CS 598sm

Probabilistic & Approximate Computing

http://misailo.web.engr.Illinois.edu/courses/cs598

Today's Topic

The software is written by humans, and thus can have errors.

How do we find errors or make sure they are absent?

Testing in a Nutshell

Test Inputs

Program Under Test

Test Oracle

Testing in a Nutshell

Test Inputs

```
1, -1, 0, 2, 3, 100…
```

Program Under Test

```
int square(x) {
  return x*x
}
```

Test Oracle

```
{ 1 : 1
 -1 : 1
  0 : 0
  2 : 4
  3 : 9
100 : 10000 }
```

But we already know:

Approximate Program Analysis =

Accuracy + Safety

How do we adapt the existing testing practices to the new computations?

Analysis Concerns for AI Systems

From: Trustworthy AI by Jeannette M. Wing:

After two decades of investment and advances in research and development, *trustworthy* has come to mean a set of (overlapping) properties:

- Reliability: Does the system do the right thing?
- Safety: Does the system do no harm?
- Security: How vulnerable is the system to attack?
- Privacy: Does the system protect a person's identity and data?
- Availability: Is the system up when I need to access it?
- Usability: Can a human use it easily?

From: Trustworthy AI by Jeannette M. Wing:

Analysis Concerns for AI Systems

AI systems raise the bar in terms of the set of properties of interest. In addition to the properties associated with trustworthy computing (from above), we also want (overlapping) properties such as:

- Accuracy: How well does the AI system do on new (unseen) data compared to data on which it was trained and tested?
- Robustness: How sensitive is the system's outcome to a change in the input?
- Fairness: Are the system outcomes unbiased?
- Accountability: Who or what is responsible for the system's outcome?
- Transparency: Is it clear to an external observer how the system's outcome was produced?
- Interpretability/Explainability: Can the system's outcome be justified with an explanation that a human can understand and/or that is meaningful to the end user?
- Ethical: Was the data collected in an ethical manner? Will the system's outcome be used in an ethical manner?
- …and others, yet to be identified

From: Trustworthy AI by Jeannette M. Wing:

Analysis Concerns for AI Systems

The machine learning community considers accuracy as a gold standard, but trustworthy AI requires us to explore tradeoffs among these properties. For example, perhaps we are willing to give up on some accuracy in order to deploy a fairer model. Also, some of the above properties

Traditional software and hardware systems are complex due to their size and the number of interactions among their components. For the most part, we can define their behavior in terms of discrete logic and as deterministic state machines.

Today's AI systems, especially those using deep neural networks, add a dimension of complexity to traditional computing systems. This complexity is due to their inherent probabilistic nature. Through probabilities, AI systems model the uncertainty of human behavior and the uncertainty of the physical world. More recent advances in machine learning, which rely on big data, add to their probabilistic nature, as data from the real world are just points in a probability space. Thus, trustworthy AI necessarily directs our attention from the primarily deterministic nature of traditional computing systems to the probabilistic nature of AI systems.

How Well Do I Test The Program?

Idea #1:

Coverage:

- Line
- Branch
- Path

```
int square(x) {
  return x*x
}

float abs(float x) {
  if x > 0
    return x;
  else
    return –x;
}

float f(float x, float eps) {
  float s = 0;
  while (x > eps) {
    s = s + x;
    x = x / 2;
  }
  return s;
}
```

How do We Extend?

Make an analogy with the traditional coverage testing

But for most approximate computations, it matters which values we passed, not just which statements branches we visited!

"Neuron Coverage"

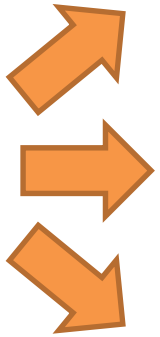
- SOSP paper
- Also: https://github.com/TrustAI/DeepCover

Various works for numerical computations

How Well Did I Test The Program?

Idea #2: Mutation Testing

```
float f(float x, float eps) {
  float s = 0;
  while (x > eps) {
    s = s + x;
    x = x / 2;
  }
  return s;
}
```



```
float f(float x, float eps) {
  float s = 0;
  while (x > eps) {
    s = s - x;
    x = x * 2;
  }
  return s;
}
```

```
float f(float x, float eps) {
  float s = 1;
  while (x < eps) {
    s = s + x;
    x = x / 2;
  }
  return s;
}
```

```
float f(float x, float eps) {
  float s = 0;
  while (x > eps) {
    s = s + eps;
    x = x / 2;
  }
  return x;
}
```

Mutants change the semantics of the program
The Test Suite is successful if it 'kills' all the mutants

Problem: Some mutants may legally pass as reasonable approximations (e.g., Hariri et al. ICST 2018)

An Important Distinction

Testing ML Model: Does the model have the desired property?

- E.g., is the label of a DNN correct wrt human expectation, or is the model robust to noise or perturbations of the inputs

Testing ML System: Is the system implementation correct?

- E.g., is the label of a DNN properly computed, or is the system not crashing on a bad input

What is the Correct Output?

```
0.2 + 0.2 ?

if Bernoulli (0.5) return 1.0 else return -1.0?

img = readImage(…)
label = DNN(img)  ?
```

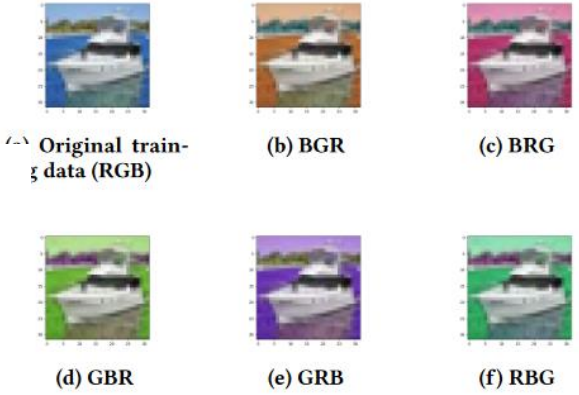
Metamorphic Testing

Test oracle problem = hard to determine the expected outcomes

Metamorphic test = establish a relation that holds between multiple inputs and also between their outputs; if the input gets transformed in a particular way, we will also know how the output will be transformed. E.g., exp(x) = Y. then exp(x+1) = Y*e

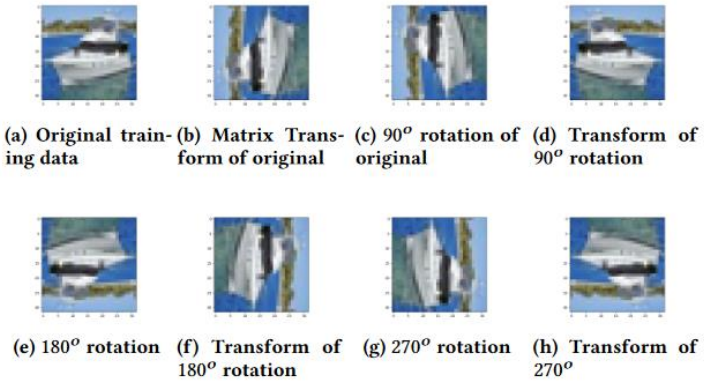
Identifying Implementation Bugs in Machine Learning Based Image Classifiers using Metamorphic Testing (ISSTA 2018)

(1) MR-1: Permutation of training & test features
(2) MR-2: Permutation of order of training instances
(3) MR-3: Shifting of training & test features by a constant (only for RBF kernel)
(4) MR-4: Linear scaling of the test features (only for linear kernel)



(a) Original training data (RGB) (b) BGR (c) BRG (d) GBR (e) GRB (f) RBG

: Permutation of RGB channels for one instance of training data. The test data is permuted in similar fashion. The results should be very similar whether the ResNet application is trained & tested on the original or permuted data.



(a) Original training data (b) Matrix Transform of original (c) 90° rotation of original (d) Transform of 90° rotation (e) 180° rotation (f) Transform of 180° rotation (g) 270° rotation (h) Transform of 270°

Figure 8: Permutation of CONV order for one instance of training data. The test data is permuted in similar fashion

Fuzz Testing

Generate many inputs randomly.

See if the system fails (crashes, hangs, etc.)

or develop some metamorphic relation to use as the oracle

Had a lot of success in traditional programming systems

- Compilers
- Security

CSmith

Generates arbitrary C programs that conform to the C99 standard.

- *Finding and Understanding Bugs in C Compilers Xuejun Yang, Yang Chen, Eric Eide, John Regehr (PLDI '11)*
- Explores atypical combinations of C language features
- Found many bugs in existing compilers
- Key challenge is targeting program generation to more likely reveal potential problems
- Trivia: the program size that helped discover most bugs was around 82KB

Fuzzing (for various purposes) is a vibrant research area these days

Checking for Correctness

1. Make sure the compiler is not behaving unexpectedly: crashing, diverging, etc.
2. Compare generated programs:
   - Compile with multiple compilers/versions or optimization levels and see if the results differ (see e.g., CSmith)
   - Change a program in some controlled manner (V. Le at al. PLDI'14, OOPSLA'15) http://web.cs.ucdavis.edu/~su/emi-project/

Test Oracles

1. Detect crashing or hanging compilers

2. Differential Testing

- Try multiple compilers – do they result in the same outcome
- Cross-compiler, Cross-optimization, Cross-version
- Compare the results of the compiled program or the code itself

3. Metamorphic testing

- Change the input program in a way that you expect the output to change.
- E.g., if the program compiles f(x) { return x } then it should compile g(x) { return 2*x }
- But more often, these relations are equivalences, even for a particular input, e.g., f(x) { return x + 2 } and g() { return 4 } are equivalent when testing f(2) and g().

