

Machine Learning and Programming Languages

Martin Vechev

Software Reliability Lab

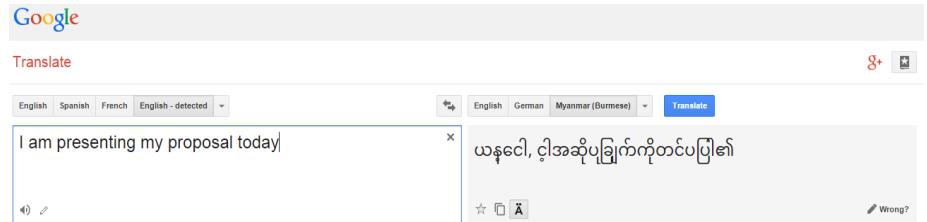
Department of Computer Science, ETH Zurich

@SRL: Two Directions

- Machine Learning based programming tools
- Probabilistic programming

Learning and probabilistic models based on Big Data have revolutionized entire fields

Natural Language Processing
(e.g., machine translation)

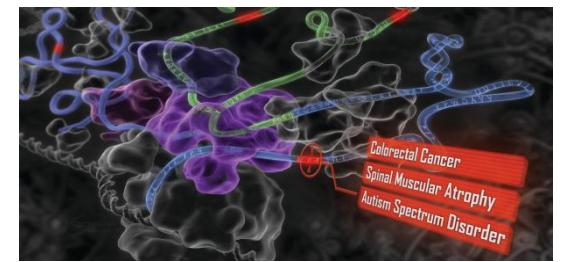
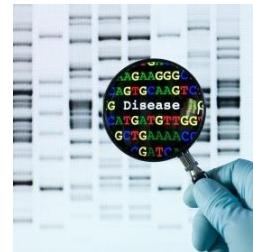


Computer Vision
(e.g., image captioning)



A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

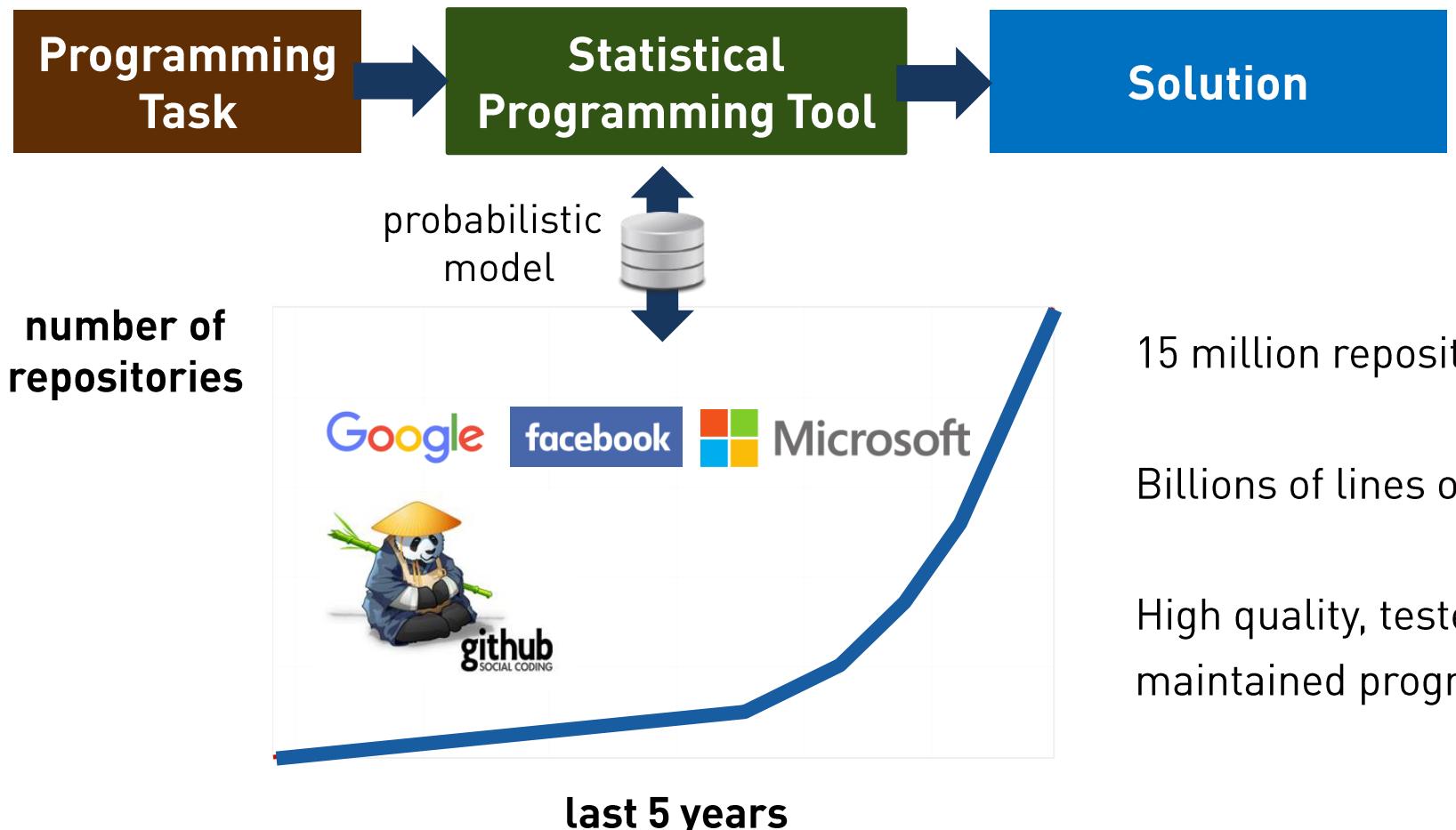
Medical Computing
(e.g., disease prediction)



Can we bring this revolution to programmers?

