# Neural Program Synthesis: Learning to Program from Trees

Tilman Hinnerichs Neil Yorke-Smith Sebastijan Dumancic

PONY lab @ TU Delft: Algorithmics Group

April 2, 2026

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## Typical use of LLMs

"Give me a program

- written in Julia/Python/... ,

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## Typical use of LLMs

"Give me a program

- written in Julia/Python/... ,
- that ...
  - translates data from format A to format B

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## Typical use of LLMs

"Give me a program

- written in Julia/Python/... ,
- that ...
  - translates data from format A to format B
  - passes these tests

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## Typical use of LLMs

"Give me a program

- written in Julia/Python/... ,
- that ...
  - translates data from format A to format B
  - passes these tests
  - fixes this existing program

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## Typical use of LLMs

"Give me a program

- written in Julia/Python/... ,
- that ...
  - translates data from format A to format B
  - passes these tests
  - fixes this existing program
  - runs `rm -rf /`"

## A familiar situation

- As computer scientists (and fans of related topics), we write a lot of code.
- Today, almost everyone uses/used LLMs somewhere in that process.

## Typical use of LLMs

"Give me a program

- written in Julia/Python/... ,
- that ...
  - translates data from format A to format B
  - passes these tests
  - fixes this existing program
  - runs `rm -rf /`"

i.e., a **target language** and a **specification**.

# Why LLMs often do/do not work.

## Why LLMs do work!

LLMs ...

- are **knowledgeable**, being trained on tons of data,

# Why LLMs often do/do not work.

## Why LLMs do work!

LLMs ...

- are **knowledgeable**, being trained on tons of data,
- are **fast***: They merge all this knowledge into a single model, and

# Why LLMs often do/do not work.

## Why LLMs do work!

LLMs ...

- are **knowledgeable**, being trained on tons of data,
- are **fast***: They merge all this knowledge into a single model, and
- often produce **plausible solutions,** or almost working ones.

# Why LLMs often do/do not work.

## Why LLMs do work!

LLMs ...

- are **knowledgeable**, being trained on tons of data,
- are **fast***: They merge all this knowledge into a single model, and
- often produce **plausible solutions,** or almost working ones.

## Why LLMs do **not** work!

1. **Syntactic Errors:** They do not follow a formal description of correct syntax, e.g., a grammar.

# Why LLMs often do/do not work.

## Why LLMs do work!

LLMs ...

- are **knowledgeable**, being trained on tons of data,
- are **fast***: They merge all this knowledge into a single model, and
- often produce **plausible solutions,** or almost working ones.

## Why LLMs do **not** work!

1. **Syntactic Errors:** They do not follow a formal description of correct syntax, e.g., a grammar.
2. **Assumed Semantics:** Has to assume what an operator means from its name.

# Why LLMs often do/do not work.

## Why LLMs do work!

LLMs ...

- are **knowledgeable**, being trained on tons of data,
- are **fast***: They merge all this knowledge into a single model, and
- often produce **plausible solutions,** or almost working ones.

## Why LLMs do **not** work!

1. **Syntactic Errors:** They do not follow a formal description of correct syntax, e.g., a grammar.
2. **Assumed Semantics:** Has to assume what an operator means from its name.
3. **Cannot Execute the Program:** What went wrong? What doesn't work yet?

# What is Program Synthesis?

| | A | B |
|---|---|---|
| 1 | Email | Column 2 |
| 2 | Nancy.FreeHafer@fourthcoffee.com | nancy freehafer |
| 3 | Andrew.Cencici@northwindtraders.com | andrew cencici |
| 4 | Jan.Kotas@litwareinc.com | jan kotas |
| 5 | Mariya.Sergienko@gradicdesigninstitute.com | mariya sergienko |
| 6 | Steven.Thorpe@northwindtraders.com | steven thorpe |
| 7 | Michael.Neipper@northwindtraders.com | michael neipper |

