Neural-network guided Grammar Filtering For Syntax-Guided Synthesis

Kairo Morton [†], William T. Hallahan*, Elven Shum*,
Ruzica Piskac*, **Mark Santolucito**[††]

*Yale University. [†]MIT, [††]Barnard College, Columbia University

Solvers are Supplemented with Additional Techniques

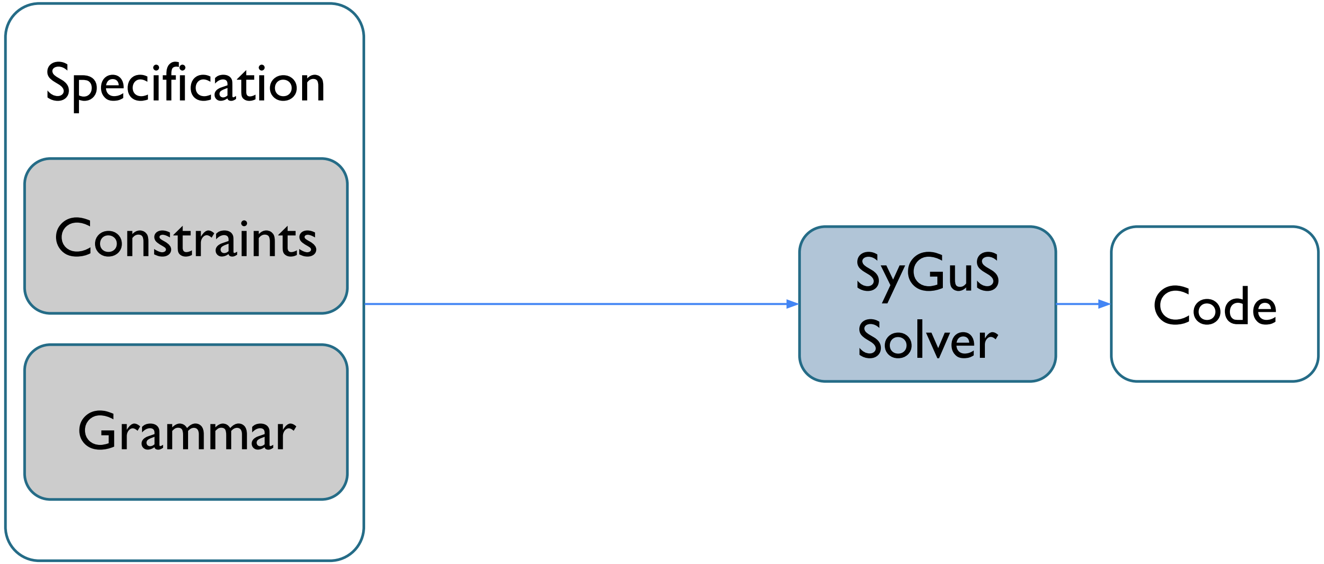
- For CEGQI:
  - Partial quantifier elimination as a preprocessing pass
  - Heuristic solution reconstruction
- For enumerative:
  - Divide-and-conquer **[Alur et al 2017]**
  - Piecewise-Independent Unification (UNIF+PI) **[Barbosa et al FMCAD2019]**
  - Theory-specific constant repair **[Abate et al 2019]**
  - Static grammar minimization and symmetry breaking
  - Variable agnostic enumeration

Solvers are Supplemented with Additional Techniques

- For CEGQI:
  - Partial quantifier elimination as a preprocessing pass
  - Heuristic solution reconstruction
- For enumerative:
  - Divide-and-conquer **[Alur et al 2017]**
  - Piecewise-Independent Unification (UNIF+PI) **[Barbosa et al FMCAD2019]**
  - Theory-specific constant repair **[Abate et al 2019]**
  - Static grammar minimization and symmetry breaking
  - Variable agnostic enumeration

