Course Overview

CS294: Program Synthesis for Everyone

Ras Bodik
Emina Torlak

Division of Computer Science
University of California, Berkeley

The name of the course

this CS294 topics course has been listed as

CS294: Programming Language Design for Everyone

Since putting the course on the books, we realized we are ready teach a superset of intended material.

In addition to

- design of domain-specific languages (DSLs) and
- their lightweight implementation

you will learn

- how to build a synthesizer in a semester

also a topic for everyone (PL students and others)

CAV tutorial

The course is based on our invited CAV 2012 tutorial

Synthesizing Programs with Constraint Solvers
slides (ppt)   slides (pdf)   screencast

We will expand all topics into standalone segments

- basics of modern verification (with solvers)
- embedding your language in a host language (Racket)
- synthesis algorithms (with solvers and without)
- creative specifications and tests, etc

Motivation: Two quotes

Computers Programming Computers?

from the an 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.

http://www.thetexaseconomy.org/business-industry/business-development/articles/article.php?name=computersProgramming

Motivation: Two quotes

Automated programming revisited

Ras Bodik

> Why is it that Moore’s Law hasn’t yet revolutionized the job of the programmer? Compute cycles have been harnessed in testing, model checking, and autotuning but programmers still code with bare hands. Can their cognitive load be shared with a computer assistant?

Discussion

Moore’s Law increased performance > 100x since the invention of C. Do you agree that this improvement has not conferred programmability benefits?

- garbage collection

- we can afford scripting overhead

reuse

What is program synthesis

Find a program P that meets a spec ϕ(input,output):

$$\underbrace{\exists P}_{\text{find P}} \, . \, \underbrace{\forall x \, . \, \phi(x, P(x))}_{\text{correctness cond.}}$$

When to use synthesis:

productivity: when writing ϕ is faster than writing P

correctness: when proving ϕ is easier than proving P

Is compiler a synthesizer

Can compilation be expressed with this formula?

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

Assume we want to compile source program $S(x)$ into target program $P(x)$. Can ϕ describe this?

$$\phi(x, y) = \quad y = S(x)$$

Compilation vs. synthesis

So where's the line between compilation & synthesis?

Compilation:

1) represent source program as abstract syntax tree (AST)

(i) parsing, (ii) name analysis, (iii) type checking

2) lower the AST from source to target language

eg, assign machine registers to variables, select instructions, …

Lowering performed with tree rewrite rules, sometimes based on analysis of the program

eg, a variable cannot be in a register if its address is in another variable

Synthesis, classical

Key mechanisms similar to compilation

- start from a *spec* = src program, perhaps in AST form
- rewrite rules lower the spec to desired program

But rewrite sequence can be non-deterministic

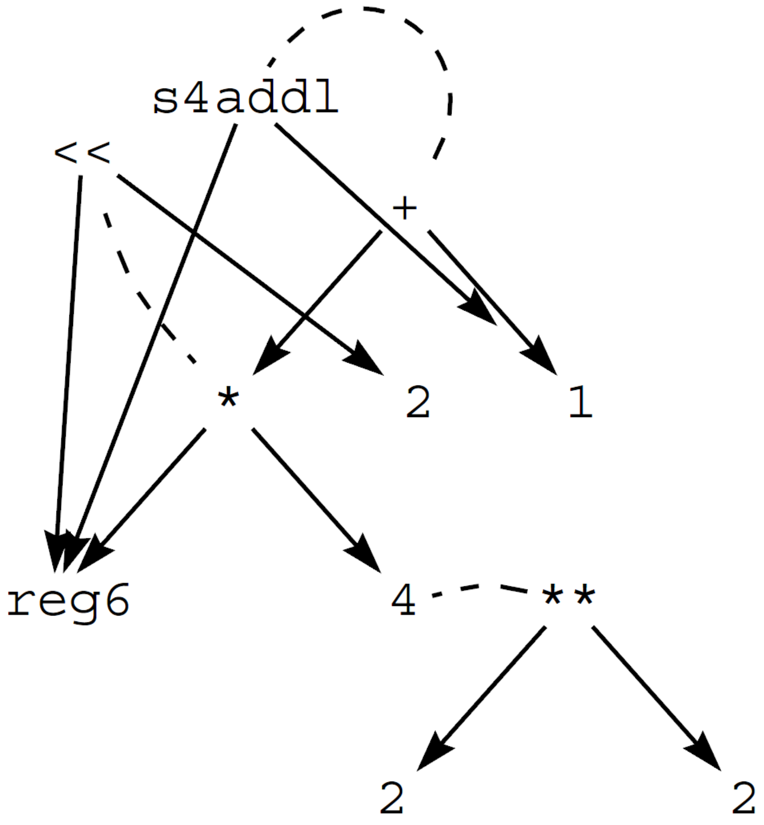
- explore many programs (one for each sequence)