(a) Specification by examples

# What is Program Synthesis?

| | A | B |
|---|---|---|
| 1 | Email | Column 2 |
| 2 | Nancy.FreeHafer@fourthcoffee.com | nancy freehafer |
| 3 | Andrew.Cencici@northwindtraders.com | andrew cencici |
| 4 | Jan.Kotas@litwareinc.com | jan kotas |
| 5 | Mariya.Sergienko@gradicdesigninstitute.com | mariya sergienko |
| 6 | Steven.Thorpe@northwindtraders.com | steven thorpe |
| 7 | Michael.Neipper@northwindtraders.com | michael neipper |

(a) Specification by examples

$$
\begin{aligned}
S &::= \texttt{concat(S, S)} \\
S &::= \texttt{at\_ind(S, Int)} \\
S &::= \texttt{"."} \\
\mathsf{Int} &::= \texttt{length(S)} \\
&\vdots
\end{aligned}
$$

(b) Some String grammar

# What is Program Synthesis?

| | A | B |
|---|---|---|
| 1 | Email | Column 2 |
| 2 | Nancy.FreeHafer@fourthcoffee.com | nancy freehafer |
| 3 | Andrew.Cencici@northwindtraders.com | andrew cencici |
| 4 | Jan.Kotas@litwareinc.com | jan kotas |
| 5 | Mariya.Sergienko@gradicdesigninstitute.com | mariya sergienko |
| 6 | Steven.Thorpe@northwindtraders.com | steven thorpe |
| 7 | Michael.Neipper@northwindtraders.com | michael neipper |

(a) Specification by examples

S ::= `concat(S, S)`
S ::= `at_ind(S, Int)`
S ::= `"."`
Int ::= `length(S)`
⋮

(b) Some String grammar

```
concat(at_ind(_arg_1, Int), ".")
```

Figure: A program

# What is Program Synthesis?

| | A | B |
|---|---|---|
| 1 | Email | Column 2 |
| 2 | Nancy.FreeHafer@fourthcoffee.com | nancy freehafer |
| 3 | Andrew.Cencici@northwindtraders.com | andrew cencici |
| 4 | Jan.Kotas@litwareinc.com | jan kotas |
| 5 | Mariya.Sergienko@gradicdesigninstitute.com | mariya sergienko |
| 6 | Steven.Thorpe@northwindtraders.com | steven thorpe |
| 7 | Michael.Neipper@northwindtraders.com | michael neipper |

(a) Specification by examples

```
S ::= concat(S, S)
S ::= at_ind(S, Int)
S ::= "."
Int ::= length(S)
⋮
```

(b) Some String grammar

```
concat(at_ind(_arg_1, Int), ".")
```

Figure: A program

- Find solution by enumerating (possibly all) programs from the **grammar** and check them against the **specification**.

# Why Program Synthesis?

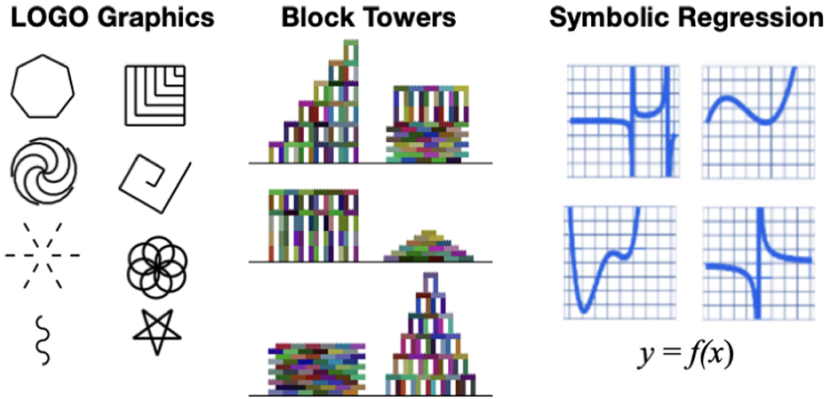



Figure: Examples of tested domains in DreamCoder [2].

[2] Ellis, K. et al. *DreamCoder: Building interpretable hierarchical knowledge representations with wake-sleep Bayesian program learning* (2020).

