Effective Program Generation using Learned Probabilistic Models

Woosuk Lee

Joint work with Kihong Heo, Pardis Pashakhanloo, Rajeev Alur, Mayur Naik

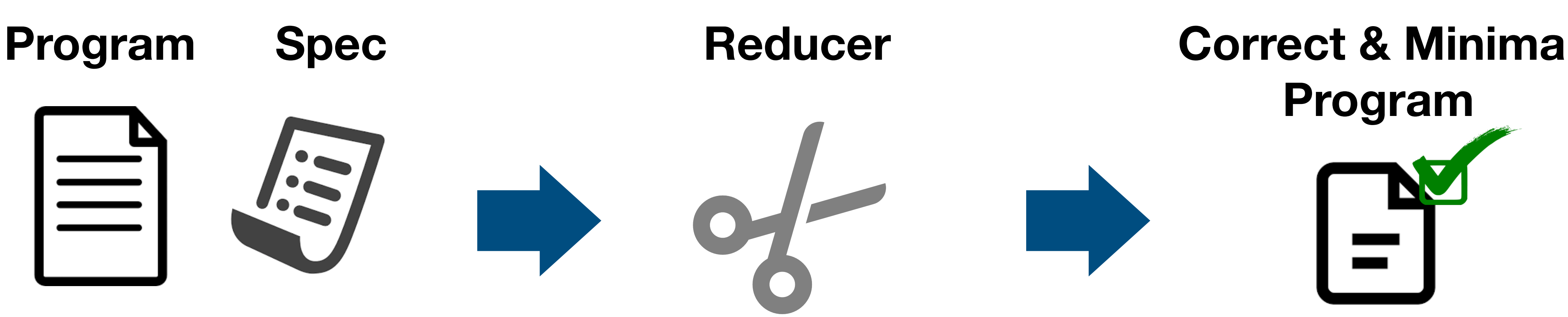
University of Pennsylvania

Hanyang University

Program Generation

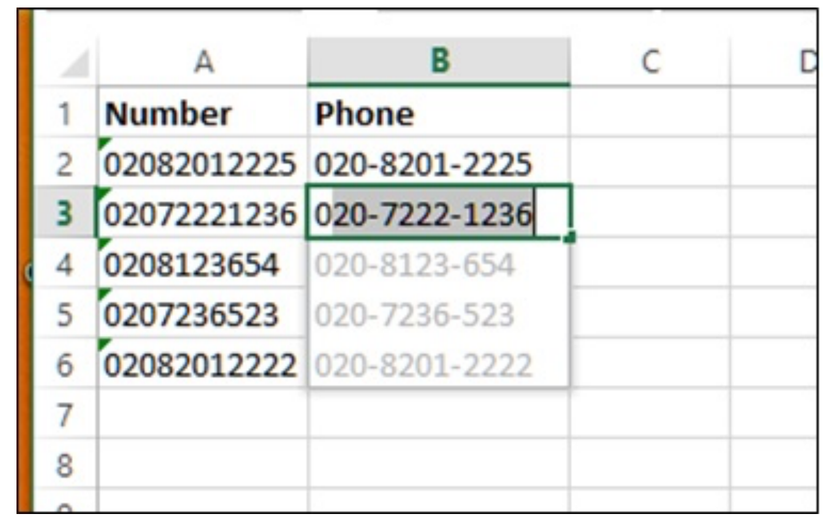
- Synthesis: automated generation from high-level specs.

- Reduction: automated minimization from high-level specs.



Applications

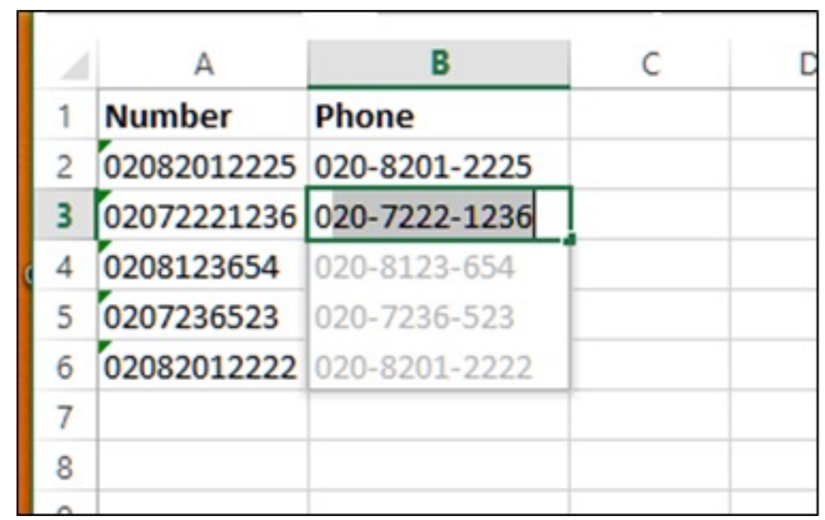
- Program synthesis
    - Programming-by-example (e.g., Microsoft Excel FlashFill)



    - Program verification
    - Program optimization
- Program reduction
    - Fault localization
    - Test case minimization (e.g., minimizing GCC bugs)
    - Attack surface reduction

Applications

- Program synthesis
  - Programming-by-example (e.g., Microsoft Excel FlashFill)
  - Program verification
  - Program optimization
- Program reduction
  - Fault localization



	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				

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

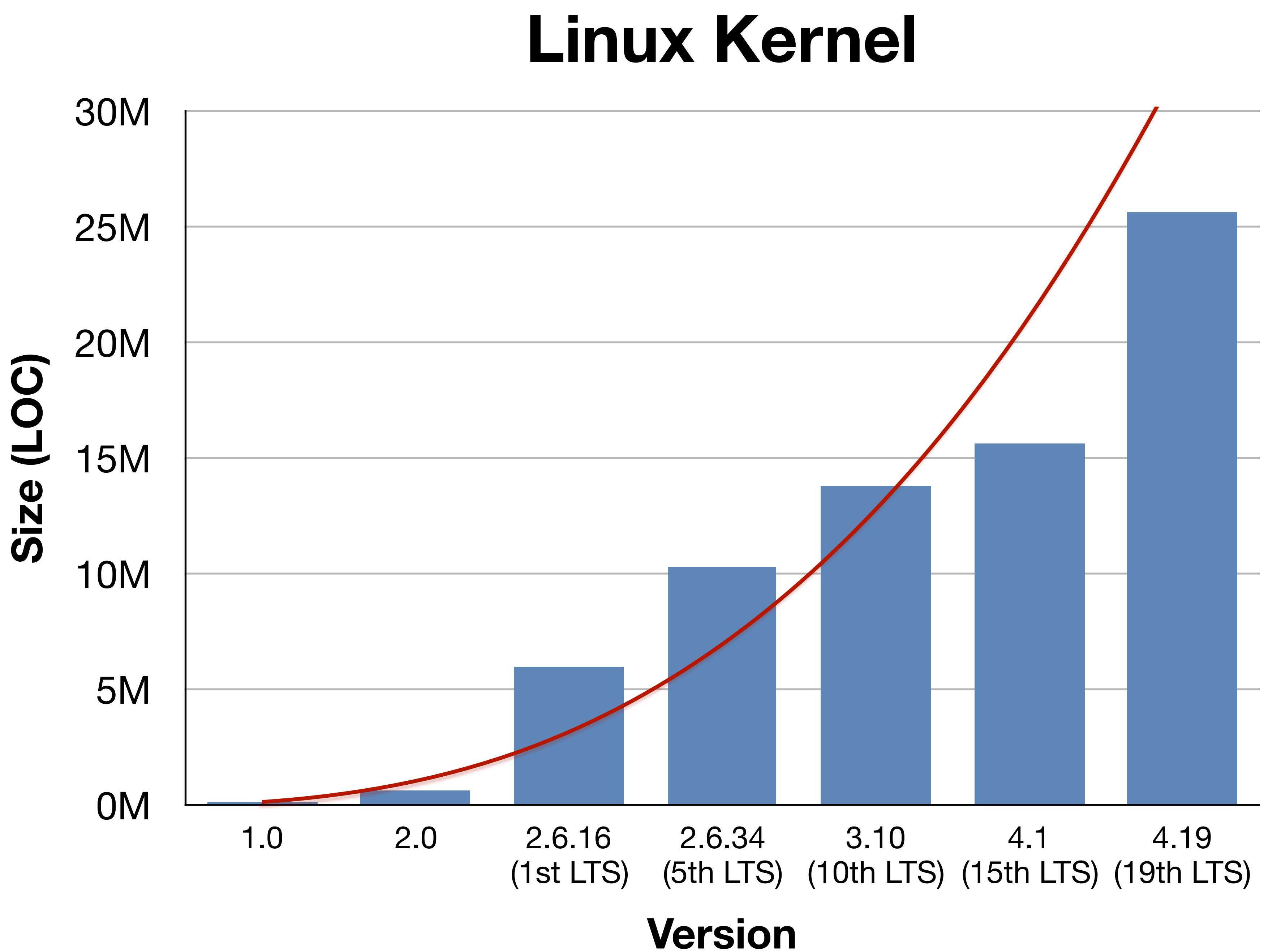
Publications

- Effective Program Debloating via Reinforcement Learning
  Kihong Heo*, **Woosuk Lee***, Pardis Pashakhanloo and Mayur Naik
  CCS 2018: 25th ACM Conference on Computer and Communications Security, 2018 (* contributed equally)

- Accelerating Search-Based Program Synthesis Using Learned Probabilistic Models
  Woosuk Lee, Kihong Heo, Rajeev Alur, Mayur Naik
  PLDI 2018: 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2018.

Effective Program Debloating via Reinforcement Learning (CCS’18)

Growth of SW Complexity

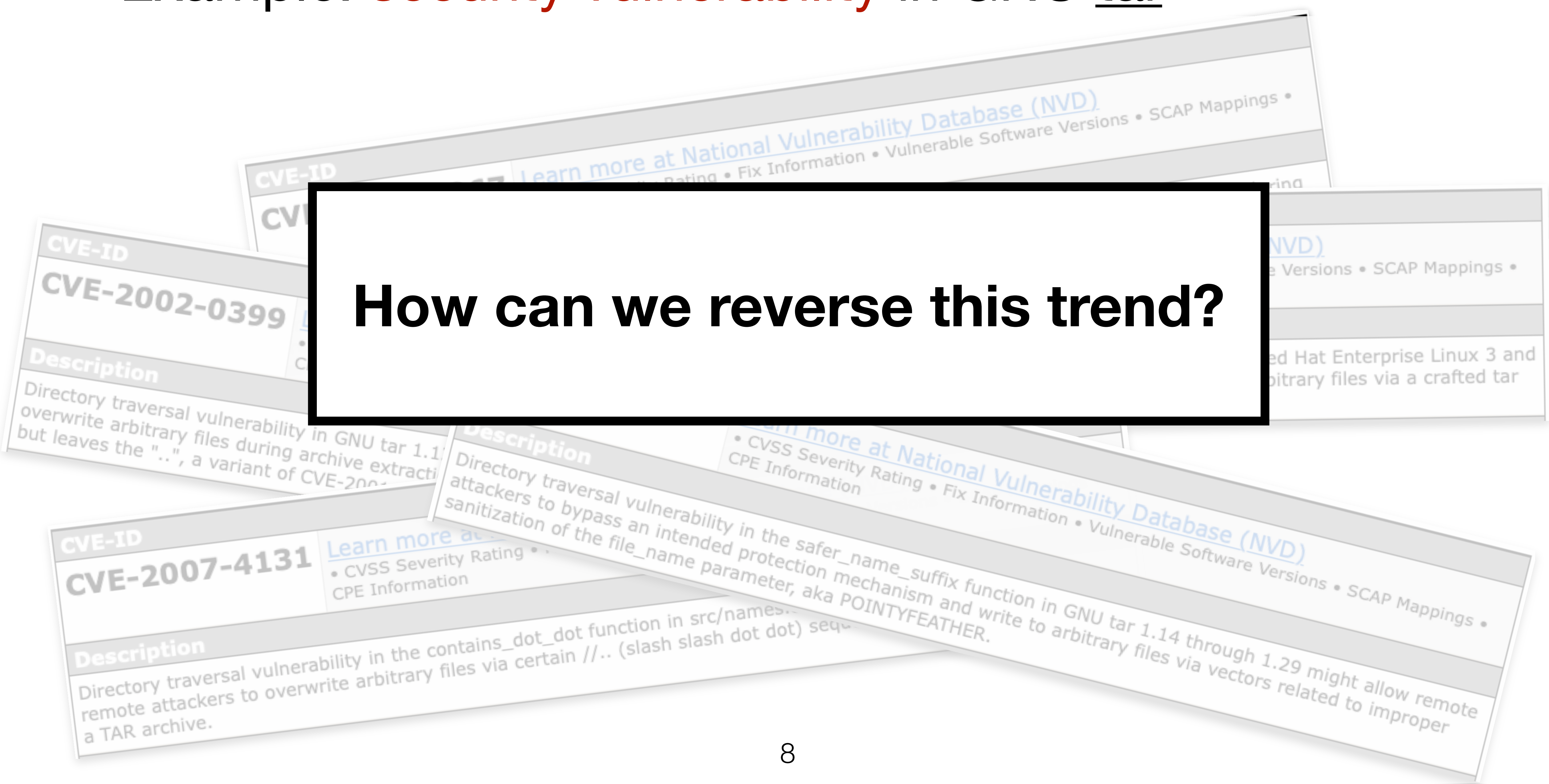


*https://www.linuxcounter.net

Consequences of SW Bloat

Performance	Maintainability	Security

- Example: security vulnerability in GNU tar



How can we reverse this trend?

State-of-the-Practice

General-purpose <u>tar</u>

- Out-of-the-box Linux

Customized <u>tar</u>

- BusyBox Utility Package*

*https://busybox.net

State-of-the-Practice

General-purpose tar

- Out-of-the-box Linux
- 97 cmd line options

Customized tar

- BusyBox Utility Package*
- 8 cmd line options

*https://busybox.net

State-of-the-Practice

General-purpose tar

- Out-of-the-box Linux
- 97 cmd line options
- 45,778 LOC
- 13,227 statements

Customized tar

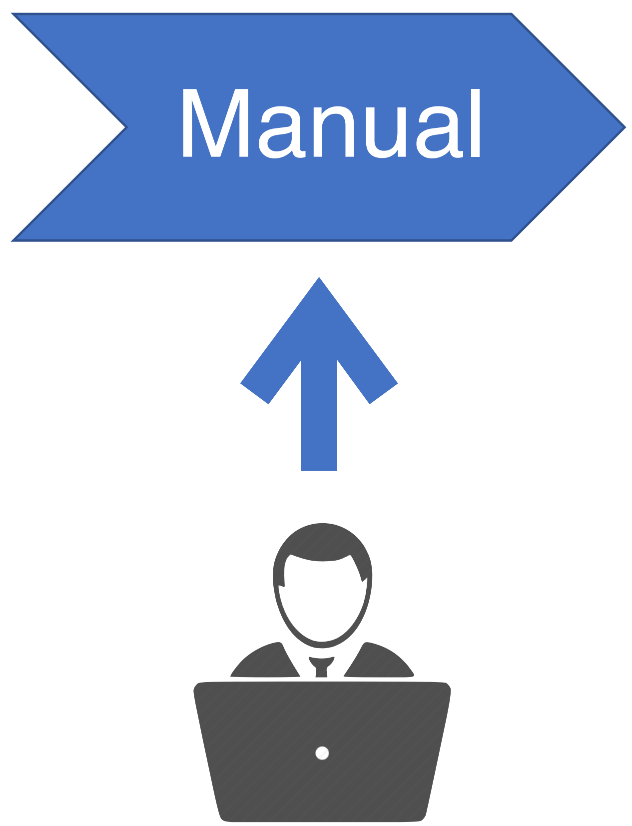
- BusyBox Utility Package*
- 8 cmd line options
- 3,287 LOC
- 403 statements

*https://busybox.net

State-of-the-Practice

General-purpose tar

- Out-of-the-box Linux
- 97 cmd line options
- 45,778 LOC
- 13,227 statements
- CVE-2016-6321



Customized tar

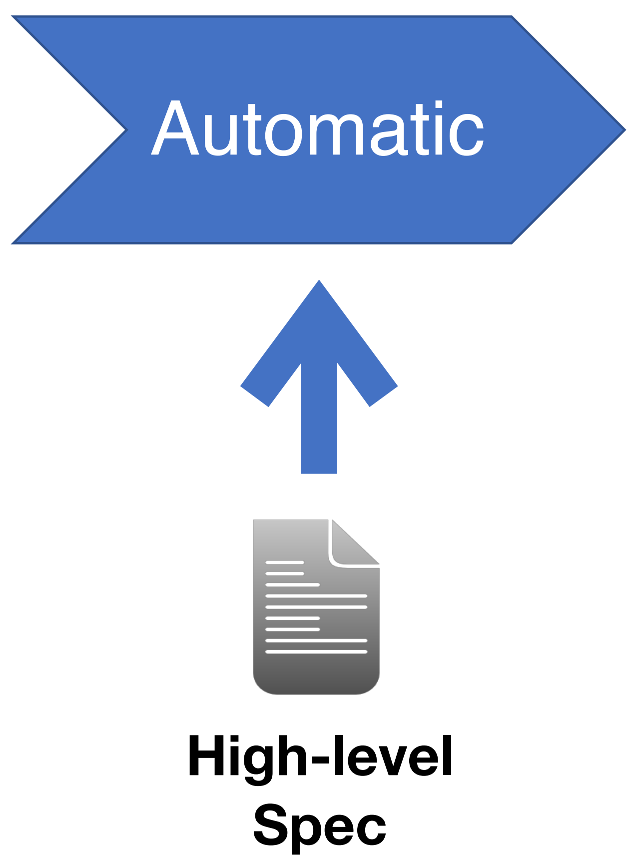
- BusyBox Utility Package*
- 8 cmd line options
- 3,287 LOC
- 403 statements
- No known CVEs

*https://busybox.net

Our Goal

General-purpose tar

- Out-of-the-box Linux
- 97 cmd line options
- 45,778 LOC
- 13,227 statements
- CVE-2016-6321



Customized tar

- BusyBox Utility Package*
- 8 cmd line options
- **1,646** ~~3,287~~ LOC
- **518** ~~403~~ statements
- No known CVEs

*https://busybox.net

Our Contribution

Chisel: an automated program debloating system†

- **minimality**: trim code as aggressively as possible
- **efficiency**: scale to large programs
- **robustness**: avoid introducing new vulnerabilities
- **naturalness**: produce maintainable code
- **generality**: handle a variety of programs and specs

† http://chisel.cis.upenn.edu

Reduced version of tar-1.14

Global variable declarations removed

```c
int absolute_names;
int ignore_zeros_option;
struct tar_stat_info stat_info;

char *safer_name_suffix (char *file_name, int link_target) {
    int prefix_len;
    char *p;

    if (absolute_names) {
        p = file_name;
    } else {
       /* CVE-2016-6321 */
       /* Incorrect sanitization if "file_name" contains ".." */
       ...
    }
    ...
    return p;
}

void extract_archive() {
    char *file_name = safer_name_suffix(stat_info.file_name, 0);
    /* Overwrite "file_name" if exists */
    ...
}

void list_archive() { ... }
```

Code containing CVE removed

Overwriting functionalities removed

```c
void read_and(void *(do_something)(void)) {
    enum read_header status;
    while (...) {
        status = read_header();
        switch (status) {
        case HEADER_SUCCESS: (*do_something)(); continue;
        case HEADER_ZERO_BLOCK:
          if (ignore_zeros_option) continue;
          else break;
        ...
        default: break;
        }
    }
    ...
}

/* Supports all options: -x, -t, -P, -i, ... */
int main(int argc, char **argv) {
    int optchar;
    while (optchar = getopt_long(argc, argv) != -1) {
        switch(optchar) {
        case 'x': read_and(&extract_archive); break;
        case 't': read_and(&list_archive); break;
        case 'P': absolute_names = 1; break;
        case 'i': ignore_zeros_option = 1; break;
        ...
        }
    }
    ...
}
```

Unnecessary functionalities removed

Unsupported options removed

Exploit

CVE-2016-6321

- `tar` uses a sanitization mechanism to handle archives containing '`../`' in their target pathname
- (e.g., '`a/../b`' → '`b`')
- This sanitization is flawed, and attackers can exploit it.

Exploit

- Suppose the root user intends to only extract a file from a downloaded archive and write to '`/etc/motd`' (at the root dir).

```
$> tar xf malicious.tar etc/motd
```

- "`malicious.tar`" contains an entry whose pathname is `'etc/motd/../etc/shadow'`
- The file '`/etc/shadow`' (containing actual passwords in encrypted format for users' (including the root) accounts) is changed **(may lead to a full system compromise)**.
- Note that `'/etc/shadow'` should not be extracted when asking for `'/etc/motd'`.

Exploit

Security vulnerability (path traversal): CVE-2016-6321

```
root:/$ cat etc/shadow
root:l1k4qj1xQWErkzQW1:0:99999:7:::
root:/$ tar xv malicious.tar etc/motd
root:/$ cat etc/shadow
Your system has been compromised :)
root:/$ _
```

