Scaling up Domain Agnostic Techniques for Program Synthesis

Théo Matricon

supervised by Nathanaël Fijalkow

IMG_20241015_013730.jpg	Today at 11:10	504 KB	JPEG image
IMG_20241101_103002.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

IMG_20241015_013730.jpg	Today at 11:10	504 KB	JPEG image
IMG_20241101_103002.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

Manual Edit

IMG_20241015_013730.jpg	Today at 11:10	504 KB	JPEG image
IMG_20241101_103002.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

Manual Edit

cat_photo_2024_10_15.jpg	Today at 11:10	504 KB	JPEG image
cat_photo_2024_11_01.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

IMG_20241015_013730.jpg	Today at 11:10	504 KB	JPEG image
IMG_20241101_103002.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

Manual Edit

cat_photo_2024_10_15.jpg	Today at 11:10	504 KB	JPEG image
cat_photo_2024_11_01.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

Automatic Edit

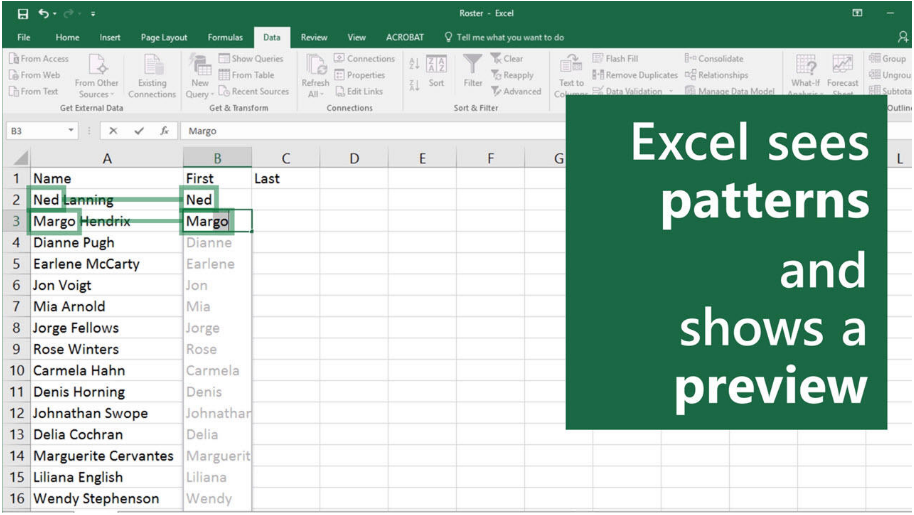
IMG_20241015_013730.jpg	Today at 11:10	504 KB	JPEG image
IMG_20241101_103002.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

Manual Edit

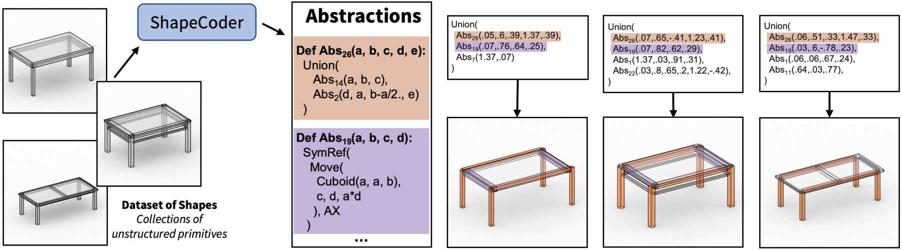
cat_photo_2024_10_15.jpg	Today at 11:10	504 KB	JPEG image
cat_photo_2024_11_01.jpg	Today at 11:10	529 KB	JPEG image
IMG_20241113_223641.jpg	Today at 11:10	343 KB	JPEG image
IMG_20241115_121123.jpg	Today at 11:10	319 KB	JPEG image
IMG_20241126_204919.jpg	Today at 11:09	270 KB	JPEG image
IMG_20241130_193000.jpg	Today at 11:09	209 KB	JPEG image

Automatic Edit

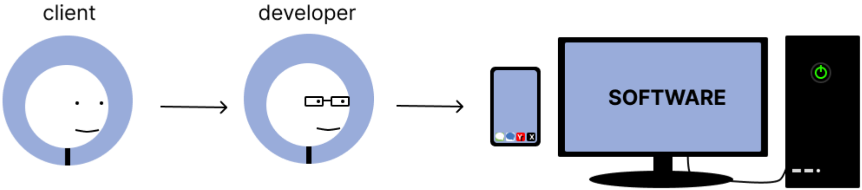
cat_photo_2024_10_15.jpg	Today at 11:10	504 KB	JPEG image
cat_photo_2024_11_01.jpg	Today at 11:10	529 KB	JPEG image
cat_photo_2024_11_13.jpg	Today at 11:10	343 KB	JPEG image
cat_photo_2024_11_15.jpg	Today at 11:10	319 KB	JPEG image
cat_photo_2024_11_26.jpg	Today at 11:09	270 KB	JPEG image
cat_photo_2024_11_30.jpg	Today at 11:09	209 KB	JPEG image

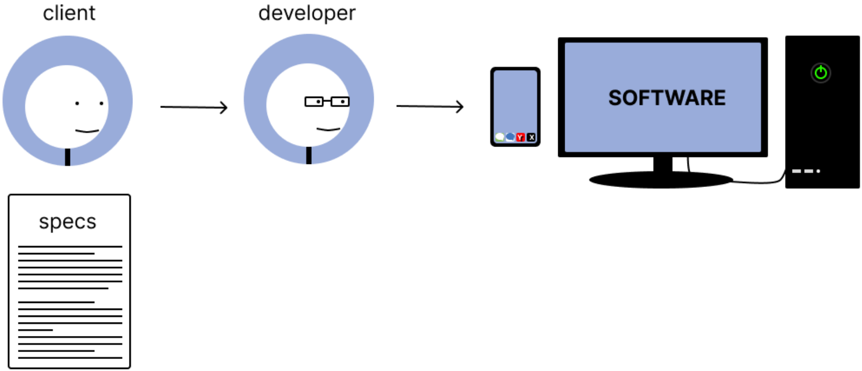


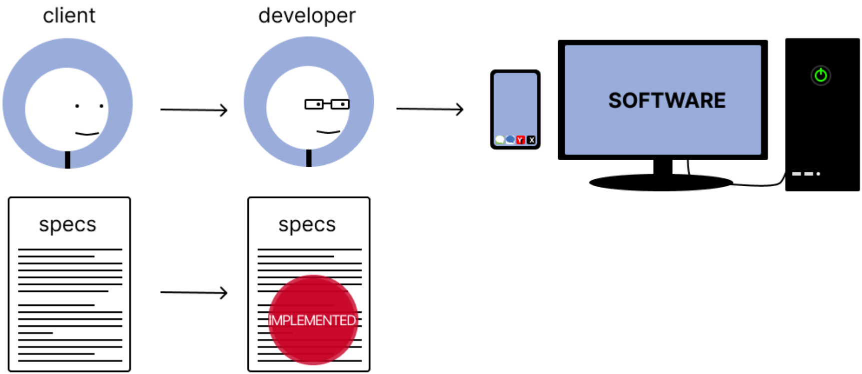
Copyright Microsoft

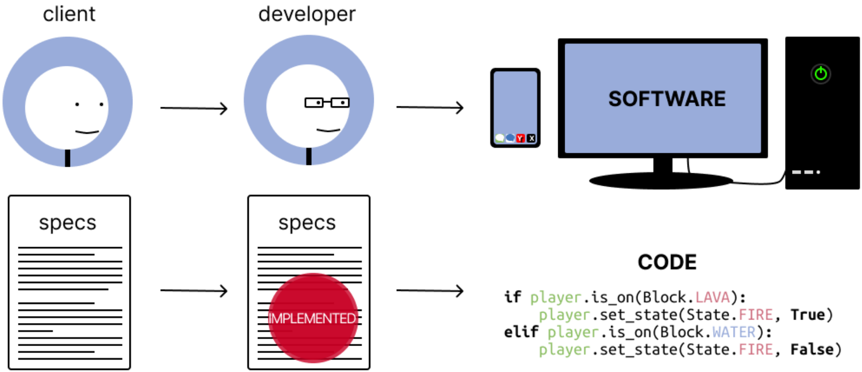


ShapeCoder [Jones et al., 2023]

