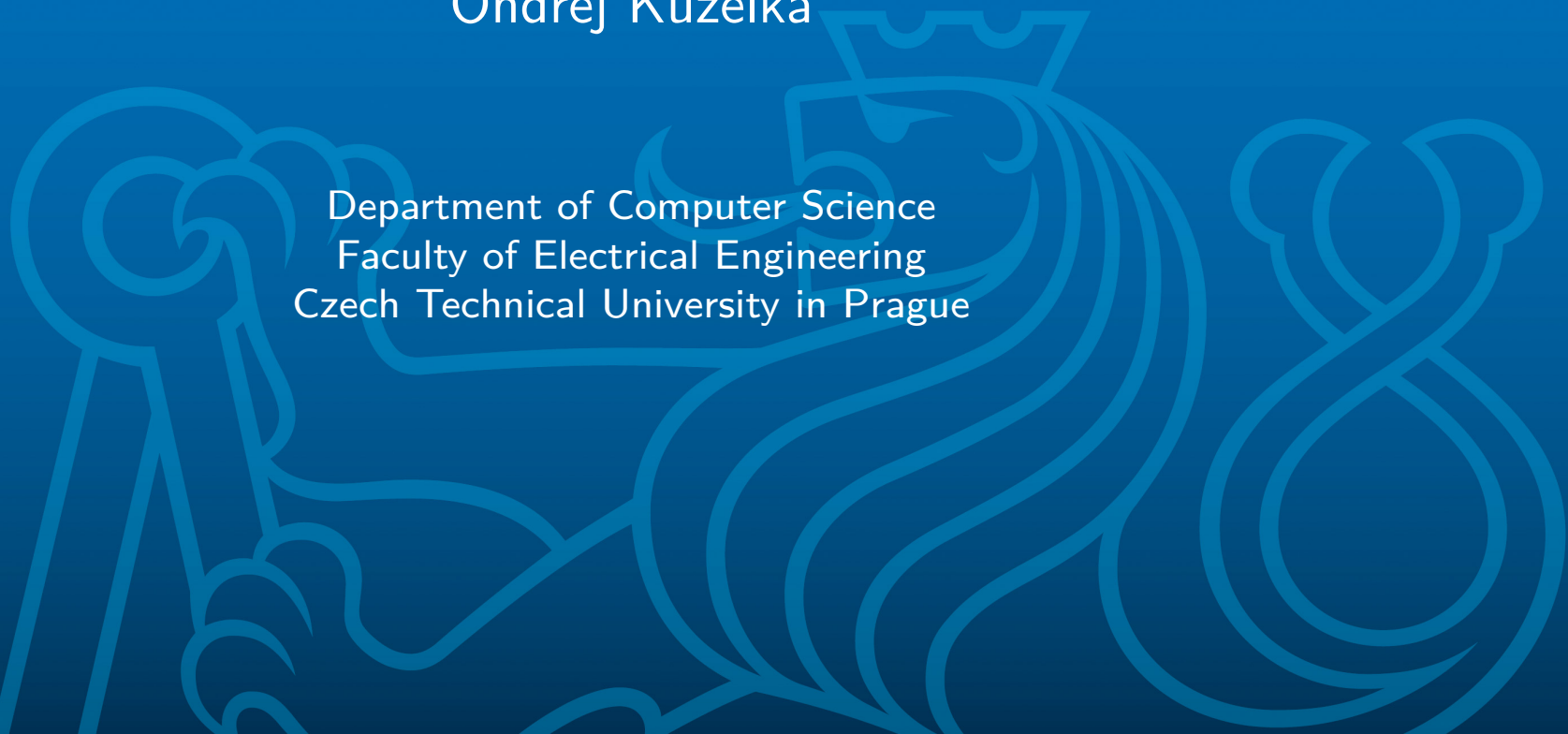
Symbolic Machine Learning

Lecture Slides on Probabilistic (Logic) Programming

Ondřej Kuželka

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague



This Lecture

Today's lecture will be a brief, not very technical introduction to probabilistic programming. In the remaining two lectures, we will focus on one concrete probabilistic logic programming language from KU Leuven, called **ProbLog**.