# Why Program Synthesis in this Seminar?

## Programs are just graphs!

- Every program has an AST representation

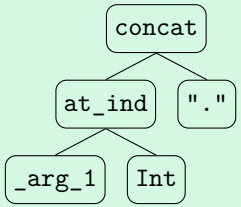
# Why Program Synthesis in this Seminar?

## Programs are just graphs!

- Every program has an AST representation

```
concat(at_ind(_arg_1, Int),
            ".")
```

$\Longrightarrow$




Program as an AST

Figure: A program is not just a string: it has tree structure.

# The gap we want to bridge

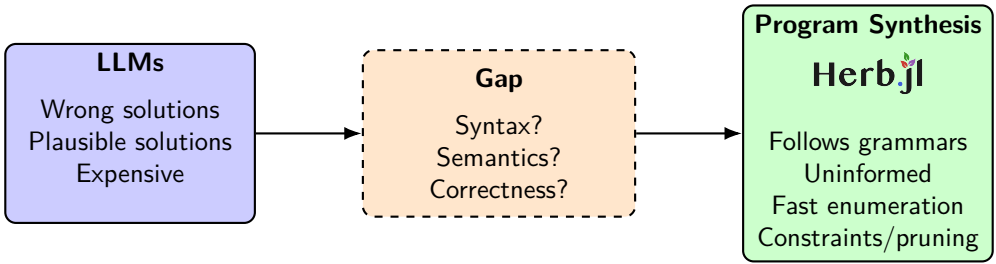
## LLMs for Program Synthesis

- We cannot just prompt the LLM for every decision. LLMs are **too expensive**

# The gap we want to bridge

## LLMs for Program Synthesis

- We cannot just prompt the LLM for every decision. LLMs are **too expensive**




## Goal

Keep the **guidance** from LLMs, but recover the **discipline** of synthesis.

# Problem statement

## Input

- an LLM
- a target language, via a grammar $\mathcal{G}$
- a specification $\mathcal{S}$

## Question

How can we follow the **guidance** of the LLM, while still following the **grammar** and finding a **satisfying program**?

# NeuGuS: Learning Neural Guidance for Synthesis

## Definition (Learning heuristics)

Given data $\mathcal{D} = \{(i, o, p) | p(i) = o, i \in \mathcal{E}\}$,

# NeuGuS: Learning Neural Guidance for Synthesis

## Definition (Learning heuristics)

Given data $\mathcal{D} = \{(i, o, p) | p(i) = o, i \in \mathcal{E}\}$,learn heuristic $h : \mathcal{E} \rightarrow \mathbb{R}^{|\mathcal{G}|}$.

## Learning from trees

Given

- specification $\mathcal{E}$/the grammar $\mathcal{G}$/program-so-far $p_0$/....

predict which grammar rule $i \in [|\mathcal{G}|]$ to pick.

# NeuGuS: Learning Neural Guidance for Synthesis

## Definition (Learning heuristics)

Given data $\mathcal{D} = \{(i, o, p) | p(i) = o, i \in \mathcal{E}\}$,learn heuristic $h : \mathcal{E} \to \mathbb{R}^{|\mathcal{G}|}$.

## Learning from trees

Given

- specification $\mathcal{E}$/the grammar $\mathcal{G}$/program-so-far $p_0$/....

predict which grammar rule $i \in [|\mathcal{G}|]$ to pick.

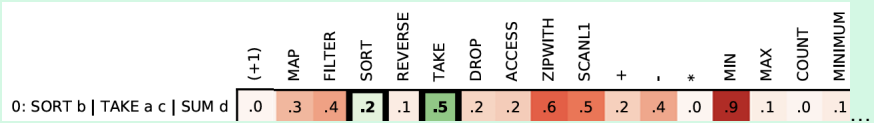
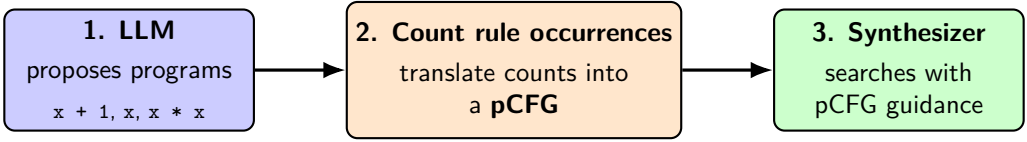



