CS 4830/7485
System Specification, Verification and Synthesis
Fall 2019

Program Synthesis

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

PROGRAM SYNTHESIS

The “modern” approach to program synthesis

- Interactive:
  - computer-aided programming
  - programmer solves key problems (e.g., provides program skeleton), synthesizer fills in (boring or tedious) details (e.g., missing guards/assignments)
- Search-for-patterns based:
  - synthesis = search among set of user-defined patterns
- Solver based:
  - heavily uses verifiers like SAT and SMT solvers
  - often in a counter-example guided loop

Example: programming by sketching
[Solar-Lezama, Bodik, et al.]

Parallel Parking by Sketching

Ref: Chaudhuri, Solar-Lezama (PLDI 2010)

```
Err = 0.0;
for(t = 0; t<T; t+=dT){
  if(stage==STRAIGHT){
    if(t > ??) stage= INTURN;
  }
  if(stage==INTURN){
    car.ang = car.ang - ??;
    if(t > ??) stage= OUTTURN;
  }
  if(stage==OUTTURN){
    car.ang = car.ang + ??;
    if(t > ??) break;
  }
  simulate_car(car);
  Err += check_collision(car);
}
Err += check_destination(car);
```

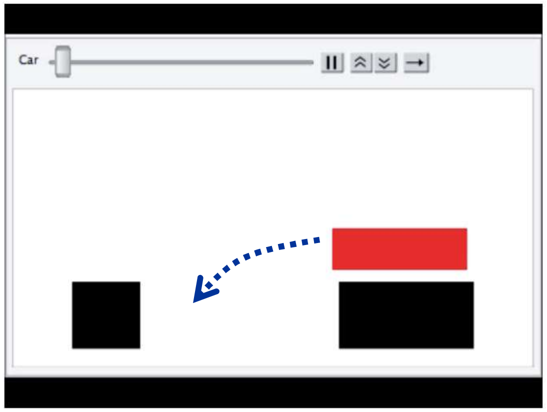
When to start turning?

Backup straight

How much to turn?

Turn

Straighten



Enables programmers to focus on high-level solution strategy

Using SAT and SMT solvers for synthesis

Recall: what is synthesis?

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

Usually re-written as:

$$\exists P : \forall x : pre(x) \rightarrow post(x, P(x))$$

i.e., if input satisfies precondition, then output will satisfy postcondition.

Using SAT and SMT solvers for synthesis

$$\exists P: \forall x: pre(x) \rightarrow post(x, P(x))$$

Example of pre(), post():

$$pre(x1, x2):\ \ number(x1) \wedge number(x2)$$

$$post(x1, x2, y):\ \ x1 \leq y \wedge x2 \leq y \wedge (x1 = y \vee x2 = y)$$

i.e., the spec for max(x1,x2).

First: using SAT and SMT solvers for <u>verification</u>

Suppose we already have a program P.

Then instead of checking whether P is **correct**

$$\forall x: pre(x) \rightarrow post(x, P(x))$$

we can check whether P is **wrong**

$$\exists x: pre(x) \wedge \neg post(x, P(x))$$

i.e., we can check **satisfiability** of the formula

$$pre(x) \wedge \neg post(x, P(x))$$

Hold on: are programs formulas?

Consider a simple loop-free program:

```
function P(int x) returns (real y)
{
  int tmp := 0;
  if (x >= 0) then {
    tmp++;
    y := tmp*x;
  }
  else
    y := -x;
  return y;
}
```

Formula:

$$P(x,y) = (x \geq 0 \wedge y = x) \vee (x < 0 \wedge y = -x)$$

Hold on: are programs formulas?

What about real programs?

Loops, data structures, libraries, pointers, threads, …

Translation to formulas much harder, but verification tools are available that do this, constantly making progress.

We will assume we have a formula *$P(x,y)$* representing the program P: *"y is the output of P for input x".*

Back to using SAT and SMT solvers for <u>verification</u>

We can check **satisfiability** of the formula

$$pre(x) \wedge \neg post(x, P(x))$$

or, writing P as predicate on both input and output variables:

$$pre(x) \wedge P(x, y) \wedge \neg post(x, y)$$

<u>Satisfiable</u> => P is wrong: we get a counter-example (x,y)

<u>Unsatisfiable</u> => P is correct (for all x)

Using SAT and SMT solvers for synthesis

What can be done when we don't have the program P ?

$$pre(x) \wedge P(x,y) \wedge \neg post(x,y)$$

Hint: what if we have a finite/small number of candidate programs?

Iterate and search!

