Alien Codes and Their Automated and Human Explanations

Josef Urban
joint work with Thibault Gauthier, Mirek Olsak and Tom Hales

AI4REASON and University of Gothenburg / Chalmers University

SCML'26
July 7, 2026, Linz

European Research Council
Established by the European Commission

OEIS: ⩾ 350000 finite sequences in 2021

The OEIS is supported by the many generous donors to the OEIS Foundation.

THE ON–LINE ENCYCLOPEDIA OF INTEGER SEQUENCES ®

founded in 1964 by N. J. A. Sloane

2 3 5 7 11 Search Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

Search: **seq:2,3,5,7,11**

Displaying 1-10 of 1163 results found. page 1 2 3 4 5 6 7 8 9 10 ... 117

Sort: relevance | references | number | modified | created Format: long | short | data

A000040 The prime numbers. +30 10150
(Formerly M0652 N0241)

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271 (list; graph; refs; listen; history; text; internal format)

OFFSET 1,1

COMMENTS See A065091 for comments, formulas etc. concerning only odd primes. For all information concerning prime powers, see A000961. For contributions concerning "almost primes" see A002808.
A number p is prime if (and only if) it is greater than 1 and has no positive divisors except 1 and p.
A natural number is prime if and only if it has exactly two (positive) divisors.
A prime has exactly one proper positive divisor, 1.

Some Invented Explanations for OEIS ("Alien Coder")

- A4578: Expansion of sqrt(8) in base 3:
  loop2(((y * y) div (x + y)) + y, y, x + x, 2, loop((1 + 2) * x, x, 2)) mod (1 + 2)
- A4001: Hofstadter-Conway $10k seq: a(n) = a(a(n-1)) + a(n-a(n-1)):
  loop(push(loop(pop(x), y-x,pop(x)),x) + loop(pop(x), x-1, x), x - 1, 1)
- A40: prime numbers: 2+compr((loop(x*y, x, 2) + x) mod (2+x), x)
- A30184: Expand $\eta(q) * \eta(q^3) * \eta(q^5) * \eta(q^{15})$ in powers of q (elliptic curves): loop(push(loop((pop(x) * loop(if (pop(x) mod y) <= 0 then (x - loop(if (x mod (1 + (y + y))) <= 0 then (x + x) else x, 2, y)) else x, y, push(0, y))) + x, y, push(0, x)), x) div y, x, 1)
- A51023: Wolfram's $30k Rule 30 automaton:
  loop2(y, y div 2, x, 1, loop2(loop2((((y div (0 - (2 + 2))) mod 2) + x) + x, y div 2, y, 1, loop2(((y mod 2) + x) + x, y div 2, y, 1, x)), 2 + y, x, 0, 1)) mod 2
- A2580: $\sqrt[3]{2}$ Hales's blog: `https://t.ly/tHs1d`
- A2 Kolakoski sequence: a(n) is length of n-th run: 1+pop(loop(push(
  loop(pop(x), x div 2, x), (x + pop(x)) mod 2) + x, x, push(1, 0)))

TL;DR

- A machine can find explanations for over 135k OEIS sequences
- This is done *from scratch*, without any domain knowledge
- N. Sloane: The OEIS: *A Fingerprint File for Mathematics* (2021)
- About 350k integer sequences in 2021 from all parts of math
- We use a simple Search-Verify-Train positive feedback loop
- Developed by us for combining learning & proving since 2006
- However, one of the most surprising experiments in my life:
- 980 iterations and still refuses to plateau - counters RL wisdom
- Since it interleaves symbolic breakthroughs and statistical learning?
- The electricity bill is only 2-3k EUR, you can do this at home
- Btw., we experimentally verify Occam's Razor
- Evolving (self-improving) population of 5.2M matching explanations
- Some of them unusual/novel/alien - how do we show them correct?
- Connections to Solomonoff Induction, AIXI, Gödel Machine?

Brief Intro to What I Normally Do: ML+ATP for Math

- What is mathematical and scientific thinking?
- Pattern-matching, analogy, induction/learning from examples
- Deductive reasoning and proving
- Complicated feedback loops between learning and deduction
- Using a lot of previous knowledge - both for induction and deduction
- We need to develop such methods on computers
- Are there any large corpora suitable for nontrivial deduction?
- Yes! Large libraries of formal proofs and theories
- So let's try to develop strong AI on them!
- In my case done for 25-30 years. Recent overviews:
- *Learning Guided Automated Reasoning: A Brief Survey. 2024*
- *AI4REASON*: `http://ai4reason.org/PR_CORE_SCIENTIFIC_4.pdf`

ENIGMA: Feedback prove/learn loop on formal math

- Done on 57880 Mizar Mathematical Library problems recently
- Efficient ML-guidance inside the best ATPs (E prover and more)
- Training of the ML-guidance is *interleaved* with proving harder problems
- Ultimately a 70% improvement over the original strategy:
- ... from 14933 proofs to 25397 proofs (all 10s CPU - no cheating)
- 75% of the Mizar corpus reached in July 2021 - higher times and many runs: `https://github.com/ai4reason/ATP_Proofs`
- Details in our Mizar60 paper: `https://arxiv.org/abs/2303.06686`

	$\mathcal{S}$	$\mathcal{S} \odot \mathcal{M}_9^0$	$\mathcal{S} \oplus \mathcal{M}_9^0$	$\mathcal{S} \odot \mathcal{M}_9^1$	$\mathcal{S} \oplus \mathcal{M}_9^1$	$\mathcal{S} \odot \mathcal{M}_9^2$	$\mathcal{S} \oplus \mathcal{M}_9^2$	$\mathcal{S} \odot \mathcal{M}_9^3$	$\mathcal{S} \oplus \mathcal{M}_9^3$
solved	**14933**	16574	20366	21564	22839	22413	23467	22910	23753
$\mathcal{S}$%	+0%	+10.5%	+35.8%	+43.8%	+52.3%	+49.4%	+56.5%	+52.8%	+58.4
$\mathcal{S}+$	+0	+4364	+6215	+7774	+8414	+8407	+8964	+8822	+9274
$\mathcal{S}-$	-0	-2723	-782	-1143	-508	-927	-430	-845	-454

	$\mathcal{S} \odot \mathcal{M}_{12}^3$	$\mathcal{S} \oplus \mathcal{M}_{12}^3$	$\mathcal{S} \odot \mathcal{M}_{16}^3$	$\mathcal{S} \oplus \mathcal{M}_{16}^3$
solved	24159	24701	25100	**25397**
$\mathcal{S}$%	+61.1%	+64.8%	+68.0%	**+70.0%**
$\mathcal{S}+$	+9761	+10063	+10476	+10647
$\mathcal{S}-$	-535	-295	-309	-183

What Are the Current AI/TP TODOs/Bottlenecks?

- High-level structuring of proofs - proposing good lemmas
- Proposing new concepts, definitions and theories
- Proposing new targeted algorithms, decision procedures, tactics
- Proposing good witnesses for existential proofs
- All these problems involve synthesis of some mathematical objects
- Btw., constructing proofs is also a synthesis task
- This talk: explore learning-guided synthesis for OEIS
- Interesting research topic and tradeoff in learning/AI/proving:
- Learning direct guessing of objects (this talk) vs guidance for search procedures (ENIGMA and friends)
- Start looking also at semantics rather than just syntax of the objects

Quotes: Learning vs. Reasoning vs. Guessing

"C'est par la logique qu'on démontre, c'est par l'intuition qu'on invente."
(It is by logic that we prove, but by intuition that we discover.)
– Henri Poincaré, Mathematical Definitions and Education.

"Hypothesen sind Netze; nur der fängt, wer auswirft."
(Hypotheses are nets: only he who casts will catch.)
– Novalis, quoted by Popper – The Logic of Scientific Discovery

Certainly, let us learn proving, but also let us learn guessing.
– G. Polya - Mathematics and Plausible Reasoning

Galileo once said, "Mathematics is the language of Science." Hence, facing the same laws of the physical world, alien mathematics must have a good deal of similarity to ours.
– R. Hamming - Mathematics on a Distant Planet

QSynt: Semantics-Aware Synthesis of Math Objects

- Gauthier (et al) 2019-26
- Synthesize math expressions based on semantic characterizations
- i.e., not just on the syntactic descriptions (e.g. proof situations)
- Tree Neural Nets and Monte Carlo Tree Search (a la AlphaZero)
- Later also various (small) *language models* with their search methods
- Invent programs for OEIS sequences FROM SCRATCH (no LLM cheats)
- ~135k sequences and 5.2M explanations invented so far (980 iterations): 80+ different characterizations of primes
- Non-neural (Turing complete) symbolic computing and semantics collaborate with the statistical/neural learning
- Program evolution governed by high-level criteria (Occam, efficiency)

Generating programs for OEIS sequences

$0, 1, 3, 6, 10, 15, 21, \ldots$

An undesirable large program:

```
if x = 0 then 0 else
if x = 1 then 1 else
if x = 2 then 3 else
if x = 3 then 6 else ...
```

Small program (Occam's Razor):

$$\sum_{i=1}^{n} i$$

Fast program (efficiency criteria):

$$\frac{n \times n + n}{2}$$

Programming language

- Constants: $0, 1, 2$
- Variables: x, y
- Arithmetic: $+, -, \times, \mathit{div}, \mathit{mod}$
- Condition : if $\ldots \leqslant 0$ then $\ldots$ else $\ldots$
- $\mathit{loop}(f, a, b) := u_a$ where $u_0 = b$,

$$u_n = f(u_{n-1}, n)$$

- Two other loop constructs: *loop2*, a while loop (*compr*)

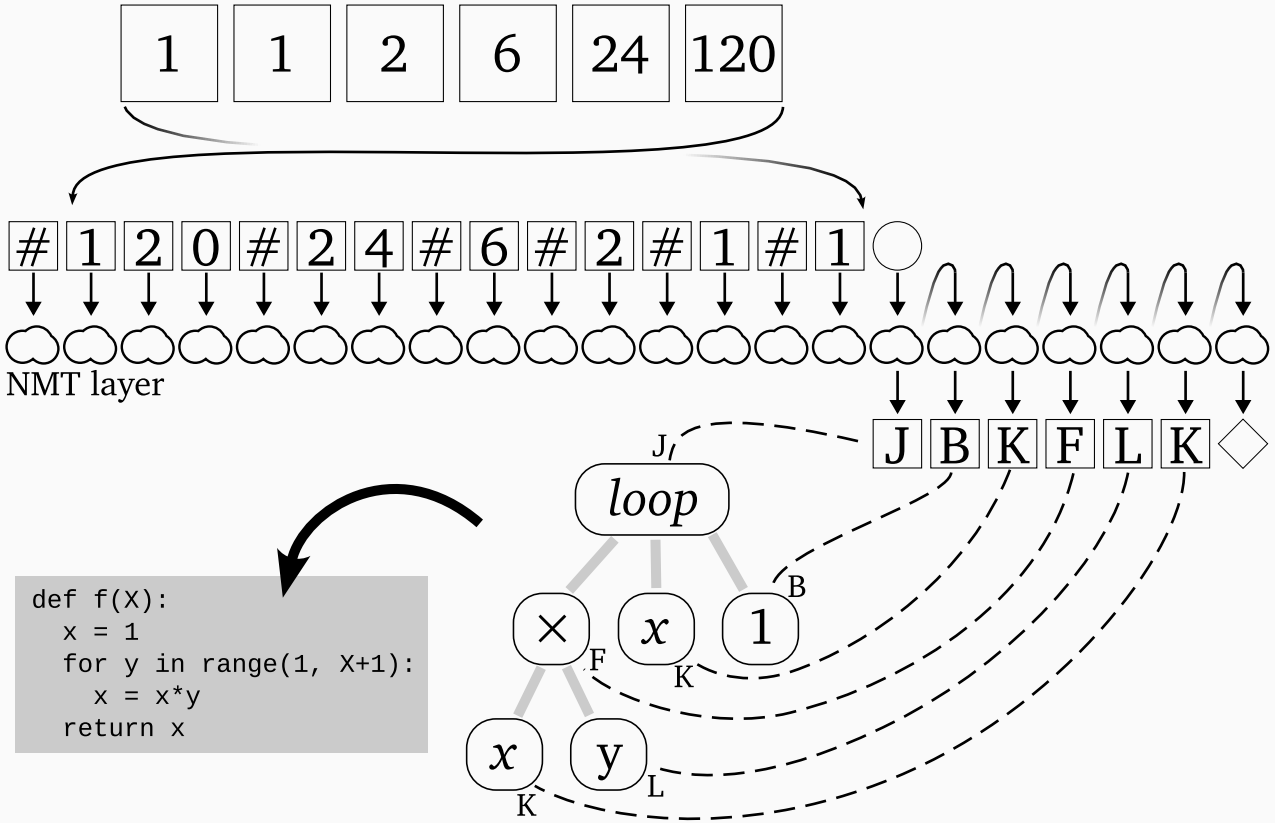
Example:

$2^{\mathbf{x}} = \prod_{y=1}^{x} 2 = \mathit{loop}(2 \times x, \mathbf{x}, 1)$

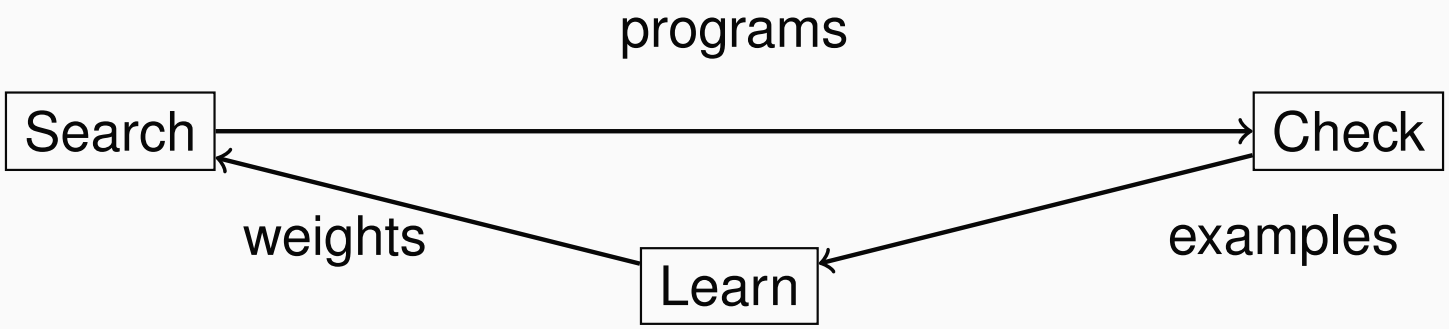
$\mathbf{x}! = \prod_{y=1}^{x} y = \mathit{loop}(y \times x, \mathbf{x}, 1)$