Why probabilistic programming? Although not yet completely mature, probabilistic programming may well be the next big thing in AI, so it is good to know at least a bit about it.

The purpose of this lecture is to get an intuitive idea of what probabilistic programming is. **The lecture is based on an intro to probabilistic programming by Adrian Sampson (Cornell University), available from http://adriansampson.net/doc/ppl.html.**

What is Probabilistic Programming? (1)

$$\texttt{(Bayesian) Machine learning + Programming languages}$$
$$=$$
$$\texttt{Probabilistic programming}$$

What is Probabilistic Programming? (2)

$$\texttt{(Bayesian) Machine learning + Programming languages}$$
$$=$$
$$\texttt{Probabilistic programming}$$

$$\texttt{Randomized algorithms} \neq \texttt{Probabilistic programming}$$

Digression: Randomized Algorithms

Can you name some randomized algorithms?

Digression: Polynomial Identity Testing

Given: A polynomial $P(x_1, \ldots, x_n)$ represented as an algebraic expression, e.g. $(x-1)^{20} \cdot y + (x-2)^{19}$.

Compute: Is $P(x_1, \ldots, x_n)$ identically equal to 0?

Digression: Schwartz-Zippel Lemma (1)

Theorem

Let $p \in \mathbf{F}[x_1, x_2, \ldots, x_n]$ *be a* **non-zero** *polynomial of degree* $d \geq 0$ *over a field* $\mathbf{F}$*. Let* $\mathbf{S}$ *be a finite subset of* $\mathbf{F}$ *and let* r_1, r_2, $\ldots$, r_n *be selected at random independently and uniformly from* $\mathbf{S}$*. Then*

$$Pr[p(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|\mathbf{S}|}.$$

Q: How can we use this theorem to design a randomized algorithm for checking if a polynomial is non-zero?

Digression: Schwartz-Zippel Lemma (2)

Theorem

Let $p \in \mathbf{F}[x_1, x_2, \ldots, x_n]$ be a **non-zero** *polynomial of degree $d \geq 0$ over a field $\mathbf{F}$. Let $\mathbf{S}$ be a finite subset of $\mathbf{F}$ and let r_1, r_2, $\ldots$, r_n be selected at random independently and uniformly from $\mathbf{S}$. Then*

$$Pr[p(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|\mathbf{S}|}.$$

Q: How can we use this theorem to design a randomized algorithm for checking if a polynomial is non-zero?

Digression: Schwartz-Zippel Lemma (3)

Q: How can we use this theorem to design a randomized algorithm for checking if a polynomial is non-zero?

A: If we evaluate the polynomial on inputs $r_1, r_2, \ldots, r_n$, selected at random independently and uniformly, and if the result is non-zero, we can output "no" (because the polynomial is clearly not identical to zero). The probability of error, i.e. that the polynomial is non-zero and, at the same time, it evaluates to 0 on the inputs $r_1, \ldots, r_n$ is bounded by the theorem. Moreover the probability of making an error can be reduced by running the algorithm multiple times ("amplification by independent trials trick").

Digression: Schwartz-Zippel Lemma (now really digressing too much...)

Q: What if $d > |\mathbf{F}|$?

Hint: Think of Boolean satisfiability!

Where are we?

OK, that was a randomized algorithm. But probabilistic programming is about something a bit different!

