

# Introduction to Probabilistic Programming

Xin Zhang

Peking University

# What the course is about

- An introductory course to an advanced topic
  - Still an advanced course
- The first course on probabilistic programming in China
- Offered to both domestic and international students
  - Part of the international graduate program at PKU
- Course materials in English

# What the course is not about

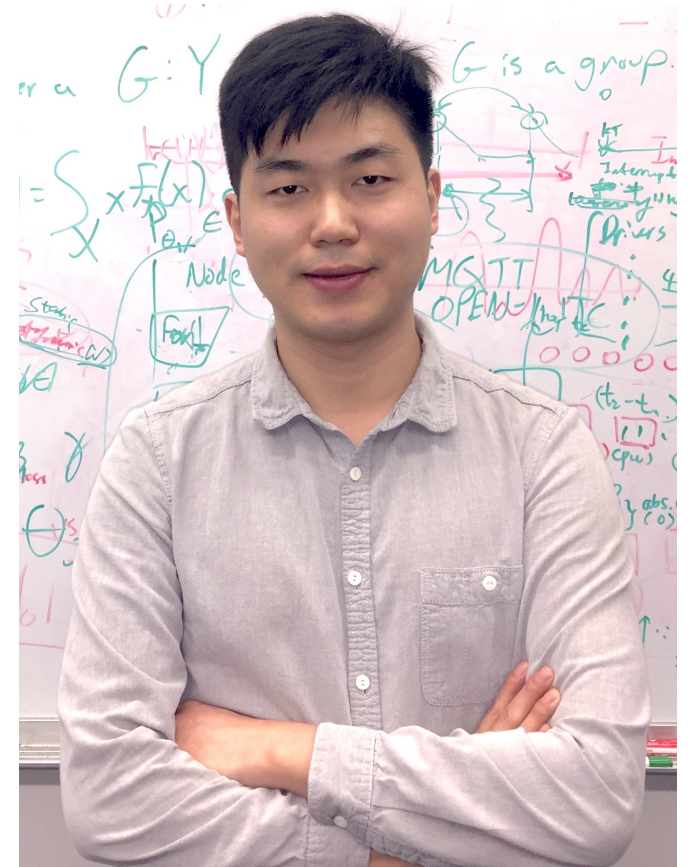
- Learning how to program
- Learning how to speak in English
  - If I end up teaching in English, its fine if you ask/answer questions in Chinese
  - You can ask me to clarify in Chinese
  - You're encouraged to communicate in English

# What needs to be decided

- Shall I teach in English?
  - I will if at least one student needs that
  - Will send out a survey later this week

# Instructor

- Xin Zhang, Assistant Professor, Computer Science
- Research Areas: Programming Languages, Software Engineering, Assured Artificial Intelligence
- Website: <http://xinpl.github.io>
- E-mail: [xin@pku.edu.cn](mailto:xin@pku.edu.cn)



# TA

- Junhao Liu, PhD Student
- Programming Languages Lab @ PKU
- Email: [liujunhao@pku.edu.cn](mailto:liujunhao@pku.edu.cn)

# Logistics

- Time:
  - Thursday 9am - 12am (9:00-9:50, 10:10 – 11:00, 11:10 – 12:00)
- Location:
  - Changping Campus: 206 Teaching Building (昌平校区教学楼206)
- Credit hours: 3
- Course Website: <http://xinpl.github.io/courses/probprog/2024/>

# Course WeChat Group



# Course Policy

- Laptops and phones allowed to try example programs
  - Mute them
  - Don't use them to do things that are unrelated to the course
- Academic integrity
  - [http://www.dean.pku.edu.cn/web/student\\_info.php?type=3&id=49](http://www.dean.pku.edu.cn/web/student_info.php?type=3&id=49)
  - 0 tolerance for academic dishonesty and cheating
- Speak out when you don't understand something

# Grading

- Assignments: 30%
- Mid-Term (In-Class Exam): 30%
- Final (In-Class Exam): 40%

**Off-campus students: we will figure out later how to take the exams**

# Past Grades

2021

Grade	Percentage
A+	13.3%
A	33.3%
B+	46.7%
B	6.7%

2022 (Covid)

Grade	Percentage
A+	11.1%
A	22.2%
P	66.7%

# What you will learn

- What probabilistic programming is
- How to write programs in popular languages
- Relevant concepts like graphical models, Bayesian learning, probabilistic inferences
- Theoretical foundations
- Inferences and learning
- Frontiers

