

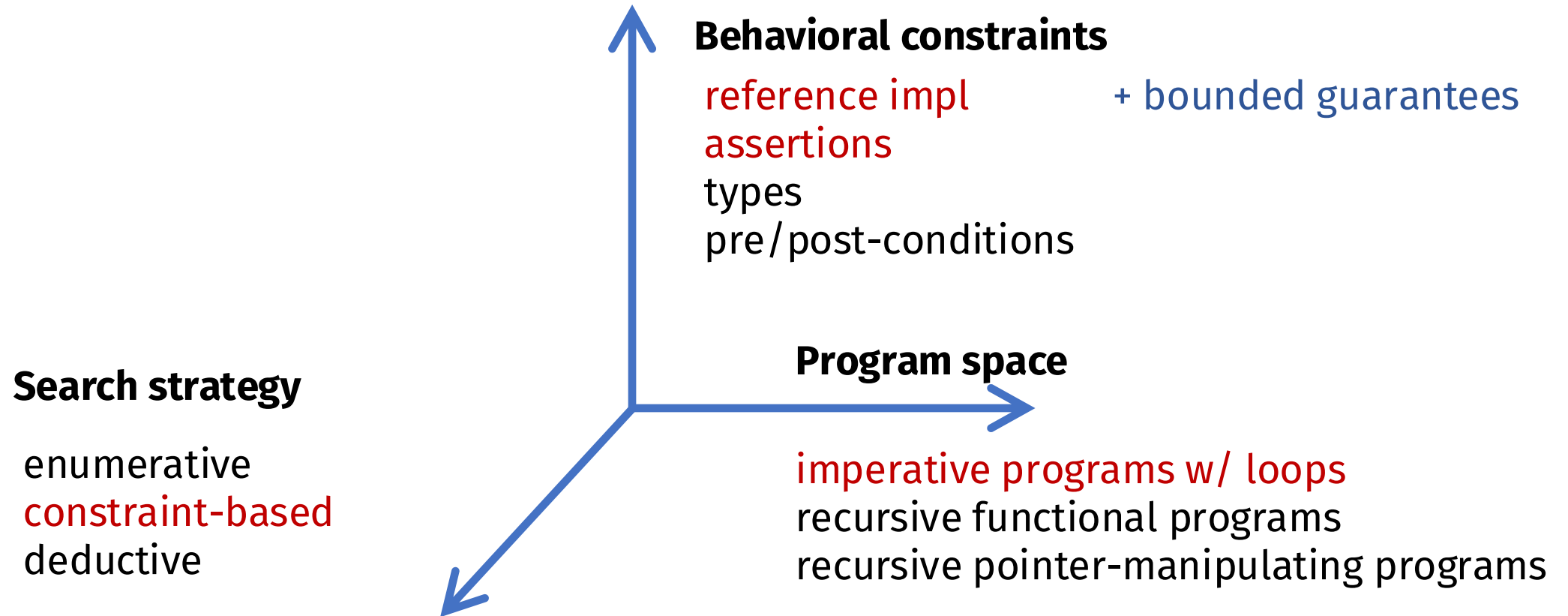
# #19: Program Sketching and CEGIS

**Sankha Narayan Guria**

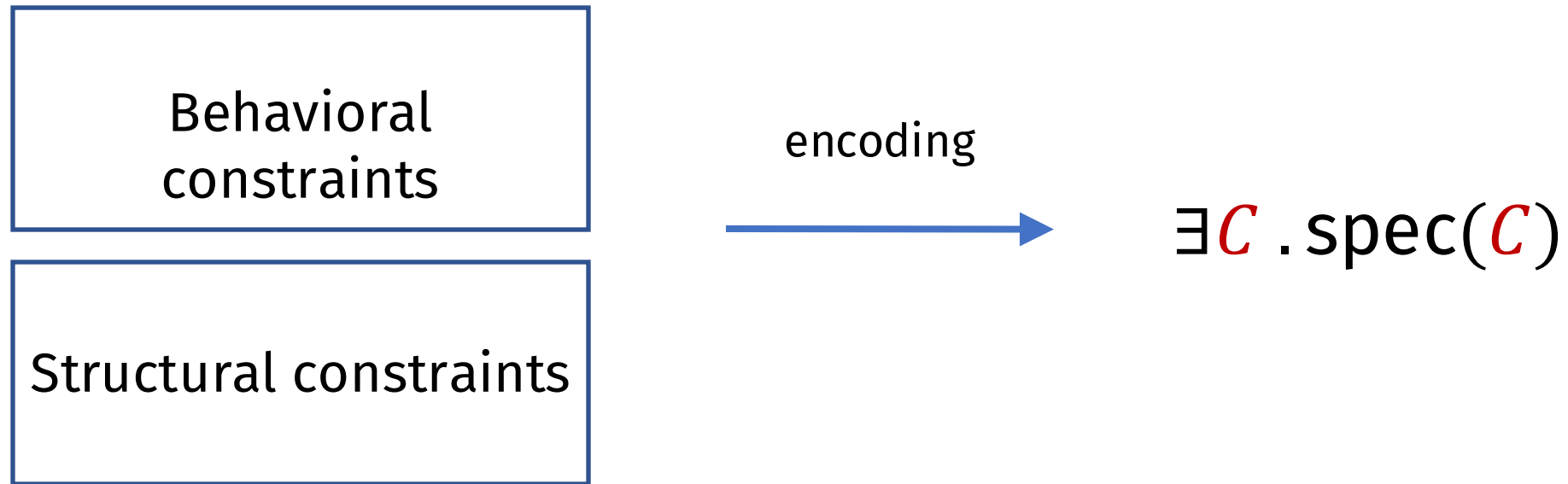
EECS 700: Introduction to Program Synthesis



# Program Sketching



# Constraint-based synthesis



# CBS for complex programs

## 2. How to encode the behavior of complex programs?

Behavioral constraints  
= assertions / reference  
