Program Synthesis

Emina Torlak
drina@cs.washington.edu
Today

Last lecture

• Solvers as angelic runtime oracle

Today

• Program synthesis: from specs to code

Reminders

• HW3 is due on Friday.
• No class next week; happy Thanksgiving!
The program synthesis problem

∃ P. ∀ x. φ(x, P(x))

Find a program P that satisfies the specification φ on all inputs.
The program synthesis problem

\[ \exists P. \forall x. \varphi(x, P(x)) \]

\( \varphi \) may be a formula, a reference implementation, input / output pairs, traces, demonstrations, etc.

Find a program \( P \) that satisfies the specification \( \varphi \) on all inputs.
The program synthesis problem

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

$\varphi$ may be a formula, a reference implementation, input / output pairs, traces, demonstrations, etc.

Synthesis improves
- Productivity (when writing $\varphi$ is easier than writing $P$).
- Correctness (when verifying $\varphi$ is easier than verifying $P$).

Find a program $P$ that satisfies the specification $\varphi$ on all inputs.
Two kinds of program synthesis

 Existence of programs satisfying a given property:

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

- Synthesis as a problem in deductive theorem proving.
- Synthesis as a search problem.
Two kinds of program synthesis

∃ P. ∀ x. φ(x, P(x))

Deductive (classic) synthesis

Inductive (syntax-guided) synthesis
Two kinds of program synthesis

Deductive (classic) synthesis
Derive the program $P$ from the constructive proof of the theorem $\forall x. \exists y. \varphi(y, x)$.

Inductive (syntax-guided) synthesis

$\exists P. \forall x. \varphi(x, P(x))$
Two kinds of program synthesis

\[ \exists P. \forall x. \varphi(x, P(x)) \]

**Deductive (classic) synthesis**
Derive the program \( P \) from the constructive proof of the theorem \( \forall x. \exists y. \varphi(y, x) \).

**Inductive (syntax-guided) synthesis**
Discover the program \( P \) by searching a restricted space of candidate programs for one that satisfies \( \varphi \) on all inputs.
Two kinds of program synthesis

**Deductive (classic) synthesis**

*Derive* the program $P$ from the constructive proof of the theorem $\forall x. \exists y. \varphi(y, x)$.

**Inductive (syntax-guided) synthesis**

*Discover* the program $P$ by searching a restricted space of candidate programs for one that satisfies $\varphi$ on all inputs.
Deductive synthesis with axioms and E-graphs

Denali Superoptimizer
[Joshi, Nelson, Randall, PLDI’02]
Deductive synthesis with axioms and E-graphs

Specification $\varphi$, given as a reference implementation.

$\text{reg6 } \ast 4 + 1$

Denali Superoptimizer
[Joshi, Nelson, Randall, PLDI’02]
Deductive synthesis with axioms and E-graphs

Specification $\varphi$, given as a reference implementation.

$\text{reg6} \times 4 + 1$

Denali Superoptimizer [Joshi, Nelson, Randall, PLDI’02]

Optimal (lowest cost) program $P$ that is equivalent to $\varphi$ on all inputs (values of $\text{reg6}$).

$s4addl(\text{reg6}, 1)$
Deductive synthesis with axioms and E-graphs

Specification $\varphi$, given as a reference implementation.

$\forall \ k, n. \ 2^n = 2^{\ast\ast n}$

$\forall \ k, n. \ k \cdot 2^n = k \ll n$

$\forall \ k, n. \ k \cdot 4 + n = s4addl(k, n)$

...*

Optimal (lowest cost) program $P$ that is equivalent to $\varphi$ on all inputs (values of $reg6$).

Denali Superoptimizer

$[\text{Joshi, Nelson, Randall, PLDI’02}]$

$s4addl(reg6, 1)$

Two kinds of axioms:

- Instruction semantics.
- Algebraic properties of functions and relations used for specifying instruction semantics.
Deductive synthesis with axioms and E-graphs

Specification \( \varphi \), given as a reference implementation.

1. Construct an E-graph.
2. Use a SAT solver to search the E-graph for a K-cycle program.

Optimal (lowest cost) program \( P \) that is equivalent to \( \varphi \) on all inputs (values of reg6).

