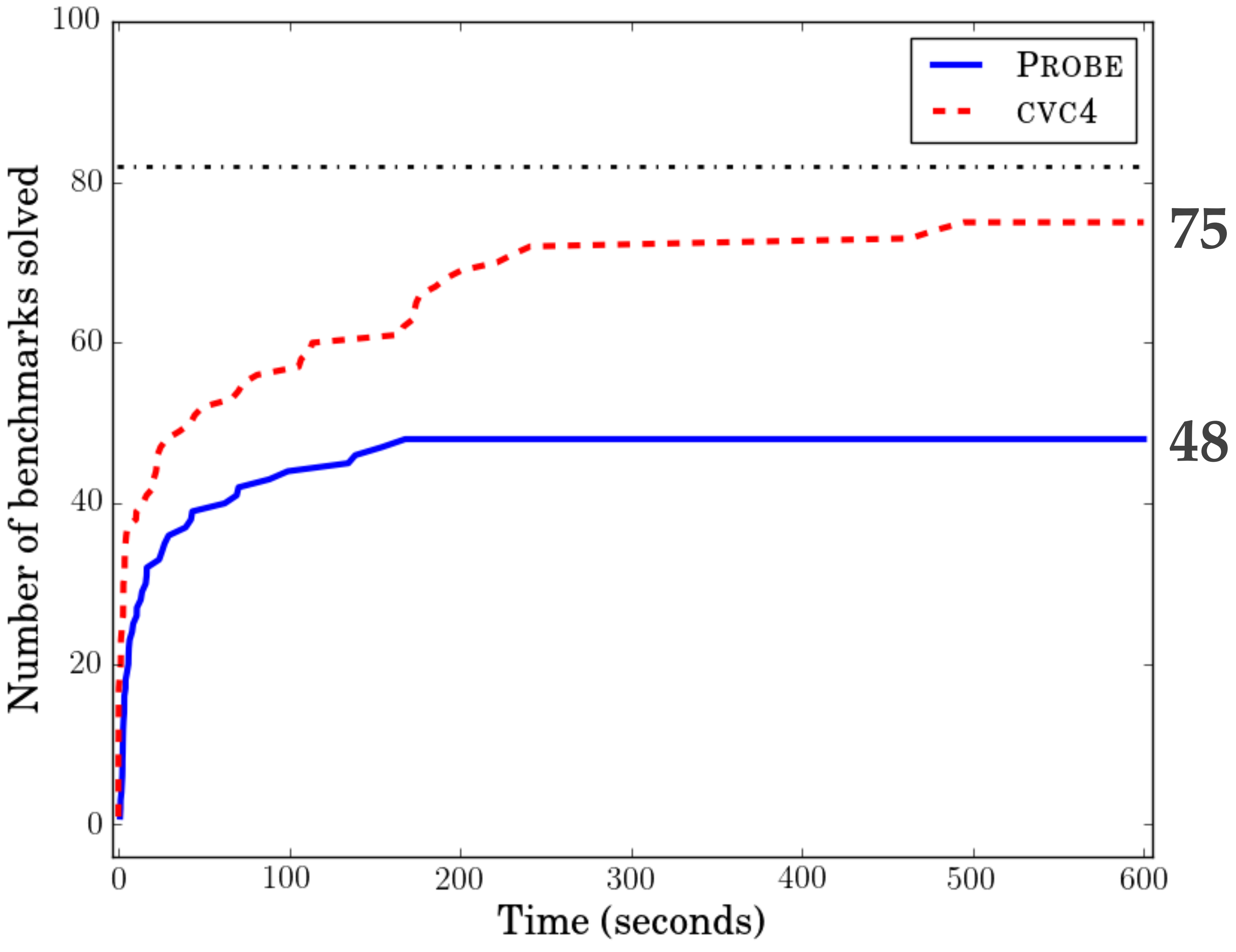
Circuit Domain

Experimental Setup: State-of-the-art Solvers

1. Euphony (top-down enumeration + pre-learned models)
2. CVC4 (Winner of the 2019 SyGuS competition)

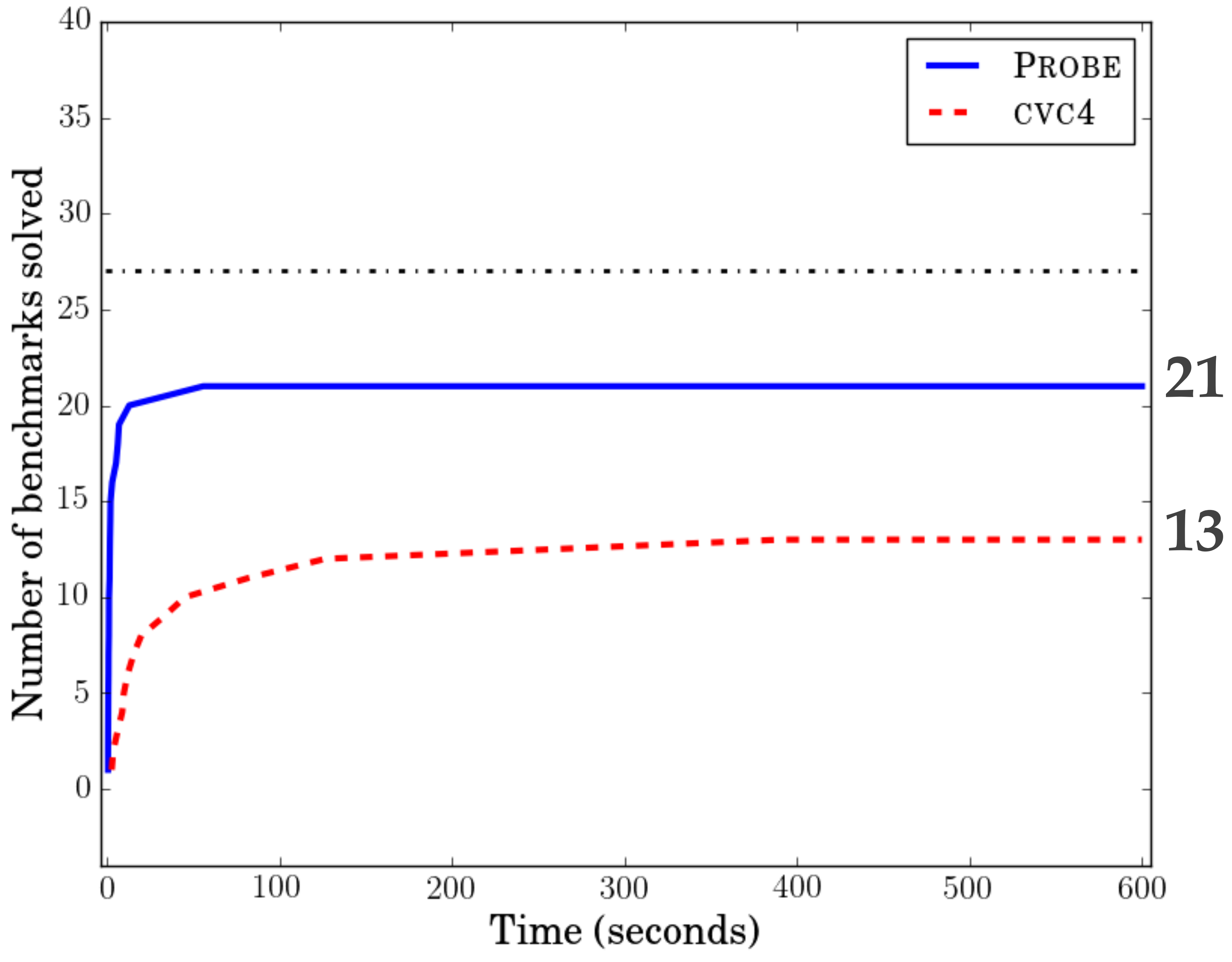
Synthesis Time (Probe VS CVC4)

