

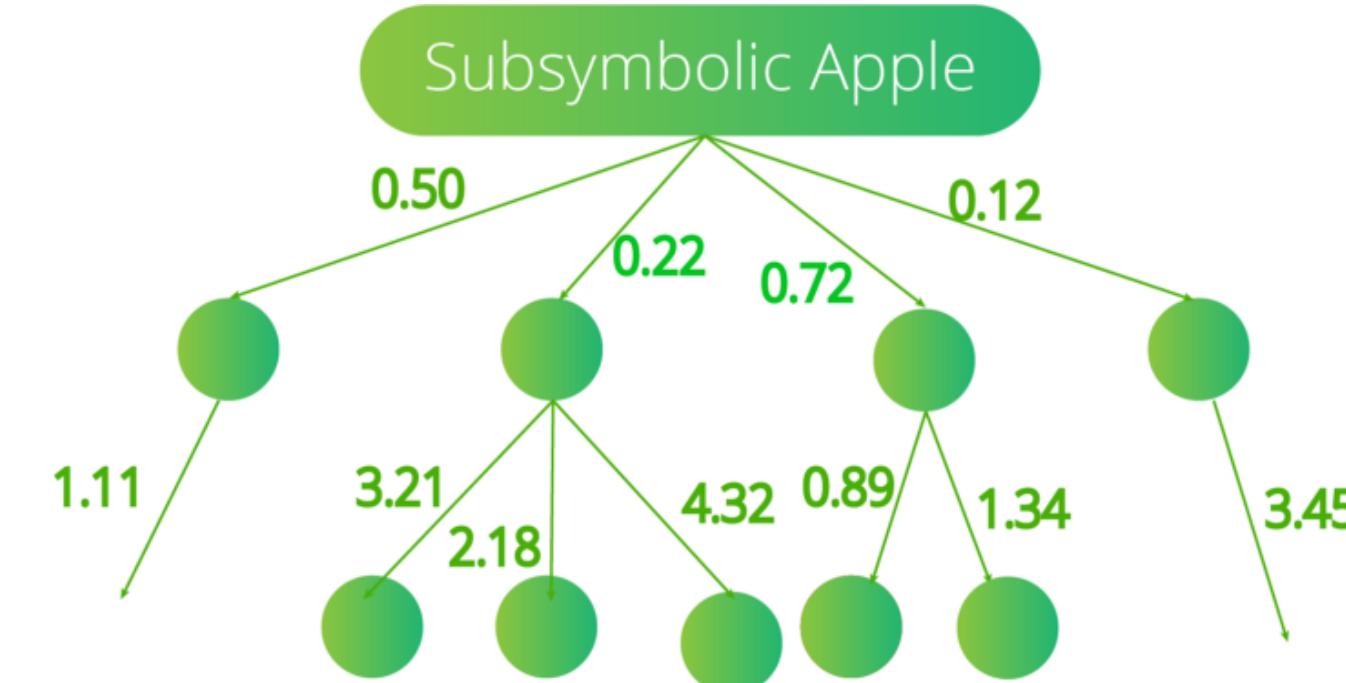
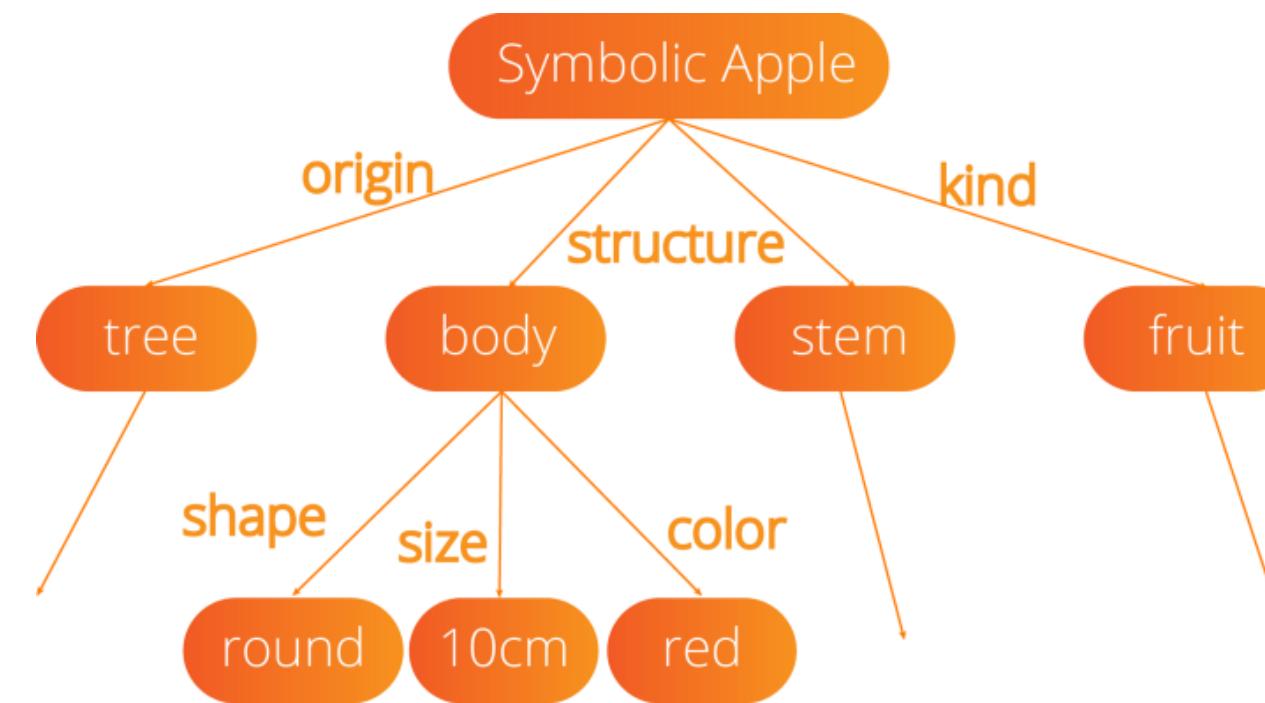
# Program Reasoning

## 16. Neuro-Symbolic AI

Kihong Heo

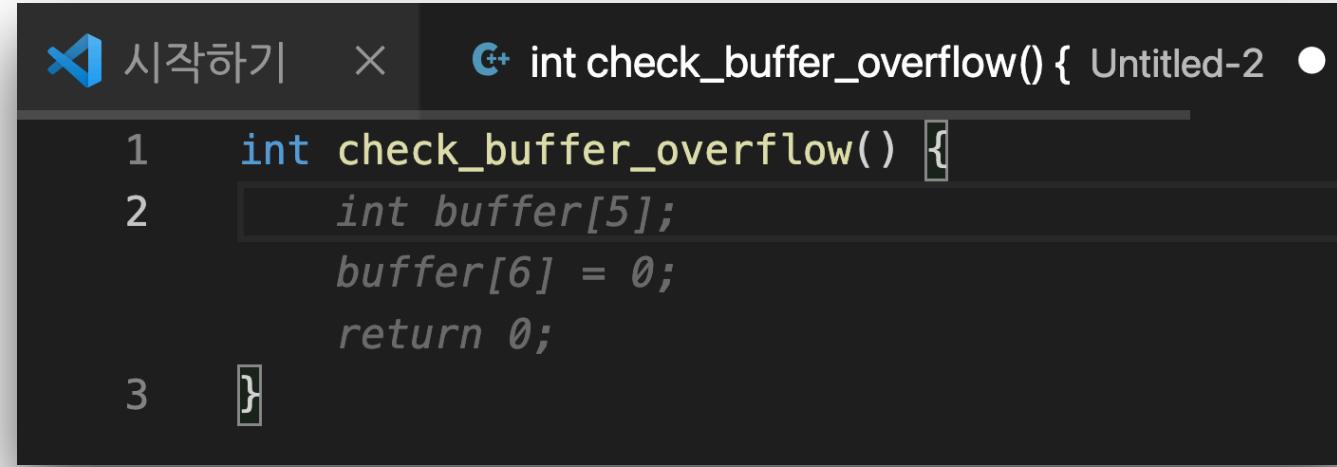


# Two Major Camps of AI



- Symbolist
- 1950s - 1980s
- High-level symbolic representation
- Human-readable & interpretable
- Logic- & search- based
- Connectionist
- 1980s - present
- Low-level neural representation
- Efficient learning
- Differentiation-based

# Artificial Ignorance (Then)



```
1 // read an image file
2 int read_file() {
3     FILE *fp;
4     int i, j, k;
5     int n;
6     int m;
7     int nc;
8     int nr;
9     int nb;
10    int nc_max;
11    int nr_max;
12    int nb_max;
13    int nc_min;
14    int nr_min;
15    int nb_min;
16    int nc_avg;
17    int nr_avg;
18    int nb_avg;
19    int nc_sum;
20    int nr_sum;
21    int nb_sum;
22    int nc_var;
23    int nr_var;
24    int nb_var;
25    int nc_std;
26    int nr_std;
27    int nb_std;
28    int nc_med;
29    int nr_med;
30    int nb_med;
31    int nc_mode;
32    int nr_mode;
33    int nb_mode;
34    int nc_min_index;
35    int nr_min_index;
36    int nb_min_index;
37    int nc_max_index;
38    int nr_max_index;
39    int nb_max_index;
40    int nc_med_index;
41    int nr_med_index;
42    int nb_med_index;
43    int nc_mode_index;
44    int nr_mode_index;
45    int nb_mode_index;
46    int nc_sum_index;
47    int nr_sum_index;
48    int nb_sum_index;
49    int nc_var_index;
50    int nr_var_index;
51    int nb_var_index;
52    int nc_std_index;
53    int nr_std_index;
54    int nb_std_index;
55    int nc_avg_index;
56    int nr_avg_index;
57    int nb_avg_index;
58    int nc_min_index_index;
59    int nr_min_index_index;
60 }
```

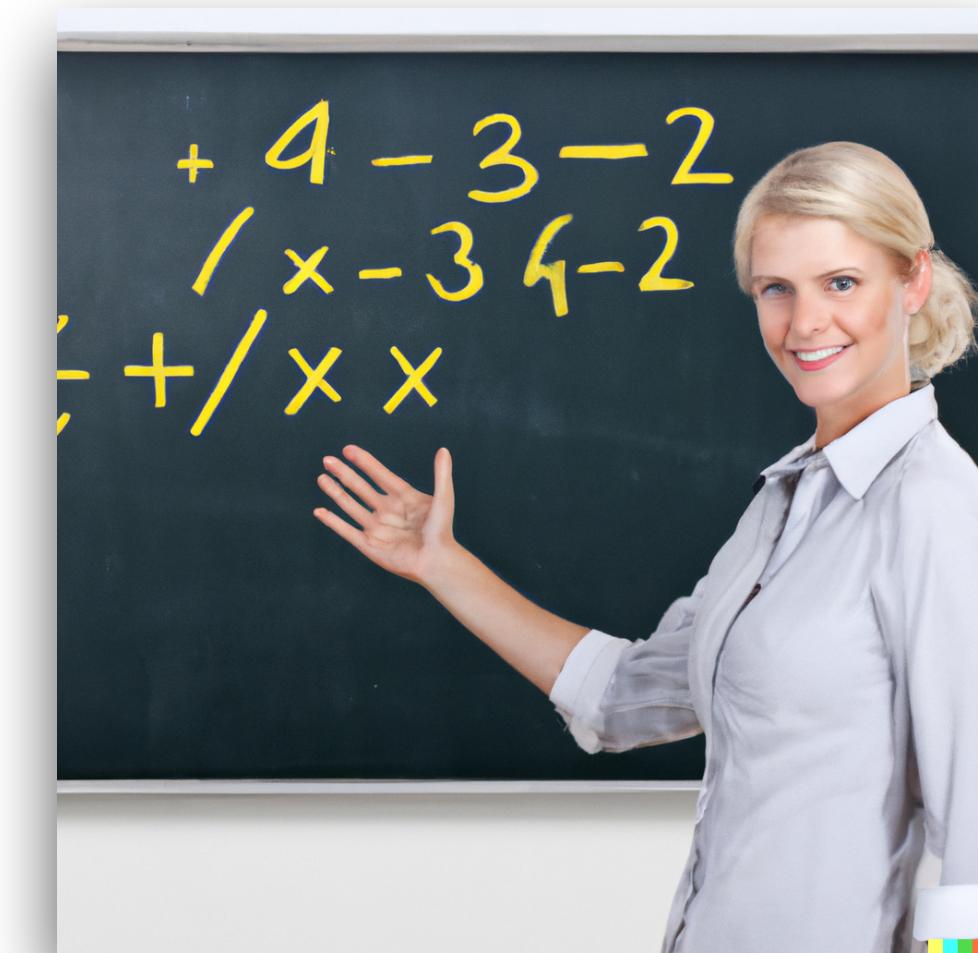
by Copilot

```
1 // read an image file
2 int read_file() {
3     FILE *fp;
4     int i, j, k;
5     int n;
6     int m;
7     int nc;
8     int nr;
9     int nb;
10    int nc_max;
11    int nr_max;
12    int nb_max;
13    int nc_min;
14    int nr_min;
15    int nb_min;
16    int nc_avg;
17    int nr_avg;
18    int nb_avg;
19    int nc_sum;
20    int nr_sum;
21    int nb_sum;
22    int nc_var;
23    int nr_var;
24    int nb_var;
25    int nc_std;
26    int nr_std;
27    int nb_std;
28    int nc_med;
29    int nr_med;
30    int nb_med;
31    int nc_mode;
32    int nr_mode;
33    int nb_mode;
34    int nc_min_index;
35    int nr_min_index;
36    int nb_min_index;
37    int nc_max_index;
38    int nr_max_index;
39    int nb_max_index;
40    int nc_med_index;
41    int nr_med_index;
42    int nb_med_index;
43    int nc_mode_index;
44    int nr_mode_index;
45    int nb_mode_index;
46    int nc_sum_index;
47    int nr_sum_index;
48    int nb_sum_index;
49    int nc_var_index;
50    int nr_var_index;
51    int nb_var_index;
52    int nc_std_index;
53    int nr_std_index;
54    int nb_std_index;
55    int nc_avg_index;
56    int nr_avg_index;
57    int nb_avg_index;
58    int nc_min_index_index;
59    int nr_min_index_index;
60 }
```