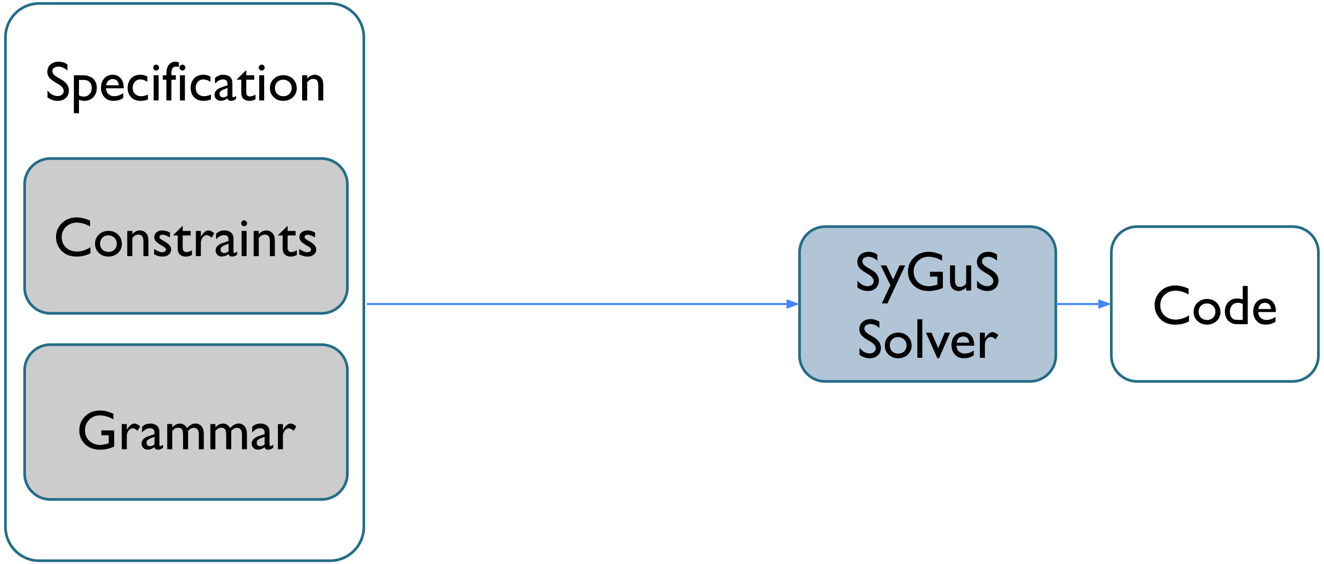
Standard SyGuS synthesis flow

Standard SyGuS synthesis flow



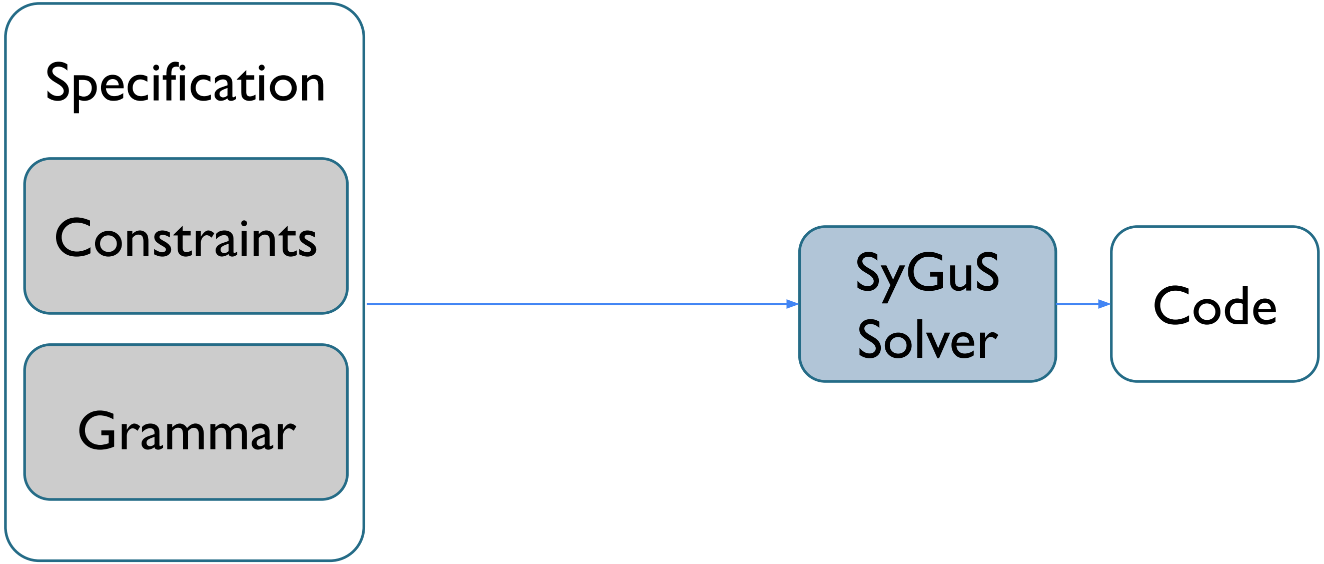
Standard SyGuS synthesis flow

```
(synth-fun f ((x String)) String
  ((Start String (ntString))
  (ntString String
    (x
    (str.++ ntString ntString)
    (str.at ntString ntInt) ))
  (ntInt Int
    (0 1 2 3 4 5
    (+ ntInt ntInt) (- ntInt ntInt)
    (str.len ntString)
    ))

(declare-var x String)
(constraint (= (f "Hello") "HelloHello"))
```