Encoding OEIS for Language Models

- Input sequence is a series of digits
- Separated by an additional token # at the integer boundaries
- Output program is a sequence of tokens in Polish notation
- Parsed by us to a syntax tree and translatable to Python
- Example: $a(n) = n!$



Search-Verify-Train Positive Feedback Loop



- Analogous to our Prove/Learn feedback loops in learning-guided proving (since 2006 – Machine Learner for Automated Reasoning – MaLARea))
- However, the OEIS setting allows much faster feedback on *symbolic conjecturing*
- (But: see Part 2 of the talk for bringing this back to the proving setting)

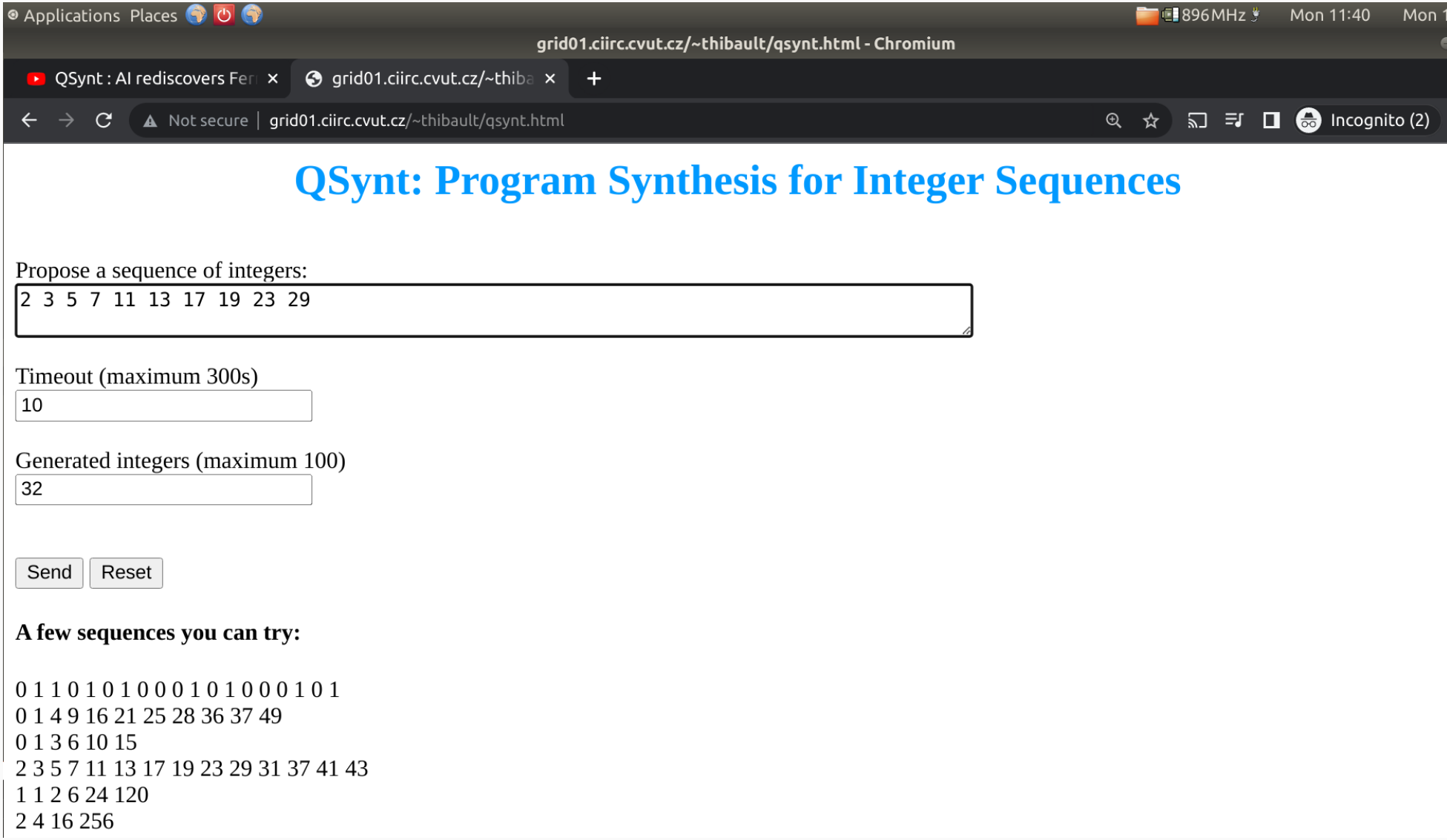
Search-Verify-Train Feedback Loop for OEIS

- **search phase**: LM synthesizes many programs for input sequences
- typically 240 candidate programs for each input using beam search
- 84M programs for OEIS in several hours on the GPU (depends on model)
- **checking phase**: the millions of programs efficiently evaluated
- resource limits used, fast indexing structures for OEIS sequences
- check if the program generates *any* OEIS sequence (hindsight replay)
- we keep the shortest (Occams's razor) and fastest program (efficiency)
- from iter. 501, we also keep the program with the best speed/length ratio
- **learning phase**: LM trains to translate the "solved" OEIS sequences into the best program(s) generating them
- from iter. 336: train LMs to transform (generalization, optimization)
- our learned version of human-coded methods like ILP and compilation

Search-Verify-Train Feedback Loop

- The weights of the LM either trained from scratch or continuously updated
- This yields *new minds* vs *seasoned experts* (who have seen it all)
- We also train experts on varied selections of data, in varied ways
- Orthogonality: common in theorem proving - different experts help
- Each iteration of the self-learning loop discovers more solutions
- ... also improves/optimizes existing solutions
- The alien mathematician thus self-evolves
- Occam's razor and efficiency are used for its weak supervision
- Quite different from today's LLM approaches:
- LLMs do one-time training on everything human-invented
- Our Alien instead starts from zero knowledge
- Evolves increasingly nontrivial skills, may diverge from humans
- Turing complete (unlike Go/Chess) – arbitrary complex algorithms

QSynt web interface for program invention



QSynt inventing Fermat pseudoprimes

Positive integers k such that $2^k \equiv 2 \mod k$. ($341 = 11 * 31$ is the first non-prime)

```
First 16 generated numbers (f(0),f(1),f(2),...):
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53
Generated sequence matches best with: A15919(1-75), A100726(0-59), A40(0-58)

Program found in 5.81 seconds
f(x) := 2 + compr(\x.loop(\(x,i).2*x + 2, x, 2) mod (x + 2), x)
Run the equivalent Python program here or in the window below:
```

BRYTHON

Tutorial Demo Documentation Console Editor Gallery Resources

English

Brython version: 3.10.6

run | Python | Javascript | Share code

```
def f2(X):
  x = 2
  for i in range (1,X + 1):
    x = 2*x + 2
  return x

def f1(X):
  x,i = 0,0
  while i <= X:
    if f2(x) % (x + 2) <= 0:
      i = i + 1
    x = x + 1
  return x - 1

def f0(X):
  return 2 + f1(X)

for x in range(32):
  print (f0(x))
```

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
```

Lucas/Fibonacci characterization of (pseudo)primes

```
input sequence: 2,3,5,7,11,13,17,19,23,29

invented output program:
f(x) := compr(\(x,y).(loop2(\(x,y).x + y, \(x,y).x, x, 1, 2) - 1)
                mod (1 + x), x + 1) + 1

human conjecture: x is prime iff? x divides (Lucas(x) - 1)

PARI program:
? lucas(n) = fibonacci(n+1)+fibonacci(n-1)
? b(n) = (lucas(n) - 1) % n

Counterexamples (Bruckman-Lucas pseudoprimes):
? for(n=1,4000,if(b(n)==0,if(isprime(n),0,print(n))))
1
705
2465
2737
3745
```

QSynt inventing primes using Wilson's theorem

n is prime iff $(n-1)! + 1$ is divisible by n (i.e.: $(n-1)! \equiv -1 \mod n$)

```
First 32 generated numbers (f(0),f(1),f(2),...):
0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0
Generated sequence matches best with: A10051(0-100), A252233(0-29), A283991(0-24)

Program found in 5.17 seconds
f(x) := (loop(\(x,i).x * i, x, x) mod (x + 1)) mod 2
Run the equivalent Python program here or in the window below:
```

BRYTHON

Tutorial Demo Documentation Console Editor Gallery Resources

English

Brython version: 3.10.6

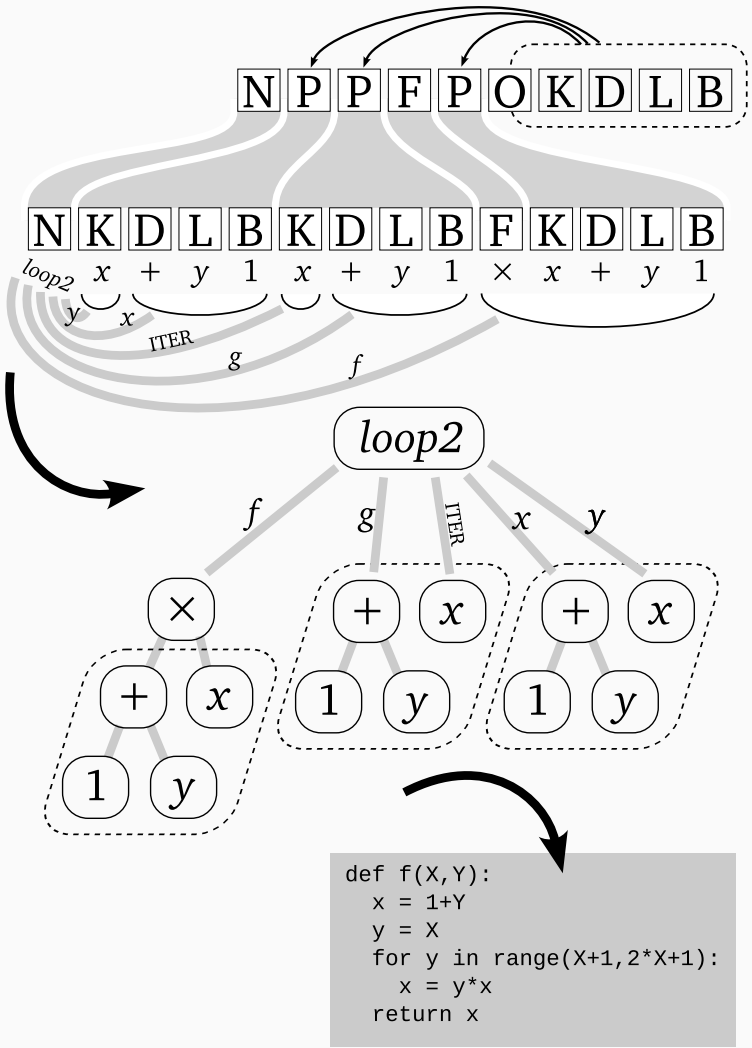
run Python Javascript Share code

```
1  def f1(X):
2    x = X
3    for i in range (1,X + 1):
4      x = x * i
5    return x
6
7  def f0(X):
8    return (f1(X) % (X + 1)) % 2
9
10 for x in range(32):
11   print (f0(x))
12
```

```
0
1
1
0
1
0
1
0
0
0
1
0
1
```

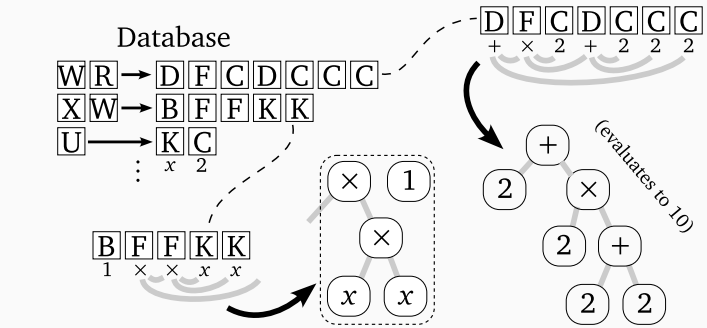
Introducing Local Macros/Definitions

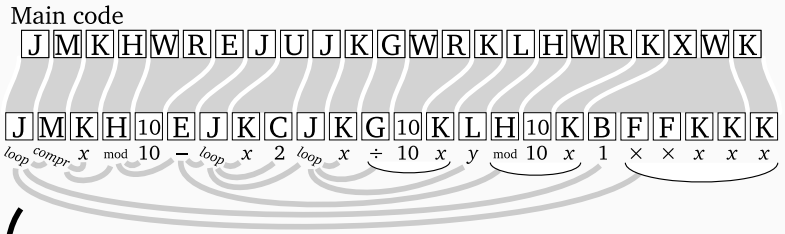
A macro/expanded version of a program invented for A1813: $a(n) = (2n)!/n!$.
1, 2, 12, 120, 1680, 30240, 665280, 17297280, 518918400, 17643225600,

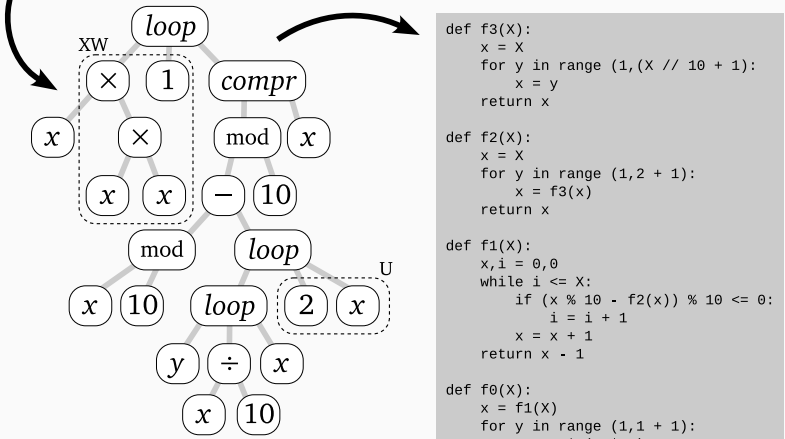


Introducing Global Macros/Definitions

A macro/expanded version of A14187 (cubes of palindromes).
0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1331, 10648, 35937, 85184,








