The Second Competition on Syntax-Guided Synthesis

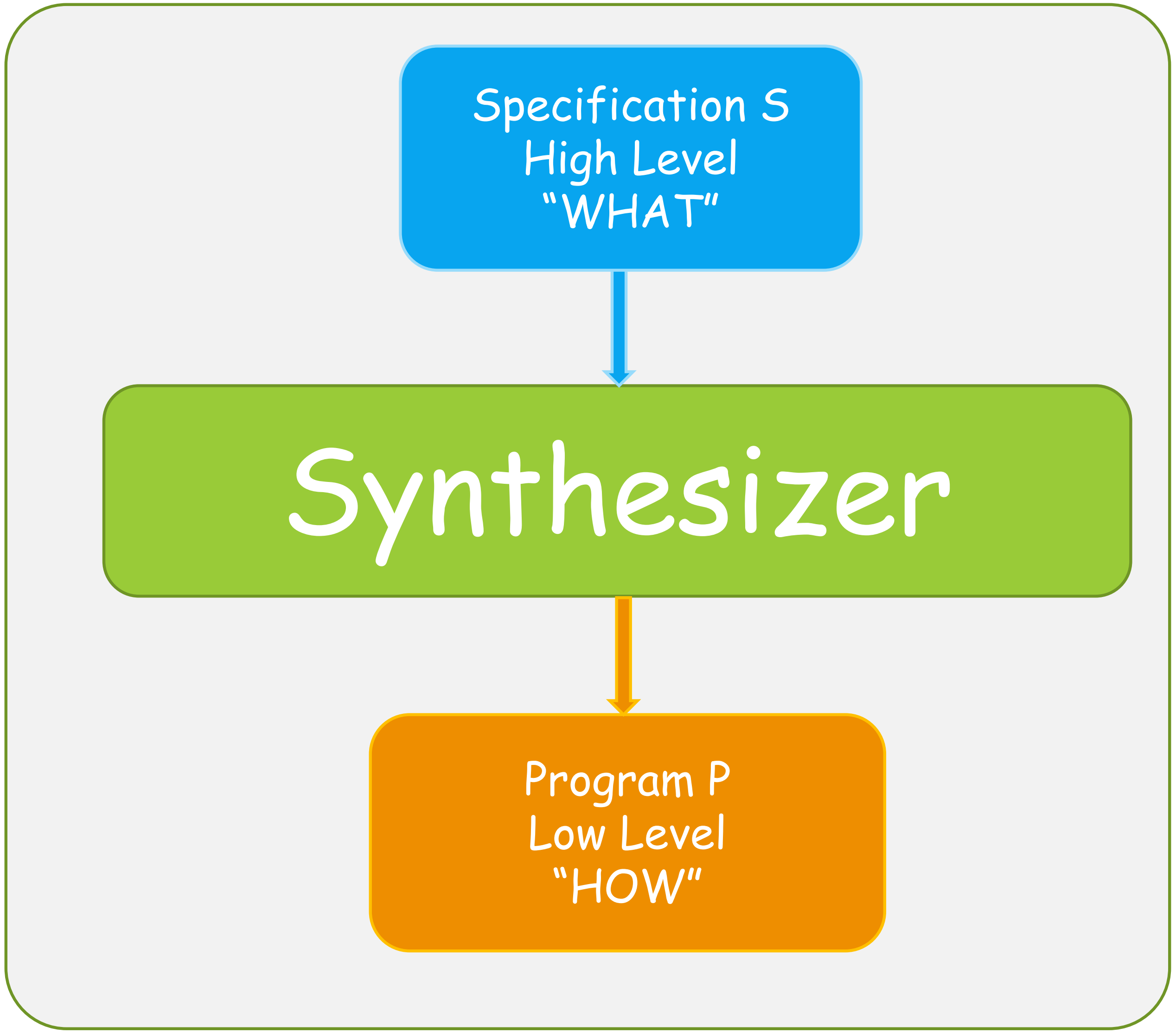
SyGuS
Syntax-Guided Synthesis

Rajeev Alur, Dana Fisman,
Rishabh Singh and Armando Solar-Lezama

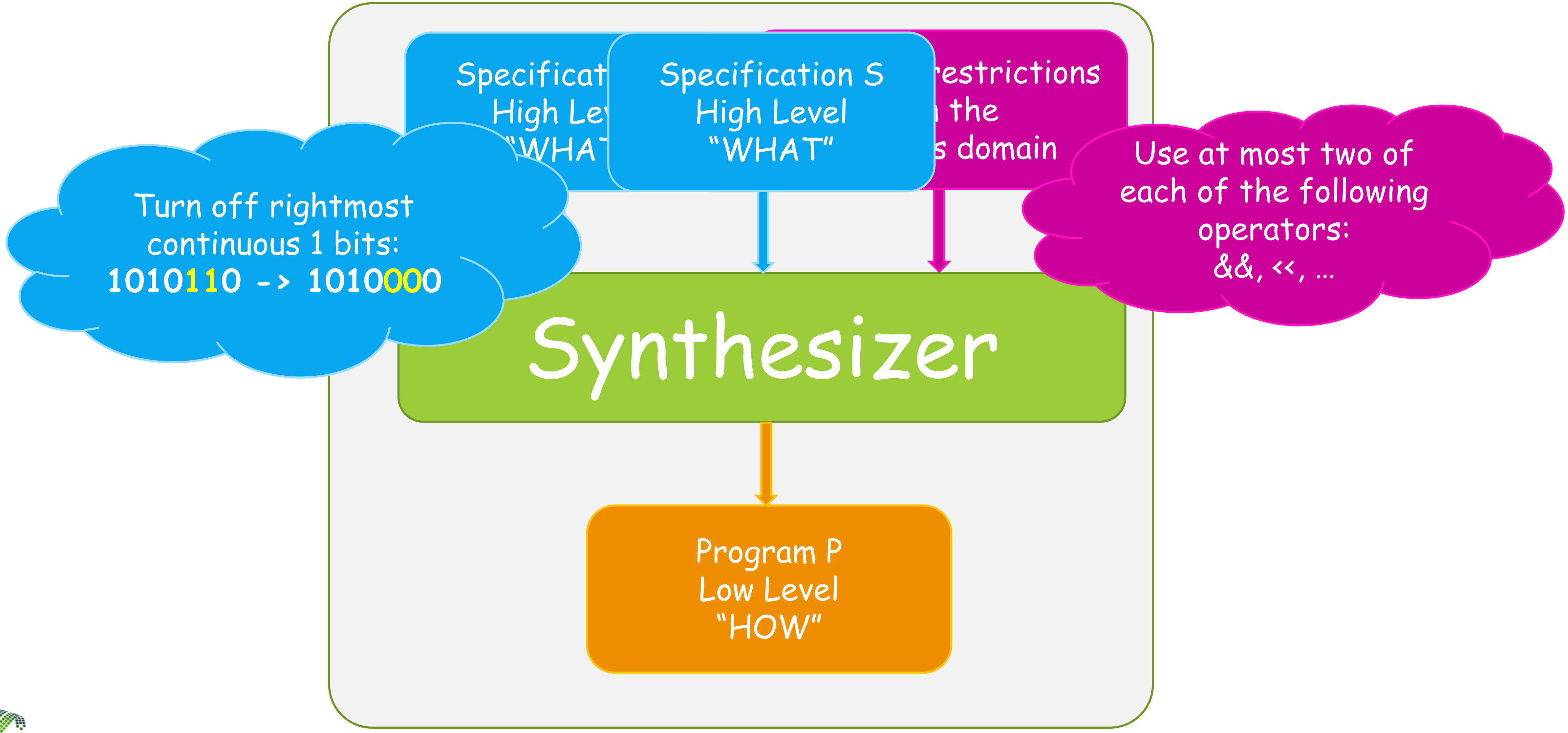
Talk Outline

- Introduction
  - Motivation: recent trends in program synthesis
  - The big picture
- Formalization of Syntax-Guided Synthesis
- SyGuS-COMP'15 Tracks
- Solution Strategies
  - Presentations by Solvers' authors
- SyGuS-COMP'15 Benchmarks
- SyGuS-COMP'15 Competition Results

Program Synthesis



New Trends in Synthesis



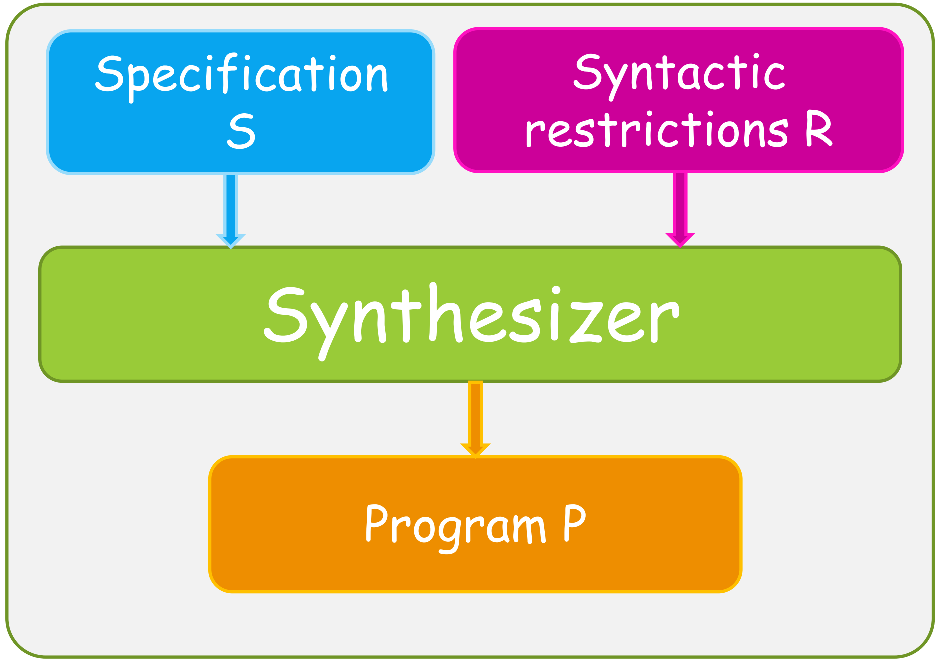
New Trends in Synthesis

Motivation:

- Tractability
- Combine

  human expert insights with

  computers exhaustiveness & rapidness
- Benefit progress SAT & SMT Solvers



Ex 1. Parallel Parking By Sketching

The challenge is finding the parameters

When to start turning?

