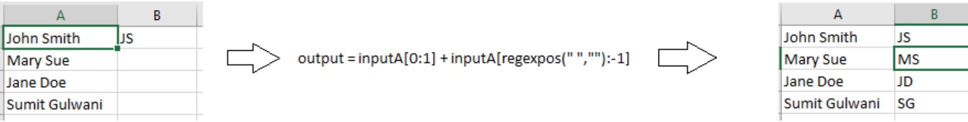
Learning to Learn Programs: Going Beyond Program Structure

Kevin Ellis & Sumit Gulwani

IJCAI 2017

The problem of programming by example

- Computers are everywhere, yet almost no one can code!
- Let *everyone* write programs by giving examples, not code



The right program is ambiguous

Input	Output
"Missing page numbers, 1993"	"1993"
"64-67, 1995"	

First number from the end
First number from the beginning
Just the constant string "1993"
...

Just the tip of the iceberg... could have on the order of 10^{100} consistent programs!

Our contribution: picking the right program

- Pick the smallest program?
  - Classic old idea: [Solomonoff 1964]
- Pick the program that "looks the most correct?"
  - Features of the program structure: [Singh et al, 2015], [Tamuz et al, 2013], [Dechter et al, 2013], [Liang et al 2010]

Our contribution: picking the right program

- Pick the smallest program?
  - Classic old idea: [Solomonoff 1964]
- Pick the program that "looks the most correct?"
  - Features of the program structure: [Singh et al, 2015], [Tamuz et al, 2013], [Dechter et al, 2013], [Liang et al 2010]

Our approach:

Program structure

+ Program execution trace

+ Program outputs

Our contribution: picking the right program

- Pick the smallest program?
  - Classic old idea: [Solomonoff 1964]
- Pick the program that "looks the most correct?"
  - Features of the program structure: [Singh et al, 2015], [Tamuz et al, 2013], [Dechter et al, 2013], [Liang et al 2010]

Our approach:

Program structure

+ Program execution trace

+ Program outputs

A Question: What should an inductive bias over programs look like?

A Problem: How can we improve PBE systems?

What kinds of programs can we learn?

Input	**Output**
[CPT-00350	[CPT-00350]
[CPT-00340	*to be predicted*
[CPT-115]	*to be predicted*

Append right bracket
Make output be delimited with brackets
Just the constant string "[CPT-00350]"
...

FlashFill & FlashExtract style problems implemented using **PROSE**: PROgram Synthesis by Example [Polozov, Gulwani, 2015]

Features of program structure: Predicting a correct program based on its appearance

Motivating example: Some program structures are more natural than others

Input	Output
John Smith	Smith, John
Jane Goodall	
Sumit Gulwani	

Motivating example: Some program structures are more natural than others

Input	Output
John Smith	Smith, John
Jane Goodall	Goodall, Jane
Sumit Gulwani	Gulwani, Sumit

`SecondWord + ', ' + FirstWord`

Input	Output
John Smith	Smith, John
Jane Goodall	
Sumit Gulwani	

Motivating example: Some program structures are more natural than others

Input	Output
John Smith	Smith, John
Jane Goodall	
Sumit Gulwani	

Input	Output
John Smith	Smith, John
Jane Goodall	Goodall, Jane
Sumit Gulwani	Gulwani, Sumit

`SecondWord + ', ' + FirstWord`

Input	Output
John Smith	Smith, John
Jane Goodall	Smith, Jane
Sumit Gulwani	Smith, Sumit

`'Smith, ' + FirstWord`

Motivating example: Some program structures are more natural than others

Input	Output
John Smith	Smith, John
Jane Goodall	
Sumit Gulwani	

Input	Output
John Smith	Smith, John
Jane Goodall	Goodall, Jane
Sumit Gulwani	Gulwani, Sumit

`SecondWord + ', ' + FirstWord`

Input	Output
John Smith	Smith, John
Jane Goodall	Smith, Jane
Sumit Gulwani	Smith, Sumit

`'Smith, ' + FirstWord`

Fewer constants, better program!