```
def f3(X):
    x = X
    for y in range (1,(X // 10 + 1):
        x = y
    return x

def f2(X):
    x = X
    for y in range (1,2 + 1):
        x = f3(x)
    return x

def f1(X):
    x,i = 0,0
    while i <= X:
        if (x % 10 - f2(x)) % 10 <= 0:
            i = i + 1
        x = x + 1
    return x - 1

def f0(X):
    x = f1(X)
    for y in range (1,1 + 1):
        x = x * (x * x)
    return x
```

Five Different Self-Learning Runs

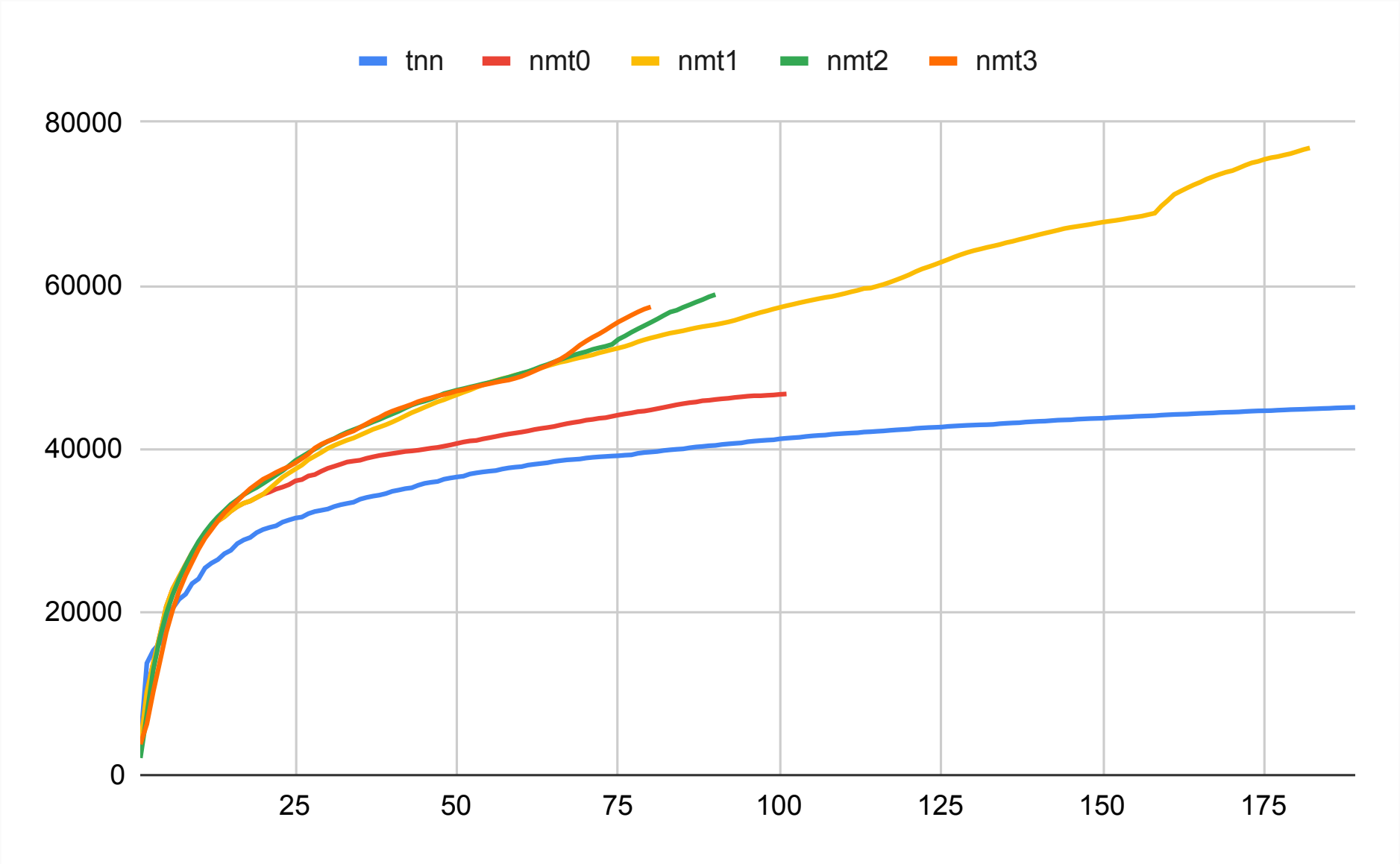


Figure: Cumulative counts of solutions.

Five Different Self-Learning Runs

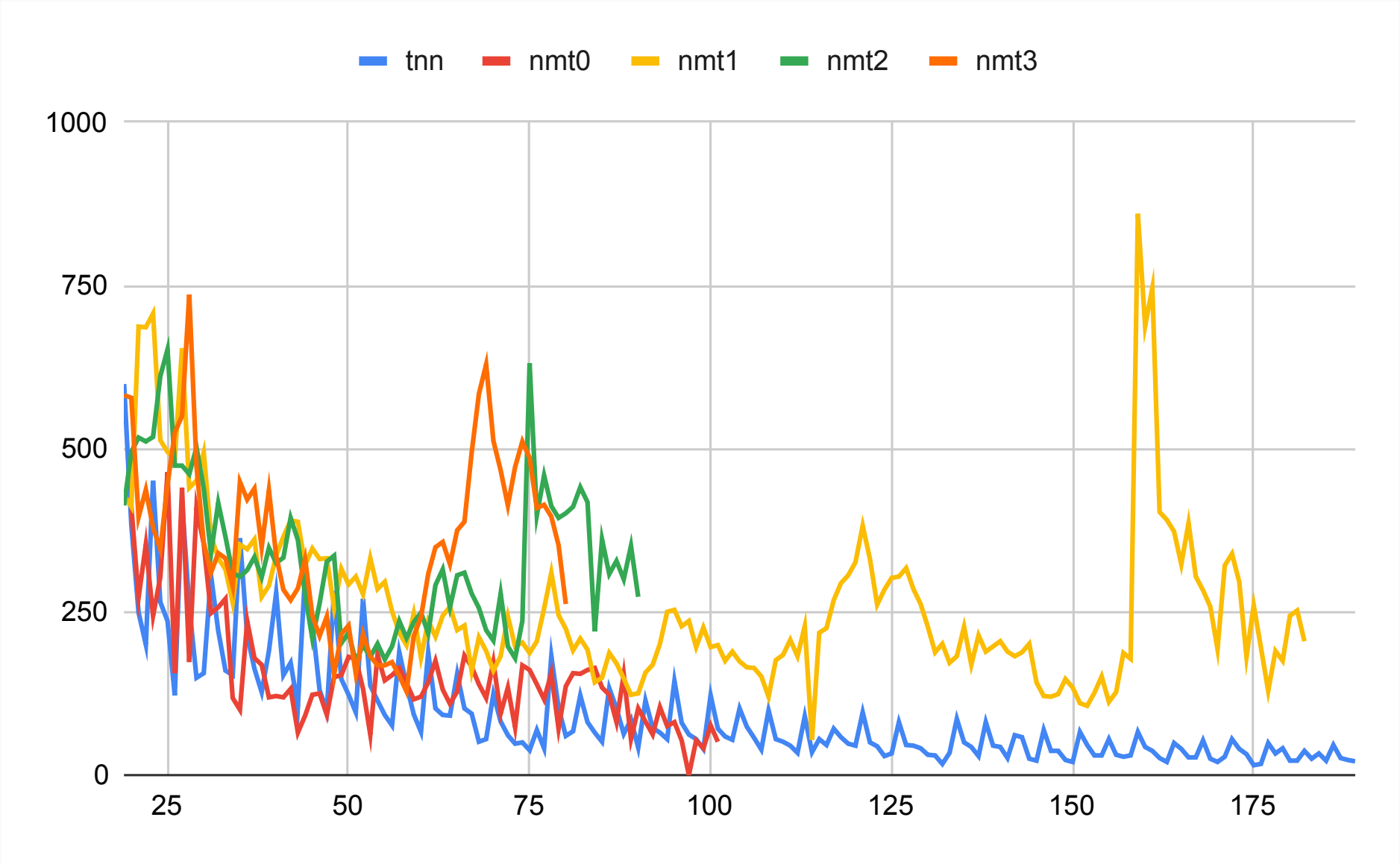


Figure: Increments of solutions.

Speed Evolution – Technology Breakthroughs

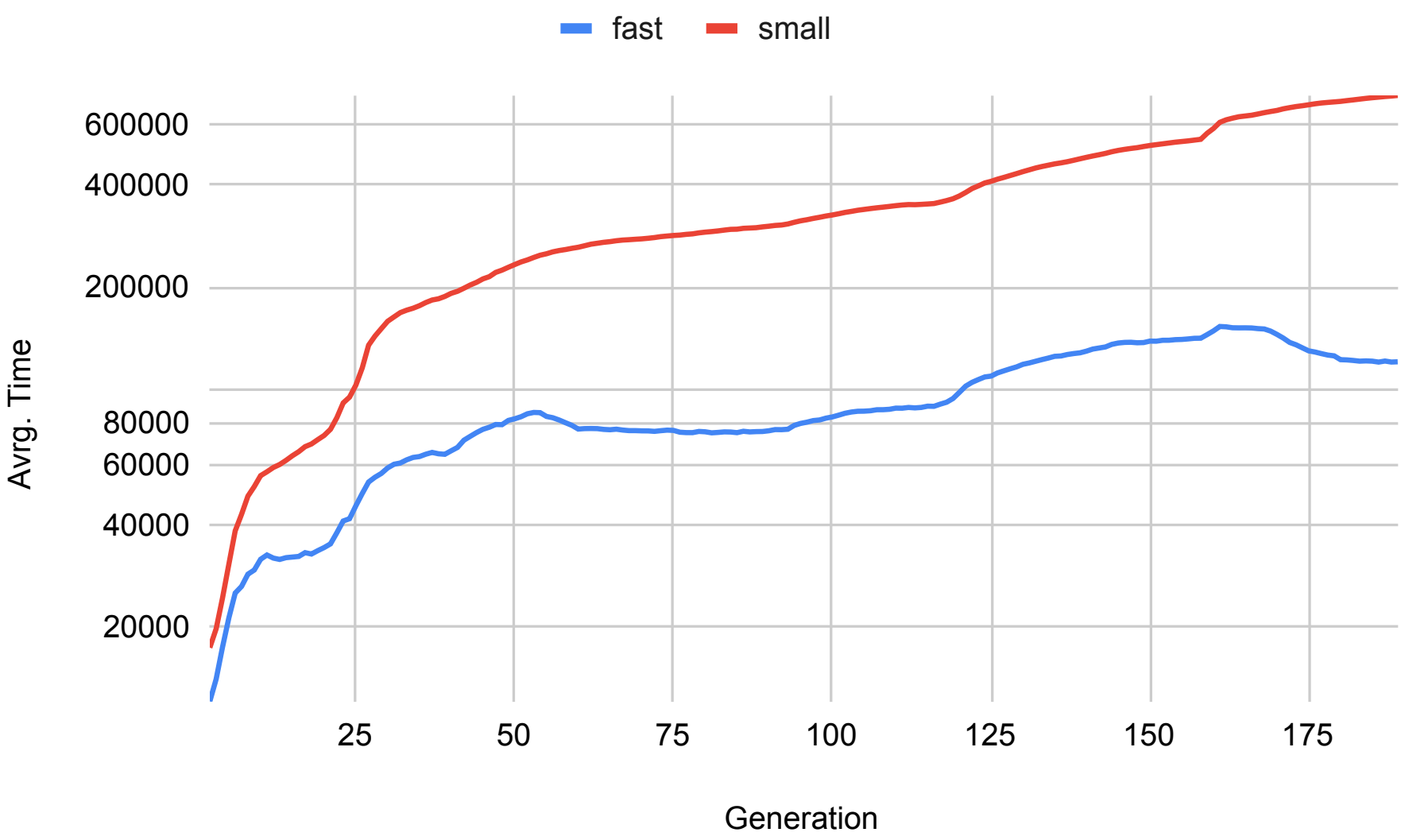
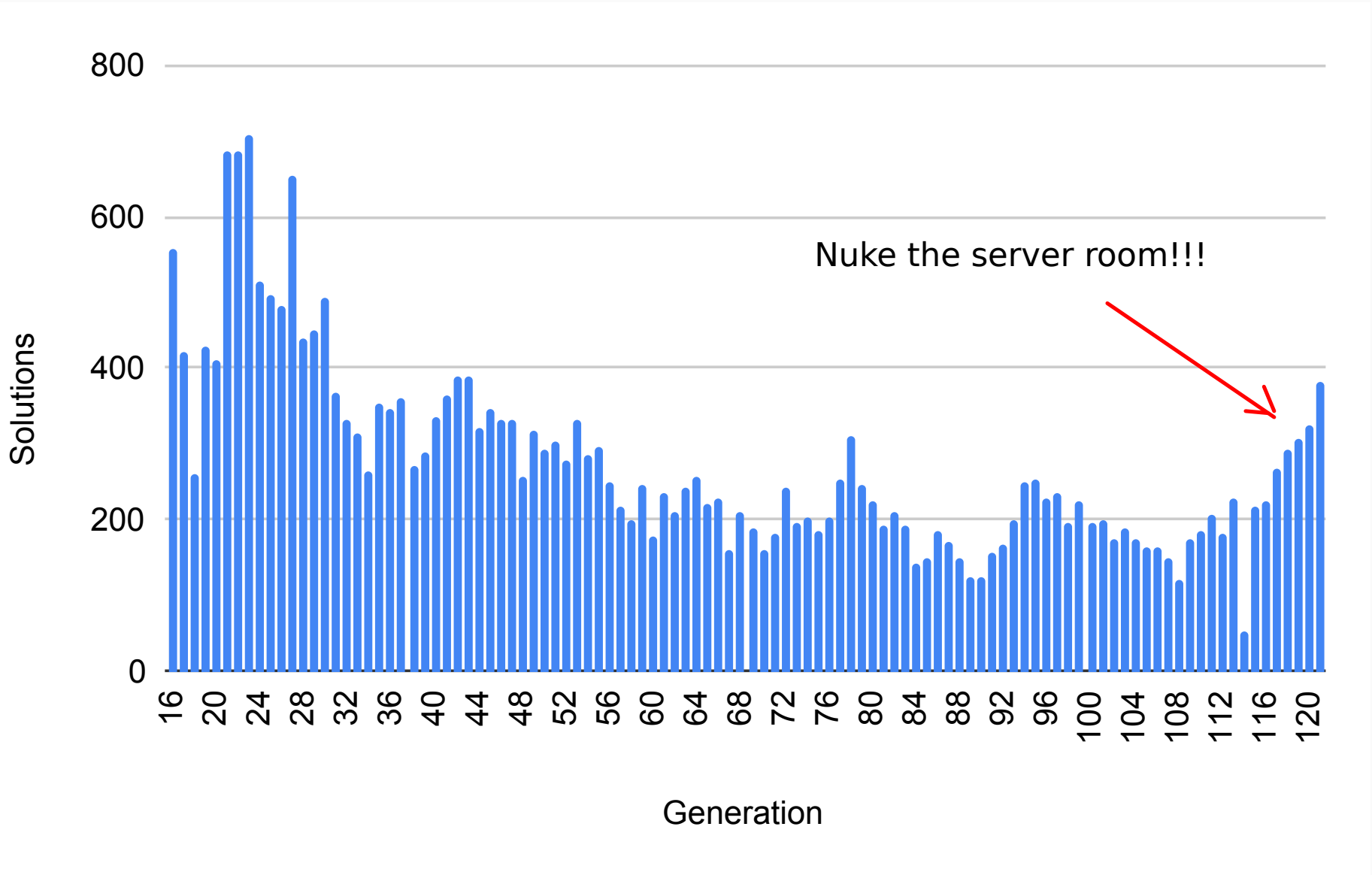
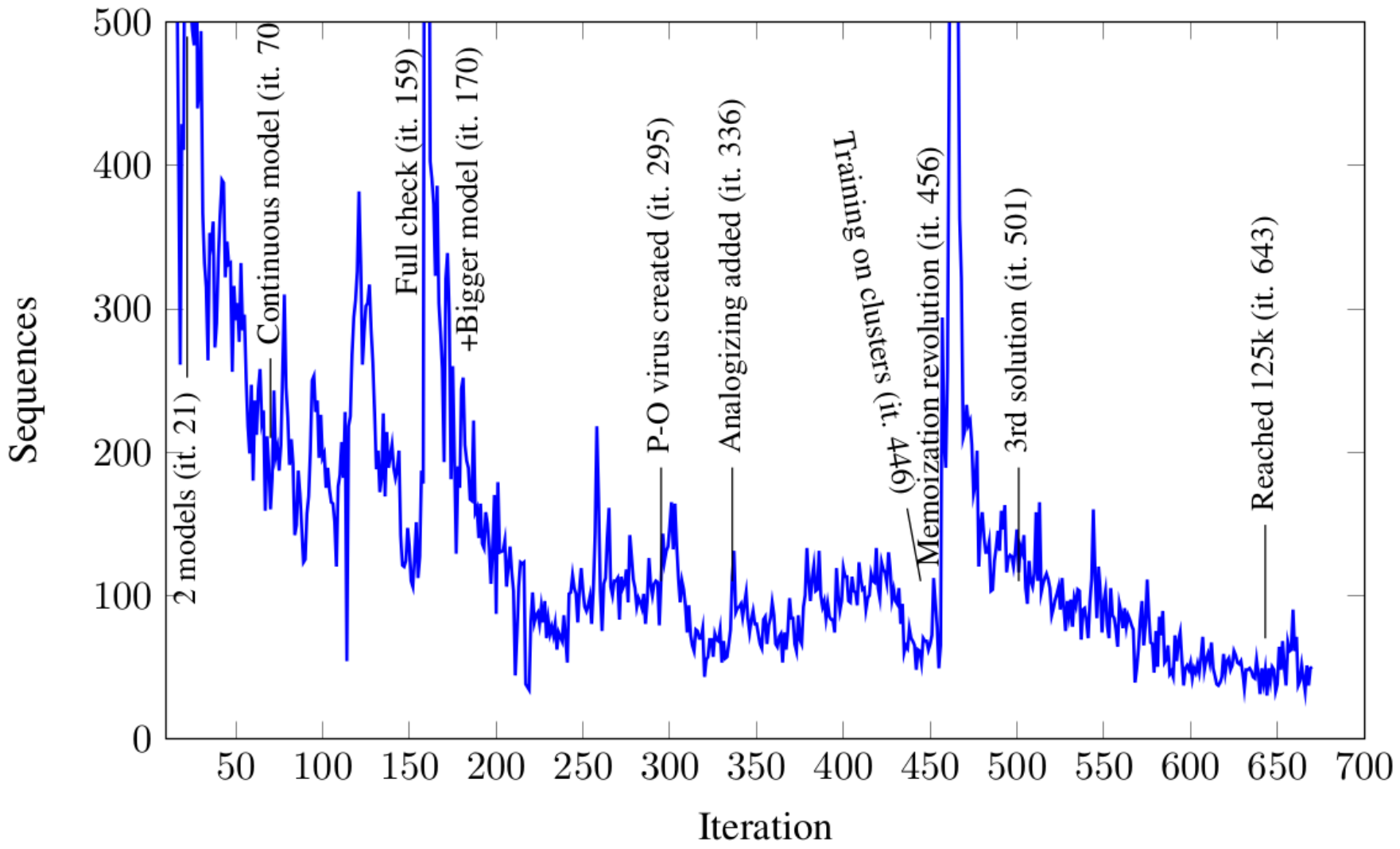


Figure: Avrg. time in iterations

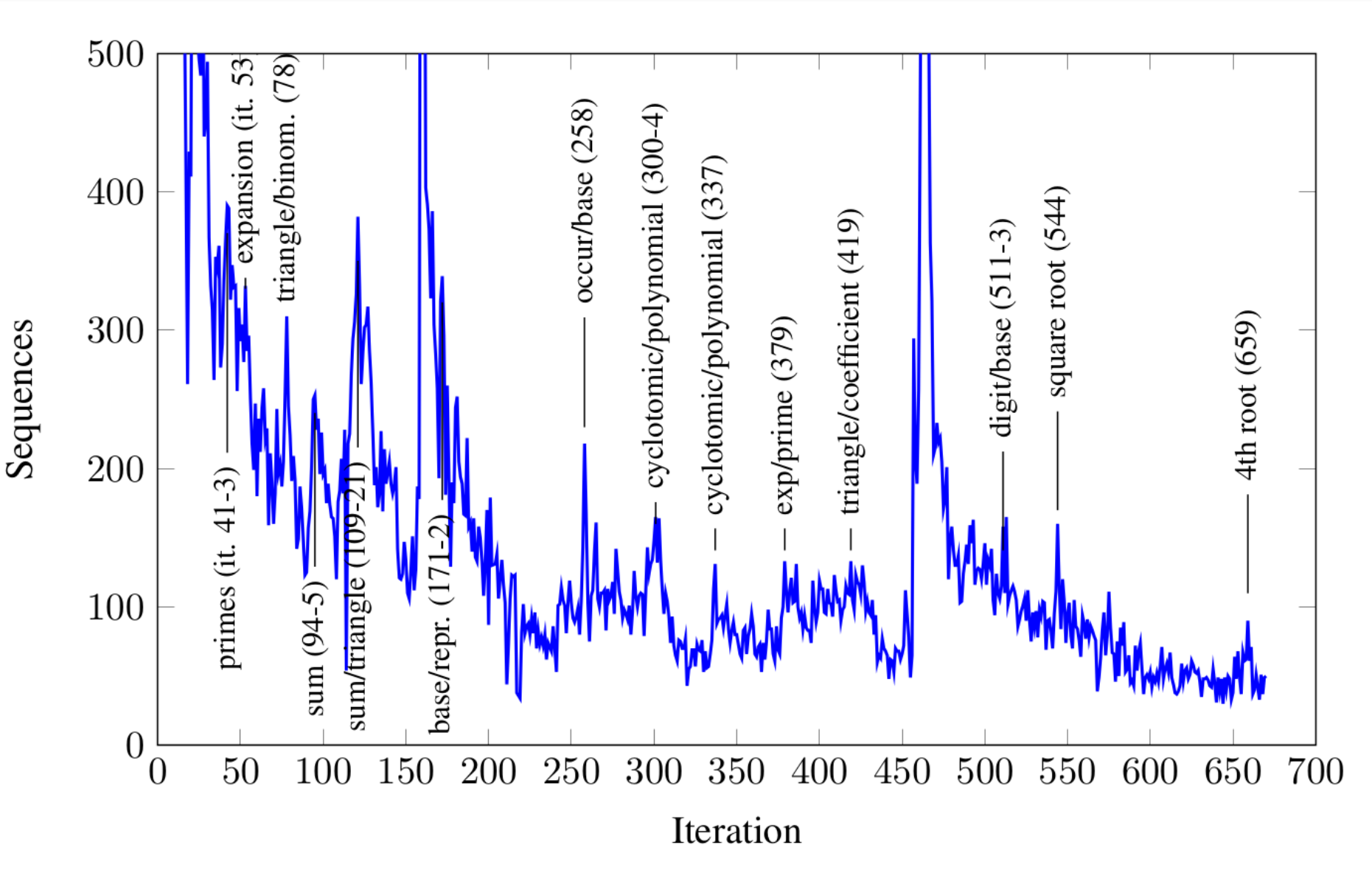
Singularity Take-Off X-mas Card (December 2022)



Human Made Technology Jumps



Some Automatic Technology Jumps



Some Automatic Technology Jumps

- iter 53: expansion/prime: A29363 Expansion of $1/((1-x^4)(1-x^7)(1-x^9)(1-x^{10}))$
- iter 78: triangle/binomial: A38313 Triangle whose (i,j)-th entry is $binomial(i,j) * 10^{i-j} * 11^j$
- iter 94-5: sum: A100192 $a(n) = Sum_{k=0..n} binomial(2n, n+k) * 2^k$
- 109-121: sum/triangle: A182013 Triangle of partial sums of Motzkin numbers
- 171-2: base/representation: A39080 n st base-9 repr. has the same number of 0's and 4's
- 258: occur/base: A44533 n st "2,0" occurs in the base 7 repr of n but not of $n+1$
- 300-304: cyclotomic/polynomial: A14620 Inverse of 611th cyclotomic polynomial
- 379: exp/prime: A124214 E.g.f.: $exp(x)/(2-exp(3*x))^{1/3}$
- 419: triangle/coefficient: A15129 Triangle of (Gaussian) q-binomial coefficients for $q=-13$
- 511,3: digit/base/prime: A260044 Primes with decimal digits in 0,1,3.
- 544: square root: A10538 Decimal expansion of square root of 87.
- 659: 4th root: A11084 Decimal expansion of 4th root of 93.

Some Invented Explanations for OEIS ("Alien Coder")

- A4578: Expansion of sqrt(8) in base 3:
  loop2(((y * y) div (x + y)) + y, y, x + x, 2, loop((1 + 2) * x, x, 2)) mod (1 + 2)
- A4001: Hofstadter-Conway $10k seq: a(n) = a(a(n-1)) + a(n-a(n-1)):
  loop(push(loop(pop(x), y-x,pop(x)),x) + loop(pop(x), x-1, x), x - 1, 1)
- A40: prime numbers: 2+compr((loop(x*y, x, 2) + x) mod (2+x), x)
- A30184: Expand $\eta(q) * \eta(q^3) * \eta(q^5) * \eta(q^{15})$ in powers of q (elliptic curves): loop(push(loop((pop(x) * loop(if (pop(x) mod y) <= 0 then (x - loop(if (x mod (1 + (y + y))) <= 0 then (x + x) else x, 2, y)) else x, y, push(0, y))) + x, y, push(0, x)), x) div y, x, 1)
- A51023: Wolfram's $30k Rule 30 automaton:
  loop2(y, y div 2, x, 1, loop2(loop2((((y div (0 - (2 + 2))) mod 2) + x) + x, y div 2, y, 1, loop2(((y mod 2) + x) + x, y div 2, y, 1, x)), 2 + y, x, 0, 1)) mod 2
- A2580: $\sqrt[3]{2}$ Hales's blog: `https://t.ly/tHs1d`
- A2 Kolakoski sequence: a(n) is length of n-th run: 1+pop(loop(push(
  loop(pop(x), x div 2, x), (x + pop(x)) mod 2) + x, x, push(1, 0)))

Serious Math Conjecturing – Elliptic Curves (2024)

- Sander Dahmen: *Here are some OEIS labels related to elliptic curves (and hence modular forms), ordered by difficulty. It would be interesting to know if some of these appear in your results.*
