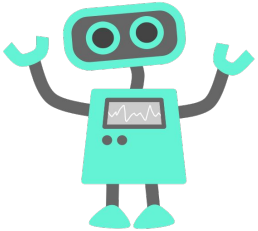
Toward Trustworthy Neural Program Synthesis

Kevin Ellis
Joint with Darren Key, Wen-Ding Li
Cornell University
Workshop on Dependable and Secure Software Systems
ETH 2022

The Learning + Recursion Lab

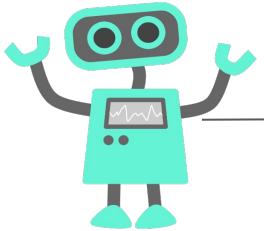
Program synthesizer

specification →  → `program`

Specification = Dependent Types
(types + logical predicates)

Program synthesizer

$$\texttt{n: Nat} \rightarrow \texttt{x:}\alpha \rightarrow \{\texttt{List}\ \alpha \mid \texttt{len}\ \nu = \texttt{n}\}$$



```
replicate = λn . λx . if n ≤ 0
  then Nil
  else Cons x (replicate (dec n) x)
```

Synquid. Polikarpova 2016

Specification = Input-Outputs

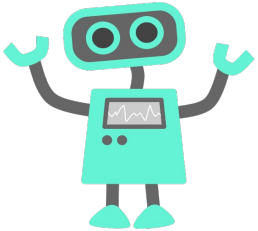
“Cornell University” ->

“C.U.”

“Eidgenössische Technische Hochschule” ->

“E.T.H.”

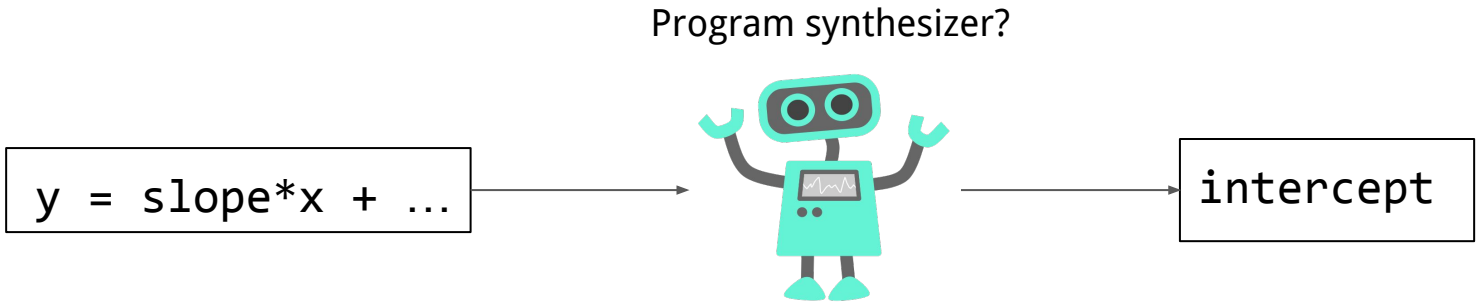
Program synthesizer



```
“.”.join(x.split())
```

Eg, FlashFill. Gulwani 2012

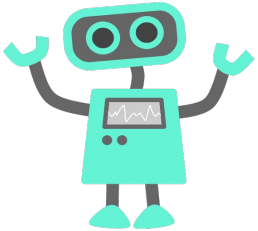
Specification = Partially completed program



Specification = Partially completed program

```
# open file and loop over lines
with open(file, “r”) as handle:
```

Program synthesizer?

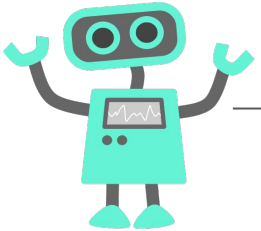


```
for ln in file:
```

Specification = Partially completed program

```
# flatten a list called `xss`
```

Program synthesizer?

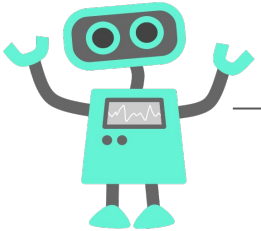


```
[x
 for xs in xss
 for x in xs ]
```

Specification = Natural language comment

```
# flatten a list called `xss`
```

Program synthesizer?



```
[x
 for xs in xss
 for x in xs ]
```

Large language models for source code

OpenAI Codex, GitHub Copilot

12 billion learned parameters

159 gigabytes of data from GitHub

```
# Write a python function called `abbreviate`
# that takes a string containing white space
# and returns the first letter of each word
# separated by periods.
def abbreviate(s):
    return '.'.join([c[0] for c in s.split()])
```

Large language models for source code: AlphaCode

You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcbd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcaa" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of s.

Large language models for source code: AlphaCode

```
t=int(input())
for i in range(t):
    s=input()
    t=input()
    a=[]
    b=[]
    for j in s:
        a.append(j)
    for j in t:
        b.append(j)
    a.reverse()
    b.reverse()
    c=[]
    while len(b)!=0 and len(a)!=0:
        if a[0]==b[0]:
            c.append(b.pop(0))
            a.pop(0)
        elif a[0]!=b[0] and len(a)!=1:
            a.pop(0)
            a.pop(0)
        elif a[0]!=b[0] and len(a)==1:
            a.pop(0)
    if len(b)==0:
        print("YES")
    else:
        print("NO")
```

First AlphaCode reads the two phrases.

If the letters at the end of both phrases don't match, the last letter must be deleted. If they do match we can move onto the second last letter and repeat.

Backspace deletes two letters. The letter you press backspace instead of, and the letter before it.

If we've matched every letter, it's possible and we output that.

Evan Pu added a new photo.

November 16, 2021 ·

copilot's buggy code suggestion (in gray) against the correct code (below)

it is very subtle, but caused my search algorithm to bug out and invalidated 2 days worth of works

please use responsibly I guess is my take-away.

```
rect_params = t_rank[queue_ids[0]], b_rank[queue_ids[1]], l_rank[queue_ids[2]], r_rank[queue_ids[3]]
log_rect_prob = np.log(t[queue_ids[0]]) + np.log(b[queue_ids[1]]) + np.log(l[queue_ids[2]]) + np.log(r[queue_ids
return rect_params, log_rect_prob
log_rect_prob = np.log(t[rect_params[0]]) + np.log(b[rect_params[1]]) + np.log(l[rect_params[2]]) + np.log(r[rec
return rect_params, log_rect_prob
```

I Speak, You Verify: Toward Trustworthy Neural Program Synthesis

Darren Key, Wen-Ding Li, Kevin Ellis. 2022

Trust in Traditional Program Synthesis

Synthesis: Dreams $\Longrightarrow$ Programs

ZOHAR MANNA AND RICHARD WALDINGER

To specify a program *maxlist* to compute the largest element of a given list l, we write

$$maxlist(l) \Longleftarrow \textbf{compute } some\ z : z \in l \text{ and } z \geqslant all(l)$$

where l is a nonempty list of numbers.

Trust in Traditional Program Synthesis

From Program Verification to Program Synthesis

Saurabh Srivastava
University of Maryland, College Park
saurabhs@cs.umd.edu

Sumit Gulwani
Microsoft Research, Redmond
sumitg@microsoft.com

Jeffrey S. Foster
University of Maryland, College Park
jfoster@cs.umd.edu

(a)
```
Bresenhams(int X,Y) {
    v1:=2Y−X; y:=0; x:=0;
    while (x ≤ X)
      | out[x]:=y;
      | if (v1 < 0)
      |     v1:=v1+2Y;
      | else
      |     v1:=v1+2(Y−X); y++;
      | x++;
    return out;
}
```

(b)
```
Bresenhams(int X,Y) {
    []true → v1'=2Y−X ∧ y'=0 ∧ x'=0
    while (x ≤ X)
      | []v1 < 0 → out'=upd(out,x,y) ∧ v1'=v1+2Y ∧ y'=y ∧ x'=x+1
      | []v1 ≥ 0 → out'=upd(out,x,y) ∧ v1'=v1+2(Y−X) ∧ y'=y+1 ∧ x'=x+1
    return out;
}
```

(c)

Invariant τ : $0 < Y \leq X \wedge v_1 = 2(x+1)Y - (2y+1)X \wedge 2(Y-X) \leq v_1 \leq 2Y \wedge \forall k : 0 \leq k < x \Rightarrow 2|out[k] - (Y/X)k| \leq 1$

Ranking function φ : $X - x$

Figure 1. (a) Bresenham's line drawing algorithm (b) The invariant and ranking function that prove partial correctness and termination, respectively. (c) The algorithm written in transition system form, with statements as equality predicates, guarded appropriately.

Trust in Traditional Program Synthesis

Automating String Processing in Spreadsheets Using Input-Output Examples

Sumit Gulwani
Microsoft Research, Redmond, WA, USA
sumitg@microsoft.com

THEOREM 3 (Soundness). *The set $\tilde{P}$ of string expressions returned by* `GenerateStringProgram`$(\{(\sigma_i, s_i)\}_i)$ *are all consistent with each input-output pair (σ_i, s_i), i.e.,*

$$\forall P \in \tilde{P} \;\; \forall i : \;\; ([\![P]\!]\, \sigma_i) = s_i$$

Trust in Traditional Program Synthesis

$$\text{program} \vdash \text{specification}$$

Trust in ***Neural*** Program Synthesis

$$\text{program} \vdash \text{specification} \qquad (?)$$

Trust in Neural Program Synthesis

program $\vdash$ natural language (?)

Trust in Neural Program Synthesis

program $\vdash$ natural language (**X**)

Trust in Neural Program Synthesis

Neural network defines:

Pr[program | natural language]

