Making Program Synthesis Faster (and making your code faster with program synthesis)

Dr Elizabeth Polgreen

Royal Academy of Engineering Research Fellow
Assistant Professor, University of Edinburgh

Automatically generating code that satisfies the user's specification

Hasn’t this all been solved?

DEVOPS GENERAL SECURITY

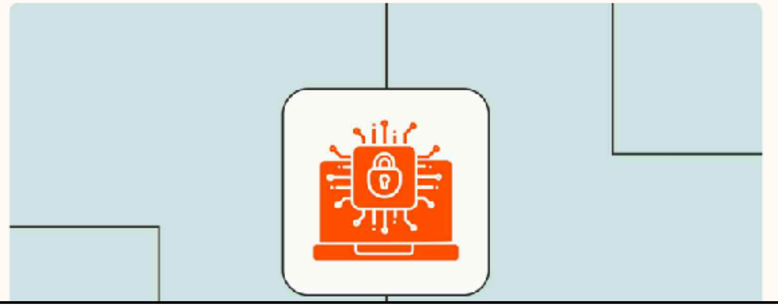
GitHub Copilot: A potential security risk?

By **arnavsharma**

MAY 10, 2023

5 security risks of generative AI and how to prepare for them

By Elisa Silverman · June 14, 2023



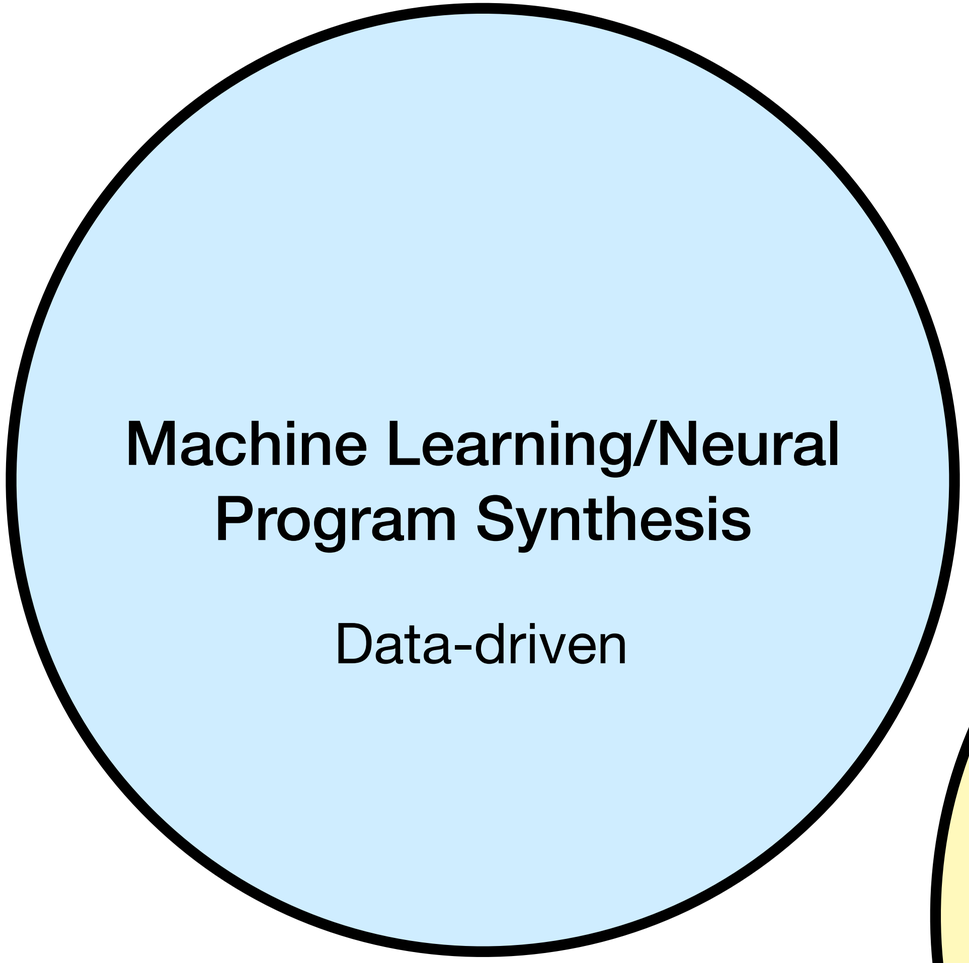
[Submitted on 20 Aug 2021 (v1), last revised 16 Dec 2021 (this version, v3)]

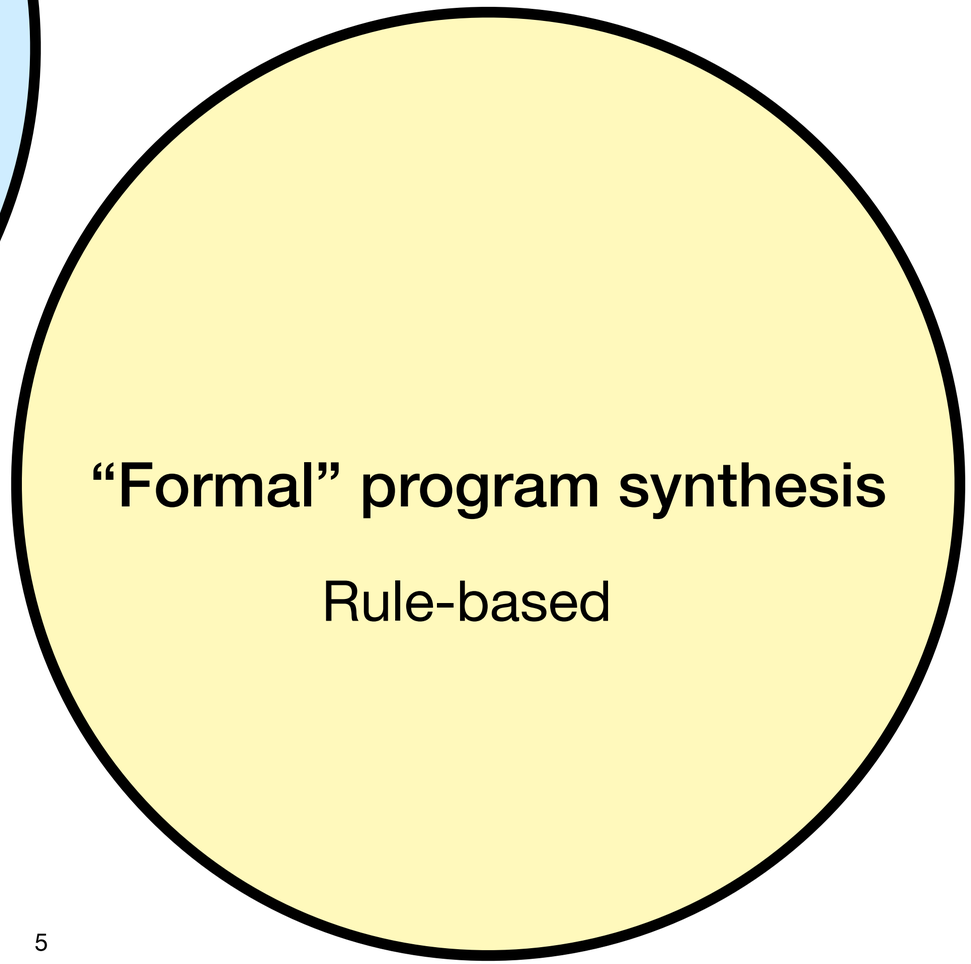
Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions

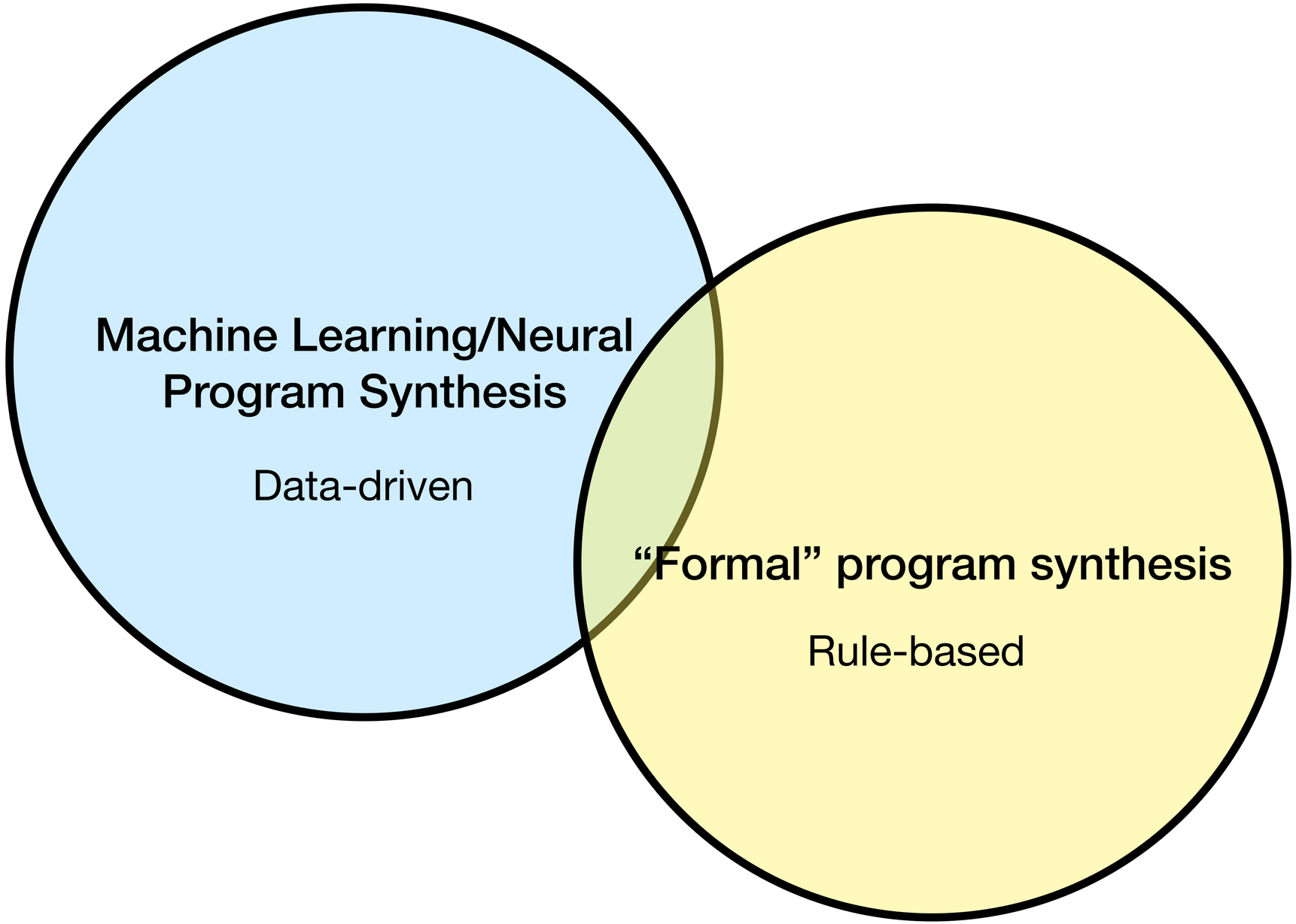
Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri

There is burgeoning interest in designing AI-based systems to assist humans in designing computing systems, including tools that automatically generate computer code. The most notable of these comes in the form of the first self-described `AI pair programmer', GitHub Copilot, a language model trained over open-source GitHub code. However, code often contains bugs – and so, given the vast quantity of unvetted code that Copilot has processed, it is certain that the language model will have learned from exploitable, buggy code. This raises concerns on the security of Copilot's code contributions. In this work, we systematically investigate the prevalence and conditions that can cause GitHub Copilot to recommend insecure code. To perform this analysis we prompt Copilot to generate code in scenarios relevant to high-risk CWEs (e.g. those from MITRE's "Top 25" list). We explore Copilot's performance on three distinct code generation axes -- examining how it performs given diversity of weaknesses, diversity of prompts, and diversity of domains. In total, we produce 89 different scenarios for Copilot to complete, producing 1,689 programs. Of these, we found approximately 40% to be vulnerable.

Comments: Accepted for publication in IEEE Symposium on Security and Privacy 2022


Machine Learning/Neural
Program Synthesis
Data-driven


“Formal” program synthesis
Rule-based


Machine Learning/Neural Program Synthesis
Data-driven
“Formal” program synthesis
Rule-based

This talk

- What is synthesis?
- Making your code faster with program synthesis:
  - C2TACO (GPCE 2023)
  - mlirSynth (PACT 2023)
- Making synthesis faster:
  - Guiding with reinforcement learning (AAAI 2024)
  - Guiding with LLMs

This talk

- What is synthesis?
- Making your code faster with program synthesis:
  - C2TACO (GPCE 2023)
  - mlirSynth (PACT 2023)
- Making synthesis faster:
  - Guiding with reinforcement learning (AAAI 2024)
  - Guiding with LLMs

Formal Program Synthesis

$$\exists P \forall x \,.\, \sigma(P, x)$$

Does there exist a function P such that, for all possible inputs x, the specification σ will evaluate to true for P and x.

Formal Program Synthesis

$$\exists P \forall x \,.\, \sigma(P, x)$$

Does there exist a function P such that, for all possible inputs x, the specification σ will evaluate to true for P and x.