Example from EMI (PLDI'14)

```
struct tiny { char c; char d; char e; };
void foo(struct tiny x) {
  if (x.c != 1) abort();
  if (x.e != 1) abort();
}
int main() {
  struct tiny s;
  s.c = 1; s.d = 1; s.e = 1;
  foo(s);
  return 0;
}
```

A bug in the LLVM optimizer causes this miscompilation. The developers believe that the Global Value Numbering (GVN) optimization turns the struct initialization into a single 32-bit load. Subsequently, the Scalar Replacement of Aggregates (SRoA) optimization decides that the 32-bit load is undefined behavior, as it reads past the end of the struct, and thus does not emit the correct instructions to initialize the struct. The developer who fixed the issue characterized it as

> *"... very, very concerning when I got to the root cause, and very annoying to fix."*

Figure 2: Reduced version of the code in Figure 1b for bug reporting. (`http://llvm.org/bugs/show_bug.cgi?id=14972`)

```
int a, b, c, d, e;
int main() {
  for (b = 4; b > -30; b--)
    for (; c;)
      for (;;) {
        e = a > 2147483647 - b;
        if (d) break;
      }
  return 0;
}
```

Partial Redundancy Elimination (PRE) detects the expression "`e2147483647 - b`" as loop invariant. Loop Invariant Motion (LIM) tries to move it up from the innermost loop to the body of the outermost loop. Unfortunately, this optimization is problematic, as GCC then detects a signed overflow in the program's optimized version and this (incorrect) belief of the existence of undefined behavior causes the compiler to generate non-terminating code (and the bogus warning at `-O2`).

Figure 3: GCC miscompiles this program to an infinite loop instead of immediately terminating with no output. (`http://gcc.gnu.org/bugzilla/show_bug.cgi?id=58731`)

Example Metamorphic Relations

Paper	Metamorphic relation	How to construct metamorphic relations
Tao et al. [114]	Equivalence relation	Constructing equivalent expressions, assignment blocks, and submodules
Le et al. [72]	Equivalence relation under a given set of test inputs	Deleting code in the dead regions under the set of test inputs
Le et al. [73]	Equivalence relation under a given set of test inputs	Deleting and inserting code in the dead regions under the set of test inputs
Sun et al. [111]	Equivalence relation under a given set of test inputs	Inserting code in both the live and dead regions by synthesizing valid semantic-preserving code snippets under the set of test inputs
Donaldson and Lascu [43]	Equivalence relation	Injecting dead code into test programs
Nakamura and Ishiura [89]	Equivalence relation	Applying a set of equivalent transformation rules on test programs
Donaldson et al. [42]	Equivalence relation	Applying a set of (essentially) semantics-preserving transformations on high-value graphics shaders
Samet [99–101]	Equivalence relation	Converting a source program and the object program into an intermediate representation, respectively

Table from A Survey of Compiler Testing, CSUR February 2020

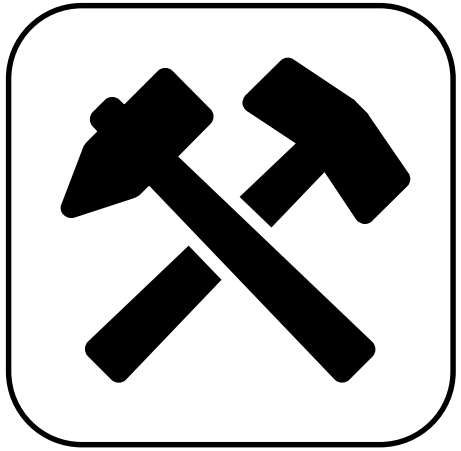
Fuzz Testing of Probabilistic Programming Systems

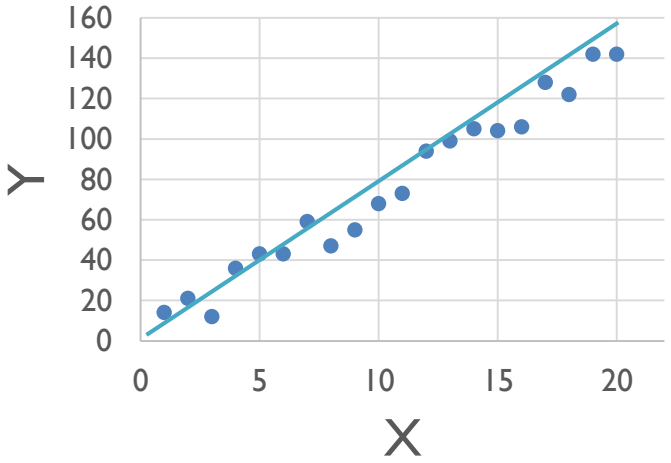
Probabilistic Programming System

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b, 1.0));
posterior w;
posterior b;
```





Probabilistic Model

Probabilistic Programming System

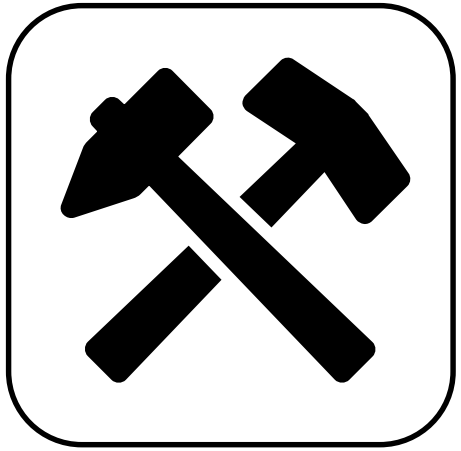
Estimates the values for the parameters

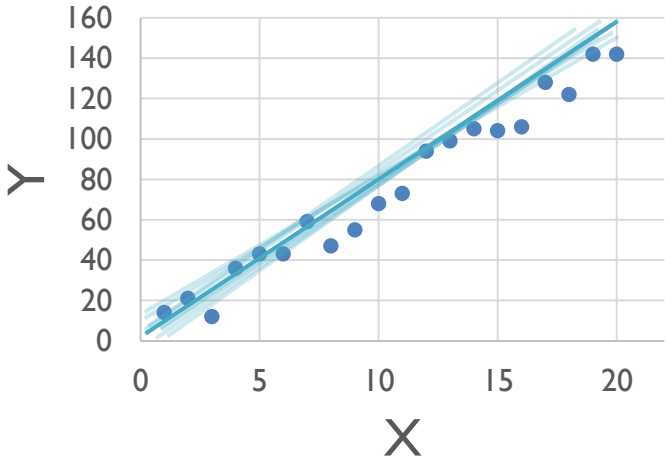
Probabilistic Programming System

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b,1.0));
posterior w;
posterior b;
```





Probabilistic Model

Probabilistic Programming System

Estimates the values for the parameters

In principle, computing probabilities is simple!

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

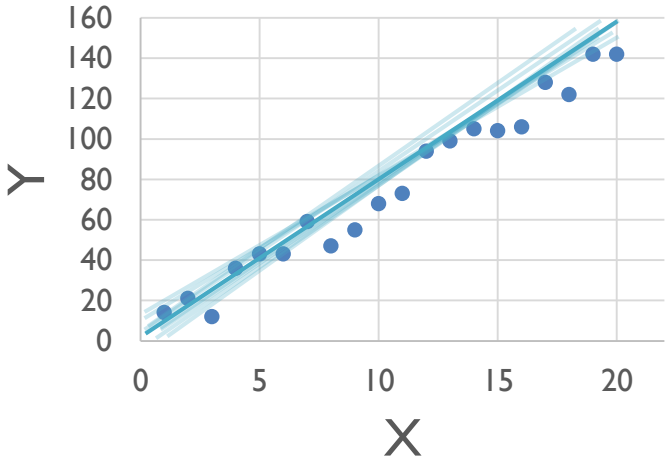
```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x +b, 1.0));
posterior w;
posterior b;
```

Probabilistic Model

$$\frac{P(x|\theta)\cdot P(\theta)}{P(x)} = P(\theta|x)$$

Bayes Rule!

Probabilistic Programming System



Estimates the values for the parameters

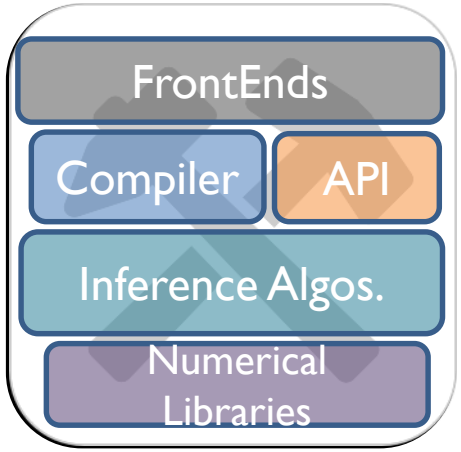
In practice, it is complicated!

