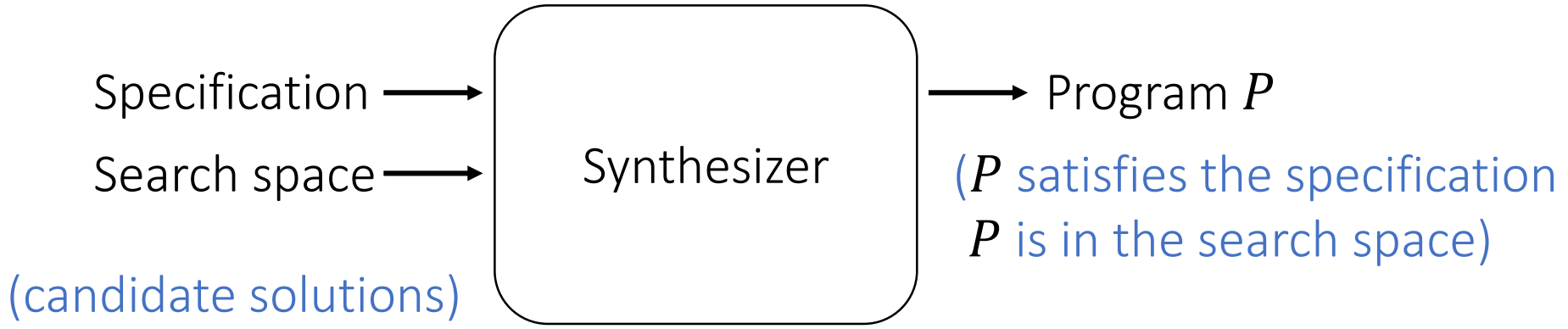
Guarantees in Program Synthesis

Qinheping Hu , Jason Breck , John Cyphert ,
Loris D'Antoni , Thomas Reps

(What you want your program to do)



Not always enough!

Specification: $\forall 0 \leq x < 100. P(x) = x + 1$

Solution 1:

```
return x+1;
```

Solution 2:

```
if (x=0) return 1;
if (x=1) return 2;
if (x=2) return 3;
…
if (x=99) return 100;
return -1;
```