implementation

Structural constraints

encoding

$\exists C . \text{spec}(C)$

## 1. How to specify for complex programs?

## 3. How to solve for complex specs?

# Program Sketching

## 2. How to encode the behavior of complex programs?

Symbolic execution

encoding

$\exists C . \text{spec}(C)$

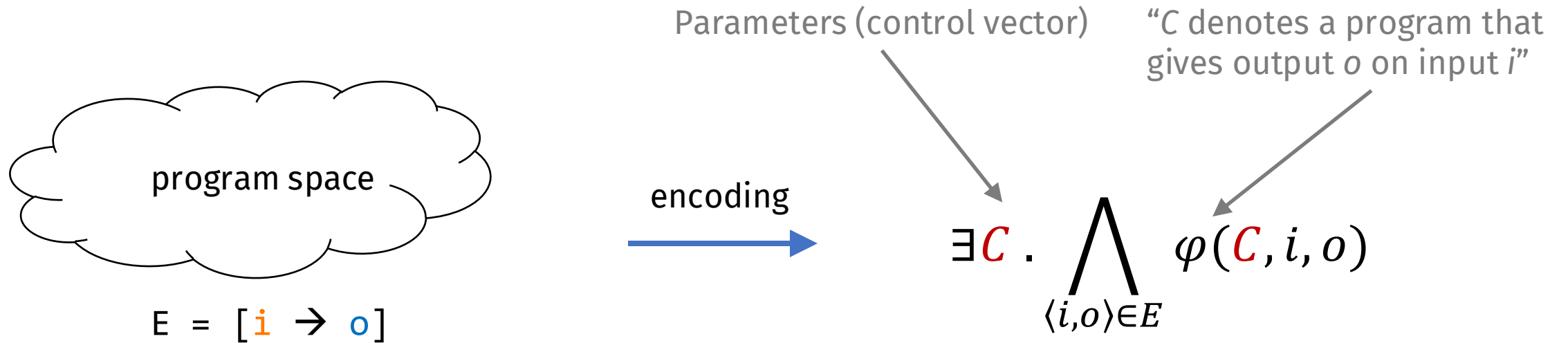
## 3. How to solve for complex specs? CEGIS

