Learning Program Synthesis for Integer Sequences

Siddharth Bhat

MSR
AI4Code

July 31st, 2023

The Problem

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.

The Problem

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.

A000142	Factorial numbers: n! = 1*2*3*4*...*n (order of symmetric group S_n, number of permutations of n letters). (Formerly M1675 N0659)	2630

The Problem

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.

A000142	Factorial numbers: n! = 1*2*3*4*...*n (order of symmetric group S_n, number of permutations of n letters). (Formerly M1675 N0659)	2630

1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, 1124000727777607680000 (list; graph; refs; listen; history; text; internal

The Problem

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.

A000142	Factorial numbers: n! = 1*2*3*4*...*n (order of symmetric group S_n, number of permutations of n letters). (Formerly M1675 N0659)	2630

1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, 1124000727777607680000 (list; graph; refs; listen; history; text; internal

PROG

```
(Axiom) [factorial(n) for n in 0..10]
(Magma) a:= func< n | Factorial(n) >; [ a(n) : n in [0..10]];
(Haskell)
a000142 :: (Enum a, Num a, Integral t) => t -> a
a000142 n = product [1 .. fromIntegral n]
a000142_list = 1 : zipWith (*) [1..] a000142_list
-- Reinhard Zumkeller, Mar 02 2014, Nov 02 2011, Apr 21 2011
(Python)
for i in range(1, 1000):
    y = i
    for j in range(1, i):
       y *= i - j
    print(y, "\n")
(Python)
import math
for i in range(1, 1000):
    math.factorial(i)
    print("")
# Ruskin Harding, Feb 22 2013
(PARI) a(n)=prod(i=1, n, i) \\ Felix Fröhlich, Aug 17 2014
(PARI) {a(n) = if(n<0, 0, n!)}; /* Michael Somos, Mar 04 2004 */
(Sage) [factorial(n) for n in (1..22)] # Giuseppe Coppoletta, Dec 05 2014
(GAP) List([0..22], Factorial); # Muniru A Asiru, Dec 05 2018
(Scala) (1: BigInt).to(24: BigInt).scanLeft(1: BigInt)(_ * _) // Alonso del Arte,
  Mar 02 2019
```

The Problem

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.

A000142	Factorial numbers: n! = 1*2*3*4*...*n (order of symmetric group S_n, number of permutations of n letters). (Formerly M1675 N0659)	2630

1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, 1124000727777607680000 (list; graph; refs; listen; history; text; internal

PROG

```
(Axiom) [factorial(n) for n in 0..10]
(Magma) a:= func< n | Factorial(n) >; [ a(n) : n in [0..10]];
(Haskell)
a000142 :: (Enum a, Num a, Integral t) => t -> a
a000142 n = product [1 .. fromIntegral n]
a000142_list = 1 : zipWith (*) [1..] a000142_list
-- Reinhard Zumkeller, Mar 02 2014, Nov 02 2011, Apr 21 2011
(Python)
for i in range(1, 1000):
    y = i
    for j in range(1, i):
       y *= i - j
    print(y, "\n")
(Python)
import math
for i in range(1, 1000):
    math.factorial(i)
    print("")
# Ruskin Harding, Feb 22 2013
(PARI) a(n)=prod(i=1, n, i) \\ Felix Fröhlich, Aug 17 2014
(PARI) {a(n) = if(n<0, 0, n!)}; /* Michael Somos, Mar 04 2004 */
(Sage) [factorial(n) for n in (1..22)] # Giuseppe Coppoletta, Dec 05 2014
(GAP) List([0..22], Factorial); # Muniru A Asiru, Dec 05 2018
(Scala) (1: BigInt).to(24: BigInt).scanLeft(1: BigInt)(_ * _) // Alonso del Arte,
  Mar 02 2019
```

The Problem Statement

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.

The Problem Statement

- Given an OEIS sequence s, generate a program p such that $s_i = p(i)$.
- What language is p written in?
- What is our training data set of (s, p) pairs?
- What is our architecture for generating a program p_t for a novel sequence t during test time?