Standard SyGuS synthesis flow

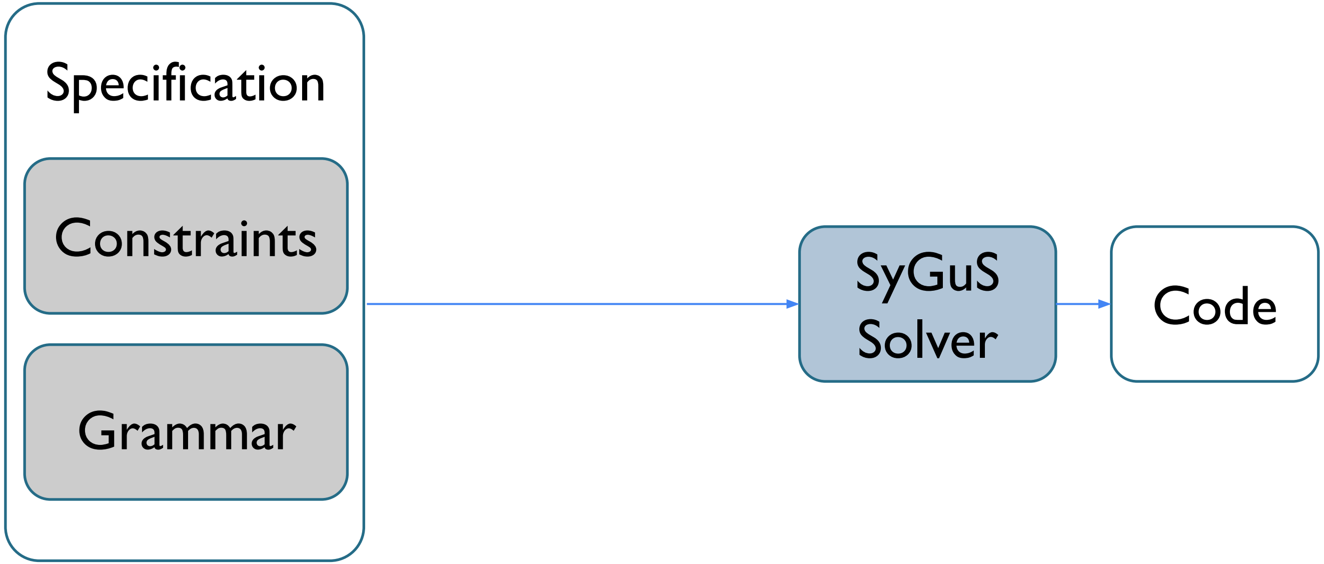
```
(synth-fun f ((x String)) String
  ((Start String (ntString))
  (ntString String
    (x
    (str.++ ntString ntString)
    (str.at ntString ntInt) ))
  (ntInt Int
    (0 1 2 3 4 5
    (+ ntInt ntInt) (- ntInt ntInt)
    (str.len ntString)
    ))

(declare-var x String)
(constraint (= (f "Hello") "HelloHello"))
```




```
(define-fun f ((x String)) String (str.++ x x))
```

