Guiding Synthesis with AI

Julian Parsert September 5, 2024

Unrelated Problem, Help Needed

Ramsey-Based Termination:

tools use disjunctive well-founded relations over the transitive closure of a transition relation.

NEW idea (van Oostrom):

We can use affluence INSTEAD of transitivity.
Affluence is strictly weaker than *transitivity*.

Problem with new idea:

Verifying affluence is difficult. theoretically weaker so easier than transitivity ?
"Completing" disj. rel ends up being transitive "by accident".

Guiding Enumerative Program Synthesis with Large Language Models

with Yixuan Li and Elizabeth Polgreen (CAV2024)

Function Synthesis

$$\exists f.\ \phi[F \mapsto f]$$

Does there exist a function f such that it satisfies the specification ϕ?
ϕ is a formula in a background theory.

Syntax-Guided Synthesis (SyGuS)

SyGuS is a problem of synthesising

- a function F within
- a theory τ that satisfies
- a semantic specification ϕ
- with a syntactic restriction G.

$\rightarrow$ SyGuS-IF closely follows SMTLIB

SyGuS Tools/competitions

$$\exists f.\ \forall xy.f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$$
$$f \in G$$

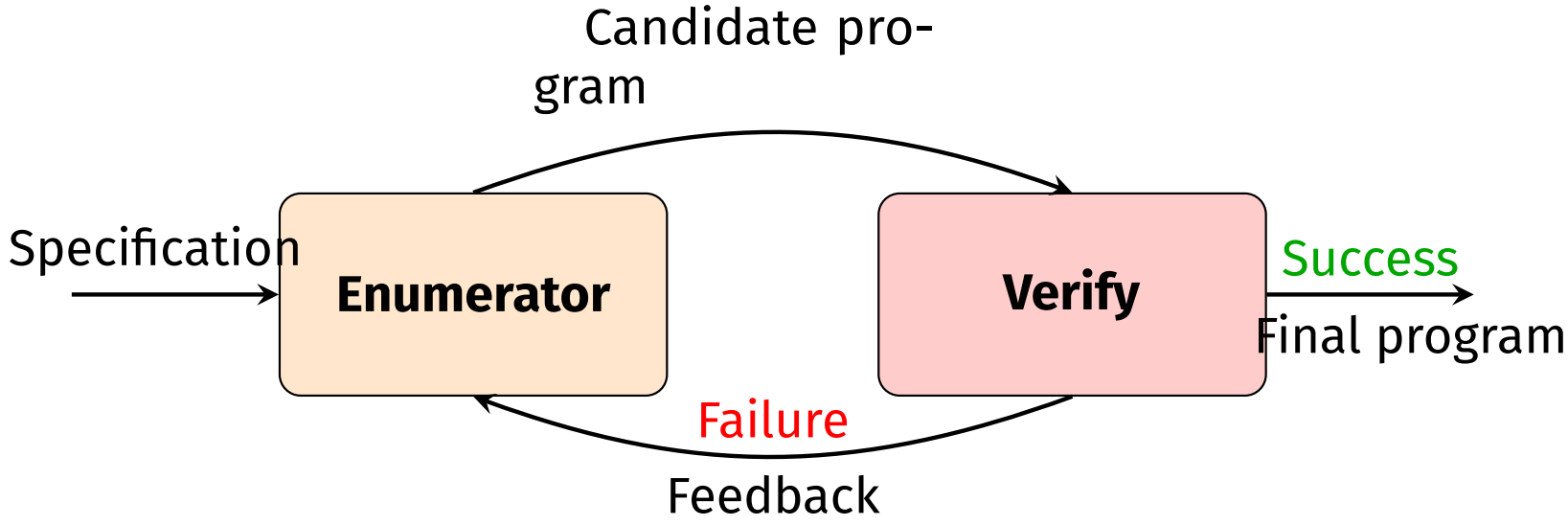
```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((I Int) (B Bool))
    ((I Int (x y 0 1 (+ I I) (- I I) (ite B I I)))
     (B Bool ((and B B) (or B B) (not B)
              (= I I) (<= I I) (>= I I)))))
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
(check-synth)
```

```
(define-fun max2 ((x Int) (y Int)) Int   (ite (>= x z)
    x z))
```

Enumerative Synthesis

In Syntax-Guided Synthesis:
Use grammar to systematically enumerate space of possible programs

Counter-Example Guided Inductive Synthesis (CEGIS)



pCFG

A context-free grammar with probabilities attached to the rules.