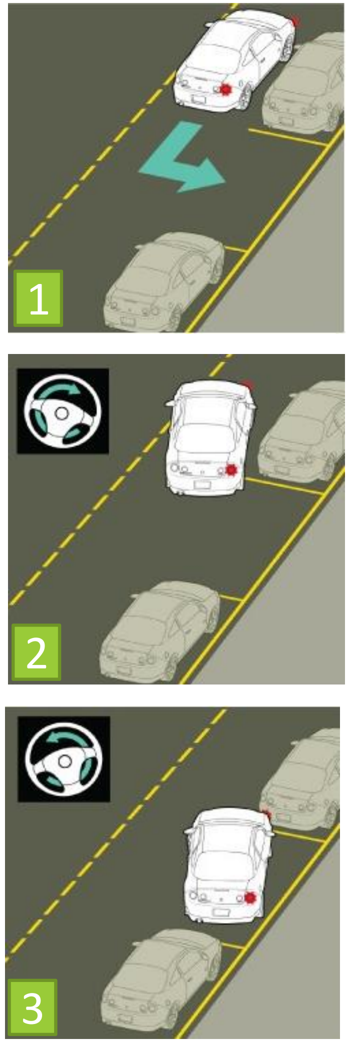
How much to turn?

```
Err = 0.0;
for(t = 0; t<T; t+=dT){
  if(stage==STRAIGHT){ // (1) Backup straight
    if(t > ??) stage= INTURN;
  }

  if(stage==INTURN){  // (2) Turn
    car.ang = car.ang - ??;
    if(t > ??) stage= OUTTURN;
  }

  if(stage==OUTTURN){ // (3) Straighten
    car.ang = car.ang + ??;
    if(t > ??) break;
  }

  simulate_car(car);
  Err += check_collision(car);
}
Err += check_destination(car);
```



Structure of the program is known

[Chaudhuri & Solar-Lezama PLDI 2010]

Ex 2. Optimizing Multiplications

Superoptimizing Compiler

Given a program P, find a “better” equivalent program P’.

```
multiply (x[1,n], y[1,n]) {
   x1 = x[1,n/2];
   x2 = x[n/2+1, n];
   y1 = y[1, n/2];
   y2 = y[n/2+1, n];
   a = x1 * y1;
   b = shift( x1 * y2, n/2);
   c = shift( x2 * y1, n/2);
   d = shift( x2 * y2, n);
   return ( a + b + c + d)
}
```

Replace with equivalent code

with only 3 multiplications

SyGuS

Ex 3. Automatic Invariant Generation

Given a program P and a post condition S,

Find invariants I_1, I_2 with which we can prove program is correct

```
SelecionSort(int A[],n) {
  i1 :=0;
  while(i1 < n-1) {          Invariant:  ???
    v1 := i1;
    i2 := i1 + 1;
    while (i2 < n) {         Invariant:  ???
      if (A[i2]<A[v1])
        v1 := i2 ;
      i2++;
    }
    swap(A[i1], A[v1]);
    i1++;
  }
  return A;
}
```

post: $\forall k : 0 \le k < n \Rightarrow A[k] \le A[k+1]$

SyGuS

Ex 3. Template-Based Invariant Generation

Given a program P and a post condition S

Find invariants $I_1, I_2, \ldots I_k$ with which we can prove program is correct

```
SelecionSort(int A[],n) {
  i1 :=0;
  while(i1 < n−1) {
    v1 := i1;
    i2 := i1 + 1;
    while (i2 < n) {
      if (A[i2]<A[v1])
        v1 := i2 ;
      i2++;
    }
    swap(A[i1], A[v1]);
    i1++;
  }
  return A;
}
```

```
Invariant:
∀k1,k2.  ???  ∧ ???
```

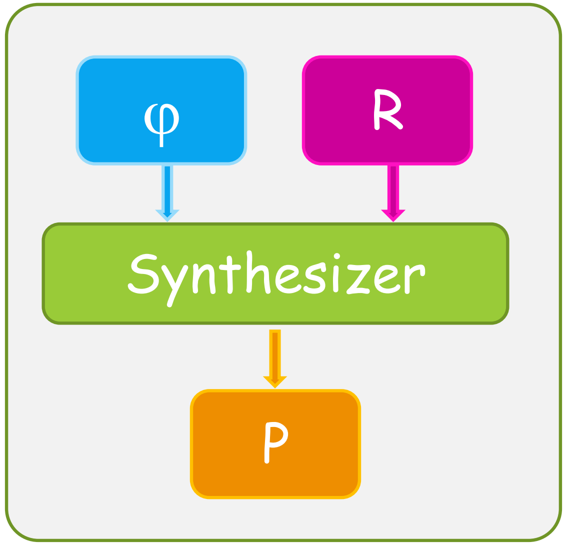
```
Invariant:
??? ∧ ??? ∧
(∀k1,k2. ??? ∧ ???) ∧
(∀k. ??? ∧ ?)
```

Constraint Solver

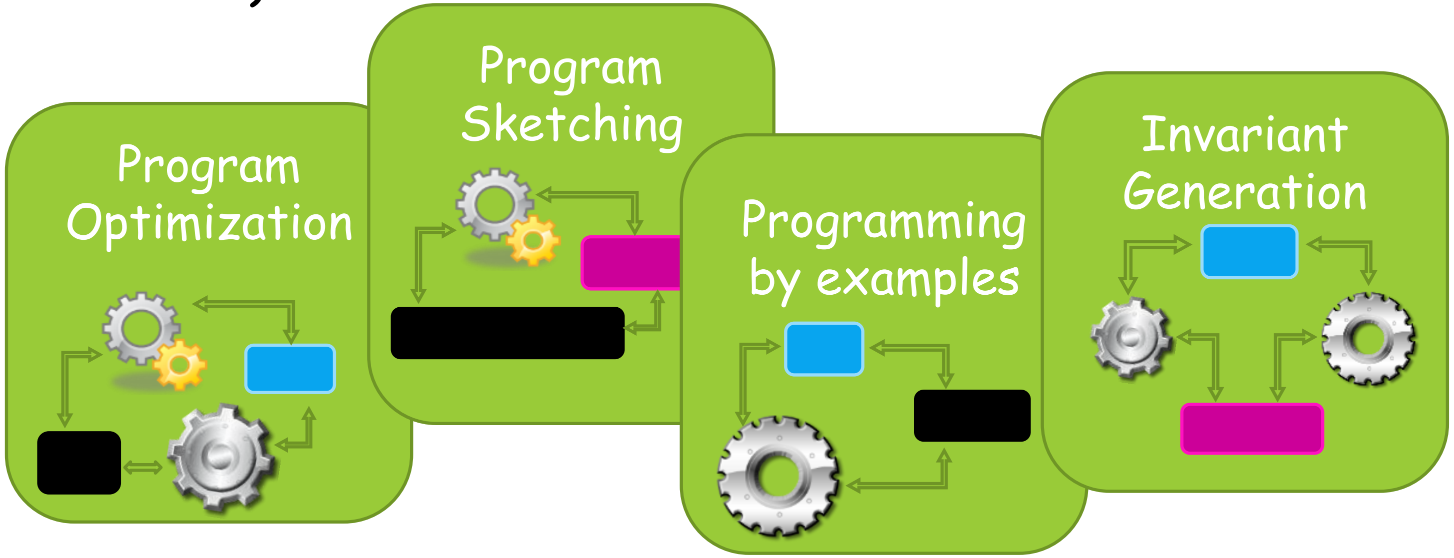
```
post:  ∀k : 0 ≤ k <n ⇒ A[k] ≤ A[k+1]
```

Syntax-Guided Program Synthesis

- Common theme to many recent efforts
  - Sketch (Bodik, Solar-Lezama et al)
  - FlashFill (Gulwani et al)
  - Super-optimization (Schkufza et al)
  - Invariant generation (Many recent efforts…)
  - TRANSIT for protocol synthesis (Udupa et al)
  - Oracle-guided program synthesis (Jha et al)
  - Implicit programming: Scala^Z3 (Kuncak et al)
  - Auto-grader (Singh et al)



But no way to have
a generic solver for all ☹



Talk Outline

- Introduction
  - Motivation: recent trends in program synthesis
  - The big picture
- Formalization of Syntax-Guided Synthesis
- SyGuS-COMP'15 Tracks and Solvers
- Solution Strategies
  - Presentations by Solvers' authors
- SyGuS-COMP'15 Benchmarks
- SyGuS-COMP'15 Competition Results

Slide adopted from a presentation of Viktor Kuncak

The Big Picture

Given list i is P(i) sorted?

Does prog P always sorts correctly?

Given Prog P Spec S	**Assertion Checking:** $P(i) \models S(i)$?	**Program Verification:** $\forall i: \quad P(i) \models S(i)$?
Given only Spec S	**Constraint Programming:** Find o: $o \models S(i)$	**Program Synthesis:** Find P: $\forall i: P(i) \models S(i)$

overall correctness

partial/ intermediate

Given list i, return it sorted

Return a sorting program P

The Big Picture

Given
Prog P
Spec S

Assertion Checking:

$P(i) \models S(i)$?

Program Verification:

$\forall i: \quad P(i) \models S(i)$?