# Schedule (Tentative)

- Introduction with WebPPL (Week 1 & 2)
- Relevant backgrounds (Week 3 & 4)
- Theoretical foundations (Week 5 & 6)
- Mid-Term (Week 7)
- Inference (Week 8 & 9)
- Learning (Week 10)
- Probabilistic Logic Programming (Week 11 & 12)
- Advanced Topics (Week 13-15)
- Final (Week 16)

# After This Class

- Hopefully, you can
  - know what graphical models are
  - Write probabilistic programs
  - Have an idea about how probabilistic programming languages work underneath

How would you build an AI?  
What should it be able to do?

# Different Styles of AI

## Five Tribes of Machine Learning

Symbolists

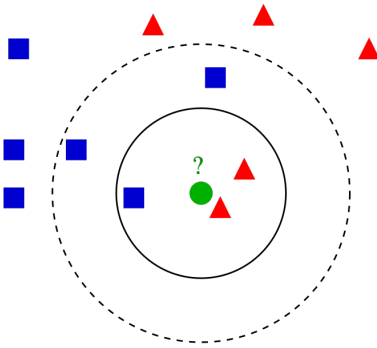
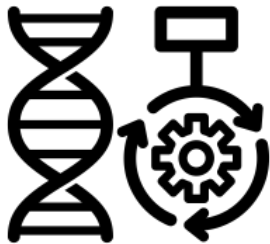
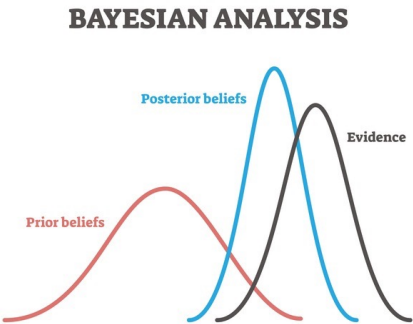
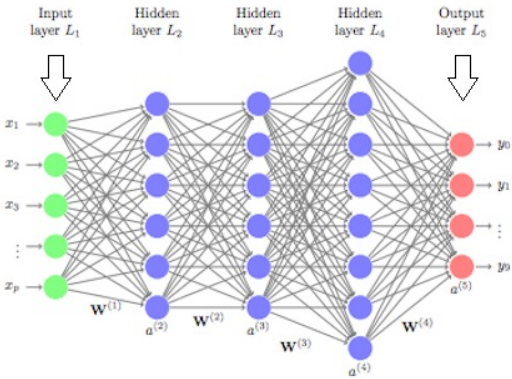
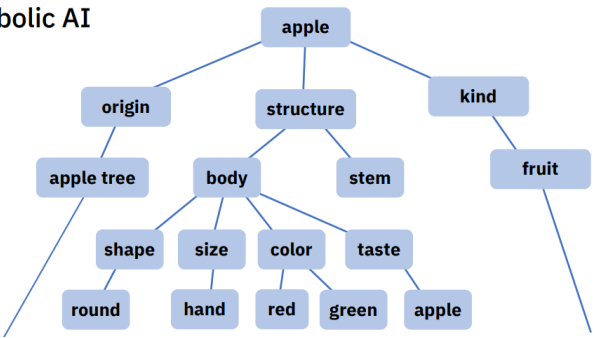
Connectionists

Bayesians

Evolutionaries

Analogizers

Symbolic AI



“The Master Algorithm”, Pedro Domingos



# Different Styles of AI

## Five Tribes of Machine Learning

Symbolists

Connectionists

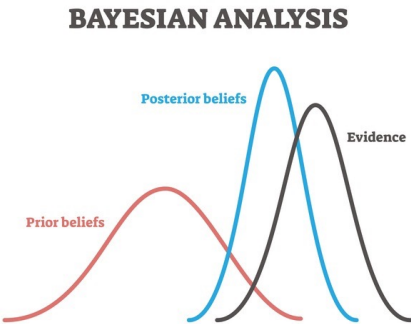
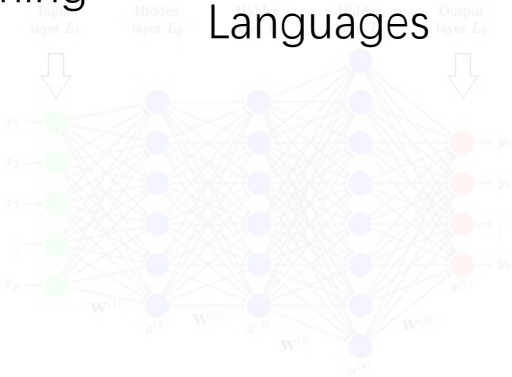
**Bayesians**

