Machine Learning and Programming Languages

Martin Vechev
Software Reliability Lab
Department of Computer Science, ETH Zurich

ETH Zürich

European Research Council
erc

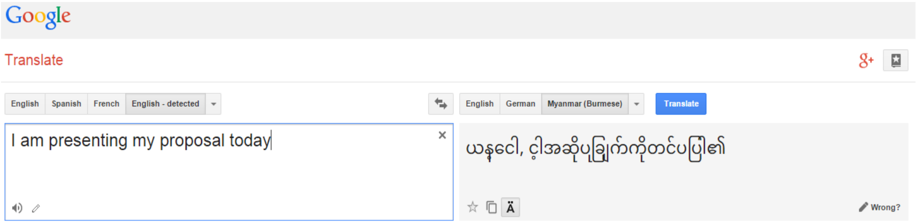
@SRL: Two Directions

- Machine Learning based programming tools

- Probabilistic programming

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

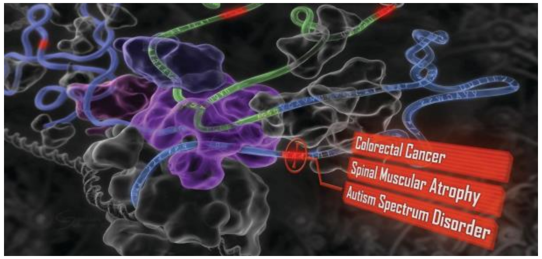
Natural Language Processing
(e.g., machine translation)



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?

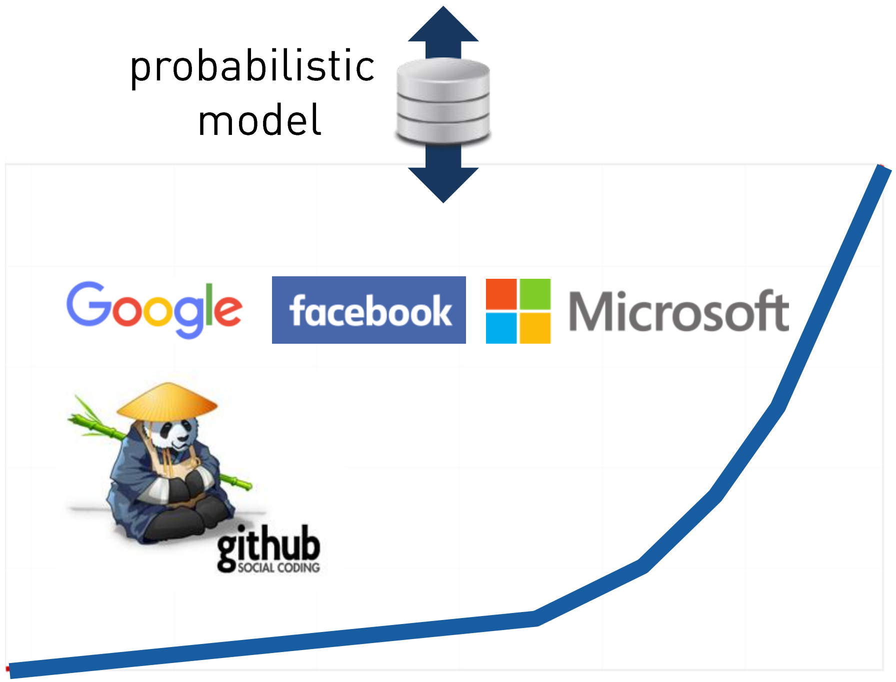
Vision

Learning from large datasets of programs

Programming Task → **Statistical Programming Tool** → **Solution**

probabilistic model

number of repositories



last 5 years

15 million repositories

Billions of lines of code

High quality, tested, maintained 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

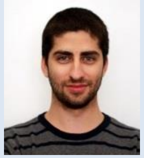

Christine Zeller


Svetoslav Karaivanov


Pascal Roos


Benjamin Bischel


Timon Gehr


Petar Tsankov


Mateo Panzacchi

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

ETH Zürich

Statistical Code Completion

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

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

SLANG

```
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.setAudioSource(MediaRecorder.AS);
r.setVideoSource(MediaRecorder.VS);
r.setOutFormat(MediaRecorder.MPEG4);
```

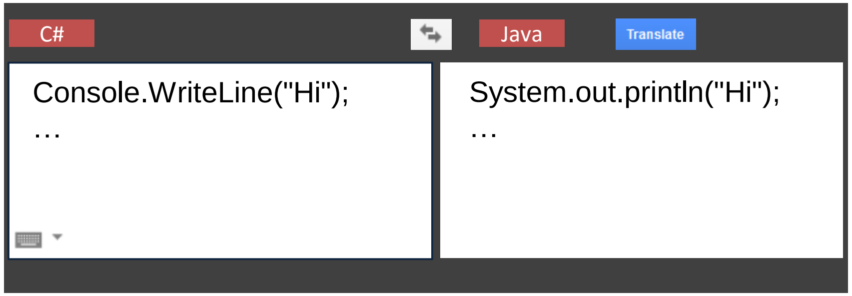
Statistical language models
Recurrent neural networks