Vision

Learning from large datasets of programs



Probabilistic Learning

Probabilistically likely solutions to problems impossible to solve otherwise

Publications

- Statistical Deobfuscation of Android Applications, **ACM CCS'16**
- Probabilistic Mode for Code with Decision Trees, **ACM OOPSLA'16**
- PHOG: Probabilistic Mode for Code, **ACM ICML'16**
- Learning Programs from Noisy Data, **ACM POPL'16**
- Predicting Program Properties from “Big Code”, **ACM POPL'15**
- Code Completion with Statistical Language Models, **ACM PLDI'14**
- Machine Translation for Programming Languages, **ACM Onward'14**

Publicly Available Tools

<http://JSNice.org>

statistical de-obfuscation

<http://Nice2Predict.org>

structured prediction framework



Martin
Vechev



Andreas
Krause



Veselin
Raychev



Pavol
Bielik



Christine
Zeller



Svetoslav
Karaivanov



Pascal
Roos



Benjamin
Bischel



Timon
Gehr



Petar
Tsankov



Mateo
Panzacchi

More information: <http://www.srl.inf.ethz.ch/>

Statistical Code Completion

[Code Completion with Statistical Language Models, ACM PLDI 2014]

```
Camera camera = Camera.open();
camera.setDisplayOrientation(90);
```



```
Camera camera = Camera.open();
camera.setDisplayOrientation(90);

camera.unlock();
SurfaceHolder holder = getHolder();
holder.addCallback(this);
holder.setType(SurfaceHolder.STP);
MediaRecorder r = new MediaRecorder();
r.setCamera(camera);
r.set AudioSource(MediaRecorder.AS);
r.set VideoSource(MediaRecorder.VS);
r.set OutFormat(MediaRecorder.MPEG4);
```

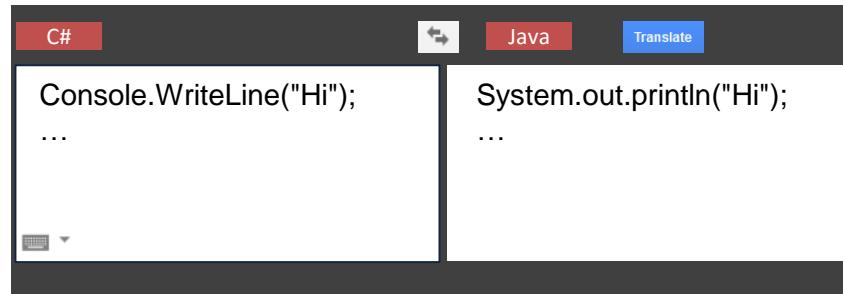
Statistical language models
Recurrent neural networks

+

Typestate analysis
Alias analysis

Programming Language Translation

[Phrase-based statistical translation of programming languages, ACM Onward 2014]



Phrase-based Statistical
Machine Translation

+

Prefix Parsing of
Context-Free Grammars

Program de-obfuscation

[Predicting program properties from Big Code, ACM POPL 2015]

```
function FZ(e, t) {  var n = [];
var r = e.length;  var i = 0;
for (; i < r; i += t)    if (i + t <
r)      n.push(e.substring(i, i +
t)); else
n.push(e.substring(i, r));
return n;
}
```



```
function chunkData(str, step) {
var colNames = [];
var len = str.length;
var i = 0;
for (; i < len; i += step)
if (i + step < len)
colNames.push(str.substring(i, i + step));
else
colNames.push(str.substring(i, len));
return colNames;
}
```

JSNice.org

[Predicting program properties from Big Code, ACM POPL 2015]

✓ Every country

✓ 170,000 users

✓ Top ranked tool



This Page Amsterdam @thispage_ams · Jul 16
Do you write ugly JavaScript code? Not to worry. JSNice will make it look like you are a **superstar** coder. Yai! - buff.ly/1HR4JL7

Ingvar Stepanyan @RReverser · Aug 6
JSNice.org became my **must-have** tool for code deobfuscation.
[Expand](#) [Reply](#) [Retweet](#) [Favorite](#) [More](#)

Brevity @seekbrevity · Jul 28
JSNice is an **amazing** tool for de-minifying **#javascript** files. JSNice.org, Its great for **#learning** and reverse engineering.
[Expand](#) [Reply](#) [Retweet](#) [Favorite](#) [More](#)

Alvaro Sanchez @alvasavi · Jun 10
This is gold.
Statistical renaming, Type inference and Deobfuscation.
jsnice.org
[Expand](#) [Reply](#) [Retweet](#) [Favorite](#) [More](#)

Alex Vanston @mrvdot · Jun 7
I've been looking for this for years: JS NICE buff.ly/1pQ5qfr **#javascript** **#unminify** **#deobfuscate** **#makeItReadable**
[Expand](#) [Reply](#) [Retweet](#) [Favorite](#) [More](#)

Kamil Tomšík @cztomsik · Jun 6
tell me **how this** works!
de-minify **#jquery** **#javascript** incl. args, vars & **#jsdoc**
impressive! jsnice.org

CodeGeekz

COMMENTS **MOST RECENT** **Fridi Komesz** on: **20 Online Code Editors and Tools for Developers**

[HOME](#) [WORDPRESS](#) [WEB DESIGN](#) [TYPOGRAPHY](#) [RESOURCES](#) [ARCHIVES](#) [CONTACT](#) [ADVERTISE](#)

20 Essential Tools for Coders

by **GAVIN** in **DEVELOPMENT - RESOURCES** — 28 JUL, 2014

20 Best Freebies for Web Designers of Year 2014

JavaScript Diagrams OCT 08, 2014 by VIKAS IN DESIGN



After spending a lot of time in a particular field, we normally get a lot of experience and we suddenly start doing things easily and in short span of time to maximize our efforts towards our goal. This same procedure also holds true in case of web designing also, Because it's all about getting an experience in a particular field or language. Learning web designing is not an easy task, but when you hold experience in the same field, It becomes much easier for you to get most of the work done in very short span of time.

