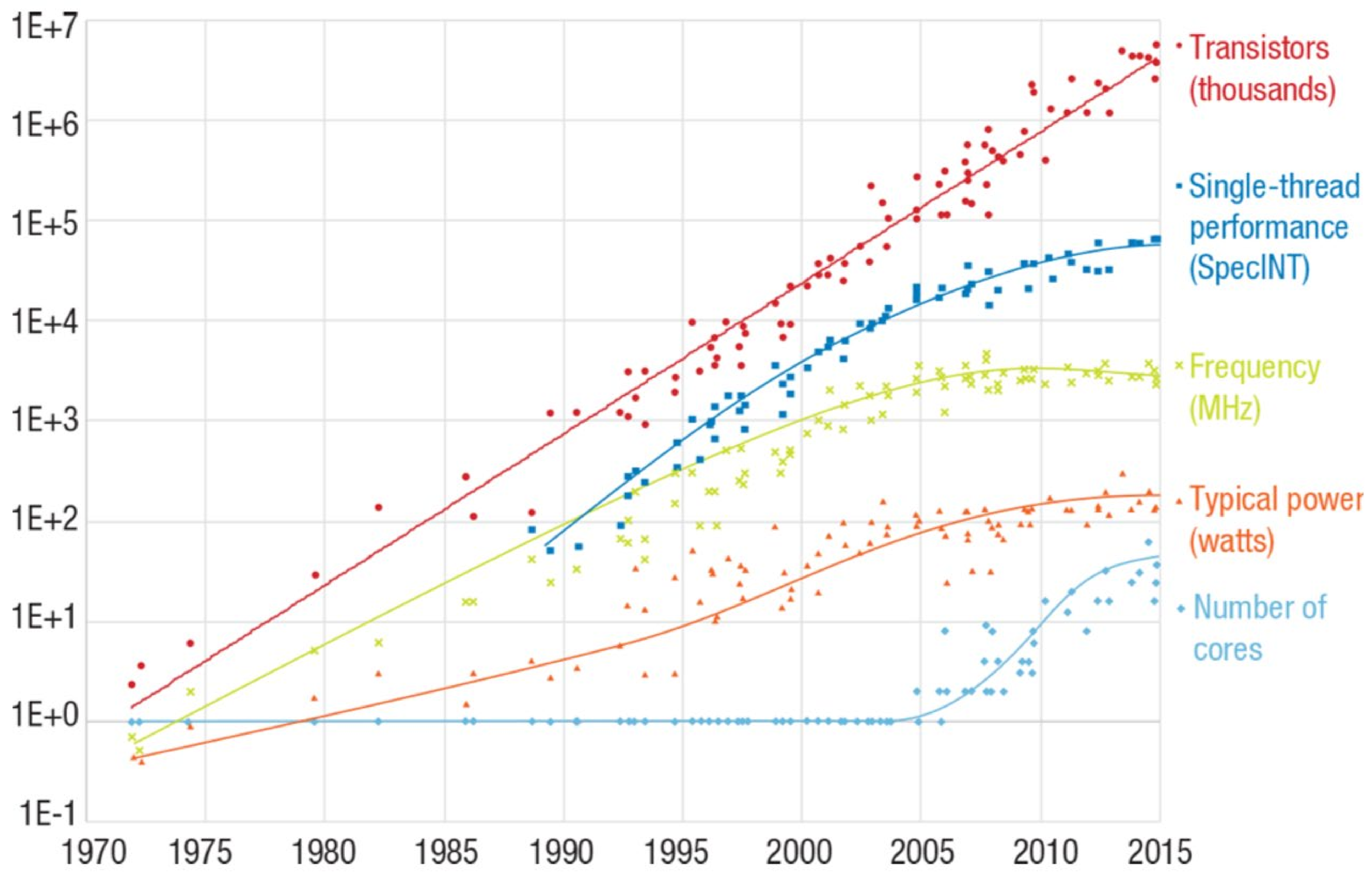
Using Program Synthesis to Make Your Code Run Faster

Dr Elizabeth Polgreen

Lecturer, University of Edinburgh
Royal Academy of Engineering Research Fellow



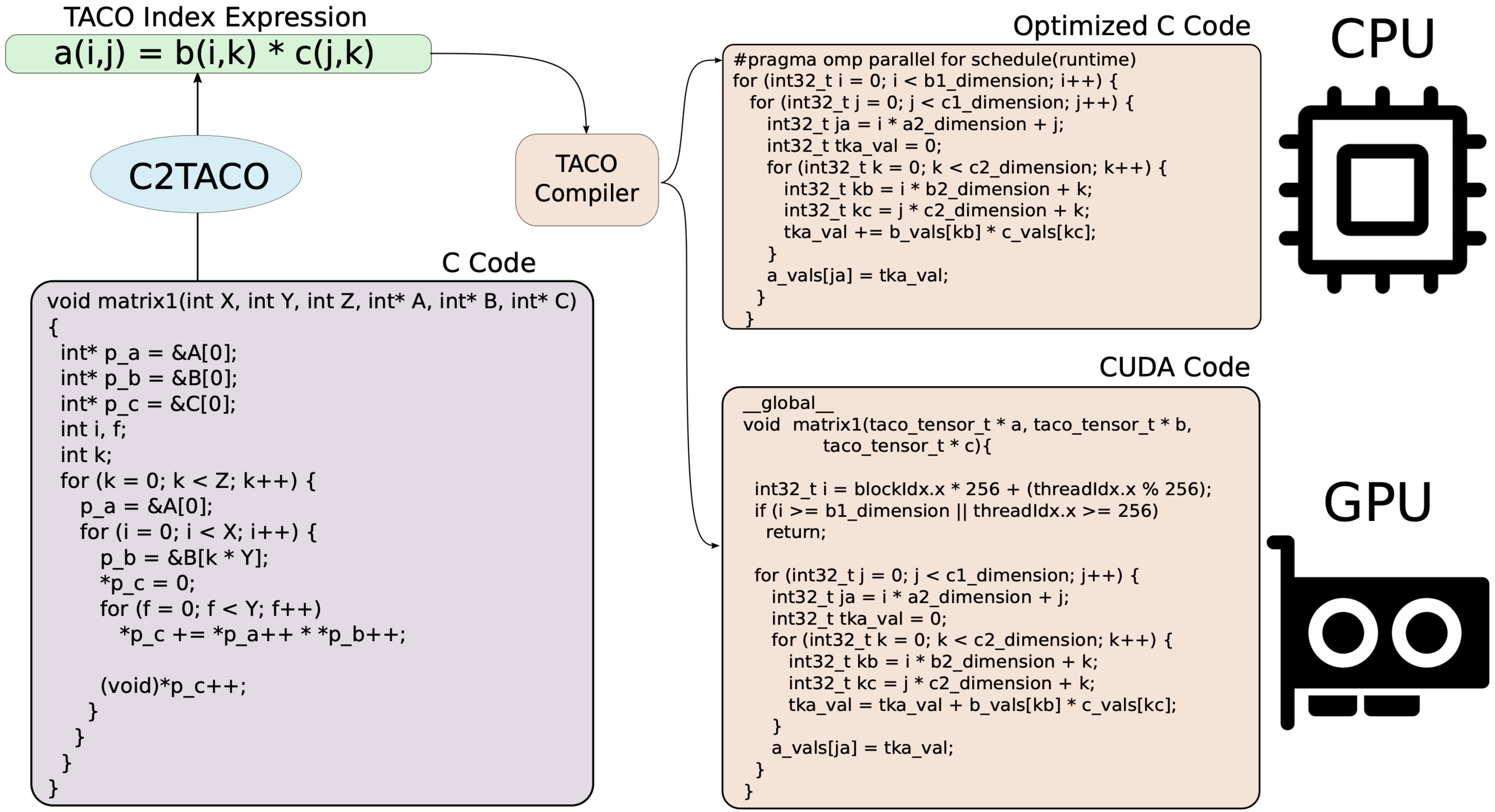
Moore's Law is dead?

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

- 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?

TACO: The Tensor Algebra Compiler

C2TACO



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

Existing approaches:

- ~~API matching/rewriting~~ Brittle!
- ~~Neural machine translation~~ Needs too much data!

How? Program synthesis!

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: $@ret \geq x \land @ret \geq y \land (@ret = x \land @ret = y)$

