Program Synthesis with Constraint Solving for the OutSystems Language

Rodrigo Bernardo

Instituto Superior Técnico

OutSystems

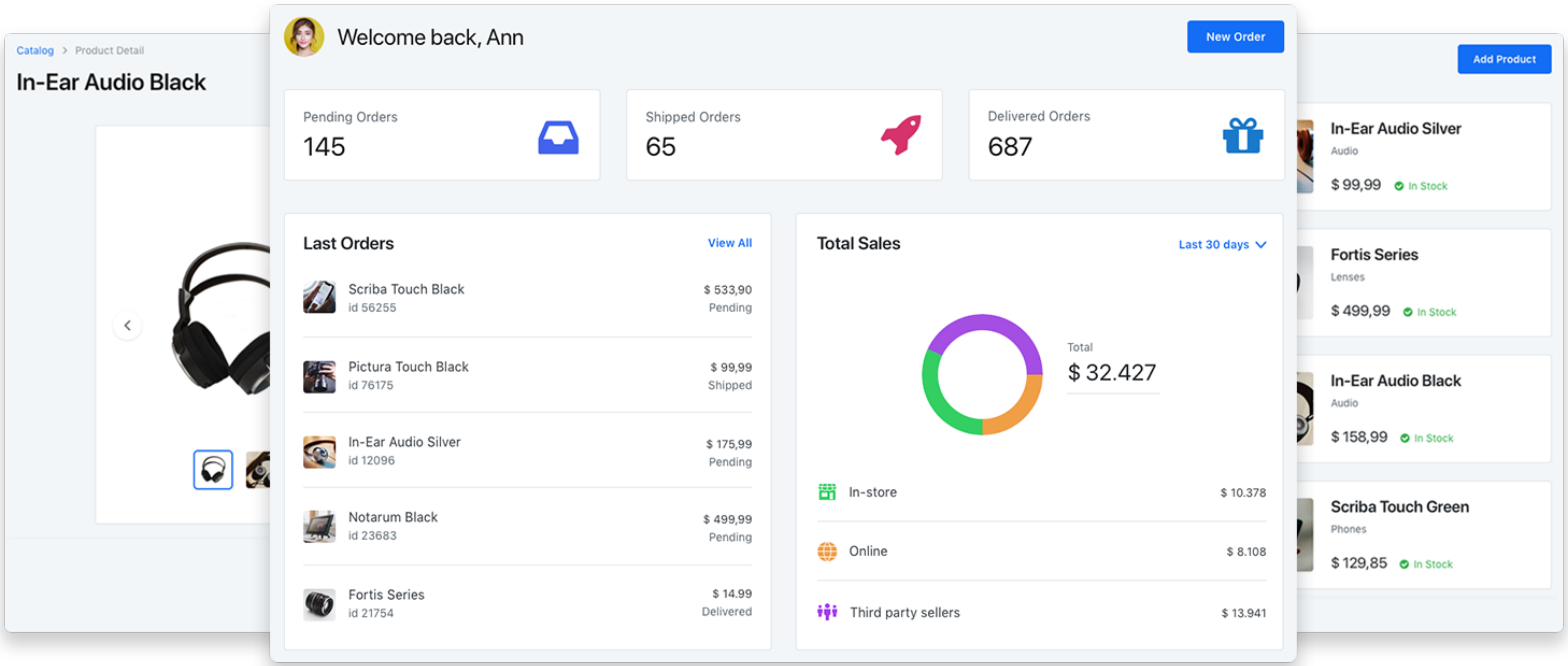
INESC-ID

Carnegie Mellon University

June 7, 2019

Introduction

OutSystems



Catalog > Product Detail

In-Ear Audio Black

Welcome back, Ann

New Order

Pending Orders
145

Shipped Orders
65

Delivered Orders
687

Last Orders — View All

Scriba Touch Black id 56255	$ 533,90 Pending
Pictura Touch Black id 76175	$ 99,99 Shipped
In-Ear Audio Silver id 12096	$ 175,99 Pending
Notarum Black id 23683	$ 499,99 Pending
Fortis Series id 21754	$ 14.99 Delivered

Total Sales — Last 30 days

Total
$ 32.427

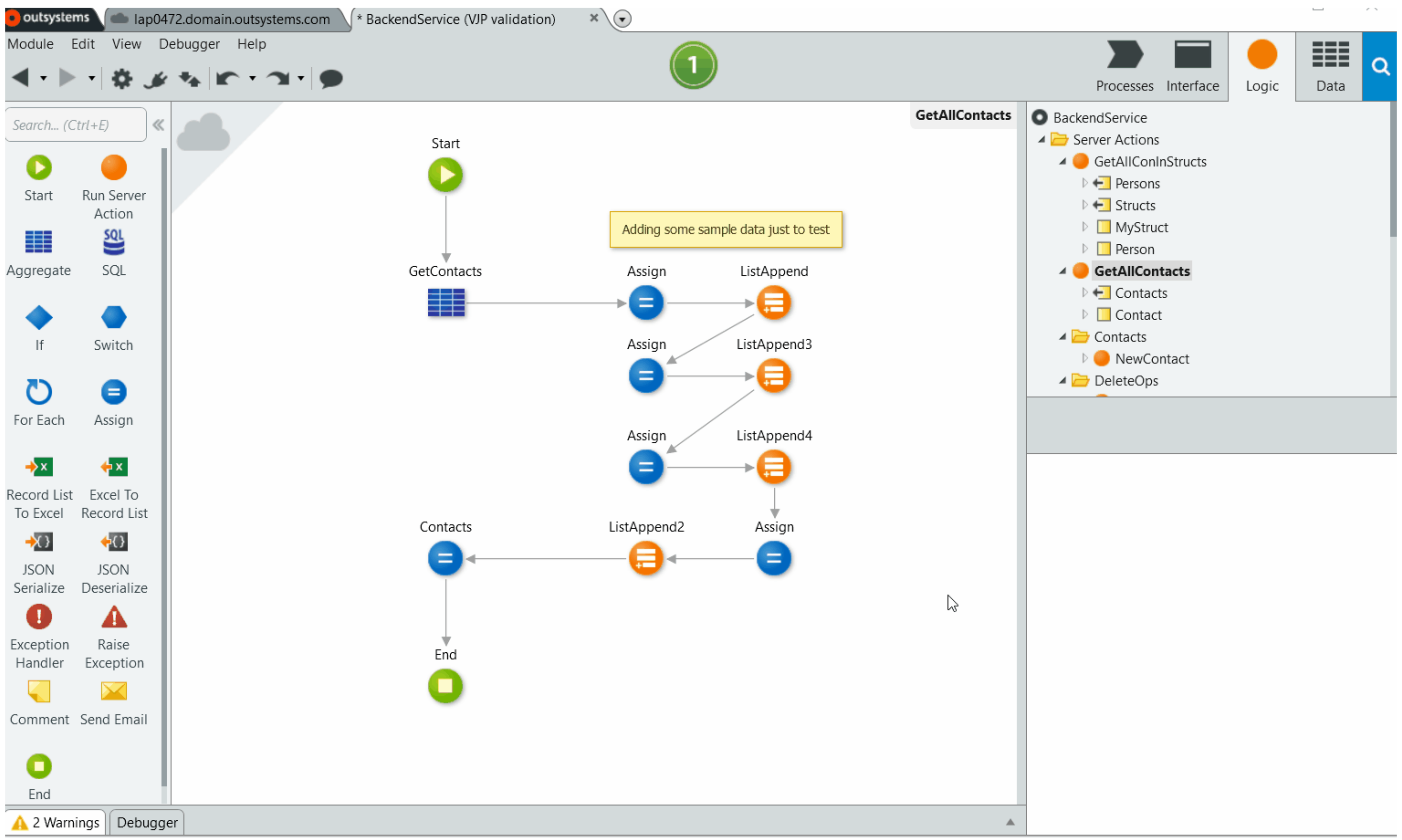
In-store	$ 10.378
Online	$ 8.108
Third party sellers	$ 13.941