Given
only
Spec S

Return a program P implementing turnoff rightmost 1's

Program Synthesis:

$\exists P: \quad \forall i: P(i) \models S(i)$

Return a program P implementing turnoff rightmost 1's using only so and so operators

Syntax-Guided Synthesis:

$\exists P \in [\![R]\!]: \quad \forall i: P(i) \models S(i)$

SyGuS

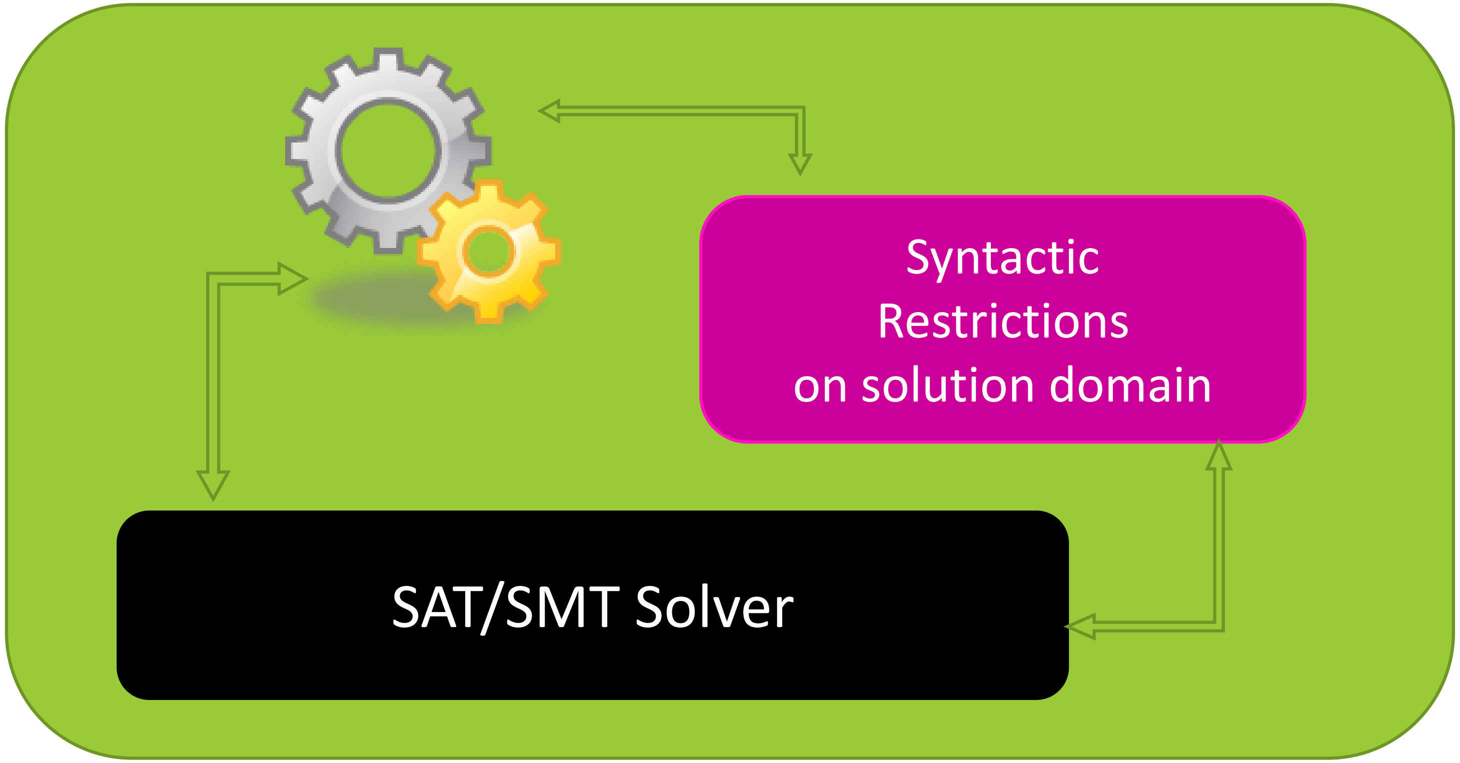
From Satisfiability to Synthesis

Recent trends in program synthesis:

$P(i) \models S(i)$?	$\forall i\colon P(i) \models S(i)$?
$\exists o\colon o \models S(i)$	$\exists P\colon \forall i\colon P(i) \models S(i)$
	$\exists P \in [\![R]\!]\colon \forall i\colon P(i) \models S(i)$

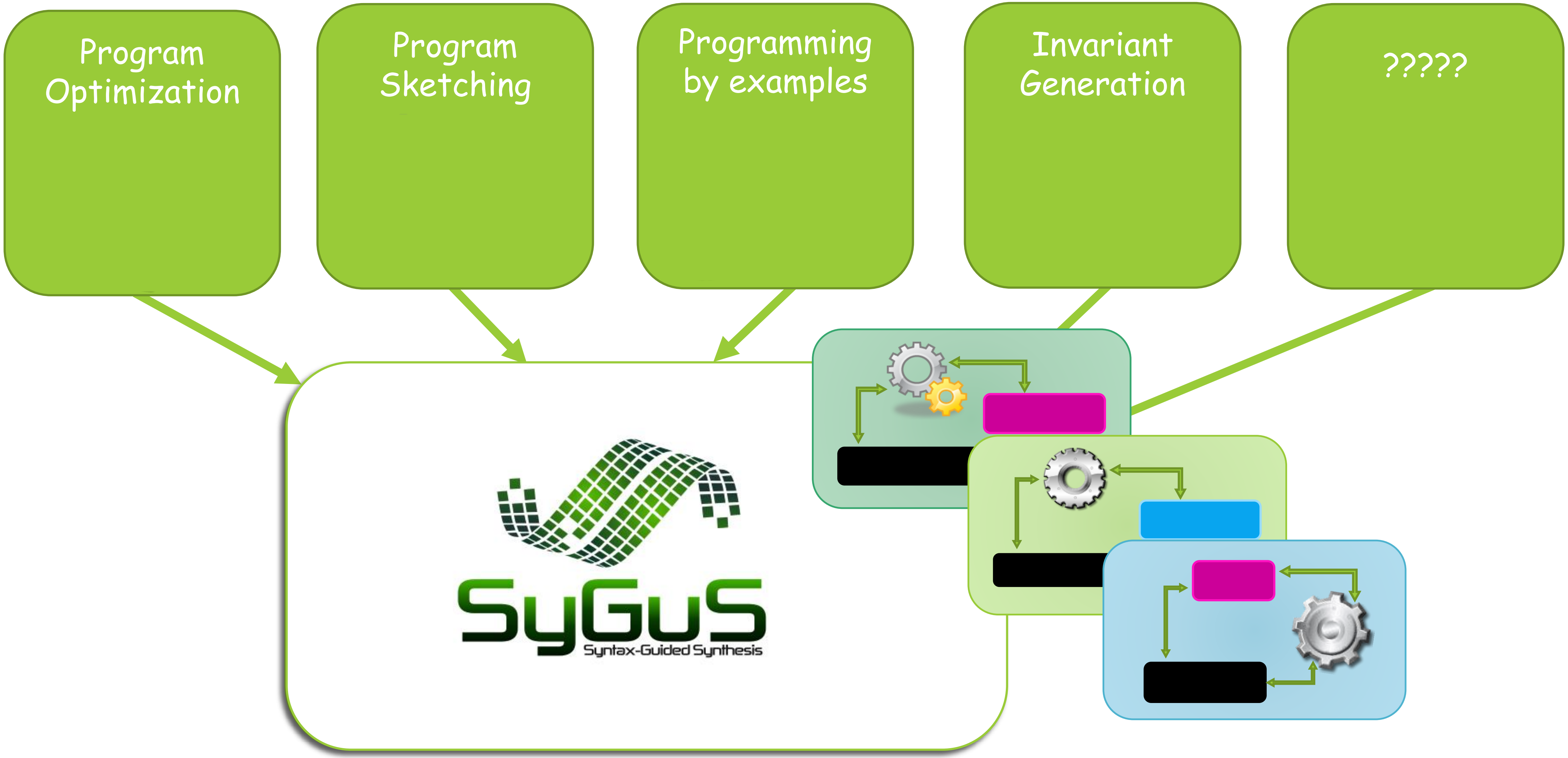
Problem

(verif/synth nature)



SyGuS

SyGuS - The Vision



Talk Outline

- Introduction
  - Motivation: recent trends in program synthesis
  - The big picture
- Formalization of Syntax-Guided Synthesis
- SyGuS-COMP'15 Tracks
- Solution Strategies
  - Presentations by Solvers' authors
- SyGuS-COMP'15 Benchmarks
- SyGuS-COMP'15 Competition Results

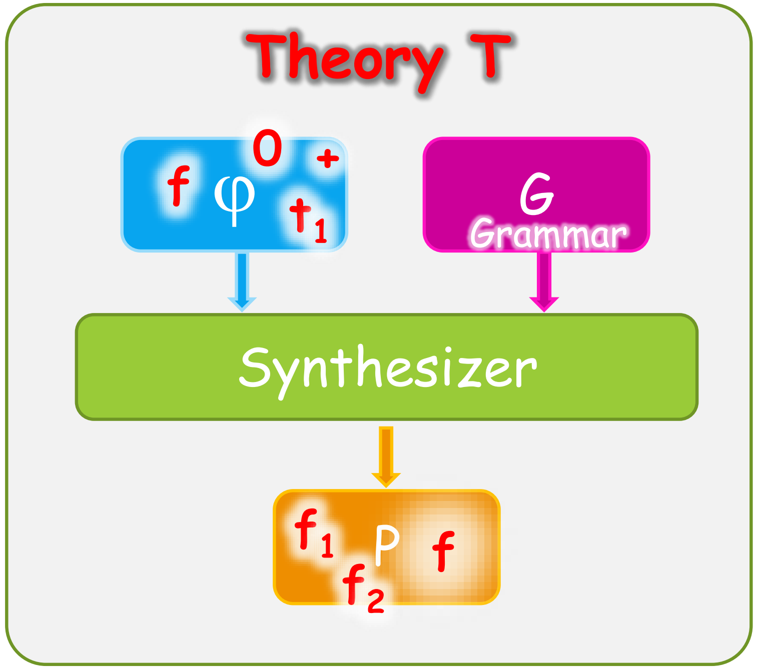
Syntax-Guided Synthesis (SyGuS) Problem

- Fix a background theory T: fixes types and operations
- Function to be synthesized: name f along with its type
  - General case: multiple functions to be synthesized
- Inputs to SyGuS problem:
  - Specification φ