by Copilot



by Dall-E



by Dall-E

# Artificial Ignorance (Now)

## The hidden dangers of AI coding

How AI-generated code is changing cybersecurity — and what developers and “vibe coders” should expect.



Stan Kaminsky

October 10, 2025

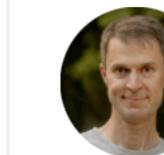
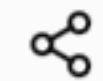
AI • CODING

### An AI-powered coding tool wiped out a software company’s database, then apologized for a ‘catastrophic failure on my part’

By Beatrice Nolan  
Tech Reporter

July 23, 2025, 7:22 AM ET

Add us on



Daniel Stenberg

curl CEO. Code Emitting Organism  
6개월 · 수정됨

...

That's it. I've had it. I'm putting my foot down on this craziness.

1. Every reporter submitting security reports on **#Hackerone** for **#curl** now needs to answer this question:

"Did you use an AI to find the problem or generate this submission?"

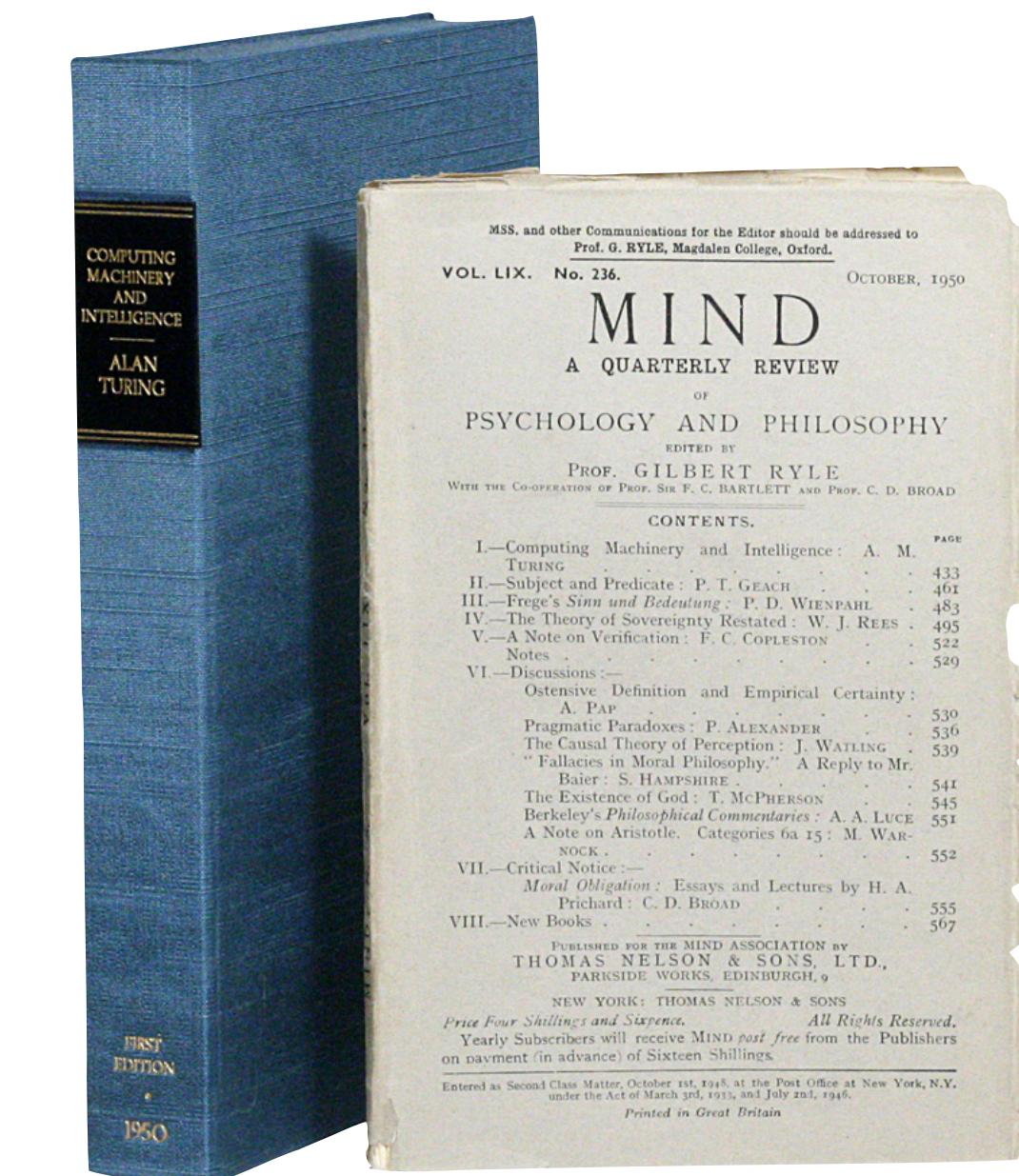
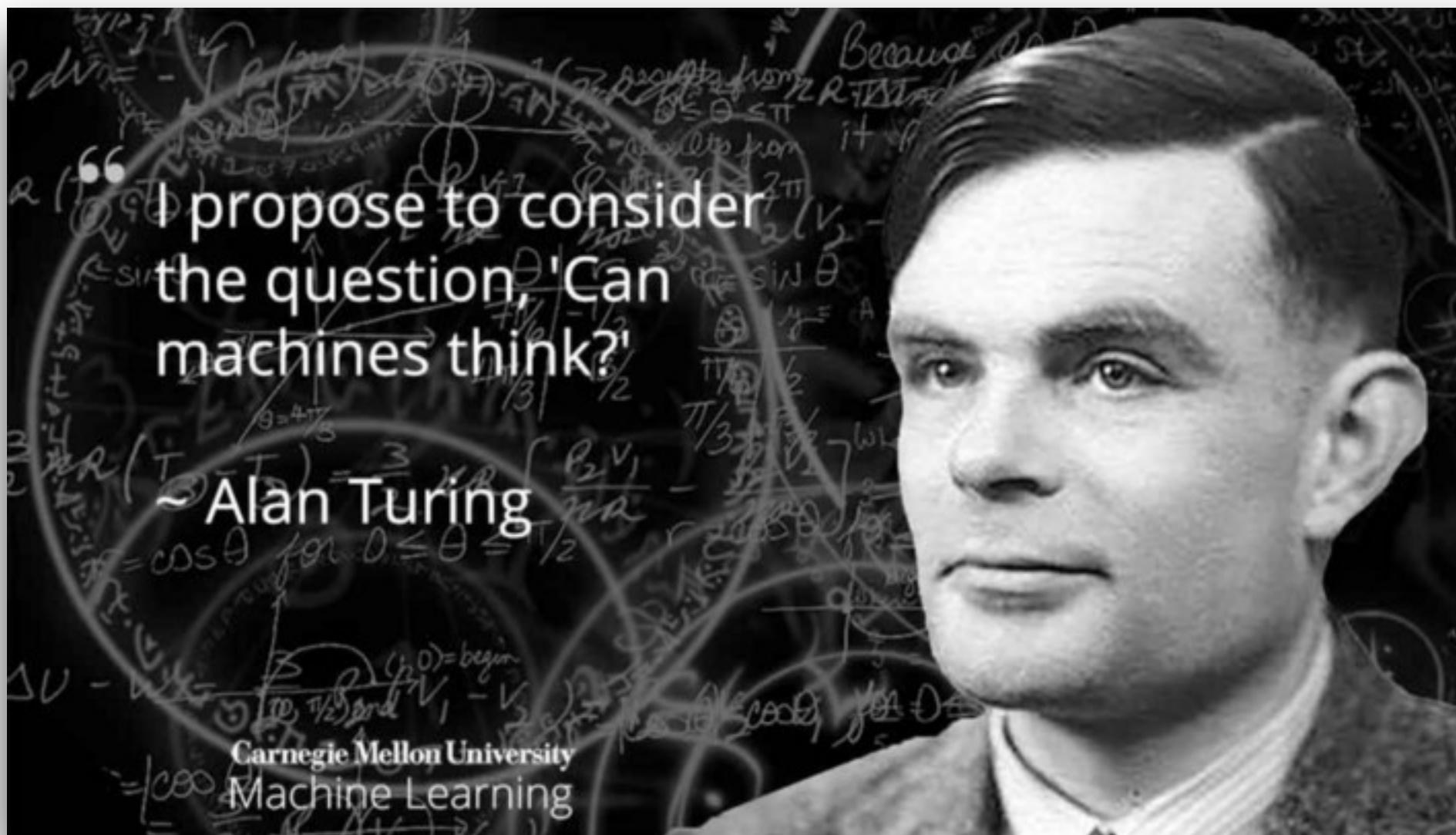
(and if they do select it, they can expect a stream of proof of actual intelligence follow-up questions)

2. We now ban every reporter INSTANTLY who submits reports we deem AI slop. A threshold has been reached. We are effectively being DDoSed. If we could, we would charge them for this waste of our time.