+

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;
}
```

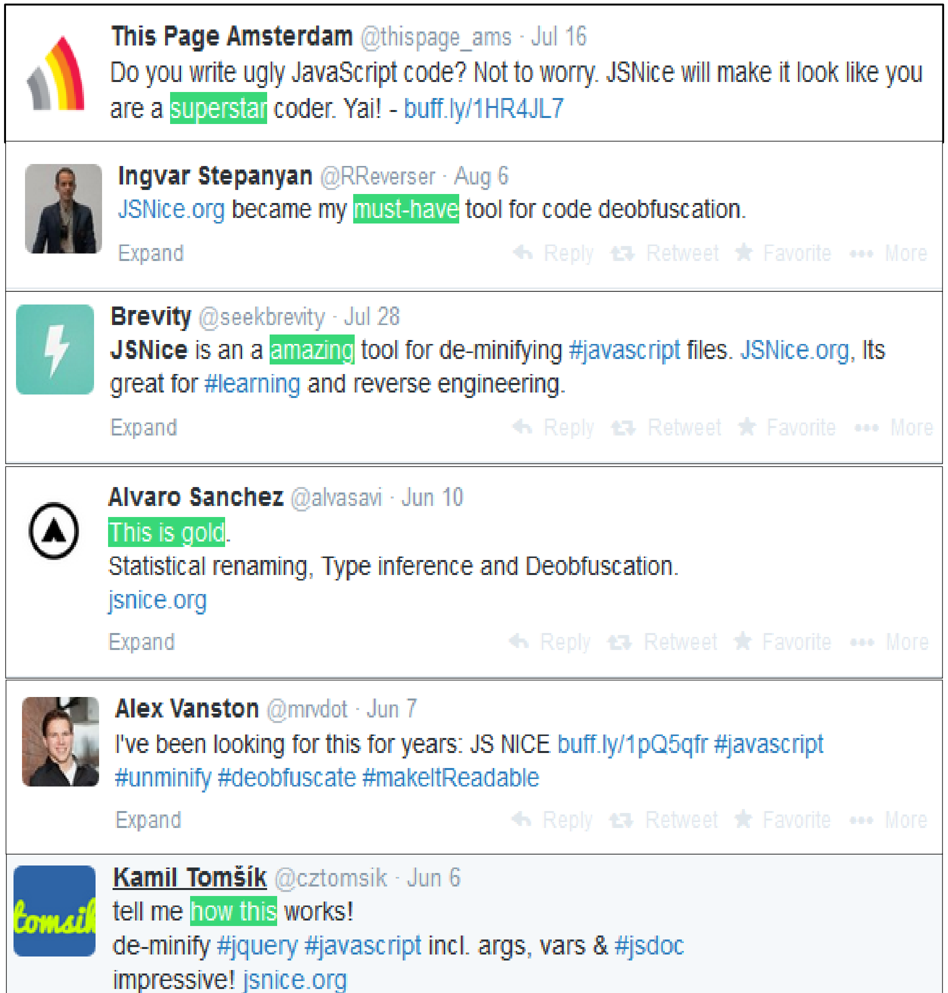
JSNice.org

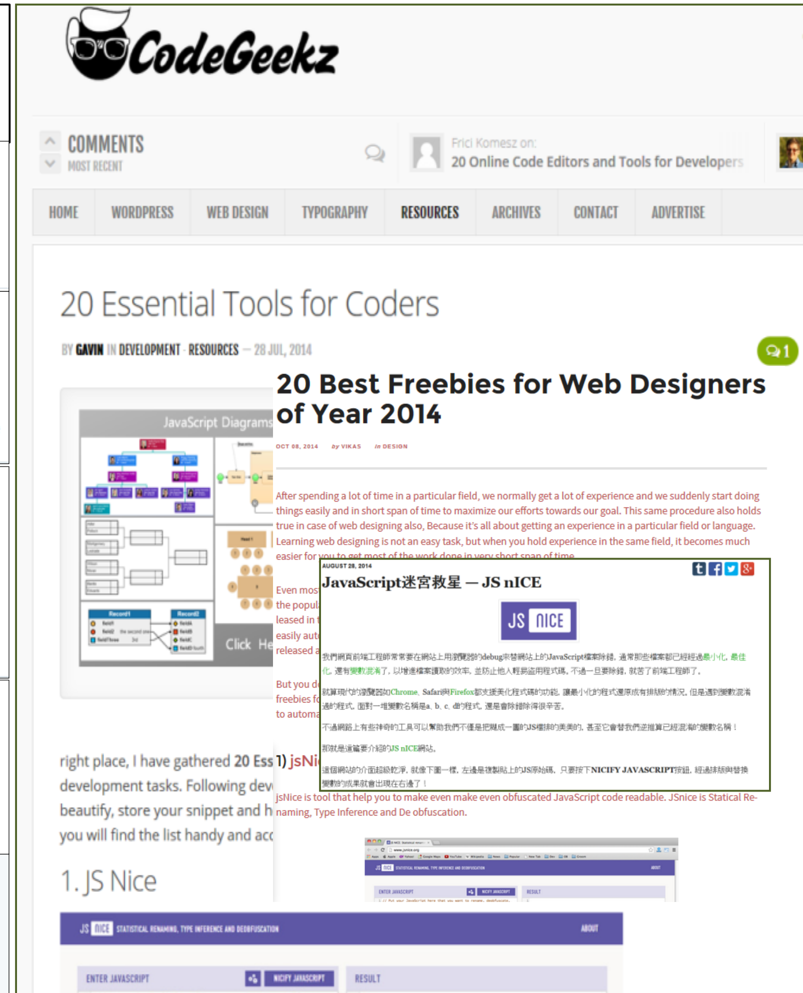
```
function chunkData(str, step) {
  var colNames = [];
  var len = str.length;
  var i = 0;
  for (; i < len; i += step)
    if (i + step < len)
      colNames.push(str.substring(i, i + step));
   else
      colNames.push(str.substring(i, len));
  return colNames;
}
```

JSNice.org

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

✓ Every country ✓ 170,000 users ✓ Top ranked tool





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)	$\underset{y \in \Omega}{\text{argmax}}\ P(y \mid x)$, Greedy MAP inference

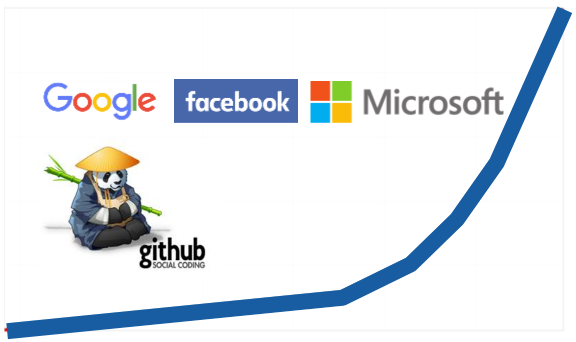
More information and tutorials at: http://www.nice2predict.org/ http://www.srl.inf.ethz.ch/

Key Challenge

Existing Programs

Learning

Model



Probabilistic Model

Widely Applicable

Efficient Learning

High Precision

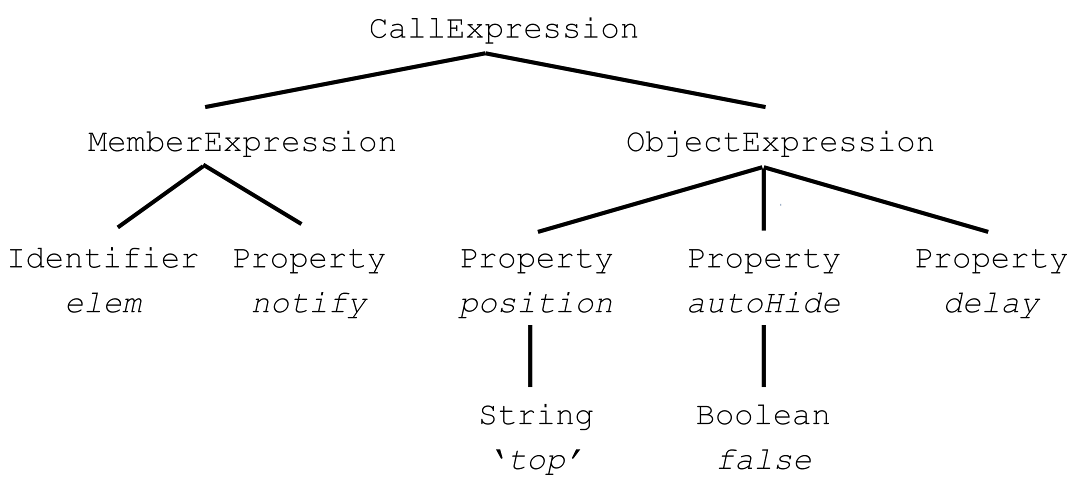
Explainable Predictions