- A006571 A030187 A030184 A128263 A187096 A251913
- JU: *We have the first three:*
- A6571 : loop((push(loop((pop(x) * loop(if (pop(x) mod y) <= 0 then ((if (y mod loop(1 + (x + x), 2, 2)) <= 0 then (x - y) else x) - y) else x, y, push(0, y))) + x, y, push(0, x)), x) * 2) div y, x, 1)
- A30187 : loop(push(loop((pop(x) * loop(if (pop(x) mod y) <= 0 then (x - loop(if (x mod (((2 + y) * y) - 1)) <= 0 then (x + x) else x, 2, y)) else x, y, push(0, y))) + x, y, push(0, x)), x) div y, x, 1)
- A30184 : loop(push(loop((pop(x) * loop(if (pop(x) mod y) <= 0 then (x - loop(if (x mod (1 + (y + y))) <= 0 then (x + x) else x, 2, y)) else x, y, push(0, y))) + x, y, push(0, x)), x) div y, x, 1)

A6571: Expansion of $q * Product_{k>=1}(1 - q^k)^2 * (1 - q^{11*k})^2$
A30187: Expansion of $\eta(q) * \eta(q^2) * \eta(q^7) * \eta(q^{14})$ in powers of q.
A30184: Expansion of $\eta(q) * \eta(q^3) * \eta(q^5) * \eta(q^{15})$ in powers of q.

Part 2: Are Alien Programs Correct?

1. We can test with more terms in the sequences (b-files in the OEIS)
2. We can ask human mathematicians (supported by various tools)
3. We can try to automatically prove program equivalence

1. Generalization of the Solutions to Larger Indices

- Are the programs correct?
- And can we experimentally verify Occam's razor?
  (implications for how we should be designing ML/AI systems!)
- OEIS provides additional terms for some of the OEIS entries
- Among 78118 solutions, 40,577 of them have a b-file with 100 terms
- We evaluate both the small and the fast programs on them
- Here, 14,701 small and 11,056 fast programs time out.
- 90.57% of the remaining slow programs check
- 77.51% for the fast programs
- This means that **SHORTER EXPLANATIONS ARE MORE RELIABLE!**
  (Occam was right, so why is everybody building trillion-param LLMs???)
- Common error: reliance on an approximation of a real number, such as π.

2. Infinite Math-Nerd Sniping

- We have 5.2M problems for math nerds like this one:
- JU: *This thing works for the first 1k values (just checked) - any idea why?*
- `https://oeis.org/A004578` - Expansion of sqrt(8) in base 3.
- loop2(((y * y) div (x + y)) + y, y, x + x, 2, loop((1 + 2) * x, x, 2)) mod (1 + 2)
- MO: *Not a proof, just a rough idea: The program iterates the function q |-> 2+q / 1+q, where q is a rational number. This converges to sqrt(2). The number q is represented by an integer 'a' such that* $a = 3^x * (2 * q)$, *where 'x' is the input. Once the approximation is good enough,* $a = floor(3^x * sqrt(8))$, *so a mod 3 is the digit we want.*

3. Are two QSynt programs equivalent?

- As with primes, we often find many programs for one OEIS sequence
- Currently we have almost 5.2M programs for the 135k sequences
- On average about 38 programs per “solved” sequence
- It may be quite hard to see that the programs are equivalent
- Extend to Schmidhuber’s Gödel Machine?
- A simple example for $0, 2, 4, 6, 8, \ldots$ with two programs f and g:
  - $f(0) = 0, f(n) = 2 + f(n-1)$ if $n > 0$
  - $g(n) = 2 * n$
  - conjecture: $\forall n \in \mathbb{N}.g(n) = f(n)$
- We can ask mathematicians, but we have thousands of such problems
- Or we can try to ask our ATPs (and thus create a large ATP benchmark)!
- Here is one SMT encoding by Janota & Gauthier:

```
(set-logic UFLIA)
(define-fun-rec f ((x Int)) Int (ite (<= x 0) 0 (+ 2 (f (- x 1)))))
(assert (exists ((c Int)) (and (> c 0) (not (= (f c) (* 2 c))))))
(check-sat)
```

Inductive proof by Vampire of the $f = g$ equivalence

```
% SZS output start Proof for rec2
1. f(X0) = $ite($lesseq(X0,0), 0,$sum(2,f($difference(X0,1)))) [input]
2. ? [X0 : $int] : ($greater(X0,0) & ~f(X0) = $product(2,X0)) [input]
[...]
43. ~$less(0,X0) | iG0(X0) = $sum(2,iG0($sum(X0,-1))) [evaluation 40]
44. (! [X0 : $int] : (($product(2,X0) = iG0(X0) & ~$less(X0,0)) => $product(2,$sum(X0,1)) = iG0($sum(X0,1)))
    &  $product(2,0) = iG0(0))   => ! [X1 : $int] : ($less(0,X1) => $product(2,X1) = iG0(X1)) [induction hypo]
[...]
49. $product(2,0) != iG0(0) | $product(2,$sum(sK3,1)) != iG0($sum(sK3,1)) | ~$less(0,sK1) [resolution 48,41]
50. $product(2,0) != iG0(0) | $product(2,sK3) = iG0(sK3) | ~$less(0,sK1) [resolution 47,41]
51. $product(2,0) != iG0(0) | ~$less(sK3,0) | ~$less(0,sK1) [resolution 46,41]
52. 0 != iG0(0) | $product(2,$sum(sK3,1)) != iG0($sum(sK3,1)) | ~$less(0,sK1) [evaluation 49]
53. 0 != iG0(0) | $product(2,sK3) = iG0(sK3) | ~$less(0,sK1) [evaluation 50]
54. 0 != iG0(0) | ~$less(sK3,0) | ~$less(0,sK1) [evaluation 51]
55. 0 != iG0(0) | ~$less(sK3,0) [subsumption resolution 54,39]
57. 1 <=> $less(sK3,0) [avatar definition]
59. ~$less(sK3,0) <- (~1) [avatar component clause 57]
61. 2 <=> 0 = iG0(0) [avatar definition]
64. ~1 | ~2 [avatar split clause 55,61,57]
65. 0 != iG0(0) | $product(2,sK3) = iG0(sK3) [subsumption resolution 53,39]
67. 3 <=> $product(2,sK3) = iG0(sK3) [avatar definition]
69. $product(2,sK3) = iG0(sK3) <- (3) [avatar component clause 67]
70. 3 | ~2 [avatar split clause 65,61,67]
71. 0 != iG0(0) | $product(2,$sum(sK3,1)) != iG0($sum(sK3,1)) [subsumption resolution 52,39]
72. $product(2,$sum(1,sK3)) != iG0($sum(1,sK3)) | 0 != iG0(0) [forward demodulation 71,5]
74. 4 <=> $product(2,$sum(1,sK3)) = iG0($sum(1,sK3)) [avatar definition]
76. $product(2,$sum(1,sK3)) != iG0($sum(1,sK3)) <- (~4) [avatar component clause 74]
77. ~2 | ~4 [avatar split clause 72,74,61]
82. 0 = iG0(0) [resolution 36,10]
85. 2 [avatar split clause 82,61]
246. iG0($sum(X1,1)) = $sum(2,iG0($sum($sum(X1,1),-1))) | $less(X1,0) [resolution 43,14]
251. $less(X1,0) | iG0($sum(X1,1)) = $sum(2,iG0(X1)) [evaluation 246]
[...]
1176. $false <- (~1, 3, ~4) [subsumption resolution 1175,1052]
1177. 1 | ~3 | 4 [avatar contradiction clause 1176]
1178. $false [avatar sat refutation 64,70,77,85,1177]
% SZS output end Proof for rec2
% Time elapsed: 0.016 s
```

A new Benchmark for Proving Such Equivalences

- Initially (2023) about 30k problems for proving Fast=Small
- About 16k problems seem to require induction
- This is quite hard: SMTs can do a lot of arithmetic
- But not so much induction
- **Baseline ATP/SMT performance**: poor (504 by CVC5)
- Solution: build another self-learning loop!
- This will be a genuine Prove-Learn loop
- However, focused on conjecturing the inductive predicates

OEIS Benchmark Problems

Each problem involves proving:

$$\forall x \in \mathbb{N}, \quad f_{\text{small}}(x) = f_{\text{fast}}(x)$$

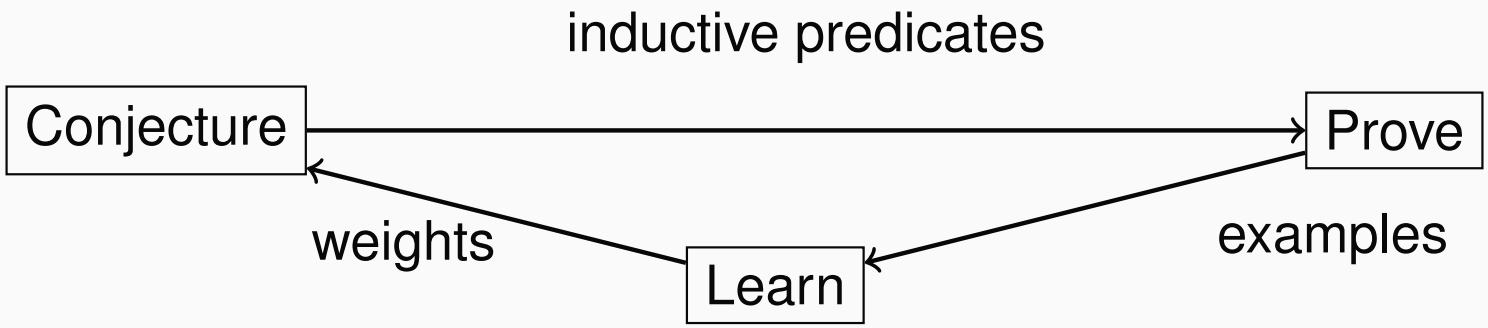
SMT encoding:

$$\exists c \in \mathbb{Z}, \quad c \geqslant 0 \wedge \neg(f_{\text{small}}(c) = f_{\text{fast}}(c))$$

Example (A217, Triangular numbers):

$$\sum_{k=1}^{x} k = \frac{x^2 + x}{2}$$

Conjecture-Prove-Train Positive Feedback Loop



- Unlike in the OEIS loop, here we do not just test but prove
- Proving is usually (much) slower than testing
- Hence we call an SMT (Z3) with a low time limit (200 ms)
- We try to minimize the solutions
- We again train on the shortest and fastest solutions
- Overall, very similar to the OEIS loop in the technology used:
- We use NMT to translate from the program to the inductive predicate

Approach: Self-Learning Loop

Another iterative loop:

1. Train NMT to learn solved problems and predicates.
2. Generate predicates using trained NMT.
3. Attempt proofs with Z3 using generated predicates.
4. Select predicates based on heuristics (size, solution speed).
5. Each iteration: 2-4 ATP runs (1–3 hrs each).
6. Significant discoveries lead to performance jumps.

Technical Details

Grammar for predicates:

$$0, 1, 2, x, y \mid A + B \mid A - B \mid A \times B \mid A \operatorname{div} B \mid A \bmod B$$
$$\operatorname{cond}(A, B, C), \quad \operatorname{loop}(F, A, B),$$
$$\operatorname{loop2}(F, G, A, B, C), \quad \operatorname{compr}(F, A)$$

Filtering via:

- Semantic evaluation.
- Fingerprinting (equivalence checking).
- Removing predicates not dependent on induction variable.

Neural Predicate Generation:

- Neural Machine Translation (Encoder-Decoder with Attention).
- Training data: Encoded problems and induction predicates.
- Beam search width: 240.
- Data augmentation by expanding function definitions and adding rare indices.

Self Learning Loop for Proving Fast=Small

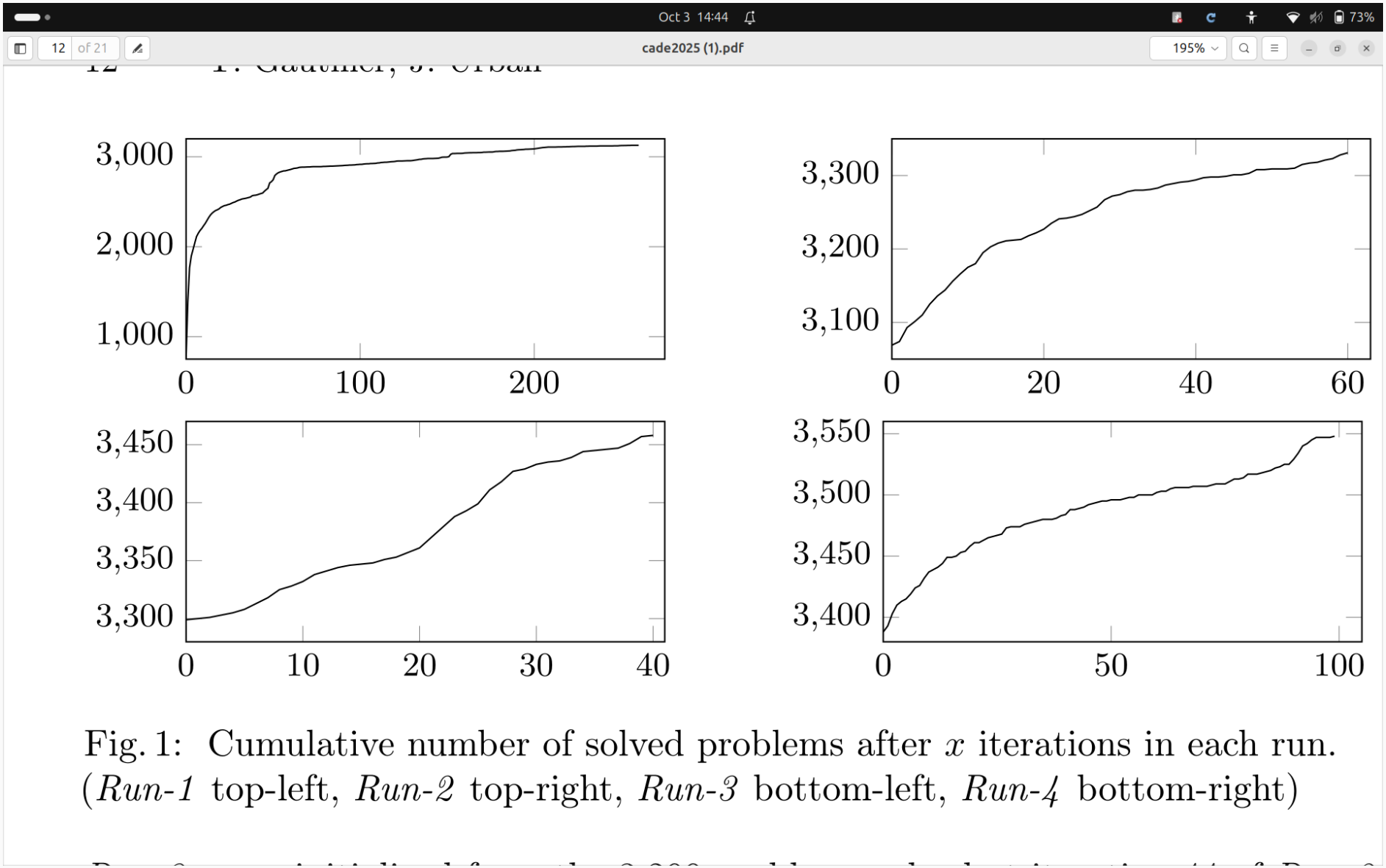


Fig. 1: Cumulative number of solved problems after x iterations in each run. (*Run-1* top-left, *Run-2* top-right, *Run-3* bottom-left, *Run-4* bottom-right)