Standard SyGuS synthesis flow

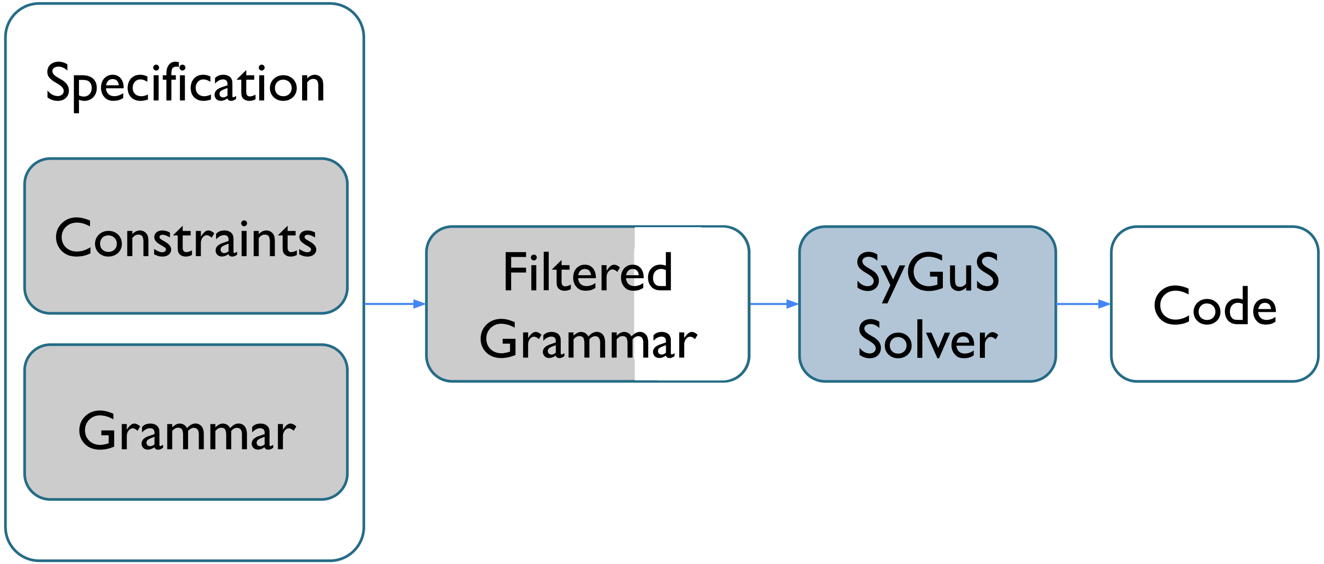
```
(synth-fun f ((x String)) String
  ((Start String (ntString))
  (ntString String
    (x
    (str.++ ntString ntString)
    (str.at ntString ntInt) ))
  (ntInt Int
    (0 1 2 3 4 5
    (+ ntInt ntInt) (- ntInt ntInt)
    (str.len ntString)
    ))

(declare-var x String)
(constraint (= (f "Hello") "HelloHello"))
```



(define-fun f ((x String)) String (str.++ x x))