Representing Programs as Trees

JavaScript expression:

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

Tree:



Probabilistic Context Free Grammars (PCFGs)

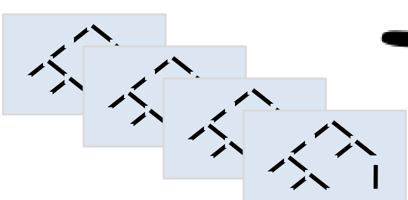
N: non-terminal symbols Σ : terminal symbols $S \in N$: start symbol

R is a finite set of production rules $\alpha \rightarrow \beta_1 \dots \beta_n$

$\alpha \in N$ $\beta_i \in (N \cup \Sigma)$

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

Existing Dataset



Training

$$q(\alpha \rightarrow \beta_1 \dots \beta_n) \approx \frac{\#(\alpha \rightarrow \beta_1 \dots \beta_n)}{\#(\alpha)}$$

$$\forall \alpha \sum_{\alpha \rightarrow \beta_1 \dots \beta_n \in R} q(\alpha \rightarrow \beta_1 \dots \beta_n) = 1$$

valid probability distribution

PCFG

PCFG

PCFG

$$\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}_1 \ldots \boldsymbol{\beta}_n$$

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

PHOG: Generalizing PCFG

PCFG	PHOG