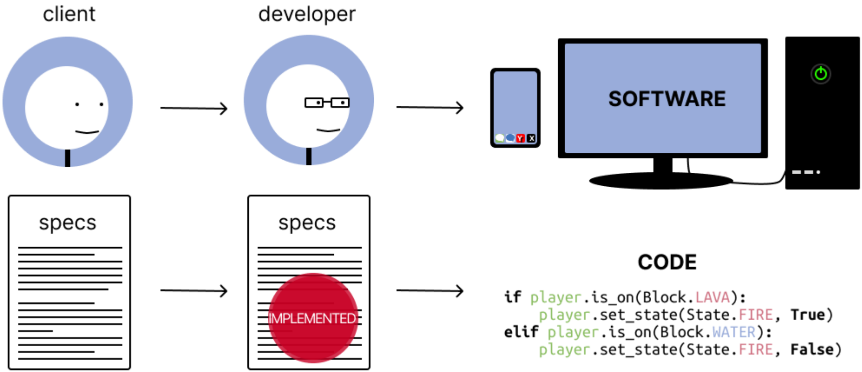

client
developer
SOFTWARE


client
developer
SOFTWARE
specs

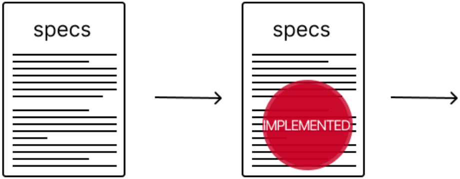

client
developer
specs
specs
IMPLEMENTED
SOFTWARE




```
if player.is_on(Block.LAVA):
    player.set_state(State.FIRE, True)
elif player.is_on(Block.WATER):
    player.set_state(State.FIRE, False)
```



Can we assist developers with automatic code generation?



CODE

```
if player.is_on(Block.LAVA):
    player.set_state(State.FIRE, True)
elif player.is_on(Block.WATER):
    player.set_state(State.FIRE, False)
```

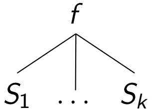
Deterministic tree grammars

a deterministic tree grammar G

derivation rules are of the form:

$$S \to f\ S_1 \ldots S_k$$

for the tree



Program Synthesis

Input:

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

Output:

- a $p \in \mathcal{L}(G)$ such that $\mathcal{C}(p) = \checkmark$

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.'

Specifications

- Logic
- Examples

- Natural Language

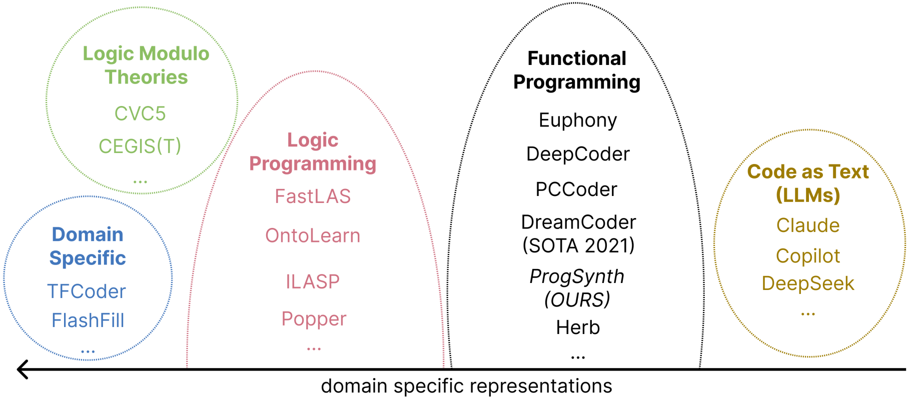
Relevant Articles of this thesis

- **Enumeration**
  - Fijalkow, Lagarde, Matricon, Ellis, Ohlmann, and Potta, *Scaling Neural Program Synthesis with Distribution-based Search*, 2022, AAAI
  - Matricon, Fijalkow, and Lagarde, *Eco Search: A No-delay Best-First Search Algorithm for Program Synthesis*, 2025, AAAI
- **Others**
  - Matricon, Fijalkow, and Marguerittte, *WikiCoder: Learning to Write Knowledge-Powered Code*, 2023, SPIN
  - Matricon and Fijalkow, *Runtime Filtering: Semantic Pruning for Program Synthesis*, 2025, Under Preparation (to be submitted)
- **Software**
  - Matricon, Fijalkow, Lagarde, and Ellis, *DeepSynth: Scaling Neural Program Synthesis with Distribution-based Search*, 2022, Journal of Open Source Software

Relevant Articles of this thesis

- **Enumeration**
  - Fijalkow, Lagarde, Matricon, Ellis, Ohlmann, and Potta, *Scaling Neural Program Synthesis with Distribution-based Search*, 2022, AAAI
  - Matricon, Fijalkow, and Lagarde, *Eco Search: A No-delay Best-First Search Algorithm for Program Synthesis*, 2025, AAAI
- **Others**
  - Matricon, Fijalkow, and Margueritte, *WikiCoder: Learning to Write Knowledge-Powered Code*, 2023, SPIN
  - Matricon and Fijalkow, *Runtime Filtering: Semantic Pruning for Program Synthesis*, 2025, Under Preparation (to be submitted)
- **Software**
  - Matricon, Fijalkow, Lagarde, and Ellis, *DeepSynth: Scaling Neural Program Synthesis with Distribution-based Search*, 2022, Journal of Open Source Software

Program Synthesis Frameworks



Enumeration

- Fijalkow, Lagarde, Matricon, Ellis, Ohlmann, and Potta, *Scaling Neural Program Synthesis with Distribution-based Search*, 2022, AAAI
- Matricon, Fijalkow, and Lagarde, *Eco Search: A No-delay Best-First Search Algorithm for Program Synthesis*, 2025, AAAI

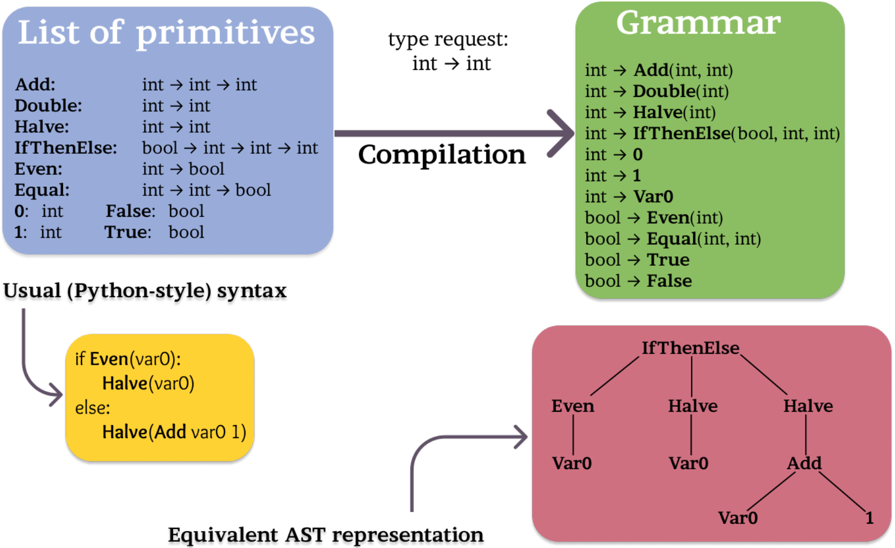
Program Synthesis

Input:

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

Output:

- a $p \in \mathcal{L}(G)$ such that $\mathcal{C}(p) = \checkmark$

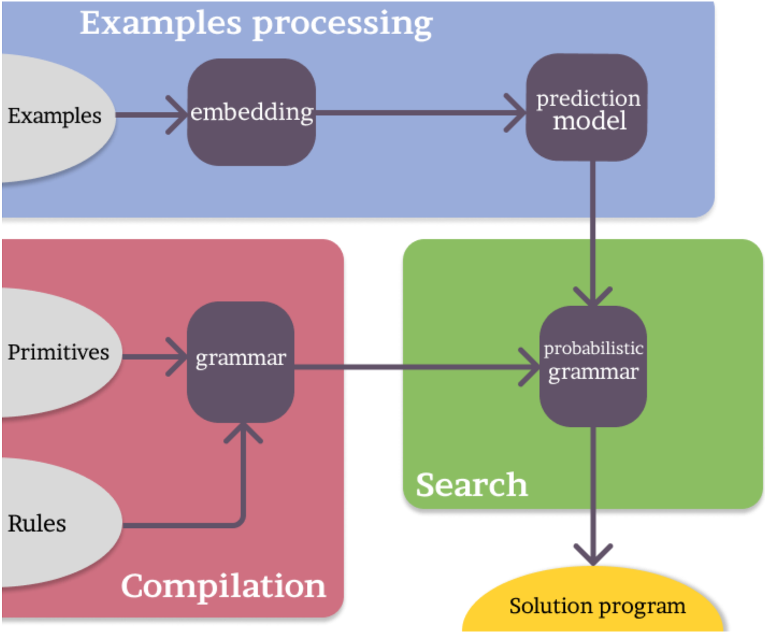


Basic If-Then-Else Grammar

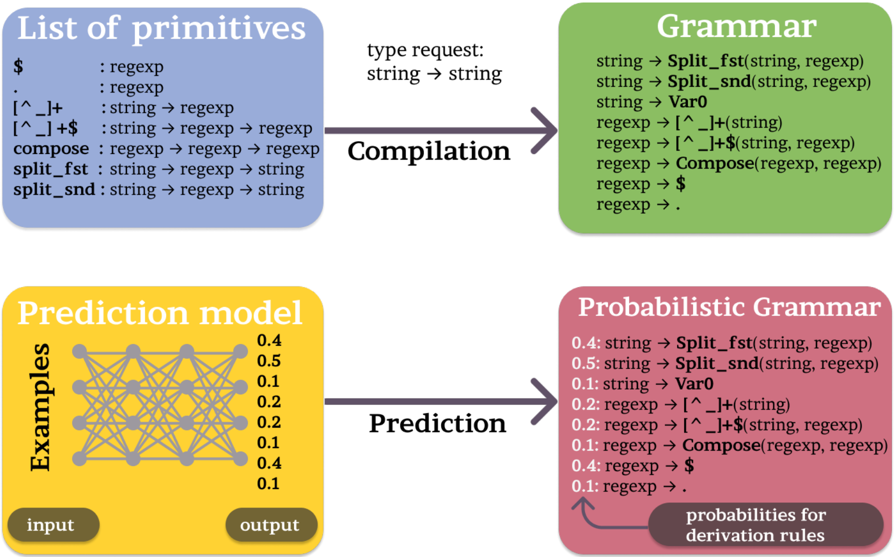
Symbolic search is not enough...

Symbolic search is not enough...

Enters machine learning [Balog et al., 2017]!



Our machine learning guided pipeline



Prediction Example

Enumeration Problem

Input:

a probabilistic(weighted) deterministic tree grammar G

Goal:

enumerate all programs of G

BFS

DFS

Threshold [Menon et al., 2013]

Sort and Add [Balog et al., 2017]

...

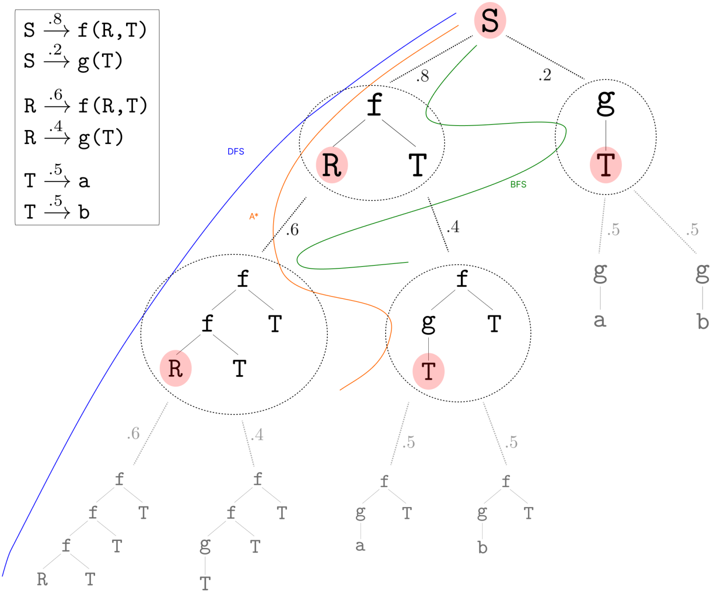
Best-first Search Problem

Input:
a probabilistic(weighted) deterministic tree grammar G

Goal:
enumerate all programs of G in order of non-increasing probabilities

Delay:
time complexity between enumeration of the n^{th} program and the next

$$S \xrightarrow{.8} f(R,T)$$
$$S \xrightarrow{.2} g(T)$$
$$R \xrightarrow{.6} f(R,T)$$
$$R \xrightarrow{.4} g(T)$$
$$T \xrightarrow{.5} a$$
$$T \xrightarrow{.5} b$$



Comparison of Best-first Search Algorithms

Time Comparison for a *simple grammar* with 3 non terminals

Best-first Search Algorithm	Time	Delay
A^* for program synthesis [Lee et al., 2018]	3h	$O(\log n)$
HeapSearch [Fijalkow et al., 2022]	1h	$O(\log n)$
BeeSearch [Ameen and Lelis, 2023]	15min	$O(\log n)$
EcoSearch w/o buckets [Matricon et al., 2025]	11min	$O(\log n)$
EcoSearch [Matricon et al., 2025]	7min30	$O(1)$

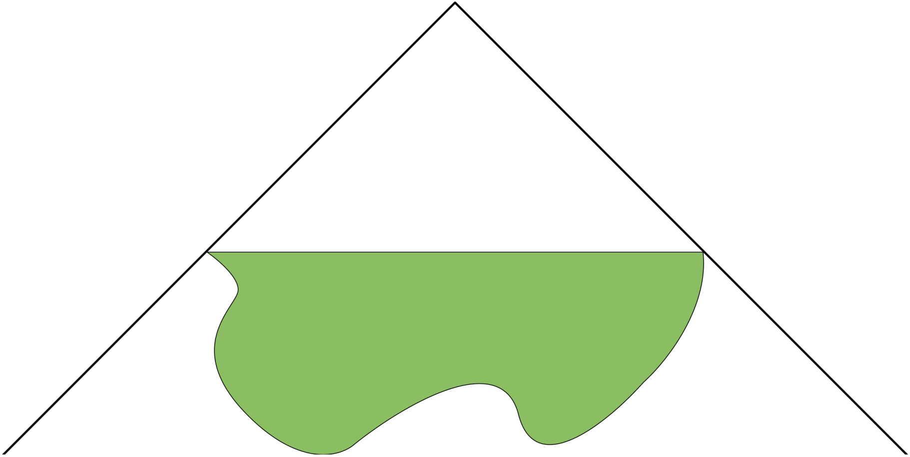


Illustration of frontier for A* [Lee et al., 2018]

