Constraint-based Search, Part II

CS560: Reasoning About Programs

Roopsha Samanta

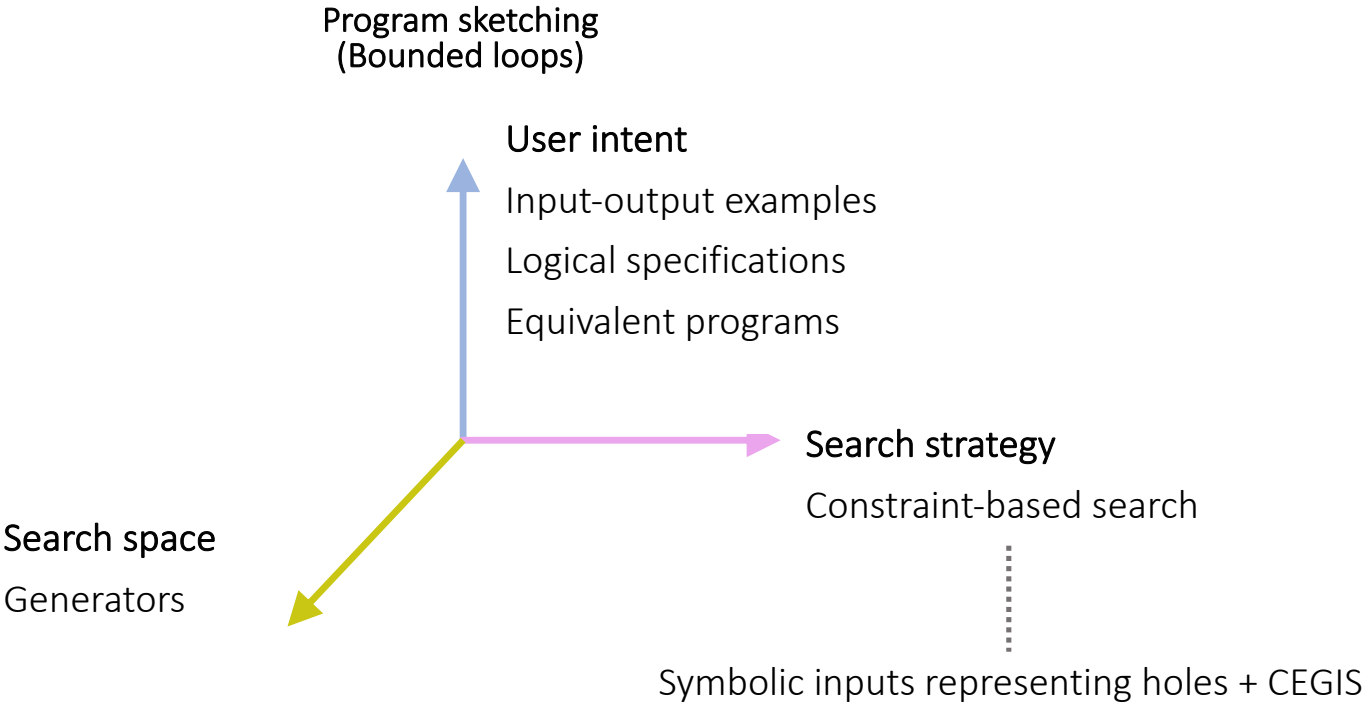
Roadmap

Previously

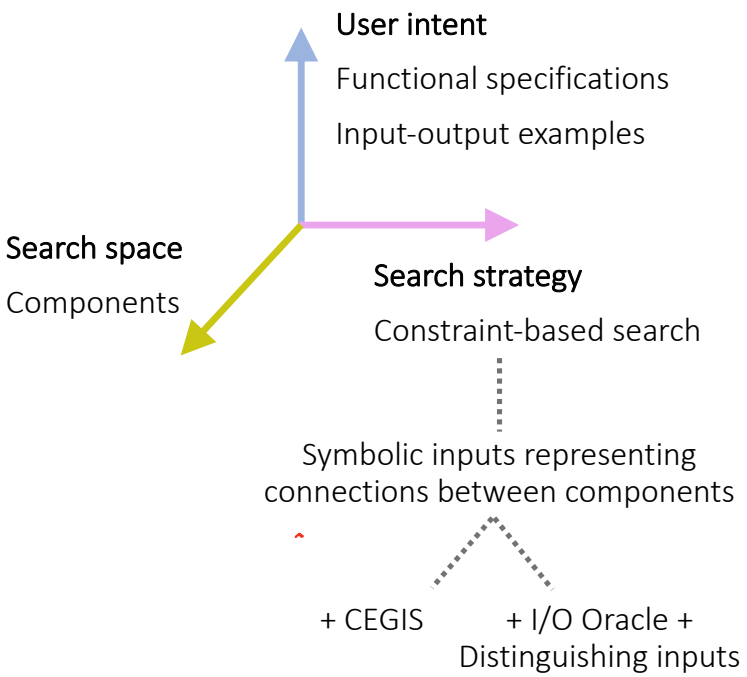
- Constraint-based search: sketching

Today

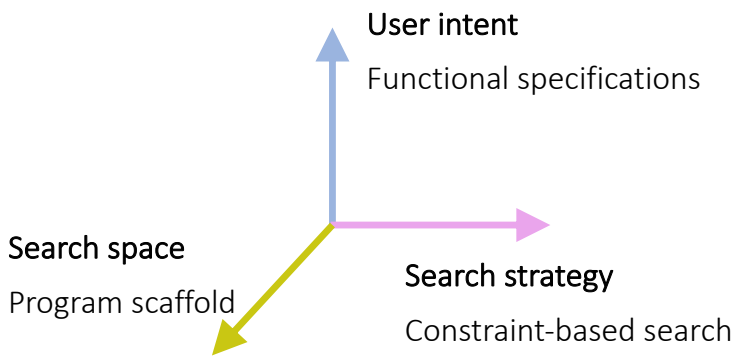
- Constraint-based search: component-based synthesis


Program sketching
(Bounded loops)
User intent
Input-output examples
Logical specifications
Equivalent programs
Search strategy
Constraint-based search
Symbolic inputs representing holes + CEGIS
Search space
Generators

Component-based synthesis (Loop-free programs)

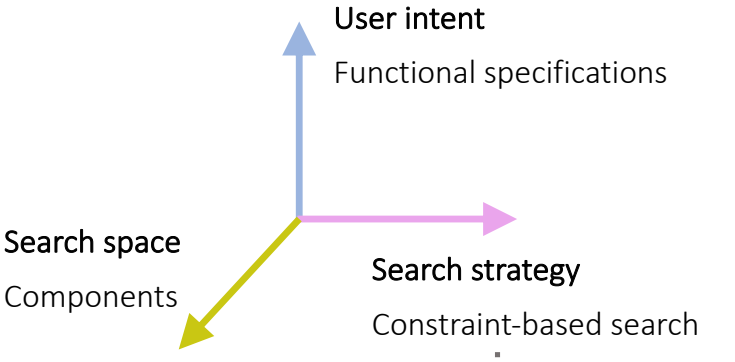


Proof-based synthesis (Loopy programs)



Extension of invariant generation to program generation.

Component-based synthesis (Loop-free programs)



[GJTV11]

Symbolic inputs representing connections between components

+ CEGIS

Example: Least significant zero bit

```
int W = 32;

bit[W] isolate0 (bit[W] x) {
        bit[W] ret = 0;
        for (int i = 0; i < W; i++)
              if (!x[i]) { ret[i] = 1; return ret;  }
}
```