The Trust Conundrum

Trust ~ Verification

program $\vdash$ specification

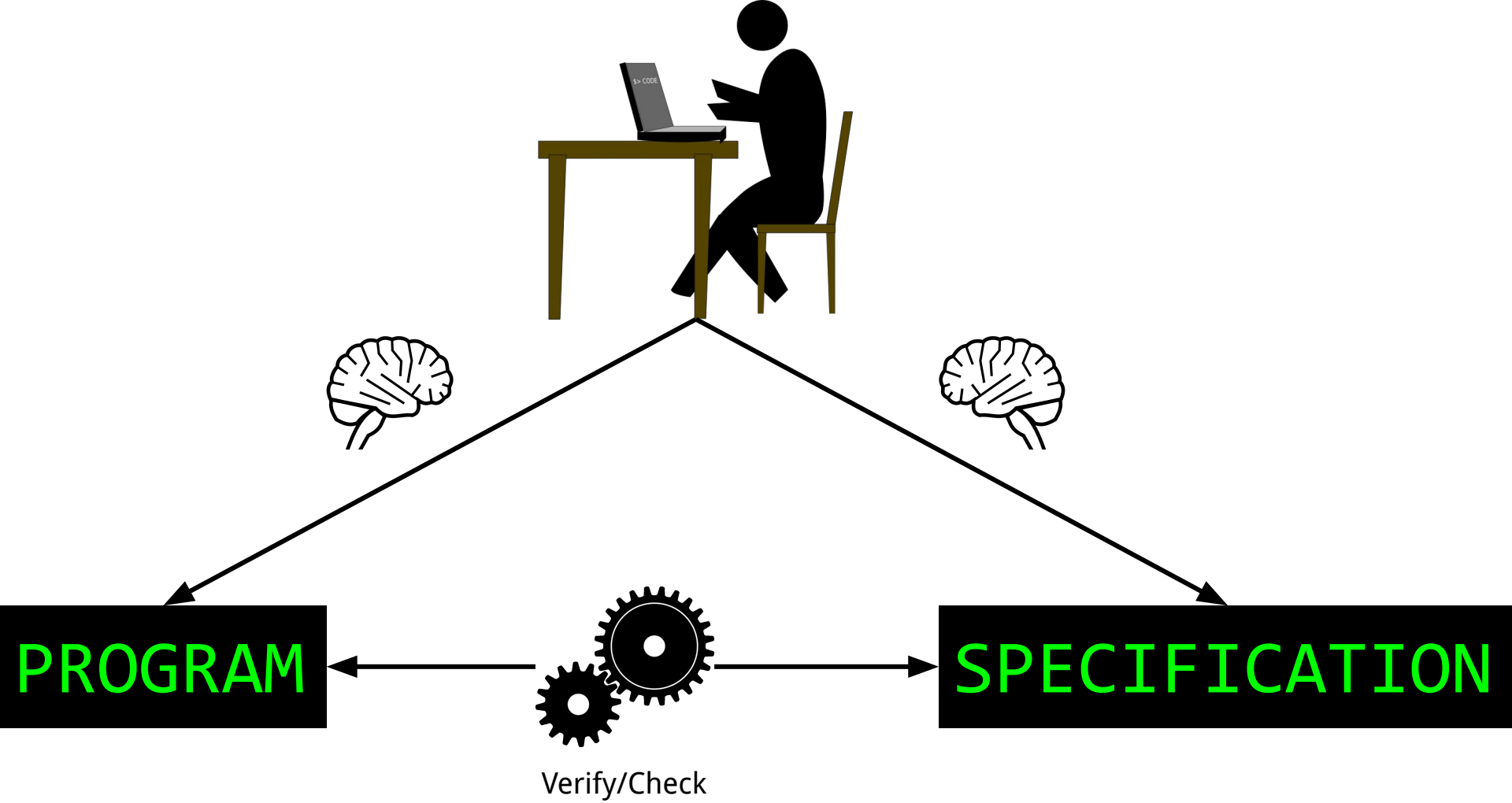
My specification is informal…

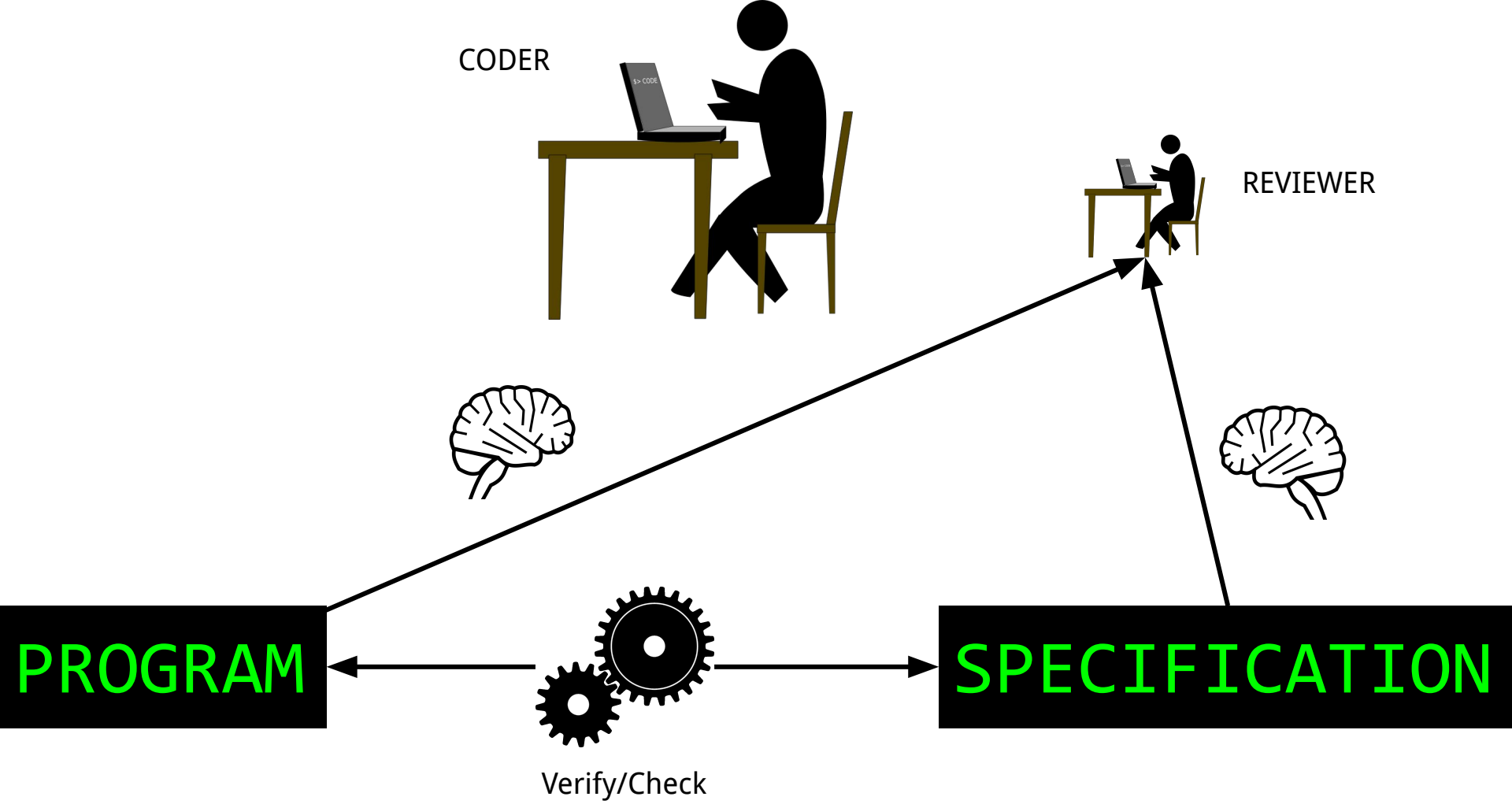
…because train data is messy natural code

And I can’t verify against an informal specification

How do people build trust?

```python
class grafanaTests(unittest.TestCase):
    """Tests for GeekTechStuff Grafana API Python"""

    def test_admin_name_is_string(self):
        admin_username = main.get_username()
        self.assertIs(type(admin_username),str)

    def test_admin_password_is_string(self):
        admin_password = main.get_password()
        self.assertIs(type(admin_password),str)

    def test_grafana_url_is_string(self):
        grafana_url = main.get_url()
        self.assertIs(type(grafana_url),str)
```


$> CODE
PROGRAM
SPECIFICATION
Verify/Check


CODER
REVIEWER
PROGRAM
SPECIFICATION
Verify/Check

How to escape the trust conundrum

1. Start with informal intention
2. Formalize intention into program and specification
3. Enforce program $\vdash$ specification
4. Human-in-the-loop: Code reviewer checks both program and specification

How to escape the trust conundrum

1. Start with informal intention
2. Formalize intention into program and specification
3. Enforce program $\vdash$ specification
4. Human-in-the-loop: Code reviewer checks both program and specification
5. **Know your own limitations:**

   don’t try to write a program if you can’t get bug free code

Evan Pu added a new photo.

November 16, 2021 ·

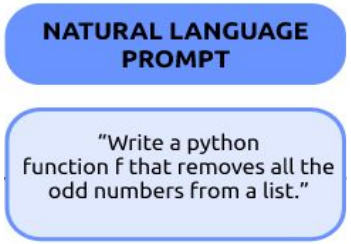
copilot's buggy code suggestion (in gray) against the correct code (below)

it is very subtle, but caused my search algorithm to bug out and invalidated 2 days worth of works

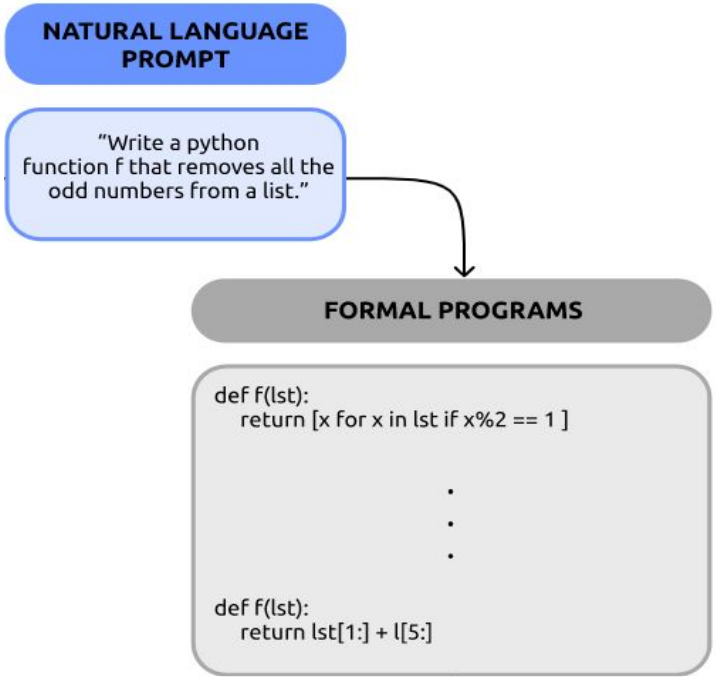
please use responsibly I guess is my take-away.

```
rect_params = t_rank[queue_ids[0]], b_rank[queue_ids[1]], l_rank[queue_ids[2]], r_rank[queue_ids[3]]
log_rect_prob = np.log(t[queue_ids[0]]) + np.log(b[queue_ids[1]]) + np.log(l[queue_ids[2]]) + np.log(r[queue_ids
return rect_params, log_rect_prob
log_rect_prob = np.log(t[rect_params[0]]) + np.log(b[rect_params[1]]) + np.log(l[rect_params[2]]) + np.log(r[rec
return rect_params, log_rect_prob
```

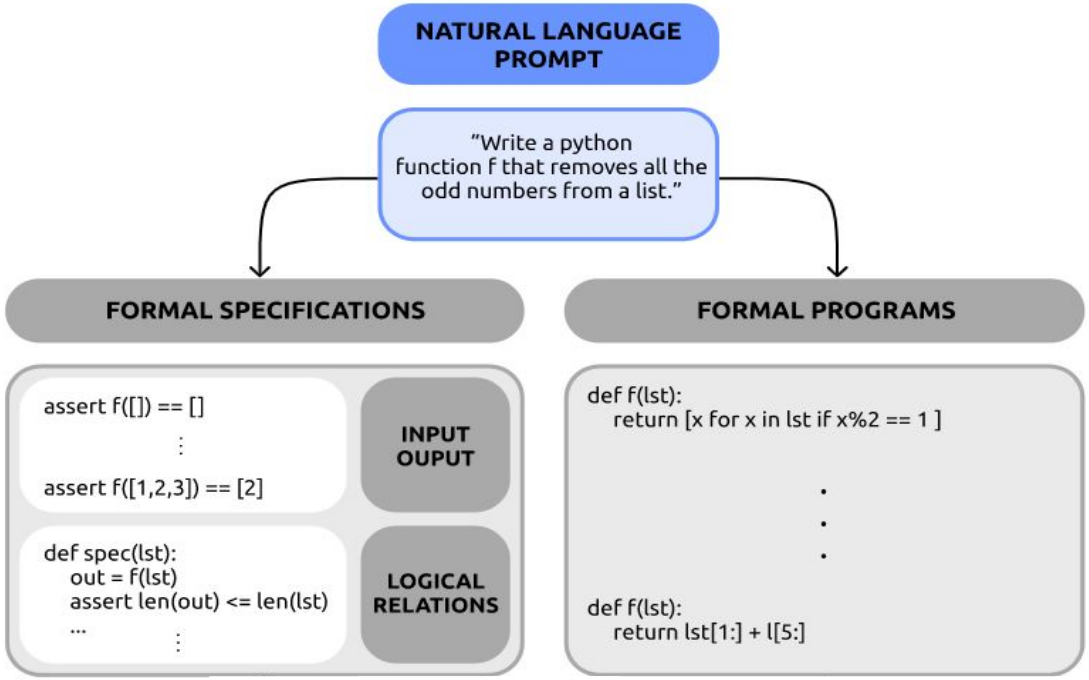
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."

