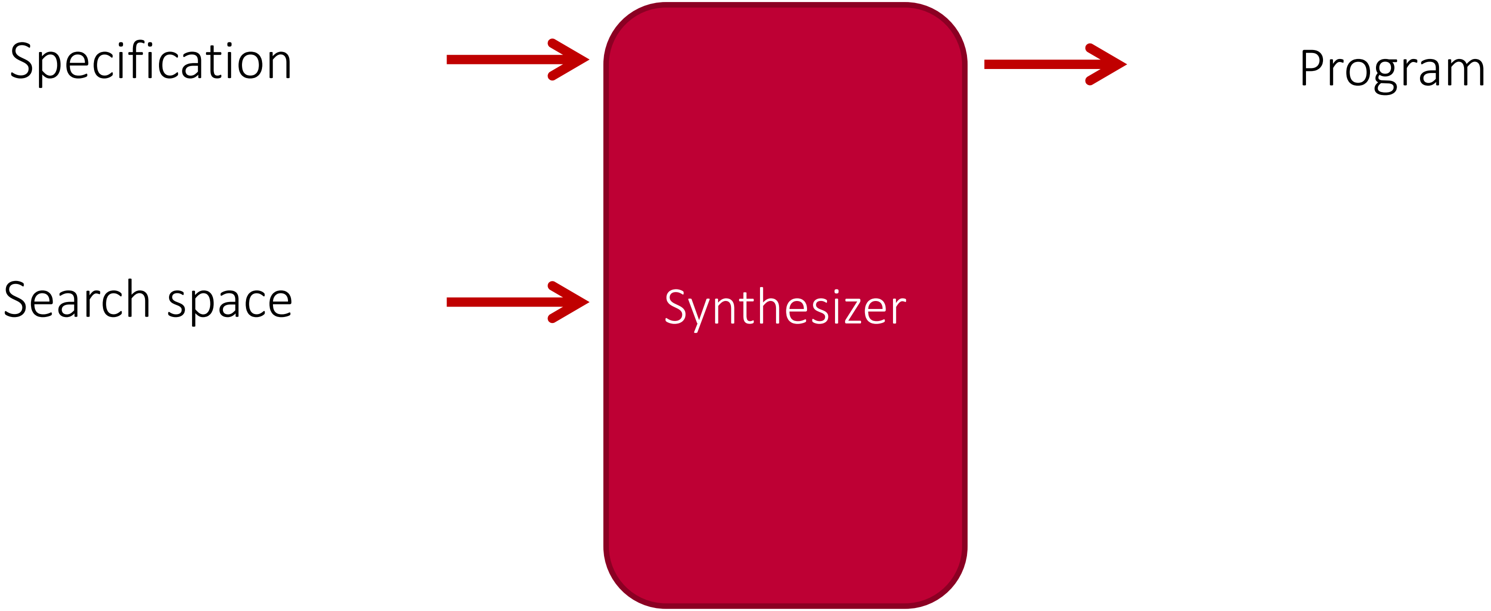
SYNTAX-GUIDED SYNTHESIS WITH GUARANTEES

Loris D’Antoni

University of Wisconsin Madison

Program synthesis with guarantees



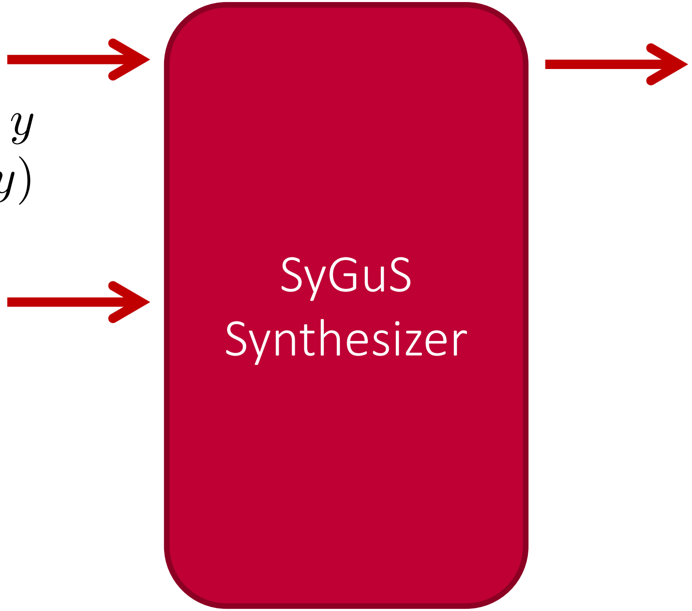
Syntax Guided Synthesis [Alur et al. 13]

Specification

$$\Phi(P) : \forall x, y. P \geq x \wedge P \geq y \\ \wedge (P = x \vee P = y)$$

Search space

```
Start   := Start+Start
         | ITE(BExpr,Start,Start)
         | x | y | 0 | 1
BExpr := NOT(BExpr)
         | Start > Start
         | Start AND Start
```



Program

ITE(x>y, x, y)

ITE(x>y, ITE(x>0, x, x), y)

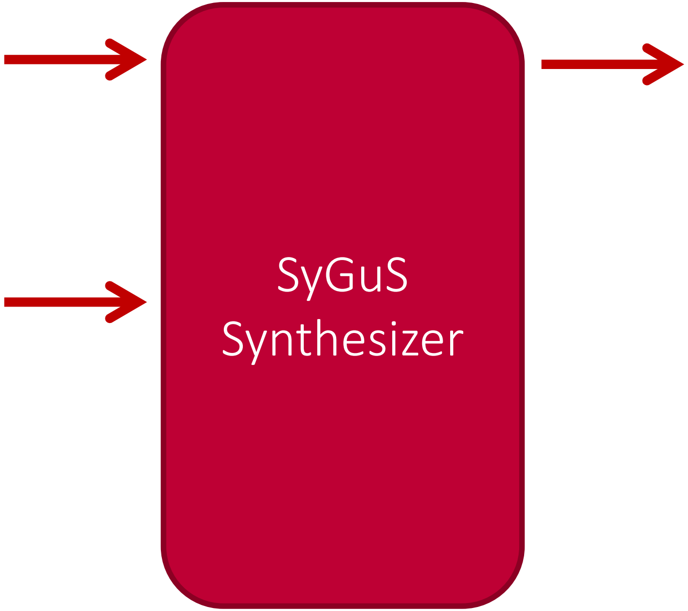
Syntax Guided Synthesis [Alur et al. 13]

Formula

$\Phi(P)$

Context-free grammar

G



Program

$$P^* \;\; s.t. \quad \Phi(P^*) \\ \wedge \quad P^* \in L(G)$$

Applications of SyGuS

Automatic Program Inversion using Symbolic Transducers

Univer

Quantified Invariants via

[1] Princ
[2] TC

Synthesis of Fault-Attack Countermeasures for Cryptographic Circuits

Accelerating Syntax-Guided Invariant Synthesis

Grigory Fedyukovich[1] and Rastislav Bodík[2]

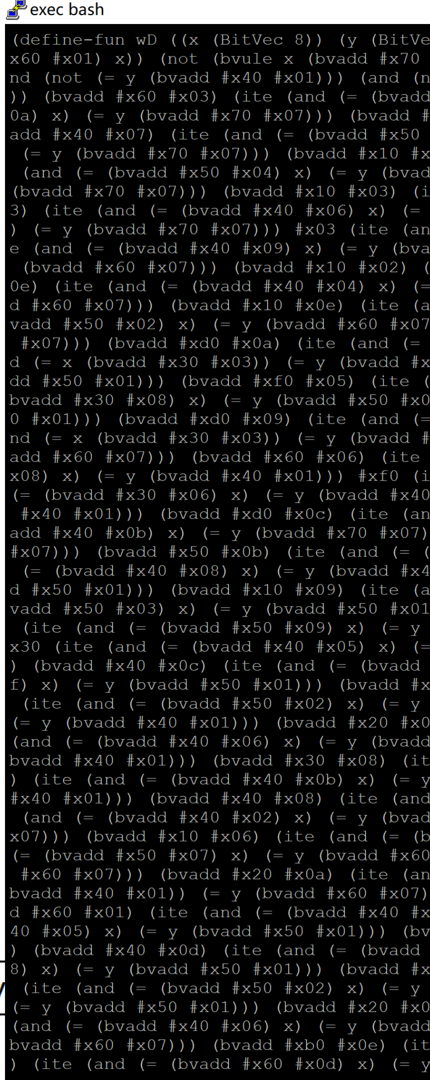
[1] Princeton University, USA, grigoryf@cs.princeton.edu
[2] University of Washington, USA, bodik@cs.washington.edu