σ is a quantifier free formula in a background theory, e.g., Linear Integer Arithmetic

NB: we can write specs with input-output examples as quantifier free formula

Formal Program Synthesis

```
int f(int x, int y)
{
   ???
}
@ensures: @ret ≥ x ∧ @ret ≥ y ∧ (@ret = x ∧ @ret = y)
```

Formal Program Synthesis

```
int f(int x, int y)
{
   ???
}
@ensures: @ret ≥ x ∧ @ret ≥ y ∧ (@ret = x ∧ @ret = y)
```

$$\exists f.\, \forall x, y.\;\; f(x,y) \geq y \;\; \wedge \;\; f(x,y) \geq x \;\; \wedge \;\; (f(x,y) = x \vee f(x,y) = y)$$

Solution: f finds the max of x and y

Defining the search space

Syntax-Guided Synthesis

```
int f(int x, int y)
{
  ???
}
@ensures: @ret ≥ x ∧ @ret ≥ y ∧ (@ret = x ∧ @ret = y)
```

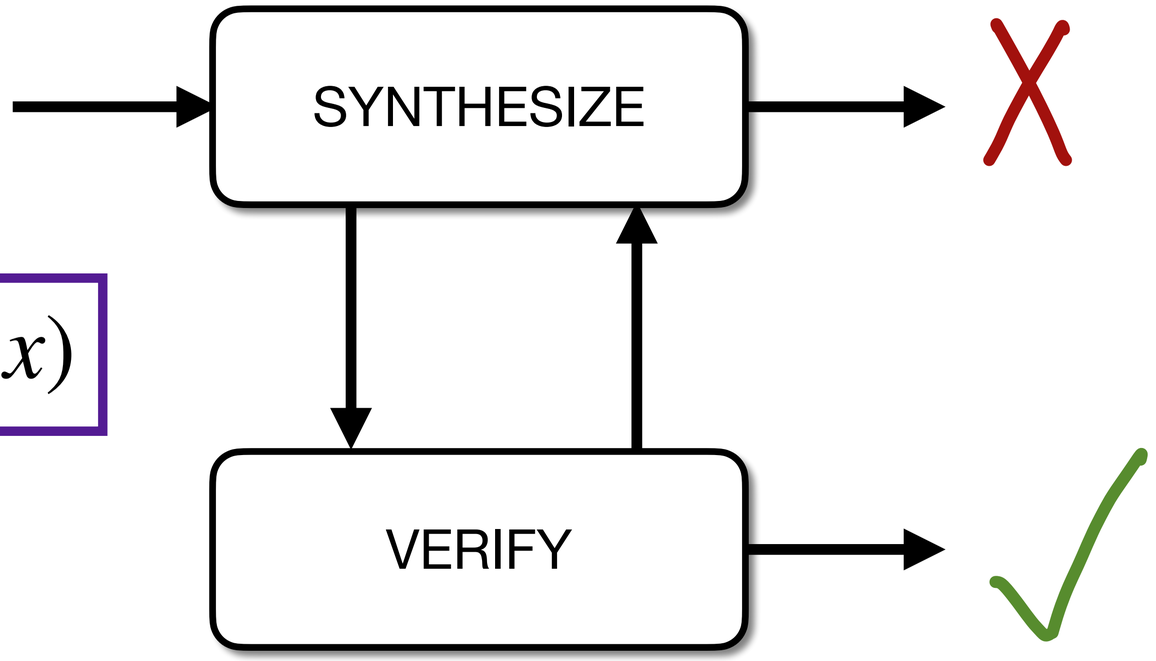
```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Context Free Grammar

Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

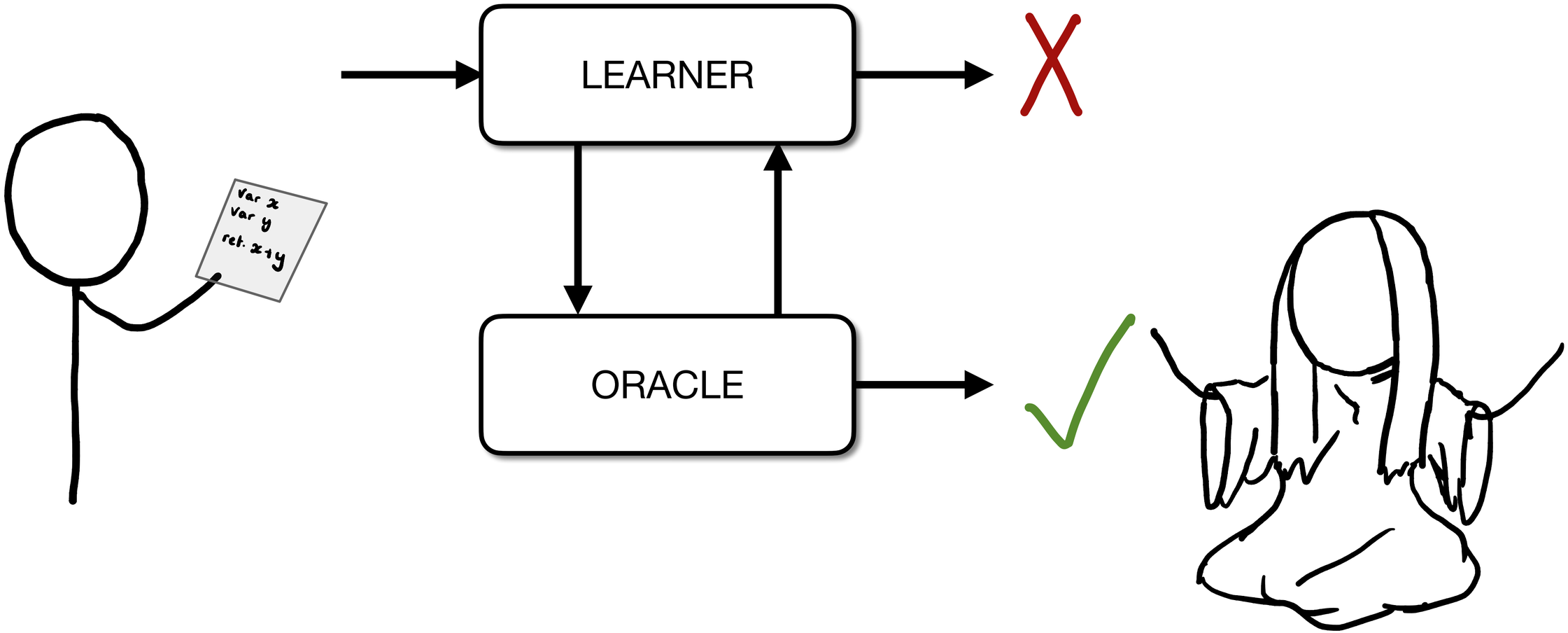
$$\exists P \forall x \,.\, \sigma(P, x)$$



Combinatorial sketching for finite programs - Solar Lezama et al

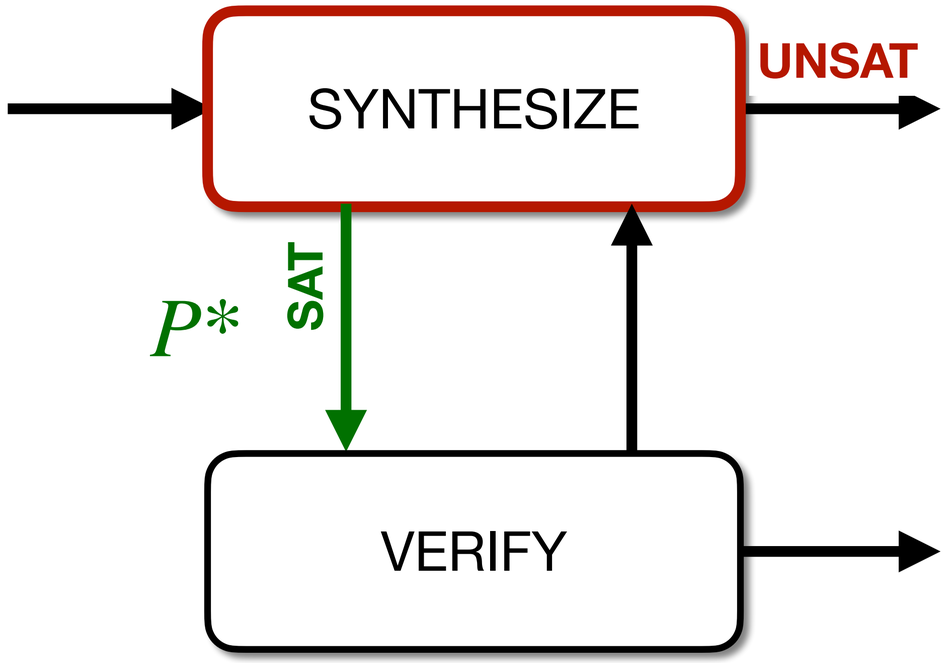
Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis



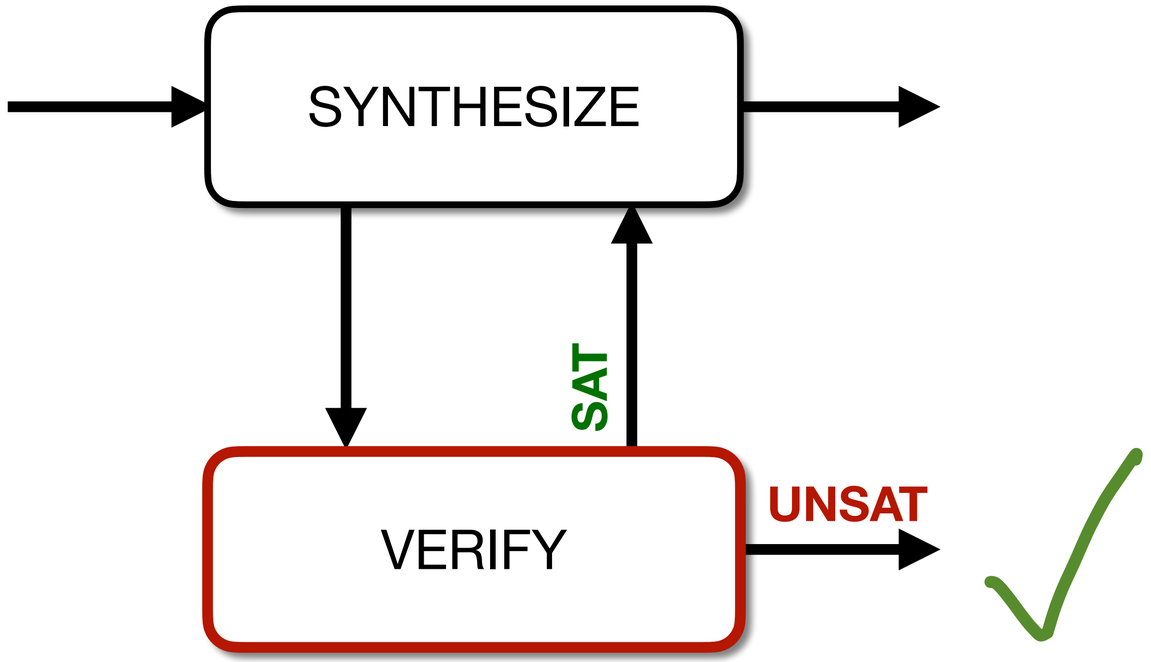
Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis



Algorithms for formal synthesis

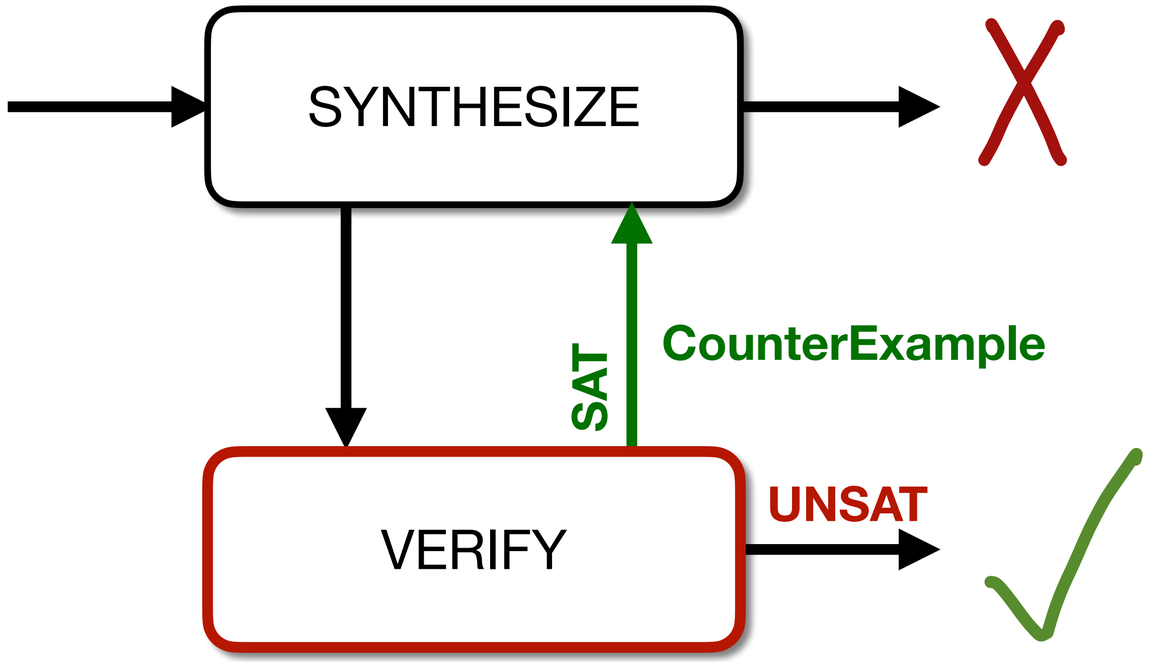
Counterexample Guided Inductive Synthesis



$$\exists x \,.\, \neg\sigma(P*, x)$$

Algorithms for formal synthesis

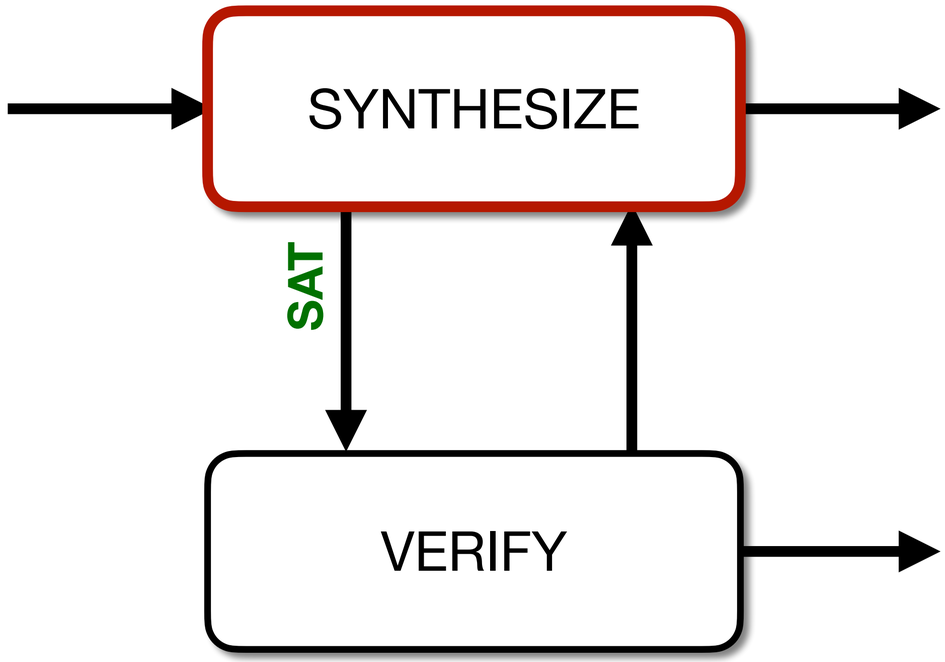
Counterexample Guided Inductive Synthesis



Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

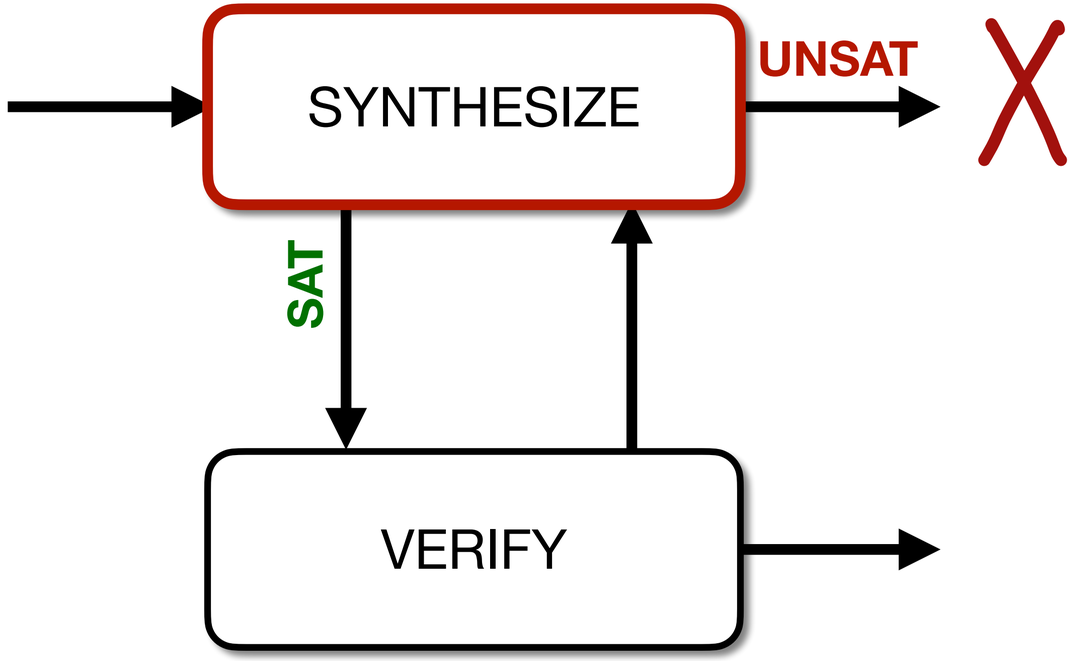
$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



Algorithms for formal synthesis

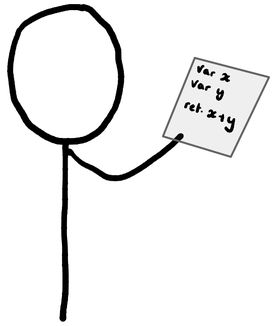
Counterexample Guided Inductive Synthesis

$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$



Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



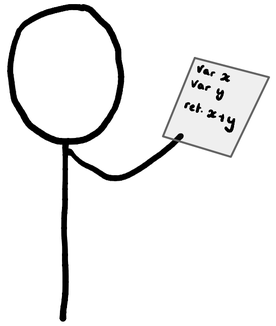
$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$

Guess	Counterexamples
x	

$\exists x\,.\,\neg\sigma(P*, x)$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



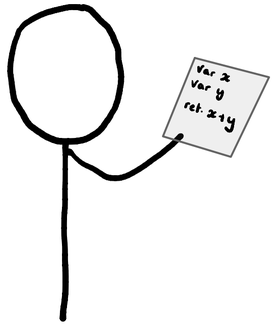
$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$

Guess	Counterexamples
x	x=1, y=2

$\exists x\,.\,\neg\sigma(P*, x)$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