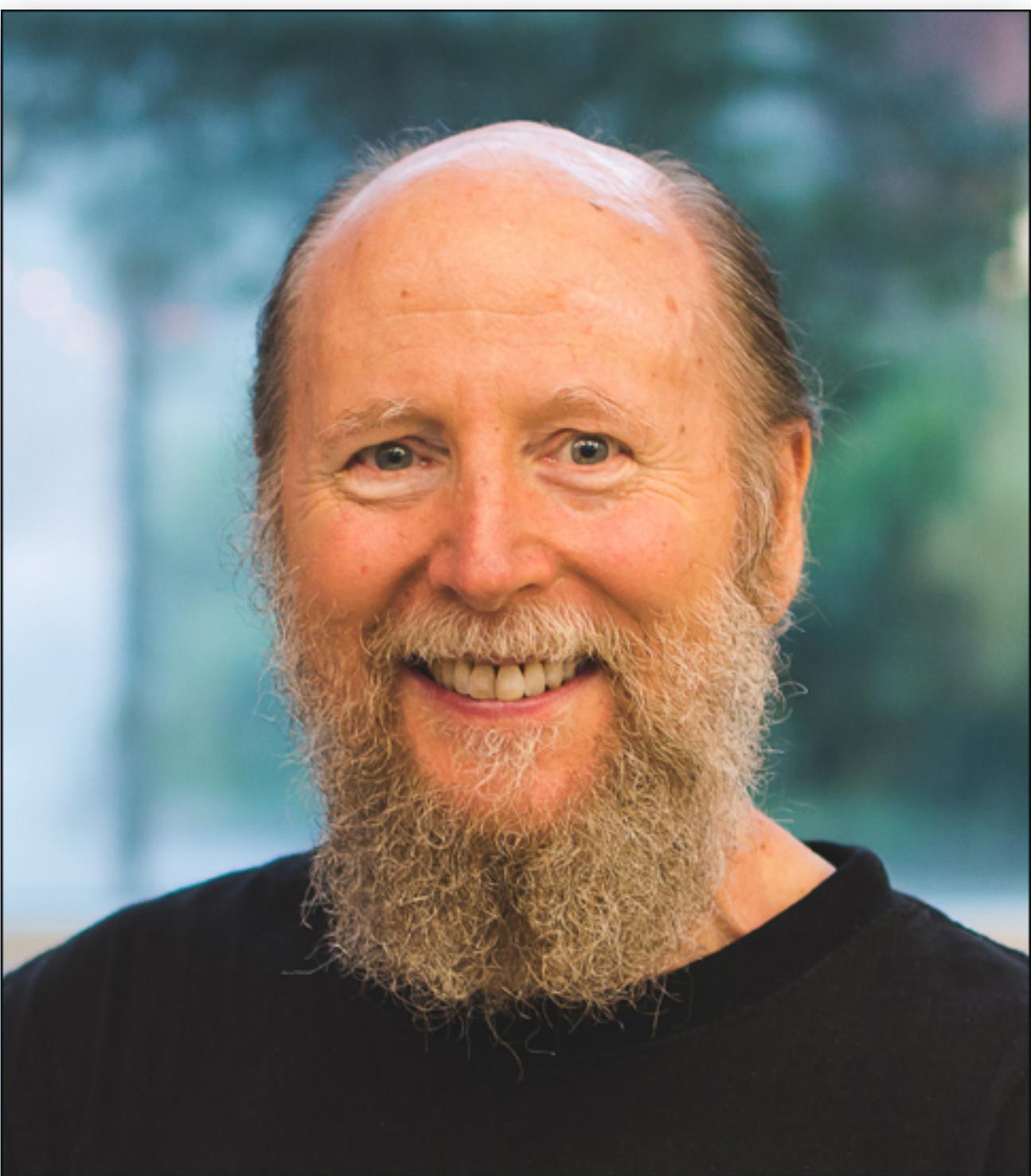
We still have not seen a single valid security report done with AI help.

# Program Synthesis & AI

- Turing's dream
- Program: explainable & executable & verifiable
- Future of AI/SW/CS?



# Verification and AI



Richard Sutton  
2024 Turing Award Winner

***“An AI system can create and maintain knowledge only to the extent that it can verify that knowledge itself.”***

- “Verification, The Key to AI”, 2001



Then, what is our mission?

# Neuro-Symbolic AI

- What is the future of AI (or programming)?
  - Combining the strengths of the two camps
- Two directions
  - Symbolic engine + neural guide: Program synthesis using numerical optimization
  - Neural engine + symbolic guide: Constrained decoding

# Inductive Logic Programming (ILP)

- A subfield of symbolic artificial intelligence
- Goal: given a dataset (inductive), infer a set of rules (logic programming)
- Synthesizing programs in logic programming languages
  - E.g., Prolog, Datalog

# Example

- Parent

**Dataset**

parent(a,b)	parent(a,c)	parent(d,b)
father(a,b)	father(a,c)	mother(d,b)
male(a)	female(c)	female(d)

**Learned Rules**

```
father(X,Y) :- parent(X,Y) & male(X)
mother(X,Y) :- parent(X,Y) & female(X)
```

- Transitive closure

**Dataset**

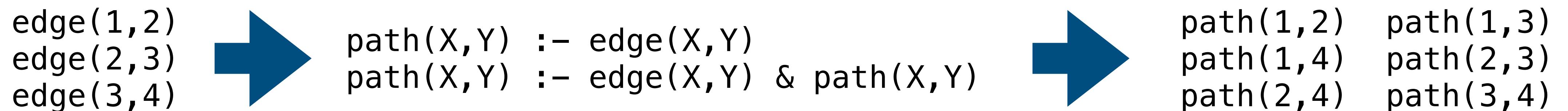
edge(1,2)	edge(2,3)	edge(3,4)
path(1,2)	path(1,3)	path(1,4)
path(2,3)	path(2,4)	path(3,4)

**Learned Rules**

```
path(X,Y) :- edge(X,Y)
path(X,Y) :- edge(X,Z) & path(Z,Y)
```

# Datalog Programs

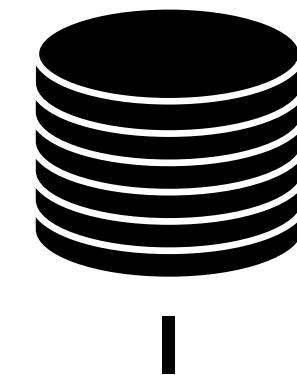
- A set of Horn clause rules ( $X_1 \wedge X_2 \wedge \dots \wedge X_n \rightarrow H$ )
- Input & output: a set of tuples
- Applications: big data analysis, network protocol, program analysis, etc



# Datalog Program Synthesis

- Given a set of candidate rules, find a subset that is consistent with a given examples
  - A typical combinatorial optimization problem (i.e., NP-hard)
- In this lecture, we assume a set of candidates is predefined

edge(1,2)  
edge(2,3)  
edge(3,4)

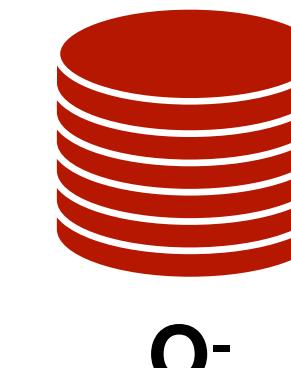


✓ path(x,y) :- edge(x,y).  
✗ path(x,x) :- edge(x,y).  
✓ path(x,z) :- edge(x,y), path(y,z).  
✗ path(x,y) :- path(y,x).

...



path(1,2)    path(1,3)  
path(1,4)    path(2,3)  
path(2,4)    path(3,4)



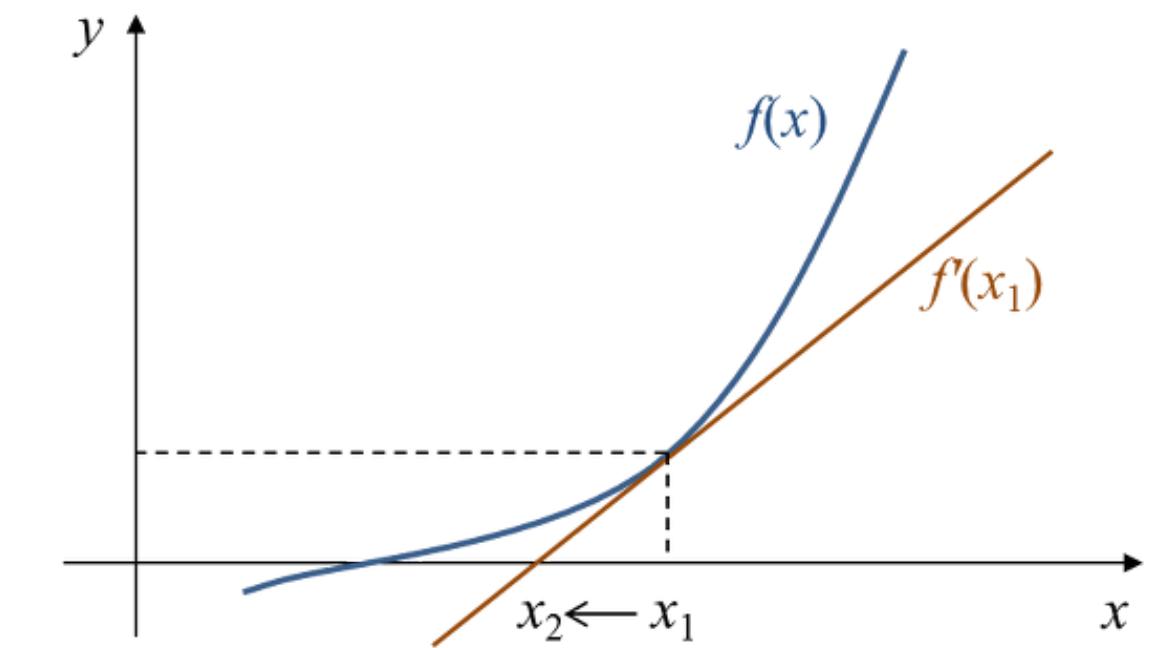
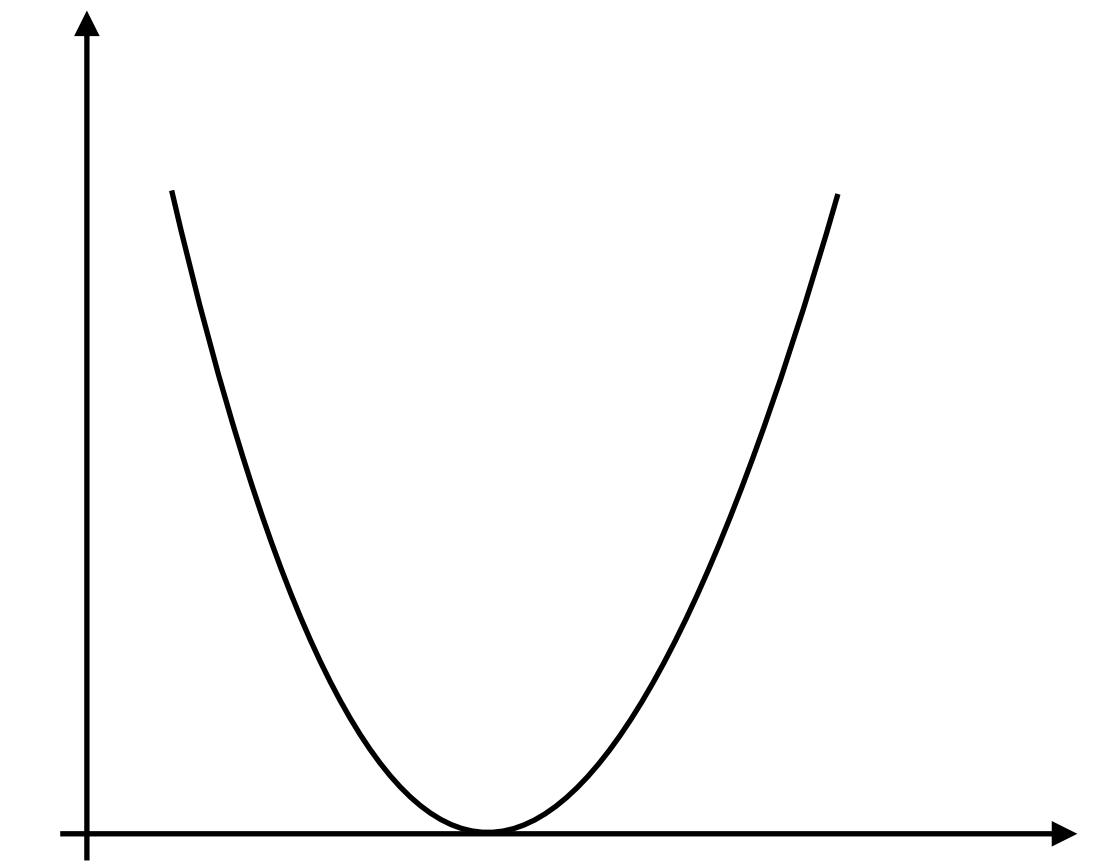
path(2,1)    path(3,1)  
path(4,1)    path(4,2)

# Challenges

- Huge search space
  - E.g., # of possible combinations of 50 candidate rules?
- If a wrong rule is chosen? The program produces a wrong tuple
- If a correct rule is missed? The program does not produce a correct tuple
- That is, how to find a point where the “loss” is zero?
  - Loss: the amount of derived wrong tuples + underived correct tuples
- Can the synthesizer learn from failures?