DOES IT REALLY WORK

```
// Axuiliary functions
fun E (x: x <= #x40) :=
   (ite (x <= #x19) (x + #x41)
      (ite (x <= #x33) (x + #x47)
        (ite (x <= #x3d) (x - #x04)
          (ite (x == #x3e) #x2b #x2f))))
fun B h l x := (x << (7 - h)) >> (7 - h + l)
// List transformations
trans B64E (l: (BitVec 8) list) : (BitVec 8) :=
   match l with
   | x::y::z::tail when true ->
     E (B 7 2 x) ::
     E (((B 1 0 x) << #x04) | (B 7 4 y)) ::
     E (((B 4 0 y) << 2) | (B 7 6 z)) ::
     E (B 5 0 z) :: B64E(tail)
   | x::y::[] when true ->
     E (B 7 2 x) ::
     E (((B 1 0 x) << 4) | (      y)) ::
     E ((B 4 0 y) << 2) :: #x      []
   | x::[] when true ->
     E (B 7 2 x)) ::
     E ((B 1 0 x) << 4) :: #x3d      :: []
   | [] when true -> []
```

CVC4

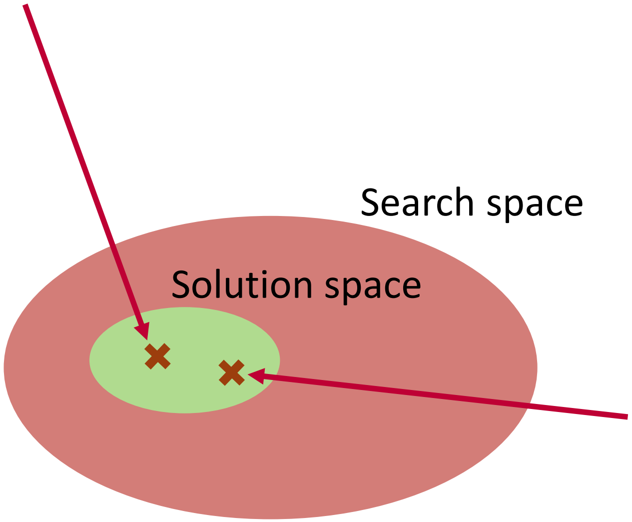
ESolver

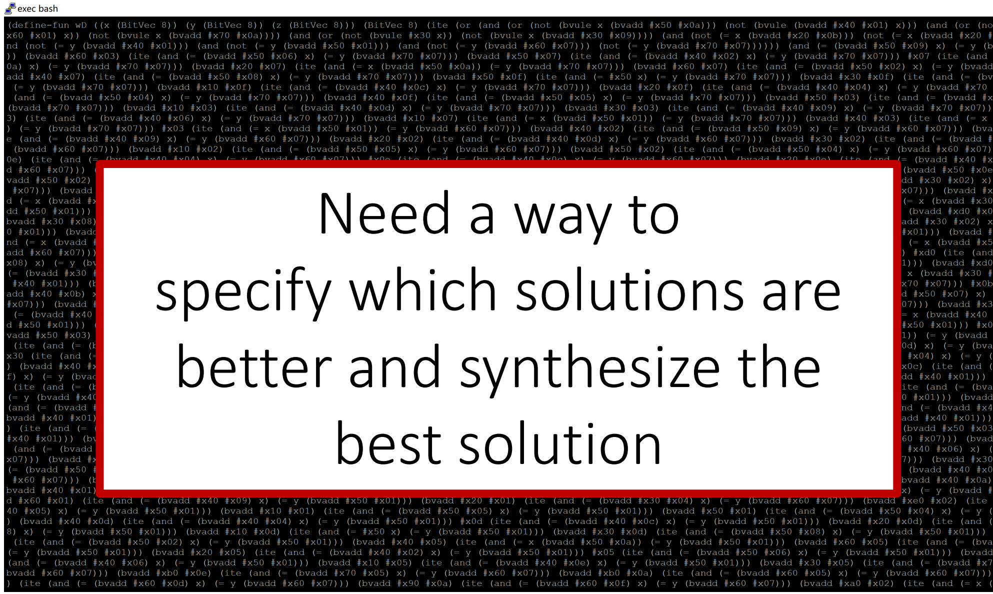



```
(define-fun ((x (BitVec 8)) (y (BitVec 8))) (bvand (bvlshl (DD x) #x02) (bv
```

Program synthesis is unpredictable – part 1

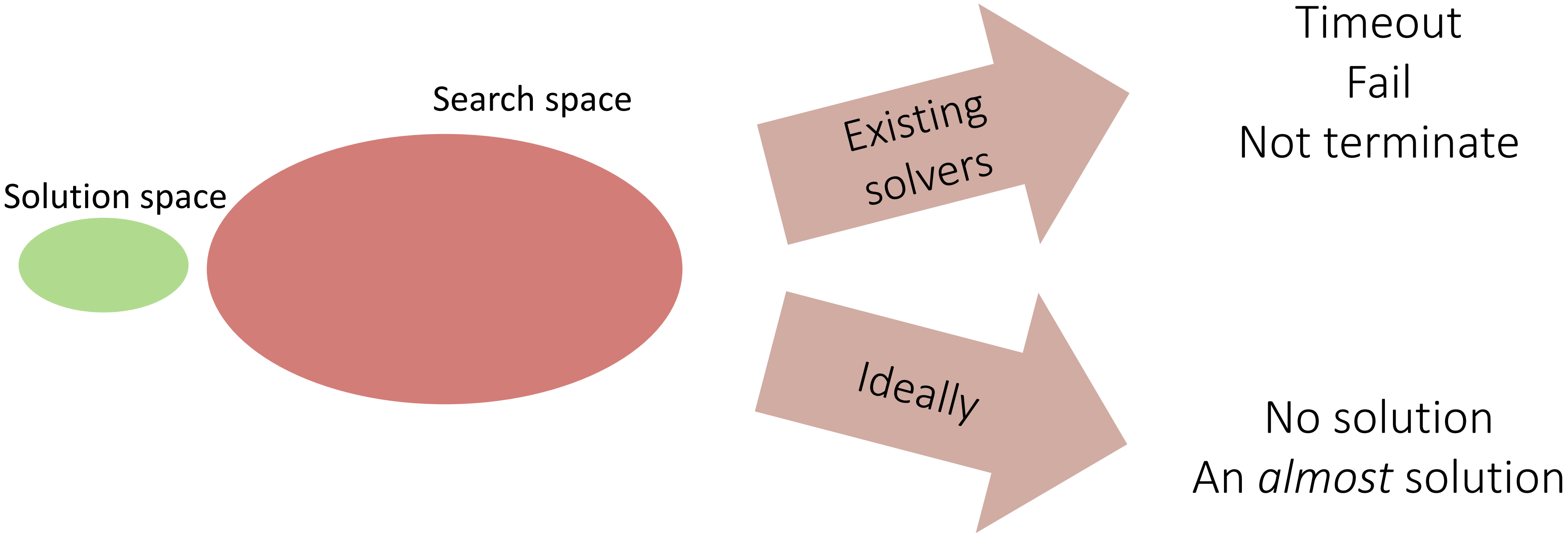
```
(define-fun ((x (BitVec 8)) (y (BitVec 8))) (bvand (bvlshl (DD x) #x02) (bvlshr (DD y) #x06)))
```





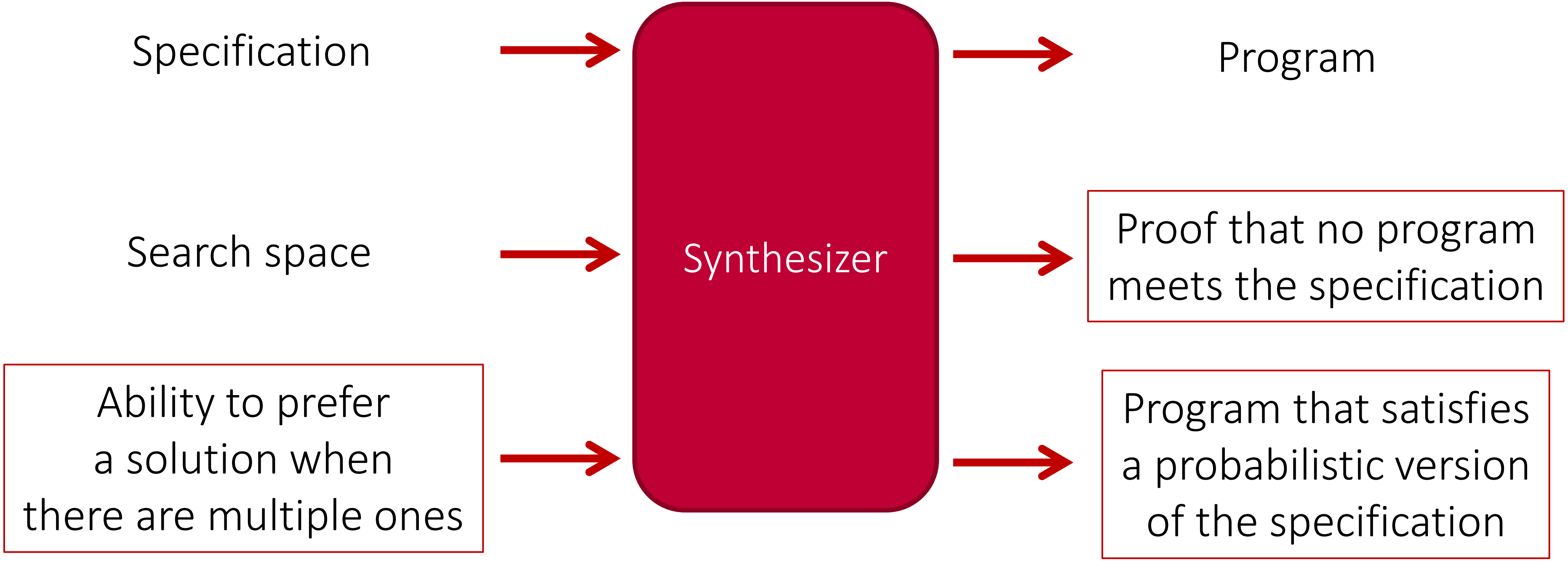
"Synthesis is like a box of chocolate,
You never know what you're gonna get"