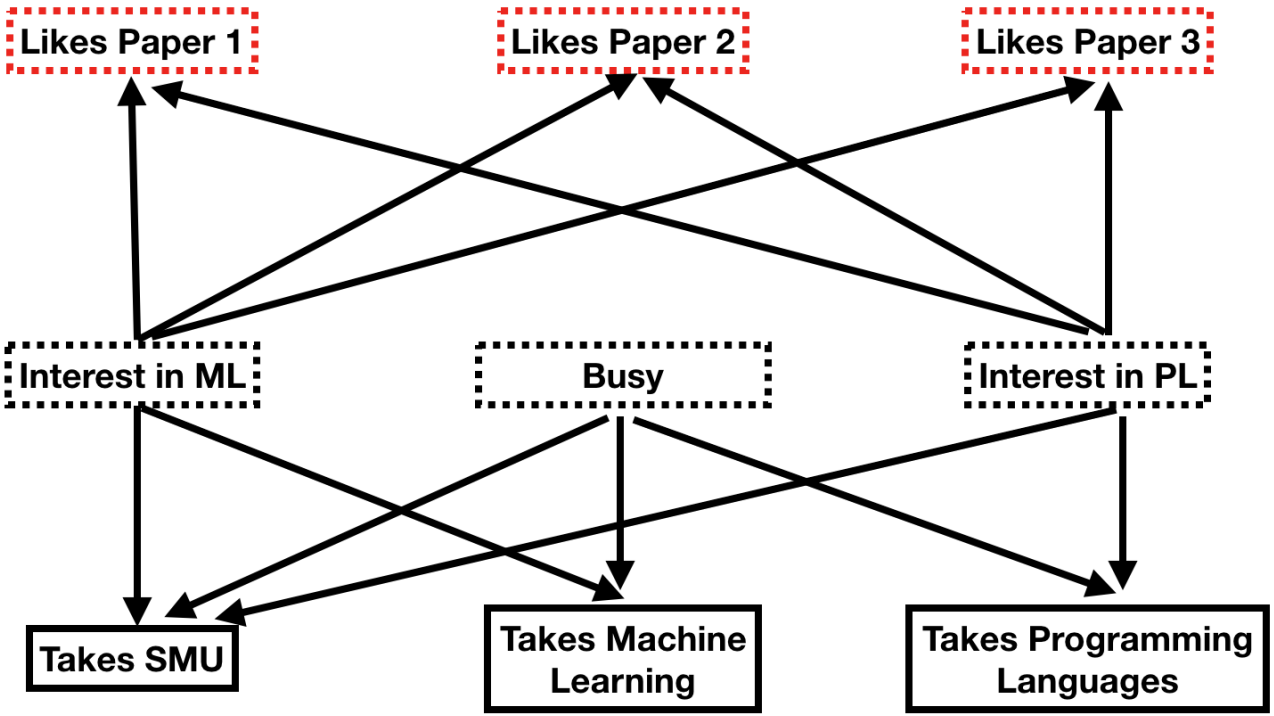
Probabilistic Programming: Informal Definitions

Probabilistic programming is a tool for statistical modeling (similarly as e.g. Bayesian networks).

Another definition from Adrian Sampson's intro: *"A probabilistic programming language is an ordinary programming language with* `rand` *and a great big pile of related tools that help you understand the program's statistical behavior."*

Example: Bayesian Networks in PP (1)

A system for recommending papers. We have 3 papers and all that we can observe about students is which courses they take. The motivations vary, some take courses based on interest, others because of their schedules ("busy"). The observed (solid black), query (red dashed) and latent (black dashed) random variables are depicted in the following Bayes net.



Example: Bayesian Networks in PP (2)

Now to fully describe the Bayes net, we would need to give it the conditional probability tables (or learn them if we had data). We would also need to do inference on the Bayes net... so we would probably use some specialized tool...

Example: Bayesian Networks in PP (2)

Now to fully describe the Bayes net, we would need to give it the conditional probability tables (or learn them if we had data). We would also need to do inference on the Bayes net... so we would probably use some specialized tool...

That sounds rather annoying. Wouldn't you prefer to "code" the model in some convenient language, e.g. Python or JavaScript?

Example: Bayesian Networks in PP (3)

To quote Adrian Sampson's text:

Even this tiny example should demonstrate the drudgery of by-hand statistical modeling. It's like writing assembly code: we're doing something that feels a bit like programming, but there are no abstractions, no reuse, no descriptive variable names, no comments, no debugger, no type systems. [...] The goal of PPLs is to bring the old and powerful magic of programming languages, which you already know and love, to the world of statistics.