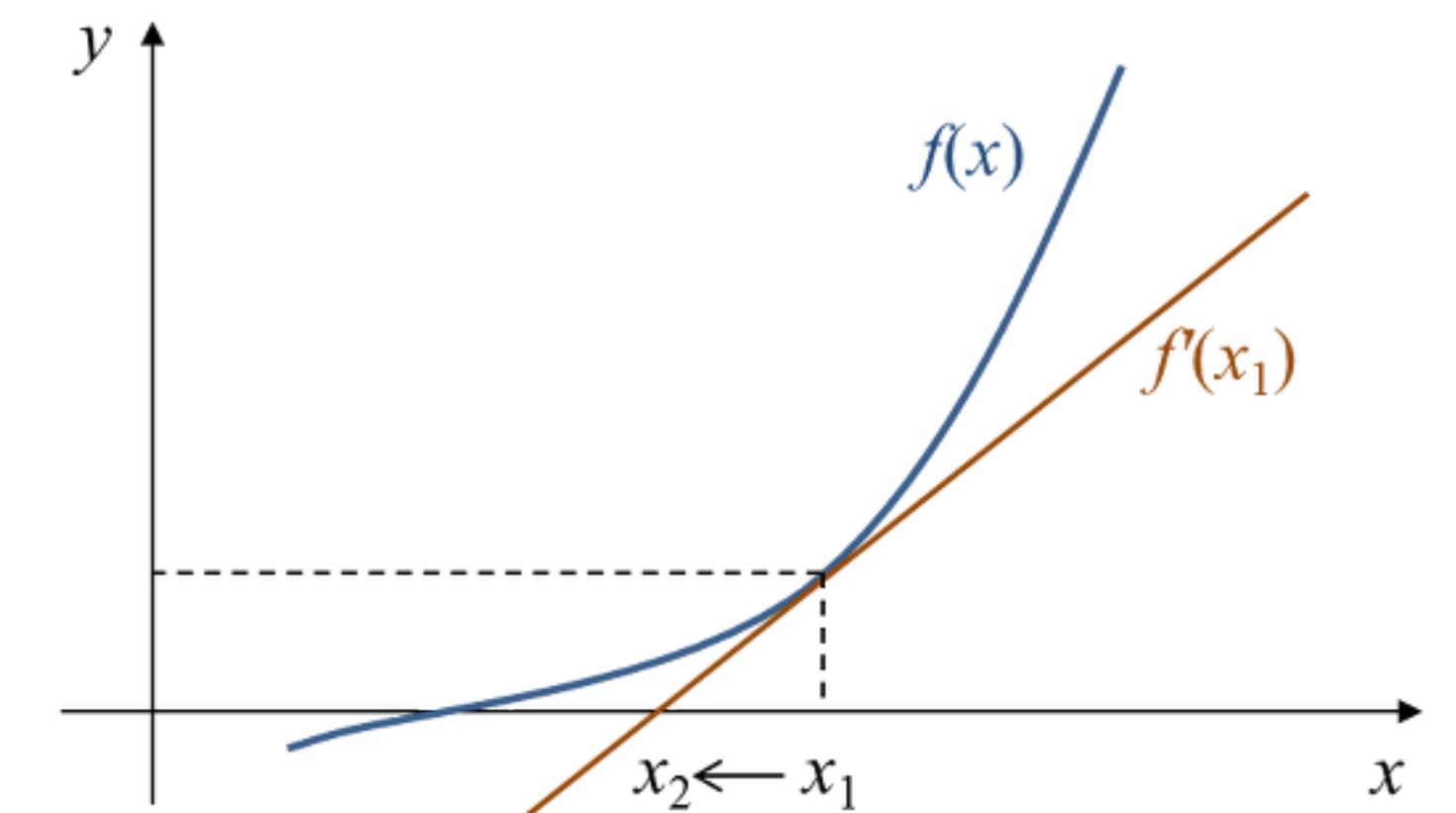
# Analogy

- In other fields of mathematics, how to find a zero point?
- Enumerative approach: general but not efficient
  - E.g., 0, 1, 2, 3, 4...
- Algebraic approach: efficient but not general
  - E.g.,  $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$  for quadratic equation, but  $\geq 5$ -degree?
- Approximative approach: moderately efficient and general
  - E.g., Newton's method



# A Solution: Difflog\*

- A Datalog synthesis algorithm using numerical optimization
- Key idea: solving combinatorial optimization via numerical optimization
- Why numerical optimization? Many powerful algorithms exist!
  - E.g., Newton's method for differentiable loss functions
- Problem: Datalog programs are not differentiable



\*Si et al., Synthesizing Datalog Programs using Numerical Relaxation, IJCAI 2019

# Idea 1: Provenance

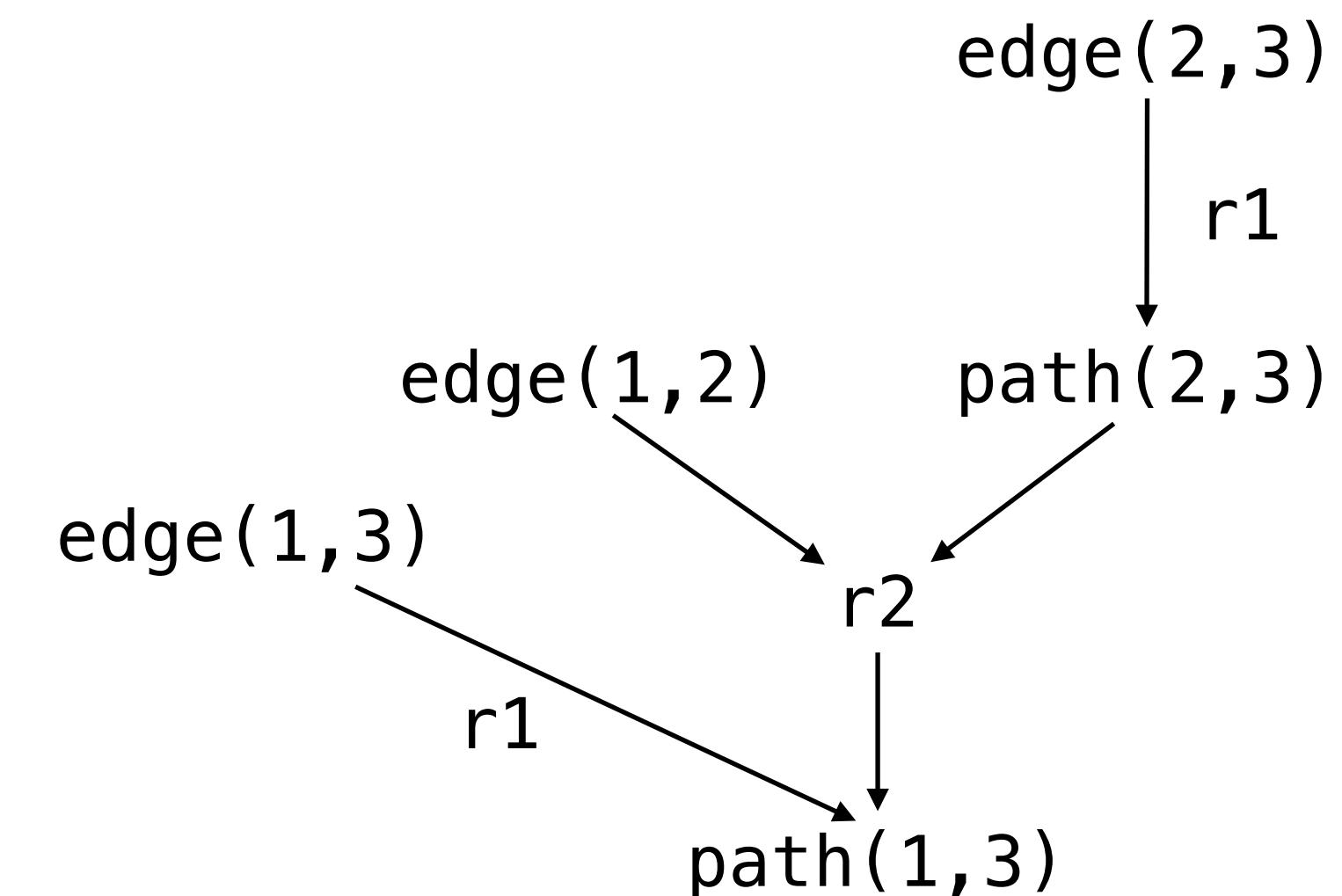
- Datalog programs produce provenance as well as output tuples
- Provenance: a proof that explains why a tuple is derived
  - If an undesired tuple is derived, we can see the reason

## Rules

```
r1: path(x,y) :- edge(x,y).  
r2: path(x,z) :- edge(x,y), path(y,z).
```

## Input tuples

```
{edge(1,2), edge(2,3), edge(1,3)}
```



# Idea 2: Continuous Semantics (1)

- Interpret Datalog programs (non-continuous function) as continuous functions
  - Existence of tuple  $\{0, 1\}$  to weight of tuple  $[0, 1]$
  - Each rule is associated with a weight
  - The weight of a tuple is computed using the weights of rules on the provenance
- Then, combinatorial optimization problem → numerical optimization problem
  - Many existing algorithms applicable

# Example (1)

**Parameters:**  $\vec{W}$

```

0.7 path(x,y) :- edge(x,y).
0.1 path(x,x) :- edge(x,y).
0.9 path(x,z) :- edge(x,y), path(y,z).
0.3 path(x,y) :- path(y,x).

```

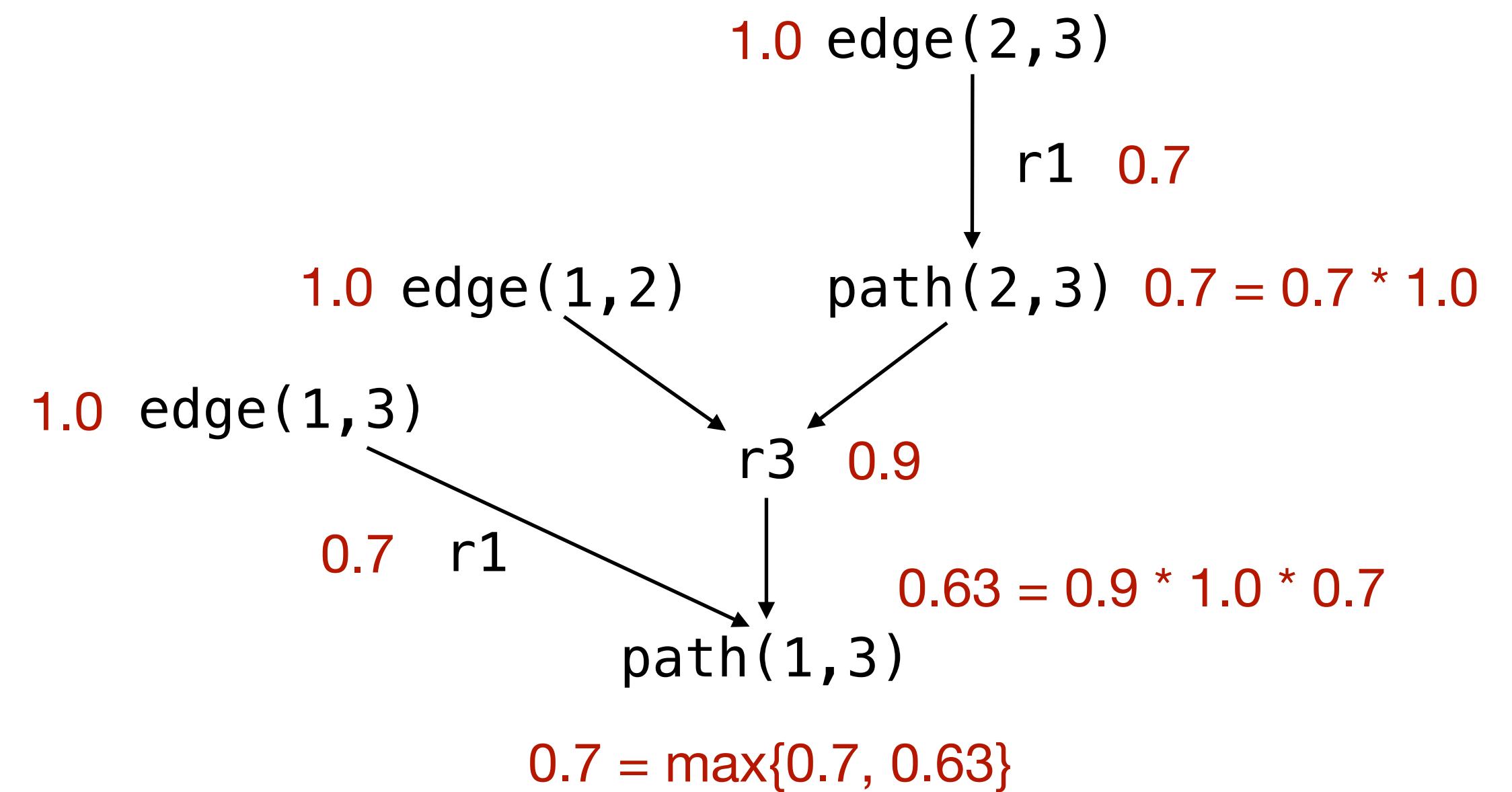
Input	edge(1,2)	edge(2,3)
Weight	1.0	1.0

**Discrete semantics**

$$v_t = \bigvee_g (v_{a_1} \wedge v_{a_2} \wedge \dots \wedge v_{a_k})$$

**Continuous semantics**

$$v_t = \max_g (w_g \times v_{a_1} \times v_{a_2} \times \dots \times v_{a_k})$$



# Example (2)

**Parameters:**  $\vec{W}$

```

0.7 path(x,y) :- edge(x,y).
0.1 path(x,x) :- edge(x,y).
0.9 path(x,z) :- edge(x,y), path(y,z).
0.3 path(x,y) :- path(y,x).