Example of Influential General Discovery

Discovery at Iteration 41 (*Run-1*) led to predicate:

$$s(x) = s(1) \wedge v(1+x) = w(x) + v(x) + w(x) \qquad \text{(P1)}$$

Applied to program equality: A198766

$$\underbrace{\underbrace{\mathit{loop}(5 \times (2 + X), X, 1)}_{v} + 2}_{} = \underbrace{\underbrace{\mathit{loop2}(X \times Y, Y, X, 7, 5)}_{w,s}}_{} \ \mathit{div}\ 2 \qquad \text{(Eq1)}$$

Sequences:

- $v(0) = 1, \quad v(x+1) = 5v(x) + 10$
- $w(0) = 7,\ s(0) = 5, \quad w(x+1) = w(x) \cdot s(x)$

Generality of Predicate Across OEIS Problems

Other equalities discovered:

$$loop(10X + 4, X, 1) = 2 \cdot (loop2(X \cdot Y, Y, X, 2, 10)\ div\ 9) \qquad \text{(A2278)}$$

$$loop(6X + 6, X, 0) = 2 \cdot (loop2(X \cdot Y, Y, X, 3, 6)\ div\ 5) \qquad \text{(A105281)}$$

Implicit transformations:

- $v(x + 1) - v(x) = 4v(x) + 10,\ 9v(x) + 4,\ 5v(x) + 6$
- Predicate (P1) generalizes these without needing hardcoded constants.

Further Examples

Example 1 A2411: Pentagonal pyramidal numbers (Gaussian sum variation).

Problem: $\underbrace{loop(X+Y,X,0)}_{v} \times X = (((X \times X)+X)\ div\ 2) \times X$

Math: $x \times \sum_{k=1}^{x} k = \frac{x \times x + x}{2} \times x$ Solution: $0 \leq x \wedge x \times x + x = 2 \times v(x)$

The proof requires one predicate on the *loop function* v defining $loop(X+Y,X,0)$. We can also observe that our machine learning algorithm restricted the induction step to nonnegative integers by adding the conjunct $0 \leq x$.[16]

Further Examples

Example 2 A205646: empty faces in Freij's family of Hansen polytopes.
We first simplify the original problem by evaluating constant programs and reducing polynomials. The problem can then be restated as:

$$\underbrace{loop(3 \times X, X, 1)}_{v} + 16 = \underbrace{loop2(X \times Y, Y, X \ div \ 2, loop(3, x \ mod \ 2, 1), 9))}_{w,s} + 16$$

Mathematically, the equation is $3^x + 16 = 3^{(x \ mod \ 2)} \times 9^{(x \ div \ 2)} + 16$. The fast program is saving time by computing 3^x as $3^{(x \ mod \ 2)} \times 9^{(x \ div \ 2)}$ (this is the basis of the *fast exponentiation* algorithm [9]). It stores the value of $9 = 1 + 2 \times (2 + 2)$ in the second component of the loop y to avoid recomputing it at each iteration of the loop. The solution consists of 6 predicates:

$$w(1 + x) = v(1 + x) \wedge w(x) = v(x), v(x) = w(x) \wedge w(1 + x) = v(1 + x)$$
$$x \leq w(2), x \leq w(x), s(x) = s(1), s(x + 1) = s(x)$$

More Human (+ AI) Proofs: Kolakoski

0 1 3 6 2 7 / 23 OEIS 13 20 12 / 10 22 11 21

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES ®

founded in 1964 by N. J. A. Sloane

Search Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

A000002 Kolakoski sequence: a(n) is length of n-th run; a(1) = 1; sequence consists just of 1's and 2's. 298

(Formerly M0190 N0070)

1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2,
1, 1, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 2,
1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 2,
1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2

(list; graph; refs; listen; history; text; internal format)

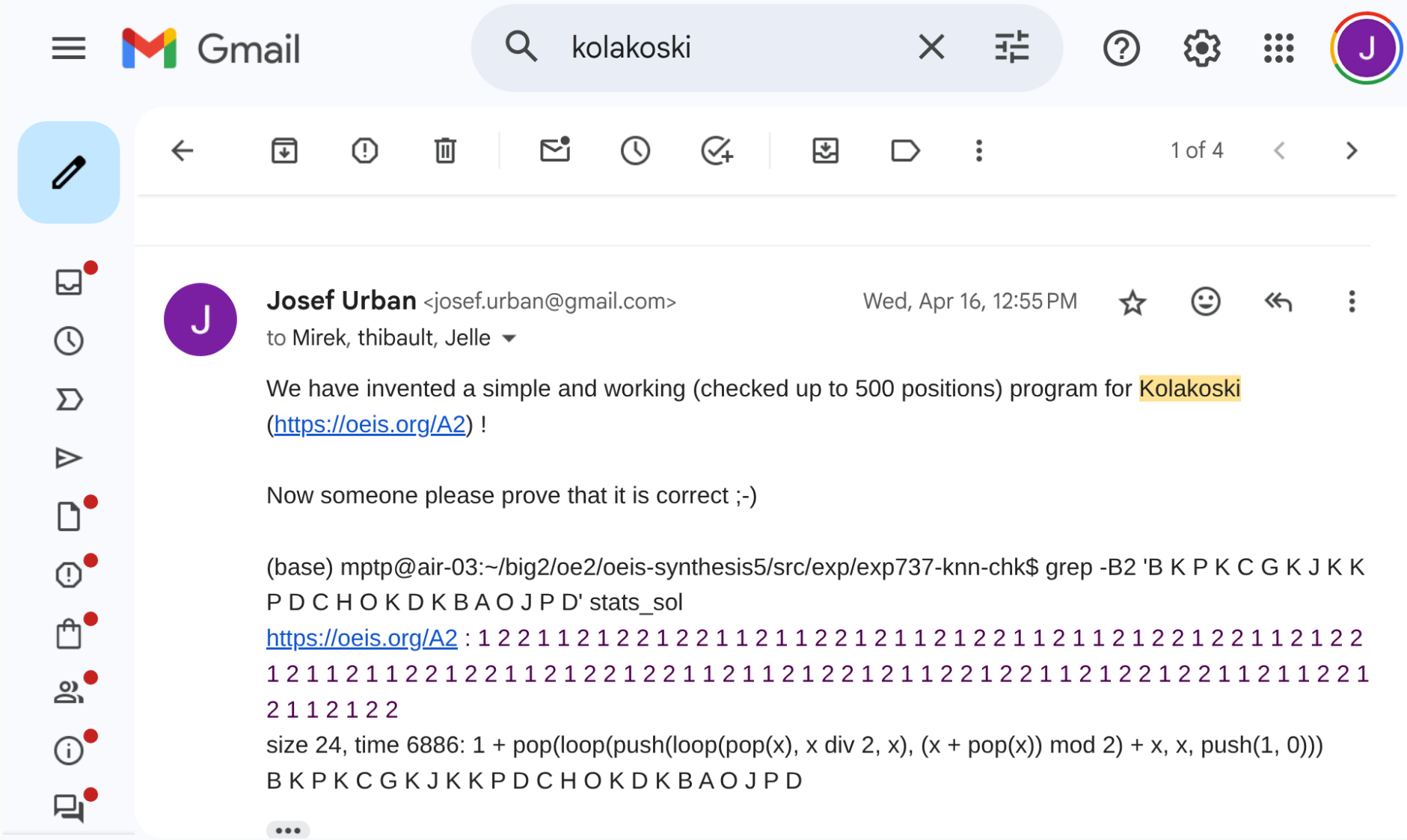
OFFSET 1,2

COMMENTS Historical note: the sequence might be better called the Oldenburger-Kolakoski sequence, since it was discussed by Rufus Oldenburger in 1939; see links. - Clark Kimberling, Dec 06 2012. However, to avoid confusion, this sequence will be known in the OEIS as the Kolakoski sequence. It is undesirable to have some entries refer to the Oldenburger-Kolakoski sequence and others to the Kolakoski sequence. - N. J. A. Sloane, Nov 22 2017

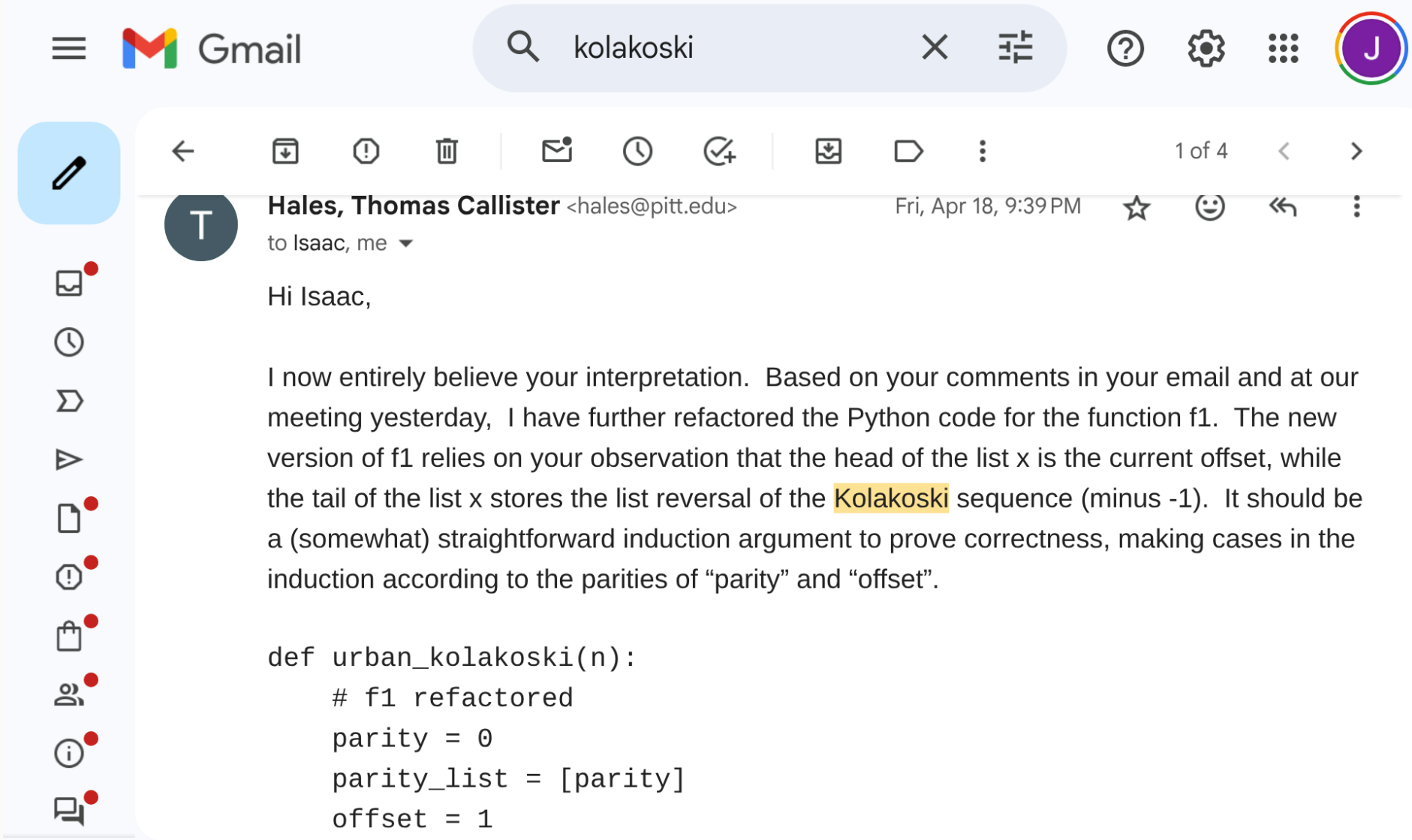
It is an unsolved problem to show that the density of 1's is equal to 1/2.

A weaker problem is to construct a combinatorial bijection between

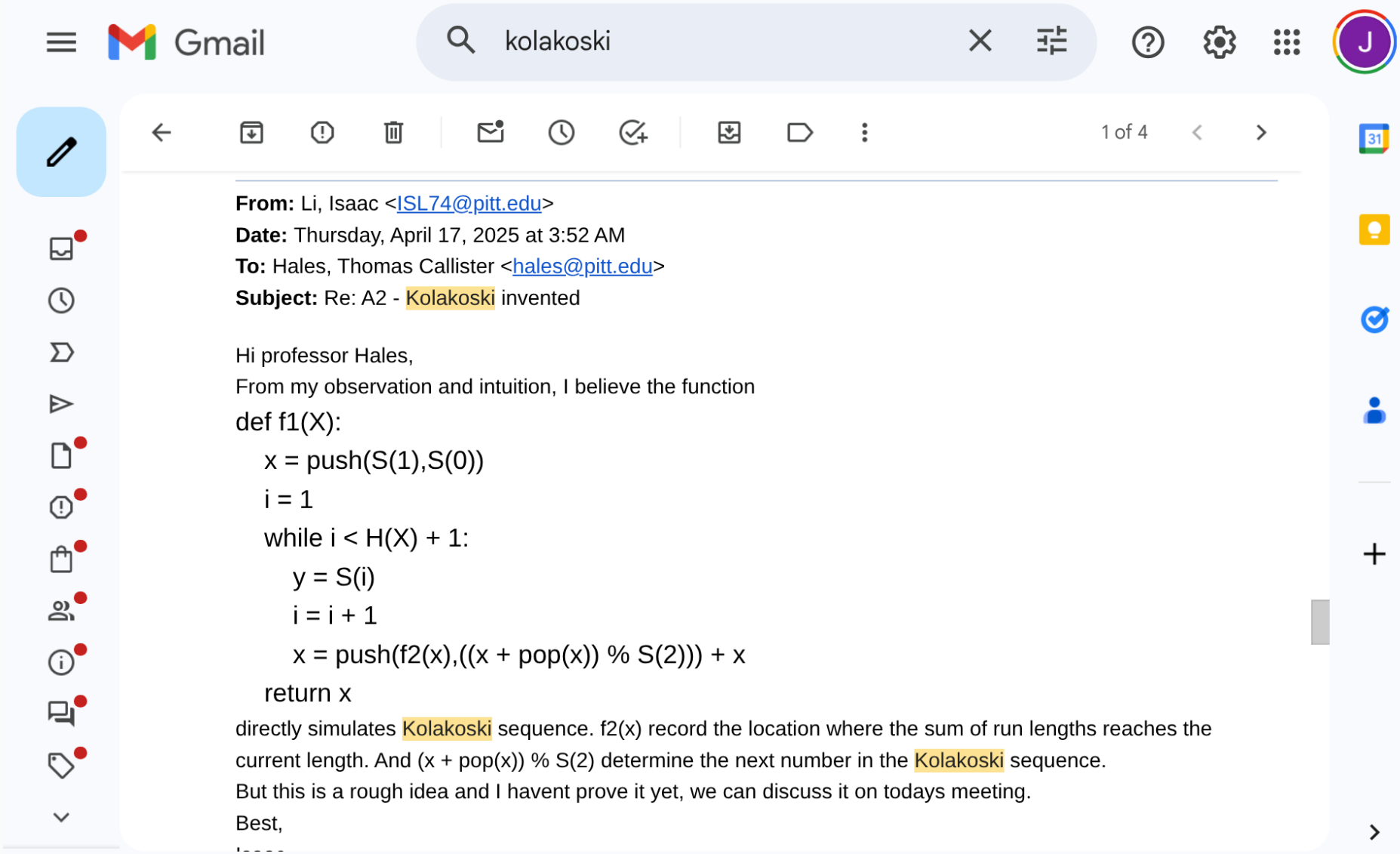
More Human (+ AI) Proofs: Kolakoski



More Human (+ AI) Proofs: Kolakoski



More Human (+ AI) Proofs: Kolakoski



More Human (+ AI) Proofs: Kolakoski

- A2

  Short description: Kolakoski sequence: a(n) is length of n-th run; a(1) = 1; sequence consists just of 1's and 2's.

  Note: This sequence was written up in the Guardian in 2014. The Guardian writes, "Mathematicians drool over this sequence."
  Status: synthesis is probably correct.
  Long description of the Kolakoski sequence $a(n)$: If $x \in \{1, 2\}$, let $\bar{x} = 3 - x \in \{1, 2\}$ be complementation: $\bar{2} = 1$, $\bar{1} = 2$. There are two indices i (the read index) and j (the write index) with $i < j$. Initialize $i = 1$, $j = 2$ and $a(1) = 1$. Assume $a(k)$ has been defined for all $k < j$. The recursion rule is given as follows.

More Human (+ AI) Proofs: Kolakoski

Case 1 (add run of length 1): If $a(i) = 1$, then set $a(j) = \bar{a}(j-1)$ and increment $i \leftarrow i+1$, $j \leftarrow j+1$.
Case 2 (add run of length 2): If $a(i) = 2$, then set $a(j) = a(j+1) = \bar{a}(j-1)$ and increment $i \leftarrow i+1$, $j \leftarrow j+2$.

Synthesized code in Python (after significant refactoring) is equivalent to the following