Add Product

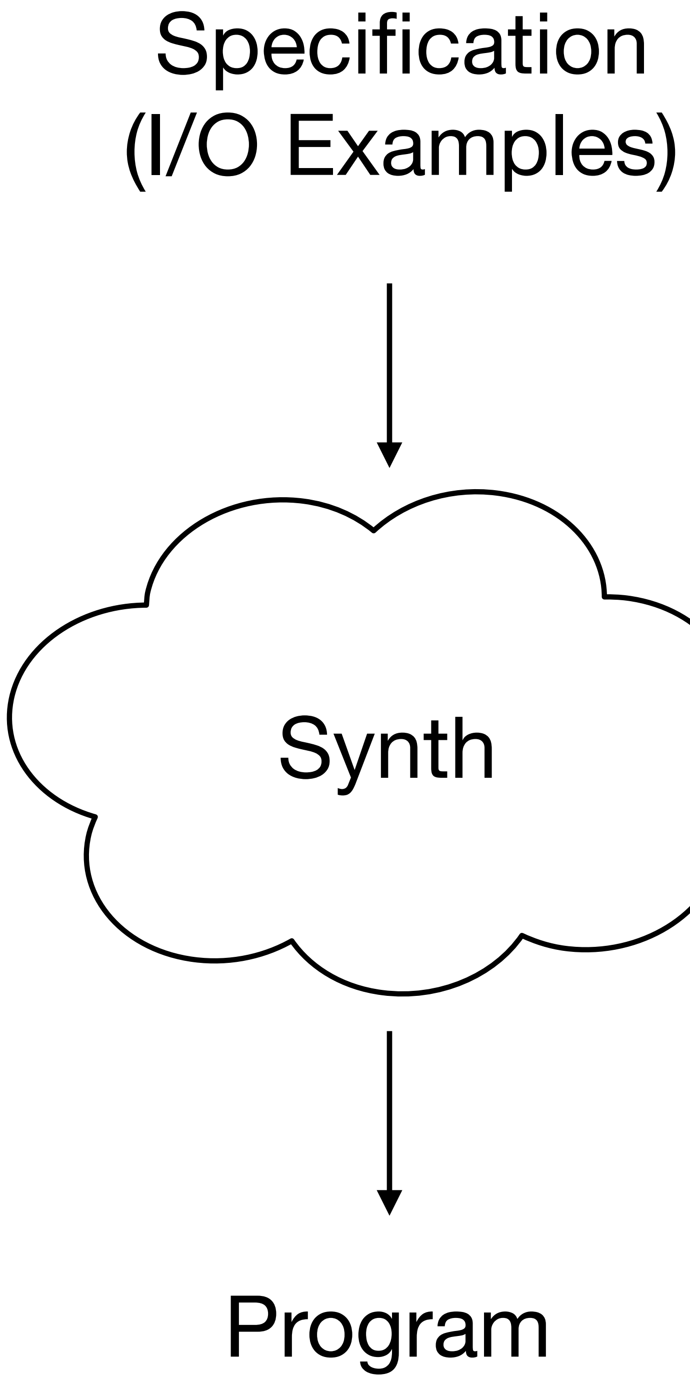
In-Ear Audio Silver	Audio	$ 99,99	In Stock
Fortis Series	Lenses	$ 499,99	In Stock
In-Ear Audio Black	Audio	$ 158,99	In Stock
Scriba Touch Green	Phones	$ 129,85	In Stock

Introduction

OutSystems



Synthesis



Introduction

Program Synthesis

Definition

Given a specification ϕ, find a program *P* that satisfies it for all input *x:*

$$\exists P.\, \forall x.\, \phi(P, x)$$

- Logical Specifications

$$in = [7,2,5,3,2,4] \quad \mapsto \quad out = [2,2,3,4,5,7]$$

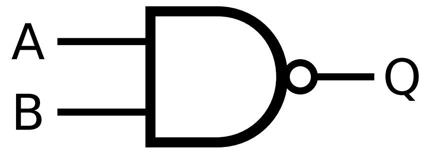
$$isSorted(out) \wedge sameContents(in, out)$$

- I/O Examples

$$\begin{tikzcd}
(1,4) \arrow[dr] & & \\
(2,16) \arrow[r] & \text{Synth} \arrow[r] & 2x^2 \\
(3,36) \arrow[ur] & &
\end{tikzcd}$$

- Components

Concat(Text, Text): Text
Index(Text, Text, Int): Text
Length(Text): Int



⋯

Introduction

OutSystems

Concat(Text, Text): Text	`Concat("", "") = ""`
	`Concat("x", "yz") = "xyz"`

Index(Text, Text, Int): Text	`Index("abcbc", "b", 0) = 1`
	`Index("abcbc", "b", 2) = 3`
	`Index("abcbc", "d", 0) = −1`

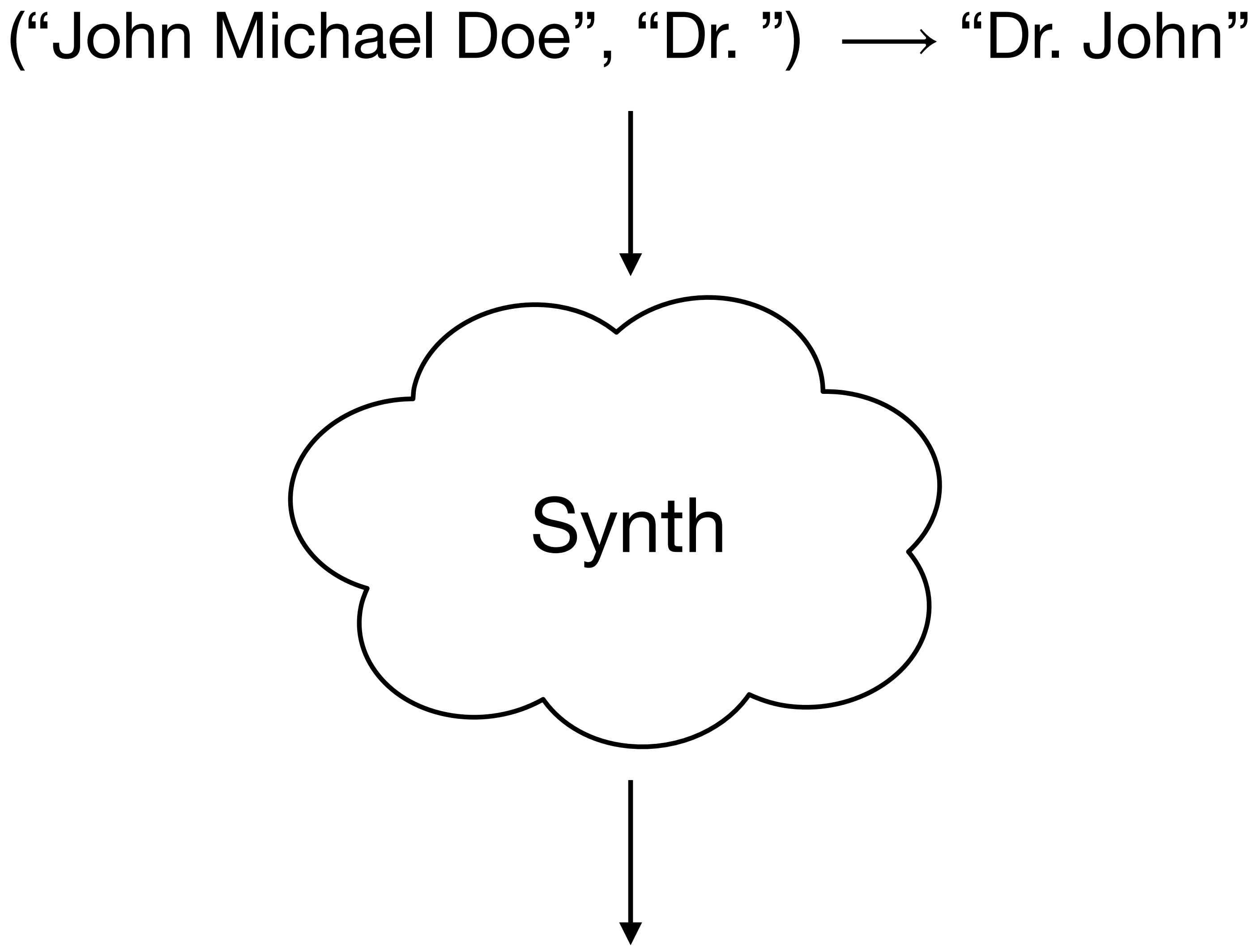
Substr(Text, Int, Int): Text	`Substr("abcdef", 2, 3) = "cde"`
	`Substr("abcdef", 2, 100) = "cdef"`