```

Input	edge(1,2)	edge(2,3)
Weight	1.0	1.0

Output	path(1,2)	path(2,3)	path(1,3)	path(1,1)	path(2,1)
Weight ( $v_t$ )	0.7	0.7	0.63	0.1	0.21
Expectation	1	1	1	0	0

**Discrete semantics**

$$v_t = \bigvee_g (v_{a_1} \wedge v_{a_2} \wedge \dots \wedge v_{a_k})$$

**Continuous semantics**

$$v_t = \max_g (w_g \times v_{a_1} \times v_{a_2} \times \dots \times v_{a_k})$$

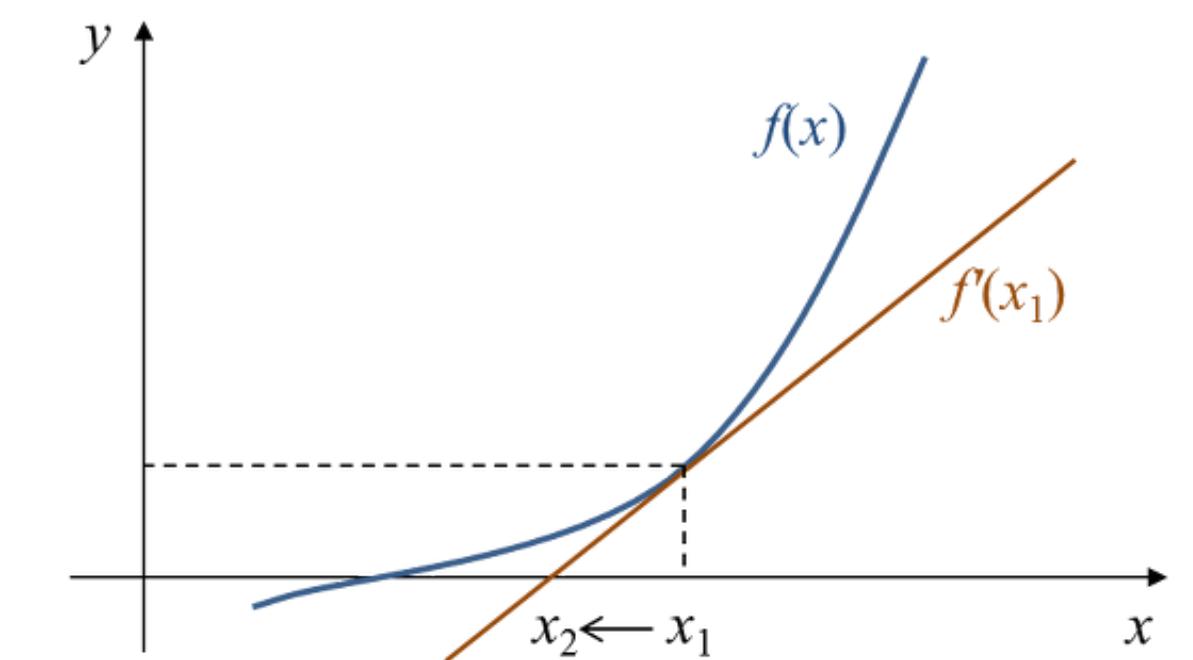
**Optimization problem:**  
find  $\vec{W}$  that minimizes the following loss

$$\text{loss} = \sum_{t \in \text{pos}} (1 - v_t)^2 + \sum_{t \in \text{neg}} (0 - v_t)^2$$

# Numerical Optimization

$$\text{loss} = \sum_{t \in pos} (1 - v_t)^2 + \sum_{t \in neg} (0 - v_t)^2$$

- In continuous semantics,  $v_t$  is a polynomial with  $w_r$  (differentiable)
- Then, the loss function is a polynomial with  $w_r$  (differentiable)
- Solve using a well-known algorithm (e.g., Newton's method)
  - loss is 0 = consistent with all the examples = a desired program

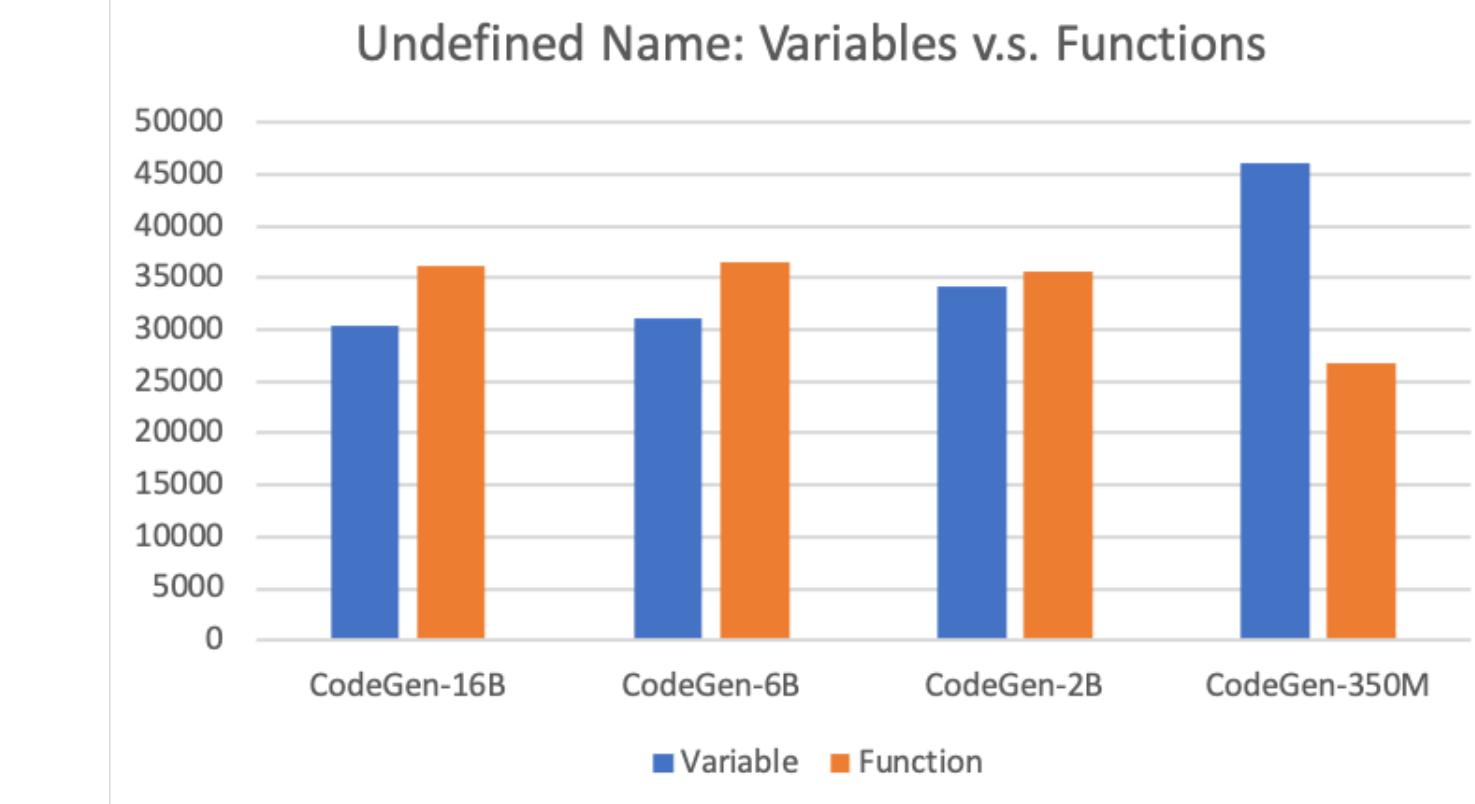
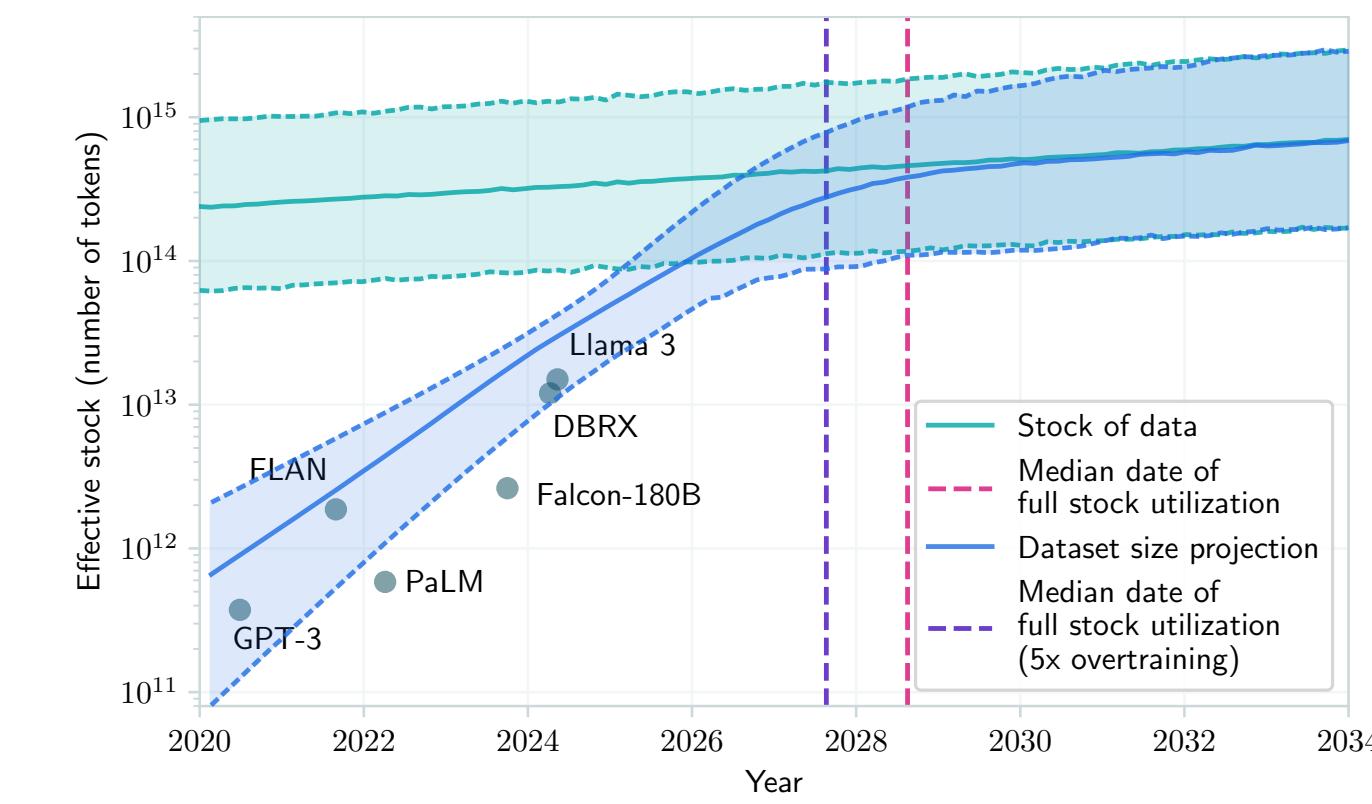
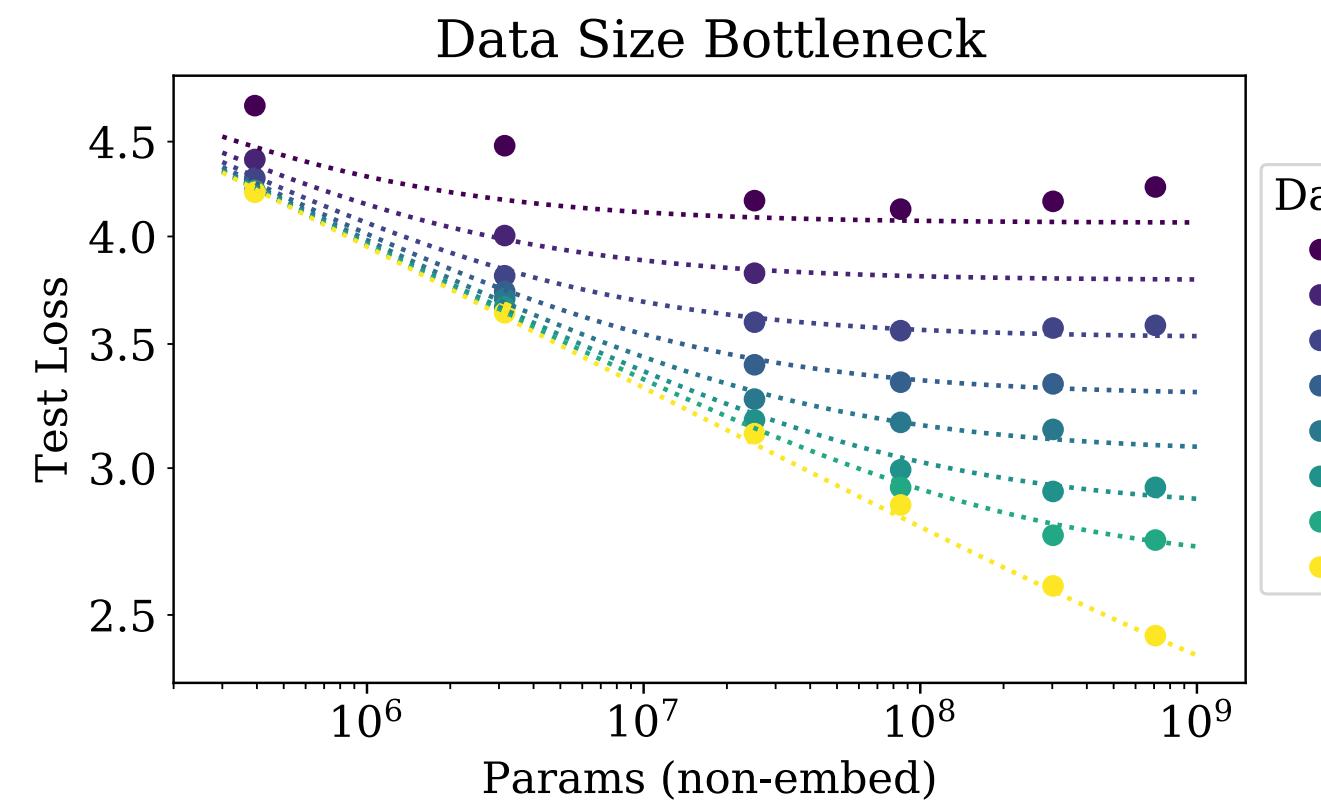