Formal Program Synthesis

```
int f(int x, int y)
{
  ???
}
```

@ensures: $@ret \geq x \wedge @ret \geq y \wedge (@ret = x \wedge @ret = y)$

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

Solution: f finds the max of x and y

Defining the search space

Syntax-Guided Synthesis

```
int f(int x, int y)
{
   ???
}
```

@ensures: $@ret \geq x \wedge @ret \geq y \wedge (@ret = x \wedge @ret = y)$

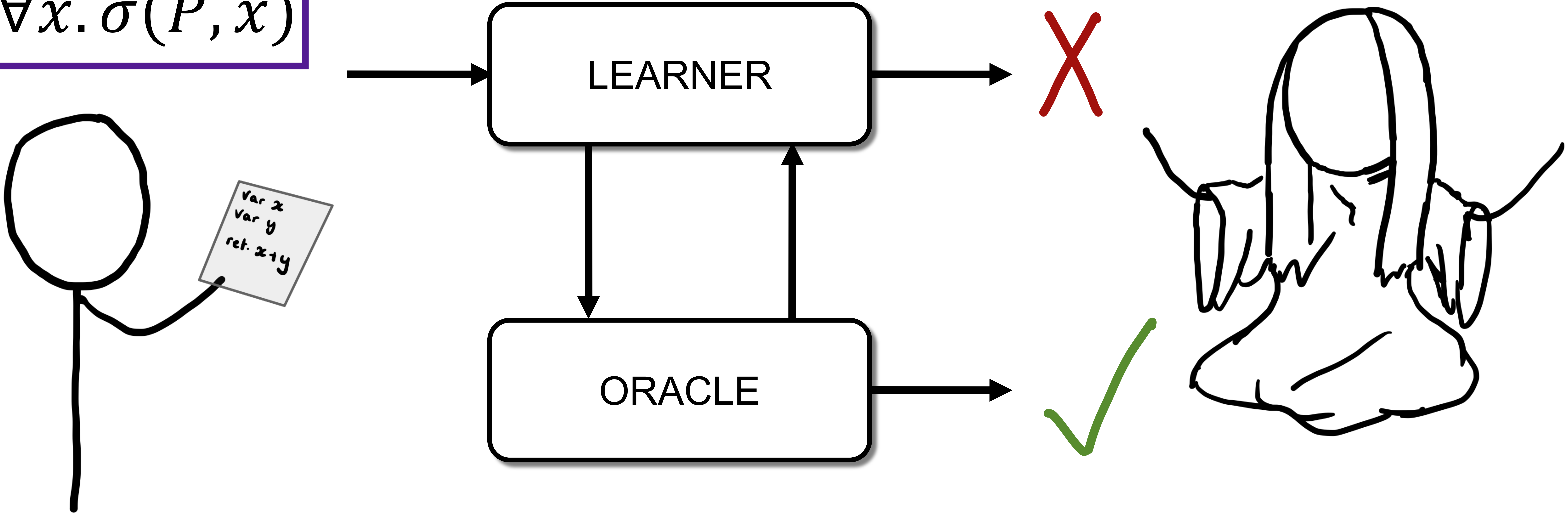
```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

Context Free Grammar

Algorithms for formal synthesis

Oracle Guided Inductive Synthesis

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



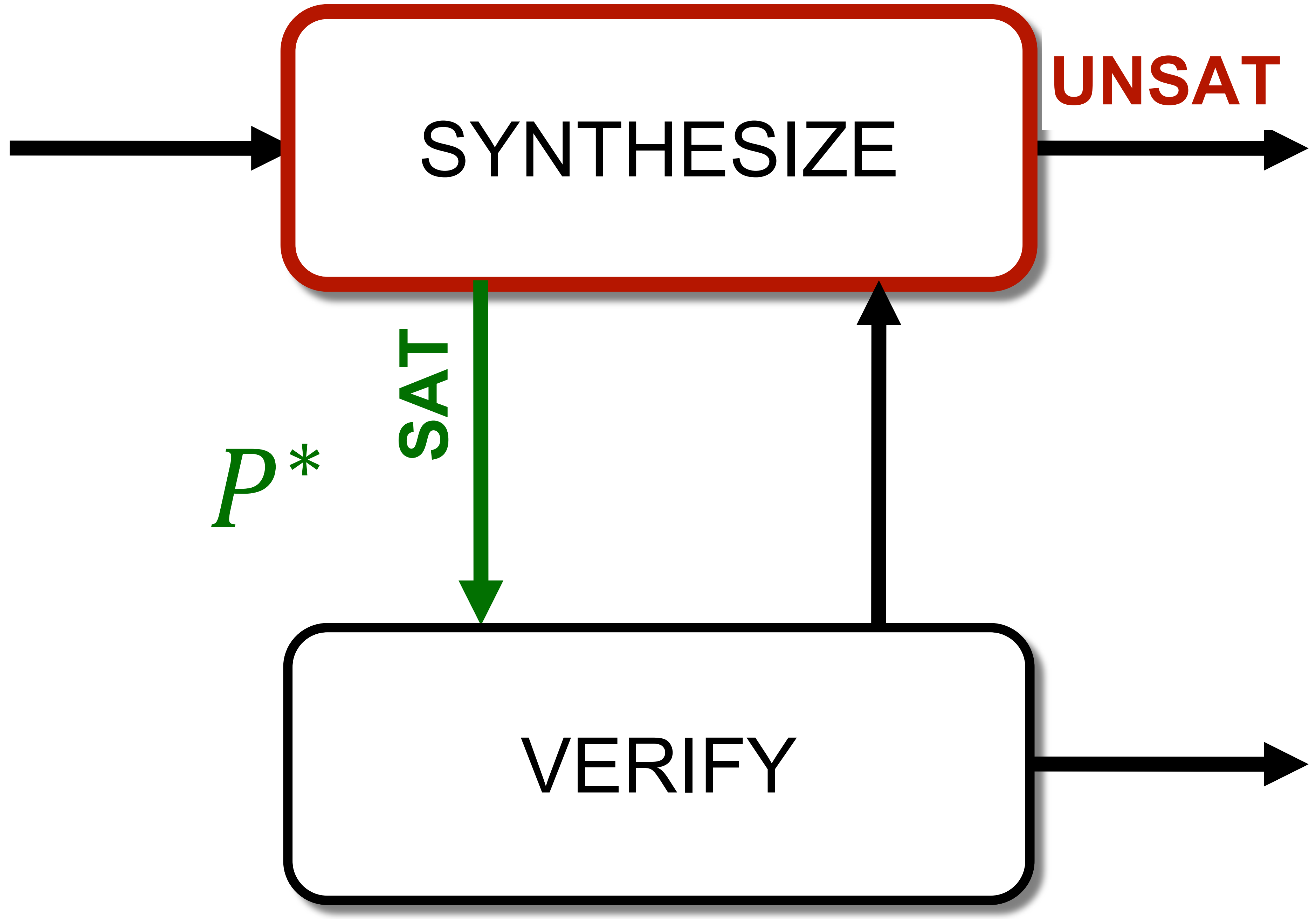
Searches program space
and guesses candidates

Says if the candidate is
correct, and guides the
search if not

Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

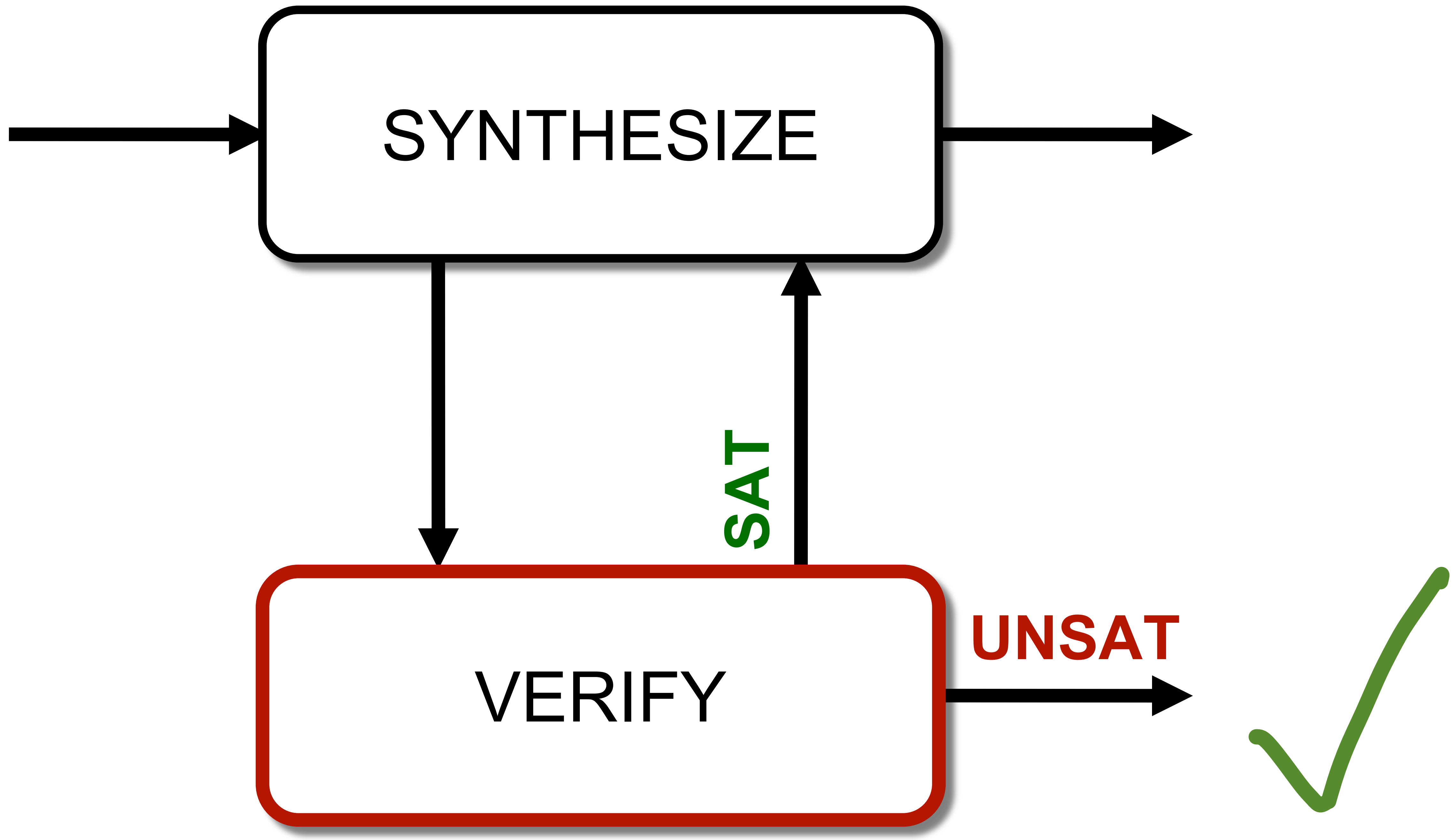
$\exists P \forall x . \sigma(P, x)$



Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

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

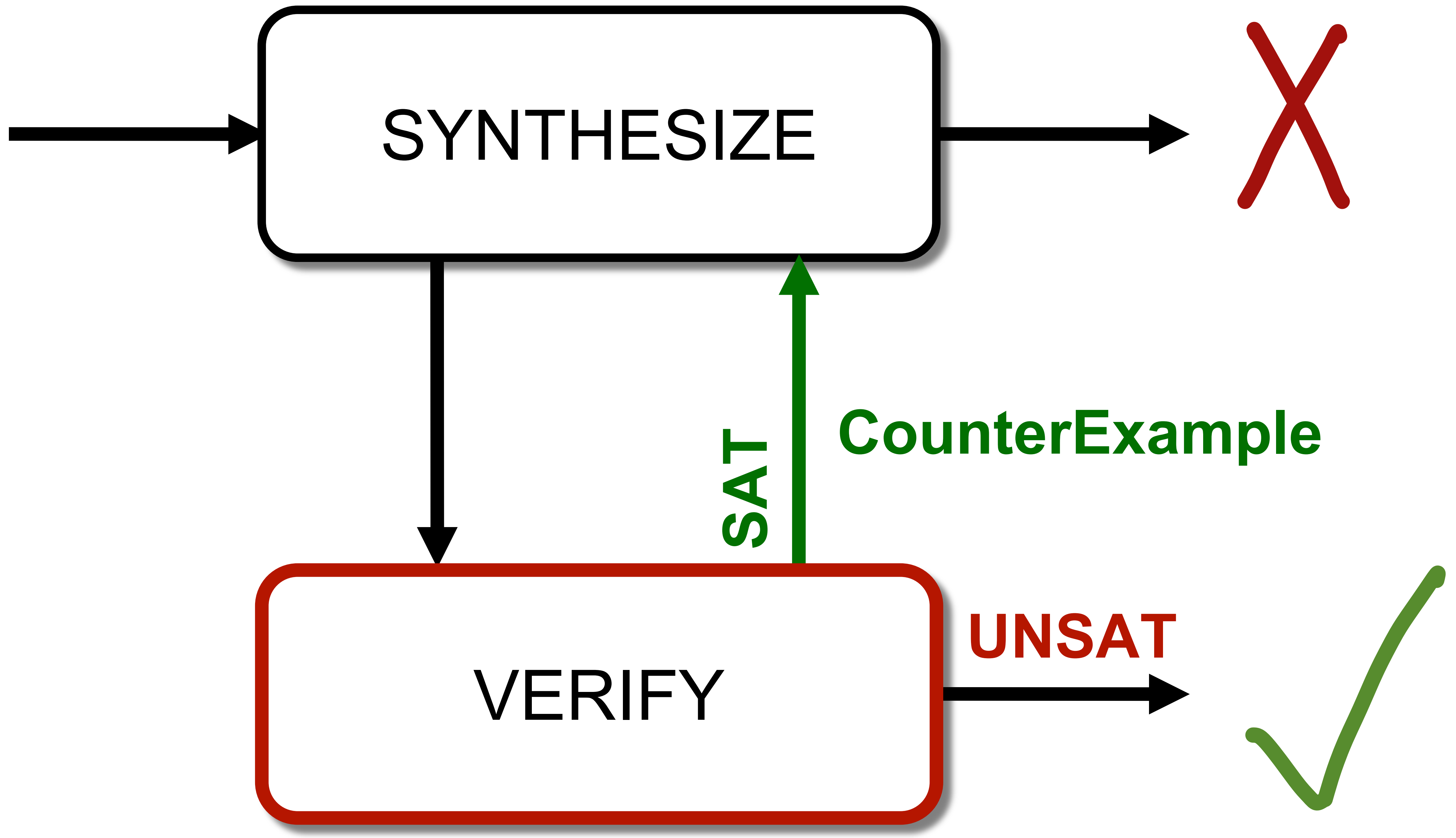


$$\exists x. \neg\sigma(P*, x)$$

Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

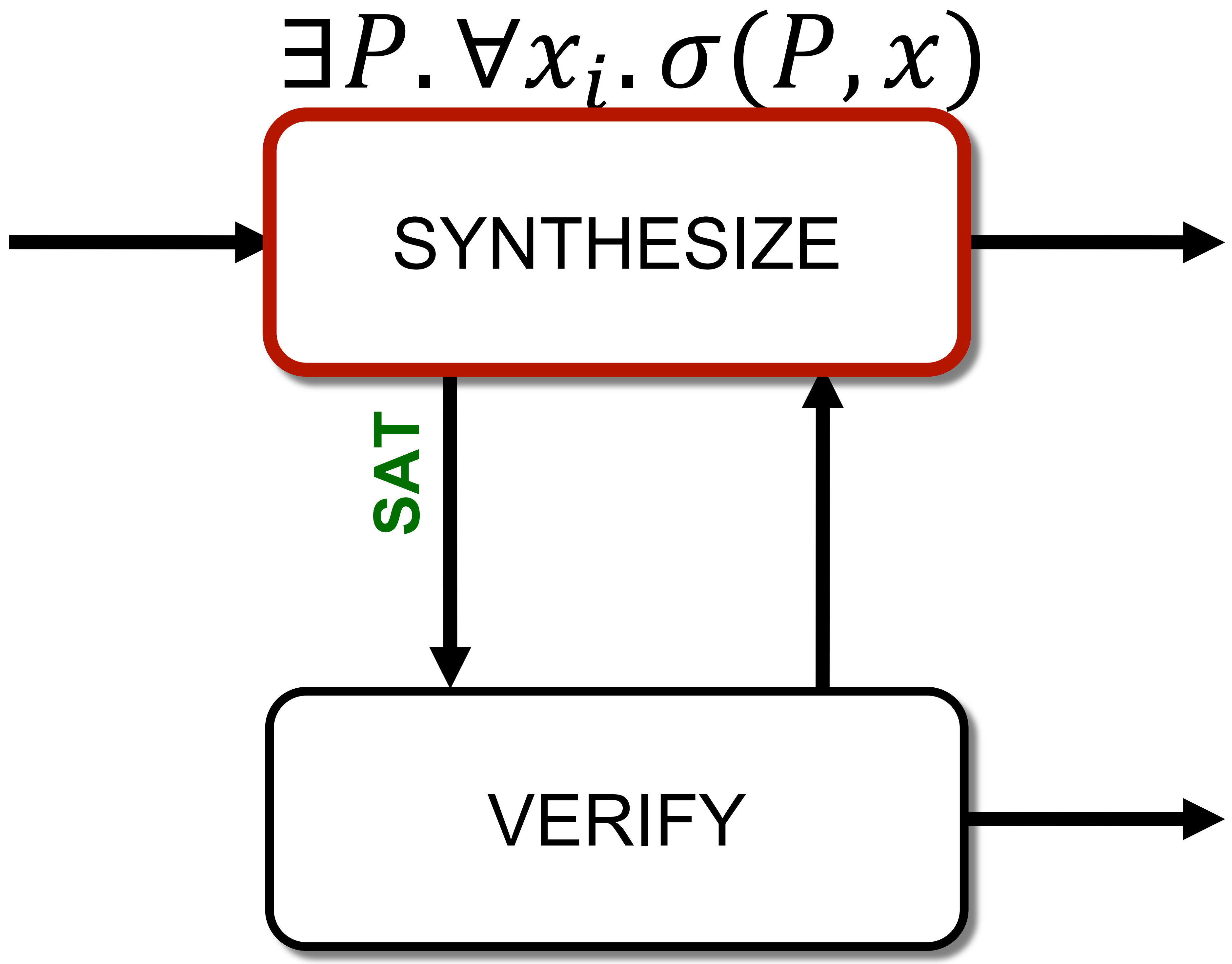
$$\exists P \forall x . \sigma(P, x)$$



Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

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

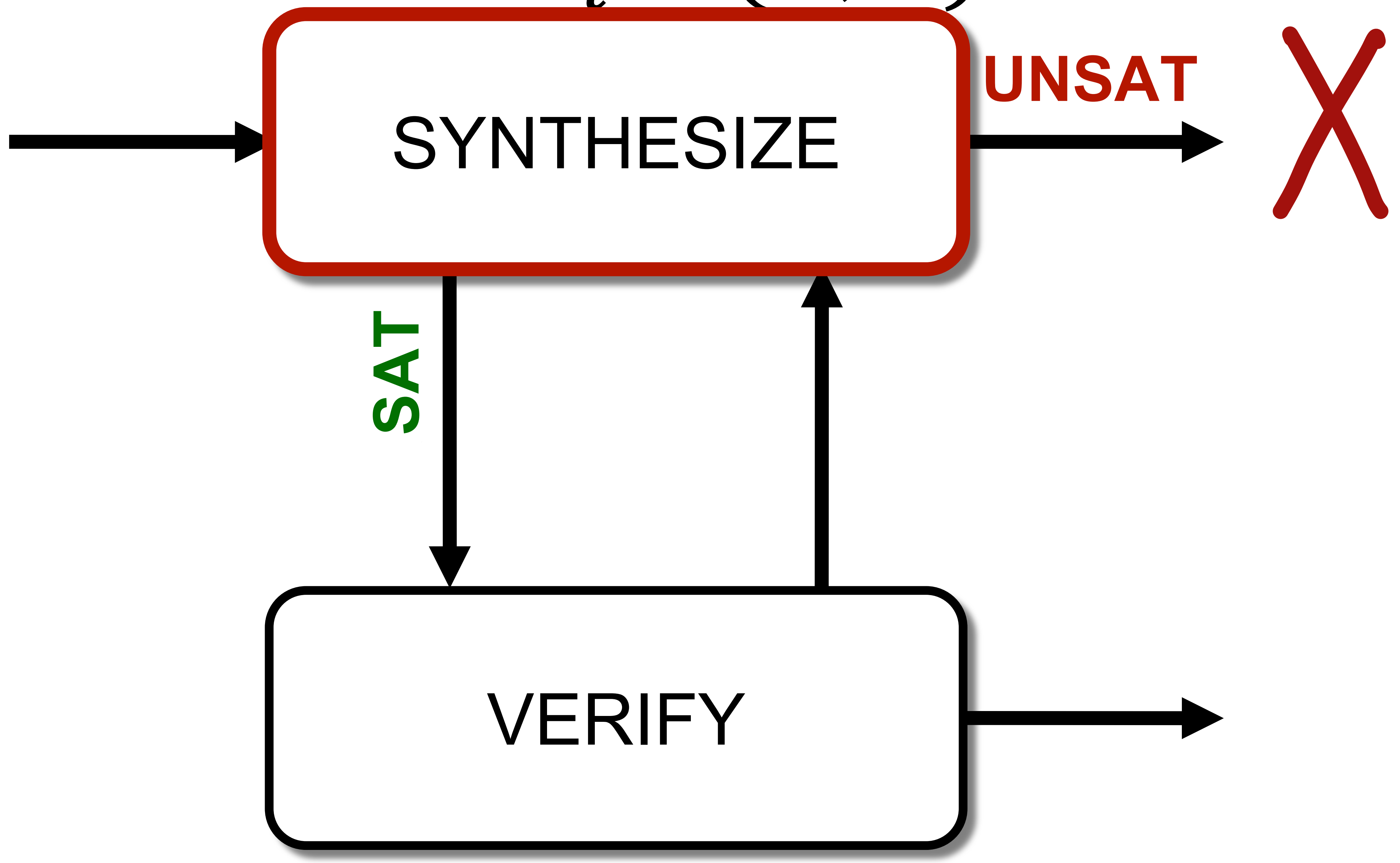


Algorithms for formal synthesis

Counterexample Guided Inductive Synthesis

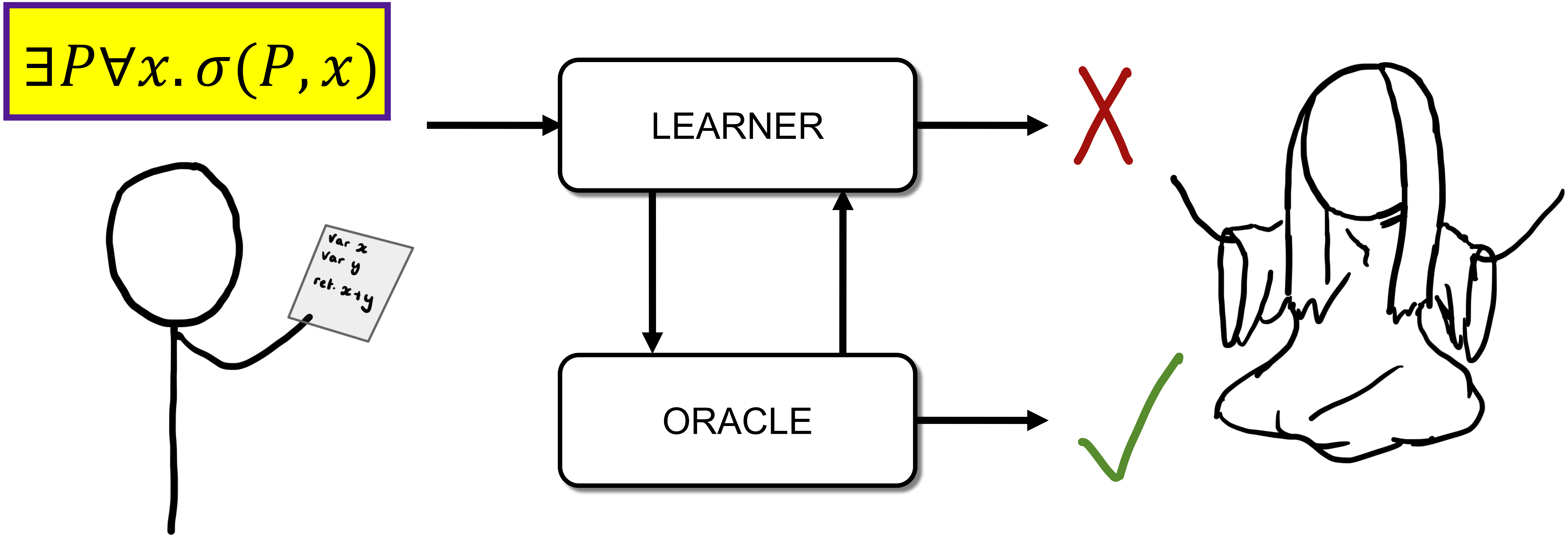
$$\exists P \forall x . \sigma(P, x)$$

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



Algorithms for formal synthesis

Oracle Guided Inductive Synthesis



Searches program space and guesses candidates

Says if the candidate is correct, and guides the search if not

C2TACO - Specification

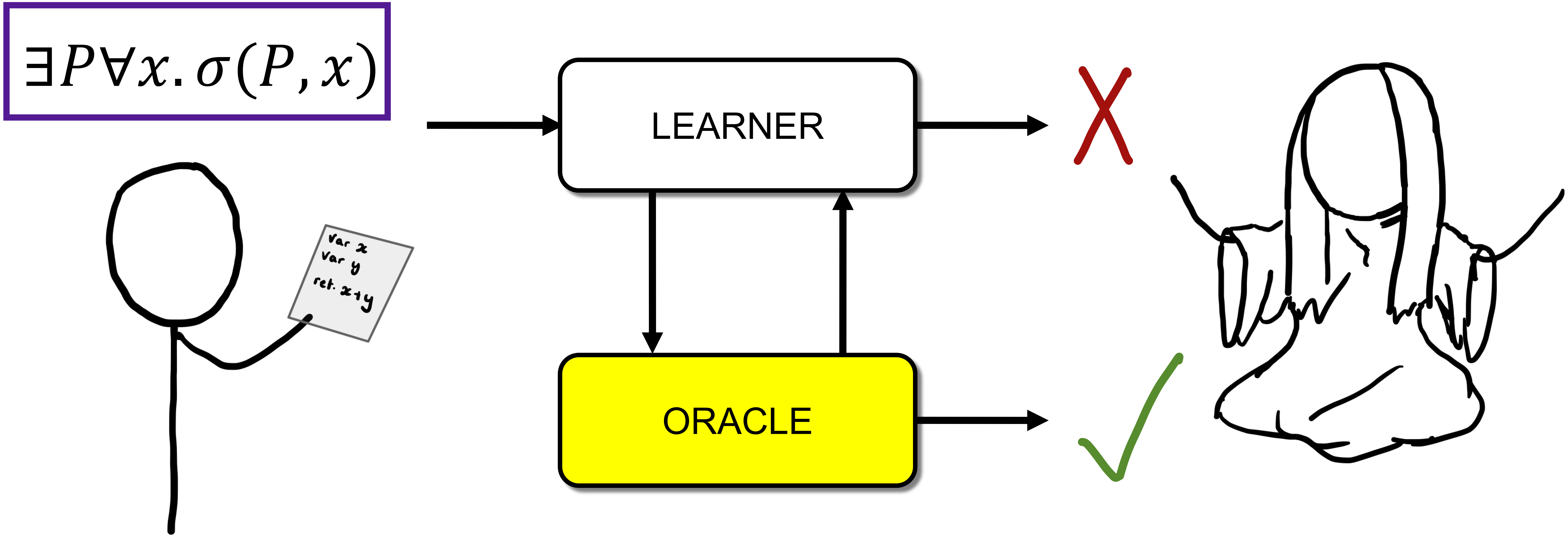
$$\exists P_T \forall x. P_T(x) = P_C(x)$$

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.

Specification: randomly generated input-output examples

Algorithms for formal synthesis

Oracle Guided Inductive Synthesis



Searches program space and guesses candidates

Says if the candidate is correct, and guides the search if not

C2TACO - Specification

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

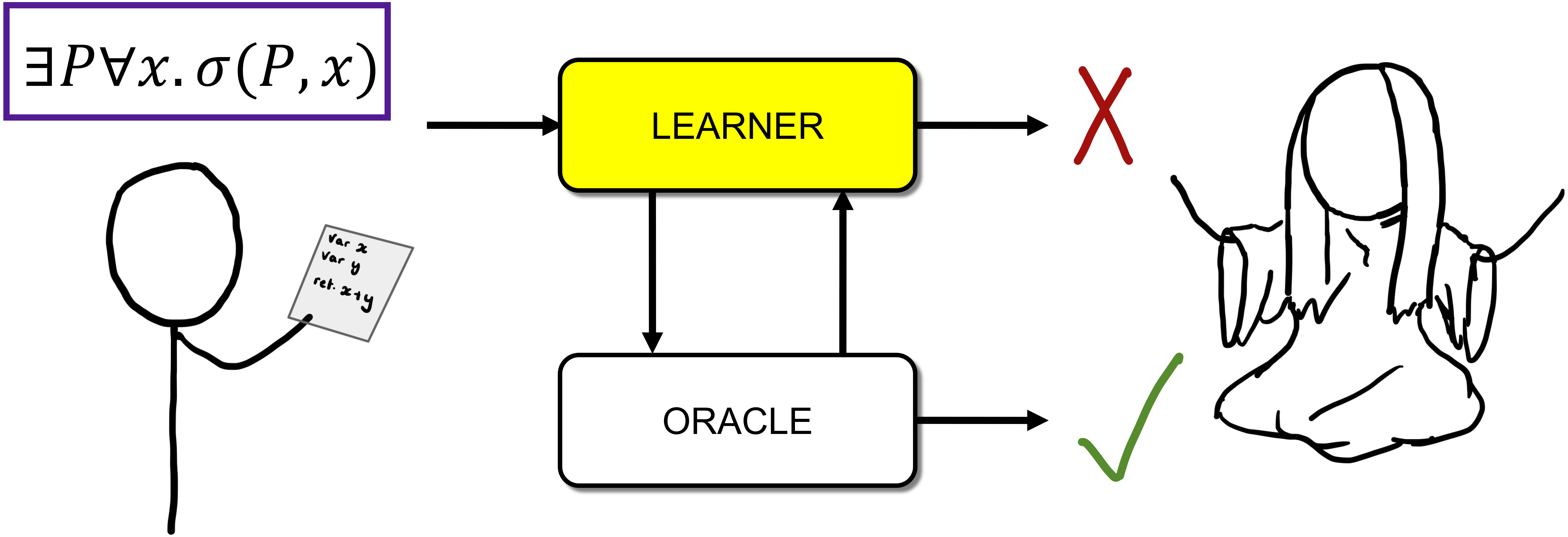
Does there exist a function P_T, in TACO, such that, for all possible inputs x, $P_T(x)$ gives the same result as the original source program $P_C(x)$ in C.

