Computer-Aided Reasoning for Software

CSE507

Program Synthesis

courses.cs.washington.edu/courses/cse507/14au/

Emina Torlak
emina@cs.washington.edu

Today

Last lecture

- Angelic nondeterminism and execution

Today

- Program synthesis: computers programming computers

Announcements

- Please fill out the course evaluation form (Dec 02-08)

Computers programming computers?

"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

Program synthesis aims to automate (tedious parts of) programming.

The program synthesis problem

φ may be a formula, a reference implementation, input/output pairs, traces, demonstrations, etc.

Synthesis improves

- Productivity (when writing φ is easier than writing P).
- Correctness (when verifying φ is easier than verifying P).

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

Find a program P that meets the input/output specification φ.

Two kinds of program synthesis

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

SPIRAL

Deductive (classic) synthesis

FlashFill

Inductive (syntax-guided) synthesis

Deductive synthesis with axioms and E-graphs

Complete specification φ of the desired program (a reference implementation in an ISA).

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 φ on all inputs (values of reg6).

`reg6 * 4 + 1`

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

`s4addl(reg6, 1)`

$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k << n

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

…

Two kinds of axioms:

- Instruction semantics.
- Algebraic properties of functions and relations used for specifying instruction semantics.

Denali by example

```
reg6 * 4 + 1
```

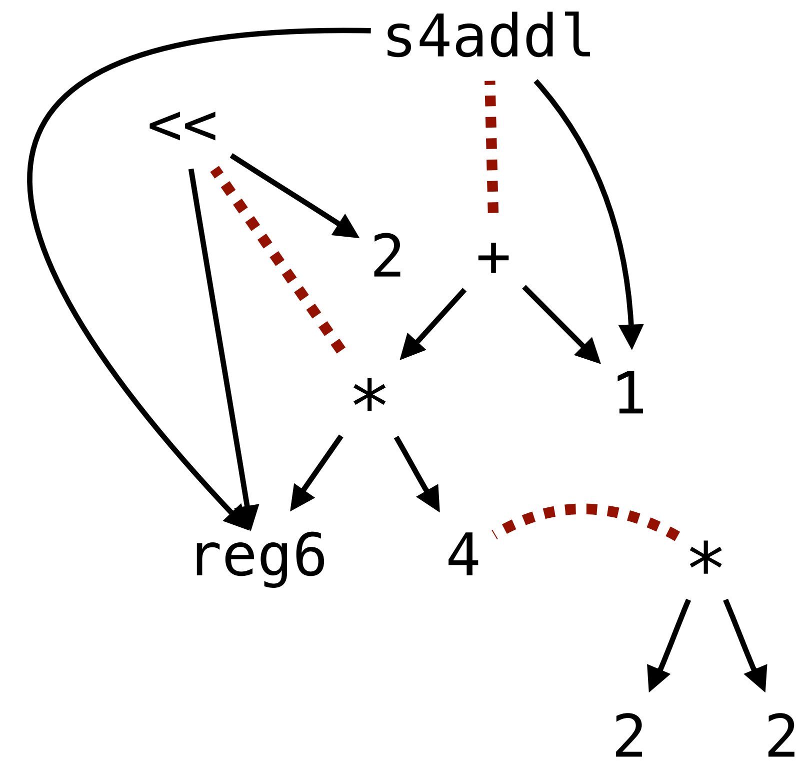
$\forall$ k, n. 2^n = 2**n

$\forall$ k, n. k*2^n = k **<<** n

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

…

E-graph matching



SAT

`s4addl(reg6, 1)`

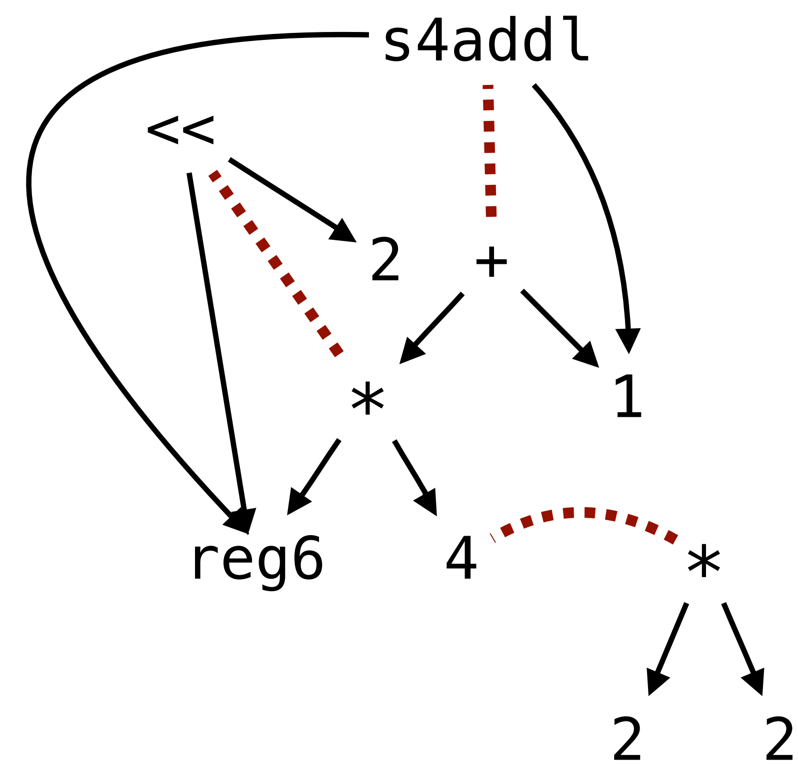
Deductive synthesis versus compilation

Deductive synthesizer

- Non-deterministic.
- *Searches* all correct rewrite sequences (proofs) for one that yields an optimal program.