We Need a Language (WebPPL – Defining a Distribution)

WebPPL = a probabilistic programming language embedded in JavaScript (according to the WebPPL website, pronounced "web people")

```
var roll = function () {
  var die1 = randomInteger(6) + 1;
  var die2 = randomInteger(6) + 1;
  return die1 + die2;
}
//
var dist = Enumerate(roll);
print(dist);
viz.auto(dist);
```

WebPPL – Example

WebPPL: `http://webppl.org`

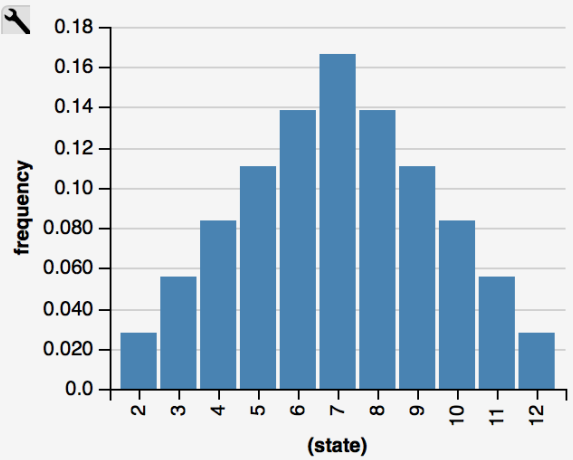
File: Default

add code | add text | .md

```
var roll = function (){
  var die1 = randomInteger(6) + 1;
  var die2 = randomInteger(6) + 1;
  return die1 + die2;
}
//
var dist = Enumerate(roll);
print(dist);
viz.auto(dist);
```

run

{"probs":[0.02777777777777777,0.05555555555555554,0.08333333333333333,0.111
11111111109,0.1388888888888889,0.16666666666666669,0.1388888888888889,0.111111
11111111109,0.08333333333333333,0.05555555555555554,0.02777777777777777],"supp
ort":[2,3,4,5,6,7,8,9,10,11,12]}



WebPPL - Conditioning

So far there was nothing too special about probabilistic programming. Here comes the magic: **conditioning**. Here "conditioning" refers to probabilistic conditioning $P[A|B] = P[A \wedge B]/P[B]$ where A and B are some events and $P[B] \neq 0$.

Example: Suppose that we know that the sum of the two dices is an even number.

```
var roll = function () {
  var die1 = randomInteger(6) + 1;
  var die2 = randomInteger(6) + 1;
  condition((die1 + die2) % 2 == 0);
  return die1 + die2;
}
```

WebPPL - Conditioning (Result)

WebPPL: `http://webppl.org`

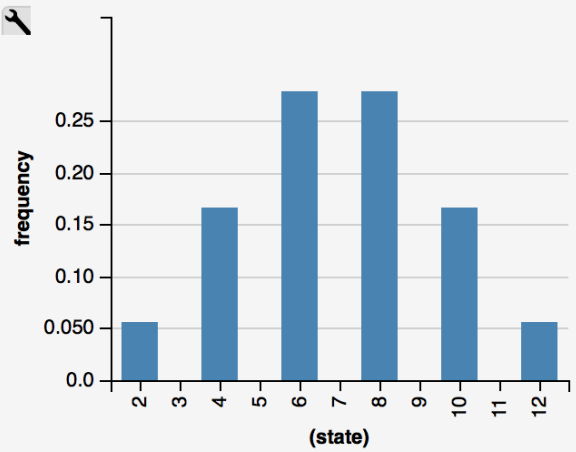
File: Default

add code | add text | .md

```
var roll = function (){
  var die1 = randomInteger(6) + 1;
  var die2 = randomInteger(6) + 1;
  condition((die1 + die2) % 2 == 0);
  return die1 + die2;
}
//
var dist = Enumerate(roll);
print(dist);
viz.auto(dist);
```

run

{"probs":[0.0555555555555554,0.1666666666666669,0.2777777777777773,0.277
77777777773,0.1666666666666669,0.0555555555555554],"support":[2,4,6,8,10,12]
}



Conditioning - Remarks

Now, with conditioning, we already have something that goes beyond randomized algorithms. We are shielded away from implementing the actual inference. We have just specified the model by describing how the random variables are sampled and then we created a new distribution by conditioning!