Rewrite rules are need not be arbitrarily composable

- rewrite seq can get stuck (a program cannot be lowered)
- hence must backtracking

Denali: synthesis with axioms and E-graphs

[Joshi, Nelson, Randall PLDI'02]



$$\forall\, n\,.\; 2^n = 2{**}n$$

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

$$\forall k,n{::}\; k * 4 + n \;=\; \mathtt{s4addl}(k,n)$$

$\mathbf{reg6} * 4 + 1$ $\longrightarrow$ $\mathtt{s4addl}(\mathbf{reg6},1)$

specification

synthesized program

Two kinds of axioms

Instruction semantics: defines (an interpreter for) the language

$$\forall\, n\,.\ 2^n = 2{*}{*}n$$

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

$$\forall k, n{::}\ k * 4 + n\ = \texttt{s4addl}(k, n)$$

Algebraic properties: associativity of add64, memory modeling, …

$$(\forall\, x, y :: \texttt{add64}(x, y) = \texttt{add64}(y, x))$$

$$(\forall\, x, y, z :: \texttt{add64}(x, \texttt{add64}(y, z)) = \texttt{add64}(\texttt{add64}(x, y), z))$$

$$(\forall\, x :: \texttt{add64}(x, 0) = x)$$

$$(\forall\, a, i, j, x\ :: i = j$$
$$\lor\ \texttt{select}(\texttt{store}(a, i, x), j) = \texttt{select}(a, j))$$

Compilation vs. classical synthesis

Where to draw the line?*

If it searches for a good (or semantically correct) rewrite sequence, it's a synthesizer.

*We don't really need this definition but people always ask.

Modern synthesis

Interactive: it's computer-aided programming

a lot of our course will be on obtaining diagnostics about (incomplete or incorrect) programs under development

Solver-based: no (less) need for sem-preserving rules

instead, search a large space of programs that are mostly incorrect but otherwise posses programmer-specified characteristics, eg, run in log(n) steps.

how to find a correct program in this space? conceptually, we use a verifier that checks the ϕ condition.