Input-Output Examples



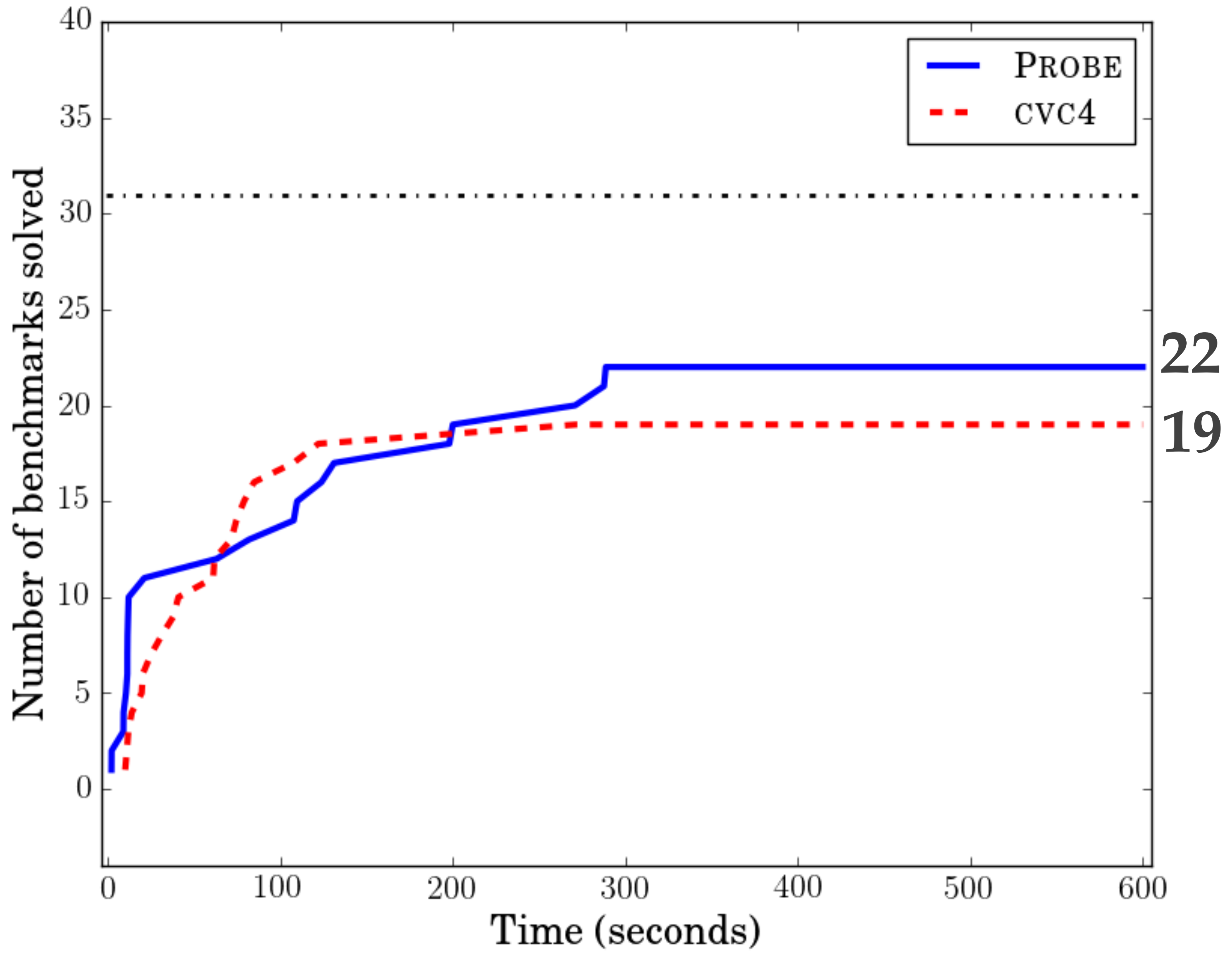
String Domain

First Order Formula



BitVector Domain

First Order Formula

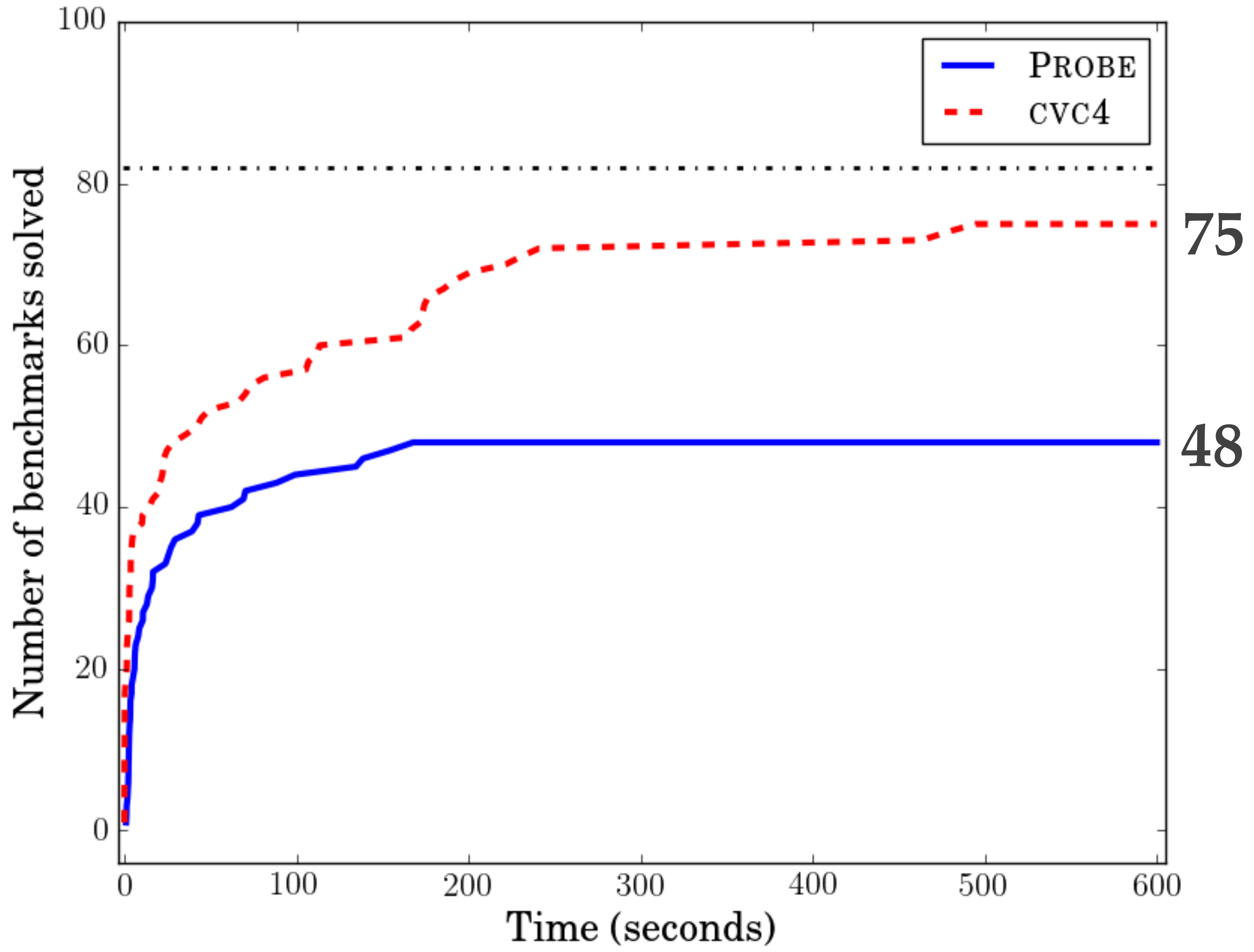


Circuit Domain

Synthesis Time (Probe VS CVC4)

Prone to overfitting

Input-Output Examples



String Domain

Solution Quality: Generalization Accuracy

Benchmark	Training Examples	Testing Examples	Probe Accuracy	CVC4 Accuracy
initials	4	54		
phone-5	7	100		
phone-6	7	100		
phone-7	7	100		
phone-10	7	100		

Solution Quality: Generalization Accuracy

Benchmark	Training Examples	Testing Examples	Probe Accuracy	CVC4 Accuracy
initials	4	54	**100%**	
phone-5	7	100	**100%**	
phone-6	7	100	**100%**	
phone-7	7	100	**100%**	
phone-10	7	100	**100%**	

Solution Quality: Generalization Accuracy

Benchmark	Training Examples	Testing Examples	Probe Accuracy	CVC4 Accuracy
initials	4	54	**100%**	**100%**
phone-5	7	100	**100%**	**100%**
phone-6	7	100	**100%**	**100%**
phone-7	7	100	**100%**	**7%**
phone-10	7	100	**100%**	**57%**

CVC4 does not generalize!