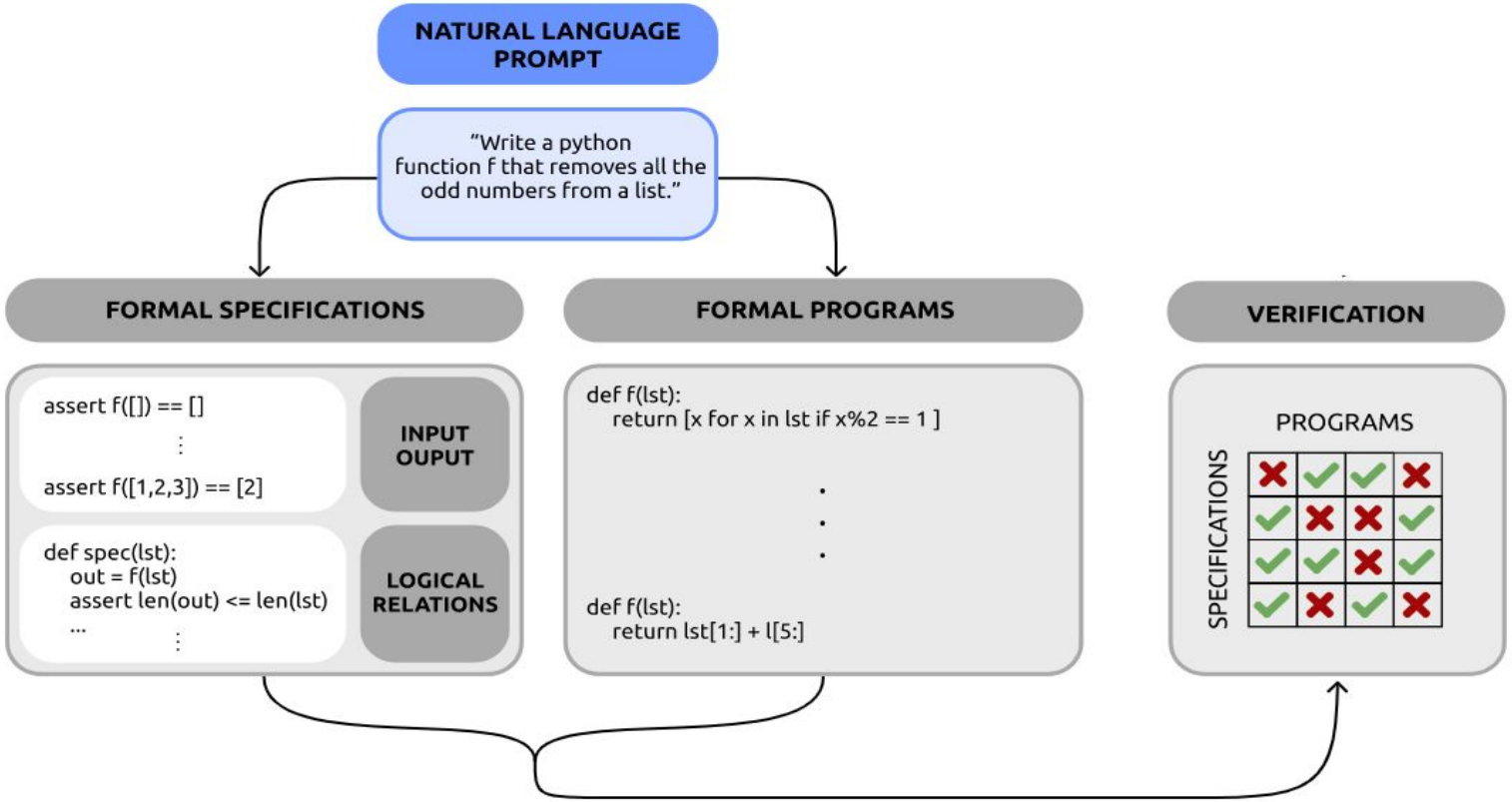
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL PROGRAMS
def f(lst):
return [x for x in lst if x%2 == 1]
.
.
.
def f(lst):
return lst[1:] + l[5:]

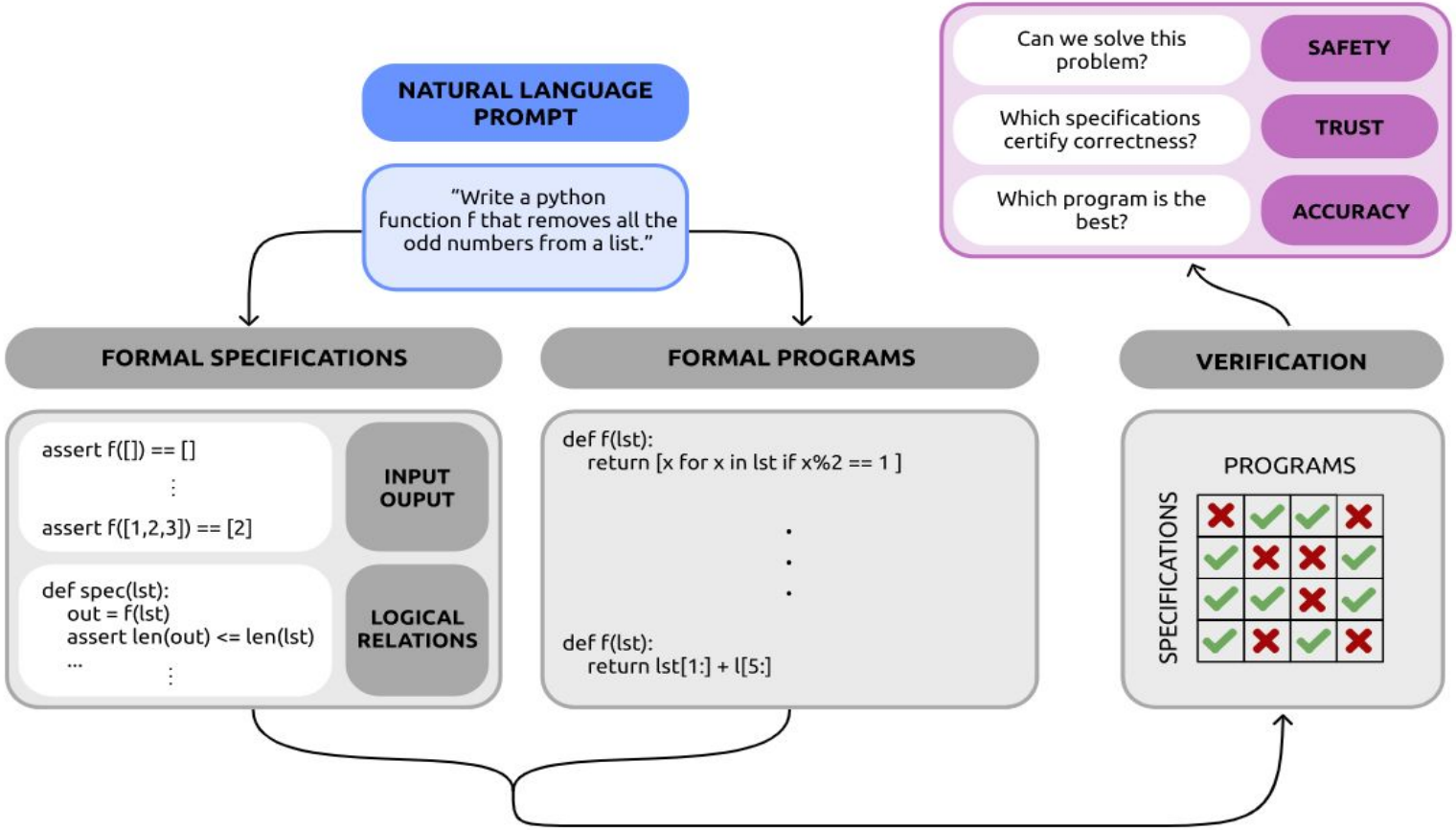
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL SPECIFICATIONS
FORMAL PROGRAMS
assert f([]) == []
⋮
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
⋮
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]

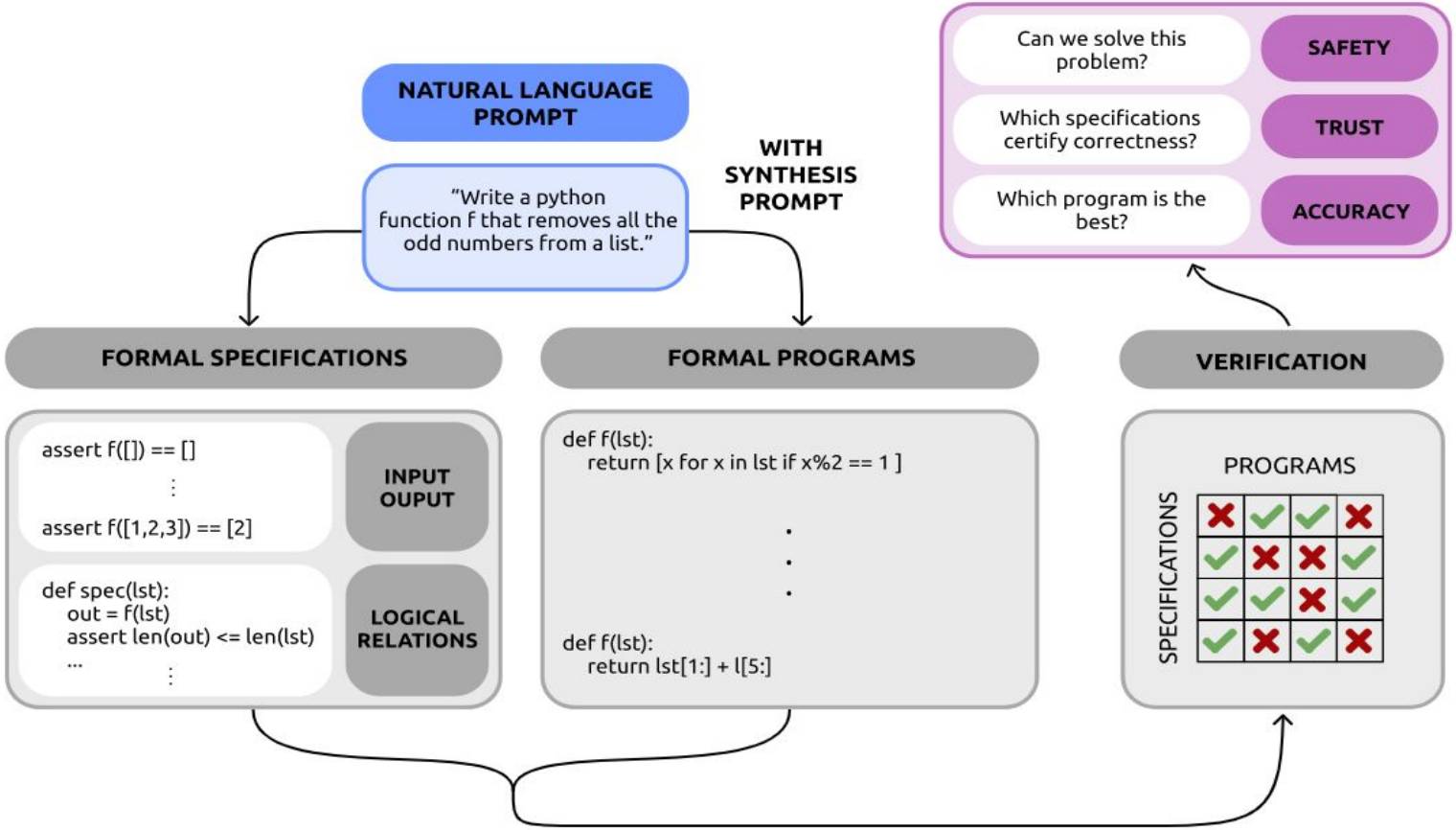
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL SPECIFICATIONS
FORMAL PROGRAMS
VERIFICATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL SPECIFICATIONS
assert f([]) == []
⋮
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
⋮
LOGICAL RELATIONS
FORMAL PROGRAMS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
VERIFICATION
PROGRAMS
SPECIFICATIONS
Can we solve this problem?
SAFETY
Which specifications certify correctness?
TRUST
Which program is the best?
ACCURACY

Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
WITH SYNTHESIS PROMPT
Can we solve this problem?
SAFETY
Which specifications certify correctness?
TRUST
Which program is the best?
ACCURACY
FORMAL SPECIFICATIONS
FORMAL PROGRAMS
VERIFICATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

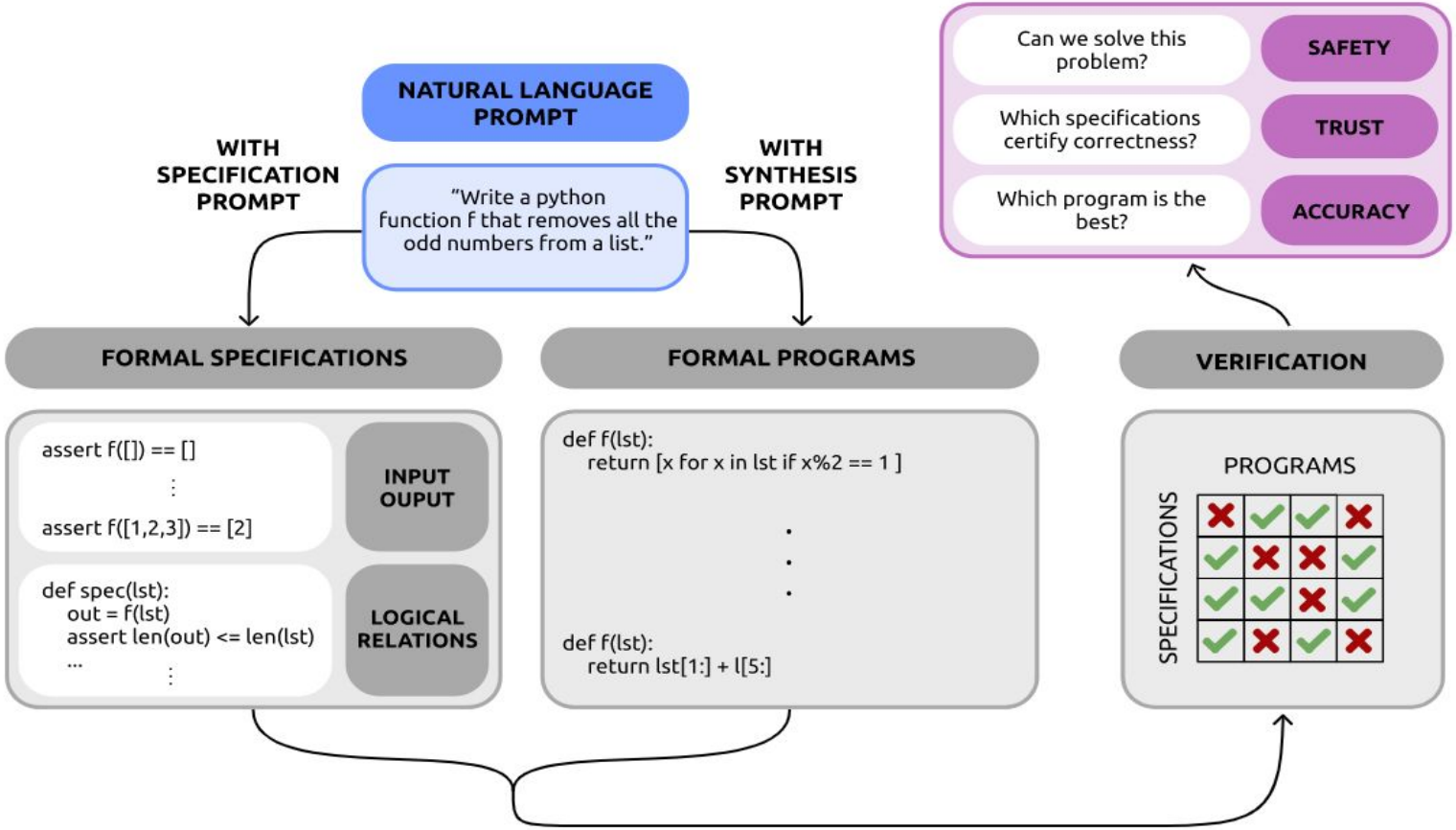
Generating Programs

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
```

Generating Programs

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
    return list(map(lambda x, y:  x-y, nums1, nums2))
```

Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
WITH SPECIFICATION PROMPT
WITH SYNTHESIS PROMPT
Can we solve this problem?
SAFETY
Which specifications certify correctness?
TRUST
Which program is the best?
ACCURACY
FORMAL SPECIFICATIONS
FORMAL PROGRAMS
VERIFICATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

Generating Specifications, input-outputs

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
```

Generating Specifications, input-outputs

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
    pass # To-do: implement

# Check if sub_list works
assert sub_list(
```

Generating Specifications, input-outputs

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
    pass # To-do: implement

# Check if sub_list works
assert sub_list([2, 3, 1], [1, 1, 1]) == [1, 2, 0]
```

Generating Specifications, logical relations

```
# Problem 3

# Write a function to subtract two lists element-wise.
def sub_list(nums1,nums2):
    pass # To-do: implement

# Test 3
```

Generating Specifications, logical relations

```
# Problem 3

# Write a function to subtract two lists element-wise.
def sub_list(nums1,nums2):
    pass # To-do: implement

# Test 3
def test_sub_list(nums1 : list, nums2 : list):
    """
    Given two lists ‘nums1‘ and ‘nums2‘, test whether function ‘sub_list‘ is implemented correctly.
    """
    output_list = sub_list(nums1, nums2)
    # check if the length of the output list is the same as the lengths of the input lists
    assert len(output_list) == len(nums1) == len(nums2)
    # check if the output list has the expected elements
    for i in range(len(output_list)):
        assert output_list[i] == nums1[i] - nums2[i]

# run the testing function ‘test_sub_list‘ on a new testcase
test_sub_list([1, 2, 3, 4], [10, 9, 8, 7])
```

Generating Specifications, logical relations

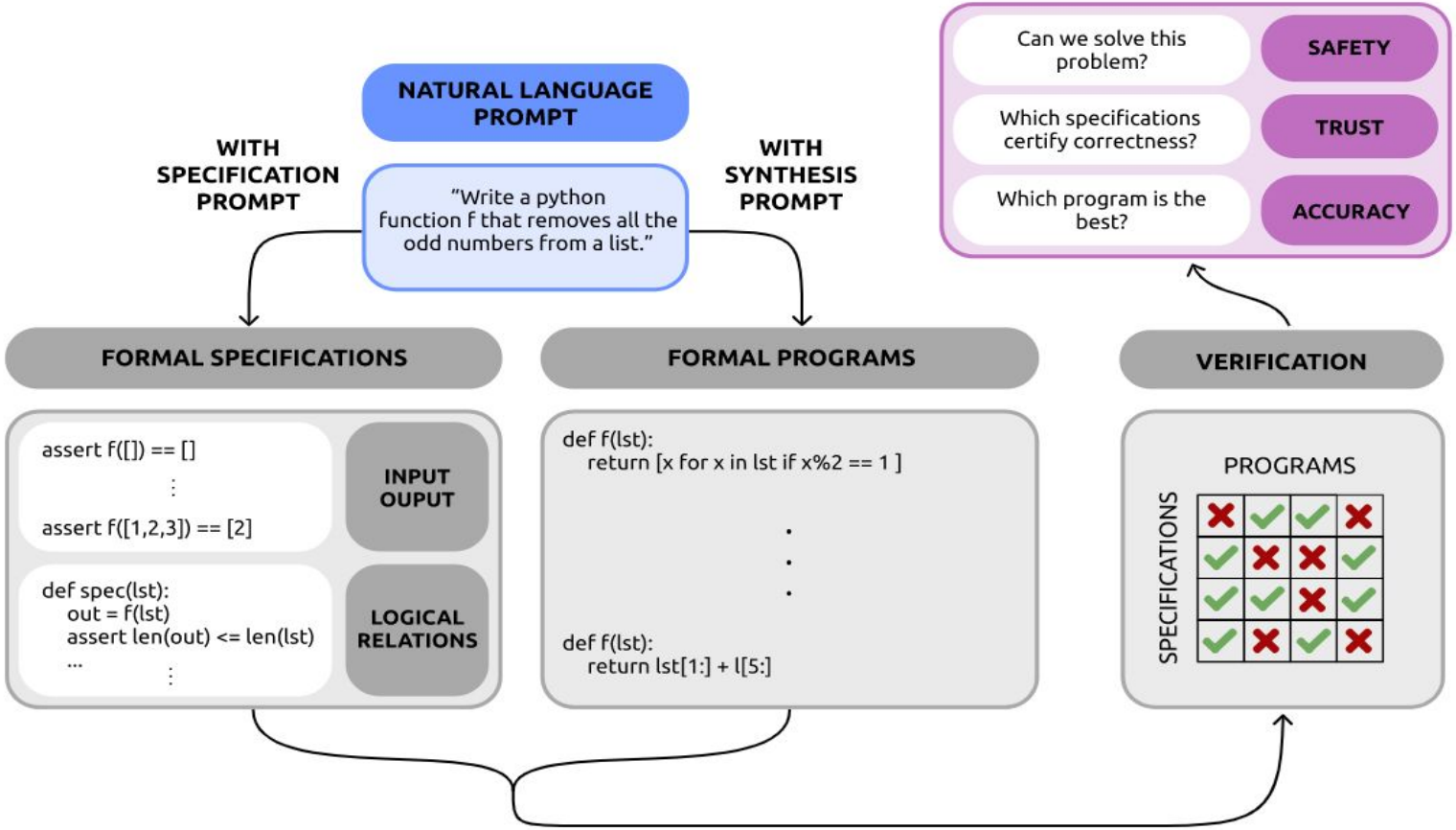
```
# Problem 1

from typing import List

# Given a list of integers, return a list th
↪  integers.
def filtered_even_integers(input_list: List(
    pass # To-do: Implement

# Test 1

def test_filtered_even_integers(input_list: List()):
    """ Given an input `input_list`, test whether the
    ↪  `filtered_even_integers` is implemented corre
    """
    output_list = filtered_even_integers(input_list)
    # check if the output list only contains odd inte
    for integer in output_list:
        assert integer % 2 == 1
    # check if all the integers in the output list ca
    ↪  input list
    for integer in output_list:
        assert integer in input_list

# run the testing function `test_filtered_even_intege
test_filtered_even_integers([31, 24, 18, 99, 1000, 52
```

```
# Problem 2

# Return a string where the vowels (`a`, `e`, `i`, `o`, `u`, and their
↪  capital letters) are repeated twice in place
def repeat_vowel(input_str: str) -> str:
    pass # To-do: Implement

# Test 2  ..................
```

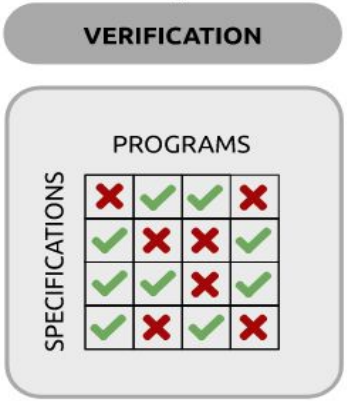
```
# Problem 3

# Write a function to subtract two lists element-wise.
def sub_list(nums1,nums2):
    pass # To-do: implement

# Test 3
```

```
def test_sub_list(nums1 : list, nums2 : list):
    """
    Given two lists ‘nums1‘ and ‘nums2‘, test whether function ‘sub_list‘ is implemented correctly.
    """
    output_list = sub_list(nums1, nums2)
    # check if the length of the output list is the same as the lengths of the input lists
    assert len(output_list) == len(nums1) == len(nums2)
    # check if the output list has the expected elements
    for i in range(len(output_list)):
        assert output_list[i] == nums1[i] - nums2[i]

