Scaling Program Synthesis by Exploiting Existing Code

James Bornholt
Emina Torlak
University of Washington
Synthesis: write programs automatically

Syntax

Semantics
Synthesis: write programs automatically

Syntax

Semantics
Synthesis: write programs automatically

Syntax

Semantics

Target Behavior

$f(x) = 4x + 1$
Synthesis: write programs automatically

Syntax

Semantics

Target Behavior

\[ f(x) = 4x + 1 \]
Synthesis: write programs automatically

Target Behavior
\[ f(x) = 4x + 1 \]

Syntax
- \( (x << 2) + 1 \)
- \( x + x + x + x + 1 \)

Semantics
- \( 4 \times x + 1 \)
Synthesis: write programs automatically

Syntax

Semantics

Target Behavior
Permutation(L, f(L))
∧ Sorted(f(L))
Synthesis: write programs automatically

Target Behavior
\[ \text{Permutation}(L, f(L)) \land \text{Sorted}(f(L)) \]

Syntax
- Bubble Sort

Semantics
- Quicksort
The success of synthesis

End-user programming by example [FlashFill, POPL’11]

Cache coherence protocols [Transit, PLDI’13]

Parallel browser layout engines [PPoPP’13]

Compilers for new spatial architectures [Chlorophyll, PLDI’14]
The success of synthesis

End-user programming by example [FlashFill, POPL’11]

Cache coherence protocols [Transit, PLDI’13]

Parallel browser layout engines [PPoPP’13]

Compilers for new spatial architectures [Chlorophyll, PLDI’14]
The success of synthesis

End-user programming by example [FlashFill, POPL’11]

Cache coherence protocols [Transit, PLDI’13]

Parallel browser layout engines [PPoPP’13]

Compilers for new spatial architectures [Chlorophyll, PLDI’14]

Approximate Computing

quality bounds into approximate programs

Synthesis

high-level intent

into

low-level detail
The success of synthesis

End-user programming by example [FlashFill, POPL’11]

Cache coherence protocols [Transit, PLDI’13]

Parallel browser layout engines [PPoPP’13]

Compilers for new spatial architectures [Chlorophyll, PLDI’14]

Approximate Computing

quality bounds into approximate programs

Hardware Synthesis

programs into hardware designs

Synthesis

turns high-level intent into low-level detail
The success of synthesis

End-user programming by example [FlashFill, POPL’11]

Cache coherence protocols [Transit, PLDI’13]

Parallel browser layout engines [PPoPP’13]

Compilers for new spatial architectures [Chlorophyll, PLDI’14]
Approximate Computing
quality bounds into approximate programs

Not all applications require perfect accuracy
Approximate Computing
quality bounds into approximate programs

Not all applications require perfect accuracy
Approximate Computing
quality bounds into approximate programs

Not all applications require perfect accuracy

Video and image quality
Not all applications require perfect accuracy

Video and image quality

Sensors and simulation

Machine learning
Approximate Computing
quality bounds into approximate programs

Not all applications require perfect accuracy

Precise Implementation
Not all applications require perfect accuracy.

*Approximate Computing*

quality bounds into approximate programs

Precise Implementation

Desired Quality
Approximate Computing
quality bounds into approximate programs

Not all applications require perfect accuracy

Precise Implementation

Desired Quality

Approximate Compiler
Not all applications require perfect accuracy.
Approximate Computing
quality bounds into approximate programs

Not all applications require perfect accuracy

Precise Implementation
Approximate Program Synthesis†
Desired Quality
Approximate Program

Hardware Synthesis

programs into hardware designs

Synthesizing circuits from high-level languages
Hardware Synthesis
programs into hardware designs

Synthesizing circuits from high-level languages

module example(input a, b, c, output y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
Hardware Synthesis
programs into hardware designs

Synthesizing circuits from high-level languages

module example(input a, b, c, output y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
Synthesizing circuits from high-level languages