    Typed formula using symbols in T + symbol f
  - Context-free grammar G

    Characterizing the set of allowed expressions ⟦G⟧ (in theory T)
- Computational problem:

  Find expression e in ⟦G⟧ such that φ[f/e] is valid (in theory T)



SyGuS - formalization example

B

Name and type of the function to be synthesized

Grammar describing the syntactic restrictions

Semantic restrictions (correctness criteria)

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
  ((Start Int (x y 0 1
               (+ Start Start)
               (- Start Start)
                  (ite StartBool Start Start)))
   (StartBool Bool ((and StartBool StartBool)
                    (or StartBool StartBool)
                    (not StartBool)
                    (<= Start Start))))
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
(check-synth)
```

SyGuS

Talk Outline

- Introduction
  - Recent trends in program synthesis (the problem)
  - The big picture
  - Formalization of Syntax-Guided Synthesis
- SyGuS-COMP'15 tracks
- Solution Strategies
- Benchmarks
- Competition Results

SyGuS-COMP'15 Tracks

- General Track
  - Background theory LIA or BV
  - **Arbitrary grammar** (as defined in the benchmark)
- Linear Integer Arithmetic Track
  - Background theory **LIA**
  - **No grammar restrictions** (any LIA expression is allowed)
- Invariant Synthesis Track
  - Background theory LIA
  - No grammar restrictions
  - **Special constructs to describe invariant synthesis** (pre-condition, transition, post-condition)

SyGuS

SyGuS LIA track example

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int )
```

```
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
(check-synth)
```

SyGuS

SyGuS-COMP'15 Tracks

- General Track
  - Background theory LIA or BV
  - **Arbitrary grammar** (as defined in the benchmark)
- Linear Integer ArithmeticTrack
  - Background theory **LIA**
  - **No grammar restrictions** (any LIA expression is allowed)
- Invariant Synthesis Track
  - Background theory LIA
  - No grammar restrictions
  - **Special constructs to describe invariant synthesis** (pre-condition, transition, post-condition)

SyGuS

SyGuS Inv track example

```
(set-logic LIA)
(synth-inv inv-f ((x Int) (y Int) (b Bool)))
(declare-primed-var b Bool)
(declare-primed-var x Int)
(declare-primed-var y Int)
(define-fun pre-f ((x Int) (y Int) (b Bool)) Bool
                    (and (and (>= x 5) (<= x 9)) (and (>= y 1) (<= y 3))))
(define-fun trans-f ((x Int) (y Int) (b Bool) (x! Int) (y! Int) (b! Bool)) Bool
                    (and (and (= b! b) (= y! x)) (ite b (= x! (+ x 10)) (= x! (+ x 12)))))
(define-fun post-f ((x Int) (y Int) (b Bool)) Bool
                    (< y x))
(inv-constraint inv-f pre-f trans-f post-f)
(check-synth)
```

SyGuS Inv track example

```
(set-logic LIA)

(synth-inv inv-f ((x Int) (y Int) (b Bool)))

(declare-primed-var b Bool)

(declare-primed-var x Int)

(declare-primed-var y Int)

(define-fun pre-f ((x Int) (y Int) (b Bool)) Bool
                    (and (and (>= x 5) (<= x 9)) (and (>= y 1) (<= y 3))))

(define-fun trans-f ((x Int) (y Int) (b Bool) (x! Int) (y! Int) (b! Bool)) Bool
                    (and (and (= b! b) (= y! x)) (ite b (= x! (+ x 10)) (= x! (+ x 12)))))

(define-fun post-f ((x Int) (y Int) (b Bool)) Bool
                    (< y x))

(inv-constraint inv-f pre-f trans-f post-f)

(check-synth)
```

```
(constraint (=> (pre-f x y b) (inv-f x y b)))
(constraint (=> (and (inv-f x y b)
                     (trans-f x y b x! y! b!))
                (inv-f x! y! b!)))
(constraint (=> (inv-f x y b) (post-f x y b)))
```

SyGuS-COMP'15 Solvers

General Track

- CVC4-1.5-sygus
- Enumerative
- Stochastic
- Sketch-2014
- Sketch-AC
- Sosy Toast
- Sosy Toast v2

LIA Track

- CVC4-1.5-sygus
- Alchemist CSDT
- Alchemist CS

Invariants Track

- CVC4-1.5-sygus
- ICE DT
- Alchemist CS

Talk Outline

- Introduction
  - Recent trends in program synthesis (the problem)
  - The big picture
  - Formalization of Syntax-Guided Synthesis
- SyGuS-COMP'15 tracks and solvers
- Solution Strategies
- Benchmarks
- Competition Results

Solving SyGuS

$P(i) \models S(i)$?

$\forall i$: $P(i) \models S(i)$?

$\exists o$: $o \models S(i)$

$\exists P$: $\forall i$: $P(i) \models S(i)$

$\exists P \in [\![G]\!]$: $\forall i$: $P(i) \models S(i)$

- Is SyGuS same as solving SMT formulas with quantifier alternation?

$$\exists P \in [\![G]\!]: \forall i: P(i) \models S(i)$$

- SyGuS can sometimes be reduced to Quantified-SMT, but not always
  - Set $[\![G]\!]$ is all linear expressions over input vars x, y

    SyGuS reduces to $\exists a,b,c.\ \forall x,y.\ \varphi$ [f / ax+by+c]
  - Set $[\![G]\!]$ is all conditional expressions

    SyGuS cannot be reduced to deciding a formula in LIA
- Syntactic structure of the set $[\![G]\!]$ of candidate implementations can be used effectively by a solver
- Existing work on solving Quantified-SMT formulas suggests solution strategies for SyGuS

SyGuS

Running Example

- Specification:

$$(x \leq f(x,y))\ \&$$
$$(y \leq f(x,y))\ \&$$
$$(f(x,y) = x \mid f(x,y) = y)$$

- Syntactic Restrictions:

  all expressions built from x,y,0,1,

  <=, =, =>, +,

  If-Then-Else

SyGuS as Active Learning (CEGIS)

[Solar-Lezama et al.]
[Seshia et al.]

- Concrete inputs I for learning

  $f(x,y) = \{ (x=a_0, y=b_0), (x=a_1, y=b_1), \ldots\}$

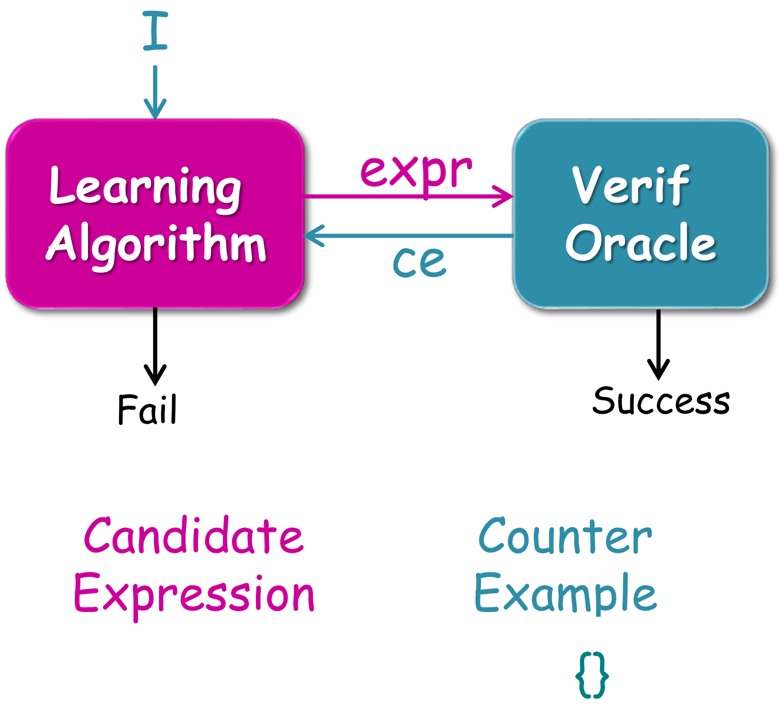
- Learning algorithm proposes candidate expression e such that φ[f/e] holds for all values in I

- Check if φ[f/e] is valid for all values using SMT solver

- If valid, then stop and return e

- If not, let (x=a, y=b, ….) be a counter-example (satisfies ~ φ[f/e])

- Add (x=a, y=b) to tests I for next iteration

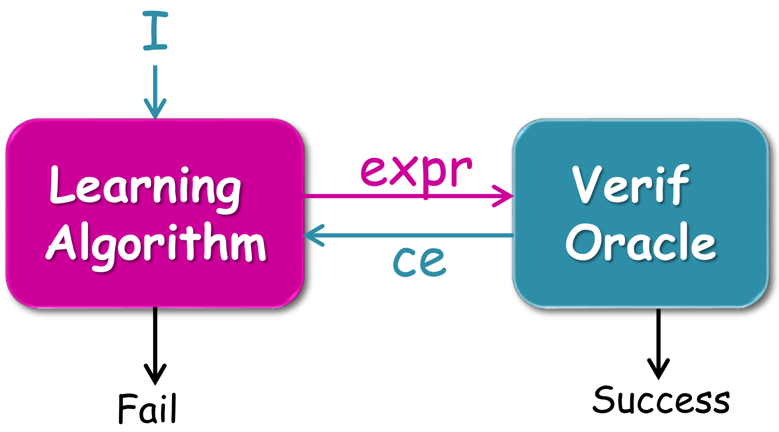


Enumerative CEGIS [Udupa et al.]