0010 0101 → 0000 0010

0010 0000 → 0000 0001

Trick for an optimized program:

Adding 1 to a string of ones turns the next zero to a one:

000111 + 1 = 001000

Example: Least significant zero bit

```
bit[W] isolateSk (bit[W] x) implements isolate0 {

       return !(x + ??) & (x + ??) ;
}
```

Sketch 1

```
generator bit[W] gen(bit[W] x, int bnd){
    assert bnd > 0;
    if(??) return x;
    if(??) return ??;
    if(??) return !gen(x, bnd-1);
    if(??){
      return {| gen(x, bnd-1) (& | +) gen(x, bnd-1) |}; }
}
```

```
bit[W] isolate0Sk (bit[W] x) implements isolate0 {
     return gen(x, 3);
}
```

Sketch 2

User still needs to write a program sketch

Example: Least significant zero bit

```
int W = 32;

bit[W] isolate0 (bit[W] x) {
        bit[W] ret = 0;
        for (int i = 0; i < W; i++)
              if (!x[i]) { ret[i] = 1; return ret;  }
}
```

0010 0101 → 0000 0010

0010 0000 → 0000 0001

$$\phi(I,O) \coloneqq \bigwedge_{t=1}^{W}\left(\left(I[t]=0 \wedge \bigwedge_{j=0}^{t-1} I[j]=1\right) \Rightarrow O[t]=1 \wedge \bigwedge_{j\neq t} O[j]=0\right)$$