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

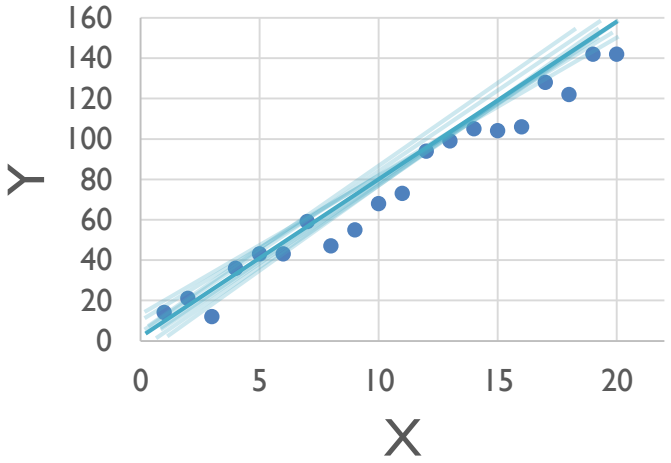
Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x+ b, 1.0));
posterior w;
posterior b;
```

Probabilistic Model



Probabilistic Programming System

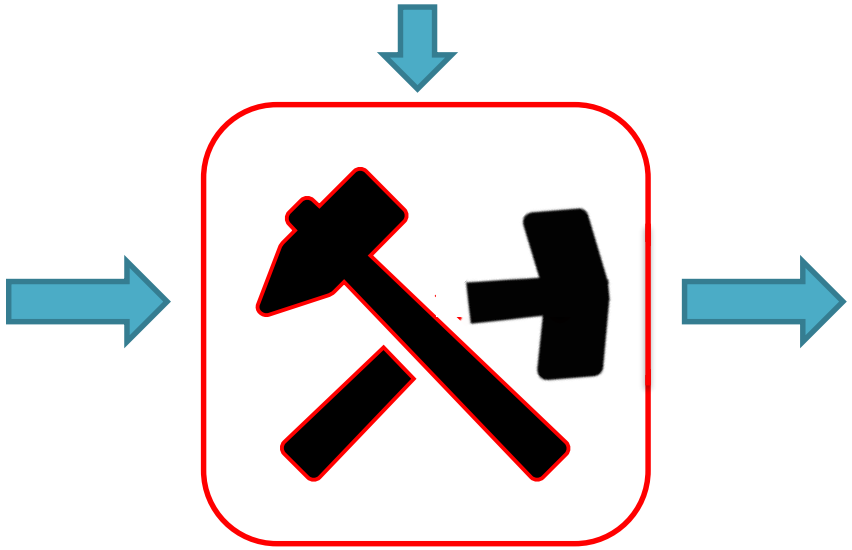


What if the system has bugs?

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x+b,1.0));
posterior w;
posterior b;
```



What if the system has bugs?

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x+b,1.0));
posterior w;
posterior b;
```

```
InvalidArgumentError:
Tensor had NaN values
```

Runtime Errors

What if the system has bugs?

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x+b,1.0));
posterior w;
posterior b;
```

```
w : nan
b : inf
```

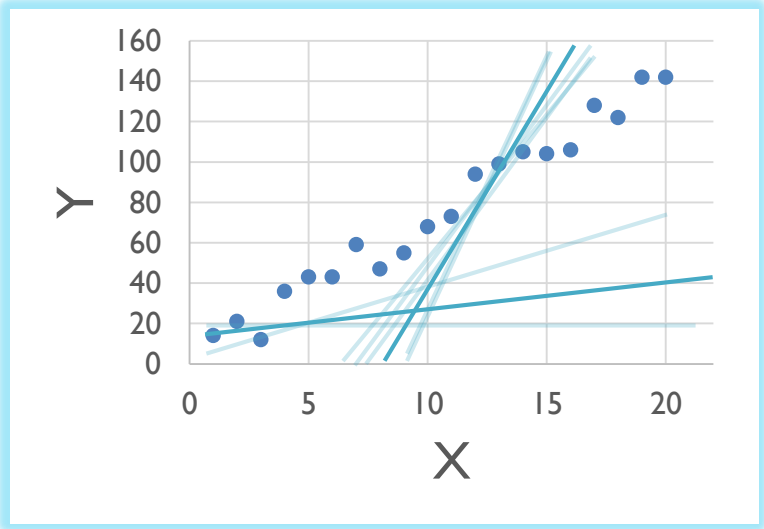
Numerical Errors

What if the system has bugs?

