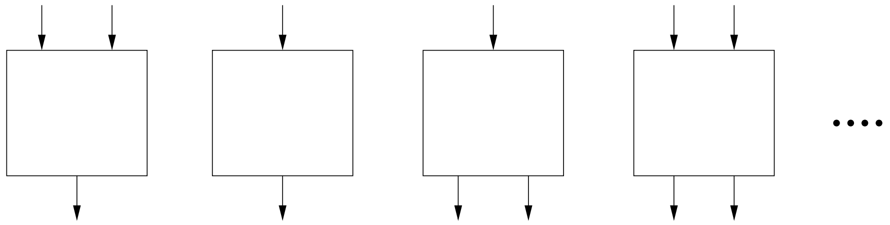
Synthesizing from Components:

Building from Blocks

Ashish Tiwari
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025

Joint work with Sumit Gulwani (MSR), Vijay Anand Korthikanti (UIUC), Susmit Jha (UC Berkeley), Sanjit Seshia (UC Berkeley), Thomas Sturm (Munich), Ankur Taly (Stanford), Ramarathnam Venkatesan (MSR)

Component-Based Synthesis



Problem: How to wire the components to synthesize a desired system ?

Concrete Examples

Desired System $F_{\texttt{spec}}$	Components f_i's
sort an array	comparators
compute $\frac{x+y}{2}$	modulo arithmetic ops
find rightmost one	bitwise ops, arithmetic ops
compute x^{243}	multiplication
accept ω-regular language	Buchi automata
safe hybrid system	multiple operating modes
geometry construction	ruler-compass steps
deobfuscated code	parts of obfuscated code
verification proof	verification inference rules

Question: $\exists C : \forall x : F_{\texttt{spec}}(x) = C(f_1, f_2, \ldots)(x)$

Synthesis Problem Classes

“This is difficult”

“This is ill posed”

“This is too general to be solvable”

$$\exists C : \forall x : F_{\mathtt{spec}}(x) = C(f_1, f_2, \ldots)(x)$$

Parameters that define the synthesis problem:

- composition operator C
- class of specifications $F_{\mathtt{spec}}$
- class of component specifications f_i

Fixing the synthesis problem:
fix these parameters, fix representation of $F_{\mathtt{spec}}, f_i$

Bounded Synthesis

The synthesis problem is still hard

We make it feasible by replacing the unbounded quantifier, $\exists C$, by a bounded quantifier

$$\exists C : \forall x : F_{\mathtt{spec}}(x) = C(f_1, f_2, \ldots)(x)$$

$$\Downarrow$$

$$\exists c : \forall x : F_{\mathtt{spec}}(x) = c(f_1, f_2, f_3)(x), c \text{ in some finite set}$$

This bounded synthesis problem is solved by deciding the $\exists\forall$ formula

Straight-Line Program Synthesis

composition operator	function composition
components	primitive functions
system	complex function

Bounded synthesis version:

- fix length of program
- fix upper bound on number of each component

$\exists P : \forall x : F_{\mathtt{spec}}(x) = P(x), \quad P$ a straight-line program composing f_i's

$$\Downarrow$$

$$\exists \pi : \forall x : F_{\mathtt{spec}}(x) = f_{\pi(1)}(f_{\pi(2)}(f_{\pi(3)}(x)))$$

Example: Straight-Line Program Synthesis

Specification: Evaluate polynomial $a * h^2 + b * h + c$

Budget: two multiplication and two addition operators

Finite search space

Synthesized Program:

1. $o_1 := a * h$;
2. $o_2 := o_1 + b$;
3. $o_3 := o_2 * h$;
4. return $o_3 + c$;

Correctness: $(a * h + b) * h + c = a * h^2 + b * h + c$

Example: Straight-Line Program Synthesis

Specification: Turn-off rightmost contiguous 1 bits

Example: 010101100 ↦ 010100000

Budget: two addition and at most four bitwise Boolean operators

Finite search space: Also need some constants

Synthesized Program:

1. $o_1 := x + (-1)$;
2. $o_2 := o_1 | x$;
3. $o_3 := o_2 + 1$;
4. return $o_3 \& x$;

Correctness on sample input:

$$010101100 \mapsto 010101011 \mapsto 010101111 \mapsto 010110000 \mapsto 010100000$$

Loop-free Program Synthesis

composition operator	function composition
components	primitive functions, if-then-else
system	complex function

Bounded synthesis version:

- fix length of program
- fix upper bound on number of each component including if-then-else

$$\exists P : \forall x : F_{\mathtt{spec}}(x) = P(x), \quad P \text{ a straight-line program composing } f_i\text{'s}$$

$$\Downarrow$$

$$\exists \pi : \forall x : F_{\mathtt{spec}}(x) = f_{\pi(\epsilon)}(f_{\pi(1)}(f_{\pi(11)}(x_1), f_{\pi(12)}(x_2, x_1)))$$

