Discovering Relational Specifications

by Calvin Smith, Gabriel Ferns, Aws Albarghouthi

Muqsit Azeem
TRDDC, Pune

July 21, 2018

Formal Methods Update Meeting, BITS Pilani, Goa Campus

What are we interested in?

Formal specifications of library functions

What are we interested in?

Formal specifications of library functions

- Problems:
  - code unavailable
  - large code
- partial behavior of these functions
- discover a rich class of specifications

Problem

- Given a function f and a data set D, a partial picture of i/o behavior of f, perhaps collected through some random testing

What can we learn about the function f by simply analyzing the dataset D?

Example 1

f		
i_1	i_2	**r**
1	2	3
3	4	7
5	6	11
4	3	7
⋮	⋮	⋮

f is commutative

Specification

Example 2

f

i_1	**r**
1	1
7	7
-10	10

$$f(x) = |x|$$

Specification

Example 2

f

i_1	**r**
1	1
7	7
-10	10

$$f(x) = x$$

Specification

D-restricted assignment (σ_D)

f

i_1	**r**
1	1
7	7
-10	10

$$f(x) = x$$

Specification

assign each variable of specification to a constant that appears in the dataset

- $\sigma_D = \{x \to 1\}$ is a D-restricted assignment to $f(x) = x$
- but $\sigma_D = \{x \to 2\}$ is not a D-restricted assignment because f is not defined for 2 in the given dataset

Example 2

f

i_1	**r**
1	1
7	7
-10	10

$$f(x) = x$$

Specification

positive evidence

- D-restricted assignments that satisfies the specification
- positive evidence is $\{x \to 1, x \to 7\}$

Example 2

f

i_1	**r**
1	1
7	7
-10	10

$$f(x) = x$$

Specification

negative evidence

- D-restricted assignments that does not satisfy the specification
- negative evidence is $\{x \rightarrow -10\}$

What does it mean for a specification to explain a data-set?

- if there exists a *negative evidence* - the specification is considered inconsistent with the data
- otherwise the specification is considered *more likely* to be true depending on a measure of the *positive evidence* that is available for it

Want to learn specifications

commutativity

$$f(x, y) = z \Leftrightarrow f(y, x) = z$$

transitivity

$$g(x, y) = t \wedge g(y, z) = t \Rightarrow g(x, z) = t$$

`sin` is periodic by 2π

$$\exists k.x = 2\pi k + y \Rightarrow sin(x) = z \Leftrightarrow sin(y) = z$$

rotating a shape by a multiple of 2π does not change the shape

$$\exists k.x = 2\pi k \Rightarrow rotate(y, x) = y$$

Example 3

f		
i_1	i_2	**r**
1	2	3
3	4	7
5	6	11
4	3	7
⋮	⋮	⋮

$$f(x, y) = z \Leftrightarrow f(y, x) = z$$

Specification

Example 3

f

i_1	i_2	**r**
1	2	3
3	4	7
5	6	11
4	3	7
⋮	⋮	⋮

$$f(x, y) = z \Leftrightarrow f(y, x) = z$$

Specification

positive and negative evidence

- positive evidence is $\{\{x \to 3, y \to 4\}, \{x \to 4, y \to 3\}\}$
- no negative evidence

Example 4

concat

i_1	i_2	**r**
a	b	ab
a	ϵ	a
ϵ	a	a
b	ϵ	b
⋮	⋮	⋮

len

i_1	**r**
a	1
ϵ	0
b	1
ab	2
⋮	⋮

Specification: $len(concat(x, y)) = z \Leftrightarrow len(x) = z$

Example 4

concat

i_1	i_2	**r**
a	b	ab
a	ϵ	a
ϵ	a	a
b	ϵ	b
⋮	⋮	⋮

len

i_1	**r**
a	1
ϵ	0
b	1
ab	2
⋮	⋮

Specification: $len(concat(x, y)) = z \Leftrightarrow len(x) = z$

positive and negative evidence

- positive evidence is $\{\{x \to a, y \to \epsilon\}, \{x \to b, y \to \epsilon\}\}$
- negative evidence is $\{\{x \to a, y \to b\}, \{x \to \epsilon, y \to a\}\}$

Example 4

add **constraint** to *weaken* the specification by finding a formula G s.t.