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
```

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
```

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
```

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
P :=
| loop(f, a, b)
# loop(F, P, P) :=
#   while a > 0:
#     b = f(a, b)
#     a -= 1
#   return b;
```

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
P :=
| loop(f, a, b)
# loop(F, P, P) :=
#   while a > 0:
#     b = f(a, b)
#     a -= 1
#   return b;
```

- For OEIS example, let sequence be $s_n \equiv n!$.

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
P :=
| loop(f, a, b)
# loop(F, P, P) :=
#   while a > 0:
#     b = f(a, b)
#     a -= 1
#   return b;
```

- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
P :=
| loop(f, a, b)
# loop(F, P, P) :=
#   while a > 0:
#     b = f(a, b)
#     a -= 1
#   return b;
```

- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
P :=
| loop(f, a, b)
# loop(F, P, P) :=
#   while a > 0:
#     b = f(a, b)
#     a -= 1
#   return b;
```

- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- This is the program `loop(λ(counter,accum). counter*accum, counter=n, start=1)`.

Programming Language

```
P := 0 | 1 | x | y | P + P | P - P | P * P | P div P | P mod P
   |  cond(P, P, P) # cond(c, t, e) := return t if c else e
F := lambda (x, y). P
P :=
| loop(f, a, b)
# loop(F, P, P) :=
#   while a > 0:
#     b = f(a, b)
#     a -= 1
#   return b;
```

- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- This is the program `loop(λ(counter,accum). counter*accum, counter=n, start=1)`.

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- []

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{\mathbf{1}} [1]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{\mathbf{1}} [1] \mapsto_{\mathbf{2}} [1; 2] \mapsto_{\mathbf{3}} [1; 2; 3] \mapsto_{*} [1; (2 * 3)]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{1} [1] \mapsto_{2} [1; 2] \mapsto_{3} [1; 2; 3] \mapsto_{*} [1; (2 * 3)] \mapsto_{+} [1 + (2 * 3)]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{\mathtt{1}} [1] \mapsto_{\mathtt{2}} [1; 2] \mapsto_{\mathtt{3}} [1; 2; 3] \mapsto_{*} [1; (2 * 3)] \mapsto_{+} [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{1} [1] \mapsto_{2} [1;2] \mapsto_{3} [1;2;3] \mapsto_{*} [1;(2*3)] \mapsto_{+} [1+(2*3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{\mathtt{1}} [1] \mapsto_{\mathtt{2}} [1; 2] \mapsto_{\mathtt{3}} [1; 2; 3] \mapsto_{*} [1; (2 * 3)] \mapsto_{+} [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_{\mathtt{1}} [1]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_{\mathtt{1}} [1] \mapsto_{\mathtt{2}} [1; 2] \mapsto_{\mathtt{3}} [1; 2; 3] \mapsto_{*} [1; (2 * 3)] \mapsto_{+} [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_{\mathtt{1}} [1] \mapsto_{\mathtt{2}} [1; 2]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_3 [1;2;3] \mapsto_* [1;(2*3)] \mapsto_+ [1+(2*3)]$
- $(1+2)*3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_+ [(1+2)]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_3 [1;2;3] \mapsto_* [1;(2*3)] \mapsto_+ [1+(2*3)]$
- $(1+2)*3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_+ [(1+2)] \mapsto_3 [(1+2);3] \mapsto_* [(1+2)*3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x] \mapsto_y [x; y]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_3 [1;2;3] \mapsto_* [1;(2*3)] \mapsto_+ [1+(2*3)]$
- $(1+2)*3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_+ [(1+2)] \mapsto_3 [(1+2);3] \mapsto_* [(1+2)*3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x] \mapsto_y [x;y] \mapsto_* [x*y] \mapsto_x [x*y;x]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_3 [1;2;3] \mapsto_* [1;(2*3)] \mapsto_+ [1+(2*3)]$
- $(1+2)*3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1;2] \mapsto_+ [(1+2)] \mapsto_3 [(1+2);3] \mapsto_* [(1+2)*3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x] \mapsto_y [x;y] \mapsto_* [x*y] \mapsto_x [x*y;x] \mapsto_1 [x*y;x;1] \mapsto_{loop} [loop(\lambda(x,y).x*y,x,1)]$
- Generate program, test if program matches sequence for some finite sequence length (say, 100). Check that $s_i =_? p(i)$ for $0 \leqslant i \leqslant 100$.