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, ...., ];
```

Data

```
w ~ Normal(1 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x+b,1.0));
posterior w;
posterior b;
```



Inaccurate outputs

How can we automatically test such systems?

ProbFuzz

Automates Testing of Probabilistic Programming Systems

Leverages domain knowledge for code and data generation

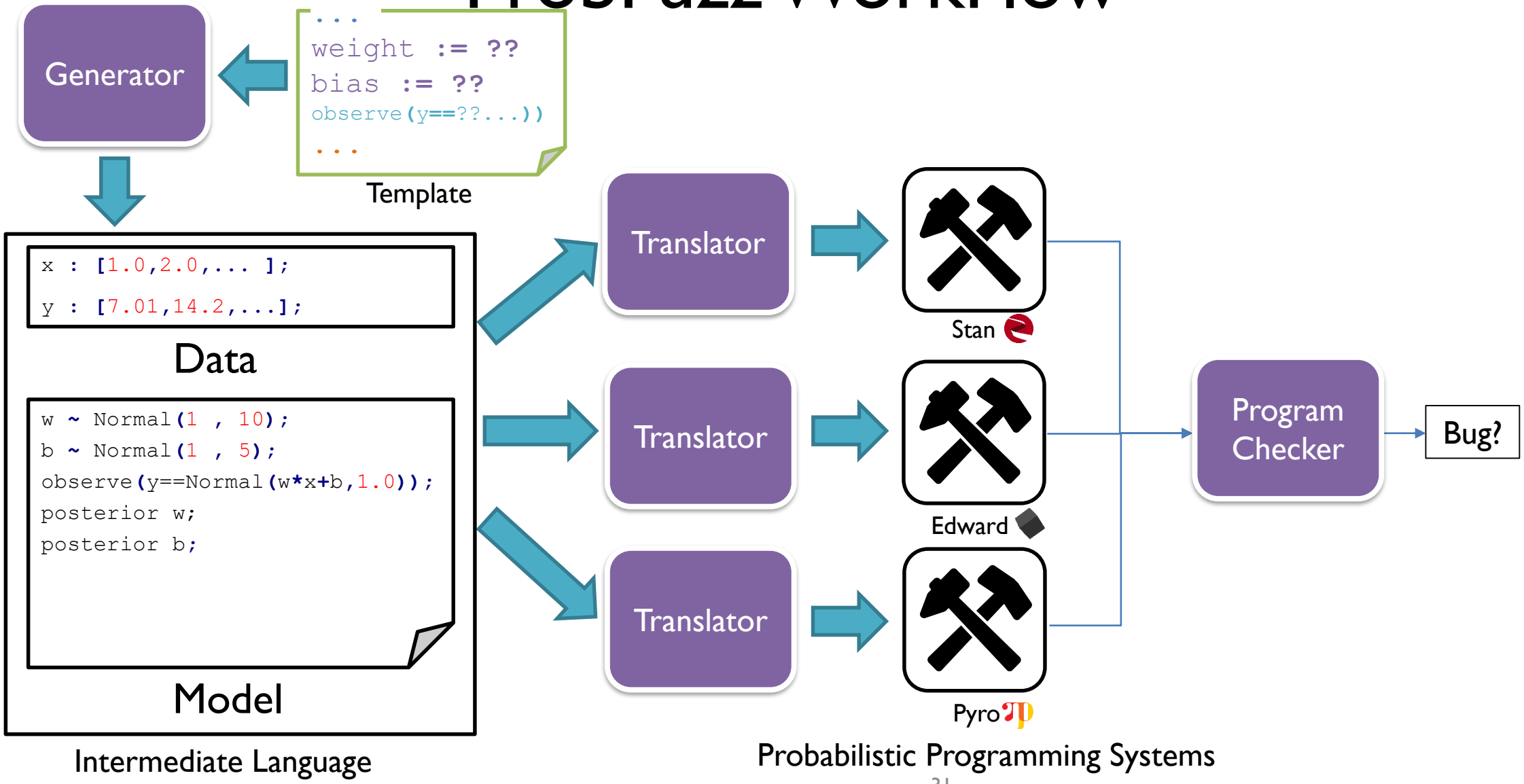
Partial programs (Templates) for targeted program generation

Common language for representing probabilistic programs

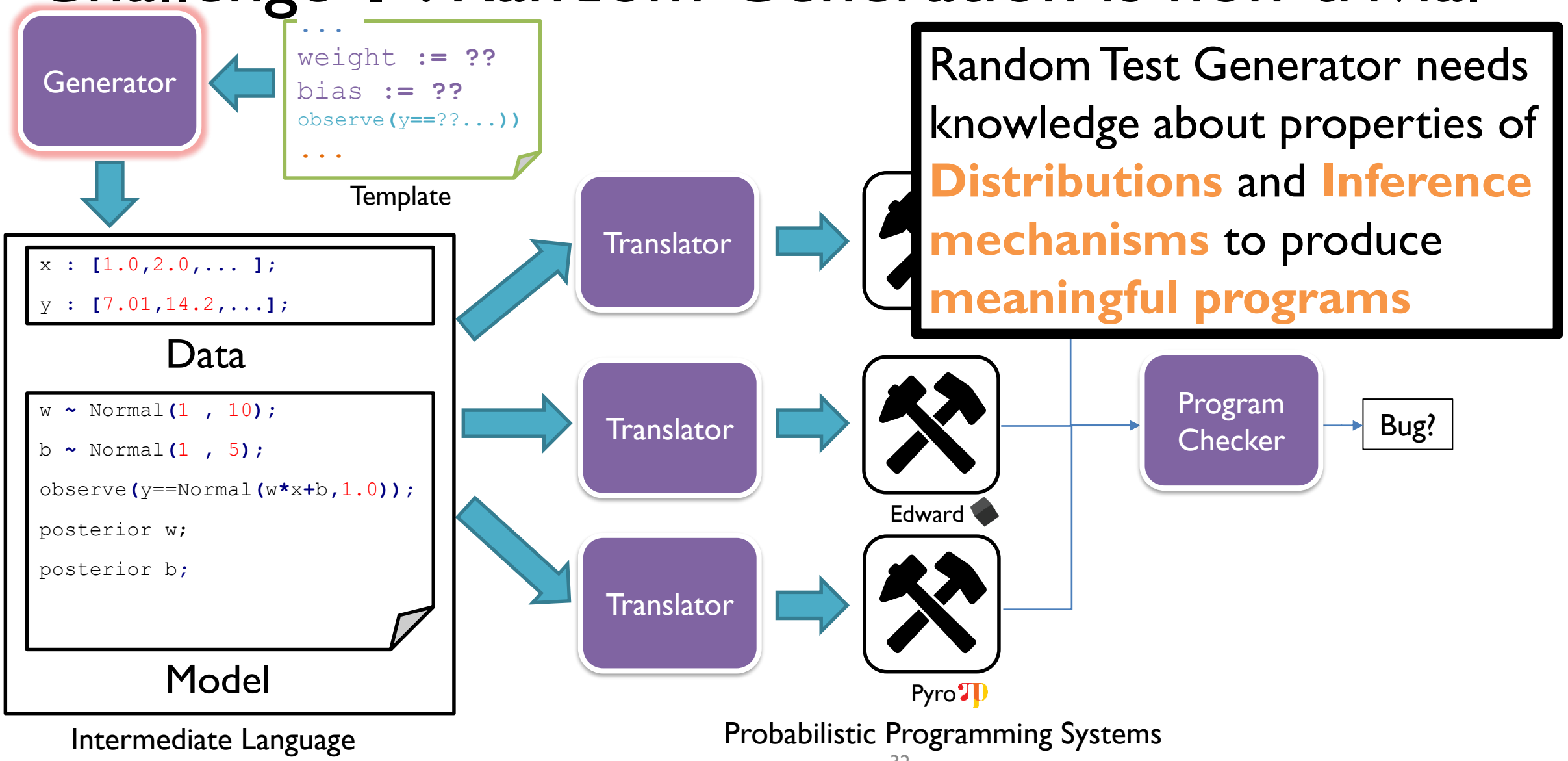
Differential testing with accuracy reasoning

ProbFuzz has found more than 50 bugs in 3 PP Systems: Stan, Edward, and Pyro and underlying frameworks: Tensorflow and Pytorch

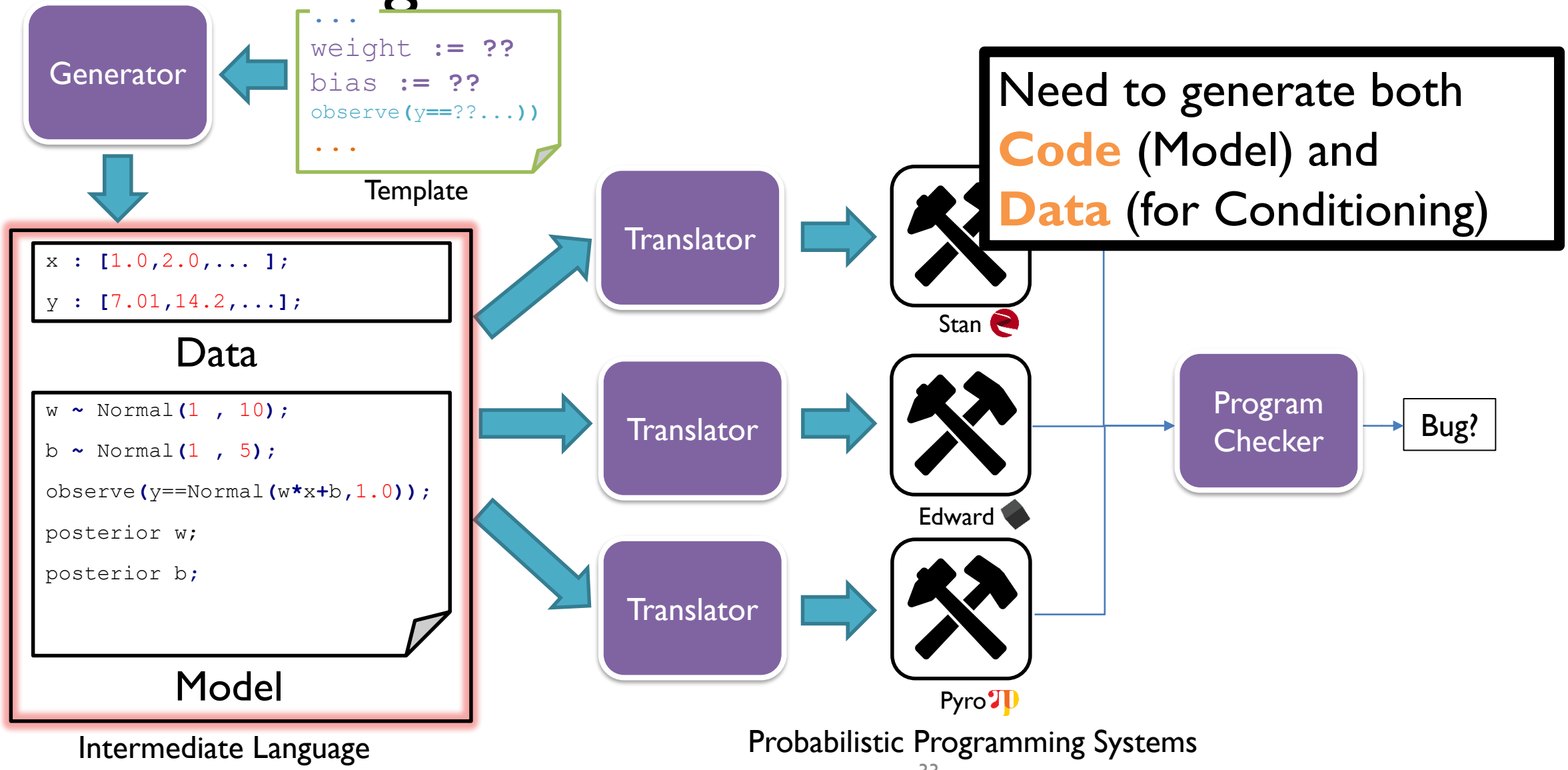
ProbFuzz WorkFlow



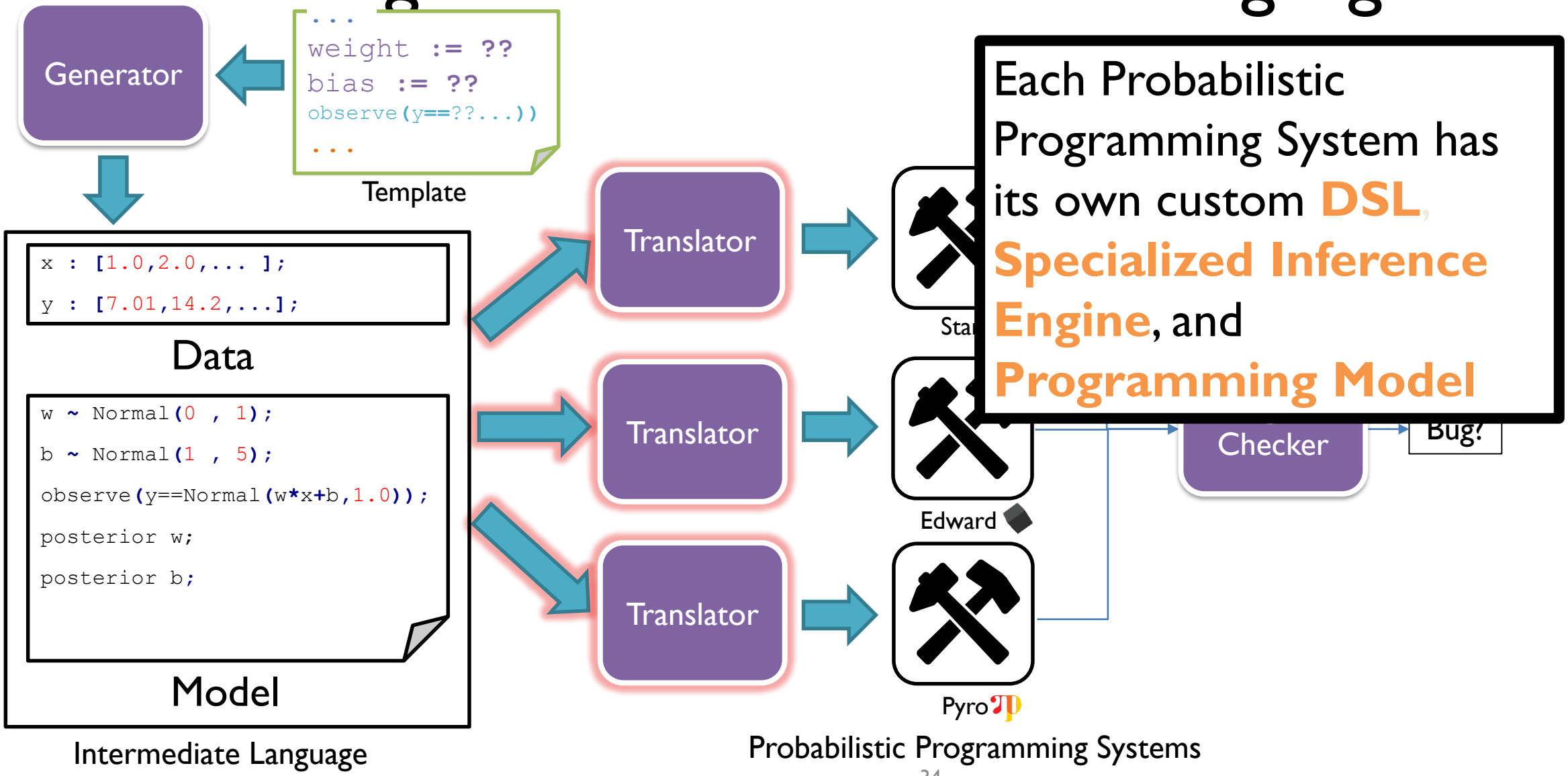
Challenge 1 : Random Generation is non-trivial



Challenge 2: Code and Data Generation



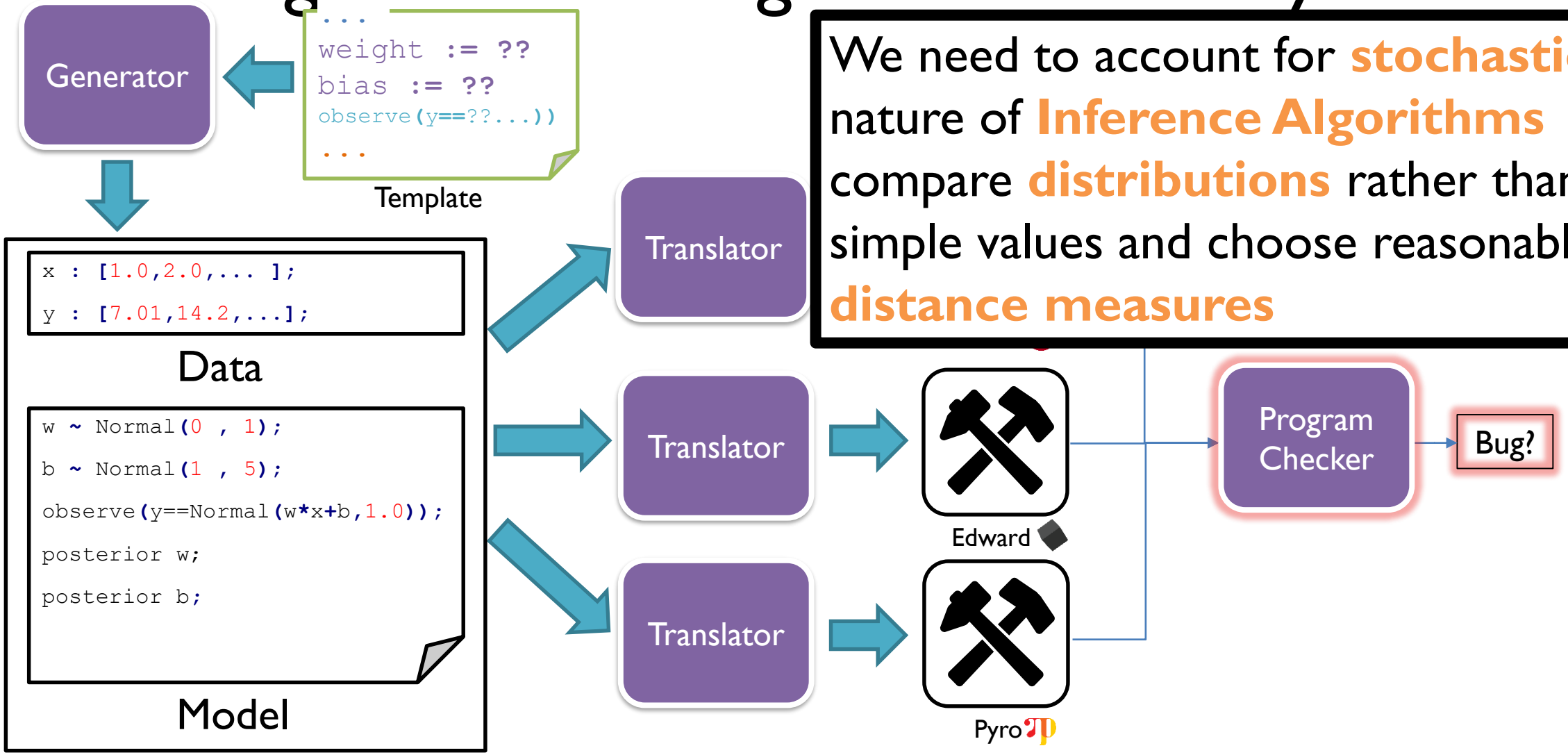
Challenge 3: Lack of Common Language



Each Probabilistic Programming System has its own custom **DSL**, **Specialized Inference Engine**, and **Programming Model**

Challenge 4: Reasoning about Accuracy is Hard

We need to account for **stochastic** nature of **Inference Algorithms** compare **distributions** rather than simple values and choose reasonable **distance measures**



ProbFuzz Intermediate Language

General Language for Probabilistic Programs with mathematical operations, distributions, conditionals and loops

`Observe` statement for conditioning

Key aspect: **Holes** (??) for Distributions and Parameters

Query statements for estimating parameters

Can be used for representing several models like **Linear Regression**, **Bayesian Networks**, **Hierarchical Models**, **Hidden Markov Models**, etc.

Generation of Concrete Programs

Data

Prior

Model

Queries

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
posterior weight;
posterior bias;
```