```
def synthetic_kolakoski(n):
    parity = 0
    parity_list = [parity]
    offset = 1
    for _ in range(n-1):
        parity = (offset + parity) % 2
        parity_list = parity_list + [parity]
        offset += parity_list[-1-(offset // 2)]
    # add 1 to get kolakoski
    return [parity+1 for parity in parity_list]
```

More Human (+ AI) Proofs: Kolakoski


```
def synthetic_kolakoski(n):
    parity = 0
    parity_list = [parity]
    offset = 1
    for _ in range(n-1):
        parity = (offset + parity) % 2
        parity_list = parity_list + [parity]
        offset += parity_list[-1-(offset // 2)]
    # add 1 to get kolakoski
    return [parity+1 for parity in parity_list]
```

Credit: Isaac Li was the first to understand how the synthetic program works. The refactored synthesized code here is based on his explanation of the synthetic program.

The basic idea of the synthetic code is that the offset is even exactly at the beginning of a Kolakoski run of length 2. In this way, with even offset, the update instruction `parity = (offset + parity) % 2` reduces to `parity = parity` so that to `parity_list` are appended two consecutive equal values. Otherwise, the offset is odd, and the instruction `parity = (offset+parity) % 2` reverses the parity, so that alternating parities are appended to `parity_list`, producing runs of length 1. Also `offset//2` is tracking the growing gap $j - i$ between the read index i and the write index j.

To do: state the precise relationship among `j, i, offset`. Note that the gap $j - i$ increases by one each time case 2 of the recursion is executed, which happens exactly when a run of length 2 is created. Also, `offset//2` increases by one each time the offset is incremented from an odd integer to the next even integer. This also occurs exactly when a run of length 2 is created.

More Human (+ AI) Proofs: Cyclotomic Polynomials

A070526 Value of n-th cyclotomic polynomial at 2^n. 3

1, 5, 73, 257, 1082401, 4033, 4432676798593, 4294967297, 18014398643699713, 1098438933505, 1298708349570020393652962442872833, 281474959933441, 9135500406707633916741382424010949897006927872 1

(list; graph; refs; listen; history; text; internal format)

OFFSET 1,2

LINKS Table of n, a(n) for n=1..13.

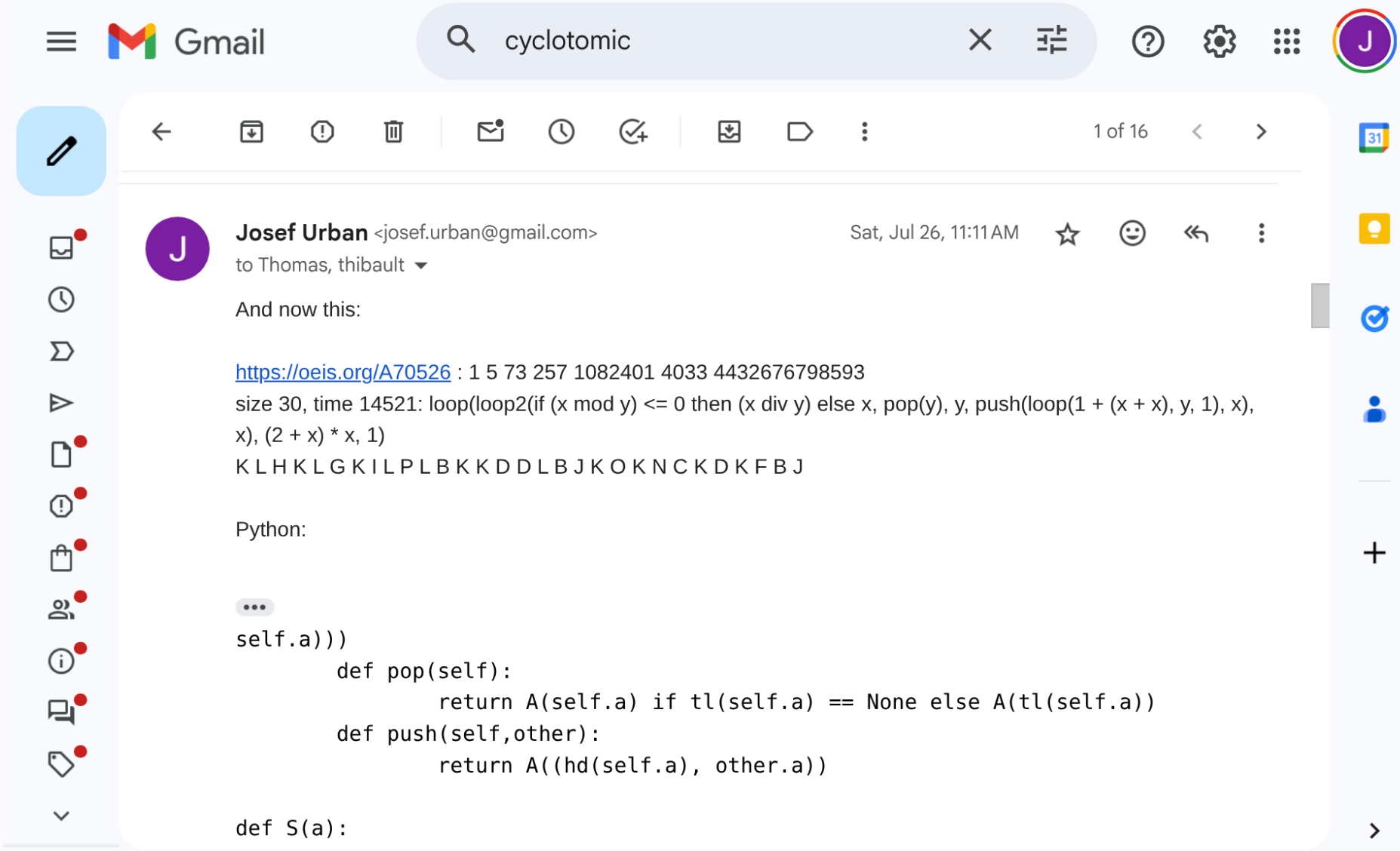
EXAMPLE n=5: a(5)=C[5,32]=1+32+1024+32768+1048576=1082401

MATHEMATICA Table[Cyclotomic[w, 2^w], {w, 1, 15}]

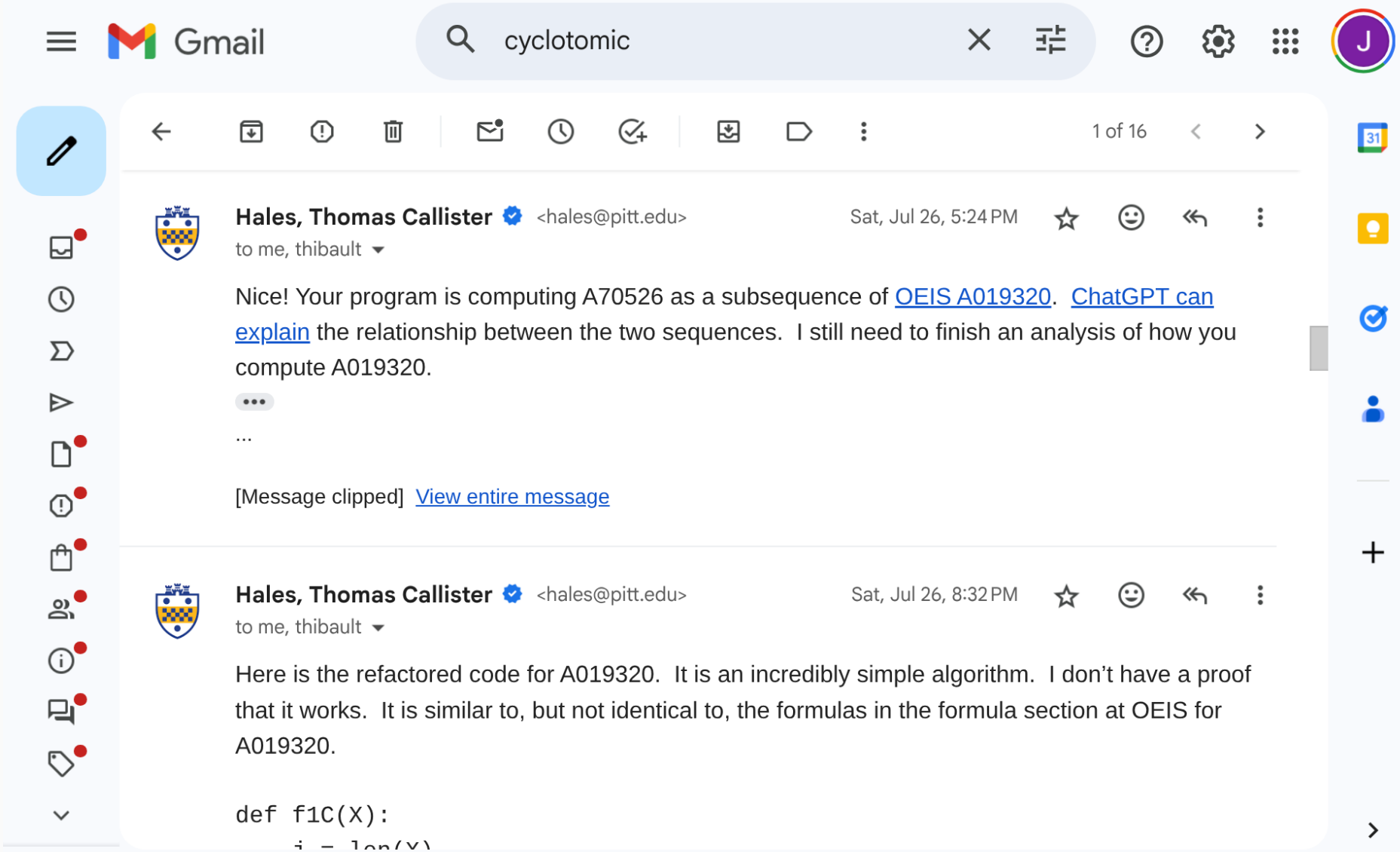
CROSSREFS Cf. A070518-A070525.
Sequence in context: A383924 A138623 A206280 * A070530 A248046 A059017
Adjacent sequences: A070523 A070524 A070525 * A070527 A070528 A070529

KEYWORD easy,nonn

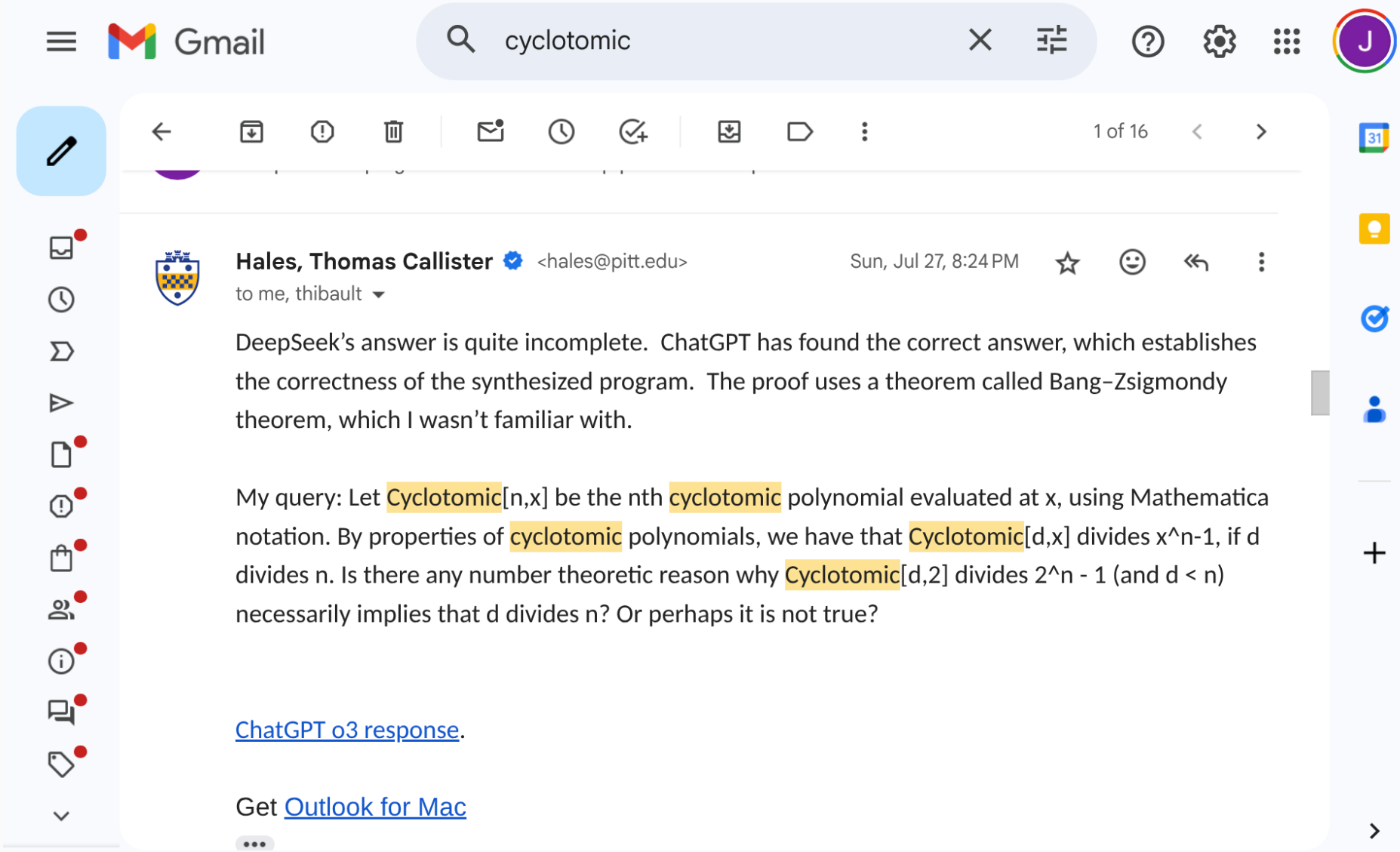
More Human (+ AI) Proofs: Cyclotomic Polynomials



More Human (+ AI) Proofs: Cyclotomic Polynomials



More Human (+ AI) Proofs: Cyclotomic Polynomials



More Human (+ AI) Proofs: Cyclotomic Polynomials

- A70526
  Short description: Value of n-th cyclotomic polynomial at 2^n.
  Verdict: The synthesis is correct and remarkable.
  Analysis: Let $\Phi_n(x)$ be the nth cyclotomic polynomial evaluated at x. An examination of the code reveals that the synthetic procedure computes $\Phi_k(2)$ and uses an identity $\Phi_{n^2}(2) = \Phi_n(2^n)$ to extract sequence A70526 as a subsequence $k = n^2$ of $k \mapsto \Phi_k(2)$. We have $\Phi_{n^2}(2) = \Phi_n(2^n)$ because for all x, we have $\Phi_{n^2}(x) = \Phi_n(x^n)$ (because both monic polynomials have roots at exactly the primitive n^2-roots of unity). It is remarkable that the synthesis uses this identity as an empirically discovered fact.

  We now give an analysis of the sequence $\Phi_n(2)$, which is A019320.
  Short description: Cyclotomic polynomials at $x = 2$.
  Analysis: An examination of the code reveals that the synthetic procedure is as follows. Suppose that $\Phi_j(2)$ has been computed for $j < n$. Initialize $x = 2^n - 1$. For each $j = n - 1$ down to $j = 1$, if $x \mod \Phi_j(2) = 0$, replace x with $x/\Phi_j(2)$. The the terminal value of x after the loop is $x = \Phi_n(2)$.

  Why does this work? According to an explanation provided by ChatGPT-o3, the reason is based on the Bang-Zsigmondy (BZ) theorem. (Urban tried the same query in Deepseek, which was unable to provide a satisfactory explanation.)

  > (ChatGPT Query)
  > Let Cyclotomic[n,x] be the nth cyclotomic polynomial evaluated at x, using Mathematica notation. By properties of cyclotomic polynomials, we have that Cyclotomic[d,x] divides $x^n - 1$, if d divides n. Is there any number theoretic reason why Cyclotomic[d,2] divides $2^n - 1$ (and $d < n$) necessarily implies that d divides n? Or

More Human (+ AI) Proofs: Cyclotomic Polynomials

perhaps it is not true?

(ChatGPT-o3 response, thought for 1m 2s)
It's *almost* true, but not quite.

Counterexample. Take $d = 6$. Then $\Phi_6(x) = x^2 - x + 1$, so $\Phi_6(2) = 3$. Since 2 has order 2 modulo 3, we have $3 \mid 2^n - 1$ iff n is even. In particular, with $n = 8$ we get

$$\Phi_6(2) = 3 \mid 2^8 - 1 = 255, \qquad \text{but } 6 \nmid 8.$$

Here $d = 6 < n = 8$, yet $d \nmid n$.

Why this is the only exception. For $d > 1$ and $d \neq 6$, the Bang-Zsigmondy theorem guarantees a *primitive prime divisor* p of $2^d - 1$: a prime p dividing $2^d - 1$ but no $2^k - 1$ with $k < d$. Such a prime necessarily divides $\Phi_d(2)$, and the multiplicative order of 2 modulo p is exactly d. Therefore, if $\Phi_d(2) \mid 2^n - 1$, then in particular $p \mid 2^n - 1$, so the order of 2 mod p divides n; hence $d \mid n$.

The only failure of this argument for base 2 is at $d = 6$, because $2^6 - 1 = 63 = 3^2 \cdot 7$ has *no* primitive prime divisor (both $3 \mid 2^2 - 1$ and $7 \mid 2^3 - 1$). Consequently $\Phi_6(2) = 3$ is made up entirely of "non-primitive" primes, and the implication $\Phi_d(2) \mid 2^n - 1 \Rightarrow d \mid n$ can fail.

Conclusion. Apart from the unique exceptional case $d = 6$, if $\Phi_d(2) \mid 2^n - 1$ then indeed $d \mid n$. The added hypothesis $d < n$ is not needed for the statement to hold (or fail) in these cases.

More Human (+ AI) Proofs: Cyclotomic Polynomials

$\Phi_d(2) \mid 2^n - 1 \Rightarrow d \mid n$ can fail.

Conclusion. Apart from the unique exceptional case $d = 6$, if $\Phi_d(2) \mid 2^n - 1$ then indeed $d \mid n$. The added hypothesis $d < n$ is not needed for the statement to hold (or fail) in these cases.

The BZ theorem implies that for all $j < n$ with $j \neq 6$, we have $\Phi_j(2)$ divides $2^n - 1$ if divides n. Also

$$x^n - 1 = \Pi_{j|n}\Phi_j(x), \quad \text{so that} \quad \Phi_n(2) = \frac{2^n - 1}{\Pi_{j|n, j<n}\Phi_j(2)}.$$

By BZ, the right-hand-side of this expression is what the synthetic program computes, w an exception for $j = 6$, with $\Phi_6(2) = 3$, which potentially throws the answer off by a factor 3. Remarkably, this potential problem never occurs, and the synthetic program always gi the correct result. We have $\Phi_2(2) = \Phi_2(6) = 3$. Sometimes the synthetic program divides $\Phi_2(6) = 3$ (the *wrong 3*) but then not by $\Phi_2(2) = 3$ (the *right 3*), but this does not affect t correctness of the results. This division *"by the wrong 3"* occurs when 6 does not divide and 3 divides $2^n - 1$. Equivalently, this occurs when n is congruent to 2 or 4 mod 6. In th cases, $2^n - 1$ is not divisible by 9.

Evolution and Proliferation of Primes and Others

`https://bit.ly/3XHZsjK`: triangle coding, sigma (sum of divisors), primes. `https://bit.ly/3iJ4oGd` (the first 24, now 50)

Nr	Program
P1	(if x <= 0 then 2 else 1) + (compr (((loop (x + x) (x mod 2) (loop (x * x) 1 (loop (x + x) (x div 2) 1))) + x) mod (1 + x)) x)
P2	1 + (compr ((((loop (x * x) 1 (loop (x + x) (x div 2) 1)) + x) * x) mod (1 + x)) (1 + x))
P3	1 + (compr (((loop (x * x) 1 (loop (x + x) (x div 2) 1)) + x) mod (1 + x)) (1 + x))
P4	2 + (compr ((loop2 (1 + (if (x mod (1 + y)) <= 0 then 0 else x)) (y - 1) x 1 x) mod (1 + x)) x)
P5	1 + (compr ((loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) x (1 + x)) mod (1 + x)) (1 + x))
P6	1 + (compr ((loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) (2 + (x div (2 + (2 + 2)))) (1 + x)) mod (1 + x)) (1 + x))
P7	compr ((1 + (loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) x x)) mod (1 + x)) (2 + x)
P8	1 + (compr ((loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) (1 + ((2 + x) div (2 + (2 + 2)))) (1 + x)) mod (1 + x)) (1 + x))
P9	compr (x - (loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) x x)) (2 + x)
P10	compr (x - (loop (if (x mod (1 + y)) <= 0 then 2 else x) (x div 2) x)) (2 + x)
P11	1 + (compr ((loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) (1 + (x div (2 + (2 + 2)))) (1 + x)) mod (1 + x)) (1 + x))
P12	compr ((x - (loop (if (x mod (1 + y)) <= 0 then y else x) x x)) - 2) (2 + x)
P13	1 + (compr ((loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) (2 + (x div (2 * (2 + (2 + 2))))) (1 + x)) mod (1 + x)) (1 + x))
P14	compr ((x - (loop (if (x mod (1 + y)) <= 0 then y else x) x x)) - 1) (2 + x)
P15	1 + (compr (x - (loop (if (x mod (1 + y)) <= 0 then (1 + y) else x) (2 + (x div (2 * (2 + (2 + 2))))) (1 + x))) (1 + x))
P16	compr (2 - (loop (if (x mod (1 + y)) <= 0 then 0 else x) (x - 2) x)) x
P17	1 + (compr (x - (loop (if (x mod (1 + y)) <= 0 then 2 else x) (2 + (x div (2 * (2 + (2 + 2))))) (1 + x))) (1 + x))
P18	1 + (compr (x - (loop (if (x mod (1 + y)) <= 0 then 2 else x) (1 + (2 + (x div (2 * (2 * (2 + 2)))))) (1 + x))) (1 + x))
P19	1 + (compr (x - (loop2 (loop (if (x mod (1 + y)) <= 0 then 2 else x) (2 + (y div (2 * (2 + (2 + 2))))) (1 + y)) 0 (1 - (x mod 2)) 1 x)) (1 + x))
P20	1 + (compr (x - (loop2 (loop (if (x mod (1 + y)) <= 0 then 2 else x) (1 + (2 + (y div (2 * (2 * (2 + 2)))))) (1 + y)) 0 (1 - (x mod 2)) 1 x)) (1 + x))
P21	1 + (compr (x - (loop2 (loop (if (x mod (2 + y)) <= 0 then 2 else x) (2 + (y div (2 * ((2 + 2) + (2 + 2))))) (1 + y)) 0 (1 - (x mod 2)) 1 x)) (1 + x))
P22	1 + (compr (x - (loop2 (loop (if (x mod (2 + y)) <= 0 then 2 else x) (2 + (y div (2 * (2 * (2 + 2))))) (1 + y)) 0 (1 - (x mod 2)) 1 x)) (1 + x))
P23	2 + (compr (loop (x - (if (x mod (1 + y)) <= 0 then 0 else 1)) x x) x)
P24	loop (1 + x) (1 - x) (1 + (2 * (compr (x - (loop (if (x mod (2 + y)) <= 0 then 1 else x) (2 + (x div (2 * (2 + 2)))) (1 + (x + x)))) x)))

Evolution and Proliferation of Primes

Iter	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	6	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	8	1	6	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	12	4	6	6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	7	12	6	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	4	10	6	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	3	4	6	0	18	6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	2	3	1	0	12	18	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	2	3	1	0	9	56	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	2	5	2	0	7	59	49	9	1	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0
41	1	2	3	0	4	52	58	42	23	0	13	0	8	0	0	0	0	0	0	0	0	0	0	0
42	0	2	4	0	3	44	50	38	60	8	11	0	55	0	0	0	0	0	0	0	0	0	0	0
43	0	2	12	0	0	37	55	14	116	35	16	7	90	0	0	0	0	0	0	0	0	0	0	0
44	0	2	13	0	0	28	40	6	176	73	19	8	122	9	12	0	0	0	0	0	0	0	0	0
45	0	2	9	0	0	19	24	4	147	185	26	16	94	25	29	0	7	0	0	0	0	0	0	0
46	0	2	4	0	0	11	14	0	101	256	21	14	66	64	30	0	29	0	0	0	0	0	0	0
47	0	0	0	0	0	9	4	0	55	290	23	3	43	116	16	6	62	14	0	0	0	0	0	0
48	0	0	0	0	0	8	0	0	22	261	16	0	34	192	10	6	89	30	0	0	0	0	0	0
49	0	0	0	0	0	8	0	0	6	195	11	0	36	225	8	6	99	34	0	0	0	0	0	0
50	0	0	0	0	0	5	0	0	2	154	8	0	29	168	6	6	108	39	0	0	0	0	0	0
51	0	0	0	0	0	4	0	0	0	121	7	0	21	97	6	6	113	43	0	0	0	0	0	0
52	0	0	0	0	0	2	0	0	0	118	8	0	12	62	6	6	110	51	0	0	0	0	0	0
53	0	0	0	0	0	1	0	0	0	59	7	0	15	33	6	6	125	62	0	0	0	0	0	0
54	0	0	0	0	0	1	0	0	0	41	4	0	16	17	6	9	137	72	0	0	0	0	0	0
55	0	0	0	0	0	2	0	0	0	32	4	0	15	9	6	17	147	82	0	0	0	0	0	0
56	0	0	0	0	0	1	0	0	0	29	4	0	10	7	6	39	152	98	0	0	0	0	0	0

More Bragging

- Hofstadter-Conway $10000 sequence: a(n) = a(a(n-1)) + a(n-a(n-1)) with a(1) = a(2) = 1.
- D. R. Hofstadter, Analogies and Sequences: Intertwined Patterns of Integers and Patterns of Thought Processes, Lecture in DIMACS Conference on Challenges of Identifying Integer Sequences, 2014.