Specification: randomly generated input-output examples

Correctness oracle: compile and execute on a small set of examples, then test on a much bigger set

Algorithms for formal synthesis

Oracle Guided Inductive Synthesis



Searches program space and guesses candidates

Says if the candidate is correct, and guides the search if not

C2TACO - Grammar

$\langle PROGRAM \rangle$ ::= $\langle TENSOR \rangle$ = $\langle EXPR \rangle$

$\langle TENSOR \rangle$ ::= $\langle ID \rangle$ ($\langle INDEX\text{-}EXPR \rangle$) | $\langle ID \rangle$

$\langle INDEX\text{-}EXPR \rangle$ ::= $\langle INDEX\text{-}VAR \rangle$
  | $\langle INDEX\text{-}VAR \rangle$, $\langle INDEX\text{-}EXPR \rangle$

$\langle INDEX\text{-}VAR \rangle$ ::= $i \mid j \mid k \mid l$

$\langle EXPR \rangle$ ::= $\langle EXPR \rangle$ + $\langle EXPR \rangle$
  | $\langle EXPR \rangle$ - $\langle EXPR \rangle$
  | $\langle EXPR \rangle$ * $\langle EXPR \rangle$
  | $\langle EXPR \rangle$ / $\langle EXPR \rangle$
  | $\langle CONSTANT \rangle$
  | $\langle TENSOR \rangle$

$\langle ID \rangle$ ::= $T_0 \mid T_1 \mid T_2 \mid \ldots$

$\langle CONSTANT \rangle$ ::= $C_0 \mid C_1 \mid C_2 \mid \ldots$

Bottom up enumeration

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