$\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}_1 \ldots \boldsymbol{\beta}_n$	$\boldsymbol{\alpha}[\boldsymbol{\gamma}] \rightarrow \boldsymbol{\beta}_1 \ldots \boldsymbol{\beta}_n$

	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:

$$\boldsymbol{\alpha}[\boldsymbol{\gamma}] \rightarrow \boldsymbol{\beta}_1 \dots \boldsymbol{\beta}_n$$

Function:

$$f: \text{AST} \rightarrow \boldsymbol{\gamma}$$

"...All problems in computer science can be solved by **another level of indirection**..."
-- David Wheeler

Key Idea:

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

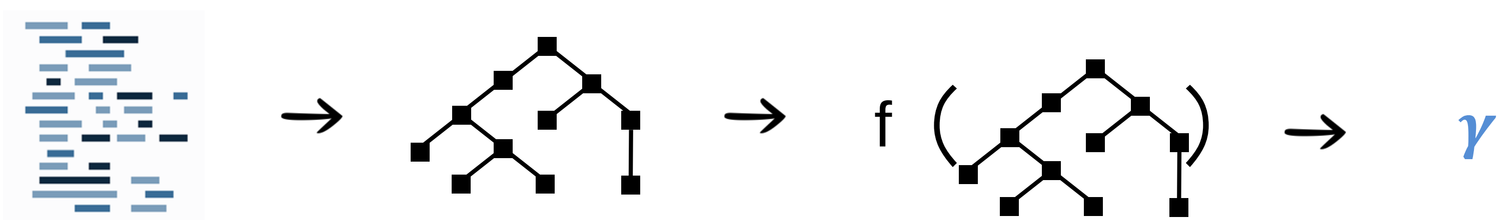
PHOG

Production Rules:

$$\boldsymbol{\alpha}[\gamma] \rightarrow \boldsymbol{\beta}_1 \ldots \boldsymbol{\beta}_n$$

Function:

$$f: \text{AST} \rightarrow \gamma$$



Source Code

Abstract Syntax Tree

Function Application

Conditioning Context

What are these functions?

in general: can be unrestricted programs (Turing complete)

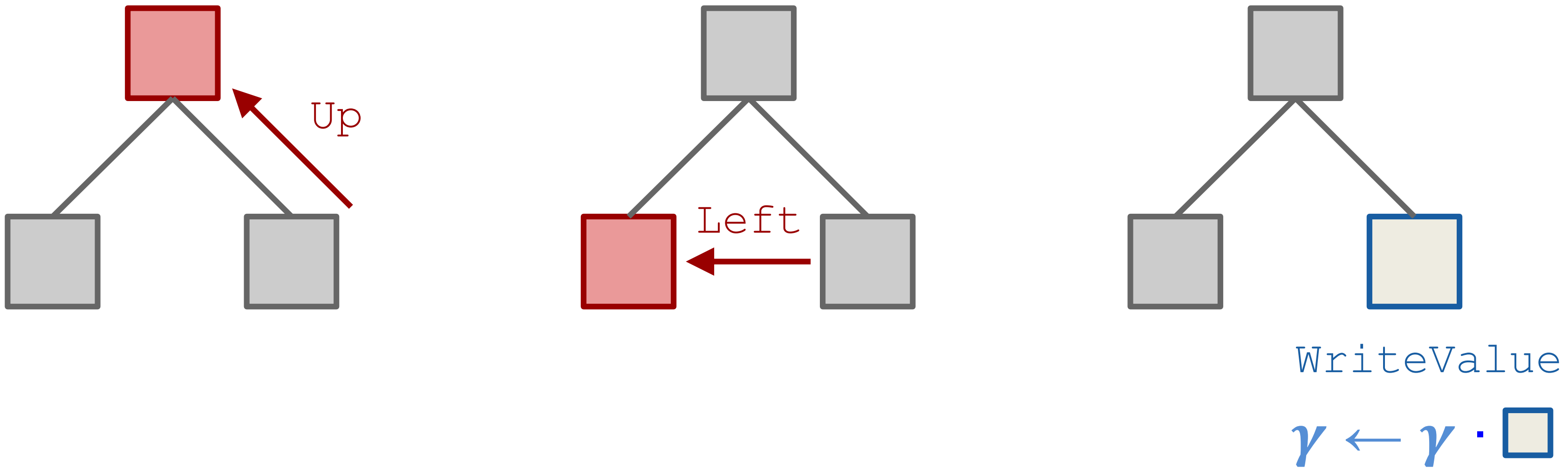
so far: language to iterate over trees, accumulate context