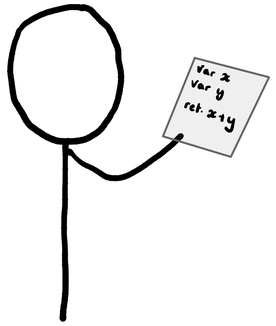
$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	

$$\exists x\,.\,\neg\sigma(P*, x)$$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



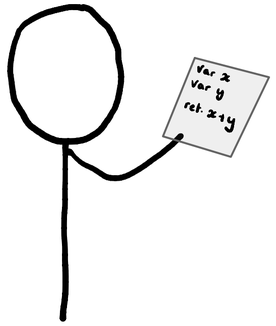
$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	x=2, y=1

$$\exists x\,.\,\neg\sigma(P*, x)$$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



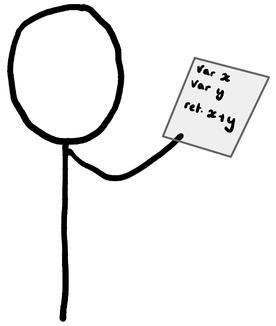
$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	x=2, y=1
$(x \geq y)?x : y$	

$$\exists x\,.\,\neg\sigma(P*, x)$$

Max

$$\exists f\,.\,\forall x, y\,.\;\; f(x,y) \geq y \;\;\wedge\;\; f(x,y) \geq x \;\;\wedge\;\; (f(x,y) = x \vee f(x,y) = y)$$



$$\exists P\,.\,\forall x_i\,.\,\sigma(P, x)$$

Guess	Counterexamples
x	x=1, y=2
y	x=2, y=1
$(x \geq y)?x : y$	Correct!

$$\exists x\,.\,\neg\sigma(P*, x)$$

The Oracle

$$\exists x \,.\, \neg \sigma(P*, x)$$

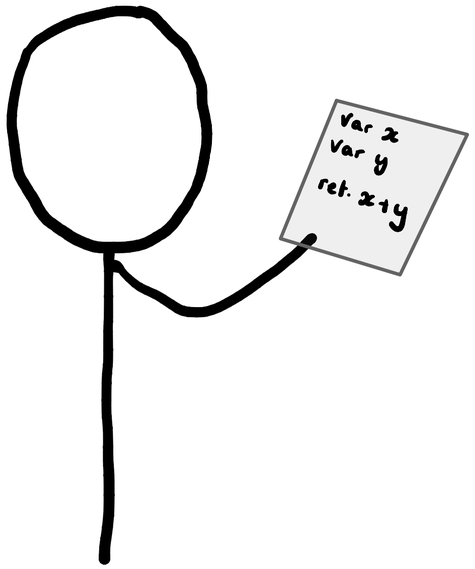
The Oracle

- Checking against input-output specifications is easy: just execute the code
- Checking against arbitrary formula is harder:
  - Use an SMT solver

$$\exists x \,.\, \neg\sigma(P*, x)$$

The Learner

$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$

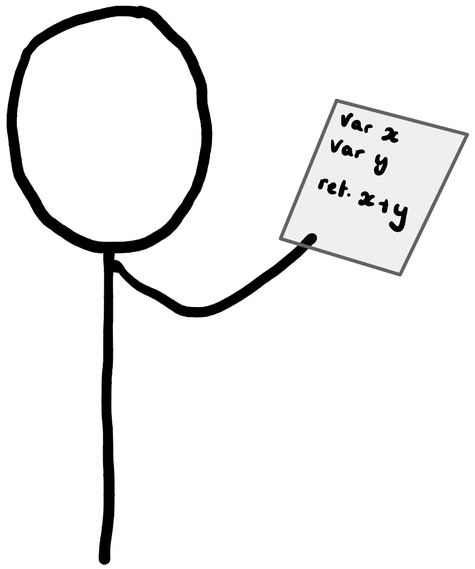


- Enumerate through the grammar (with heuristics)

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

The Learner

$$\exists P \,.\, \forall x_i \,.\, \sigma(P, x)$$

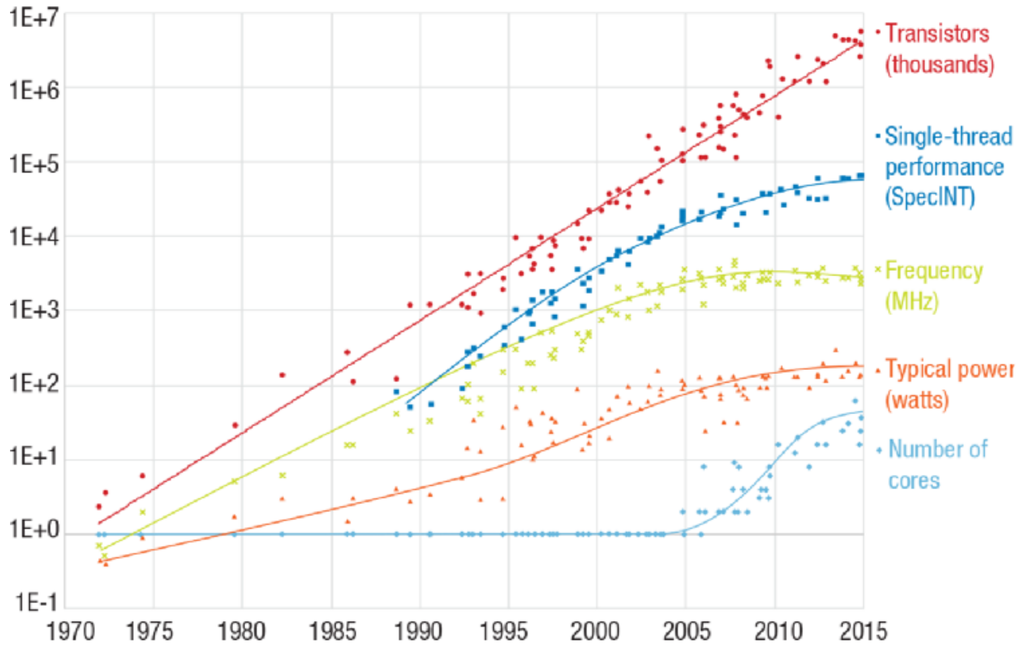


- Enumerate through the grammar (with heuristics)
- Symbolic encoding:
  - use selector variables to choose which parts of the grammar to use.
  - Ask an SMT solver to assign values to those selector variables.

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Making your code faster

Making your code faster

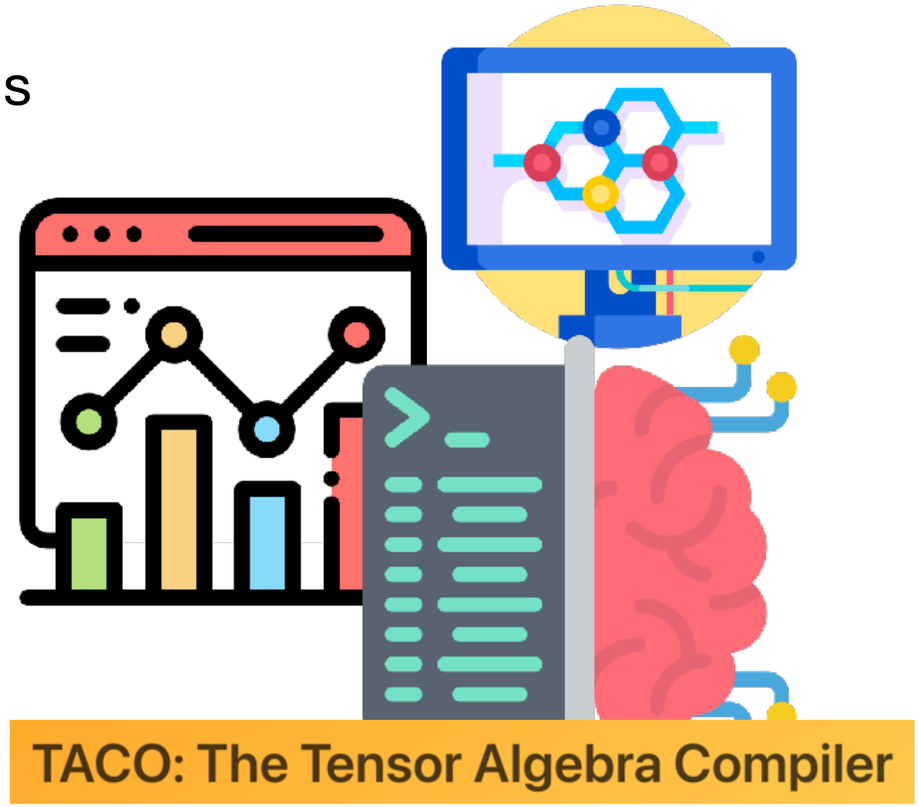


Moore's Law is dead?

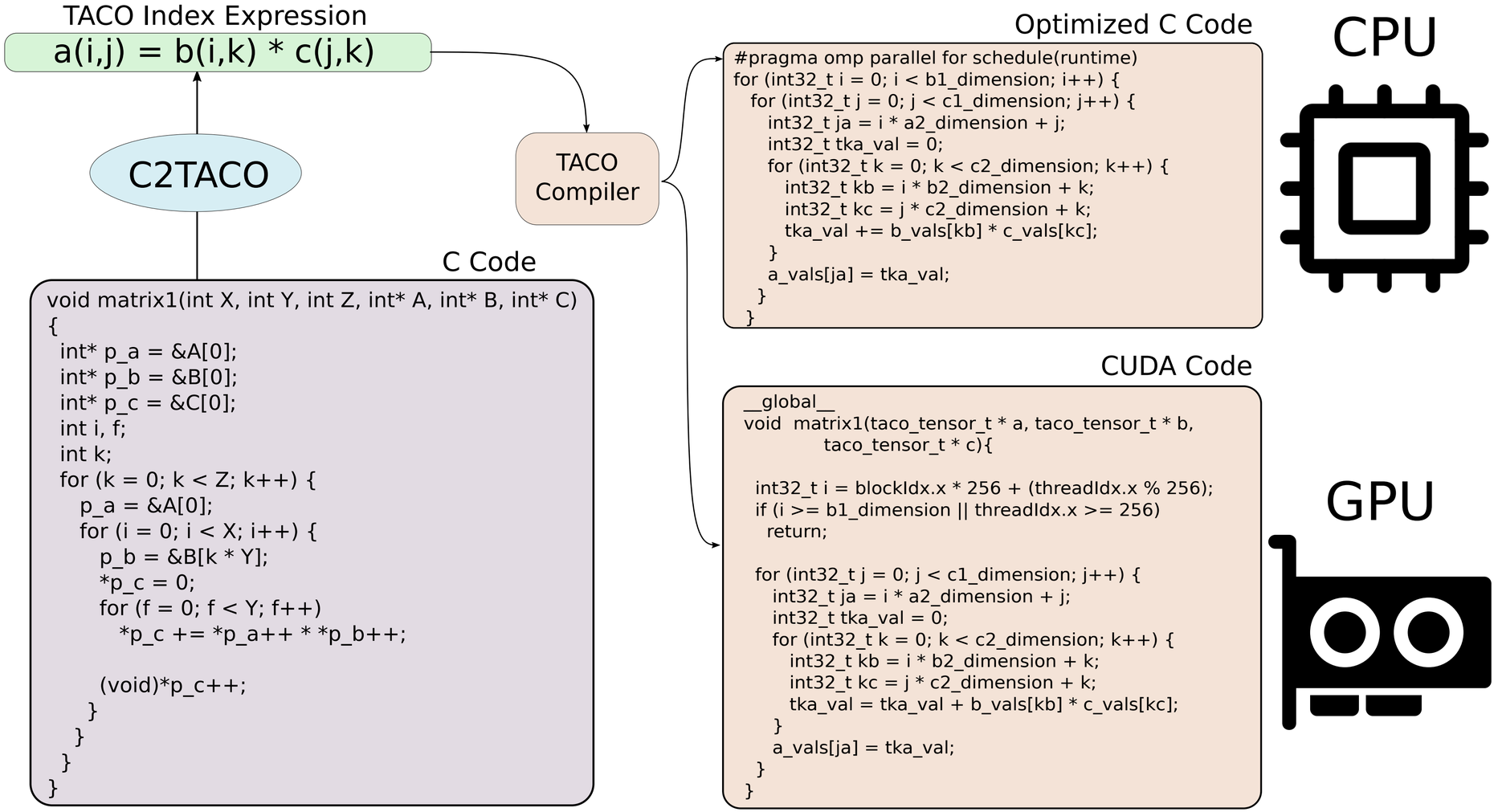
- Previously all code worked on all hardware
- Hardware is now becoming more specialized, with correspondings DSLs
- Using this specialized hardware gives performance gains
- What about legacy code?

Making your code faster

- Machine learning workloads are dominated by tensor code
- Key to efficiency: highly parallelised dense algebra
- DSLs like TACO make this easy for new applications
- What about legacy code?



C2TACO



C2TACO: Lifting Tensor Code to TACO - José Wesley de Souza Magalhães, Jackson Woodruff, Elizabeth Polgreen, Michael O'Boyle