Even more the popular leased in easily auto released a

Click Here

JavaScript 迷宮救星 – JS nICE AUGUST 28, 2014



我們網頁前的工具常常需要在網站上用到強大的debug來替網站上的JavaScript碼做分析。通常那些檔案都已經被縮小, 壓縮過了。還有被混淆過了。以達成保護資訊的目的。並防止他人將其應用程式碼。不過一旦要修改, 就需要了的確工程量了。這讓我們的開發者非常的頭痛。b. c. 在時代。還是有時掛在那邊卡住。

不過網路上有些神奇的小工具可以幫助我們不僅是把複雜一團的JS碼拆開來變的更美觀的。甚至它會替我們逆向算出已經混淆的變數名稱！那就是這篇要介紹的JS nICE網站。

這個網站的小面超級乾淨, 就像下面一樣, 只須在搜尋框上的JS旁邊填, 只要按下 **NICIFY JAVASCRIPT**按鈕, 組合結果與替換變數的結果就會出現在右邊了！

JS nICE is a tool that help you to make even make even obfuscated JavaScript code readable. JSnICE is Statistical Renaming, Type Inference and De obfuscation.

1. JS Nice



ENTER JAVASCRIPT

1 // Put your JavaScript/JS here that you want to rename, deobfuscate,

RESULTS

Dimensions:

Probabilistic Learning from Big Code

Applications	Code completion Deobfuscation	Program synthesis	Feedback generation	Translation
Intermediate Representation	Sequences (sentences) Trees	Translation Table	Graphical Models (CRFs) Feature Vectors	
Analyze Program (PL)	typestate analysis scope analysis	control-flow analysis	alias analysis	
Train Model (ML)	Neural Networks N-gram/back-off	SVM PCFG	Structured SVM	Pseudo-Likelihood
Query Model (ML)	$\text{argmax}_{y \in \Omega} P(y x)$		Greedy MAP inference	

More information
and tutorials at:

<http://www.nice2predict.org/>
<http://www.srl.inf.ethz.ch/>

Key Challenge

Existing Programs



Learning

Model

Probabilistic
Model



Widely
Applicable

Efficient
Learning

High
Precision

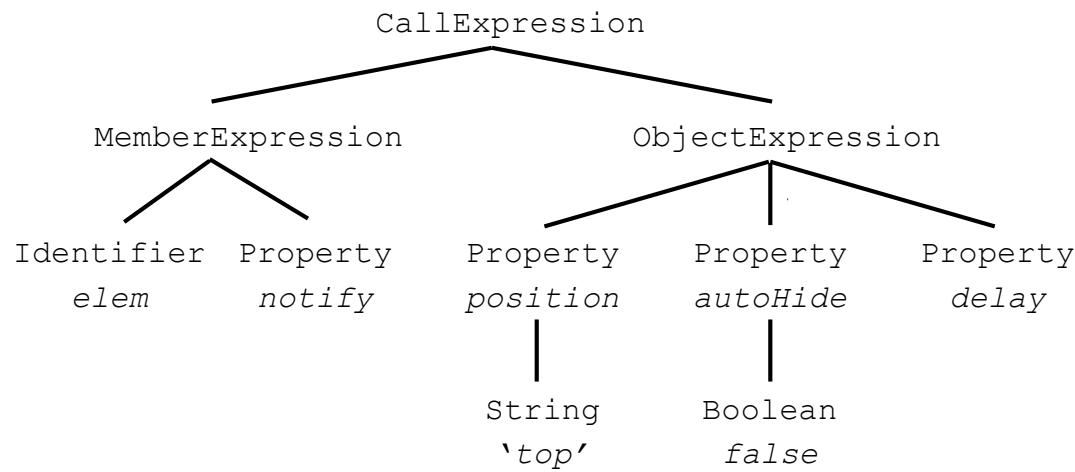
Explainable
Predictions

Representing Programs as Trees

JavaScript expression:

```
elem.notify({  
    position: 'top',  
    autoHide: false,  
    delay: 100  
});
```

Tree:

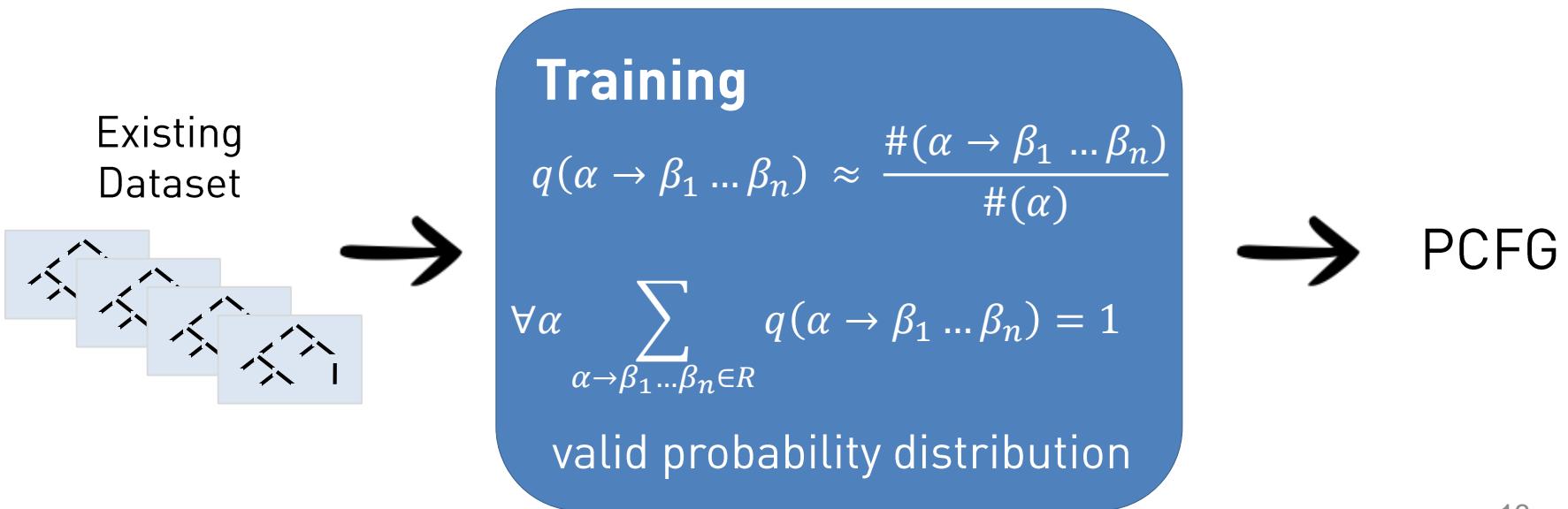


Probabilistic Context Free Grammars (PCFGs)

N : non-terminal symbols Σ : terminal symbols $S \in N$: start symbol

R is a finite set of production rules $\alpha \rightarrow \beta_1 \dots \beta_n$
 $\alpha \in N$ $\beta_i \in (N \cup \Sigma)$

$q: R \rightarrow \mathbb{R}^{(0,1)}$ is a conditional probability of choosing a production rule



PCFG

PCFG

$\alpha \rightarrow \beta_1 \dots \beta_n$

P

Property → x	0.05
Property → y	0.03
Property → promise	0.001

PHOG: Generalizing PCFG

PCFG
 $\alpha \rightarrow \beta_1 \dots \beta_n$

PHOG
 $\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n$

	P
Property → x	0.05
Property → y	0.03
Property → promise	0.001

	P
Property[reject, promise] → promise	0.67
Property[reject, promise] → notify	0.12
Property[reject, promise] → resolve	0.11

PHOG

Production Rules:

$$\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n$$

Function:

$$f: \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{AST} \\ \text{---} \\ \text{---} \end{array} \rightarrow \gamma$$



Key Idea:

“...All problems in computer science can be solved by **another level of indirection...**”

-- David Wheeler

**Parametrize the grammar by a function
used to dynamically obtain the context**

PHOG

Production Rules:

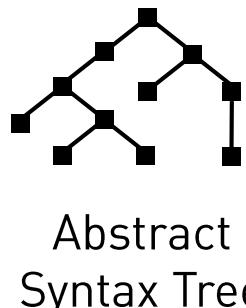
$$\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n$$

Function:

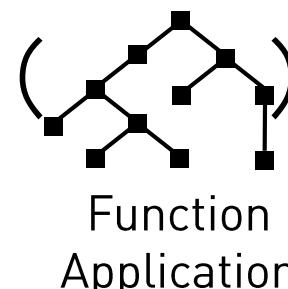
$$f: \begin{matrix} \text{Source Code} \\ \text{AST} \\ \text{Conditioning Context} \end{matrix} \rightarrow \gamma$$



Source
Code



Abstract
Syntax Tree



Function
Application



Conditioning
Context

What are these functions?

in general: can be unrestricted programs (Turing complete)

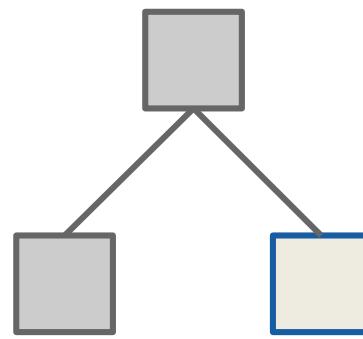
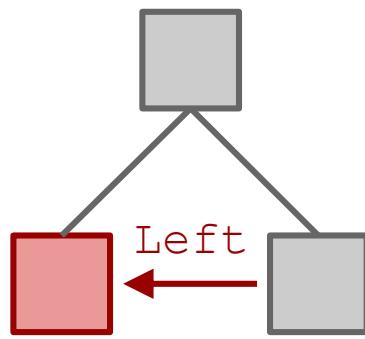
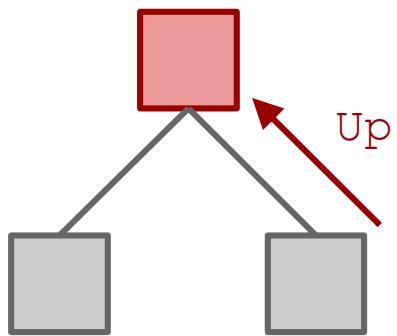
so far: language to iterate over trees, accumulate context

TCond ::= ε | WriteOp TCond | MoveOp TCond

MoveOp ::= Up, Left, Right, DownFirst, DownLast,
NextDFS, PrevDFS, NextLeaf, PrevLeaf,
PrevNodeType, PrevNodeValue, PrevNodeContext

WriteOp ::= WriteValue, WriteType, WritePos

Expressing functions



WriteValue

$$\gamma \leftarrow \gamma \cdot \square$$

Function Evaluation: Example

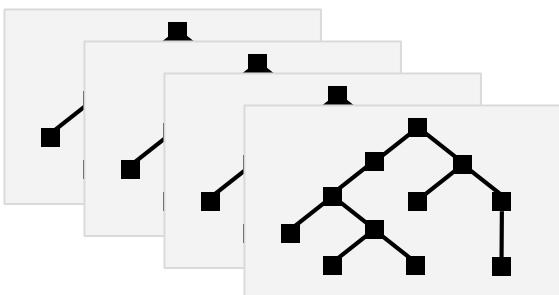
Query	F	γ
elem.notify(... , ... , { position: 'top', hide: false, ? });	Left WriteValue Up WritePos Up DownFirst DownLast WriteValue	{ } { hide } { hide } { hide, 3 } { hide, 3 } { hide, 3 } { hide, 3 } { hide, 3, notify }