Solution Quality: Generalization Accuracy

Benchmark	Training Examples	Testing Examples	Probe Accuracy	CVC4 Accuracy
phone-9	7	100	-	7%
univ_4	8	20	-	73%
univ_5	8	20	-	68%
univ_6	8	20	-	100%

CVC4 does not generalize!

Solution Quality: Generalization Accuracy

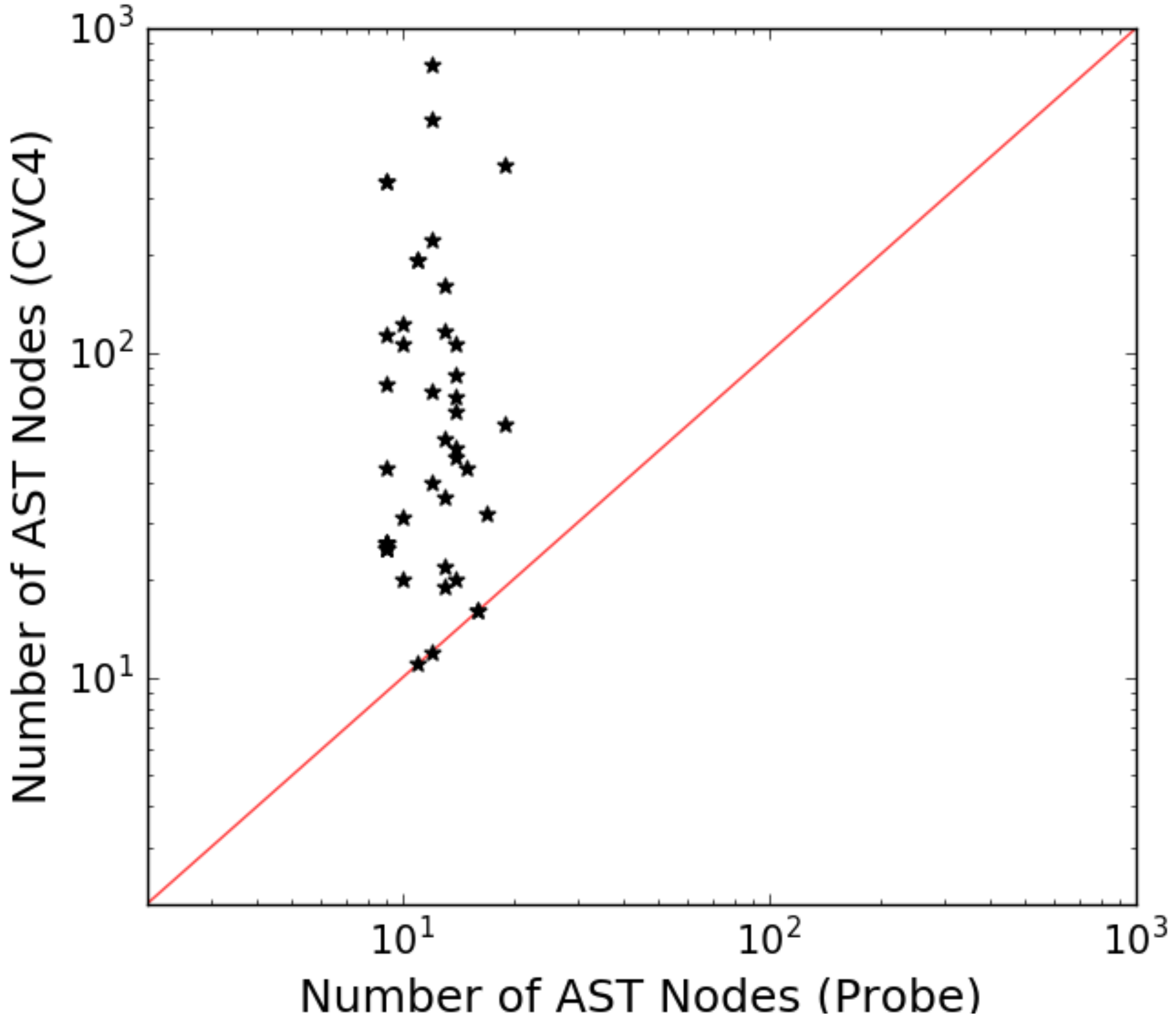
PROBE 100% Average Accuracy

CVC4 68% Average Accuracy

Solution Quality: Size of Solutions

- Size is a surrogate for program simplicity.
- Smaller solutions are more readable and usable.
- Smaller solutions generalize well to additional examples.

Solution Quality: Size of Solutions (CVC4)



Scatter plot of String solution sizes (# of AST nodes)

Solution Quality: Size of Solutions (CVC4)

Probe Solution - 19 AST nodes

(rep (rep (rep (rep (rep (rep arg '<' ' ') '<' ' ') '<' ' ') '>' ' ') '>' ' ') '>' ' ')

CVC4 Solution - 380 AST nodes!

(ite (contains (rep x "<" ") "<") (++ (rep (substr x 0 (indexof x ">" 1)) "<" ") (at x (indexof (rep x " " (rep x " " x)) "<" 1))) (rep " " (at x 1) (rep x "<" " "))) (at x (indexof (rep x "<" x) "<" (indexof x ">" 1)))) (at x (indexof (rep x " " x) ">" 1))) " ") (str.at x (+ -1 (indexof x " " 1)))) (at (rep x " " ") 1)) (at x (indexof (rep x " " (rep x " " x)) ">" 1))) (at x (indexof (rep x "<" x) (++ ">" " ") 1))) (at x (indexof (rep x "<" x) ">" 1))) (at (rep x ">" x) (len x))) (rep " " (at x 1) x)) (at (rep x " " x) (len x))) (at x (+ -1 (indexof (rep x "<" ") "<" 1)))) (rep " " (str.at x 1) ">")) (at x (+ 1 (indexof x "<" (indexof x ">" 1))))) (rep (at x 1) " " (rep x "<" ">"))) (at x (indexof (rep x " " x) " " 1))) (at x (- -1 (- 1 (indexof x ">" 1))))) (at x (indexof (rep x " " (rep x ">" x)) ">" 1))) (at x (+ -1 (indexof (rep x "<" ") "<" 1)))) (at x (indexof (++ (rep x ">" ") ">") ">" 1))) (rep " " (at x 1) (++ x x))) (at x (+ 1 (+ 1 (indexof x "<" 1))))) (at x (indexof (rep x " " (rep x ">" ")) ">" 1))) (rep " " (at x 1) (rep x ">" "))) (at x (+ 1 (indexof (rep x " " "<") " " 1)))) (rep " " (at x 1) (rep x "<" x))) (at x (indexof (rep x " " x) "<" (indexof x ">" 1)))) (at x (indexof (rep x "<" x) (++ ">" " ") 1))) (at (rep x " " x) (+ -1 (len x)))) (at (rep x " " x) (len x))) (at (rep x " " x) (+ 1 (len x)))) (at (rep x " " x) (indexof x ">" 1))) (at x (+ -1 (len x)))) (rep (rep x "<" ") ">" "))

Evaluation Conclusion

1. Probe outperforms Euphony on all 3 domains
2. CVC4 solutions - 2 orders of magnitude larger than Probe's

Conclusion

Just-in-Time Learning + Bottom-up Search - works well!