Denali Superoptimizer
[Joshi, Nelson, Randall, PLDI'02]

reg6 * 4 + 1 \rightarrow s4addl(reg6, 1)

\( \forall k, n. \ 2^n = 2^{**n} \)
\( \forall k, n. \ k*2^n = k \ll n \)
\( \forall k, n. \ k*4 + n = \text{s4addl}(k, n) \)

Two kinds of axioms:
• Instruction semantics.
• Algebraic properties of functions and relations used for specifying instruction semantics.
Denali by example

∀ k, n. 2^n = 2**n
∀ k, n. k*2^n = k << n
∀ k, n. k*4 + n = s4addl(k, n)
...

E-graph matching

 SAT

s4addl(reg6, 1)
Denali by example

\[ \forall \, k, n. \, 2^n = 2^{*n} \]
\[ \forall \, k, n. \, k*2^n = k \ll n \]
\[ \forall \, k, n. \, k*4 + n = s4addl(k, n) \]
\[ \ldots \]

E-graph matching

\[ \begin{align*}
\text{reg6} & \ast 4 + 1 \\
\forall \, k, n. \, 2^n &= 2^{*n} \\
\forall \, k, n. \, k*2^n &= k \ll n \\
\forall \, k, n. \, k*4 + n &= s4addl(k, n) \\
\ldots 
\end{align*} \]

SAT

\[ s4addl(\text{reg6}, 1) \]
Denali by example

reg6 * 4 + 1

∀ k, n. $2^n = 2^{*n}$
∀ k, n. $k*2^n = k << n$
∀ k, n. $k*4 + n = s4addl(k, n)$
...

E-graph matching

SAT

$s4addl(reg6, 1)$
Denali by example

∀ k, n. 2^n = 2**n
∀ k, n. k*2^n = k << n
∀ k, n. k*4 + n = s4addl(k, n)
...

reg6 * 4 + 1

E-graph matching

SAT

s4addl(reg6, 1)
Denali by example

∀ k, n. $2^n = 2^{**n}$
∀ k, n. $k \cdot 2^n = k \ll n$
∀ k, n. $k \cdot 4 + n = s4addl(k, n)$
...

E-graph matching

SAT

$s4addl(reg6, 1)$
Deductive synthesis versus compilation

**Deductive synthesizer**
- Non-deterministic.
- Searches *all correct rewrites* for one that is optimal.

**Compiler**
- Deterministic.
- Lowers a source program into a target program using a *fixed sequence of rewrite steps*.

\[
reg6 \times 4 + 1 \\
\downarrow \\
reg6 \ll 2 + 1
\]
Deductive synthesis versus inductive synthesis

Deductive synthesis

- Efficient and provably correct: thanks to the semantics-preserving rules, only correct programs are explored.
- Requires sufficient axiomatization of the domain.
- Requires complete specifications to seed the derivation.
Deductive synthesis versus inductive synthesis

In Deductive synthesis:
- Efficient and provably correct: thanks to the semantics-preserving rules, only correct programs are explored.
- Requires sufficient axiomatization of the domain.
- Requires complete specifications to seed the derivation.

Inductive synthesis:
- Works with multi-modal and partial specifications.
- Requires no axioms.
- But often at the cost of lower efficiency and weaker (bounded) guarantees on the correctness/optimality of synthesized code.

\[ \exists P \forall x. \varphi(x, P(x)) \]
Inductive syntax-guided synthesis

CEGIS: Counterexample-Guided Inductive Synthesis
[Solar-Lezama et al, ASPLOS'06]
Inductive syntax-guided synthesis

A partial or multimodal specification $\varphi$ of the desired program (e.g., assertions, i/o pairs).

$\text{reg6} \ast 4 + 1$

CEGIS: Counterexample-Guided Inductive Synthesis
[Solar-Lezama et al, ASPLOS'06]
Inductive syntax-guided synthesis

A partial or multimodal specification \( \varphi \) of the desired program (e.g., assertions, i/o pairs).

\[ \text{reg6} \times 4 + 1 \]

**expr** :=
- \text{const} | \text{reg6} |
- \text{s4addl}(\text{expr}, \text{expr}) |
- ...

CEGIS: Counterexample-Guided Inductive Synthesis
[Solar-Lezama et al, ASPLOS'06]

A syntactic sketch (e.g., a grammar) describing the shape of the desired program \( P \).
This defines the space of candidate programs to search. Can be fine-tuned for better performance.
Inductive syntax-guided synthesis

A partial or multimodal specification \( \varphi \) of the desired program (e.g., assertions, i/o pairs).

\[ \text{reg6} \times 4 + 1 \]

A syntactic sketch (e.g., a grammar) describing the shape of the desired program \( P \).

