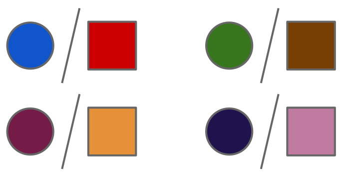
Learning Programs from Noisy Data

Veselin Raychev
Pavol Bielik
Martin Vechev
Andreas Krause

ETH Zurich

Why learn programs from examples?

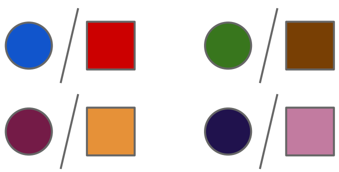
Input/output
examples



often easier to provide
examples than specification
(e.g. in FlashFill)

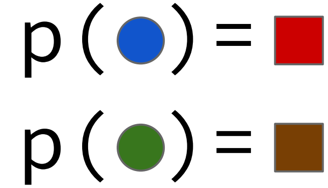
Why learn programs from examples?

Input/output examples



often easier to provide examples than specification (e.g. in FlashFill)

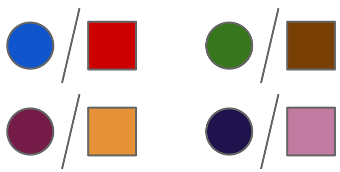
learn a function p such that



. . . ?

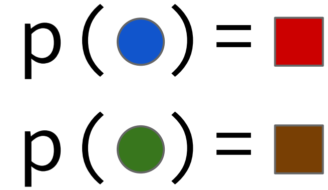
Why learn programs from examples?

Input/output
examples



often easier to provide
examples than specification
(e.g. in FlashFill)

learn a function p such that



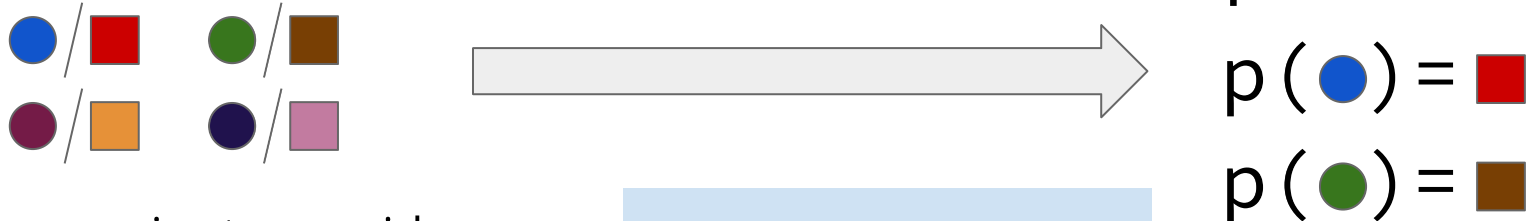
the user may make a **mistake** in the examples

. . . ?

Why learn programs from examples?

Input/output examples

learn a function p such that



often easier to provide examples than specification (e.g. in FlashFill)

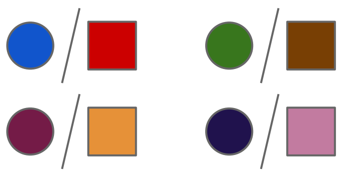
the user may make a **mistake** in the examples

. . . ?

Actual goal: produce p that the **user really wanted** and tried to specify

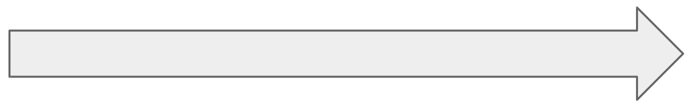
Why learn programs from examples?

Input/output
examples

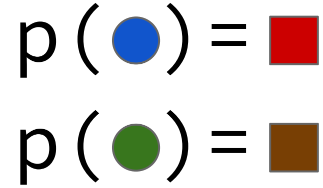


often easier to provide examples than specification (e.g. in FlashFill)

learn a function



p such that



. . . ?

the user may make a **mistake** in the examples

Actual goal: produce p that the **user really wanted** and tried to specify

Key problem of synthesis: overfits, not robust to noise

Learning Programs from Data: Defining Dimensions

Handling errors in the dataset

Number of Examples

Learned program complexity

Learning Programs from Data: Defining Dimensions

	Program synthesis (PL)
Handling errors in the dataset	**no**
Number of Examples	**tens**
Learned program complexity	**interesting programs**

Learning Programs from Data: Defining Dimensions

	Program synthesis (PL)	Deep learning (ML)
Handling errors in the dataset	**no**	**yes**
Number of Examples	**tens**	**millions**
Learned program complexity	**interesting programs**	**simple, but unexplainable functions**

Learning Programs from Data: Defining Dimensions

	Program synthesis (PL)	This paper bridges a gap	Deep learning (ML)
Handling errors in the dataset	**no**	**yes**	**yes**
Number of Examples	**tens**	**millions**	**millions**
Learned program complexity	**interesting programs**	**interesting programs**	**simple, but unexplainable functions**

Learning Programs from Data: Defining Dimensions

	Program synthesis (PL)	This paper bridges a gap	Deep learning (ML)
Handling errors in the dataset	**no**	**yes**	**yes**
Number of Examples	**tens**	**millions**	**millions**
Learned program complexity	**interesting programs**	**interesting programs**	**simple, but unexplainable functions**

expands capabilities of existing synthesizers

Learning Programs from Data: Defining Dimensions

	Program synthesis (PL)	This paper bridges a gap	Deep learning (ML)
Handling errors in the dataset	**no**	**yes**	**yes**
Number of Examples	**tens**	**millions**	**millions**
Learned program complexity	**interesting programs**	**interesting programs**	**simple, but unexplainable functions**

expands capabilities of existing synthesizers

new state-of-the-art precision for programming tasks

Learning Programs from Data: Defining Dimensions

	Program synthesis (PL)	This paper bridges a gap	Deep learning (ML)
Handling errors in the dataset			
Number of Examples			
Learned program complexity			

Bridges gap between ML and PL

Advances both areas

expands capabilities of existing synthesizers

new state-of-the-art precision for programming tasks

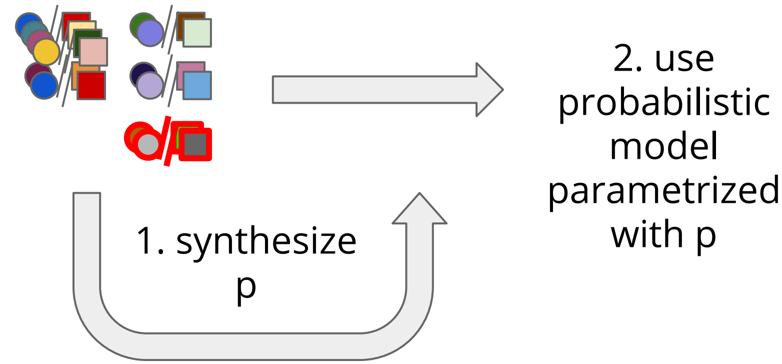
In this paper

- A general framework that handles
  - errors in training dataset
  - learns statistical models on data
  - handles synthesis with millions of examples