Dataset Generation: A Program for an OEIS sequence s

- Idea: generate a program in reverse polish notation!
- $1 + (2 * 3) \mapsto$ `"1 2 3 * +"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_3 [1; 2; 3] \mapsto_* [1; (2 * 3)] \mapsto_+ [1 + (2 * 3)]$
- $(1 + 2) * 3 \mapsto$ `"1 2 + 3 *"`
- $[] \mapsto_1 [1] \mapsto_2 [1; 2] \mapsto_+ [(1 + 2)] \mapsto_3 [(1 + 2); 3] \mapsto_* [(1 + 2) * 3]$
- For OEIS example, let sequence be $s_n \equiv n!$.
- This is given by the recurrence $s_0 = 1$, $s_{n+1} = n \cdot s_n$
- This is the program `loop(λ(x,y).x*y, x, 1)`.
- A valid program generation is $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$
- Generate program, test if program matches sequence for some finite sequence length (say, 100). Check that $s_i =_? p(i)$ for $0 \leqslant i \leqslant 100$.
- Randomly explore space of possible programs.

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \ldots]$).

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \dots]$).
- We found the action sequence $[] \mapsto_x [x]$

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \dots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y]$

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \dots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \ldots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.
- This is the program $p \equiv$ `loop(λ(x,y).x*y, x, 1)` such that $s_i = p(i)$.

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \dots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.
- This is the program $p \equiv$ `loop(λ(x,y).x*y, x, 1)` such that $s_i = p(i)$.
- We want to learn the function:

Program Stack	Sequence	Next Action
$[]$	$[1, 2, 6, 24, \dots]$	$\mapsto_x$

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \ldots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.
- This is the program $p \equiv$ `loop(λ(x,y).x*y, x, 1)` such that $s_i = p(i)$.
- We want to learn the function:

Program Stack	Sequence	Next Action
$[]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_x$
$[x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_y$

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \ldots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.
- This is the program $p \equiv$ `loop(λ(x,y).x*y, x, 1)` such that $s_i = p(i)$.
- We want to learn the function:

Program Stack	Sequence	Next Action
$[]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_x$
$[x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_y$
$[x; y]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_*$
$[x * y; x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_1$
$[x * y; x; 1]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_{loop}$
$[loop(\lambda(x, y).x * y, x, 1)]$	$[1, 2, 6, \ldots]$	$\mapsto_\blacksquare$

- Learn this function using an NN.

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \ldots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.
- This is the program $p \equiv$ `loop(λ(x,y).x*y, x, 1)` such that $s_i = p(i)$.
- We want to learn the function:

Program Stack	Sequence	Next Action
$[]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_x$
$[x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_y$
$[x; y]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_*$
$[x * y; x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_1$
$[x * y; x; 1]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_{loop}$
$[loop(\lambda(x, y).x * y, x, 1)]$	$[1, 2, 6, \ldots]$	$\mapsto_\blacksquare$

- Learn this function using an NN.
- **Input**: embedding of current program stack + embedding of first 16 terms of OEIS sequence.

Training from the Dataset

- Consider the sequence $s_i \equiv i!$. ($s \equiv [1, 2, 6, 24, \ldots]$).
- We found the action sequence $[] \mapsto_x [x] \mapsto_y [x; y] \mapsto_* [x * y] \mapsto_x [x * y; x] \mapsto_1 [x * y; x; 1] \mapsto_{loop} [loop(\lambda(x, y).x * y, x, 1)]$.
- This is the program $p \equiv$ `loop(λ(x,y).x*y, x, 1)` such that $s_i = p(i)$.
- We want to learn the function:

Program Stack	Sequence	Next Action
$[]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_x$
$[x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_y$
$[x; y]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_*$
$[x * y; x]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_1$
$[x * y; x; 1]$	$[1, 2, 6, 24, \ldots]$	$\mapsto_{loop}$
$[loop(\lambda(x, y).x * y, x, 1)]$	$[1, 2, 6, \ldots]$	$\mapsto_{\blacksquare}$

- Learn this function using an NN.
- **Input**: embedding of current program stack + embedding of first 16 terms of OEIS sequence.
- **Output:** one-hot vector over next actions.