C2TACO - Specification

⟨*PROGRAM*⟩ ::= ⟨*TENSOR*⟩ = ⟨*EXPR*⟩

⟨*TENSOR*⟩ ::= ⟨*ID*⟩ **(** ⟨*INDEX-EXPR*⟩ **)** | ⟨*ID*⟩

⟨*INDEX-EXPR*⟩ ::= ⟨*INDEX-VAR*⟩
| ⟨*INDEX-VAR*⟩, ⟨*INDEX-EXPR*⟩

⟨*INDEX-VAR*⟩ ::= i | j | k | l

⟨*EXPR*⟩ ::= ⟨*EXPR*⟩ + ⟨*EXPR*⟩
| ⟨*EXPR*⟩ - ⟨*EXPR*⟩
| ⟨*EXPR*⟩ * ⟨*EXPR*⟩
| ⟨*EXPR*⟩ / ⟨*EXPR*⟩
| ⟨*CONSTANT*⟩
| ⟨*TENSOR*⟩

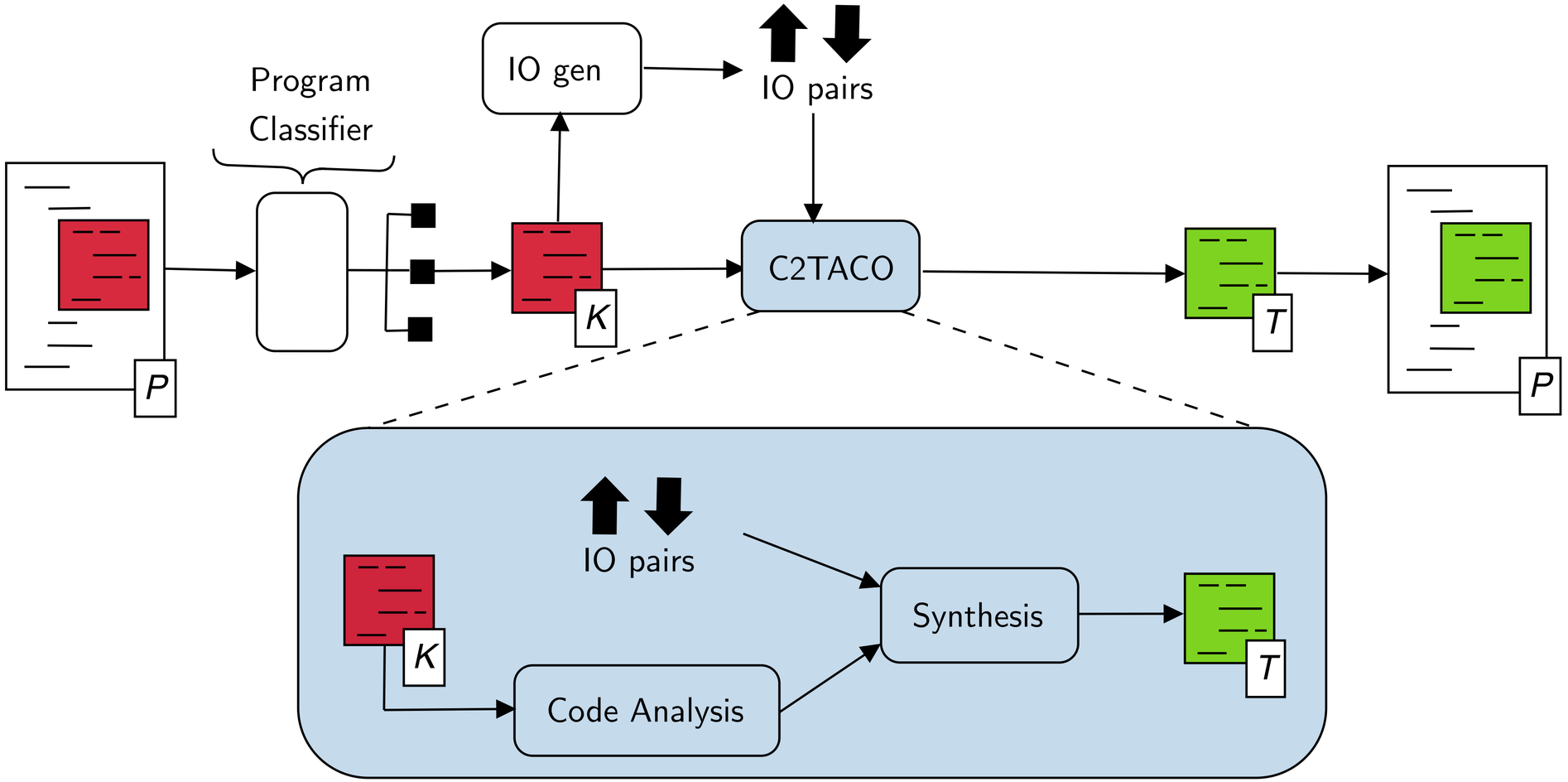
⟨*ID*⟩ ::= T_0 | T_1 | T_2 | ...

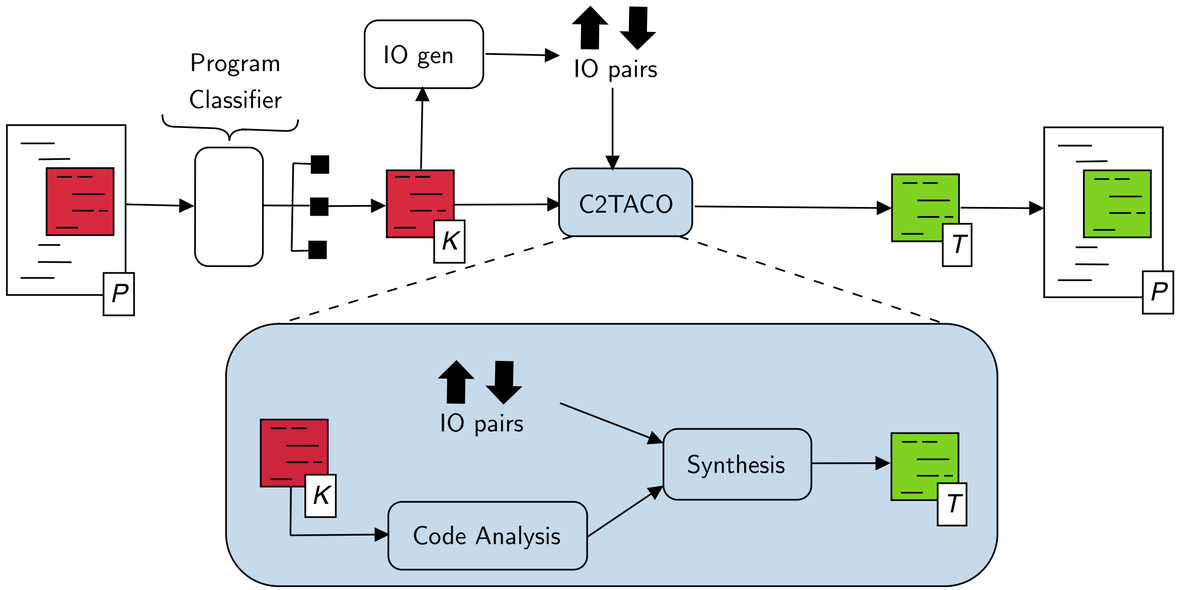
⟨*CONSTANT*⟩ ::= C_0 | C_1 | C_2 | ...

$$\exists P_T \forall x \,.\, P_T(x) = P_C(x)$$

Does there exist a function P_T, in TACO, such that, for all possible inputs x, $P_T(x)$ gives the same result as the original source program $P_C(x)$ in C.

C2TACO - Overview





Specification

- Generate Input/Output example

Bottom-up enumerative search of templates

- Progressively grow a candidate set by combining simpler to more complex ones
- Initialization: Basic programs (returning arguments, constants)
- Terminate when specification matched

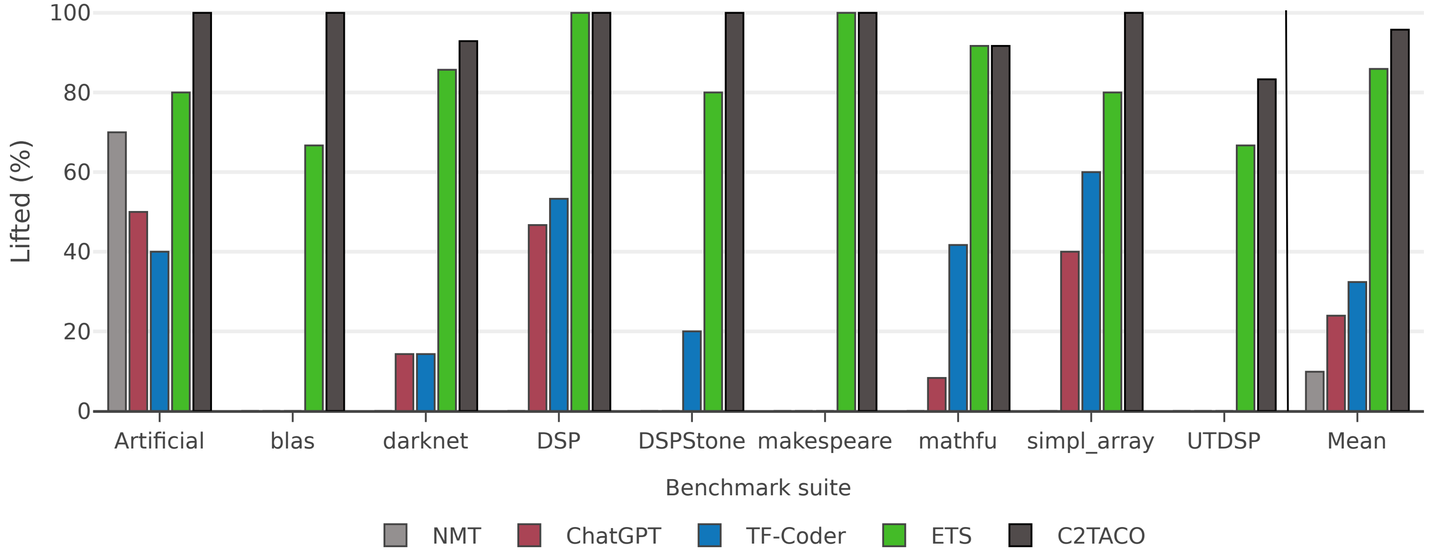
Optimization techniques

- Type correct by construction
- Identify classes of observationally equivalent candidates
- Heuristics based on analysis of source code

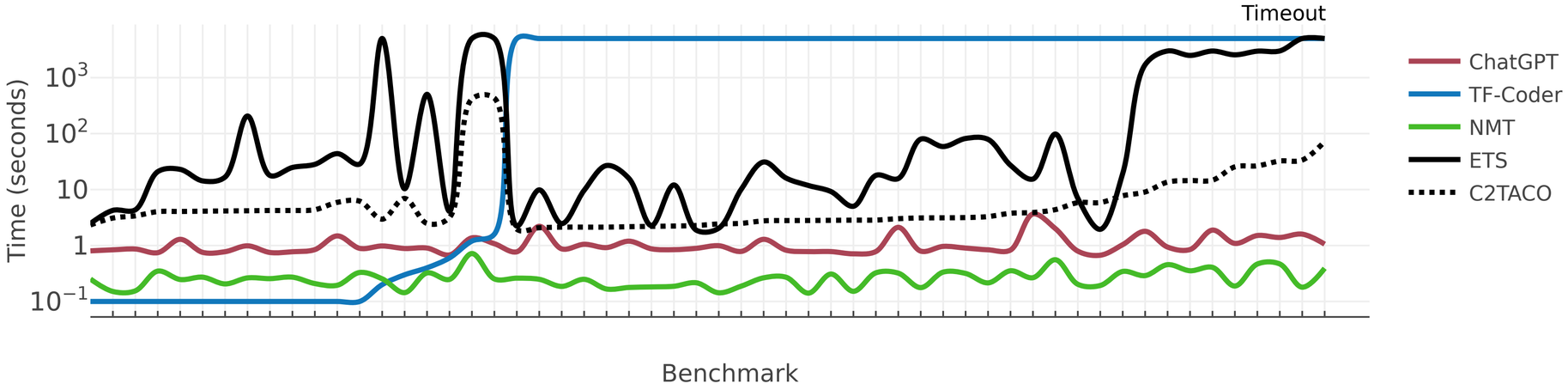
Oracle:

- Complete templates and compile and execute code

C2TACO - Performance

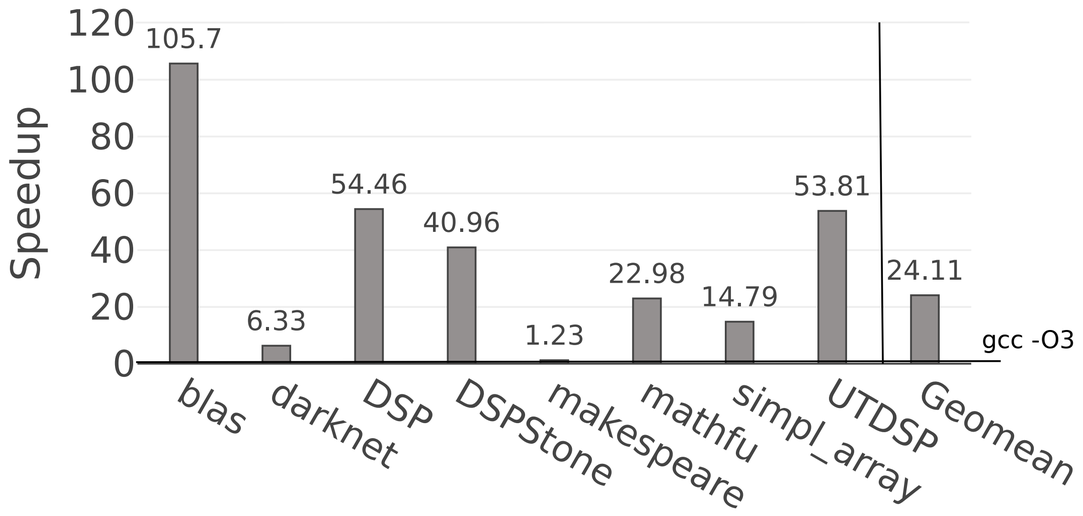


- Better than NMT and chatGPT
- 5.6s average synthesis time

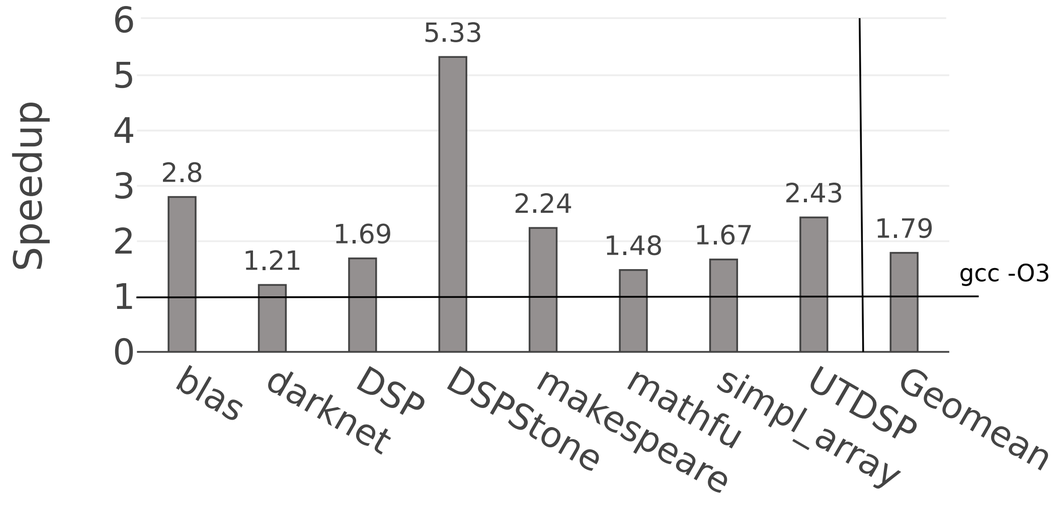


C2TACO - Performance

Average speedup 1.79x on a multi-core platform and 24.1x on a GPU



GPU

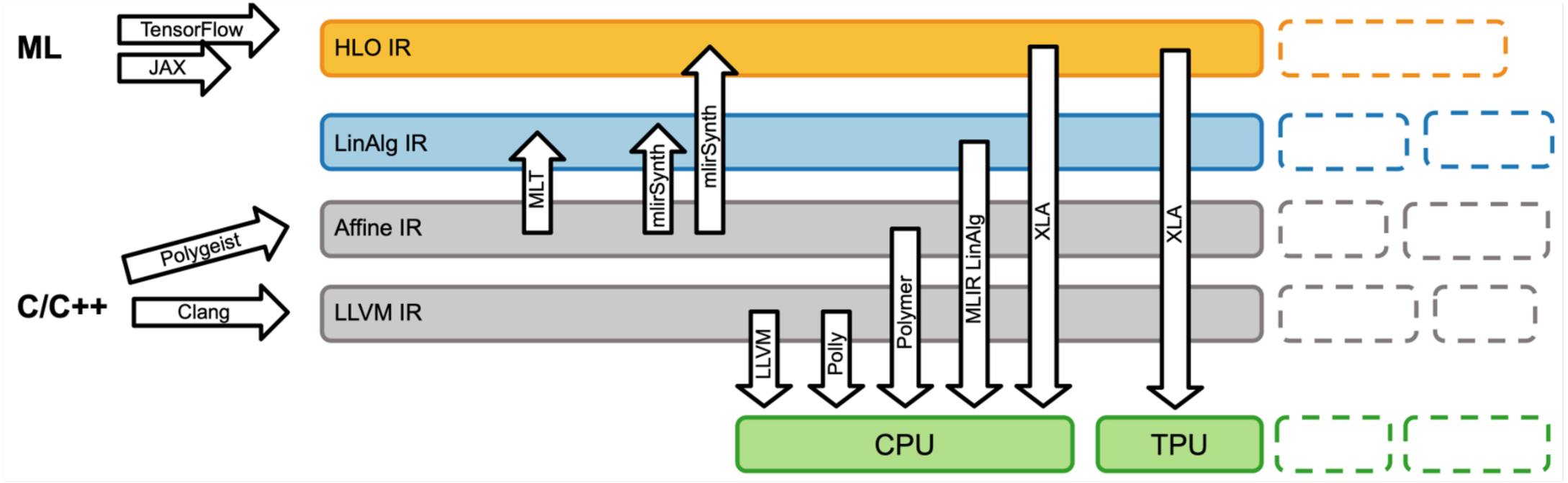


CPU

Speedup obtained by the synthesized TACO programs on different hardware platforms. The baseline is the average running time of the original implementations when compiled with `gcc -O3`

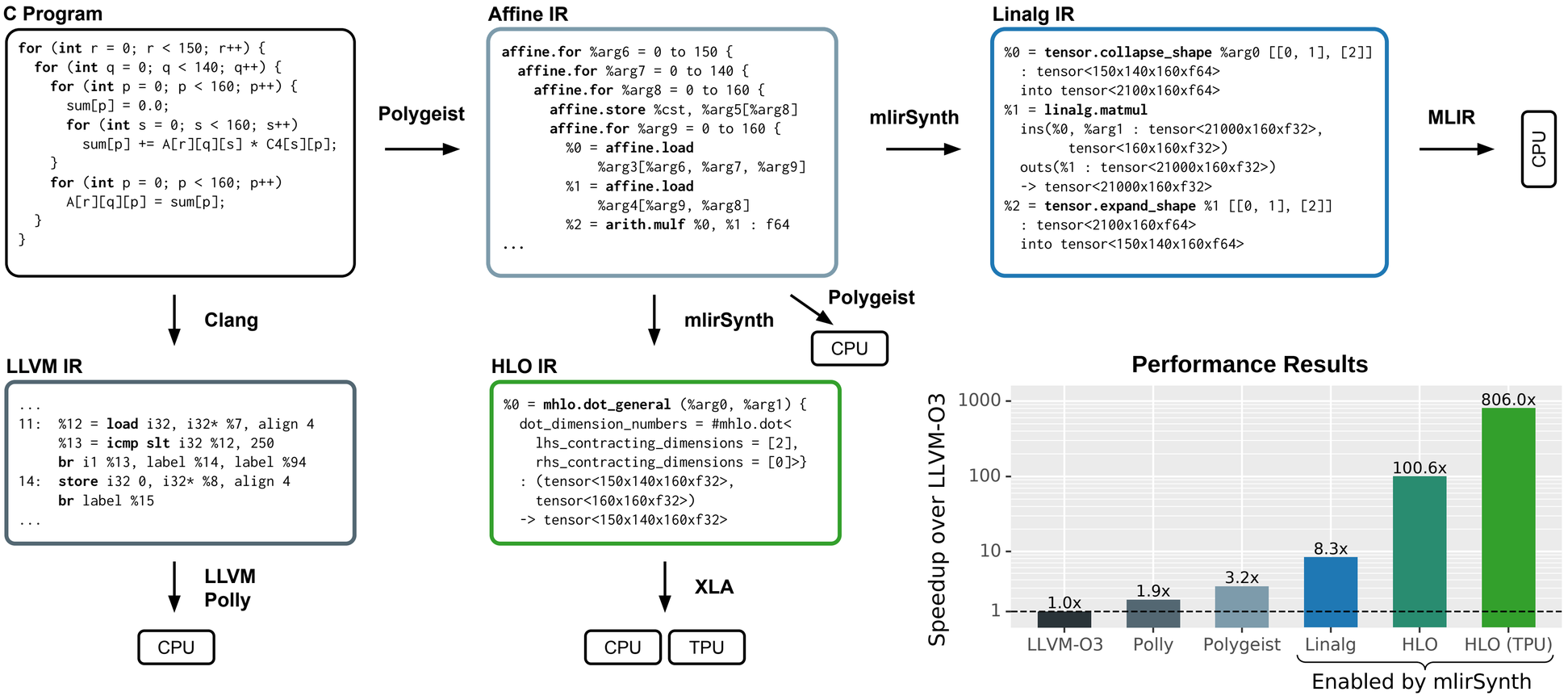
mlirSynth

- MLIR = extensible high-level representation within LLVM
- Vendors develop compilations paths for different MLIR dialects

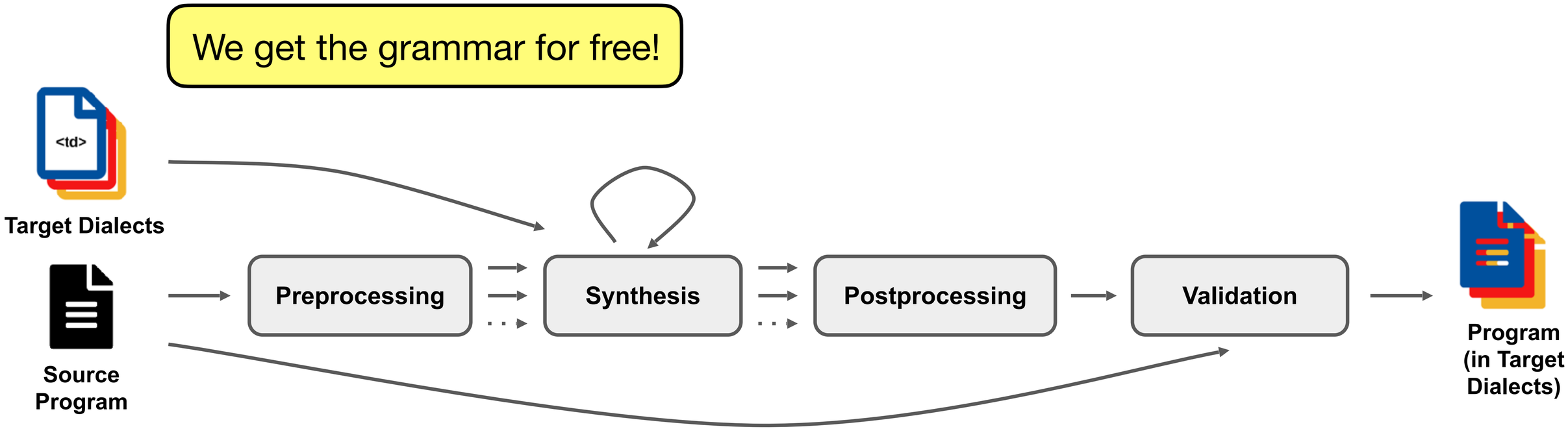


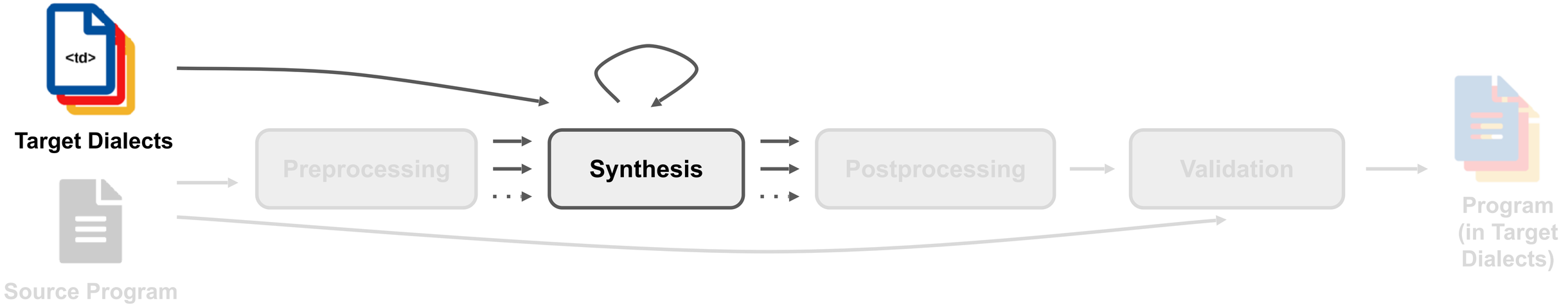
mlirSynth: Automatic, Retargetable Program Raising in Multi-Level IR using Program Synthesis - Alexander Brauckmann, Elizabeth Polgreen, Tobias Grosser, Michael O'Boyle

mlirSynth


C Program
for (int r = 0; r < 150; r++) {