- Instantiated with two synthesizers
  - generalize existing works

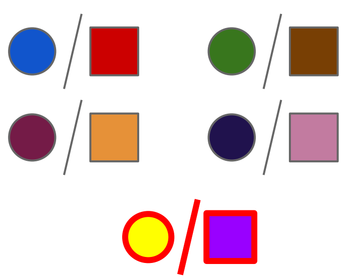
Contributions

New probabilistic models



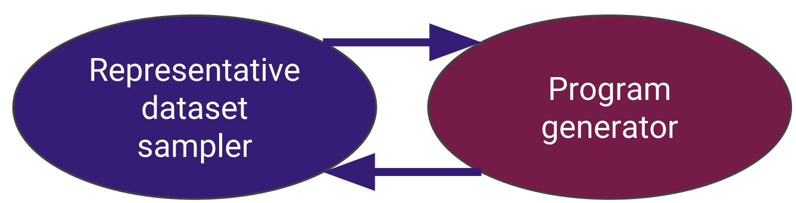
Handling noise

Input/output examples



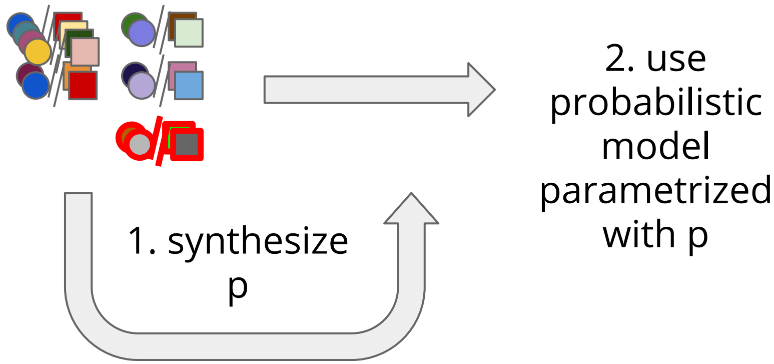
incorrect examples

Handling large datasets



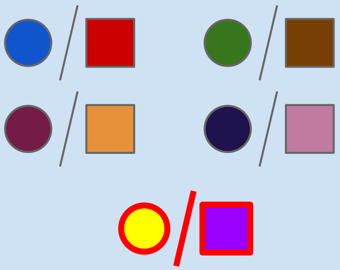
Contributions

New probabilistic models



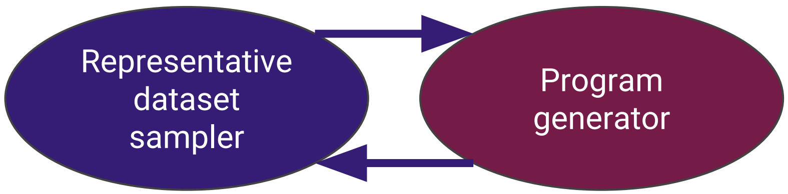
Handling noise

Input/output examples



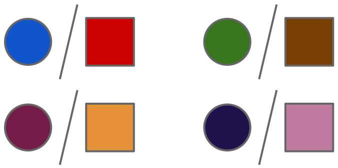
incorrect examples

Handling large datasets



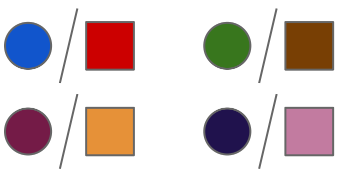
Synthesis with noise: usage model

Input/output
examples



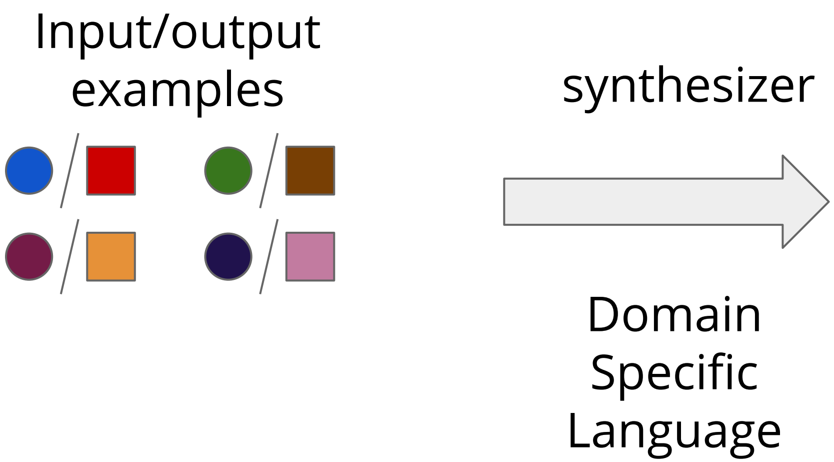
Synthesis with noise: usage model

Input/output examples

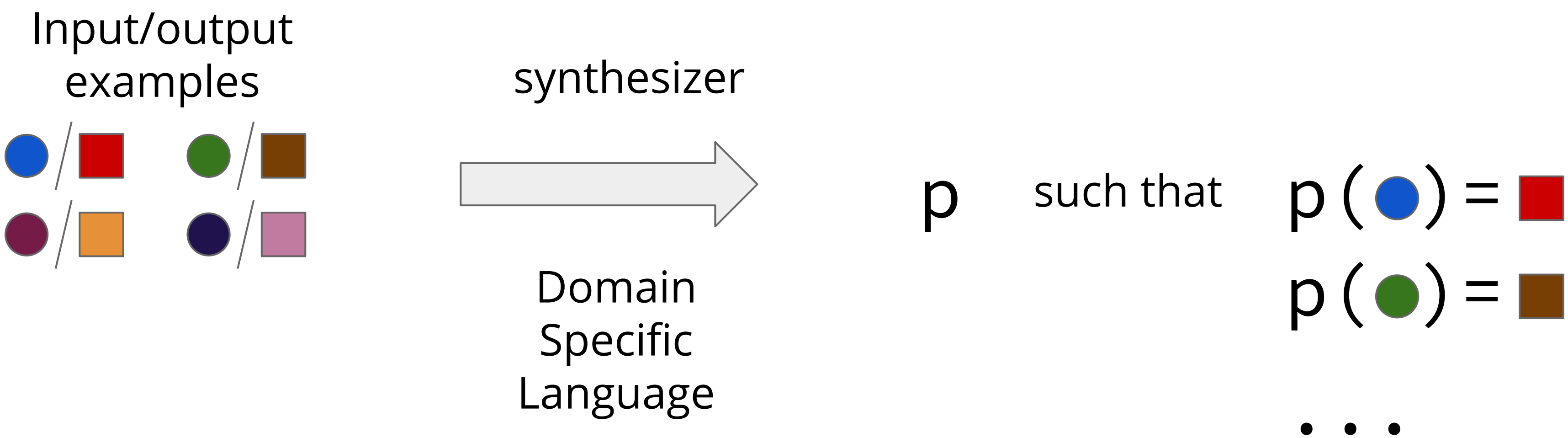


Domain Specific Language

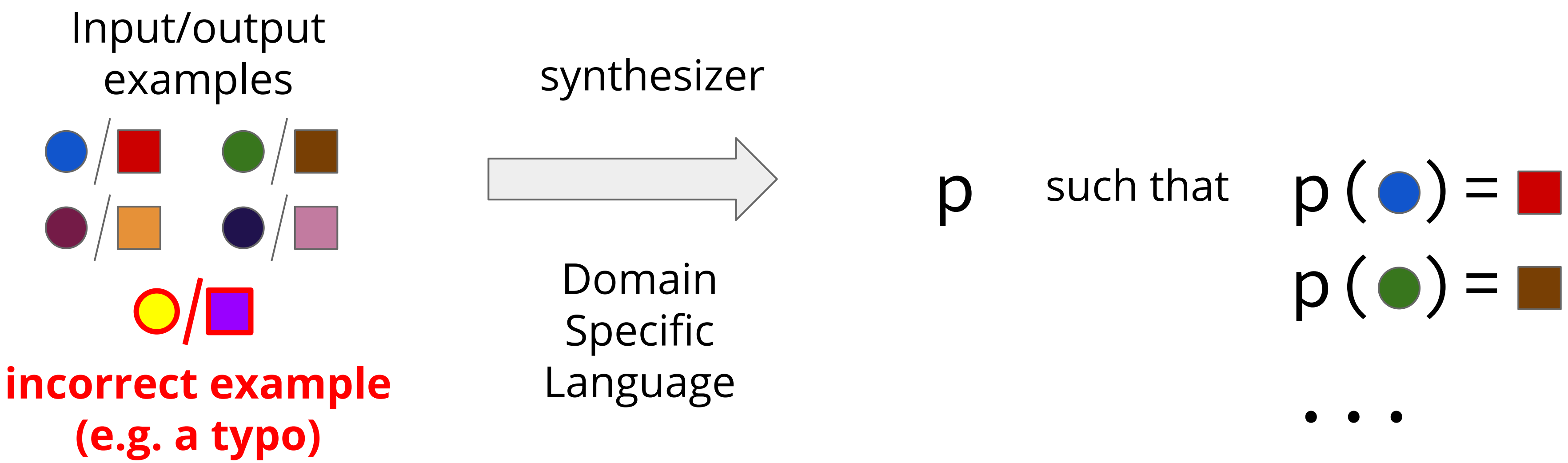
Synthesis with noise: usage model



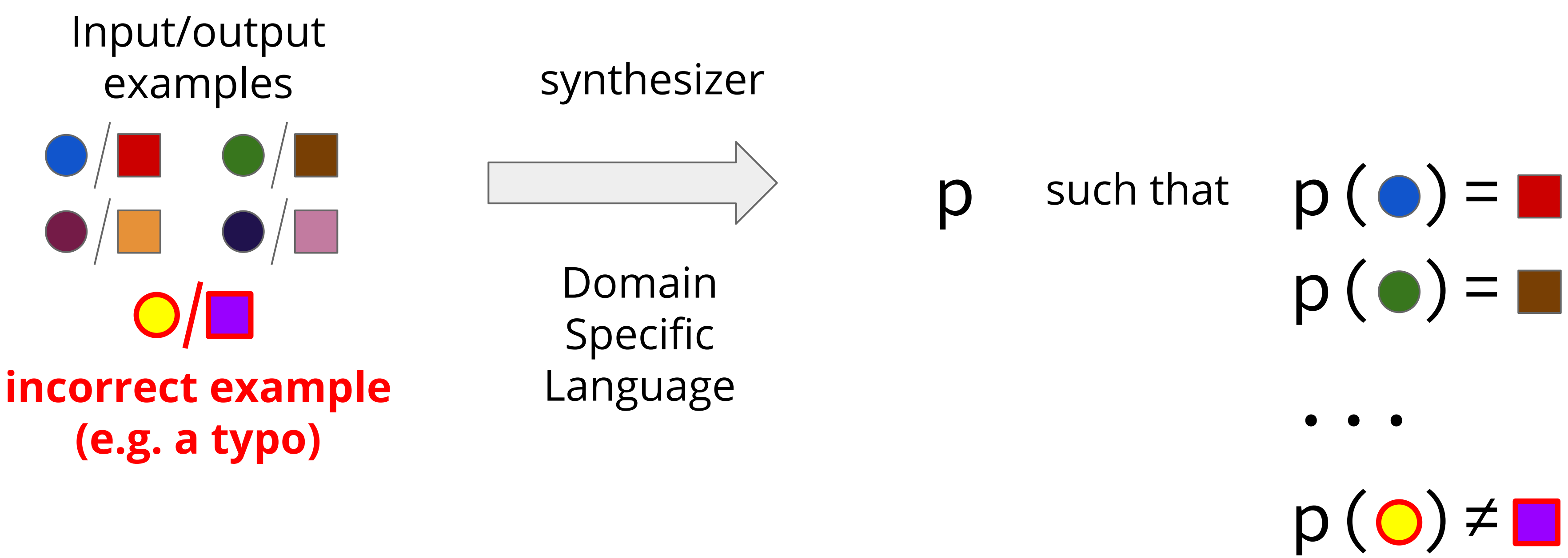
Synthesis with noise: usage model



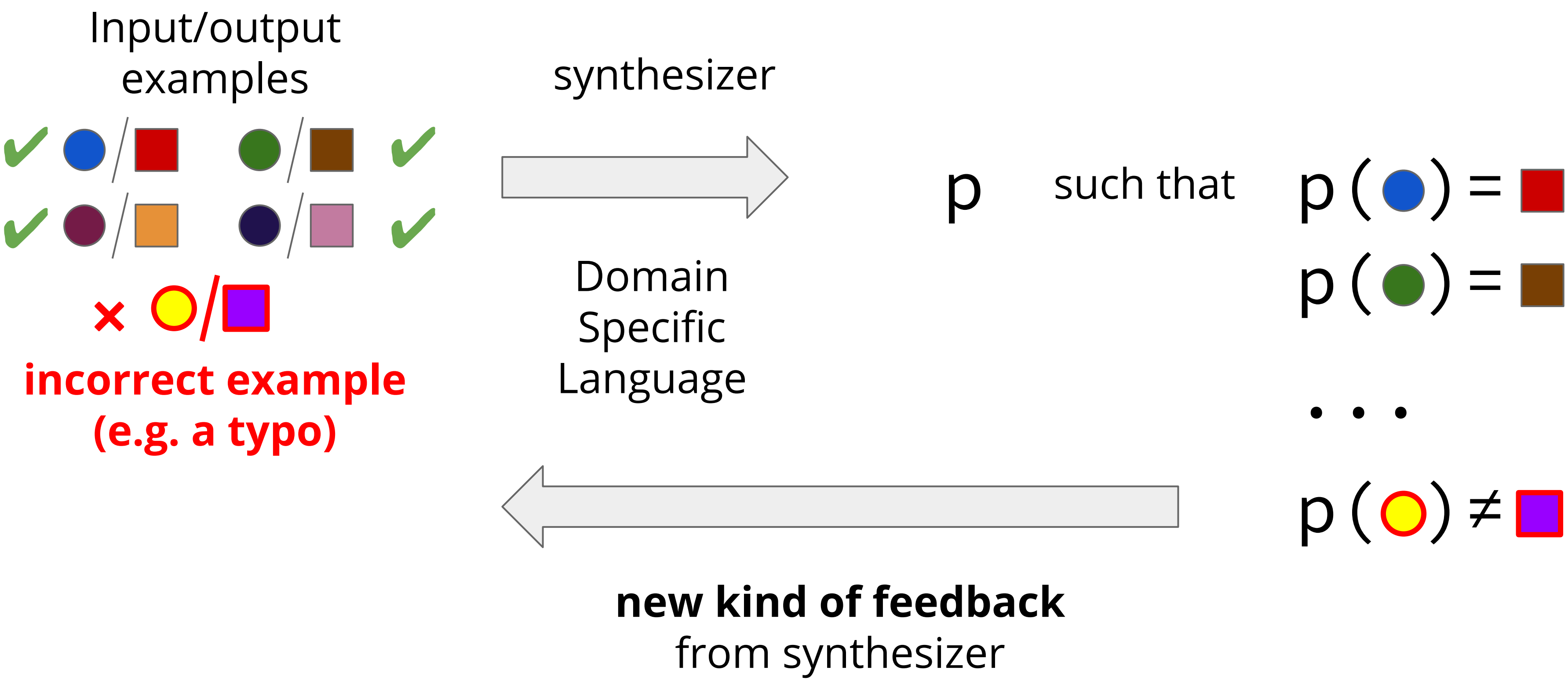
Synthesis with noise: usage model



Synthesis with noise: usage model

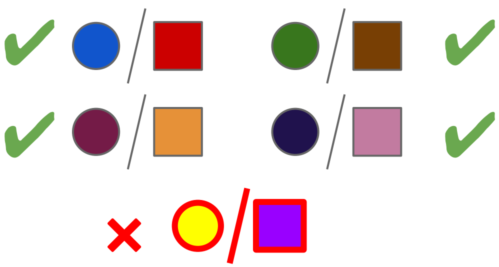


Synthesis with noise: usage model



Synthesis with noise: usage model

Input/output
examples

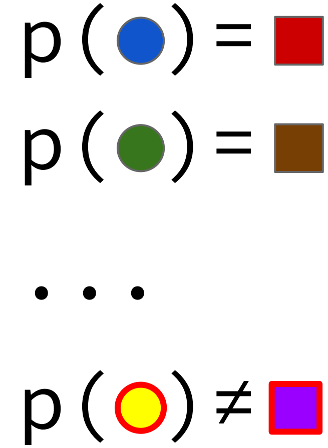


incorrect example
(e.g. a typo)

synthesizer

Domain
Specific
Language

p such that

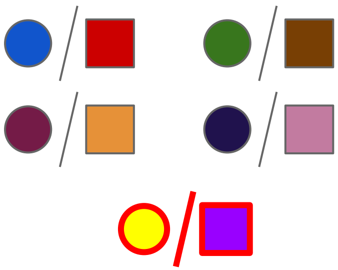


new kind of feedback
from synthesizer

- Tell user to remove **suspicious** example, or
- Ask for more examples

Handling noise: problem statement

D: Input/output examples



incorrect examples

synthesizer

dataset of input/output examples

$$p_{best} = \underset{p \in P}{\arg\min}\ \mathrm{errors}(D, p)$$

space of possible programs in DSL

Too long program, hardcodes the input/outputs.
Synthesis must penalize such answers

Our problem formulation:

regularization constant

$$p_{best} = \underset{p \in P}{\arg\min}\ \mathrm{errors}(D, p) + \lambda r(p)$$

error rate

regularizer penalizes long programs

Noisy synthesis using SMT

total solution

cost

$$p_{best} = \arg\min_{p \in P} \boxed{\text{errors}(D,\ p) + \lambda r(p)}$$

number of instructions

Noisy synthesis using SMT

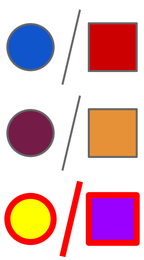
total solution

cost

$$p_{best} = \arg\min_{p \in P} \boxed{\mathrm{errors}(D, p) + \lambda r(p)}$$

number of

instructions

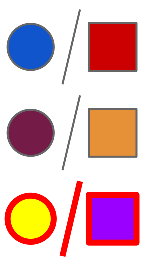


Noisy synthesis using SMT

$$p_{best} = \arg\min_{p \in P} \boxed{errors(D, p) + \lambda r(p)}$$

total solution **cost**

number of instructions



encoding

err$_1$ = if p( ) = then 0 else 1
err$_2$ = if p( ) = then 0 else 1
err$_3$ = if p( ) = then 0 else 1

errors = err$_1$ + err$_2$ + err$_3$
p $\in$ P_r (with **r** instructions)

Ψ

formula given to SMT solver

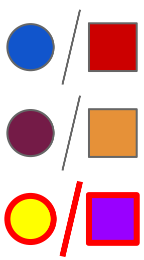
Noisy synthesis using SMT

total solution

cost

$$p_{best} = \arg\min_{p \in P} \boxed{errors(D,\ p)\ +\ \lambda r(p)}$$

number of instructions



encoding

err$_1$ = if p( ) = then 0 else 1
err$_2$ = if p( ) = then 0 else 1
err$_3$ = if p( ) = then 0 else 1

errors = err$_1$ + err$_2$ + err$_3$
p ∈ P$_r$ (with **r** instructions)

Ψ

formula given to SMT solver

Ask a number of SMT queries in increasing value of solution cost

e.g. for

$\lambda = 0.6$

costs are

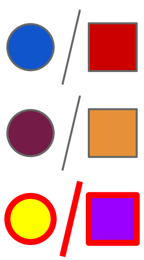
cost		number of **errors**			
		0	1	2	3
r	1	**0.6**	**1.6**	**2.6**	**3.6**
	2	**1.2**	**2.2**	**3.2**	**4.2**
	3	**1.8**	**2.8**	**3.8**	**4.8**

Noisy synthesis using SMT

total solution
cost

$$p_{best} = \arg\min_{p \in P} \boxed{errors(D, p) + \lambda r(p)}$$

number of
instructions



encoding

err_1 = if p( ) = then 0 else 1
err_2 = if p( ) = then 0 else 1
err_3 = if p( ) = then 0 else 1

errors = $err_1 + err_2 + err_3$
$p \in P_r$ (with **r** instructions)

Ψ

formula given
to SMT solver

Ask a number of
SMT queries
in increasing value
of solution cost

e.g. for
$\lambda = 0.6$
costs are

cost		number of **errors**			
		0	1	2	3
r	1	**UNSAT**	**1.6**	**2.6**	**3.6**
	2	**1.2**	**2.2**	**3.2**	**4.2**
	3	**1.8**	**2.8**	**3.8**	**4.8**

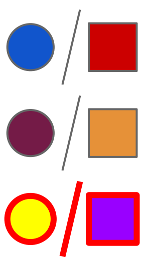
Noisy synthesis using SMT

total solution

cost

$$p_{best} = \arg\min_{p \in P} \boxed{\text{errors}(D, p) + \lambda r(p)}$$

number of instructions



encoding

err_1 = if p( )= then 0 else 1
err_2 = if p( )= then 0 else 1
err_3 = if p( )= then 0 else 1

errors = $err_1 + err_2 + err_3$
$p \in P_r$ (with **r** instructions)

Ψ

formula given to SMT solver

Ask a number of SMT queries in increasing value of solution cost

e.g. for

$\lambda = 0.6$

costs are

cost		number of **errors**			
		0	1	2	3
r	1	UNSAT	**1.6**	**2.6**	**3.6**
	2	UNSAT	**2.2**	**3.2**	**4.2**
	3	**1.8**	**2.8**	**3.8**	**4.8**

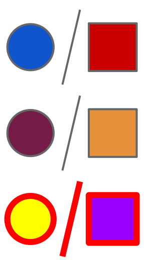
Noisy synthesis using SMT

$$p_{best} = \underset{p \in P}{\arg\min} \boxed{errors(D, p) + \lambda r(p)}$$

total solution

cost

number of instructions



encoding

err$_1$ = if p( )= then 0 else 1
err$_2$ = if p( )= then 0 else 1
err$_3$ = if p( )= then 0 else 1

errors = err$_1$ + err$_2$ + err$_3$
p ∈ P$_r$ (with **r** instructions)

Ψ

formula given to SMT solver

Ask a number of SMT queries in increasing value of solution cost

e.g. for

$\lambda = 0.6$

costs are

cost		number of **errors** 0	1	2	3
r	1	UNSAT	UNSAT	**2.6**	**3.6**
	2	UNSAT	**2.2**	**3.2**	**4.2**
	3	**1.8**	**2.8**	**3.8**	**4.8**

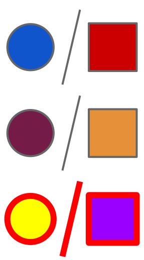
Noisy synthesis using SMT

$$p_{best} = \underset{p \in P}{\arg\min} \boxed{\mathrm{errors}(D, p) + \lambda r(p)}$$

total solution

cost

number of instructions



encoding

err_1 = if p( ) = then 0 else 1
err_2 = if p( ) = then 0 else 1
err_3 = if p( ) = then 0 else 1

errors = err_1 + err_2 + err_3
p ∈ P_r (with **r** instructions)

Ψ

formula given to SMT solver

Ask a number of SMT queries in increasing value of solution cost

e.g. for

$\lambda = 0.6$

costs are

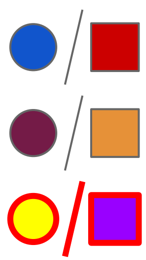
cost		number of **errors**			
		0	1	2	3
r	1	UNSAT	UNSAT	**2.6**	**3.6**
	2	UNSAT	**2.2**	**3.2**	**4.2**
	3	UNSAT	**2.8**	**3.8**	**4.8**

Noisy synthesis using SMT

$$p_{best} = \underset{p \in P}{\arg\min} \boxed{errors(D, p) + \lambda r(p)}$$

total solution
cost

number of
instructions



encoding

err_1 = if $p(\ ) = \ $ then 0 else 1
err_2 = if $p(\ ) = \ $ then 0 else 1
err_3 = if $p(\ ) = \ $ then 0 else 1

errors = $err_1 + err_2 + err_3$
$p \in P_r$ (with **r** instructions)