1. Guided Bottom-up search enumerates programs in the order of cost.
2. On-the-fly guidance is obtained from just-in-time learning.
3. Solutions generated are readable and generalize across 3 domains.

https://github.com/shraddhabarke/probe.git

Artifacts Available acm

Artifacts Evaluated Functional acm

Grammar Statistics

Domain	Operations	Literals	Variables
String Domain	16	11	1
BitVector Domain	17	3	1
Circuit Domain	4	0	6

String Domain Grammar

```
Start → S
    S → arg0 | arg1| ...           string variables
      | lit-1 | lit-2 | ...        string literals
      | (replace S S S)            replace s x y replaces first occurrence of x in s with y
      | (concat S S)               concat x y concatenates x and y
      | (substr S I I)             substr x y z extracts substring of length z, from index y
      | (ite B S S)                ite x y z returns y if x is true, otherwise z
      | (int.to.str I)             int.to.str x converts int x to a string
      | (at S I)                   at x y returns the character at index y in string x
    B → true | false               bool literals
      | (= I I)                    = x y returns true if x equals y
      | (contains S S)             contains x y returns true if x contains y
      | (suffixof S S)             suffixof x y returns true if x is the suffix of y
      | (prefixof S S)             prefixof x y returns true if x is the prefix of y
    I → arg0 | arg1| ...           int variables
      | lit-1 | lit-2 | ...        int literals
      | (str.to.int S)             str.to.int x converts string x to a int
      | (+ I I)                    + x y sums x and y
      | (- I I)                    - x y subtracts y from x
      | (length S)                 length x returns length of x
      | (ite B I I)                ite x y z returns y if x is true, otherwise z
      | (indexof S S I)            indexof x y z returns index of y in x, starting at index z
```

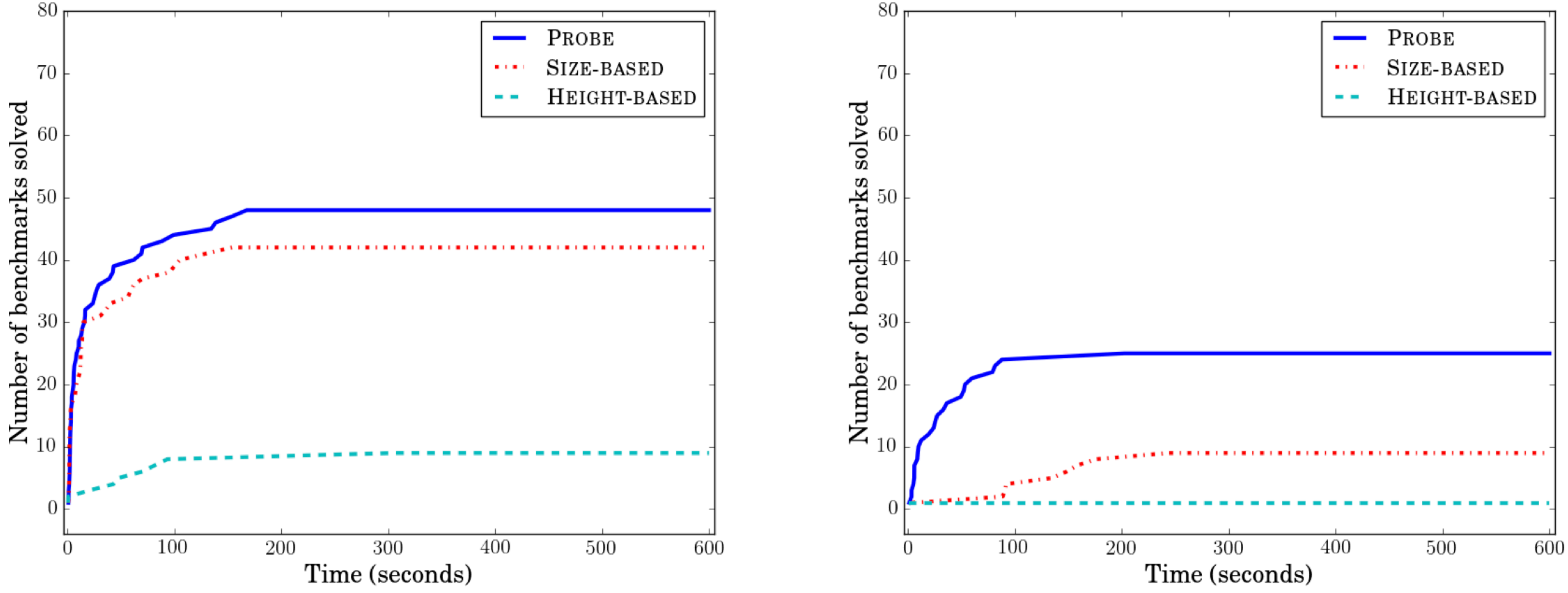
BitVector Domain Grammar

Start →	*BV*	
BV →	`arg0` | `arg1` | `...`	bit-vector variables
|	`lit-1` | `lit-2` | `...`	bit-vector literals
|	(`xor` *BV BV*)	`xor x y` performs bitwise xor between `x` and `y`
|	(`and` *BV BV*)	`and x y` performs bitwise and operation between `x` and `y`
|	(`or` *BV BV*)	`or x y` performs bitwise or operation between `x` and `y`
|	(`neg` *BV*)	`neg x` returns the two's complement of `x`
|	(`not` *BV*)	`not x` returns the one's complement of `x`
|	(`add` *BV BV*)	`add x y` adds `x` and `y`
|	(`mul` *BV BV*)	`mul x y` multiplies `x` and `y`
|	(`udiv` *BV BV*)	`udiv x y` returns the unsigned quotient of dividing `x` by `y`
|	(`urem` *BV BV*)	`urem x y` returns the unsigned remainder of dividing `x` by `y`
|	(`lshr` *BV BV*)	`lshr x y` returns the logical right shift of `x` by `y` bits
|	(`ashr` *BV BV*)	`ashr x y` returns the arithmetic right shift of `x` by `y`
|	(`shl` *BV BV*)	`shl x y` returns the logical left shift of `x` by `y`
|	(`sdiv` *BV BV*)	`sdiv x y` returns the signed quotient of dividing `x` by `y`
|	(`srem` *BV BV*)	`srem x y` returns the signed remainder of dividing `x` by `y`
|	(`sub` *BV BV*)	`sub x y` subtracts `y` from `x`
|	(`ite` *B BV BV*)	`ite x y z` returns `y` if `x` is true, otherwise `z`
B →	`true` | `false`	bool literals
|	(`=` *BV BV*)	`= x y` returns true if `x` equals `y`
|	(`ult` *BV BV*)	`ult x y` returns true if `x` is unsigned less than `y`
|	(`ule` *BV BV*)	`ule x y` returns true if `x` is unsigned less than equal to `y`
|	(`slt` *BV BV*)	`slt x y` returns true if `x` is signed less than `y`
|	(`sle` *BV BV*)	`sle x y` returns true if `x` is signed less than equal to `y`
|	(`ugt` *BV BV*)	`ugt x y` returns true if `x` unsigned greater than `y`
|	(`redor` *BV*)	`redor x` performs bit-wise or reduction of `x`
|	(`and` *BV BV*)	`and x y` returns the logical and of `x` and `y`
|	(`or` *BV BV*)	`or x y` returns the logical or of `x` and `y`
|	(`not` *BV*)	`not x` returns the logical not of `x`