Length(Text): Text	`Length("") = 0`
	`Length("abc") = 3`

Add(Int, Int): Int	1 + 2 = 3 0 + 1 = 1
Sub(Int, Int): Int	1 - 2 = -1 2 - 1 = 1

Synthesis

Programming by Examples

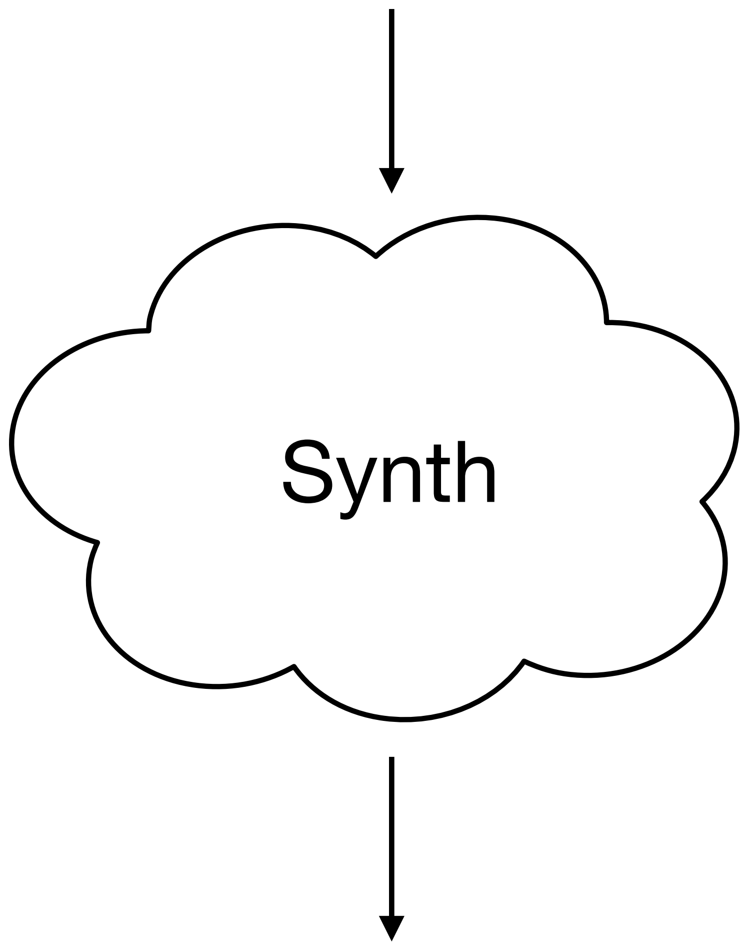


```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Programming by Examples

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”



Index(“John Michael Doe”, “ ”, 0) = 4

Substr(“John Michael Doe”, 0, 4) = “John”

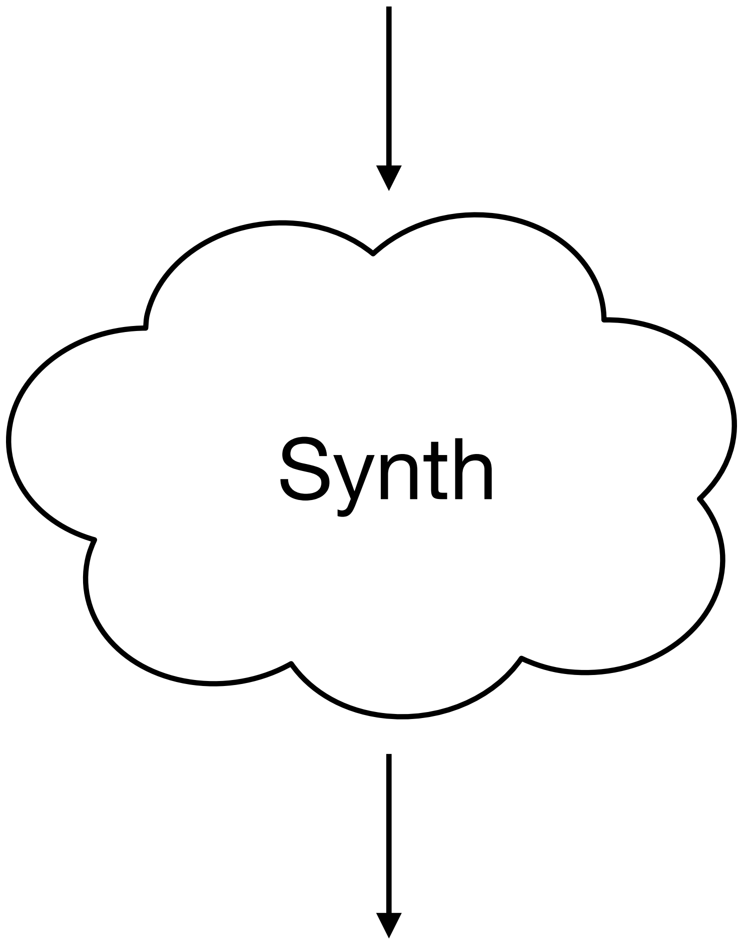
Concat(“Dr. ”, “John”) = “Dr. John”

```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Programming by Examples

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”



r1 = Index(“John Michael Doe”, “ ”, 0)

r2 = Substr(“John Michael Doe”, 0, r1)

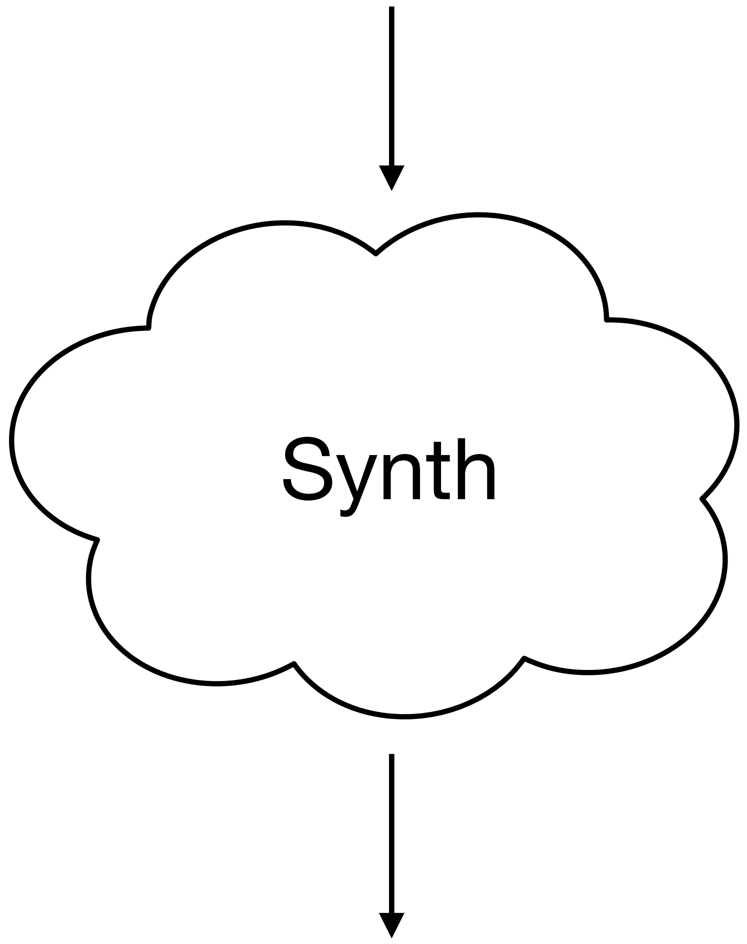
r3 = Concat(“Dr. ”, r2)

```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Programming by Examples

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”



c0 = “ ”

c1 = 0

r1 = Index(“John Michael Doe”, c0, c1)

r2 = Substr(“John Michael Doe”, c1, r1)

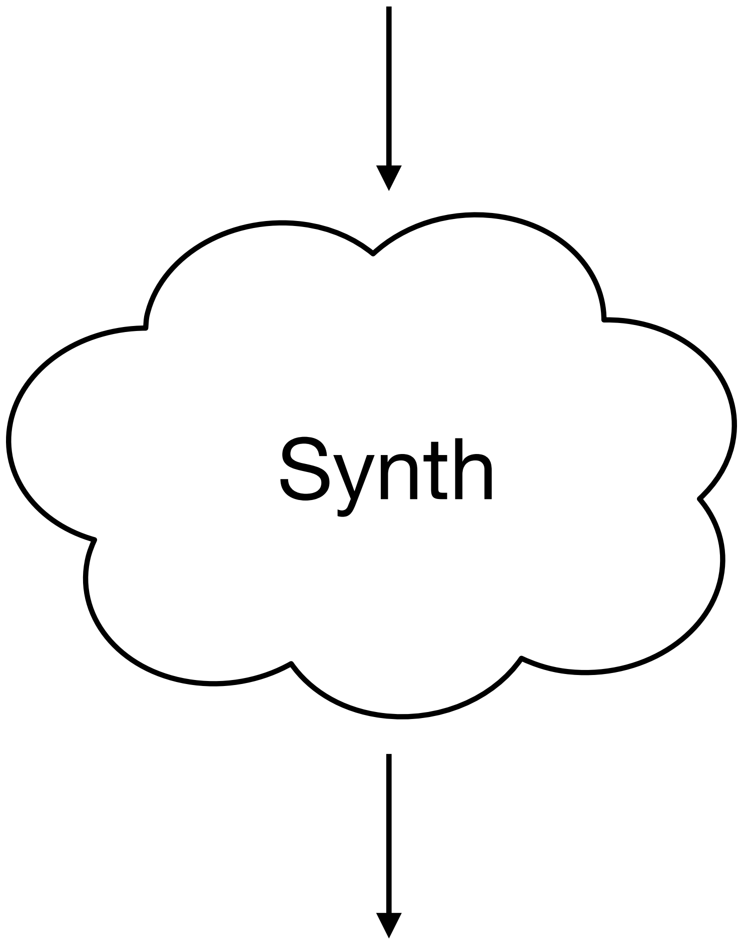
r3 = Concat(“Dr. ”, r2)