Hardware Synthesis
programs into hardware designs

module example(input a, b, c, output y);
  assign y = ~a & ~b & ~c | a & ~b & c | a & ~b & c;
endmodule
Hardware Synthesis
programs into hardware designs

Synthesizing circuits from high-level languages

High-Level Synthesis (HLS)
Hardware Synthesis

programs into hardware designs

Synthesizing circuits from high-level languages

High-Level Synthesis (HLS)

Place and route
Timing closure
Crosstalk and feedback
Quantum!
Hardware Synthesis
programs into hardware designs

Synthesizing circuits from high-level languages

Mark Wyse
UW grad student and HLS extraordinaire
Hardware Synthesis

programs into hardware designs

Synthesizing circuits from high-level languages

float dist(float a[3], float b[3]) {
    float r = 0;
    r += (a[0] - b[0]) * (a[0] - b[0]);
    r += (a[1] - b[1]) * (a[1] - b[1]);
    return sqrt(r);
}
Black Box Systems
observed behaviors into specifications

Defining system behavior by observation
Defining system behavior by observation
Black Box Systems
observed behaviors into specifications

Defining system behavior by observation
Black Box Systems

observed behaviors into specifications

Defining system behavior by observation
Black Box Systems
observed behaviors into specifications

Defining system behavior by observation

Programming by example
Black Box Systems
observed behaviors into specifications

Defining system behavior by observation

Programming by example
Synthesizing x86 instruction specs

Godefroid and Taly. Automated Synthesis of Symbolic Instruction Encodings from I/O Samples. PLDI’12.
Approximate Computing
quality bounds into approximate programs

Hardware Synthesis
programs into hardware designs

Black Box Systems
observed behaviors into specifications
Machine Learning and Synthesis
Learning programs from examples

Black Box Systems
observed behaviors
into specifications

\[ \exists P. \bigwedge_{x_i \in X} \varphi(x_i, P(x_i)) \]
Learning programs from examples

Black Box Systems

observed behaviors into specifications

\[ \exists P. \bigwedge_{x_i \in X} \varphi(x_i, P(x_i)) \]
Learning programs from examples

FlashFill

Version Space Algebra
Machine learning and synthesis

- Program Synthesis
- Statistical Machine Learning
Machine learning and synthesis

Program Synthesis → Millions of examples/parameters → Statistical Machine Learning
Machine learning and synthesis

100 instructions

Program Synthesis

Millions of examples/parameters

Statistical Machine Learning
Machine learning and synthesis

- Approximate Computing
- Hardware Synthesis
- Black Box Systems

100 instructions

Program Synthesis

Statistical Machine Learning

Millions of examples/parameters
Machine learning and synthesis

100 instructions

Program Synthesis

Millions of examples/parameters

Statistical Machine Learning

Approximate Computing

Hardware Synthesis

Black Box Systems

100 instructions

Millions of examples/parameters
Programs are not uniformly distributed.
Programs are not uniformly distributed.
Programs are not uniformly distributed.
Programs are not uniformly distributed.
Programs are not uniformly distributed.
Component-Based Synthesis
Synthesis of Loop-Free Programs

Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]
Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

- and
- xor
- add
- >> 1

Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Synthesis of Loop-Free Programs

\[
f(x, y) = \left[ \frac{x + y}{2} \right]
\]

- `add`
- `and`
- `xor`
- `>> 1`

Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Synthesis of Loop-Free Programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Higher-level components

\[ f(L) = \min \{ l_i \mid l_i \in L \} \]
Higher-level components

\[ f(L) = \min\{l_i \mid l_i \in L\} \]
Higher-level components

\[ f(L) = \min \{ l_i \mid l_i \in L \} \]
Higher-level components

\[ f(L) = \min \{ l_i \mid l_i \in L \} \]

\[ f(L) = \text{quicksort}(L)[0] \]
Higher-level components

\[ f(L) = \min\{l_i \mid l_i \in L\} \]

\[ f(L) = \text{quicksort}(L)[0] \]