Program synthesis is unpredictable – part 2

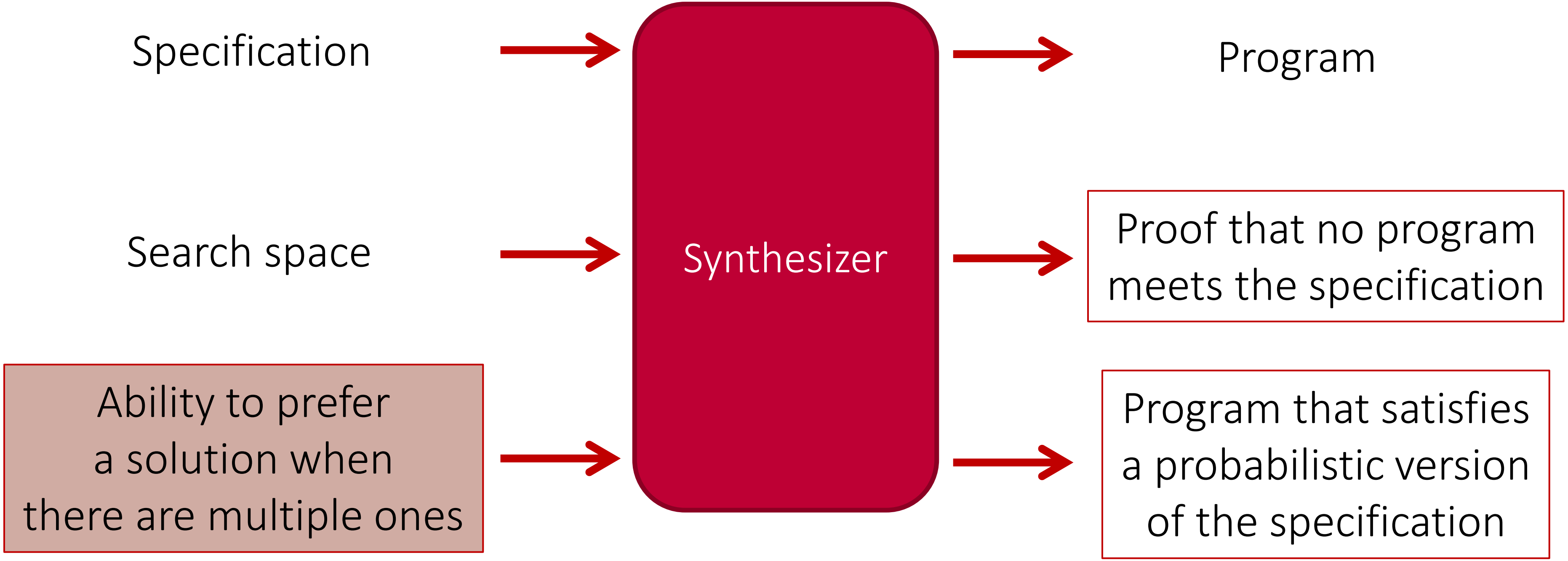


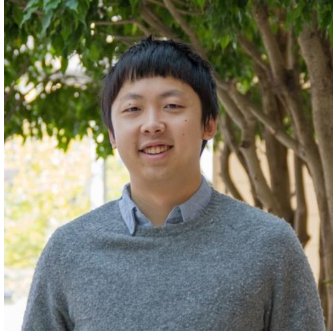
PROGRAM SYNTHESIS WITH GUARANTEES

Program synthesis with guarantees



Program synthesis with guarantees





SYNTAX-GUIDED SYNTHESIS WITH QUANTITATIVE OBJECTIVES

Q. HU, L. D'ANTONI [CAV18]

```
(define-fun wD ((x (BitVec 8)) (y (BitVec 8)) (z (BitVec 8))) (BitVec 8) (ite (or (and
x60 #x01) x)) (not (bvule x (bvadd #x70 #x0a)))) (and (or (not (bvule #x30 x)) (not (bvu
nd (not (= y (bvadd #x40 #x01))) (and (not (= y (bvadd #x50 #x01))) (and (not (= y (bva
)) (bvadd #x60 #x03) (ite (and (= (bvadd #x50 #x06) x) (= y (bvadd #x70 #x07))) (bvadd
0a) x) (= y (bvadd #x70 #x07))) (bvadd #x20 #x07) (ite (and (= x (bvadd #x50 #x0a)) (=
add #x40 #x07) (ite (and (= (bvadd #x50 #x08) x) (= y (bvadd #x70 #x07))) (bvadd #x50 #
 (= y (bvadd #x70 #x07))) (bvadd #x10 #x0f) (ite (and (= (bvadd #x40 #x0c) x) (= y (bva
 (and (= (bvadd #x50 #x04) x) (= y (bvadd #x70 #x07))) (bvadd #x40 #x0f) (ite (and (= (
(bvadd #x70 #x07))) (bvadd #x10 #x03) (ite (and (= (bvadd #x40 #x0d) x) (= y (bvadd #x7
3) (ite (and (= (bvadd #x40 #x06) x) (= y (bvadd #x70 #x07))) (bvadd #x10 #x07) (ite (a
) (= y (bvadd #x70 #x07))) #x03 (ite (and (= x (bvadd #x50 #x01)) (= y (bvadd #x60 #x07
e (and (= (bvadd #x40 #x09) x) (= y (bvadd #x60 #x07))) (bvadd #x20 #x02) (ite (and (=
 (bvadd #x60 #x07))) (bvadd #x10 #x02) (ite (and (= (bvadd #x50 #x05) x) (= y (bvadd #x
0e) (ite (and (= (bvadd #x40 #x04) x) (= y (bvadd #x60 #x07))) #x0e (ite (and (= (bvadd
d #x60 #x07))) (bvadd #x10 #x0e) (ite (and (= #x50 x) (= y (bvadd #x60 #x07))) (bvadd #
vadd #x50 #x02) x) (= y (bvadd #x60 #x07))) (bvadd #x40 #x06) (ite (and (= x (bvadd #x3
 #x07))) (bvadd #xd0 #x0a) (ite (and (= (bvadd #x30 #x06) x) (= y (bvadd #x60 #x07))) (
d (= x (bvadd #x30 #x03)) (= y (bvadd #x60 #x07))) (bvadd #xd0 #x0e) (ite (and (= (bvad
dd #x50 #x01))) (bvadd #xf0 #x05) (ite (and (= (bvadd #x30 #x01) x) (= y (bvadd #x50 #x
bvadd #x30 #x08) x) (= y (bvadd #x50 #x01))) (bvadd #xf0 #x01) (ite (and (= (bvadd #x30
0 #x01))) (bvadd #xd0 #x09) (ite (and (= (bvadd #x30 #x06) x) (= y (bvadd #x50 #x01)))
nd (= x (bvadd #x30 #x03)) (= y (bvadd #x50 #x01))) (bvadd #xd0 #x0d) (ite (and (= (bva
add #x60 #x07))) (bvadd #x60 #x06) (ite (and (= (bvadd #x30 #x01) x) (= y (bvadd #x40 #
x08) x) (= y (bvadd #x40 #x01))) #xf0 (ite (and (= (bvadd #x30 #x04) x) (= y (bvadd #x4
(= (bvadd #x30 #x06) x) (= y (bvadd #x40 #x01))) (bvadd #xe0 #x08) (ite (and (= (bvadd
 #x40 #x01))) (bvadd #xd0 #x0c) (ite (and (= (bvadd #x50 #x03) x) (= y (bvadd #x70 #x07
add #x40 #x0b) x) (= y (bvadd #x70 #x07))) (bvadd #x20 #x0b) (ite (and (= x (bvadd #x40
```

Adding quantitative objectives

Specification

$$\Phi(P) : \forall x, y. P \geq x \wedge P \geq y \wedge (P = x \vee P = y)$$

Search space

```
Start   := Start+Start
         | ITE(BExpr,Start,Start)
         | x | y | 0 | 1
BExpr := NOT(BExpr)
         | Start > Start
         | Start AND Start
```

SyGuS Synthesizer

Program

ITE(x>y, x, y)

ITE(x>y, ITE(x>0, x, x), y)

Need a way to prefer the first solution

Adding quantitative objectives

Specification

$$\Phi(P) : \forall x, y. P \geq x \wedge P \geq y$$
$$\wedge (P = x \vee P = y)$$

Search space

Start := Start+Start

| ITE(BExpr,Start,Start) **1**

| x | y | 0 | 1

BExpr := NOT(BExpr)

| Start > Start

| Start AND Start

weight

SyGuS Synthesizer

Program

ITE(x>y, x, y)

ITE(x>y, ITE(x>0, x, x), y)

Need a way to prefer the first solution

What is the weight of a program?