Example: Loop-free Program Synthesis

Specification: Obfuscated code

Example: We are given

```
if (h(x))
   if (x*(x+1)% 2 == 1) y := f(x) else y := g(x)
else y := f(g(x))
```

Components Budget: f, g, h, if-then-else

Synthesized Program:

```
o := g(x);
if (h(x)) y := o; else y := f(o);
```

Correctness: Equivalence of two loop-free programs

Loop-free Program Synthesis

$$\exists \pi : \forall x : F_{\texttt{spec}}(x) = f_{\pi(\epsilon)}(f_{\pi(1)}(f_{\pi(11)}(x_1), f_{\pi(12)}(x_2, x_1)))$$

Enumerate all possible programs and check

Enumerate all permutations π and check

Checking if a synthesized program is the desired program is a verification problem

Bounded Synthesis := iteratively perform verification

But we can learn from failures ...

$\exists\forall\phi$ Solvers

Bounded Synthesis $\mapsto$ $\exists\forall$ solving

How to solve $\exists u : \forall x : \phi$ formulas?

A1 Counter-example guided iterative solver

A2 Distinguishing input solver

- Applies even when ϕ not fully known

A3 Numerical solver

A1: Solving $\exists\forall\phi$

Counter-example guided iterative procedure for solving $\exists\vec{u} : \forall\vec{x} : \phi(\vec{u},\vec{x})$

1. Guess $\vec{u}_0$ for $\vec{u}$
2. (Verification) Check if

$$\forall\vec{x} : \phi(\vec{u}_0,\vec{x})$$

3. If true, then return $\vec{u}_0$
4. Get counterexample $\vec{x}_0$, add it to X
5. (Finite Synthesis) Find new $\vec{u}_0$ such that

$$\exists\vec{u}_0 : \bigwedge_{\vec{x}_0\in X} \phi(\vec{u}_0,\vec{x}_0)$$

6. Go to Step 2

A1: Counter-example Guided Iterative $\exists\forall$ Solving

Needs a backend quantifier-free solver

That can return counterexamples

We use an SMT solver

The structure of ϕ, and additional knowledge about what ϕ encodes, is used optimize the above procedure to expedite convergence

Related Work: Sketch, Aha

Reference: Synthesis of loop-free programs, PLDI 2011

A2: Distinguishing Input Solver

Solving $\exists \vec{u} : \forall \vec{x} : \phi(\vec{u}, \vec{x})$

1. X := some finite set of choices for $\vec{x}$
2. Find two programs that work for X, but differ on some $\vec{x}_0$

$$\exists \vec{u}_1, \vec{u}_2, \vec{x}_0 : \left(\bigwedge_{\vec{x} \in X} (\phi(\vec{u}_1, \vec{x}) \ \wedge \ \phi(\vec{u}_2, \vec{x})) \right) \ \wedge \ (\phi(\vec{u}_1, \vec{x}_0) \ \not\Leftrightarrow \ \phi(\vec{u}_2, \vec{x}_0))$$

3. If satisfiable, we add $\vec{x}_0$ to X and go to (2)
4. If unsatisfiable, then find one program that works for X

$$\exists \vec{u}_1 : \bigwedge_{\vec{x} \in X} \phi(\vec{u}_1, \vec{x})$$

5. If satisfiable, return $\vec{u}_1$
6. Otherwise, return “not synthesizable”

A2: Properties of the A2 Solver

The second algorithm for solving $\exists \vec{u} : \forall \vec{x} : \phi(\vec{u}, \vec{x})$

- Does not need the full specification of the desired program
- We only need the knowledge of the specification on the set X
- Does not perform the verification step

An interative implementation of A2:

1. Tool asks user for the expected output on input $\vec{x}_0$
2. Tool synthesizes internally two programs that work correctly for $X := \{\vec{x}_0\}$, but differ on input $\vec{x}_1$
3. Tool asks user for the expected output on input $\vec{x}_1$
4. Add $\vec{x}_1$ to X and repeat

A3: Nonsymbolic $\exists\forall$ Solver

A third algorithm for solving $\exists\vec{u} : \forall\vec{x} : \phi(\vec{u}, \vec{x})$

1. Find finite set X of input-output pairs of the specification
2. Synthesize program that works for finite set X
3. Verify the synthesized program on randomly sampled inputs

We solved Step (2) using an SMT solver previously

We can avoid the SMT solver and instead

1. hierarchical program synthesis: first synthesize high-level components
2. enumerate composition of high-level components guided by goal

Example: Synthesis Without Symbolic Reasoning

Specification: Construct a triangle, given its base, a base angle and sum of the other two sides.