- Top-down
- $O(\log n)$ delay

Key Idea: takes advantage of grammar structure

Key Idea: takes advantage of grammar structure

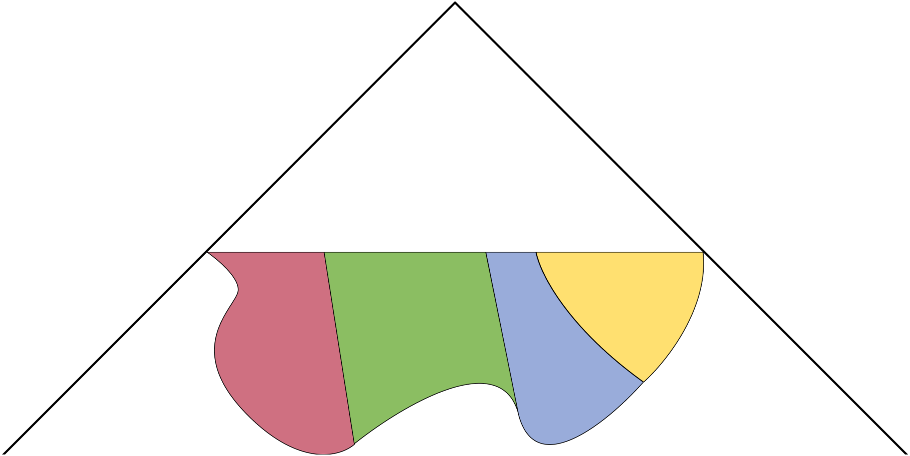
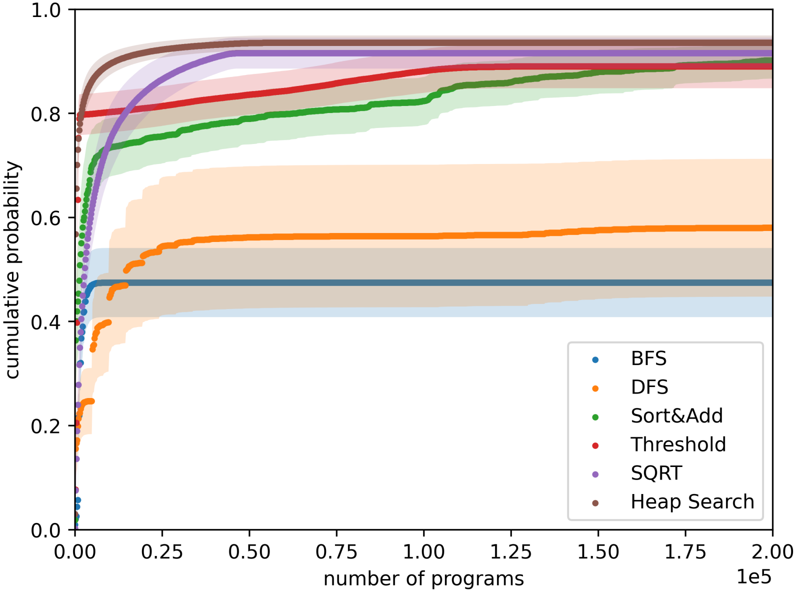


Illustration of frontier for HeapSearch [Fijalkow et al., 2022]

- Bottom-up: fast evaluation + observational equivalence
- $O(\log n)$ delay



Cumulative probability w.r.t. number of programs enumerated

DeepCoder

integer list manipulation benchmark

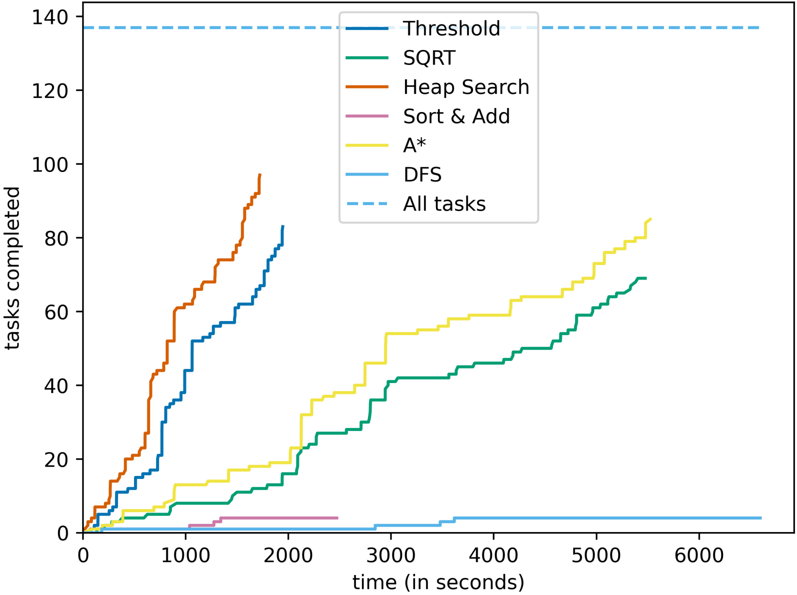
500 tasks with programs of depth < 5

introduced in DeepCoder [Balog et al., 2017]

simple grammar with 2 non terminals

```
def f(x: list[int]) -> list[int]:
    y = sort(x)
    return filter(is_even, y)

example = {
    input=[236, 147, -158, 99, 170],
    output=[-158, 17     0, 236]
}
```



Tasks solved using different enumeration algorithms on DeepCoder

Key Idea: structured frontier expansion

Key Idea: structured frontier expansion

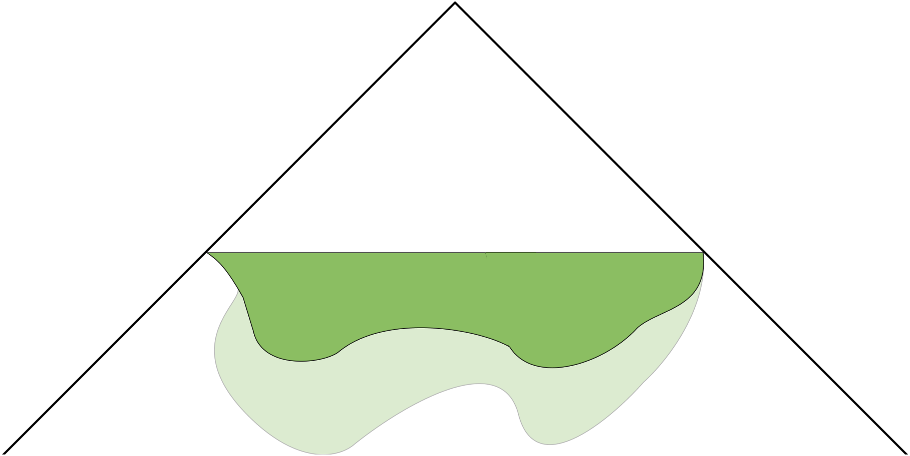


Illustration of frontier for BEESEARCH [Ameen and Lelis, 2023]

- Introduce cost tuple representation
- Better frontier expansion

Key Idea: unification of HeapSearch and BeeSearch

Key Idea: unification of HEAPSEARCH and BEESEARCH

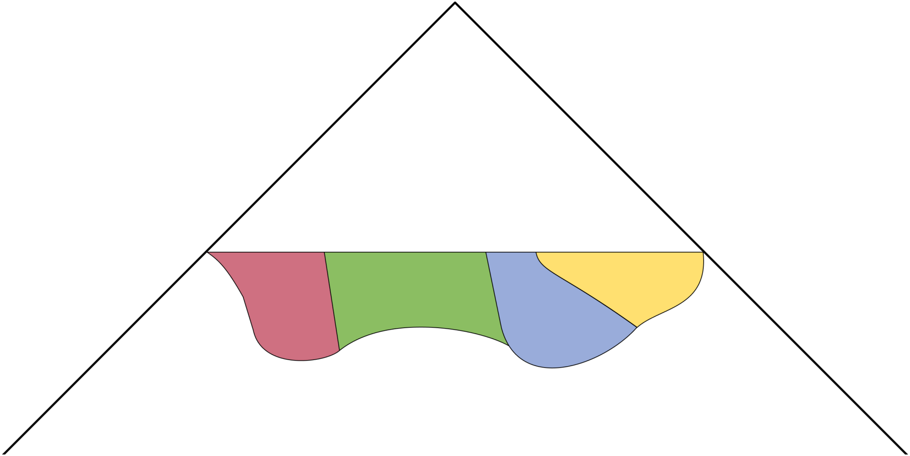
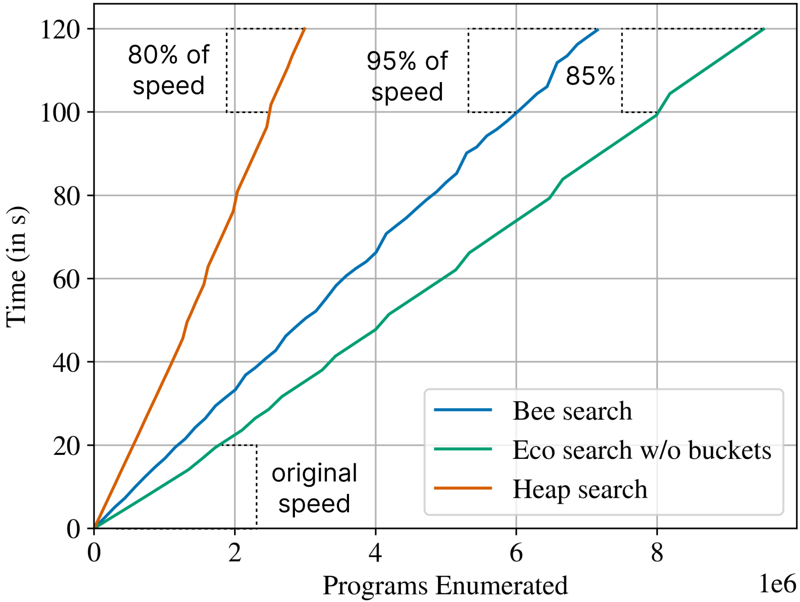