Circuit Domain Grammar

$Start \rightarrow$	B	
$B \rightarrow$	`arg0` | `arg1` | ...	boolean variables
|	(`and` B B)	`and x y` returns the logical and of `x` and `y`
|	(`not` B)	`not x` returns the logical not of `x`
|	(`or` B B)	`or x y` returns the logical or of `x` and `y`
|	(`xor` B B)	`xor x y` returns the logical xor of `x` and `y`

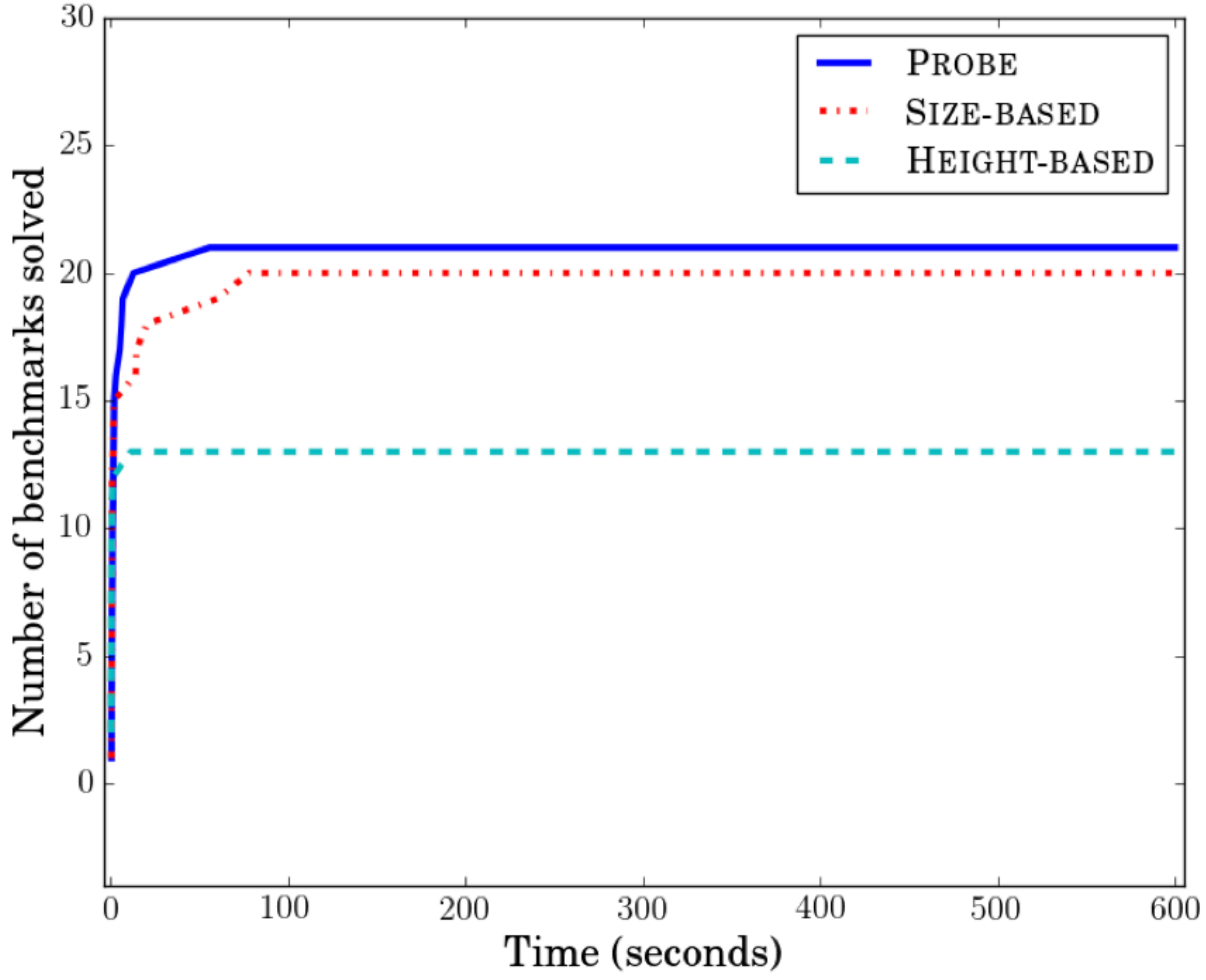
Synthesis Time (Probe VS Traditional Synthesis)



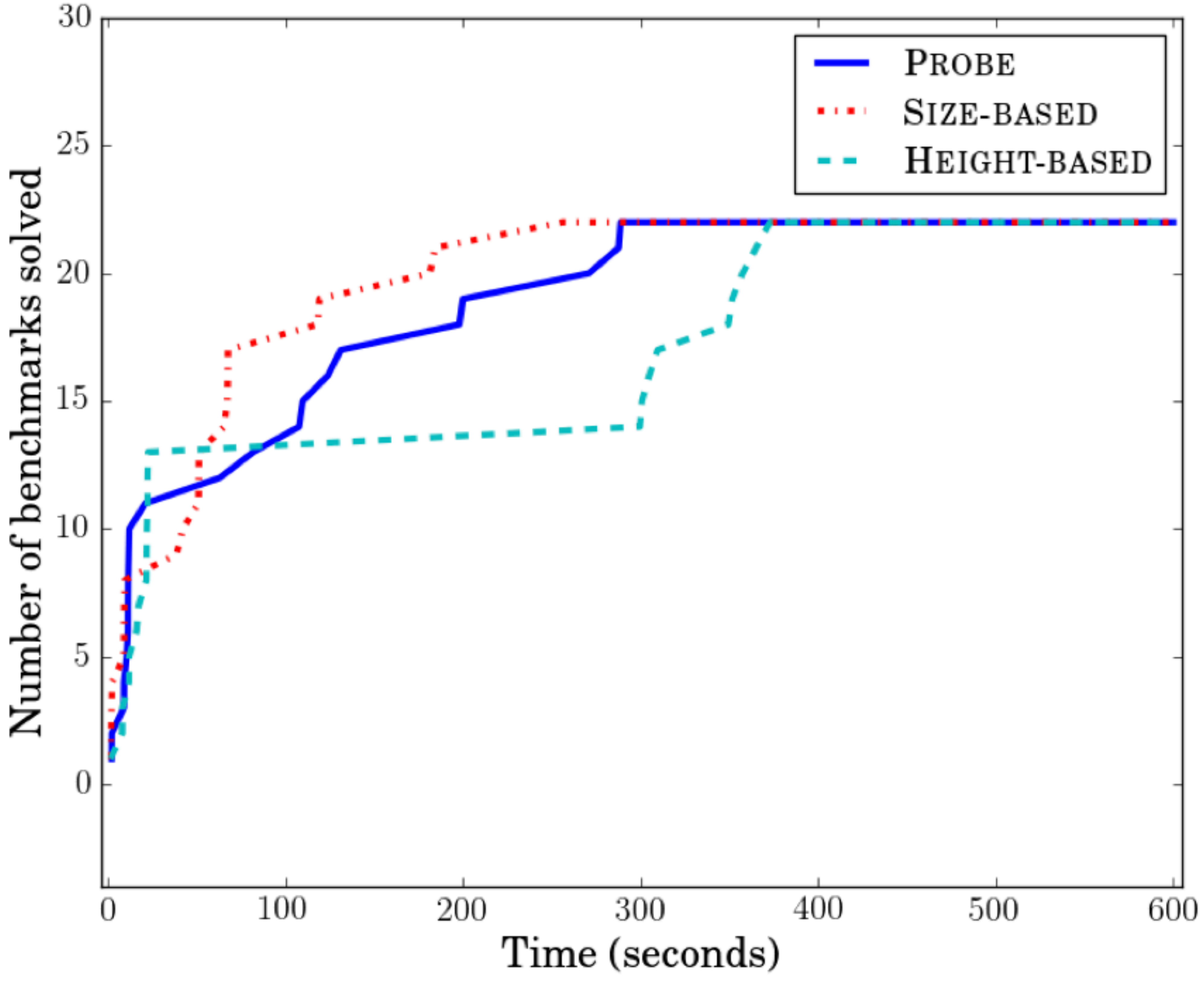
(a) STRING domain with regular grammar.

