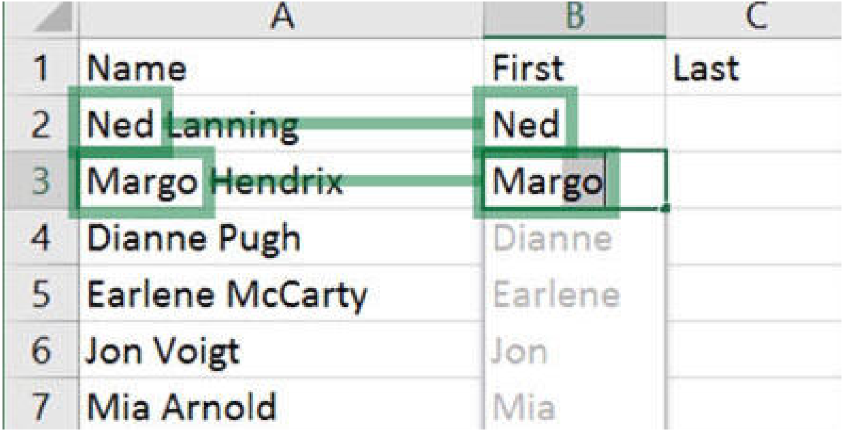
EcoSearch: A Constant-Delay Best-First Search Algorithm for Program Synthesis

T. Matricon, N. Fijalkow and G. Lagarde

AAAI 2025

Program Synthesis in the wild



Copyright Microsoft for syntactic manipulation,
based on papers by *S. Gulwani*

In practice

Specifications:

Logic:

$$\forall a, b$$
$$f(a,b) \geq a$$
$$f(a,b) \geq b$$
$$f(a,b) \in \{a, b\}$$

Examples:

$$f(1,5) = 5$$
$$f(2,1) = 2$$
$$f(-3,-9) = -3$$

Natural language:

'Write a function that takes the maximum of its two arguments.'

Produced Algorithms:

```
def max(a: int, b: int) ->int:
    if a <=b:
        return b
    else:
        return a
```

Program Synthesis: Problem

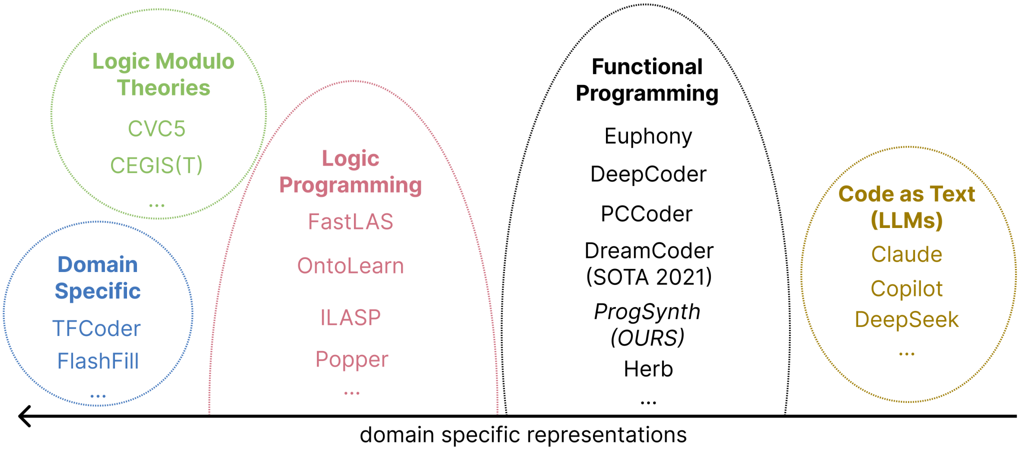
Input:

- the search space G:
  a deterministic tree grammar
- a specification $\mathcal{C}$:
  it checks if a program $p \in \mathcal{L}(G)$ matches the specification

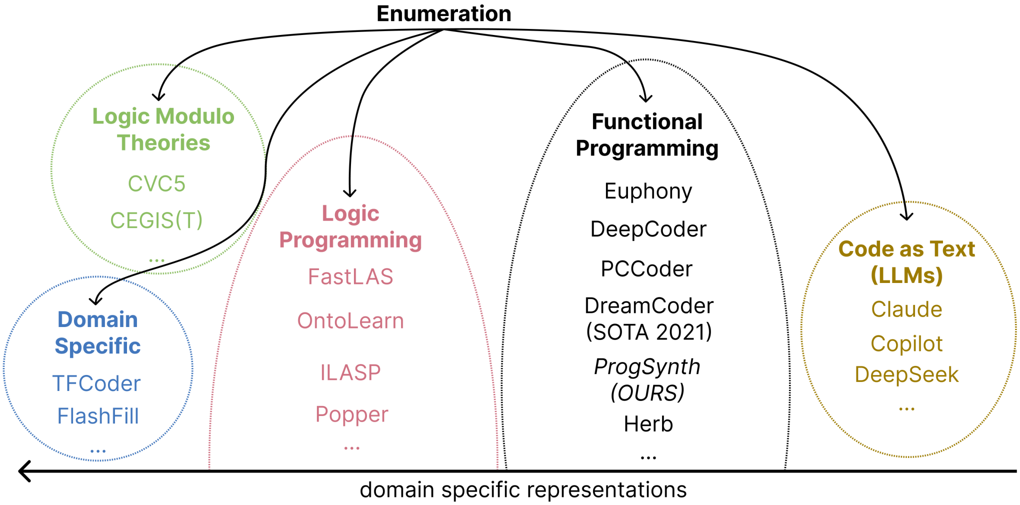
Output:

- a program in the search space that matches the specification:
  a $p \in \mathcal{L}(G)$ such that $\mathcal{C}(p) = \checkmark$

Frameworks


Logic Modulo Theories
CVC5
CEGIS(T)
...
Domain Specific
TFCoder
FlashFill
...
Logic Programming
FastLAS
OntoLearn
ILASP
Popper
...
Functional Programming
Euphony
DeepCoder
PCCoder
DreamCoder (SOTA 2021)
ProgSynth (OURS)
Herb
...
Code as Text (LLMs)
Claude
Copilot
DeepSeek
...
domain specific representations

Frameworks


Enumeration
Logic Modulo Theories
CVC5
CEGIS(T)
...
Domain Specific
TFCoder
FlashFill
...
Logic Programming
FastLAS
OntoLearn
ILASP
Popper
...
Functional Programming
Euphony
DeepCoder
PCCoder
DreamCoder (SOTA 2021)
ProgSynth (OURS)
Herb
...
Code as Text (LLMs)
Claude
Copilot
DeepSeek
...
domain specific representations

Program Synthesis in the age of LLMs?

A reviewer said "isn't this research now useless?".
No, we do not solve the same problem.

LLM: takes natural language as input
Program Synthesis: takes a checker as input
Enumeration can also be used on top of LLMs

Enumeration Problem

Input:

- the search space G:
  a deterministic tree grammar with a cost for each tree
- a specification $\mathcal{C}$:
  it checks if a program $p \in \mathcal{L}(G)$ matches the specification

Goal:

Enumerate all programs in order of non-decreasing costs

Delay:

Time complexity in terms of n: number of programs enumerated

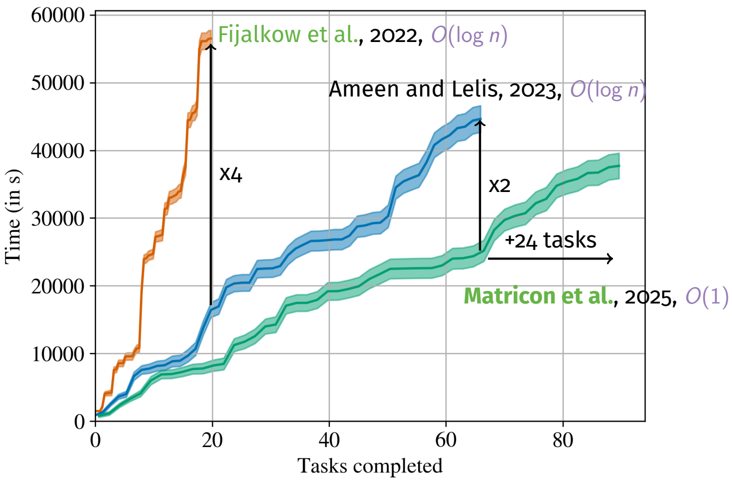
Between enumeration of the n^{th} program and the next

Overview

Major papers:

- 2017, machine learning + enumeration, *Balog et al.*, ICLR
- 2018, $O(\log n)$, *Lee et al.*, PLDI

$\longrightarrow$



Skeleton of an enumeration algorithm

At step n:

$S = O(n)$
$k = O(1)$

priority queue: S pairs of (program, cost)

$(p_1, c_1) \quad (p_2, c_2) \quad \cdots \quad (p_S, c_S)$

pop

p_n

push k successors

check is solution

$\mathcal{C}(p_n) = \checkmark$

Our enumeration algorithm

At step n:

$M = O(1)$
$k = O(1)$

bucket queue: M buckets of (programs, cost)

$(\mathcal{P}_1, c_1) \quad (\mathcal{P}_2, c_2) \quad \cdots \quad (\mathcal{P}_M, c_M)$

pop

$\mathcal{P}$

for each $p_n \in \mathcal{P}$

p_n

push k successors

check is solution

$\mathcal{C}(p_n) = \checkmark$

Our contribution

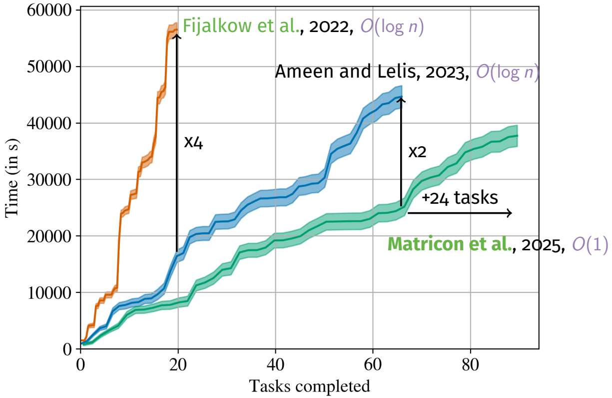
We prove bounded differences in cost:
$\exists M, \ \forall n, \ cost\_next(p_n) - cost(p_n) \leq M$

This implies: priority queues $O(\log n) \to$ bucket queues $O(1)$.

Impact

Fastest ranked enumeration for program synthesis in practice.
First algorithm with $O(1)$ delay $\to$ closes open question.

Futures



- Combine with LLMs
- Implement in other Program Synthesis solvers
- Memoryless version?
- Parallelized version? GPU?