*https://seclists.org/fulldisclosure/2016/Oct/96

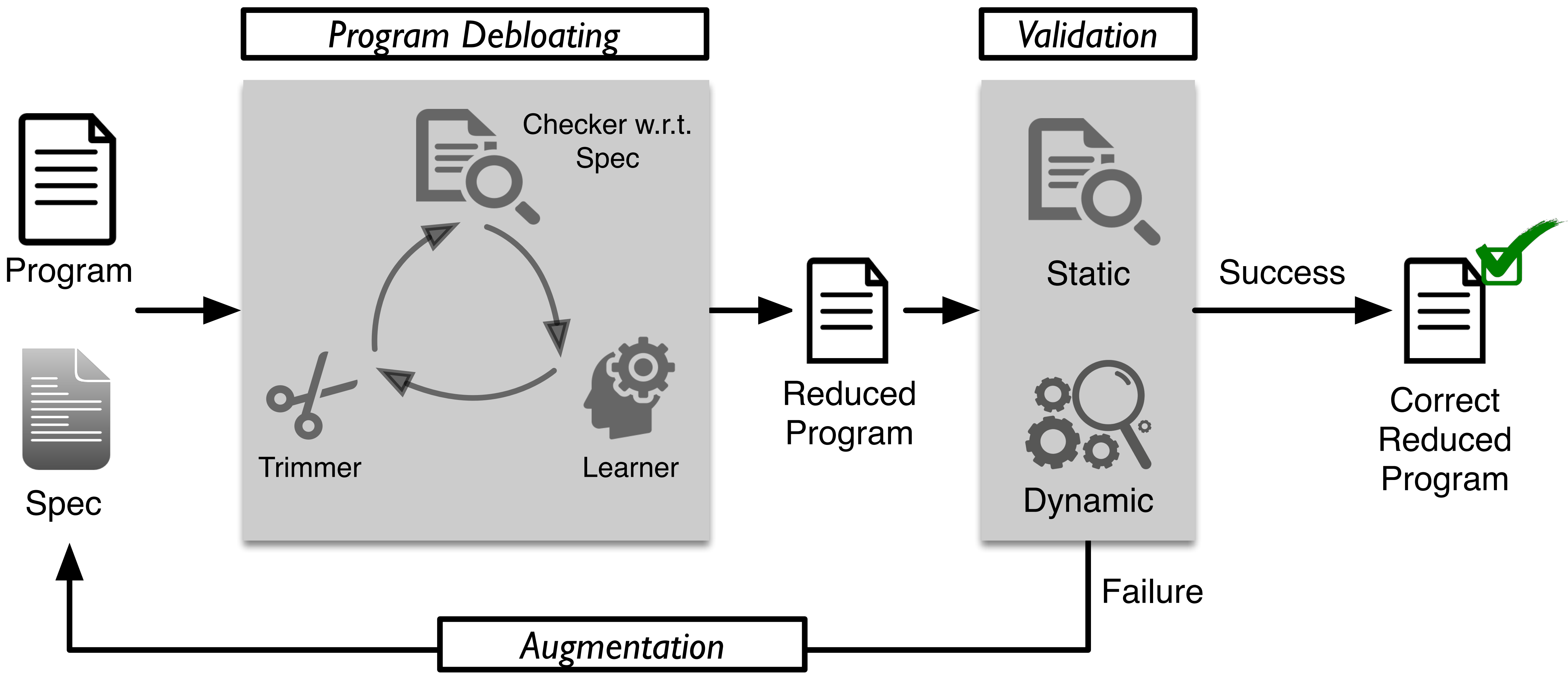
Exploit Removed

Original Ver. of Tar
45,778 LOC

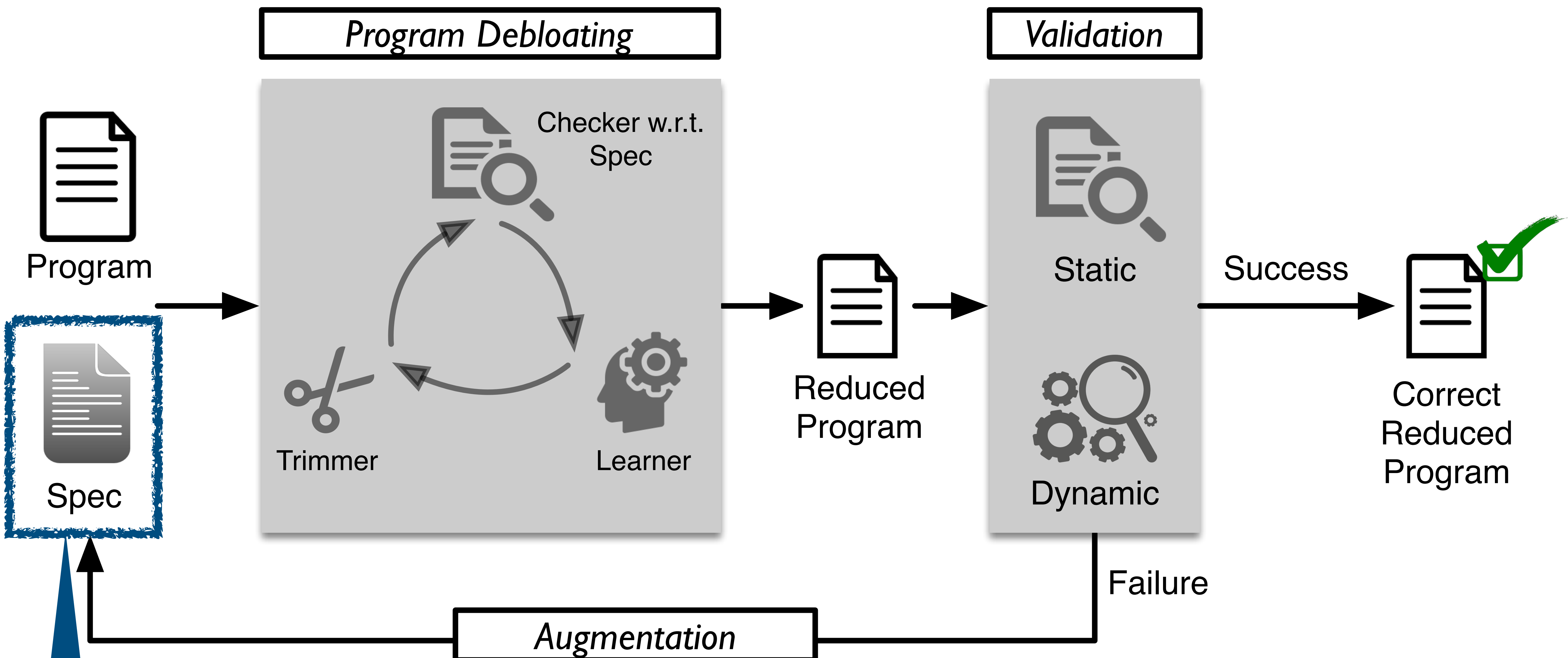
Debloated Ver. of Tar
1,646 LOC

```
root:/$ cat etc/shadow
root:l1k4qj1xQWErkzQW1:0:99999:7:::
root:/$ tar xv malicious.tar etc/motd
root:/$ cat etc/shadow
root:l1k4qj1xQWErkzQW1:0:99999:7:::
root:/$ _
```

System Architecture

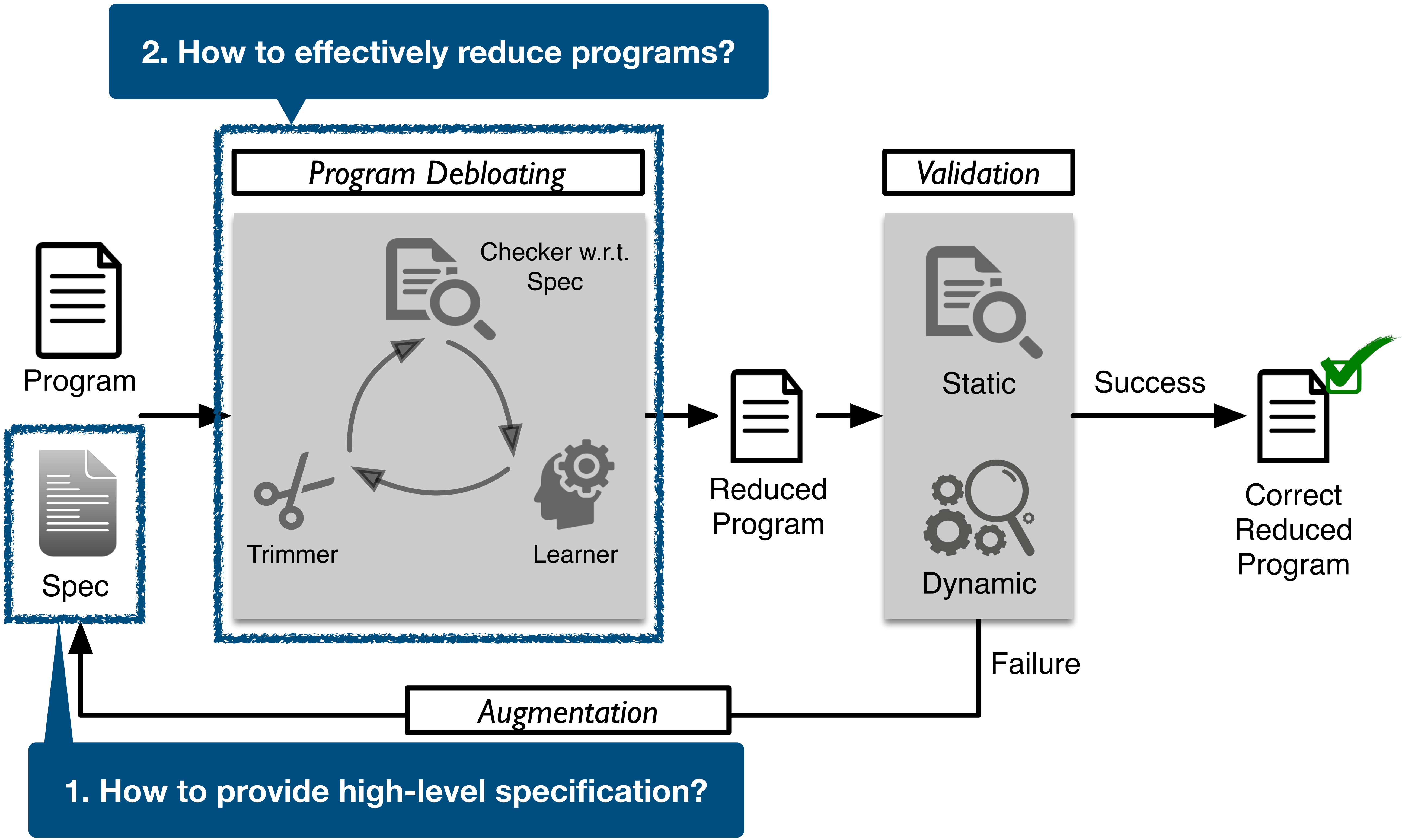


Key Questions

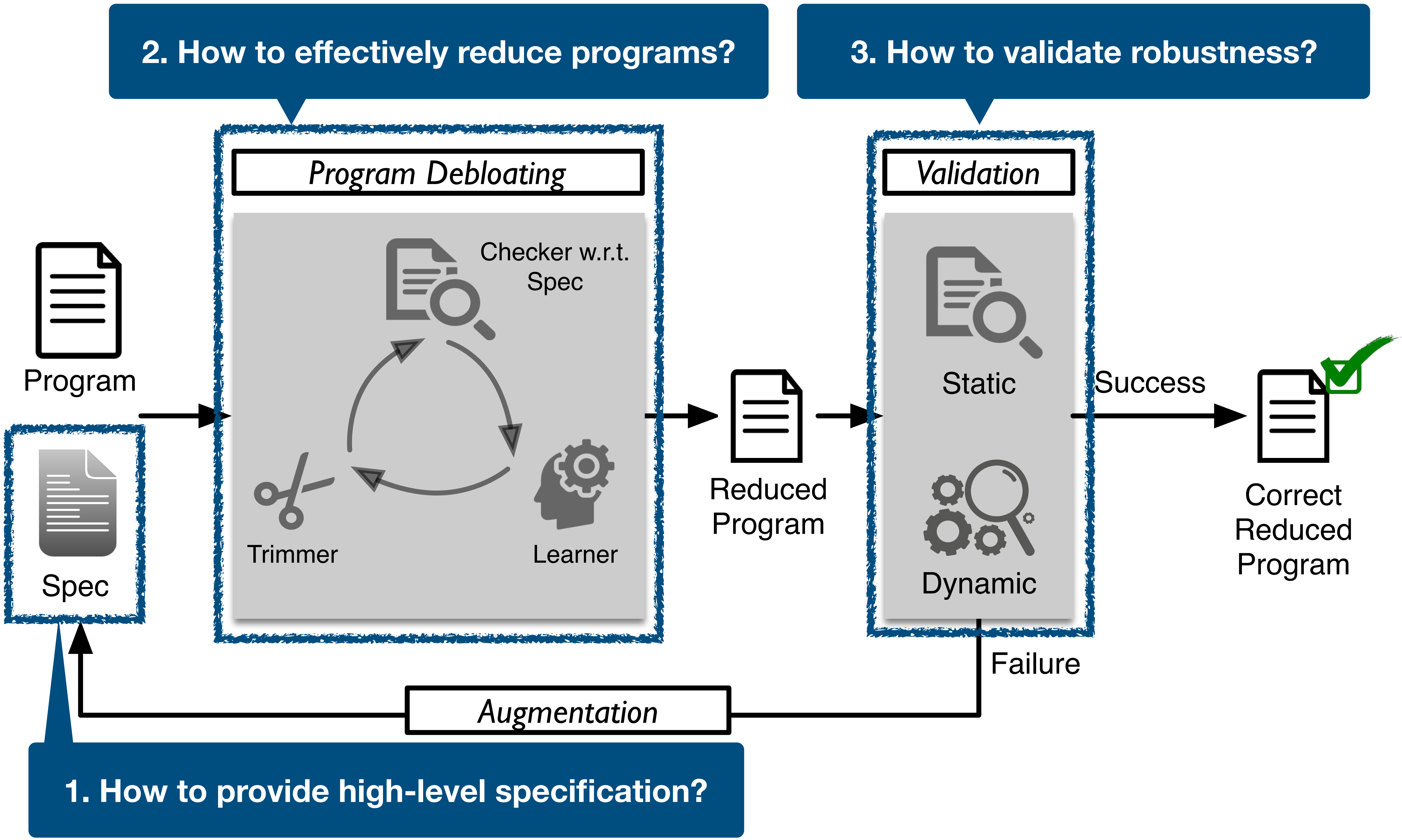


1. How to provide high-level specification?

Key Questions



Key Questions



High-level Specification

```bash
#!/bin/bash

function compile {
  clang -o tar.debloat tar-1.14.c
  return $?
}

# tests for the desired functionalities
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

```bash
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}

compile || exit 1
core || exit 1
non_core || exit 1
```

High-level Specification

```
#!/bin/bash

function compile {
  clang -o tar.debloat tar-1.14.c
  return $?
}

# tests for the desired functionalities
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

1. The program is compilable.

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}

compile || exit 1
core || exit 1
non_core || exit 1
```

High-level Specification

```
#!/bin/bash

function compile {
  clang -o tar.debloat tar-1.14.c
  return $?
}
```

```
# tests for the desired functionalities
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

2. The program produces the same results with the desired functionalities.
(e.g., using regression test suites)

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}

compile || exit 1
core || exit 1
non_core || exit 1
```

High-level Specification

```
#!/bin/bash
```

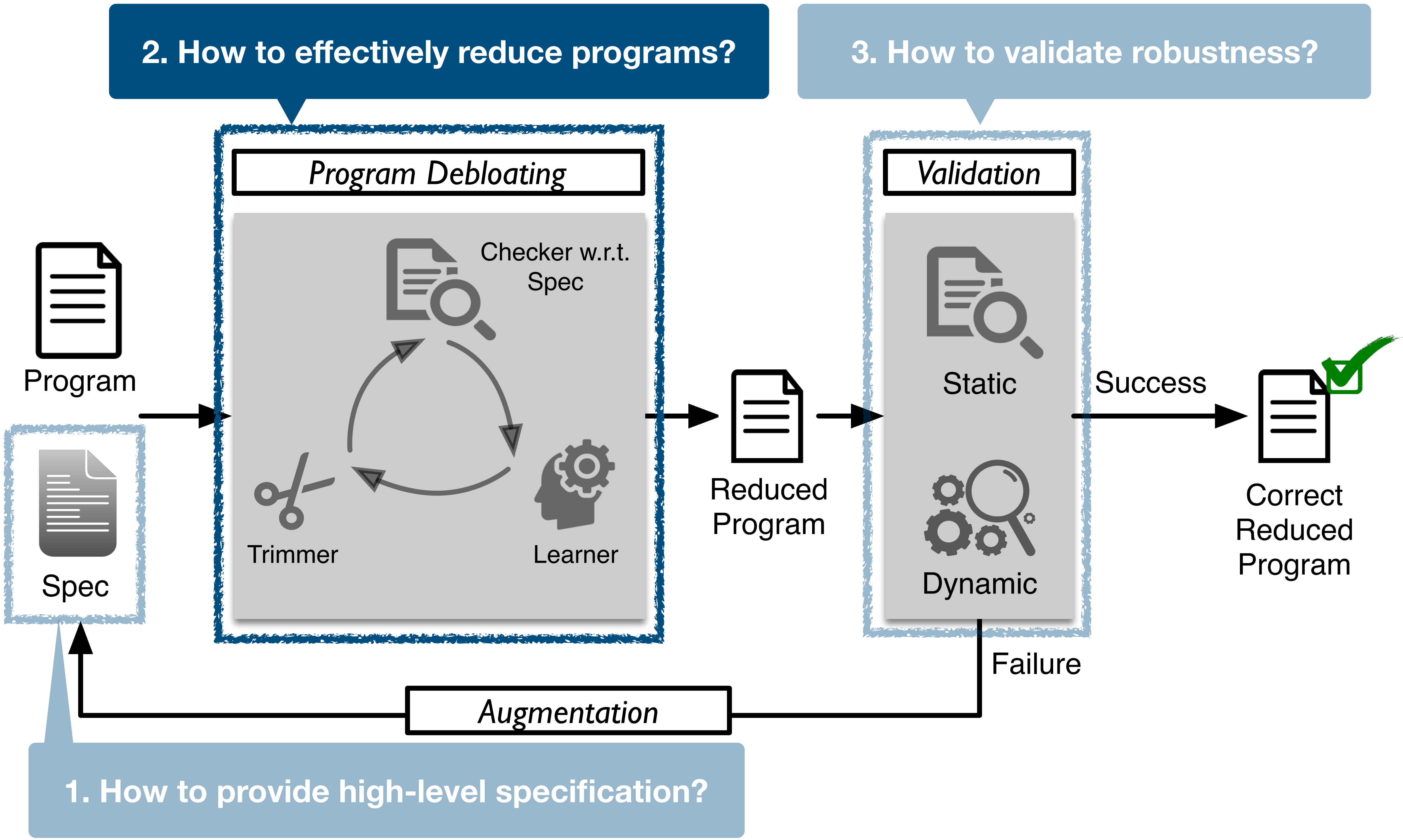
3. The program does not crash with the undesired functionalities. (e.g., using Clang sanitizers)

```
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}
```

```
compile || exit 1
core || exit 1
non_core || exit 1
```

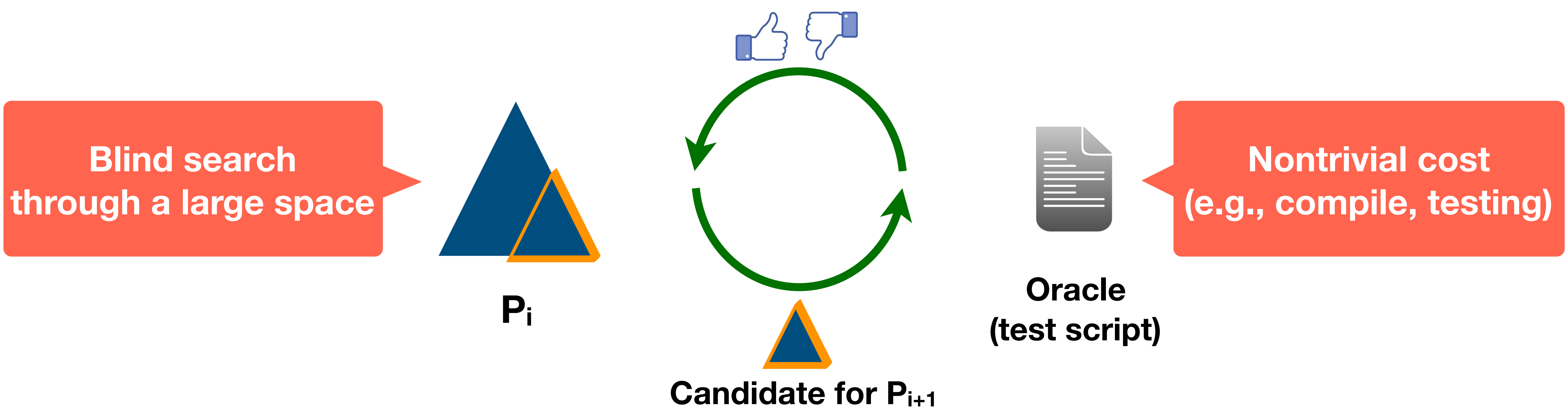
Key Questions



DD: Key Challenges

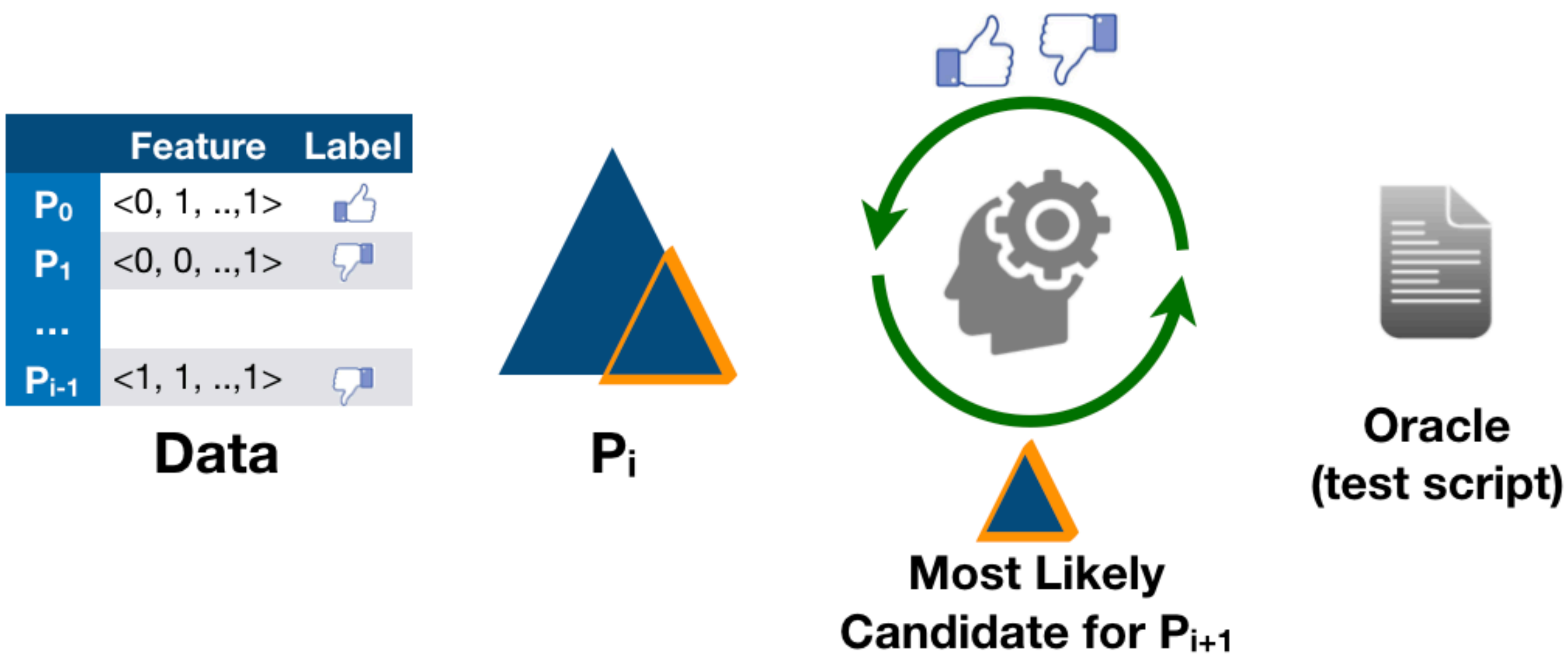
[Zeller and Hildebrandt, 2002]

- Oracle O takes a program and returns Pass or Fail
- Given a program ***P***, find a **1-minimal** ***P****** such that O(***P*****) = Pass
- **1-minimal:** removing any element of ***P****** does not pass O
- Time complexity: O(|***P***|^2)

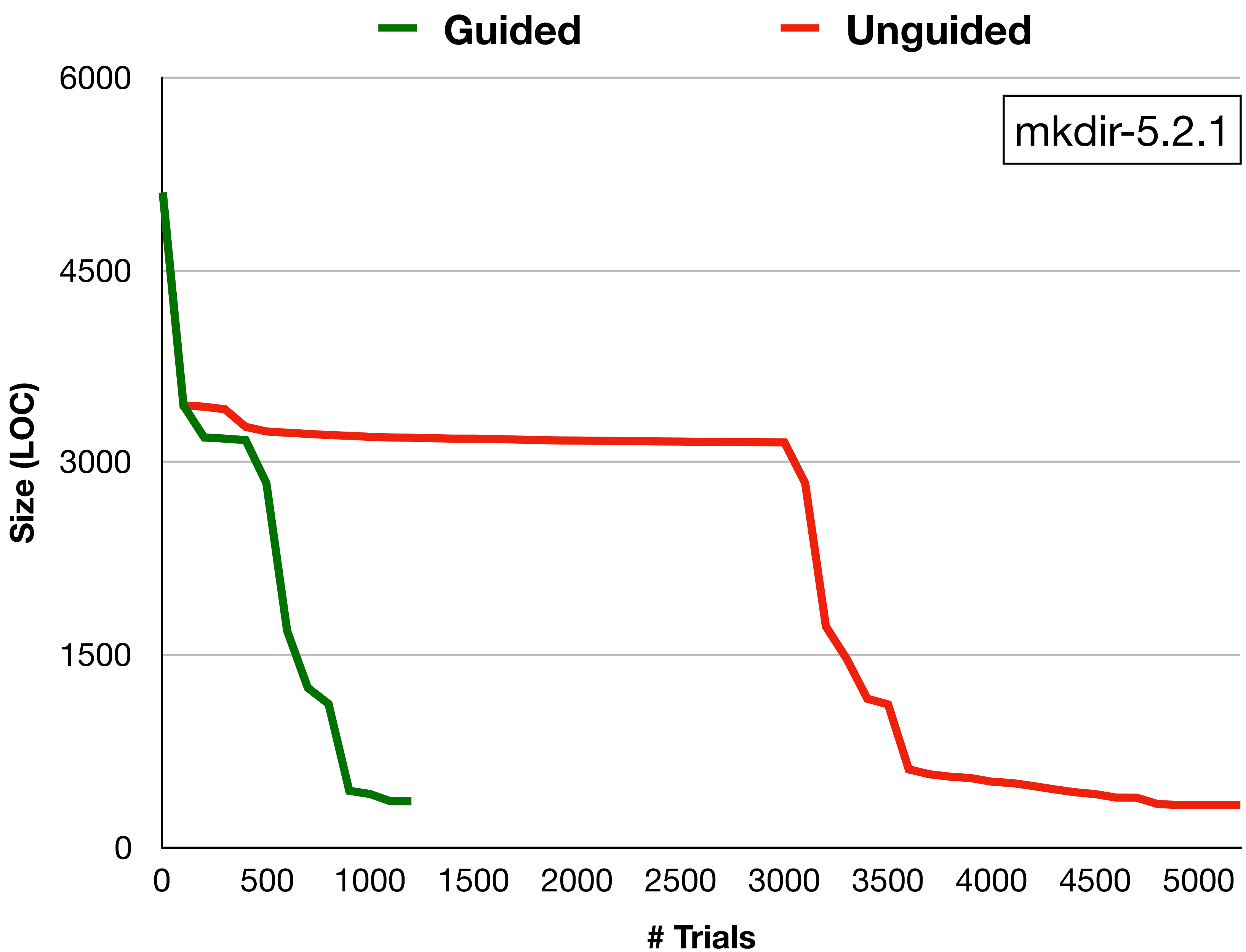


Our Solution: Learning-guided DD

- **Learn a policy** for DD using reinforcement learning (RL)
- **Guide the search** based on the prediction of the learned policy
- Still guarantee **1-minimality** and **O(|*P*|^2) time complexity**
- Discard nonsensical programs upfront
  (e.g., invalid syntax, no main, uninitialized variables, etc)



Our Solution: Learning-guided DD



Example