(b) STRING domain with extended grammar.

Synthesis Time (Probe VS Traditional Synthesis)

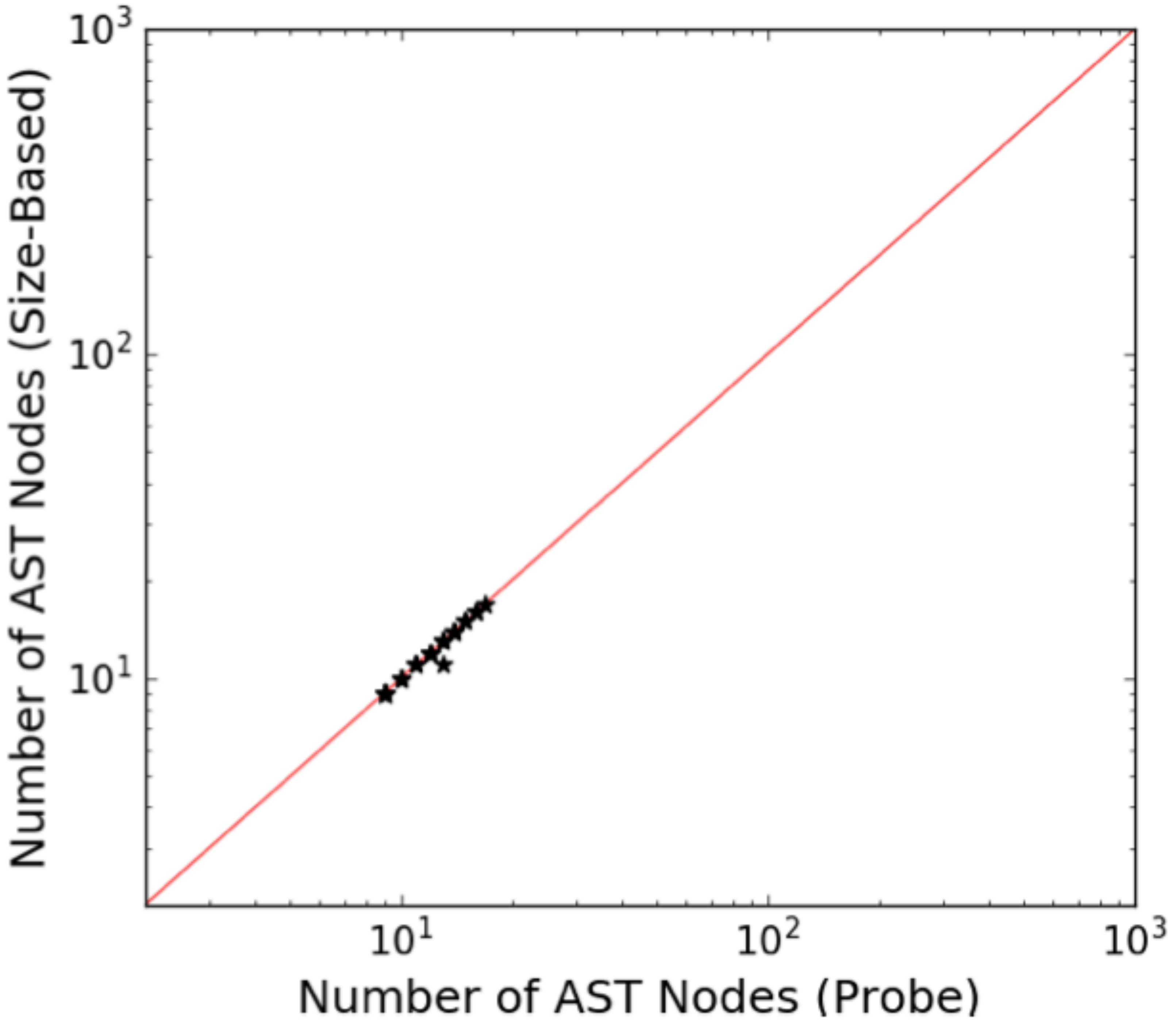


(c) BitVec domain