Template for Linear Regression

Specification of Distributions and legal ranges

Generator

```
x: [1.0, 2.0, 3.0, … ]
w: 5.0
b: 0.5
y: [5.51, 10.48, 15.0, … ]
weight := Normal(0.0,1.0)
bias := Gamma(1.0,1.0)
observe(y==Normal(weight*x+bias,1.5))
posterior weight;
posterior bias;
```

Generated Instances of Code and Data

Generation of Concrete Programs

Data

Prior

Model

Querie
s

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
posterior weight;
posterior bias;
```

Template for Linear Regression

Specification of Distributions and legal ranges

Generator

```
x: [2.1, 6.9, 111.0, … ]
w: 1.0
b: 0.1
y: [2.21, 7.1, 111.11, … ]
weight := Gamma(21.0,7.0)
bias := Gamma(1.0,1.0)
observe(y==Gamma(weight*x+bias,1.5))
posterior weight;
posterior bias;
```

Generated Instances of Code and Data

Generation of Concrete Programs

Data

Prior

Model

Queries

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
posterior weight;
posterior bias;
```

Template for Linear Regression

Specification of Distributions and legal ranges

Generator

```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.39, … ]
weight := Beta(21.0,7.0)
bias := Exponential(2.0)
observe(y==Normal(weight*x+bias,1.5))
posterior weight;
posterior bias;
```

Generated Instances of Code and Data

Generating Data

```
x: Float[10]
w: Float
b: Float
y: w*x + b
```

Generator

```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.43, … ]
```

Template for Linear Regression

Generated Datasets

Generating Data with Noise

```
x: Float[10]
w: Float
b: Float
e: Normal(0.1,0.1)[10]
y: w*x + b + e
```

Template for Linear Regression


```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.88, 157.1, 0.43, … ]
```

Generated Datasets

Generating Distributions and Parameters

Priors

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
```


```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.43, … ]
```

Can we fill in any random distribution here?

No! We need to do dependence analysis and interval analysis

Generating Distributions and Parameters

Specification of Distributions and legal ranges

Distribution	Support	μ	σ
Normal	(-∞, ∞)	(-∞, ∞)	(0, ∞)
…	…	…	…

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
```

Generator

```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.43, … ]
weight := ??
bias := ??
```

The distribution support must match the dataset **y**

Generating Distributions and Parameters

Specification of Distributions and legal ranges

Distribution	Support	μ	σ
Normal	$(-\infty, \infty)$	$(-\infty, \infty)$	$(0, \infty)$
…	…	…	…

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
```

Generator

```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.43, … ]
weight := ??
bias := ??
observe(y==Normal(weight*x+bias,1.5))
```

The distribution support must match the dataset **y**

The other missing parameter must be a positive value (variance of Normal distribution)

Generating Distributions and Parameters

Specification of Distributions and legal ranges

Distribution	Support	μ	σ
Normal	(-∞, ∞)	(-∞, ∞)	(0, ∞)
…	…	…	…

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
```

Generator

```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.43, … ]
weight := Gamma(21.0,7.0)
bias := Exponential(2.0)
observe(y==Normal(weight*x+bias,1.5))
```

The distribution support must match the dataset **y**

The other missing parameter must be a positive value (variance of Normal distribution)

The priors for weight and bias can have any support

Generating Distributions and Parameters

Specification of Distributions and legal ranges

Distribution	Support	μ	σ
Normal	(-∞, ∞)	(-∞, ∞)	(0, ∞)
…	…	…	…

```
x: Float[10]
w: Float
b: Float
y: w*x + b
weight := ??
bias := ??
observe(y==??(weight*x+bias,??))
posterior weight;
posterior bias;
```