Components: Ruler compass constructions

Formal specification: Given points p_1, p_2 and numbers a, r, find point p

$$\begin{aligned} \phi_{pre} &:= r > \texttt{length}(p_1, p_2) \\ \phi_{post} &:= \texttt{Angle}(p, p_1, p_2) = a \ \wedge \ \texttt{length}(p, p_1) + \texttt{length}(p, p_2) = r \end{aligned}$$

Construction:

L1 := ConstructLineGivenAngleLine(L,a);

C1 := ConstructCircleGivenPointLength(p1,r);

(p3,p4) := LineCircleIntersection(L1,C1);

L2 := PerpendicularBisector2Points(p2,p3);

p5 := LineLineIntersection(L1,L2);

Example: Geometry Construction Synthesis

Step 1 find concrete input-output pair consistent with specification

$$\begin{aligned} L &= \texttt{Line}(\langle 81.62, 99.62\rangle, \langle 99.62, 83.62\rangle) \\ r &= 88.07 \\ a &= 0.81 \text{ radians} \end{aligned}$$

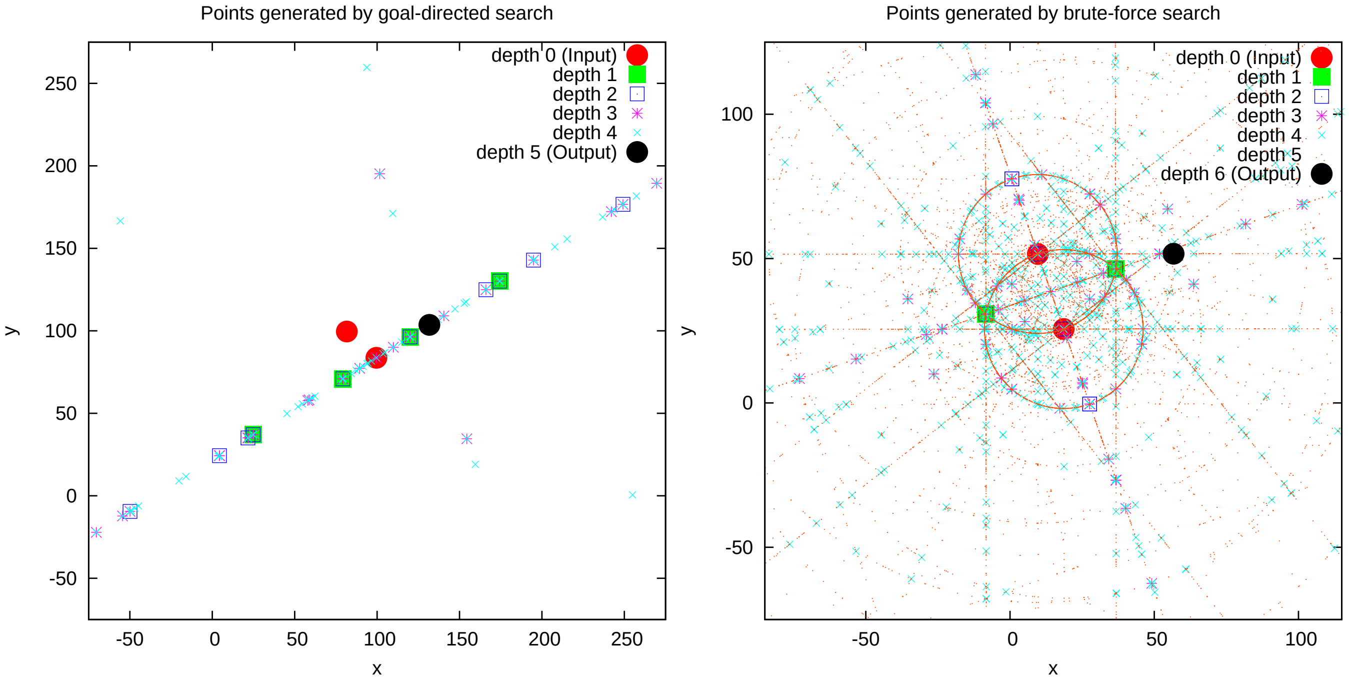
Compute output for this input: $p := \langle 131.72, 103.59\rangle$

Step 2 Start enumerating partial programs built using an extended library

Step 3 Evaluate if intermediate objects generated by the partial program are good and try other choices in Step (2) otherwise

Geometry Construction Synthesis

Evaluating effect of making search goal directed



Points visited in a goal-directed search (left) and a brute-force search (right).