Ψ

formula given
to SMT solver

Ask a number of
SMT queries
in increasing value
of solution cost

e.g. for
$\lambda = 0.6$
costs are

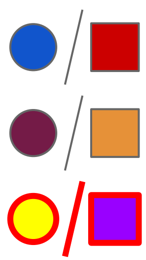
cost		number of **errors**			
		0	1	2	3
r	1	UNSAT	UNSAT	**2.6**	**3.6**
	2	UNSAT	SAT	**3.2**	**4.2**
	3	UNSAT	**2.8**	**3.8**	**4.8**

Noisy synthesis using SMT

$$p_{best} = \underset{p \in P}{\arg\min} \boxed{\mathrm{errors}(D, p) + \lambda r(p)}$$

total solution
cost

number of
instructions



encoding

err_1 = if p(●) = ■ then 0 else 1
err_2 = if p(●) = ■ then 0 else 1
err_3 = if p(●) = ■ then 0 else 1

errors = $err_1 + err_2 + err_3$
$p \in P_r$ (with **r** instructions)

Ψ

formula given
to SMT solver

Ask a number of
SMT queries
in increasing value
of solution cost

e.g. for

$\lambda = 0.6$

costs are

cost		number of **errors**			
		0	1	2	3
r	1	UNSAT	UNSAT	2.6	
	2	UNSAT	SAT		
	3	UNSAT	2.8	3.8	

best program is
with two
instructions and
makes one error

Noisy synthesizer: example

Take an actual synthesizer and
show that we can make it handle noise

Implementation: BitSyn

For BitStream programs, using Z3

similar to Jha et al.[ICSE'10] and Gulwani et al.[PLDI'11]

Example program:

```
function check_if_power_of_2(int32 x) {
    var o = add(x, 1)
    return bitwise_and(x, o)        ← synthesized, short loop-free programs
}
```

Implementation: BitSyn

For BitStream programs, using Z3

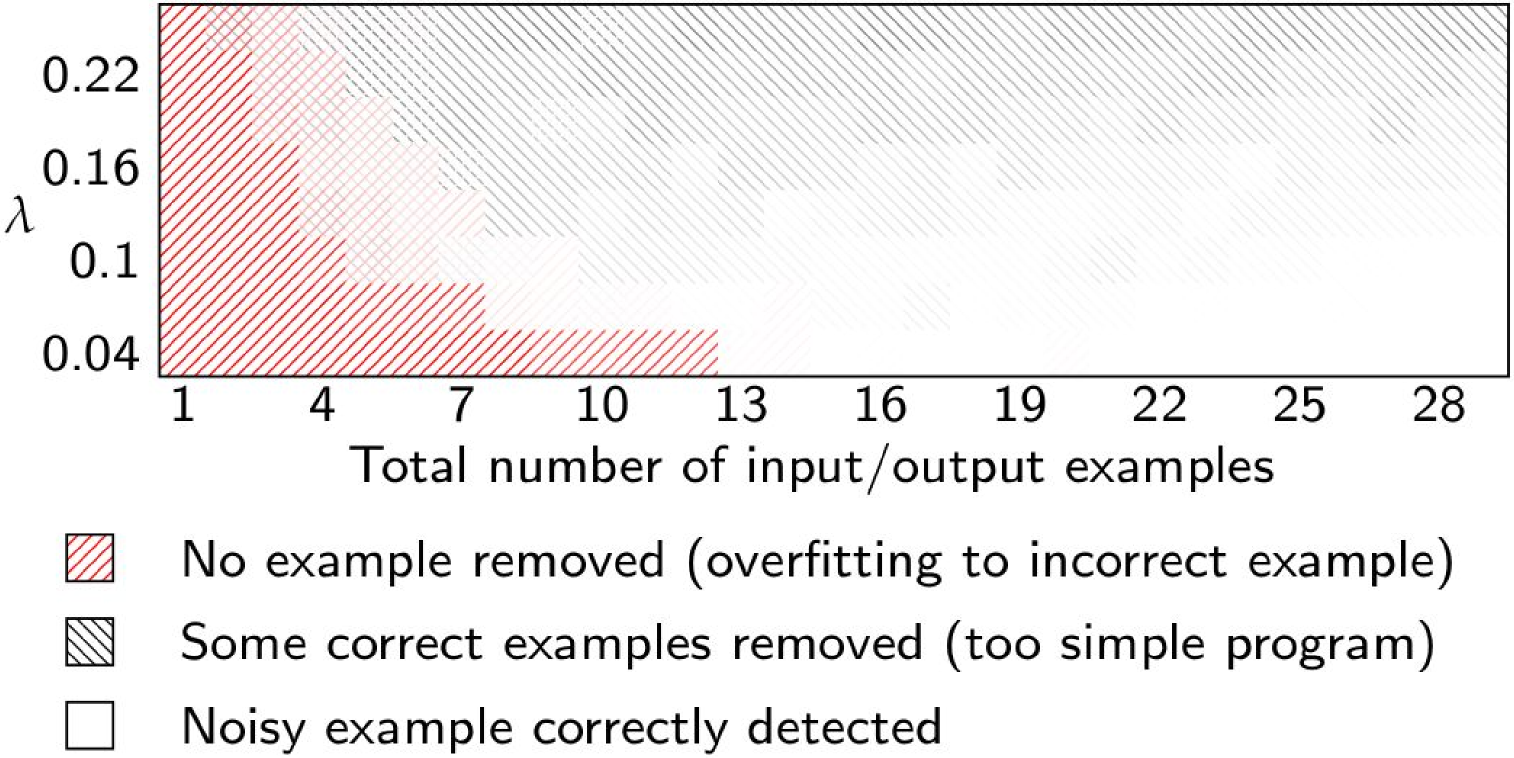
similar to Jha et al.[ICSE'10] and Gulwani et al.[PLDI'11]

Example program:

```
function check_if_power_of_2(int32 x) {
    var o = add(x, 1)
    return bitwise_and(x, o)      <--- synthesized, short loop-free programs
}
```

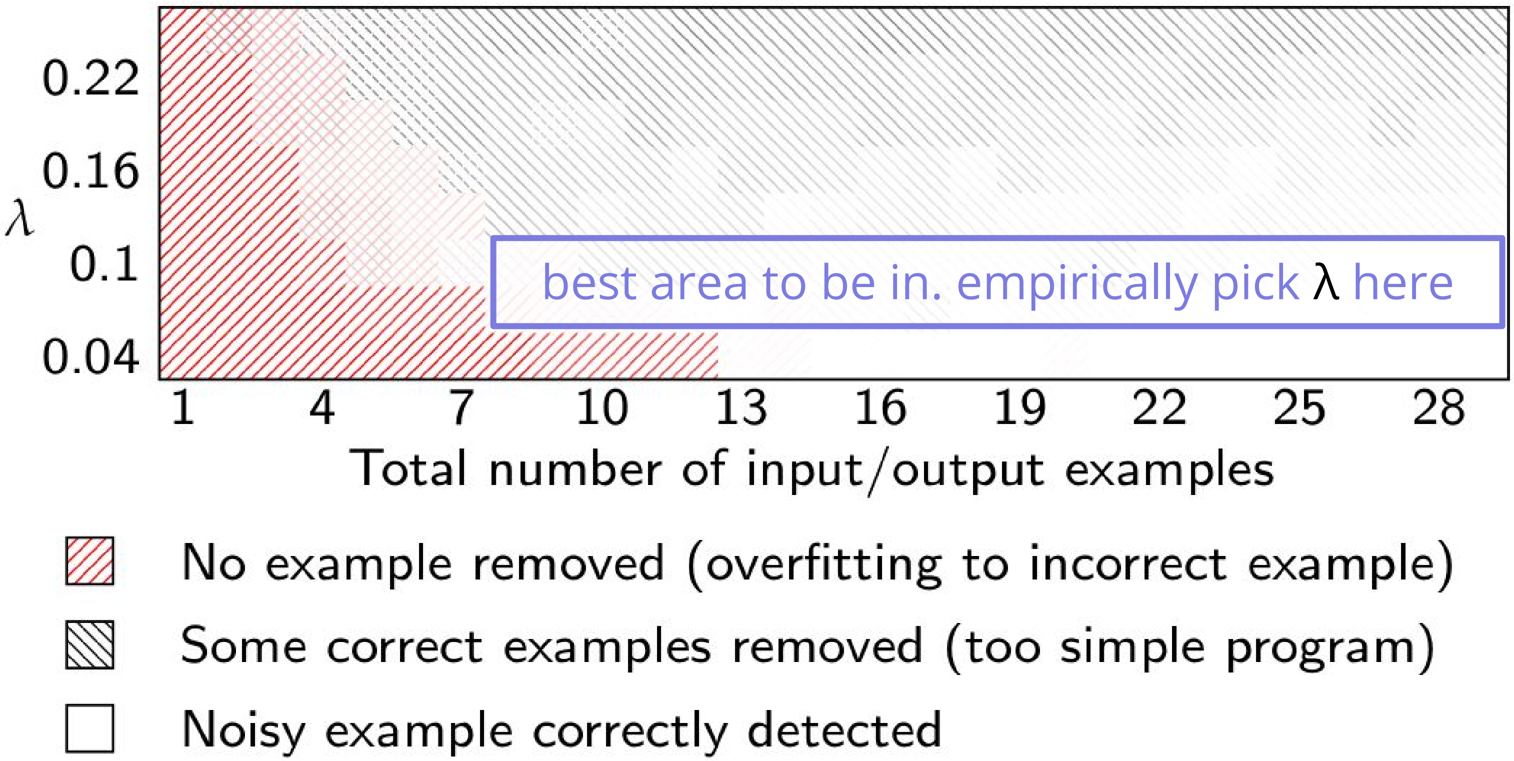
Question: how well does our synthesizer discover noise?
(in programs from prior work)

Implementation: BitSyn



Question: how well does our synthesizer discover noise?
(in programs from prior work)

Implementation: BitSyn



Question: how well does our synthesizer discover noise?
(in programs from prior work)

So far... handling noise

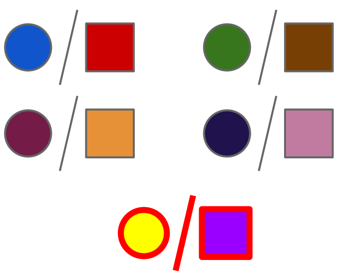
- Problem statement and regularization
- Synthesis procedure using SMT
- Presented one synthesizer

Handling noise enables us to solve new classes of problems beyond normal synthesis

Contributions

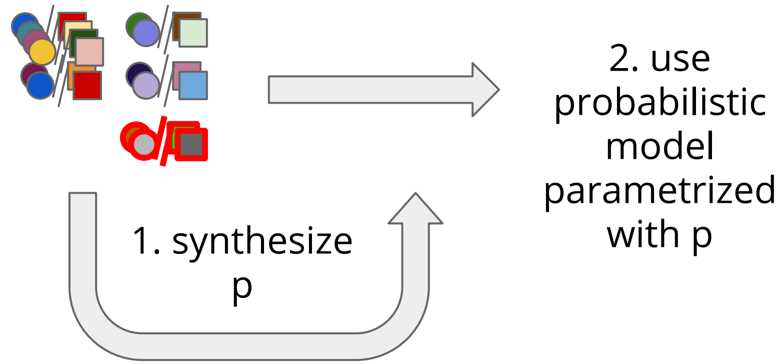
Handling noise

Input/output examples

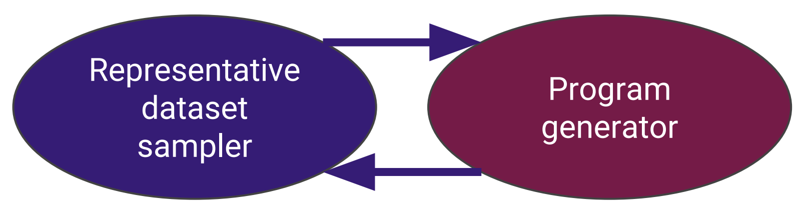


incorrect examples

New probabilistic models



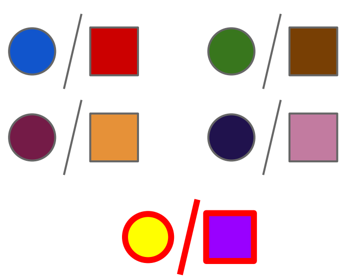
Handling large datasets



Contributions

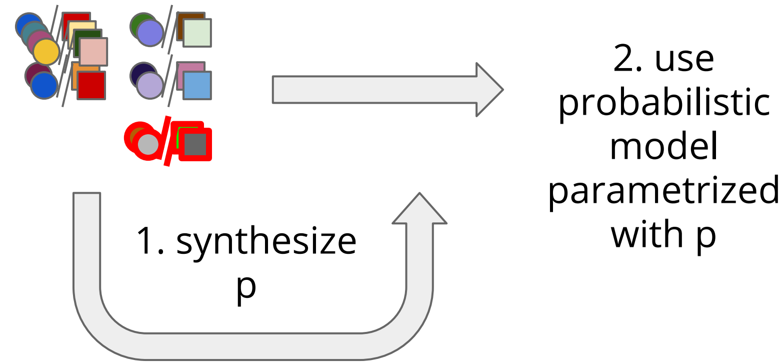
Handling noise

Input/output examples

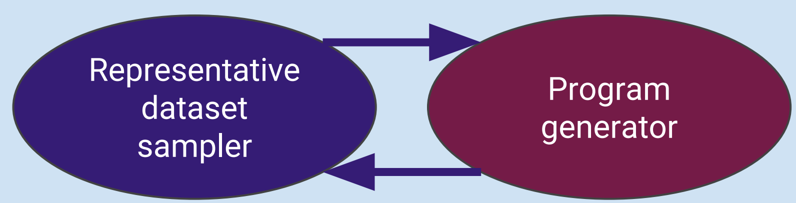


incorrect examples

New probabilistic models



Handling large datasets



Fundamental problem

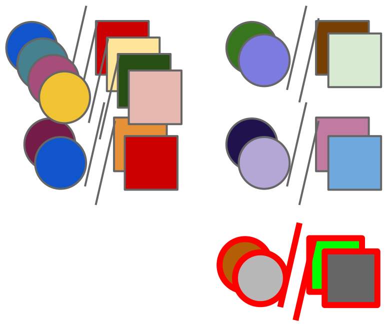
Large number of examples:

$$p_{best} = \arg\min_{p \in P} \mathrm{cost}(D, p)$$

Fundamental problem

Large number of examples:

$$p_{best} = \arg\min_{p \in P} \text{cost}(D, p)$$



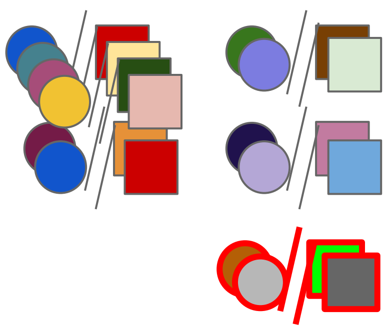
D

Millions of input/output examples

Fundamental problem

Large number of examples:

$$p_{best} = \arg\min_{p \in P} \mathrm{cost}(D, p)$$



D

Millions of input/output examples

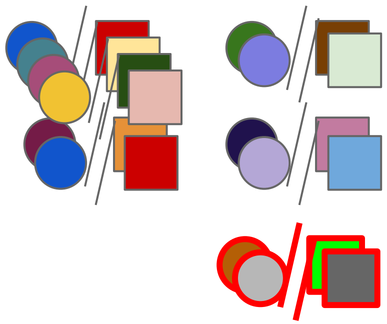
computing cost(D, p)