```
/* mkdir-5.2.1 */
int xstrtol(char *s, char **ptr, int strtol_base, strtol_t *val,
            char *valid_suffixes) {
err = 0;
assert(0 <= strtol_base && strtol_base <= 36);
p = ptr ? ptr : &t_ptr;
q = s;
while(ISSPACE (*q)) ++q;
if (*q == ‘-‘) return LONGINT_INVALID;
errno = 0;
tmp = strtol(s, p, strtol_base);
if (*p == s) { … }
if (!valid_suffixes) { … }
if (**p != ‘\0’) { … }
*val = tmp;
return err;
}
```

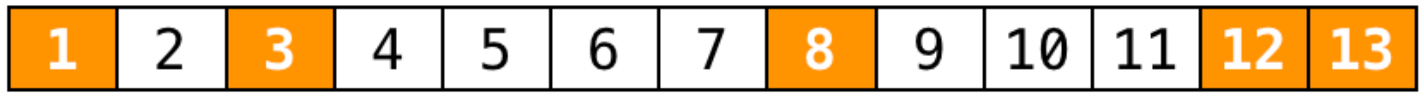
: removed code

Example

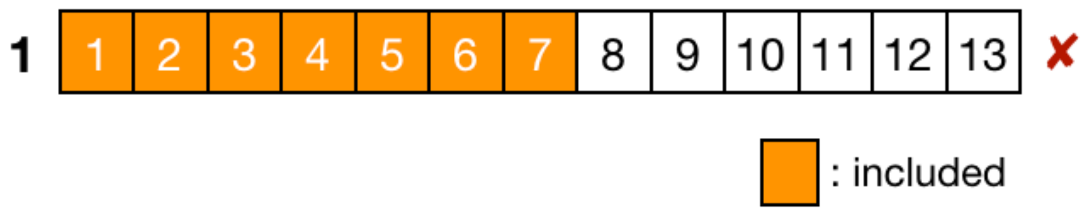
```c
/* mkdir-5.2.1 */
int xstrtol(char *s, char **ptr, int strtol_base, strtol_t *val,
            char *valid_suffixes) {
 1: err = 0;
 2: assert(0 <= strtol_base && strtol_base <= 36);
 3: p = ptr ? ptr : &t_ptr;
 4: q = s;
 5: while(ISSPACE (*q)) ++q;
 6: if (*q == ‘-‘) return LONGINT_INVALID;
 7: errno = 0;
 8: tmp = strtol(s, p, strtol_base);
 9: if (*p == s) { … }
10: if (!valid_suffixes) { … }
11: if (**p != ‘\0’) { … }
12: *val = tmp;
13: return err;
}
```

Minimal Desired Program:

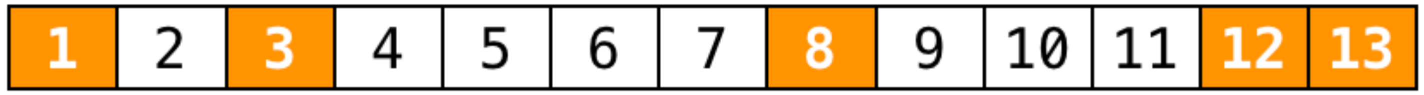
1	2	**3**	4	5	6	7	**8**	9	10	11	**12**	**13**



Unguided Delta-Debugging

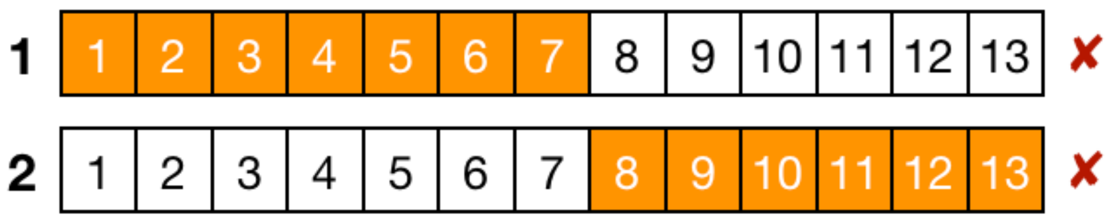


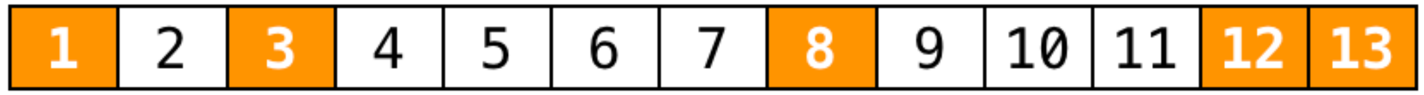
Guided Delta-Debugging



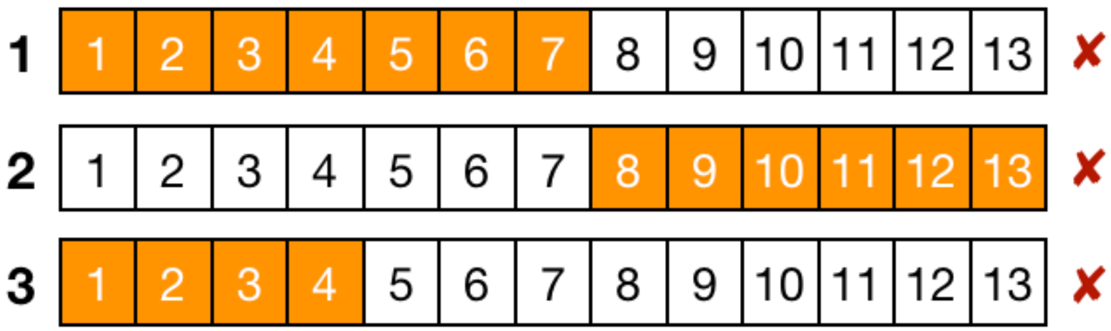
Unguided Delta-Debugging

Guided Delta-Debugging



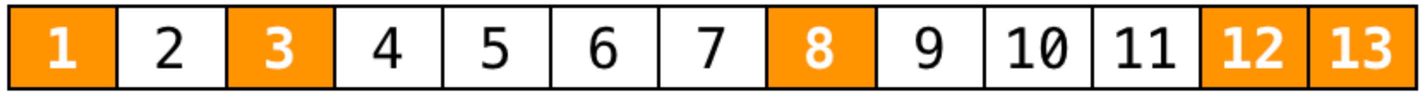


Unguided Delta-Debugging

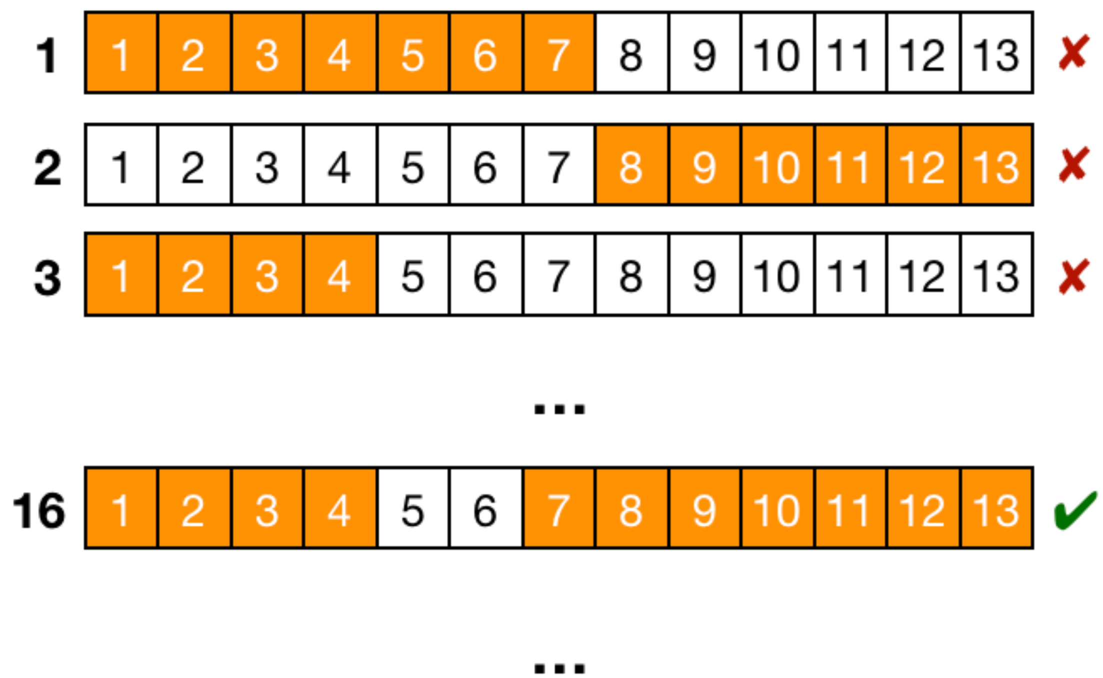


...

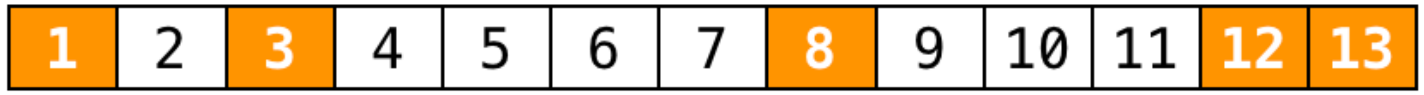
Guided Delta-Debugging



Unguided Delta-Debugging

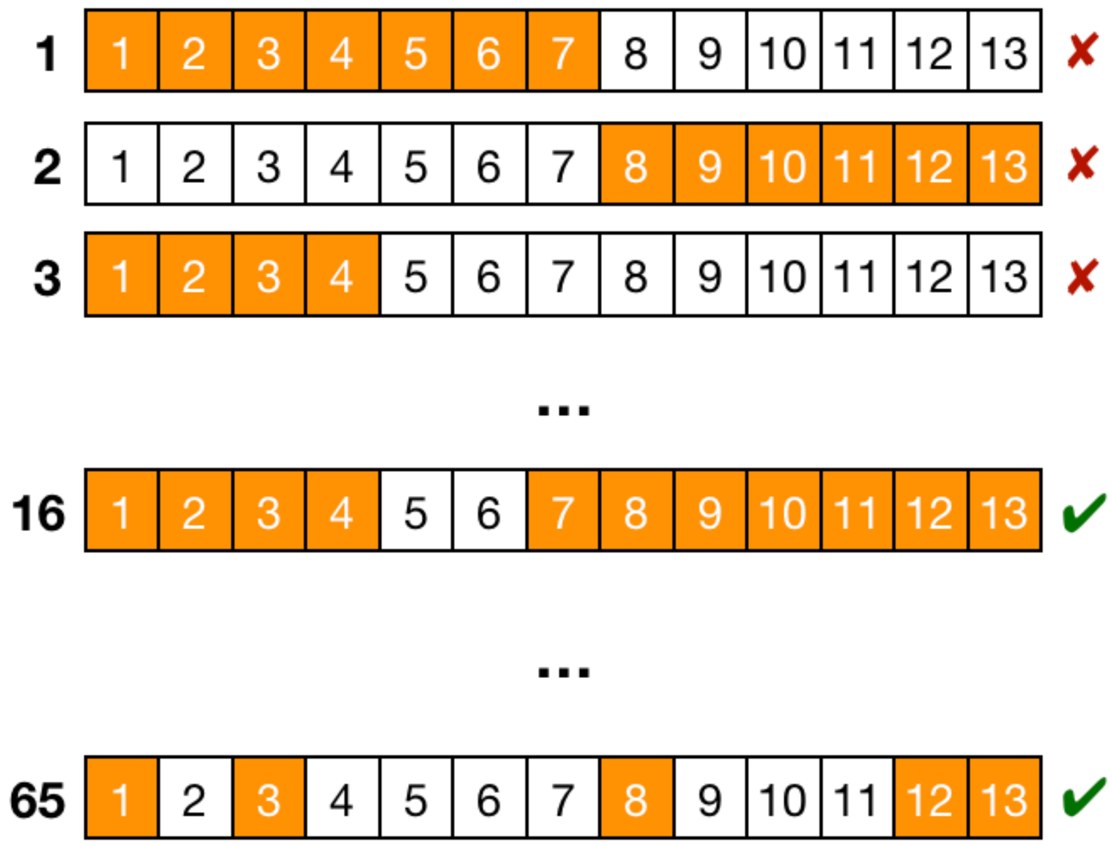