Behavioral constraints  
= assertions / reference  
implementation

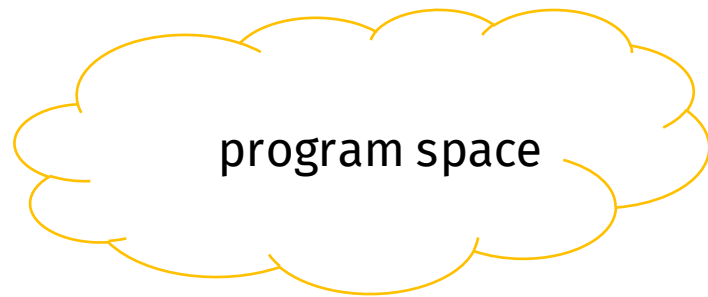
Structural constraints

## 1. How to specify for complex programs? Sketches

# CBS from examples



# CBS from specifications



$$E = [i \rightarrow o]$$

$$\forall i \, o . \psi(i, o)$$

encoding



$$\exists \mathcal{C} . \bigwedge_{\langle i, o \rangle \in E} \varphi(\mathcal{C}, i, o)$$

" $\mathcal{C}$  denotes a program that gives output  $o$  on input  $i$ "

$$\boxed{\exists \mathcal{C} . \forall i \, o . \varphi(\mathcal{C}, i, o) \Rightarrow \psi(i, o)}$$

doubly-quantified constraint:  
not solver-friendly

# Example

```
harness void main(int x) {  
  int y := ?? * x + ??;  
  assert y - 1 == x + x;  
}
```

encoding

$$\exists C . \forall i \ o . \varphi(C, i, o) \Rightarrow \psi(i, o)$$

$$\begin{aligned} \exists c_1 c_2 . \forall x \ y . y &= c_1 * x + c_2 \\ &\Rightarrow y - 1 = x + x \end{aligned}$$

simplify

$$\exists c_1 c_2 . \forall x . c_1 * x + c_2 - 1 = x + x$$

How do we solve this constraint?



# CEGIS

$$\exists c . \forall x . Q(c, x)$$

- **Idea 1: Bounded Observation Hypothesis**

- Assume there exists a small set of inputs  $X = \{x_1, x_2, \dots, x_n\}$  such that whenever  $c$  satisfies

$$\bigwedge_{i \in 1..n} Q(c, x_i)$$


it also satisfies

$$\forall x . Q(c, x)$$

← No quantifiers here, can give to SAT / SMT

# Example

This is a linear constraint,  
two inputs are enough!