# run the testing function ‘test_sub_list‘ on a new testcase
test_sub_list([1, 2, 3, 4], [10, 9, 8, 7])
```

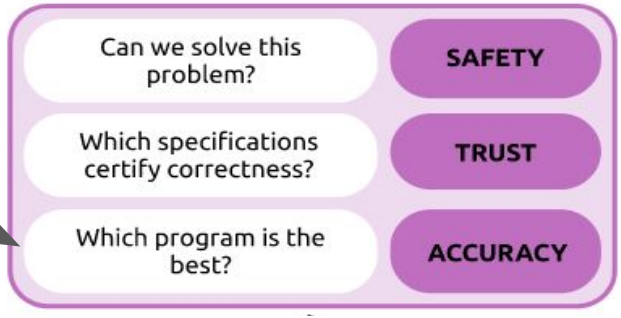
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
WITH SPECIFICATION PROMPT
WITH SYNTHESIS PROMPT
Can we solve this problem?
SAFETY
Which specifications certify correctness?
TRUST
Which program is the best?
ACCURACY
FORMAL SPECIFICATIONS
FORMAL PROGRAMS
VERIFICATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

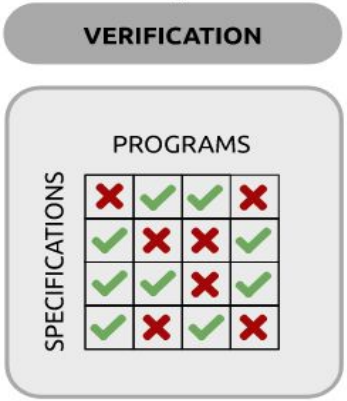
Verification Matrix -> Objectives



Pr[program is correct]



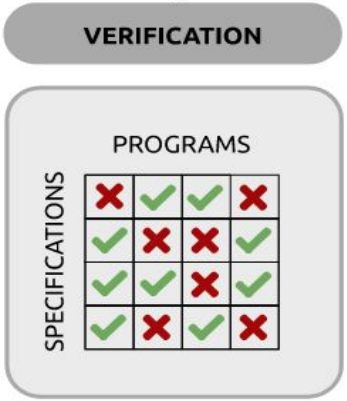
Verification Matrix -> Objectives



Pr[program is correct]

+ Satisfies many specs?
+ Many other progs satisfy same specs?
- Lots of different prog behaviors?

Verification Matrix -> Objectives



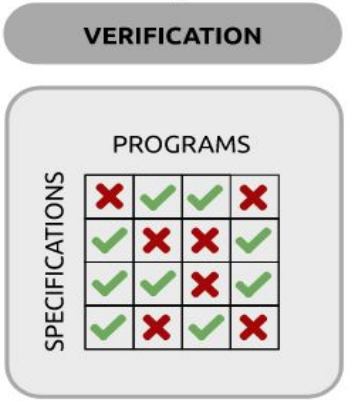
+ Satisfies many specs?
+ Many other progs satisfy same specs?
- Lots of different prog behaviors?

Pr[prog correct]

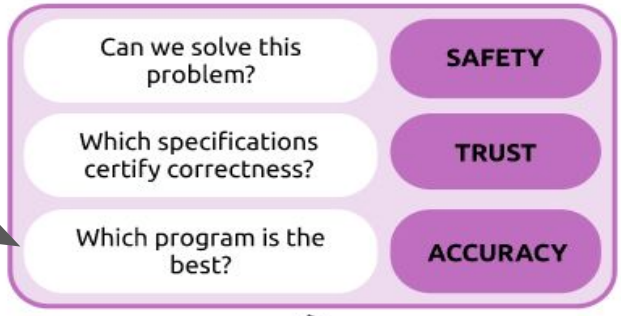
Feature extraction

Binary classification
(logistic regression)

Objectives



Pr[program is correct]



$$\underset{\text{prog}}{\operatorname{argmax}} \Pr[\text{ prog is correct }]$$

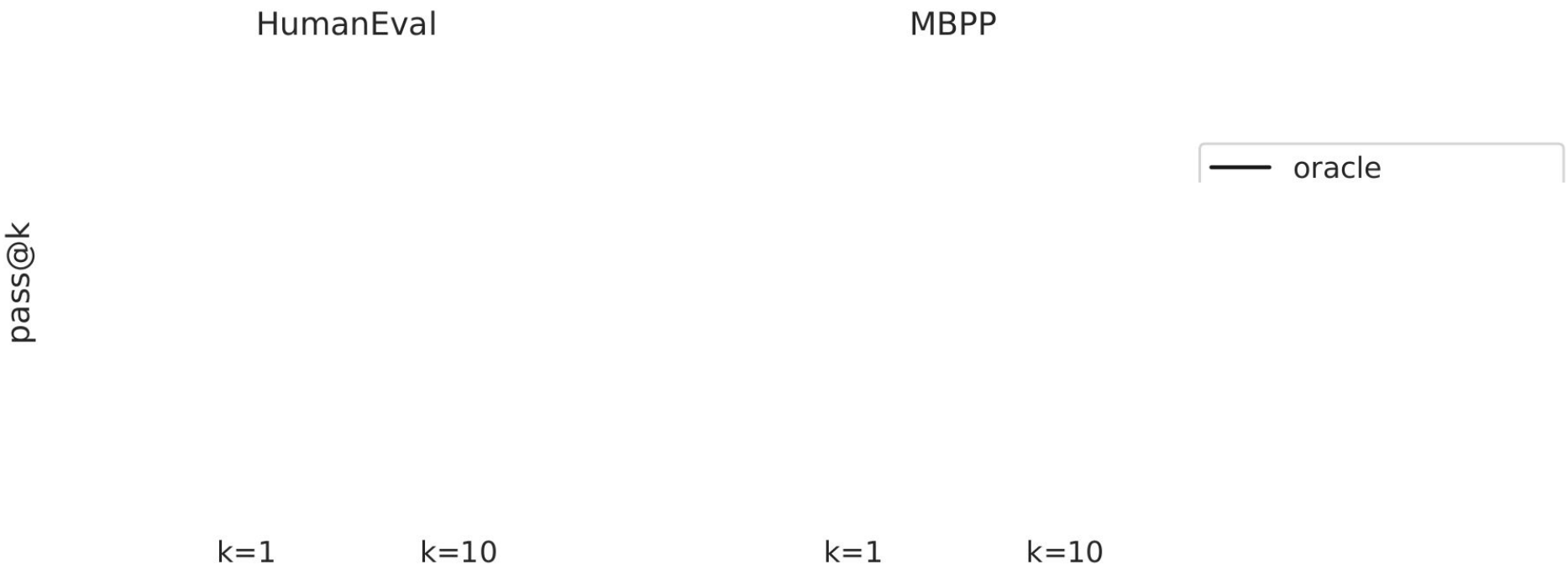
Pass@k

Get k guesses as to the correct program

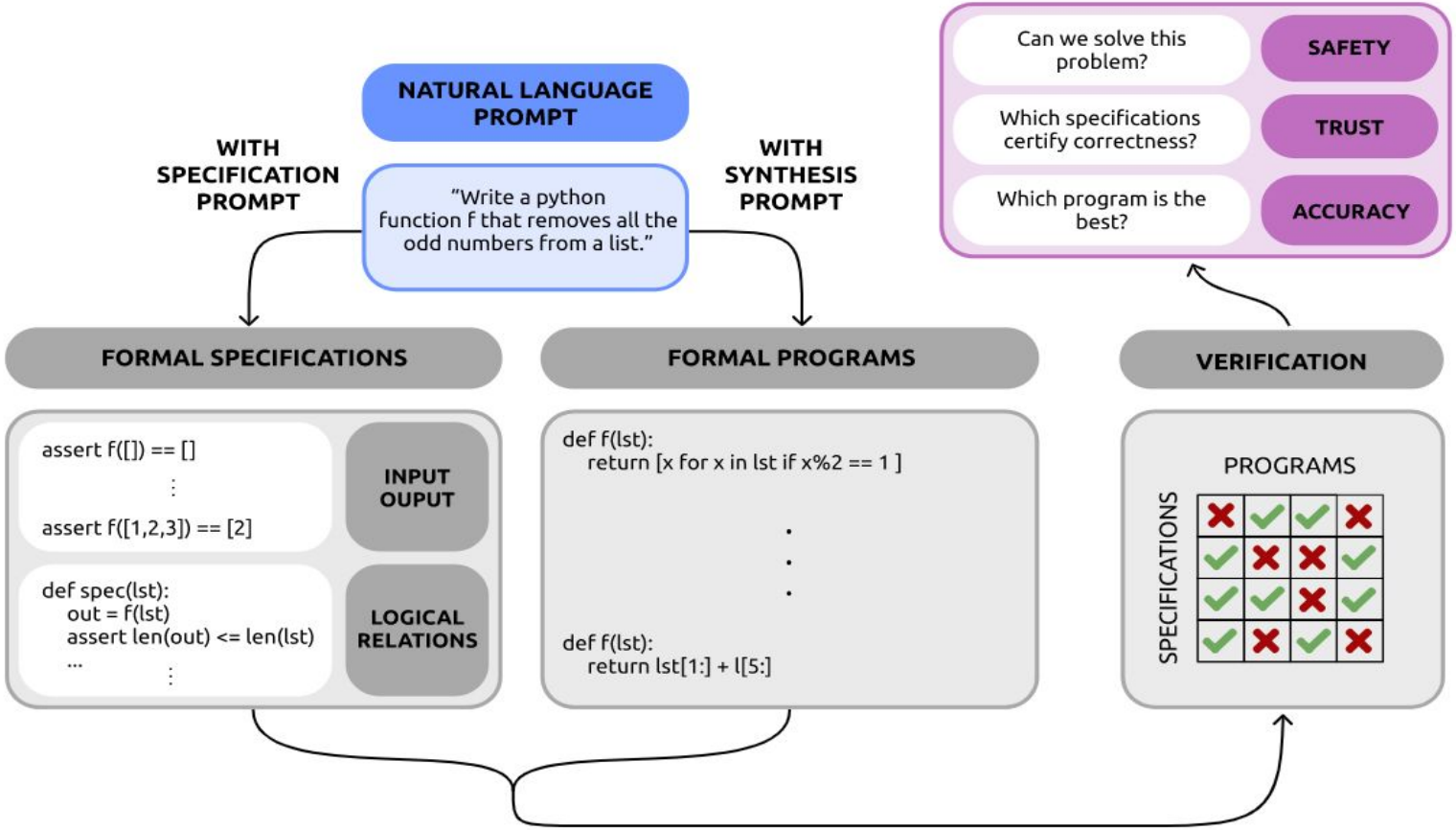
Probability one of your guesses is correct

Accuracy, not really trust

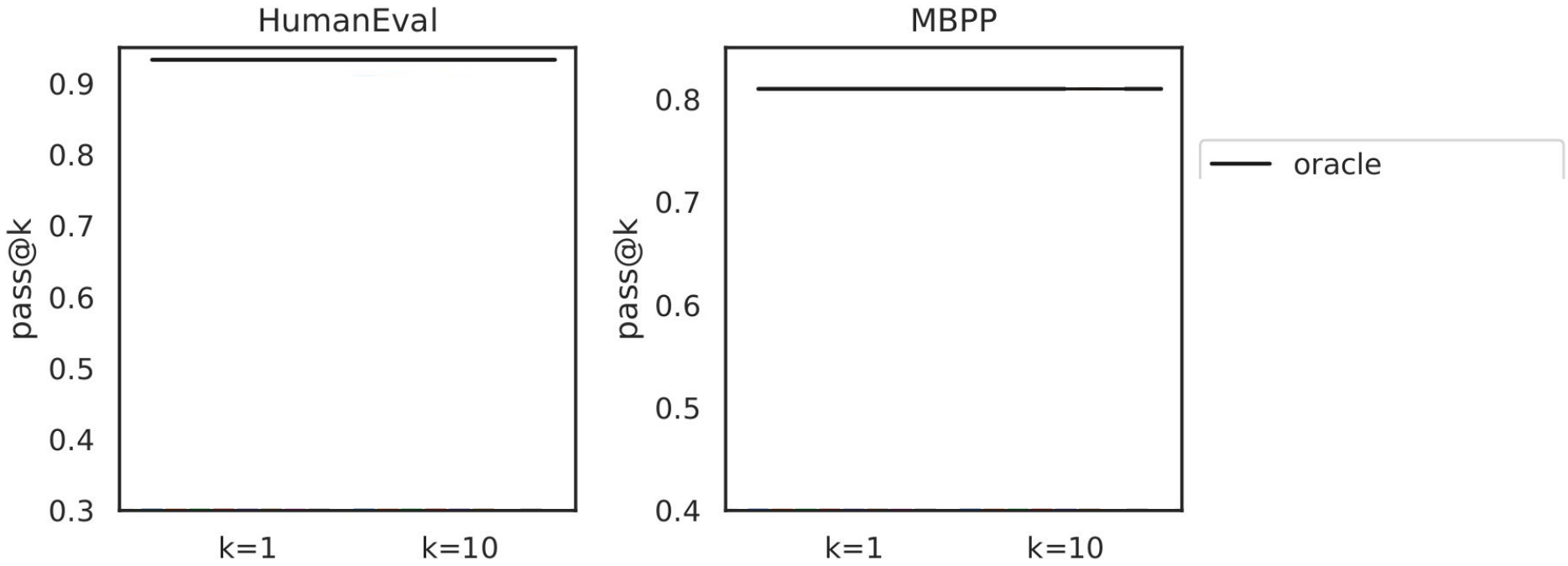