for (int q = 0; q < 140; q++) {
for (int p = 0; p < 160; p++) {
sum[p] = 0.0;
for (int s = 0; s < 160; s++)
sum[p] += A[r][q][s] * C4[s][p];
}
for (int p = 0; p < 160; p++)
A[r][q][p] = sum[p];
}
}
Polygeist
Affine IR
affine.for %arg6 = 0 to 150 {
affine.for %arg7 = 0 to 140 {
affine.for %arg8 = 0 to 160 {
affine.store %cst, %arg5[%arg8]
affine.for %arg9 = 0 to 160 {
%0 = affine.load
%arg3[%arg6, %arg7, %arg9]
%1 = affine.load
%arg4[%arg9, %arg8]
%2 = arith.mulf %0, %1 : f64
...
mlirSynth
Linalg IR
%0 = tensor.collapse_shape %arg0 [[0, 1], [2]]
: tensor<150x140x160xf64>
into tensor<2100x160xf64>
%1 = linalg.matmul
ins(%0, %arg1 : tensor<21000x160xf32>,
tensor<160x160xf32>)
outs(%1 : tensor<21000x160xf32>)
-> tensor<21000x160xf32>
%2 = tensor.expand_shape %1 [[0, 1], [2]]
: tensor<2100x160xf64>
into tensor<150x140x160xf64>
MLIR
CPU
Clang
LLVM IR
...
11: %12 = load i32, i32* %7, align 4
%13 = icmp slt i32 %12, 250
br i1 %13, label %14, label %94
14: store i32 0, i32* %8, align 4
br label %15
...
LLVM
Polly
CPU
mlirSynth
Polygeist
CPU
HLO IR
%0 = mhlo.dot_general (%arg0, %arg1) {
dot_dimension_numbers = #mhlo.dot<
lhs_contracting_dimensions = [2],
rhs_contracting_dimensions = [0]>}
: (tensor<150x140x160xf32>,
tensor<160x160xf32>)
-> tensor<150x140x160xf32>
XLA
CPU
TPU
Performance Results
Speedup over LLVM-O3
1000
100
10
1
1.0x
1.9x
3.2x
8.3x
100.6x
806.0x
LLVM-O3
Polly
Polygeist
Linalg
HLO
HLO (TPU)
Enabled by mlirSynth

mlirSynth - Overview





Specification

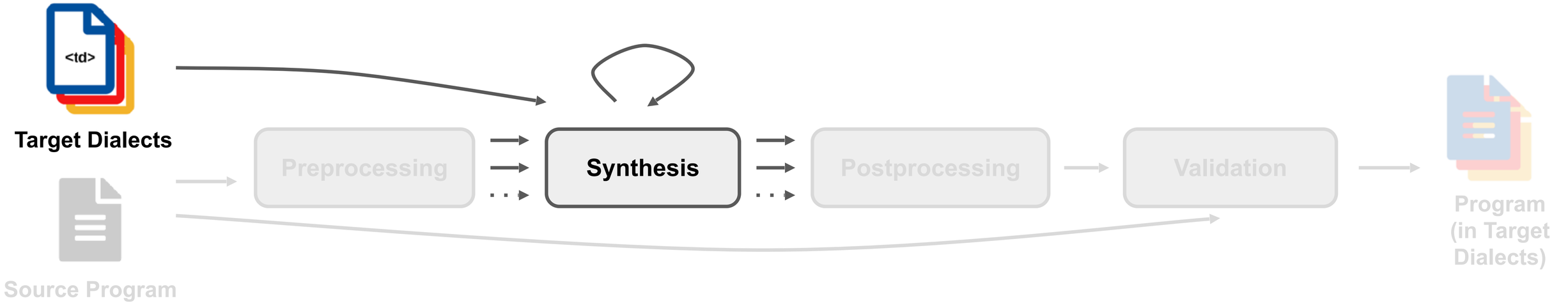
- Generate Input/Output example

Bottom-up enumerative search

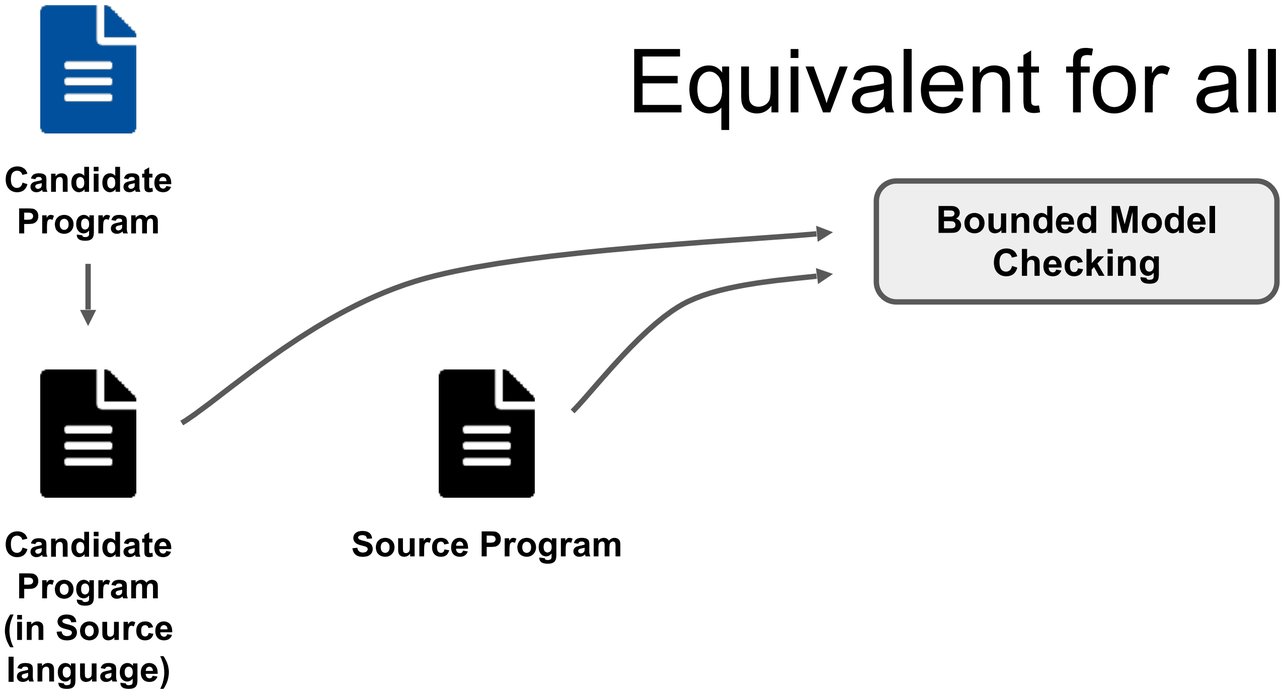
- Progressively grow a candidate set by combining simpler to more complex ones
- Initialization: Basic programs (returning arguments, constants)
- Terminate when specification matched

Optimization techniques

- Type correct by construction
- Identify classes of observationally equivalent candidates
- Polyhedral-based heuristics for guiding synthesis



Equivalent for all inputs?

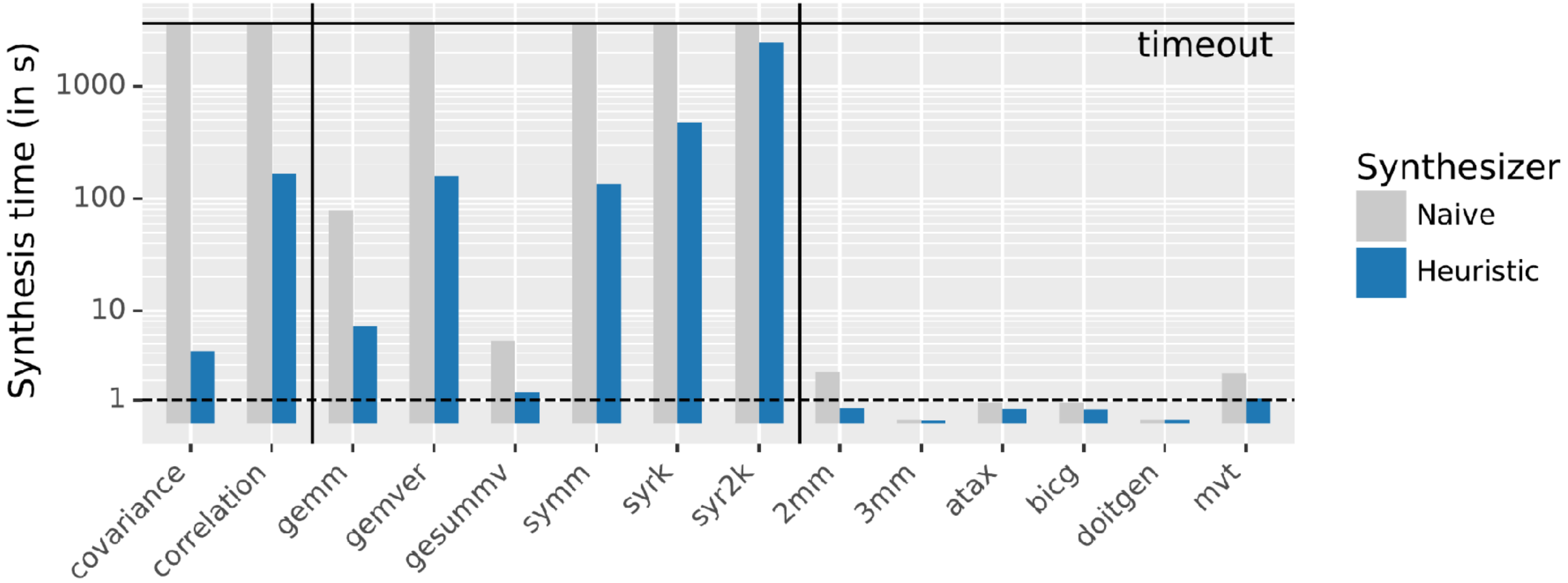


Equivalence Guarantees

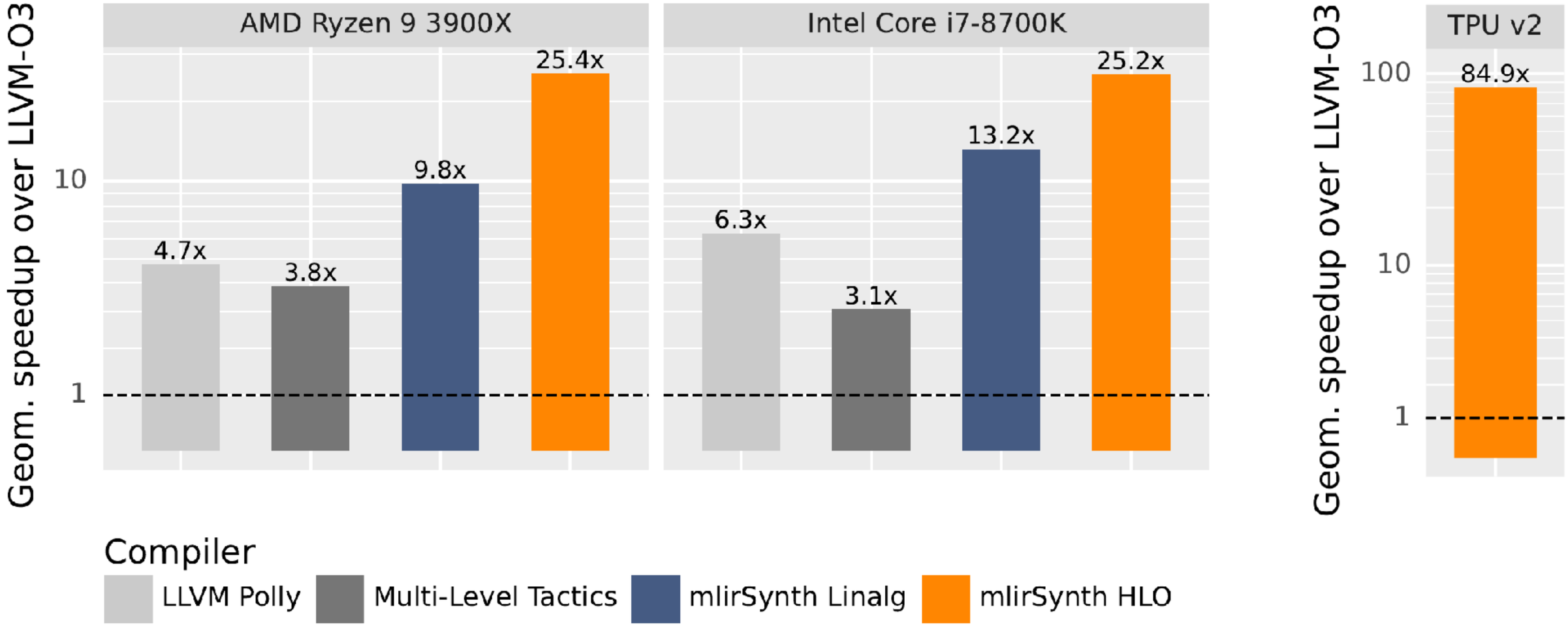
a) Float arithmetic
b) Float arithmetic, permitting small δ
c) Integer arithmetic

Testing I/O equivalence

mlirSynth - Performance



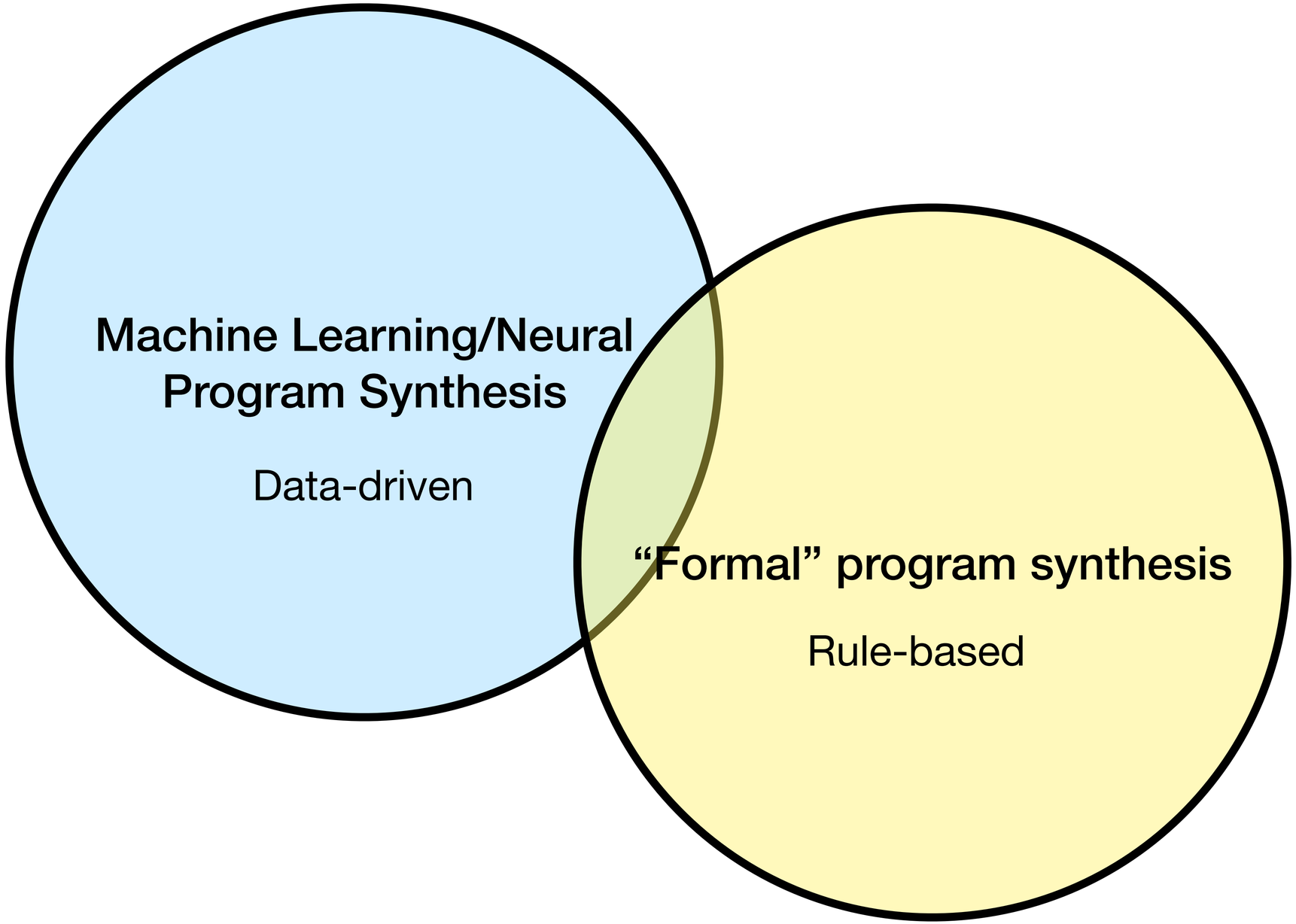
mlirSynth - Performance



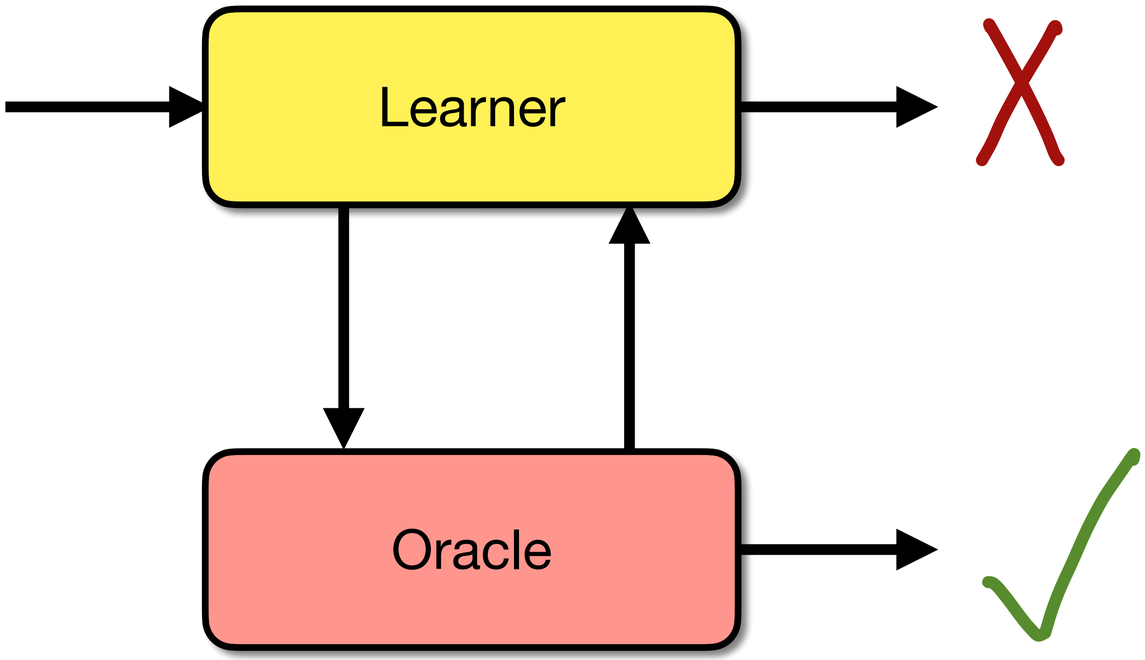