Generator

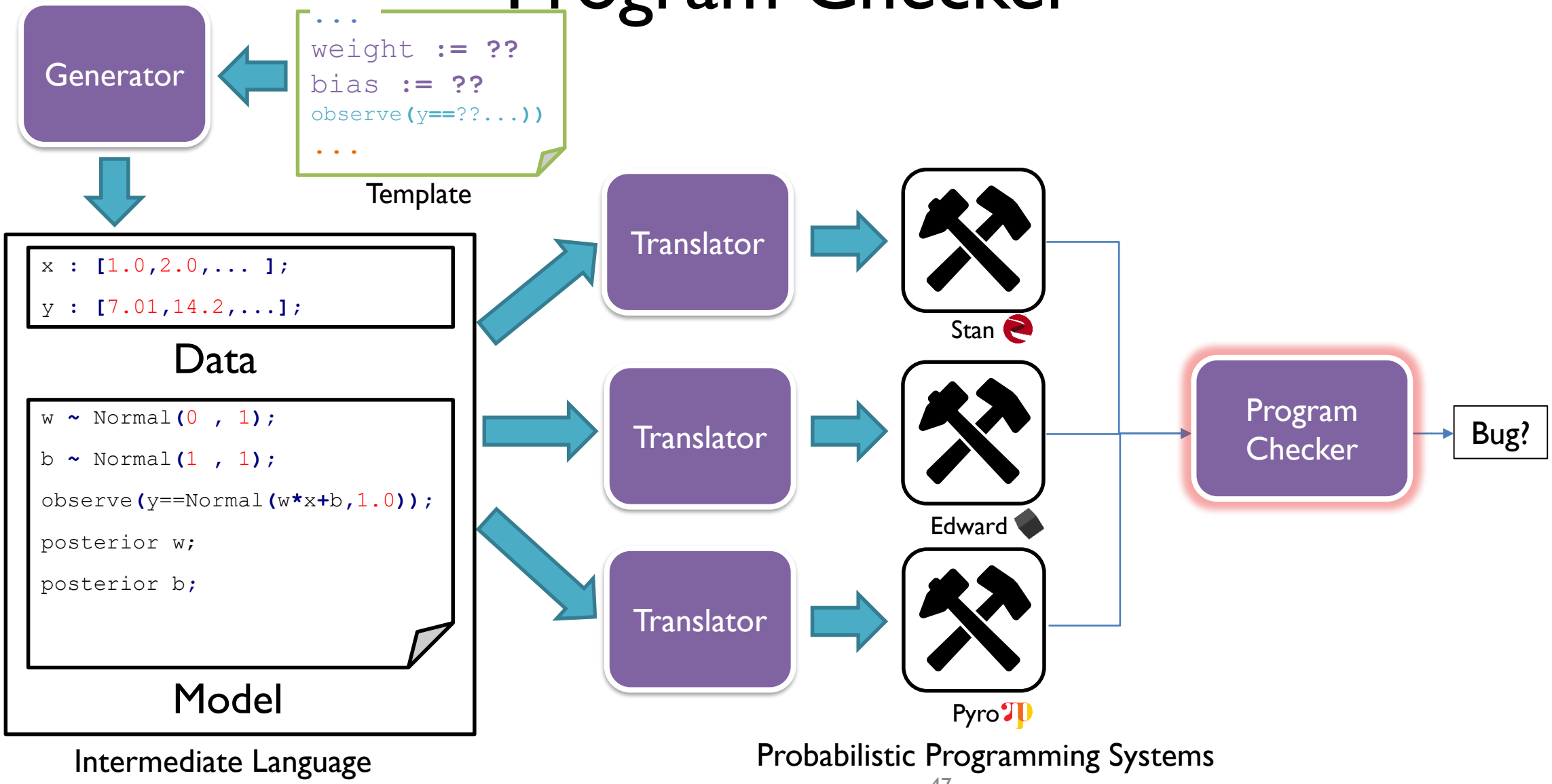
```
x: [-1.0, 52.3, 0.1, … ]
w: 3.0
b: 0.1
y: [-2.89, 157.1, 0.43, … ]
weight := Gamma(21.0,7.0)
bias := Exponential(2.0)
observe(y==Normal(weight*x+bias,1.5))
posterior weight;
posterior bias;
```

The distribution support must match the dataset **y**

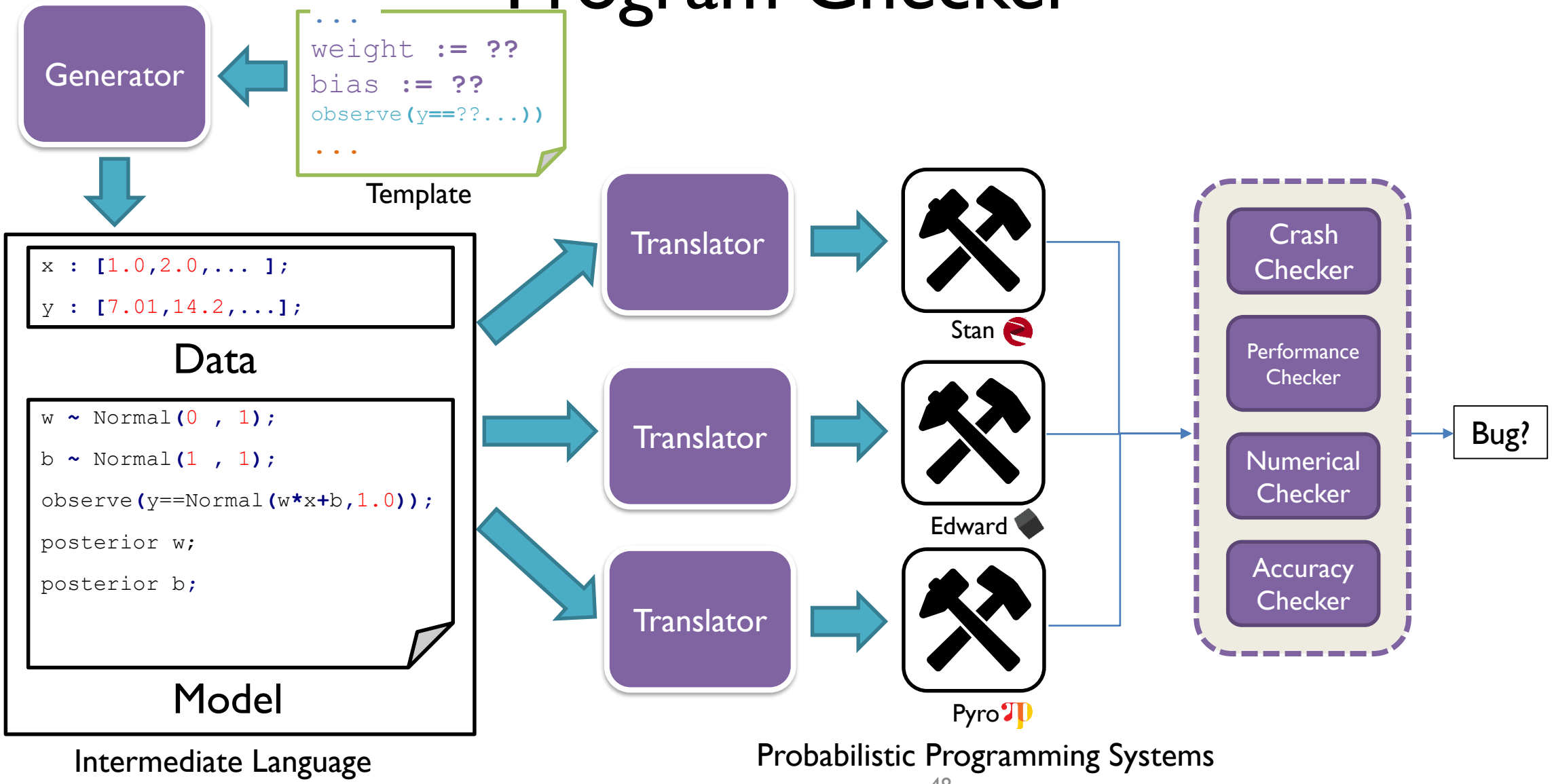
The other missing parameter must be a positive value (variance of Normal distribution)

The priors for weight and bias can have any support

Program Checker



Program Checker



Reasoning about Accuracy

Accuracy Checker implements various kinds of checks:

- Comparison with Reference Result, e.g.: Symbolic Inference: PSI*, Hakaru+
- Comparison amongst the Probabilistic Programming Systems

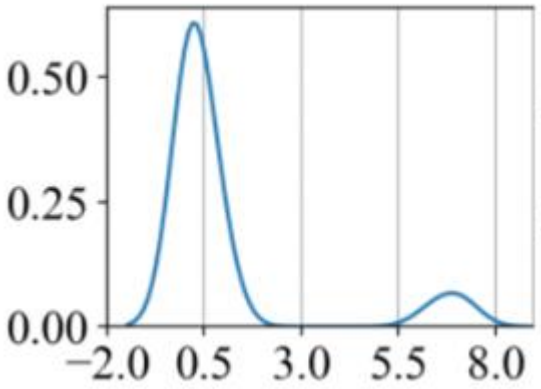
We used several metrics for comparing posterior distributions, e.g :

SMAPE for linear regression $SMAPE(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n) = \frac{1}{n}\Sigma_{\{i=1\}}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$

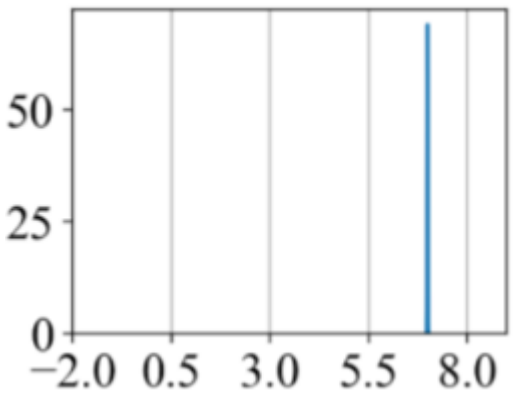
```
x : [1.0,2.0,... ];
y : [7.01,14.02,...];
```

```
w ~ Gamma(97.5, 86.2);
b ~ Beta(44.0, 44.0);
observe(y==Normal(w*x,b));
posterior w;
```