Programs so far

Programs of length 1:

X Y 0 1 ⊥

Bottom up enumeration

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

Programs so far

X Y 0 1 ⊥

Programs of length 2:

X+X Y+Y X+0 X+1 Y+0 Y+1 X+Y

-X -Y -0 -1

ite(⊥, X, X) ite(⊥, X, Y) ite(⊥, X, 0). ...

X=X X=Y Y=Y Y=0 Y=1 X=1 X=0

...

Bottom up enumeration

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

Programs so far

X Y 0 1 ⊥ X+X Y+Y X+0 X+1 Y+0 Y+1 X+Y -X -Y -0 -1 ite(⊥, X, X) ite(⊥, X, Y) ite(⊥, X, 0). …	X=X X=Y Y=Y Y=0 Y=1 X=1 X=0

…

Programs of length 3:

Bottom up enumeration

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

Programs so far

X Y 0 1 ⊥

X+X Y+Y X+0
X+1 Y+0 Y+1 X+Y

-X -Y -0 -1

ite(⊥, X, X) ite(⊥, X, Y) ite(⊥, X, 0).
...

X=X
X=Y
Y=Y
Y=0
Y=1
X=1
X=0

...

Programs of length 3:

Problem: exponential search space!

Bottom up enumeration of templates

$\langle PROGRAM \rangle$ ::= $\langle TENSOR \rangle$ = $\langle EXPR \rangle$

$\langle TENSOR \rangle$ ::= $\langle ID \rangle$ ($\langle INDEX\text{-}EXPR \rangle$) | $\langle ID \rangle$

$\langle INDEX\text{-}EXPR \rangle$ ::= $\langle INDEX\text{-}VAR \rangle$
| $\langle INDEX\text{-}VAR \rangle$, $\langle INDEX\text{-}EXPR \rangle$

$\langle INDEX\text{-}VAR \rangle$ ::= $i \mid j \mid k \mid l$