- Find an expression consistent with a given set of concrete examples

- Enumerate expressions in increasing size, and evaluate each expression on all concrete inputs to check consistency

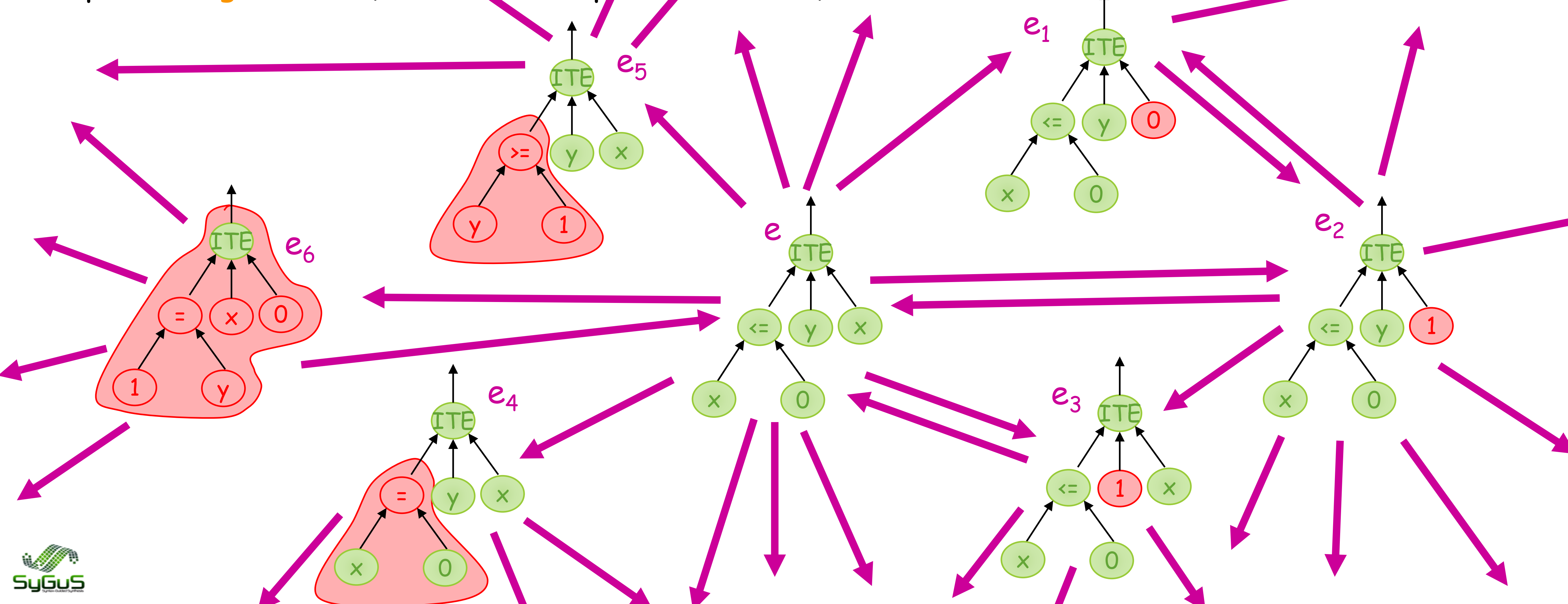
- Key optimization for efficient pruning of search space:
  - Expressions e_1 and e_2 are equivalent if $e_1(a,b)=e_2(a,b)$ on all concrete values $(x=a,y=b)$ in Examples
  - E.g. If-Then-Else ($0 \le x$, e_1, e_2) considered equivalent to e_1 if in current set of Examples x has only non-negative values
  - Only one representative among equivalent sub-expressions needs to be considered for building larger expressions



Stochastic [adaptation of Schufza et al.]

Idea:

Find desired expression e by probabilistic walk on graph where nodes are expressions and edges capture single-edits (for a fixed expression size n)



Stochastic

- Metropolis-Hastings Algorithm: Given a probability distribution P over domain X, and an ergodic Markov chain over X, samples from X

- Because the graph is strongly connected, we can reach each node with some probability

- Let Score(e) be the "Extent to which e meets the spec φ"
  Having $P(e) \propto Score(e)$ we increase the chances of getting to expressions with better score.
  To escape "local minima" we allow with some probability moving to expressions with lower score.

- Specific choice of score:
  For a given set I of concrete inputs, $Score(e) = \exp(-\frac{1}{2}\, Wrong(e))$
  where Wrong(e) = No of examples in I for which $\sim \varphi\,[f/e]$

- Score(e) is large when Wrong(e) is small
  => Expressions e with Wrong(e) = 0 more likely to be chosen in the limit than any other expr

SyGuS

Stochastic

- Initial candidate expression e sampled uniformly from E_n

- When $Score(e) = 1$, return e

- Pick node v in parse tree of e uniformly at random.

  Replace subtree rooted at v with subtree of same size, sampled uniformly

- With probability $\min\{ 1, Score(e')/Score(e) \}$, replace e with e'

- Outer loop responsible for updating expression size n

Solvers Presentations

- *Andrew Reynolds:*

  CVC4-1.5 sygus

- *Daniel Neider:*

  ICE and Alchemist

- *Heinz Riener:*

  Sosy Toast

SyGuS

Participating Solvers

- CVC4-1.5 Sygus Solver (Andrew Reynolds, Viktor Kuncak, Cesare Tinelli, Clark Barrett, Morgan Deters, Tim King)
- ICE-DT Solver (Daniel Neider, P. Madhusudan, Pranav Garg)
- Skech-AC (Jinseong Jeon, Xiaokang Qiu, Armando Solar-Lezama, Jeff Foster)
- Sosy Toast, Sosy Toast Variant2 (Heinz Riener, Ruediger Ehlers)
- Enumerative Solver (Abhishek Udupa)
- Stochastic Solver (Mukund Raghothaman)
- Alchemist CSDT (Shambwaditya Saha, Daniel Neider, P. Madhusudan)
- Alchemist CS (Daniel Neider, Shambwaditya Saha, P. Madhusudan)
- Sketch-Based (Rishabh Singh, Armando Solar-Lezama)

SyGuS

Track Participation

Solver	GEN	LIA	INV
Sosy Toast			
Sosy Toast v2			
CVC4 1.5			
Enumerative			
Stochastic			
AlchemistCSDT			
AlchemistCS			
ICE DT			
Sketch-AC			
Sketch-based			

Benchmarks

- Hacker's Delight (bit manipulation problems)
- Invariant Generation (for program verification)
- Vehicle Control (autonomous cars on routes with an intersection point)
- Conditional integer arithmetic (complex branching structure)
- ICFP (bit vector algorithms from functional programming competition)
- Integer Arithmetic (Shambwaditya Saha)
- Motion Planning (Sarah Chasins)
- Invariant Synthesis (Pranav Garg)
- Compiler Optimization (Nissim Ofek)

Evaluation Setup

StarExec Platform

Timeout of **3600s**

4 cores machines

256 GB RAM

General Track (309)

Enumerative Solver

139

CVC4-1.5

179

Stochastic Solver

106

LIA Track (73)

Alchemist
CSDT

47

CVC4-1.5

70

Alchemist
CS

33

INV Track (67)

Alchemist CS

53

ICE DT

57

CVC4-1.5

29

Some Stories

The Story of Expression Sizes

Expression Sizes

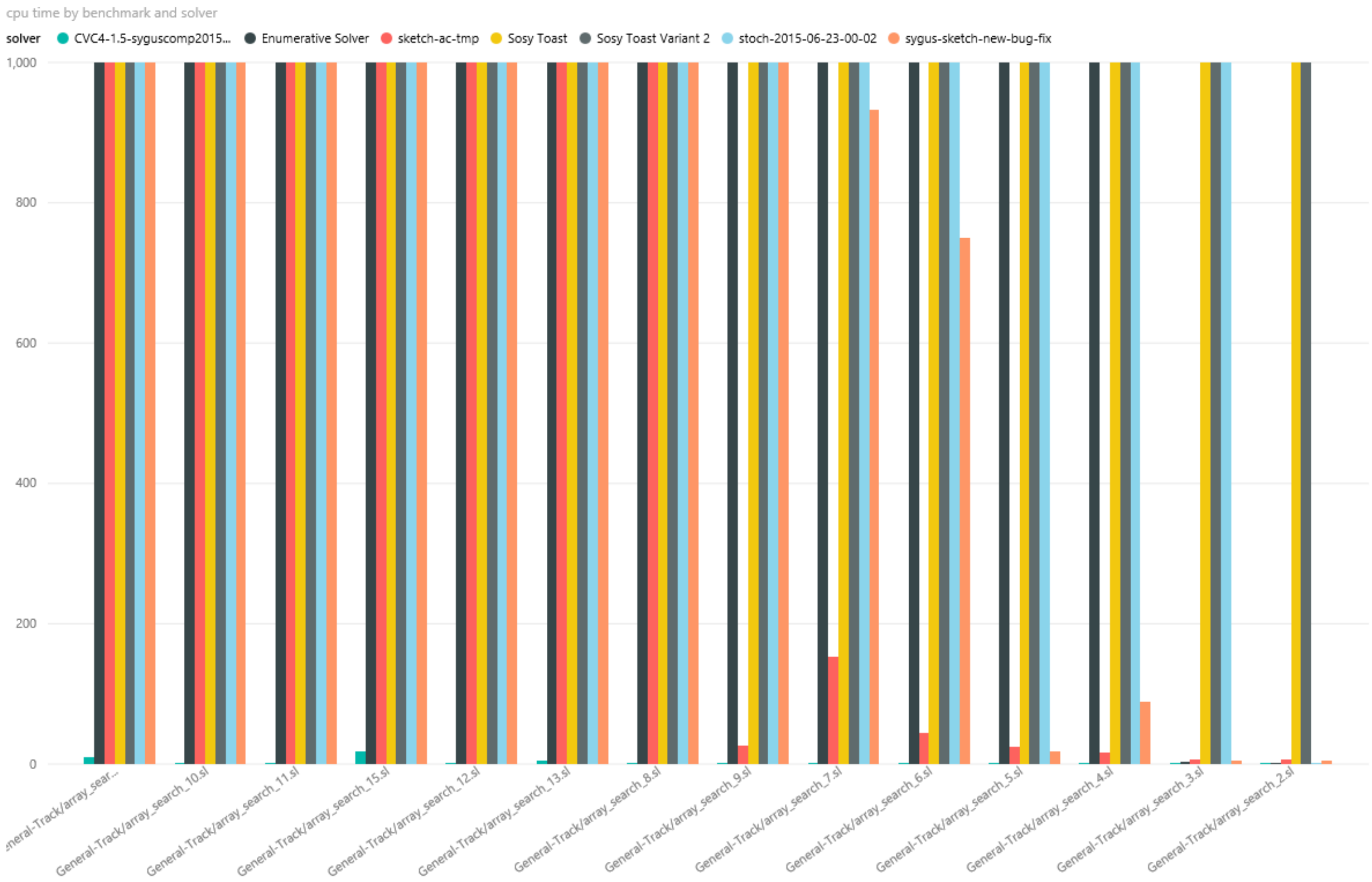
GENERAL Track (309)			
Solver	#Solved	Total-expr-size	Average-expr-size
CVC4-1.5-v4	179	6130193	34246.89
Enumerative Solver	139	1664	11.97
stoch-2015-06-23-00-02	106	2494	23.53
sygus-sketch-new-bug-fix	87	1919	22.06
sketch-ac	80	1749	21.86
Sosy Toast Variant 2	53	545	10.28
Sosy Toast	50	484	9.68

