Algorithmic Program Synthesis
Looks familiar?

Produce small code fragments that satisfy the given specification

Does this look like something you know in the context of programming?
Compilation vs Synthesis

**Compilation:** it does not produce unknowns just translates.

- Represent **source** program as abstract syntax tree (AST)
- **Lower** the AST from source to target language
- Lowering performed with tree **rewrite rules**, which are guaranteed to be correct.

**(Deductive) Synthesis:**

- Similar mechanism: start from **spec**, and use rules to **lower** the spec to a desired program.
- Rewrite sequence can be **non-deterministic**
- Rewrite rules **need not** be arbitrarily **composable**.
A \textit{partial} program defines a \textit{search space}.

Goal: find a candidate in the space that satisfies $\phi$

The space is huge!

Try to describe it symbolically and use the power of solvers to do the search.
Search Over Candidate Programs

spec

program-to-formula translator

\( \phi \)
solver
“synthesis engine”

\( h \mapsto 1 \)
code generator

\( P[1] \)

sketch

sketch \( P[h] \)
How does this work?
Synthesis as Search

spec: int foo (int x) {
    return x + x;
}

sketch: int bar (int x) implements foo {
    return x << ?;  
}

result: int bar (int x) implements foo {
    return x << 1;
}
Program as a Formula

We have a program:

\[ f(x) \{ \text{return } x + x \} \]

And a formula that represents it:

\[ S_f(x, y) : y = x + x \]

Now a solver is an interpreter:

\[ S_f(x, y) \land x = 3 \quad y \mapsto 6 \]

And, a program inverter:

\[ S_f(x, y) \land y = 6 \quad x \mapsto 3 \]

This bidirectionally enables synthesis!
Constraints Solving for Search

We have a spec and a partial program:

```plaintext
spec(x) { return x + x }

sketch(x) { return x << ?? }
```

The solver finds $h$, and therefore, synthesizes the program:

$$S_{sketch}(x, y, h) : y = x \times 2^h$$

We may not always get lucky with the choice of input:

$$S_{sketch}(x, y, h) \land x = 2 \land y = 4 \quad h \mapsto 1$$

$$S_{sketch}(x, y, h) \land x = 0 \land y = 0 \quad h \mapsto 1, 2, 3, 4, \ldots$$

$$\land S_{sketch}(x', y', h) \land x' = 3 \land y' = 6 \quad h \mapsto 1$$
Inductive Synthesis

Small world hypothesis:

There is a small set of inputs where if the program is correct for these, then it is correct for every input.

So, instead of solving this:

$$\exists h \forall x. \phi(x, P(x, h))$$

We solve this:

$$\exists h. \phi(x_1, P(x_1, h)) \land \cdots \land \phi(x_n, P(x_n, h))$$

But where do these magical inputs come from?
CounterExample-Guided Inductive Synthesis (CEGIS)

Inductive Synthesizer
compute a candidate implementation from concrete inputs.

$$(x_1, o_1), ..., (x_k, o_k)$$

unrealizable

fail

succeed

candidate implementation

add a counterexample input

verifier/checker

Your verifier/checker goes here
Small World Hypothesis

C = size of candidate space = \exp(\text{bits of controls})
End of Synthesis Algorithm