```
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



Function Evaluation: Example

Query	F	γ
	Left	{}
	WriteValue	{hide}
	Up	{hide}
	WritePos	{hide, 3}
	Up	{hide, 3}
	DownFirst	{hide, 3}
	DownLast	{hide, 3}
	WriteValue	{hide, 3, notify}

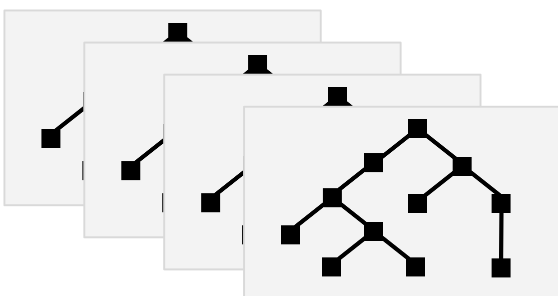
```
elem.notify(
   ... ,
   ... ,
  {
     position: 'top',
     hide: false,
     ?
  }
);
```

↓

{ 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]

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

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

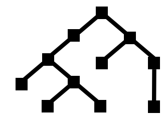
TCond Language

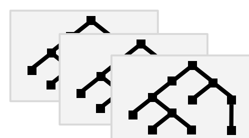
PHOG: Key Ideas

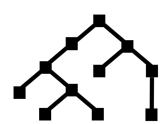
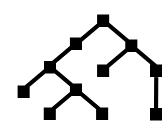
$\alpha\ [\gamma] \xrightarrow[0.02]{} \beta_1 \ldots \beta_n$