- for all negative evidences, G is unsat.
- for some positive evidences, G is sat.
- $G \Rightarrow \mathit{len}(\mathit{concat}(x, y) = z) \Leftrightarrow \mathit{len}(x) = z$ has some positive evidences and has no negative evidence.

$$y = \epsilon \Rightarrow \mathit{len}(\mathit{concat}(x, y)) = z \Leftrightarrow \mathit{len}(x) = z$$

Bach

A technique for discovering *likely specifications* from data generated for a number of standard libraries.

Bach

A techn*ique for discovering likely specifications from data generated* for a number

Discovers rich array of specifications

by combining novel insights of program synthesis and databases.

Specification

Consider specification as a formula over an interpreted theory

Specification ($\mathcal{F}$):

$$\forall V.G \Rightarrow (\Psi \Leftrightarrow \Phi) \text{ or } \forall V.G \Rightarrow (\Psi \Rightarrow \Phi),$$

where $\Psi = \wedge_i \psi_i$ and $\Phi = \wedge_j \phi_j$

- V : set of variables
- G : a formula over interpreted set of predicates and function symbols
- each ψ_i is an atom of the form $t = o$ (analogously, ϕ_i)
- t is a nested function application over $V, \sum$
- $\sum$ is a finite set of uninterpreted functions $\{f_1, ..., f_n\}$
- $o \in V$

Specification

Consider specification as a formula over an interpreted theory

Specification ($\mathcal{F}$):

$$\forall V.G \Rightarrow (\Psi \Leftrightarrow \Phi) \text{ or } \forall V.G \Rightarrow (\Psi \Rightarrow \Phi),$$

where $\Psi = \wedge_i \psi_i$ and $\Phi = \wedge_j \phi_j$

- V : set of variables
- G : a formula over interpreted set of predicates and function symbols
- each ψ_i is an atom of the form $t = o$ (analogously, ϕ_i)
- t is a nested function application over $V, \sum$
- $\sum$ is a finite set of uninterpreted functions $\{f_1, ..., f_n\}$
- $o \in V$

E.g. $\forall x, y.x > 0 \Rightarrow (f(g(x)) = y \Leftrightarrow g(f(x)) = y)$

Searching of specifications: Specification Induction

- iteratively constructs specifications by traversing set of programs and connections between them
- in order from smallest to largest based on a set of rules

Searching of specifications: Specification Induction

- iteratively constructs specifications by traversing set of programs and connections between them
- in order from smallest to largest based on a set of rules

Enumerative synthesis

Specification preference

- Given Ψ, Φ
  - learn $\Psi \Leftrightarrow \Phi$
  - if fail, learn either $\Psi \Rightarrow \Phi$ or $\Phi \Rightarrow \Psi$
  - If no implication can be learned, Bach resorts to abduction

Guard abduction

Bach solves a number of abduction problems to learn guard

- $G \Rightarrow (\Psi \Leftrightarrow \Phi)$, $G \Rightarrow (\Psi \Rightarrow \Phi)$, $G \Rightarrow (\Phi \Rightarrow \Psi)$
  - Each provided predicate is instantiated with every combination of variables
  - **E.g.** if $a > b$ is provided and $vars(\mathcal{F}) = \{x, y\}$, abduction will use $x > y$ and $y > x$

Specification Preference: Example

h_1	
i_1	**r**
1	true
2	false
3	true
⋮	⋮

h_2	
i_1	**r**
1	true
2	true
3	true
⋮	⋮

- $\Psi : h_1(x) = p, \Phi : h_2(x) = p$, where $p = \{true, false\}$

Specification: $h_1(x) = p \Leftrightarrow h_2(x) = p$

$(\Rightarrow)$

$$h_1(x) = p \Rightarrow h_2(x) = p$$

$(\Rightarrow)$

$$h_1(x) = p \Rightarrow h_2(x) = p$$

Negative evidence

$$\{x = 2, p = \mathit{false}\}$$

$(\Leftarrow)$

$$h_2(x) = p \Rightarrow h_1(x) = p$$

$(\Leftarrow)$

$$h_2(x) = p \Rightarrow h_1(x) = p$$

Negative evidence

$$\{x = 2, p = \mathit{true}\}$$

Guard abduction

- $G \Rightarrow (h_1(x) = p \Leftrightarrow h_2(x) = p)$
- $G \Rightarrow (h_1(x) = p \Rightarrow h_2(x) = p)$
- $G \Rightarrow (h_2(x) = p \Rightarrow h_1(x) = p)$

Guard abduction

- $G \Rightarrow (h_1(x) = p \Leftrightarrow h_2(x) = p)$
- $G \Rightarrow (h_1(x) = p \Rightarrow h_2(x) = p)$
- $G \Rightarrow (h_2(x) = p \Rightarrow h_1(x) = p)$

Learned specification

$$p = \mathit{true} \Rightarrow (h_1(x) = p \Rightarrow h_2(x) = p)$$

Specification Consistency Verification

How to efficiently verify the consistency of the specification with the dataset?