{ Previous Property, Parameter Position, API name }

Learning a PHOG

Existing Dataset D



Synthesis: practically intractable

Key Idea: iterative synthesis on fraction of examples, [POPL'16]

$$f_{best} = \arg \min_{f \in TCond} cost(D, f)$$

```
TCond ::= ε | WriteOp TCond | MoveOp TCond
MoveOp ::= Up, Left, Right, ...
WriteOp ::= WriteValue, WriteType, ...
```



TCond Language

PHOG: Key Ideas

$$\alpha [\gamma] \xrightarrow{0.02} \beta_1 \dots \beta_n$$

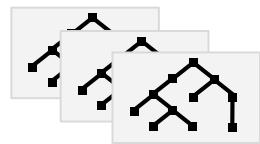
PHOG: Probabilistic Mode for Code, **ACM ICML'16**

P.Bielik, V.Raychev, M.V.

Parametrize grammar by a function $F : \text{graph} \rightarrow \gamma$

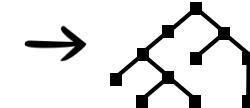


Synthesize F from Dataset D



$$F = \arg \min_{F'} \text{cost}(D, F')$$

Apply F to a new program



$$F(\text{tree}) \rightarrow \gamma$$

Current and Future Research

Learning Reasoning Engines

Beyond code
(e.g., NLP)

New Models of Code:
Neural + PHOG

Spinoff ETH zürich
New AI Programming
Assistants

Learning Points-To Analysis for JavaScript

v1 = v2 [Assignment]



VarPtsTo(v2, h)

Assign(v1, v2)

VarPtsTo(v1, h)

```
function isBig(v) {  
    return v < this.length  [This]  
}  
[12, 5].filter(isBig);
```



VarPtsTo("global", h)

checkIfInsideMethodCall checkMethodName
checkReceiverType checkNumberOfArguments ...

VarPtsTo(this, h)



does not handle these:

Function.prototype

call()

apply()

Array.prototype

map()

some()

forEach()

every()

filter()

...

Array

from()

JSON

stringify()



Goal: Can we learn the production rules?

VarPtsTo(v2, h)

v2 = f(v1)

VarPtsTo(v1, h)

How do I get started?



Reading:

Veselin
Raychev

Learning from Large Codebases, PhD Thesis, ETH Zurich, 2016

Dagstuhl Seminar on Big Code Analytics, Nov 2015

- ML, NLP, PL, SE
- Link to materials, people in the general area

<http://learningfrombigcode.org>

- data sets, tools, challenges

<http://nice2predict.org>

- open source, online demo, build your own tool

@SRL: Two Directions

- Machine Learning based programming tools
- Probabilistic programming

Probabilistic Programming

Promise: simplifies the construction and querying of probabilistic models, even by non-experts.

Applications: reliable machine learning, vision, robotics, networks, security, cyber-physical systems, etc.

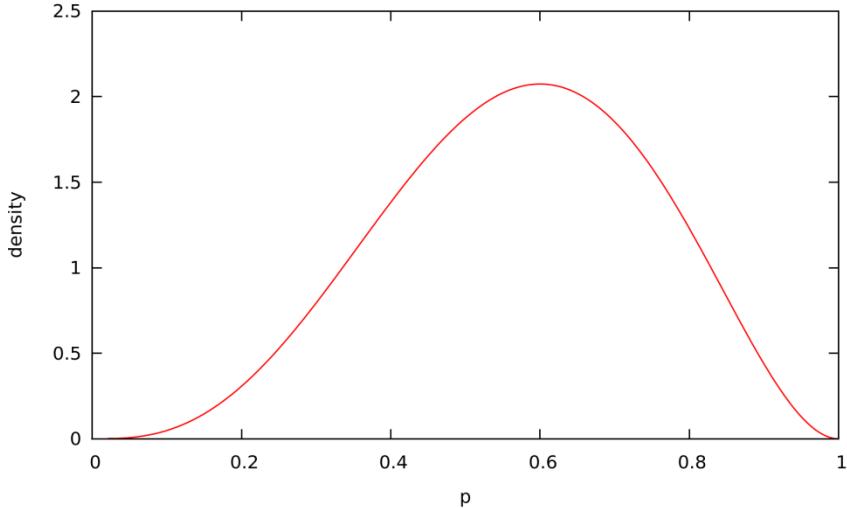
Probabilistic Programs

Express a probabilistic model

```
def main() {  
    p := Uniform(0,1);  
    r := [1,1,0,1,0];  
  
    for i in [0..r.length] {  
        observe(Bernoulli(p) == r[i]);  
    }  
    return p;  
}
```