O(|D|)

Fundamental problem

Large number of examples:

$$p_{best} = \arg\min_{p \in P} \mathrm{cost}(D,\ p)$$



D

Millions of input/output examples

computing $\mathrm{cost}(D,\ p)$

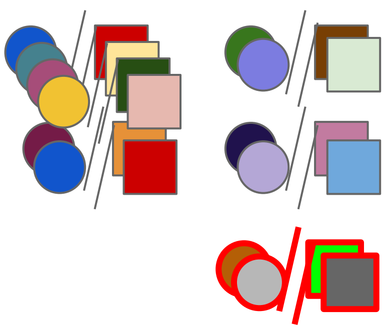
$O(\ |D|\ )$

Synthesis: practically intractable

Fundamental problem

Large number of examples:

$$p_{best} = \underset{p \in P}{\arg\min}\ \mathrm{cost}(D, p)$$



D

Millions of input/output examples

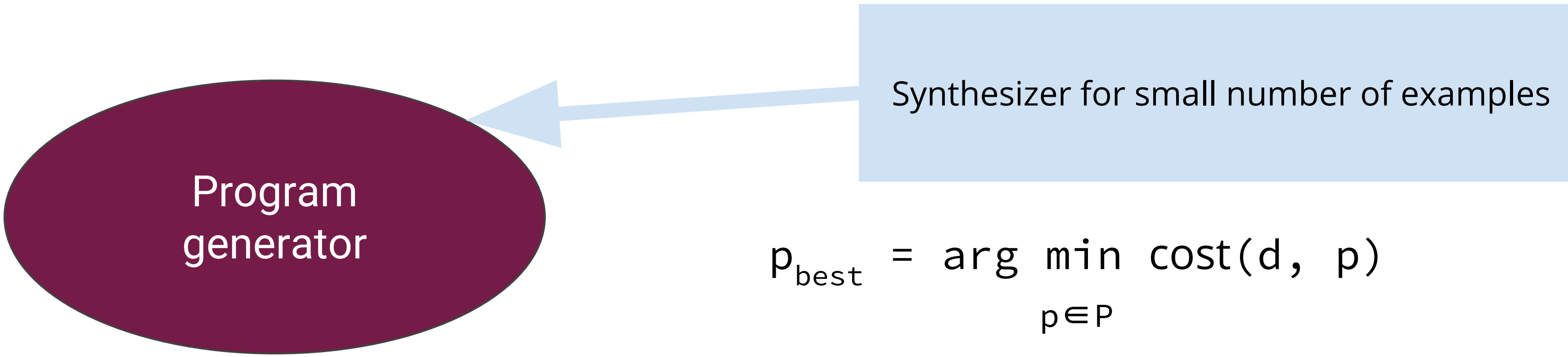
computing $\mathrm{cost}(D, p)$

$O(\ |D|\ )$

Synthesis: practically intractable

Key idea: iterative synthesis on fraction of examples

Our solution: two components



$$p_{best} = \arg\min_{p \in P} \mathrm{cost}(d, p)$$

given dataset d, finds best program

Our solution: two components

Synthesizer for small number of examples

Program generator

$$p_{best} = \arg\min_{p \in P} \mathrm{cost}(d, p)$$

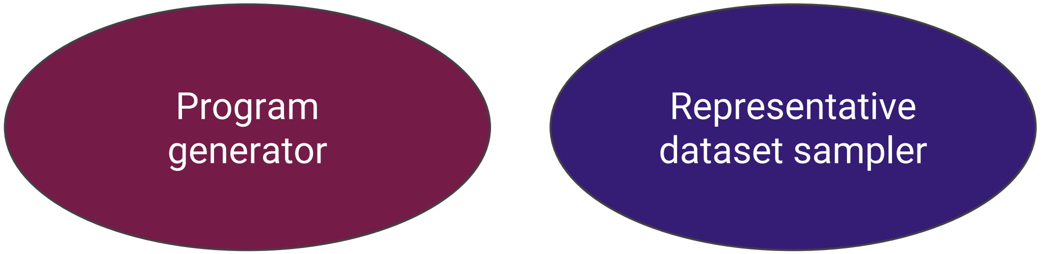
given dataset d, finds best program

Dataset sampler

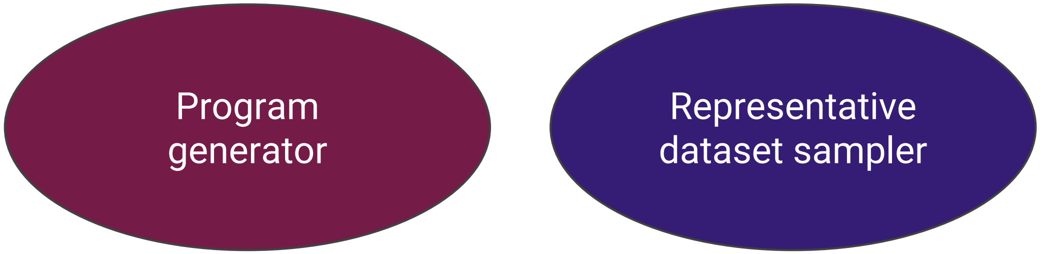
Picks dataset d ⊆ D

We introduce representative dataset sampler
Generalize a user providing input/output examples

In a loop



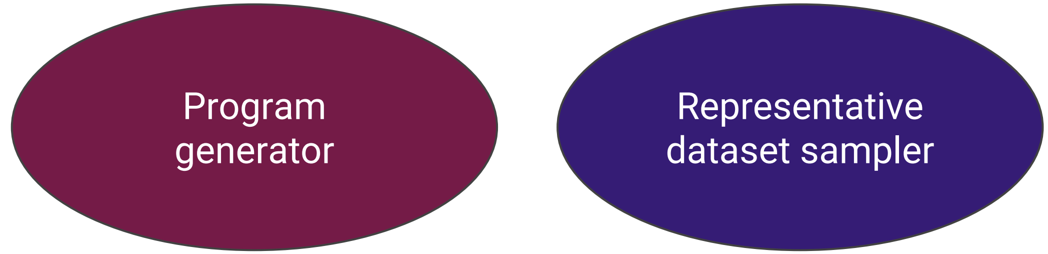
In a loop



Start with a small random sample d⊆D

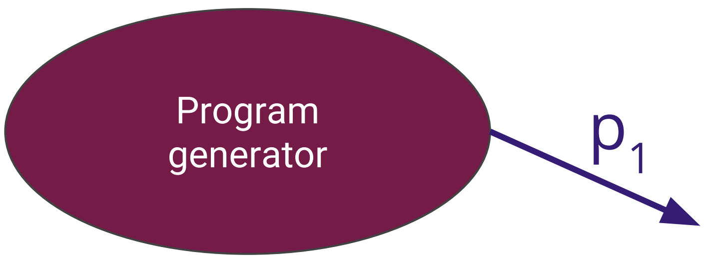
Iteratively generate programs and samples.

In a loop

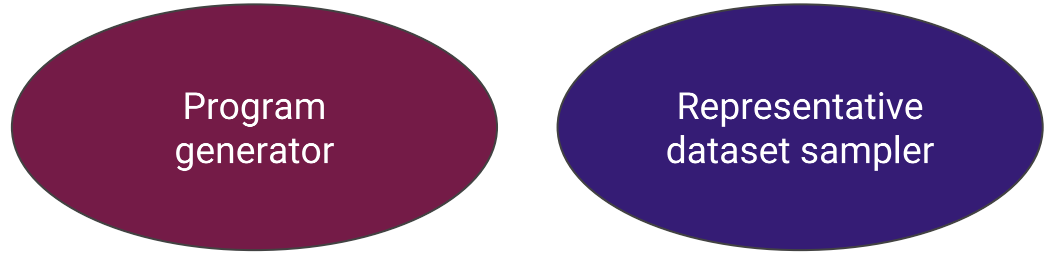


Start with a small random sample d⊆D

Iteratively generate programs and samples.

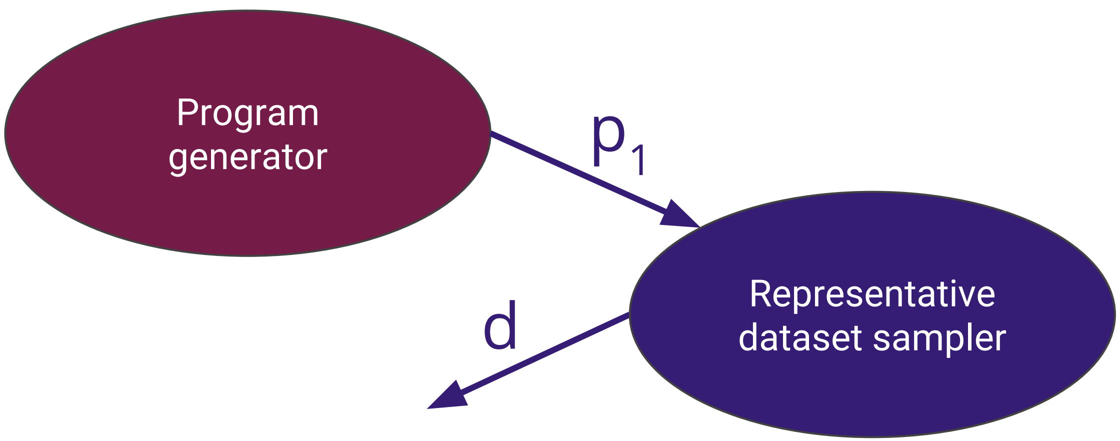


In a loop

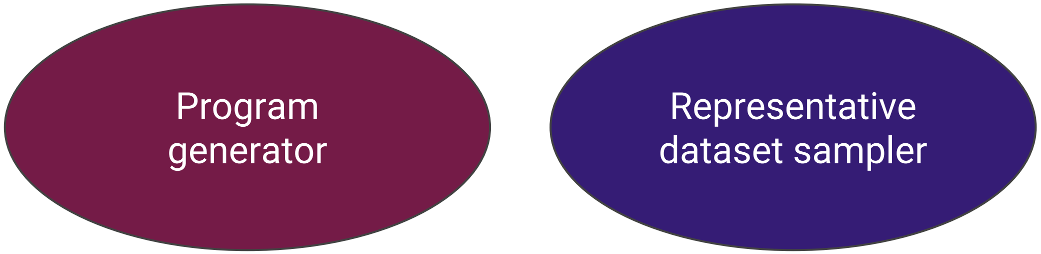


Start with a small random sample d⊆D

Iteratively generate programs and samples.

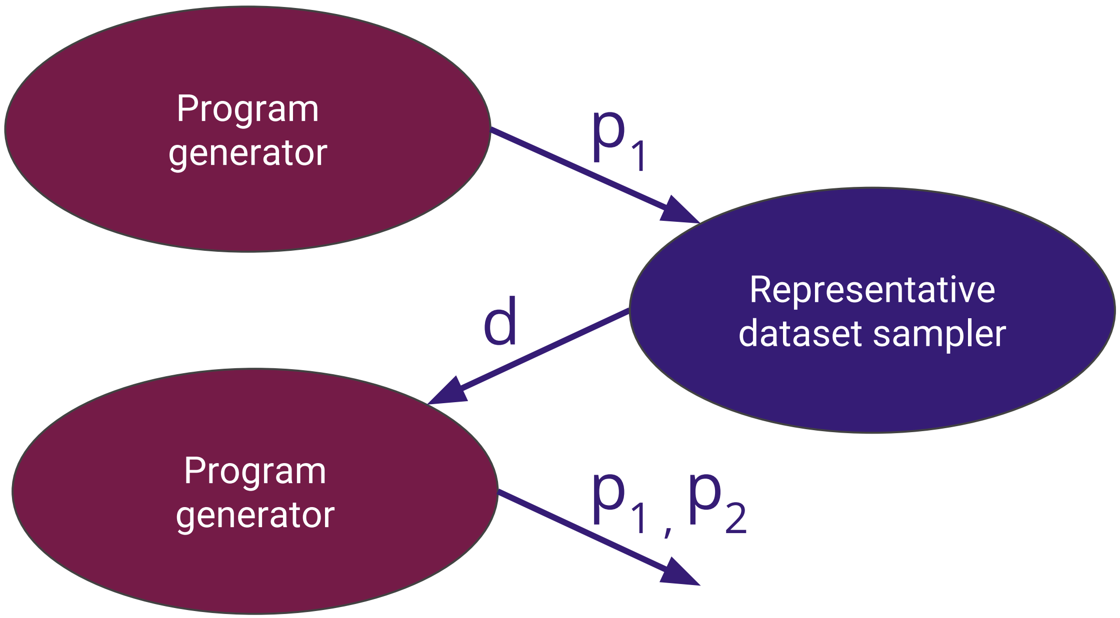


In a loop

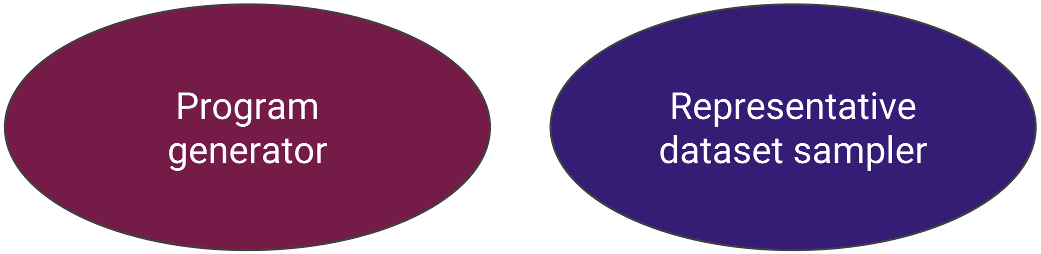


Start with a small random sample d⊆D

Iteratively generate programs and samples.