```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Programming by Examples

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”



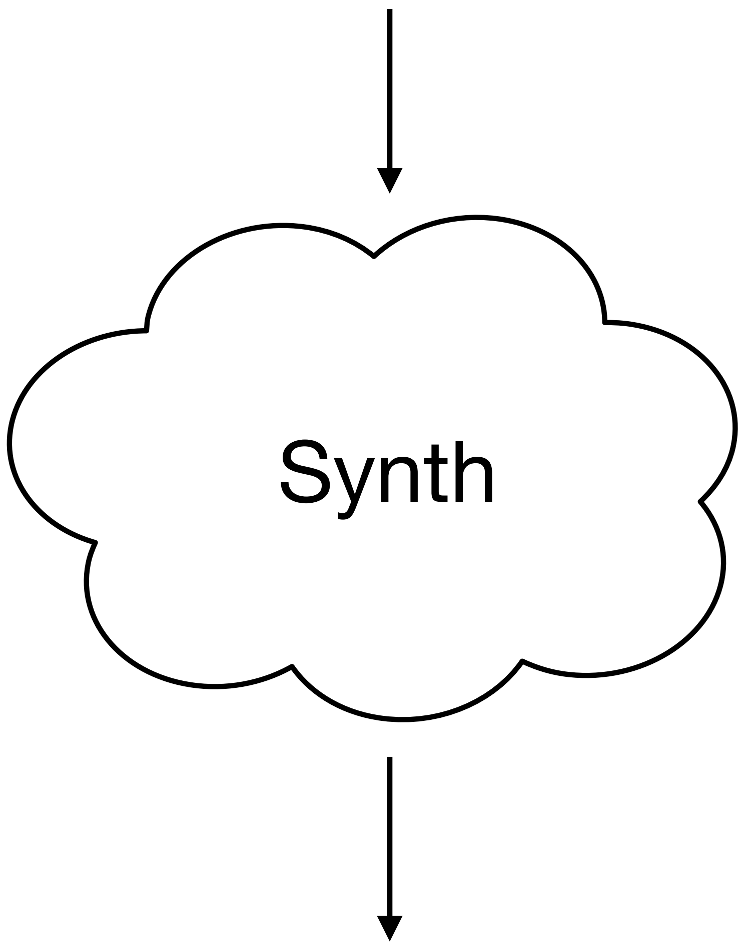
```
prog(name, prefix):
    c0 = “ ”
    c1 = 0
    r1 =  Index(name, c0, c1)
    r2 =  Substr(name, c1, r1)
    r3 =  Concat(prefix, r2)
```

```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Programming by Examples

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

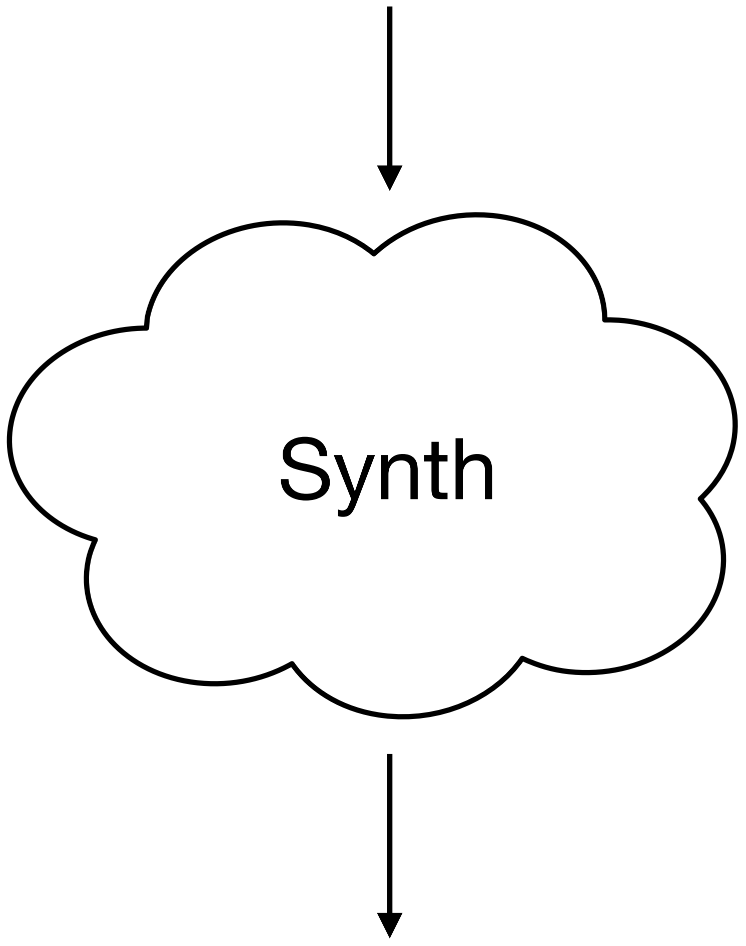