$\langle EXPR \rangle$ ::= $\langle EXPR \rangle$ + $\langle EXPR \rangle$
| $\langle EXPR \rangle$ - $\langle EXPR \rangle$
| $\langle EXPR \rangle$ * $\langle EXPR \rangle$
| $\langle EXPR \rangle$ / $\langle EXPR \rangle$
| $\langle CONSTANT \rangle$
| $\langle TENSOR \rangle$

$\langle ID \rangle$ ::= $T_0 \mid T_1 \mid T_2 \mid \ldots$

$\langle CONSTANT \rangle$ ::= $C_0 \mid C_1 \mid C_2 \mid \ldots$

- Instead of enumerating complete programs, enumerate programs with holes in place of arguments
- Extend the correctness oracle to check all possible combinations of asignments to the holes

Observational Equivalence

Programs so far

X Y 0 1 ⊥

X+X Y+Y X+0 X+1 Y+0
Y+1 X+Y

-X -Y -0 -1

ite(⊥, X, X)
ite(⊥, X, Y)
ite(⊥, X, 0). ...

X=X
X=Y
Y=Y
Y=0
Y=1
X=1
X=0

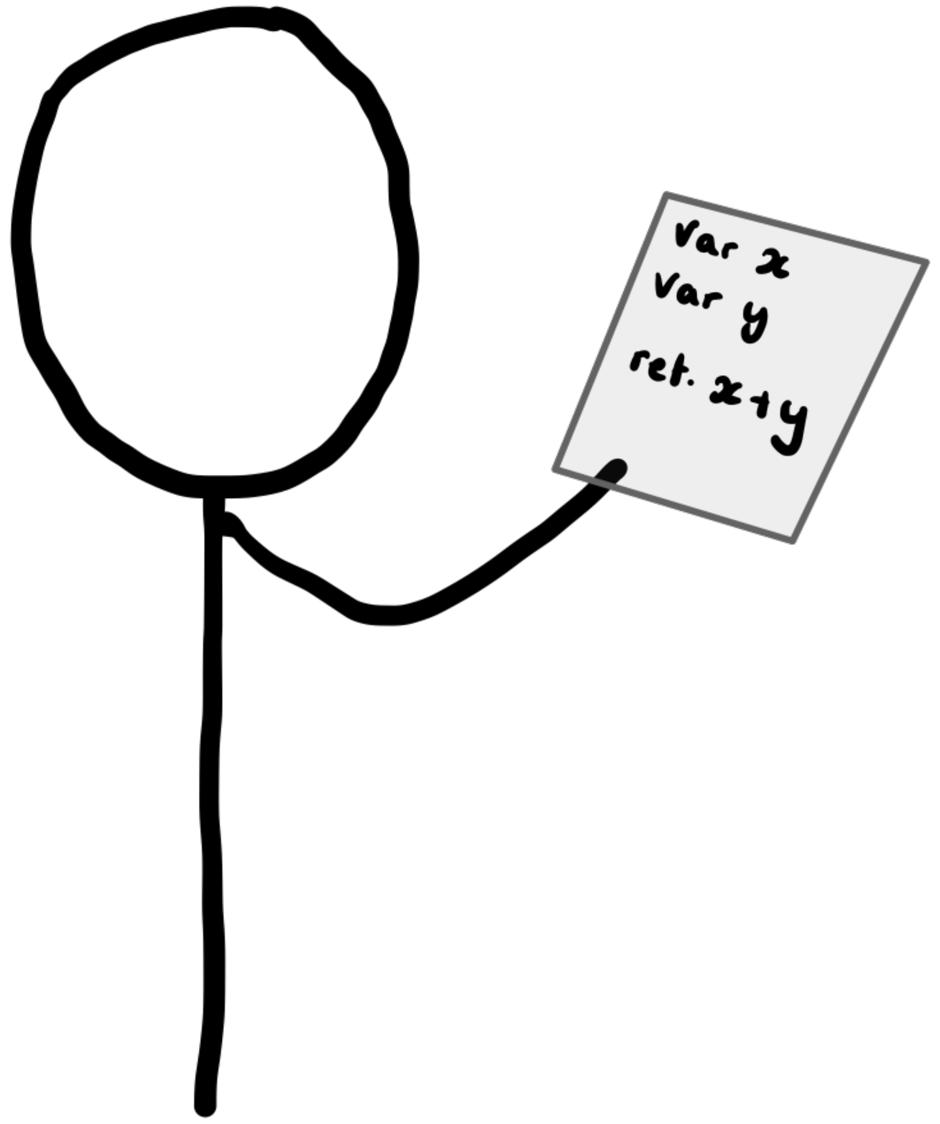
...

- If multiple candidate programs behave the same on all the inputs, we can discard all but one
- Tames exponential growth.. a bit

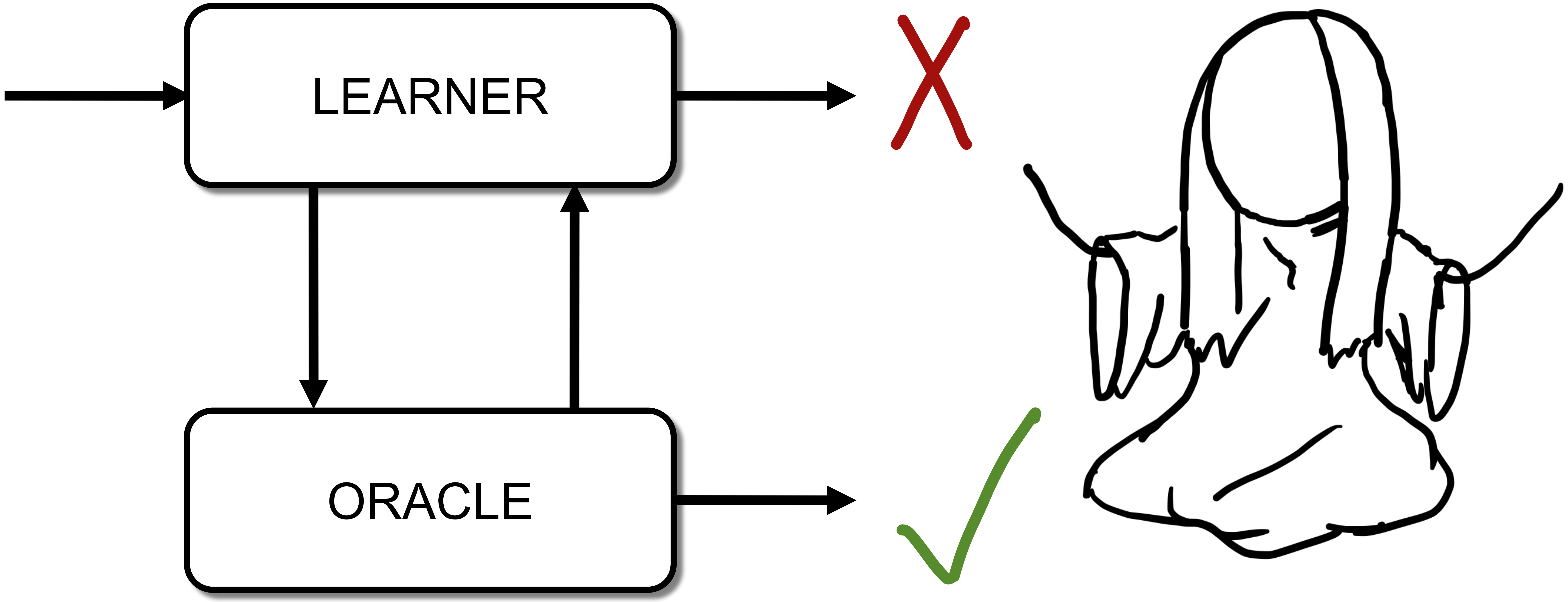
Algorithms for formal synthesis

Oracle Guided Inductive Synthesis

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

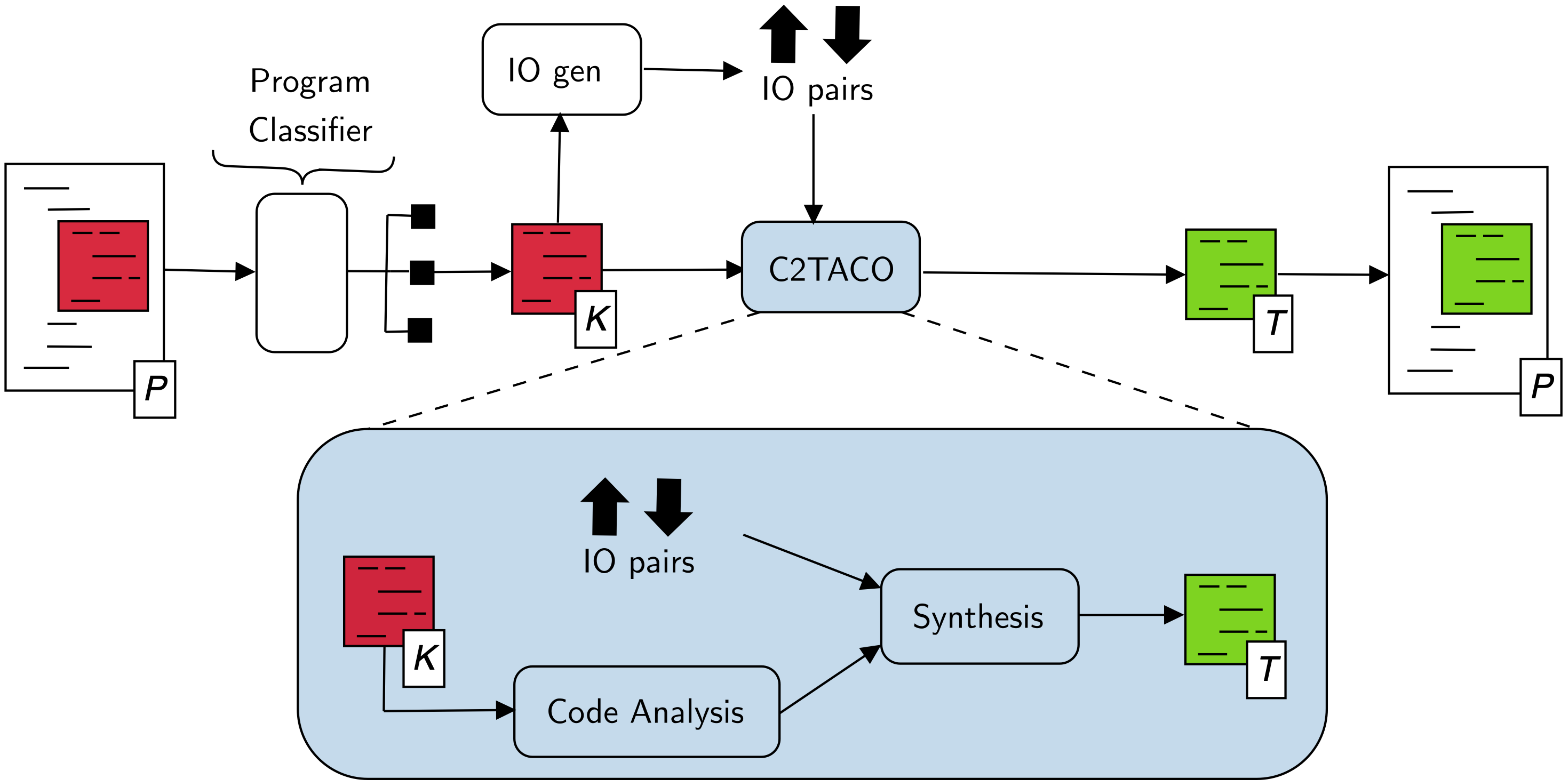


Searches program space and guesses candidates

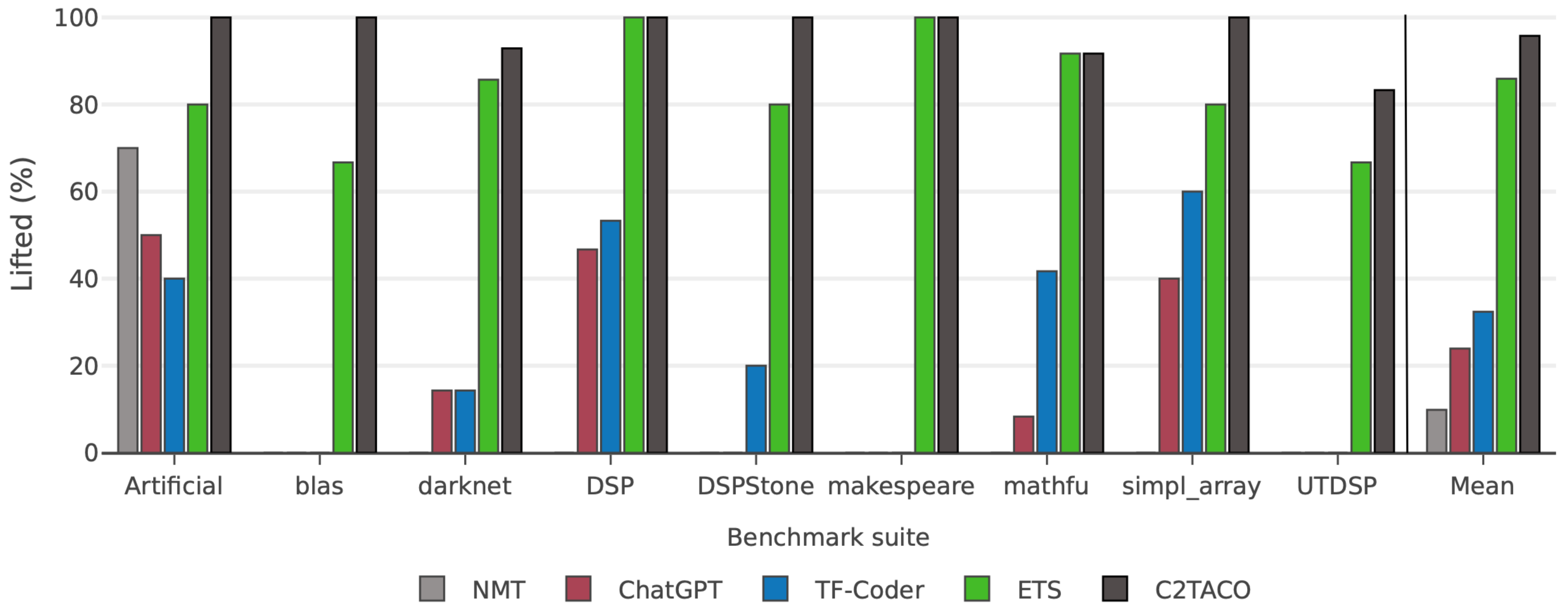