Pass@k



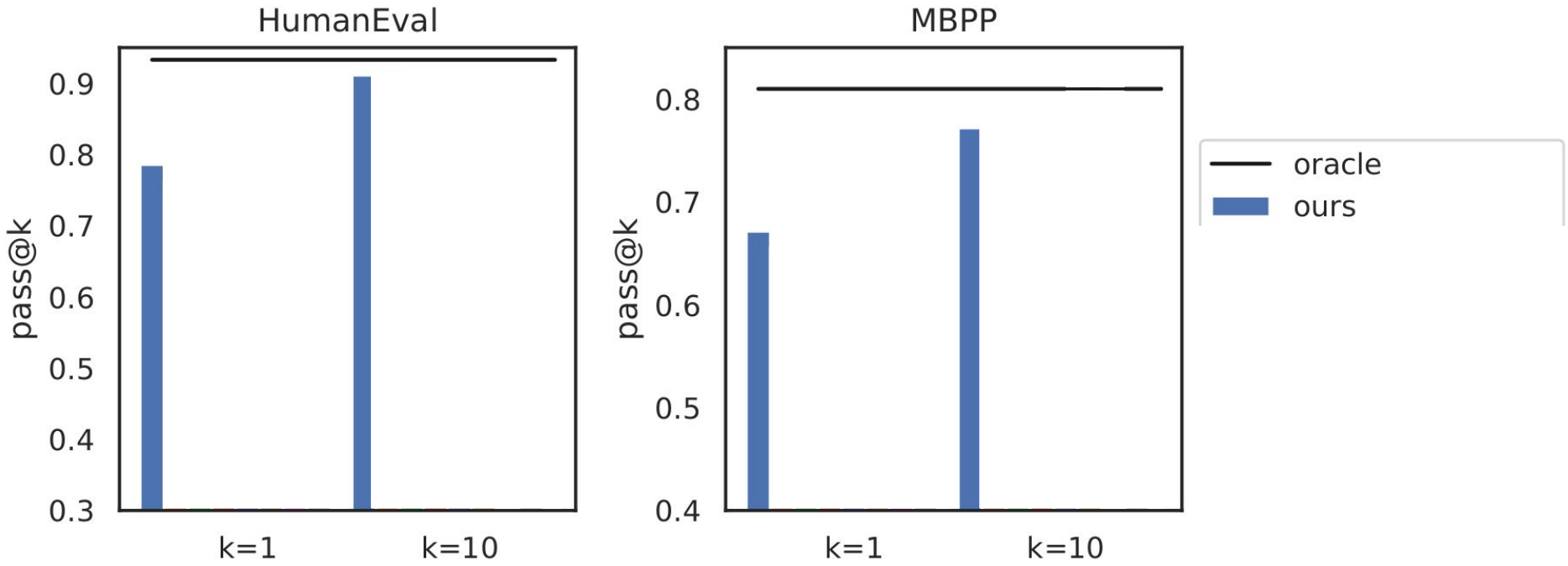
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
WITH SPECIFICATION PROMPT
WITH SYNTHESIS PROMPT
Can we solve this problem?
SAFETY
Which specifications certify correctness?
TRUST
Which program is the best?
ACCURACY
FORMAL SPECIFICATIONS
FORMAL PROGRAMS
VERIFICATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

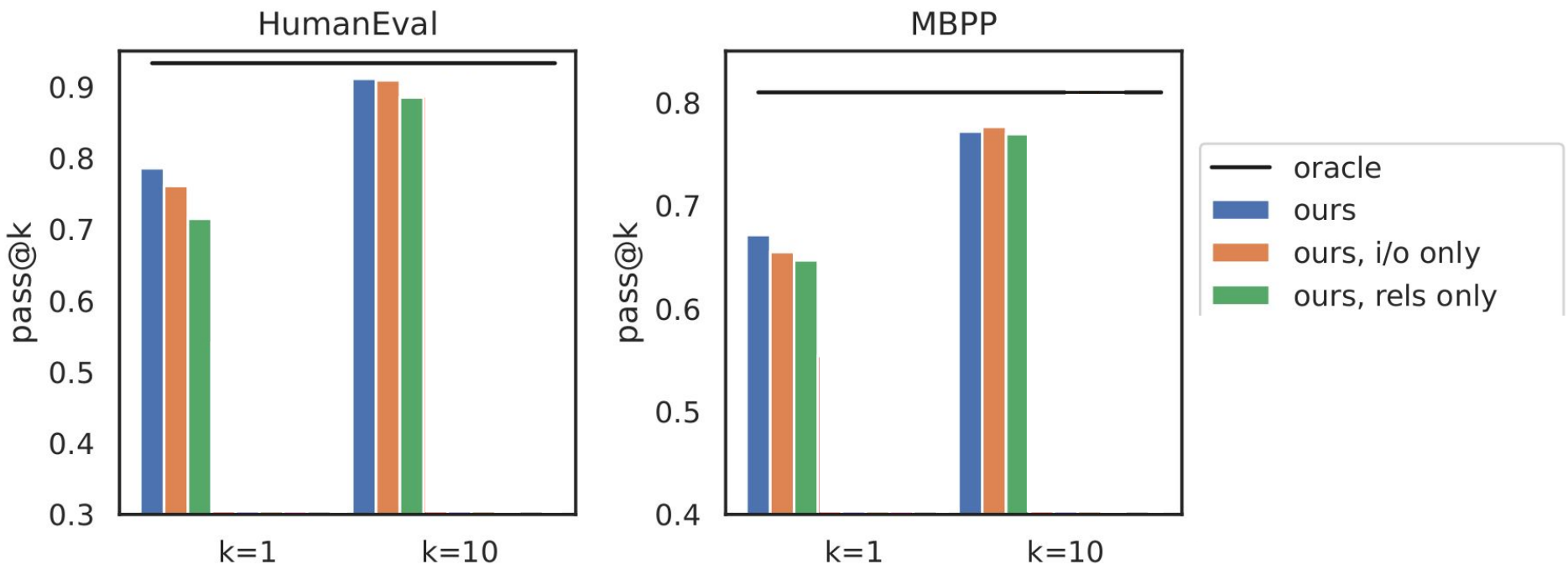
Pass@k



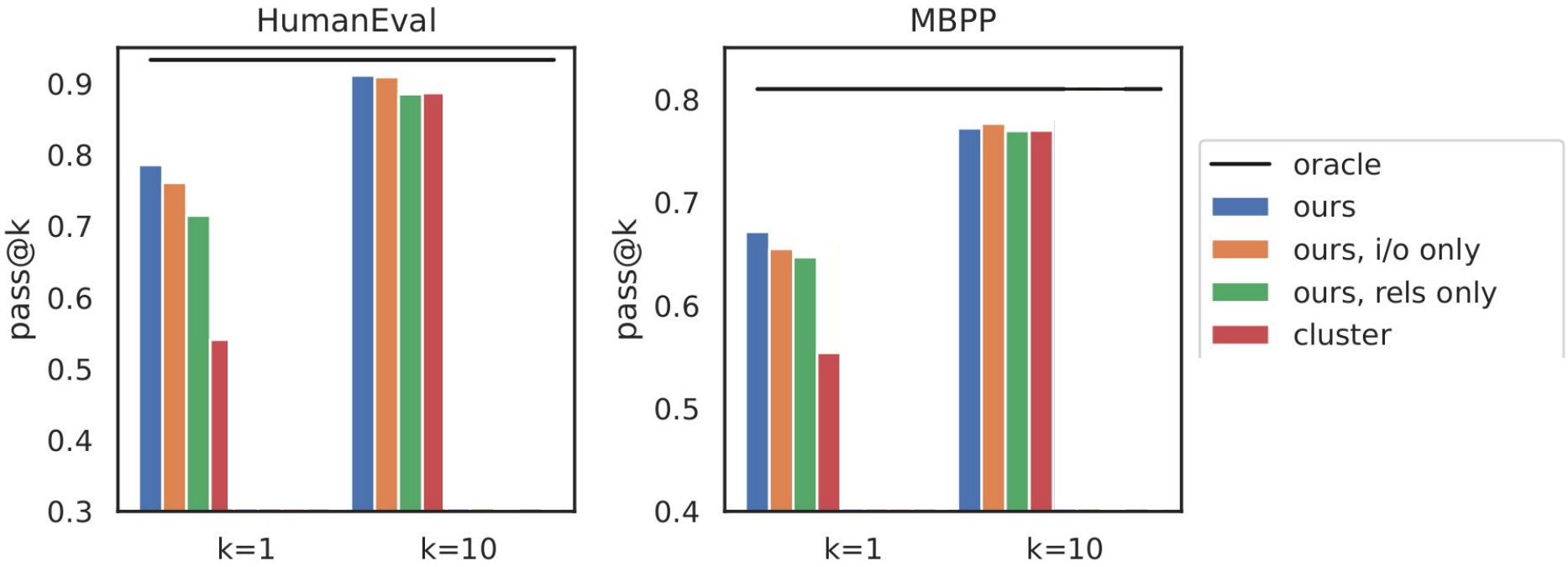
Pass@k



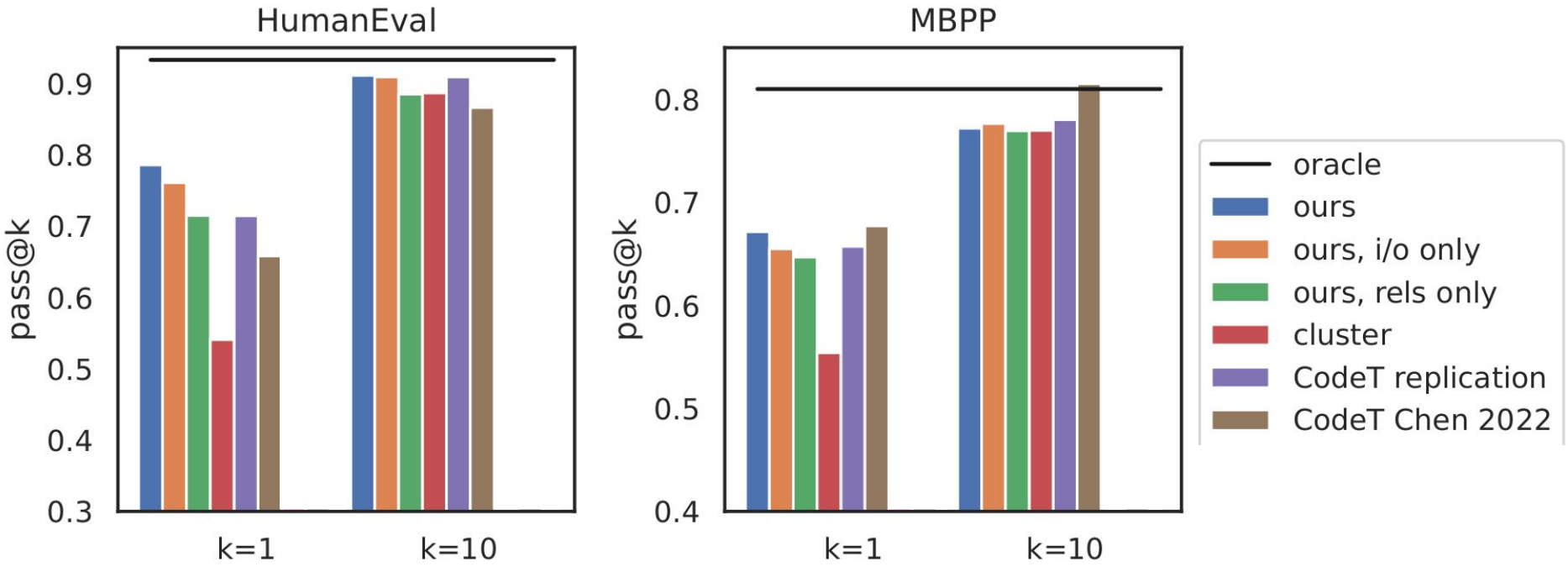
Pass@k


HumanEval
MBPP
pass@k
0.9
0.8
0.7
0.6
0.5
0.4
0.3
k=1
k=10
oracle
ours
ours, i/o only
ours, rels only

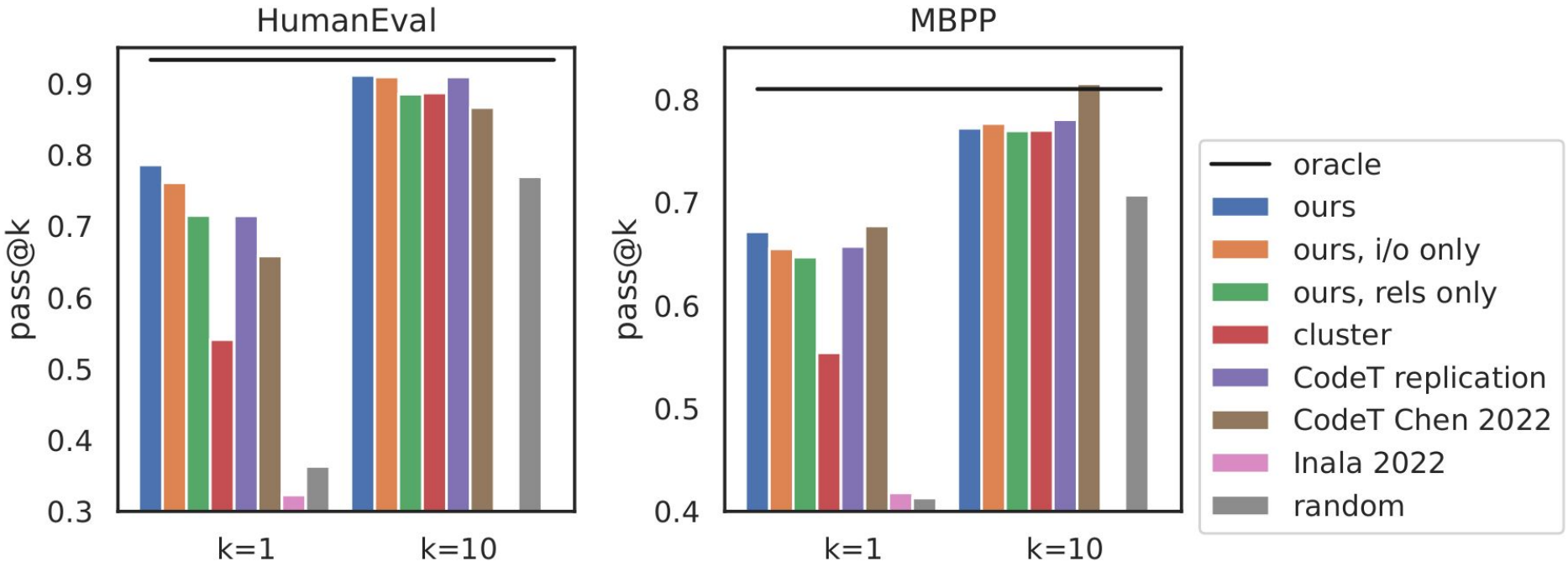
Pass@k


HumanEval
MBPP
pass@k
0.9
0.8
0.7
0.6
0.5
0.4
0.3
k=1
k=10
oracle
ours
ours, i/o only
ours, rels only
cluster

Pass@k


HumanEval
MBPP
pass@k
0.9
0.8
0.7
0.6
0.5
0.4
0.3
k=1
k=10
oracle
ours
ours, i/o only
ours, rels only
cluster
CodeT replication
CodeT Chen 2022

Pass@k

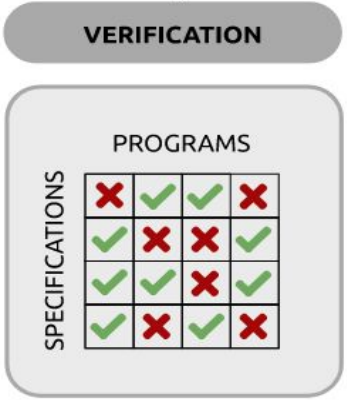

HumanEval
MBPP
pass@k
0.9
0.8
0.7
0.6
0.5
0.4
0.3
k=1
k=10
oracle
ours
ours, i/o only
ours, rels only
cluster
CodeT replication
CodeT Chen 2022
Inala 2022
random

Pass@k

Speculyzer: >= recent works

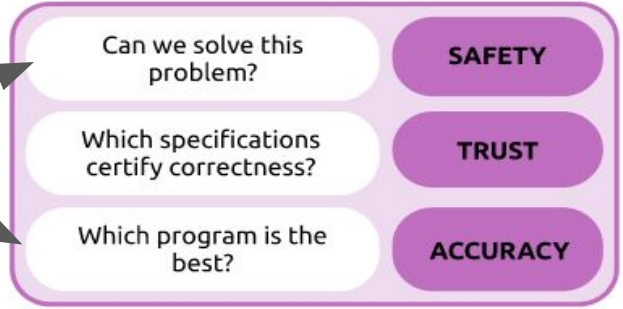
Pass@k: Not really the same thing as trust/safety

Objectives



Pr[program is correct]

$$\max_{prog} \Pr[\text{ prog is correct }] > \text{threshold}$$



$$\operatorname{argmax}_{prog} \Pr[\text{ prog is correct }]$$