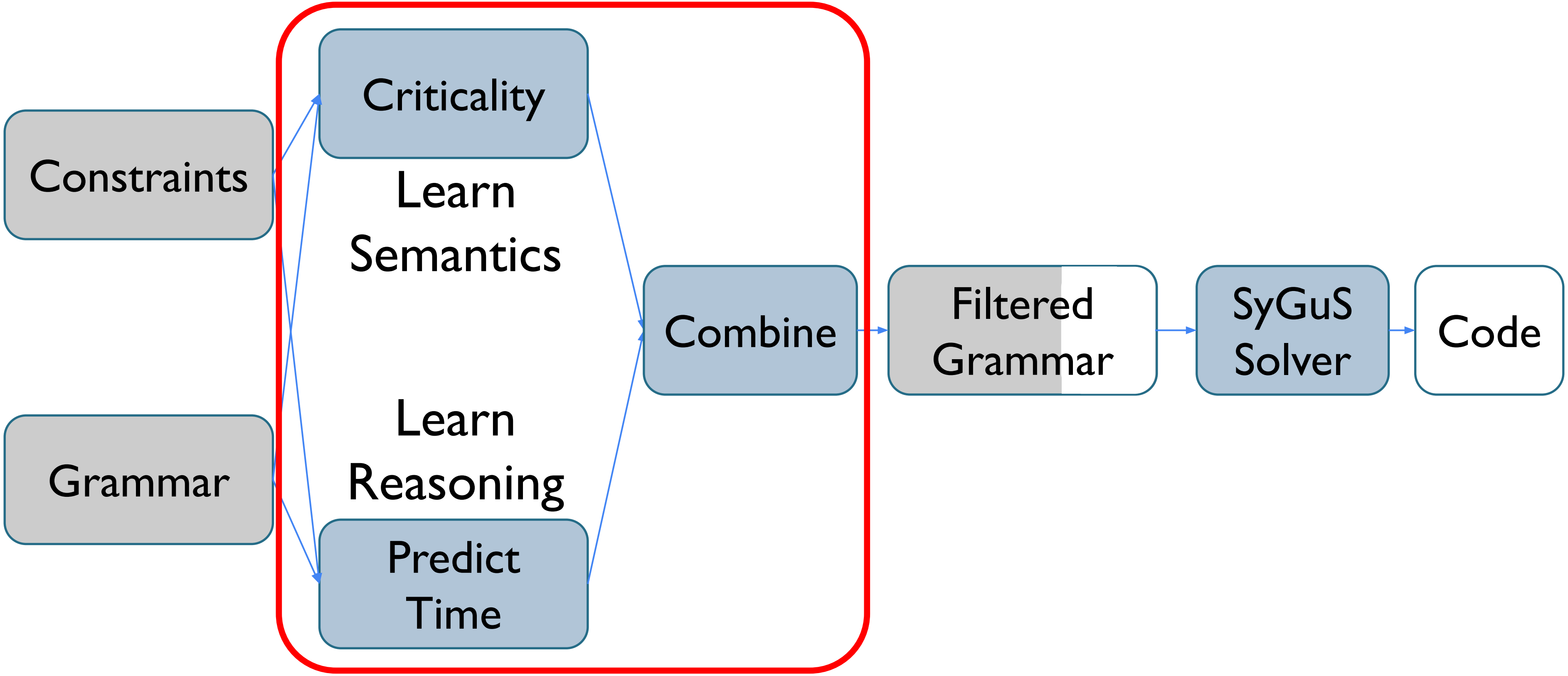
Predicting the grammar to reduce the search space

```
(synth-fun f ((x String)) String
  ((Start String (ntString))
  (ntString String
    (x
    (str.++ ntString ntString)
    (str.at ntString ntInt) ))
  (ntInt Int
    (0 1 2 3 4 5
    (+ ntInt ntInt) (- ntInt ntInt)
    (str.len ntString)
    ))

(declare-var x String)
(constraint (= (f "Hello") "HelloHello"))
```



(define-fun f ((x String)) String (str.++ x x))

Components of grammar prediction



Component Models

Criticality : (IOExample, Terminal) → [0,1]

Predict Time : Terminal → TimeSaved ($\mathbb{R}$)

Combine : $[[0,1], \text{TimeSaved}]^{|G|} \to$ Boolean

Component Models

Criticality : (IOExample, Terminal) → [0,1]

What is the probability a terminal is critical for this constraint?