$$\exists c_1 c_2 . \forall x . c_1 * x + c_2 - 1 = x + x$$

$$X = \{0, 1\}$$

$$Q(c_1, c_2, 0) \equiv c_2 - 1 = 0$$

$$Q(c_1, c_2, 1) \equiv c_1 + c_2 - 1 = 2$$

$$\{c_1 \rightarrow 2, c_2 \rightarrow 1\}$$

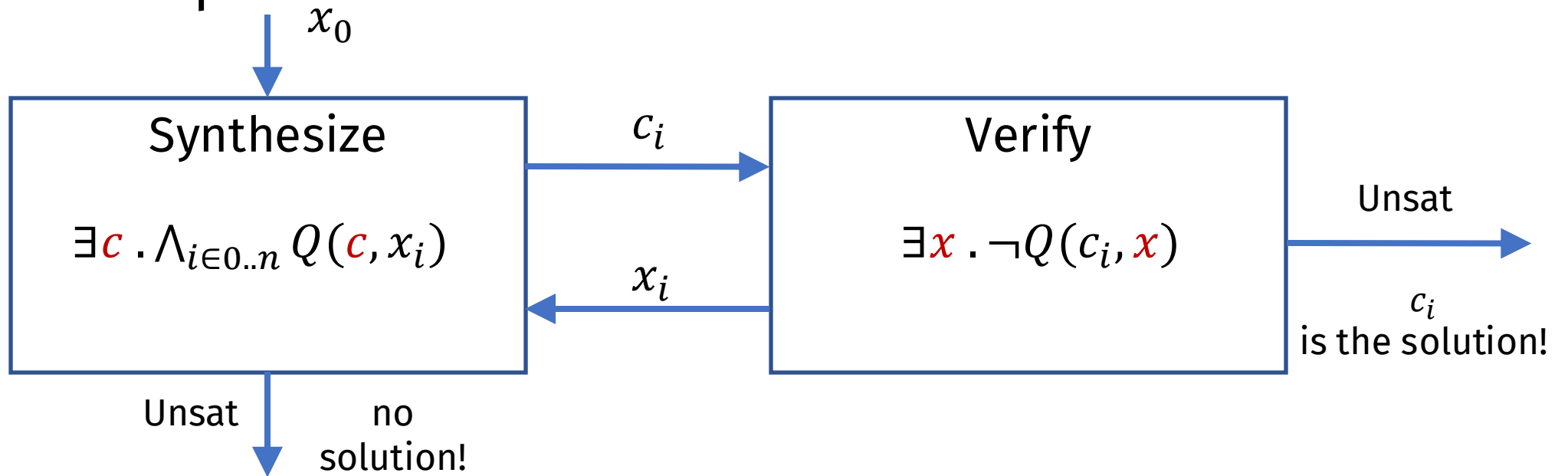
```
harness void main(int x) {  
    int y := 2 * x + 1;  
    assert y - 1 == x + x;  
}
```

How do we find X in a general case?