How to efficiently verify the consistency?

- model positive and negative evidence of a formula $\mathcal{F}$ and data-set D as a union of conjunctive query (UCQ).
- the evaluation should return the positive and negative evidence
- formulation as a database query evaluation allow us to leverage efficient, highly engineered database engines and Datalog server
- query is typically small and data is large

Encoding Specifications

Specification($\mathcal{F}$):

$$\forall \bar{\mathbf{x}}.\Psi \Leftrightarrow \Phi,$$

where $\Psi = \wedge_i \psi_i$ and $\Phi = \wedge_j \phi_j$

Encoding Specifications

Specification($\mathcal{F}$):

$$\forall \bar{\mathbf{x}}.\Psi \Leftrightarrow \Phi,$$

where $\Psi = \wedge_i \psi_i$ and $\Phi = \wedge_j \phi_j$

- for each var $x \in \bar{\mathbf{x}}$ in formula $\mathcal{F}$, create a Datalog variable $X_x \in \bar{\mathbf{X}}$, i.e.
  - $x \in \bar{\mathbf{x}} \Rightarrow X_x \in \bar{\mathbf{X}}$
- for each n-ary function f, create $(n+1)$-ary relation R_f

Encoding Specifications: Example

f

i_1	i_2	**r**
1	2	3
3	4	7
5	6	11
4	3	7
⋮	⋮	⋮

$$f(x, y) = z \Leftrightarrow f(y, x) = z$$

Specification

Encoding Specifications: Example

f		
i_1	i_2	**r**
1	2	3
3	4	7
5	6	11
4	3	7
⋮	⋮	⋮

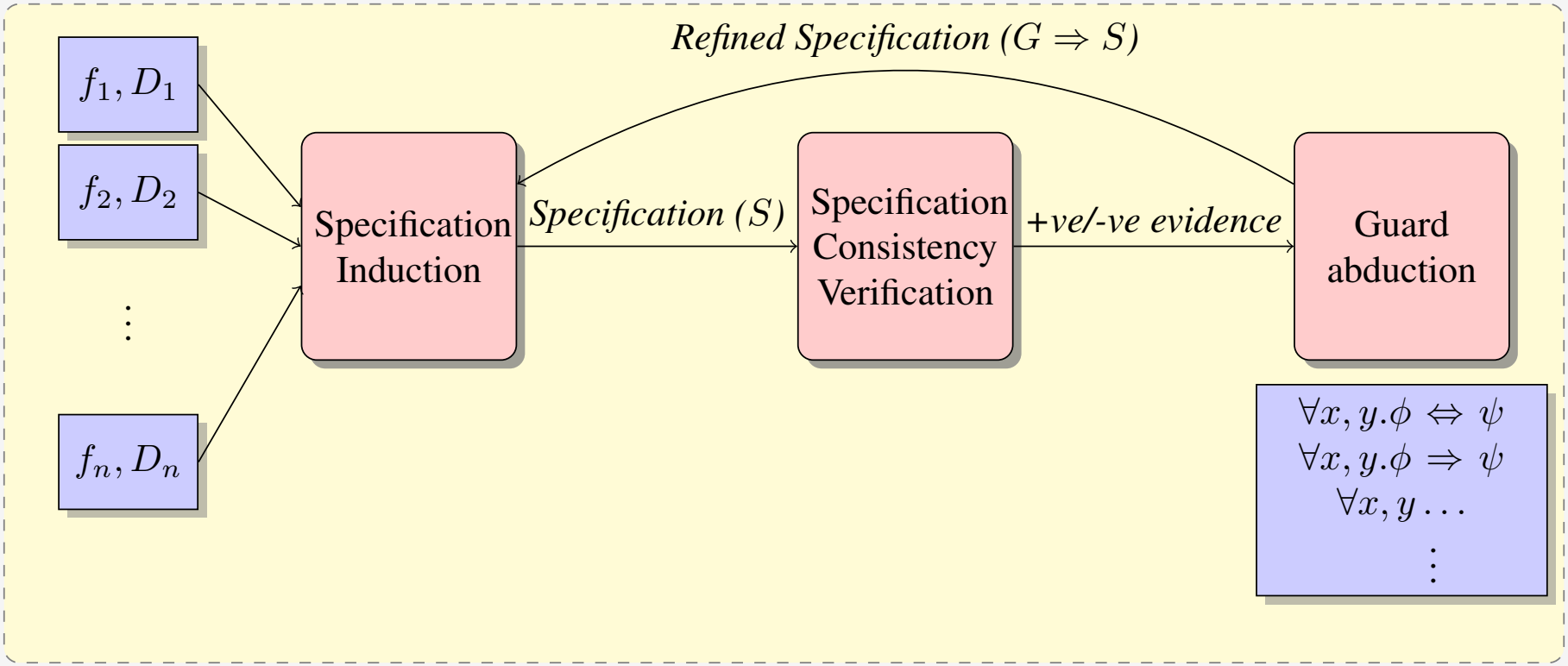
$$f(x, y) = z \Leftrightarrow f(y, x) = z$$

