Synthesizing Programs with Constraint Solvers

ASPLOS Symposium

Ras Bodik

Division of Computer Science
University of California, Berkeley

Prepare your language for synthesis

Extend the language with two constructs

spec:

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

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

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

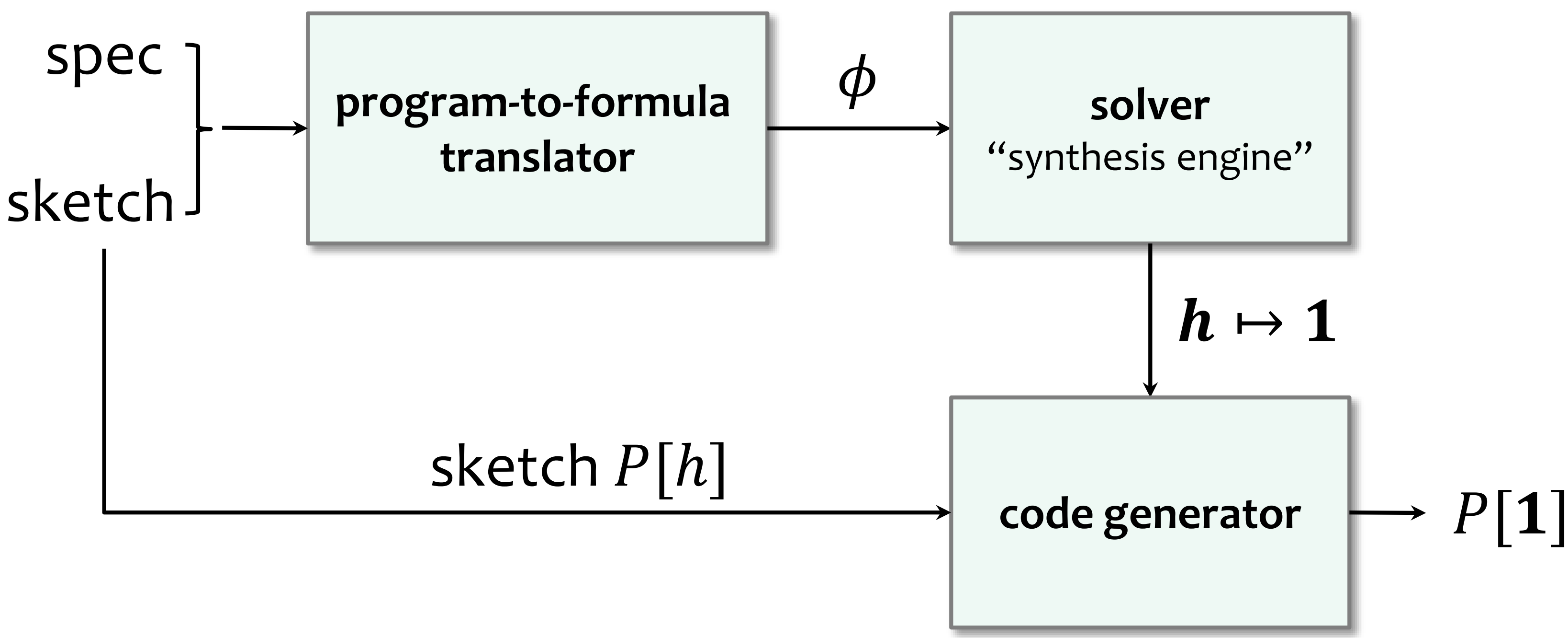
space too large ($\gg 10^{10}$)

Describe the space **symbolically**

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

What constraints? Essentially encode semantics in SAT

Synthesis from partial programs



Example: Parallel Matrix Transpose

Example: 4x4-matrix transpose with SIMD

a functional (executable) specification:

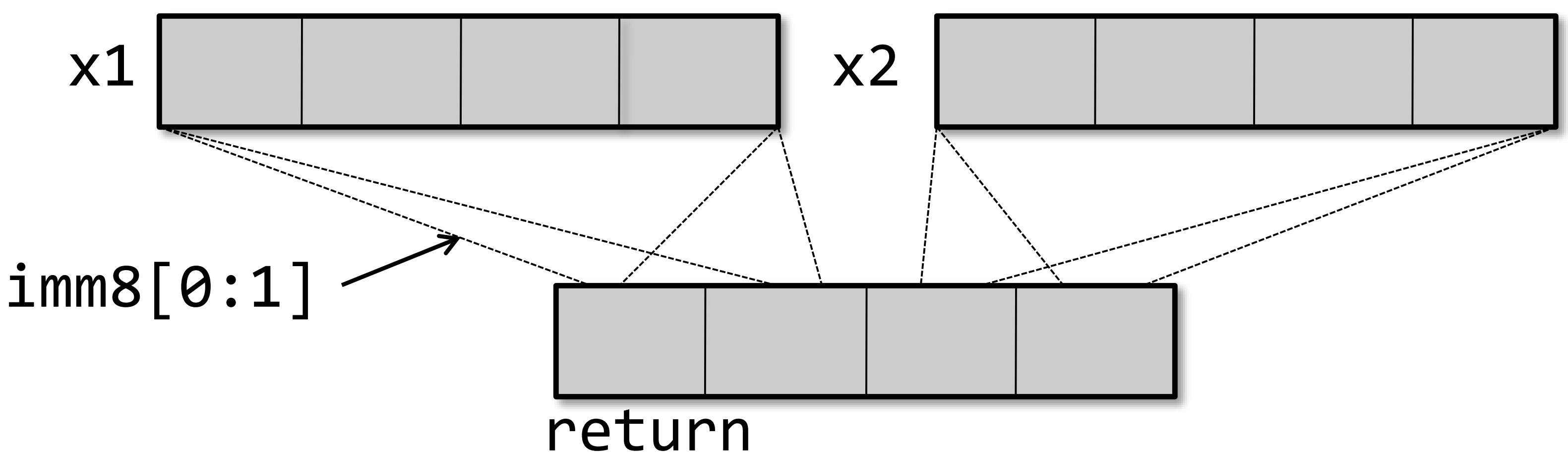
```
int[16] transpose(int[16] M) {
  int[16] T = 0;
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
      T[4 * i + j] = M[4 * j + i];
  return T;
}
```