Embeddings in Greater Detail

- `[x*y;x]   [1, 2, 6, 24]` $\mapsto_1 ?$

Embeddings in Greater Detail

- `[x*y;x]` `[1, 2, 6, 24]` $\mapsto_1$?

Embedding Computation

`action_onehot = NNPolicy(vprogram, vseq)`

Data Embedded

`[x*y;x]` `[1, 2, 6, 24]` $\mapsto_1$?

Embeddings in Greater Detail

- `[x*y;x]  [1, 2, 6, 24]` $\mapsto_1$?

Embedding Computation	Data Embedded
`action_onehot = NNPolicy(vprogram, vseq)`	`[x*y;x]  [1, 2, 6, 24]` $\mapsto_1$?
`v1 = NNProgramMul(NNVarEmbed(x), NNVarEmbed(y))`	`x*y`

Embeddings in Greater Detail

- `[x*y;x]   [1, 2, 6, 24]` $\mapsto_1$?

Embedding Computation	Data Embedded
`action_onehot = NNPolicy(vprogram, vseq)`	`[x*y;x]   [1, 2, 6, 24]` $\mapsto_1$?
`v1 = NNProgramMul(NNVarEmbed(x), NNVarEmbed(y))`	`x*y`
`vprogram = NNProgramCons(v1, VarEmbed(x))`	`[(x*y); x]`

Embeddings in Greater Detail

- `[x*y;x]  [1, 2, 6, 24]` $\mapsto_1$?

Embedding Computation	Data Embedded
`action_onehot = NNPolicy(vprogram, vseq)`	`[x*y;x]  [1, 2, 6, 24]` $\mapsto_1$?
`v1 = NNProgramMul(NNVarEmbed(x), NNVarEmbed(y))`	`x*y`
`vprogram = NNProgramCons(v1, VarEmbed(x))`	`[(x*y); x]`
`v3 = NNSequenceCons(NNNumber(6), NNNumber(24))`	`[6; 24]`

Embeddings in Greater Detail

- `[x*y;x]   [1, 2, 6, 24]` $\mapsto_1 ?$

Embedding Computation	Data Embedded
`action_onehot = NNPolicy(vprogram, vseq)`	`[x*y;x]   [1, 2, 6, 24]` $\mapsto_1 ?$
`v1 = NNProgramMul(NNVarEmbed(x), NNVarEmbed(y))`	`x*y`
`vprogram = NNProgramCons(v1, VarEmbed(x))`	`[(x*y); x]`
`v3 = NNSequenceCons(NNNumber(6), NNNumber(24))`	`[6; 24]`
`v4 = NNSequenceCons(NNNumber(2), v3)`	`[2; 6; 24]`

Embeddings in Greater Detail

- `[x*y;x]  [1, 2, 6, 24]` $\mapsto_1 ?$

Embedding Computation	Data Embedded
`action_onehot = NNPolicy(vprogram, vseq)`	`[x*y;x]  [1, 2, 6, 24]` $\mapsto_1 ?$
`v1 = NNProgramMul(NNVarEmbed(x), NNVarEmbed(y))`	`x*y`
`vprogram = NNProgramCons(v1, VarEmbed(x))`	`[(x*y); x]`
`v3 = NNSequenceCons(NNNumber(6), NNNumber(24))`	`[6; 24]`
`v4 = NNSequenceCons(NNNumber(2), v3)`	`[2; 6; 24]`
`vseq = NNSequenceCons(NNNumber(1), v4)`	`[1; 2; 6; 24]`