Functional specification

Component-based synthesis

$$\phi(I,O) \coloneqq \bigwedge_{t=1}^{W}\left(\left(I[t]=0 \wedge \bigwedge_{j=0}^{t-1} O[j]=1\right) \Rightarrow O[t]=1 \wedge \bigwedge_{j\neq t} O[j]=0\right)$$

Functional specification

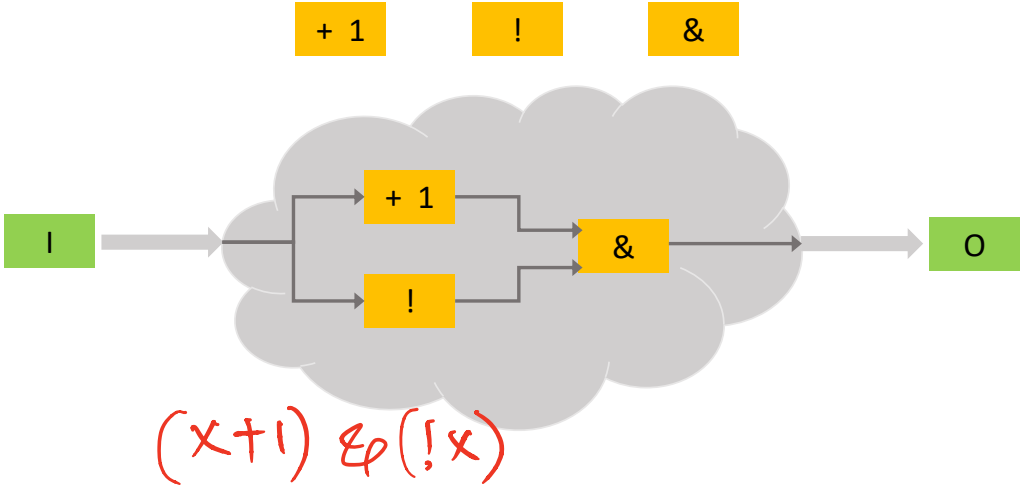
+

A library of components
+
functional specifications ϕ_{Comp} for components

Find a program that implements $\phi(I,O)$ using only given components

Component-based synthesis

$$\phi(I,O) \coloneqq \bigwedge_{t=1}^{W}\left(\left(I[t]=0 \wedge \bigwedge_{j=0}^{t-1} O[j]=1\right) \Rightarrow O[t]=1 \wedge \bigwedge_{j\neq t} O[j]=0\right)$$



Functional specification

+

A library of components
+
functional specifications ϕ_{Comp} for components

Find a program that implements $\phi(I,O)$ using only given components

Component-based synthesis

Enables hierarchical synthesis!
User can extend library by specifying a synthesized program as a new component in the extended library.

- Component-Based Synthesis for Complex APIs. POPL, 2017
- Look for the Proof to Find the Program: Decorated-Component-Based Program Synthesis. CAV, 2017
- Component-Based Synthesis of Table Consolidation and Transformation Tasks from Examples. PLDI, 2017
- Jungloid Mining: Helping to Navigate the API Jungle. PLDI, 2005

Functional specification

+

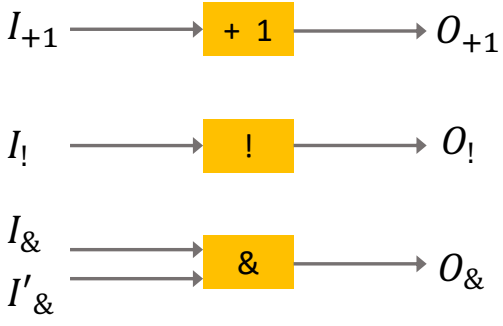
A library of components
+
Functional specifications for components

Find a program that implements $\phi(I, O)$ using only given components

SMT encoding?

SMT encoding: component semantics

Lib: component library
T_{in}: ∪(component inputs)
T_{out}: ∪(component outputs)



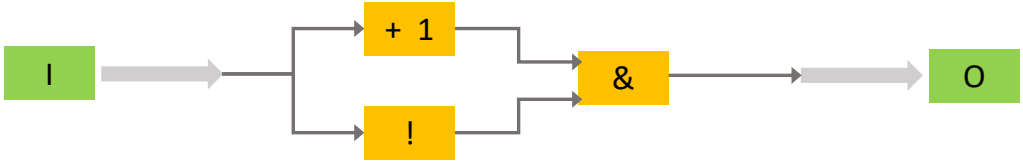
$\phi_{+1}(I_{+1}, O_{+1})$: $O_{+1} = I_{+1} + 1$