Specification

positive and negative evidence

- positive evidence $P(X) \leftarrow R_f(X_x, X_y, O), R_f(X_y, X_x, O'), O = O'$
- negative evidence $N(X) \leftarrow R_f(X_x, X_y, O), R_f(X_y, X_x, O'), O \neq O'$

Components of Bach



Implementation

- Implemented in OCaml.
- **Input:**
  - a signature of simply typed functions
  - i/o data for each function
  - a set of predicates to compute guards
- Uses Souffle Datalog engine to compute +ve and -ve evidence

Exploratory Evaluation

Targeted 9 set of python libraries. Each benchmark consists of

- a finite set of signature
- a set of predicates
- a data-set of 1000 randomly samples executions for each function

Benchmark	Description
`list` (7)	standard list operations, including `hd`, `tl`, `cons`, etc.
`matrix` (7)	matrix operations from Python's `sympy` library.
`trig` (4)	trig. functions (`sin`, `cos`, etc.) in Python's `math` module.
`z3` (5)	API to Python's `z3` library, including `sat` and `valid`.
`geometry` (5)	manipulations of shapes from Python's `sympy` library.
`sets` (10)	functions from Python's `set` module.
`dict` (5)	functions from Python's `dict` (dictionary) module.
`fp199` (4)	arithmetic on $\mathbb{F}_{199}$, the finite field of order 199.
`strings` (9)	string operations from Python's `string` module.

Figure: List of benchmarks; number of functions is in parentheses

z3 specifications

The z3 benchmark contains functions from a subset of Python's z3 API.

- Learned specification for z3
  - $p = true \Rightarrow (valid(x) = p \Rightarrow sat(x) = p)$
  - $and(x, y) = z \Leftrightarrow and(y, x) = z$
  - $valid(x) = p \wedge valid(y) = p \Rightarrow valid(and(x, y)) = p$

strings specifications

The strings benchmark contains the typical set of funtions for manipulating strings.

- Learned specification for strings
  - $lstrip(x) = y \Rightarrow lstrip(y) = y$
  - $p = true \Rightarrow (prefix(x, x) = p)$
  - $concat(y, reverse(y)) = x \Rightarrow reverse(x) = x$

trig specifications

The trig benchmark contains trigonometric functions from Python's math module.

- Learned specification for trig
  - $\exists k.x = 2\pi k + y \Rightarrow (sin(x) = z \Leftrightarrow sin(y) = z)$
  - $arcsin(z) = x \Rightarrow sin(x) = z$

geometry specifications

The geometry benchmark contains functions from sympy's geometry module.

