TOMER ULLMAN: So so far, we've talked about just examples of running things forward. I hope I've given you some examples of different procedures that you can run forward to get some interesting stuff, whether it's a mixture of Gaussians, whether it's sort of this mixture of Gaussian plus uniform, whether it's just flipping the coin.

But the question is, OK, I've written down my forward model-- and hopefully, you saw that, even if it was a little bit broken, even if you didn't get the full details, it wasn't that hard to write it down, right? Someone could say, listen, I think the way the bombing works is this. You're going to put a Gaussian. You're going to put another Gaussian maybe, or maybe you're going to put three. I don't know how many.

And you can write that down. And then you say, OK, I actually want to do inference on that. And that's when it becomes a little bit painful to do. And if only there was a way of running forward your model and, having written the forward direction, you can do inference. And it looks like we're talking about something completely different, but actually, it's not. We're basically going to run our models, but we're going to run our models in a way that it's going to do inference.

So let's see. How would we possibly do that? So the basic syntax for any sort of query-- any sort of inference-- in Church is by stating the following procedure. You start out with saying query, where query is not itself a command. It's just there all sorts of queries. There's rejection query. There's mh-query-- Metropolis Hastings. There's explicit enumeration.

But the point is, you would write down that particular query, then you would write down the generative model. This is a list of things-- the way that you think the world works. So here, for example, you would put in the London bombing example. You would put in the list of things like, I don't know, it's either uniform or not uniform, it's either Gaussian or not Gaussian. You're going to put some uncertainty in priors and things like that.

Once you finish defining your forward model of how you think the world works, you're going to

ask it a particular thing. The penultimate statement that you're going to give it is what we want to know. For example, in this particular case, suppose that, before I started, I said, I don't know if the bombing is targeted or not. I'm 50/50 either way. I don't have to be. But let's say I'm 50/50 either way.

So I say, I'm going to flip a coin. It's either targeted or it's not targeted. And that's going to come up either true or false. And what you're going to basically say-- you're going to query on that. You want to say, did the coin come up true or false given the data?

So the last thing-- the ultimate statement that you're going to write-- is basically the conditional statement. And the conditional statement is basically the thing that has to evaluate as true. The usual thing that we would write down there-- I'll give you some examples of that-- but what we would usually write is something like, given that the observed data matches the sample data from my model.

So if you want to do something like the probability of a particular hypothesis given the data, this is the way that you would say, this is my data. This is what I know. And the way that it would know is, you're sort of running down a program, and you're constraining it to give you a sample that matches the actual thing that you see. In the London bombing example, what you would do is you would write something like query, a bunch of defines-- targeted bombing, random bombing, things like that-- I want to know-- is it targeted or random? How did the coin fall?

And what you're going to do is, you're going to say, listen, run this model forward under the following condition. If I just run it forward, I would either get targeted or not. It would be 50/50-- under the condition that whatever this model samples has to match the actual data. So I'm going to run it forward, but the thing that it needs to evaluate as true is that the samples I got from the model are equal to the actual data that I got.

Now once you do that-- once you define that particular thing-- what you've done is change the probability distribution that the generative model describes. Remember how we talked earlier-- I was sort of trying to hammer it home-- that anything that you write down as Church is actually a probability distribution. You write down the program and you run it an infinite number of times and you get some distribution.

Your generative model describes a particular distribution. If you condition that model on

something, you get a different distribution. And that different distribution is now what you're going to sample from. You're going to sample from the posterior. You have some prior, you condition it on some data, and you're going to sample from the posterior.

And sampling from the posterior can be something like-- and we'll give it some examples, like, I know how the world works in terms of their objects, and I know how light works, and I know how vision works. I don't know what the particular objects in this world are. That's what I want to know. I condition on their retinal display being equal to something. And now, my posterior probability distribution is going to basically sample from, say, your face or these objects or these chairs.

Well, the same thing could work, for example, if you're trying to query a sentence. You're trying to parse a sentence from sound, or you're trying to predict how the next step in a physics engine is going to work, or many, many, many, many, many different other things that you can find in probmods.org. So like I said, the "what we know" is the condition. And if you set the condition to true, that's a sampling from the generative model, because you're always going to evaluate as true.

Now, how could you possibly implement this sort of magical procedure? So how could you take some probability distribution and change it into a different probability distribution that does what you want it to do? And there are many, many different ways of doing that. But the easiest way of doing that is by something called rejection query.

How many of you know about rejection sampling? How many don't know about rejection sampling? OK. The way rejection sampling works is that I have some sort of distribution that I'm trying to sample from. And suppose that it's really hard to sample from that distribution exactly.