mlirSynth and C2TACO - Performance

- Simple synthesis gave us impressive speed-ups
- Handwritten heuristics were key
- Correctness is still a challenge for floating point
- Missed some benchmarks due to scalability

Making Synthesis Faster

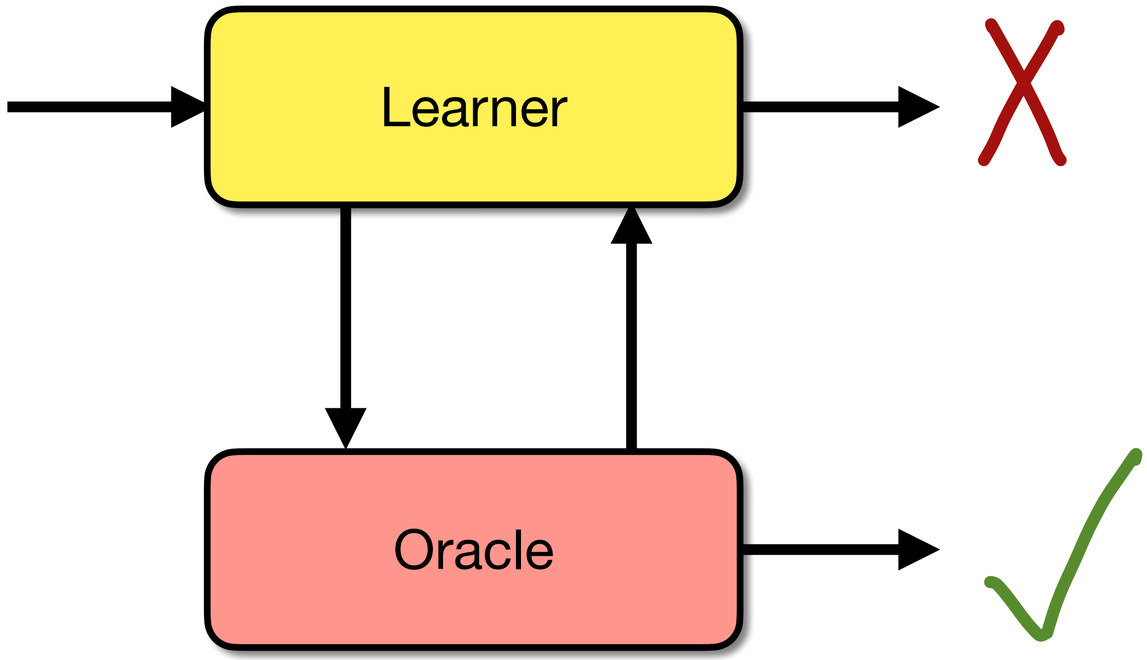

Machine Learning/Neural Program Synthesis
Data-driven
“Formal” program synthesis
Rule-based

New synthesis algorithms:



- New learners: clever heuristics, use machine learning
  - Type directed
  - Reinforcement learning
  - Probabilistic grammars
- New oracles: pass more information back to the learner
  - “Can this partial program ever satisfy the specification”
- New ways to break the problem into smaller problems

New synthesis algorithms:

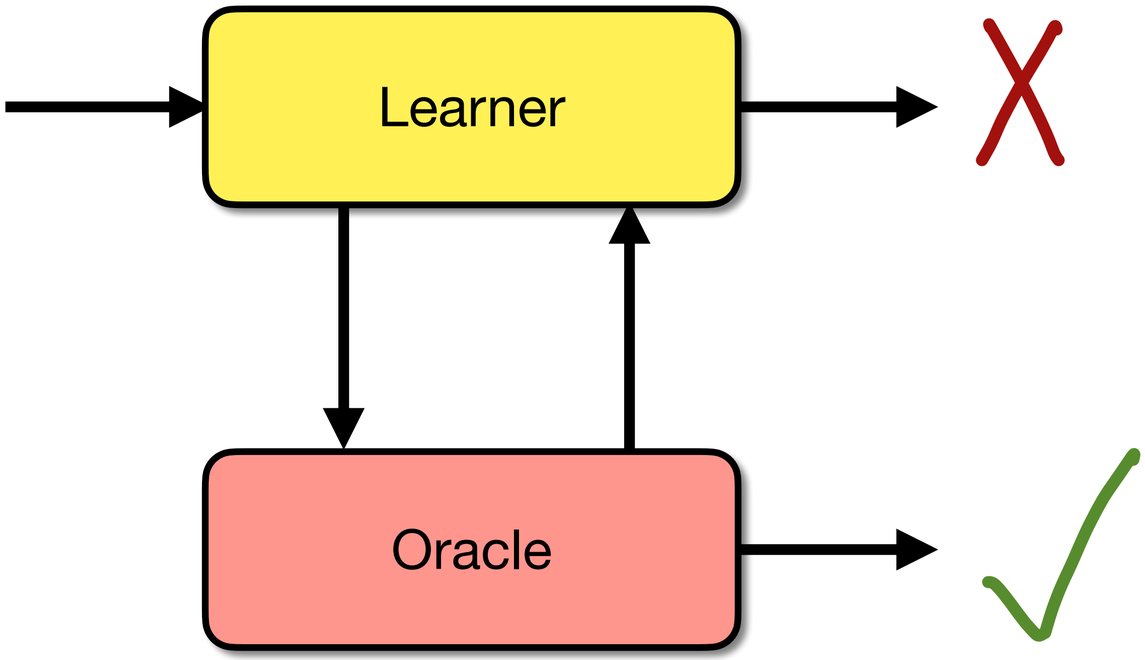


- New learners: clever heuristics, use machine learning
  - Type directed
  - Reinforcement learning
  - Probabilistic grammars (with LLMs)

Parsert & Polgreen, AAAI 2024

- New oracles: pass more information back to the learner
  - “Can this partial program ever satisfy the specification”
- New ways to break the problem into smaller problems

New synthesis algorithms:



- New learners: clever heuristics, use machine learning
  - Type directed
  - Reinforcement learning
  - Probabilistic grammars (with LLMs)

Parsert & Polgreen, AAAI 2024

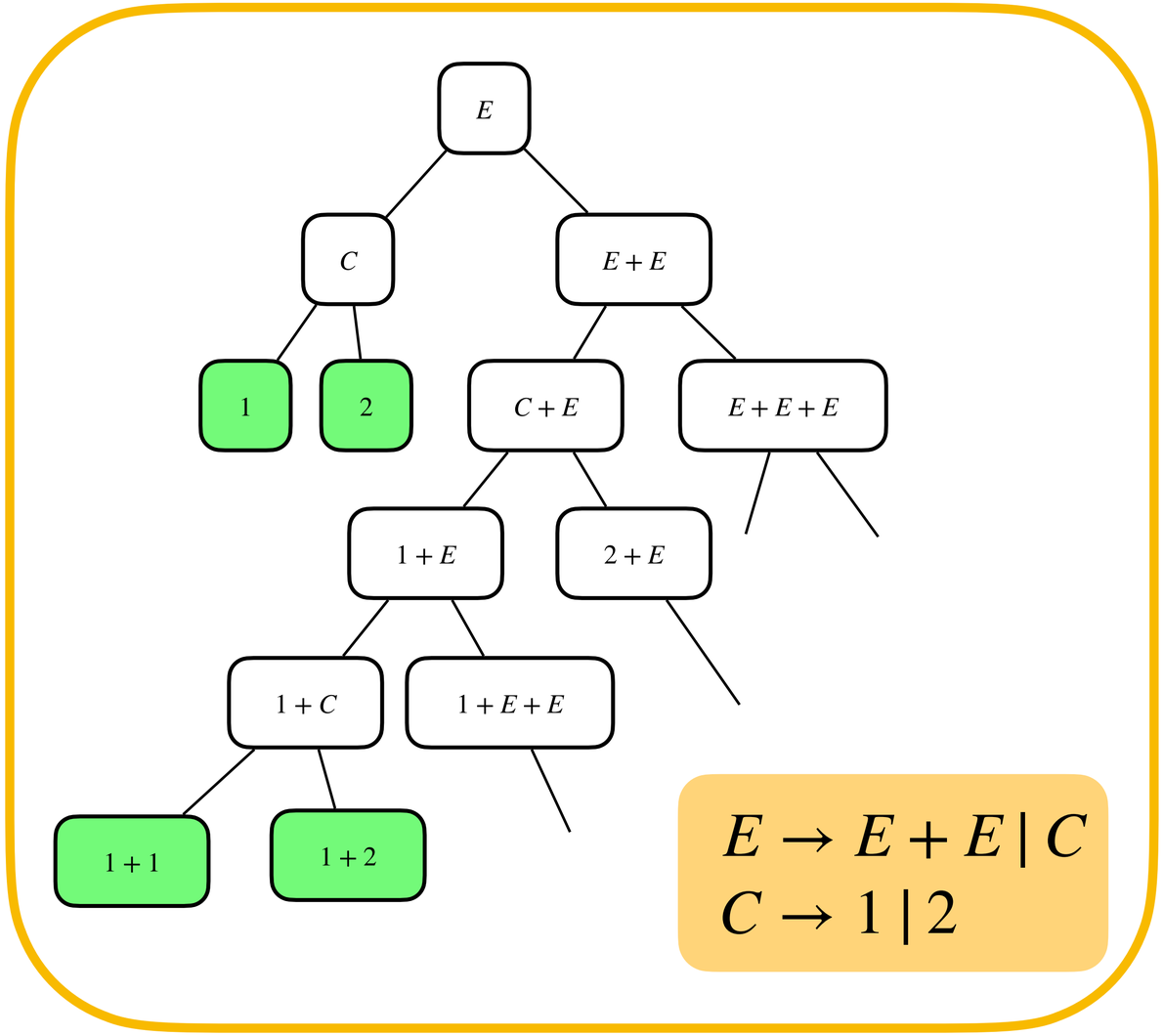
- New oracles: pass more information back to the learner
  - “Can this partial program ever satisfy the specification”
- New ways to break the problem into smaller problems

Guidance from LLMs

- Potentially unrealiable, not able to use counterexamples
- Insight: syntax-guidance can be easily handled in a probabilistic manner
- We can convert the output from an LLM into syntactic guidance

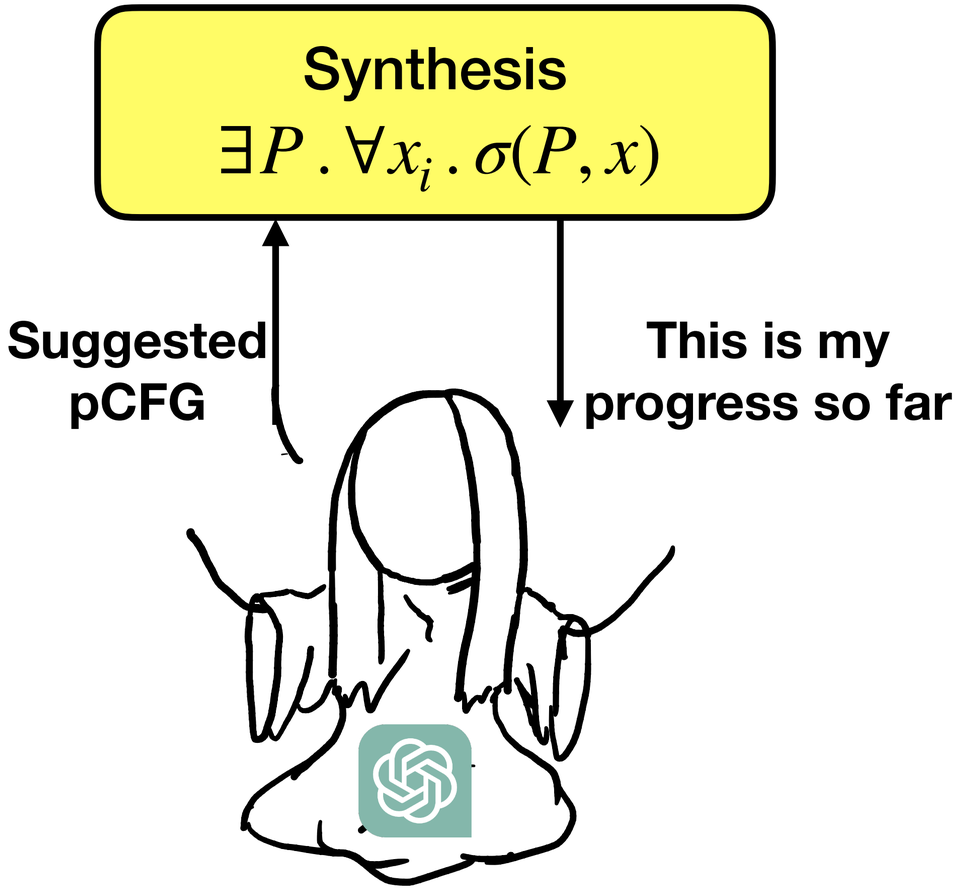
Work with Yixuan Li and Julian Parsert

Top down enumeration



- Initialize expression with the Start symbol
- Repeatedly choose production rules to replace any non-terminals in the
- Can be guided by a probability distribution over production rules (pCFG)
- Extension of this: weighted enumerative search using A*[1]

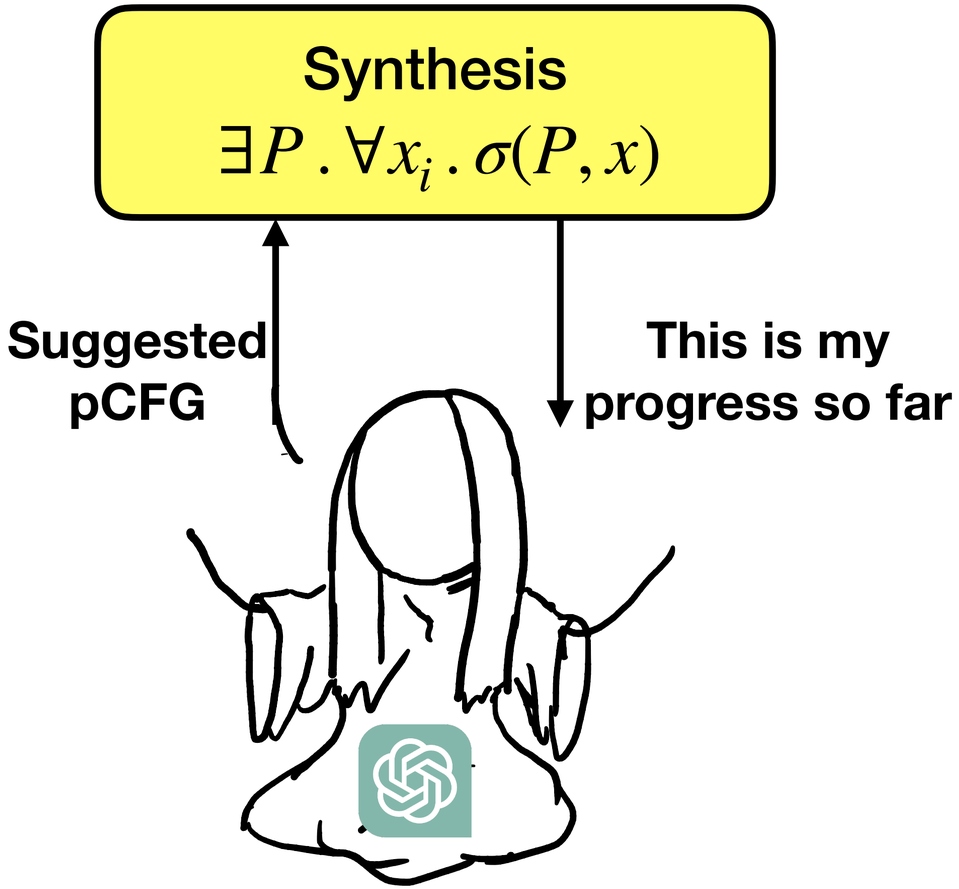
[1] Accelerating search-based program synthesis using learned probabilistic models - Lee et al



- Given a large language model, how do we update the probability distribution over the grammar?