CVC4's Large Expression Sizes

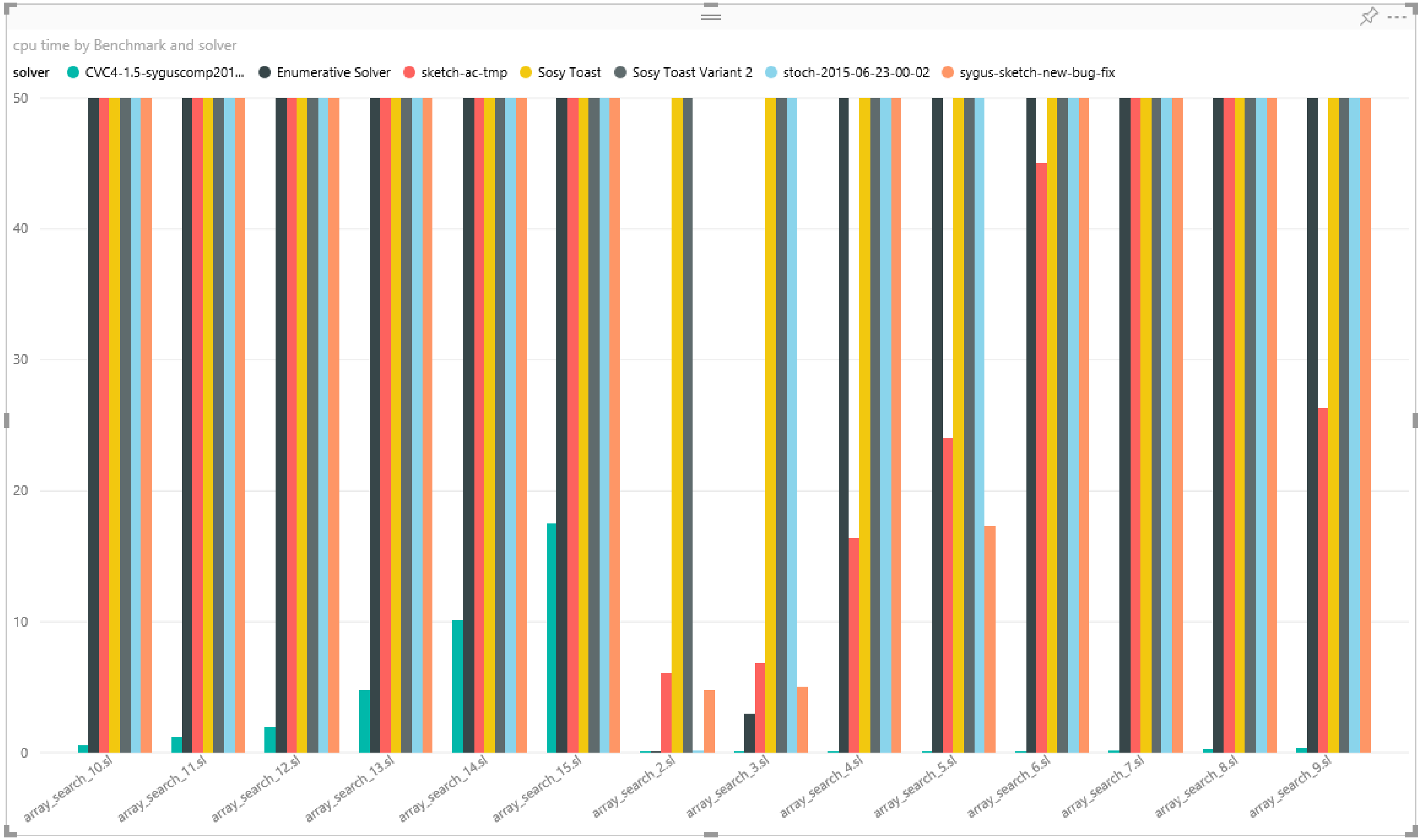
GENERAL Track (309)			
Solver	#Solved	Total-expr-size	Average-expr-size
CVC4-1.5-v4	179	6130193	34246.89
Enumerative Solver	139	1664	11.97
stoch-2015-06-23-00-02	106	2494	23.53
sygus-sketch-new-bug-fix	87	1919	22.06
sketch-ac	80	1749	21.86
Sosy Toast Variant 2	53	545	10.28
Sosy Toast	50	484	9.68