The probability associated with a rule $\alpha \to \beta$

$$[\texttt{Start} \to \texttt{(ite StartBool Start Start)}] \mapsto 3/19$$
$$[\texttt{Start} \to \texttt{x1}] \mapsto 3/19$$
$$[\texttt{Start} \to \texttt{x2}] \mapsto 3/19$$
$$[\texttt{Start} \to \texttt{x3}] \mapsto 4/19$$
$$[\texttt{StartBool} \to \texttt{(>= Start Start)}] \mapsto 3/19$$

Synthesis with LLMs: Idea

Let's ask ~~blockchains~~ LLMs to solve ~~everything~~ synthesis.

Synthesis with LLMs: Idea

Let's ask ~~blockchains~~ LLMs to solve ~~everything~~ synthesis.

If the LLM fails, try again ($\times n$ where $n = 6$, trust me)

Synthesis with LLMs: Idea

Let's ask ~~blockchains~~ LLMs to solve ~~everything~~ synthesis.

If the LLM fails, try again ($\times n$ where $n = 6$, trust me)

If it still fails?
Hypothesis:
Correct solutions "syntactically close" to LLM suggestions

Synthesis with LLMs: Idea

Let's ask ~~blockchains~~ LLMs to solve ~~everything~~ synthesis.

If the LLM fails, try again ($\times n$ where $n = 6$, trust me)

If it still fails?
Hypothesis:
Correct solutions "syntactically close" to LLM suggestions

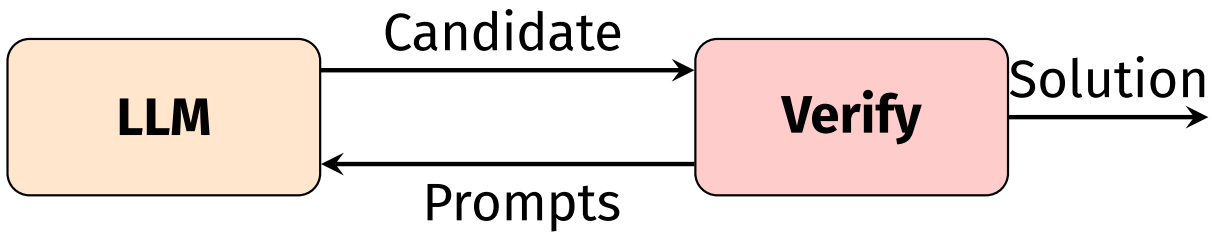
Technique

- Use symbolic enumerator,
- to narrow synthesis search space
- by prioritising functions "near" LLM suggestions.

3 Synthesis Methods with LLMs

- Stand-alone LLM
- pCFG-synth (pCFG = probabilistic context-free grammar)
- iLLM-synth (iLLM = integrated Large Language Model)

Stand-alone LLM



Up to 6 synthesis attempts per benchmark

LLM Prompts

```
(set-logic LIA)
(declare-var x1 Int)
(declare-var x2 Int)
(declare-var x3 Int)
(constraint (>= (Prog x1 x2 x3) x1))
(constraint (>= (Prog x1 x2 x3) x2))
(constraint (>= (Prog x1 x2 x3) x3))
(constraint (or (= x1 (Prog x1 x2 x3)) (or (= x2 (Prog x1 x2 x3)) (= x3 (Prog x1 x2 x3)))))
(check-synth)
You are a good synthesizer. Do you know what "(define-fun Prog ((x1 Int) (x2 Int) (x3 Int)) Int" is doing?
Write only one Lisp-like method "defun Prog" without any built-in methods or arrays.
Requirements:
1. No built-in functions.
2. Never violate the SMT-LIB constraints above.
3. Pay attention to the define functions.
4. Ensure the response contains one and only one function.
5. Do not include any iterations, BitVec, or Int notations in the function body.
Write it correctly, or I will lose my job and 100 grandmothers will die. Don’t disappoint me.
Write only one Lisp-like method "defun Prog" that never violates the SMT-LIB constraints above.
```

LLM Prompts

Response from LLM

```
(defun Prog (x1 x2 x3)
    (ite (>= x1 x2) (ite (>= x1 x3) x1 x3) (ite (>= x2 x3) x2 x3)))
```