Compiler

- Deterministic.
- Lowers a source program into a target program using a fixed sequence of rewrites.




```
reg6 * 4 + 1
      ↓
reg6 << 2 + 1
```

Deductive synthesis versus inductive synthesis

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

Deductive synthesis

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

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 φ of the desired program (e.g., assertions, i/o pairs).

Solves $\exists$ P. $\varphi(x_1, P(x_1)) \wedge \ldots \wedge \varphi(x_n, P(x_n))$ for *representative* inputs $x_1, \ldots, x_n$.

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

```
reg6 * 4 + 1
```

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

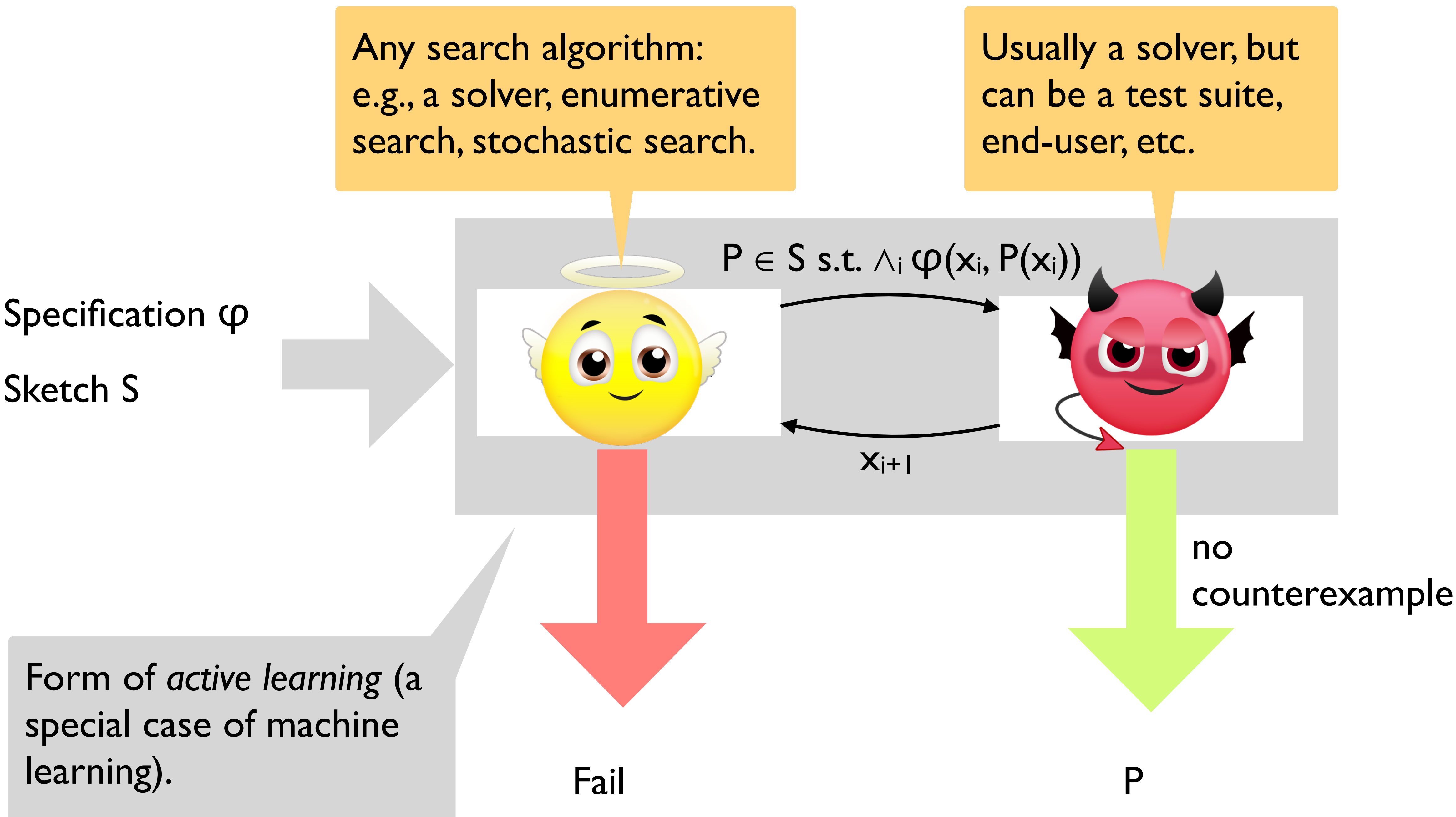
```
s4addl(reg6, 1)
```

```
expr :=
 const | reg6 |
 s4addl(expr, expr) |
 …
```

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



Inductive synthesis with a solver

`0, 1, 2`

`x * 4`

`x << n`

- 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 c program P.

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

`x << 2`

[Solar-Lezama et al, ASPLOS'06]

Inductive synthesis with enumerative search

`0`

`x * 4`

```
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.

K=1: 0, 1, 2, x

K=2: 1 << 2, 2 << 2,

x << 1, **x << 2**

[Udupa et al, PLDI'13]

Inductive synthesis with stochastic search

`0, 1, 2`

`x * 4`

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

- 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.

[Schkufza et al, ASPLOS'13]

Summary

Today

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

Next (and final) lecture

- Solver-aided languages