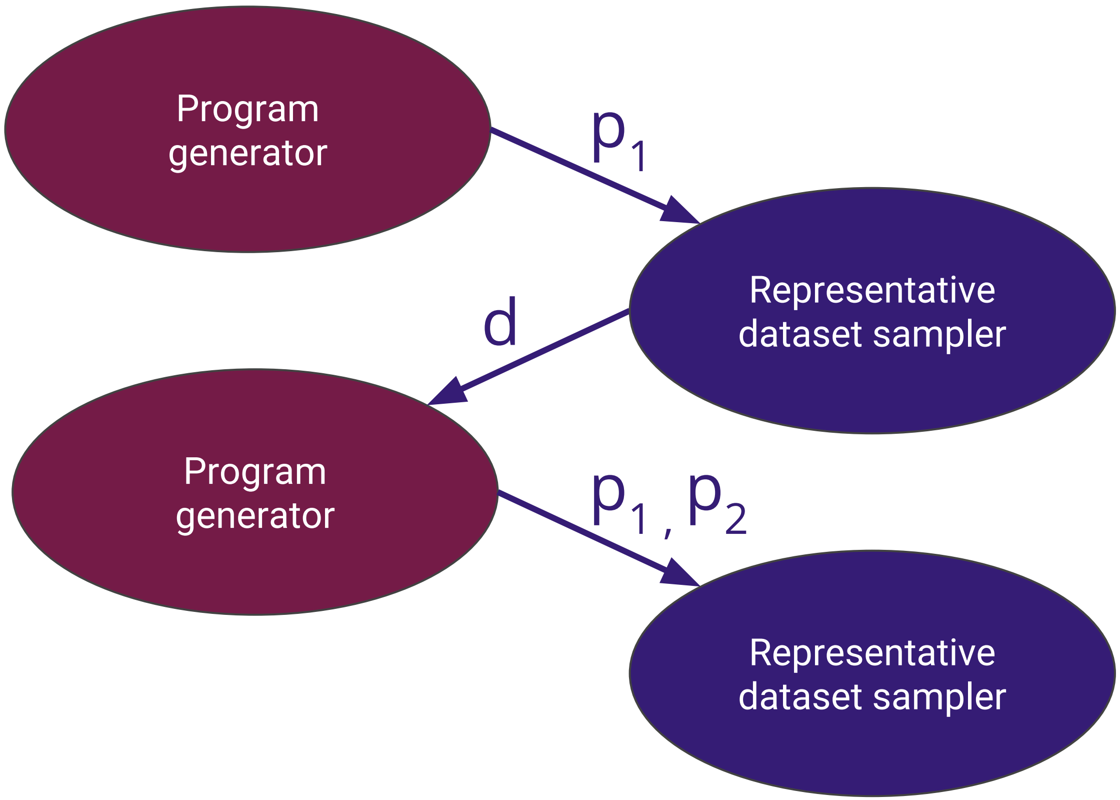


In a loop

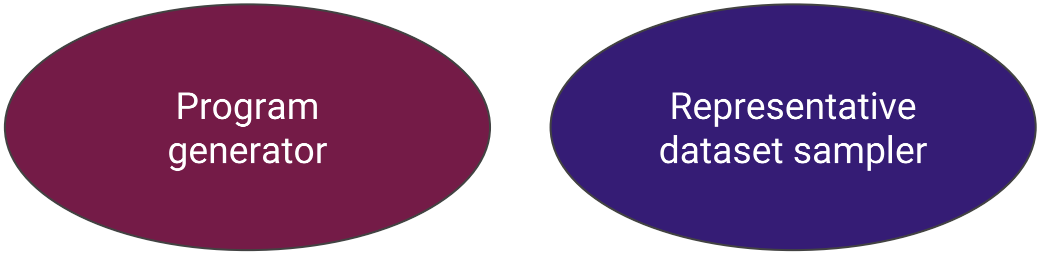


Start with a small random sample d⊆D

Iteratively generate programs and samples.

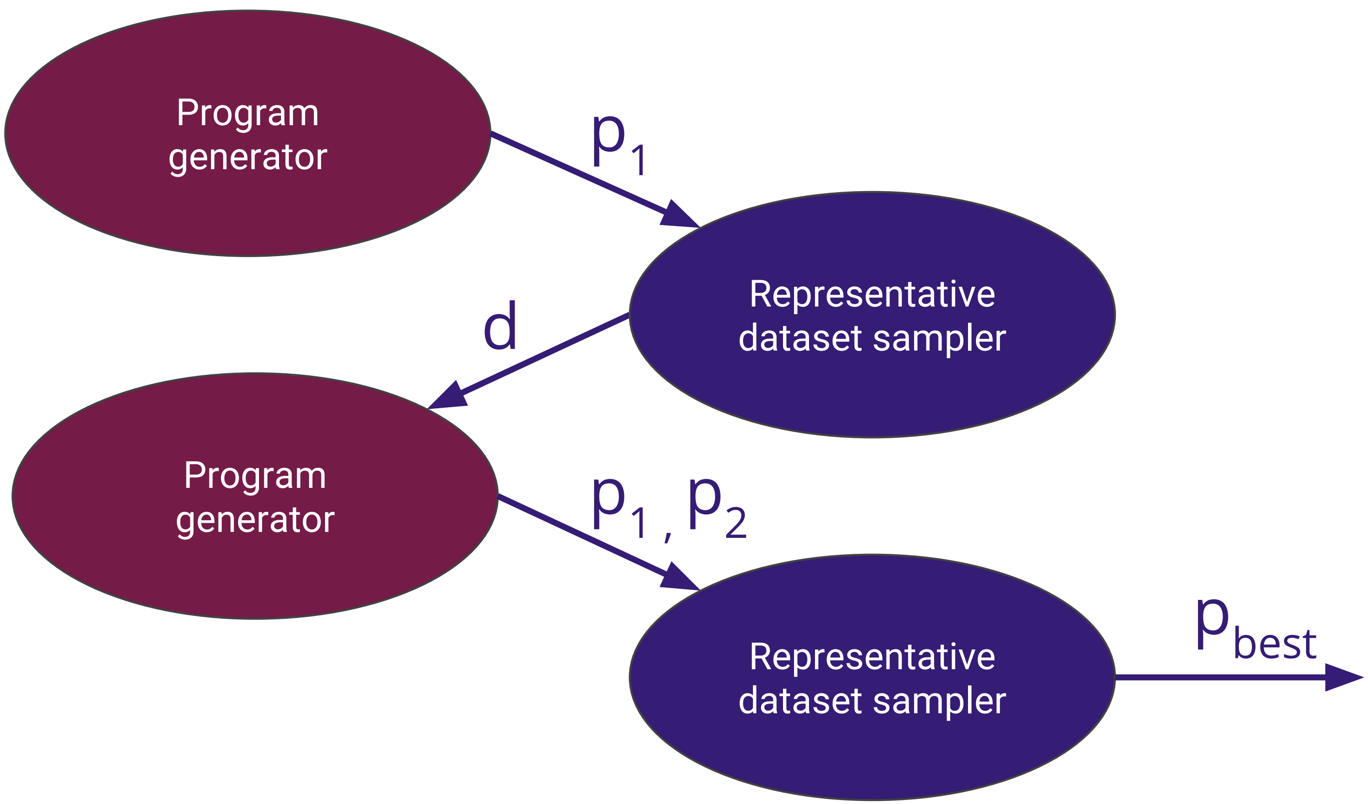


In a loop

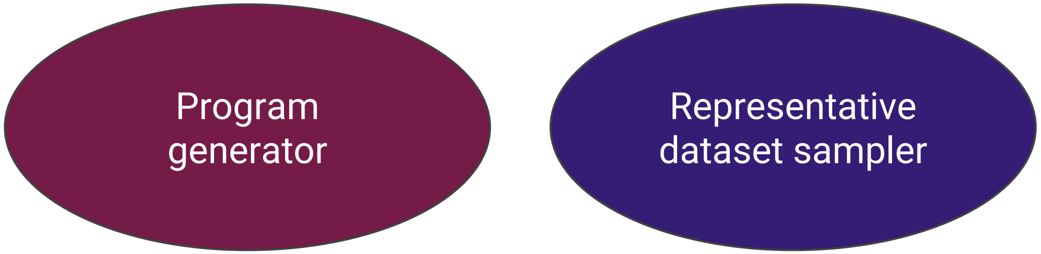


Start with a small random sample d⊆D

Iteratively generate programs and samples.

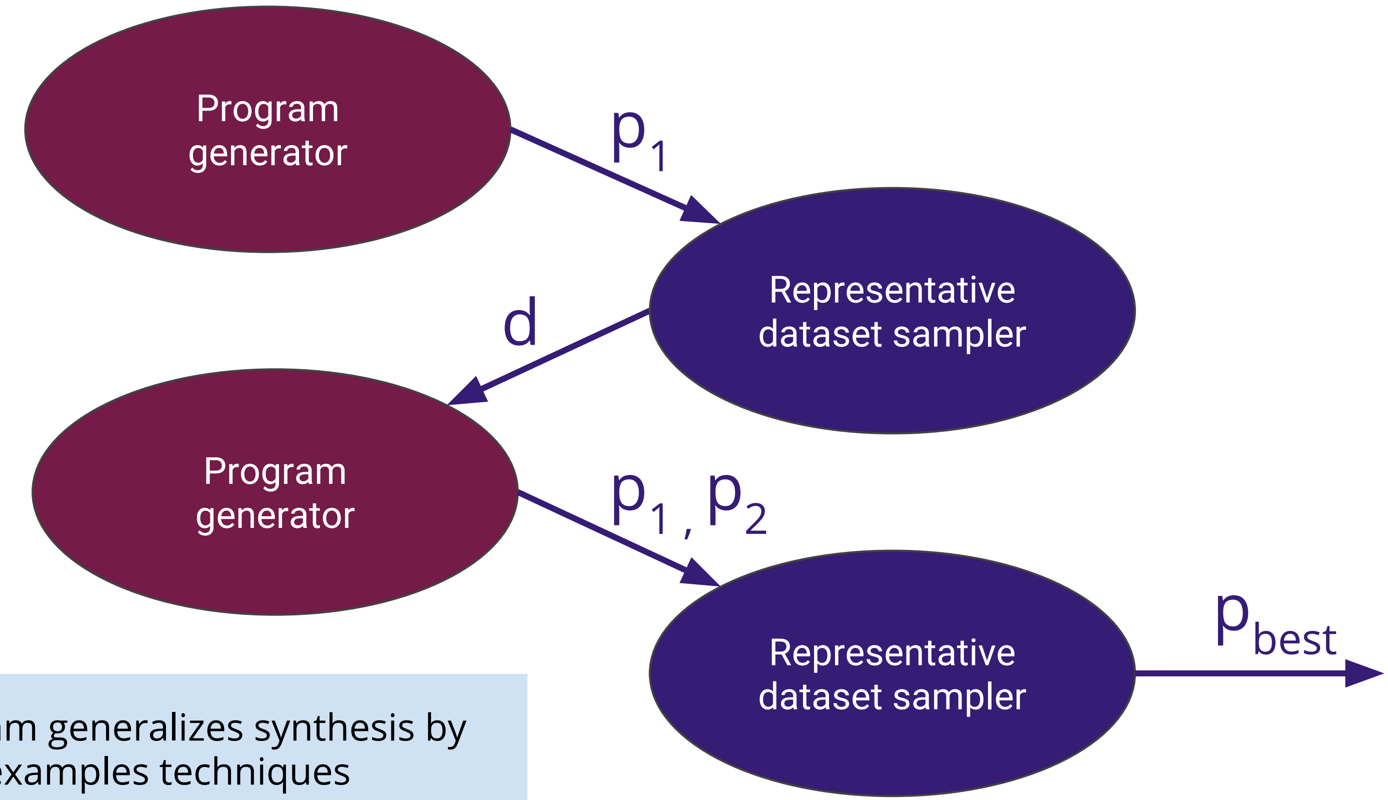


In a loop



Start with a small random sample d⊆D

Iteratively generate programs and samples.



Algorithm generalizes synthesis by examples techniques

Representative dataset sampler

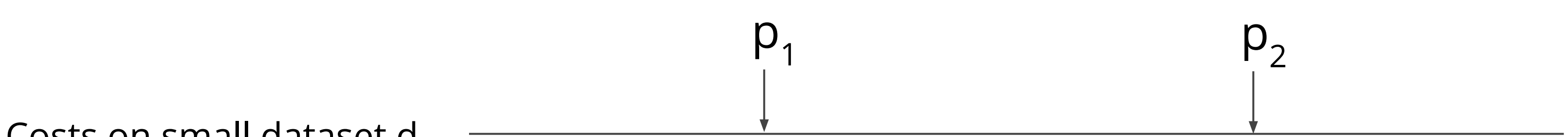
Idea: pick a small dataset d for which a set of already generated programs $p_1,\ldots,p_n$ behave like on the full dataset

$$d = \arg\min_{d \subseteq D} \max_{i \in 1..n} | \mathrm{cost}(d, p_i) - \mathrm{cost}(D, p_i) |$$

Representative dataset sampler

Idea: pick a small dataset d for which a set of already generated programs $p_1,\ldots,p_n$ behave like on the full dataset

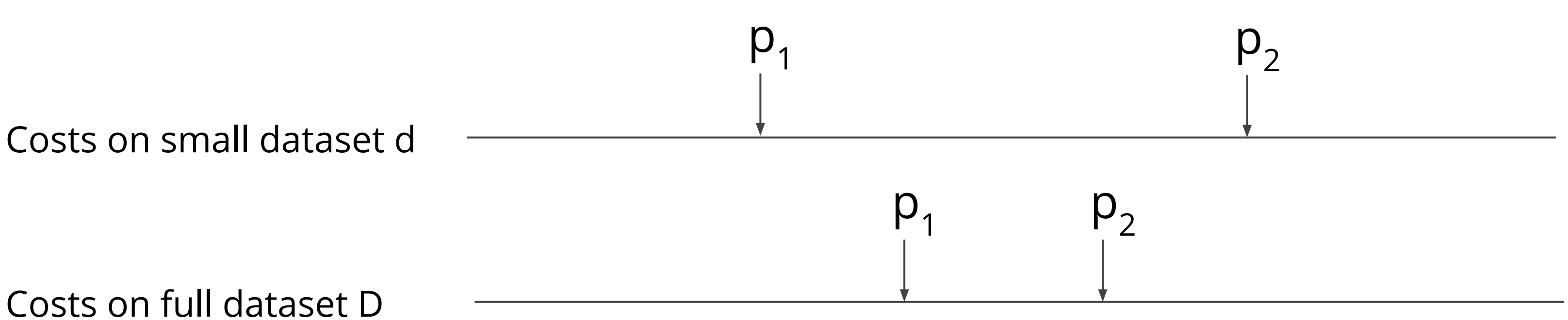
$$d = \arg\min_{d \subseteq D} \max_{i \in 1..n} | \mathrm{cost}(d, p_i) - \mathrm{cost}(D, p_i) |$$



Representative dataset sampler

Idea: pick a small dataset d for which a set of already generated programs $p_1,\ldots,p_n$ behave like on the full dataset

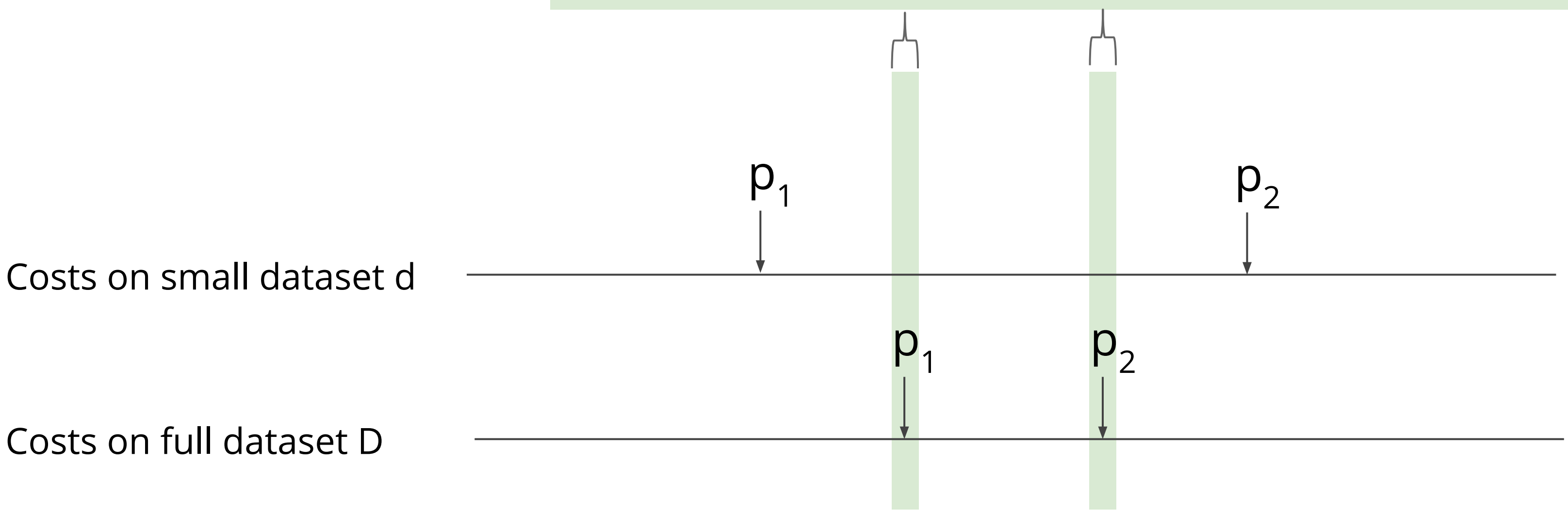
$$d = \arg\min_{d \subseteq D} \max_{i \in 1..n} | cost(d, p_i) - cost(D, p_i) |$$