So let's say that my distribution is this circle. And for whatever reason, it's really, really hard to sample from that circle. I don't want to try to define the probability distribution that describes this circle. It's really hard to sample from it. This is trivial, but there are probability distributions that are really hard to sample from.

What do you do? You can construct a really simple distribution that you can sample from. Let's say that it's really, really simple for me to sample from a uniform square that encompasses the circle. So now, I have some probability distribution. There's a uniform distribution over the square that I can sample from. What does it mean I can sample from? It means each time I

run the procedure I get some point in the square.

But I don't want the points in the square. I want only points from the circle. So what I would do is basically sample from the square, and each time it falls outside the square, I'm going to say, throw that out. That's called rejection sampling. Because you sample from some sort of procedure that you know how to sample from, then you check that sample. And if that sample didn't meet your desiderata, you throw it away. And what you're left with is the circle-- the distribution that you're trying to get.

So what is the simple thing and what is the hard thing in what we're describing so far? The simple thing is the generative model. It's relatively easy to sample from the generative model. We just wrote it down. So we know how to sample from it. But we're looking for something else. We're looking for some sort of different program. We're looking for some sort of setting of a program that would generate the data that I saw, not just the generative model that I wrote.

The way that we would do that is, in rejection query, we would sample from the generative model. We would check, does that fit what we know? Suppose I just sampled something. And then I check, does that apply to the condition that I want?

So let's say I have a particular world. It's a sort of a silly world. But let's just make sure it works given the last time. OK.

So what I'm going to do is, I'm going to describe a world in which there is Legolas. Gimli, and Arwen. Anyone get *Lord of the Rings* references or something like that? OK. Each one of them is going to take out a particular number of orcs. And let's say that's my generative model is that I don't know how many each one of them took out. Let's say that they take anything between zero and 20.

They're having a brawl. Each one of them is going to take out some number of orcs. So we're going to define the number of orcs that Legolas took out as some random integer-- 20. OK? Gimli's the same. Arwen's the same.

And we're going to also define the total number of orcs that they took out as just plus each one of these things. So we're going to look at the pile of orcs that they took out in the end. That's my generative model. That's it.

What I'm going to wonder about is, how many orcs did Gimli take out? Not knowing anything,

how many orcs did Gimli take out? Should we just switch to kill or something? I feel bad for the orcs.

But OK. How many orcs did Gimli take out? Well, we don't know. We just said we don't know. It's a random integer between zero and 20. It's anyone's guess. If I just ran this model forward, it would give me any number between zero and 20. If I ran it 1,000 times, I would get a uniform distribution over zero and 20.

Now, I'm going to give you a condition. It's a simple condition. The total number of orcs that they all took out is greater than 45. Altogether, Gimli, Arwen, Legolas took out more than 45 orcs. Now, how many orcs do you think that Gimli took out?

Now, the point is that you would somehow shift your distribution. This is a very simple problem. You could probably write it down on a notepad. But you're trying to do the posterior of the number of orcs that Gimli took out, given-- conditioned on-- the fact that all of them together took out more than 45. Is this making sense? OK.

How would I write down that as a rejection query without using any syntax that you haven't seen already. Without using anything like query yet, I'm just going to use, basically, recursion to write down a rejection query for that. And later on, you can use a rejection query.

But what I would do is, I would just say, here's a procedure that's going to give me back the number of orcs that Gimli took out conditioned on everyone taking out more than 45. So I write down a particular generative model. Like, I know that Legolas took out somewhere between zero and 20, Gimli took out somewhere zero and 20, Arwen took out somewhere between zero and 20, and they all took out the number of total orcs.

Now I say this. If the total number of orcs is greater than 45, that's a good sample. So I'm going to go through this. I'm going to sample my program. I'm going to get, Gimli did-- I don't know-- 15. Legolas did 10. So now, say, Arwen did 20 or something like that. So now we got to 45.

Is that a good world? Yes, we're over 45. Fine. Give me back whatever it was for Gimli. I don't even remember what it was. Give me that back.

Suppose it didn't add up to 45. Try again. This is basically the circle and square example from before. The if statement is telling us, if this matches my condition, sample randomly from the world. Sample from the square. Sample from the generative model. If the thing that you got

matches the condition that you want, give me back the sample. Give me back that answer. If it didn't, try again.

And now, if we do this, and we repeat this procedure 1,000 times, then it's no longer uniform distribution. It's greater than five or whatever. It's going to be zero on that. Because if he took out zero, they're never going to get to 45, right? And it's probably going to be sort of skewed in this direction. So that's the posterior distribution on how many orcs Gimli took out conditioned on all of them taking out more than 45.

That's amazing, you guys. You've just understood rejection query. You've just written down, in a few very simple lines of code, a sampling with rejection. And in fact, you can define all of conditioning using this thing.