Mine existing code bases for common idioms
Higher-level components

\[ f(L) = \min \{ l_i | l_i \in L \} \]

\[ f(L) = \text{quicksort}(L)[0] \]

Mine existing code bases for common idioms

Hardware Synthesis
programs
into
hardware designs
Producing candidate programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

- and
- xor
- add
- >> 1

Diagram:
- x
- y
- f(x, y)
Producing candidate programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

\[ \text{xor} \quad \text{add} \quad >> 1 \]
Producing candidate programs

$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$
Producing candidate programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Learn search heuristics from existing code
Producing candidate programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Learn search heuristics from existing code

Programming by example
[FlashFill, POPL'11]

Autocomplete with synthesis
[CodeHint, ICSE'14]
Producing candidate programs

\[ f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor \]

Mine existing code bases for common idioms

Learn search heuristics from existing code

Programming by example
[FlashFill, POPL'11]

Autocomplete with synthesis
[CodeHint, ICSE'14]
Producing candidate programs

- Learn search heuristics from existing code
- Mine existing code bases for common idioms

Black Box Systems
observed behaviors into specifications

Selecting among many candidate solutions
Sketch-Based Synthesis
Sketches make synthesis more tractable
Sketches make synthesis more tractable

\[
\text{bit}[W] \ \text{popCount}(\text{bit}[W] \ x) \ \{ \\
\quad x = (x \ & \ 0x5555) + ((x \ >> \ 1) \ & \ 0x5555); \\
\quad x = (x \ & \ 0x3333) + ((x \ >> \ 2) \ & \ 0x3333); \\
\quad x = (x \ & \ 0x0077) + ((x \ >> \ 8) \ & \ 0x0077); \\
\quad x = (x \ & \ 0x000F) + ((x \ >> \ 4) \ & \ 0x000F); \\
\quad \text{return} \ x; \\
\}
\]
Sketches make synthesis more tractable

```c
bit[W] popCount(bit[W] x) {
    x = (x & 0x5555) + ((x >> 1) & 0x5555);
    x = (x & 0x3333) + ((x >> 2) & 0x3333);
    x = (x & 0x0077) + ((x >> 8) & 0x0077);
    x = (x & 0x000F) + ((x >> 4) & 0x000F);
    return x;
}

bit[W] popSketched(bit[W] x) {
    loop (??) {
        x = (x & ??) + ((x >> ??) & ??);
    }
    return x;
}
```
Sketches make synthesis more tractable

```c
bit[W] popSketched(bit[W] x) {
    loop (??) {
        x = (x & ??) + ((x >> ??) & ??);
    }
    return x;
}
```
Sketches make synthesis more tractable

```c
bit[W] popSketched(bit[W] x) {
    loop (??) {
        x = (x & ??) + ((x >> ??) & ??);
    }
    return x;
}

int[] map(int[] xs) {
    int[] ys = {};
    for (int i=0; i<xs.length; i++) {
        ys.append(??(xs[i]));
    }
    return ys;
}

int[] foldl(int[] xs) {
    int acc = ??;
    for (int i=0; i<xs.length; i++) {
        acc = ??(acc, xs[i]);
    }
    return ys;
}

int[] filter(int[] xs) {
    int[] ys = {};
    for (int i=0; i<xs.length; i++) {
        if (??(xs[i])) ys.append(xs[i]);
    }
}
```
Sketches make synthesis more tractable

Identify common sketches from existing code
Sketches make synthesis more tractable

- Mine existing code bases for common idioms
- Identify common sketches from existing code
Sketches make synthesis more tractable

Mine existing code bases for common idioms

Identify common sketches from existing code

Approximate Computing
quality bounds into approximate programs
Sketches make synthesis more tractable

Mine existing code bases for common idioms

Identify common sketches from existing code

Approximate Computing

quality bounds into approximate programs

Precise Implementation

Approximate Program Synthesis
Thanks!