Representative dataset sampler

Idea: pick a small dataset d for which a set of already generated programs $p_1,\ldots,p_n$ behave like on the full dataset

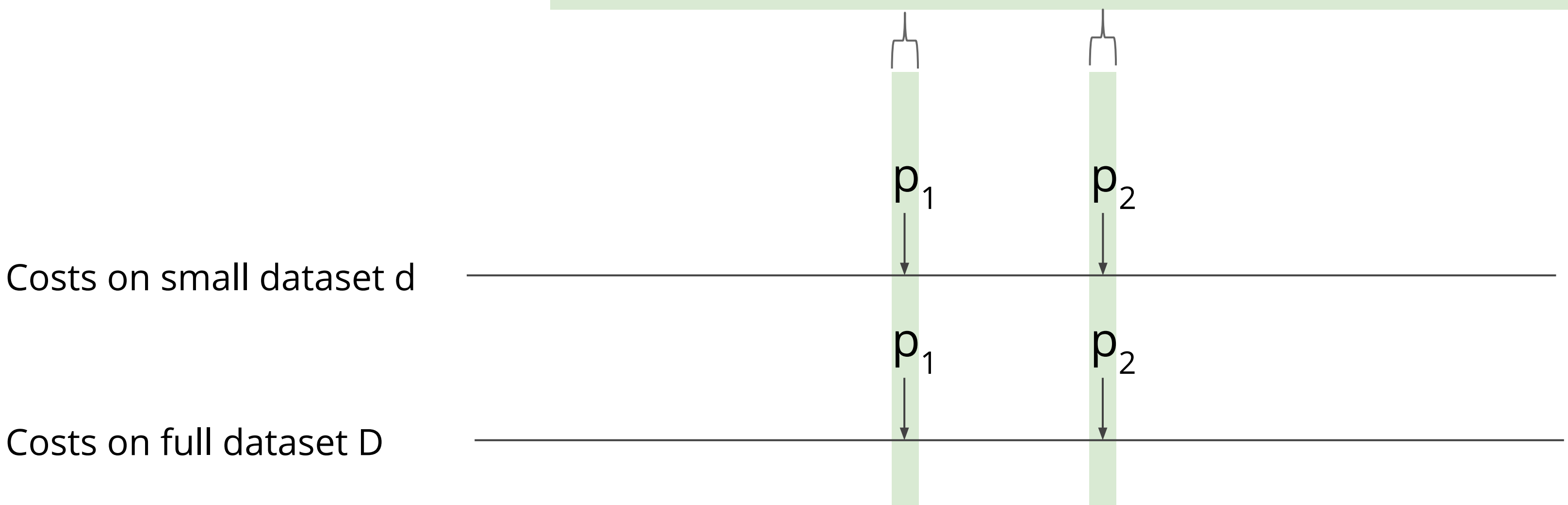
$$d = \arg\min_{d \subseteq D} \max_{i \in 1..n} \left| \mathrm{cost}(d, p_i) - \mathrm{cost}(D, p_i) \right|$$



Representative dataset sampler

Idea: pick a small dataset d for which a set of already generated programs $p_1,\ldots,p_n$ behave like on the full dataset

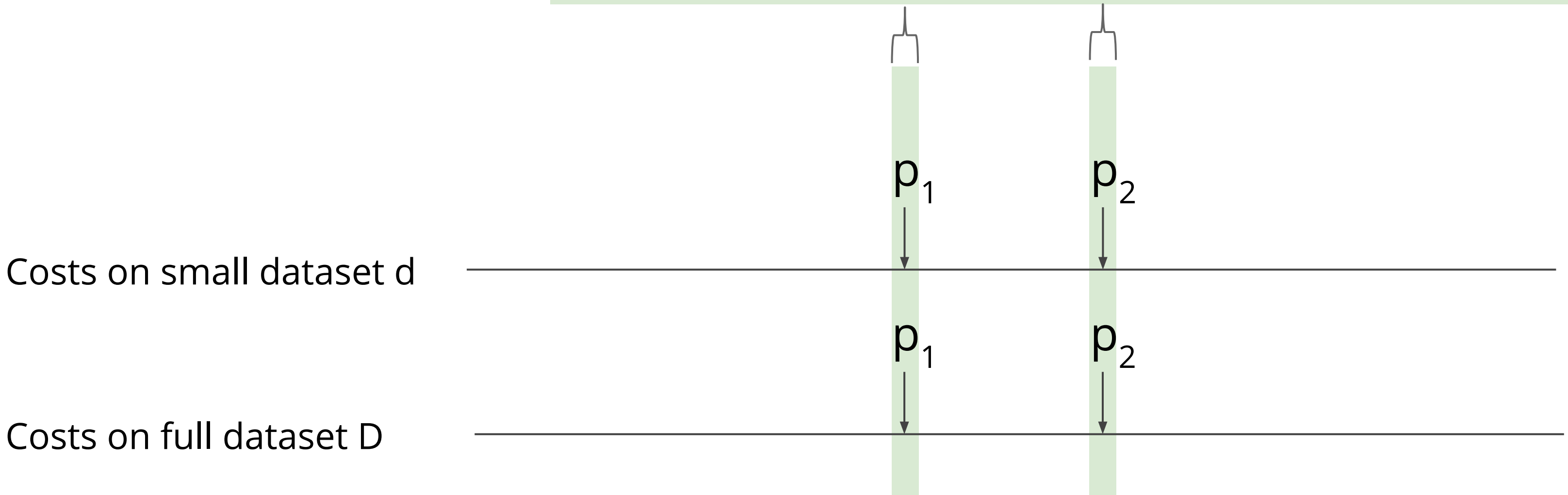
$$d = \arg\min_{d \subseteq D} \max_{i \in 1..n} \left| \mathrm{cost}(d, p_i) - \mathrm{cost}(D, p_i) \right|$$



Representative dataset sampler

Idea: pick a small dataset d for which a set of already generated programs $p_1,\ldots,p_n$ behave like on the full dataset

$$d = \arg\min_{d \subseteq D} \ \max_{i \in 1..n} \ | \ \mathrm{cost}(d, p_i) - \mathrm{cost}(D, p_i) \ |$$



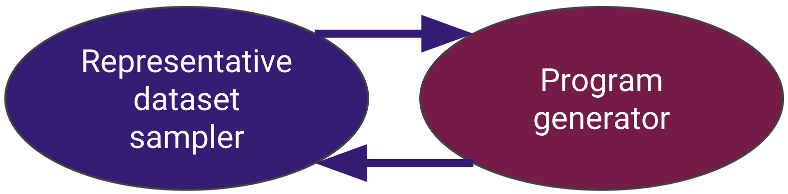
Theorem: this sampler shrinks the candidate program search space
In evaluation: significant speedup of synthesis

So far... handling large datasets

- Iterative combination of synthesis and sampling

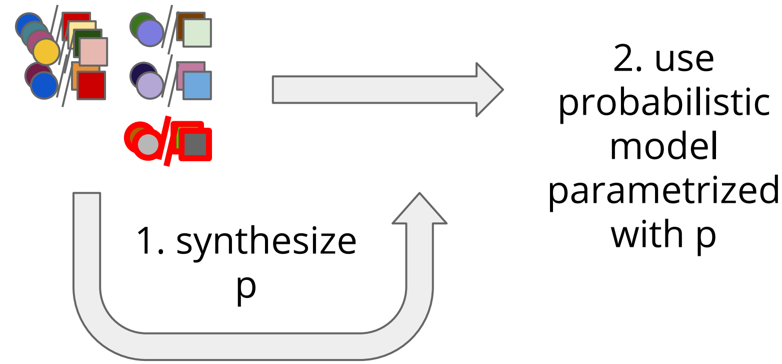
- New way to perform approximate empirical risk minimization

- Guarantees (in the paper)



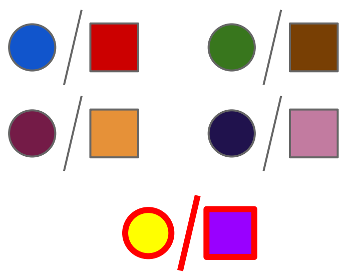
Contributions

New probabilistic models



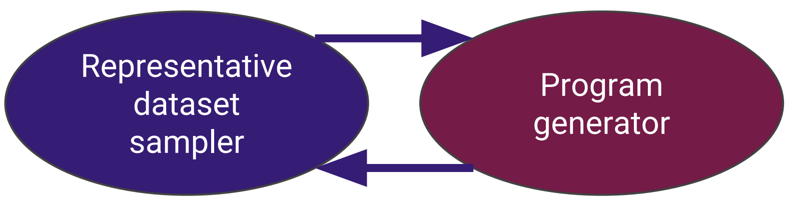
Handling noise

Input/output examples



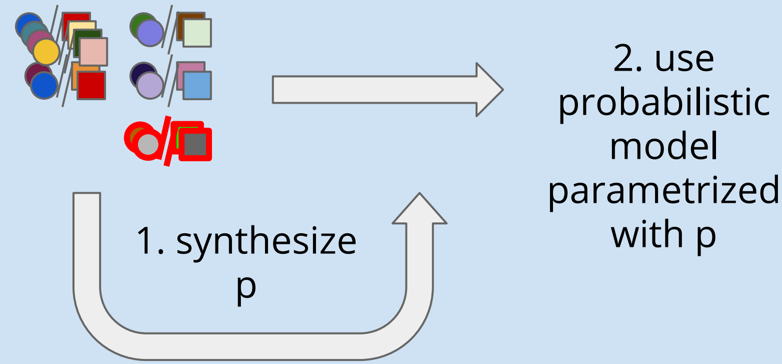
incorrect examples

Handling large datasets



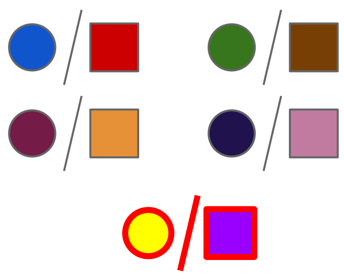
Contributions

New probabilistic models



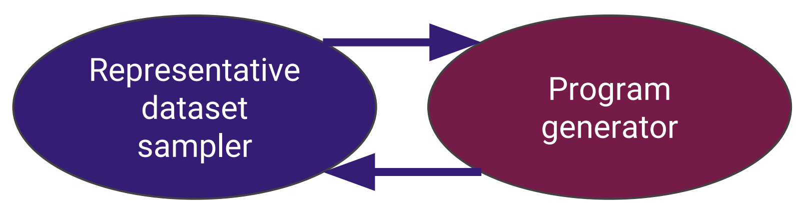
Handling noise

Input/output examples



incorrect examples

Handling large datasets



Statistical programming tools

A new breed of tools:

Learn from large existing codebases (e.g. Big Code) to make predictions about programs

Statistical programming tools

A new breed of tools:

Learn from large existing codebases (e.g. Big Code) to make predictions about programs

1. Train machine learning model

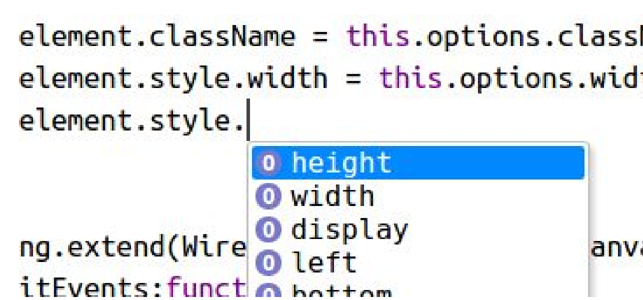
Statistical programming tools

A new breed of tools:

Learn from large existing codebases (e.g. Big Code) to make predictions about programs

1. Train machine learning model

2. Make predictions with model



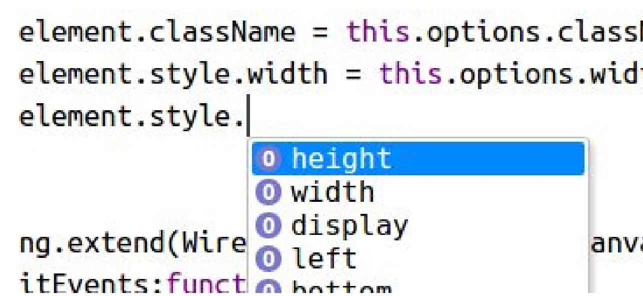
Statistical programming tools

A new breed of tools:

Learn from large existing codebases (e.g. Big Code) to make predictions about programs

1. Train machine learning model

2. Make predictions with model



hard-coded model
low precision

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE’12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

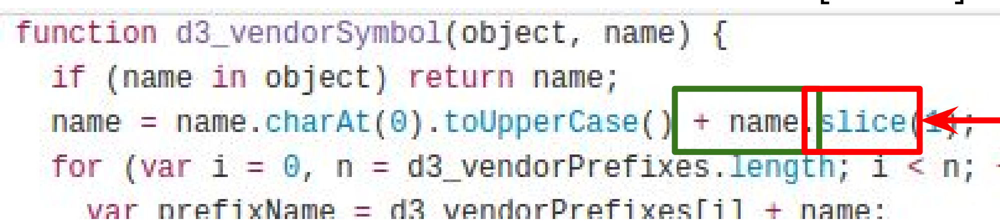
Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]



```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n;
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Raychev et al.[PLDI'14]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n;
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Raychev et al.[PLDI'14]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n;
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Raychev et al.[PLDI'14]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Raychev et al.[PLDI'14]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Existing machine learning models

Essentially remember mapping from context in training data to prediction (with probabilities)

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Raychev et al.[PLDI'14]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`charAt` → `slice`

Model will predict `slice` when it sees it after "`charAt`"
Relies on static analysis

Problem of existing systems

Precision. They rarely predict the next statement

Hindle et al.[ICSE'12]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n;
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after "`+ name .`"

This model comes from NLP

Raychev et al.[PLDI'14]

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.slice(1);
  for (var i = 0, n = d3_vendorPrefixes.length; i < n;
    var prefixName = d3_vendorPrefixes[i] + name;
```

Learn a mapping

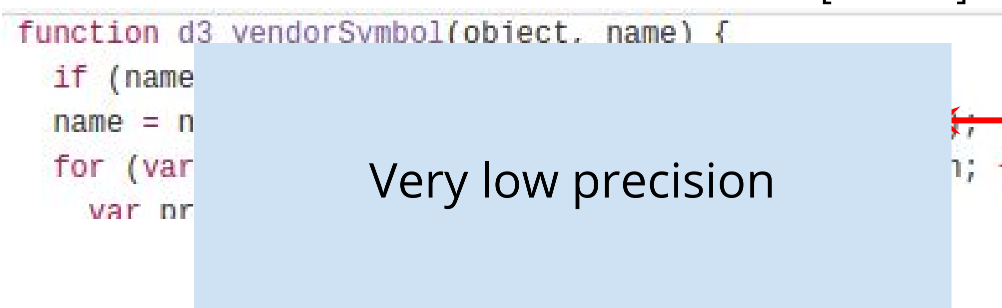
`charAt` → `slice`

Model will predict `slice` when it sees it after "`charAt`"
Relies on static analysis