Says if the candidate is correct, and guides the search if not

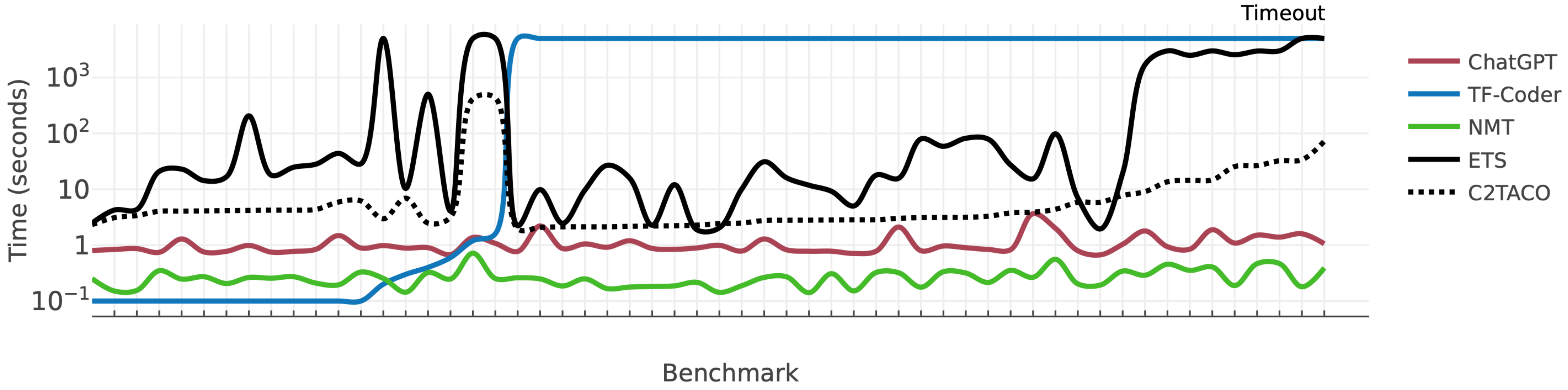
C2TACO - Overview



C2TACO - Performance

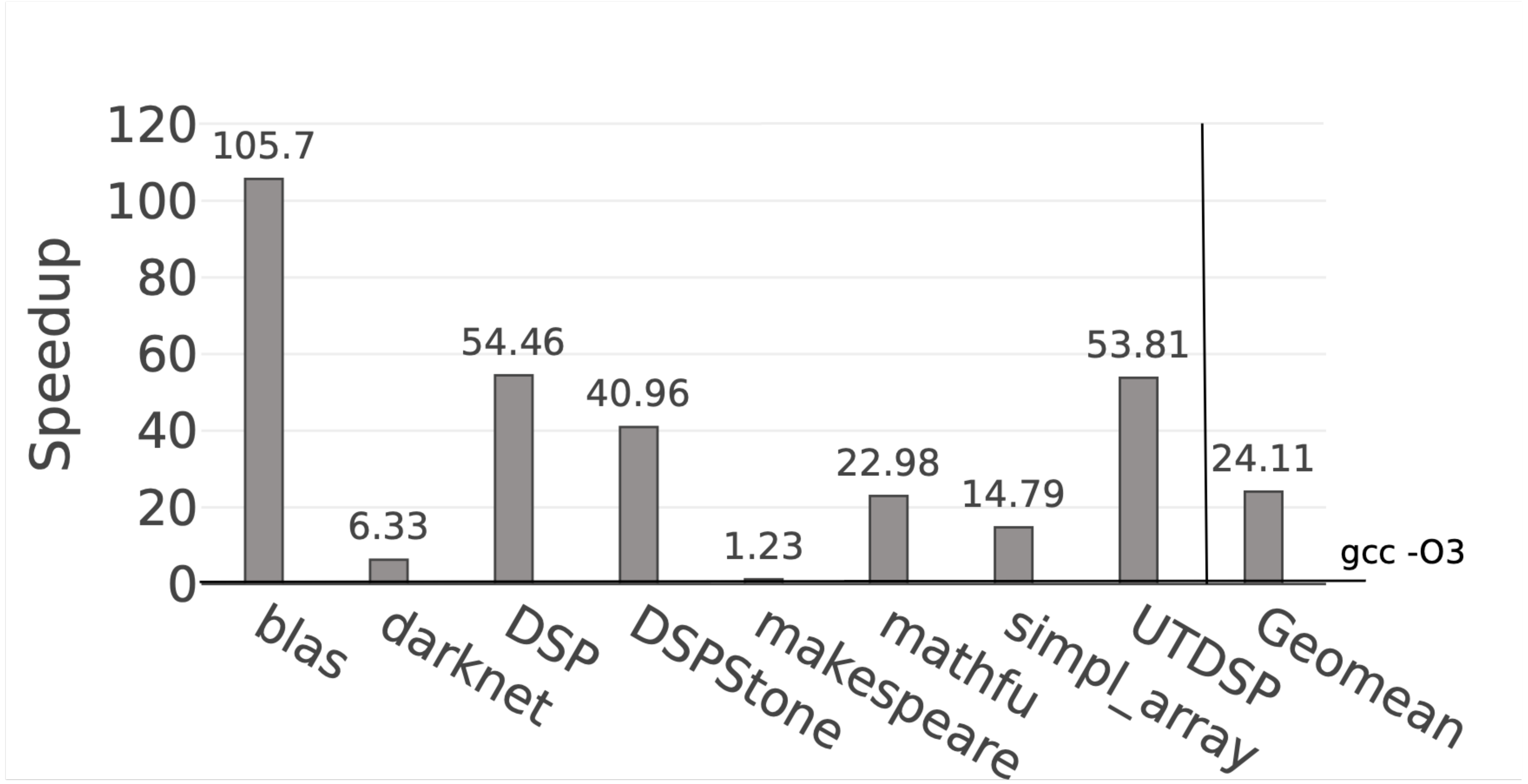


- Better than NMT and chatGPT
- 5.6s average synthesis time

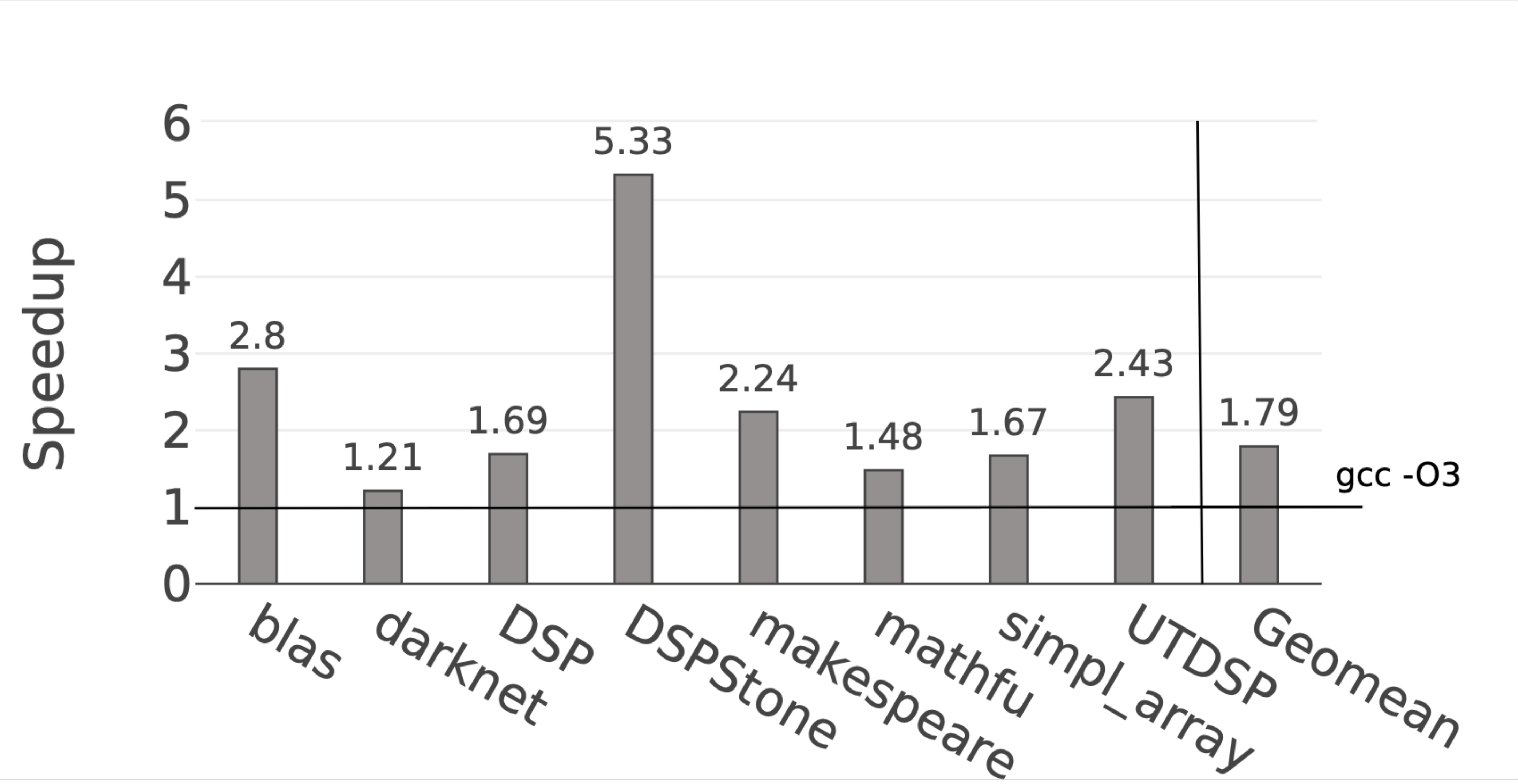


C2TACO - Performance

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



GPU

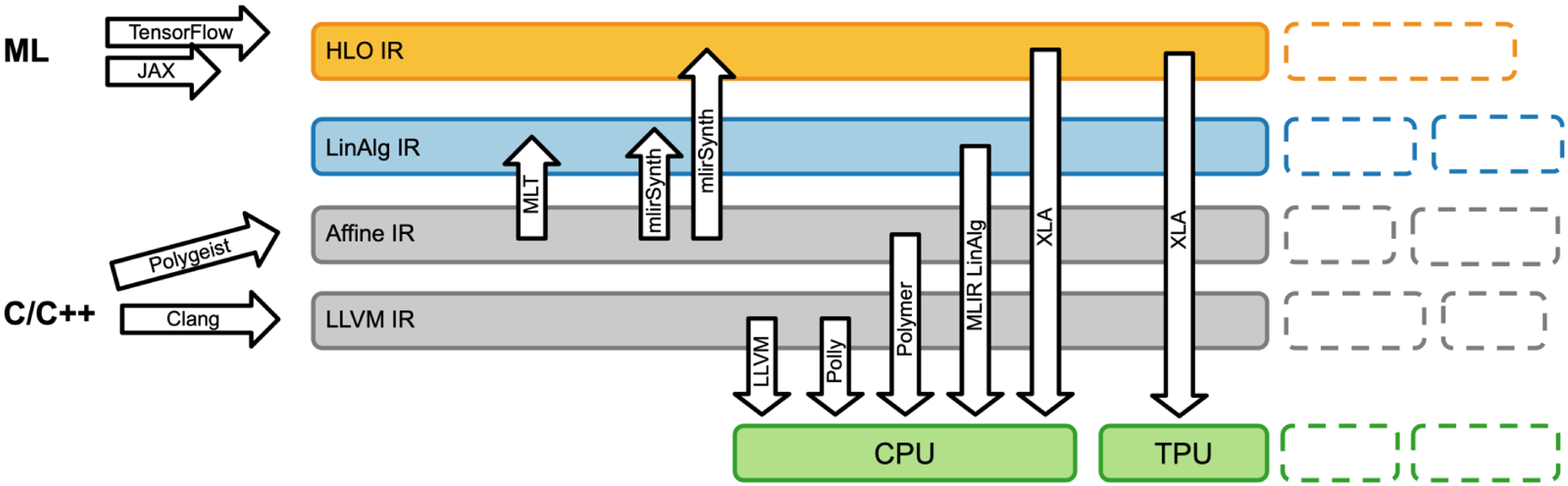


CPU

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`

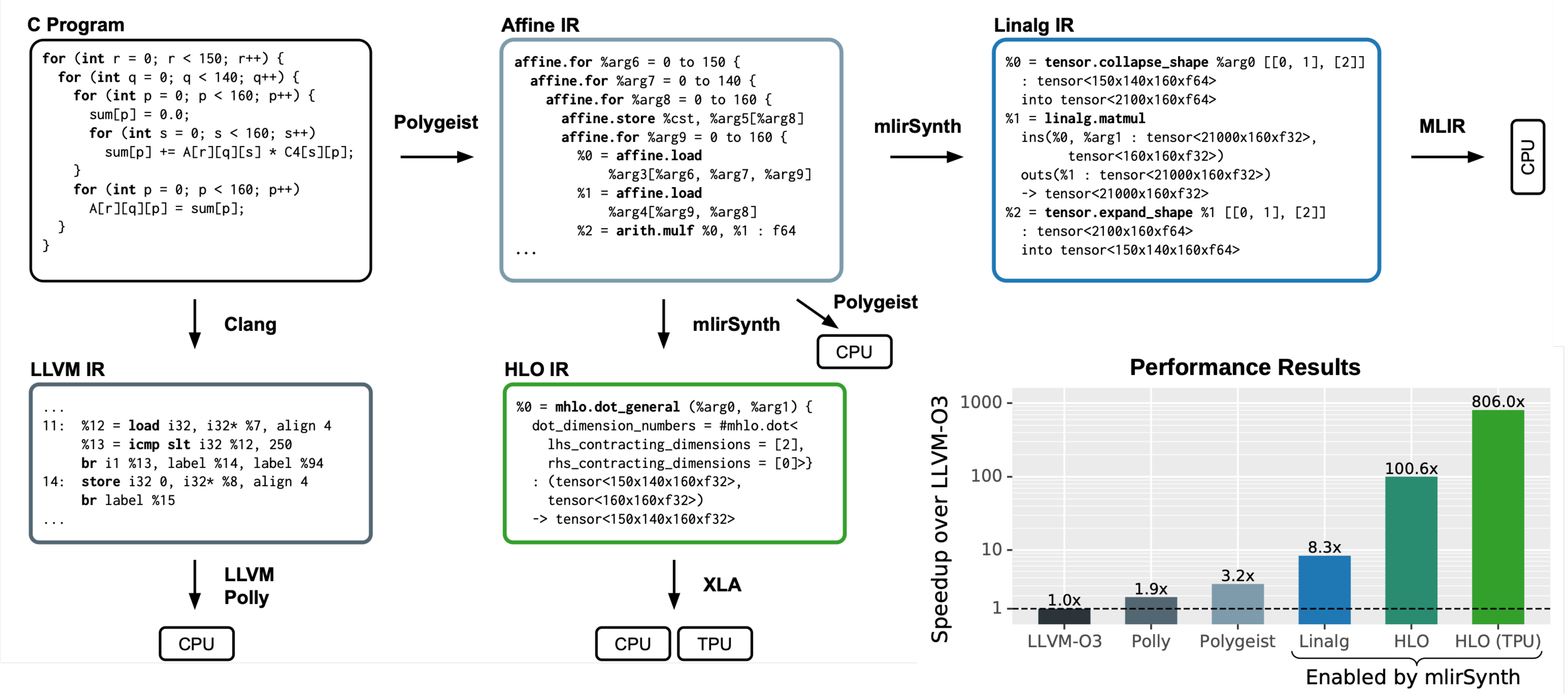
mlirSynth

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



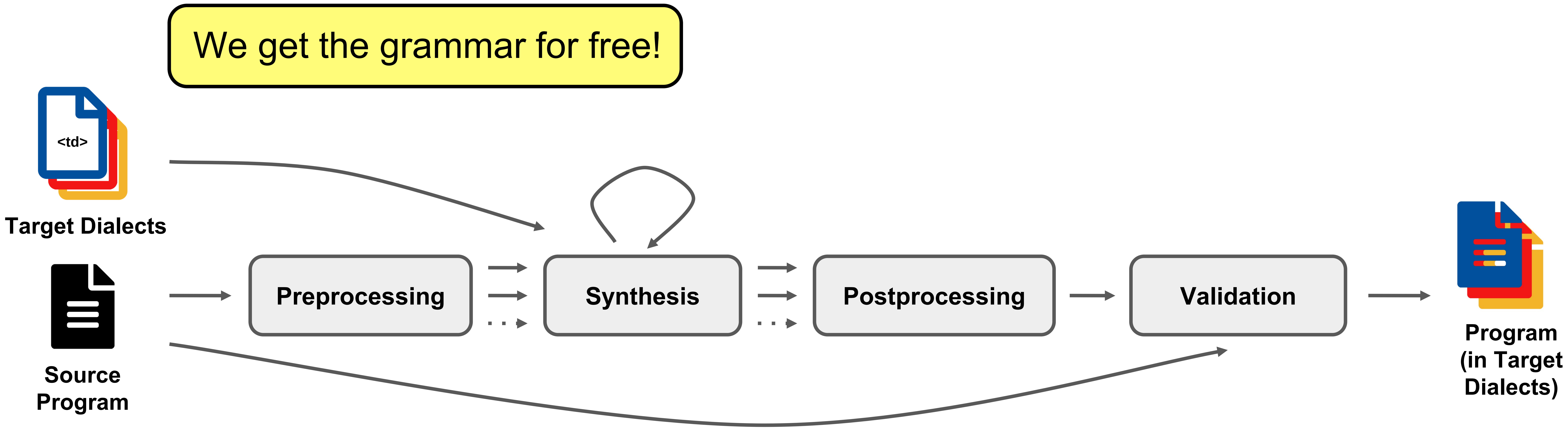
mlirSynth: Automatic, Retargetable Program Raising in Multi-Level IR using Program Synthesis - Alexander Brauckmann, Elizabeth Polgreen, Tobias Grosser, Michael O'Boyle

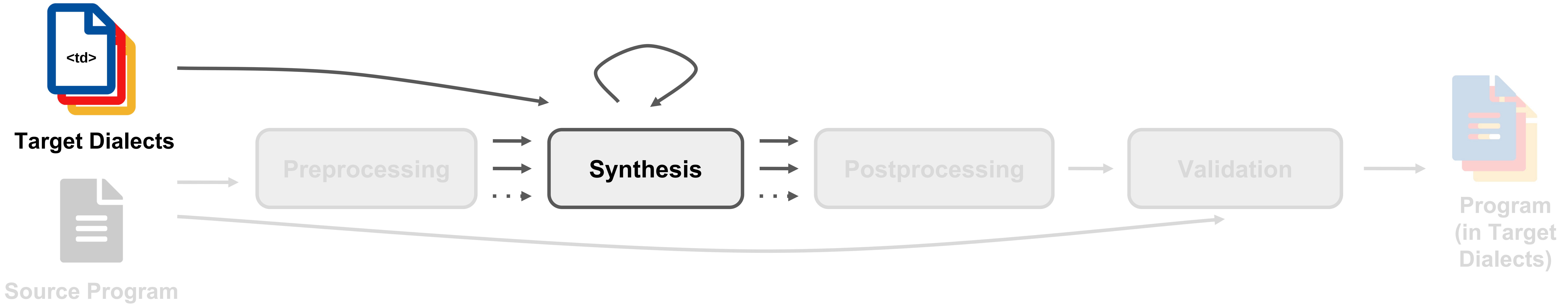
mlirSynth



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

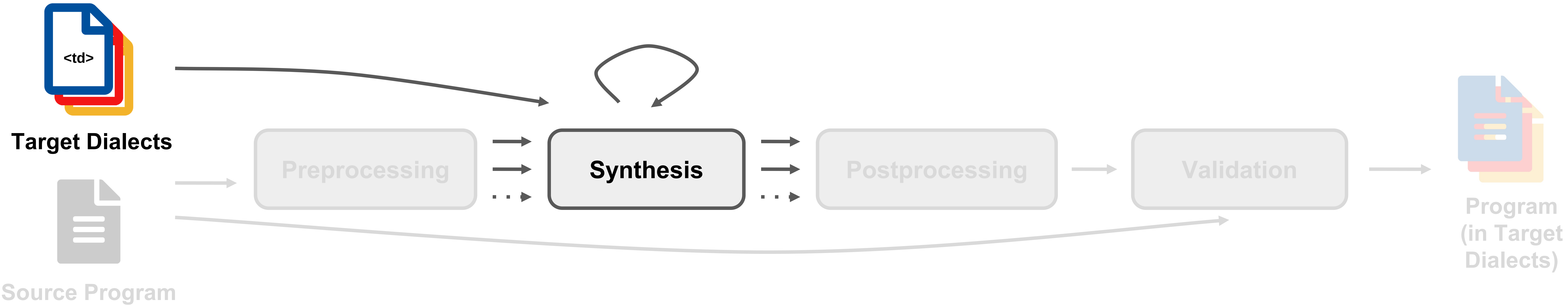
- Generate Input/Output example

Bottom-up enumerative search

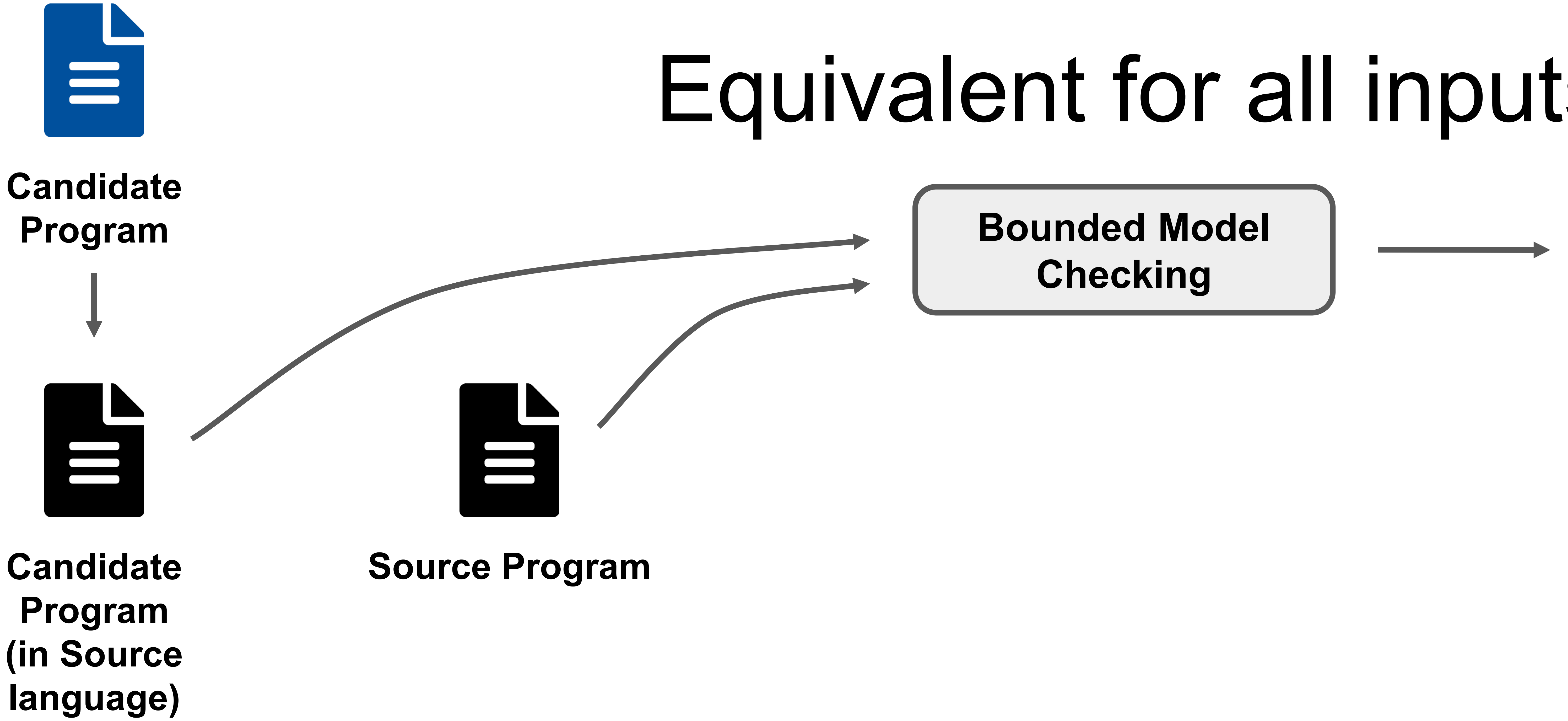
- Progressively grow a candidate set by combining simpler to more complex ones
- Initialization: Basic programs (returning arguments, constants)
- Terminate when specification matched

Optimization techniques