Embeddings in Greater Detail

- `[x*y;x]` `[1, 2, 6, 24]` $\mapsto_1 ?$

Embedding Computation	Data Embedded
`action_onehot = NNPolicy(vprogram, vseq)`	`[x*y;x]` `[1, 2, 6, 24]` $\mapsto_1 ?$
`v1 = NNProgramMul(NNVarEmbed(x), NNVarEmbed(y))`	`x*y`
`vprogram = NNProgramCons(v1, VarEmbed(x))`	`[(x*y); x]`
`v3 = NNSequenceCons(NNNumber(6), NNNumber(24))`	`[6; 24]`
`v4 = NNSequenceCons(NNNumber(2), v3)`	`[2; 6; 24]`
`vseq = NNSequenceCons(NNNumber(1), v4)`	`[1; 2; 6; 24]`

Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)

Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.

Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.
- Generation 0: Random exploration of program space. Generates 993 solutions.

Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.
- Generation 0: Random exploration of program space. Generates 993 solutions.
- Use Generation 0 to learn policy. Run generation 1.

Evaluation

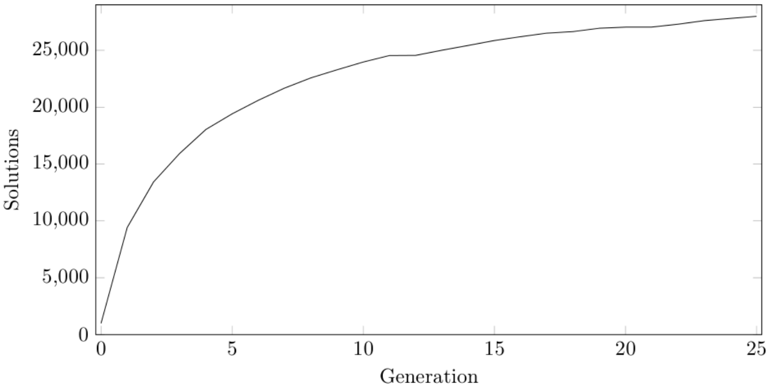
- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.
- Generation 0: Random exploration of program space. Generates 993 solutions.
- Use Generation 0 to learn policy. Run generation 1.
- Generation 1 discovers additional 8483 solutions.

Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.
- Generation 0: Random exploration of program space. Generates 993 solutions.
- Use Generation 0 to learn policy. Run generation 1.
- Generation 1 discovers additional 8483 solutions.
- Use generation k to learn policy. Run generation $k + 1$.

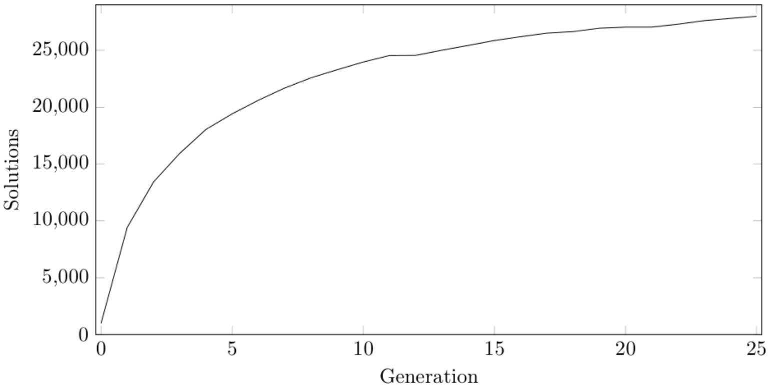
Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.
- Generation 0: Random exploration of program space. Generates 993 solutions.
- Use Generation 0 to learn policy. Run generation 1.
- Generation 1 discovers additional 8483 solutions.
- Use generation k to learn policy. Run generation $k+1$.



Evaluation

- How many OEIS sequences from the set of all sequences S have valid programs generated for them? (timeout=10min)
- Pick an OEIS sequence $s \in S$. Try to generate program p such that $p(i) = s_i$ for 10 minutes. Fail if sequence was not generated.
- Generation 0: Random exploration of program space. Generates 993 solutions.
- Use Generation 0 to learn policy. Run generation 1.
- Generation 1 discovers additional 8483 solutions.
- Use generation k to learn policy. Run generation $k+1$.