```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Programming by Examples

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

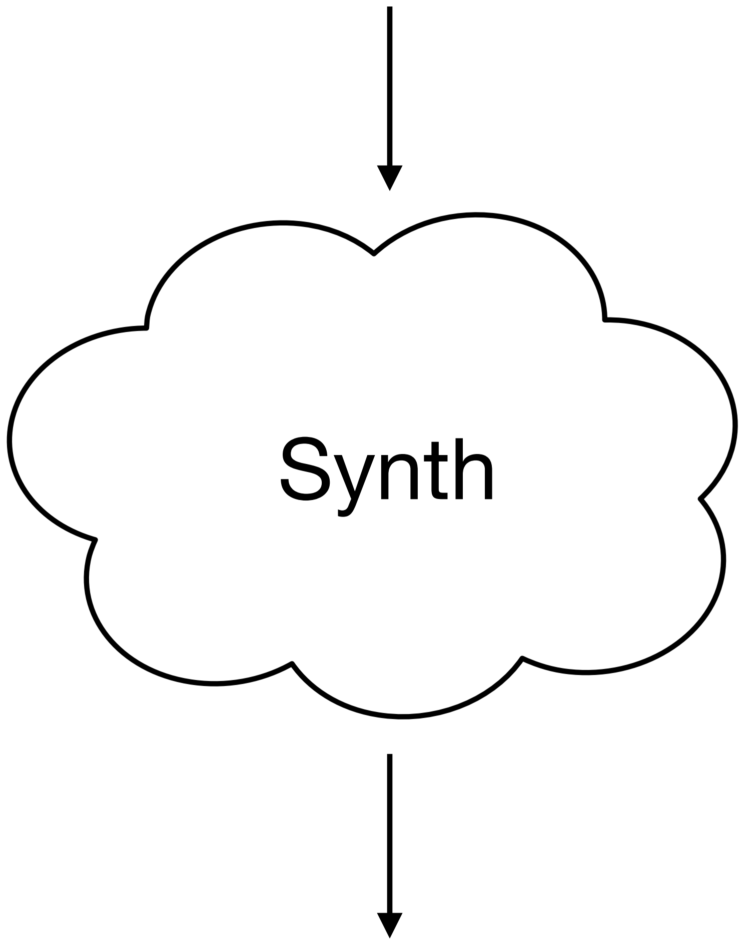


```
prog(name, prefix) =
  Concat(“Dr. ”,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

(“John Michael Doe”, “Mr. ”) $\longrightarrow$ “Mr. John”

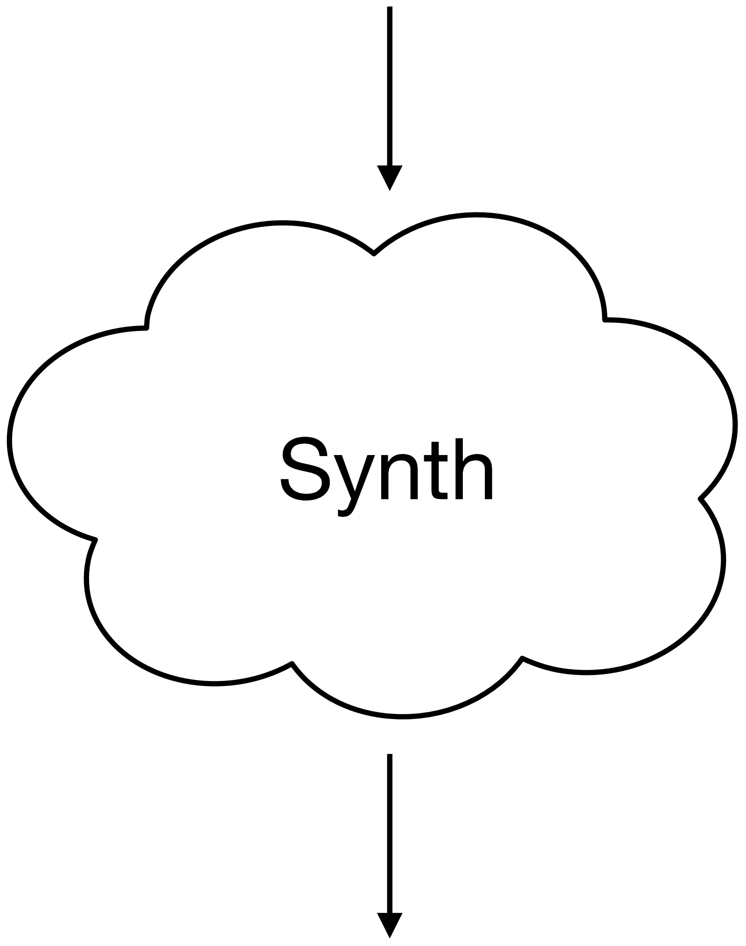


```
prog(name, prefix) =
  Concat(“Dr. ”,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

(“John Michael Doe”, “Mr. ”) $\longrightarrow$ “Mr. John”



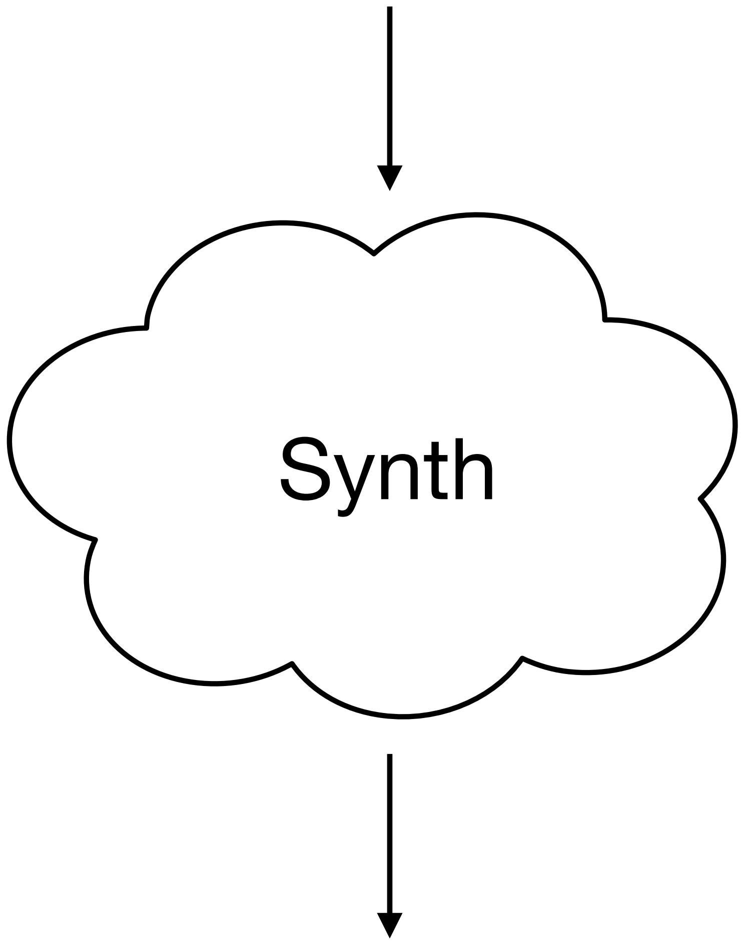
```
prog(name, prefix) =
  Concat(prefix, “John”)
```

Synthesis

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

(“John Michael Doe”, “Mr. ”) $\longrightarrow$ “Mr. John”

(“Catherine Marie Joe”, “Dr. ”) $\longrightarrow$ “Dr. Catherine”



```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

Synthesis

Constraint Solving

Definition

Given a signature $\sum$ and a $\sum$-theory T, the Satisfiability Modulo Theories (SMT) problem is the problem of determining the T-satisfiability of $\sum$-formulas.

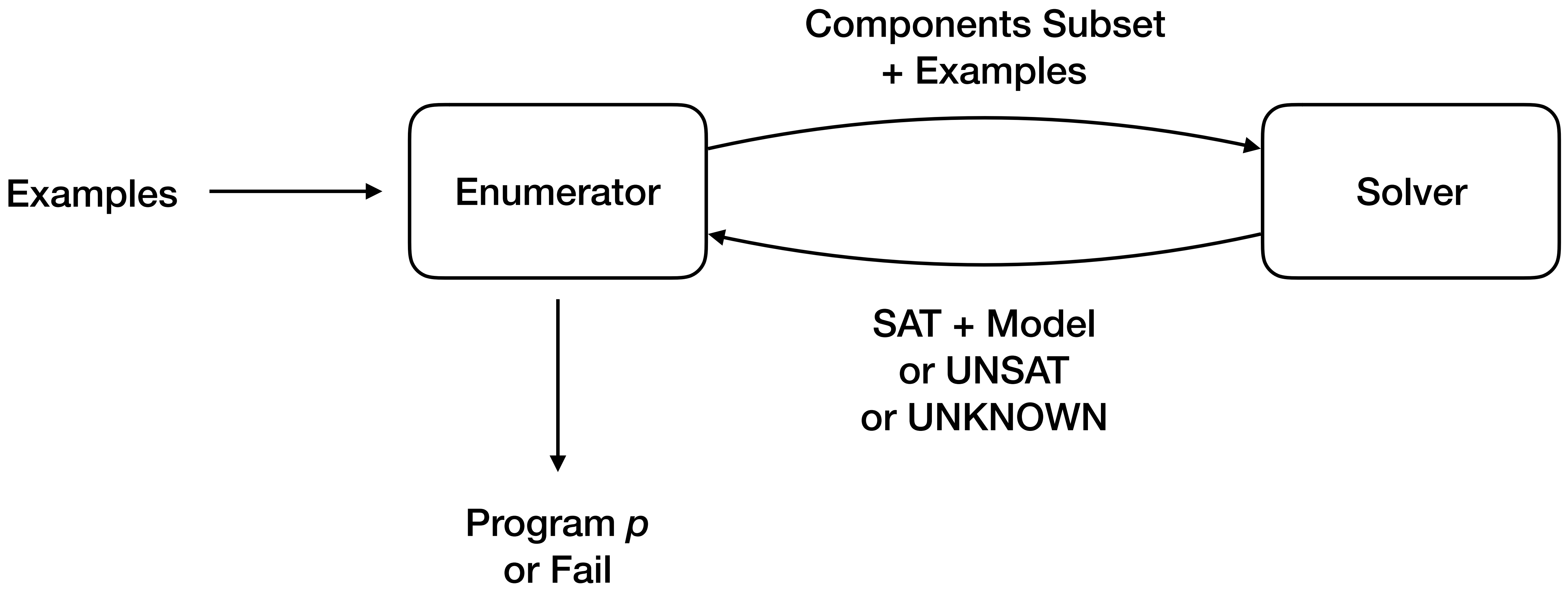
$$\begin{aligned}\Sigma = \{&\text{""}, \text{"}a\text{"}, \text{"}aa\text{"}, \ldots, \text{"}ab\text{"}, \ldots, {+}{+}, substr, index, \ldots\} \\ \cup\, \{&\ldots, -1, 0, 1, \ldots +, -\}\end{aligned}$$

$$\text{“Dr. John”} = x \mathbin{{+}{+}} Substr(\text{“John Michael Doe”}, y, z)$$

$$x = \text{“Dr. ”}, \quad y = 0, \quad z = 4$$

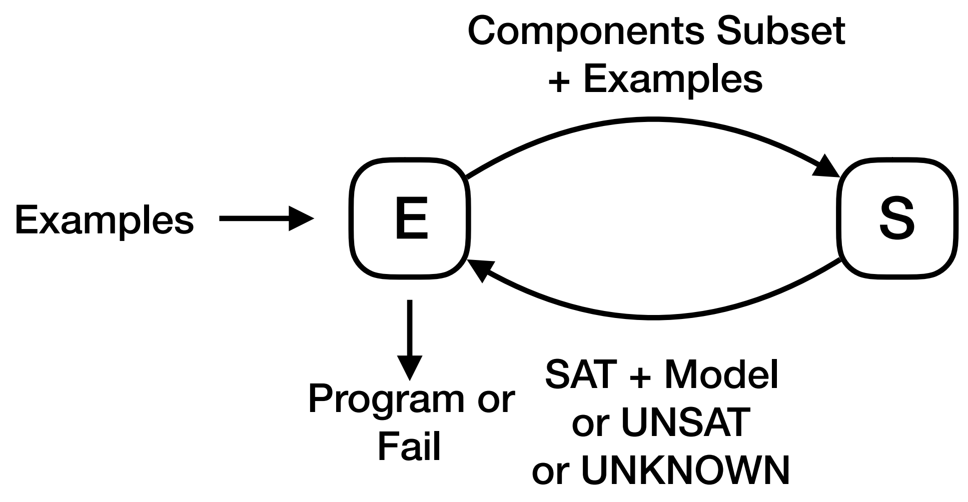
Synthesis

Setwise



Synthesis

Setwise



(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