Figure: Heuristic values over grammar functions

# The common bridge: probabilistic grammars

## Most common idea

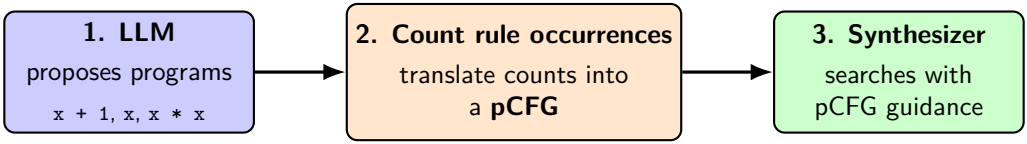
Use an LLM to produce a multiple programs, estimate grammar-rule probabilities from them, and guide synthesis with the resulting **pCFG**.



# The common bridge: probabilistic grammars

## Most common idea

Use an LLM to produce a multiple programs, estimate grammar-rule probabilities from them, and guide synthesis with the resulting **pCFG**.




## Interpretation

Likely programs are explored earlier. Unlikely rules and programs are delayed.

# Why pCFGs are not the full answer

## Key limitation

A context-free grammar (CFG) captures **syntax**. A pCFG adds **rule preferences**. But both are still mostly **context-free**.

# Why pCFGs are not the full answer

## Key limitation

A context-free grammar (CFG) captures **syntax**. A pCFG adds **rule preferences**. But both are still mostly **context-free**.

## What gets lost

1. structural context, and
2. useful sub-programs

# Why pCFGs are not the full answer

## Key limitation

A context-free grammar (CFG) captures **syntax**. A pCFG adds **rule preferences**. But both are still mostly **context-free**.

## What gets lost

1. structural context, and
2. useful sub-programs

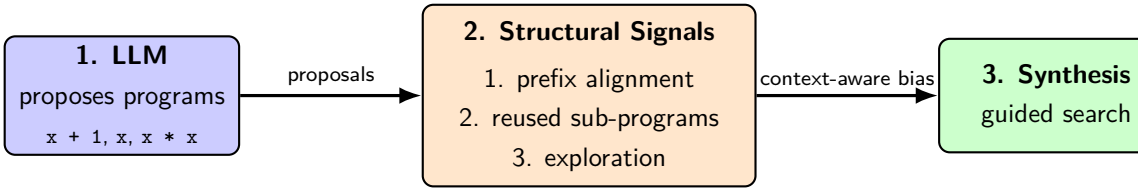
## Intuition

An LLM can prefer a rule *because of the program-so-far*. A plain pCFG usually cannot express that well.

# 1. Narcissus: Context-Aware LLM Guidance

## Goal

Inject **structural bias** into LLM guidance, while keeping correctness in the synthesizer.



# 1. Narcissus: What guidance do we want?

## Combined guidance

Let $\pi$ be the current partial program, and let $r$ be the possible next rule.

$$s(r \mid \pi) = \lambda_1 s_{\text{pref}}(r \mid \pi) + \lambda_2 s_{\text{sub}}(r \mid \pi) + \lambda_3 s_{\text{expl}}(r \mid \pi)$$

$$p(r \mid \pi) \propto s(r \mid \pi)$$

# 1. Narcissus: What guidance do we want?

## Combined guidance

Let $\pi$ be the current partial program, and let $r$ be the possible next rule.

$$s(r \mid \pi) = \lambda_1 s_{\text{pref}}(r \mid \pi) + \lambda_2 s_{\text{sub}}(r \mid \pi) + \lambda_3 s_{\text{expl}}(r \mid \pi)$$

$$p(r \mid \pi) \propto s(r \mid \pi)$$

## Signal 1: Prefix alignment

**Intuition:** If similar proposal programs used rule $r$ after the same prefix $\pi$, prefer $r$ again.

# 1. Narcissus: What guidance do we want?

## Combined guidance

Let $\pi$ be the current partial program, and let $r$ be the possible next rule.