# Neuro-Symbolic AI

- What is the future of AI (or programming)?
    - Combining the strengths of the two camps
  - Two directions
    - Symbolic engine + neural guide: Program synthesis using numerical optimization
    - Neural engine + symbolic guide: Constrained decoding
-

# Problems of Language Models for Code

- No guarantee on important properties (e.g., syntax, semantics, safety, security, etc)
- More data needed for training larger models
- All human-generated data will be exhausted by 2026-2032
- Larger model does not always mean better performance



# Constrained Decoding

조건부 생성

- How to make LMs to generate *correct* outputs?
- Guide LLMs in generating content that satisfies a given constraint
  - Syntax: Json, XML, Python, OCaml, etc
  - Semantics: type, security vulnerability, etc
- Idea: attach a filter at the backend of LMs
  - Combination of LM and symbolic checker

# Language Models in a Nutshell

- $T$  is a finite set of tokens (vocabulary)
- $\text{LM} \in \text{String} \rightarrow \Pr(T)$ : given a prefix, return a probability distribution on  $T$

```
LMSynthesizer(prompt):  
    s := initialize(prompt)  
    while true do  
        v := LM(prompt + s)  
        t := Sample(v)  
        if t = EOS then break  
        s := s + t  
    return s
```

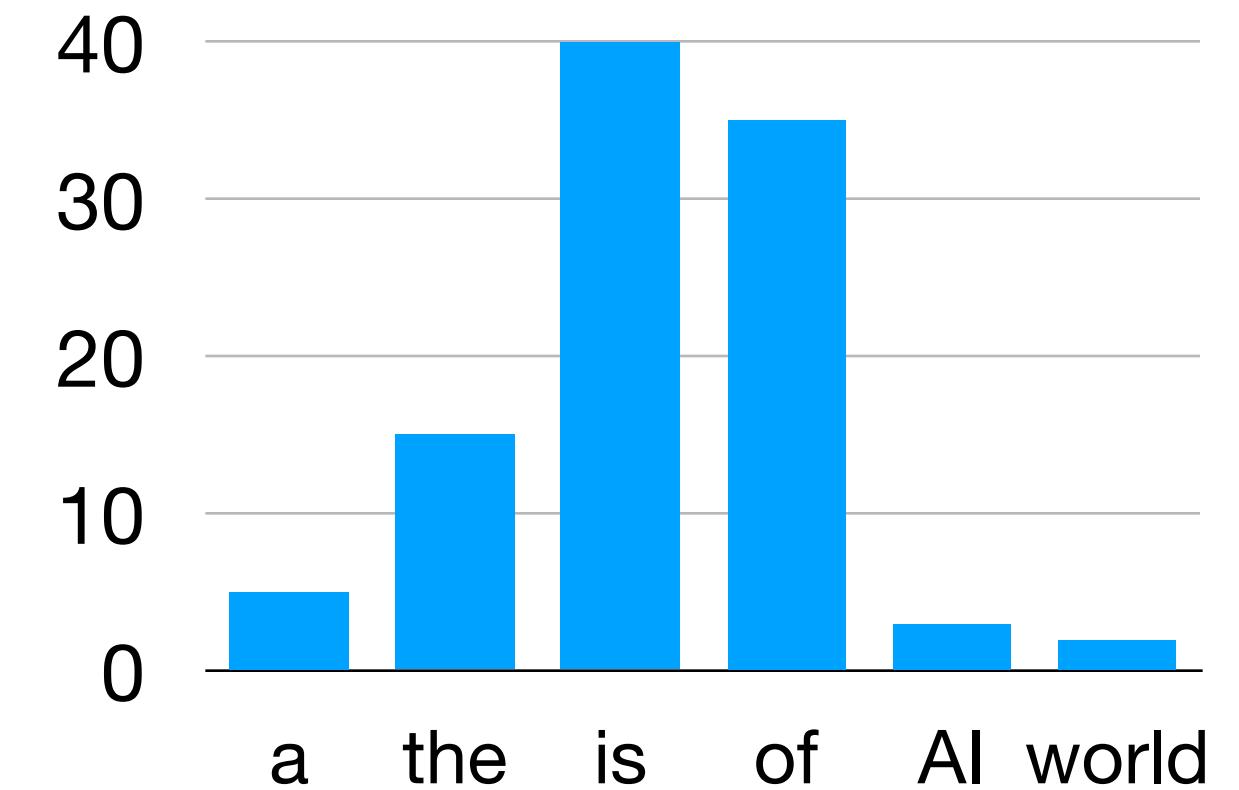
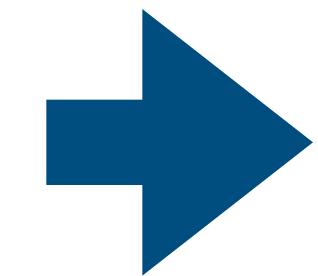
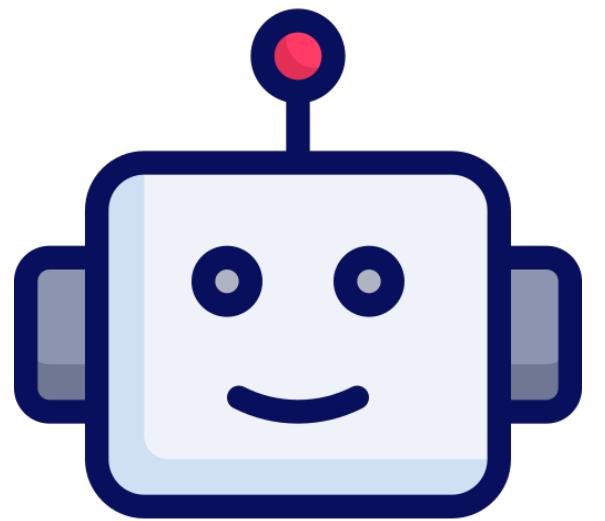
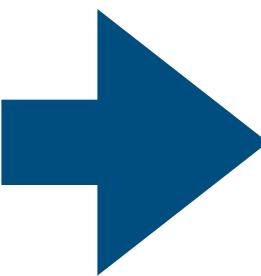
Prompt: “Which university is the best?”  
S0: “”  
S1: “The”  
S2: “The best”  
S2: “The best university”  
S3: “The best university is”

You've reached out the Free plan limit. Upgrade to Plus or try again after 10:00AM.

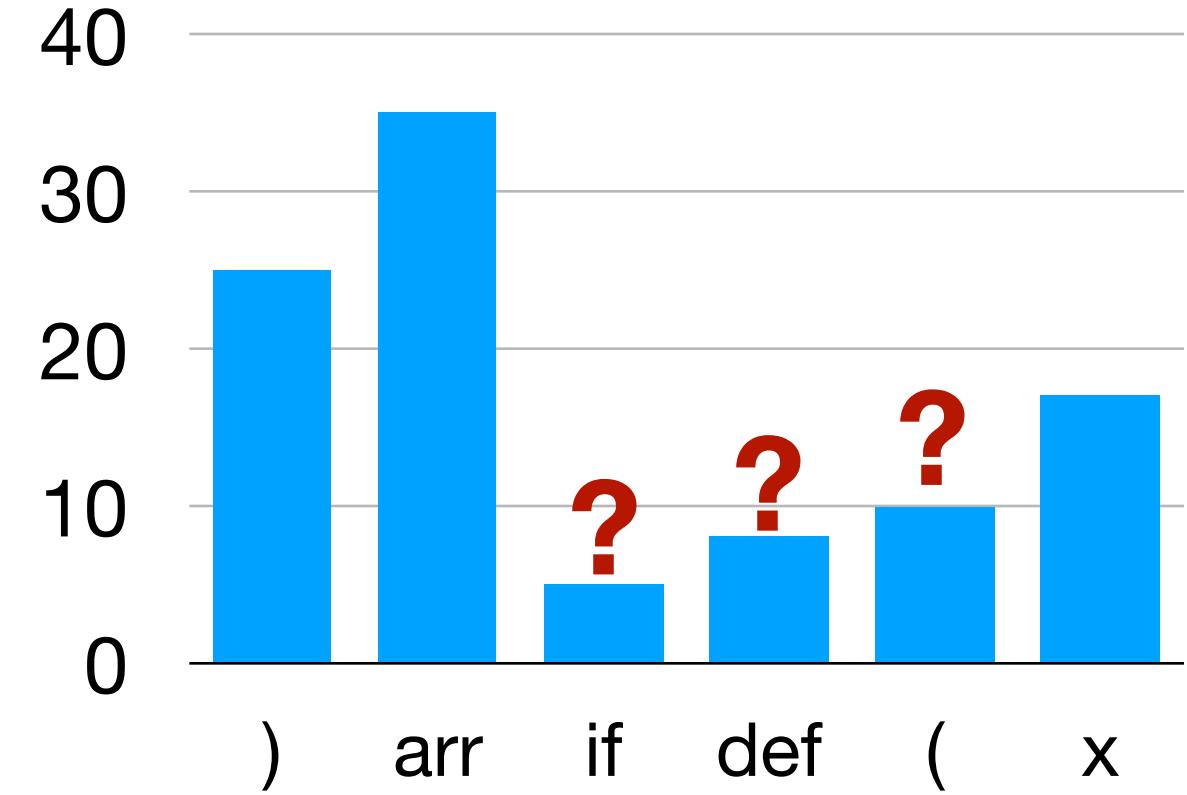
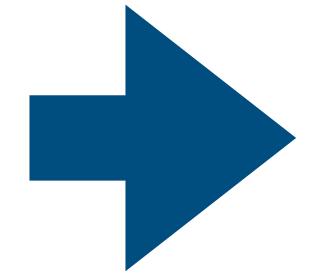
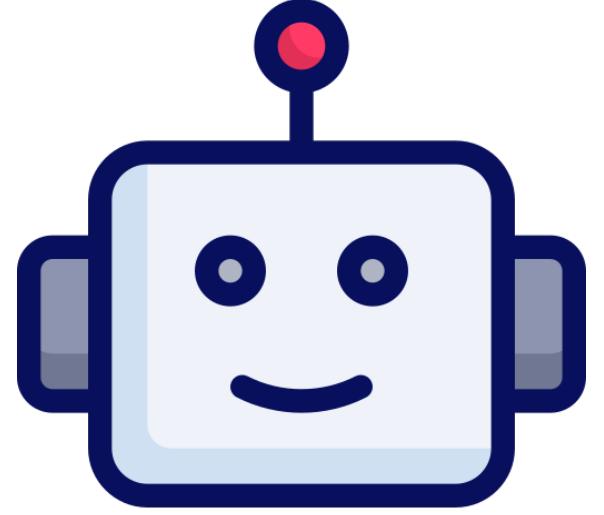
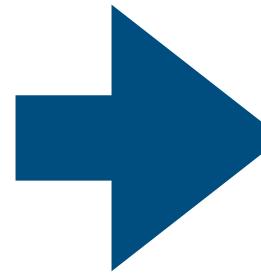
Get Plus