```
prog(name, prefix) =
  Concat(prefix,
    Substr(name, 0,
      Index(name, “ ”, 0)))
```

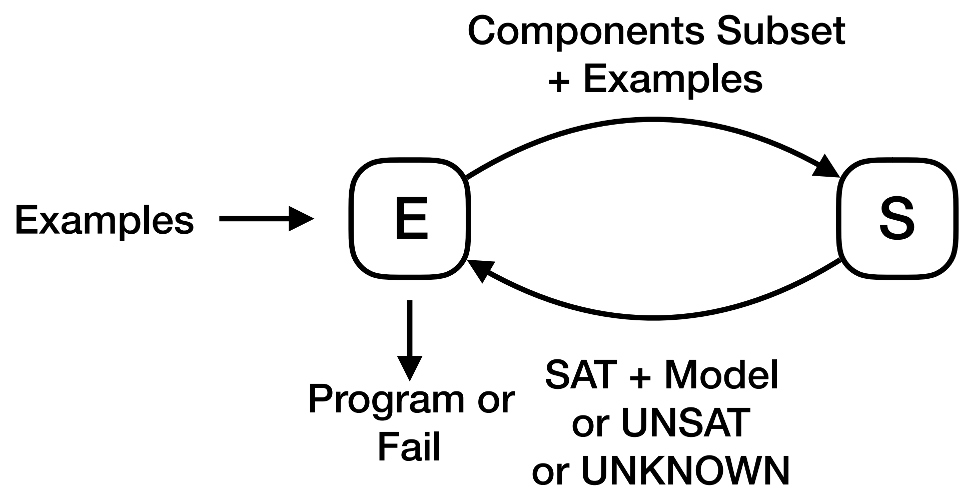
- Concat
- Index
- Length
- Substr
- Add
- Sub

{Concat}, {Index}, {Length}, {Substr}, …,
{Concat, Concat}, {Concat, Index}, {Concat, Length}, …,
{Concat, Concat, Concat}, {Concat, Concat, Index},
{Concat, Concat, Length}, …,
{Concat, Index, Index}, …, {Concat, Index, Substr}

Synthesis

Setwise

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

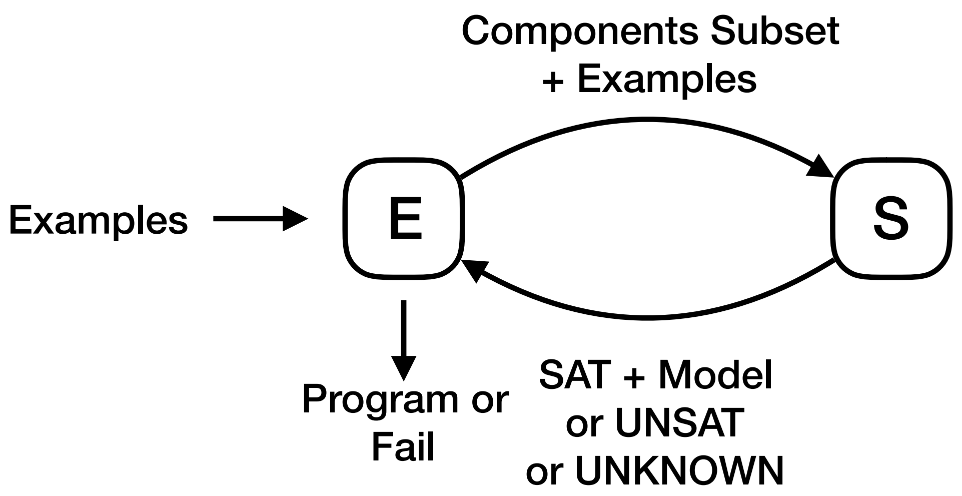


```
prog(name, prefix):
    c0 = “ ”
    c1 = 0
    r1 = Index(name, c0, c1)
    r2 = Substr(name, c1, r1)
    r3 = Concat(prefix, r2)
```

Synthesis

Setwise

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”



```
prog(name, prefix):
    c0 = “ ”
    c1 = 0
    r1 = Index(name, c0, c1)
    r2 = Substr(name, c1, r1)
    r3 = Concat(prefix, r2)
```

c_0, c_1 r_1, r_2, r_3 p_{11}, p_{12}, p_{13}

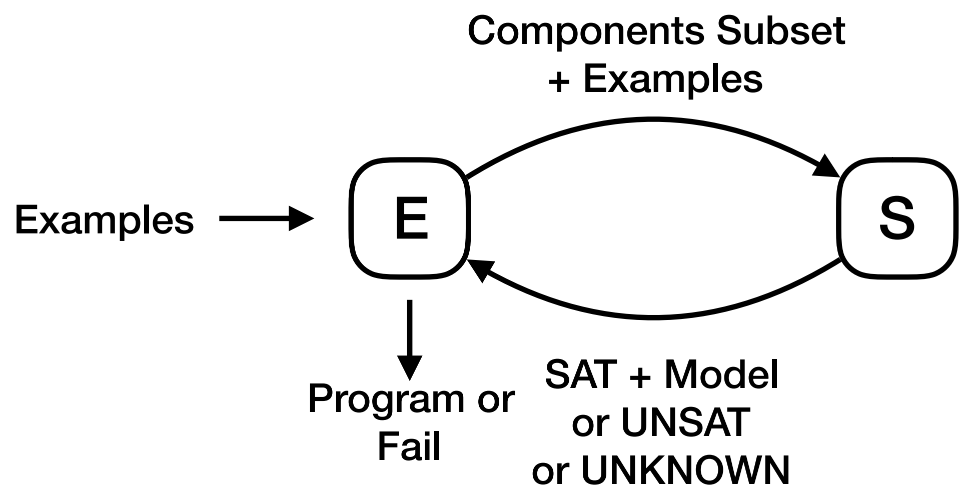
p_{21}, p_{22}, p_{23}

p_{31}, p_{32}, p_{33}

i_0, i_1 o_0

Synthesis

Setwise



(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