$\phi_!(I_!, O_!)$: $O_! = \neg I_!$

$\phi_\&(I_\&, I'_\&, O_\&)$: $O_\& = I_\& \wedge I'_\&$

$\phi_{Lib}(T_{in}, T_{out})$

SMT encoding: syntactic well-formedness

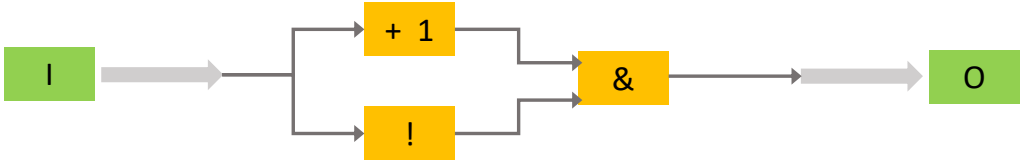



```
f(I):
1. O_1 := f+1 (I);
2. O_2 := f! (I);
3. O_3 := f& (O_1, O_2);
return O_3;
```

To obtain program, we need to find:

- which component goes into which location
- from which location or program input does component get input arguments

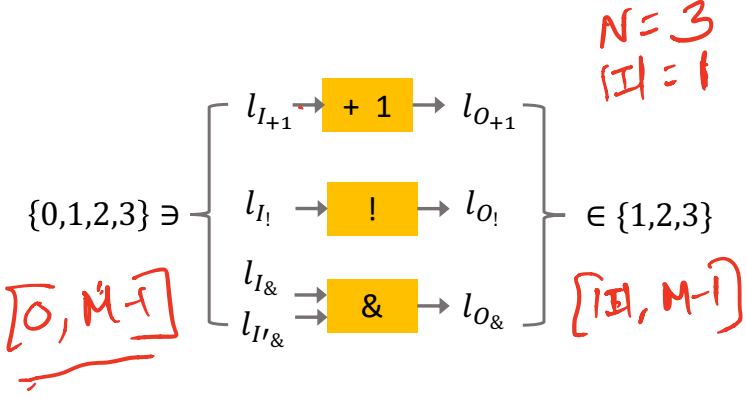
SMT encoding: syntactic well-formedness




```
f(I):
1. O_1 := f+1 (I);
2. O_2 := f! (I);
3. O_3 := f& (O_1, O_2);
return O_3;
```

Use **location variables** for
every input and output of components!

SMT encoding: syntactic well-formedness



If x is an output variable for component c, l_x is the location where c goes.

If x is an input variable for component c, l_x is the location from where c get its input.

```
f(I):
1. O_1 := f+1 (I);
2. O_2 := f! (I);
3. O_3 := f& (O_1, O_2);
return O_3;
```

$$l_{O_{+1}} = 1 \quad l_{I_{+1}} = 0$$
$$l_{O_!} = 2 \quad l_{I_!} = 0$$
$$l_{O_\&} = 3 \quad l_{I_\&} = 1 \quad l_{I'_\&} = 2$$

SMT encoding: syntactic well-formedness

L : set of location variables
Lib: component library
N: number of components
M: $N + |I|$
T_{in}: ∪(component inputs)
T_{out}: ∪(component outputs)
T: $T_{in} \cup T_{out}$

$$\psi_{wfp}(L)$$

$$\bigwedge_{x,y \in T_{out},\, x \neq y} l_x \neq l_y \quad \wedge$$

Every program location has exactly one component

$$\bigwedge_{c \in Lib,} \; \bigwedge_{x \in I_c,\, y = O_c} l_x < l_y \quad \wedge$$

Every variable is initialized before use

$$\bigwedge_{x \in T_{in}} 0 \leq l_x \leq M - 1 \quad \wedge \quad \bigwedge_{x \in T_{out}} |I| \leq l_x \leq M - 1$$

SMT encoding: dataflow semantics

L : set of location variables
Lib: component library
N: number of components
M: $N + |I|$
T_{in}: ∪(component inputs)
T_{out}: ∪(component outputs)
T: $T_{in} \cup T_{out}$

$$\psi_{Conn} : \bigwedge_{x,y \in I \cup O \cup T} l_x = l_y \rightarrow x = y$$

Relating the input/output variables of the components and the program

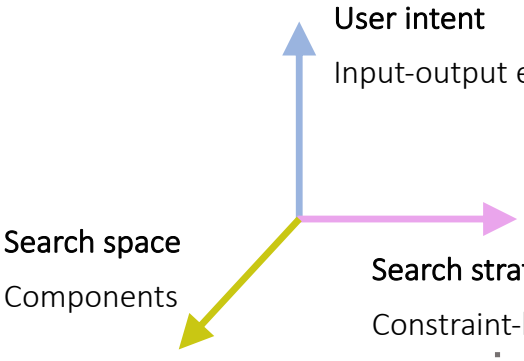
SMT encoding: synthesis constraint

$$\exists L, T. \forall I, O.\ \ \psi_{wfp}(L) \wedge \ \left(\phi_{Lib}(T) \wedge \psi_{Conn}(I, O, T, L) \rightarrow \phi(I, O)\right)$$