Geometry Construction Synthesis

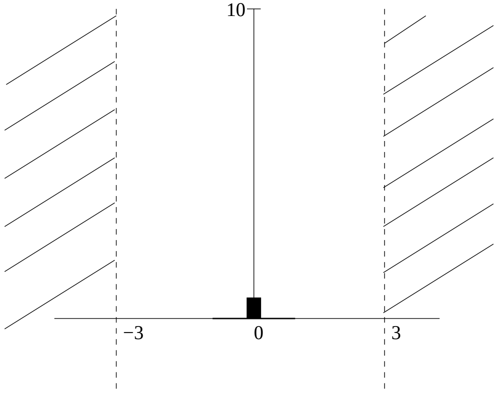
- Extended library is forward search
  - Encodes knowledge / concept taught in class
- Goal directness is backward search
  - Corresponds to reasoning student expected to do
- Sample input-output points generated using numerical techniques

Switching Logic Synthesis

Given a multimodal dynamical system

Synthesize conditions for switching between modes such that some requirements are met

Example: Driving a Robot



The goal is to drive the robot starting from `Init` to `Reach` while remaining inside `Safe`:

$$\begin{aligned}\texttt{Init} &:= (x \in [-1,1],\ y=0,\ v_x=0,\ v_y=0)\\ \texttt{Reach} &:= (y \geq 10)\\ \texttt{Safe} &:= (|x| \leq 3)\end{aligned}$$

Using the 2 modes:

- Mode 1: Force applied in $(1,1)$-direction

$$\frac{dx}{dt} = v_x, \quad \frac{dv_x}{dt} = 1 - v_x, \quad \frac{dy}{dt} = v_y, \quad \frac{dv_y}{dt} = 1 - v_y$$

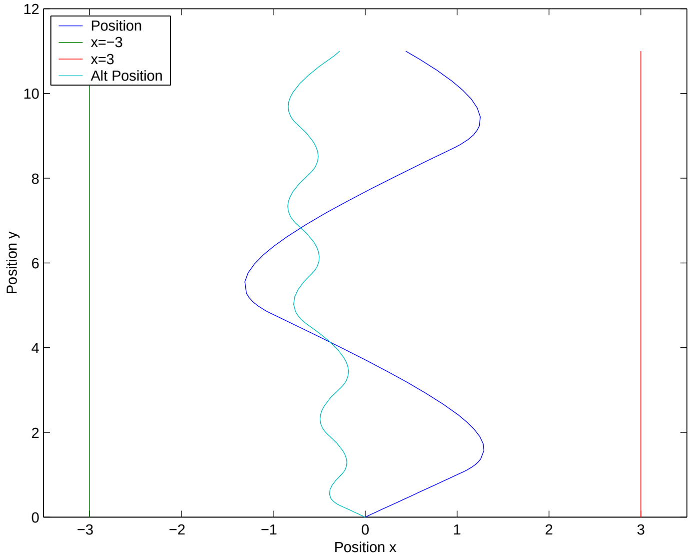
- Mode 2: Force applied in $(-1,1)$-direction

$$\frac{dx}{dt} = v_x, \quad \frac{dv_x}{dt} = -1 - v_x, \quad \frac{dy}{dt} = v_y, \quad \frac{dv_y}{dt} = 1 - v_y$$

Example: Driving a Robot

We synthesize a non-deterministic controller: a set of different possible switchings that each satisfy the requirement `Safe`U`Reach`.

Two possible trajectories:



How to discover the correct switching logic?

Switching Logic Synthesis

$\exists$switching conditions : $\forall$state variables : correctness

We can again bound the search for switching conditions

But that is a bad solution

Need to go back to verification

Verification Techniques

1. Reachability-Based Verification
2. Abstraction-Based Verification
3. Certificate-Based Verification

Key Observation: Verification = searching for right certificate

Property	Witness/Certificate
Stability	Lyapunov function
Safety	Inductive Invariant
Liveness	Ranking function

Certificate-Based Verification

Verifying property P in system S :=

$$\exists C : \text{C is a certificate for } P \text{ in } S$$

Can do a bounded search for C

Also known as the constraint-based approach

Certificates for Synthesis Problem:

Property	Witness/Certificate
Safety	Controlled Inductive Invariant
Stability	Controlled Lyapunov function

Bounded Synthesis of Switching Logic

Given multimodal dynamical system, and property `Safe`:

- Guess templates for the certificate for controlled-safety
- Generate the $\exists a, b, \ldots : \forall x, y, \ldots : \phi$
- Solve the formula to get values for $a, b, \ldots$

$\exists\forall$ Solvers

Need $\exists u : \forall x : \phi$ solvers for the reals

We can use the same ideas as before

- Symbolic Numeric Approach:
  - Symbolic: A combination of QEPCAD, redlog, slfq to eliminate inner $\forall$
  - Numeric: Gradient descent to find u from resulting formula
- Iterative learning: Iteratively prune out u values based on simulations

Conclusion

- Synthesis: $\exists\forall$ solving
- Bounded synthesis: Make problem tractable by making $\exists$ a finite quantification
- Component-based Synthesis
- Various approaches to solve $\exists\forall$ depending on application
- Switching logic synthesis : search for controlled certificates