Illustration of frontier for ECOSEARCH without buckets [Matricon et al., 2025]

- $O(\log n)$ delay
- Frugal frontier expansion

Key Issue: $O(\log n)$ delay implies a slow-down over time

Key Issue: $O(\log n)$ delay implies a slow-down over time



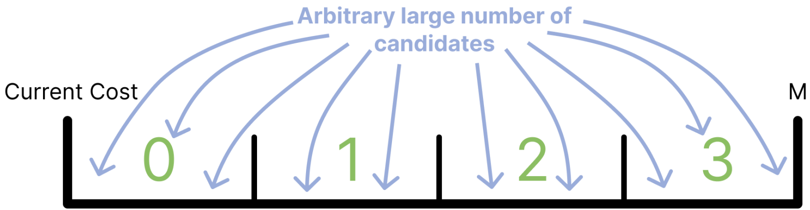
Key Theoretical Insight

There exists a constant $M \geq 0$ such that,
for any program p and its successor p'
we have $cost(p') - cost(p) \leq M$.

Key Theoretical Insight

There exists a constant $M \geq 0$ such that,
for any program p and its successor p'
we have $cost(p') - cost(p) \leq M$.

M does not depend on the number of programs enumerated.


Arbitrary large number of candidates
Current Cost
M
0
1
2
3

Key Idea: take advantage of our theoretical insight

Key Idea: take advantage of our theoretical insight

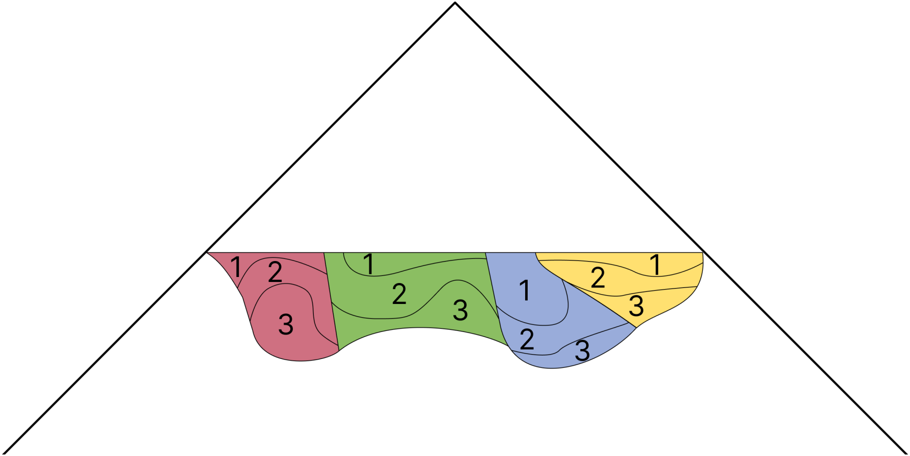


Illustration of frontier for EcoSearch [Matricon et al., 2025]

- $O(1)$ delay
- Integer costs

FlashFill

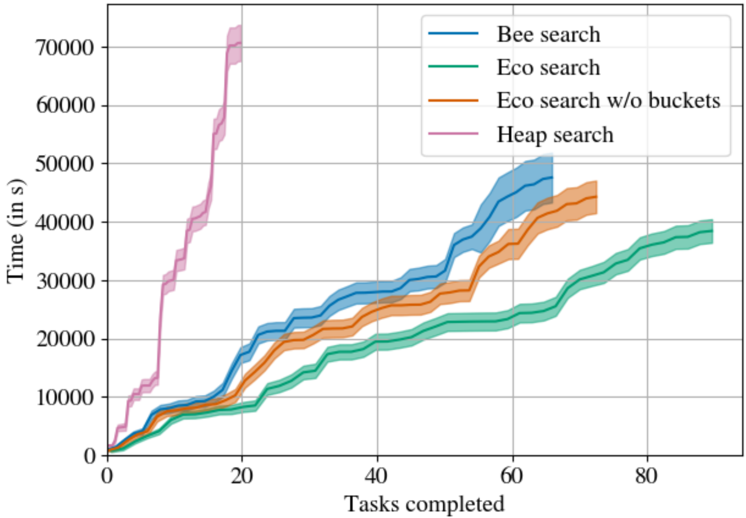
string manipulation benchmark

100 tasks

introduced in FlashFill [Gulwani, 2011]

simple grammar with 3 non terminals

```
    examples = [{
            input="736 miles",
            output="736"
        },
        {
            input="1255 miles",
            output="1255"
        },
        {
            input="790 miles",
            output="790"
        }
     ]
```



Tasks solved using different enumeration algorithms on FlashFill

Conclusion

- From $O(\log n)$ top-down to $O(\log n)$ bottom-up
- From $O(\log n)$ bottom-up to $O(1)$ bottom-up
- Faster program synthesis

But also (*not mentioned*)

- Introduced distribution-based search framework
- A "loss-optimal" sampling algorithm
- Grammar splitting to parallelise the search
- Better scaling with grammar complexity

Wikicoder

- Matricon, Fijalkow, and Margueritte, *WikiCoder: Learning to Write Knowledge-Powered Code*, 2023, SPIN

$$\begin{array}{lll} \text{Paris} & \longrightarrow \text{France} & \text{code: } 33 \\ \text{Berlin} & \longrightarrow \text{Germany} & \text{code: } 49 \\ \text{Warsaw} & \longrightarrow \text{Poland} & \text{code: } 48 \end{array}$$

Paris $\longrightarrow$ France code: 33
Berlin $\longrightarrow$ Germany code: 49
Warsaw $\longrightarrow$ Poland code: 48

Syntactic processing cannot solve these tasks.

Paris ⟶ France code: 33

Berlin ⟶ Germany code: 49

Warsaw ⟶ Poland code: 48

Syntactic processing cannot solve these tasks.