$$\text{PDF}(p) = (60p^2 - 120p + 60) * [-1 + p \leq 0] * [-p \leq 0] * p^3$$



Can contain a mixture of Discrete and Continuous Distributions

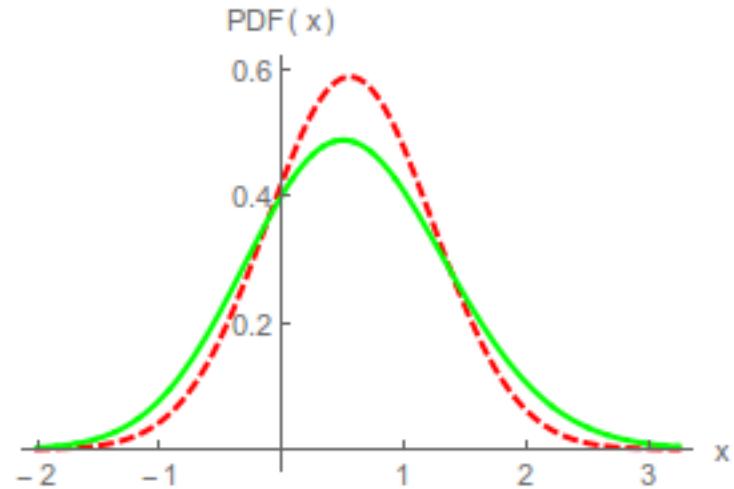
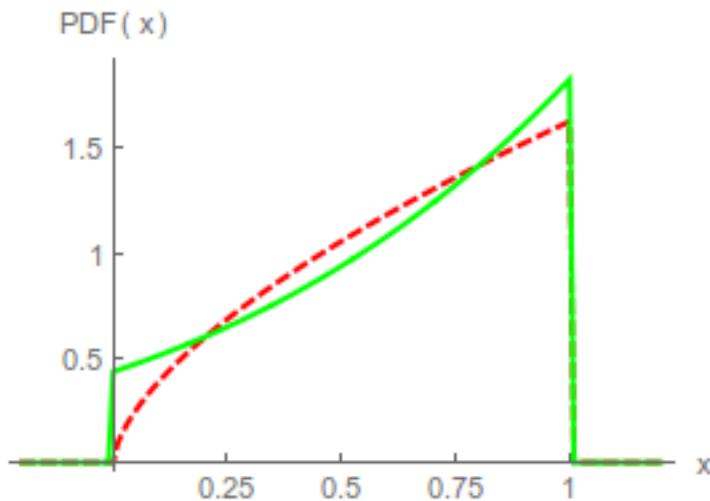
Ultimate Goal

Automatically and exactly answer queries on
the distribution represented by the program

Example queries: probabilities, expectations

Hard problem: complex programs, mixture of discrete and continuous distributions, etc.

Most Existing Work: Approximate



Exact (solid) and Infer.NET (dashed)

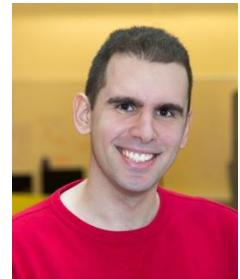
Heuristic solutions, no guarantees

Our Work: PSI

PSI: Exact Symbolic Inference for Probabilistic Programs, CAV'16



Timon
Gehr



Sasa
Misailovic

PSI \approx SAT/SMT for probabilistic programming

<http://psisolver.org/>

PSI Solver

```
def max(a,b) {
    r := a;
    if b > r { r = b; }
    return r;
}

def main() {
    x := Uniform(0,1);
    y := Gauss(0,1);
    z := Uniform(0,1);
    r := max(x,max(y,z));
    observe(x < 0.75);
    if Bernoulli(1/2)
        { assert(r < 0.9); }
    return r + UniformInt(1,3);
}
```

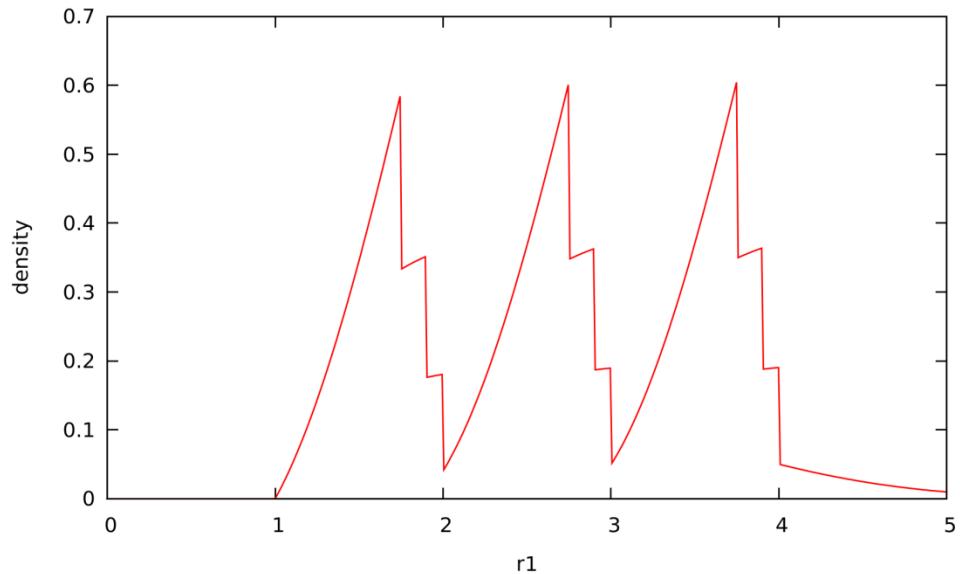
PSI Solver

PSI Solver

```
def max(a,b) {  
    r := a;  
    if b > r { r = b; }  
    return r;  
}  
  
def main() {  
    x := Uniform(0,1);  
    y := Gauss(0,1);  
    z := Uniform(0,1);  
    r := max(x,max(y,z));  
    observe(x < 0.75);  
    if Bernoulli(1/2)  
    { assert(r < 0.9); }  
    return r + UniformInt(1,3);  
}
```



A little nicer to look at...