- Learned specification for geometry
  - $b = true \Rightarrow$
    $(encl(x, y) = b \wedge encl\_pt(y, p) = true \Rightarrow encl\_pt(x, p) = true)$
  - $\exists k.x = 2\pi k \Rightarrow rotate(y, x) = y$

Empirical Evaluation: Scalability

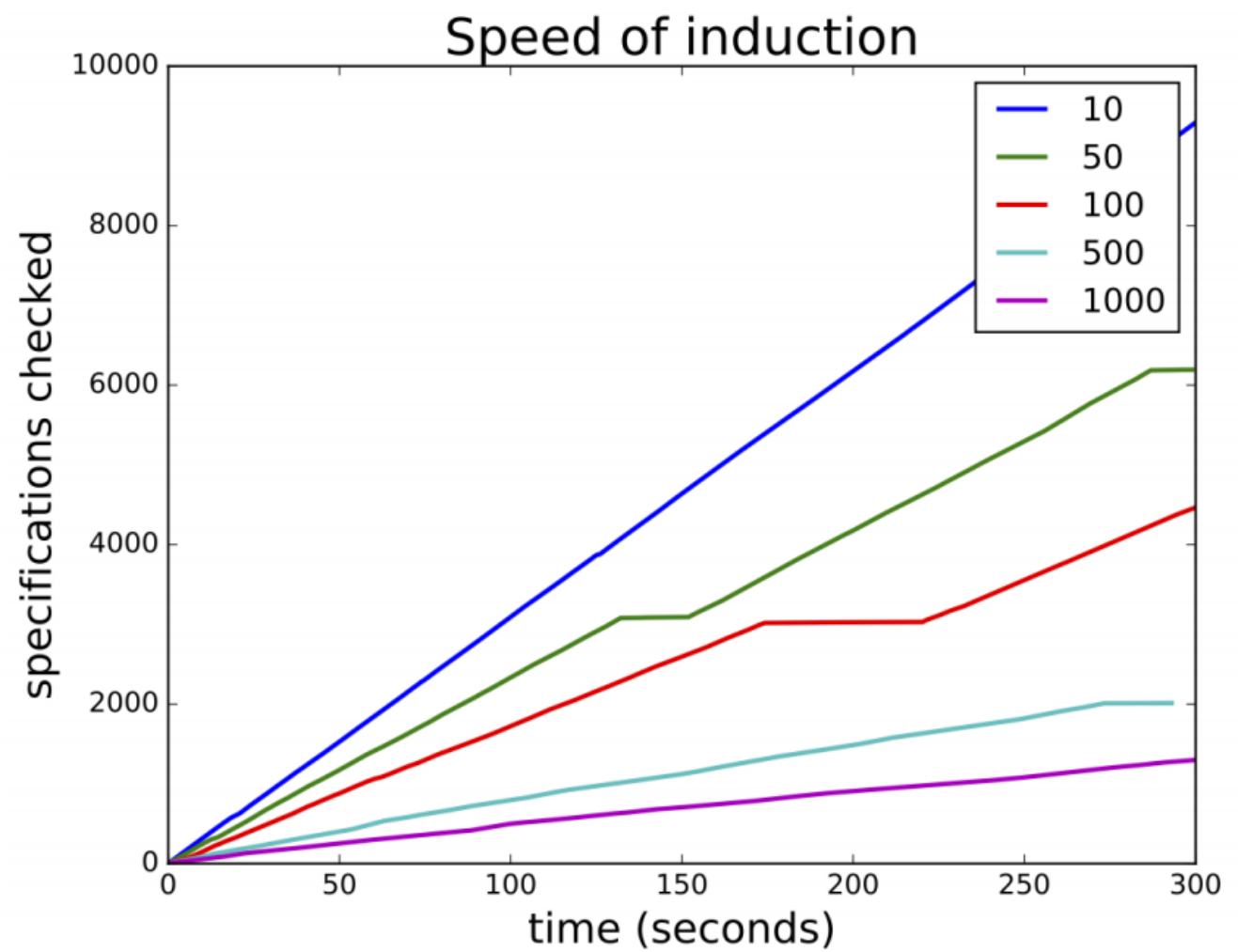


Figure: with more data, Bach checks less specification in same amount of time

Empirical Evaluation: Error Analysis

	10 observations			50 observations			100 observations			500 observations		
Benchmark	T_1	T_2	Size	T_1	T_2	Size	T_1	T_2	Size	T_1	T_2	Size
`ff199`	1.8	17.6	4.2	5.6	13.2	12.4	6.4	10.2	16.2	6.4	6.2	20.2
`trig`	2	19.2	10.8	2	0.8	29.2	0	0	28	0	0	28
`dict`	5.2	0.2	20	1.4	0	16.4	1	0	16	1	0	16
`geometry`	18	12	25	9	3.8	24.2	4	1	22	1	0	20
`lists`	40	17.6	52.4	4.8	0.4	34.4	1.4	0	31.4	0.4	0	30.4
`matrices`	18.2	11.2	25	15.4	3.2	30.2	7.8	0.6	25.2	5.2	0	19.4
`sets`	52.8	53.4	79.4	6.2	3.8	82.4	0.4	0	80.4	0	0	80
`strings`	159.6	234	215.6	85.6	50.8	324.8	22.2	1.6	310.6	0.2	0	290.2