ITE(x>y, ITE(x>0, x, x), y)   weight=2

Start := Start+Start

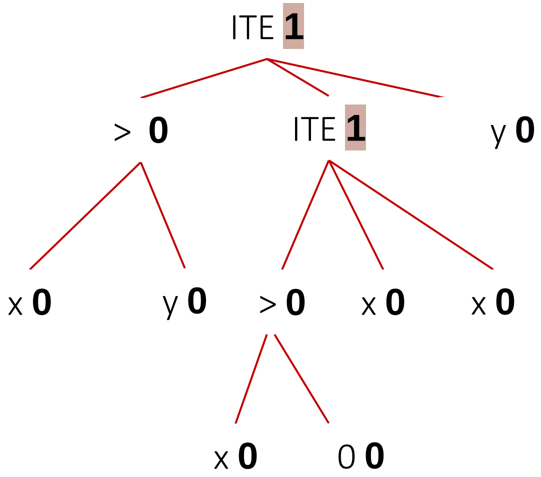
| ITE(BExpr,Start,Start)   **1**

| x | y | 0 | 1

BExpr := NOT(BExpr)

| Start > Start

| Start AND Start



Adding quantitative objectives

Specification

$$\Phi(P) : \forall x, y . P \geq x \wedge P \geq y$$
$$\wedge (P = x \vee P = y)$$

Search space

Start := Start+Start

| ITE(BExpr,Start,Start) **1**

| x | y | 0 | 1

BExpr := NOT(BExpr)

| Start > Start

| Start AND Start

SyGuS Synthesizer

Program

ITE(x>y, x, y) **1**

ITE(x>y, ITE(x>0, x, x), y) **2**

Programs now have weights/costs

Syntax Guided Synthesis with Quantitative Objectives

Formula

$$\Phi(P)$$

Weighted grammar

$$W$$

Quantitative objective

$$minimize\ cost$$

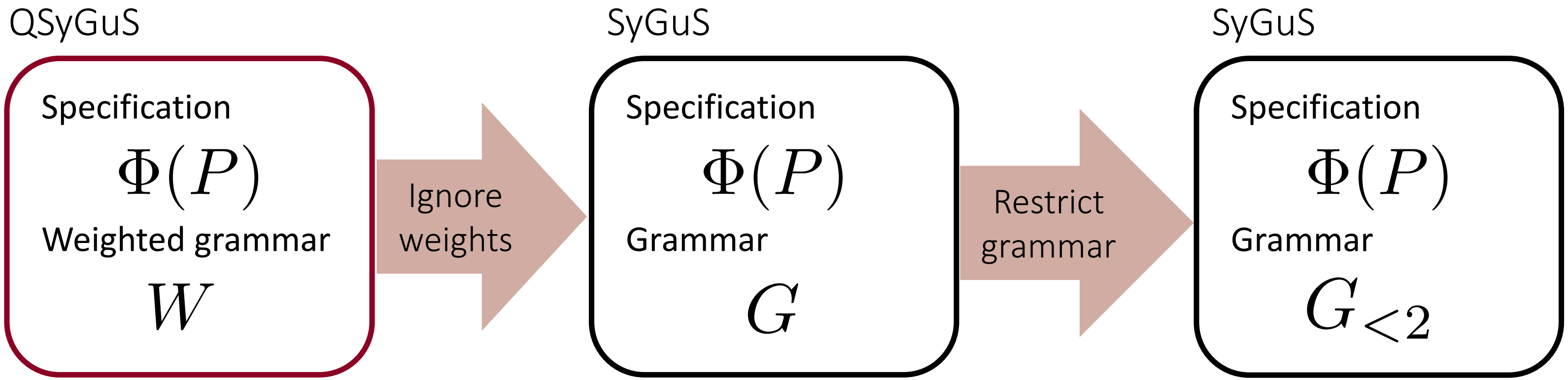
QSyGuS Synthesizer

Program

$$\begin{aligned} P^* \ s.t. \quad & \Phi(P^*) \\ \land \quad & P^* \in L(W) \\ \neg\exists Q \ s.t. \quad & \Phi(Q) \\ \land \quad & Q \in L(W) \\ \land \quad & cost(Q) < cost(P^*) \end{aligned}$$

How do we solve a QSyGuS problem?

Solving QSyGuS problems

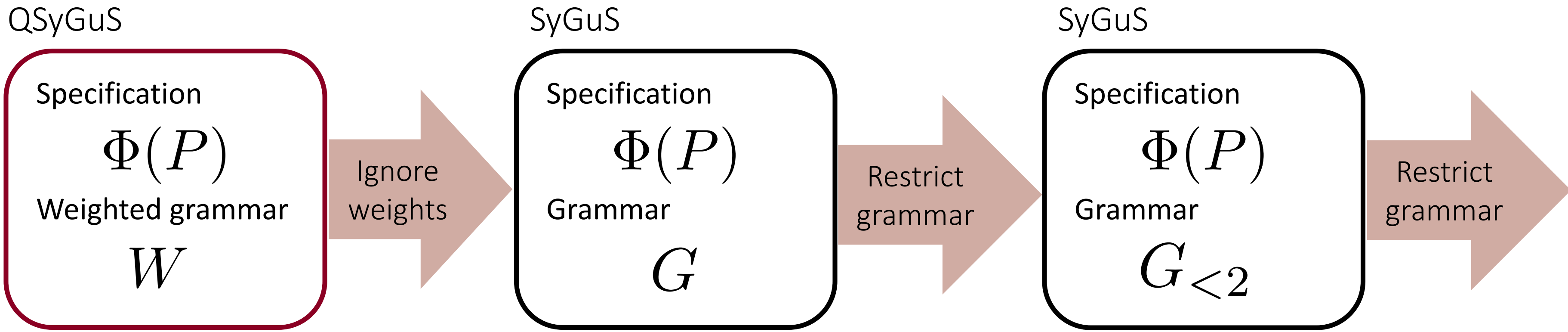



```
Start   := Start+Start
         | ITE(BExpr,Start,Start)      1
         | x | y | 0 | 1
...
```

```
Start   := Start+Start
         | ITE(BExpr,Start,Start)
         | x | y | 0 | 1
...
```

ITE(x>y, ITE(x>0, x, x), y)   **2**

Solving QSyGuS problems



Start := Start+Start
| ITE(BExpr,Start,Start) **1**
| x | y | 0 | 1
...

ITE(x>y, ITE(x>0, x, x), y) **2**

Start := Start**0** | Start**1**

Start**0** := Start**0**+Start**0**

| x | y | 0 | 1

Start1 := ITE(BExpr, Start**0**, Start**0**)

| Start**0**+Start**1** | Start**1**+Start**0**

ITE(x>y, x, y) **1**

...