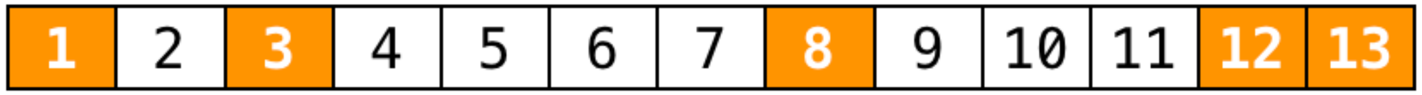
Guided Delta-Debugging



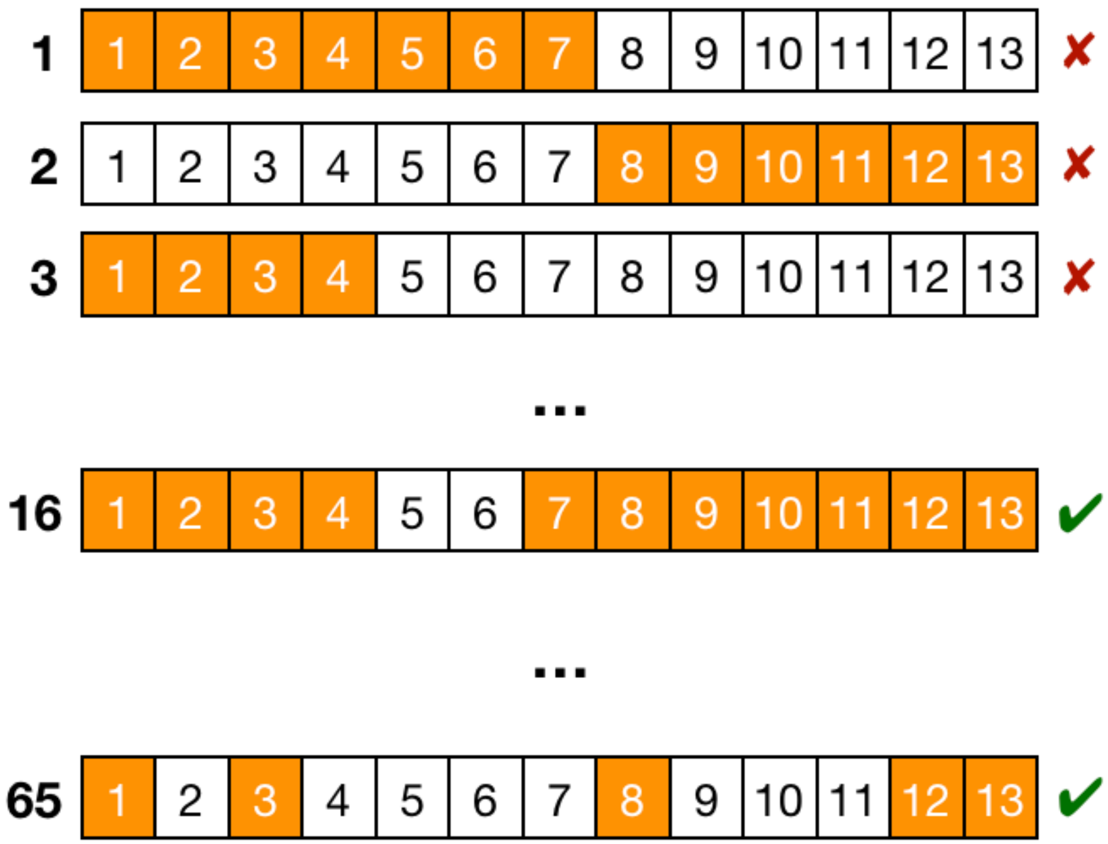
Unguided Delta-Debugging

Guided Delta-Debugging





Unguided Delta-Debugging



Guided Delta-Debugging

1	2	3	4	5	6	7	8	9	10	11	12	13

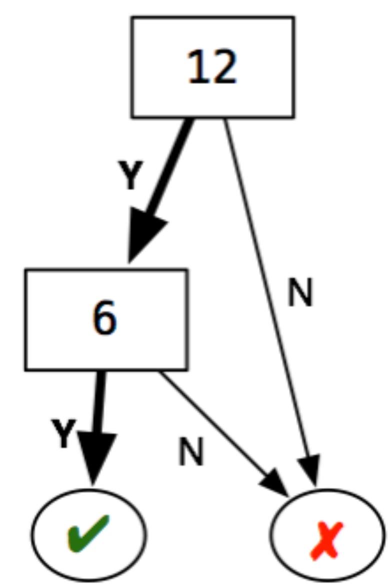
Unguided Delta-Debugging

1	1	2	3	4	5	6	7	8	9	10	11	12	13	✗
2	1	2	3	4	5	6	7	8	9	10	11	12	13	✗
3	1	2	3	4	5	6	7	8	9	10	11	12	13	✗
							...							
16	1	2	3	4	5	6	7	8	9	10	11	12	13	✔
							...							
65	1	2	3	4	5	6	7	8	9	10	11	12	13	✔

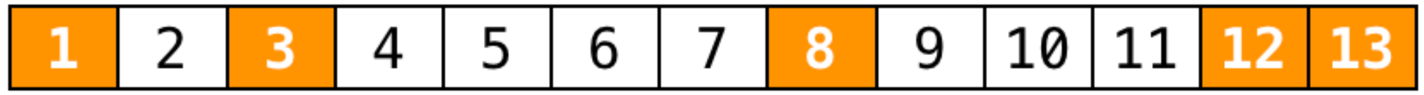
Guided Delta-Debugging

1	1	2	3	4	5	6	7	8	9	10	11	12	13	✗
	1	1	1	1	1	1	1	0	0	0	0	0	0	0

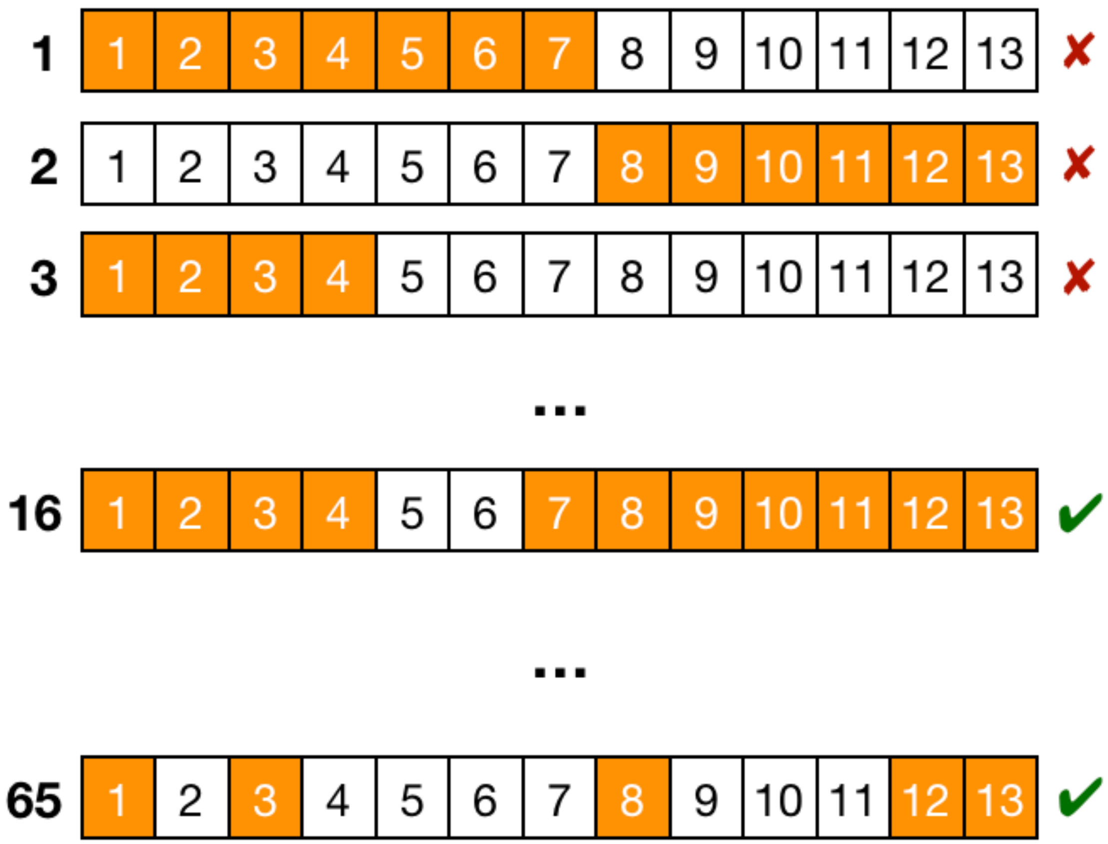
Feature vector Label



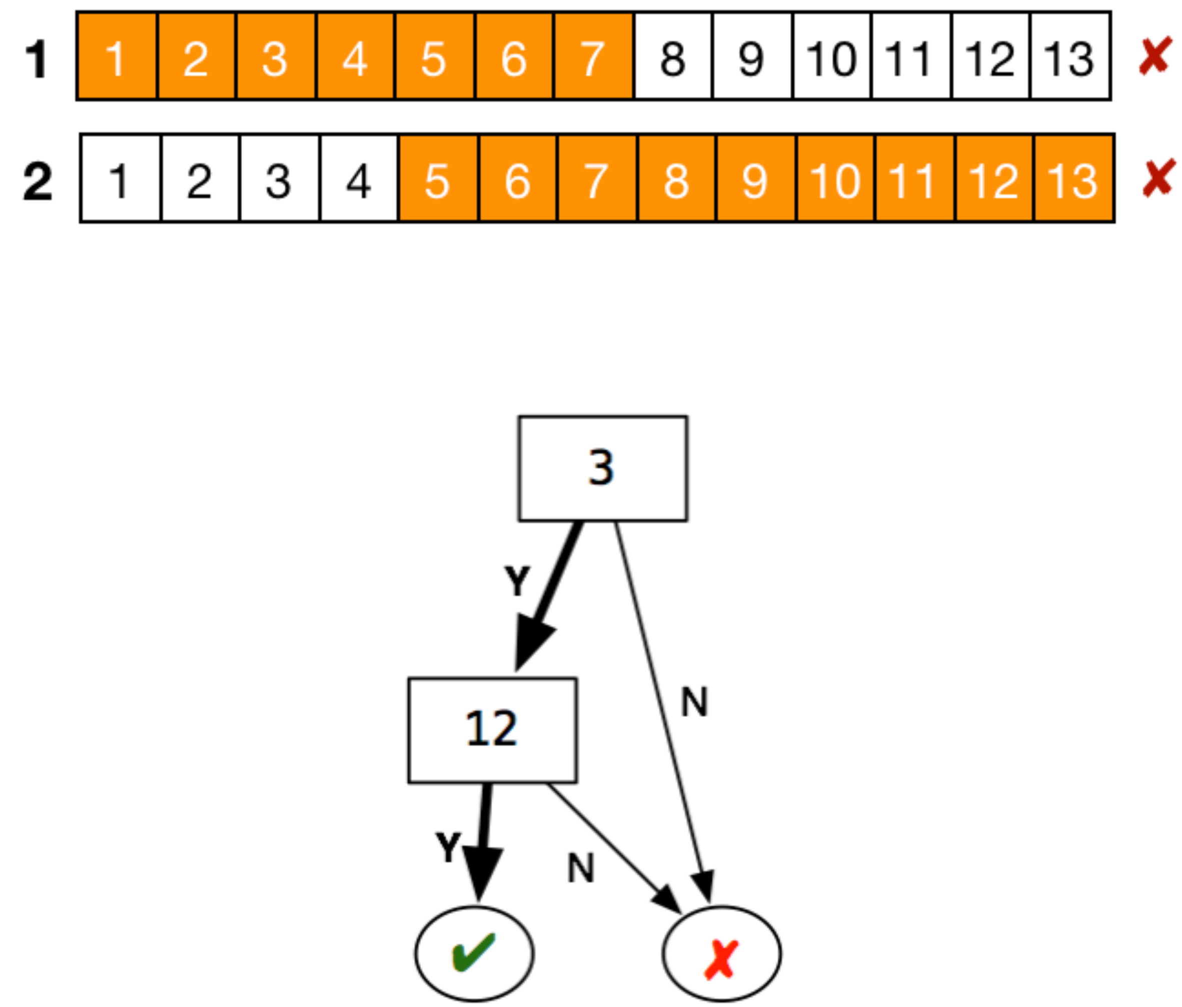
*P** should include 6 and 12

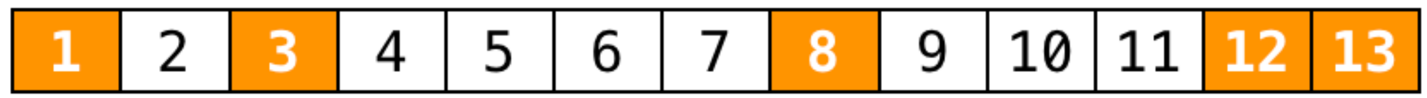


Unguided Delta-Debugging

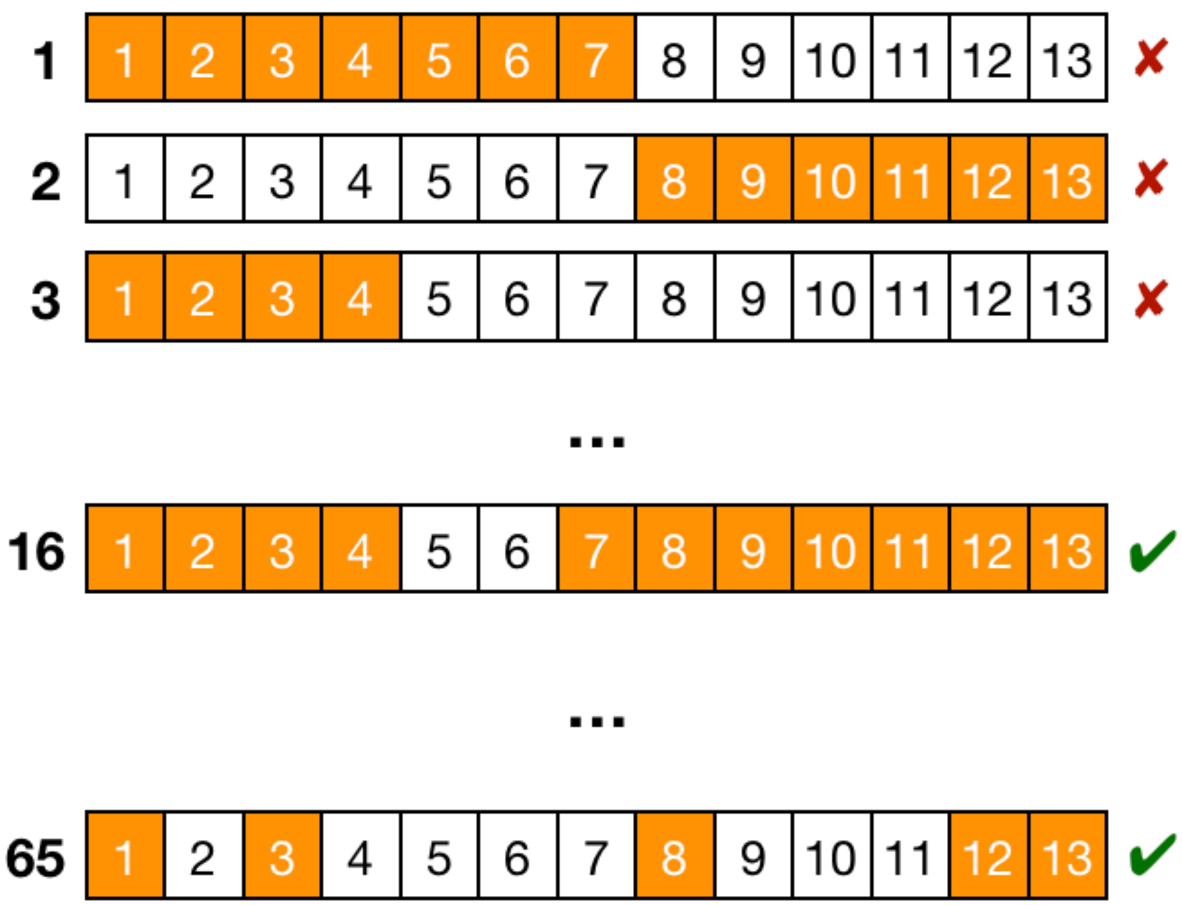


Guided Delta-Debugging



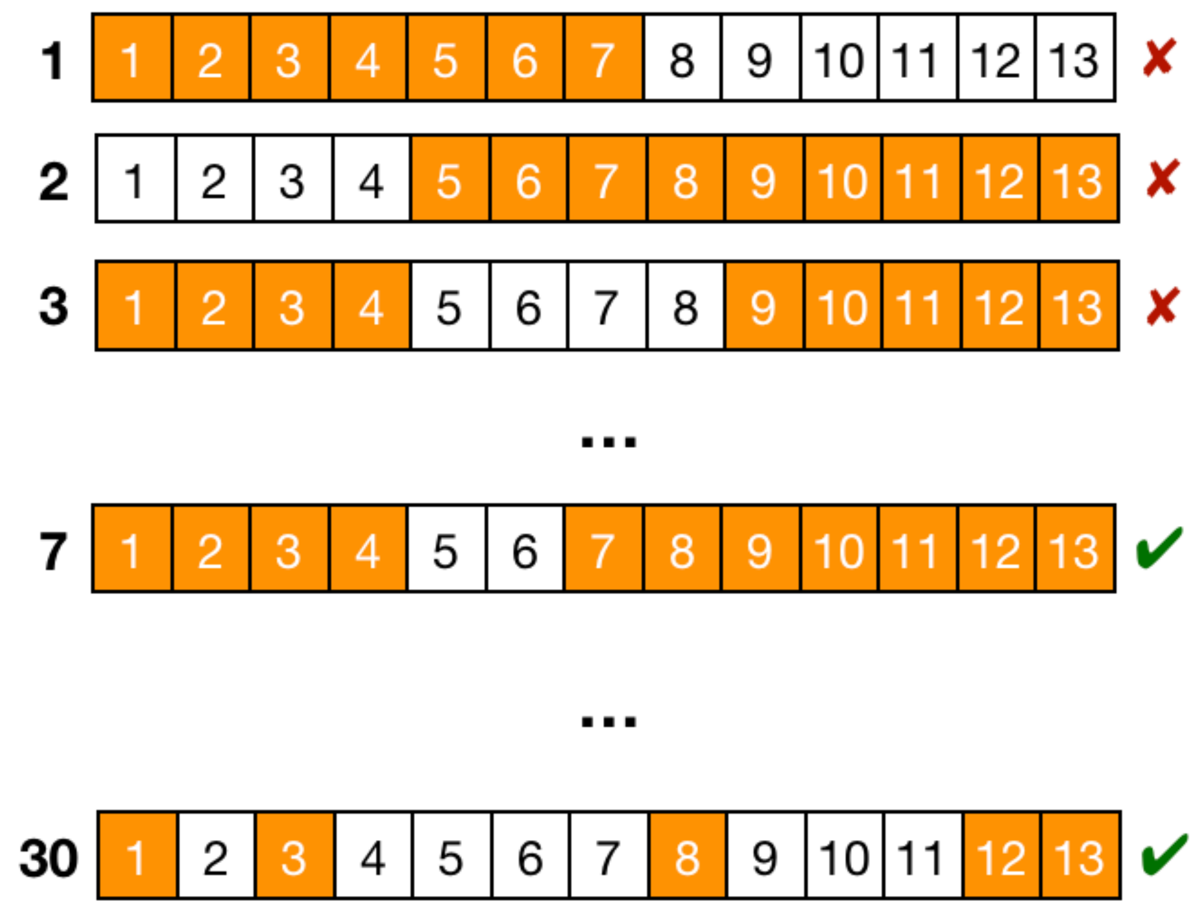


Unguided Delta-Debugging



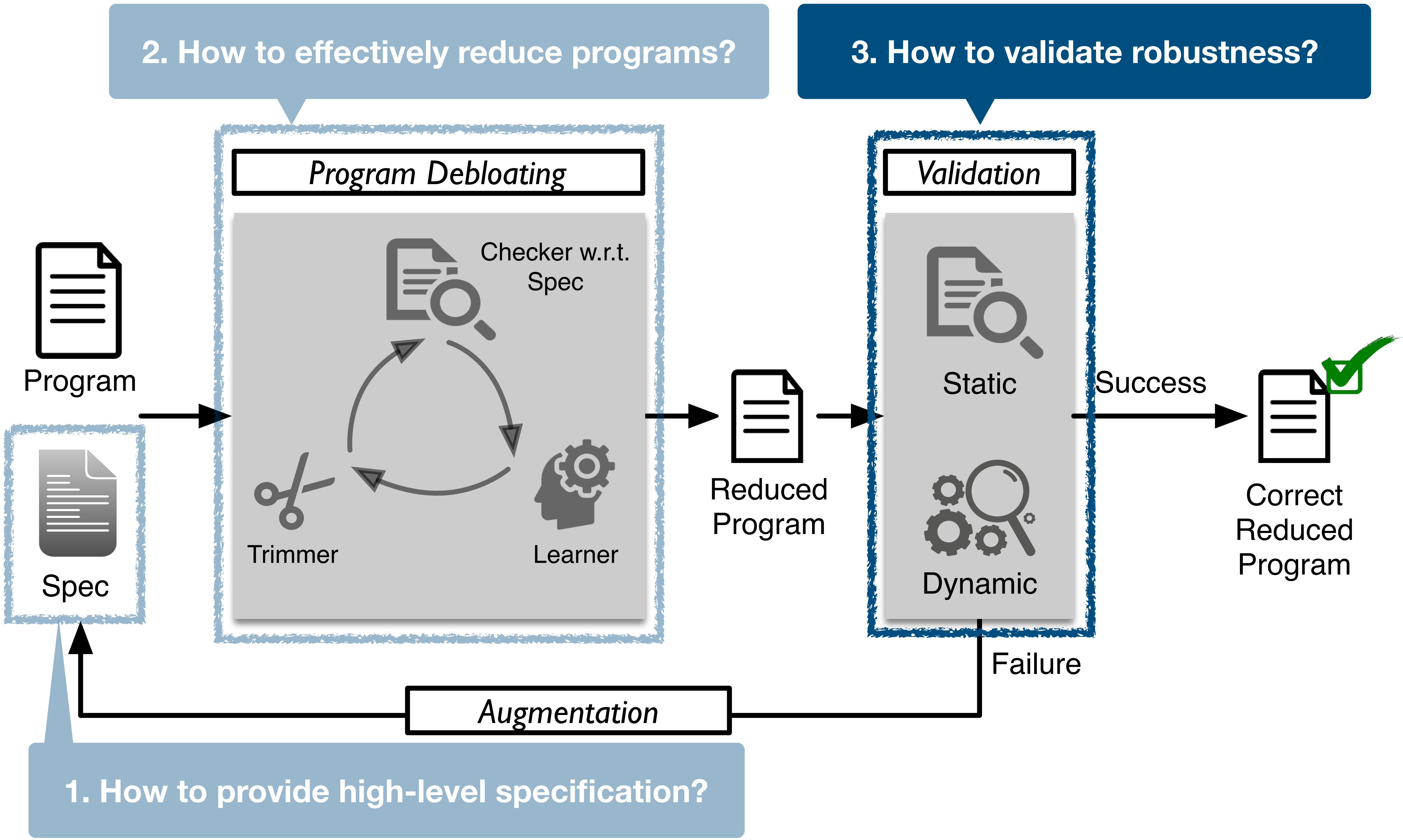
5,169 trials (4,872 failures)

Guided Delta-Debugging



1,174 trials (901 failures)

Key Questions



Validation

- Check the **robustness** of the reduced program
  - preventing newly introduced security holes
- Sound static buffer overflow analyzer (**Sparrow**)
  - #alarms in <u>tar</u>: **1,290** → **19** (feasible for manual inspection)
- Random fuzzer (AFL)
  - no crashing input found in **3 days** for <u>tar</u>

Augmentation

- Augment the test script with crashing inputs by AFL
- Typically converges in up to 3 iterations in practice
- But, may be incomplete