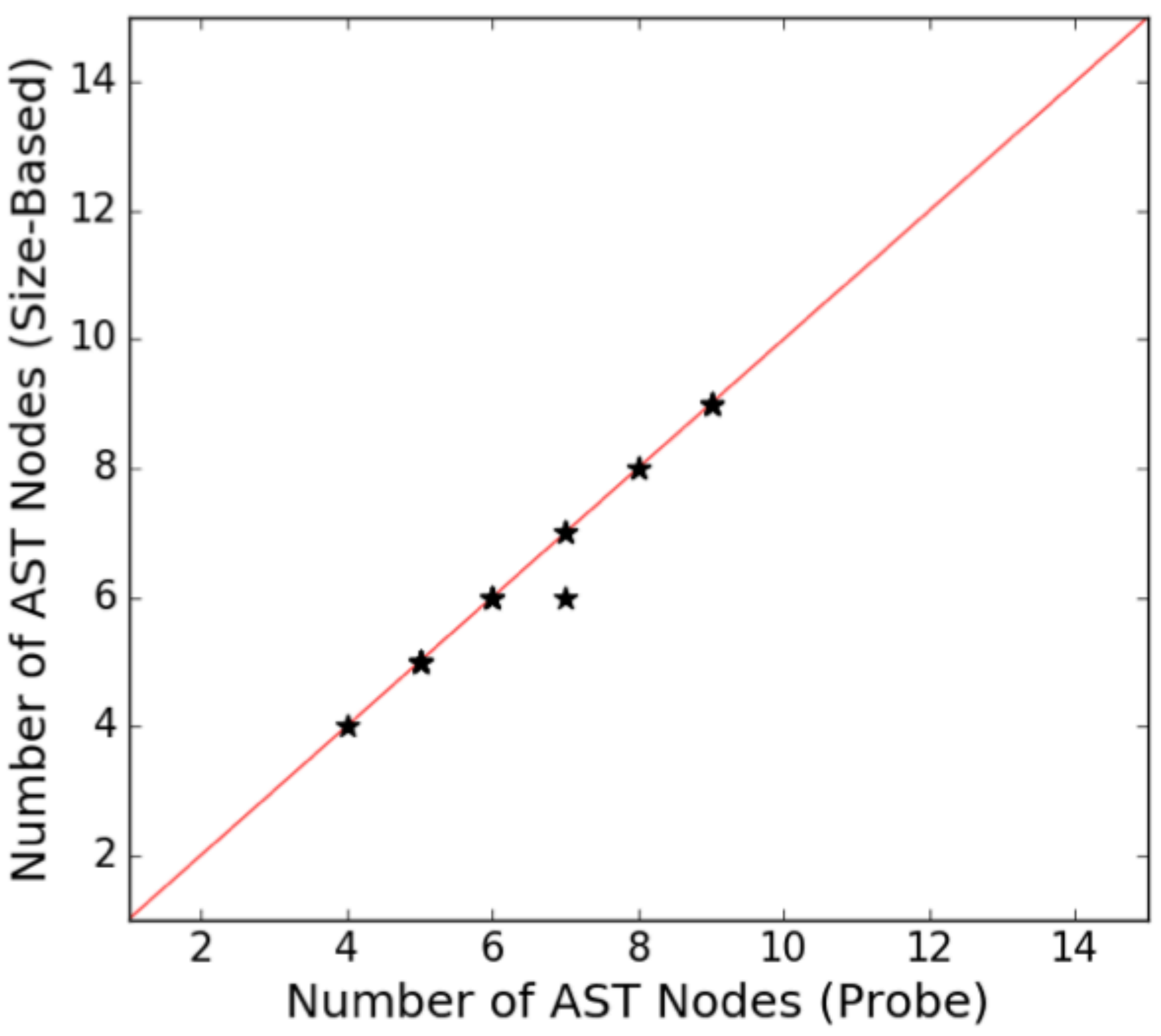


(d) Circuit domain

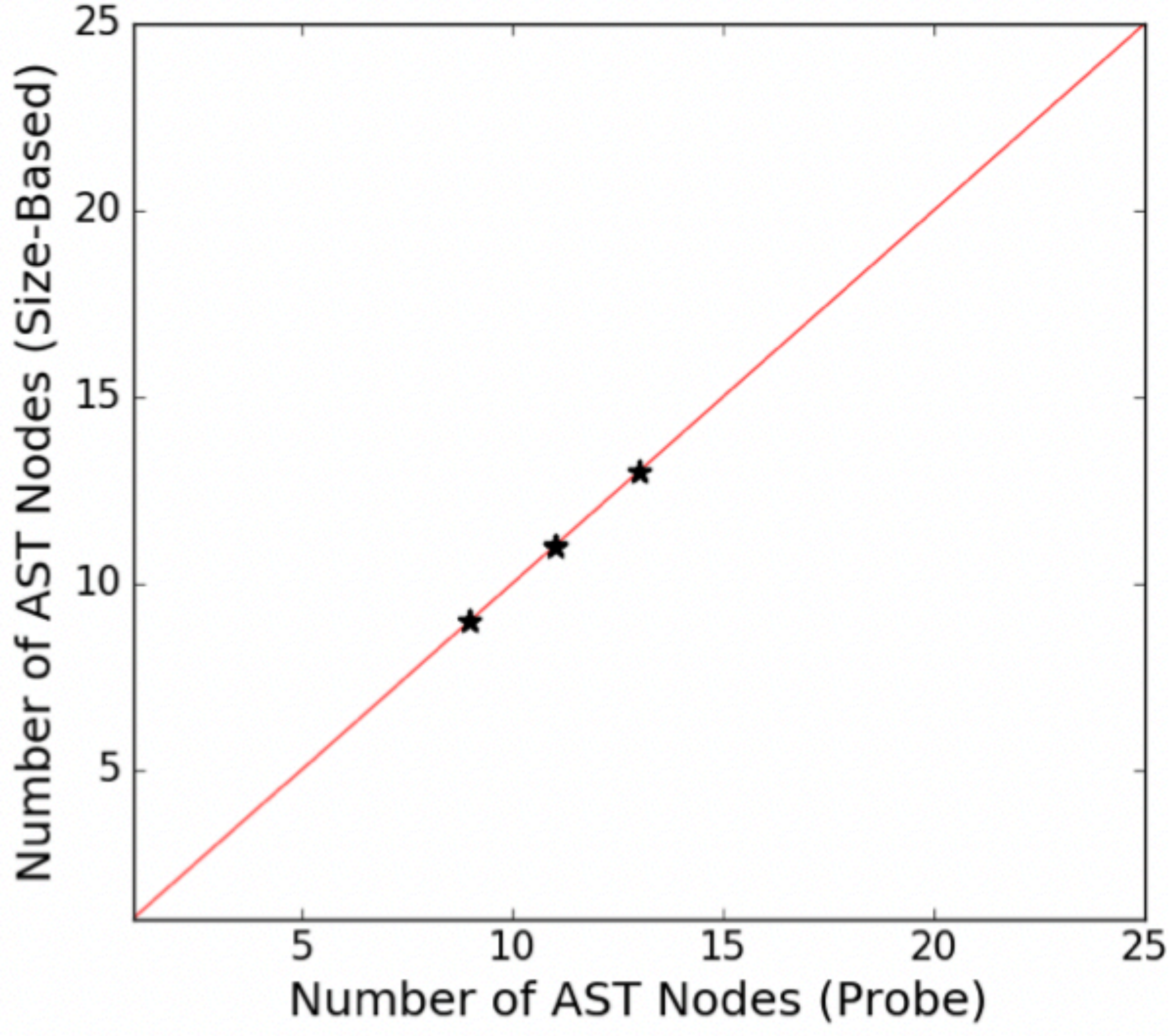
Program Size (Probe VS Traditional Synthesis)