The story of Array-search Benchmarks

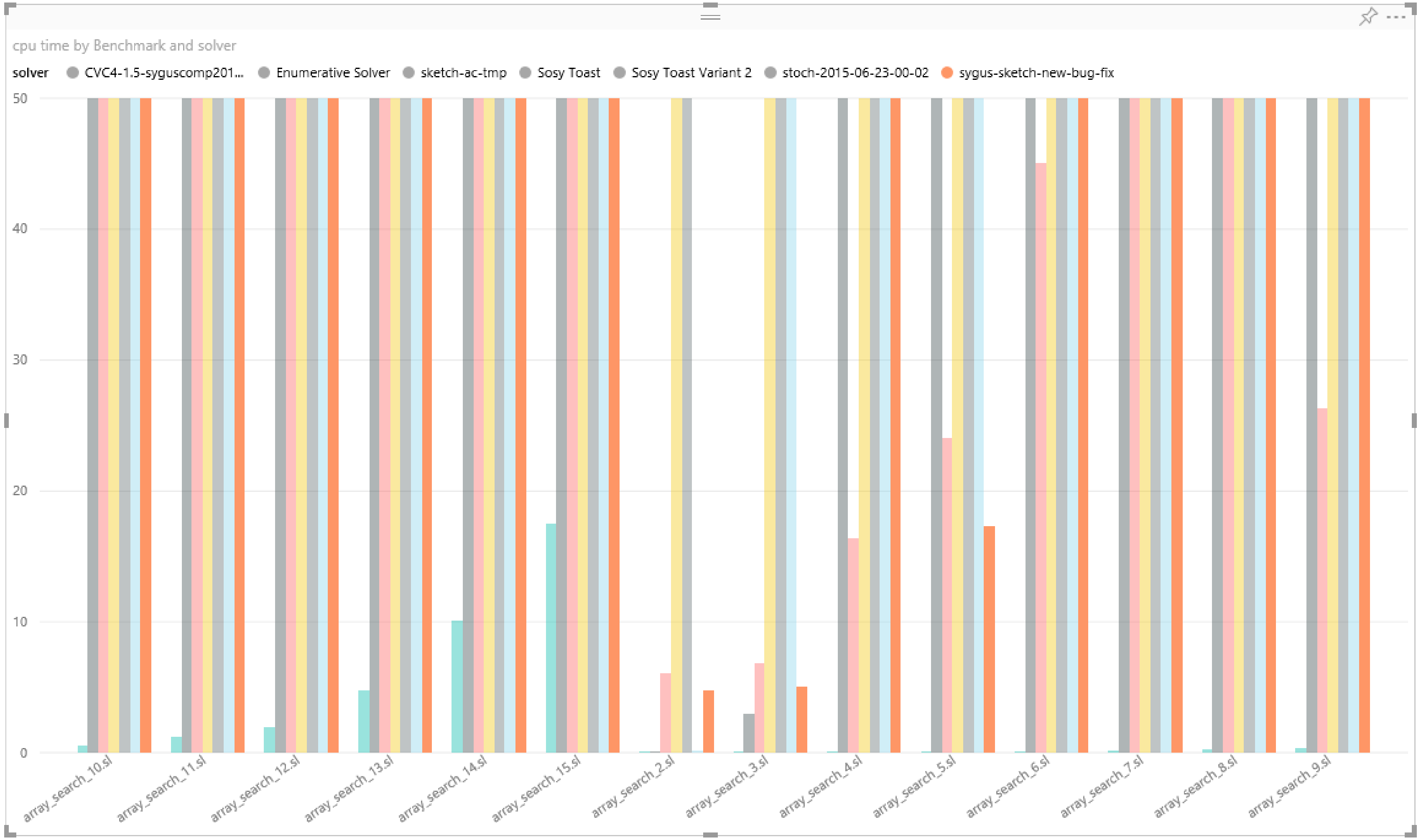
Array-search Benchmarks



Array-search Benchmarks

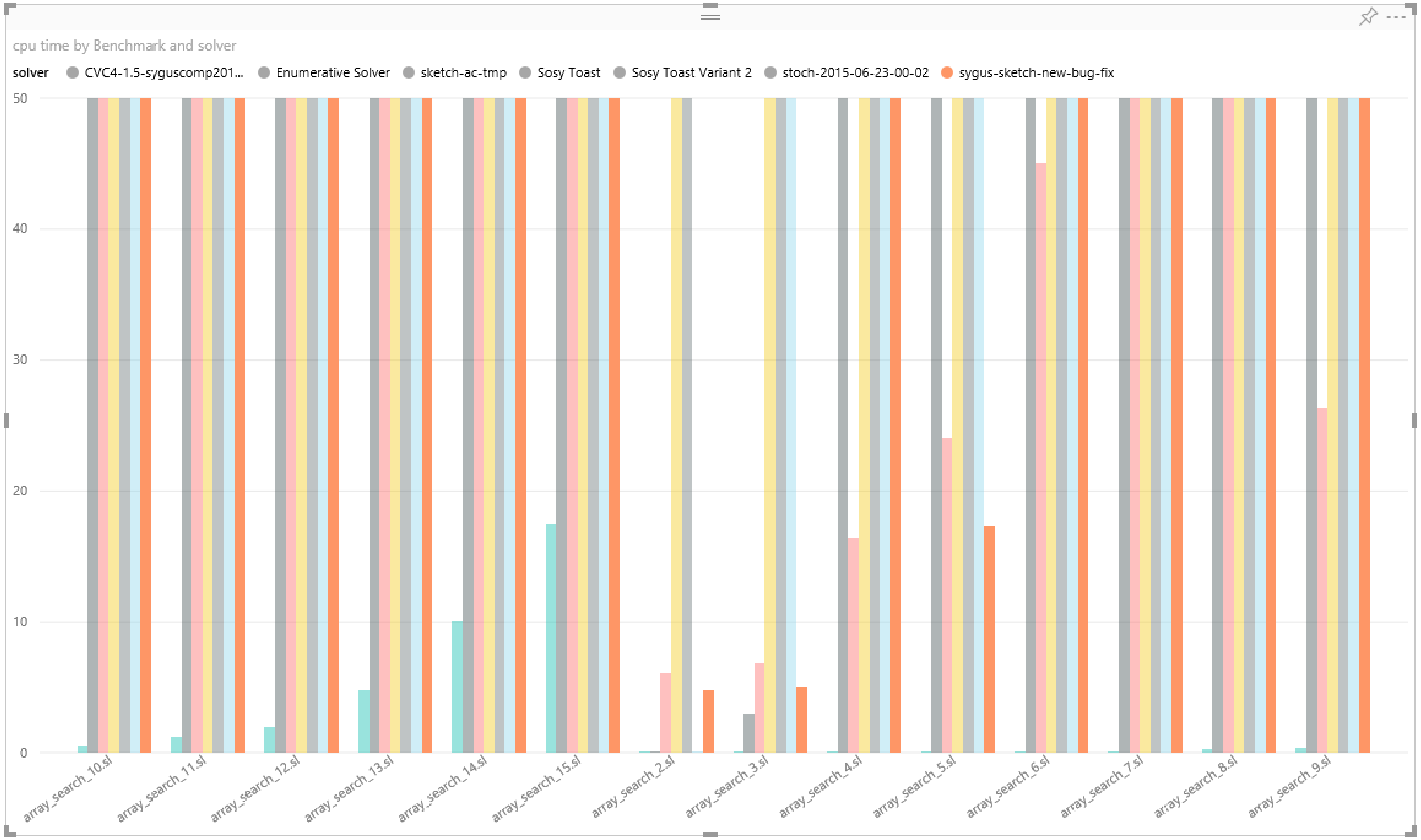


Sketch-based solves upto size 6

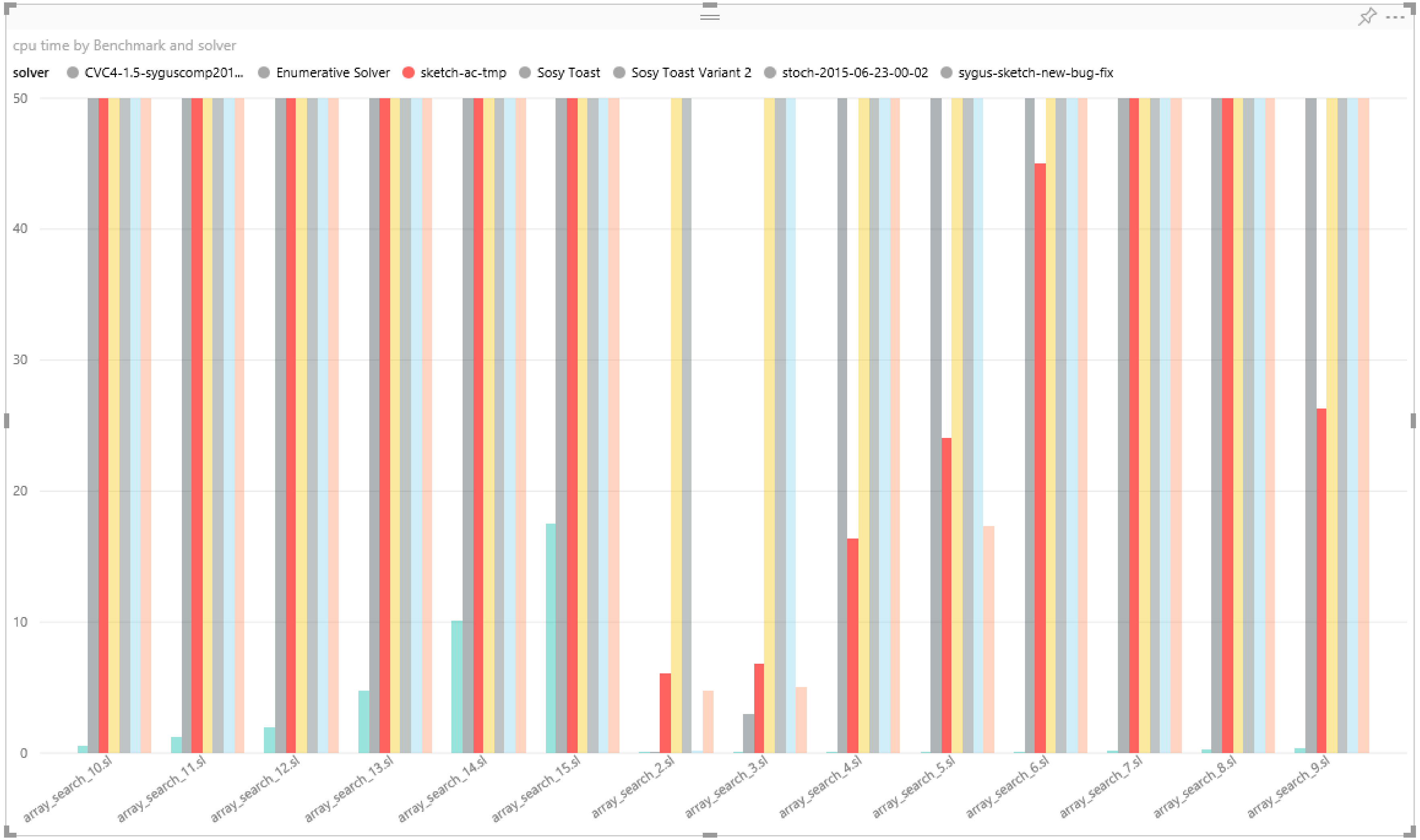


SyGuS

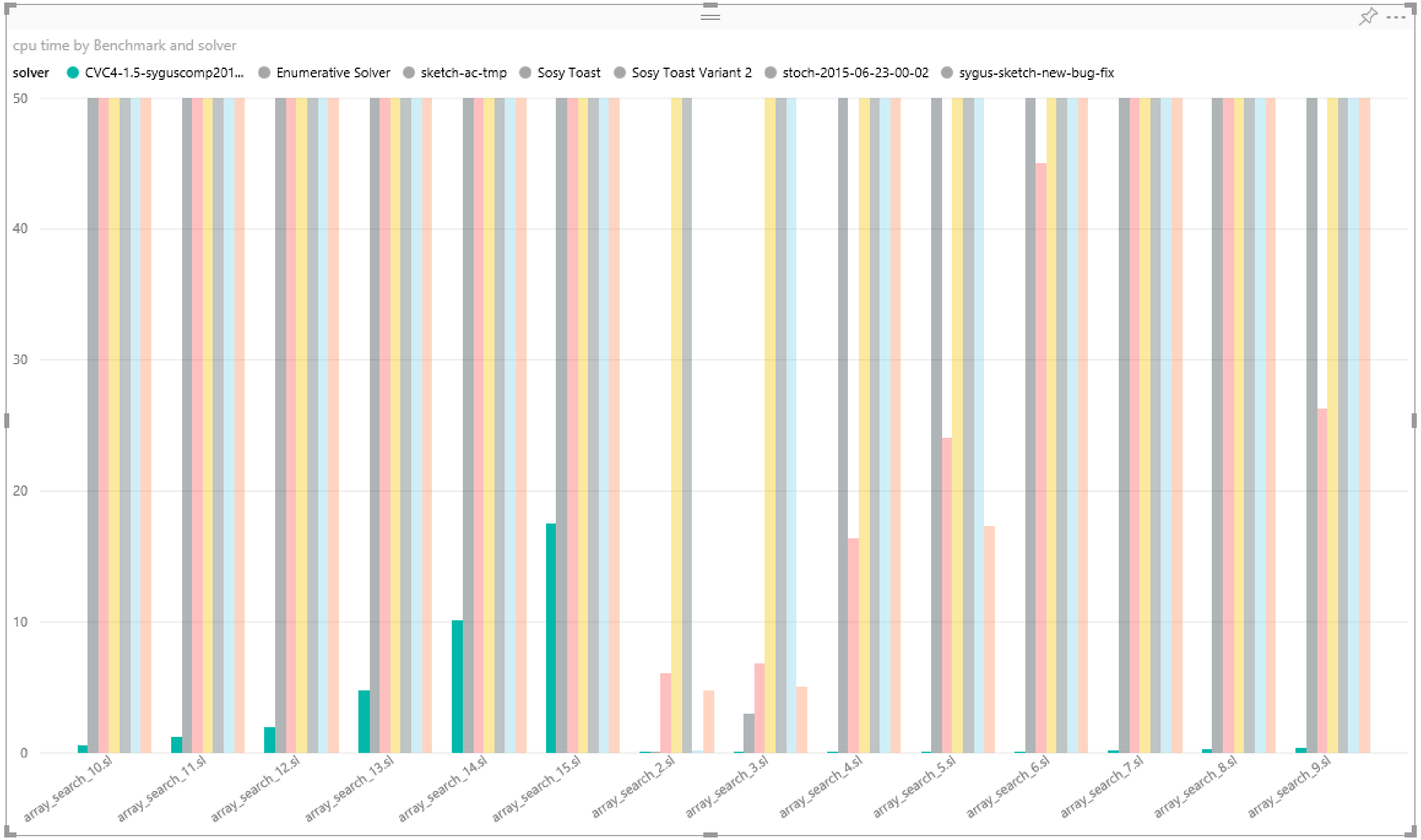
Sketch-based solves upto size 6



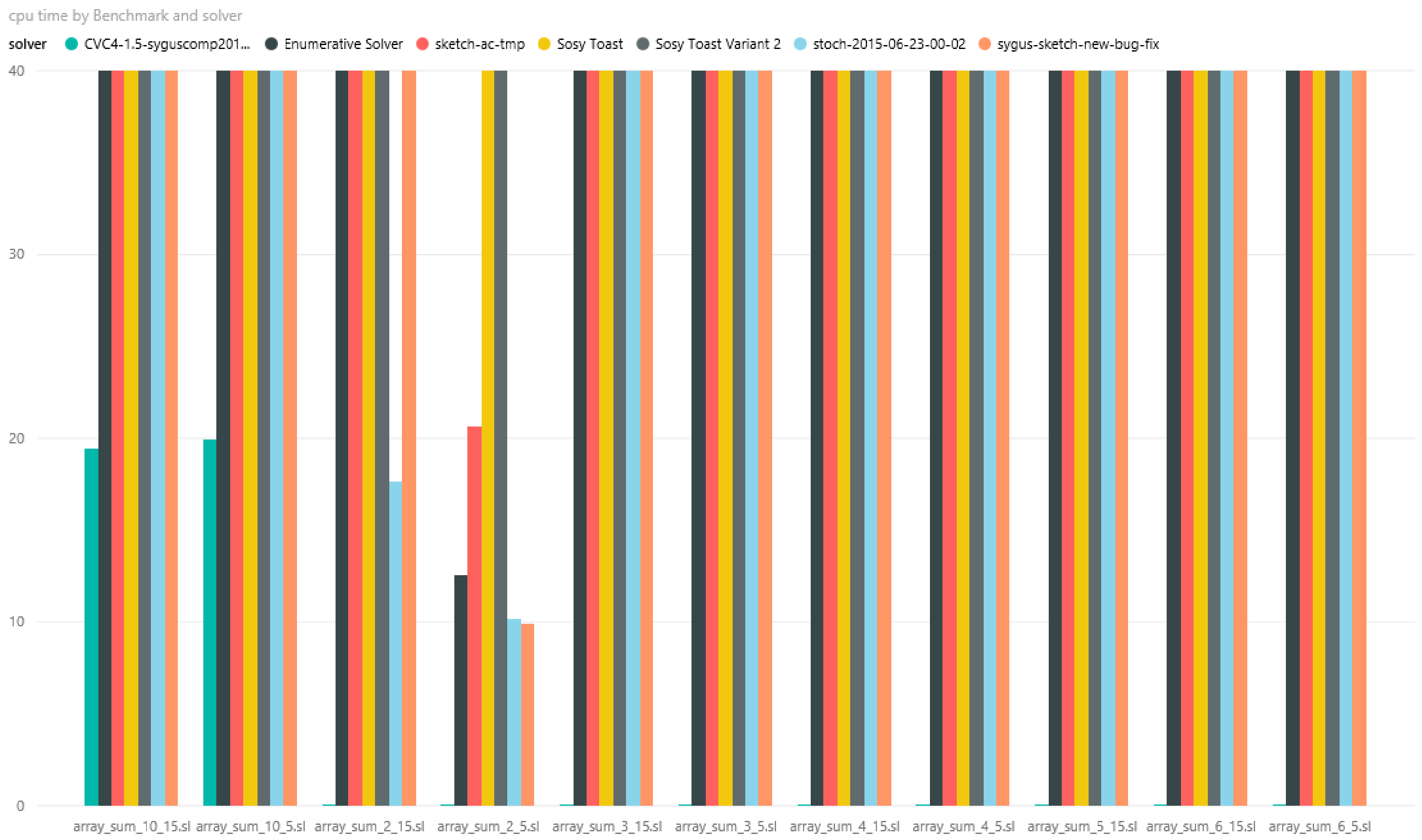
Sketch-AC solves upto size 9



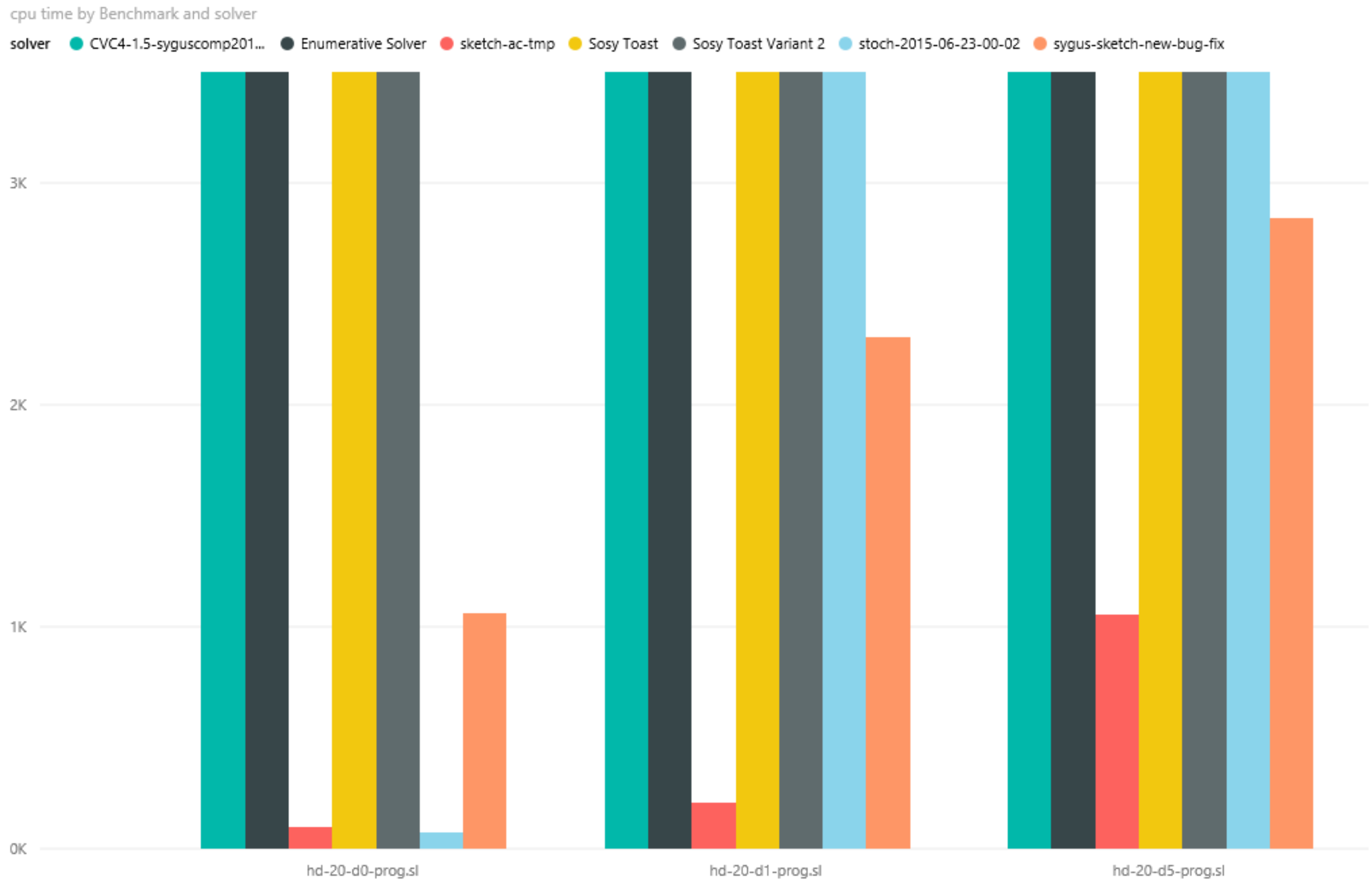
CVC4-1.5 solves all upto size 15!



Similar story for Array-sum



HackerDelight-20



The sad story of ICFP Benchmarks

No solver *could solve any*
but one of the ICFP Benchmarks

Growing Excitement around SyGuS

CVC4 [CAV 2015]
Sketch-AC [CAV 2015]
Alloy* [ICSE 2015]
Unification-based Synthesis [CAV 2015]

Solvers being used for Motion Planning, Quantum Error Correction, Vehicle Control, Compiler Optimizations, Super Compilation, …

FMSD Special Issue on Sygus

Discussion Points

- Add more theories – Arrays, UF, Strings
- Expression Sizes
- Revisit Scoring Mechanism
- More Benchmarks

www.sygus.org

synthlib@cis.upenn.edu

Thanks!

StarExec for providing computational infrastructure

Input format extends SMTLib-2

NSF Expeditions project ExCAPE and its team members

Benchmarks and Solver Participants

www.sygus.org

synthlib@cis.upenn.edu

Glory Awaits You for SyGuS-COMP 2016!