Predict Time : Terminal → TimeSaved ($\mathbb{R}$)

Combine : [[0,1], TimeSaved] $^{|G|}$ → Boolean

Component Models

Criticality : (IOExample, Terminal) → [0,1]

What is the probability a terminal is critical for this constraint?

Predict Time : Terminal → TimeSaved ($\mathbb{R}$)

What is the estimated time saved by removing this terminal?

Combine : $[[0,1], \text{TimeSaved}]^{|G|} \rightarrow \text{Boolean}$

Component Models

Criticality : (IOExample, Terminal) $\rightarrow [0,1]^{|G|}$

What is the probability a terminal is critical for this constraint?

Predict Time : Terminal $\rightarrow$ TimeSaved ($\mathbb{R}$)

What is the estimated time saved by removing this terminal?

Combine : $[[0,1], \text{TimeSaved}]^{|G|} \rightarrow$ Boolean

Of the terminals that save the most time by being removed, which are least likely to be critical?

Obtaining training data for Criticality

Criticality : (IOExample, Terminal) → [0,1]

Obtaining training data for Criticality

Criticality : (IOExample, Terminal) → [0,1]

1) Generate functions **f** from the grammar (`--sygus-stream` with no constraints)

```
(define-fun f ((x String)) String x)
(define-fun f ((x String)) String (str.++ x x))
(define-fun f ((x String)) String (str.++ x (str.++ x x)))
[...]
```

Obtaining training data for Criticality

Criticality : (IOExample, Terminal) → [0,1]

1) Generate functions **f** from the grammar (`--sygus-stream` with no constraints)
2) Apply each function **f** : String -> String to randomly generated inputs

```
(define-fun f ((x String)) String x)
(define-fun f ((x String)) String (str.++ x x))
(define-fun f ((x String)) String (str.++ x (str.++ x x)))
[…]
```

f(x) = x ++ x ++ x
f("rand") = "randrandrand"
f("AIII") = "AIIIAIIIAIII"
…

Obtaining training data for Criticality

Criticality : (IOExample, Terminal) → [0,1]

1) Generate functions **f** from the grammar (`--sygus-stream` with no constraints)
2) Apply each function **f** : String -> String to randomly generated inputs
3) Build "one-hot vector" of IOExamples and terminals used in **f**

```
(define-fun f ((x String)) String x)
(define-fun f ((x String)) String (str.++ x x))
(define-fun f ((x String)) String (str.++ x (str.++ x x)))
[...]
```