```
/* grep-2.19 */
void add_tok (token t) {
    /* removed in the first trial and restored after augmentation */
    if (dfa->talloc == dfa->tindex)
        dfa->tokens = (token *) realloc (/* large size */);
    *(dfa->tokens + (dfa->tindex++)) = t;
}
```

Talk Outline

- Motivation
- System Architecture
- **Evaluation**
- Conclusion

Experimental Setup

- 10 widely used **UNIX utility programs** (13—90 KLOC)
  - each program has a **known CVE**
  - **remove unreachable code** by static analysis upfront
- Specification:
  - supporting **the same cmd line options** as BusyBox
  - with the **test suites** by the original developers

Size of Reduced Program

#Statement

Program	Original	Chisel	Hand-written
bzip-1.05	6,316	1,575	2,342
chown-8.2	3,422	186	141
date-8.21	4,100	913	107
grep-2.19	10,816	1,071	355
gzip-1.2.4	4,069	1,042	1,058
mkdir-5.2.1	1,746	142	94
rm-8.4	3,470	73	89
sort-8.16	[illegible]	[illegible]	[illegible]
tar-1.14	[illegible]	[illegible]	[illegible]
uniq-8.16	[illegible]	[illegible]	[illegible]
Total	55,848	6,111	4,729

Reachable code by static analysis

Chisel reduced 89%

Comparable to hand-written versions

Security Hardening

Remove 4 and 2 CVEs in undesired and desired functionalities.
4 CVEs are not easily fixable by reduction (e.g., race condition).

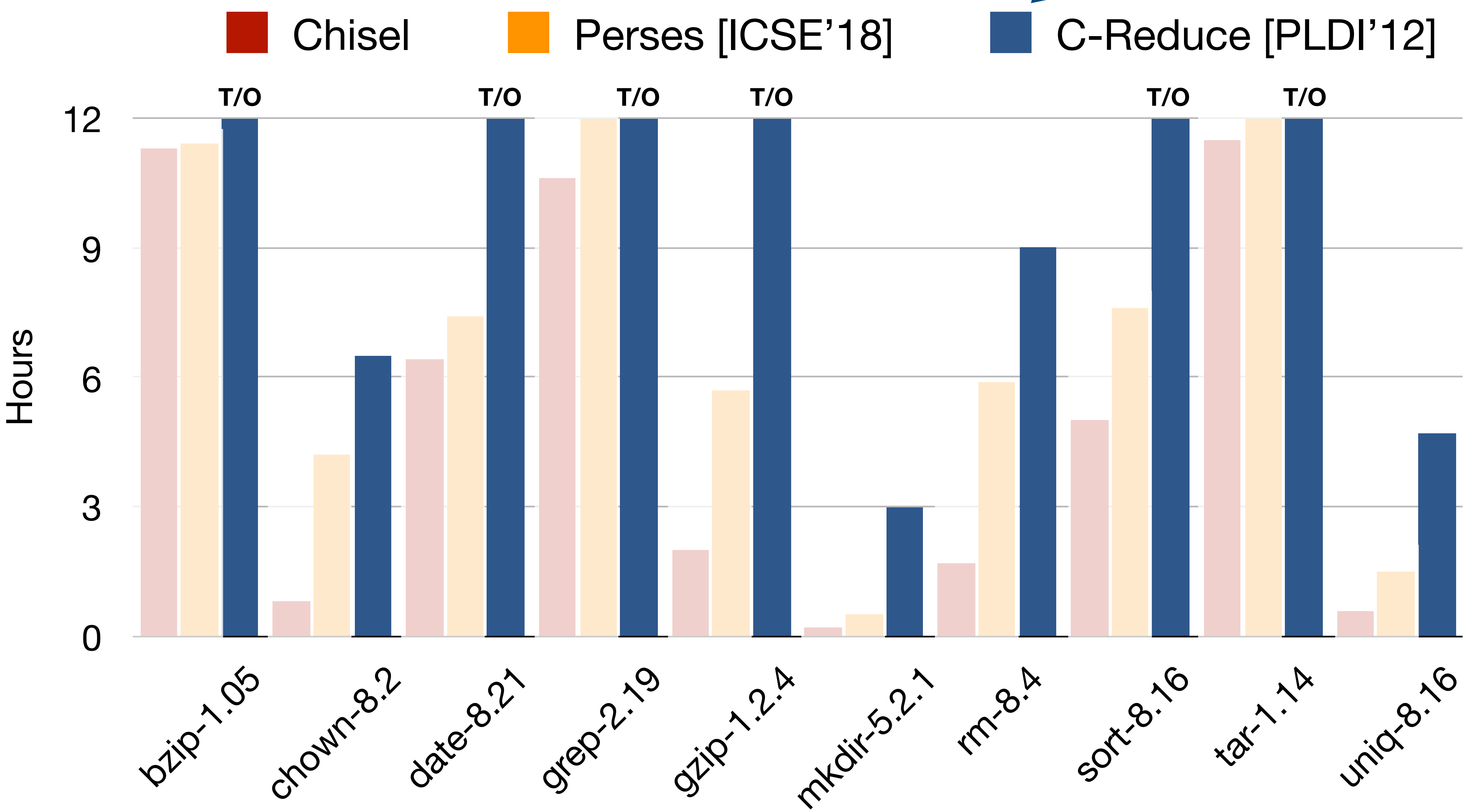
Program	CVE	#ROP Gadgets Original	#ROP Gadgets Reduced		#Alarms Original	#Alarms Reduced	
bzip-1.05	✗	662	298	(55%)	1,991	33	(98%)
chown-8.2	✔	534	162	(70%)	47	1	(98%)
date-8.21	✔	479	233	(51%)	201	23	(89%)
grep-2.19	✔	1,065	411	(61%)	619	31	(95%)
gzip-1.2.4	✔	456	340	(25%)	326	128	(61%)
mkdir-5.2.1	✗	229	124	(46%)	43	2	(95%)
rm-8.4	✗	565	95	(83%)	48	0	(100%)
sort-8.16	✔						
tar-1.14	✔						
uniq-8.16	✗	349	[illegible]	(69%)	60	1	(98%)
Total		6,752	2,285	(66%)	5,298	243	(95%)

Reduced potential attack surface

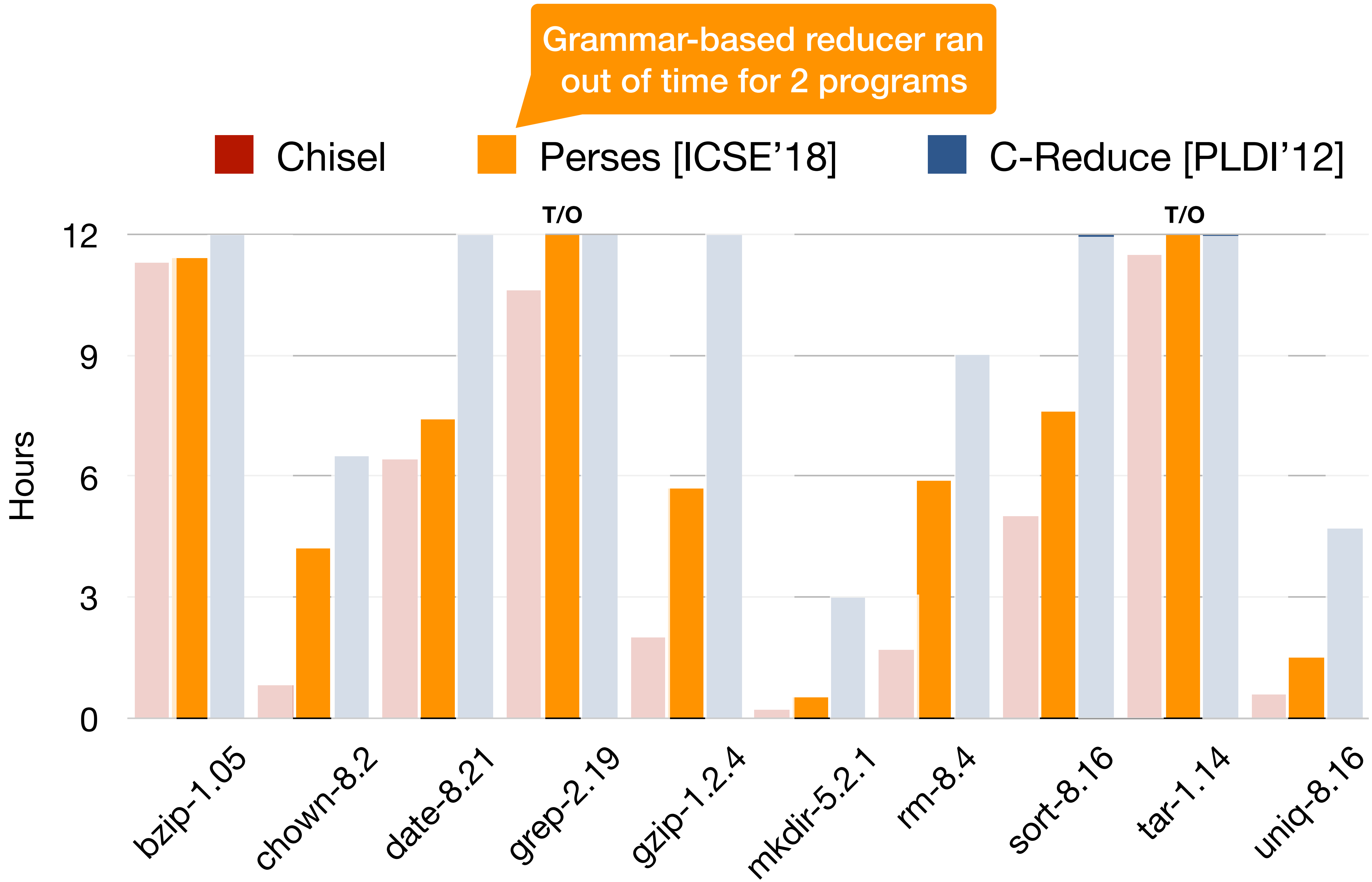
Make it feasible for manual alarm inspection

Reduction Time

Line-based reducer ran out of time for 6 programs

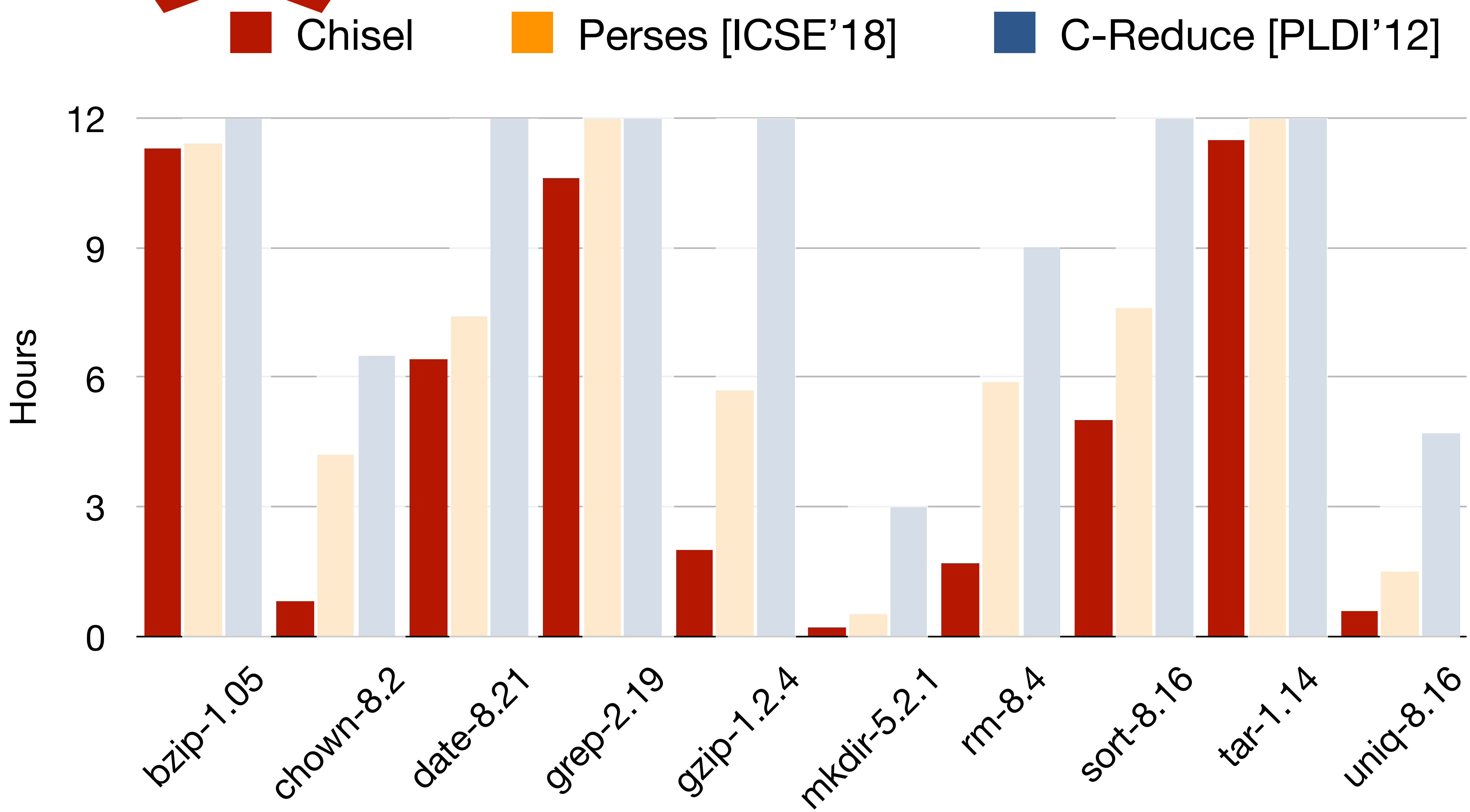


Reduction Time



Reduction Time

7x and 4x faster than C-Reduce and Perses



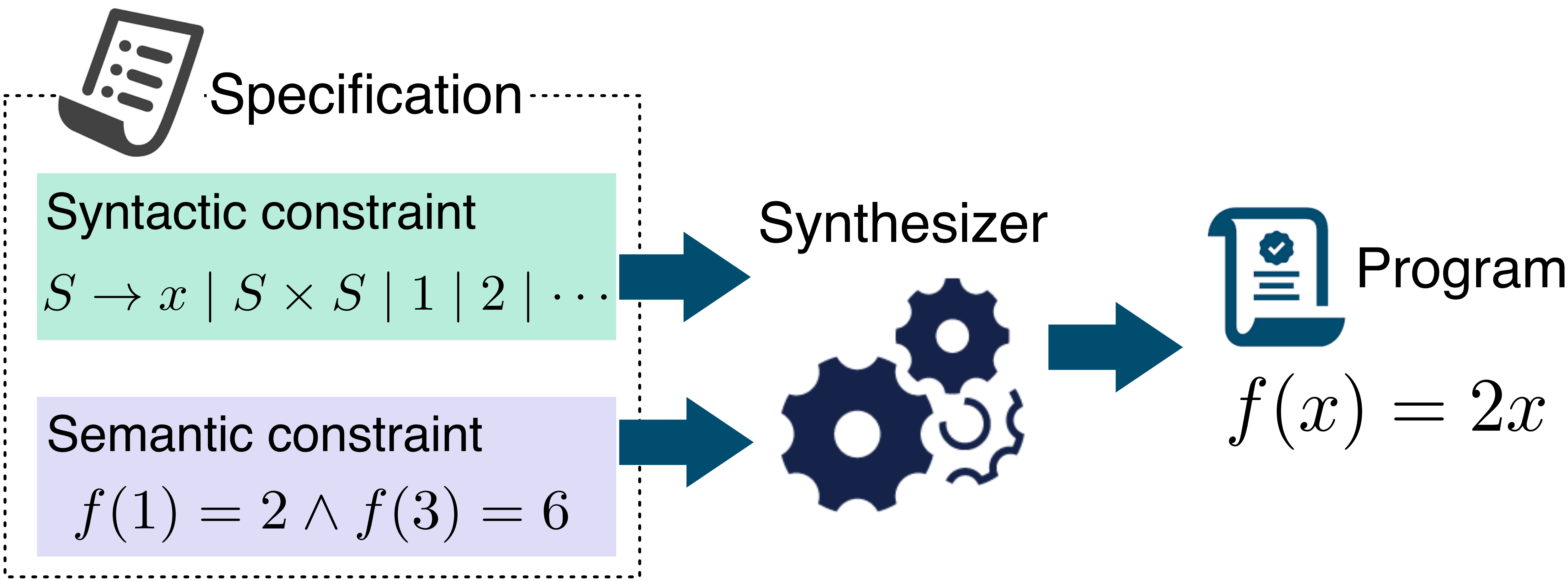
Conclusion

- **Chisel**: automated software debloating system
  - **tractable search** via learning-guided delta debugging
  - **security hardening** by removing undesired features
  - **robustness** via static & dynamic analyses
  - http://chisel.cis.upenn.edu
- **In the paper**,
  - reduction algorithm details
  - learning a debloating policy
  - engineering issues and design choices

Acknowledgment: Total Platform Cyber Protection (TPCP)

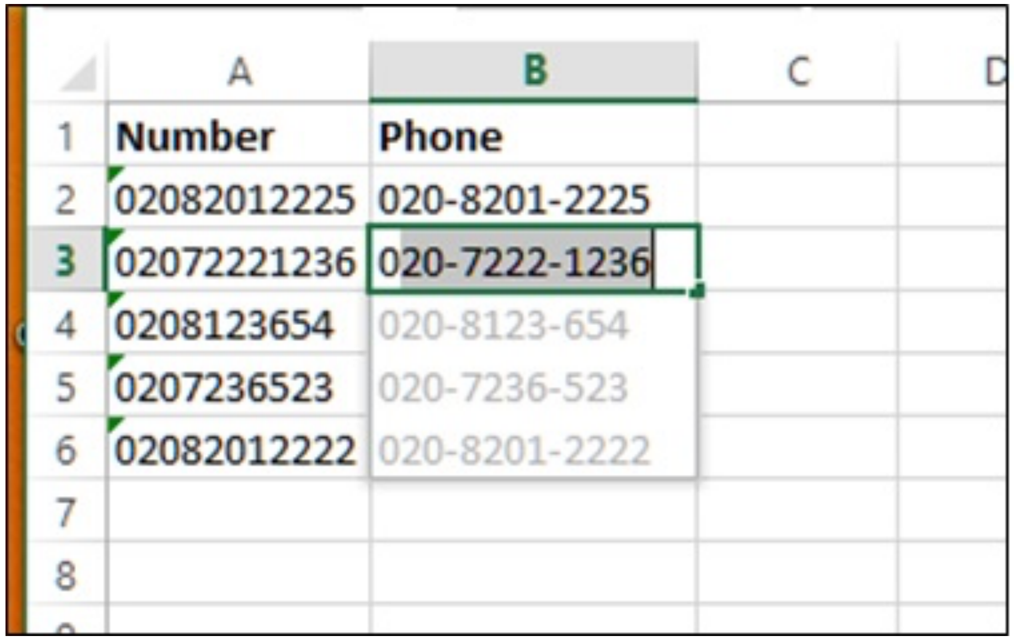
Accelerating Search-Based Program Synthesis Using Learned Probabilistic Models (PLDI'18)

Syntax-Guided Program Synthesis (SyGuS)†

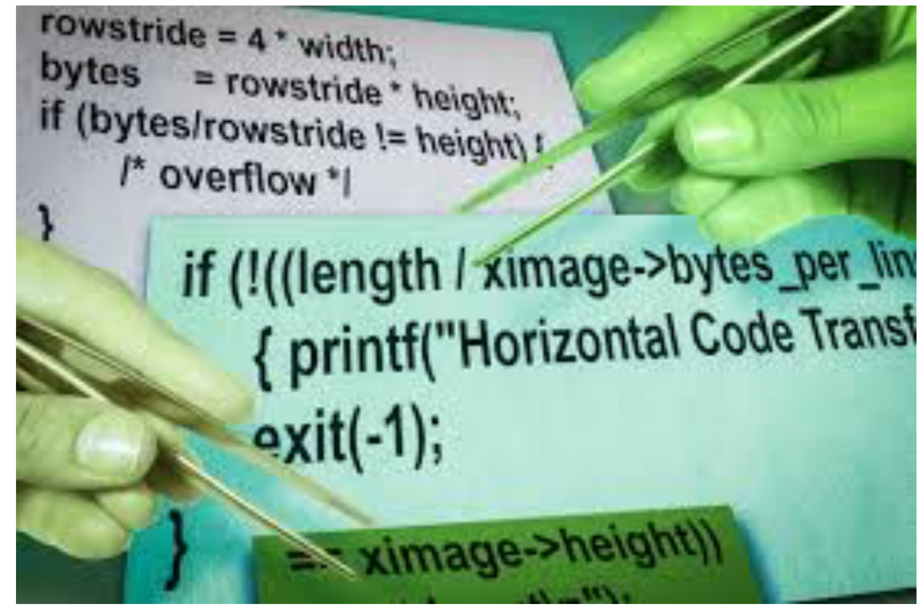


†http://www.sygus.org

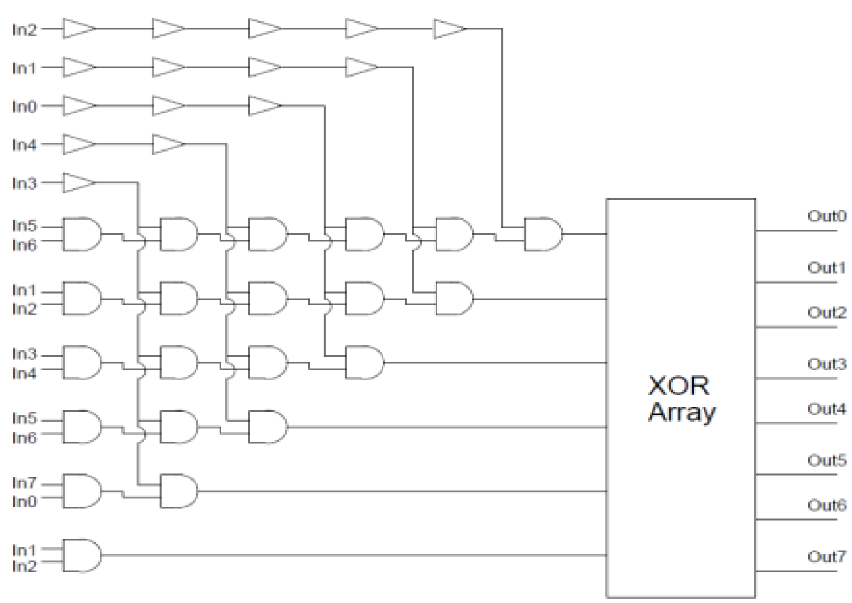
Applications of Program Synthesis



End-user Programming
(e.g., Excel Flash Fill)



Program Repair



Circuit Transformation

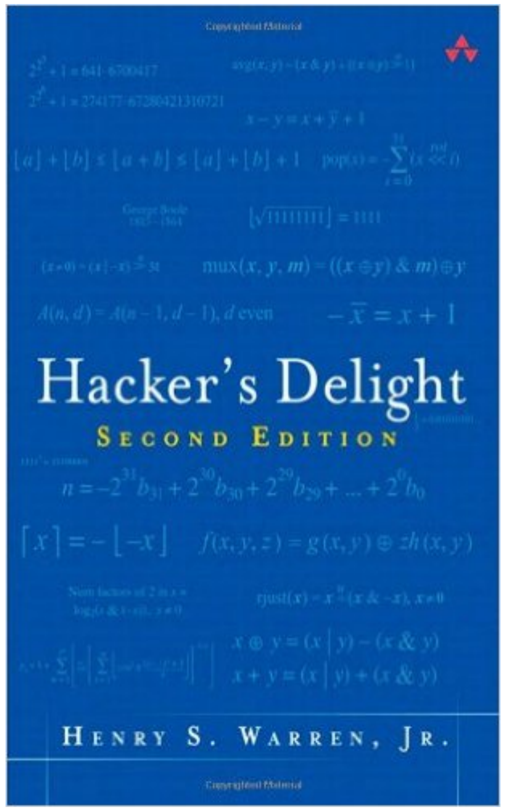
Others

- Invariant generation
- Super-optimization
- Autograding for coding assignment

...

Example

- Find a program P for bit-vector transformation such that
  - P is constructed from standard bit-vector operations (|, &, ~, +, -, <<, >>, 0, 1, …)