Figure: Average correctness results

(T_1 is the type-1 error, T_2 is the type-2 error, Size is the number of specifications produced)

Empirical Evaluation: Error Analysis

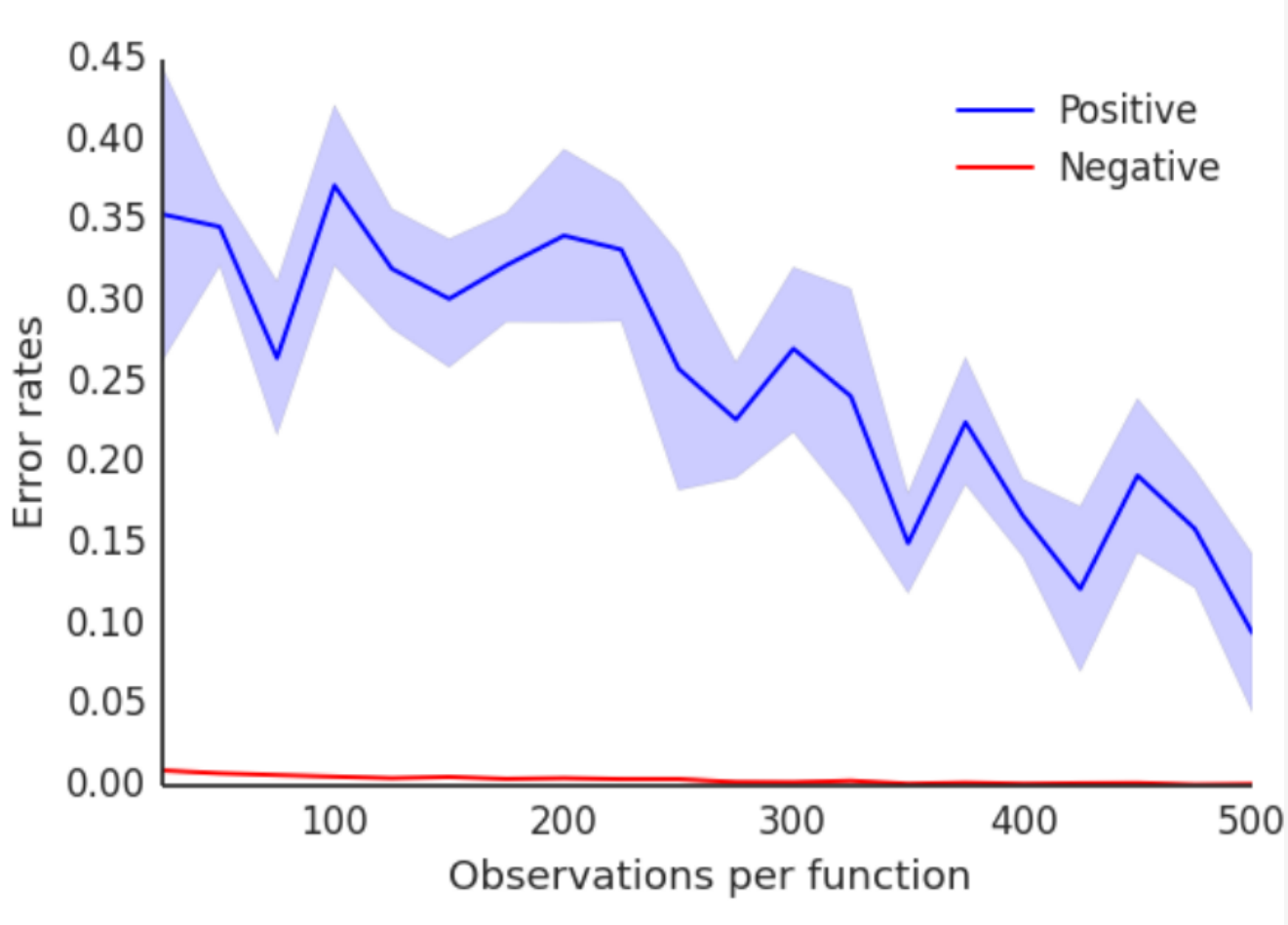


Figure: Worst-case benchmark's error rates with respect to number of observations