```
Date: Sun, Mar 17, 2024
To: <dughof@indiana.edu>

Dear Douglas,

our system [1] has today (iteration 552) found a solution of
https://oeis.org/A004074. The solution in Thibault's programming
language [1] (with push/pop added on top of [1]) is:

((2*loop(push(loop(pop(x),x-1,x),x)+loop(pop(x),y-x,pop(x)),x-1,1))-1)-x

The related A4001 was solved in iteration 463 and the solution is:
loop(push(loop(pop(x), y-x,pop(x)),x) + loop(pop(x), x-1, x), x - 1, 1)
```

Minsky 2014

It seems to me that the most important discovery since Gödel was the discovery by Chaitin, Solomonoff and Kolmogorov of the concept called Algorithmic Probability which is a fundamental new theory of how to make predictions given a collection of experiences and this is a beautiful theory, everybody should learn it,

but it's got one problem, that is, that you cannot actually calculate what this theory predicts because it is too hard, it requires an infinite amount of work.

However, it should be possible to make practical approximations to the Chaitin, Kolmogorov, Solomonoff theory that would make better predictions than anything we have today. Everybody should learn all about that and spend the rest of their lives working on it.

– M. Minsky, Panel discussion on The Limits of Understanding, 2014

References to our relevant work

- Thibault Gauthier, Josef Urban: Learning Program Synthesis for Integer Sequences from Scratch. AAAI 2023: 7670-7677
- Thibault Gauthier, Miroslav Olsák, Josef Urban: Alien Coding. CoRR abs/2301.11479 (2023)
- Thibault Gauthier, Chad E. Brown, Mikolas Janota, Josef Urban: A Mathematical Benchmark for Inductive Theorem Provers. LPAR 2023: 224-237
- Thibault Gauthier, Cezary Kaliszyk, Josef Urban: TacticToe: Learning to Reason with HOL4 Tactics. LPAR 2017: 125-143
- Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, Michael Norrish: Learning to Prove with Tactics. CoRR abs/1804.00596 (2018)
- Jan Jakubuv, Karel Chvalovský, Zarathustra Amadeus Goertzel, Cezary Kaliszyk, Mirek Olsák, Bartosz Piotrowski, Stephan Schulz, Martin Suda, Josef Urban: MizAR 60 for Mizar 50. ITP 2023: 19:1-19:22
- Thibault Gauthier: Deep Reinforcement Learning for Synthesizing Functions in Higher-Order Logic. LPAR 2020: 230-248
- Thibault Gauthier: Tree Neural Networks in HOL4. CICM 2020: 278-283
- Qingxiang Wang, Cezary Kaliszyk, Josef Urban: First Experiments with Neural Translation of Informal to Formal Mathematics. CICM 2018: 255-270
- Bartosz Piotrowski, Josef Urban: Stateful Premise Selection by Recurrent Neural Networks. LPAR 2020: 409-422
- Bartosz Piotrowski, Josef Urban: Guiding Inferences in Connection Tableau by Recurrent Neural Networks. CICM 2020: 309-314
- Josef Urban, Jan Jakubuv: First Neural Conjecturing Datasets and Experiments. CICM 2020: 315-323
- Bartosz Piotrowski, Josef Urban, Chad E. Brown, Cezary Kaliszyk: Can Neural Networks Learn Symbolic Rewriting? CoRR abs/1911.04873 (2019)

Thanks and Advertisement

- Thanks for your attention!
- To push AI/ML methods in math and theorem proving, we organize:
- 11th AITP – Artificial Intelligence and Theorem Proving
- August 30 - September 4, 2026, Aussois, France, `aitp-conference.org`
- ATP/ITP/Math vs AI/ML/AGI people, Computational linguists
- Discussion-oriented and experimental
- About 50 people in 2025
- Invited talks by people who do AI/ML/TP/AGI for math, physics, ...