  - P is consistent with the following input-output examples
    (00101 → 00100,
    10111 → 10000,
    00111 → 00000)



- Resets rightmost substring of contiguous 1's to 0's.
- Desired solution: x & (1 + (x | (x - 1)))

Existing General-Purpose Strategies

- Enumerative: search with pruning
  - EUSolver: Udupa et al. (PLDI'13, TACAS'17)
- Symbolic: constraint solving
  - CVC4: Reynolds et al. (CAV'15, CAV'18, IJCAR'18)
- Stochastic: probabilistic walk
  - STOKE: Schkufza et al. (ASPLOS'13, ASPLOS'17)

Existing General-Purpose Strategies

- Enumerative: search with pruning
  - EUSolver: Udupa et al. (PLDI'13, TACAS'17)
- Symbolic: constraint solving
  - CVC4: Reynolds et al. (CAV'15, CAV'18, IJCAR'18)
- Stochastic: probabilistic walk
  - STOKE: Schkufza et al. (ASPLOS'13, ASPLOS'17)

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

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

$$Pr(\texttt{??} \rightarrow \texttt{i++} \mid \texttt{for (i = 0; i < 100; ??)}) = 0.80$$
$$Pr(\texttt{??} \rightarrow \texttt{i--} \mid \texttt{for (i = 0; i < 100; ??)}) = 0.01$$

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

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

Exploiting Statistical Regularities

Can we leverage statistical program models
to accelerate program synthesis?

Key Challenges:

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

2. How to learn a good statistical model?

Our Contributions

- A general approach to accelerate CEGIS-based program synthesis

  – by using a probabilistic model to guide the search towards likely programs

  – supports a wide range of models (e.g., n-gram, PCFG, PHOG, neural nets, …)

- Transfer learning-based method to mitigate overfitting

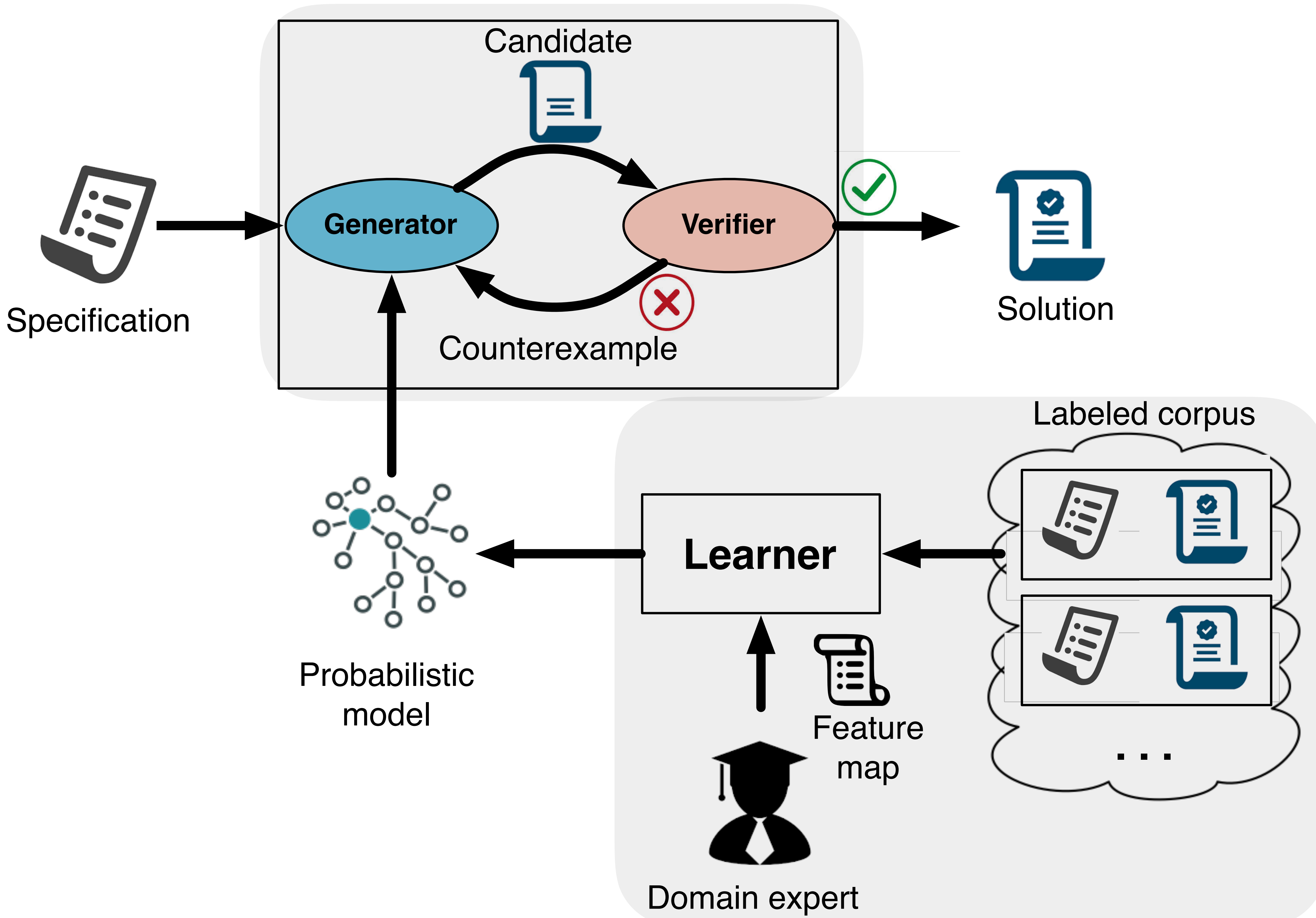
- Tool (Euphony) and evaluation on widely applicable domains

  https://github.com/wslee/euphony

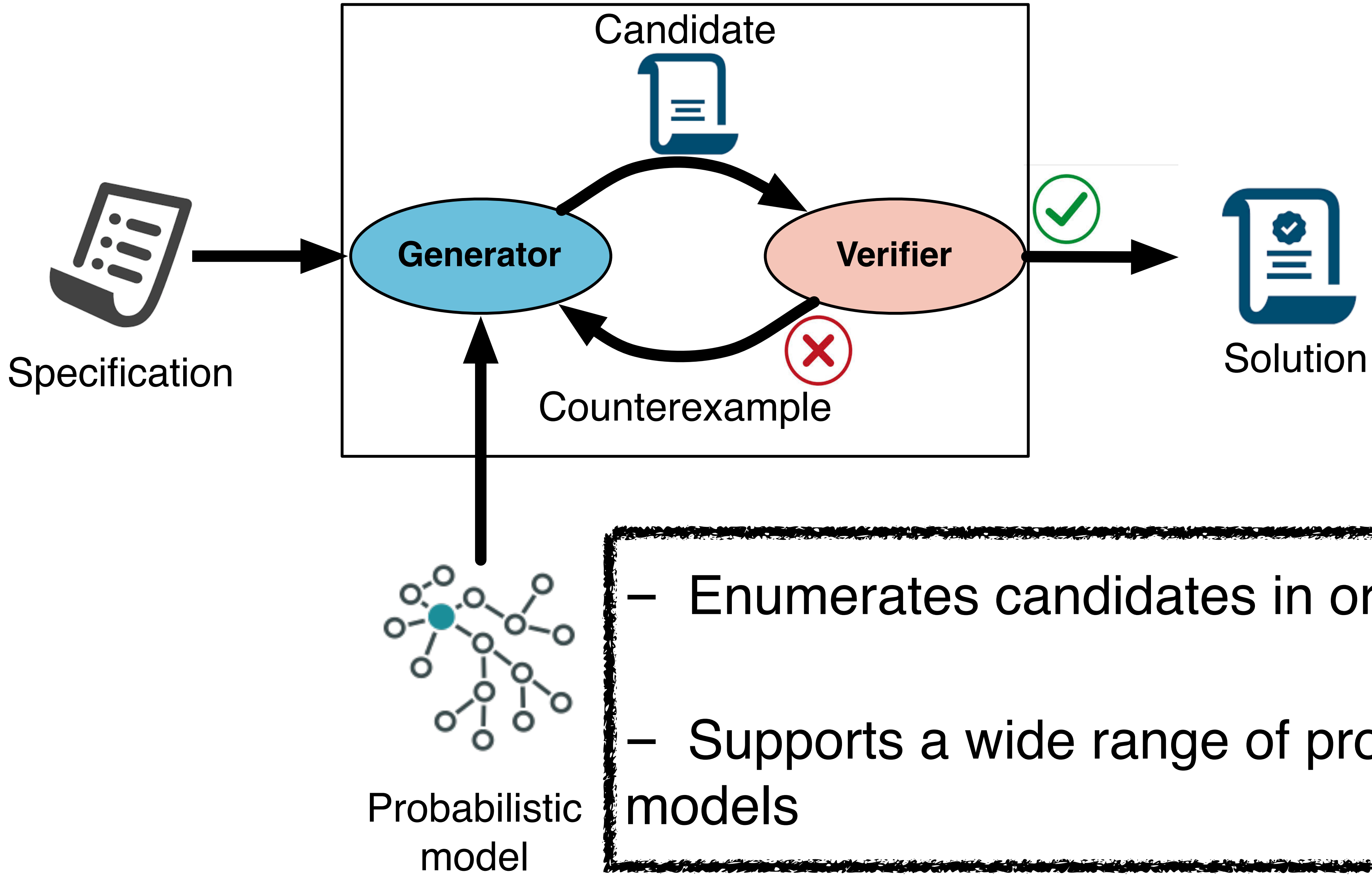
Talk Outline

- Overall Architecture
- Illustrative Example
- Empirical Evaluation

Overall Architecture



CEGIS with Guided Search



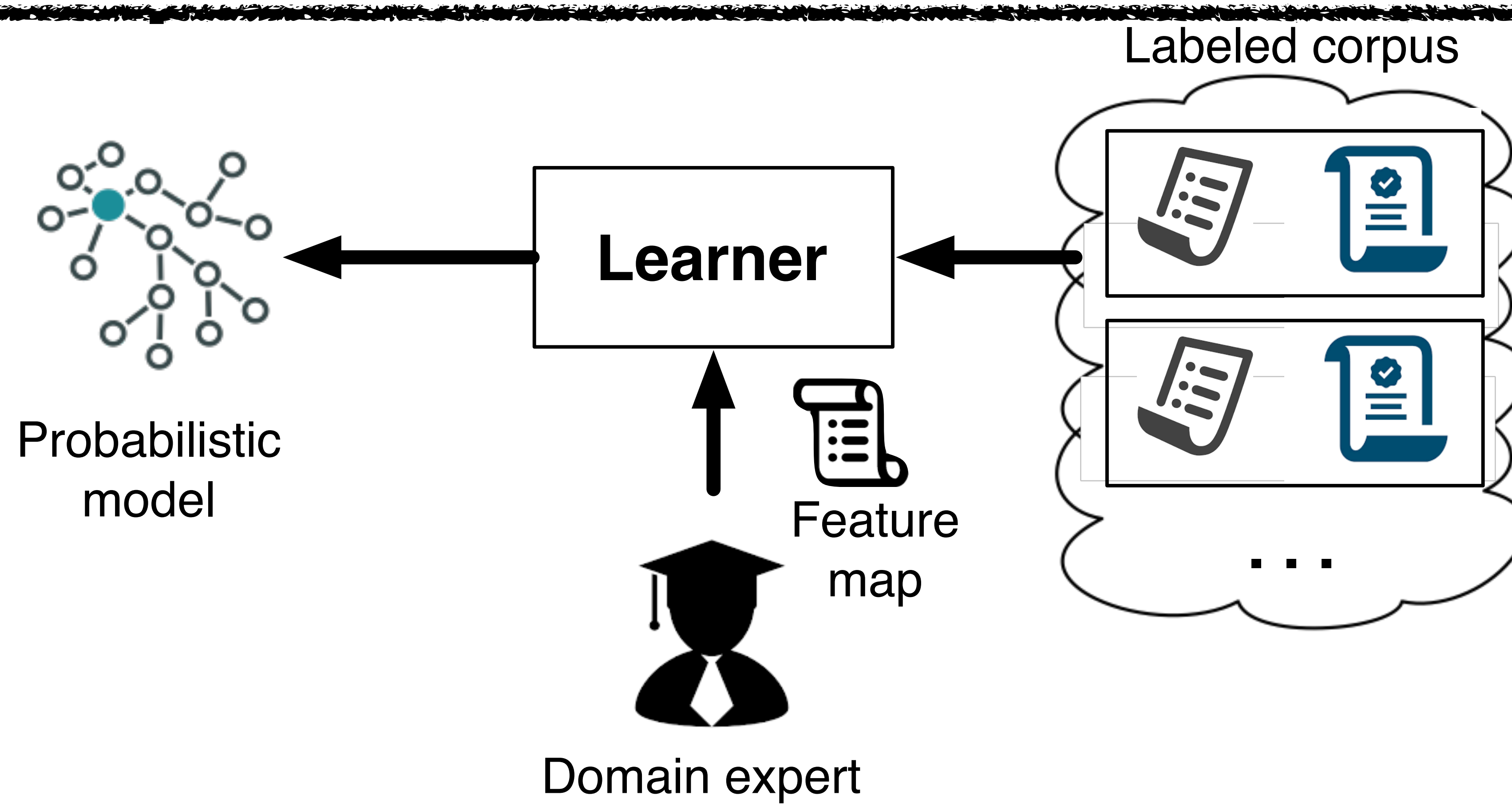
– Enumerates candidates in order of likelihood

– Supports a wide range of probabilistic models

Transfer Learning

– Problem: overfitting

– Our solution: generalize to unseen programs better using a feature map designed by domain expert



Talk Outline

- Overall Architecture
- Illustrative Example
- Empirical Evaluation

Example

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

- Specification

Syntactic specification:

String concatenation

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

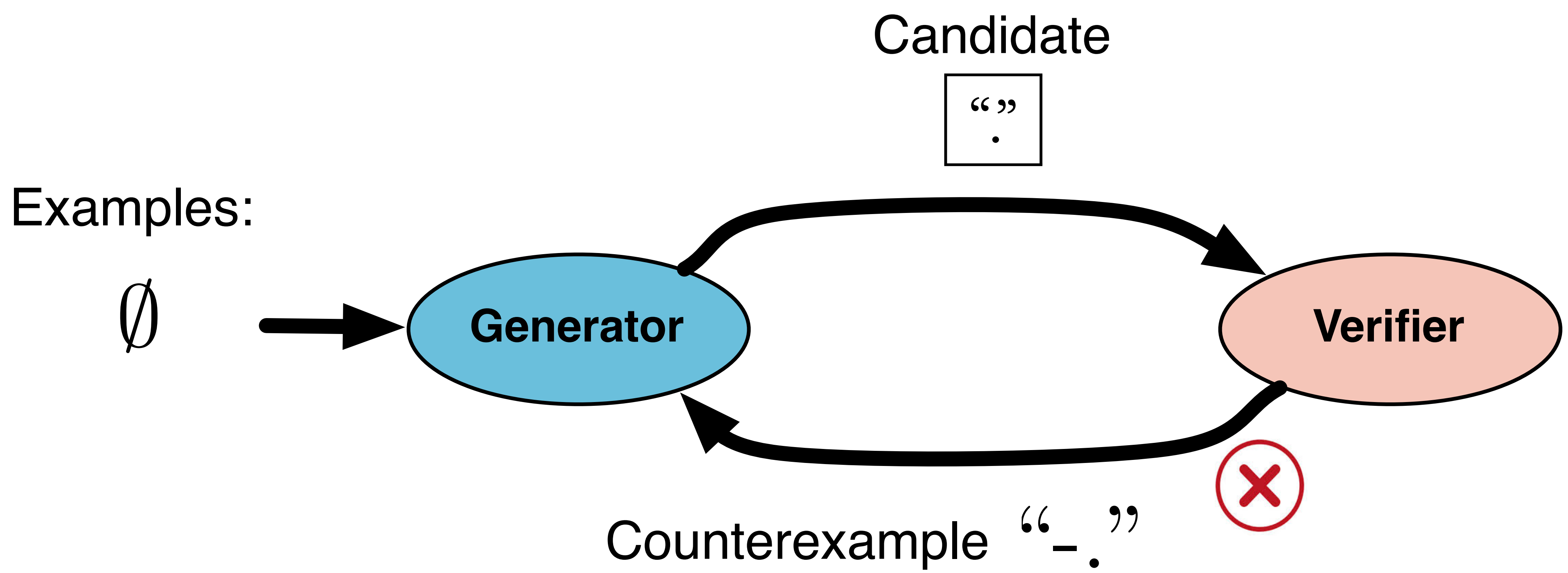
Rep(s, t1, t2): t1 in **s** is replaced by **t2**

Semantic specification:

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

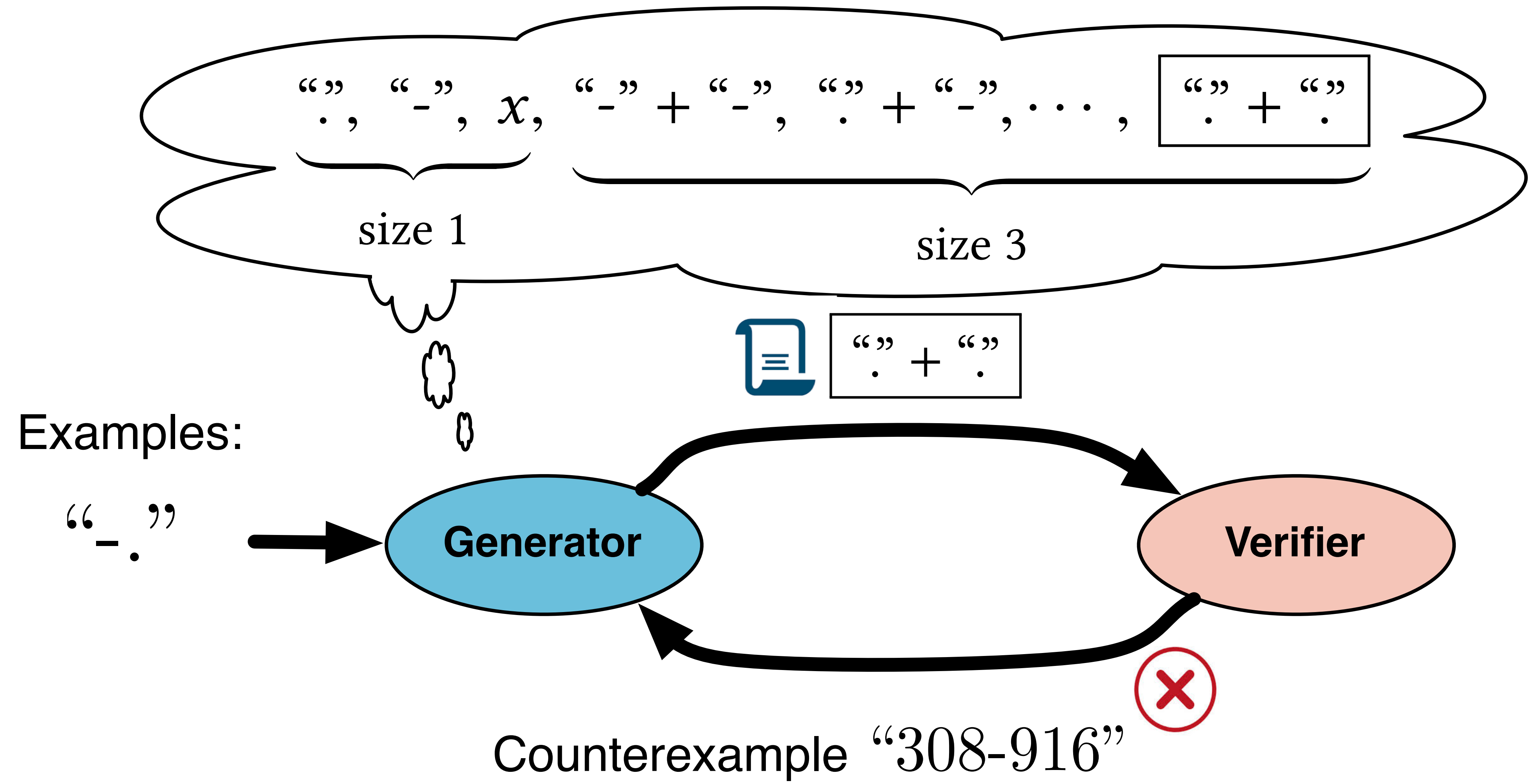
- Solution $\boxed{\mathsf{Rep}(x, \text{"-"}, \text{"."}).}$

Enumerative Search: Unguided



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

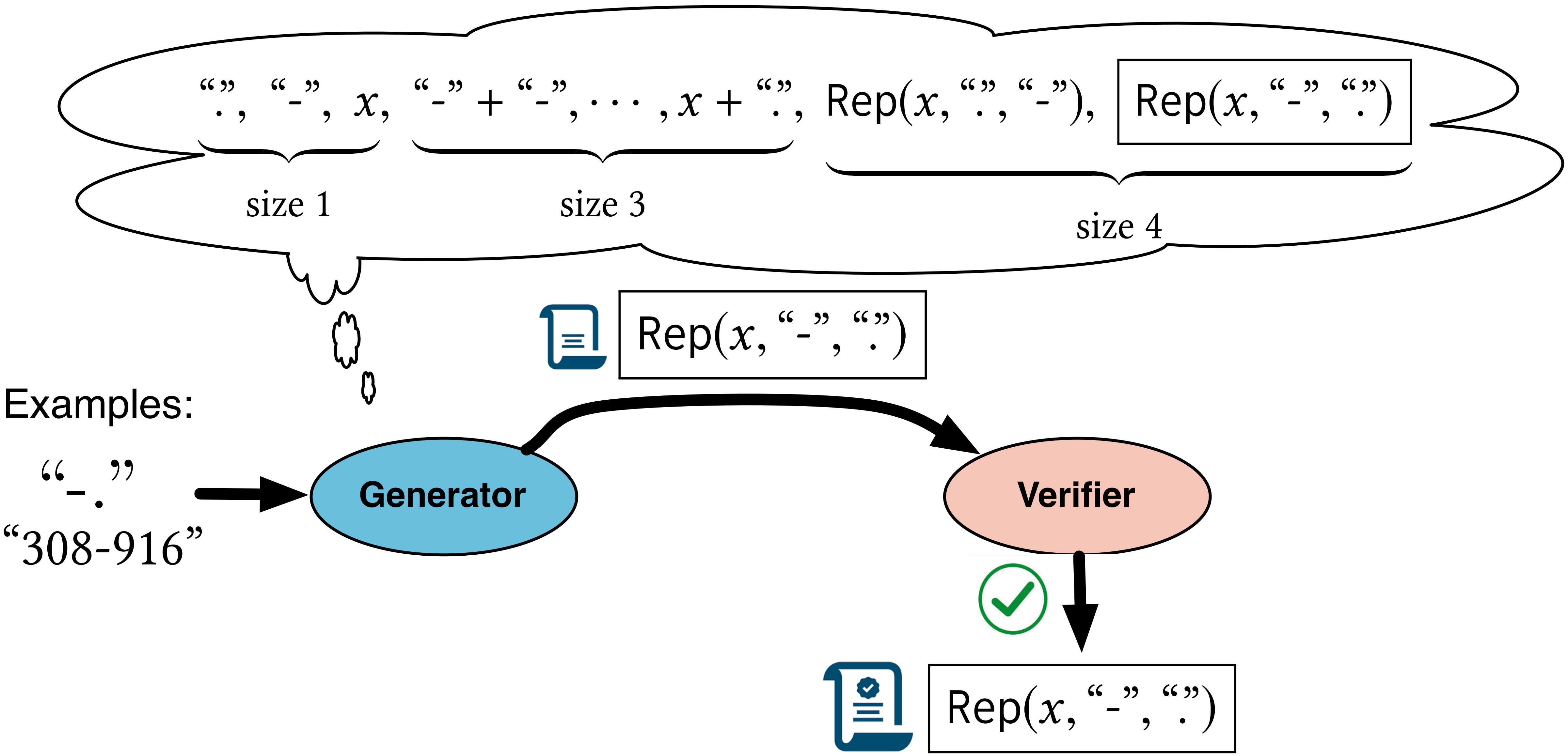
Enumerative Search: Unguided



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

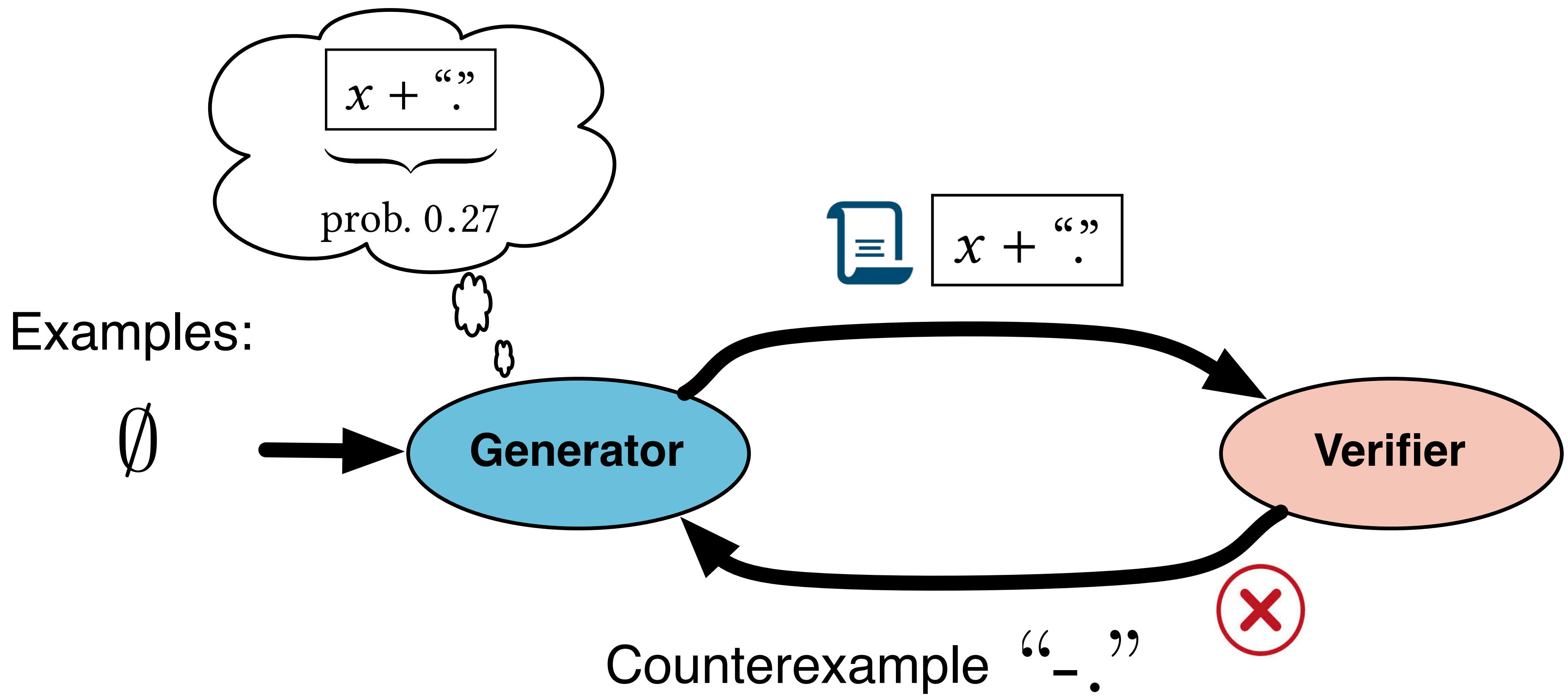
Pruning optimization: “-” + “.” is not explored.

Enumerative Search: Unguided



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