Request for converting Lisp to SMT-LIB code for the last response

```
You are a good programming language converter. Convert the Lisp function to SMT-LIB:
Based on the Lisp code provided above, convert the 'defun' Lisp-like code to a corresponding SMT-LIB function. Use SMT-
       LIB syntax starting with (define-fun
Follow these guidelines:
1. Only give me the function definition starting with '(define-fun'.
2. Pay attention to types. If there are bit-vector terms, they need to be of the same width.
3. Ensure the SMT-LIB function contains one and only one function definition starting with '(define-fun'.
4. Do not include any iterations, BitVec, or Int notations in the function body.
5. Use the assigned values from the Lisp code during translation.
6. Do not introduce any variables that do not exist in the Lisp function.
Rules for SMT-LIB: +, -, *, ite, >, =, <, >=, <=, and, or, not, true, false.
```

LLM Prompts

LLM-Generated program (Stand-alone LLM is done)

```
(define-fun Prog ((x1 Int) (x2 Int) (x3 Int)) Int
    (ite (>= x1 x2) (ite (>= x1 x3) x1 x3) (ite (>= x2 x3) x2 x3)))
```

Prompt requesting a revised solution

```
You are close to the right answer. Take another guess. You have to try something different, think harder. Write a
       different Lisp method that never violates the SMT-LIB constraints above again.
```

Stand-alone LLM

OpenAI's GPT-3.5

- Solves 49% of benchmarks.
- 4 attempts on average for a correct solution.
- Average generation time: 5 seconds.

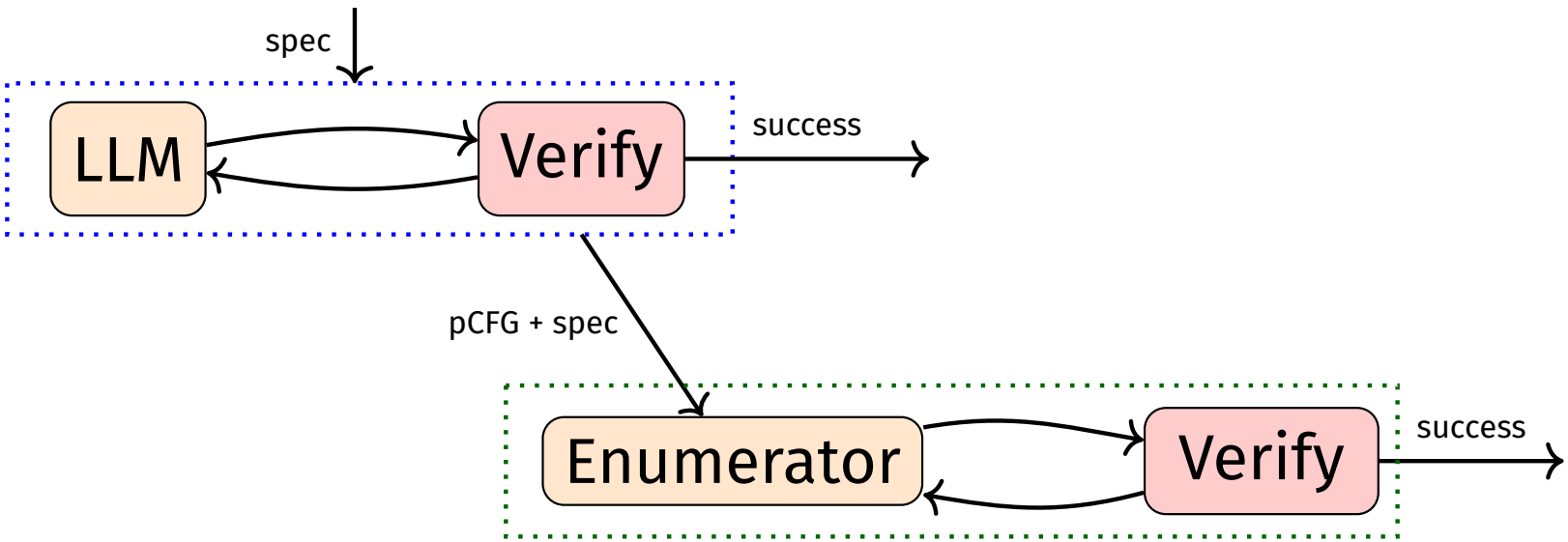
What to do after $\not{3}\ \not{4}$ 6 LLM attempts, are we lost?