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



Equivalent for all inputs?

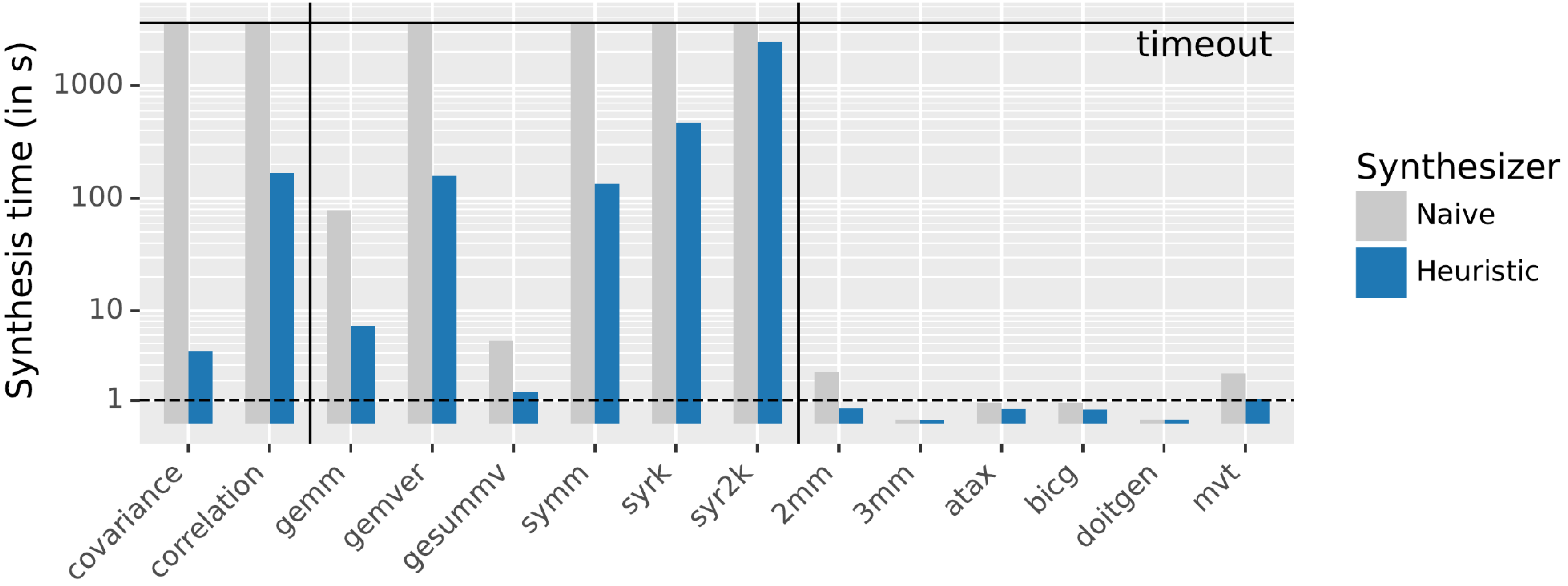


Equivalence Guarantees

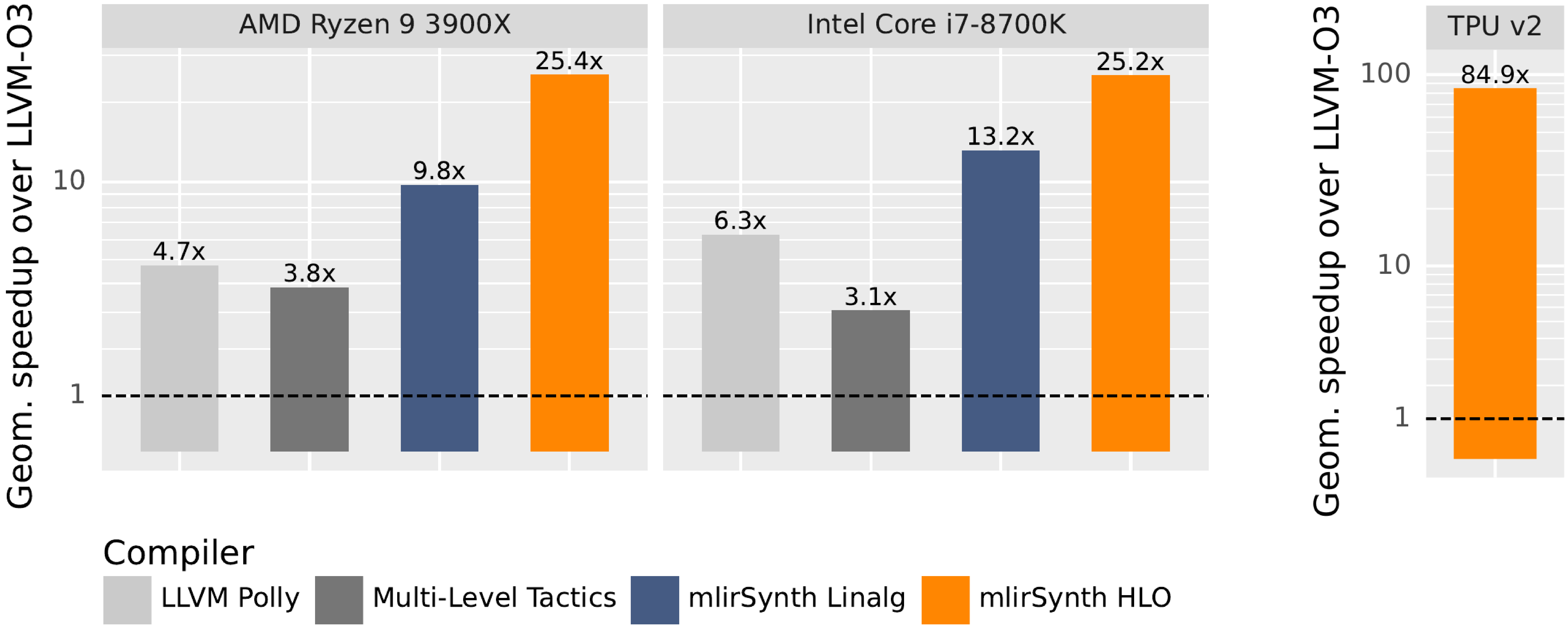
a) Float arithmetic
b) Float arithmetic, permitting small δ
c) Integer arithmetic

Testing I/O equivalence

mlirSynth - Performance



mlirSynth - Performance



Challenges

- Taking simple techniques from formal synthesis gave us big performance speed-ups
- Limitations:
  - Hand-writing heuristics
  - Correctness guarantees

Current work:

- Learning heuristics for formal synthesis:
  - Using reinforcement learning [1]
  - Using Large Language Models [2]
- Can we learn heuristics for “real-world” problems?
- Can we provide stronger guarantees of correctness?

[1] Data-Generation and Reinforcement Learning for Syntax-Guided Synthesis – Julian Parsert and Elizabeth Polgreen. AAAI 2024

[2] Guiding Enumerative Synthesis with Large Language Models – Yixuan Li, Julian Parsert and Elizabeth Polgreen. CAV 2024

Conclusions

- Taking simple techniques from formal synthesis gave us big performance speed-ups
- But there's still lots of work to do…!
- Currently recruiting for PhD students (international or home fees)

elizabeth.polgreen@ed.ac.uk