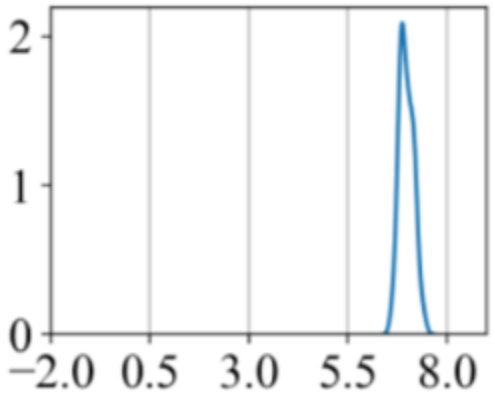
Linear Regression Model



Stan Result



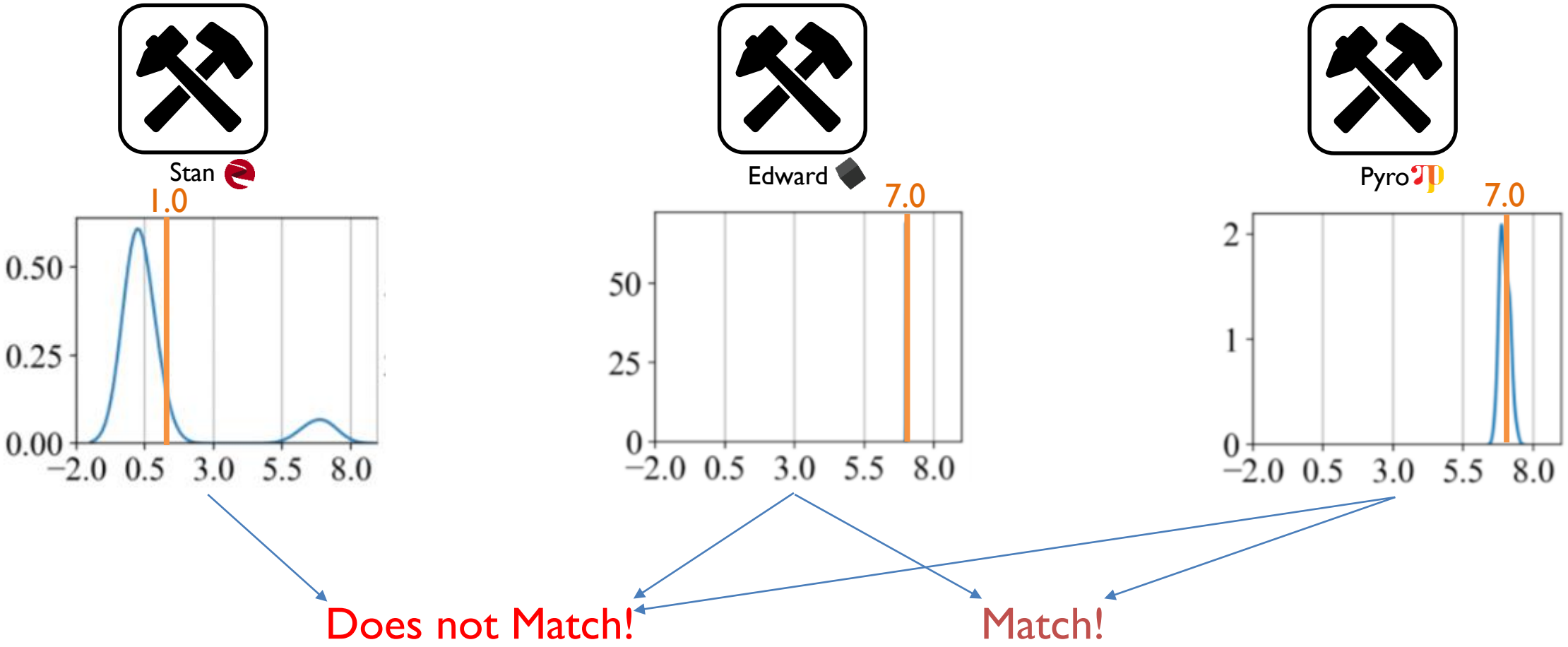
Edward Result



Pyro Result

*CAV '16, +FLOPS '16

Accuracy Checker



Compiler Fuzzing

Key idea: Generate many programs and compile them. The compiler should still be able to produce (ideally correct) code for these programs

Questions:

- How to generate programs?
- How to know they are correct?
- How to identify where the error may be?

RQ 1: New Bugs Discovered by ProbFuzz

Category	Edward	Pyro	Stan	Total
Algorithmic/Accuracy	2	1	2	5
Dimension/Boundary	13	41	0	54
Numerical	0	0	3	3
Language/Translation	1	3	1	5
Total	16	45	6	67

- So far, Developers have **Accepted 51**, Rejected 8, 7 are Pending, and 1 was already fixed
- Out of the 51 accepted bugs, we fixed 50 and 1 reported bug was fixed by developers

Lessons Learned From Studying and Fixing Bugs

- **Dimension/Boundary-Value** bugs are often found in Probabilistic Programming Systems

Lessons Learned From Studying and Fixing Bugs

- **Dimension/Boundary-Value** bugs are often found in Probabilistic Programming Systems

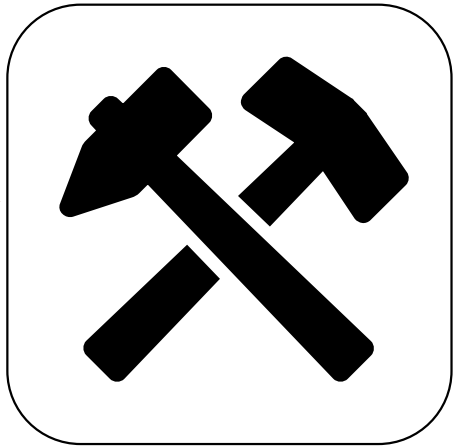
Category	Edward	Pyro	Stan	Total
Algorithmic/Accuracy	2	1	2	5
Dimension/Boundary	13	41	0	54
Numerical	0	0	3	3
Language/Translation	1	3	1	5
Total	16	45	6	67

Example Dimension/Boundary Value Bug : Pyro

```
x : [ 1.1, 3.2, … ]
y : [ 2.1, 60.0, …  ]
```

Data

```
w ~ Bernoulli(2.0)
b ~ Normal(1.0, 2.0)
observe(y==Normal(x*w + b,1.0))
expectation(w);
expectation(b);
```



Pyro

Leads to incorrect results instead of catching the error

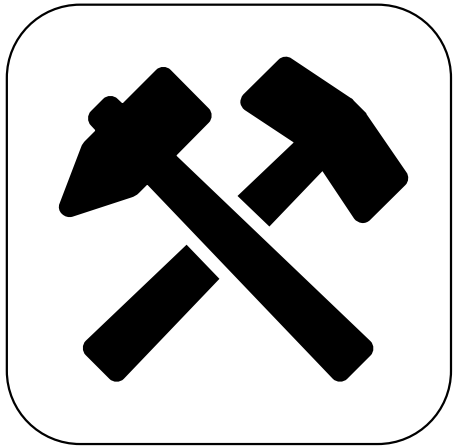
Simple fix : Add a check for boundary conditions : $0.0 < \theta < 1.0$

Example Dimension/Boundary Value Bug 2 : Pyro

```
x : [ 23.79, 81.77, ....]
y : [ 24.79, 82.77, ....]
```

Data

```
w ~ …
b ~ …
observe(Normal(x*w + b, 1.0), y)
posterior(w);
posterior(b);
```



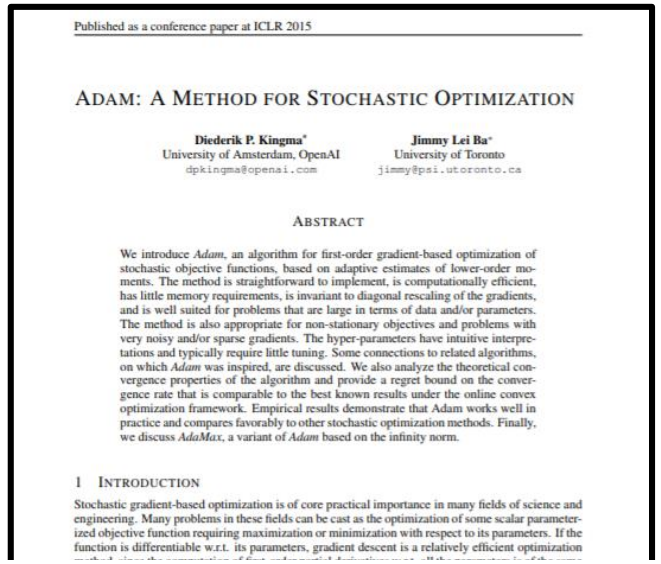
Pyro

ZeroDivisionError:
float division by zero

Program crashes even though model is correct

What went wrong?

- Missing check for β parameter in AdamOptimizer
- Crash when β contains 1
- Fix requires knowledge of the theory involved
- The bug was actually found in **Pytorch**!



Published as a conference paper at ICLR 2015

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

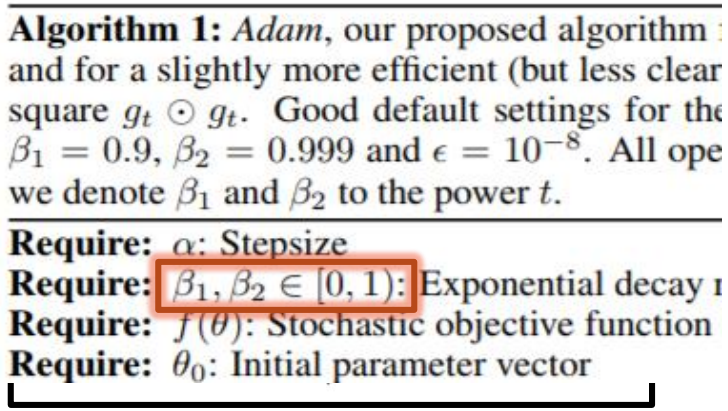
Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

1 INTRODUCTION

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. Many problems in these fields can be cast as the optimization of some scalar parameterized objective function requiring maximization or minimization with respect to its parameters. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization



Algorithm 1: *Adam*, our proposed algorithm
and for a slightly more efficient (but less clear
square $g_t \odot g_t$. Good default settings for the
$\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All ope
we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay r
Require: $f(\theta)$: Stochastic objective function
Require: θ_0: Initial parameter vector

Lessons Learned From Studying and Fixing Bugs

- **Dimension/Boundary-Value** bugs are often found in Probabilistic Programming Systems

- Fixes might **extend across the boundaries** of individual systems

Fixing Dimension/boundary bugs: Pyro-Pytorch

- Found missing boundary checks for several distributions and algorithms
- Proposed Fix in Pytorch for a global validation flag

  "@lazypanda1 Sure, I think an argument validate_args=False make sense. You can even add a generic test to tests/test_distributions.py"

  The fix was accepted

- Following this fix, Pyro introduced "**`enable_validation(True)`**" flag for boundary value checking!