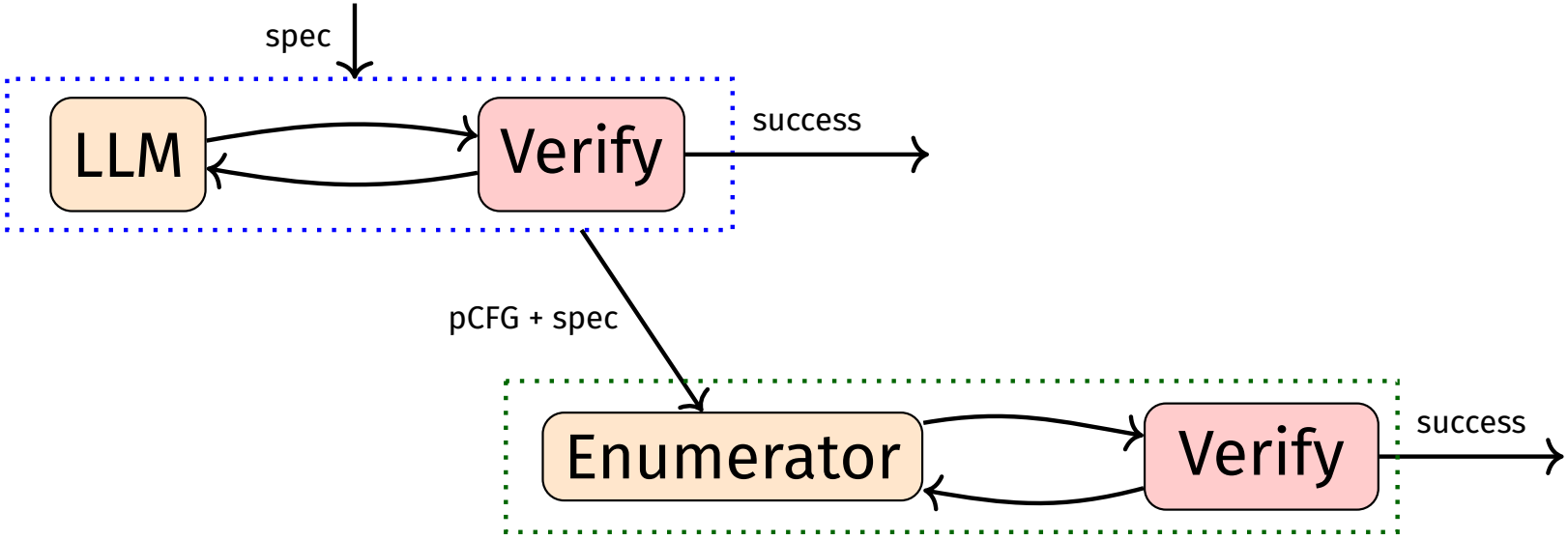
Let's apply hypothesis:

Correct solutions might be "syntactically close" to LLM suggestions.

pCFG-synth

1. ask LLM for solutions (as before)
2. generate pCFG from LLM candidates
3. enumerate according to pCFG (sampling)





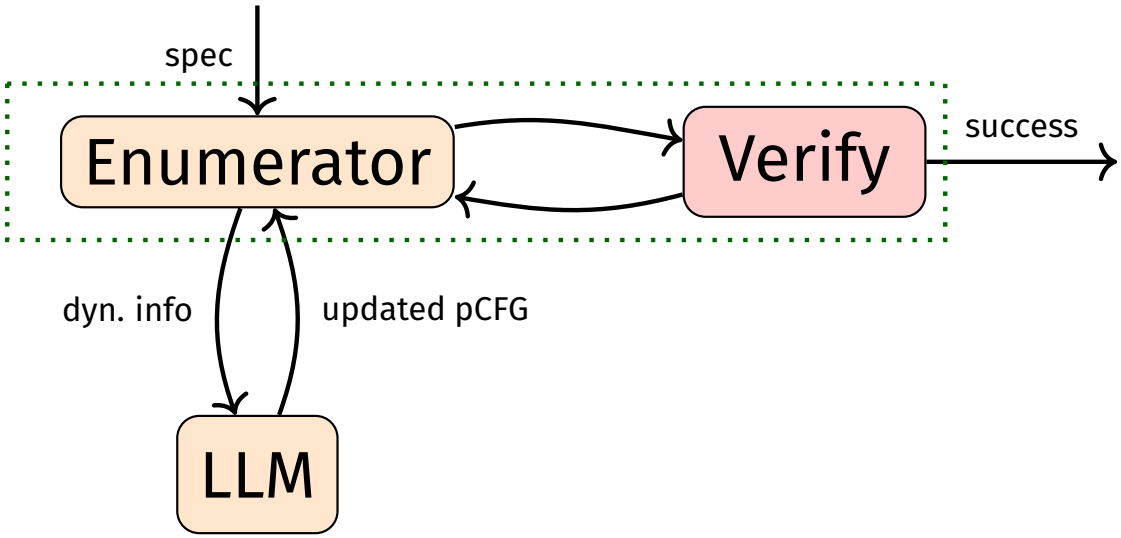
But we get info during enumeration:

- Partial Programs
- Previous candiates
- CounterExamples (CEGIS)
- …

Let’s use this dynamic information

iLLM-synth

- integrates LLM prompts for dynamic information use
- LLM suggests helper functions for partial programs
- expand and update weights of pCFG using response

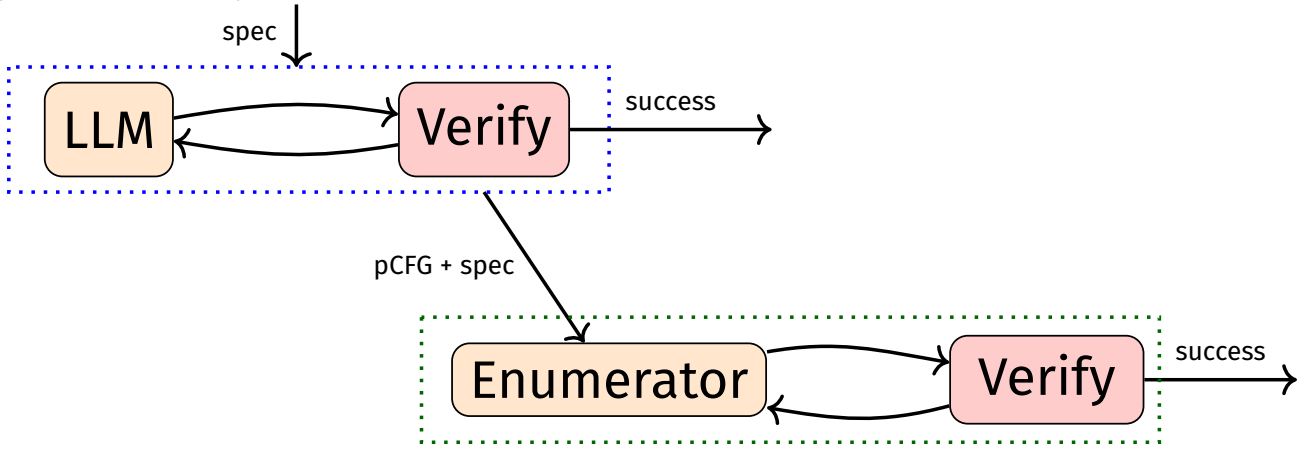


iLLM-synth Prompts

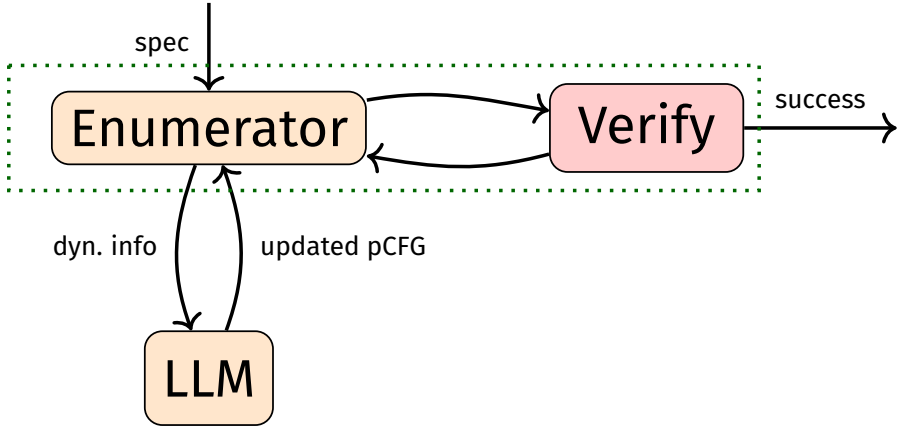
```
You are teaching a student to write SMT-LIB. The student must write a function that satisfies the following constraints:
(constraint ...
...
So far, the student has written this code:
(define-fun Prog ((x1 Int) (x2 Int) (x3 Int)) Int
    (ite ?? ?? ??)
Can you suggest some helper functions for the student to use to complete this code and replace the ??
You must print only the code and nothing else.
```

```
You are teaching a student to write SMT-LIB. The student may find the following functions useful:
(define-fun Prog ...
...
The student must write a function that satisfies the following constraints:
(constraint ...
...
The last solution the student tried was this, but the teacher marked this solution incorrect:
(define-fun Prog ...
This solution was incorrect because it did not work for the following inputs:
x3  =  (- 3)
x2  =  (- 2)
x1  =  (- 4)
```