Preparing your language for synthesis

Extend the language with two constructs

spec:

```
int foo (int x) {
    return x + x;
}
```

$\phi(x,y): y = \texttt{foo}(x)$

sketch:

```
int bar (int x) implements foo {
    return x << ??;
}
```

?? substituted with an int constant meeting ϕ

result:

```
int bar (int x) implements foo {
    return x << 1;
}
```

instead of **`implements`**, assertions over safety properties can be used

Synthesis as search over candidate programs

Partial program (sketch) defines a candidate space

we search this space for a program that meets ϕ

Usually can't search this space by enumeration

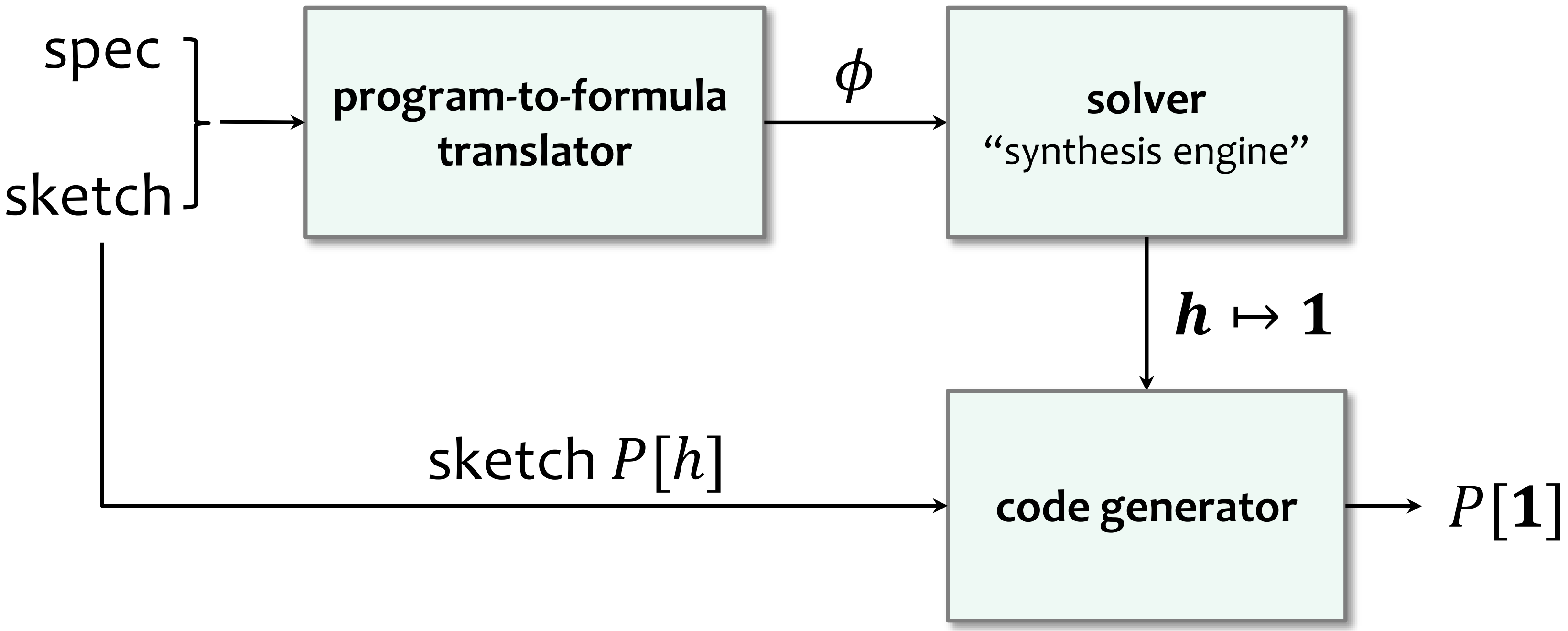
space too large ($\gg 10^{10}$)

Describe the space **symbolically**

solution to constraints encoded in a logical formula gives values of holes, indirectly identifying a correct program

What constraints? We'll cover this shortly.

Synthesis from partial programs



spec

sketch

program-to-formula translator

ϕ

solver

“synthesis engine”

$\boldsymbol{h} \mapsto \mathbf{1}$

sketch $P[h]$

code generator

$P[\mathbf{1}]$

What to do with a program as a formula?

Assume a formula $S_P(x,y)$ which holds iff program P(x) outputs value y

program: `f(x) { return x + x }`

formula: $S_f(x,y)$: $y = x + x$

This formula is created as in program verification with concrete semantics [CMBC, Java Pathfinder, …]

With program as a formula, solver is versatile

Solver as an **interpreter:** given x, evaluate f(x)

$S(x, y) \wedge x = 3$ solve for y $\boldsymbol{y} \mapsto \mathbf{6}$

Solver as a program **inverter:** given f(x), find x

$S(x, y) \wedge y = 6$ solve for x $\boldsymbol{x} \mapsto \mathbf{3}$

This solver “bidirectionality” enables synthesis

Search of candidates as constraint solving

$S_P(x, h, y)$ holds iff sketch $P[h](x)$ outputs y.

```
spec(x) { return x + x }
sketch(x) { return x << ?? }
```

$S_{sketch}(x, y, h)\colon\ y = x * 2^h$

The solver computes h, thus synthesizing a program correct for the given x (here, x=2)

$S_{sketch}(x, y, h) \wedge x = 2 \wedge y = 4$ solve for h $\boldsymbol{h} \mapsto \mathbf{1}$

Sometimes h must be constrained on several inputs

$$S(x_1, y_1, h) \wedge x_1 = 0 \wedge y_1 = 0 \wedge$$
$$S(x_2, y_2, h) \wedge x_2 = 3 \wedge y_2 = 6$$

solve for h $\boldsymbol{h} \mapsto \mathbf{1}$

Inductive synthesis

Our constraints encode **inductive synthesis:**

We ask for a program P correct on a few inputs.