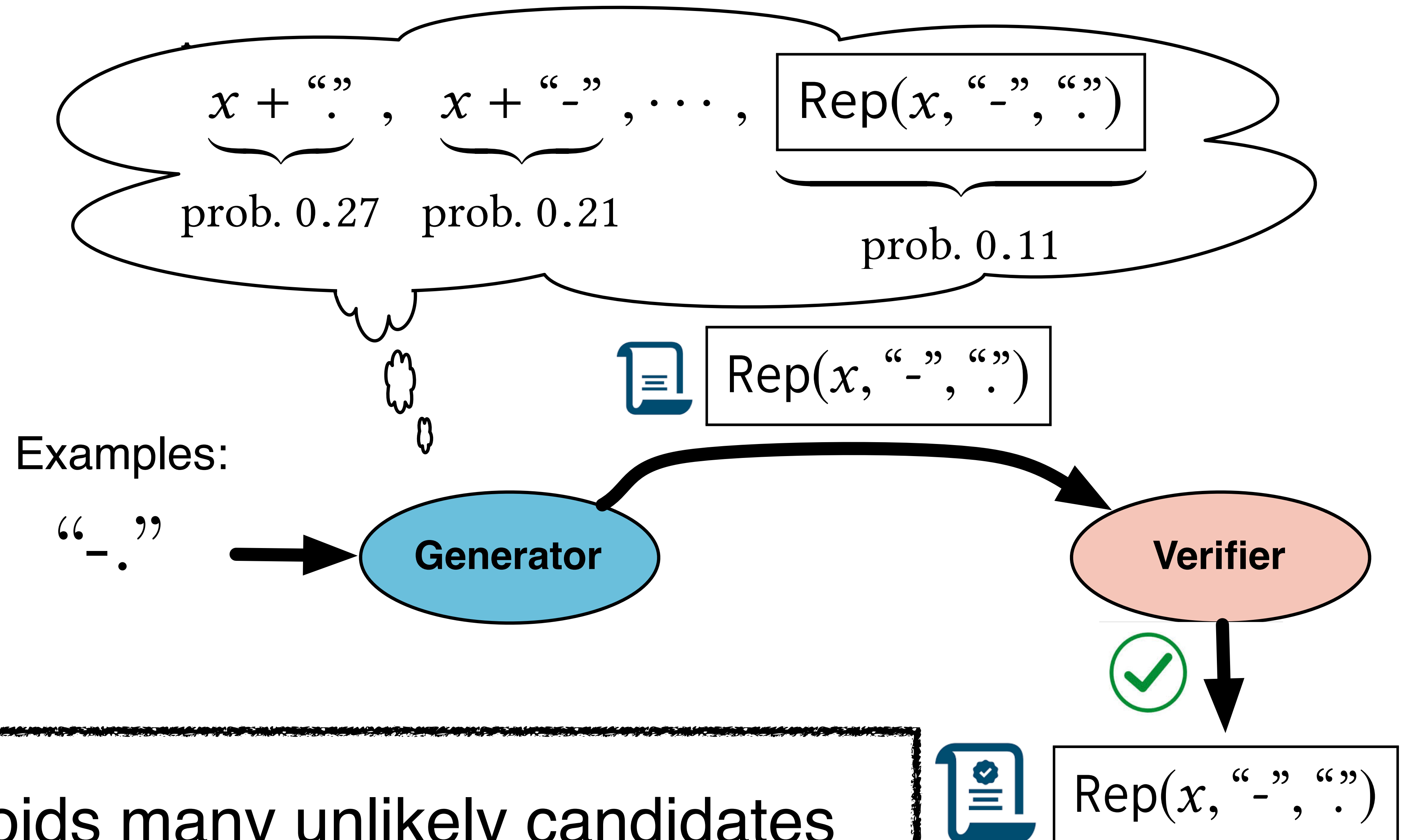
Enumerative Search: Guided



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

Enumerates in order of likelihood instead of size

Enumerative Search: Guided



- Avoids many unlikely candidates
- Preserves the pruning optimization

A Uniform Interface to Statistical Program Models

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

$$
\begin{gathered}
Pr(S \rightarrow \text{``.''} \mid \texttt{Rep}(x, \text{``-''}, S)) = 0.72 \\
Pr(S \rightarrow \text{``-''} \mid \texttt{Rep}(x, \text{``-''}, S)) = 0.001 \\
\cdots
\end{gathered}
$$

A Uniform Interface to Statistical Program Models

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

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

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

$$Pr(S \rightarrow S + S \mid S)$$

$$\times \; Pr(S \rightarrow x \mid S + S)$$

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

Examples of Models

- Probabilistic context-free grammar (PCFG)

$$A \to \beta$$

$$\begin{array}{lllr} & & & P \\ S & \to & \text{“.”} & 0.2 \\ S & \to & \text{“\_”} & 0.2 \\ S & \to & x & 0.1 \\ S & \to & S + S & 0.1 \\ S & \to & \texttt{Rep}(S, S, S) & 0.4 \end{array}$$

Examples of Models

- <u>Probabilistic Higher-order Grammar</u> (PHOG) (the model we use)

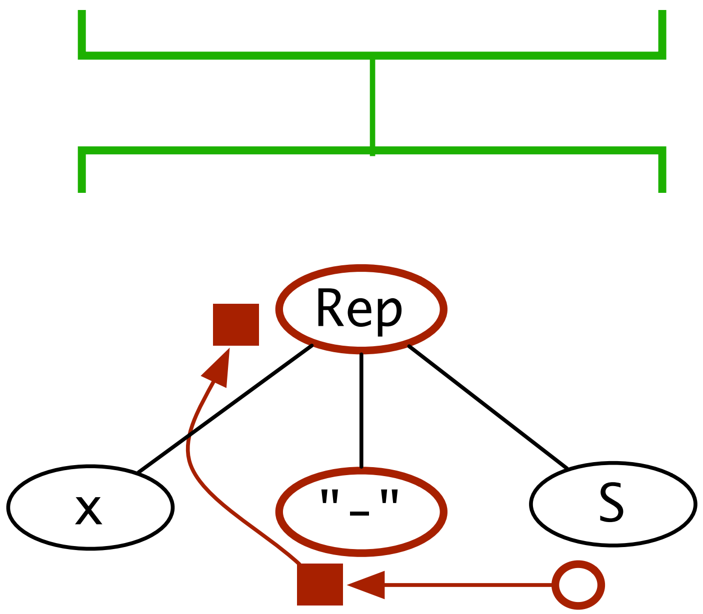
$$A[\textcolor{darkred}{context}] \rightarrow \beta$$

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

$$\ldots$$

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

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

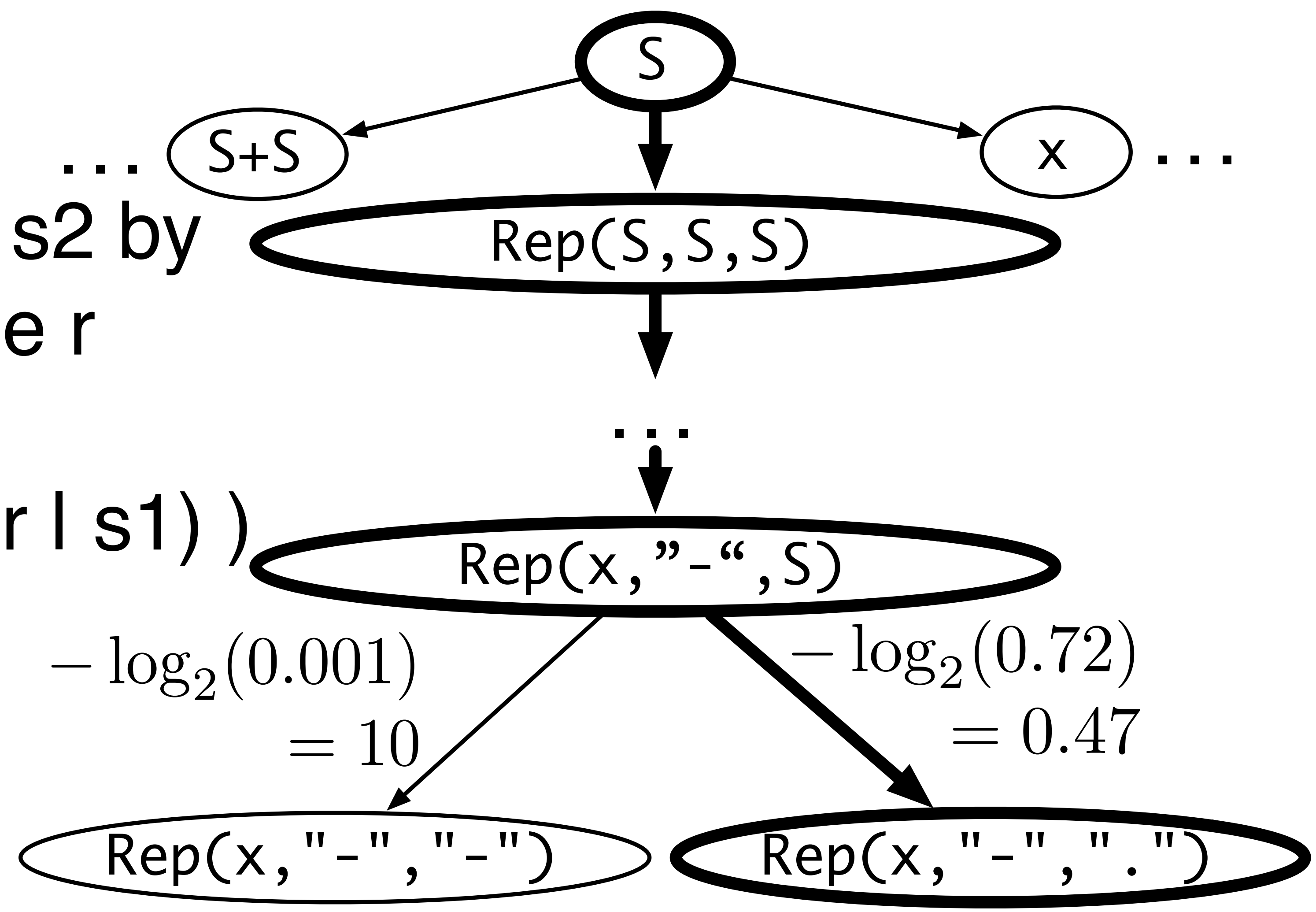


- Others: n-gram, a neural network-based model, log-bilinear model …

Guided Enumeration via Path Finding

Given a model, we construct a directed graph.

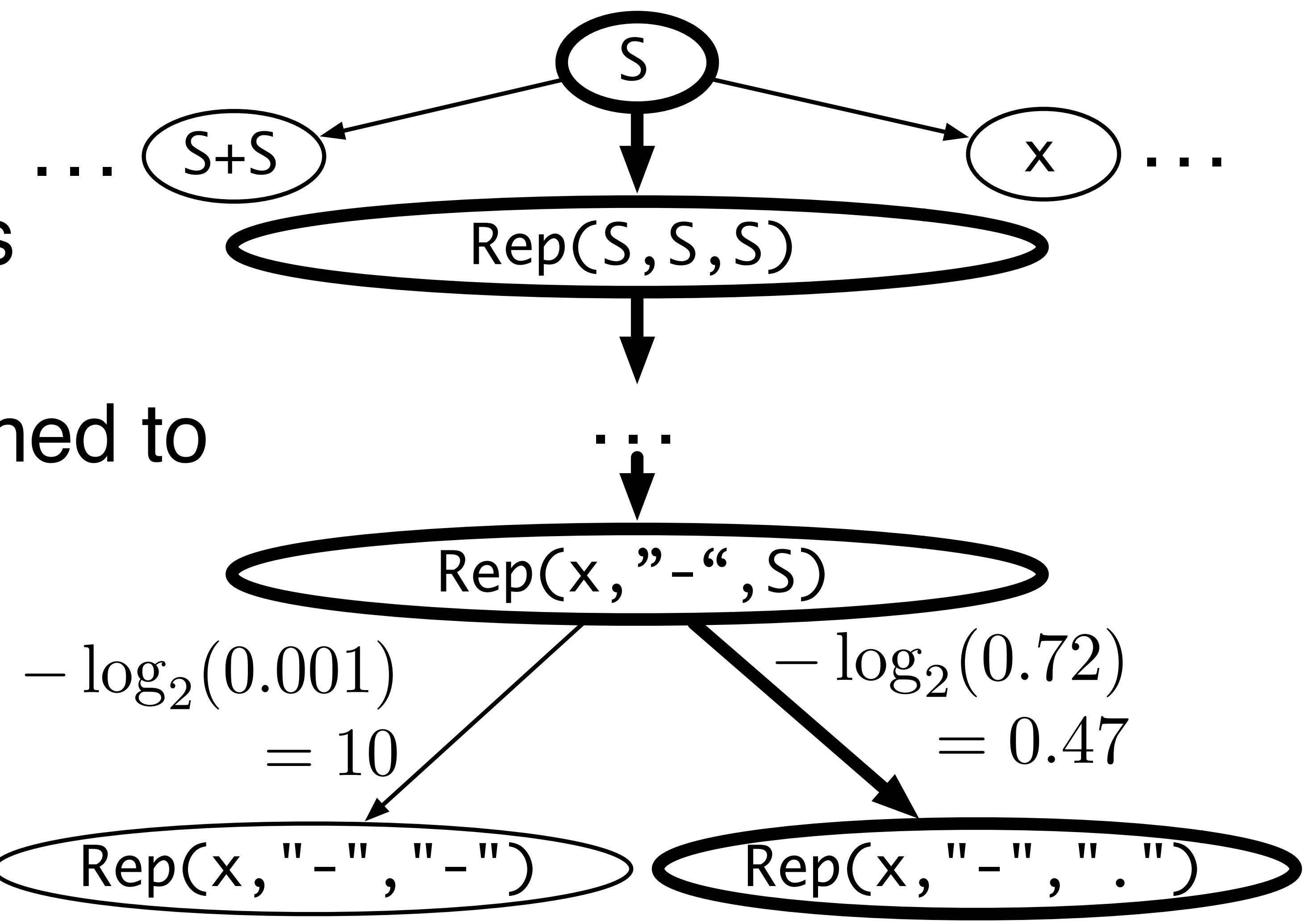
- Nodes: sentential forms
- s1 $\xrightarrow{r}$ s2: s1 expands to s2 by applying a production rule r
- w(s1 $\xrightarrow{r}$ s2) = − log (Pr (r | s1))



Guided Enumeration via Path Finding

Idea: solving a shortest pathfinding problem via A* search

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



Problem of Overfitting

- Suppose we are given a similar synthesis problem with the following semantic specification:

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

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

  (the inverse of the previous solution $\texttt{Rep}(x, \text{“-”}, \text{“.”})$.

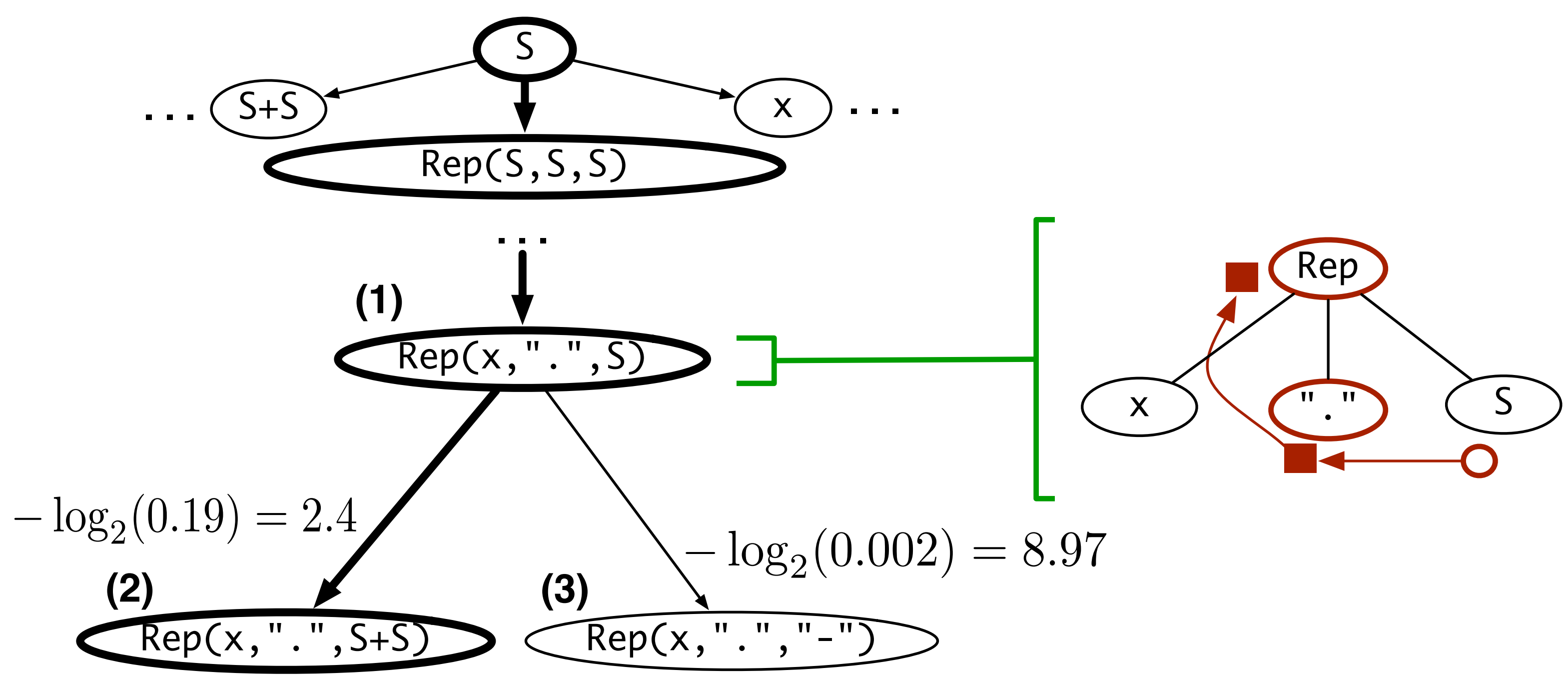
Problem of Overfitting

- Suppose we want to complete $\texttt{Rep}(x, \text{“.”}, S)$ (node (1))
- Search is not guided toward the solution (node (2) is chosen instead of (3)).

$A[context] \rightarrow \beta$

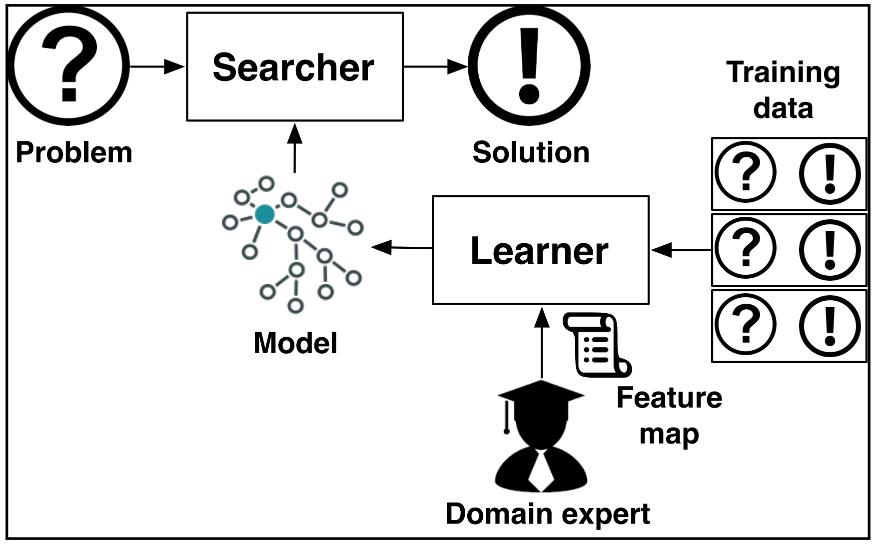
			P
$S[\text{“-”}, \texttt{Rep}]$	$\rightarrow$	“.”	**0.72**
$S[\text{“-”}, \texttt{Rep}]$	$\rightarrow$	“-”	**0.001**
$S[\text{“-”}, \texttt{Rep}]$	$\rightarrow$	x	0.12
$S[\text{“-”}, \texttt{Rep}]$	$\rightarrow$	$S + S$	0.02
	$\ldots$		

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



- PHOG sticks to syntactic information, which may lead to overfitting.

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 using Common Features

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

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

A constant string appearing in the **inputs**