Problem of existing systems

Precision. They rarely predict the next statement

Hindle et al.[ICSE’12]



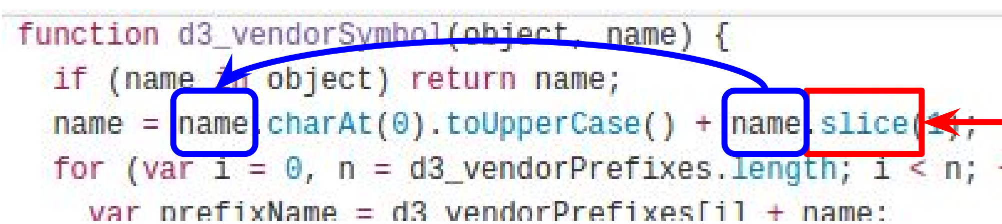
Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after “`+ name .`”

This model comes from NLP

Raychev et al.[PLDI’14]



Learn a mapping

`charAt` → `slice`

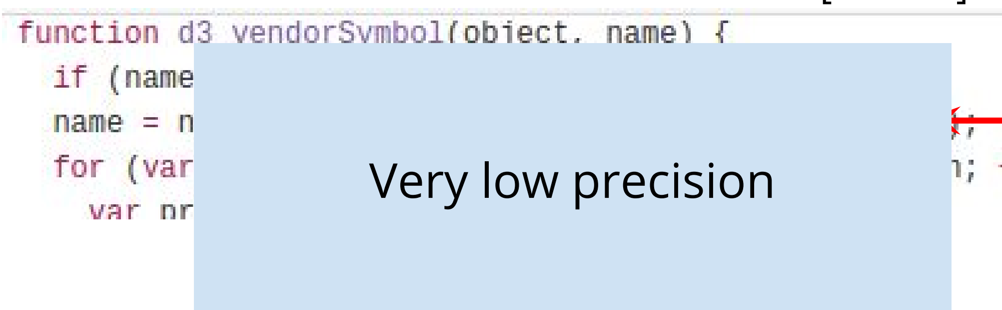
Model will predict `slice` when it sees it after “`charAt`”

Relies on static analysis

Problem of existing systems

Precision. They rarely predict the next statement

Hindle et al.[ICSE’12]



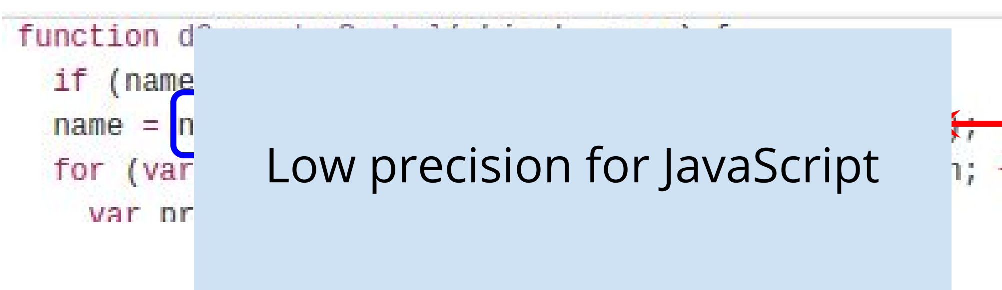
Learn a mapping

`+ name .` → `slice`

Model will predict `slice` when it sees it after “`+ name .`”

This model comes from NLP

Raychev et al.[PLDI’14]



Learn a mapping

`charAt` → `slice`

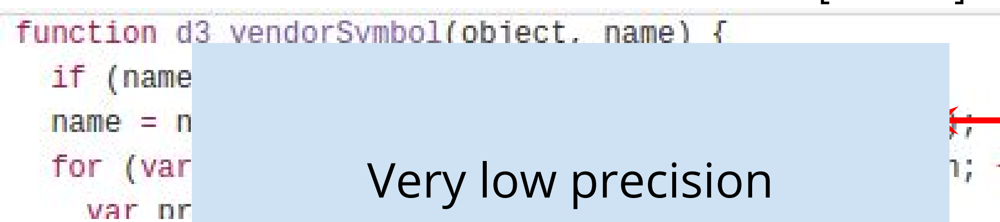
Model will predict `slice` when it sees it after “`charAt`”

Relies on static analysis

Problem of existing systems

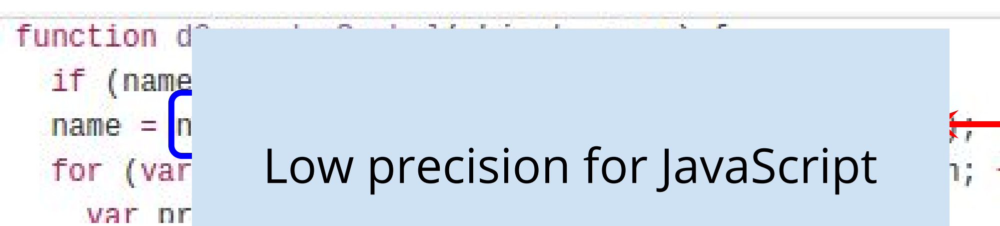
Precision. They rarely predict the next statement

Hindle et al.[ICSE'12]



Very low precision

Raychev et al.[PLDI'14]



Low precision for JavaScript

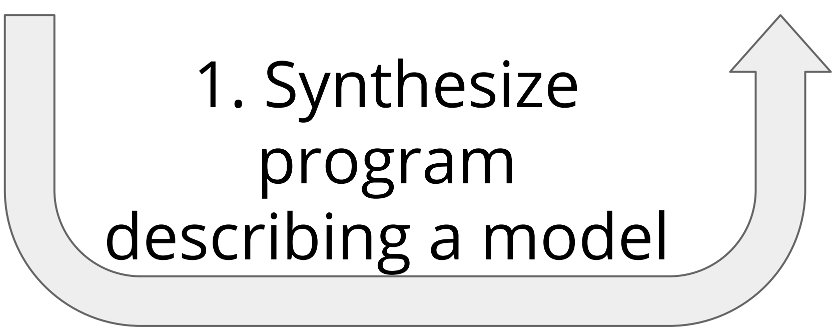
Core problem:
Existing machine learning models are limited and not expressive enough

Key idea: second-order learning

Learn a program that parametrizes a probabilistic model that makes predictions.

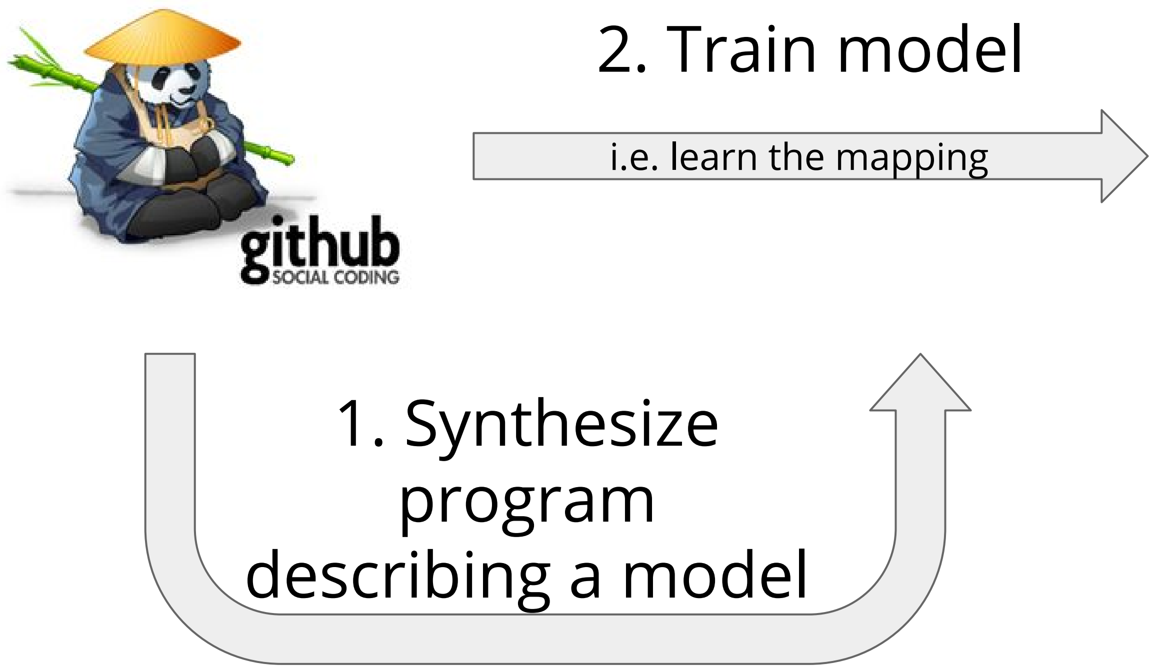
Key idea: second-order learning

Learn a program that parametrizes a probabilistic model
that makes predictions.



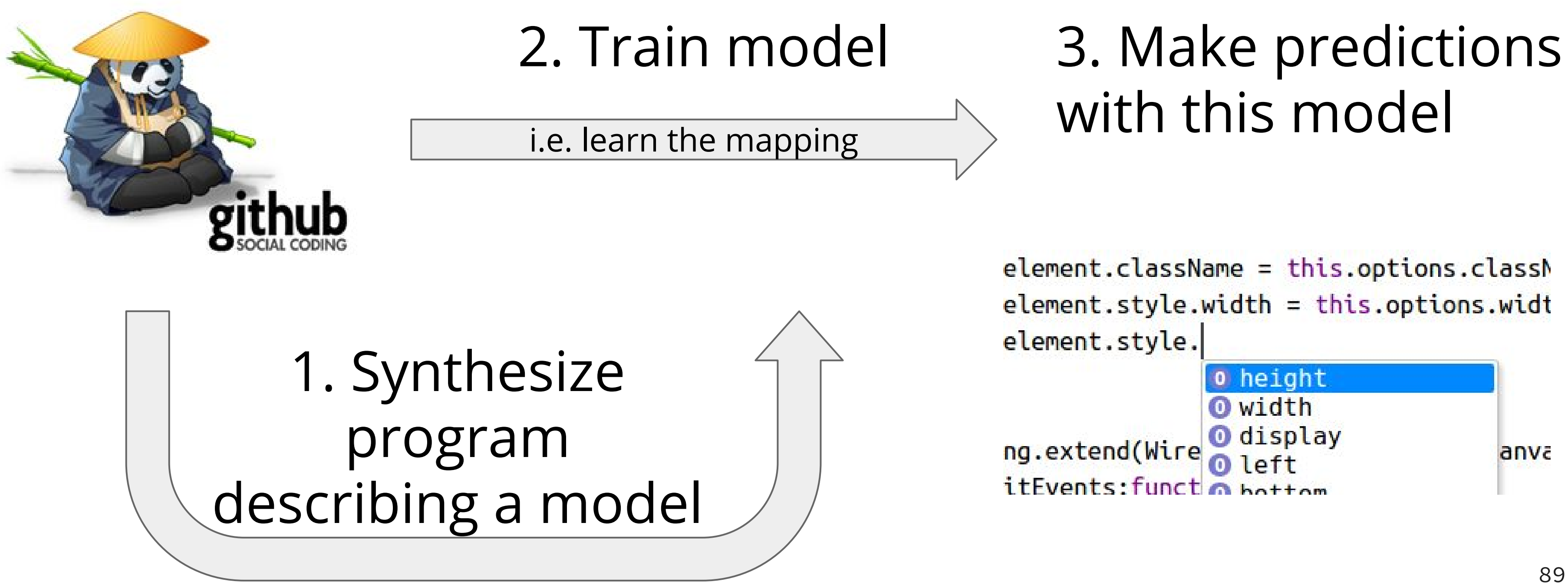
Key idea: second-order learning

Learn a program that parametrizes a probabilistic model that makes predictions.



Key idea: second-order learning

Learn a program that parametrizes a probabilistic model that makes predictions.



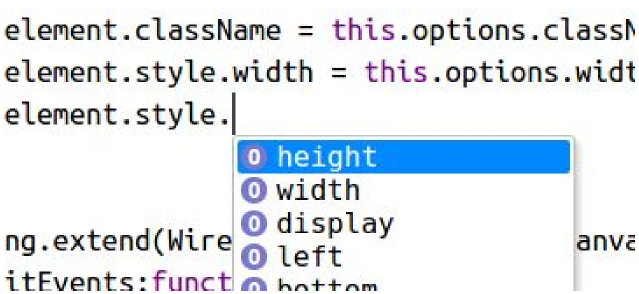
Key idea: second-order learning

Learn a program that parametrizes a probabilistic model that makes predictions.

2. Train model

i.e. learn the mapping

3. Make predictions with this model



prior models are described by simple hard-coded programs

Our approach:
learn a better program

Training and evaluation

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

`slice`

output

Training and evaluation

Compute context with program p

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

`slice`

output

Training and evaluation

Compute context
with program p

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

`slice`

Learn a mapping

`toUpperCase` → `slice`

Training and evaluation

Compute context with program p

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

slice

Learn a mapping

toUpperCase → slice

Evaluation example:

```
/)
cc, word) => {
acc + ' ' + word[0].toUpperCase() +  word.[redacted]
```

Training and evaluation

Compute context with program p

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

slice

Learn a mapping

toUpperCase → slice

Evaluation example:

Compute context with program p

```
/)
cc, word) => {
acc + ' ' + word[0].toUpperCase() +  word.[redacted]
```

Training and evaluation

Compute context
with program p

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; +
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

slice

Learn a mapping

toUpperCase → slice

Evaluation example:

Compute context
with program p

```
/)
cc, word) => {
acc + ' ' + word[0].toUpperCase() +  word.[redacted]
```

predict completion

slice

Training and evaluation

Compute context with program p

Training example:

```
function d3_vendorSymbol(object, name) {
  if (name in object) return name;
  name = name.charAt(0).toUpperCase() + name.[redacted]
  for (var i = 0, n = d3_vendorPrefixes.length; i < n; -
    var prefixName = d3_vendorPrefixes[i] + name;
```

input

`slice`

Learn a mapping

`toUpperCase` → `slice`

Evaluation example:

Compute context with program p

```
/)
cc, word) => {
acc + ' ' + word[0].toUpperCase() +  word.[redacted]
```

predict completion

`slice` ✔

Observation

Synthesis of probabilistic model can be done with the same optimization problem as before!

evaluation data:
input/output
examples

Our problem formulation:

regularization constant

$$p_{best} = \arg\min_{p \in P} \text{errors}(D, p) + \lambda r(p)$$

regularizer
penalizes long
programs

Observation

Synthesis of probabilistic model can be done with the same optimization problem as before!

evaluation data: input/output examples

cost(D, p)

Our problem formulation:

regularization constant

$$p_{best} = \arg\min_{p \in P} \boxed{\texttt{errors}(D, p) + \lambda r(p)}$$

regularizer penalizes long programs

So far...