Empirical Evaluation: Error Analysis

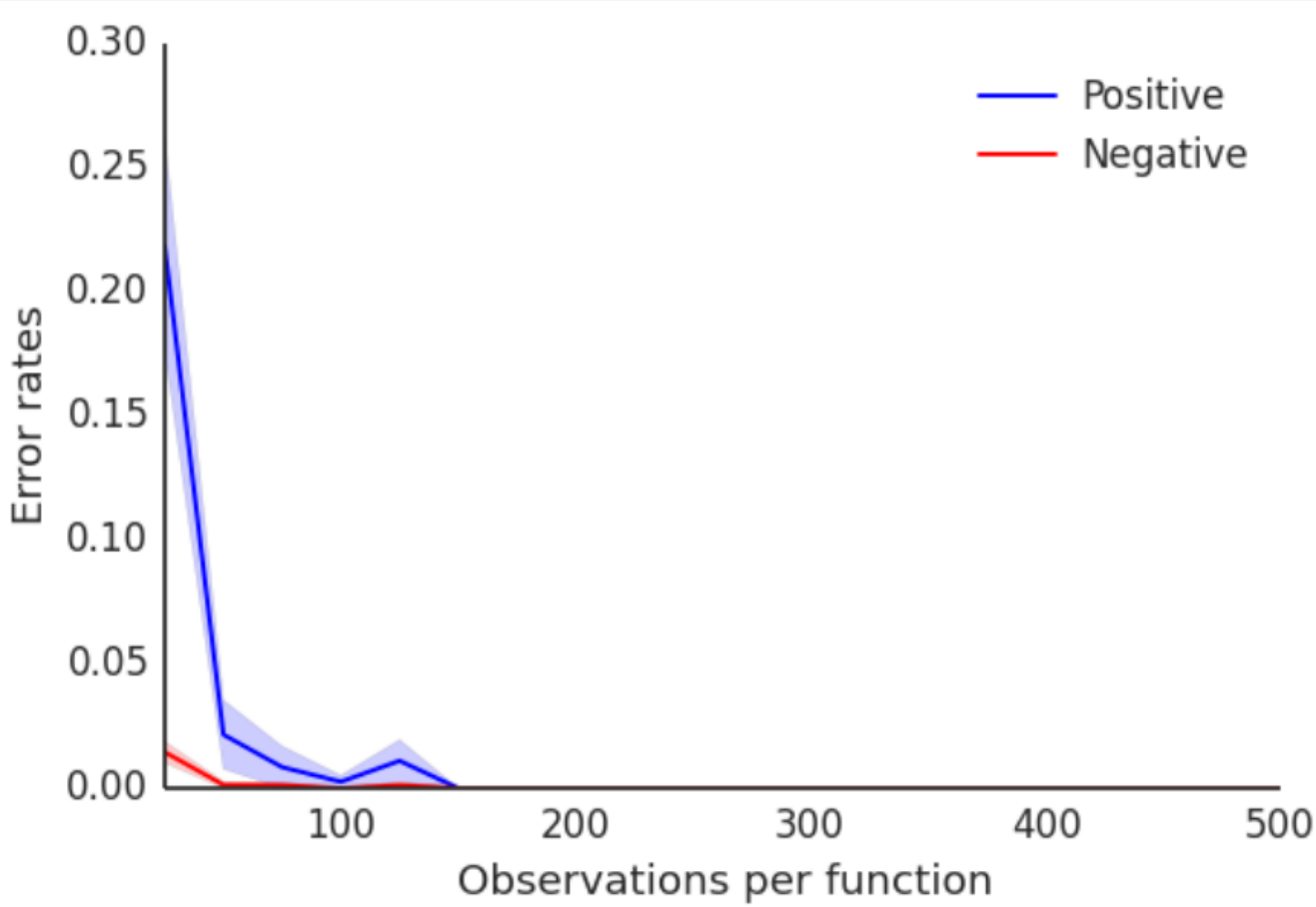


Figure: Best-case benchmark's error rates with respect to number of observations

Conclusion

- a technique for learning relational specification from i/o data
- learns specification that correlates different executions of multiple functions
- novel idea combining program synthesis and databases
- learns interesting specifications of real world libraries
- useful in program verification and development tasks

Questions?