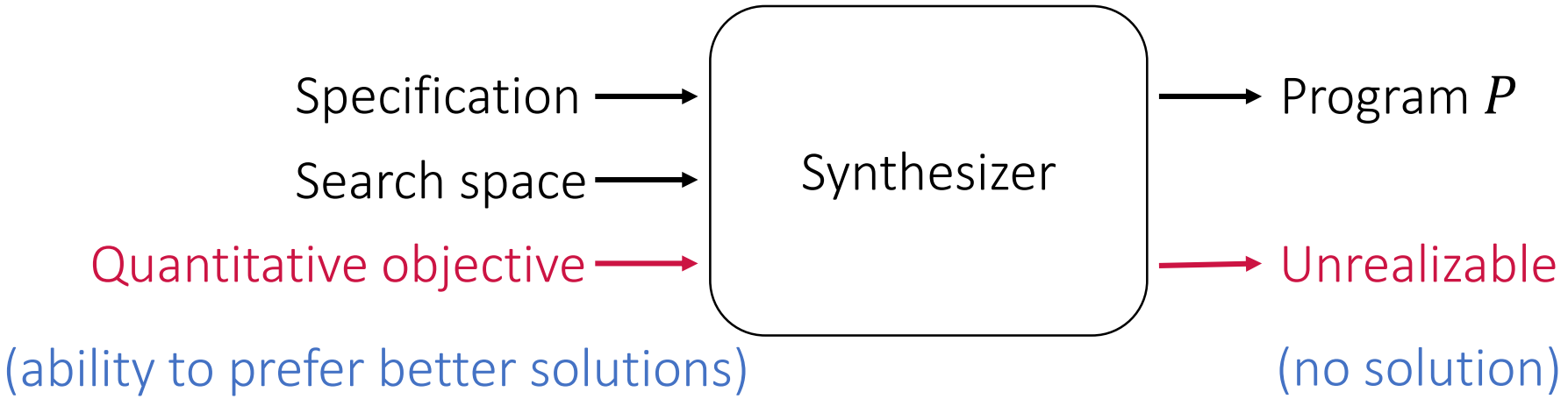
Automatic Program Inversion

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

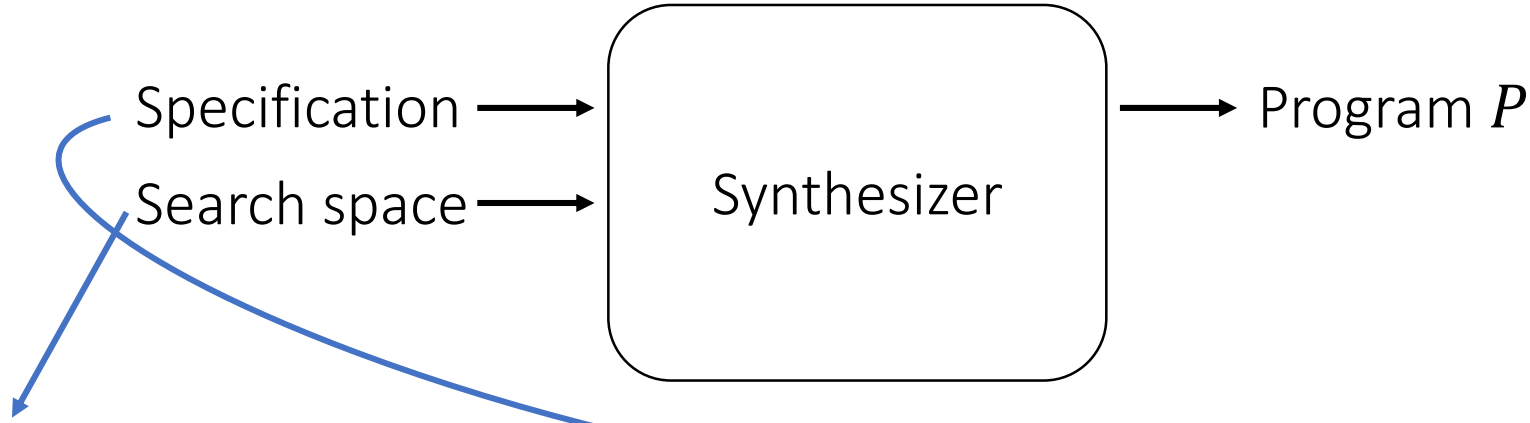
exec bash

```
(define-fun wD ((x (BitVec 8)) (y (BitVec 8)) (z (BitVec 8))) (BitVec 8) (ite (or (and (or (not (b
x60 #x01) x)) (not (bvule x (bvadd #x70 #x0a)))) (and (or (not (bvule #x30 x)) (not (bvule x (bvac
nd (not (= y (bvadd #x40 #x01))) (and (not (= y (bvadd #x50 #x01))) (and (not (= y (bvadd #x60 #x0
)) (bvadd #x60 #x03) (ite (and (= (bvadd #x50 #x06) x) (= y (bvadd #x70 #x07))) (bvadd #x50 #x07)
0a) x) (= y (bvadd #x70 #x07))) (bvadd #x20 #x07) (ite (and (= x (bvadd #x50 #x0a)) (= y (bvadd #x
add #x40 #x07) (ite (and (= (bvadd #x50 #x08) x) (= y (bvadd #x70 #x07))) (bvadd #x50 #x0f) (ite
 (= y (bvadd #x70 #x07))) (bvadd #x10 #x0f) (ite (and (= (bvadd #x40 #x0c) x) (= y (bvadd #x70 #x0
 (and (= (bvadd #x50 #x04) x) (= y (bvadd #x70 #x07))) (bvadd #x40 #x0f) (ite (and (= (bvadd #x50
(bvadd #x70 #x07))) (bvadd #x10 #x03) (ite (and (= (bvadd #x40 #x0d) x) (= y (bvadd #x70 #x07)))
3) (ite (and (= (bvadd #x40 #x06) x) (= y (bvadd #x70 #x07))) (bvadd #x10 #x07) (ite (and (= x (bv
) (= y (bvadd #x70 #x07))) #x03 (ite (and (= x (bvadd #x50 #x01)) (= y (bvadd #x60 #x07))) (bvadd
e (and (= (bvadd #x40 #x09) x) (= y (bvadd #x60 #x07))) (bvadd #x20 #x02) (ite (and (= (bvadd #x40
 (bvadd #x60 #x07))) (bvadd #x10 #x02) (ite (and (= (bvadd #x50 #x05) x) (= y (bvadd #x60 #x07)))
0e) (ite (and (= (bvadd #x40 #x04) x) (= y (bvadd #x60 #x07))) #x0e (ite (and (= (bvadd #x40 #x0c)
d #x60 #x07))) (bvadd #x10 #x0e) (ite (and (= #x50 x) (= y (bvadd #x60 #x07))) (bvadd #x30 #x0e)
vadd #x50 #x02) x) (= y (bvadd #x60 #x07))) (bvadd #x40 #x06) (ite (and (= x (bvadd #x30 #x09)) (=
 #x07))) (bvadd #xd0 #x0a) (ite (and (= (bvadd #x30 #x06) x) (= y (bvadd #x60 #x07))) (bvadd #xe0
d (= x (bvadd #x30 #x03)) (= y (bvadd #x60 #x07))) (bvadd #xd0 #x0e) (ite (and (= (bvadd #x30 #x05
dd #x50 #x01))) (bvadd #xf0 #x05) (ite (and (= (bvadd #x30 #x01) x) (= y (bvadd #x50 #x01))) (bvac
bvadd #x30 #x08) x) (= y (bvadd #x50 #x01))) (bvadd #xf0 #x01) (ite (and (= (bvadd #x30 #x04) x)
0 #x01))) (bvadd #xd0 #x09) (ite (and (= (bvadd #x30 #x06) x) (= y (bvadd #x50 #x01))) (bvadd #xe0
nd (= x (bvadd #x30 #x03)) (= y (bvadd #x50 #x01))) (bvadd #xd0 #x0d) (ite (and (= (bvadd #x30 #x0
add #x60 #x07))) (bvadd #x60 #x06) (ite (and (= (bvadd #x30 #x01) x) (= y (bvadd #x40 #x01))) (bva
x08) x) (= y (bvadd #x40 #x01))) #xf0 (ite (and (= (bvadd #x30 #x04) x) (= y (bvadd #x40 #x01))) #
(= (bvadd #x30 #x06) x) (= y (bvadd #x40 #x01))) (bvadd #xe0 #x08) (ite (and (= (bvadd #x30 #x07)
 #x40 #x01))) (bvadd #xd0 #x0c) (ite (and (= (bvadd #x50 #x03) x) (= y (bvadd #x70 #x07))) (bvadd
add #x40 #x0b) x) (= y (bvadd #x70 #x07))) (bvadd #x20 #x0b) (ite (and (= x (bvadd #x40 #x07)) (=
```