Warning: The complexity of probabilistic inference does not disappear. It is still computationally hard (after all, next we will see how to do inference in Bayes nets in PP).

Randomized algorithms vs PP: Randomized algorithms get their power from randomness (check the complexity class BPP; it is not known whether P = BPP). PP is more about manipulating (by conditioning and marginalizing) and analyzing distributions that are described as programs that use random generators inside.

Another Example

We note that WebPPL does not support full JavaScript but only a certain "functional" subset of it. Below is an example of a code generating binomial distribution.

```
var binom = function (){
  var r = function(j){
    if (j <= 0){
      return 0;
    }
    var result = r(j-1);
    if (flip(0.5)){
      return result+1;
    }
    return result;
  }
  return r(10); }
...
```

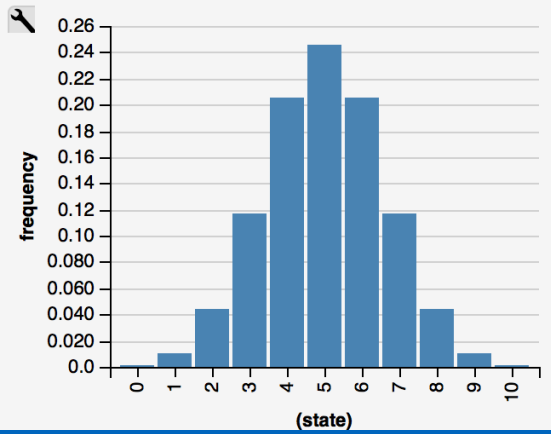
Another Example (Result)

WebPPL: `http://webppl.org`

```
var binom = function (){
  var r = function(j){
    if (j <= 0){
      return 0;
    }
    var result = r(j-1);
    if (flip(0.5)){
      return result+1;
    }
    return result;
  }
  return r(10);
}
//
var dist = Enumerate(binom);
print(dist);
viz.auto(dist);
```