Evolutionaries

Analogizers

Probabilistic programming = Programming Languages +

Symbolic AI



“The Master Algorithm”, Pedro Domingos

# Spirit of Probabilistic Programming

- Express your beliefs and uncertainties to generate data
- Adjust the model based on observed data

**Bayesian**

# Spirit of Probabilistic Programming

- Express your beliefs and uncertainties to generate data

If Xiaoming stays up, there is 50% chance he will drink coffee. If he stays up and doesn't drink coffee, he will fall asleep in class.

- Adjust the model based on observed data

Given Xiaoming didn't fall asleep today, how likely did he stay up?

**Bayesian**

# Spirit of Probabilistic Programming

- Express your beliefs and uncertainties to generate data **in programs**
- Adjust the model based on observed data using **general algorithms**

Bayesian

+

**Programming Languages**

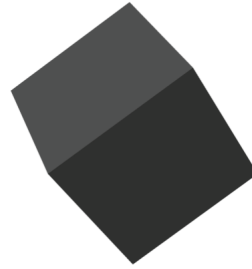
# Old Idea that Resurged Recently

- “High level probabilistic languages have been in use since the earliest versions of FORTRAN and BASIC.”- Semantics of Probabilistic Programs, Dexter Kozen, FOCS 1979
- Resurged due to
  - Novel applications
  - New inference algorithms
  - More computing power



Oxford, UBC, etc.

Edward



Google



Facebook

General-purpose programming languages + probabilistic constructs

**Venture**

MIT

**Omega**

MIT (I was part of it)




Columbia, etc.

**ProbLog**

KU LEUVEN

Wikipedia lists  
55 probabilistic  
programming  
languages!

List of probabilistic programming languages [\[ edit \]](#)



This article **may contain an excessive amount of intricate detail that may interest only a particular audience**. Please help by [spinning off](#) or [relocating](#) any relevant information, and removing excessive detail that may be against [Wikipedia's inclusion policy](#). *(October 2019)* ([Learn how and when to remove this template message](#))