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

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

Parametrize grammar by a function **F** :  $\rightarrow \gamma$

Synthesize **F** from Dataset D  **F** = arg min *cost* (D, F')
F'

Apply **F** to a new program  $\rightarrow$  $\rightarrow$ **F** () $\rightarrow \gamma$

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` **[Assigment]** →

$$\frac{\texttt{VarPtsTo}(\mathit{v2},\ \mathit{h}) \quad \texttt{Assign}(\mathit{v1},\ \mathit{v2})}{\texttt{VarPtsTo}(\mathit{v1},\ \mathit{h})}$$

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

→

$$\frac{\texttt{VarPtsTo}(\text{“global”},\ \mathit{h}) \quad \mathit{checkIfInsideMethodCall} \quad \mathit{checkMethodCallName} \quad \mathit{checkReceiverType} \quad \mathit{checkNumberOfArguments}\ \ldots}{\texttt{VarPtsTo}(\mathit{this},\ \mathit{h})}$$

flow does not handle these:

Function.prototype
- `call()`
- `apply()`

Array
- `from()`

JSON
- `stringify()`

Array.prototype
- `map()`
- `some()`
- `forEach()`
- `every()`
- `filter()`
- …

…

→

Goal: Can we learn the production rules?

$$\frac{\texttt{VarPtsTo}(\mathit{v2},\ \mathit{h}) \quad \mathbf{v2 = f(v1)}}{\texttt{VarPtsTo}(\mathit{v1},\ \mathit{h})}$$

How do I get started?

Veselin
Raychev

Reading:

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.

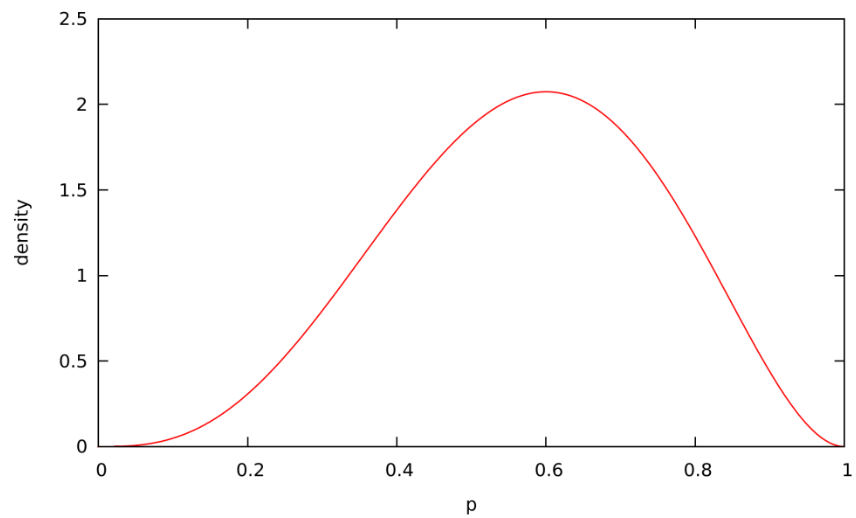
Probabilistic Programs

Express a probabilistic model

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

```
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;
}
```



Can contain a mixture of Discrete and Continuous Distributions

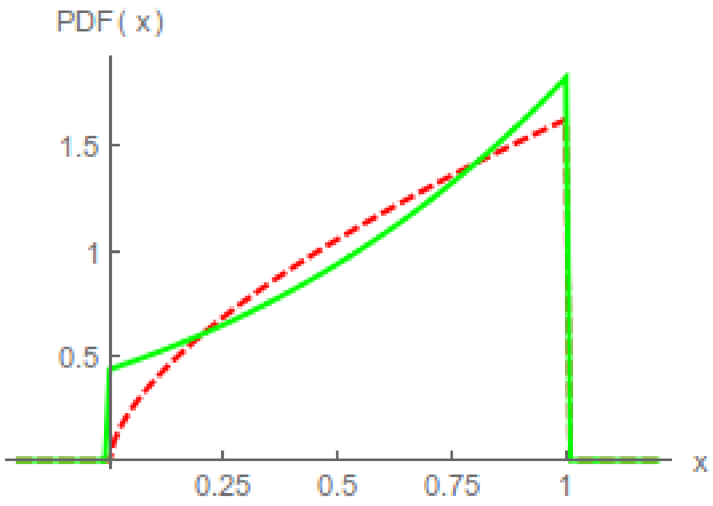
Ultimate Goal

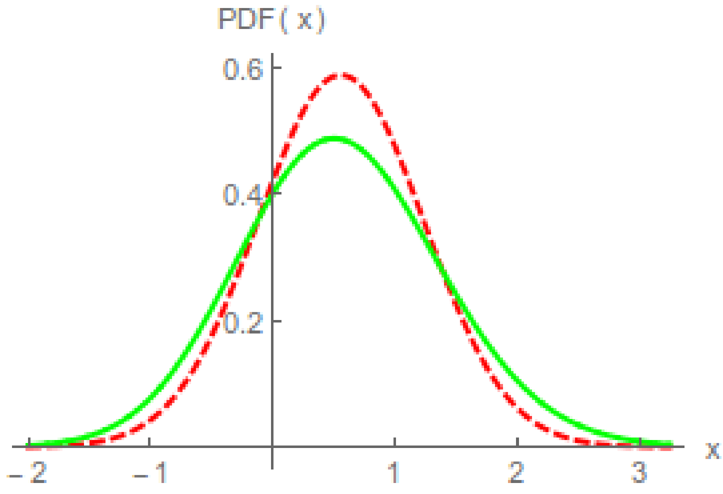
Automatically and exactly answer queries on the distribution represented by the program