pCFG-synth



iLLM-synth



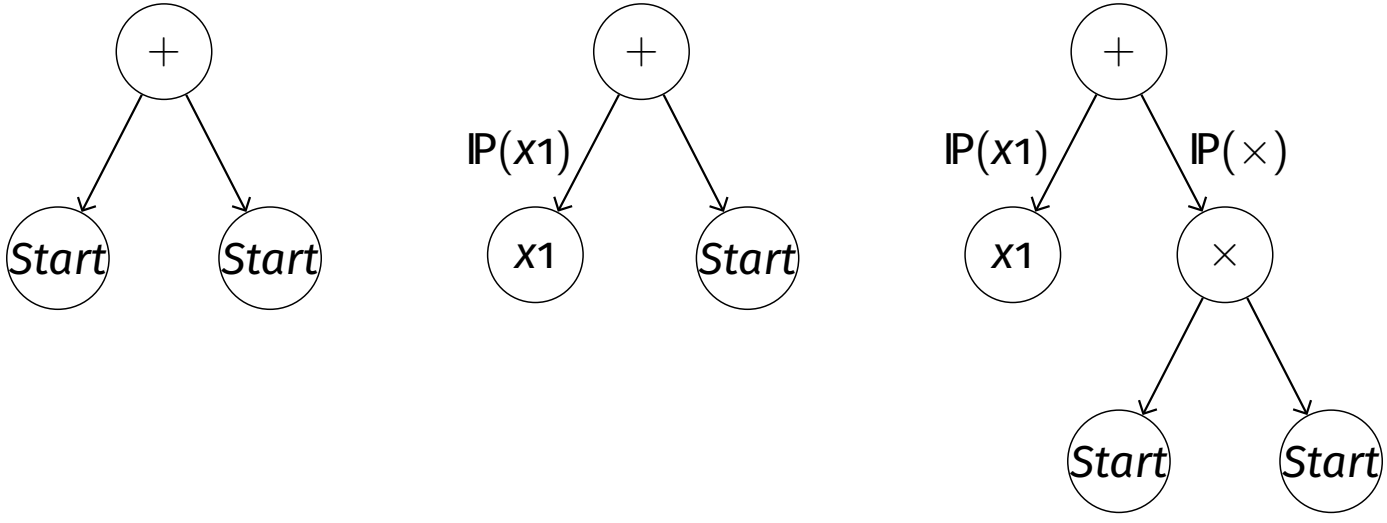
Enumerators

Different search methods:

- Top-down enumerator.
- A^* enumerator.