Programs with "holes"

Almost-complete programs:

```
Err = 0.0;
for(t = 0; t<T; t+=dT){
  if(stage==STRAIGHT){
    if(t > ??) stage= INTURN;
  }
  if(stage==INTURN){
    car.ang = car.ang - ??;
    if(t > ??) stage= OUTTURN;
  }
  if(stage==OUTTURN){
    car.ang = car.ang + ??;
    if(t > ??) break;
  }
  simulate_car(car);
  Err += check_collision(car);
}
Err += check_destination(car);
```

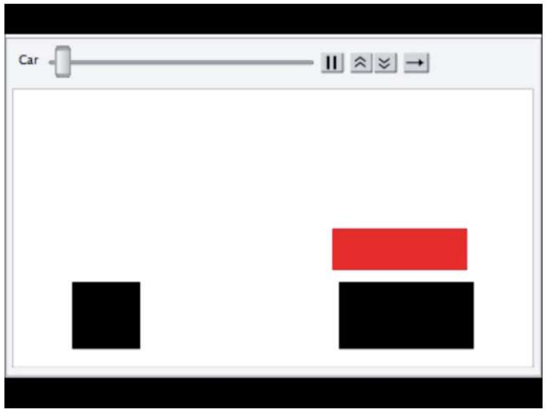
When to start turning?

Backup straight

How much to turn?

Turn

Straighten



Programs with “holes”

What should we replace “??” with?

Patterns:

- integer constants
- linear expressions of the form $ax + by + c$ where x, y are variables in the program
- …

Even with these restrictions, infinite set of candidates …

Search may take a long time or never terminate.

Can we do better?

Asking the solver to find the program

Suppose our program has 1 hole, to be filled with an integer variable.

Then, the formula characterizing the program becomes

$$P(h, x, y)$$

Can we use the solver to find the right h ?

Check satisfiability of

Free variable: solver must find right value

$$\forall x, y: pre(x) \land P(h, x, y) \rightarrow post(x, y)$$

Problem: universal quantification …

$$\forall x, y: pre(x) \land P(h, x, y) \rightarrow post(x, y)$$

Today's solvers check satisfiability of quantifier-free formulas (mostly).

What can we do about that?

Hint: what if we have a finite number of **positive examples**? i.e., I/O pairs (x, y) satisfying $pre(x) \land post(x, y)$.

Example-guided synthesis

Suppose we have a finite number of positive examples, say 2: $(x_1, y_1), (x_2, y_2)$.

That is: we know that these hold:

$$pre(x_1), pre(x_2), post(x_1, y_1), post(x_2, y_2)$$

So it suffices to check satisfiability of

$$P(h, x_1, y_1) \wedge P(h, x_2, y_2)$$

Example-guided synthesis

In general, for n positive examples and k hole variables:

$$\bigwedge_{i=1}^{n} P(h_1, h_2, \dots, h_k, x_i, y_i)$$

We turned universal quantification into finite conjunction!

Example-guided synthesis

What if solver finds this formula unsatisfiable ?

$$\bigwedge_{i=1}^{n} P(h_1, h_2, \ldots, h_k, x_i, y_i)$$

Unsatisfiable => no program exists!

This is **sound**: if no program exists that works even in this finite set of examples, we cannot hope to find a program that works for all examples.

Example-guided synthesis

What if solver finds this formula satisfiable ?

$$\bigwedge_{i=1}^{n} P(h_1, h_2, \dots, h_k, x_i, y_i)$$

Satisfiable => $P(h_1, h_2, \dots, h_k)$ is only a candidate.

It still needs to be verified for **all** I/O pairs.

We can again use the solver for that!

Example-guided synthesis

$$\bigwedge_{i=1}^{n} P(h_1, h_2, \dots, h_k, x_i, y_i)$$

Satisfiable => $P(h_1, h_2, \dots, h_k)$ is only a candidate.

Verify it by checking satisfiability of

$$pre(x) \wedge P(\underbrace{h_1, h_2, \dots, h_k}_{\text{These are now fixed}}, x, y) \wedge \neg post(x, y)$$

If formula is unsatisfiable then we are done!

What if formula is satisfiable?

Our candidate is wrong. We get a counter-example: (x^*, y^*)

What then?

