Program Synthesis
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

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

\(\phi\) 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

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

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

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

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

s4addl(reg6, 1)
Denali by example

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

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

E-graph matching

\[ \text{s4addl} \]

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.
Deductive synthesis versus inductive synthesis

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

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.
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.

Reg6 * 4 + 1

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

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

Searches for a program $P \in S$ such that $\forall i \phi(x_i, P(x_i))$.

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 $S$

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.

[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.

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

[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

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

Enumeration-based synthesis

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.

$X * 4$

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

[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.

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

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

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

- 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 stochastic search

A candidate program consistent with current inputs.

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

[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.

\[
\begin{align*}
expr & := \\
0 & | 1 & | 2 & | x & | \\
expr & \ll expr
\end{align*}
\]

[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.