```
prog(name, prefix):
    c0 = “ ”
    c1 = 0
    r1 = Index(name, c0, c1)
    r2 = Substr(name, c1, r1)
    r3 = Concat(prefix, r2)
```

c_0, c_1 $\quad$ r_1, r_2, r_3 $\quad$ p_{11}, p_{12}, p_{13}

p_{21}, p_{22}, p_{23}

p_{31}, p_{32}, p_{33}

i_0, i_1 $\quad$ o_0

l_{c_0}, l_{c_1} $\quad$ $l_{r_1}, l_{r_2}, l_{r_3}$ $\quad$ $l_{p_{11}}, l_{p_{12}}, l_{p_{13}}$

$l_{p_{21}}, l_{p_{22}}, l_{p_{23}}$

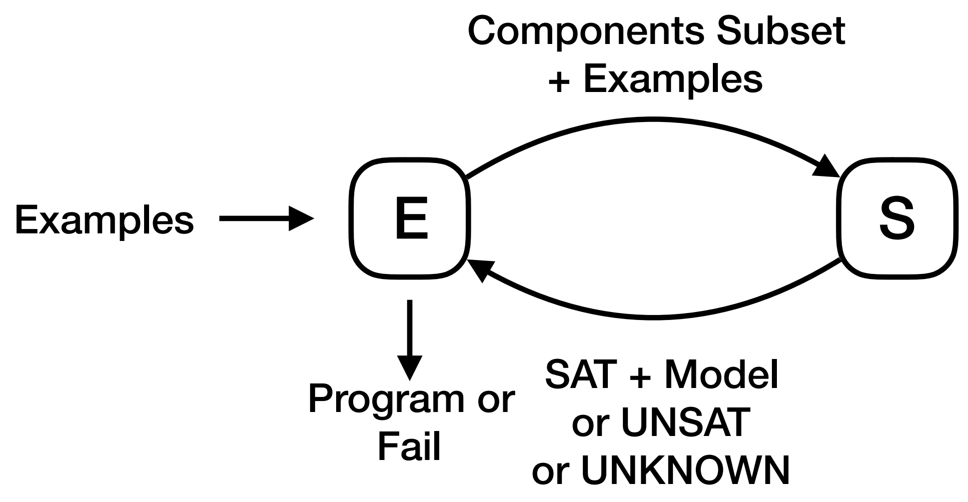
$l_{p_{31}}, l_{p_{32}}, l_{p_{33}}$

l_{i_0}, l_{i_1} $\quad$ l_o

Synthesis

Setwise

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”



i_0 = “John Michael Doe” $\wedge$ i_1 = “Dr. ”

o_0 = “Dr. John”

prog(name, prefix):

c0 = “ ”

c1 = 0

r1 = Index(name, c0, c1)

r2 = Substr(name, c1, r1)

r3 = Concat(prefix, r2)

$l_{i_0} = 1 \wedge l_{i_1} = 2 \wedge l_{c_0} = 3 \wedge l_{c_1} = 4$

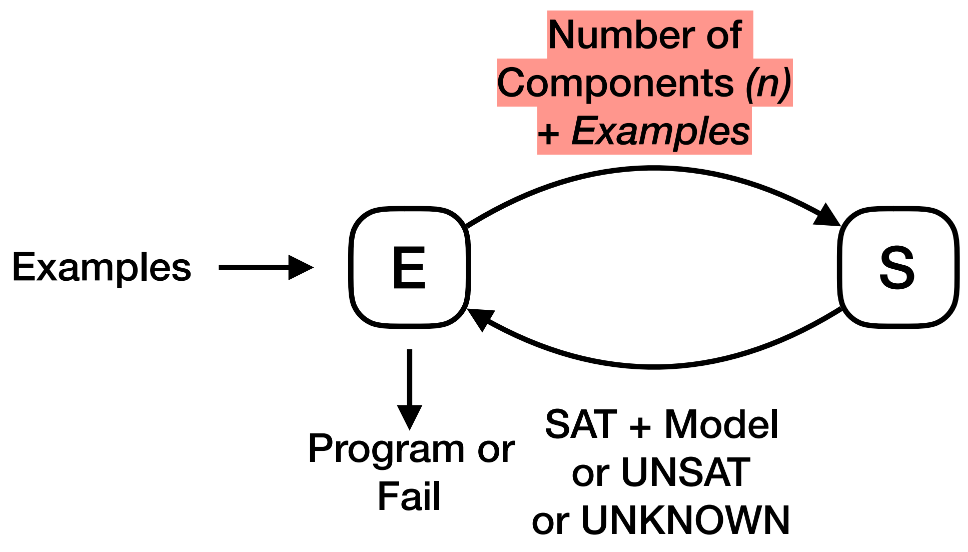
$$1 \le l_{p_{11}} \le l_{r_1} \wedge 1 \le l_{p_{12}} \le l_{r_1} \wedge 1 \le l_{p_{13}} \le l_{r_1}$$
$$1 \le l_{p_{21}} \le l_{r_2} \wedge 1 \le l_{p_{22}} \le l_{r_2} \wedge 1 \le l_{p_{23}} \le l_{r_2}$$
$$1 \le l_{p_{31}} \le l_{r_3} \wedge 1 \le l_{p_{32}} \le l_{r_3} \wedge 1 \le l_{p_{33}} \le l_{r_3}$$

$l_{r_3} = l_o \Rightarrow r_3 = o$

…

Synthesis

Whole

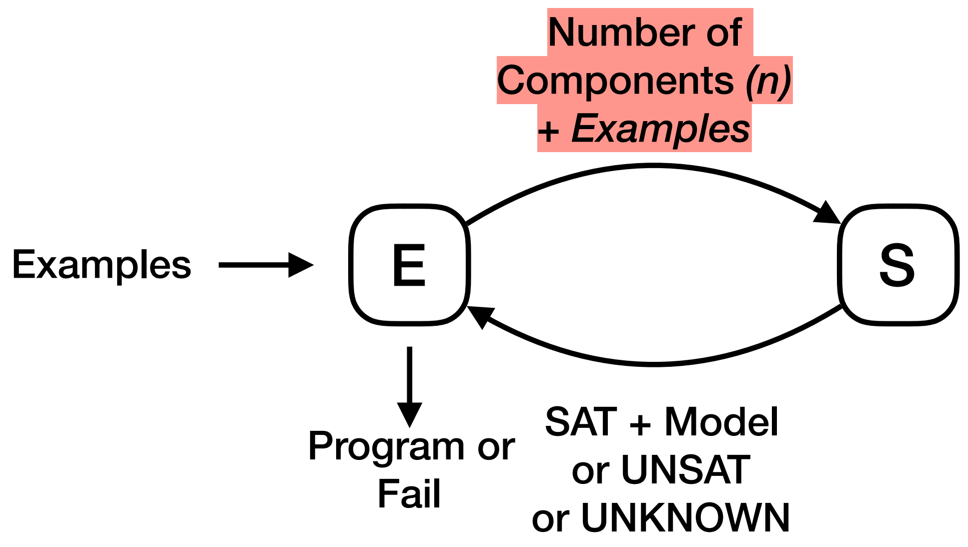

Number of Components *(n)* + *Examples*
Examples
E
S
Program or Fail
SAT + Model or UNSAT or UNKNOWN

(“John Michael Doe”, “Dr. ”) ⟶ “Dr. John”

```
prog(name, prefix):
    c0 = “ ”
    c1 = 0
    r1 = Index(name, c0, c1)
    r2 = Substr(name, c1, r1)
    r3 = Concat(prefix, r2)
```

Synthesis

Whole



Number of Components *(n)* + *Examples*
Examples
E
S
Program or Fail
SAT + Model or UNSAT or UNKNOWN

(“John Michael Doe”, “Dr. ”) $\longrightarrow$ “Dr. John”

$a_1, a_2, \ldots, a_n$

```
prog(name, prefix):
    c0 = “ ”
    c1 = 0
    r1 = Index(name, c0, c1)
    r2 = Substr(name, c1, r1)
    r3 = Concat(prefix, r2)
```