Guarantees in Program Synthesis



Syntax-Guided Synthesis



Grammar:

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

BExpr := NOT(BExpr)
| Start > Start
| Start AND Start

(Linear terms with IfThenElse)

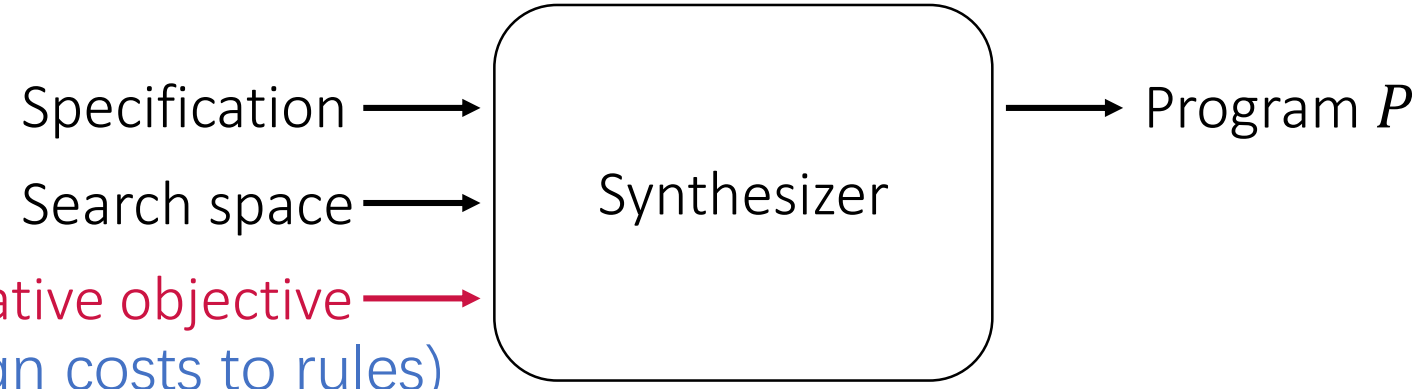
Logic formula:

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

(i.e., P is max2)

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

Syntax-Guided Synthesis + Quantitative objective



Grammar:

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

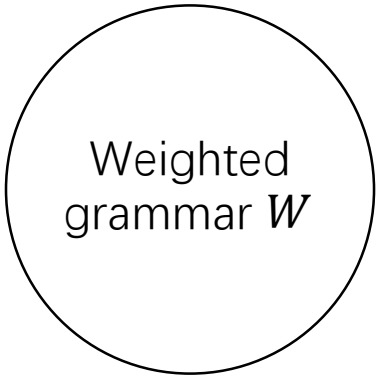
Logic formula:

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

Quantitative objective:

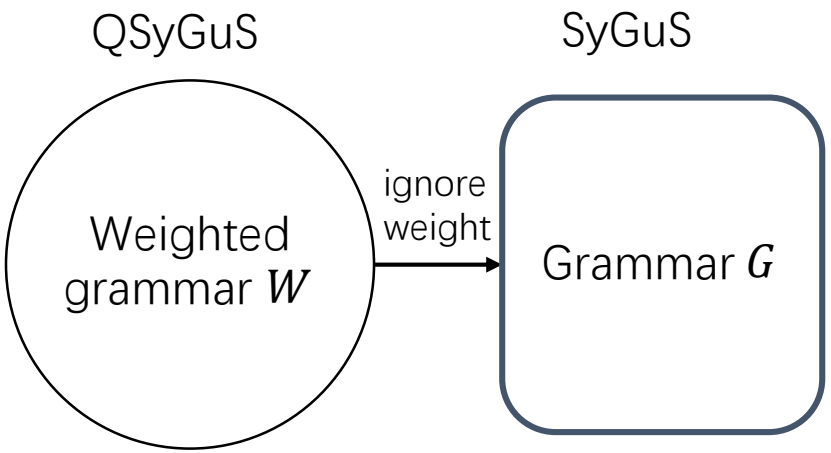
Minimize the number of ITE

QSyGuS



Grammar W :

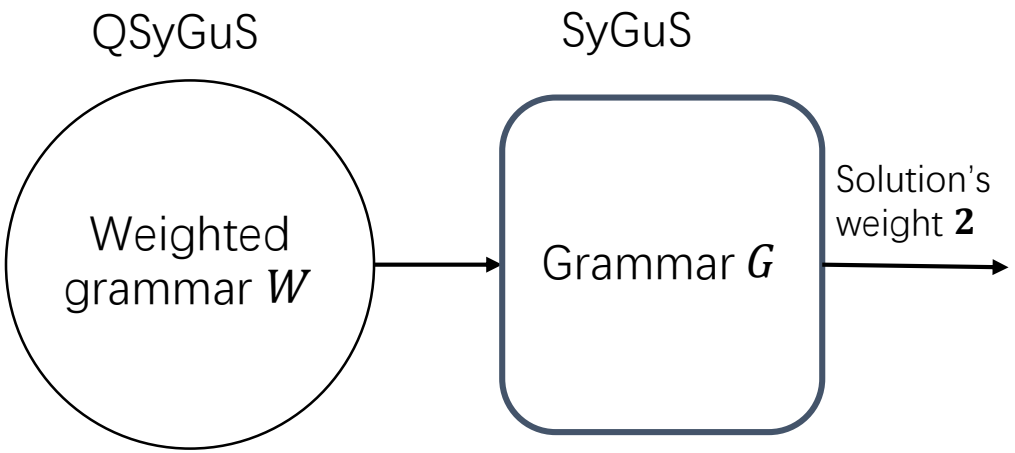
Start := Start+Start	0
| ITE(BExpr,Start,Start)	1
| x | y | 0 | 1	0
BExpr := NOT(BExpr)	0
| Start > Start	0
| Start AND Start	0