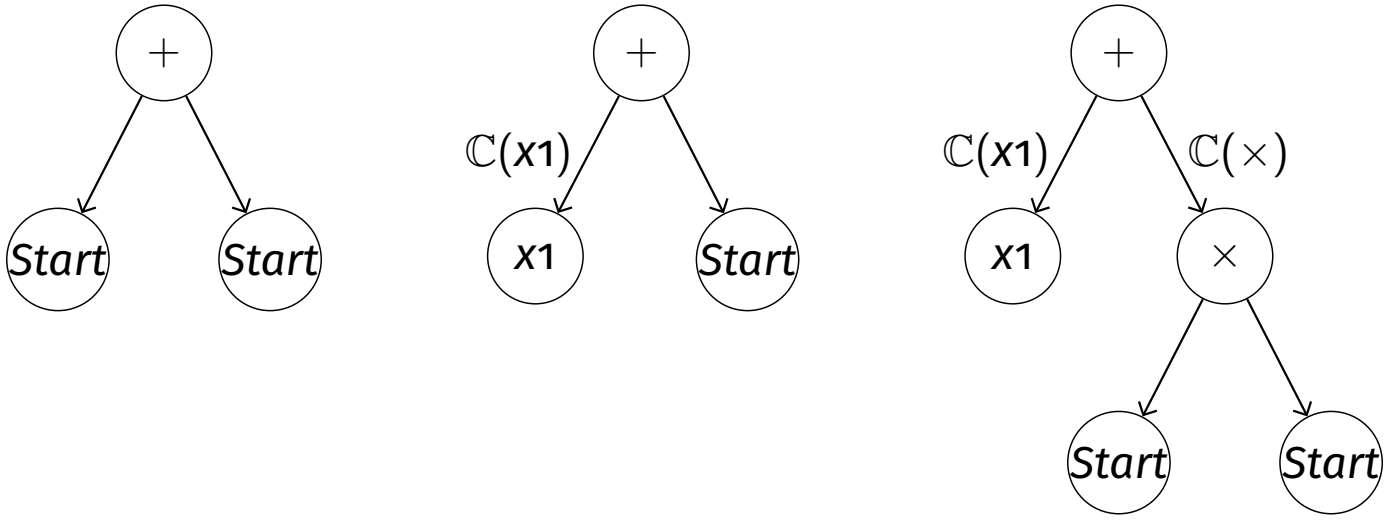
Top-down Enumerator

- Uses probabilistic rule to navigate the grammar tree.
- Generates unique programs, discarding duplicates and respecting a depth limit.
- Prioritizes new and complete programs to improve search productivity.

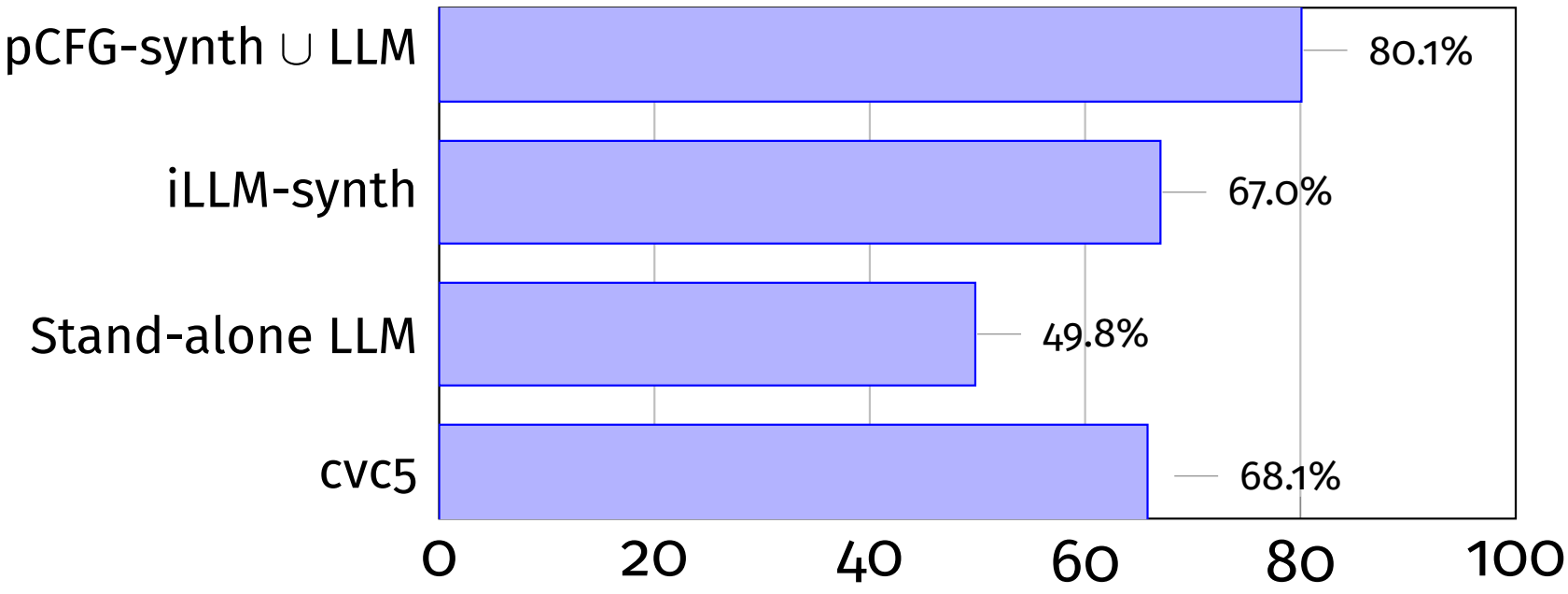


A^* Enumerator

- Chooses paths based on minimizing current path cost plus estimated cost to goal.
- Focuses on paths with lower combined actual and predicted costs.