Example queries: probabilities, expectations

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

Most Existing Work: Approximate





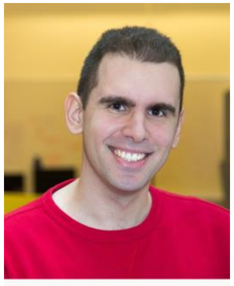
Exact (solid) and Infer.NET (dashed)

Heuristic solutions, no guarantees

Our Work: PSI

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

Timon
Gehr



Sasa
Misailovic

PSI ≈ SAT/SMT for probabilistic programming

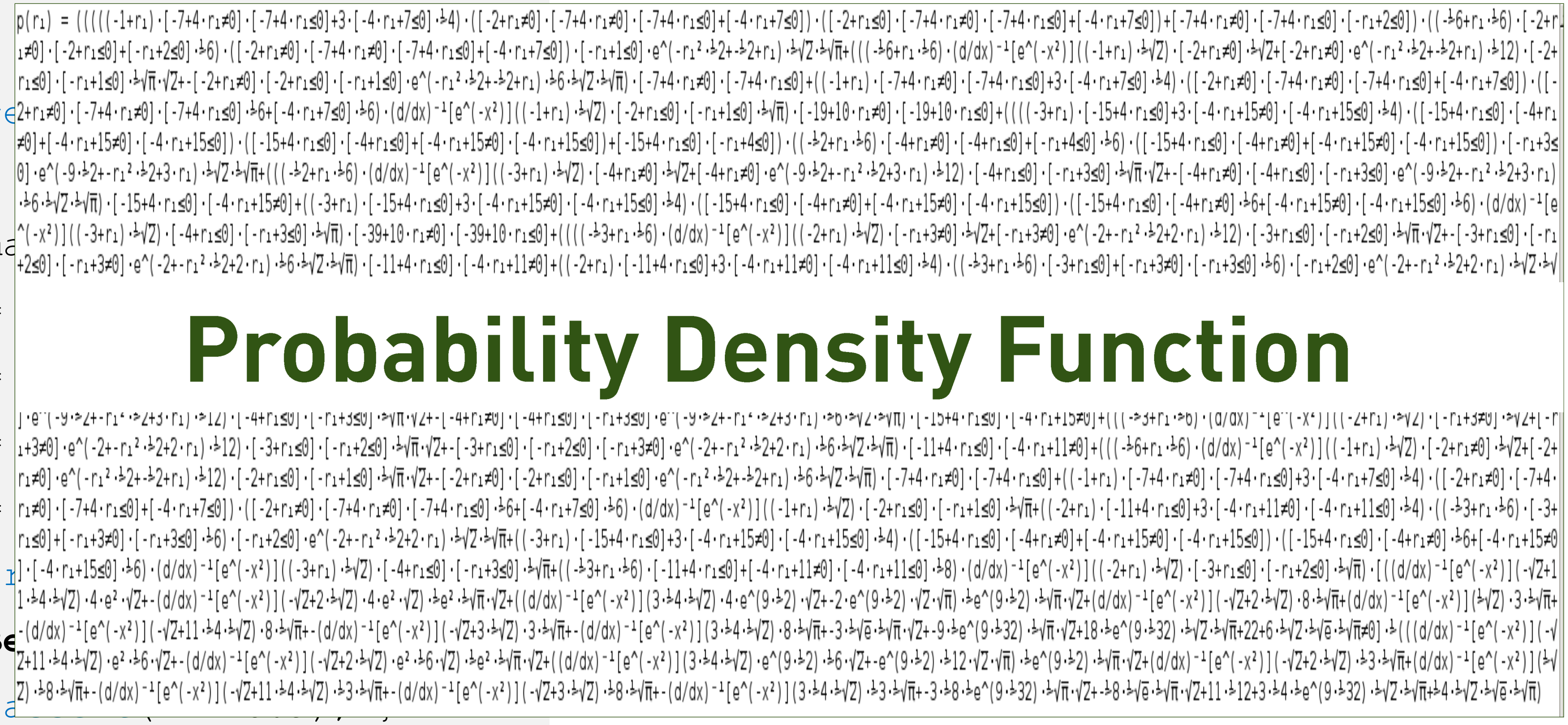
http://psisolver.org/

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

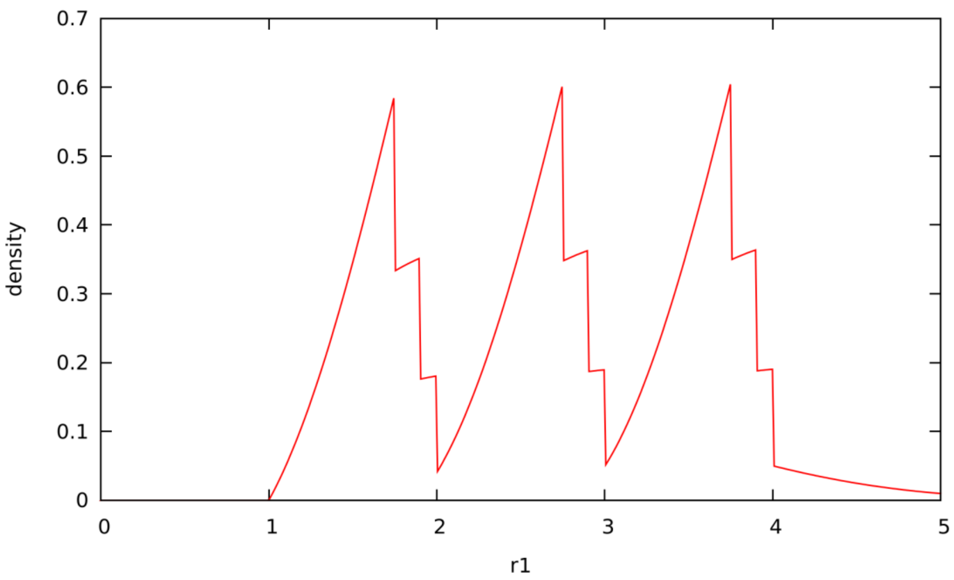
```
def max(a,b) {
  r := a;
  if
    re
}
def ma
 x :=
 y :=
 z :=
 r :=
 obser
 if Be
   { a
 return r + UniformInt(1,3);
}
```



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);
}
```

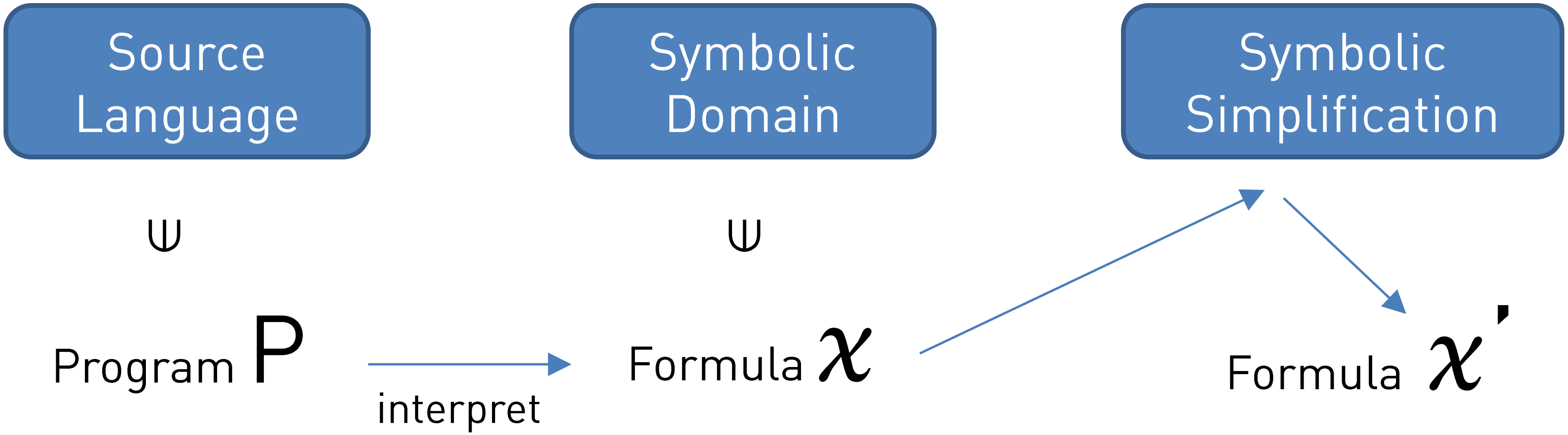
A little nicer to look at...



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

E[result] ≈ 2.6929
Pr[error] ≈ 0.132827

PSI: Ingredients and Flow



PSI: Symbolic Domain

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

Encodes probability density functions

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

PSI: From Language to Domain