A constant string appearing in the **outputs**

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

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

A constant string appearing in the **inputs**

A constant string appearing in the **outputs**

Transfer Learning using Common Features

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

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

A constant string appearing in the **inputs**

A constant string appearing in the **outputs**

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

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

Types of Constant Strings

- $const_{IO}$ represents the set of substrings of all the strings in $I \cap O$
- $const_{I}$ represents the set of substrings of all the strings in I
- $const_{O}$ represents the set of substrings of all the strings in O
- $const_{\perp}$ represents all the remaining strings.

Pivot PHOG

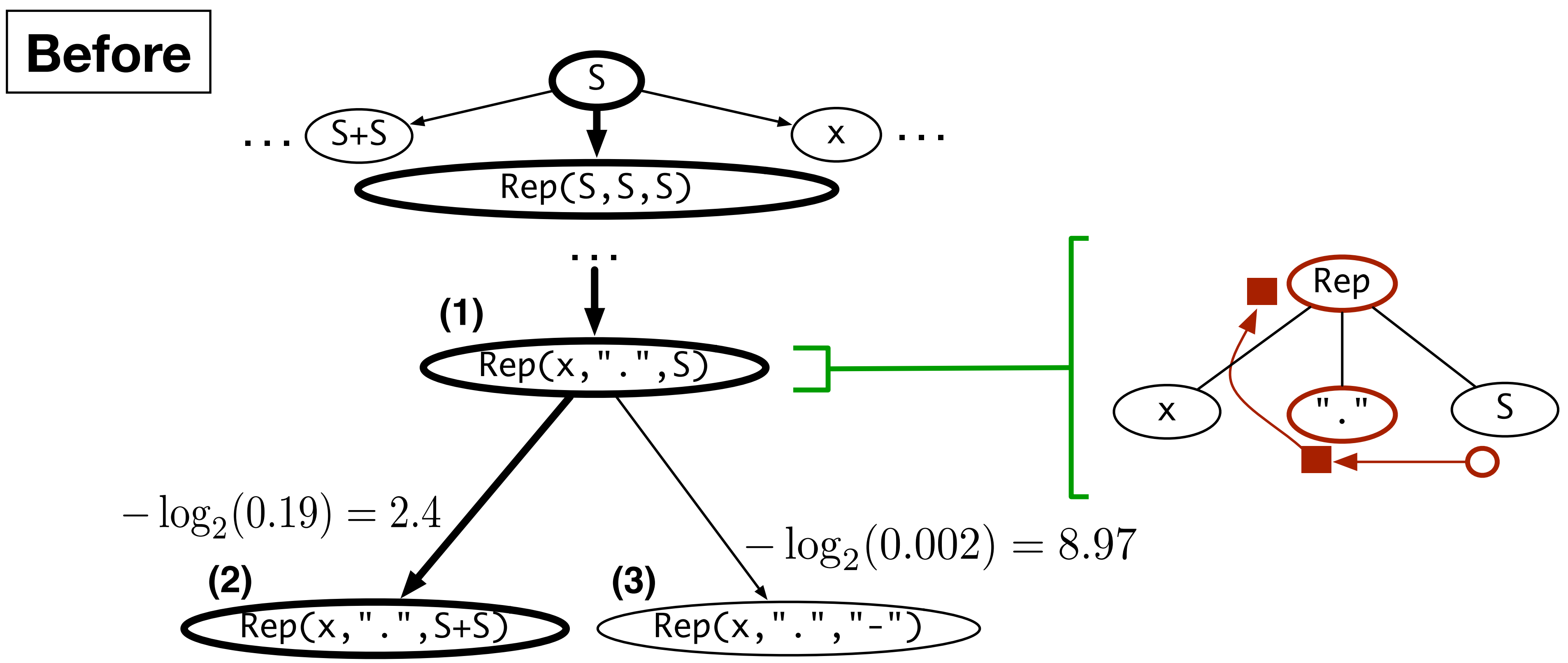
$$
\begin{array}{rcl}
S & \rightarrow & x \mid S + S \\
& \mid & \texttt{Rep}(S, S, S) \\
& \mid & \texttt{const}_{IO} \mid \texttt{const}_{I} \\
& \mid & \texttt{const}_{O} \mid \texttt{const}_{\perp}
\end{array}
$$

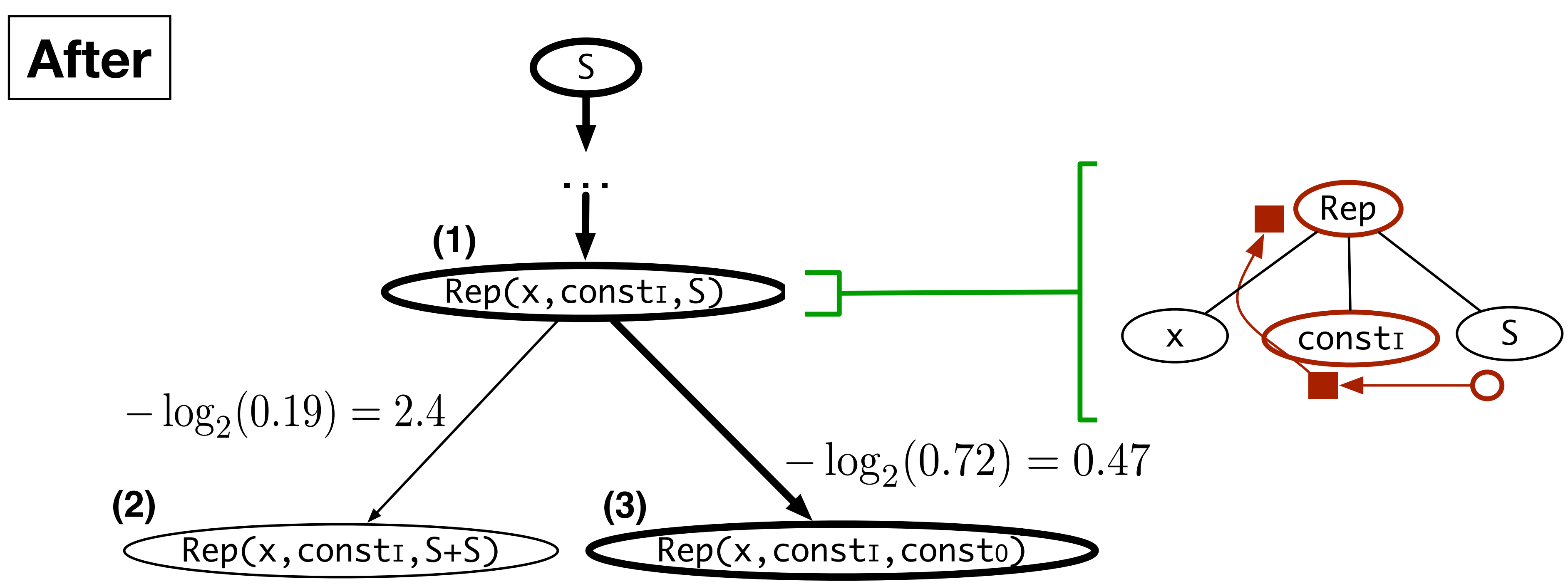
(a) A pivot grammar for string manipulation tasks

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

(b) A pivot PHOG learned using the pivot grammar

Search with the Pivot PHOG





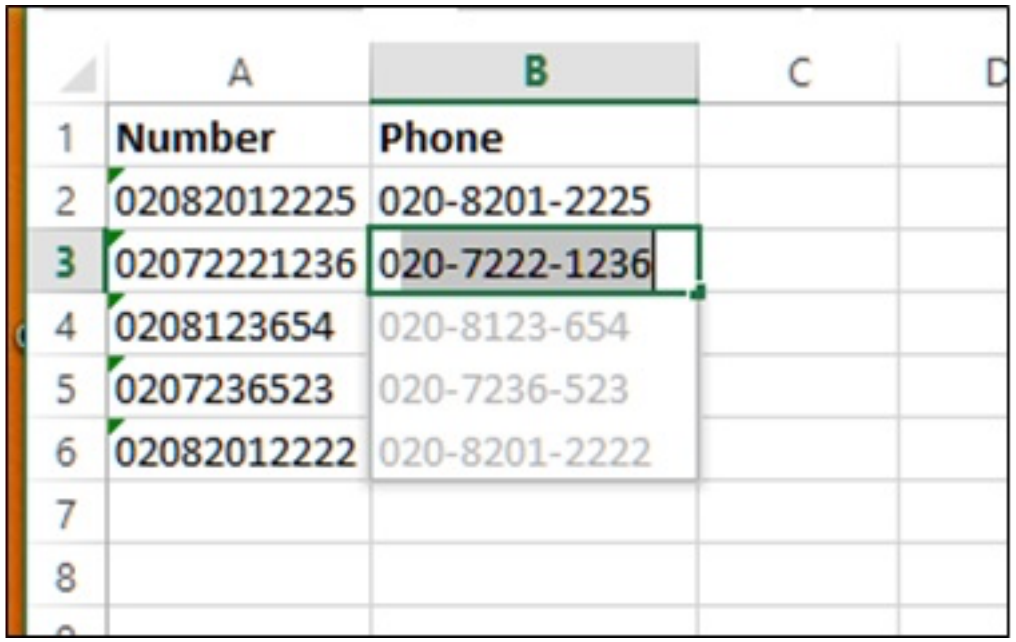
Talk Outline

- Overall Architecture
- Illustrative Example
- Empirical Evaluation

Evaluation Setup

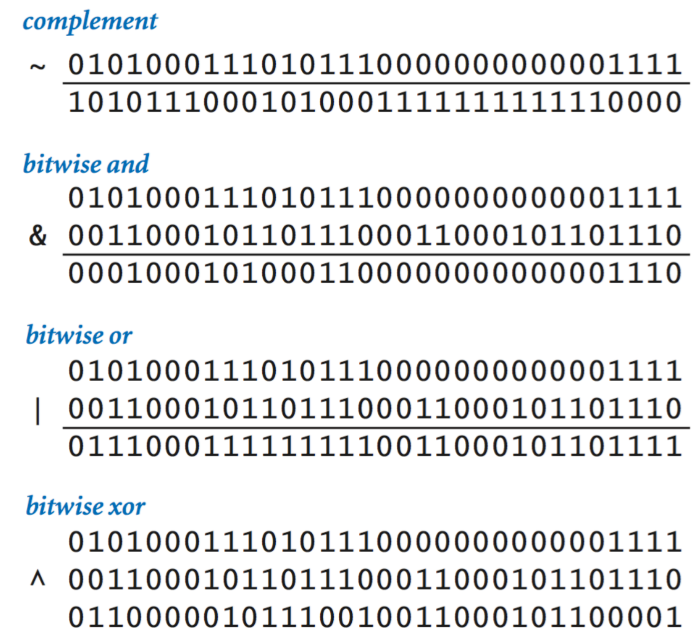
- Benchmarks:
  - 1,167 problems from 2017 SyGuS competition and online forums
- Comparison to two baselines:
  - EUSolver (general-purpose): winner of 2017 SyGuS competition
  - FlashFill (domain-specific): string processing in spreadsheets

Benchmarks

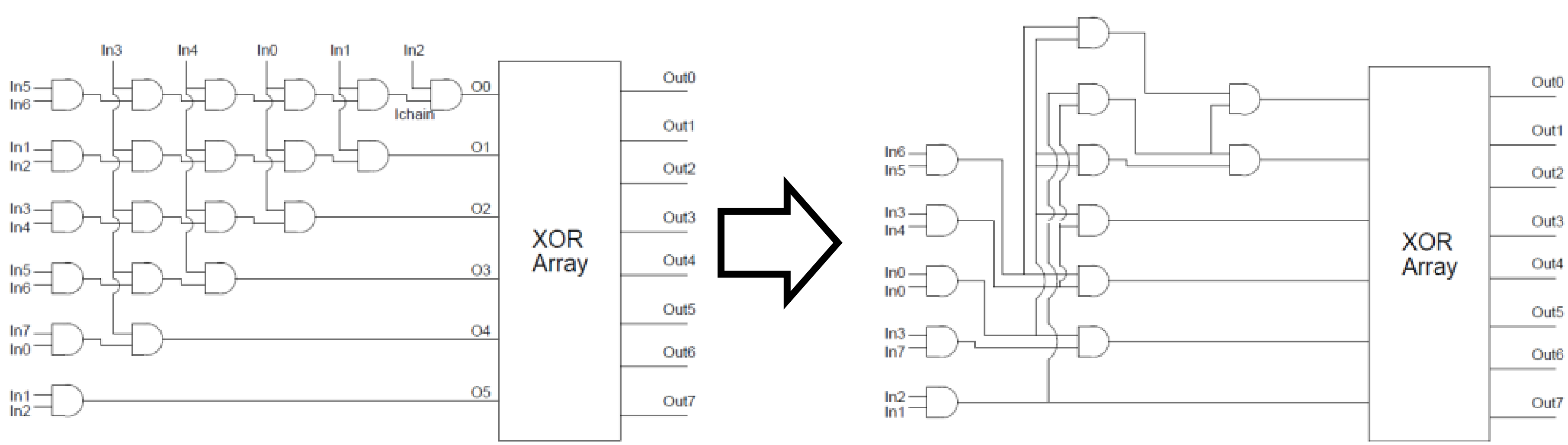


	A	B	C	D
1	Number	Phone		
2	02082012225	020-8201-2225		
3	02072221236	020-7222-1236		
4	0208123654	020-8123-654		
5	0207236523	020-7236-523		
6	02082012222	020-8201-2222		
7				
8				

STRING: End-user Programming
205 problems



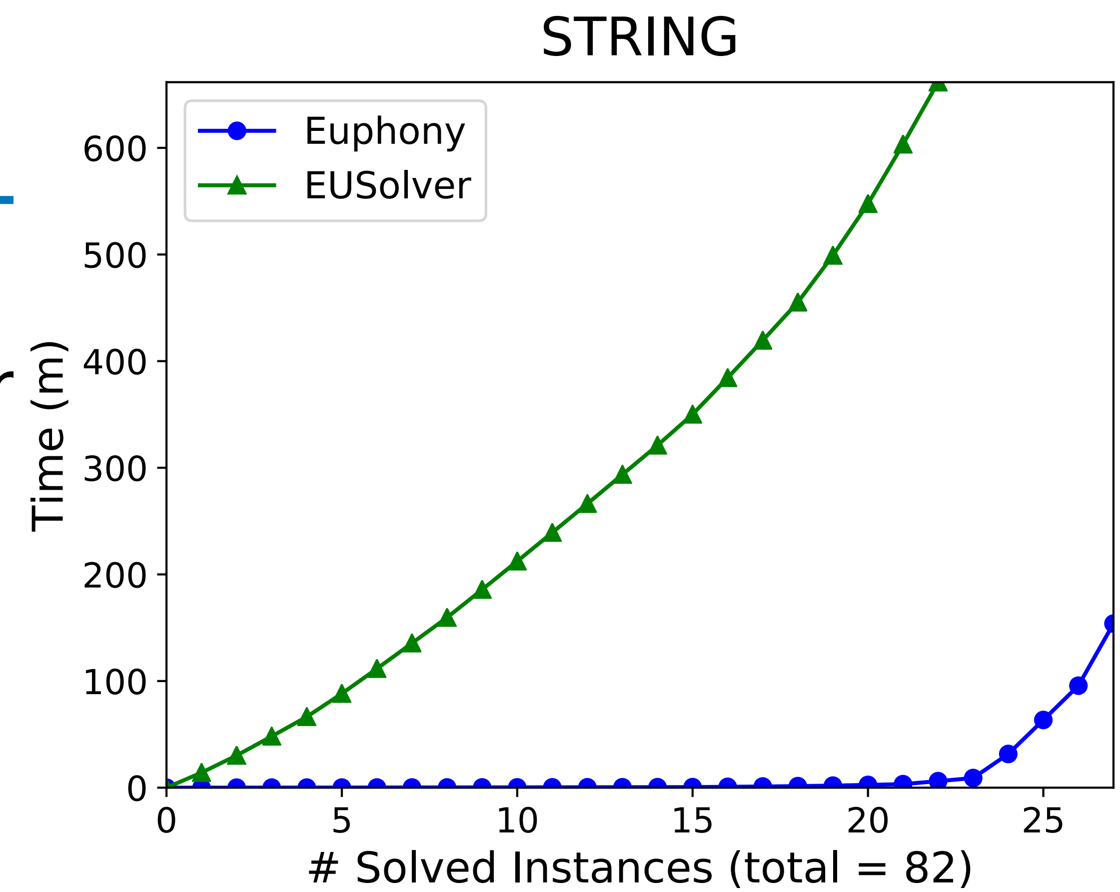
BITVEC: Efficient low-level algorithm
750 problems

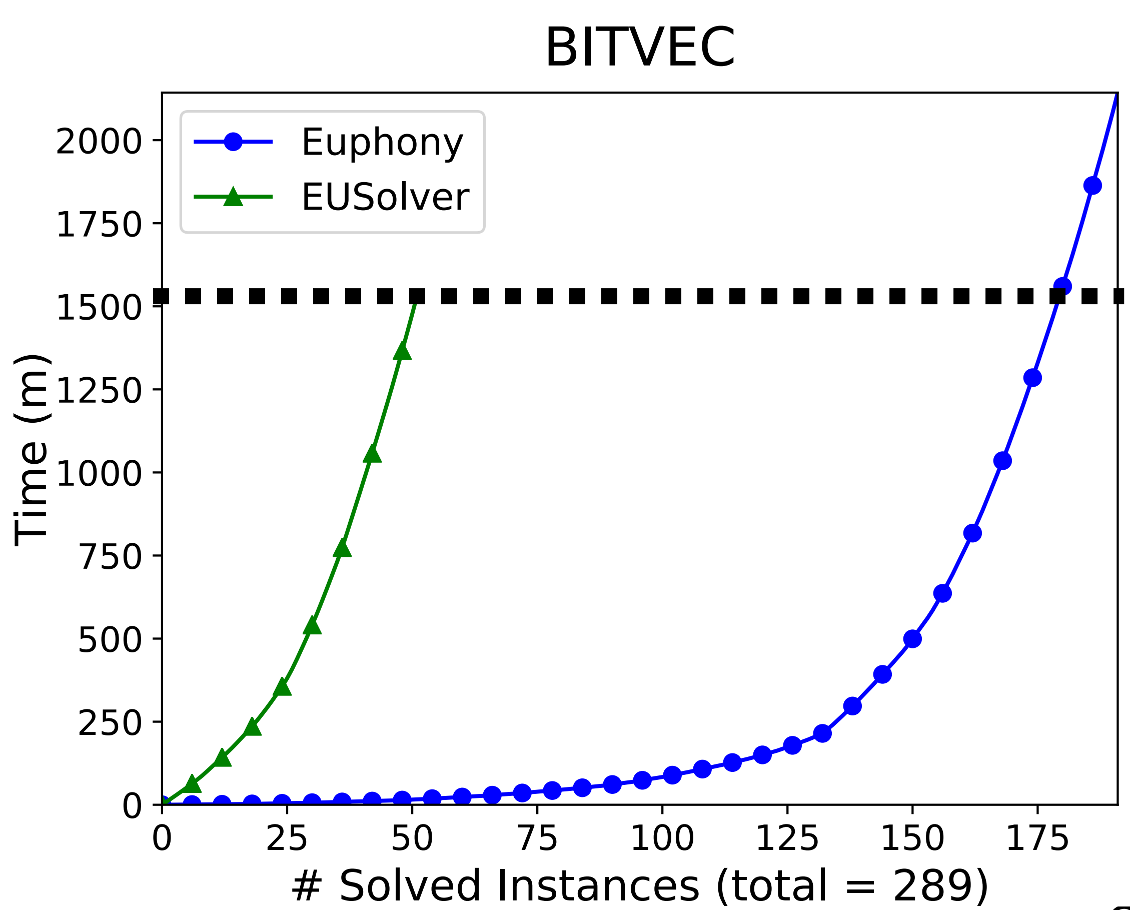


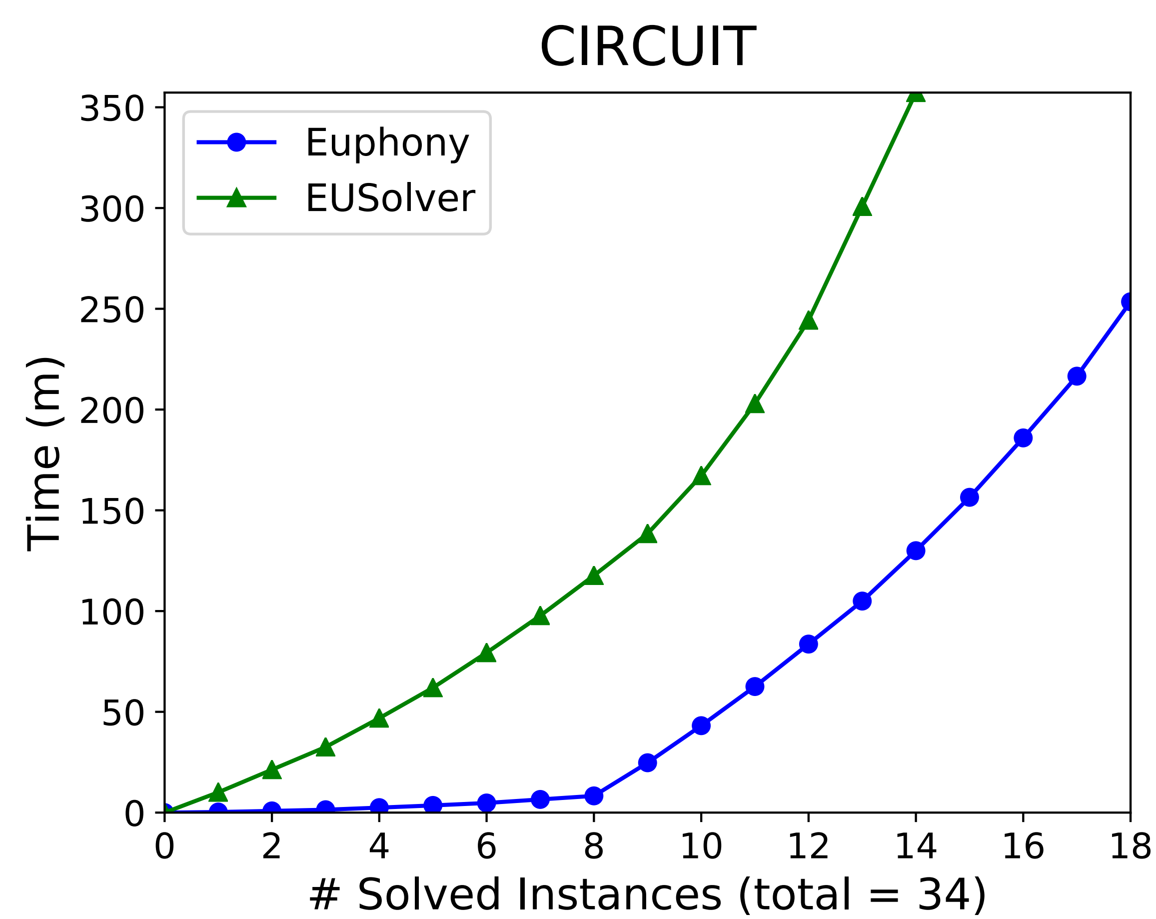
CIRCUIT: Attack-resistant crypto circuits
212 problems

Comparison with EUSolver

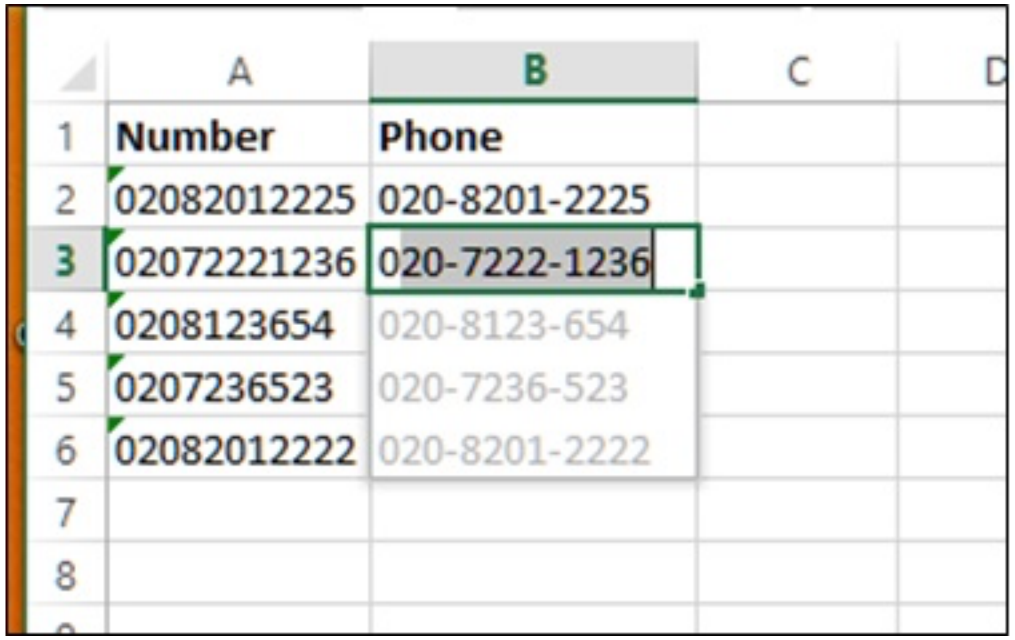
- Training: 762 solved by EUSolver in 10 mir
- Testing: 405 (timeout: 1 hour)
- # solved: Euphony 236, EUSolver 87



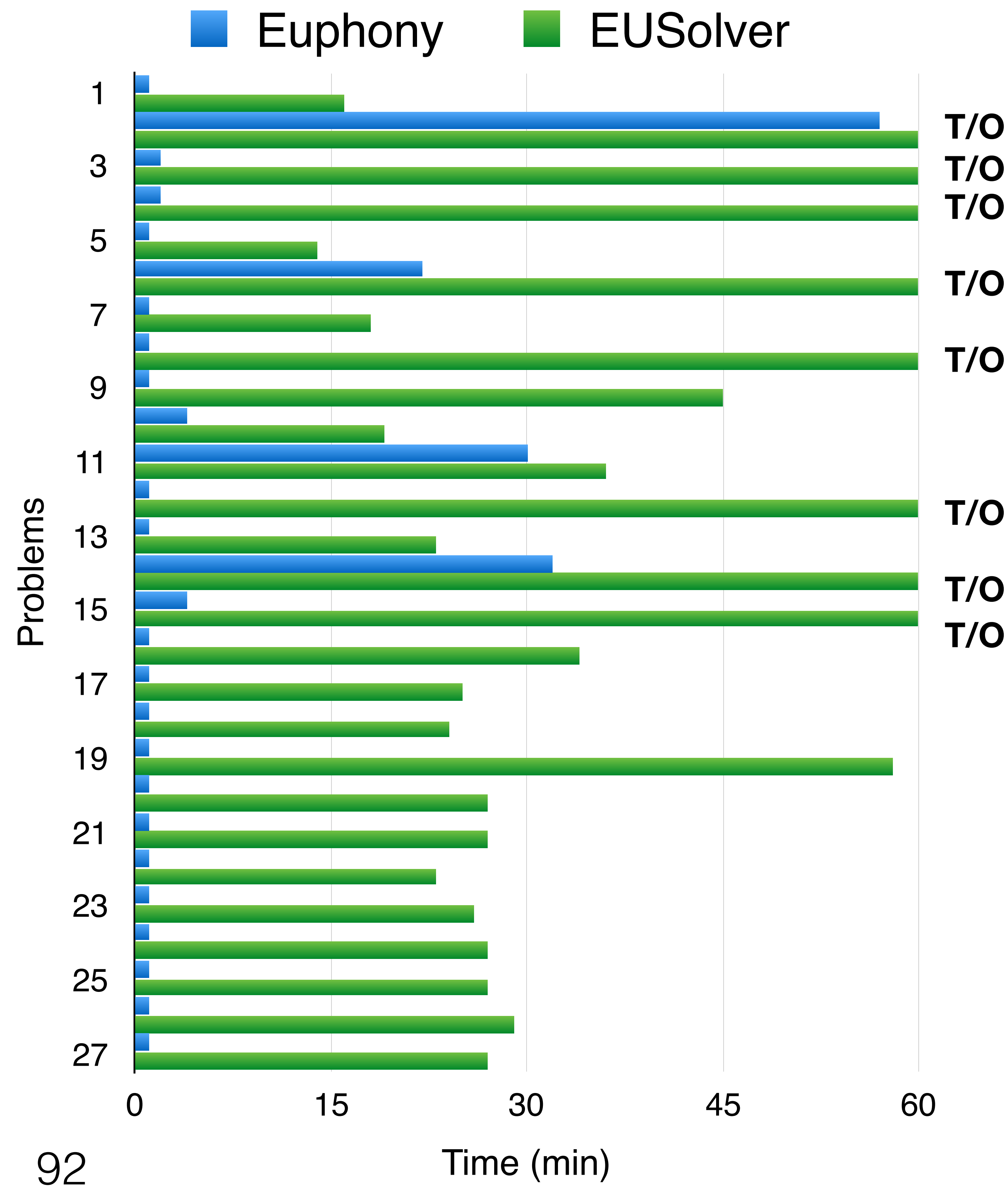




Result for STRING benchmarks



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

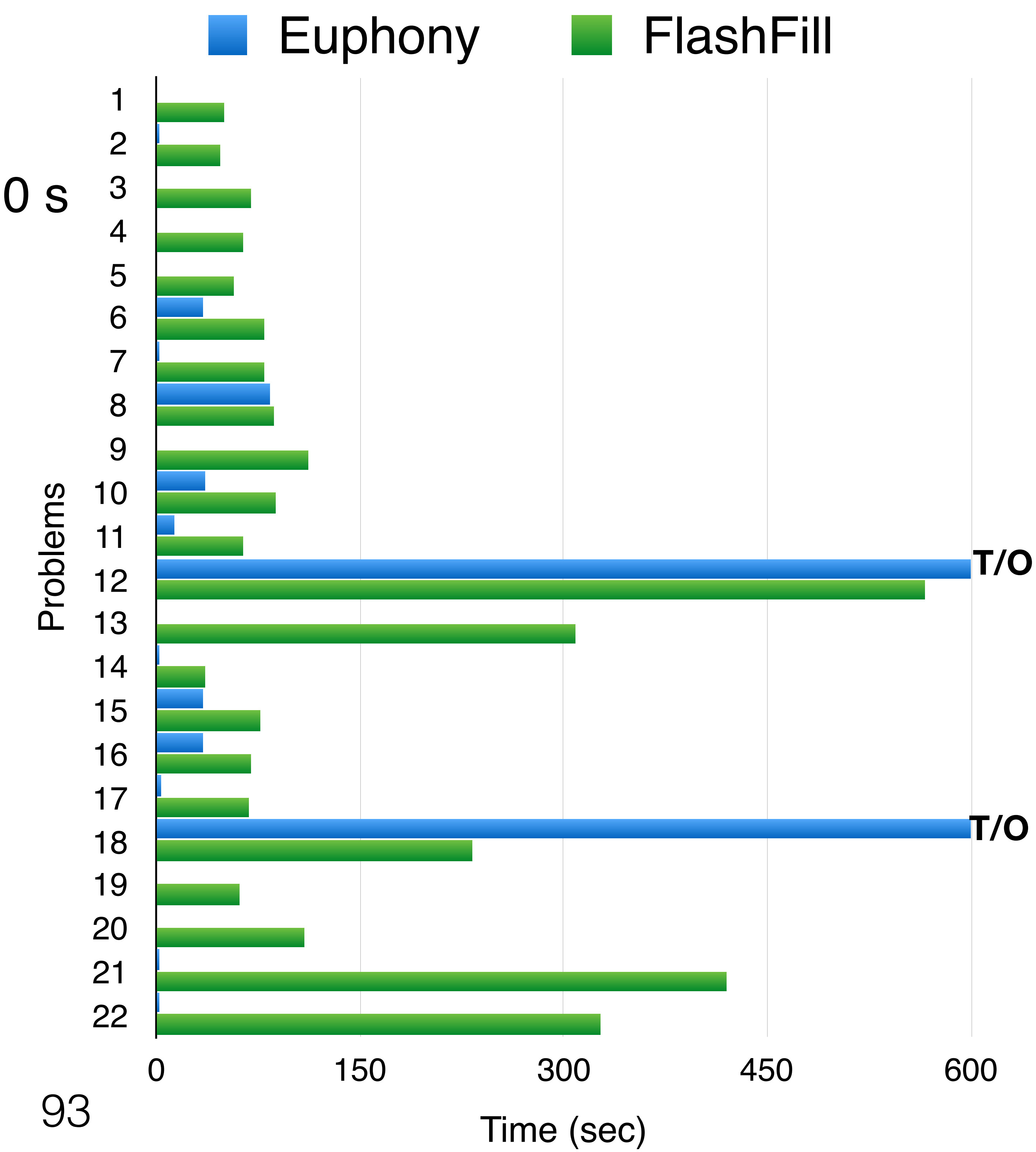


- Euphony solved 78% within 1 min
- solved 8 on which EUSolver timed out
- outperformed EUSolver on all

Comparison with FlashFill (STRING)

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

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



Efficacy of A* Search

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

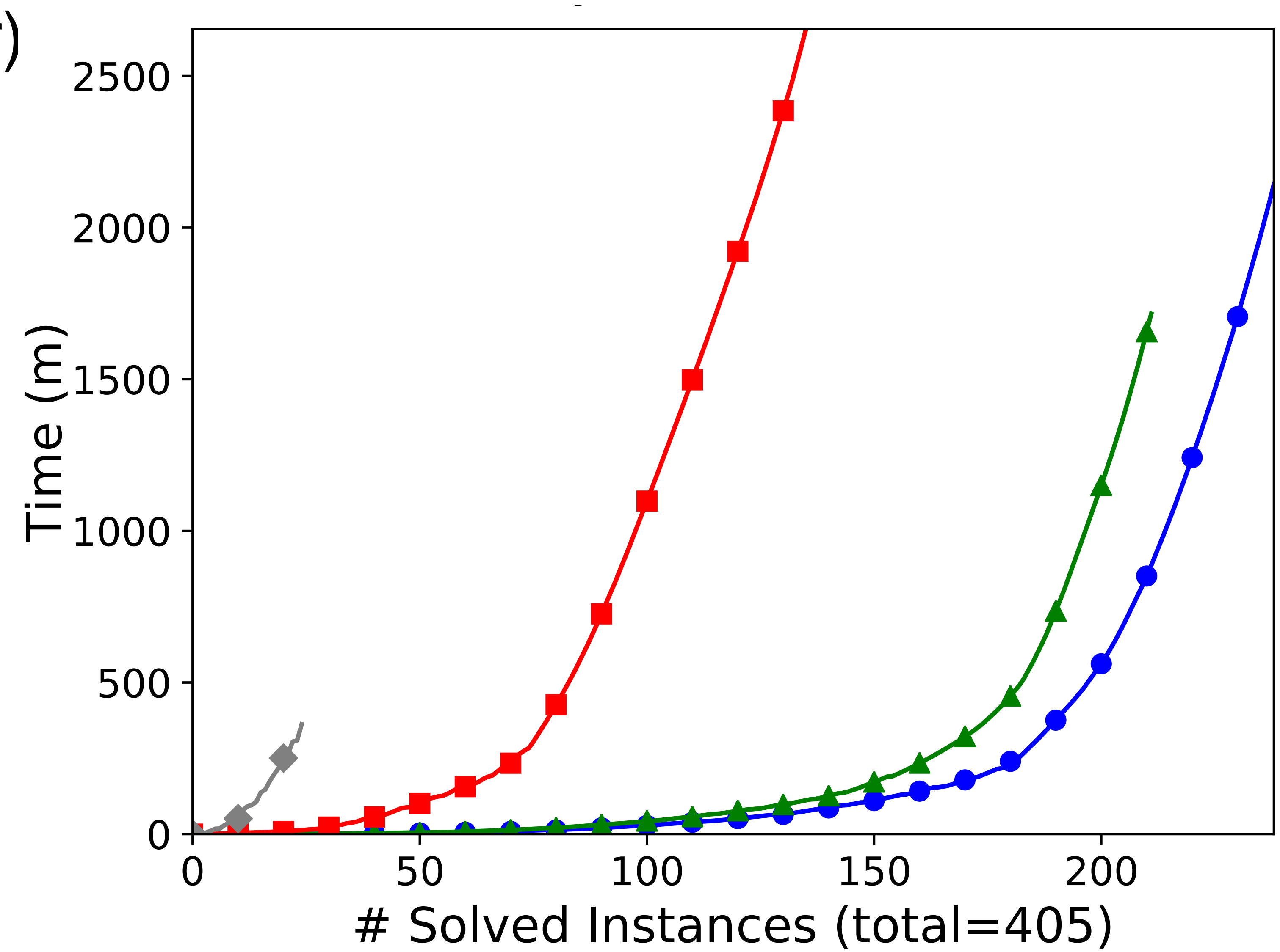
- # Solved (timout: 1 hour)

A* + PHOG: 236

Dijkstra + PHOG: 209

A* + PCFG: 133

Dijkstra + PCFG: 22



In the paper …

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

Thank you.