Soundness of grammar restriction

Weighted grammar does not contain negative weights

$$W$$

Reduced grammar accepts all and only the terms of weight < c

$$G_{<c}$$

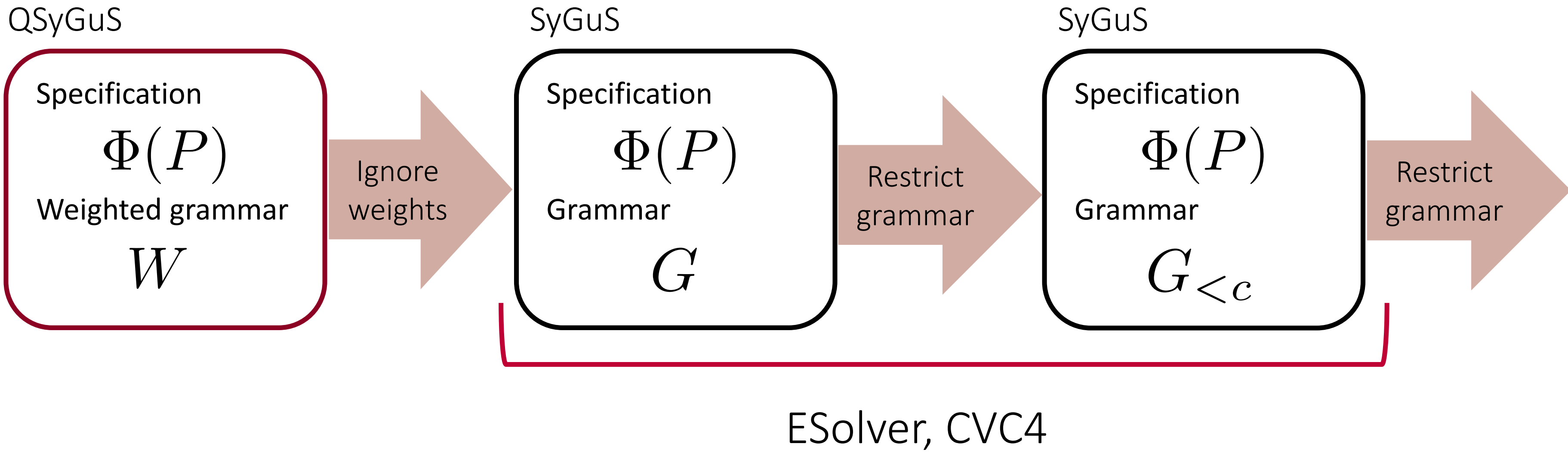
Results also generalizes to multiplicative weights

Beyond minimization constraints

$minimize\ cost$	Linear search
$cost > 3$	$complement(G_{<4})$
$2 < cost < 5$	$G_{>2} \cap G_{<5}$
$3 < cost_1 \wedge cost_2 < 0.5$	$G_{cost_1 > 3} \cap G_{cost_2 < 0.5}$

DOES IT WORK?

Implementation



Benchmarks

26 SyGuS benchmarks

$(\mathbb{R},+)$: minimize number of specified operator

minimize solution size

$([0,1],*)$: maximize solution probability

$(\mathbb{R},+)\times(\mathbb{R},+)$: find sorted optimal for (# of specified operators, size)

find Pareto optimal for (# of specified operators, size)

	Problem
Trop	max_ite(2,3)
	max_ite(2,15)
	max_ite(3,15)
	max_ite(10,15)
	parity_not
	max3_ite
	array_search_3
	array_search_5
	hackers_5
	hackers_7
	hackers_17
	hackers_19
	icfp_7
	LinExpr_eq1ex
Prob	hackers_2_prob
	hackers_5_prob
	hackers_7_prob
	hackers_17_prob

	Problem
Trop × Trop	array_search_sorted
	hackers_5_sorted
	hackers_7_sorted
	hackers_17_sorted
	array_search_pareto
	hackers_5_pareto
	hackers_7_pareto
	hackers_17_pareto

Summary of results

Solution with **better cost** than one without QSyGuS for **16/26** benchmarks

Found **optimal** solution for **14/26 benchmarks**

Average time **3.1x** compared to SyGuS

Why couldn't we prove optimality?

QSyGuS

Specification

$\Phi(P)$

Weighted grammar

W

...

Restrict grammar

SyGuS

Specification

$\Phi(P)$

Grammar

$G_{<2}$

Restrict grammar

SyGuS

Specification

$\Phi(P)$

Grammar

$G_{<1}$

Start := Start+Start
| ITE(BExpr,Start,Start) **1**
| x | y | 0 | 1
...

ITE(x>y, x, y) **1**

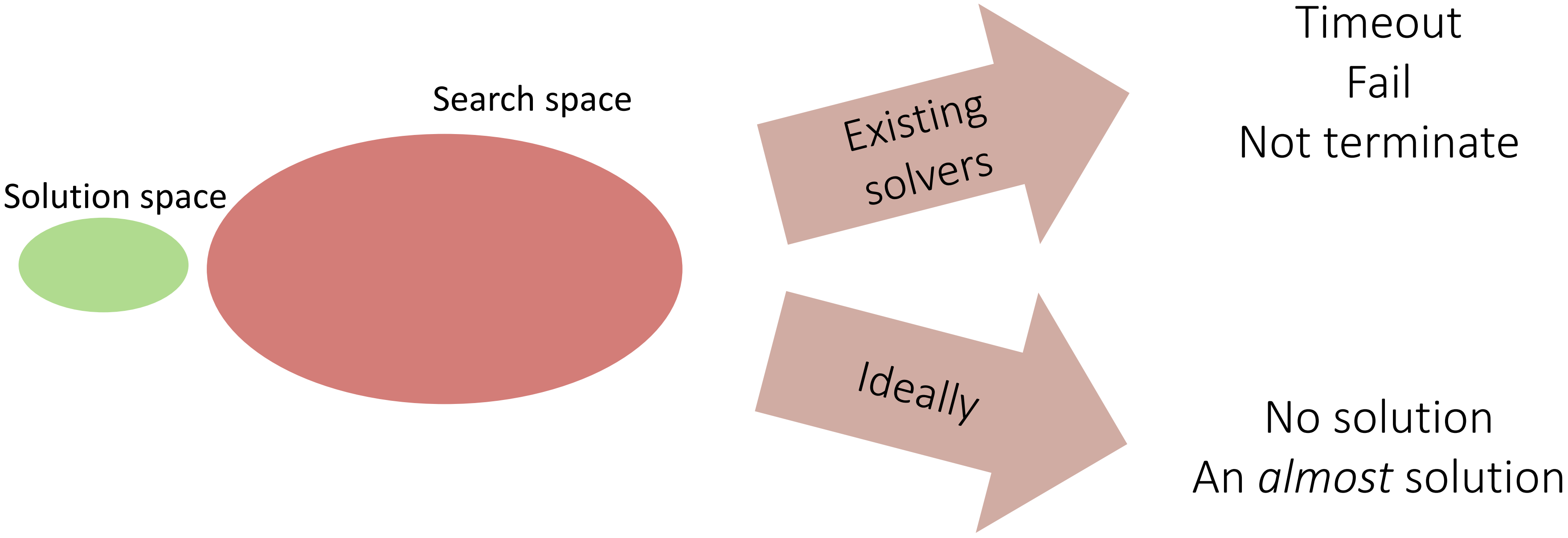
Start := Start**0**

Start**0** := Start**0**+Start**0**

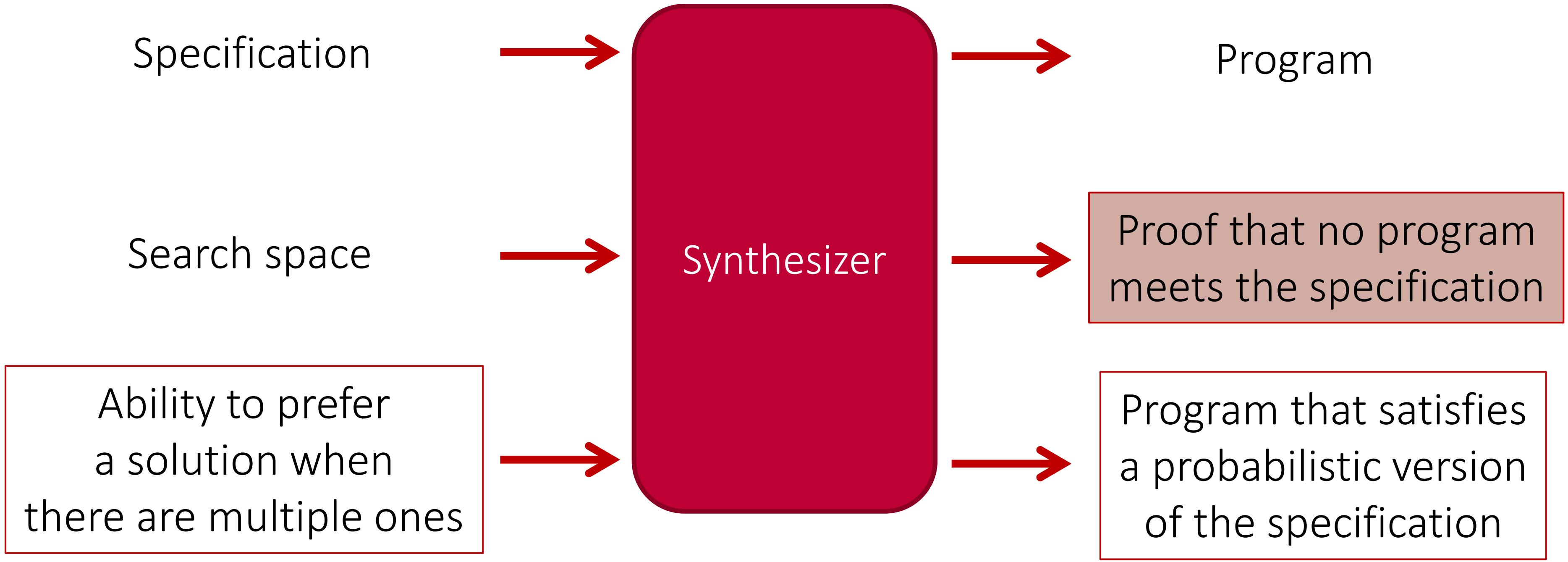
| x | y | 0 | 1

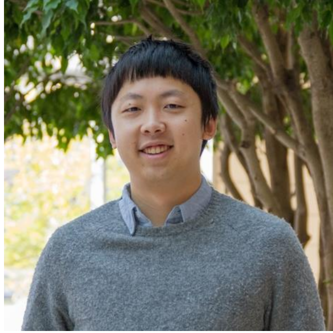
No solution!