```
def main() {
 x := Uniform(0,1);
 y := Uniform(0,1);
 z := x * y
 return z;
}
```

$p() = 1$

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

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

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

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

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

PSI: Symbolic Simplification

- Basic algebraic simplifications

  $x + x \to 2 \cdot x, \quad x \cdot x \to x^2, \ \ldots$

- Simplifications on constraints

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

  $\delta(x) \cdot [1 \leq x] \to 0, \ \ldots$

- Linearize Guards

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

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

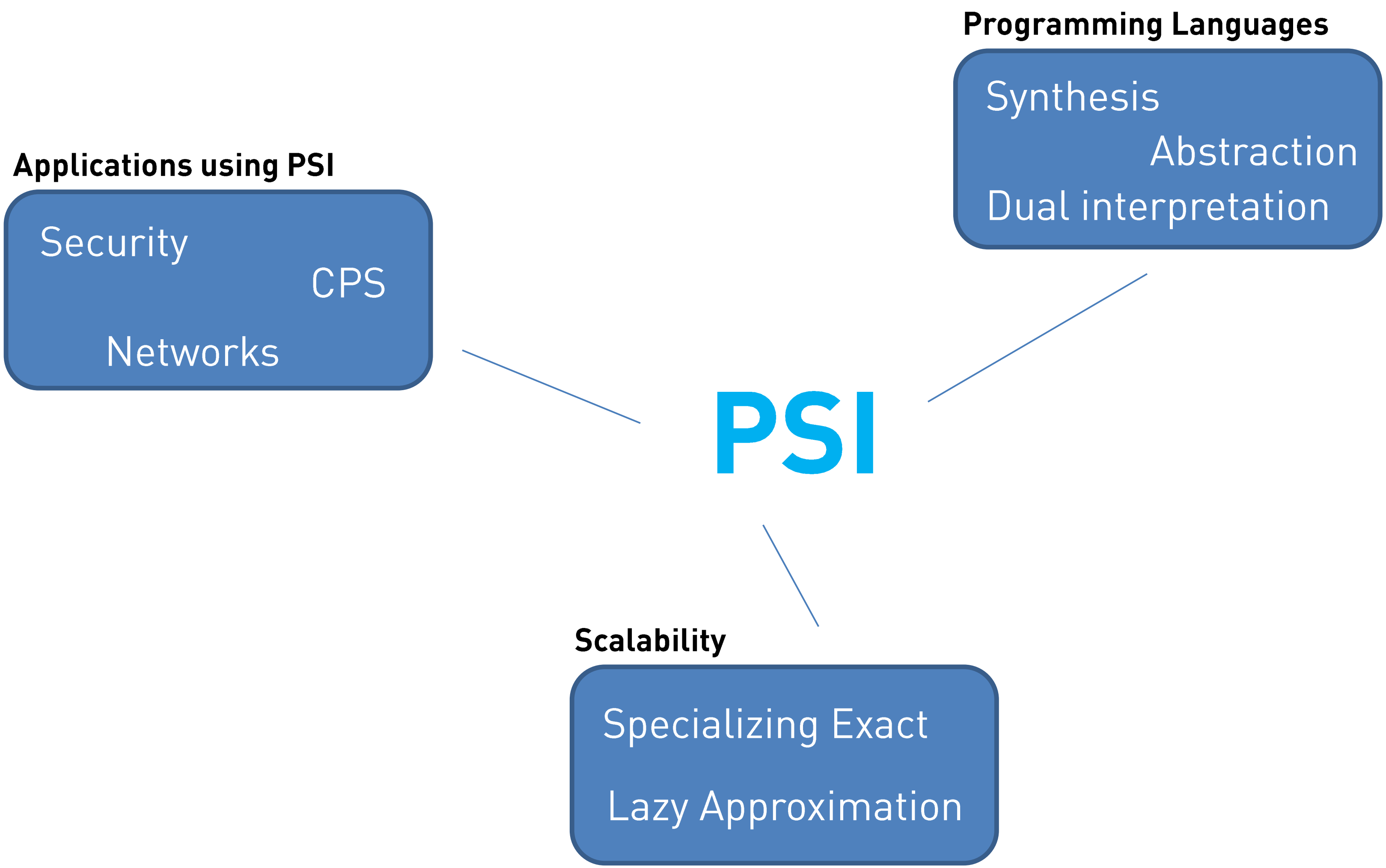
  $\ldots$

- Symbolic Integration

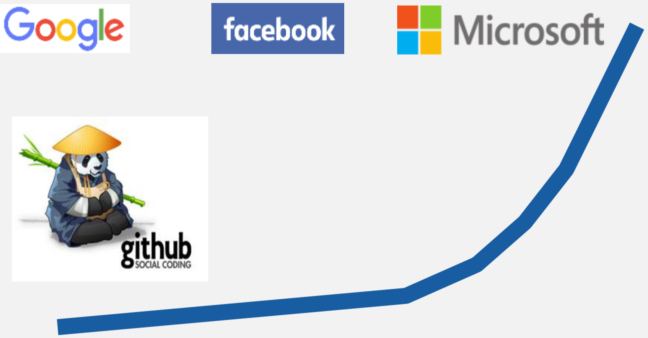
  $\int_{\mathbb{R}} \mathrm{d}x \ \int_{\mathbb{R}} \mathrm{d}y \, [0 \leq x] \cdot [x \leq 1] \cdot [0 \leq y] \cdot [y \leq 1] \cdot \delta(z - x \cdot y) \to$

  $-[0 < z] \cdot [z \leq 1] \cdot \log(z), \ \ldots$

PSI: Current and Future Research



Machine learning based programming tools



$\alpha\,[\gamma] \longrightarrow \beta_1 \ldots \beta_n$

0.02

JS NICE

Parametrize grammar by a function **F** : $\rightarrow \gamma$

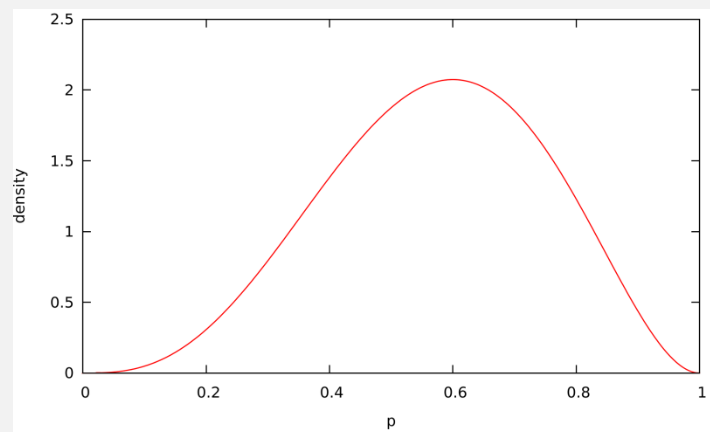
Synthesize **F** from D $\Rightarrow$ **F** = arg min *cost* (D, F')
F'

Apply **F** toprogram $\rightarrow$ $\rightarrow$ **F** ( ) $\rightarrow \gamma$

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/

ETH Zürich