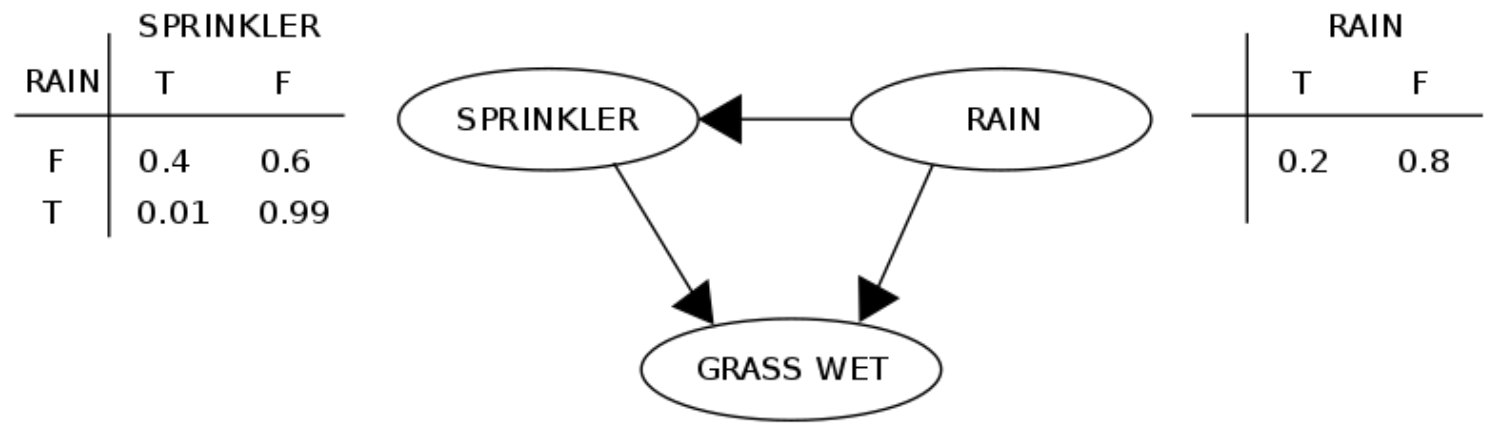
Name ↕	Extends from ↕	Host language ↕
<a href="#">Analytica</a> <sup>[17]</sup>		C++
<a href="#">bayesloop</a> <sup>[18][19]</sup>	Python	Python
<a href="#">CuPPL</a> <sup>[20]</sup>	<a href="#">NOVA</a> <sup>[21]</sup>	
<a href="#">Venture</a> <sup>[22]</sup>	<a href="#">Scheme</a>	C++
<a href="#">Probabilistic-C</a> <sup>[23]</sup>	<a href="#">C</a>	C
<a href="#">Anglican</a> <sup>[24]</sup>	<a href="#">Clojure</a>	Clojure
<a href="#">IBAL</a> <sup>[25]</sup>	<a href="#">OCaml</a>	
<a href="#">BayesDB</a> <sup>[26]</sup>	<a href="#">SQLite</a> , <a href="#">Python</a>	
<a href="#">PRISM</a> <sup>[13]</sup>	<a href="#">B-Prolog</a>	
<a href="#">Infer.NET</a> <sup>[12]</sup>	.NET Framework	.NET Framework
<a href="#">dimple</a> <sup>[10]</sup>	<a href="#">MATLAB</a> , <a href="#">Java</a>	
<a href="#">chimple</a> <sup>[11]</sup>	<a href="#">MATLAB</a> , <a href="#">Java</a>	
<a href="#">BLOG</a> <sup>[27]</sup>	<a href="#">Java</a>	
<a href="#">diff-SAT</a> <sup>[28]</sup>	<a href="#">Answer set programming</a> , <a href="#">SAT (DIMACS CNF)</a>	
<a href="#">PSQL</a> <sup>[29]</sup>	<a href="#">SQL</a>	
<a href="#">BUGS</a> <sup>[14]</sup>		
<a href="#">FACTORIE</a> <sup>[30]</sup>	<a href="#">Scala</a>	Scala
<a href="#">PMTK</a> <sup>[31]</sup>	<a href="#">MATLAB</a>	<a href="#">MATLAB</a>
<a href="#">Alchemy</a> <sup>[32]</sup>	<a href="#">C++</a>	
<a href="#">Dyna</a> <sup>[33]</sup>	<a href="#">Prolog</a>	
<a href="#">Figaro</a> <sup>[34]</sup>	<a href="#">Scala</a>	Scala
<a href="#">Church</a> <sup>[35]</sup>	<a href="#">Scheme</a>	Various: <a href="#">JavaScript</a> , <a href="#">Scheme</a>
<a href="#">ProbLog</a> <sup>[36]</sup>	<a href="#">Prolog</a>	<a href="#">Python</a> , <a href="#">Jython</a>
<a href="#">ProBT</a> <sup>[37]</sup>	<a href="#">C++</a> , <a href="#">Python</a>	
<a href="#">Stan</a> <sup>[15]</sup>		C++
<a href="#">Hakaru</a> <sup>[38]</sup>	<a href="#">Haskell</a>	Haskell
<a href="#">BAli-Phy (software)</a> <sup>[39]</sup>	<a href="#">Haskell</a>	C++