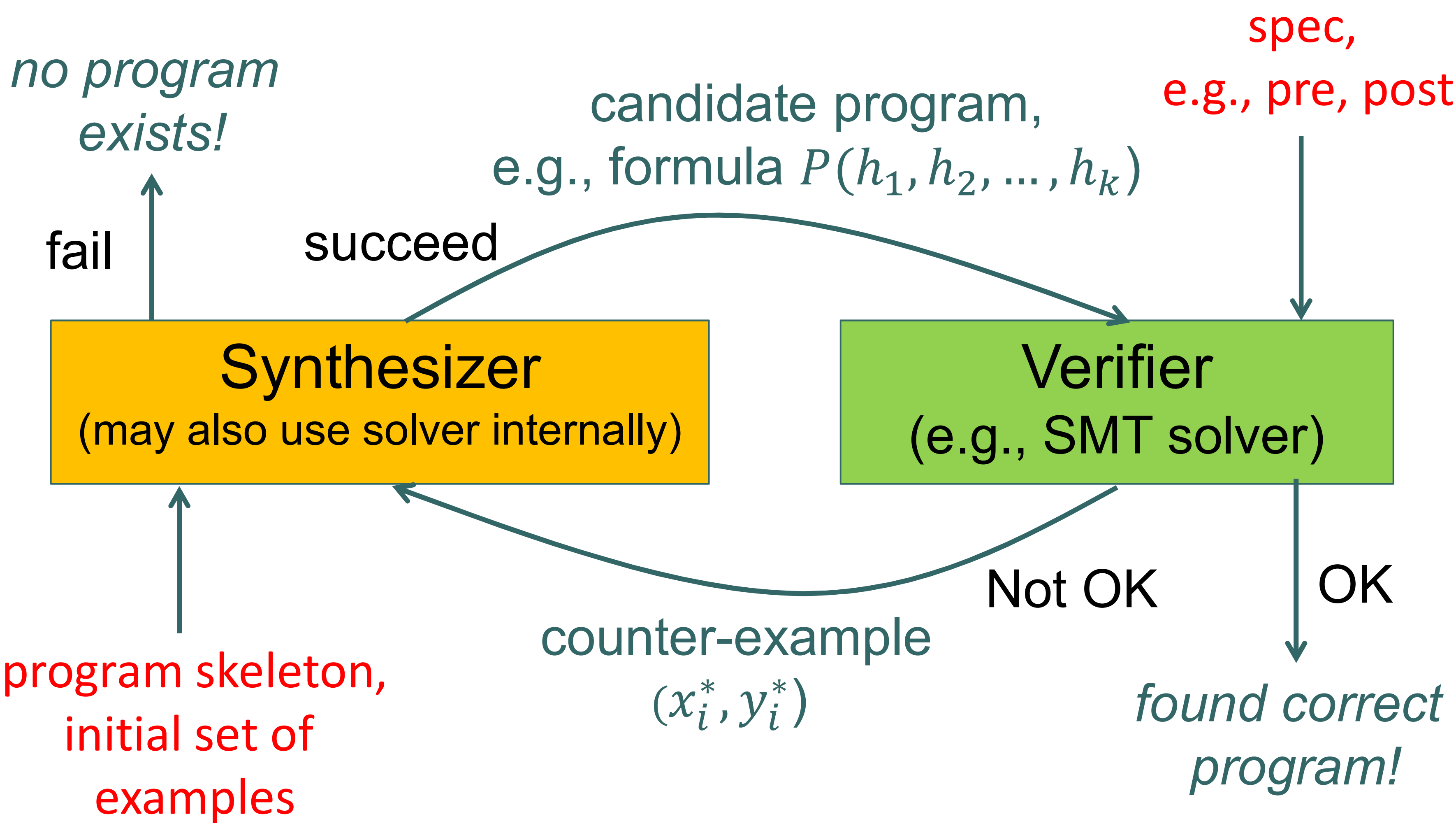
Adding negative examples to the synthesizer's inputs

In general, for n positive examples, m negative examples, and k hole variables:

$$\bigwedge_{i=1}^{n} P(h_1, h_2, \dots, h_k, x_i, y_i) \wedge \bigwedge_{i=1}^{m} \neg P(h_1, h_2, \dots, h_k, x_i^*, y_i^*)$$

Alternative: the user could provide the correct output for the counter-example input, or we could use a reference (correct and deterministic) program.

Counter-example guided synthesis



References

1. Solar-Lezama. *Program sketching*. STTT Vol 15, Issue 5-6, Oct 2013.
2. Alur, Bodik, et al. *Syntax-Guided Synthesis*. FMCAD 2013.
3. International Journal on Software Tools for Technology Transfer, Special Issue on Synthesis, Volume 15, Issue 5-6, October 2013.
4. Course by Ras Bodik and Emina Torlak. CS294 – *Program Synthesis for Everyone*. https://homes.cs.washington.edu/~bodik/ucb/cs294fa12.html