This example comes from a Sketch grad-student contest

Implementation idea: parallelize with SIMD

Intel SHUFP (shuffle parallel scalars) SIMD instruction:

```
return = shufps(x1, x2, imm8 :: bitvector8)
```



High-level insight of the algorithm designer

Matrix M transposed in two shuffle phases

Phase 1: shuffle M into an intermediate matrix S with some number of `shufps` instructions

Phase 2: shuffle S into an result matrix T with some number of `shufps` instructions

Synthesis with partial programs helps one to complete their insight. Or prove it wrong.

The SIMD matrix transpose, sketched

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;

  S[??::4] = shufps(M[??::4], M[??::4], ??);
  S[??::4] = shufps(M[??::4], M[??::4], ??);
  …
  S[??::4] = shufps(M[??::4], M[??::4], ??);

  T[??::4] = shufps(S[??::4], S[??::4], ??);
  T[??::4] = shufps(S[??::4], S[??::4], ??);
  …
  T[??::4] = shufps(S[??::4], S[??::4], ??);

  return T;
}
```

Phase 1

Phase 2

The SIMD matrix transpose, sketched

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;
  repeat (??) S[??::4] = shufps(M[??::4], M[??::4], ??);
  repeat (??) T[??::4] = shufps(S[??::4], S[??::4], ??);
  return T;
}
```

```
int[16] trans_sse(int[16] M) implements trans { // synthesized code
  S[4::4]   = shufps(M[6::4],   M[2::4],  11001000b);
  S[0::4]   = shufps(M[11::4],  M[6::4],  10010110b);
  S[12::4]  = shufps(M[0::4],   M[2::4],  10001101b);
  S[8::4]   = shufps(M[8::4],   M[12::4], 11010111b);
  T[4::4]   = shufps(S[11::4],  S[1::4],  10111100b);
  T[12::4]  = shufps(S[3
  T[8::4]   = shufps(S[4
  T[0::4]   = shufps(S[1
}
```

From the contestant email:

Over the summer, I spent about 1/2 a day manually figuring it out.

Synthesis time: <5 minutes.

Demo: transpose on Sketch

Try Sketch online at http://bit.ly/sketch-language

Demo notes (1)

In the demo, we accelerated synthesis by changing

```
repeat(??) loop body
repeat(??) loop body
```

to

```
int steps = ??
repeat(steps) loop body
repeat(steps) loop body
```

→ can improve efficiency by adding more "insight"

here, the "insight" constraints state that both loops have same (unknown) number of iterations

Demo notes (2)

How did the student come up with the insight that two phases are sufficient?

- We don't know but the synthesizer can prove that one phase is insufficient (a one-phase sketch has no solution)

Program Synthesis with Constraint Solvers

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 $\boldsymbol{y \mapsto 6}$

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

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

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 << ?? }
```

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

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

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

Sometimes h must be constrained on several inputs

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

How to select suitable inputs?

Verify a candidate program. If it fails verification, the counterexample (input) is added as an input to synthesis

More information

Learn:

- CAV 2012 invited tutorial (with Emina Torlak)
- graduate seminar (cs294-fa12)

Play:

- SKETCH synthesizer
- Rosette lightweight synthesizer

Acknowledgements

UC Berkeley

Gilad Arnold
Shaon Barman
Prof. Ras Bodik
Prof. Bob Brayton
Joel Galenson
Thibaud Hottelier
Sagar Jain
Chris Jones
Ali Sinan Koksal
Leo Meyerovich
Evan Pu
Casey Rodarmor
Prof. Koushik Sen
Prof. Sanjit Seshia
Lexin Shan
Saurabh Srivastava
Liviu Tancau
Nicholas Tung

MIT

Prof. Armando Solar-Lezama
Rishabh Singh
Kuat Yesenov
Jean Yung
Zhiley Xu

IBM

Satish Chandra
Kemal Ebcioglu
Rodric Rabbah
Vijay Saraswat
Vivek Sarkar