Program synthesis is unpredictable – part 2



Program synthesis with guarantees





PROVING UNREALIZABILITY IN SYNTAX-GUIDED SYNTHESIS

Q. HU, J. BRECK, J. CYPHERT, L. D'ANTONI, T. REPS [CAV19]

Why is this hard?

Specification

$$\Phi(P) : \forall x, y. P \geq x \land P \geq y$$
$$\land (P = x \lor P = y)$$

Search space

Start := Start+Start

| x | y | 0 | 1

SyGuS solver

Unrealizable

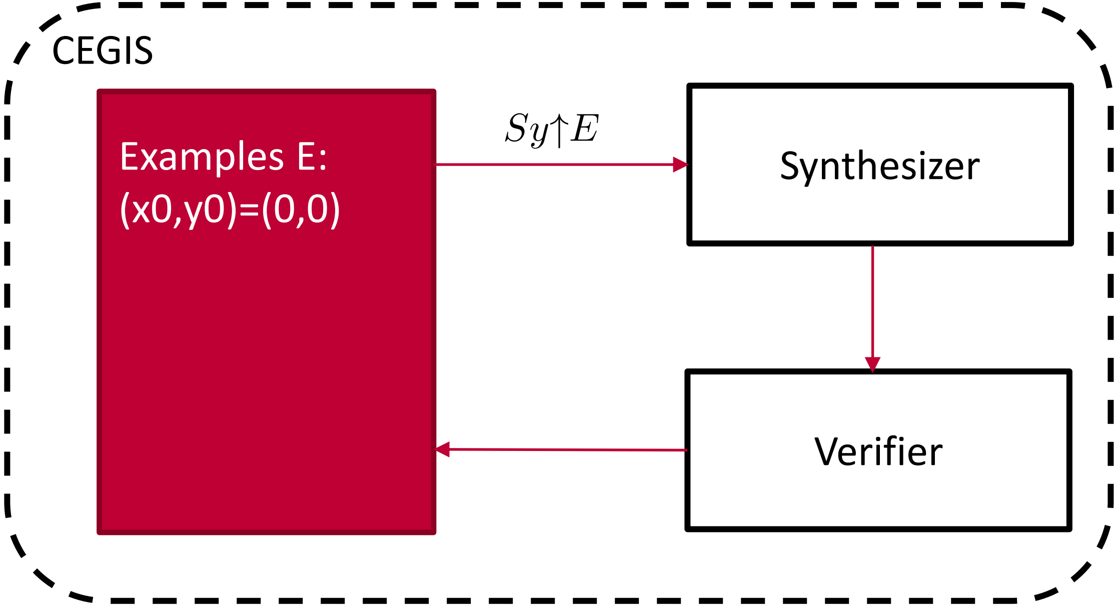
Proof that

$$\neg \exists Q \; s.t. \quad \Phi(Q)$$
$$\land \quad Q \in L(G)$$

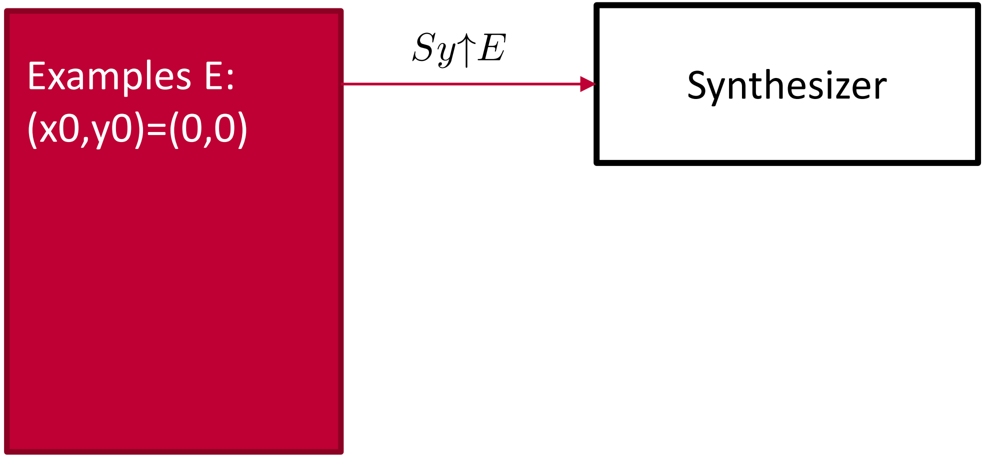
Infinite grammar makes the problem undecidable

Sy: $\Phi(f) : \forall x, y. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$

G: Start := Start+Start | x | y | 0 | 1

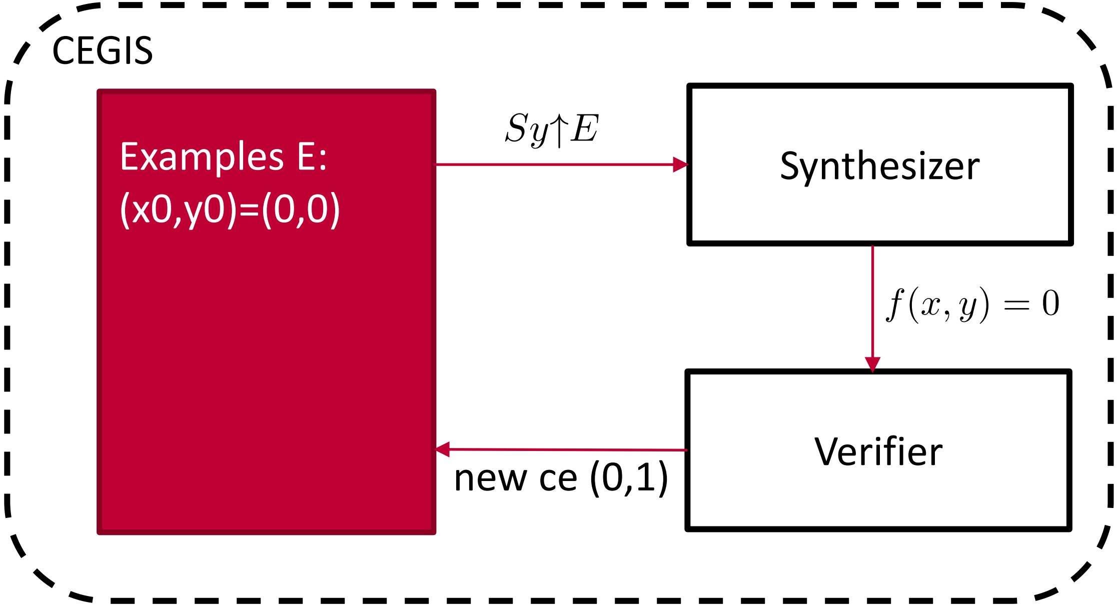


$$Sy{\uparrow}E = \bigwedge_{(x,y)\in E} \Phi(f, x, y)$$



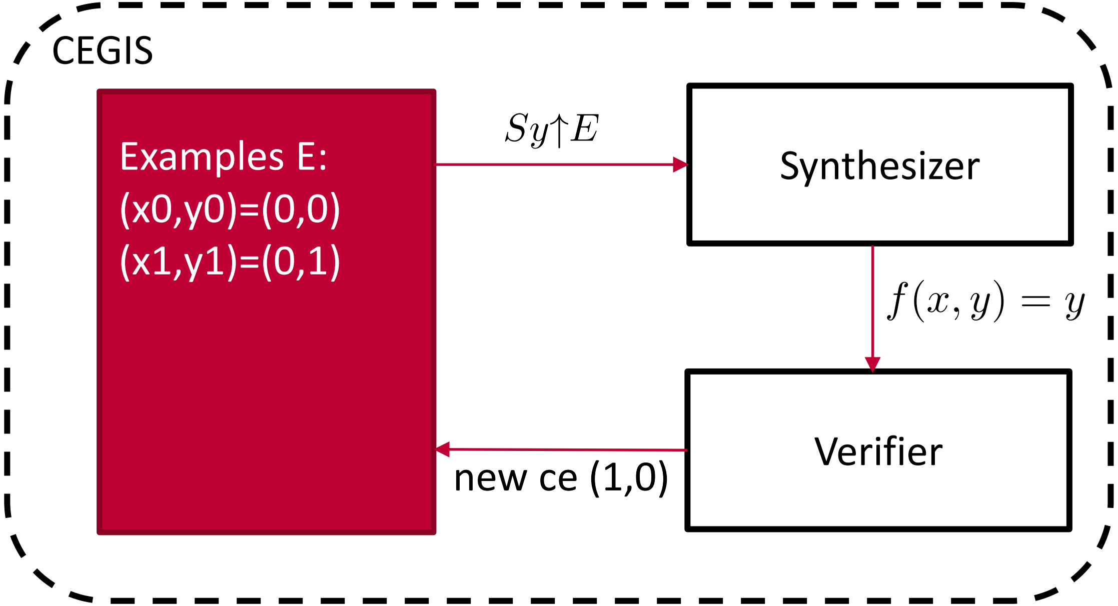
Sy: $\Phi(f): \forall x, y. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$

