This lecture: “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) {
    \text{Backup straight}
  }
  if(stage==INTURN) {
    car.ang = car.ang - \theta;
    if(t > ??) stage=OUTTURN;
  }
  if(stage==OUTTURN) {
    car.ang = car.ang + \theta;
    if(t > ??) break;
  }
  simulate_car(car);
  Err += \text{check_collision}(car);
}
Err += \text{check_destination}(car);
```

Enables programmers to focus on high-level solution strategy

Using SAT and SMT solvers for synthesis

Recall: what is synthesis?

\[ \exists P: \forall x: \varphi(x, P(x)) \]

A more detailed version:

\[ \exists P: \forall x: \text{pre}(x) \rightarrow \text{post}(x, P(x)) \]
Using SAT and SMT solvers for synthesis

\[ \exists P: \forall x: \text{pre}(x) \rightarrow \text{post}(x, P(x)) \]

Example of pre(), post():

\[ \text{pre}(x_1, x_2): \text{number}(x_1) \land \text{number}(x_2) \]
\[ \text{post}(x_1, x_2, y): x_1 \leq y \land x_2 \leq y \land x_1 = y \lor x_2 = y \]

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

First: using SAT and SMT solvers for verification

Suppose we already have a program \( P \).

Then instead of checking whether \( P \) is correct

\[ \forall x: \text{pre}(x) \rightarrow \text{post}(x, P(x)) \]

we can check whether \( P \) is wrong

\[ \exists x: \text{pre}(x) \land \neg \text{post}(x, P(x)) \]

i.e., we can check \textbf{satisfiability} of the formula

\[ \text{pre}(x) \land \neg \text{post}(x, P(x)) \]
Hold on: are programs formulas?

Consider a simple loop-free program:

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

Formula:

\[ P(x, y) = (x \geq 0 \land y = x) \lor (x < 0 \land y = -x) \]

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 verification

We can check **satisfiability** of the formula

\[ pre(x) \land \neg post(x, P(x)) \]

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

\[ pre(x) \land P(x, y) \land \neg post(x, y) \]

Satisfiable \( \Rightarrow \) \( P \) is wrong: we get a **counter-example** \((x, y)\)

Unsatisfiable \( \Rightarrow \) \( P \) is correct (for all \( x \))

Using SAT and SMT solvers for synthesis

What can be done when we don’t have the program \( P \) ?

\[ pre(x) \land P(x, y) \land \neg post(x, y) \]

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

Iterate!
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?
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** …
Iteration 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

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

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
\[
P(h, x_1, y_1) \land P(h, x_2, y_2)
\]

Example-guided synthesis

In general, for \(n\) positive examples and \(k\) hole variables:
\[
\bigwedge_{i=1}^{n} P(h_1, h_2, \ldots, h_k, x_i, y_i)
\]

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?

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

Satisfiable => \( P(h_1, h_2, \ldots, h_k) \) 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

\[ \bigwedge_{i=1}^{n} P(h_1, h_2, ..., h_k, x_i, y_i) \]

Satisfiable \(\Rightarrow\) \(P(h_1, h_2, ..., h_k)\) is only a candidate.
Verify it by checking satisfiability of

\[ pre(x) \land P(h_1, h_2, ..., h_k, x, y) \land \neg post(x, y) \]

These are now fixed

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:

\[ \bigwedge_{i=1}^{n} P(h_1, h_2, ..., h_k, x_i, y_i) \land \bigwedge_{i=1}^{m} \neg P(h_1, h_2, ..., h_k, x_i^*, y_i^*) \]

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

- **Synthesizer** (may also use solver internally)
- **Verifier** (e.g., SMT solver)

- Candidate program, e.g., formula \( P(h_1, h_2, \ldots, h_k) \)
- Spec, e.g., pre, post
- Program skeleton, initial set of examples
- Counter-example \((x_i, y_i)\)
- OK
- Found correct program!
- Not OK
- No program exists!
- Fail
- Succeed

References