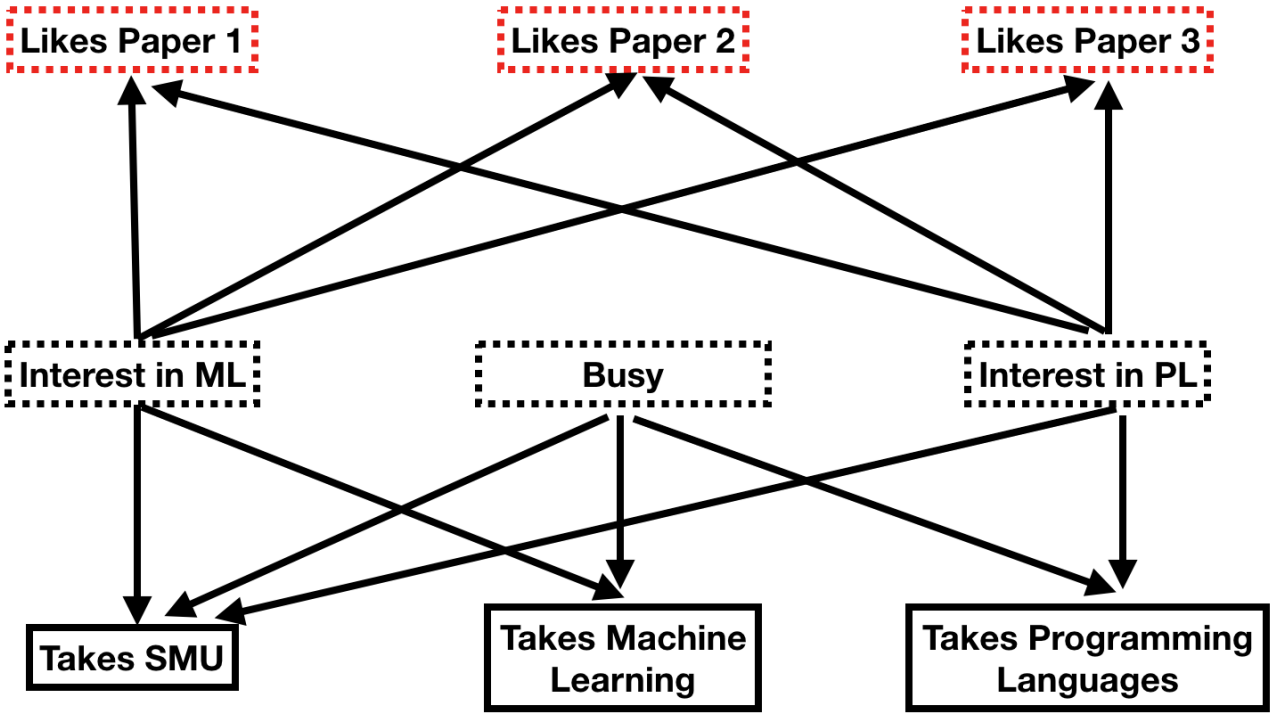
run

{"probs":[0.000976562500000009,0.0097656250000001,0.043945312500000056,0.
18750000000003,0.20507812499999992,0.24609374999999986,0.20507812499999992,0.1
1718750000000003,0.043945312500000056,0.0097656250000001,0.000976562500000000
9],"support":[0,1,2,3,4,5,6,7,8,9,10]}



Back to the Example: Bayesian Networks in PP (4)

This is what we had: *A system for recommending papers. We have 3 papers and all that we can observe about students is which courses they take. The motivations vary, some take courses based on interest, others because of their schedules ("busy"). The observed (solid black), query (red dashed) and latent (black dashed) random variables are depicted in the following Bayes net.*



Coding a "Model of a Student" in WebPPL (1)

```
var attendance = function(i_pl, i_ml, busy) {
  var attendance = function (interest, busy) {
    if (interest) {
      return busy ?  flip(0.3) :  flip(0.8);
    } else {
      return flip(0.1);
    }
  }
  var a_pl = attendance(i_pl, busy);
  var a_ml = attendance(i_ml, busy);
  var a_smu = attendance(i_pl && i_stats, busy);
  return {pl:  a_pl, ml:  a_ml, smu:  a_smu};
}
```

Coding a "Model of a Student" in WebPPL (2)

```
// Relevance of our three papers.
var relevance = function(i_pl, i_ml) {
  var rel1 = i_pl && i_ml;
  var rel2 = i_pl;
  var rel3 = i_ml;
  return {paper1:  rel1, paper2:  rel2, paper3:  rel3};
}
```

Coding a "Model of a Student" in WebPPL (3)