# Probabilistic Programming

- Introduces new machine learning models
- Introduces new programming models



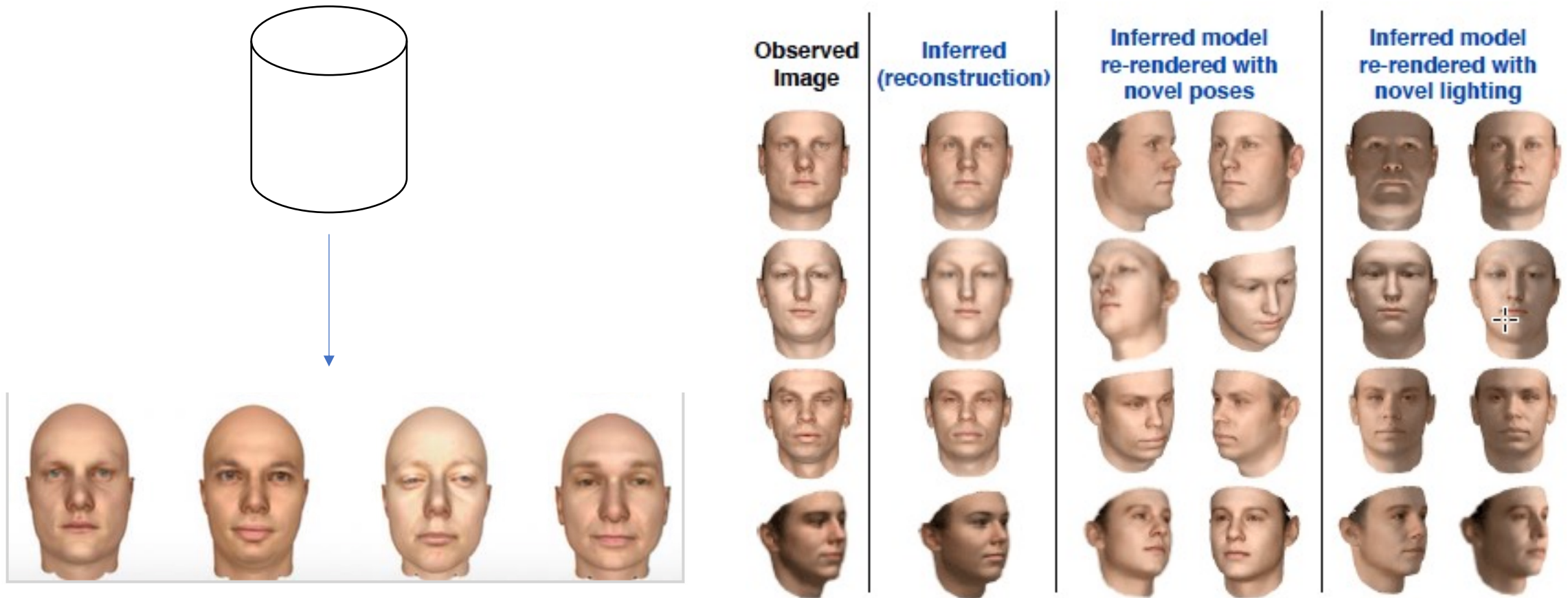
# Old-School Bayesian Models



Given the grass is wet, how likely did it rain?

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

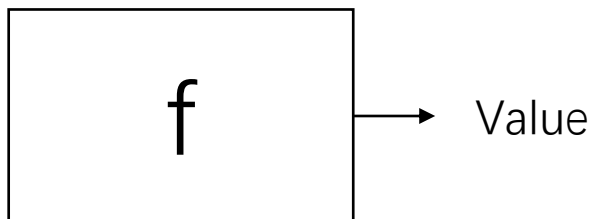
# Probabilistic Programming Models



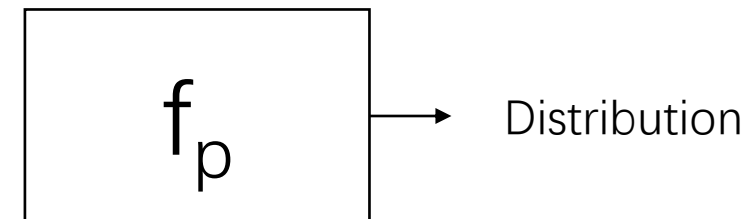
**Inverse Graphics** – 3D faces rendered from 2D images using only 50 lines of PPL code.

Reference: <http://news.mit.edu/2015/better-probabilistic-programming-0413>

# Old-School Languages vs. PPL



The semantics of the program is defined on a given execution.



Built-in support for random variables and operations on them

The semantics of the program is defined on a distribution of executions.

# First Probabilistic Program in WebPPL

- There is a gacha game:
  - 10% chance to get a SSR card every pull.
  - A pull costs 10 RMB.
  - Xiaoming usually pulls 0-9 times a day.
- How many SSRs can he get a day?
- We can model it using a probabilistic program in WebPPL!



Subset of Javascript + Probabilistic Constructs

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- Xiaoming usually pulls 0-9 times a day.