Syntactic Extraction

Semantic Processing

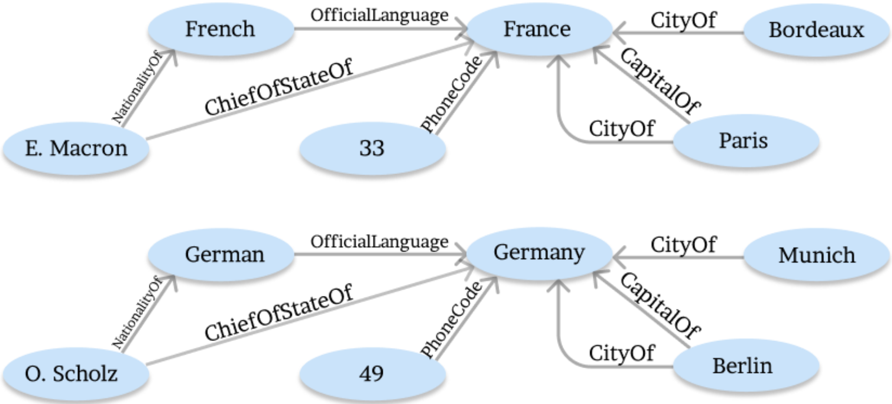
President Obama $\longrightarrow$ Obama
Prime Minister de Pfeffel Johnson $\longrightarrow$ de Pfeffel Johnson

Knowledge Post-Processing

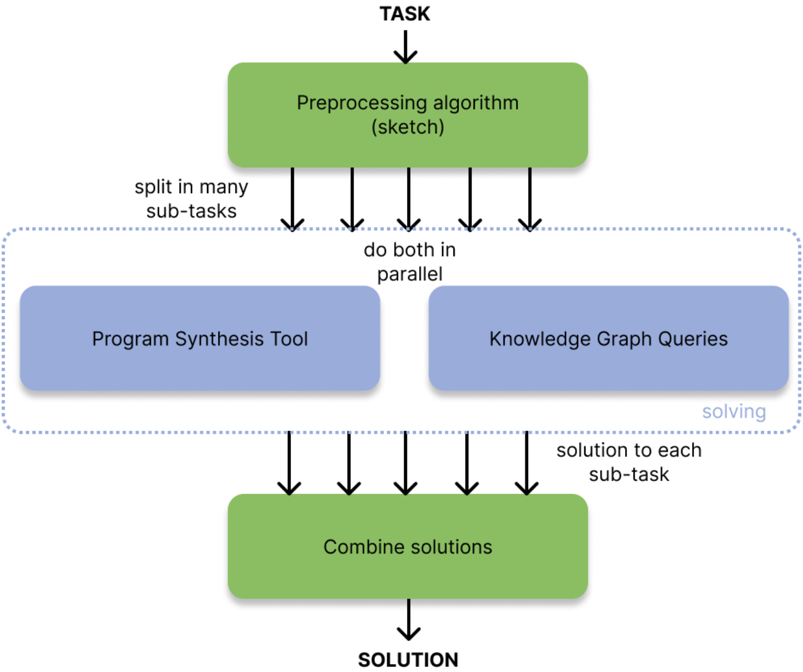
Paris $\longrightarrow$ Frnc

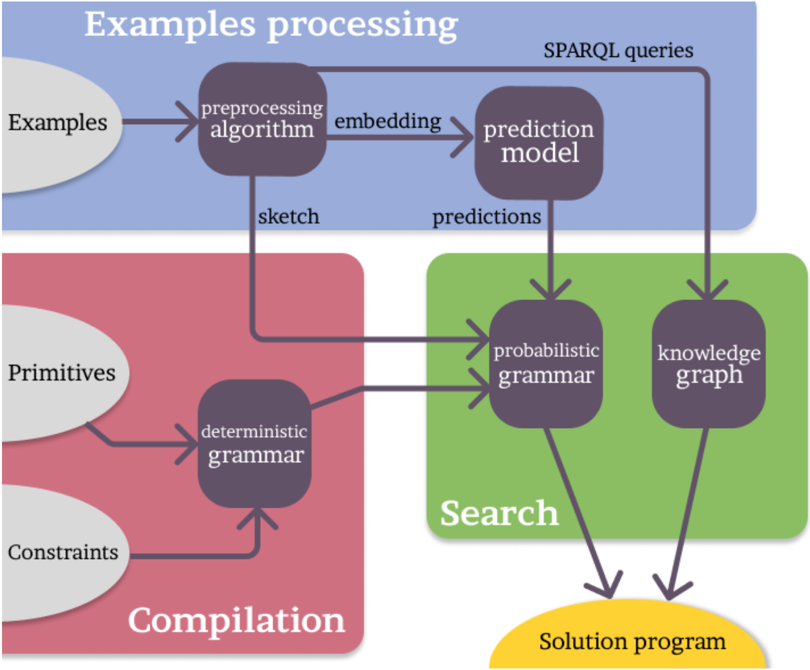
Berlin $\longrightarrow$ Grmn

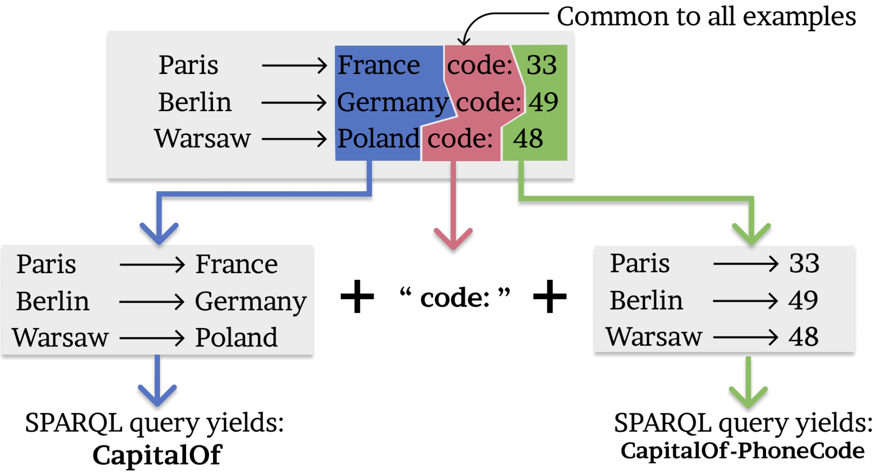
Warsaw $\longrightarrow$ Plnd

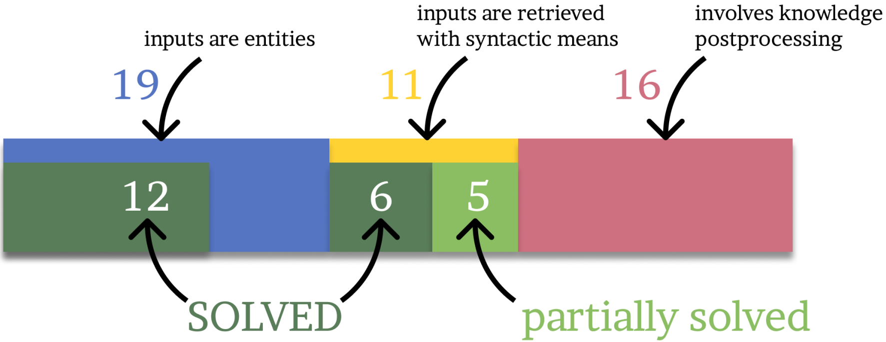


Idealized extract from YAGO/WikiData


TASK
Preprocessing algorithm (sketch)
split in many sub-tasks
do both in parallel
Program Synthesis Tool
Knowledge Graph Queries
solving
solution to each sub-task
Combine solutions
SOLUTION


Examples processing
SPARQL queries
Examples
preprocessing algorithm
embedding
prediction model
sketch
predictions
Primitives
deterministic grammar
Constraints
Compilation
probabilistic grammar
knowledge graph
Search
Solution program


Common to all examples
Paris ⟶ France code: 33
Berlin ⟶ Germany code: 49
Warsaw ⟶ Poland code: 48
Paris ⟶ France
Berlin ⟶ Germany
Warsaw ⟶ Poland
+ " code: " +
Paris ⟶ 33
Berlin ⟶ 49
Warsaw ⟶ 48
SPARQL query yields:
CapitalOf
SPARQL query yields:
CapitalOf-PhoneCode


inputs are entities
inputs are retrieved with syntactic means
involves knowledge postprocessing
19
11
16
12
6
5
SOLVED
partially solved

Conclusion

- Different levels of knowledge in programs
- Tackled entities and syntactic extraction
- Strong hypothesis on the domain

Conclusion

- HeapSearch: first $O(\log n)$ bottom-up
- EcoSearch: first $O(1)$ and bottom-up
- Speed-up: A^*: 3h EcoSearch: 7min30

- Knowledge-powered programs
- Different levels of complexity in knowledge-powered program synthesis
- Tackled entities and syntactic extraction

Conclusion

But also (*not mentioned*)

- Introduced distribution-based search framework
- A "loss-optimal" sampling algorithm
- Grammar splitting to parallelise the search
- Improved scaling of enumeration with grammar complexity

- Generate semantic equalities automatically
- Prune semantic redundant programs in $O(1)$ at runtime

ProgSynth

Generic Synthesis Library

- 10k lines of code
- 2.5k lines of test

Programming By Examples Specific

- 8k lines of code

Perspectives

- How can we remove memory constraints of enumeration algorithms?
- How can we have GPU-friendly implementations?
- How can we parallelise program synthesis?