Handling noise

Synthesizing a model

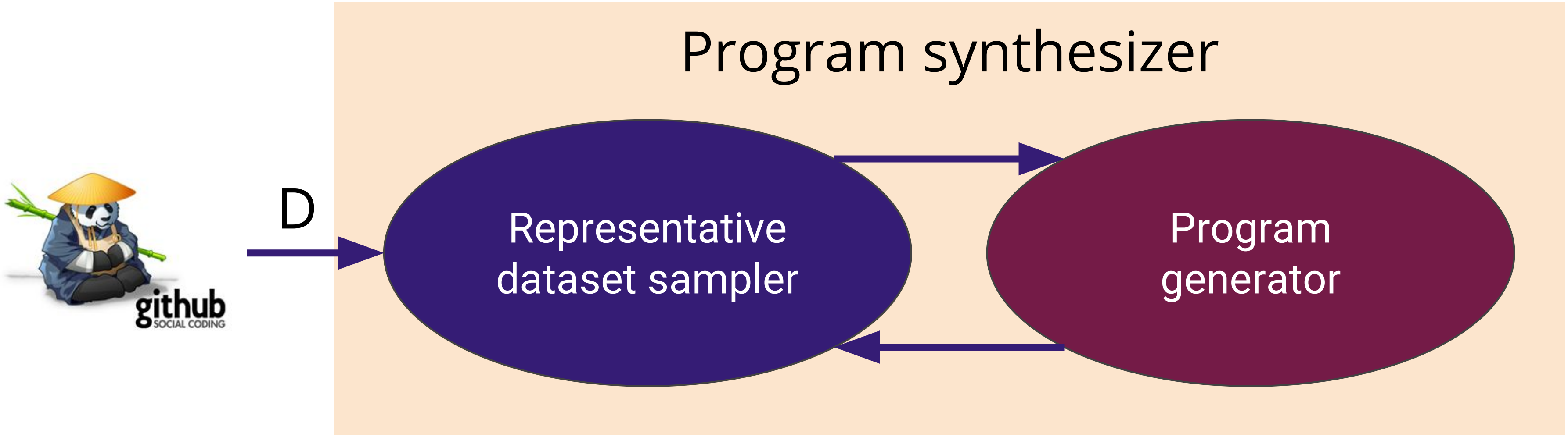
Representative dataset sampler

Techniques are generally applicable to program synthesis

Next, application for “Big Code” called DeepSyn

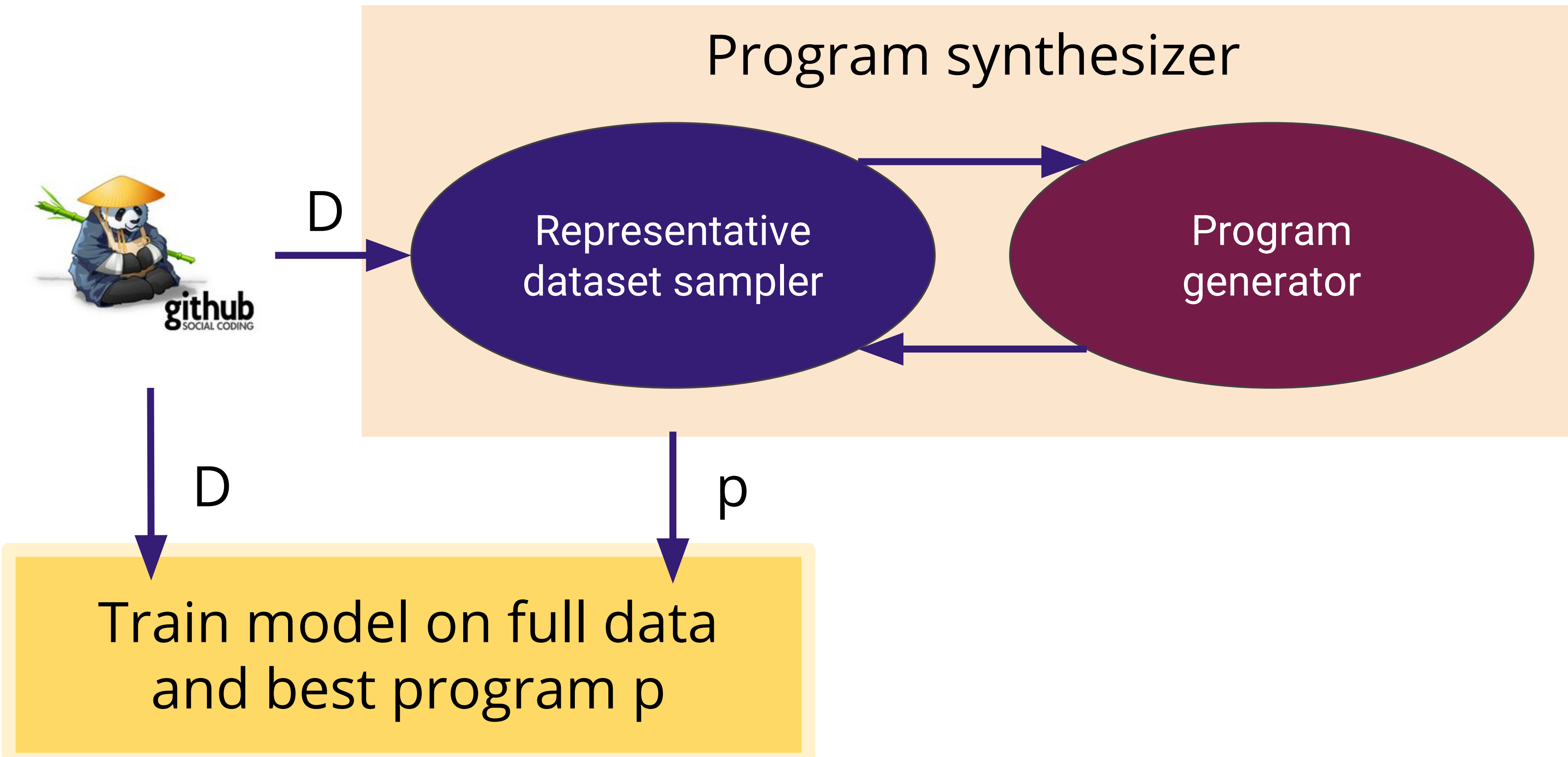
DeepSyn: Training

Trained on 100’000 JavaScript files from GitHub

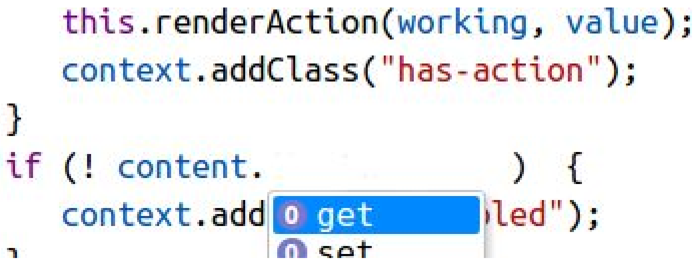


DeepSyn: Training

Trained on 100’000 JavaScript files from GitHub



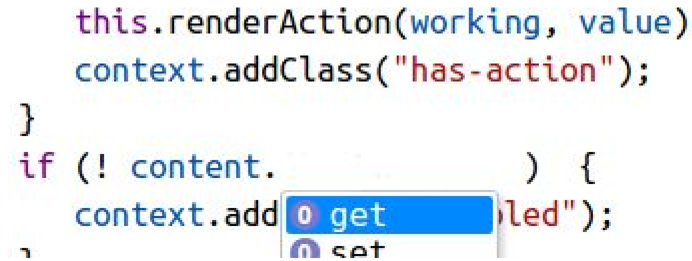
DeepSyn: Evaluation



50’000 evaluation files (not used in training or synthesis)

API completion task

DeepSyn: Evaluation

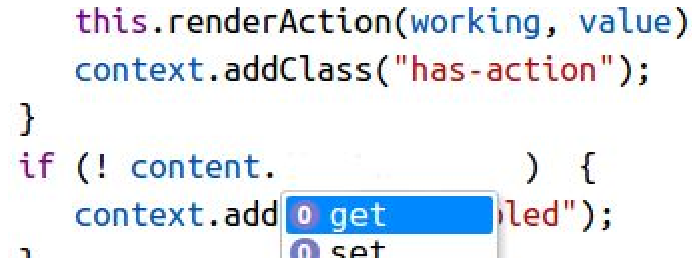


50’000 evaluation files (not used in training or synthesis)

API completion task

	Conditioning program p	Accuracy
	Last two tokens, Hindle et al.[ICSE’12]	22.2%
	Last two APIs per object, Raychev et al.[PLDI’14]	30.4%
This work	**Program synthesis with noise**	**46.3%**
	Program synthesis with noise + dataset sampler	**50.4%**

DeepSyn: Evaluation



50’000 evaluation files (not used in training or synthesis)

API completion task

Conditioning program p	Accuracy
Last two tokens, Hindle et al.[ICSE’12]	22.2%
Last two APIs per object, Raychev et al.[PLDI’14]	30.4%
Program synthesis with noise	**46.3%**
Program synthesis with noise + dataset sampler	**50.4%**

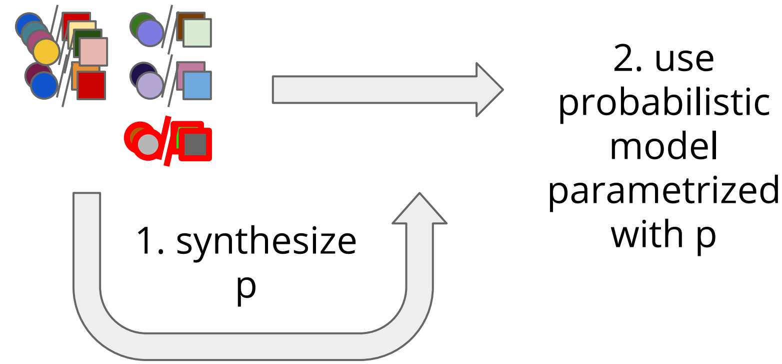
This work

We can explain best program. It looks at API preceding completion position and at tokens prior to these APIs.

Q&A

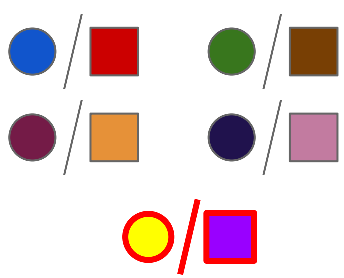
Synthesis of probabilistic models

Second-order learning



Handling noise

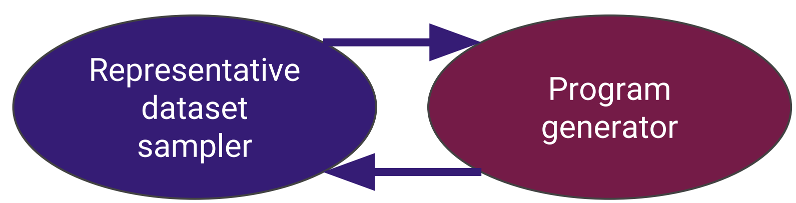
Input/output examples



incorrect examples

Extending synthesizers to handle noise

Handling large datasets

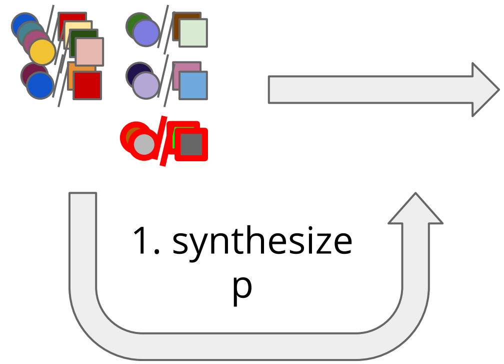


Scalability

Q&A

Synthesis of probabilistic models

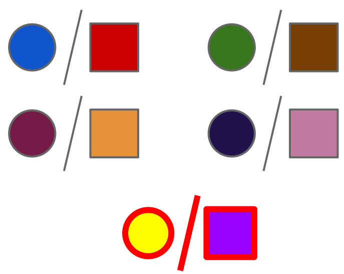
Second-order learning



2. use probabilistic model parametrized with p

Handling noise

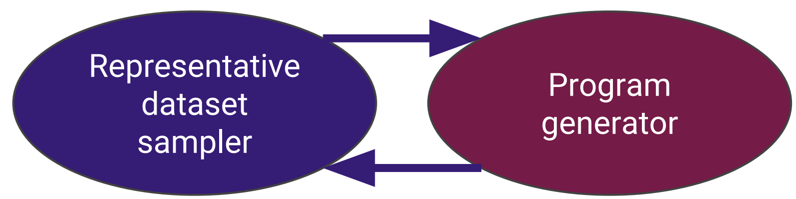
Input/output examples



incorrect examples

Bridges gap between ML and PL
Advances both areas

Handling large datasets



Extending synthesizers to handle noise

Scalability

What did we synthesize?

Left PrevActor WriteAction WriteValue PrevActor WriteAction PrevLeaf WriteValue PrevLeaf WriteValue

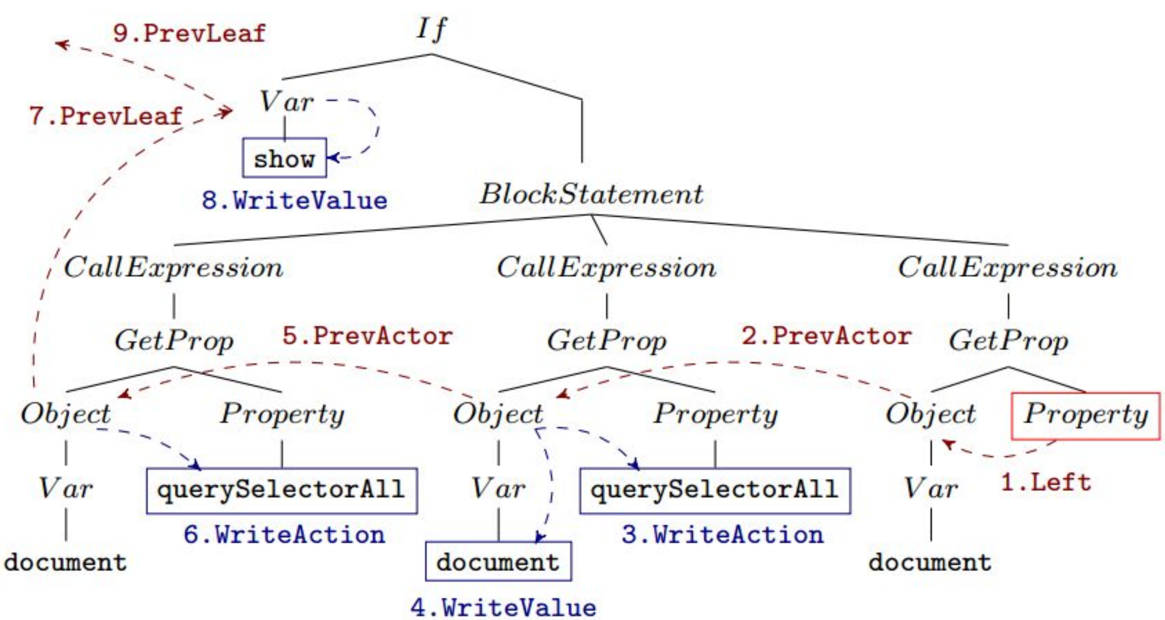
$p_{\approx best}$ =

```
Left PrevActor WriteAction WriteValue
PrevActor WriteAction PrevLeaf
WriteValue PrevLeaf WriteValue
```

(a) TCOND program

```
if (show) {
 var cws = document.querySelectorAll(...);
 for (var i = 0, slide; slide = cws[i]; i++) {
   slide.classList.add("hidden");
 }
 var iap = document.querySelectorAll(...);
 for (var i = 0, slide; slide = iap[i]; i++) {
   slide.classList.add("hidden");
 }
 var dart = document.
 ...                   └──── Completion position
}
```

(b) JavaScript code snippet



(c) Execution of $p_{\approx best}$ on the AST representation of the code snippet from (b)