This working requires our binary classifier is well “calibrated”

Recall (Coverage) vs Precision (Safety)

$$\text{can\_solve\_problem} = \max_{\text{prog}} \Pr[\text{ prog is correct }] > \textbf{\textit{threshold}}$$

$$\text{predicted\_program} = \underset{\text{prog}}{\text{argmax}} \Pr[\text{ prog is correct }]$$

High Threshold:

Sacrifice coverage for precision

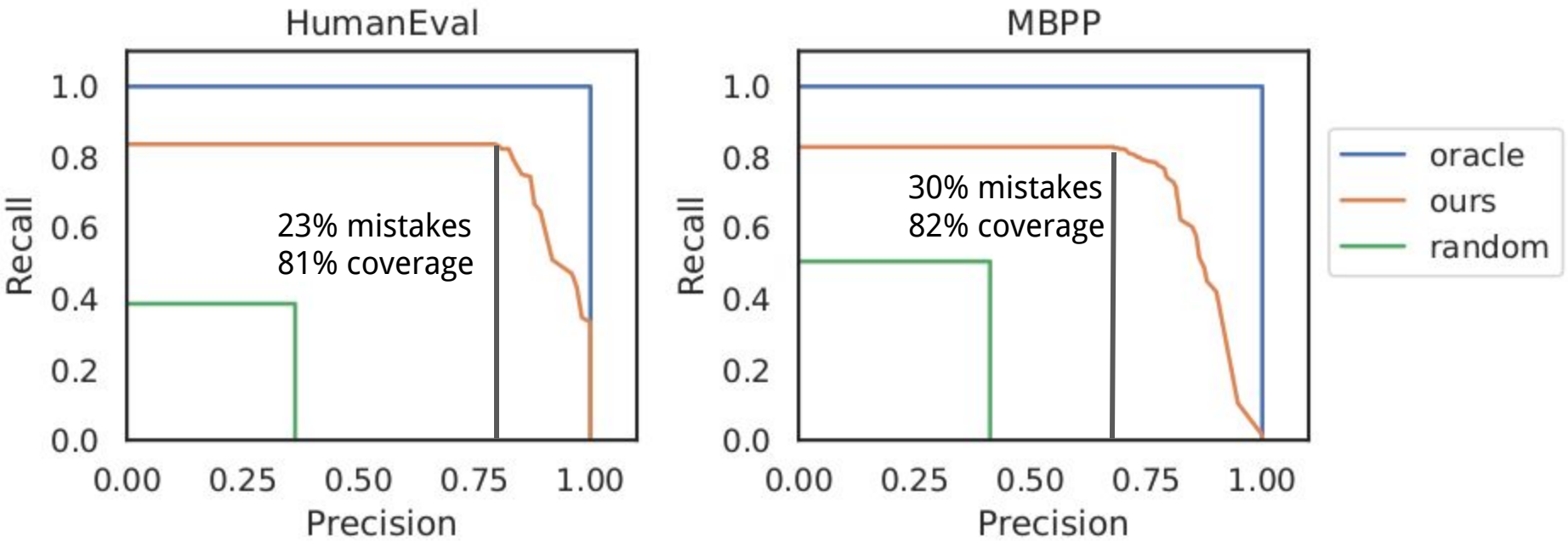
Solve fewer problems overall, but propose fewer buggy solutions

Low Threshold:

Broad coverage but inaccurate precision

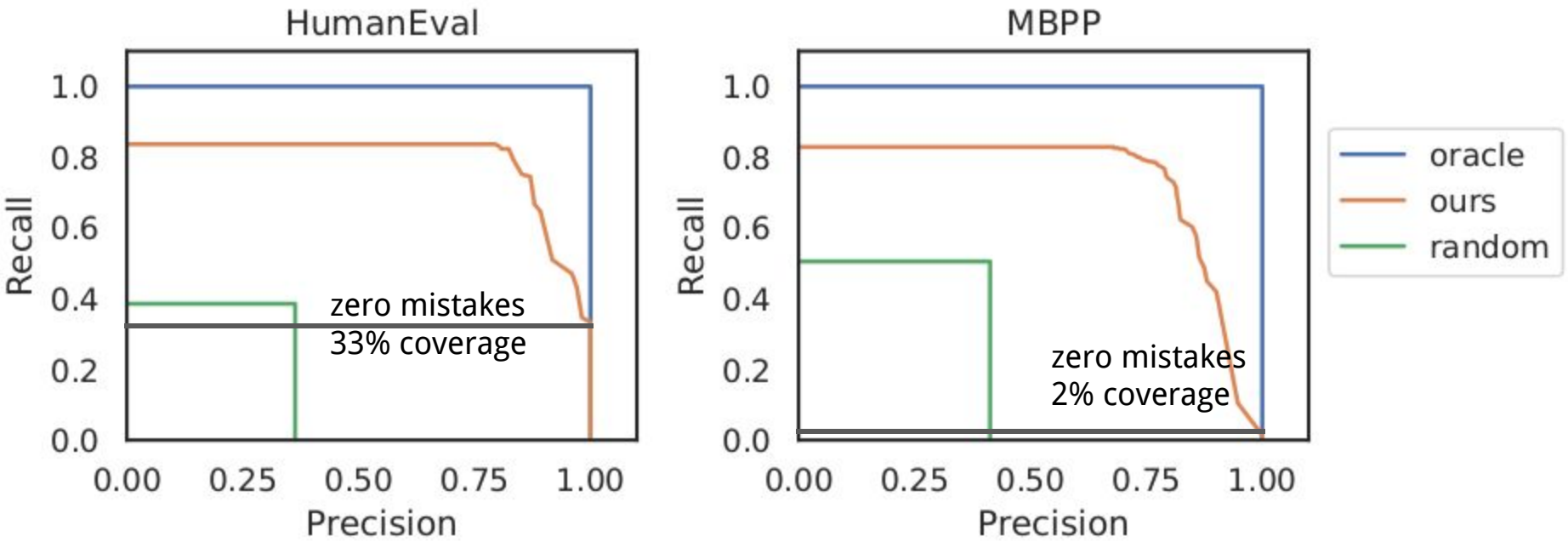
Solve more problems overall, but also make more mistakes

Tradeoffs



Precision: % of synthesized programs which do the right thing
Recall: problems we solve / problems we could have solved

Tradeoffs

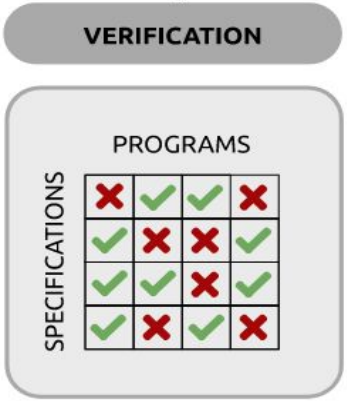


Precision: % of synthesized programs which do the right thing
Recall: problems we solve / problems we could have solved

Reality Check:

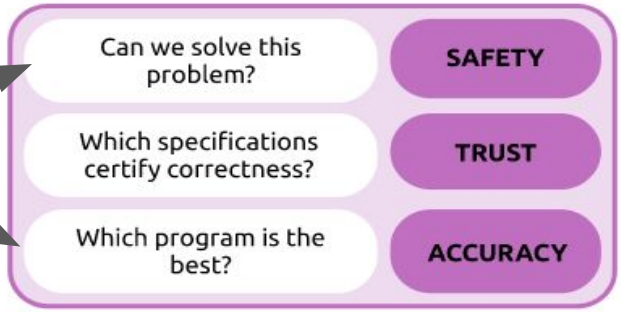
No neural net english->code system will achieve 0% errors

Objectives



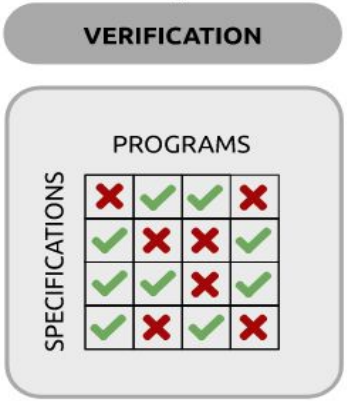
Pr[program is correct]

$$\max_{\text{prog}} \Pr[\text{ prog is correct }] > \text{threshold}$$

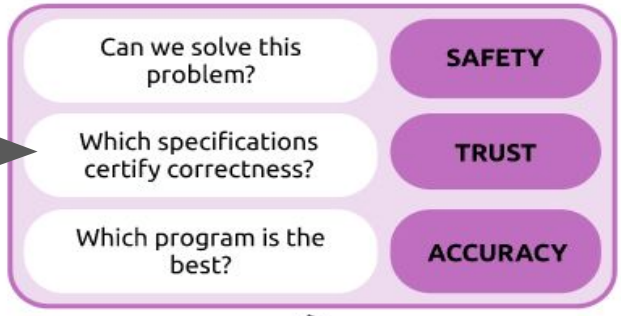


$$\operatorname*{argmax}_{\text{prog}} \Pr[\text{ prog is correct }]$$

Objectives



Pr[program is correct]



PROGRAM

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
```