# CEGIS

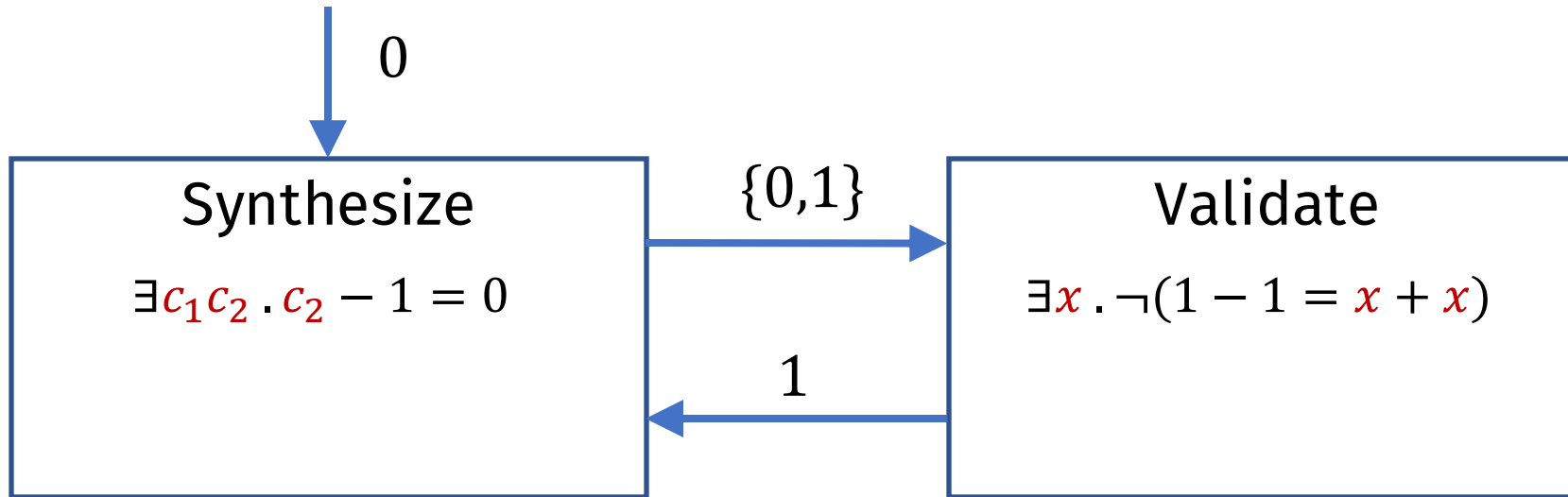
$$\exists c . \forall x . Q(c, x)$$

- **Idea 2:** Rely on verification oracle to generate counterexamples



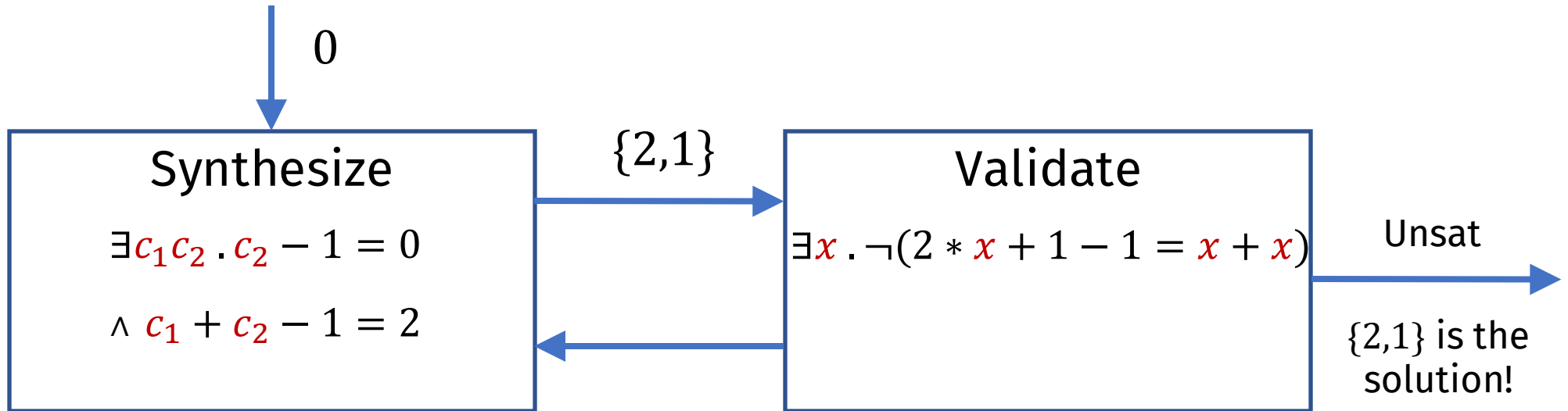
# Example

$$\exists c_1 c_2 . \forall x . c_1 * x + c_2 - 1 = x + x$$



# Example

$$\exists c_1 c_2 . \forall x . c_1 * x + c_2 - 1 = x + x$$



# Program Sketching

## 2. How to encode the behavior of complex programs?

Symbolic execution

encoding

Behavioral constraints  
= assertions / reference  
implementation

Structural constraints

## 1. How to specify for complex programs?

Sketches

$$\exists C . \forall x . Q(C, x)$$

## 3. How to solve for complex specs?

CEGIS



# Structural constraints in Sketch

- Different constraints good for different problems
  - CFGs
  - Components
  - Just figure out the constants
- **Idea:** Allow the programmer to encode all kinds of constraints using... programs (duh!)