You can install WebPPL locally or directly try it at <http://webppl.org>!

```
var gacha = function(){  
  var pull = Bernoulli({p:0.1});  
  var VS = rangeArray(0,10);  
  var num_pull = Categorical({vs: VS});  
  var num_pull_inst = sample(num_pull);  
  var performPull = function(c){  
    if( c == 0){  
      return 0;  
    }  
    return sample(pull)+performPull(c-1);  
  }  
  return performPull(num_pull_inst)  
};  
  
gacha()
```

Built-in  
distributions/  
random variables

## Let's see it!

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- Xiaoming usually pulls 0-9 times a day.

- What if I want to know the average number of SSR Xiaoming gets?
- What if I want to know what is the chance that Xiaoming gets two SSRs?
- What if I want to know the full distribution?

```
var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c){
    if( c == 0){
      return 0;
    }
    return sample(pull)+performPull(c-1);
  }
  return performPull(num_pull_inst)
};

gacha()
```

Built-in  
distributions/  
random variables

Well, you can do it manually, but WebPPL has built-in support for these!

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- Xiaoming usually pulls 0-9 times a day.

- What if I want to know the average number of SSR Xiaoming gets?
- What if I want to know what is the chance that Xiaoming gets two SSRs?
- What if I want to know the full distribution?

```
var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c){
    if( c == 0){
      return 0;
    }
    return sample(pull)+performPull(c-1);
  }
  return performPull(num_pull_inst)
};
```

```
gacha()
```

```
var gacha_model = Infer({model: gacha})
```

```
display(expectation(gacha_model))
display(Math.exp(gacha_model.score(3)))
viz(gacha_model)
```

Built-in  
distributions/  
random variables

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- Xiaoming usually pulls 0-9 times a day.

“Black magic” behind this function invocation. It performs **Marginal Inference**, which creates a map from values to probabilities.

```
var gacha = function(){  
  var pull = Bernoulli({p:0.1});  
  var VS = rangeArray(0,10);  
  var num_pull = Categorical({vs: VS});  
  var num_pull_inst = sample(num_pull);  
  var performPull = function(c){  
    if( c == 0){  
      return 0;  
    }  
    return sample(pull)+performPull(c-1);  
  }  
  return performPull(num_pull_inst)  
};
```

Built-in  
distributions/  
random variables

```
var gacha_model = Infer({model: gacha})
```

```
display(expectation(gacha_model))  
display(Math.exp(gacha_model.score(3)))  
viz(gacha_model)
```



# More on Marginal Probabilities

- We can enumerate all possible worlds and calculate their probabilities.
- Marginal probability of  $x = X$  is defined as

$$P(x = X) = \sum_{w \in \{w | x=X \text{ in } w\}} P(w)$$

We will talk about how to calculate them later in the course.

Let's make the problem more interesting

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- **If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.**
- Xiaoming usually pulls 0-9 times a day.

```
var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c){
    if( c == 0){
      return 0;
    }
    return sample(pull)+performPull(c-1);
  }
  return performPull(num_pull_inst)
};
```

```
gacha()
```

```
var gacha_model = Infer({model: gacha})
```

```
display(expectation(gacha_model))
display(Math.exp(gacha_model.score(3)))
viz(gacha_model)
```

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- **If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.**
- Xiaoming usually pulls 0-9 times a day.

- What if I want to know the average number of SSR Xiaoming gets?
- What if I want to know what is the chance that Xiaoming gets three SSRs?
- What if I want to know the full distribution?

```
var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return 0;
    }
    if (num_no_ssr == 5){
      return 1 + performPull(c-1, 0);
    }
    else{
      var cp = sample(pull);
      if (cp)
        return 1 + performPull(c-1, 0);
      else
        return 0 + performPull(c-1, num_no_ssr+1);
    }
  }
  return performPull(num_pull_inst, 0)
};
```

# Informal Semantics of Probabilistic Programs

- Probabilistic programs define a distribution of executions
  - The randomness comes from random variables
  - Given an execution  $e$ , let  $V$  be the set of random variables that are evaluated in  $e$ , we use  $v(e)$  to denote the value a random variable  $v$  takes in  $e$ , then we have

$$P(e) = \prod_{v \in V} P(v = v(e))$$

- We will introduce formal semantics later in the course!

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- **If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.**
- Xiaoming usually pulls 0-9 times a day.

Hey! What you assume seems quite reasonable. But Xiaoming never gets more than two SSRs a day!



- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day.
- **Xiaoming never gets more than 2 SSRs a day.**

This describes the outcome rather than the process. How should we modify the program?

**New Language Construct!**

```
var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return 0;
    }
    if (num_no_ssr == 5){
      return 1 + performPull(c-1, 0);
    }
    else{
      var cp = sample(pull);
      if (cp)
        return 1 + performPull(c-1, 0);
      else
        return 0 + performPull(c-1, num_no_ssr+1);
    }
  }
  return performPull(num_pull_inst);
};
```

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day.
- **Xiaoming never gets more than 2 SSRs a day.**

This describes the outcome rather than the process. How should we modify the program?

**New Language Construct!**

```
var gacha = function(){
  ...
  return performPull(num_pull_inst)
};
```

```
var gacha1 = function(){
  var num_ssrs = gacha()
  condition(num_ssrs <= 2)
  return num_ssrs
}
```

Filter out executions that do not satisfy external knowledge, and computes a conditional distribution

```
var gacha_model = Infer({model: gacha1})
```

```
display(expectation(gacha_model))
```

```
display(Math.exp(gacha_model.score(3)))
```

```
viz(gacha_model)
```



# More on Condition

```
var d = sample(Categorical({ps: [0.2, 0.3, 0.5], vs: [1,2,3]}))
```

```
condition(d != 3)
```

Value of d	Probability
1	0.2
2	0.3
3	0.5

# More on Condition

```
var d = sample(Categorical({ps: [0.2, 0.3, 0.5], vs: [1,2,3]}))
```

```
condition(d != 3)
```

Value of d	Probability
1	0.2
2	0.3
3	0.5

# More on Condition

```
var d = sample(Categorical({ps: [0.2, 0.3, 0.5], vs: [1,2,3]}))
```

```
condition(d != 3)
```

Value of d	Probability
1	$0.2 / (0.2+0.3)$
2	$0.3 / (0.2+.3)$
3	0.5

# More on Condition

```
var d = sample(Categorical({ps: [0.2, 0.3, 0.5], vs: [1,2,3]}))
```

```
condition(d != 3)
```

Value of d	Probability
1	0.4
2	0.6
3	0.5

# More on Condition

- Recall our informal semantics

$$P(e) = \prod_{v \in V} P(v = v(e))$$

- We use  $P(e|c)$  to denote the probability of an execution  $e$  after a condition statement  $c$  is added to the program, and we use  $c(e)$  to represent whether  $c$  holds on  $e$ , then

$$P(e|c) = \begin{cases} 0, & \text{if } c(e) = \text{false} \\ \frac{P(e)}{\sum_{e' \in \{e'' | c(e'')\}} P(e')}, & \text{if } c(e) = \text{true} \end{cases}$$

# More on Condition

- It might be obvious, but **condition** does construct a conditional distribution:

$$P(e|c) = \frac{P(e \cap c)}{P(c)} = \frac{P(e \cap c)}{\sum_{e' \in E} P(c \cap e')} = \frac{P(c|e) * P(e)}{\sum_{e' \in E} P(c|e') * P(e')} = \begin{cases} 0, & \text{if } c(e) = \text{false} \\ \frac{P(e)}{\sum_{e' \in \{e'' | c(e'')\}} P(e')}, & \text{if } c(e) = \text{true} \end{cases}$$

# Main Probabilistic Constructs in WebPPL

- Built-in random variables and **sample** together describe a random process in a constructive way
- **Condition** provides a way to incorporate external knowledge about the output

They together provide a mechanism to do Bayesian learning

What you believe + What you observe

# A Simple Bayesian Inference Example

- A disease can happen to 1% of the population
- A test method has the following property:

	Tested Positive	Tested Negative
Actually Positive	90%	10%
Actually Negative	10%	90%

- Xiaoming is tested positive. How likely does he carry the disease?