We hope (or test, verify) that P is correct on rest of inputs.

Segment on Synthesis Algorithm will describe how to select suitable inputs

Why synthesis now?

Three trends in computing (during last 10-15 years)

more data => need for scalable algos

parallelism , heterogeneity

- multi-level machines (SIMD to cluster), concurrency

the Web

- distributed computation, lost messages, security

programming by non-programmers

- scientists, designers, end users

Lessons:

- We need to write programs that are more complex.
- Programming must me more accessible.

energy efficiency

Example real-world synthesizers: Spiral

Derives efficient linear filter codes (FFT, …)

exploits divide-and-conquer nature of these problems

A rewrite rule for Cooley/Tukey FFT:

$$DFT_4 = (DFT_2 \otimes I_2)\ T^4_2\ (I_2 \otimes DFT_2)\ L^4_2$$

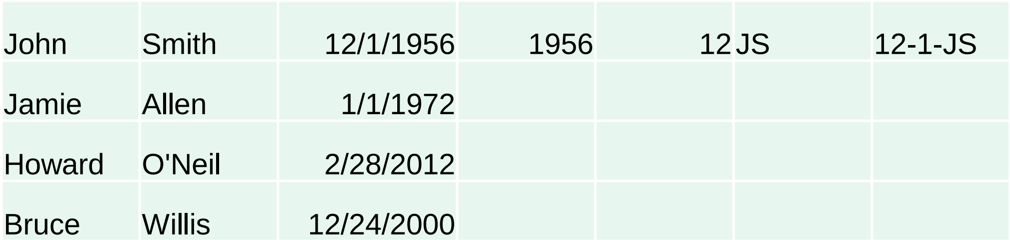
Similar rules are used to describe parallelization and locality

So, rewrite rules nicely serve three purposes: algo, para, local

http://www.spiral.net/

Example real-world synthesizer (FlashFill)

Demo



John	Smith	12/1/1956	1956	12	JS	12-1-JS
Jamie	Allen	1/1/1972				
Howard	O'Neil	2/28/2012				
Bruce	Willis	12/24/2000				

For video demos, see Sumit Gulwani’s page:
http://research.microsoft.com/en-us/um/people/sumitg/flashfill.html

What artifacts might be synthesizable?

Anything that can be viewed as a program:

reads input, produces output, could be non-terminating

Exercise: what such "programs" run in your laptop?

log structure highlighting

shopping filter

calendar planning

code convention refactorer

A liberal view of a program

networking stack

==> TCP protocol is a program ==> synthesize protocols

interpreter

==> embeds language semantics ==> languages may be synthesizable

spam filter

==> classifiers ==> learning of classifiers is synthesis

image gallery

==> compression algorithms or implementations

A liberal view of a program (cont)

file system

==> "inode" data structure

OS scheduler

==> scheduling policy

multicore processor

==> cache coherence protocol

UI

==> ???

What do we need to synthesize these?

specs

verifier (or at least a ~~verifie~~ tester)