G: Start := Start+Start | x | y | 0 | 1



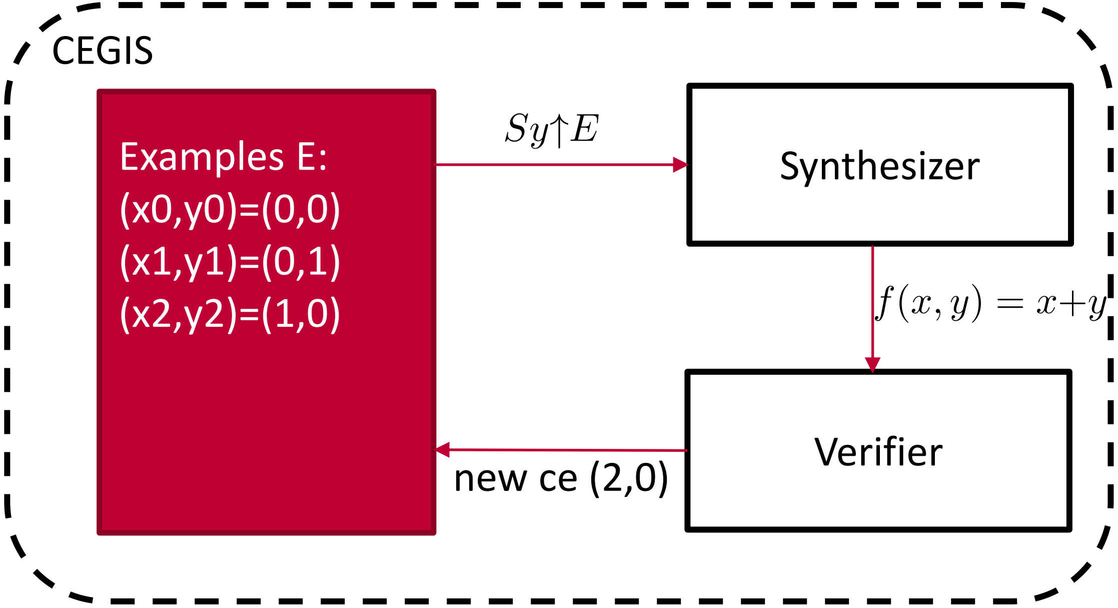
Sy: $\Phi(f) : \forall x, y. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$

G: Start := Start+Start | x | y | 0 | 1



Sy: $\Phi(f) : \forall x, y. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$

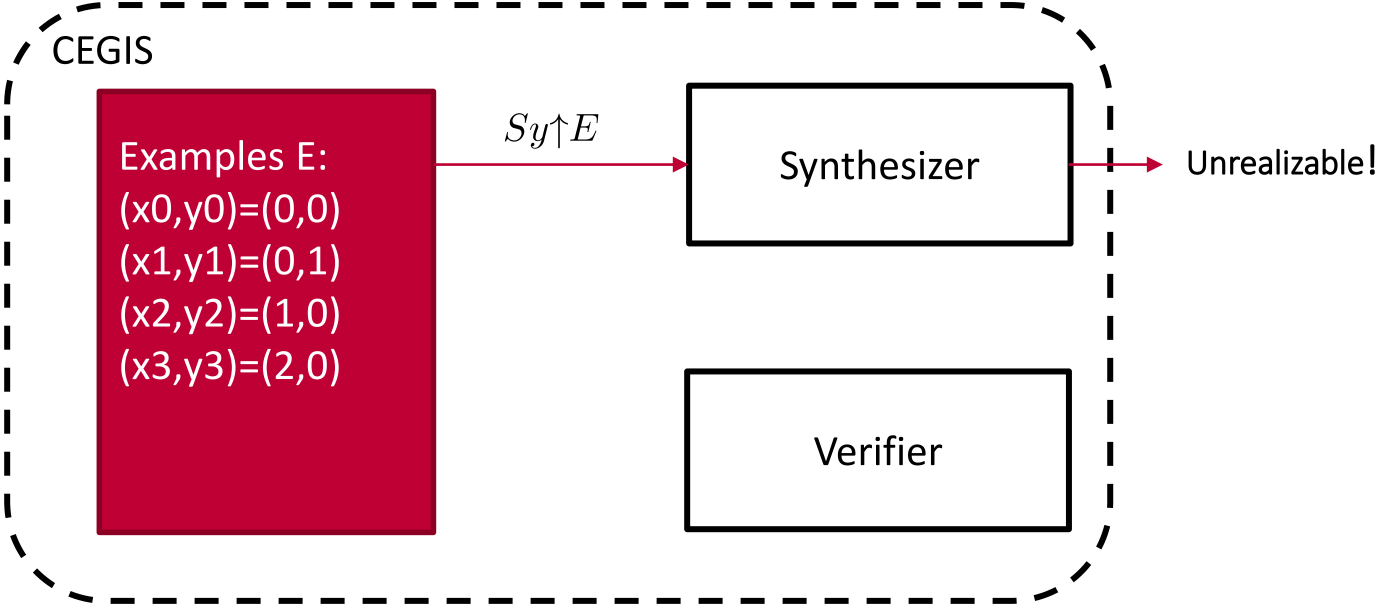
G: Start := Start+Start | x | y | 0 | 1



Sy:

$$\Phi(f) : \forall x, y. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$$

G: Start := Start+Start | x | y | 0 | 1



Soundness of CEGIS for unrealizability

$Sy{\uparrow}E$ unrealizable

No solution over E

$\Rightarrow$

Sy unrealizable

No solution

PROVING UNREALIZABILITY FOR SYGUS OVER EXAMPLES

Outline of the algorithm

$$Sy{\uparrow}E := (\Phi, G, E)$$

construct

```
int[4] Start(x_0,y_0,x_1,y_1,x_2,y_2,x_3,y_3){
    if(??){return (0,0,0,0);}           // Start -> 0
    if(??){return (1,1,1,1);}           // Start -> 1
    if(??){return (x_0,x_1,x_2,x_3);}   // Start -> x
    if(??){return (y_0,y_1,y_2,y_3);}   // Start -> y
    else{                             // Start -> Start + Start
        int[4] L = Start(x_0,y_0,x_1,y_1);
        int[4] R = Start(x_0,y_0,x_1,y_1);
        return (L[0]+R[0],L[1]+R[1],L[2]+R[2],L[3]+R[3]);}
}
int[4] P = Start(0,0,0,1,1,0,2,0);
assert (P[0]!=0 || P[1]!=1 || P[2]!=1 || P[3]!=2);
```

$Sy{\uparrow}E$ unrealizable $\Longleftrightarrow$ `assert` always holds

Reachability Problem

```
void main(){
      int x = 0;
      while(nd()){
            x++;
      }
      assert(x<0)
}
```

Nondeterministic choice

Reachability solvers:

CPA-checker

Uautomizer

Seahorn

Goal: can the `assert` be falsified?