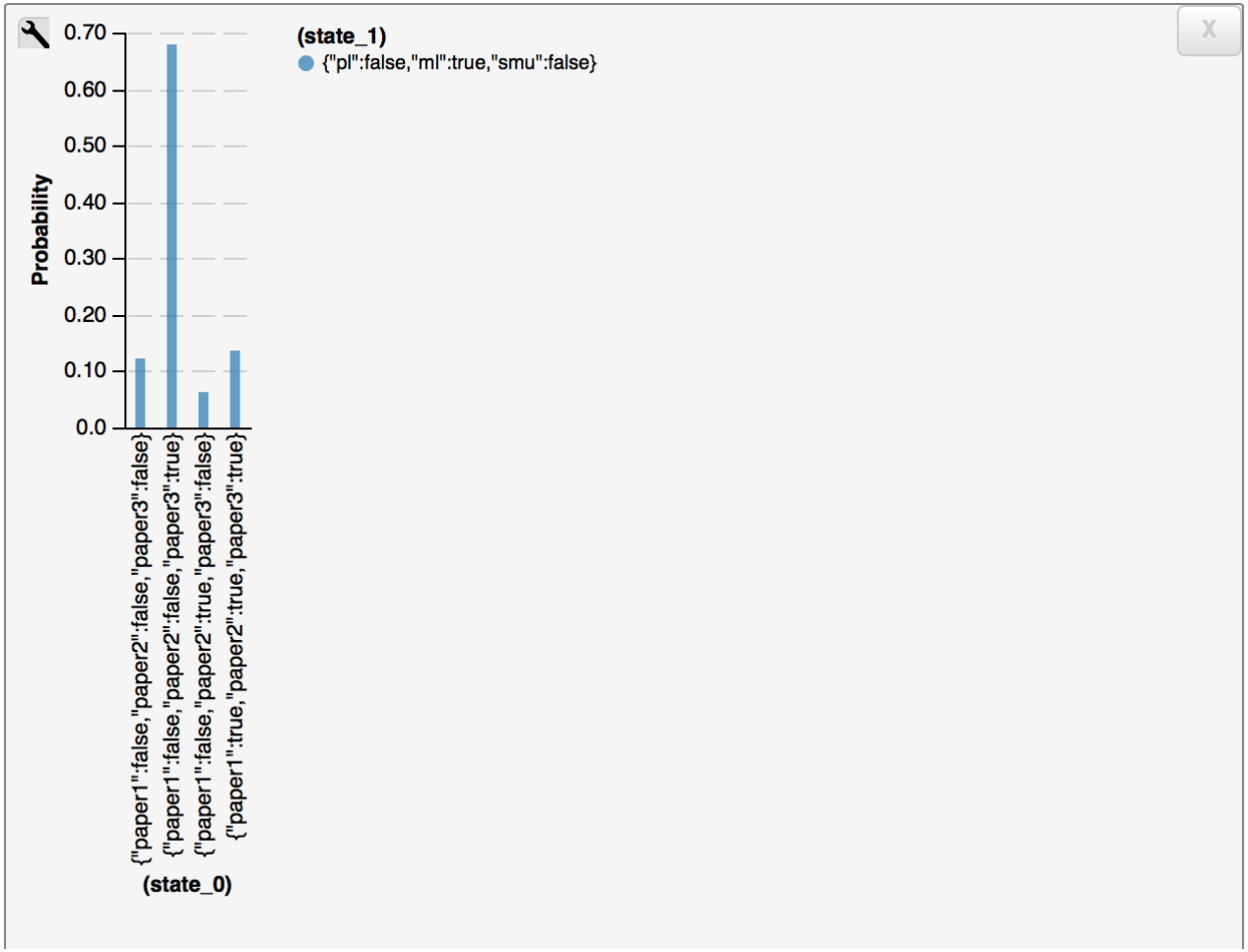
```
// A combined model.
var model = function() {
  // Some even random priors for our "student profile."
  var i_pl = flip(0.5);
  var i_ml = flip(0.5);
  var busy = flip(0.5);
  var att = attendance(i_pl, i_ml, busy);
  return [relevance(i_pl, i_ml), att];
}
```

Coding a "Model of a Student" in WebPPL – Conditioning

Example: We see a student taking ML but neither SMU nor PL? Which papers should we recommend?

```
// A combined model.
var model = function() {
  // Some even random priors for our "student profile."
  var i_pl = flip(0.5);
  var i_ml = flip(0.5);
  var busy = flip(0.5);
  var att = attendance(i_pl, i_ml, busy);
  condition(!att.pl && att.ml && !att.smu);
  return [relevance(i_pl, i_ml), att];
}
```

Coding a “Model of a Student” in WebPPL – Result



Coding a “Model of a Student” in WebPPL

Just imagine how annoying it would be to write this model directly as a Bayes net...

There is much more to WebPPL...

There is much more to WebPPL that we have not really touched here, e.g. different inference methods (we have just used inference by enumeration which is onyl useful for small examples).

For more information: `http://webppl.org/`.

Also there are many other probabilistic programming languages and it is not clear yet which ones will prevail.

Take Away: The difference between randomized algorithms and probabilistic programming.

Next two lectures: Probabilistic **logic** programming (that will also be a bit more technical).