- How can we combine enumerative search paradigm with LLMs?
- And more generally, can we combine multiple paradigms?

Runtime Filtering

- Matricon and Fijalkow, *Runtime Filtering: Semantic Pruning for Program Synthesis*, 2025, Under Preparation

1. `0+1` is useless: it does not use the input variable `Var0`;
2. `Double(Halve(P))` is redundant: it is equivalent to `P`;
3. `Add(Add(P,Q),R)` and `Add(P, Add(Q,R))` are equivalent
4. `Add(P,Q)` and `Add(Q,P)` are equivalent.

1. `Var0` must be used at least once rules out the program `0+1`;
2. Forbidding `Double(Halve)` rules out `Double(Halve(P))`;
3. Forbidding `Add(_,Add)` rules out programs associating addition to the right;
4. Choosing between `Add(P,Q)` and `Add(Q,P)` would imply ordering all programs, which context-free grammars cannot do.

$$B \Longrightarrow \texttt{And}(B, B) \mid \texttt{Or}(B, B) \mid \texttt{Not}(B) \mid \texttt{Var0} \mid \texttt{Var1}$$

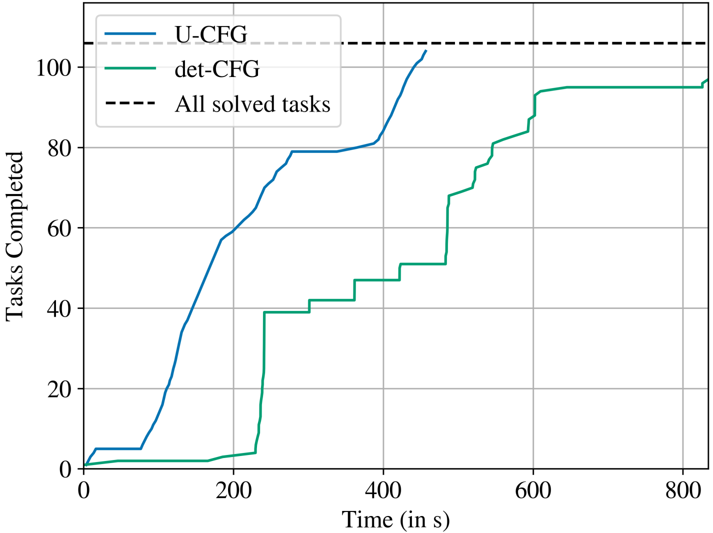
$$
\begin{aligned}
B_1 &\implies \texttt{And}(B_1, B_1) \mid \texttt{Or}(B_2, B_2) \mid \texttt{Not}(B_3) \mid \texttt{Var0} \mid \texttt{Var1} \\
B_2 &\implies \texttt{Or}(B_2, B_2) \mid \texttt{Not}(B_3) \mid \texttt{Var0} \mid \texttt{Var1} \\
B_3 &\implies \texttt{Not}(B_3) \mid \texttt{Var0} \mid \texttt{Var1}
\end{aligned}
$$

1. We enumerate all programs where each variable appears at most once, up to some fixed depth and some fixed number of variables;
2. We check for program equivalence amongst all generated programs;
3. For each equation found where one program is larger than the other one, we add a rule to forbid the larger program;

- The compilation of rules are performed on DBTAs.
- Minimisation are performed on DBTAs.
- Enumeration is performed on det-CFG and pruned by the DBTA.

Number of programs and respective proportions (prop.) with respect to maximum depth in the List Programming DSL with type ‘int list $\rightarrow$ int list’.

depth	no rules	with rules (prop.)
3	8.77e+04	0.83
4	3.34e+16	0.51
5	9.20e+52	0.13
6	4.79e+165	0.0015
7	4.14e+510	10^{-9}



Number of tasks solved with respect to cumulative time on the set of all solved List Programming tasks

Conclusion

- Find rules once and for all
- Compile the rules
- Almost-free pruning even with a smaller grammar model for bottom-up processes

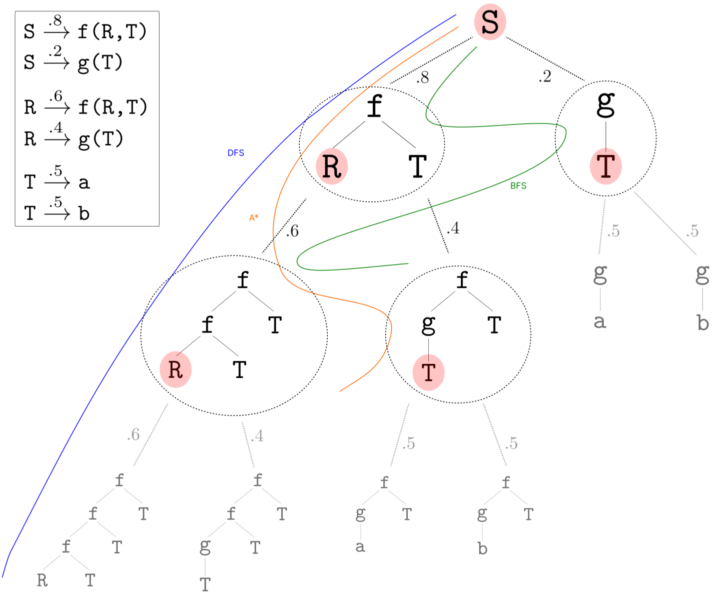
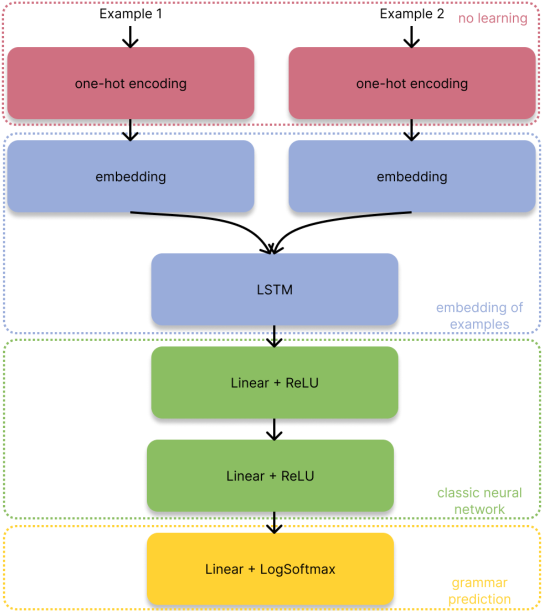
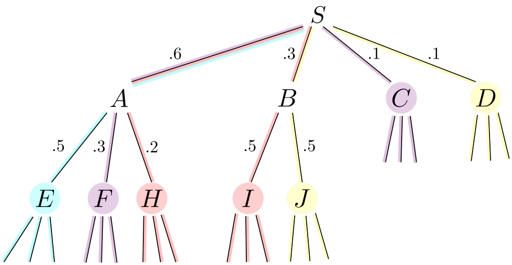
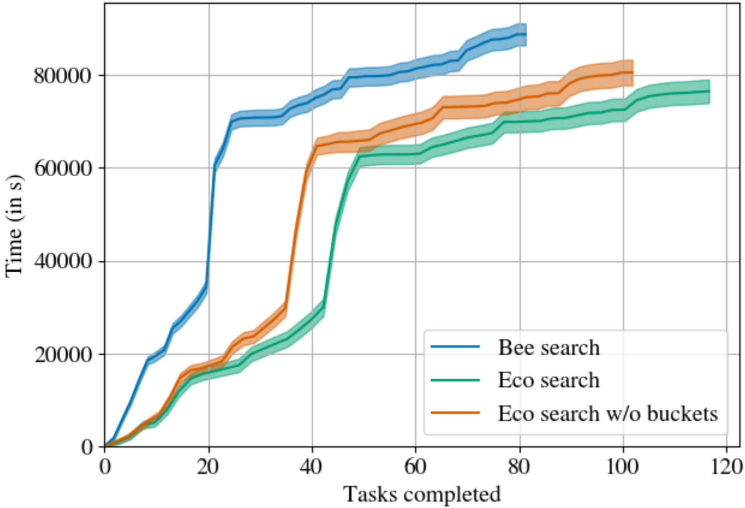


Illustration of the tree of leftmost derivations.