Sy^E to Re^E

Set input to E

$$\vec{x} \leftarrow E$$

f_G is non-deterministically drawn from $L(G)$

$$\vec{o} \leftarrow f_G(\vec{x})$$

Check if $\vec{o}$ doesn't satisfy φ $\Longleftrightarrow$ $f_G(\vec{x})$ satisfy φ on E

$\Updownarrow$

```
assert( ¬φ(o, x) , x_i))
```

$Sy{\uparrow}E$ unrealizable

Set input to E

$$\vec{x} \leftarrow E$$

Examples E:
(x0,y0)=(0,0)
(x1,y1)=(0,1)

```
x0 = 0;
y0 = 0;
x1 = 0;
y1 = 1;
```

Sy^E to Re^E

Set input to E

$$\vec{x} \leftarrow E$$

f_G is non-deterministically drawn from $L(G)$

$$\vec{o} \leftarrow f_G(\vec{x})$$

Check if $\vec{o}$ doesn't satisfy φ

`assert(` $\neg\varphi(o, \bar{x})$ $, x_i))$

Check if $\vec{o}$ doesn't satisfy φ

$$\texttt{assert}(\neg \bigwedge x_i \in E.\,\varphi(o_i, x_i))$$

```
void main(){
    …
    assert(!(spec(x0,y0,o0)&&spec(x1,y1,o1)));
}
bool spec(x,y,o){
    return (o>=x)&&(o>=y)&&(o==x||o==y);
}
```

$$\Phi(f) : \forall x, y. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$$

Sy^E to Re^E

Set input to E

$$\vec{x} \leftarrow E$$

f_G is non-deterministically drawn from $L(G)$

$$\vec{o} \leftarrow f_G(\vec{x})$$

Check if $\vec{o}$ doesn't satisfy φ

assert($\neg\varphi(o, \bar{x})$, x_i))

f_G is non-deterministically drawn from $L(G)$

$$\vec{o} \leftarrow f_G(\vec{x})$$

```
o0 = fStart(x0,y0);
```

```
int fStart(x0,y0){
    if(nd()){ return 0;}   \\ Start -> 0
    if(nd()){ return 1;}   \\ Start -> 1
    if(nd()){ return x0;}  \\ Start -> x
    if(nd()}{ return y0;}  \\ Start -> y
    if(nd()){              \\ Start -> +(Start,Start)
        left = fStart(x0,y0);
        right = fStart(x0,y0);
        return left + right;}
}
```

Example 0

o0=fStart(x0,y0);
o0 is f_G(x0,y0)for some f_G in $L(G)$

Example 1

o1=fStart(x1,y1);
o1 is f_G(x1,y1)for some f_G in $L(G)$

The two f_G can be different!

f_G is non-deterministically drawn from $L(G)$

$$\vec{o} \leftarrow f_G(\vec{x})$$

```
(o0,o1) = fStart(x0,y0,x1,y1);

<int,int> fStart(x0,y0,x1,y1){
   if(nd()){ return (0,0);}   \\ Start -> 0
   if(nd()){ return (1,1);}   \\ Start -> 1
   if(nd()){ return (x0,x1);} \\ Start -> x
   if(nd()}{ return (y0,y1);} \\ Start -> y
   if(nd()){                  \\ Start -> +(Start,Start)
      (a0,a1) = fStart(x0,y0,x1,y1);
      (b0,b1) = fStart(x0,y0,x1,y1);
      return (a0+b0,a1+b1);}
}
```

Outline of the algorithm

$$Sy{\uparrow}E := (\Phi, G, E)$$

construct

```
int[4] Start(x_0,y_0,x_1,y_1,x_2,y_2,x_3,y_3){
     if(??){return (0,0,0,0);}             // Start -> 0
     if(??){return (1,1,1,1);}             // Start -> 1
     if(??){return (x_0,x_1,x_2,x_3);}     // Start -> x
     if(??){return (y_0,y_1,y_2,y_3);}     // Start -> y
     else{                            // Start -> Start + Start
          int[4] L = Start(x_0,y_0,x_1,y_1);
          int[4] R = Start(x_0,y_0,x_1,y_1);
          return (L[0]+R[0],L[1]+R[1],L[2]+R[2],L[3]+R[3]);}
}
int[4] P = Start(0,0,0,1,1,0,2,0);
assert (P[0]!=0 || P[1]!=1 || P[2]!=1 || P[3]!=2);
```

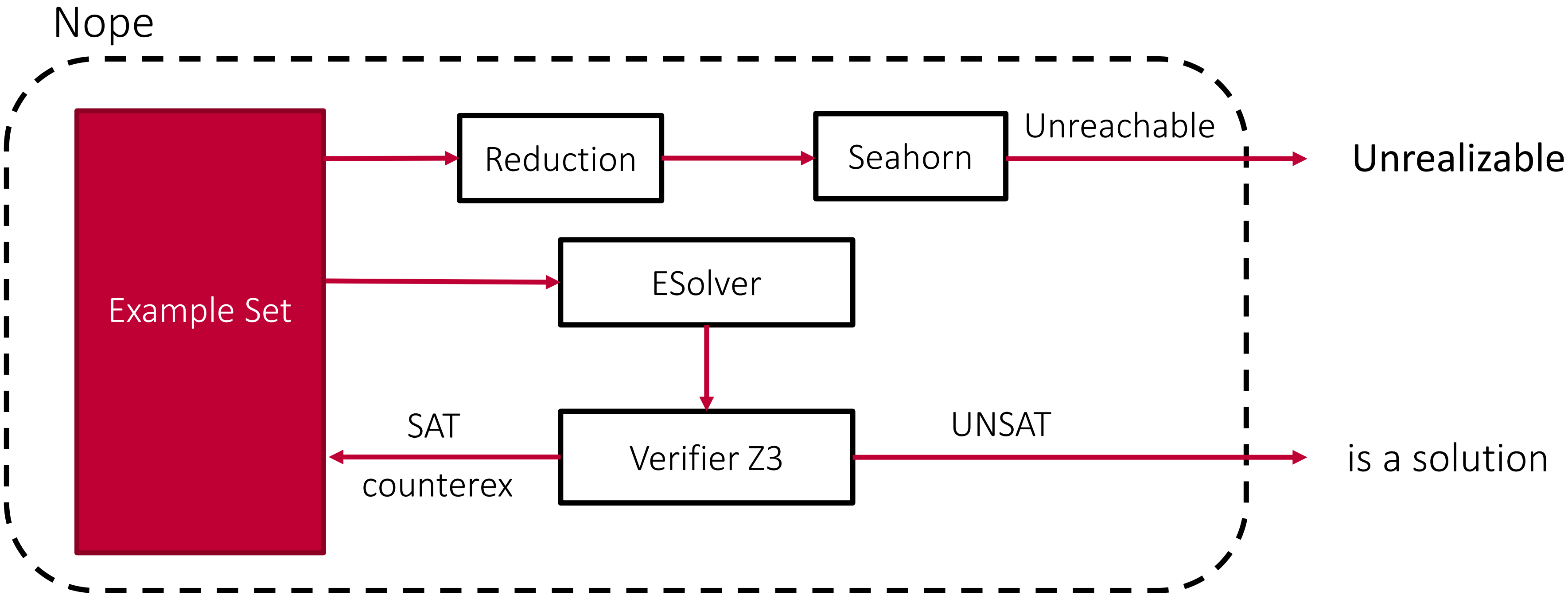
$Sy{\uparrow}E$ unrealizable $\Longleftrightarrow$ `assert` always holds

$\Downarrow$

Sy unrealizable

DOES IT WORK?

The tool NOPE



Benchmarks

QSyGuS

Specification

$\Phi(P)$

Weighted grammar

W

...

Restrict grammar

SyGuS

Specification

$\Phi(P)$

Grammar

$G_{<2}$

Restrict grammar

SyGuS

Specification

$\Phi(P)$

Grammar

$G_{<1}$

No solution!

60 SyGuS benchmarks over LIA

QSyGuS

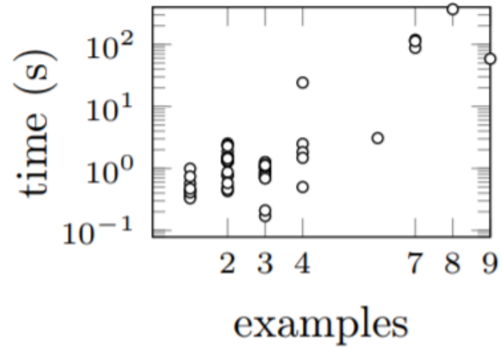
132 benchmarks that should be unrealizable

Summary of evaluation

132 variants of benchmarks taken from SyGuS	Solved
1. bounded number of if-operators	13/57
2. bounded number of plus-operators	1/30
3. restricted range of constants	45/45
	59/132

Limitation 1 of NOPE: number of examples

	number of nonterminals	number of productions	number of examples	total time (s)	SeaHorn time (s)
array_sum_4_5	5	34	14	✗	✗
array_sum_4_15	5	34	16	✗	✗



Limitation 2 of NOPE: size of grammars

	number of nonterminals	number of productions	number of examples	total time (s)	SeaHorn time (s)
mpg_example1	59	815	1	✗	✗
mpg_example2	21	178	1	✗	✗
mpg_example3	143	4186	1	✗	✗
mpg_example4	443	36745	1	✗	✗

Large sized reachability problem

Exact and Approximate Methods for Proving Unrealizability of Syntax Guided Synthesis [PLDI20]

Start := Start + Start | x

$x_0 = 0$, $x_1 = 1$

Generate equations

S = S + S ∪ (0,1)

Solution

S = { a*(0,1) | a>0 }

Ingredients:

Equations over semirings

Newton's method for solving equations

Special techniques for SyGuS operations

Program synthesis with guarantees

Specification

Search space

Ability to prefer a solution when there are multiple ones

Synthesizer

Program

Proof that no program meets the specification

Program that satisfies a probabilistic version of the specification

On Thursday!