Solve using CEGIS

Quadratic in the number of components

Component-based synthesis (Loop-free programs)



User intent

Input-output examples

Search space

Components

Search strategy

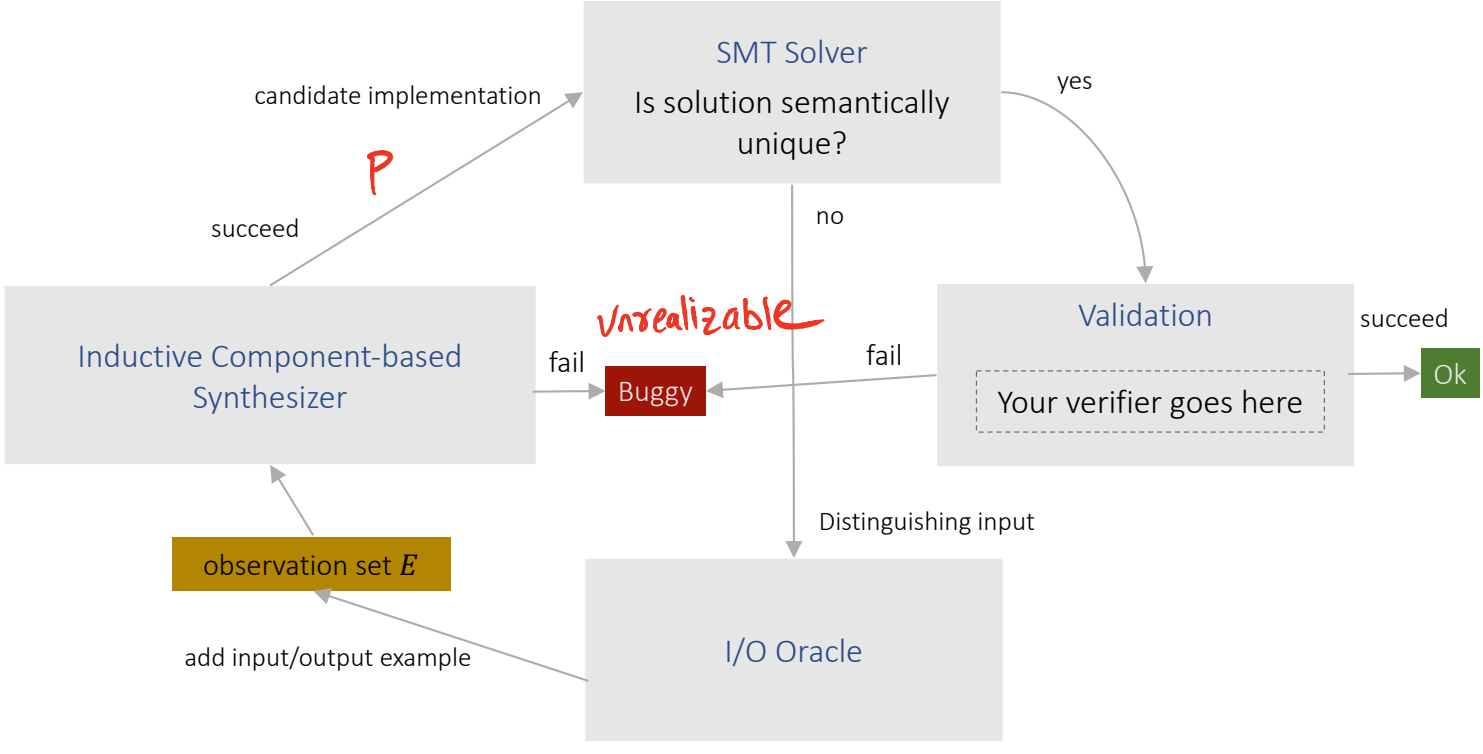
Constraint-based search

Symbolic inputs representing connections between components

[JGST10]

+ I/O Oracle +
Distinguishing inputs

P is incorrect $\Rightarrow \exists x.\ P(x)$ is incorrect.
If synthesis problem is realizable, there exists P' such that $P'(x)$ is correct



Summary

Today

- Constraint-based search: component-based synthesis

Next

- Stochastic search