Program Synthesis

Emina Torlak
emina@cs.washington.edu
Today

Last lecture

• Solvers as angelic runtime oracle

Today

• Program synthesis: computers programming computers

Reminders

• HW3 is due.
• Please fill out the course evaluation (March 5-12)!
“Information technology has been praised as a labor saver and cursed as a destroyer of obsolete jobs. But the entire edifice of modern computing rests on a fundamental irony: the software that makes it all possible is, in a very real sense, handmade. Every miraculous thing computers can accomplish begins with a human programmer entering lines of code by hand, character by character.”

**Interview** with Moshe Vardi
“Information technology has been praised as a labor saver and cursed as a destroyer of obsolete jobs. But the entire edifice of modern computing rests on a fundamental irony: the software that makes it all possible is, in a very real sense, handmade. Every miraculous thing computers can accomplish begins with a human programmer entering lines of code by hand, character by character.”

Interview with Moshe Vardi
The program synthesis problem

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

Find a program P that satisfies the specification \( \varphi \) 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

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

\( \varphi \) 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

Deductive (classic) synthesis

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
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.

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

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

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

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

- Synthesis as a search problem.

SPIRAL

FlashFill
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.

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

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).

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

Denali Superoptimizer

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

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

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

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

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

$\ast 4 + 1$  $\rightarrow$  $s4addl(reg6, 1)$

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.

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

reg6 * 4 + 1

$\text{Denali Superoptimizer}$
$[\text{Joshi, Nelson, Randall, PLDI'02}]$

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

$s4addl(reg6, 1)$

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

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

...
Denali by example

\[\forall k, n. \ 2^n = 2^{\star n}\]

\[\forall k, n. k \star 2^n = k \ll n\]

\[\forall k, n. k \star 4 + n = s4addl(k, n)\]

\[s4addl(reg6, 1)\]

E-graph matching

SAT
Denali by example

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

\[ \forall k, n. k \cdot 2^n = k \ll n \]

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

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

E-graph matching

SAT

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

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

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

E-graph matching

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

SAT
Denali by example

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

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

E-graph matching

\[ \text{s4addl} \]

\[ \text{reg6} \]

\[ \text{2} \]
\[ \text{+} \]
\[ \text{1} \]
\[ \text{*} \]
\[ \text{4} \]
\[ \text{2} \]
\[ \text{2} \]

\[ \text{SAT} \]

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

```plaintext
reg6 * 4 + 1

reg6 << 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.

∃ P. ∀ x. φ(x, P(x))
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.

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

CEGIS: Counterexample-Guided Inductive Synthesis
[Solar-Lezama et al, ASPLOS'06]
A partial or multimodal specification $\varphi$ 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).

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

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

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 $\varphi$ 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.

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

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

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

$s4\text{addl}(\text{reg6}, 1)$
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.

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

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

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

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

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

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

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 $\varphi$.

Specification $\varphi$

Sketch $S$

Synthesizer

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

Verifier

Failed

no counterexample

$P$
Overview of CEGIS

Specifications $\varphi$

Sketch $S$

Synthesizer

Verifier

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. $\land_i \varphi(x_i, P(x_i))$

Fail

no counterexample

$P$
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 $x_{i+1}$ on which $P$ violates $\phi$.

Specification $\phi$
Sketch $S$

$P \in S \text{ s.t. } \bigwedge_i \phi(x_i, P(x_i))$

$X_{i+1}$

Fail

no counterexample
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.

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

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

$x_{i+1}$

Fail

P

no counterexample
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

Replace each `??` with fresh symbolic constant.

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

[x * 4]

[x << n]

[0, 1, 2]

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

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

\[ x \times 4 \]
\[ x \ll n \]
\[ 0, 1, 2 \]

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

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

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

\[
x \times 4
\]

\[
x \ll n
\]

\[
0, 1, 2
\]

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

\[
x \ll 2
\]

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

$0$,

A candidate program consistent with current inputs.

$x \times 4$

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

[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.

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

A candidate program consistent with current inputs.

[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 << 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]
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.

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

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

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

A candidate program consistent with current inputs.

\[ x \times 4 \]

\[
expr ::= \\
0 \mid 1 \mid 2 \mid x \mid \\
e_expr \ll expr
\]

[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.

A candidate program consistent with current inputs.

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

[Schkufza et al, ASPLOS'13]
Summary

Today

• Deductive synthesis with axioms and E-graphs
• Inductive synthesis with solvers, enumeration, and stochastic search

Next lecture

• Solver-aided languages