Motivating example: Some program structures are more natural than others

Input	Output
John Smith	Smith, John
Jane Goodall	
Sumit Gulwani	

Input	Output
John Smith	Smith, John
Jane Goodall	Goodall, Jane
Sumit Gulwani	Gulwani, Sumit

`SecondWord + ', ' + FirstWord`

$$\text{program}^* = \operatorname*{arg\,max}_{\text{programs consistent with examples}} \text{score}(\text{program})$$

Input	Output
John Smith	Smith, John
Jane Goodall	Smith, Jane
Sumit Gulwani	Smith, Sumit

`'Smith, ' + FirstWord`

Features of program execution trace: Predicting a correct program based on how it computed its output

Predicting a correct program based on its execution trace

Motivating example:

Input	**Output (baseline)**	**Output (ours)**
Rebecca	Dr. Rebecca	Dr. Rebecca
Oliver	*Do. Oliver*	*Dr. Oliver*

Predicting a correct program based on its execution trace

Motivating example:

Input	**Output (baseline)**	**Output (ours)**
Rebecca	Dr. Rebecca	Dr. Rebecca
Oliver	*Do. Oliver*	*Dr. Oliver*

$\boxed{\text{Rebecca}} \longrightarrow$ Dr. Rebecca

Execution trace:

Overlapping extractions!

Features of program outputs: Predicting a correct program based on the data it produces

Predicting a correct program based on its outputs

Motivating example:

Input	**Output (baseline)**	**Output (ours)**
[CPT-00350	[CPT-00350]	[CPT-00350]
[CPT-00340	*[CPT-00340]*	*[CPT-00340]*
[CPT-11536	*[CPT-11536]*	*[CPT-11536]*
[CPT-115]	*[CPT-115]]*	*[CPT-115]*

Ask yourself: Which is more likely to be the output of a correct program?

- [CPT-00350] [CPT-11222] [CPT-115] [CPT-00350] [CPT-11222] **[CPT-115]]**
- [CPT-00350] [CPT-11222] [CPT-115] [CPT-00350] [CPT-11222] **[CPT-115]**

Describing the outputs of a program

Program outputs should be "smooth"; smoothness = good description, called a *descriptor*.

Descriptor	"[CPT-" · Digits · "]"	"[CPT-" · Digits · "]" ∨ "[CPT-" · Digits · "]]"	Name∨ Name · Digits
Outputs	[CPT-00350] [CPT-00340] [CPT-115]	[CPT-00350] [CPT-00340] [CPT-115]]	Mary John Sue0481

Using the descriptor to score a program

Does this look more like the output of the intended program or the output of an unintended program?

- User labeld outputs: y_1, ..., y_L
- Program proposes outputs y_{L+1}, ..., y_N
- $\mathbb{P}[y_1 \cdots y_N | \text{intended}]$ vs
  $\mathbb{P}[y_1 \cdots y_L | \text{intended}] \times \mathbb{P}[y_{L+1} \cdots y_N | \text{unintended}]$

Let D be the descriptor of all the outputs. Log odds ratio:

$$\approx \log \mathbb{P}[y_1 \cdots y_L | D] + \theta \cdot \phi(D)$$

where $\phi(D) =$ descriptor features. We learn θ.

Using the descriptor to score a program

Does this look more like the output of the intended program or the output of an unintended program?

- User labeld outputs: y_1, ..., y_L
- Program proposes outputs y_{L+1}, ..., y_N
- $\mathbb{P}[y_1 \cdots y_N | \text{intended}]$ vs $\mathbb{P}[y_1 \cdots y_L | \text{intended}] \times \mathbb{P}[y_{L+1} \cdots y_N | \text{unintended}]$

Let D be the descriptor of all the outputs. Log odds ratio:

$$\approx \log \mathbb{P}[y_1 \cdots y_L | D] + \theta \cdot \phi(D)$$

where $\phi(D) =$ descriptor features. We learn θ.
User provided outputs treated specially!