You don't have to get fancy with Metropolis Hastings and things like that if you're just trying to sort of prove things. If you're into computer science and things like that, you can just define conditioning using this. You're saying, I have some probability distribution. I'm trying to condition it. I get a different probability distribution-- the posterior. How well behaved is it? Can I write down? Things like that-- you can prove it. All the sort of things that you want to prove, you can prove using something like this construction.

Why shouldn't you use that? So you can do that if you're into theoretical computer science. What is bad about rejection query? Does anyone know? Can you guess? Sorry?

AUDIENCE: It's costly.

TOMER ULLMAN: Costly in what sense?

AUDIENCE: [INAUDIBLE]

TOMER ULLMAN: Right. Exactly. Depending on the condition, it might be a very, very bad idea. Here, I could sort of do the condition, because I ran the model forward, and sometimes, I got over 45.

So yeah, why is rejection query a particularly bad example? Because your condition might be really, really, really hard to satisfy. So if, for example, I change this to 59, what will happen now?

I think because of the way I wrote this, it will never reach that. But thanks for that point. But let's change this to this. Don't run this, by the way, because it will never stop. Or it will take a

long time.

So now, it's only going to be fulfilled if each one of them took out 20 orcs, right? They would all need to take out 20 orcs for the total number of orcs to be equal to 60. When is that going to happen? It's going to happen 1 in 20 times 1 in 20 times 1 in 20. You're going to waste a lot of samples on something that's never going to happen. And it doesn't even matter that much.

And you can easily look at this and you can sort of say, well, obviously, Gimli took out more than this, because I know how to program and I can figure it out. But oftentimes, you will find that you can't exactly say. You look at some convoluted program and you won't exactly know how this should look or whether it's easy or whether it's hard. But in that sense, rejection query is probably a bad idea.

Another example of why it's a bad idea is something like precision. So let's see. Don't run this, because it'll take forever. Let's do an estimate of pi.

And again, I was sort of going to give this as an exercise. But since we don't have a lot of time, here's one thing that you could do with rejection query, or you can do it with any sort of query. You could try to estimate pi.

Literally, using that example that I just did, you could try to say, OK, sample from some square, only accept the Xs that are in the circle, and then sort of try to estimate how many samples you got in the circle out of the total number of samples you took. So run 1,000 samples, and see how many of those fall in the square. And if you run 1,000, you'll get three point something.

You can try this. I did it as an exercise for you. You can see I sort of set up some of this syntax. If you want to try this out later, please do.

You run 10,000 samples, it's like 3.1. I think if you do it 100,000, it'll probably do 3.14-- maybe not that much better. Like, 100,000 samples, seriously, to get 3.14-- we all know it's 3.1415. Sometimes, if you can use math, you should use math, in some cases.

On the other hand-- this is, by the way, a reason why you should probably not use sampling in general when you can use math. We can solve it analytically-- if you're interested in precision and things like that. But suppose you're not interested in precision. I'm actually going to try and help that.

So you can actually get a pretty good estimate from about 10 samples. If you just do 10 samples on this thing, you'll probably hit something like three as an estimate. You can do the histogram for where it will fall. Most of the samples-- like 70% of the samples-- will fall between 2.8 and 3.6. If that's what you care about, then that might be what all your vision system cares about, or different things that might require sampling. Then that's fine.

And I'm not going to go too much into this, because, like I said, the dream of probabilistic programming is sort of to free you from thinking too much about the sampling. But those of you that are interested in sampling, that are interested in algorithmic learning and things like that, there's been a lot of research on exactly that. When are we OK with just taking one sample? We write down some sort of model and we see how well the model does by taking one sample, 10 samples. What's the precision that you can get? And does that precision match people trying to perform a similar task of estimating that thing?

That's, again, like the sampling hypothesis-- not sampling hypothesis for neurons, sampling hypothesis for the way people answer questions. They sample one or two or a few number of points from their generative model-- not that many. And the claim is that you can sort of see that they get better with time, that they probably don't take that much if you give them more time to think. It looks a bit like a sampling procedure that takes more samples.

And it seems like you can get away with quite a lot if you just do 10 samples or 100 samples for pi. And there's this SMBC cartoon that I quite like, which is like, why shouldn't physicists teach geometry? And they're like, well, you know, how do I remember the value of pi? It's quite simple. I look at my fingers and there's five of them, and that's about pi.

What else could we do if we don't want to use rejection query, if we don't want to use rejection sampling? Suppose we don't. We probably don't. We could try to do exhaustive enumeration. If our model is small enough, we can just consider all the possibilities and explicitly score them.