$$s(r \mid \pi) = \lambda_1 s_{\text{pref}}(r \mid \pi) + \lambda_2 s_{\text{sub}}(r \mid \pi) + \lambda_3 s_{\text{expl}}(r \mid \pi)$$

$$p(r \mid \pi) \propto s(r \mid \pi)$$

## Signal 1: Prefix alignment

**Intuition:** If similar proposal programs used rule $r$ after the same prefix $\pi$, prefer $r$ again.
**Example:** Suppose 8 proposals contain prefix $\pi$, and in 6 of them the previous rules were $Int \to Int + 1$. Then

$$s_{\text{pref}}(Int \to Int + Int \mid \pi) = \frac{6}{8} = 0.75.$$

# Narcissus: What guidance do we want?

## Signal 2: Reused sub-programs

**Intuition:** If a fragment appears often across proposals, rules that help build that fragment should get a bonus.

# Narcissus: What guidance do we want?

## Signal 2: Reused sub-programs

**Intuition:** If a fragment appears often across proposals, rules that help build that fragment should get a bonus.
**Example:** If the fragment $(x + 1)$ appears in many proposals, then a rule that expands toward that fragment gets a high $s_{\text{sub}}$ score.

# Narcissus: What guidance do we want?

## Signal 2: Reused sub-programs

**Intuition:** If a fragment appears often across proposals, rules that help build that fragment should get a bonus.
**Example:** If the fragment $(x + 1)$ appears in many proposals, then a rule that expands toward that fragment gets a high $s_{\text{sub}}$ score.

## Signal 3: Exploration

**Intuition:** The LLM proposals may be faulty. Thus, give some chance to less-used rules (regularization term).

# Narcissus: What guidance do we want?

## Signal 2: Reused sub-programs

**Intuition:** If a fragment appears often across proposals, rules that help build that fragment should get a bonus.
**Example:** If the fragment $(x + 1)$ appears in many proposals, then a rule that expands toward that fragment gets a high $s_{\text{sub}}$ score.

## Signal 3: Exploration

**Intuition:** The LLM proposals may be faulty. Thus, give some chance to less-used rules (regularization term).
**Example:** If $Int \to x$ has already been heavily favored many times, then $Int \to Int * Int$ may get an exploration bonus, so search still checks an alternative region.

# Search intuition

## Rule of thumb

Change weights $\lambda_i$ according to these rules of thumb:

- near the root: use strong contextual guidance
- deeper down: reuse useful sub-programs
- throughout: keep some exploration alive

# Search intuition

## Rule of thumb

Change weights $\lambda_i$ according to these rules of thumb:

- near the root: use strong contextual guidance
- deeper down: reuse useful sub-programs
- throughout: keep some exploration alive

## Interpretation

We want to approximate the LLM proposals, without getting getting stuck in local minima.

# Narcissus Conclusion: Current state + ongoing work

## What we do want

Cheap, context-aware guidance for reliable program synthesis.

# Narcissus Conclusion: Current state + ongoing work

## What we do want

Cheap, context-aware guidance for reliable program synthesis.

## Initial results show:

- fewer candidates tested
- faster discovery of satisfying programs

# Narcissus Conclusion: Current state + ongoing work

## What we do want

Cheap, context-aware guidance for reliable program synthesis.

## Initial results show:

- fewer candidates tested
- faster discovery of satisfying programs

## Ongoing work:

- Formalize $\lambda$ dependency
- Study on LLM proposal quality
- Evaluation on other benchmarks

# 2. Symbolic Program Synthesis

**Grammars are dumb!**

Grammars admit many syntactically correct but undesirable programs:

# 2. Symbolic Program Synthesis

## Grammars are dumb!

Grammars admit many syntactically correct but undesirable programs:

- *useless* programs, that, e.g., do not contain the input symbol

# 2. Symbolic Program Synthesis

## Grammars are dumb!

Grammars admit many syntactically correct but undesirable programs:

- *useless* programs, that, e.g., do not contain the input symbol
- *redundant* programs, such as $1 * x, 1 * (x + 1), 0 + (2 * x)$

# 2. Symbolic Program Synthesis

## Grammars are dumb!

Grammars admit many syntactically correct but undesirable programs:

- *useless* programs, that, e.g., do not contain the input symbol
- *redundant* programs, such as $1 * x, 1 * (x + 1), 0 + (2 * x)$

## Constraints!

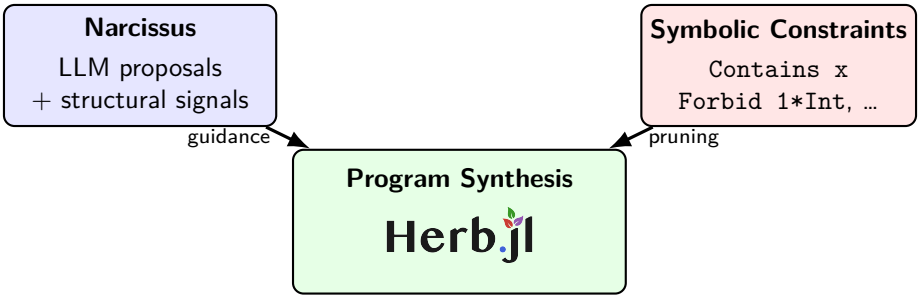
We introduce a

- language of constraints over programs,
  - `Forbid 1*Int`, `Contains x`, `Forbid 0+Int`, ...
- a constraint solver, to propagate these efficiently (called BART)

# 3. Neuro-Symbolic Program Synthesis

## Idea

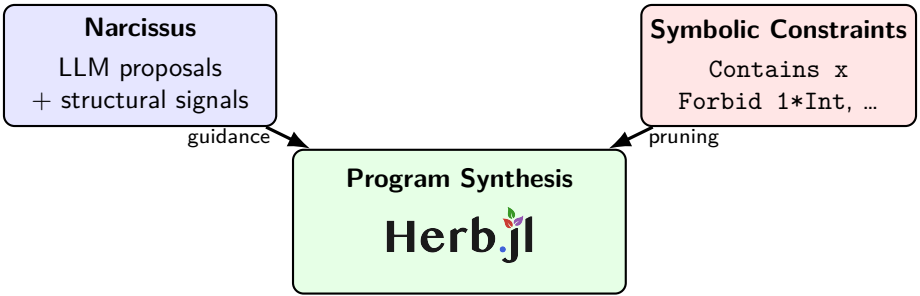
Combine **LLM guidance** with **symbolic pruning**.



# 3. Neuro-Symbolic Program Synthesis

## Idea

Combine **LLM guidance** with **symbolic pruning**.




LLMs guide search toward **plausible** programs, symbolic constraints prune **undesirable** ones.

# Summary and Conclusion

## Core messages

- LLMs are excellent at proposing plausible programs.

# Summary and Conclusion

## Core messages

- LLMs are excellent at proposing plausible programs.
- Program synthesis is excellent at enforcing correctness.

# Summary and Conclusion

## Core messages

- LLMs are excellent at proposing plausible programs.
- Program synthesis is excellent at enforcing correctness.
- We can attempt to combine both analyzing the underlying graph structure.

# Summary and Conclusion

## Core messages

- LLMs are excellent at proposing plausible programs.
- Program synthesis is excellent at enforcing correctness.
- We can attempt to combine both analyzing the underlying graph structure.
- Everything is Program Synthesis!

# Summary and Conclusion

## Core messages

- LLMs are excellent at proposing plausible programs.
- Program synthesis is excellent at enforcing correctness.
- We can attempt to combine both analyzing the underlying graph structure.
- Everything is Program Synthesis!

## Talk to me!

If you are tired of writing

- code,
- differential equations, or
- process models

yourself, come talk to me afterwards.

Why LLMs do/do not work

What is Program Synthesis?

Neural Program Synthesis via LLM guidance: Narcissus

Symbolic Program Synthesis