```
You are teaching a student to write SMT-LIB. The student must write a
     function that satisfies the following constraints:
(constraint (>= (fn vr0 vr1 vr2) vr0))
(constraint (>= (fn vr0 vr1 vr2) vr1))
(constraint (>= (fn vr0 vr1 vr2) vr2))
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (=
     vr2 (fn vr0 vr1 vr2)))))
So far, the student has written this code:
(define-fun fn ((vr0 Int) (vr1 Int) (vr2 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.
```

Sure, here are some helper functions:



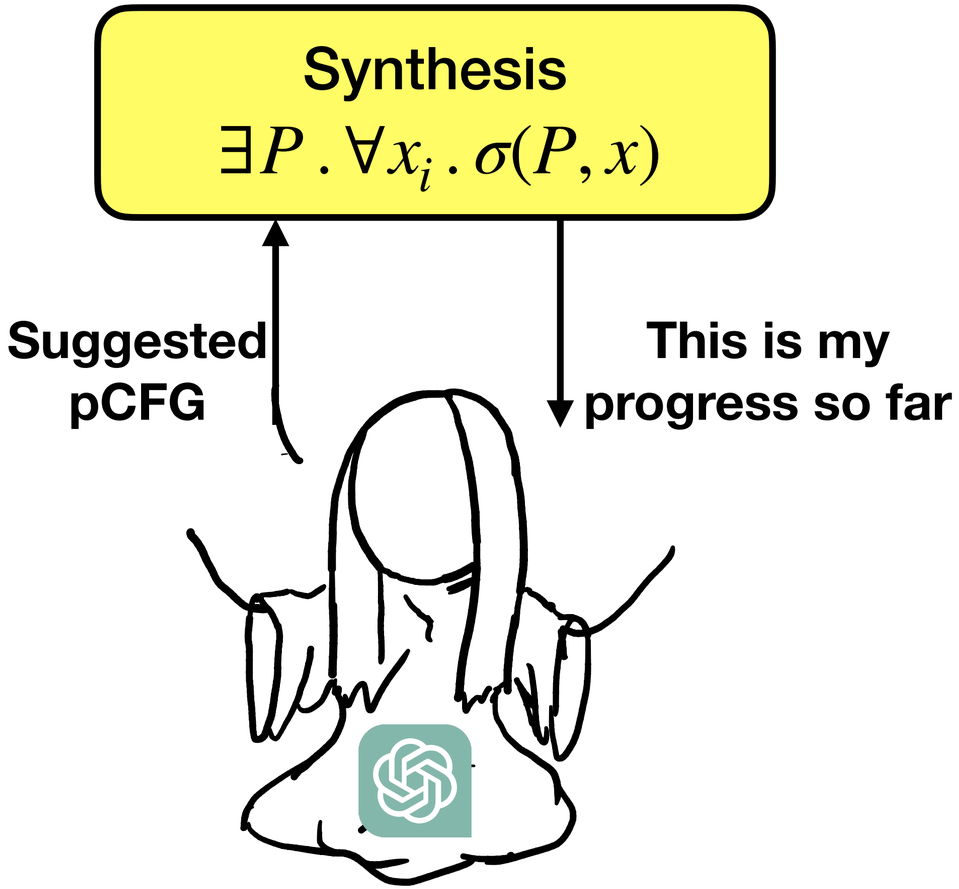
- Given a large language model, how do we update the probability distribution over the grammar?
- Prompt $\Longrightarrow$ Parse responses $\Longrightarrow$ Update weights on production rules.