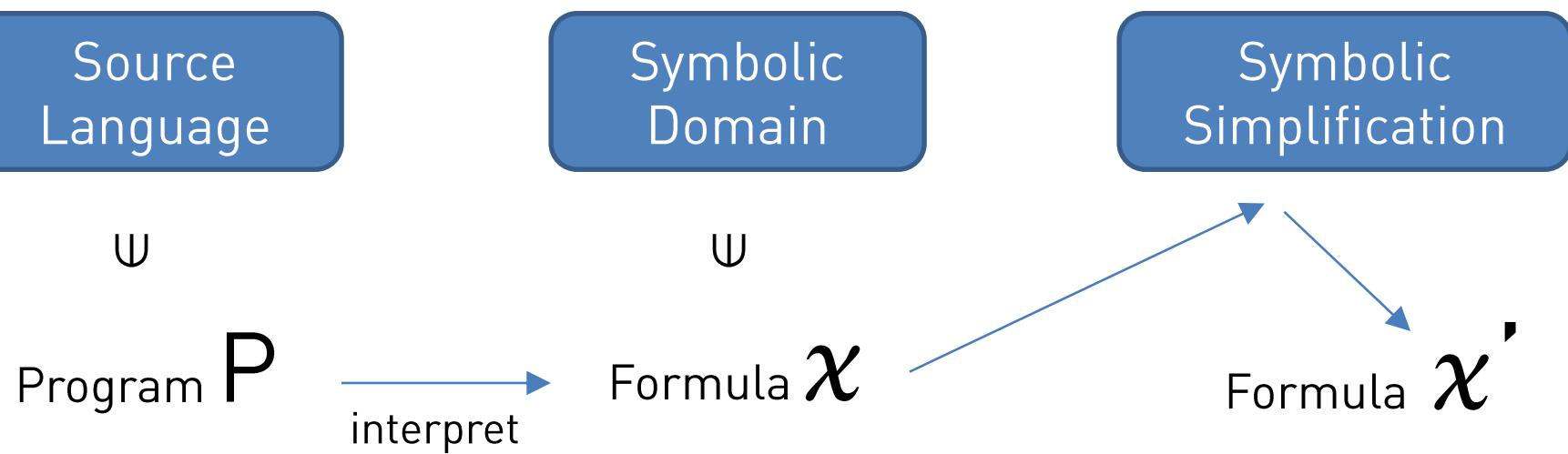


Can compute various queries on the PDF:
e.g., expectations, marginal probabilities

$$\mathbf{E}[\text{result}] \approx 2.6929$$

$$\mathbf{Pr}[\text{error}] \approx 0.132827$$

PSI: Ingredients and Flow



PSI: Symbolic Domain

$$\begin{aligned} e \in E ::= & \quad x \mid \text{e} \mid \pi \mid 0 \mid 1 \mid 2 \mid \dots \\ & \mid \log(e) \mid -e \mid e_1 + \dots + e_n \mid e_1 \cdot \dots \cdot e_n \mid e_1^{e_2} \\ & \mid \delta(e) \mid [e_1 = e_2] \mid [e_1 \leq e_2] \mid [e_1 \neq e_2] \mid [e_1 < e_2] \\ & \mid \int_{\mathbb{R}} dx \, e[\![x]\!] \mid \sum_{x \in \mathbb{Z}} e[\![x]\!] \mid \varphi(e_1, \dots, e_n) \\ & \mid (\mathrm{d}/\mathrm{d}x)^{-1}[\!\exp^{-x^2}\!](e) \text{ (Error function)} \end{aligned}$$

Encodes probability density functions

- ▶ $\text{Bernoulli}(x; p) = p \cdot \delta(1 - x) + (1 - p) \cdot \delta(x)$
- ▶ $\text{Gauss}(x; \mu, \nu) = [\nu = 0] \cdot \delta(x - \mu) + [\nu \neq 0] \cdot \frac{\exp^{-(x-\mu)^2/(2\nu)}}{(2\pi\nu)^{\frac{1}{2}}}$
- ▶ $\text{UniformInt}(x; a, b) = \frac{\sum_{x' \in \mathbb{Z}} \delta(x - x') \cdot [a \leq x'] \cdot [x' \leq b]}{\sum_{x' \in \mathbb{Z}} [a \leq x'] \cdot [x' \leq b]}$

PSI: From Language to Domain

```
def main() {
```

$$p() = 1$$

```
    x := Uniform(0,1);
```

$$p(x) = [0 \leq x] * [x \leq 1]$$

```
    y := Uniform(0,1);
```

$$p(x,y) = [0 \leq x] * [x \leq 1] * [0 \leq y] * [y \leq 1]$$

```
    z := x * y
```

$$p(x,y,z) = [0 \leq x] * [x \leq 1] * [0 \leq y] * [y \leq 1] * \delta(z - x * y)$$

```
return z;
```

$$p(z) =$$

$$\int dx \int dy [0 \leq x] * [x \leq 1] * [0 \leq y] * [y \leq 1] * \delta(z - x * y)$$

$$p(z) = - [z - 1 \leq 0] * [z \neq 0] * [-z \leq 0] * \log(z)$$

PSI: Symbolic Simplification

- ▶ Basic algebraic simplifications

$$x + x \rightarrow 2 \cdot x, \quad x \cdot x \rightarrow x^2, \dots$$

- ▶ Simplifications on constraints

$$[x = 0] + [x \neq 0] \rightarrow 1, \quad [x \leq 0] \cdot [0 \leq x] \rightarrow [x = 0],$$