I/O pair selections

DSL interpreter

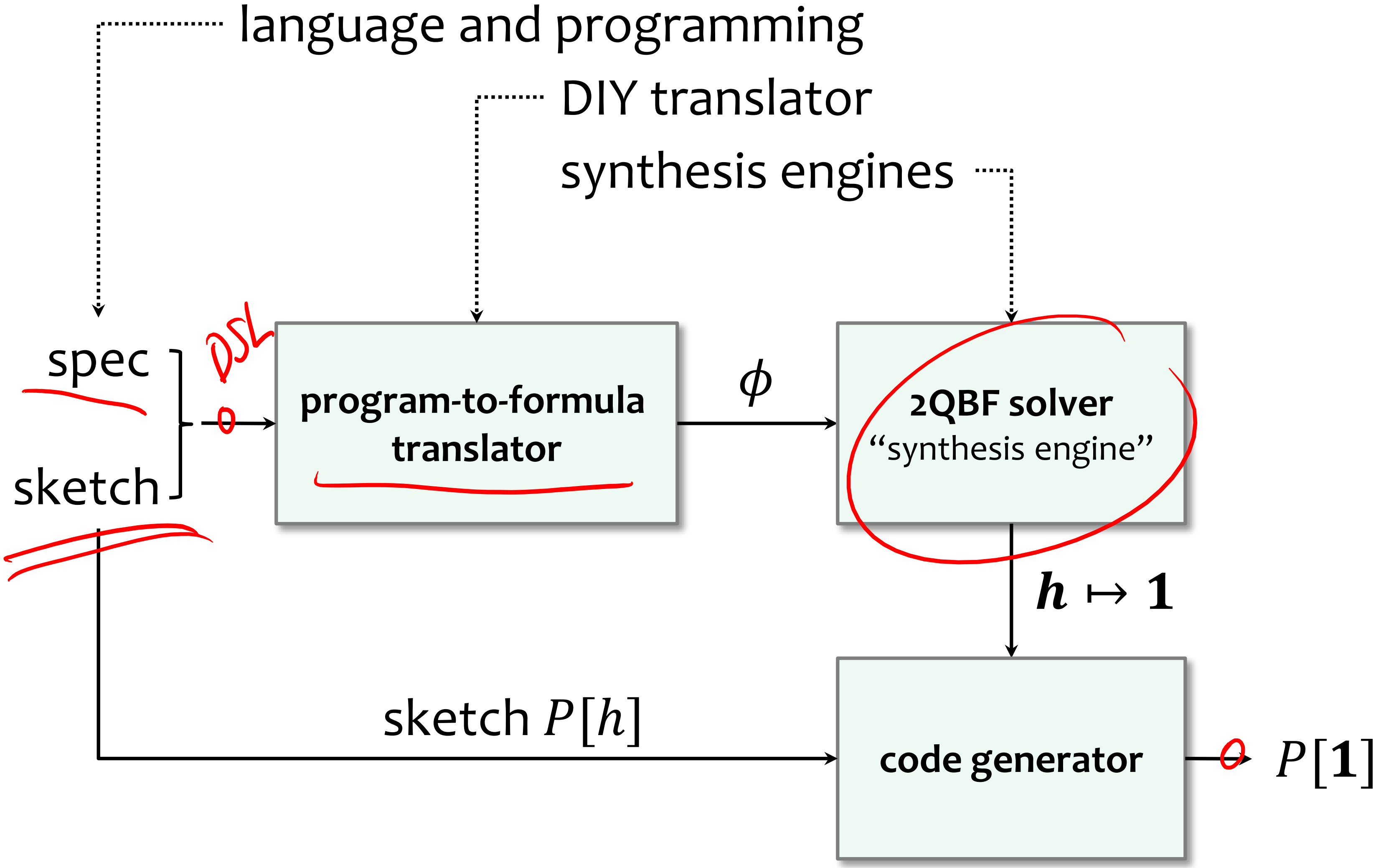
inductive synthesis

Your project

Seven milestones (each a short presentation)

- problem selection (what you want to synthesize)
- what’s your DSL (design language for your programs)
- what’s your specification (how to describe behavior)
- how to translate your DSL into logic formulas
- your synthesis algorithm
- scaling up with domain knowledge (how to sketch it)
- final posters and demos

Your project



Example of projects

Synthesis of cache-coherence protocols

Incrementalizer of document layout engines

Web scraping scripts from user demonstrations

Models of biological cells from wet-lab experiments

…

Some open synthesis problems

how should synthesizer interact with programmers

in both directions; it's psychology and language design

how to do modular synthesis?

we cannot synthesize 1M LOC at once; how to break it up?

constructing a synthesizer quickly

we'll show you how to do it in a semester; but faster would be even better. Also, how to test, maintain the synthesizer?

Homework (due in a week, Aug 30 11am)

Suggest an application for synthesis

ideally from your domain of expertise.

1) Background: Teach us about your problem.

Eg, I want to implement X but failed to debug it in 3 months

2) Problem statement:

What specific code artifact would be interesting to synthesize? Why is it hard to write the artifact by hand?

3) What are you willing to reveal to the synthesizer

That is, what's your spec? Is this spec easy to write precisely? What other info would you like to give to the synthesizer?

Next lecture

Constraint solvers can help you write programs:

Four programming problems solvable when a program is translated into a logical constraint: verification, fault-localization, angelic programming, and synthesis.

Example of a program encoding with an SMT formula (Experimenting with Z3).