Descriptors use world knowledge

Program input	Program output
457 124th St S, Seattle, WA 98111	Seattle-WA
98743 Edwards Ave, Los Angeles, CA 78911	
One Microsoft Way, Redmond, WA 98052	
11 Main, Bizmark, ND 54891	

Which are more likely to be the output of the user intended program?

- Seattle-WA, **Los-CA**, Redmond-WA, Bizmark-ND
- Seattle-WA, **Angeles-CA**, Redmond-WA, Bizmark-ND
- Seattle-WA, **Los Angeles-CA**, Redmond-WA, Bizmark-ND

World knowledge is useful in practice

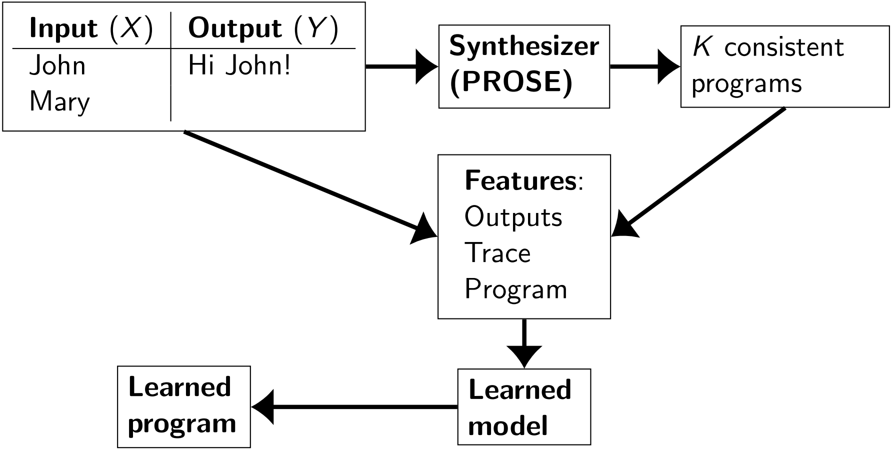
Most correct descriptors use common sense dictionaries!

Input	**Output (old system)**	**Output (ours)**
Brenda Everroad	Brenda	Brenda
Daymon Brodhacker	*Daymon*	*Daymon*
Dr. Catherine Ramsey	*Catherine*	*Catherine*
Judith K. Smith	*Judith K.*	*Judith*
Cheryl J. Adams and Binnie Phillips	*Cheryl J. Adams and Binnie*	*Cheryl*

Table: Learning a program from one example (top row) and applying it to other inputs (bottom rows, outputs italicized). Our semisupervised approach uses simple common sense reasoning, knowing about names, places, words, dates, etc, letting us get the last two rows correct.

Learning to use the features to pick a correct program

Pipeline



Input (X)
Output (Y)
John
Hi John!
Mary
Synthesizer (PROSE)
K consistent programs
Features:
Outputs
Trace
Program
Learned model
Learned program

A probabilistic model for picking a sequence of outputs

- E = input/output examples + some extra inputs
- p = a program
- $\phi(p, E)$ = feature vector from program structure, trace, & outputs
- θ = vector of weights for each feature

Log linear model:

$$\mathbb{P}[p|E, \theta] \propto \exp(\theta \cdot \phi(p, E))$$

A probabilistic model for picking a sequence of outputs

- E = input/output examples + some extra inputs
- p = a program
- $\phi(p, E)$ = feature vector from program structure, trace, & outputs
- θ = vector of weights for each feature

Log linear model:

$$\mathbb{P}[p|E,\theta] \propto \exp(\theta \cdot \phi(p, E))$$

Probability of outputting Y:

$$\mathbb{P}[Y|E,\theta] \propto \sum_{\substack{p: \\ p \text{ evaluates to } Y \\ Y \text{ consistent with } E}} \mathbb{P}[p|E,\theta]$$

Don't just take the outputs of the highest scoring program. Aggregate all scores of all programs that have a particular output!