Pyro now includes a global flag for checking all distributions and parameters

0.2.0

dafa1dd

0.2.0

fritzo released this on Apr 24 · 243 commits to dev since this release

Validation

Model validation is now available via three toggles:

```
pyro.enable_validation()
pyro.infer.enable_validation()
# Turns on validation for PyTorch distributions.
pyro.distributions.enable_validation()
```

Negative Response: Edward-Tensorflow

- Found missing boundary checks for distributions in Edward leading to NaNs
- Proposed Fix in Edward for a global validation flag

  ”That's an interesting suggestion. I can see that as a potentially useful utility. Can you raise this in TensorFlow? It seems like a feature request that should be considered upstream in TF Distributions. ***@dev”***

- Proposed Fix in Tensorflow

  “While we appreciate the sometimes inconvenience of the current design, we're reluctant to introduce a global mechanism for setting validate_args. This means we can't accept this PR.“

  Fix was rejected!

Lessons Learned From Studying and Fixing Bugs

- **Dimension/Boundary-Value** bugs are often found in Probabilistic Programming Systems
- Fixes might extend across the boundaries of individual systems
- Analyzing **accuracy** problems is **hard** and requires extensive knowledge about probabilistic inference and the underlying systems

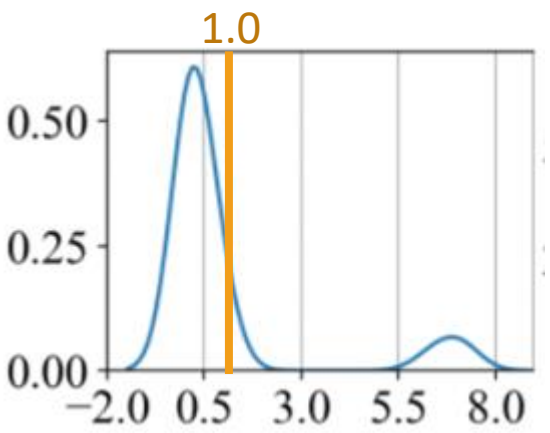
Reasoning about Accuracy

- Bug is reproduced for any value of the parameters for Beta
- Bounding the parameter ***p*** to (0,1) – **support of Beta** - produces correct result

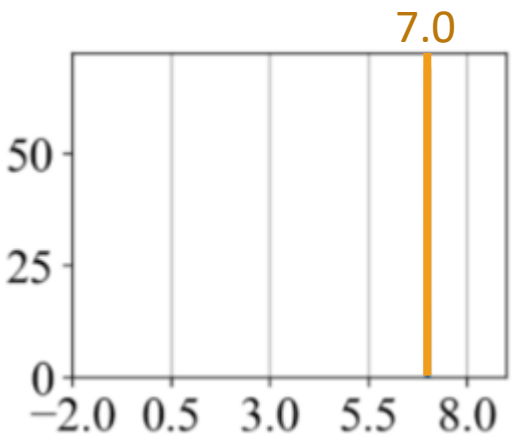
```
x : [1.0,2.0,... ];
y : [7.01,14.02,...];
```

```
w ~ Gamma(97.5, 86.2);
p ~ Beta(44.0, 44.0);
observe(y==Normal(w*x,p));
posterior w;
```

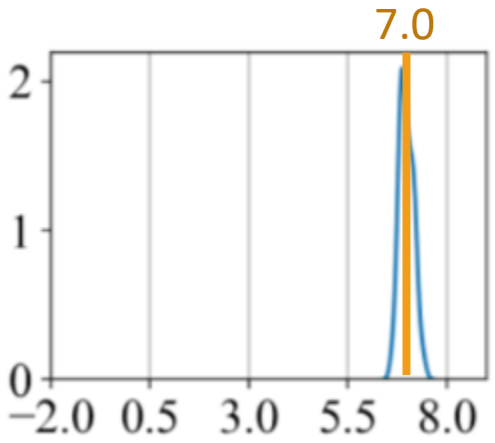
Linear Regression Model



Stan Result



Edward Result



Pyro Result

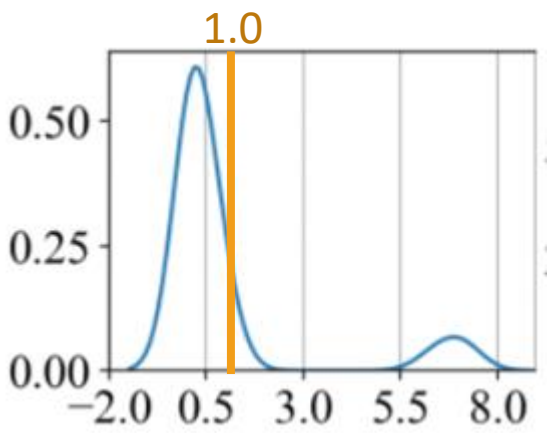
Reasoning about Accuracy

- Bug is reproduced for any value of the parameters for Beta
- Bounding the parameter ***p*** to (0,1) – **support of Beta** - produces correct result
- Stan actually interprets the sampling statement differently, which runs into errors when the bounds are not defined

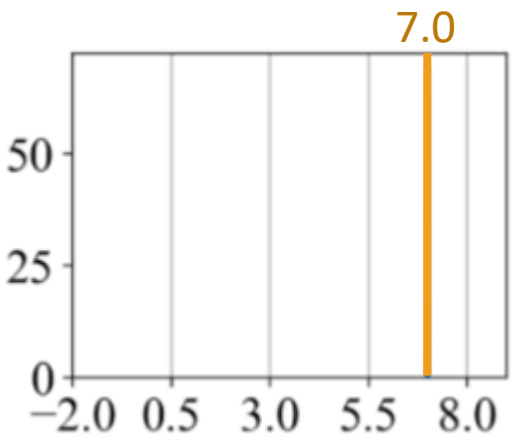
```
x : [1.0,2.0,... ];
y : [7.01,14.02,...];
```

```
w ~ Gamma(97.5, 86.2);
p ~ Beta(44.0, 44.0);
observe(y==Normal(w*x,p));
posterior w;
```

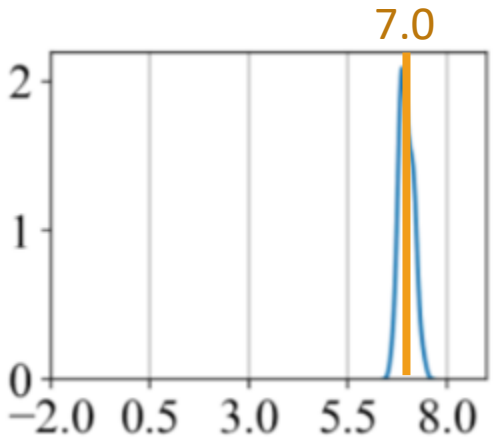
Linear Regression Model



Stan Result



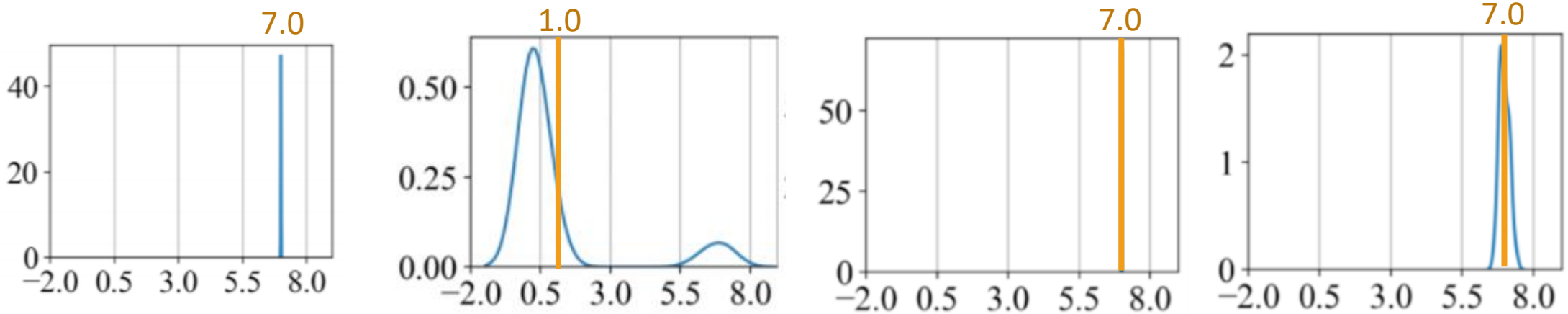
Edward Result



Pyro Result

Reasoning about Accuracy

- Bug is reproduced for any value of the parameters for Beta
- Bounding the parameter ***p*** to (0,1) – **support of Beta** - produces correct result
- Stan actually interprets the sampling statement differently, which runs into errors when the bounds are not defined
- BUGS produces correct result!



BUGS Result | Stan Result | Edward Result | Pyro Result

NEXT: HANDLING RANDOMNESS IN MACHINE LEARNING TESTS