String Domain



BitVector Domain

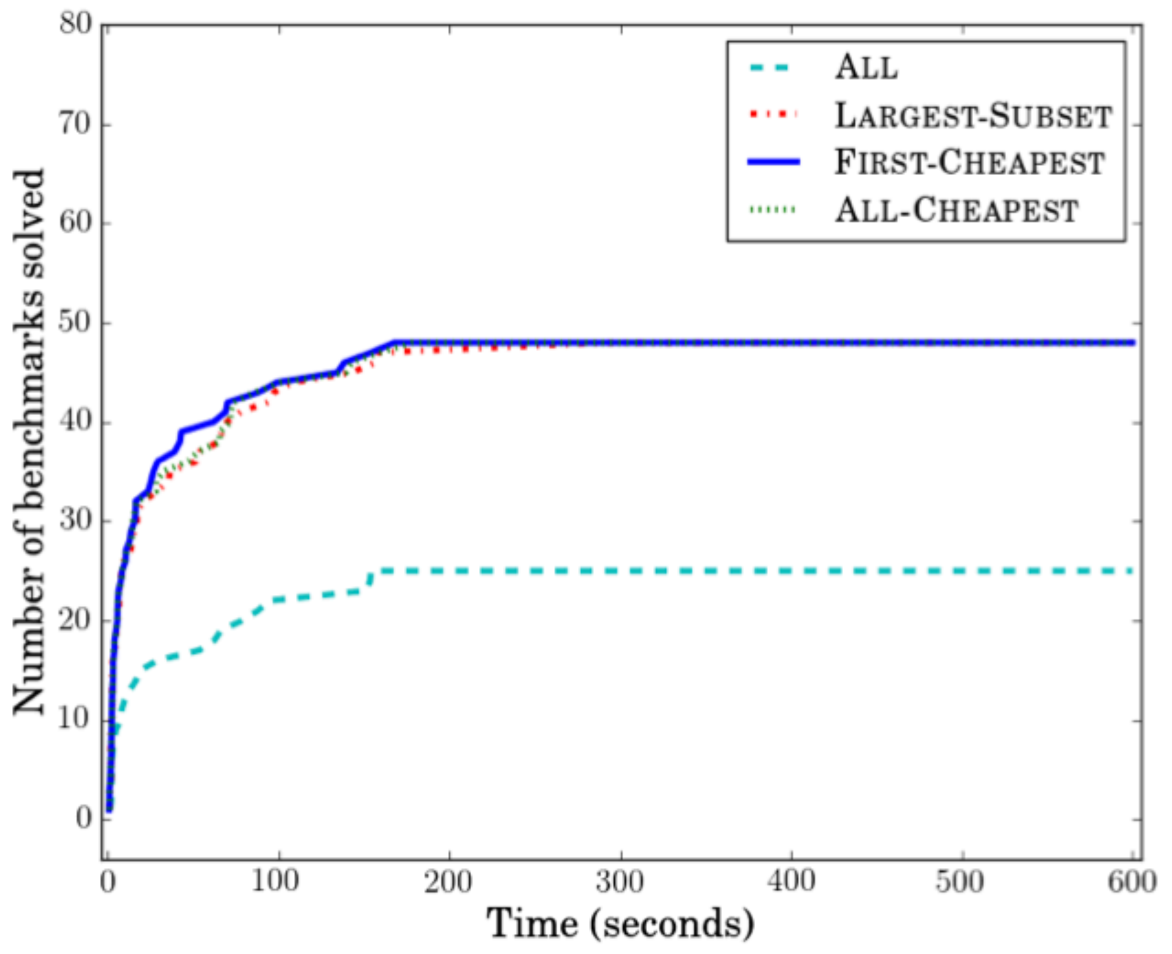


Circuit Domain

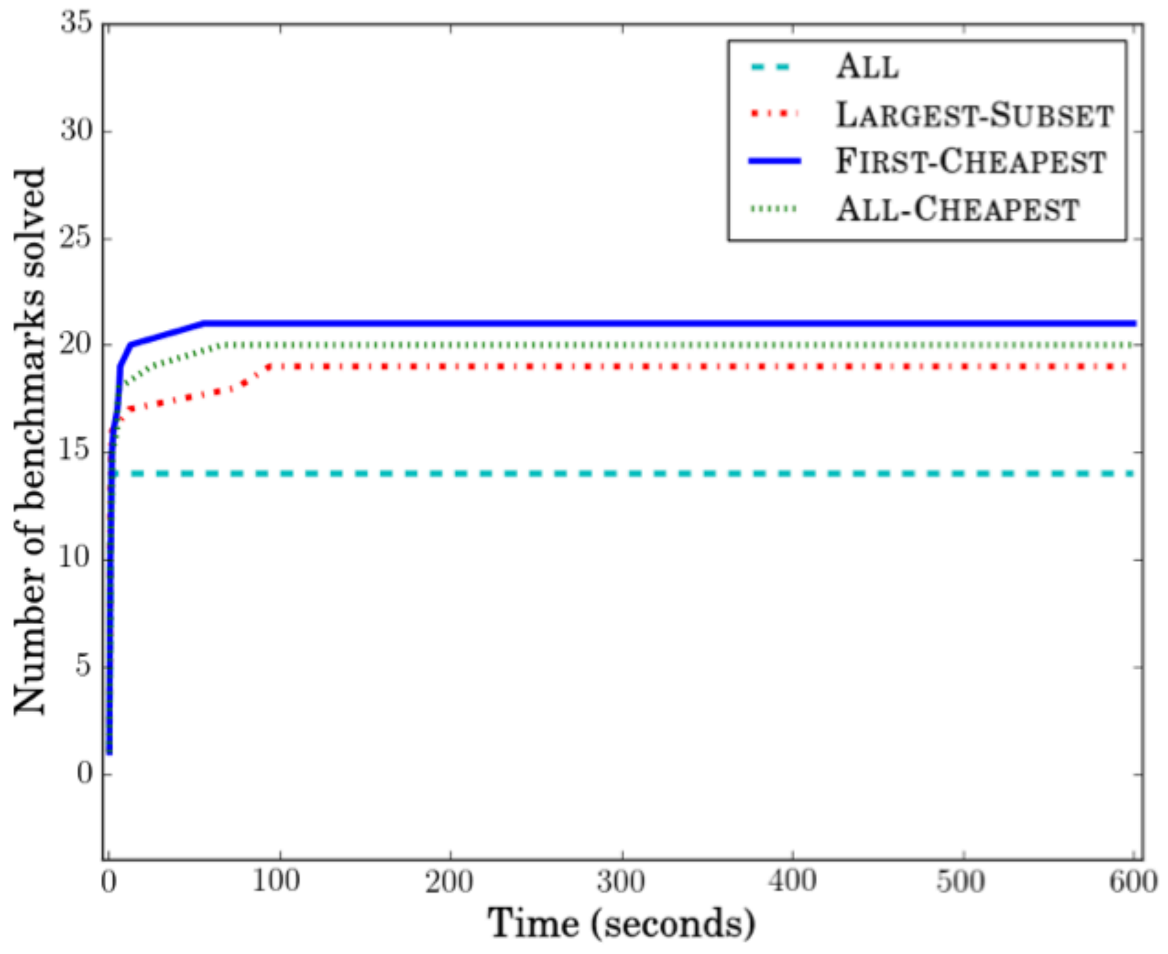
Partial Solution Selection Strategies

- Largest Subset - Single cheapest program that satisfies the largest subset of examples
- First Cheapest - Single cheapest program that satisfies a unique subset of examples
- All Cheapest - All cheapest programs that satisfy a unique subset of examples

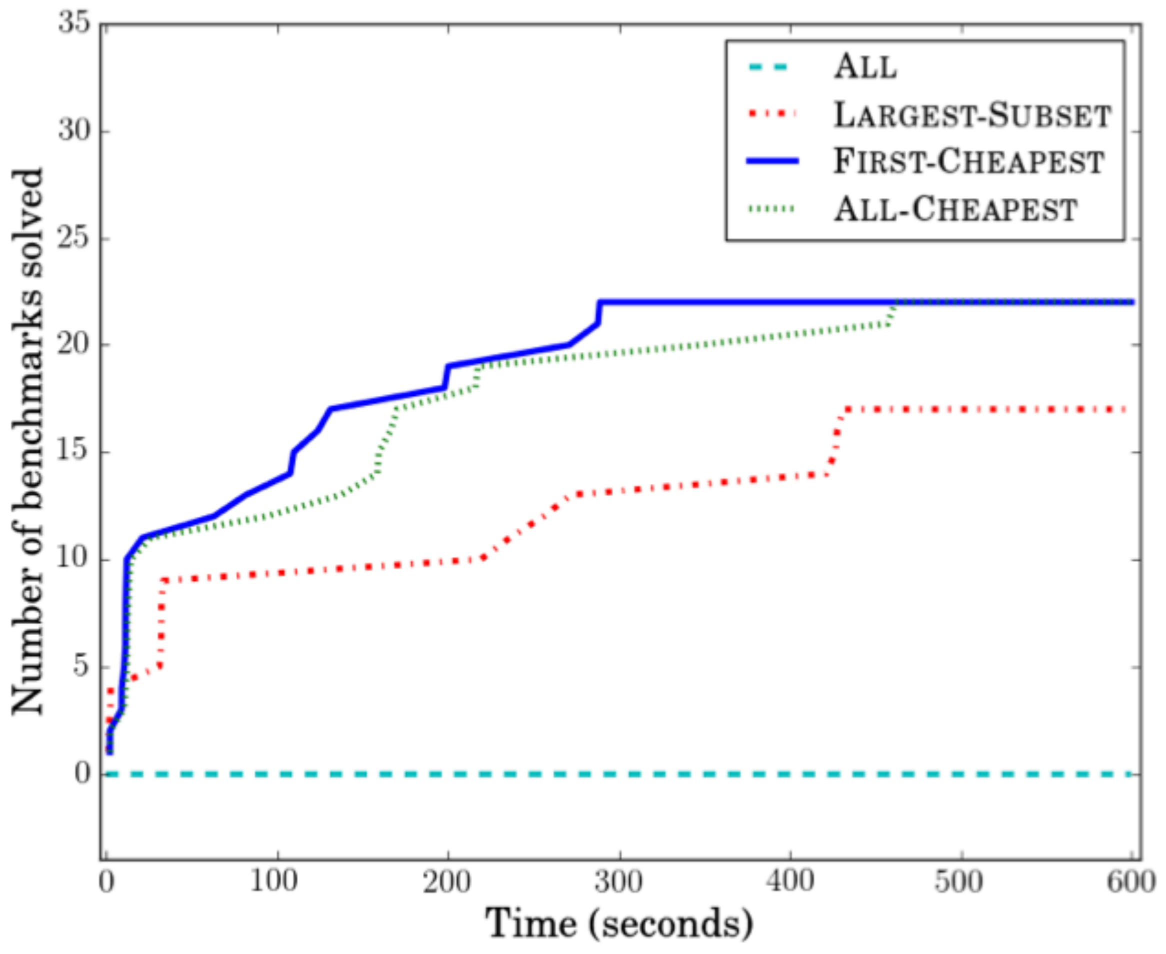
Partial Solution Selection Strategies



(a) String domain



(b) BitVec domain



(c) Circuit domain

TF-Coder results