Grammar G:

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

BExpr := NOT(BExpr)
| Start > Start
| Start AND Start



Grammar G:

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

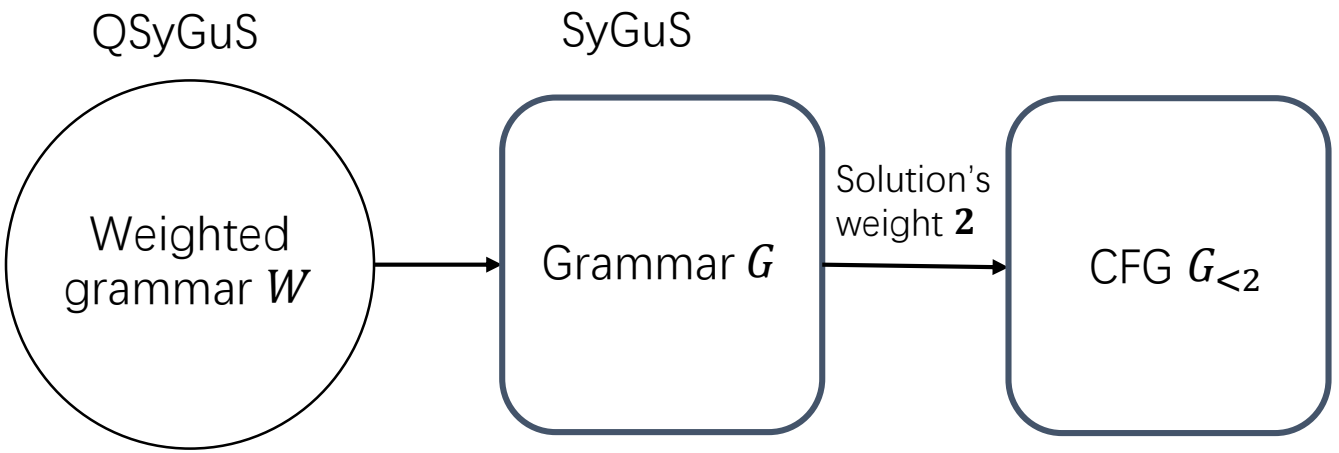
BExpr := NOT(BExpr)
| Start > Start
| Start AND Start

Solution in G :

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

with weight 2

(there are **2** ITE in the solution)



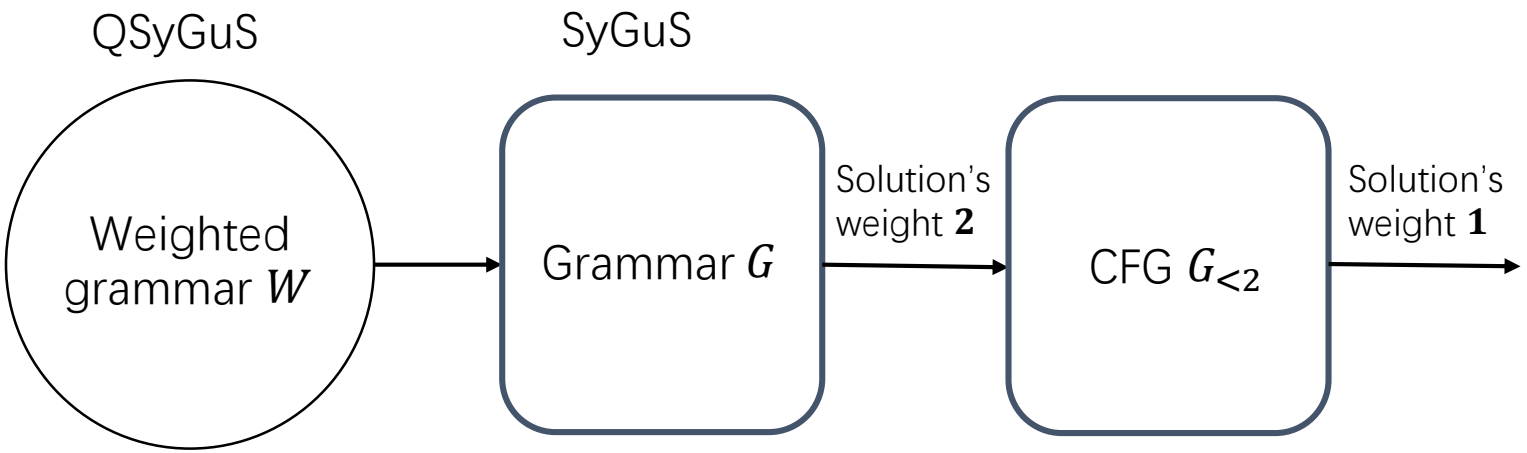
Grammar $G_{<2}$:

Start := Start0 | Start1

Start1 := Start1+Start0
| ITE(BExpr0,Start0,Start0)
| x | y | $\mathbf{0}$ | $\mathbf{1}$

Start0 := Start1+Start0
| x | y | $\mathbf{0}$ | $\mathbf{1}$

BExpr0 := NOT(BExpr0)
| Start0 > Start0
| Start0 AND Start0



Grammar $G_{<2}$:

Start := Start0 | Start1

Start1 := Start1+Start0

| ITE(BExpr0,Start0,Start0)

| x | y | $\mathbf{0}$ | $\mathbf{1}$

Start0 := Start1+Start0

| x | y | $\mathbf{0}$ | $\mathbf{1}$

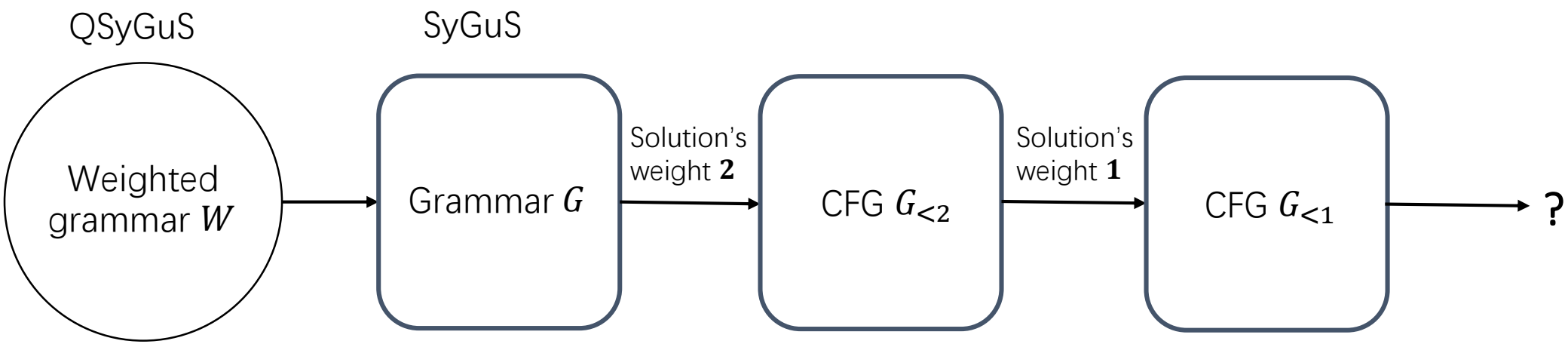
BExpr0 := NOT(BExpr0)

| Start0 > Start0

| Start0 AND Start0

Solution in $G_{<2}$:

ITE $(x > y, x, y)$

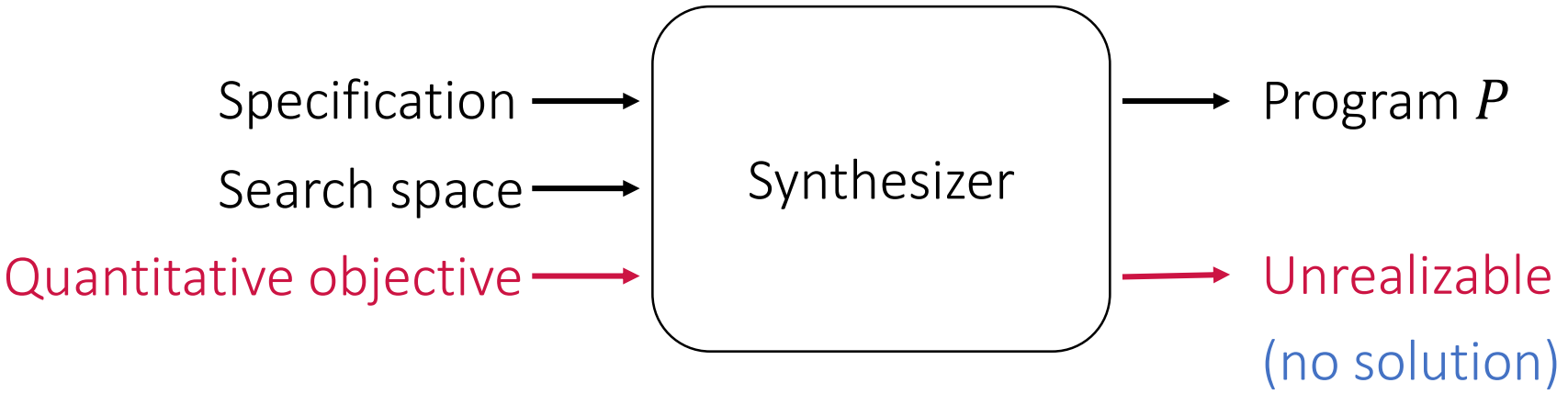


Grammar $G_{<1}$:

$\text{Start} \quad := \text{Start}+\text{Start}$

$\quad\quad\quad\quad | \; x \; | \; y \; | \; 0 \; | \; 1$

Solution $ITE(x > y, x, y)$ is minimized $\Longleftrightarrow$ There is **no solution** in $G_{<1}$



Search-based synthesizer + infinite search space
= Timeout!

Proving a SyGuS problem is unrealizable

Grammar $\mathrm{G}_{<1}$:

$$\mathrm{Start} := \mathrm{Start}+\mathrm{Start} \mid x \mid y \mid 0 \mid 1$$

Specification:

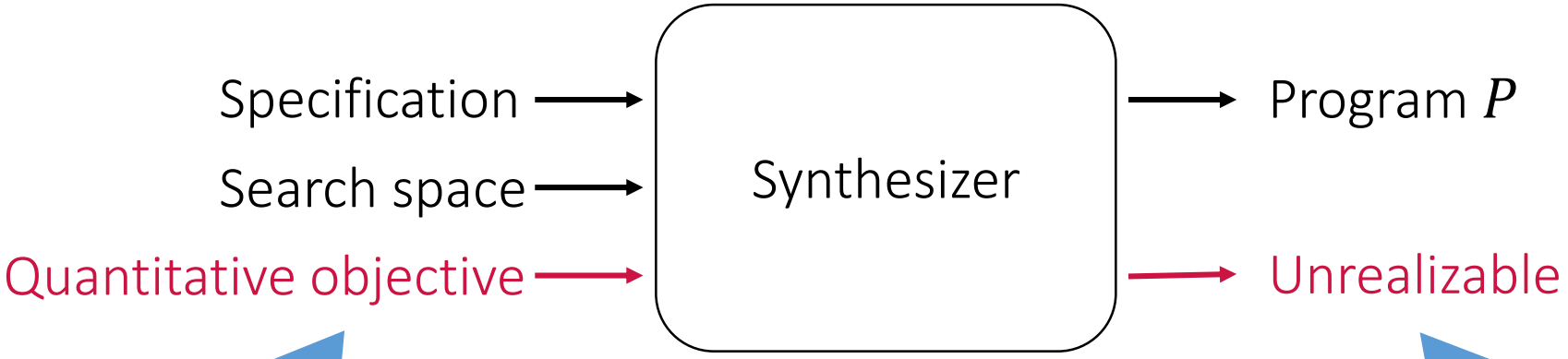
$$P(0,0) = 0 \wedge P(0,1) = 1 \wedge P(1,0) = 1 \wedge P(2,0) = 2$$

```
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);
```

The assertion always holds  The SyGuS problem is unrealizable

```
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);
```

Guarantees in Program Synthesis



More quantitative objectives

- Semantic quantitative objectives
- Resource bounded synthesis

Proving unrealizability beyond SyGuS