Occam's Razor

- When building dataset, if two programs p_1, p_2 are found for sequence s, keep the smaller one.

Occam's Razor

- When building dataset, if two programs p_1, p_2 are found for sequence s, keep the smaller one.
- Baseline dataset: always pick smallest program that generates a sequence s.

Occam's Razor

- When building dataset, if two programs p_1, p_2 are found for sequence s, keep the smaller one.
- Baseline dataset: always pick smallest program that generates a sequence s.
- Ablated dataset: pick random program that generates a sequence s.

Occam's Razor

- When building dataset, if two programs p_1, p_2 are found for sequence s, keep the smaller one.
- Baseline dataset: always pick smallest program that generates a sequence s.
- Ablated dataset: pick random program that generates a sequence s.
- Model trained on ablated dataset generates 10% fewer programs!

Occam's Razor

- When building dataset, if two programs p_1, p_2 are found for sequence s, keep the smaller one.
- Baseline dataset: always pick smallest program that generates a sequence s.
- Ablated dataset: pick random program that generates a sequence s.
- Model trained on ablated dataset generates 10% fewer programs!

  Thus, we can say that selecting the smallest solutions instead of the random ones for training helps finding new solutions in later generations.

10 Conclusion

Our system has created from scratch programs that gener
in the OEIS for 27987 sequences. Based on *Occam's razo*
shorter programs is more likely to generate better explanat
have also shown that the solutions discovered are correct fo
have observed that preferring shorter programs during the
of the system.

Fancier Dataset Construction

- No need to pick a sequence s and then try to randomly generate programs for it.

Fancier Dataset Construction

- No need to pick a sequence s and then try to randomly generate programs for it.
- Can start by randomly generating programs with tree search, and then matching these programs to known sequences.

Fancier Dataset Construction

- No need to pick a sequence s and then try to randomly generate programs for it.
- Can start by randomly generating programs with tree search, and then matching these programs to known sequences.
- Also provides automatic caching: If we try to produce a node that already exists, skip the node.

Fancier Dataset Construction

- No need to pick a sequence *s* and then try to randomly generate programs for it.
- Can start by randomly generating programs with tree search, and then matching these programs to known sequences.
- Also provides automatic caching: If we try to produce a node that already exists, skip the node.

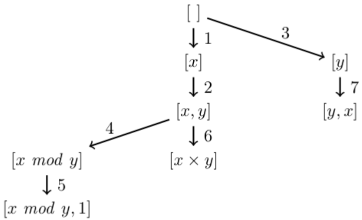


Figure 3: 7 iterations of the search loop gradually extending the search tree. The iteration number leading to the creation of a given node/stack is indicated on the arrow/action leading to it. The set of the synthesized programs after the 7th iteration is $\{1, x, y, x \times y, x \ mod \ y\}$.

Fancier Dataset Construction

- No need to pick a sequence s and then try to randomly generate programs for it.
- Can start by randomly generating programs with tree search, and then matching these programs to known sequences.
- Also provides automatic caching: If we try to produce a node that already exists, skip the node.

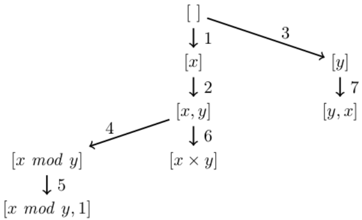


Figure 3: 7 iterations of the search loop gradually extending the search tree. The iteration number leading to the creation of a given node/stack is indicated on the arrow/action leading to it. The set of the synthesized programs after the 7th iteration is $\{1, x, y, x \times y, x \ mod \ y\}$.

compr

```
compr(f, a) :=
| failure if a < 0
| min{ m | m >= 0 and f(m, 0) <= 0 } if a = 0
| min{ m | m > compr(f, a-1) and f(m, 0) <= 0} otherwise
```