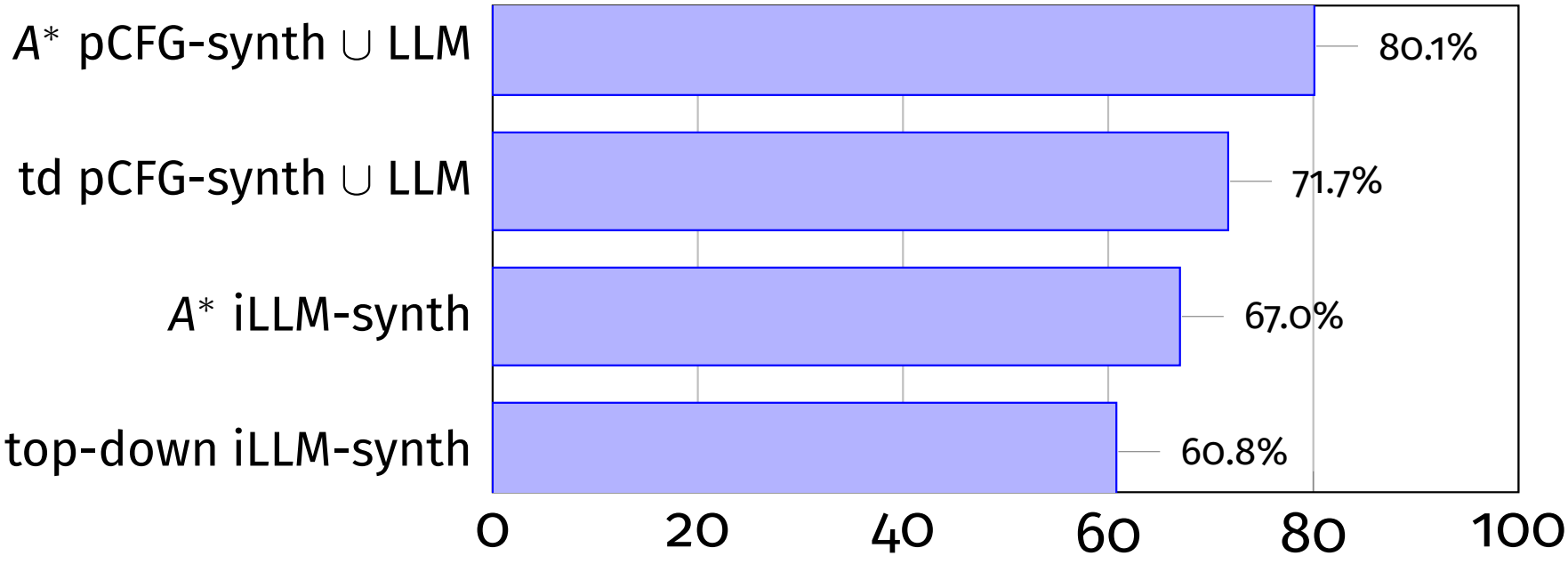


Results (600s Timeout)



- The average length of a solution: LLM is 4.7x than cvc5.

Results based on Enumerators



- A^* $>$ top-down.

Big Ugly Table

	BV (384)		LIA (87)		INV (138)		Total (609)	
Methods	#	time(s)	#	time(s)	#	time(s)	#	%
LLM only	137	*13.5*	54	*7.10*	112	*29.2*	303	49.8%
e-pCFG-synth ◇	196.0	48.3	24.0	40.0	25.4	100.5	245.4	40.3%
A^*-pCFG-synth	262	60.1	35	72.7	25	99.7	322	52.9%
LLM ∪ *e*-pCFG-synth	255.0	*37.0*	64.0	*17.20*	117.7	*40.4*	436.7	71.7%
LLM ∪ A^*-pCFG-synth	**305.0**	*35.0*	65.0	*18.1*	**118.0**	*33.6*	**488.0**	80.1%
e-iLLM-synth ◇	241.0	*88.2*	63.4	*9.3*	65.3	*25.4*	370.0	60.8%
A^*-iLLM-synth ◇	272.3	*24.6*	**68.3**	*20.8*	67.3	*43.6*	408.0	67.0%
enumerator◇	142.7	7.2	25.0	1.53	21.0	3.2	188.7	31.0%
A^*	253.0	25.4	34.0	73.19	22.0	31.1	309.0	50.7%
cvc5	292.0	17.1	43.0	19.53	80.0	23.6	415.0	68.1%

Failure of standalone LLM

- Constraints too long/complex
- simple syntactic errors (wrong place for operators)
- wrong nesting (e.g. if-then-else)
- ...

Neuro-symbolic approach can help with most problems.

Thank you

Feel free to contact me for questions, ideas, collaboration, ...

julian.parsert@gmail.com