This defines the space of candidate programs to search. Can be fine-tuned for better performance.

A program \( P \) from the given space of candidates that satisfies \( \varphi \) on all (usually bounded) inputs.

\[ \text{s4addl(reg6, 1)} \]

 expr := const | \text{reg6} | \text{s4addl(expr, expr)} | ...

CEGIS: Counterexample-Guided Inductive Synthesis

[Solar-Lezama et al, ASPLOS'06]
Inductive syntax-guided synthesis

A partial or multimodal specification \( \varphi \) of the desired program (e.g., assertions, i/o pairs).

Guess a program that works on a finite set of inputs, verify it, and learn from bad guesses.

A program \( P \) from the given space of candidates that satisfies \( \varphi \) on all (usually bounded) inputs.

\begin{align*}
\mathit{expr} & := \\
& \mathit{const} \mid \mathit{reg6} \mid \\
& \mathit{s4addl}(\mathit{expr}, \mathit{expr}) \mid \\
& \ldots
\end{align*}

\( \mathit{reg6} \ast 4 + 1 \)

\( \mathit{s4addl}(\mathit{reg6}, 1) \)

CEGIS: Counterexample-Guided Inductive Synthesis
[\text{Solar-Lezama et al, ASPLOS'06}]

A syntactic sketch (e.g., a grammar) describing the shape of the desired program \( P \).
This defines the space of candidate programs to search. Can be fine-tuned for better performance.
Overview of CEGIS

 Specification $\varphi$
 Sketch S

 Synthesizer
 Verifier
Overview of CEGIS

Specification $\varphi$
Sketch $S$

Searches for a program $P \in S$ that satisfies $\varphi$ on all inputs $x_i$ seen so far.

Synthesizer
Verifier
Overview of CEGIS

Specification $\varphi$
Sketch $S$

Searches for a program $P \in S$ that satisfies $\varphi$ on all inputs $x_i$ seen so far.

Synthesizer
Verifier

Fail
Overview of CEGIS

Searches for a program $P \in S$ that satisfies $\varphi$ on all inputs $x_i$ seen so far.

Searches for an input $x_{i+1}$ on which $P$ violates $\varphi$.

- Specification $\varphi$
- Sketch $S$

Synthesizer

Verifier

$P \in S$ s.t. $\land_i \varphi(x_i, P(x_i))$

Fail
Overview of CEGIS

Searches for a program \( P \in S \) that satisfies \( \varphi \) on all inputs \( x_i \) seen so far.

Searches for an input \( x_{i+1} \) on which \( P \) violates \( \varphi \).

\[ P \in S \text{ s.t. } \bigwedge_i \varphi(x_i, P(x_i)) \]

Speciation \( \varphi \)

Sketch \( S \)

Synthesizer

Verifier

Fail

no
counterexample

\( P \)
Overview of CEGIS

Specification $\varphi$
Sketch $S$

Searches for a program $P \in S$ that satisfies $\varphi$ on all inputs $x_i$ seen so far.

Searches for an input $x_{i+1}$ on which $P$ violates $\varphi$.

$P \in S$ s.t. $\land_i \varphi(x_i, P(x_i))$

Synthesizer
Verifier

Fail
no counterexample

$P$
Overview of CEGIS

Searches for a program $P \in S$ that satisfies $\varphi$ on all inputs $x_i$ seen so far.

Searches for an input $x_{i+1}$ on which $P$ violates $\varphi$.

Specification $\varphi$

Sketch $S$

Fail

$P \in S \text{ s.t. } \land_i \varphi(x_i, P(x_i))$

$X_{i+1}$

no counterexample
Overview of CEGIS

Searches for a program $P \in S$ that satisfies $\varphi$ on all inputs $x_i$ seen so far.

Usually a solver, but can be a test suite, end-user, etc.