```
S → 0 | 1 | 2
S → y | x
S → B ? S:S
B → S = S
B → S ≥ S
B → S ≤ S
B → !B
B → B ∨ B
B → B ∧ B
```

```
 y   x

(x ≥ 0)? x : y

(x ≥ y)? x : 0

 (y ≤ x)? x : y

 (y ≥ x)? x : y
```

Parser

```
S → 0 (2)| 1 | 2
S → y (6)| x (9)
S → B ? S:S (4)
B → S = S
B → S ≥ S (3)
B → S ≤ S (1)
B → !B
B → B ∨ B
B → B ∧ B
```

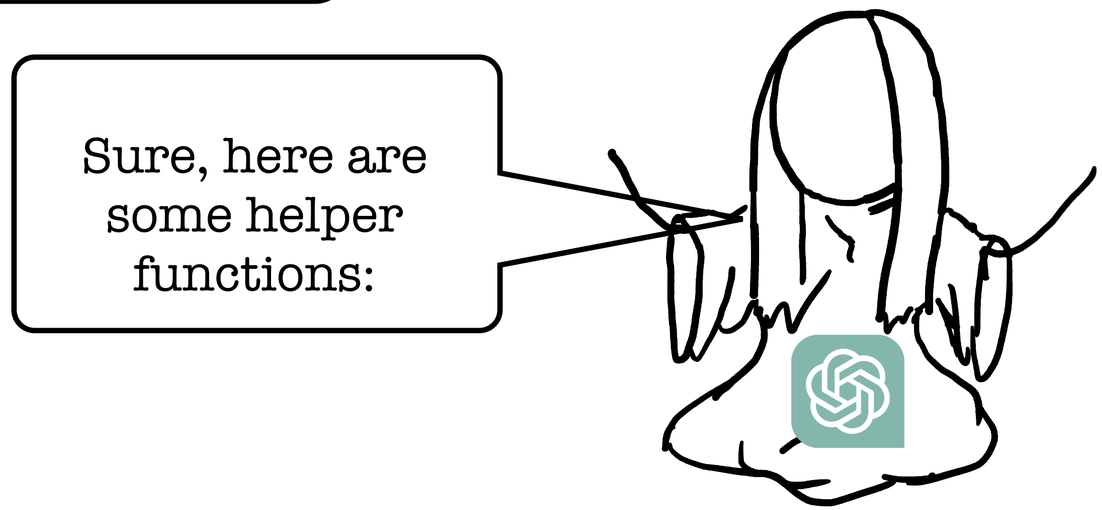


- Dynamically updating the probability distribution over grammar rules allows the oracle to make mistakes
- Prompting the LLM *during* the synthesis phase allows us to prompt with more information, which results in better performance from the LLM

The right prompts?

```
You are teaching a student to write SMT-LIB. The student must write a
     function that satisfies the following constraints:
(constraint (>= (fn vr0 vr1 vr2) vr0))
(constraint (>= (fn vr0 vr1 vr2) vr1))
(constraint (>= (fn vr0 vr1 vr2) vr2))
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (=
     vr2 (fn vr0 vr1 vr2)))))
So far, the student has written this code:
(define-fun fn ((vr0 Int) (vr1 Int) (vr2 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.
```

- Chain of thought
- Asking in a language that is common in the dataset (python, lisp) and then asking it to translate
- Emotional stimulii
- …



LLM-guided search evaluation

Problem	total # 	cvc5	Just the LLM # 	A* synth # 	A* + LLM guidance #
BV	384	292	137	253.0	**305.0**
LIA	87	43	54	34.0	**65.0**
Inv	138	80	112	22.0	**118.0**
Total	609	415	303	309.0	**488.0**

Conclusions

- Even simple synthesis can give us big performance speed-ups
  - But we have to handwrite heuristics
- Machine learning can generate heuristics for SyGuS
  - Now we need to automatically generate heuristics for “real” problems
- Can we apply the same techniques to generating more secure code? Updating old code?