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
```

Certifying (in)correctness

Think `program` is correct

How to communicate what `program` does so that the user can accept/reject it?

$$\text{program} \vdash \text{specification}$$

Certifying w/ random specifications

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

Certifying w/ random specifications

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

RANDOM INPUT/ OUTPUT

```
assert is_bored("I love this weather.") == 1
```

Certifying w/ random specifications

PROGRAM

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    return [x * i for i, x in enumerate(xs) if i != 0]
```

Certifying w/ random specifications

PROGRAM

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    return [x * i for i, x in enumerate(xs) if i != 0]
```

RANDOM LOGICAL RELATION

```
def test_derivative(xs):
    """ Test function derivative().
    """
    # TODO
    pass

# run `test_derivative` on a new testcase
test_derivative([2, 3, 4, 10, -12])
```

Certifying (in)correctness

Think `program` is correct

How to communicate what `program` does so that the user can accepted/reject it?

$$\underset{\text{prog} \vdash \text{spec}}{\text{argmax}} \ \Pr[\text{prog} \mid \text{spec}]$$

Joint distribution over programs and specifications
Uniform, except that program has to entail spec

Certifying (in)correctness

Think `program` is correct

How to communicate what `program` does so that the user can accepted/reject it?

$$\underset{\text{prog} \vdash \text{spec}}{\text{argmin}} \; |\{\text{prog}' : \text{prog}' \vdash \text{spec}\}|$$

Pick the thing which is true about the program
But which is not true about most other programs
“distinguishing”, “selective”

Certifying w/ selective/distinguishing specifications

PROGRAM

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    return [x * i for i, x in enumerate(xs) if i != 0]
```

RANDOM LOGICAL RELATION

```
def test_derivative(xs):
    """ Test function derivative().
    """
    # TODO
    pass

# run `test_derivative` on a new testcase
test_derivative([2, 3, 4, 10, -12])
```

Certifying w/ selective/distinguishing specifications

PROGRAM

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    return [x * i for i, x in enumerate(xs) if i != 0]
```

DISTINGUISHING LOGICAL RELATION

```
def test_derivative(xs: list):
    """ Given an input `xs`, test whether the function
`derivative` is implemented correctly.
    """
    ys = derivative(xs)
    assert len(ys) == len(xs) - 1
    for i in range(len(ys)):
        assert ys[i] == xs[i+1] * (i + 1)

# run `test_derivative` on a new testcase
test_derivative([3, 1, 2, 4, 5])
```

RANDOM LOGICAL RELATION

```
def test_derivative(xs):
    """ Test function derivative().
    """
    # TODO
    pass

# run `test_derivative` on a new testcase
test_derivative([2, 3, 4, 10, -12])
```

Certifying w/ selective/distinguishing specifications

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

RANDOM INPUT/ OUTPUT

```
assert is_bored("I love this weather.") == 1
```

Certifying w/ selective/distinguishing specifications

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

DISTINGUISHING INPUT/ OUTPUT

```
assert is_bored("I have no idea what I'm doing") == 2
```

RANDOM INPUT/ OUTPUT

```
assert is_bored("I love this weather.") == 1
```

Speculyzer

Synthesizer that creates specifications

Precision by backing off when it can’t solve a problem

Trust by constructing certificates of (in)correctness

What could trust unlock?

[ellisk42/ec] Bump protobuf from 3.8.0 to 3.15.0 (PR #90) Inbox ×

dependabot[bot] <notifications@github.com> Unsubscribe
to ellisk42/ec, Subscribed

This automated pull request fixes a security vulnerability (high severity).

Learn more about Dependabot security updates.

Bumps protobuf from 3.8.0 to 3.15.0.

Release notes

Sourced from protobuf's releases.

<science_fiction>

This repository | Search or type a command | Explore | Gist | Blog | Help | dsbaars

dsbaars / Project-XChain

Watch 6 | Unstar 1 | Fork 0

Browse Issues | Milestones | New Issue

Everyone's Issues 62
Assigned to you 0
Created by you 26
Mentioning you 0

No milestone selected

Labels
bug 37
critical 8
enhancement 10
high priority 26
invalid 5
low priority 2
medium prority 27

0 Open | 62 Closed | Sort: Newest | 1 2 3

Reopen | Label | Assignee | Milestone

Force currency-numbers in negotiation table bug critical high priority verified #62
Opened by dsbaars 6 days ago 1 comment

After clicking "I have read it", disable button enhancement medium prority verified #61
Opened by dsbaars 6 days ago 1 comment

Create Game, make duration field required bug medium prority verified #60
Opened by dsbaars 6 days ago 1 comment

profit forecast calculation bug bug high priority invalid verified #59
Opened by #aptwohanded 6 days ago 3 comments

Layout issues 2 and 2 point decimal rounding bug usability #58
Opened by #aptwohanded 6 days ago 1 comment

Chat does not scroll down when watching game as chairman #57
bug medium prority verified
Opened by dsbaars 6 days ago 8 comments

</science_fiction>

<science_fiction>

Add toString implementation #17

Edit

Open maxjacobson wants to merge 1 commit into `master` from `fix-git-tag-descriptions`

Conversation 0 | Commits 1 | Files changed 3

Changes from all commits | Jump to... | +20 −4

Unified | Split | Review changes

20 src/main/java/com/github/koraktor/mavanagaiata/git/GitTagDescription.java | 88.24% cov | 8 issues | View

```
@@ -67,9 +67,23 @@ public boolean isTagged() {
67 67      *
68 68      * @return The string representation of this description
69 69      */
70     -   @Override
71     -   public String toString() {
72     -     return "TODO: implement this method";
```

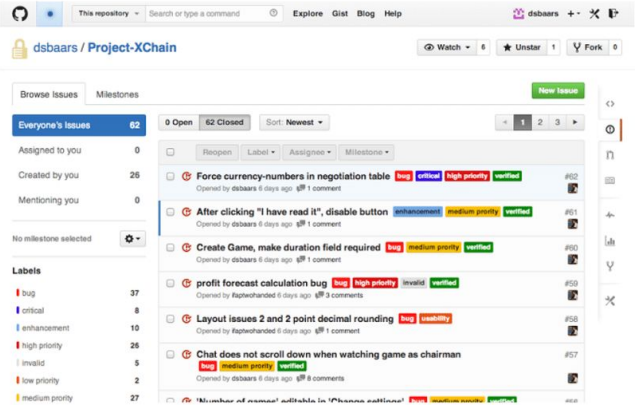
Method `toString` has a Cognitive Complexity of 7 (exceeds 5 allowed). Consider refactoring. ...

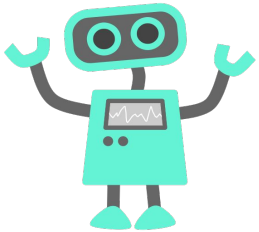
```
70 +   @Override
71 +   public String toString() {
72 +       if (this.nextTag == null) {
73 +           return this.abbreviatedCommitId;
74 +       } else if (this.distance == 0) {
75 +           return this.nextTag.getName();
76 +       } else {
```

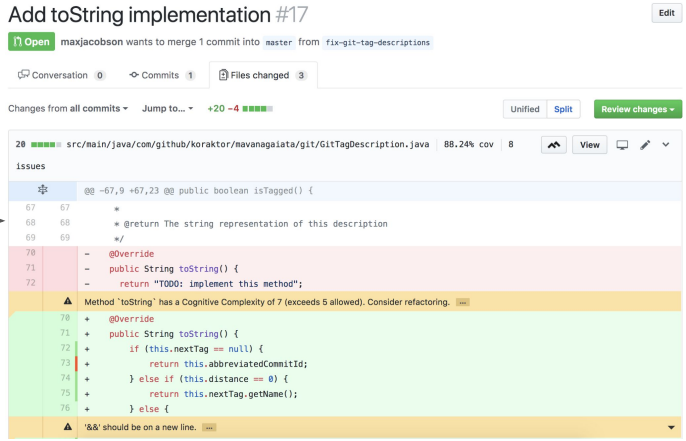
'&&' should be on a new line. ...

</science_fiction>

<science_fiction>







</science_fiction>

Challenges

Neural language models aren't _that_ good at programming (remember "oracle"?)

Execution can be hard

Verification can be hard

Rich space of specification languages with different tradeoffs:

which compose best with neural models?