$$\delta(x) \cdot [1 \leq x] \rightarrow 0, \dots$$

- ▶ Linearize Guards

$$\delta(y - x^2) \rightarrow$$

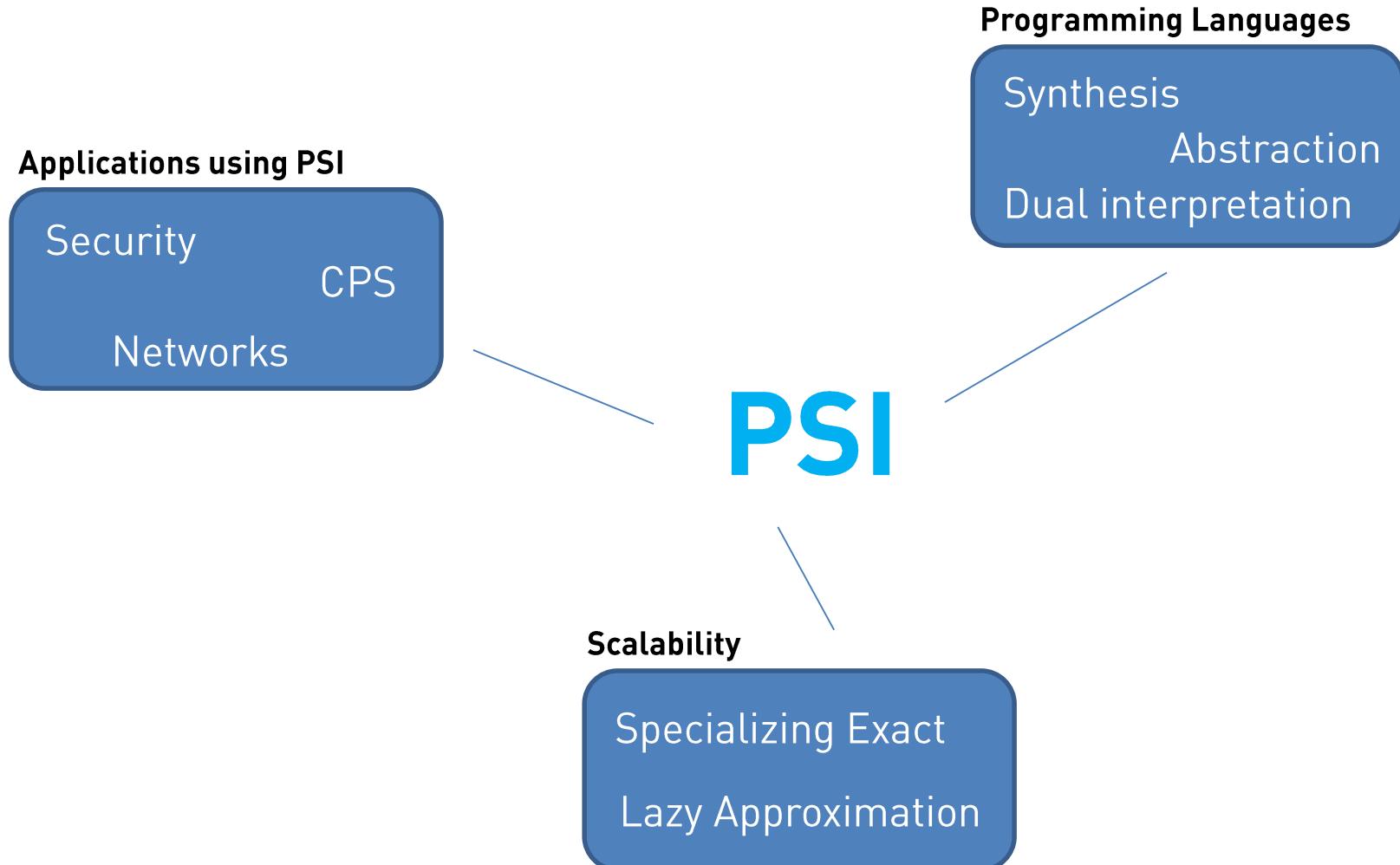
$$[-y \leq 0] \cdot ([x = 0] \cdot \delta(y) + [x \neq 0] \cdot \tfrac{1}{2\sqrt{y}}(\delta(x - \sqrt{y}) + \delta(x + \sqrt{y}))),$$

...

- ▶ Symbolic Integration

$$\int_{\mathbb{R}} dx \int_{\mathbb{R}} dy [0 \leq x] \cdot [x \leq 1] \cdot [0 \leq y] \cdot [y \leq 1] \cdot \delta(z - x \cdot y) \rightarrow \\ -[0 < z] \cdot [z \leq 1] \cdot \log(z), \dots$$

PSI: Current and Future Research



Machine learning based programming tools



$$\alpha [y] \rightarrow \beta_1 \dots \beta_n$$

0.02



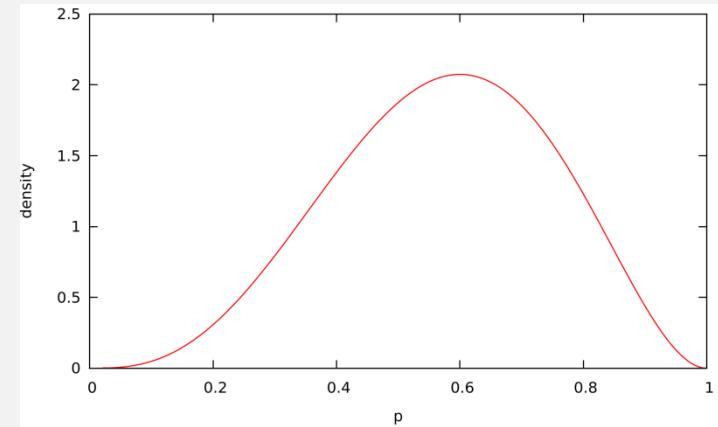
Parametrize grammar by a function $F : \text{file} \rightarrow y$

Synthesize F from D $\Rightarrow F = \arg \min_{F'} \text{cost}(D, F')$

Apply F to program $\rightarrow \text{file} \rightarrow F(\text{file}) \rightarrow y$

Probabilistic Programming

```
def main() {  
    p := Uniform(0,1);  
    r := [1,1,0,1,0];  
  
    for i in [0..r.length] {  
        observe(Bernoulli(p) == r[i]);  
    }  
    return p;  
}
```



More: <http://www.srl.inf.ethz.ch/>