Program Synthesis

Emina Torlak
emina@cs.washington.edu
Today

Last lecture
  • Solvers as angelic runtime oracle

Today
  • Program synthesis: from specs to code

Reminders
  • HW3 is due on Friday.
The program synthesis problem

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

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

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

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

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

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

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

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

Find a program \( P \) that satisfies the specification \( \phi \) on all inputs.
Two kinds of program synthesis

- Synthesis as a problem in deductive theorem proving.
- \[ \exists P. \forall x. \phi(x, P(x)) \]

- Synthesis as a search problem.
Two kinds of program synthesis

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

Deductive (classic) synthesis

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

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

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

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

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

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

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

**Inductive (syntax-guided) synthesis**
*Discover* the program $P$ by searching a restricted space of candidate programs for one that satisfies $\phi$ 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 $\Phi$, given as a reference implementation.

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

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

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

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

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

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

$s4\text{addl}(\text{reg6}, 1)$
Deductive synthesis with axioms and E-graphs

- Specification $\Phi$, given as a reference implementation:
  - $\forall k, n. 2^n = 2^{\cdot\cdot^n}$
  - $\forall k, n. k \cdot 2^n = k << n$
  - $\forall k, n. k \cdot 4 + n = s4\text{addl}(k, n)$
  - ...

- Optimal (lowest cost) program $P$ that is equivalent to $\Phi$ on all inputs (values of $\text{reg6}$):
  - $\text{Denali Superoptimizer}$
    - [Joshi, Nelson, Randall, PLDI’02]
  - $s4\text{addl}(\text{reg6}, 1)$

- 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 $\phi$, given as a reference implementation.

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

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

$\forall k, n. \quad k \cdot 4 + n = \text{s4addl}(k, n)$

...$

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

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

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

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

reg6 * 4 + 1

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)
...

E-graph matching

SAT

s4addl(reg6, 1)
Denali by example

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

reg6 \* 4 + 1

E-graph matching

\[
\begin{array}{c}
\ast \\
+ \\
\text{reg6} \\
4 \\
\text{s4addl(reg6, 1)}
\end{array}
\]

SAT
Denali by example

\[
\forall k, n. 2^n = 2^{\ast n}
\]
\[
\forall k, n. k \ast 2^n = k \ll n
\]
\[
\forall k, n. k \ast 4 + n = s4\text{addl}(k, n)
\]
\[
\ldots
\]

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

E-graph matching

\[
\text{s4\text{addl}(reg6, 1)}
\]

SAT
Denali by example

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

E-graph matching

SAT

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

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

…

E-graph matching

\[ s4addl \]

\[ \ll \]

\[ 2 + \]

\[ * \]

\[ 1 \]

\[ \text{reg6} \]

\[ 4 \]

\[ * \]

\[ 2 \]

\[ 2 \]

SAT

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

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.

$\exists P. \forall x. \phi(x, P(x))$
Inductive syntax-guided synthesis

CEGIS: Counterexample-Guided Inductive Synthesis

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

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

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

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

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

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

$\text{expr} := \text{const} \mid \text{reg6} \mid \text{s4addl} (\text{expr}, \text{expr}) \mid \ldots$

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 $\phi$ 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.

$\text{expr} :=\ 
\text{const} \mid \text{reg6} \mid \text{s4addl}(\text{expr}, \text{expr}) \mid \ldots$

CEGIS: Counterexample-Guided Inductive Synthesis

[Solar-Lezama et al, ASPLOS'06]

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

$s4addl(\text{reg6}, 1)$
Inductive syntax-guided synthesis

A partial or multimodal specification $\phi$ 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 $\phi$ on all (usually bounded) inputs.

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

$\text{s4addl}($reg6, 1$)$

$\text{expr} ::= \text{const} \mid \text{reg6} \mid \text{s4addl}(\text{expr}, \text{expr}) \mid \ldots$

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.
Overview of CEGIS

Specification $\Phi$
Sketch $S$

Synthesizer
Verifier
Overview of CEGIS

Specification $\phi$
Sketch $S$

Searches for a program $P \in S$ that satisfies $\phi$ on all inputs $x_i$ seen so far.
Overview of CEGIS

**Specification** $\Phi$

**Sketch** $S$

- Synthesizer
- Verifier

Searches for a program $P \in S$ that satisfies $\Phi$ on all inputs $x_i$ seen so far.
Overview of CEGIS

- **Specification $\phi$**
- **Sketch $S$**

### Synthesizer

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

### Verifier

- Searches for a program $P \in S$ that satisfies $\phi$ on all inputs $x_i$ seen so far.
- Searches for an input $x_{i+1}$ on which $P$ violates $\phi$.

**Fail**
Overview of CEGIS

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

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

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

Specication $\phi$
Sketch $S$

Synthesizer

Verifier

Fail

no counterexample

$P$
Overview of CEGIS

Specification $\Phi$
Sketch $S$

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

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

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

Synthesizer
Verifier

Fail
P

no counterexample
Overview of CEGIS

Speciﬁcation \( \phi \)

Sketch \( S \)

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

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

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

Fail

no counterexample

\( P \)
Overview of CEGIS

Specification $\Phi$

Sketch $S$

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

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

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

$x_{i+1}$

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 $\Phi$

Sketch $S$

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

$X_{i+1}$

Fail

P

no counterexample
Synthesizing programs with a solver

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

Solver-based synthesis

\[
x * 4
\]

\[
x \ll ???
\]

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

• Replace each ?? with a fresh symbolic constant.

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

[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.

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

[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.

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

0

x * 4

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

Enumeration-based synthesis

[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
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]
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

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

- 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.

$x \times 4$

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

$K = \{0, 1, 2, x\}$

[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.

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

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

$K=2: 1 \ll 2, 2 \ll 2, x \ll 1, x \ll 2$

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

A candidate program consistent with current inputs.

$\times \times 4$

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

[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.

$x \times 4$

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

[Schkufza et al, ASPLOS'13]
Summary

Today

• Deductive and inductive synthesis

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

Next

• Two exciting guest lectures!

• Program verification in the real world.