f(x) = x ++ x ++ x
f("rand") = "randrandrand"
f("AIII") = "AIIIAIIIAIII"
…

Evaluation

SyGuS-Comp is a yearly SyGuS solver competition

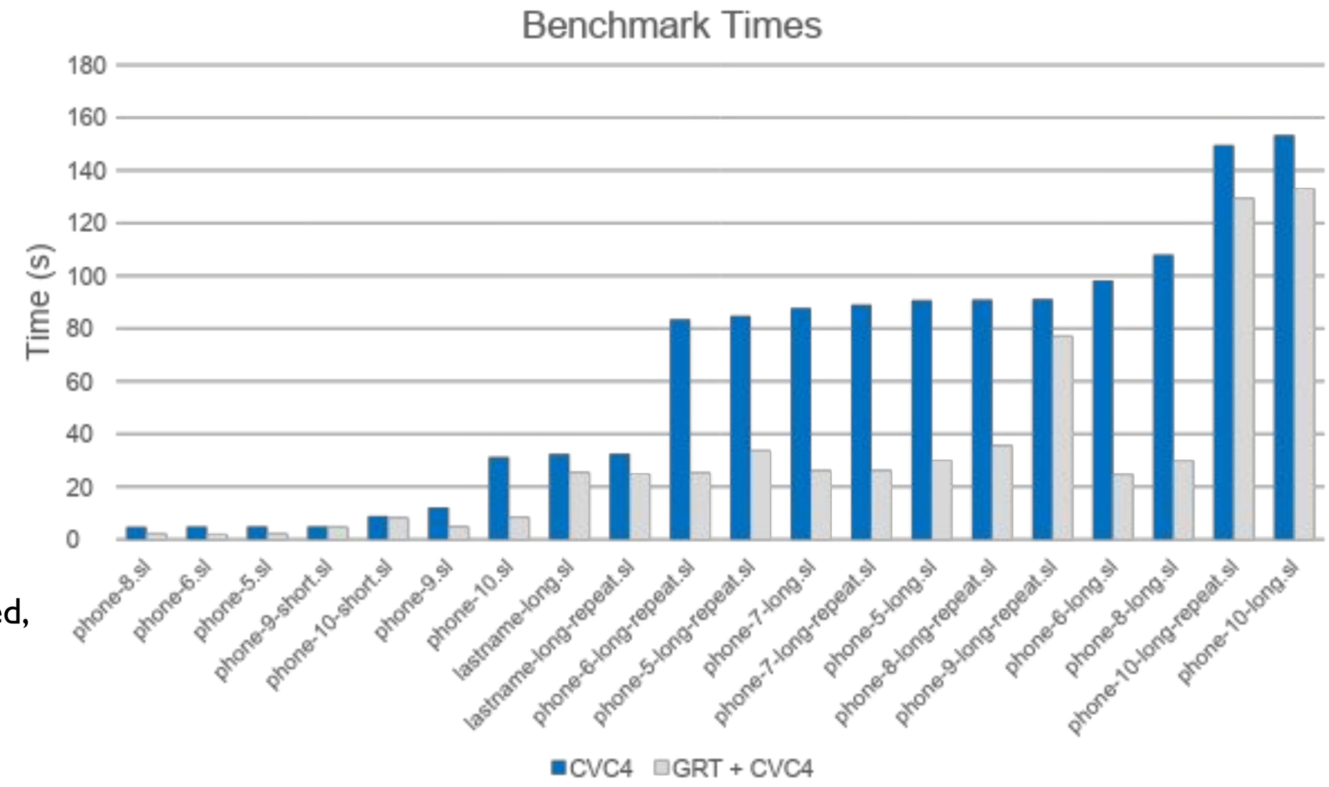
PBE-Strings track has 64 programming by example problems over strings, integers and Booleans

Solvers have an hour to solve each problem

In 2019, CVC4 won the PBE-Strings track

Q: Can GRT + CVC4 outperform CVC4 on the PBE-Strings competition benchmarks?

Evaluation



47.6% faster

When timing data is not considered, and we consider only criticality, synthesis succeeds on only 11 out of the 64 benchmarks.

On **32 out of 64** benchmarks **GRT + CVC4 is faster** than just CVC4.
On 31 benchmarks, performance is the same (within ± .1 second.)
In total, **CVC4 solves 62** of the benchmarks, **GRT + CVC4 solves 63**.
The extra benchmark is solved in 3516.21 seconds.

Future directions

Taking not just criticality, but also potential time savings into account
is needed to effectively filter SyGuS grammars.

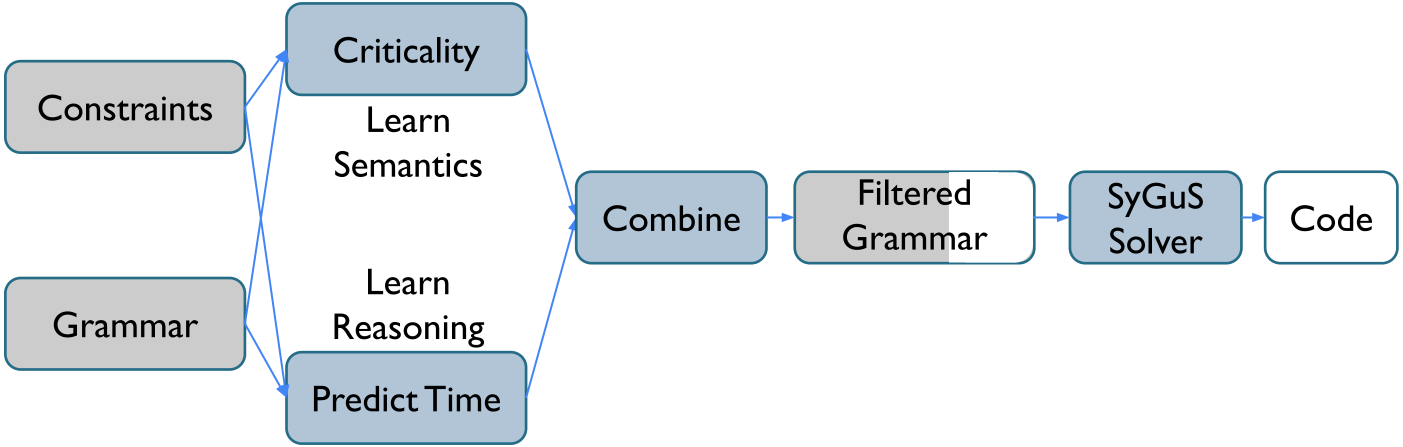
Open questions:

- Would a more sophisticated time prediction heuristic be more effective?
- Could all examples be encoded and given to the NN together,
  to allow learning about interaction between constraints?
- How could this be adapted to work with other common input types (including integers, bitvectors, and algebraic datatypes)?
- How could general first-order logic constraints be encoded in the NN?

Read more!

Kairo Morton, William T. Hallahan, Elven Shum, Ruzica Piskac, Mark Santolucito
Grammar Filtering for Syntax-Guided Synthesis. AAAI 2020

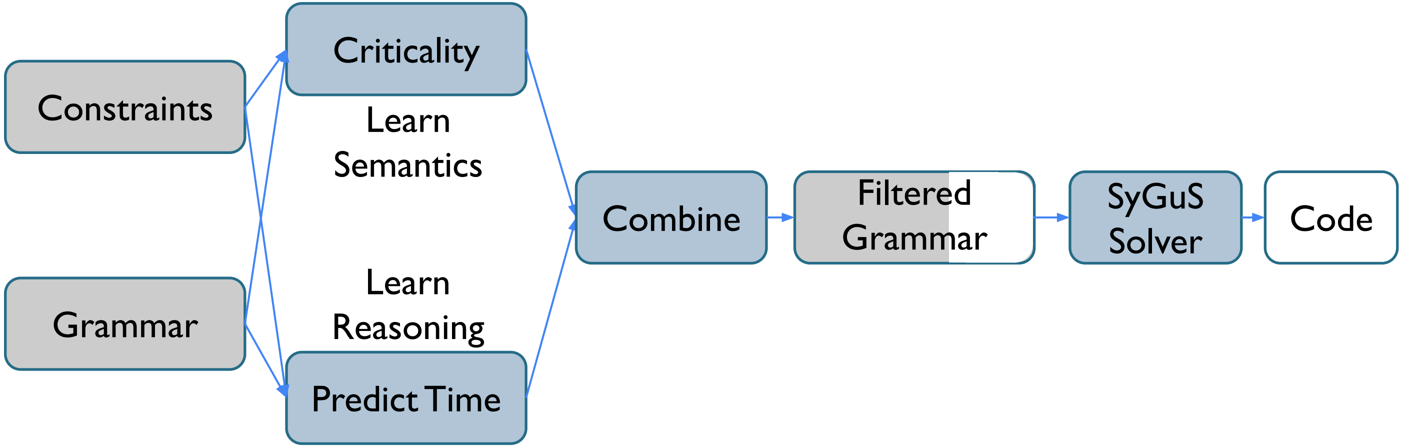
https://github.com/KTMorton/Live_Programming_Research



Read more!

Kairo Morton, William T. Hallahan, Elven Shum, Ruzica Piskac, Mark Santolucito
Grammar Filtering for Syntax-Guided Synthesis. AAAI 2020

https://github.com/KTMorton/Live_Programming_Research



Nicolas Chan, Elizabeth Polgreen, Sanjit A. Seshia
Gradient Descent over Metagrammars for Syntax-Guided Synthesis.
CoRR abs/2007.06677 (2020)