# Language Design Strategy

Extend base language with one construct

Constant hole: ??

```
int bar (int x)
{
    int t = x * ??;
    assert t == x + x;
    return t;
}
```



```
int bar (int x)
{
    int t = x * 2;
    assert t == x + x;
    return t;
}
```

Synthesizer replaces ?? with a natural number



# Constant holes $\rightarrow$ sets of expressions

- Expressions with **??** == sets of expressions

- linear expressions

$x * ?? + y * ??$

- polynomials

$x * x * ?? + x * ?? + ??$

- sets of variables

$?? \ ? \ x : y$

# Example: swap without a temporary

- Swap two integers without an extra temporary

```
void swap(ref int x, ref int y){  
    x = ... // sum or difference of x and y  
    y = ... // sum or difference of x and y  
    x = ... // sum or difference of x and y  
}
```

```
harness void main(int x, int y){  
    int tx = x; int ty = y;  
    swap(x, y);  
    assert x==ty && y == tx;  
}
```

# Syntactic sugar

- `{| RegExp |}`
- RegExp supports choice `|` and optional `?`
  - can be used arbitrarily within an expression
    - to select operands `{| (x | y | z) + 1 |}`
    - to select operators `{| x (+ | -) y |}`
    - to select fields `{| n(.prev | .next)? |}`
    - to select arguments `{| foo( x | y, z) |}`
- Set must respect the type system
  - all expressions in the set must type-check
  - all must be of the same type

# Complex program spaces

- **Idea:** To build complex program spaces from simple program spaces, borrow abstraction devices from programming languages
- Function: abstracts expressions
- Generator: abstracts set of expressions
  - Like a function with holes...
  - ...but different invocations → different code

# Example: swap without a temporary

```
generator int sign() {  
    if ?? {return 1;} else {return -1;}  
}
```

```
void swap(ref int x, ref int y){  
    x = x + sign()*y;           ➔ 1  
    y = x + sign()*y;           ➔ -1  
    x = x + sign()*y;           ➔ -1  
}
```

```
harness void main(int x, int y){  
    int tx = x; int ty = y;  
    swap(x, y);  
    assert x==ty && y == tx;  
}
```

# Recursive generators

- Can generators encode a CFG?

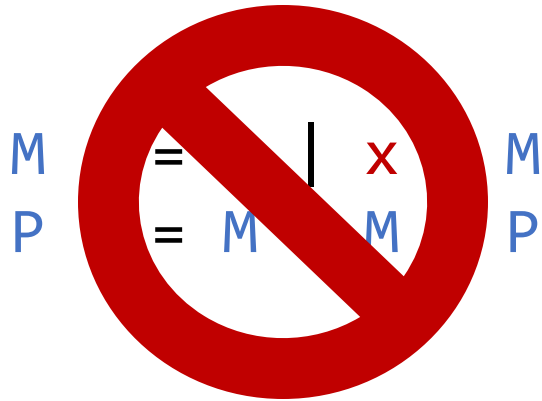
$$\begin{array}{l} M ::= n \mid x * M \\ P ::= M \mid M + P \end{array}$$

```
generator int mono(int x) {  
    if (??) {return ??;}  
    else {return x * mono(x);}  
}
```

```
generator int poly(int x) {  
    if (??) {return mono(x);}  
    else {return mono(x) + poly(x);}  
}
```

# Recursive generators

- What if monomial of every degree can occur at most once?



```
generator int mono(int x, int n) {  
    if (n <= 0) {return ??;}  
    else {return x * mono(x, n - 1);}  
}
```

```
generator int poly(int x, int n) {  
    if (n <= 0) {return mono(x,0);}  
    else {return mono(x,n) + poly(x, n - 1);}  
}
```

# Program Sketching

## 2. How to encode the behavior of complex programs?

Symbolic execution

encoding

Program c has no  
assertion  
violations on input  
x

$$\exists C . \forall x . Q(C, x)$$

## 3. How to solve for complex specs? CEGIS



Behavioral constraints  
= assertions / reference  
implementation

Structural constraints

## 1. How to specify for complex programs?

Sketches