Learning the weights on each of the features

Maximize expected number of problems where the correct outputs are predicted:

$$\arg\max_{\theta} \sum_{\text{problem}} \mathbb{P}[Y_{\text{problem}} | E_{\text{problem}}, \theta]$$

- **Smoothed** version of # problems we get correct
- Gradient-based search

Experimental results: string transformation & text extraction

Experimental results

String transformation: 447 problems

	Training	Test
Random	13.7%	13.7%
PROSE	76.4%	–
Trace	56.6%	$46.1 \pm 2\%$
Output	68.2%	$66.5 \pm 2\ \%$
Program	77.9%	$57.9 \pm 4\ \%$
All	88.4%	$\mathbf{83.5 \pm 3\%}$

91%/8%/1% from 1,2,3 examples

Text extraction: 488 problems

	Training	Test
Random	14.7%	14.7%
PROSE	65.8%	–
Output	70.5%	$\mathbf{68.2 \pm 1\%}$
Program	63.9%	$49.9 \pm 1\%$
All	79.3%	$\mathbf{69.2 \pm 2\%}$

100% from 1 example

Features of the output are less prone to overfitting

Contributions

Toward answering the question: **What should an inductive bias over programs look like?**

We suggest: go beyond syntactic structure and consider outputs and execution traces

Contributions

Toward answering the question: **What should an inductive bias over programs look like?**
We suggest: go beyond syntactic structure and consider outputs and execution traces

Improved real-world program learners on large data sets used in industry

Contributions

Toward answering the question: **What should an inductive bias over programs look like?**
We suggest: go beyond syntactic structure and consider outputs and execution traces

Improved real-world program learners on large data sets used in industry

This is a collaboration with the PROSE team at Microsoft:
Vu Le, Daniel Perelman, Alex Polozov, Danny Simmons, Abhishek Udupa, and Adam Smith

Semisupervised analogies

abc$\rightarrow$abd

- ijk$\rightarrow$ijl

(Melanie Mitchell and Douglas Hofstadter style analogy problems)

Semisupervised analogies

abc→abd

- ijk→ijl
- iijjkk→iijjll

(Melanie Mitchell and Douglas Hofstadter style analogy problems)

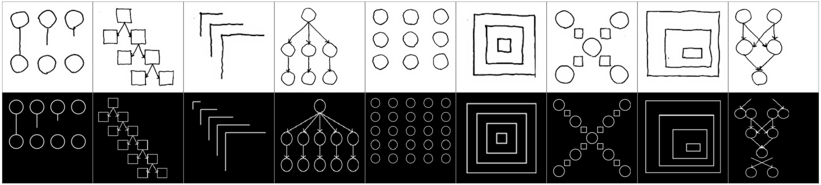
Semisupervised analogies

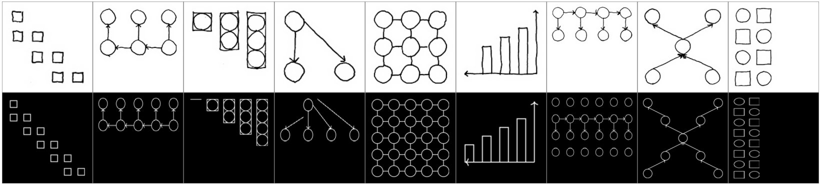
abc→abd

- ijk→ijl
- iijjkk→iijjll
- xyz→xya, or wyz, or ...

(Melanie Mitchell and Douglas Hofstadter style analogy problems)

Inferring and extrapolating graphics programs





See our paper on arxiv: “Learning to Infer Graphics Programs from Hand-Drawn Images”