# A Simple Bayesian Inference Example

- We can calculate the probability using Bayes' theorem

$$\begin{aligned}P(\text{positive} | \text{tested positive}) &= P(\text{positive}) \times \frac{P(\text{tested positive} | \text{positive})}{P(\text{tested positive})} \\&= 0.01 \times \frac{0.9}{0.01 * 0.9 + 0.99 * 0.1} \\&= 0.083\end{aligned}$$

# A Simple Bayesian Inference Example

- The probabilistic program is as follow:

```
var medic_test = function(){  
  var positive = sample(Bernoulli({p:0.01}))  
  var test_positive = function(pg){  
    if(pg){  
      return sample(Bernoulli({p:0.9}))  
    }  
    else  
      return !sample(Bernoulli({p:0.9}))  
  }  
  condition(test_positive(positive))  
  return positive  
}  
  
var m = Infer({model:medic_test})  
display(Math.exp(m.score(true)))  
viz(m)
```

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- **20% of players believe the rate is not as advertised, but only 8%.**
- **To test if the assumption is true, Xiaoming pulled 20 times, and got 4 SSRs.**

What is the chance that the rate is actually 8%?

Try to calculate it using Bayes' theorem!



- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- **20% of players believe the rate is not as advertised, but only 8%.**
- **To test if the assumption is true, Xiaoming pulled 20 times, and got 4 SSRs.**

In WebPPL, it is easy!

```
var gacha = function(){
  var cheated = sample(Bernoulli({p:0.2}));
  var pull = function(){
    if (cheated){
      return Bernoulli({p:0.08});
    }
    else
      return Bernoulli({p:0.1});
  }
  var num_pull_inst = 20;

  var performPull = function(c, num_no_ssr){
    ...
  }
  var num_ssrs = performPull(num_pull_inst, 0)
  condition(num_ssrs == 4)
  return cheated;
};

var gacha_model = Infer({model: gacha})

display(expectation(gacha_model))
```

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- **When spending over 1000 RMB, for every new pull, there is a chance that the bank would call Xiaoming.**
- **Suppose the chance is  $(x-1000)/1000$ , and Xiaoming got called and then stopped, how many SSRs has he pulled so far?**

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- **When spending over 1000 RMB, for every new pull, there is a chance that the bank would call Xiaoming.**
- **Suppose the chance is  $(x-1000)/1000$ , and Xiaoming got called and then stopped, how many SSRs has he pulled so far?**

Can you express this problem using a conventional graphical model like a Bayesian network?

```
var gacha = function(){
  ...
  var VS = rangeArray(0,1000);
  var num_pull = Categorical({vs: VS});
  ...
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return [0,false];
    }
    var cost = (num_pull_inst - c)*10;
    if(cost > 2000)
      return [0, true]
    if(cost > 1000)
      if(sample(Bernoulli({p: (cost - 1000)/1000.0}))) {
        return [0, true]
      }
    ...
  }
  var pull_result = performPull(num_pull_inst, 0)
  condition(pull_result[1])
  return pull_result[0]
};
```

# Probabilistic Programming So Far

- Built-in support for random variables
  - **Categorical**, **Bernoulli** ....
  - **Sample**
- A general language to describe the sampling process
  - Subet of Javascript
- The ability to impose conditions on any state
  - **Condition**

# Probabilistic Programming So Far

- Built-in support for random variables
  - **Categorical**, **Bernoulli** ....
  - **Sample**
- A general language to describe the sampling process
  - Subet of Javascript
- The ability to impose conditions on any state
  - **Condition**

A convenient way to express highly complex distributions

**Don't worry about how to calculate it! Just think about what is it!**



That sounds too good to be true.  
There must be a catch here ...



That sounds too good to be true.  
There must be a catch here ...



Well, some of the probabilistic programs can be really slow to run.

People have been working on how to make probabilistic programs run fast, which we will discuss later in the course.

```

var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,1000);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return 0;
    }
    if (num_no_ssr == 5){
      return 1 + performPull(c-1, 0);
    }
    else{
      var cp = sample(pull);
      if (cp)
        return 1 + performPull(c-1, 0);
      else
        return 0 + performPull(c-1, num_no_ssr+1);
    }
  }
  return performPull(num_pull_inst);
};
}

```

```

var gacha1 = function(){
  var num_ssrs = gacha()
  condition(num_ssrs <= 2)
  return num_ssrs
}

```

```
var gacha_model = Infer({model: gacha1})
```

```
display(expectation(gacha_model))
```

```
viz(gacha_model)
```

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- **Xiaoming usually pulls 0-999 times a day.**
- **Xiaoming never gets more than 2 SSRs a day.**

# Next Class

- More about WebPPL
  - We have talked about the core probabilistic constructs
- Representative applications using WebPPL