```
prog($i_0$, $i_1$):
    $c_0$ = ?
    $c_1$ = ?
    $r_1$ = $f_{a_1}(p_{11}, p_{12}, p_{13})$
    $r_2$ = $f_{a_2}(p_{21}, p_{22}, p_{23})$
    $r_3$ = $f_{a_3}(p_{31}, p_{32}, p_{33})$
```

Experimental Results

Setup

- 51 expressions out of more than 285,000
- 3 I/O examples per expression
- 600 seconds timeout
- 16 GB memory
- Z3 SMT solver, version 4.8.5

Experimental Results

SyPet

- Component-based synthesiser
- Does not compute constants automatically
- Baseline with two configurations:
    - User-provided constants (Sypet-User)
    - Every constant is a component (Sypet-All)

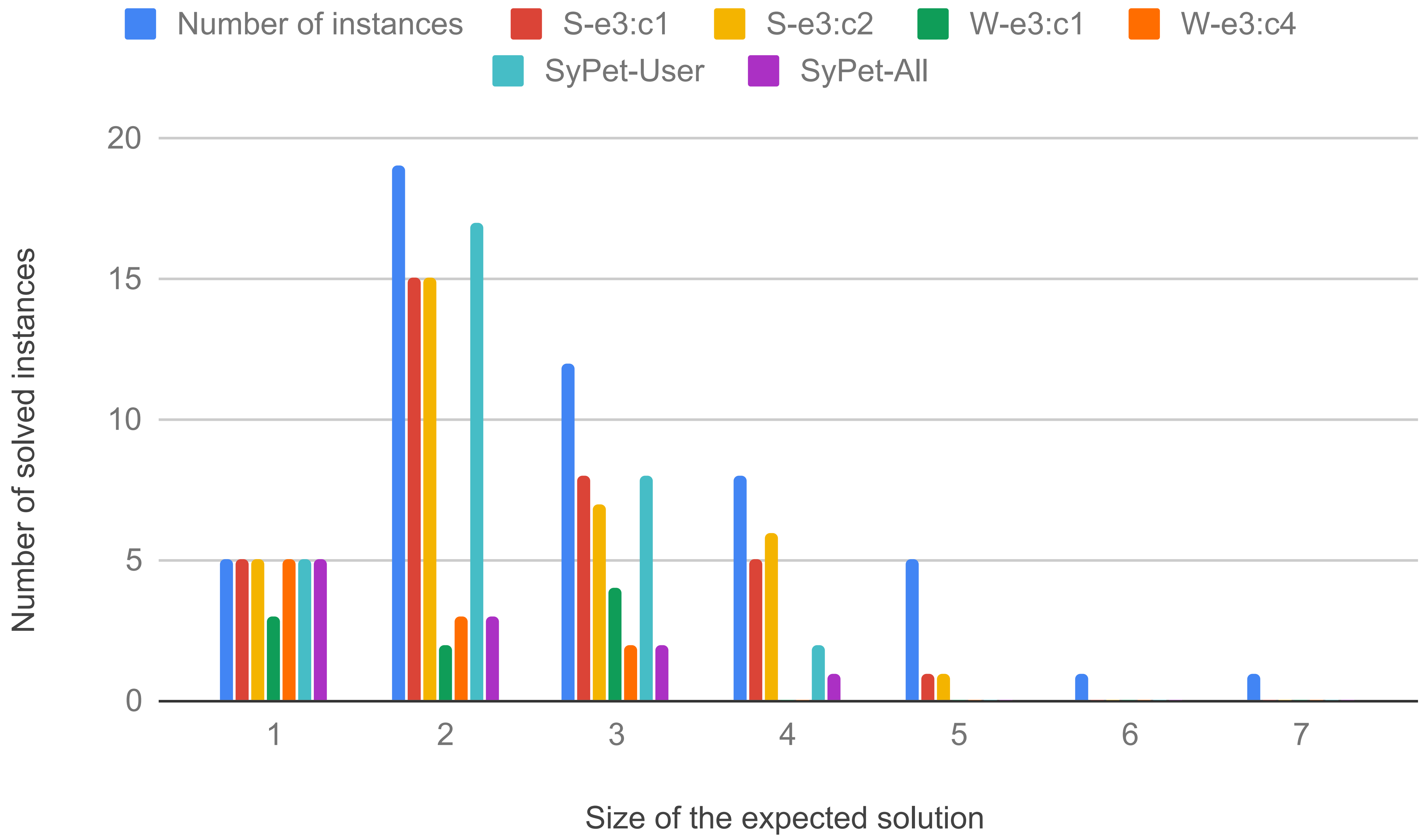
Experimental Results

Configurations

- Interested in:
    - Number of examples
    - Number of constants
- Configuration name format is *X*-e*Y*:c*Z*
    - For example, S-e1:c1, or W-e3:c4
- SyPet-User and SyPet-All, only with 3 I/O examples

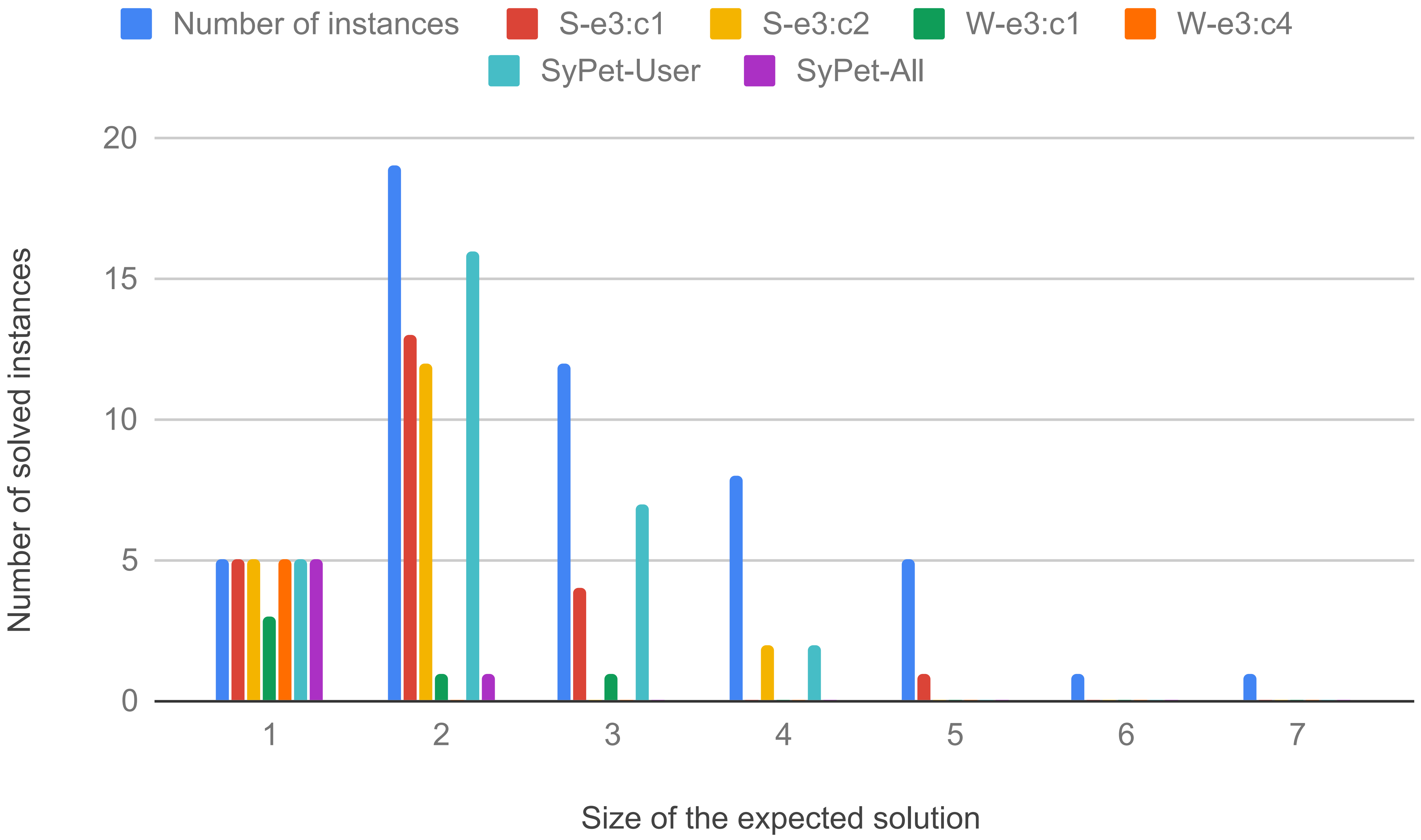
Experimental Results

Outputs Matching Input-Output Examples



Experimental Results

Outputs Matching Expected Solutions



Conclusion

- Developed two synthesizers for OutSystems expressions
- Working for small instances and small library of components
- Future work:
  - Add feature for user-provided constants
  - Apply machine learning to guide the enumerative search
  - User-study to ascertain the practicality of such synthesizers

Questions?