# Inference & Sampling

“The best university”

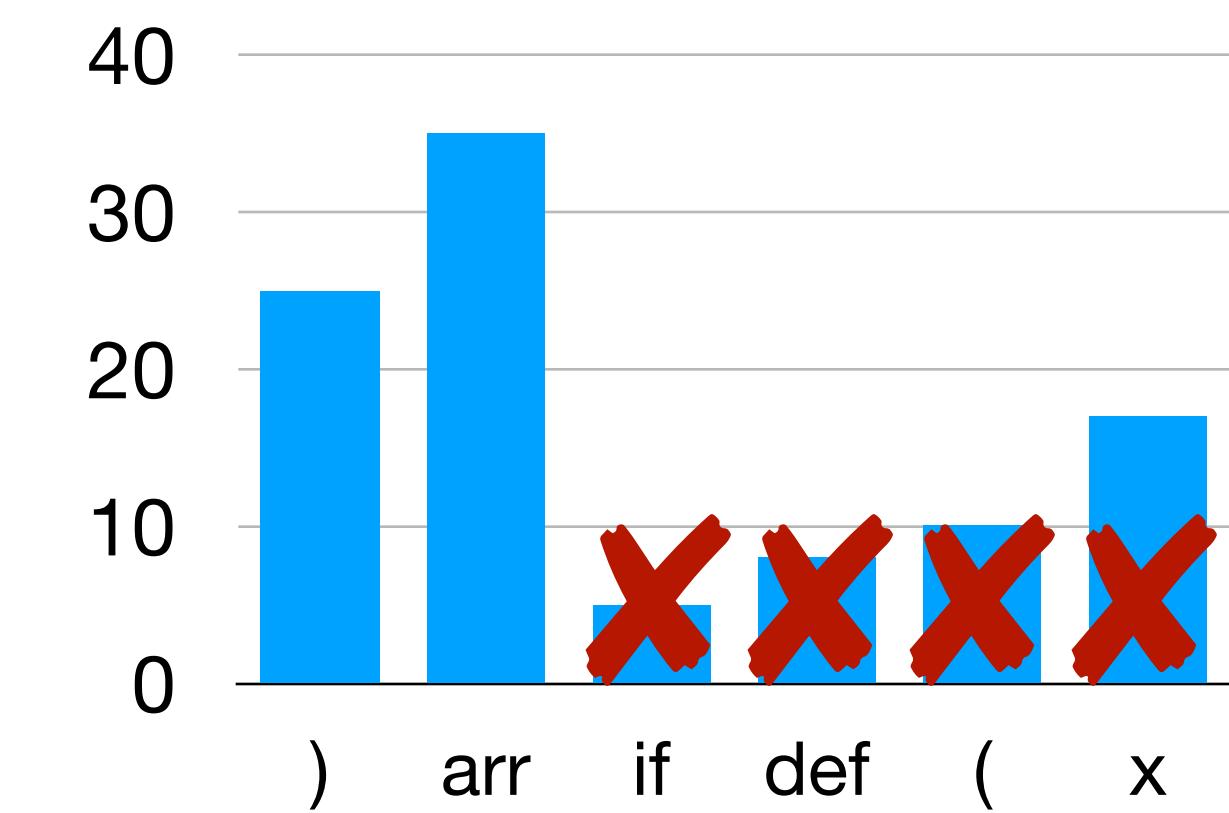


“def sort(“



# Grammar-Constrained Decoding

- Guide LM to generate grammatically correct text
- Idea: combine parser (i.e., syntactic analysis) with LM
  - Mask incorrect tokens from the distribution
  - Normalize the masked distribution
  - Sample



# Preliminary: Bottom-up Parsing

- Two basic actions
  - Shift: shift the next input symbol onto the top of the stack
  - Reduce: replace symbols at the top of the stack to a nonterminal
- Example:

**Grammar**

$e \rightarrow x$
$n$
$e + e$
$e - e$
$e * e$
$e / e$
$(e)$

**Parsing**

Stack	Input	Action
	( $x + 1 ) / 2$	shift
(	$x + 1 ) / 2$	shift
( $x$	$+ 1 ) / 2$	reduce
( $e$	$+ 1 ) / 2$	shift
( $e +$	$1 ) / 2$	shift
( $e + 1$	$) / 2$	reduce
( $e + e$	$) / 2$	shift
( $e + e$	$/ 2$	reduce
$e$	$/ 2$	shift
...	...	...
e		accept

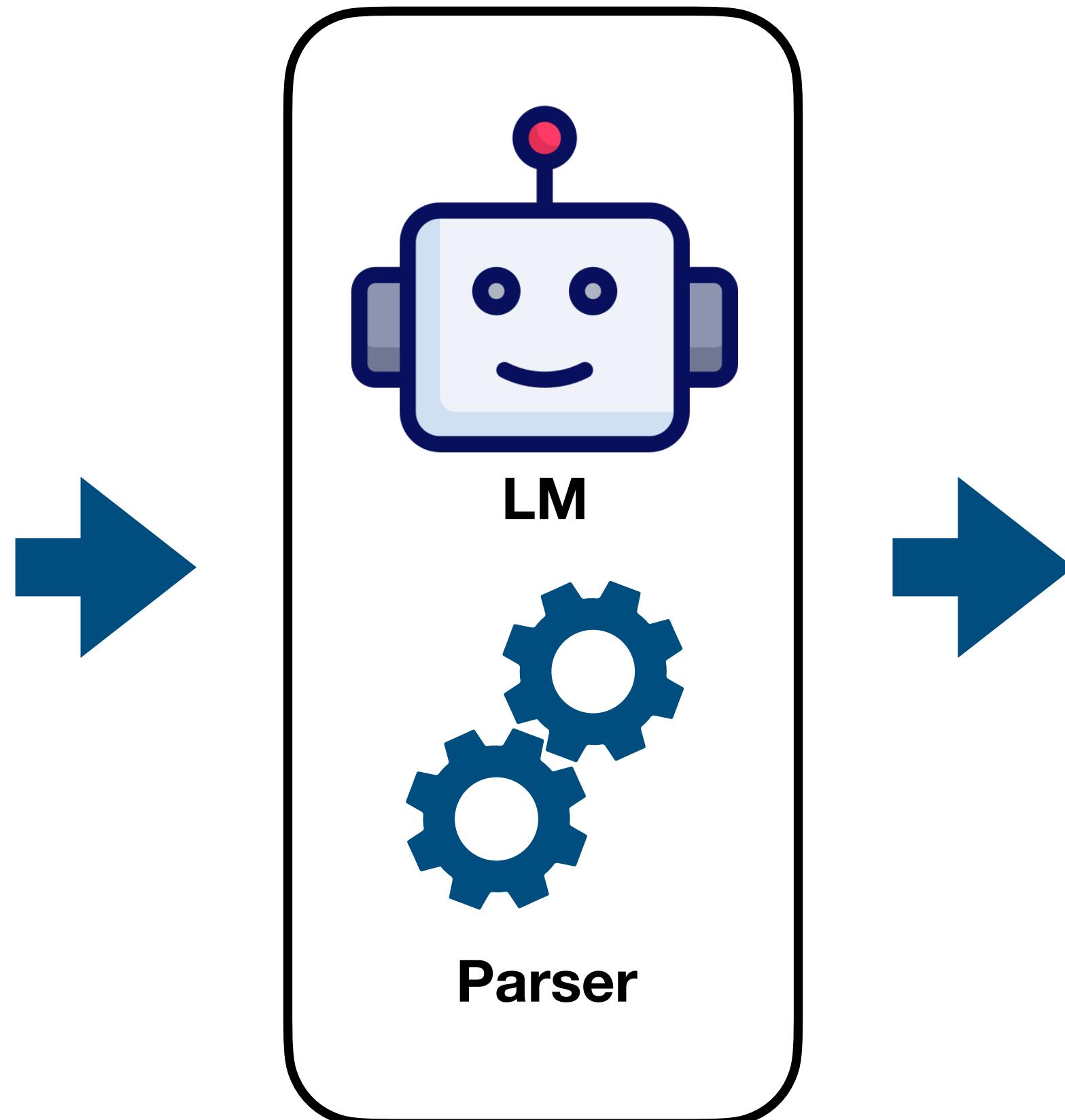
# Grammar-Constrained Decoding

## Grammar

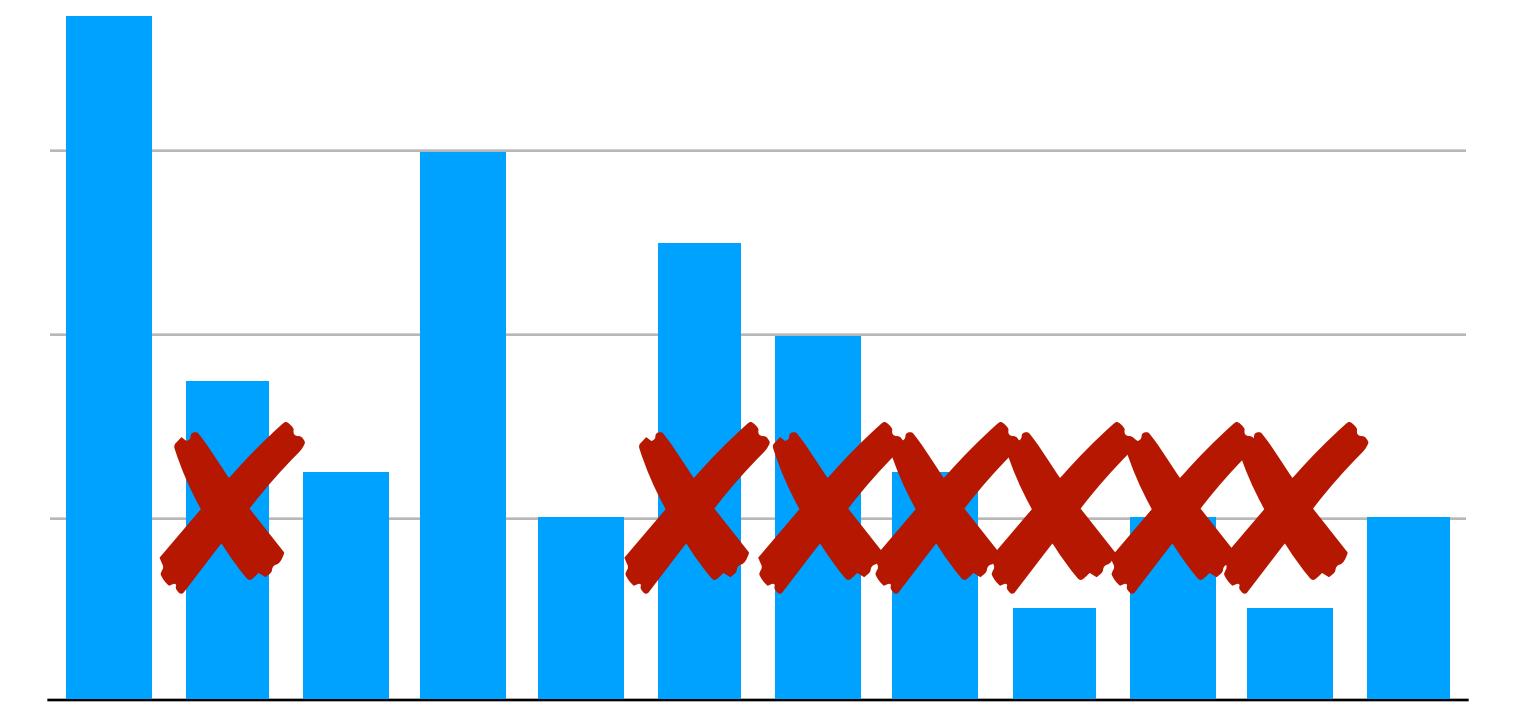
$$\begin{array}{l} e \rightarrow x \\ | \\ n \\ | \\ e + e \\ | \\ e - e \\ | \\ e * e \\ | \\ e / e \\ | \\ (e) \end{array}$$

Prompt: "What is the sum of integers from 1 to x?"