Specification $\varphi$

Sketch $S$

$P \in S$ s.t. $\land_i \varphi(x_i, P(x_i))$

$x_{i+1}$

Fail

No counterexample

$P$
Overview of CEGIS

Any search algorithm: e.g., a solver, enumerative search, stochastic search.

Usually a solver, but can be a test suite, end-user, etc.

Specification $\varphi$

Sketch $S$

$P \in S \text{ s.t. } \land_{i} \varphi(x_i, P(x_i))$

$x_{i+1}$

Fail

no counterexample

$P$
Synthesizing programs with a solver

Logical encoding of the synthesis problem for the inputs 0, 1, 2.

[Solar-Lezama et al, ASPLOS'06]
Synthesizing programs with a solver

Logical encoding of the synthesis problem for the inputs 0, 1, 2.

- Replace each ?? with a fresh symbolic constant.

[x * 4]

[x <= n]

[Solar-Lezama et al, ASPLOS'06]
Synthesizing programs with a solver

- Replace each `??` with a fresh symbolic constant.
- Translate the resulting problem to constraints w.r.t. the current inputs.

\[
\begin{align*}
&0, 1, 2 \\
&x \times 4 \\
&x \ll n
\end{align*}
\]

\[
\begin{align*}
&(0 \ll n = 0) \wedge \\
&(1 \ll n = 4) \wedge \\
&(2 \ll n = 8)
\end{align*}
\]

[Solar-Lezama et al, ASPLOS'06]
Synthesizing programs with a solver

- Replace each `??` with a fresh symbolic constant.
- Translate the resulting problem to constraints w.r.t. the current inputs.
- If SAT, convert the model to a program $P$.

$x \times 4$
$x \ll n$

$0, 1, 2$

$(0 \ll n = 0) \land (1 \ll n = 4) \land (2 \ll n = 8)$

[Solar-Lezama et al, ASPLOS'06]
Synthesizing programs with enumerative search

\[ x \times 4 \]

\[ expr := 0 | 1 | 2 | x | expr \ll expr \]

[Udupa et al, PLDI'13]

A candidate program consistent with current inputs.
Synthesizing programs with enumerative search

• Iteratively construct all programs of size $K$ until one is consistent with the current inputs.

• If two programs produce the same output on all current inputs, keep just one of the two.

[Udupa et al, PLDI'13]
Synthesizing programs with enumerative search

- Iteratively construct all programs of size $K$ until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

[Udupa et al, PLDI'13]
Synthesizing programs with enumerative search

- Iteratively construct all programs of size $K$ until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

$expr ::= 0 | 1 | 2 | x | expr \ll expr$

[Udupa et al, PLDI'13]
Iteratively construct all programs of size $K$ until one is consistent with the current inputs.

If two programs produce the same output on all current inputs, keep just one of the two.

[Udupa et al, PLDI'13]
Synthesizing programs with enumerative search

- Iteratively construct all programs of size $K$ until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

$expr := 0 \mid 1 \mid 2 \mid x \mid expr \ll expr$

$K = 1: 0, 1, 2, x$

[Udupa et al, PLDI'13]
Synthesizing programs with enumerative search

- Iteratively construct all programs of size $K$ until one is consistent with the current inputs.
- If two programs produce the same output on all current inputs, keep just one of the two.

$\text{expr} := \begin{cases} 0 \mid 1 \mid 2 \mid x \mid \\ \text{expr} \ll \text{expr} \end{cases}$

[Udupa et al, PLDI'13]
Synthesizing programs with stochastic search

0, 1, 2

x * 4

expr := 0 | 1 | 2 | x |
expr <= expr

Stochastic synthesis

A candidate program consistent with current inputs.

[Schkufza et al, ASPLOS'13]
Synthesizing programs with stochastic search

- Use Metropolis-Hastings to sample expressions.
- Mutate the current candidate program and keep the mutation with probability proportional to its correctness w.r.t. the current inputs.

A candidate program consistent with current inputs.

expr := 0 | 1 | 2 | x | expr << expr

[Schkufza et al, ASPLOS'13]
Summary

Today

• Deductive and inductive synthesis

• Syntax-guided synthesis with symbolic, enumerative, and stochastic search

Next

• Applications of program synthesis and verification