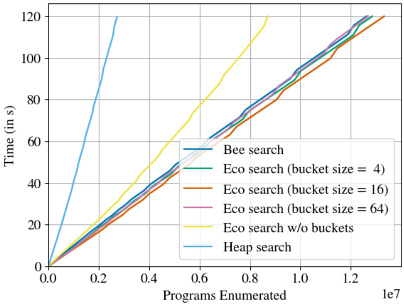

Example 1
Example 2
no learning
one-hot encoding
one-hot encoding
embedding
embedding
LSTM
embedding of examples
Linear + ReLU
Linear + ReLU
classic neural network
Linear + LogSoftmax
grammar prediction

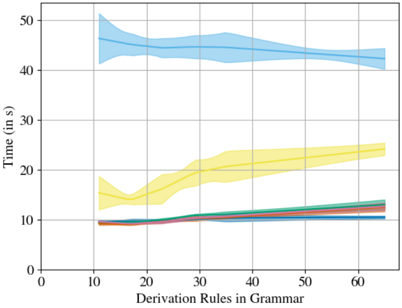


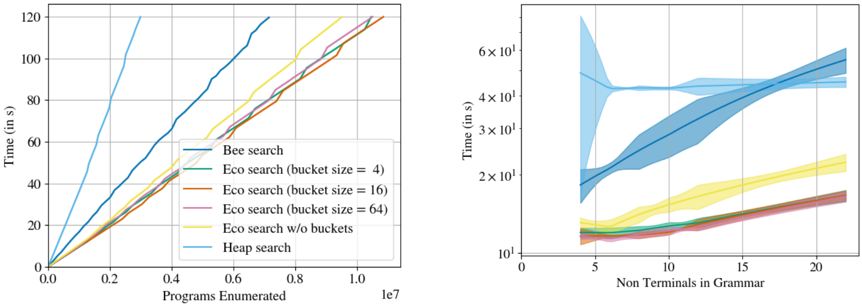
$$
\begin{array}{|l|}\hline
S \xrightarrow{.6} A \\
S \xrightarrow{.3} B \\
S \xrightarrow{.1} C \\
S \xrightarrow{.1} D \\
A \xrightarrow{.1} E \\
A \xrightarrow{.3} F \\
A \xrightarrow{.2} H \\
B \xrightarrow{.5} I \\
B \xrightarrow{.5} J \\
E \longrightarrow \cdots \\
\vdots \quad \vdots \quad \vdots \\ \hline
\end{array}
= .3 \times
\begin{array}{|l|}\hline
S \xrightarrow{1} A \\
A \xrightarrow{1} E \\
E \rightarrow \cdots \\
\vdots \quad \vdots \quad \vdots \\ \hline
\end{array}
+ .28 \times
\begin{array}{|l|}\hline
S \xrightarrow{.857} A \\
S \xrightarrow{.143} C \\
A \xrightarrow{1} F \\
F \longrightarrow \ldots \\
\vdots \quad \vdots \quad \vdots \\ \hline
\end{array}
+ .27 \times
\begin{array}{|l|}\hline
S \xrightarrow{.667} A \\
S \xrightarrow{.333} B \\
A \xrightarrow{1} H \\
B \xrightarrow{1} I \\
H \longrightarrow \ldots \\
\vdots \quad \vdots \quad \vdots \\ \hline
\end{array}
+ .25 \times
\begin{array}{|l|}\hline
S \xrightarrow{.75} B \\
S \xrightarrow{.25} D \\
B \xrightarrow{1} J \\
J \longrightarrow \ldots \\
\vdots \quad \vdots \quad \vdots \\ \hline
\end{array}
$$

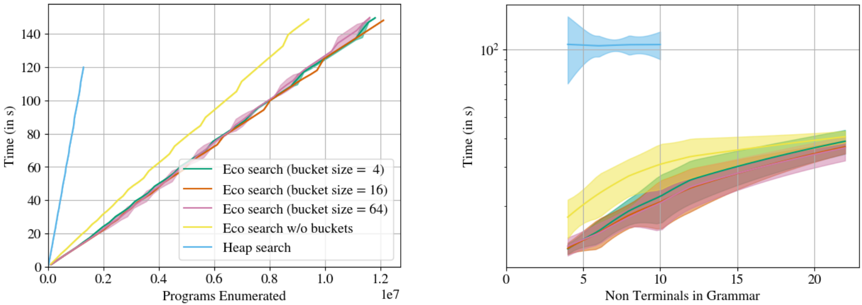


Tasks solved using different enumeration algorithms on DeepCoder


120
100
80
60
40
20
0
Time (in s)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Programs Enumerated
1e7
Bee search
Eco search (bucket size = 4)
Eco search (bucket size = 16)
Eco search (bucket size = 64)
Eco search w/o buckets
Heap search


50
40
30
20
10
0
Time (in s)
0
10
20
30
40
50
60
Derivation Rules in Grammar


Time (in s)
Programs Enumerated
1e7
Bee search
Eco search (bucket size = 4)
Eco search (bucket size = 16)
Eco search (bucket size = 64)
Eco search w/o buckets
Heap search
Time (in s)
Non Terminals in Grammar



Time (in s)
Programs Enumerated
1e7
Eco search (bucket size = 4)
Eco search (bucket size = 16)
Eco search (bucket size = 64)
Eco search w/o buckets
Heap search
Time (in s)
Non Terminals in Grammar

S. Ameen and L. H. Lelis. Program synthesis with best-first bottom-up search. *Journal of Artificial Intelligence Research*, 77:1275–1310, 2023.

M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. In *International Conference on Learning Representations, ICLR*, 2017. URL `https://openreview.net/forum?id=ByldLrqlx`.

N. Fijalkow, G. Lagarde, T. Matricon, K. Ellis, P. Ohlmann, and A. Potta. Scaling neural program synthesis with distribution-based search. In *AAAI*, 2022. URL `https://arxiv.org/abs/2110.12485`.

S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, 2011. URL `https://doi.org/10.1145/1926385.1926423`.

R. K. Jones, P. Guerrero, N. J. Mitra, and D. Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics (TOG), Siggraph 2023*, 42 (4), 2023.

W. Lee, K. Heo, R. Alur, and M. Naik. Accelerating search-based program synthesis using learned probabilistic models. *SIGPLAN Not.*, 53(4):

436449, June 2018. ISSN 0362-1340. doi: 10.1145/3296979.3192410. URL https://doi.org/10.1145/3296979.3192410.

T. Matricon and N. Fijalkow. Runtime filtering: Semantic pruning for program synthesis. In *Under Preparation*, volume 33, 2025.

T. Matricon, N. Fijalkow, G. Lagarde, and K. Ellis. Deepsynth: Scaling neural program synthesis with distribution-based search. *Journal of Open Source Software*, 7(78):4151, 2022. doi: 10.21105/joss.04151.

T. Matricon, N. Fijalkow, and G. Margueritte. Wikicoder: Learning to write knowledge-powered code. In G. Caltais and C. Schilling, editors, *SPIN*, pages 123–140. Springer Nature Switzerland, 2023. ISBN 978-3-031-32157-3.

T. Matricon, N. Fijalkow, and G. Lagarde. Eco search: A no-delay best-first search algorithm for program synthesis. In *AAAI*, 2025.

A. K. Menon, O. Tamuz, S. Gulwani, B. W. Lampson, and A. Kalai. A machine learning framework for programming by example. In *International Conference on Machine Learning, ICML*, 2013. URL http://proceedings.mlr.press/v28/menon13.html.