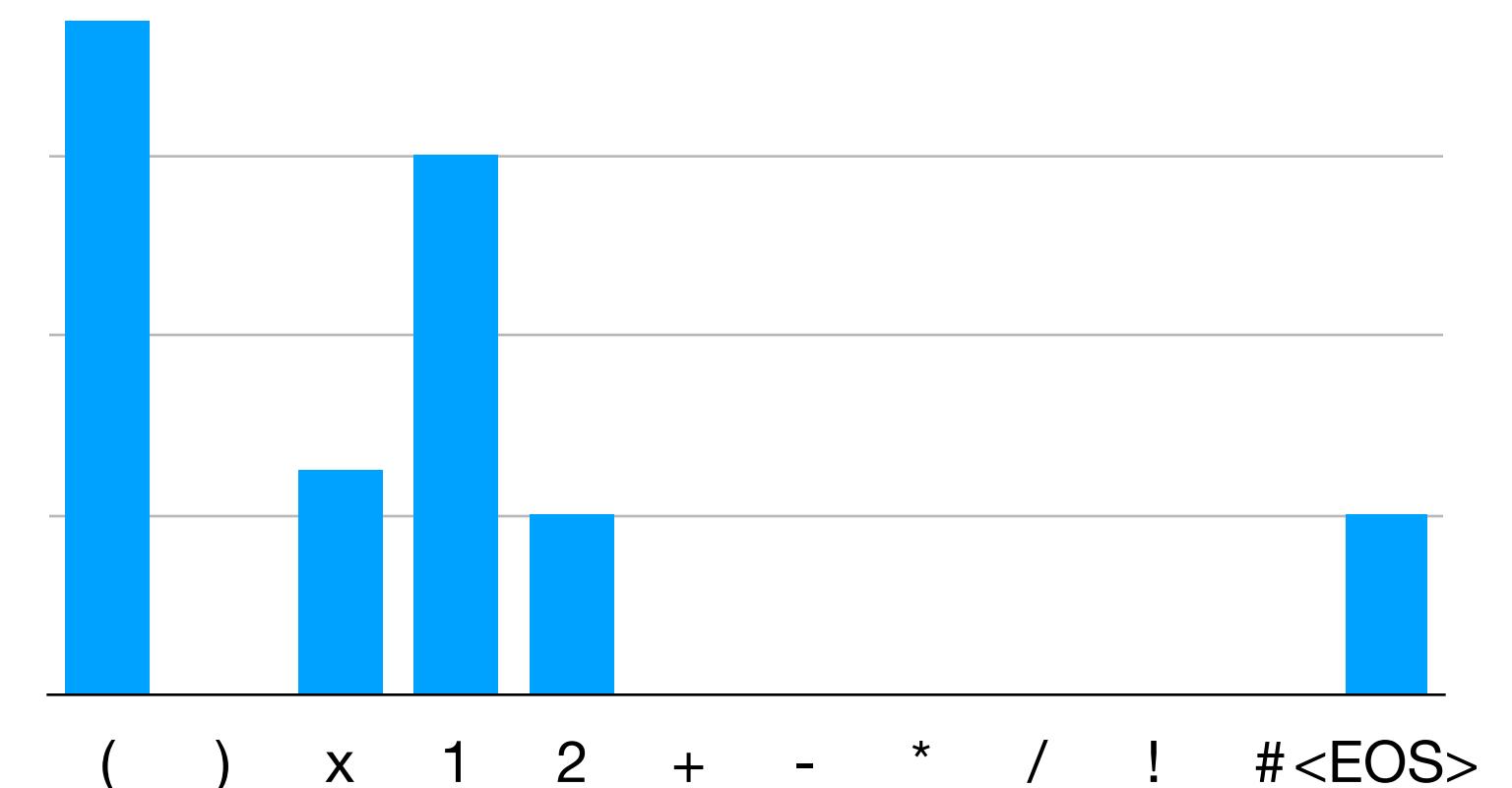
S<sub>0</sub>: "



Original Distribution

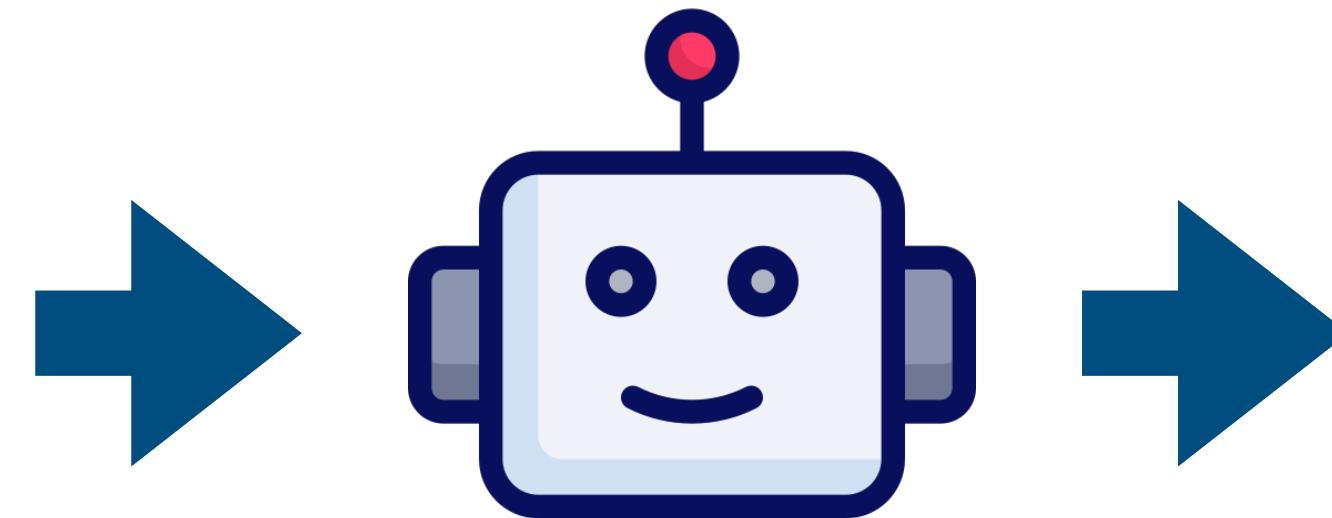


Masked Distribution

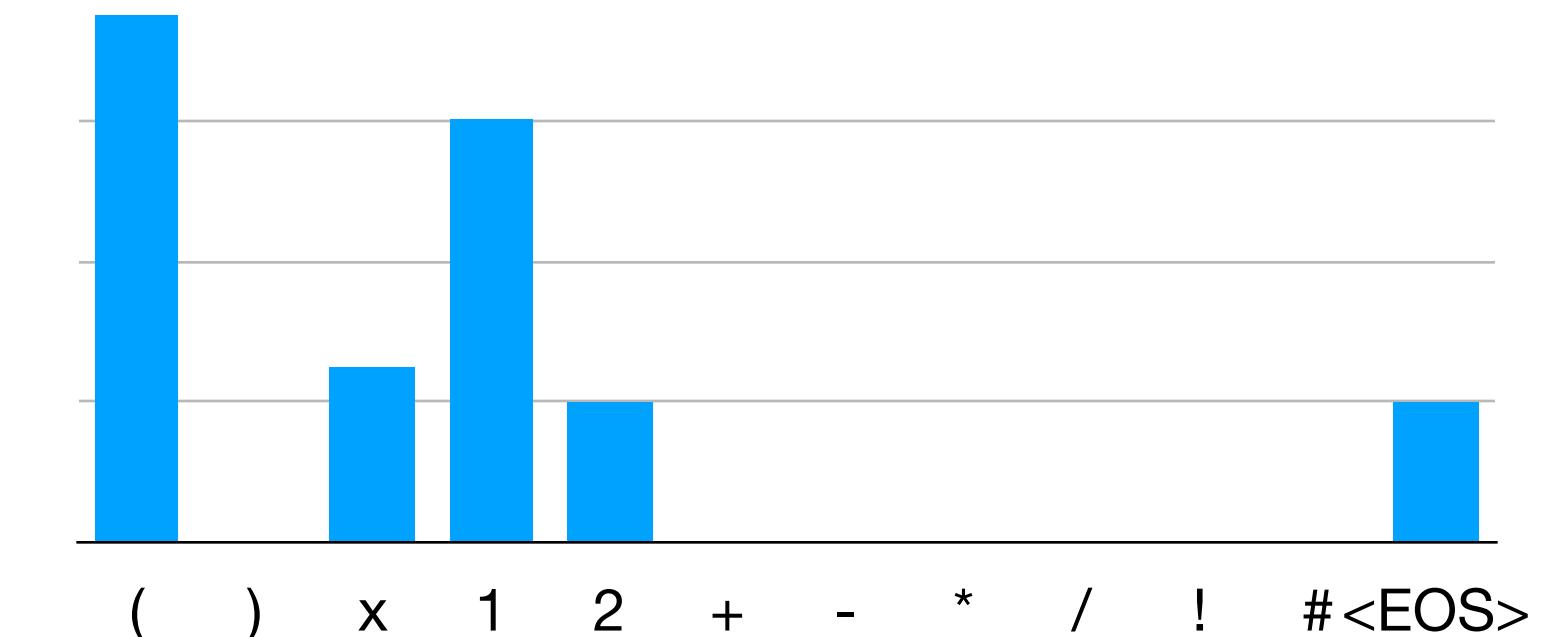


# Example: Arithmetic Expression

Prompt: "What is the sum of integers from 1 to x?"  
S0: ""



Masked Distribution



Stack	Possible Input	Input	Action
	(, x, 1, 2, <EOS>	(	shift
(	(, x, 1, 2, <EOS>	x	shift
(x			reduce
(e	+, -, *, /	+	shift
(e +	x, 1, 2	1	shift
(e + 1		)	reduce
(e + e	)	)	shift
(e + e)			reduce
e		/	shift
...		...	...
e	+, -, *, /, <EOS>	<EOS>	accept

# Practical Aspect

- Different processing units between parser and LM
  - Lexeme (parser): defined by lexer using regular expression, e.g., while, length, if, {
  - Token (LM): learned during training, e.g., wh, ile, len, th, if{
- Important: correctly and efficiently handle the mismatch

# When & Why Constrained Decoding?

- Low-resource programming languages
  - E.g., 69.8% parse errors of GPT-4 Dafny code (2025)
- New domain-specific languages and rules
  - E.g., coding guidelines
- Other domains that have constraints
  - E.g., robots, automobiles, biochemistry, etc

# Summary

- A long-standing argument in AI: connectionism vs symbolism
  - Connectionism (neural network): good at image recognition, machine translation, etc
  - Symbolism (logic): good at equation solving, logical reasoning, etc
- What is the future of AI (or programming)?
  - Neuro-symbolic AI: how to effectively combine both paradigms?