The other thing that we could do is something like Metropolis Hastings. How many of you are familiar with Metropolis Hastings? OK. Why don't we raise our hands to the degree that we are familiar with Metropolis Hastings, where here is really familiar, here is not familiar. OK.

Metropolis Hastings-- I'm not going to go too much into the details. I'll just be doing it a disservice. But the way to think about it is to say, instead of just sampling at random from the entire space of things that could be really, really bad equally, I'm going to try and sample from something that I think is likely to be good. And the way I'm going to do that is, I'm going to

construct, basically-- what's the best way to explain this?

It's to say, I'm at a particular point in the space. I've gotten my sample. I already have it. What should I do now?

Rejection sampling just says, well, just sample another one from the generative model and see if that works. That's a bad idea. What you should actually do is try to use the sample that you already have and how good it is as a sample to inform your next move.

So now, what you're going to do is, you're in this particular point in space, and you're going to move to a different point in space that depends on the point that you are now. So for example, if you're in some two-dimensional space, and you're over here, you're not going to sample from this square. You're going to sample from a point next to it, let's say. That's your proposal. You sample according to your proposal distribution. Your proposal distribution tells you where you should sample next.

So you're here in the space of all possible programs or your theory space. Metropolis Hastings is more than just programs-- anything. You're in the space of possible things that you're trying to sample from. You're here. You've got one sample.

Now, you're sort of looking around, and you take another sample. You do that according to your proposal distribution. Your jump there. And you evaluate this point. And the way that you evaluate it is, you just look, how well does this thing fit the data? And we can get into that-- but the ways you sort of score your model according to the data.

And you say, well, this one fits the data pretty well. How well does this one fit the data? Not so great. So I should probably move over there. I should move to that point in program space. I'm going to move over here.

Now I sample another point. I go over here. How well does this do? Not as great, but I might still accept it. The point is, you're going to accept and reject new points, new executions of the program according to how they predict the data, according to how well they answer the condition. If the condition is a simple true-false, that just means that you have an absolute yes or no.

But many of these conditions are going to be something like, well, it's good if it matches it. You get some score from the likelihood and the prior. So you can score this new point in program

space and either accept or reject it. And this thing of moving around in program space and sampling according to some new proposal distribution and accepting or rejecting and moving around like that is a lot more efficient in most cases than rejection sampling.

And in the limit, if you keep on doing this-- if you keep on walking around, walking around, taking samples, accepting or rejecting them depending on how well this new point in program states does-- what you'll end up with is the posterior distribution that you're trying to sample from. And I should say, what is a point in program space? It just means a program that I have completely evaluated.

Like in the case of the London bombing, it would be, I have two Gaussians, and this one's center is here, and this one's center is here. I sort of walk through all the things that I can sample. I've gotten one particular run of the program.

And then I try to move to somewhere else. Like, I might change the center of this Gaussian. Or I might say, well, you know what? Actually, there aren't two Gaussians. Let's run it again. Let's run it again. There was actually 10.

So what I do, the way I change, the way I move around in program space is to go to a particular point along the tree of the evaluation and say, what if I change that? What would I end up with? I sort of re-sample. And I re-sample. I end up with some other program. I basically say, how good is that? Yes, no-- and I accept or reject that.

As I said, I'm doing this a little bit of a disservice. But if you keep that mental image in your head of something bouncing around and accepting or rejecting new proposals according to how well they do compared to one another in a sort of pairwise fashion, you won't go far wrong.

What this also tells you is that it's a little bit important where you start out. So if I start out in this particular point in program space, or in any space, and I look locally, I might accept or reject and things like that, but, actually, the really good stuff is over here. The high probability stuff is over here.

But it'll take me a long time to get to that. Because I started out here. Does everyone sort of understand what I mean when I say "here?"

So supposing you have a random square, and you're trying to sample from a probability distribution over the square, and in the corner is this much less probability. But you started on

the corner for whatever reason. And now, you're trying to figure out how to get to those good samples in the center, but you can only move locally.

You will eventually get to the center. If you run this on long enough, you will eventually get to those good probability spaces. But it depends a lot on where you started out. And that just means that, oftentimes, in Metropolis Hastings and MCMC and things like that, you hear about burn-in, which is just to say, we want to get rid of the initial x samples, because those samples are going to be biased. They're going to depend on where we started out. And the hope is that, after x samples, we're no longer biased. We no longer remember where we started from. We're just sort of sampling around in the space.

So the way Metropolis Hastings works-- and this is the backbone of inference in Church-- is, you would write down something like mh-query, then you would write down the number of samples that you want from your posterior distribution. You would write down the lag. The lag is just to say, forget every x steps. If you want to talk about this, we can talk about it later. It's not particularly interesting.

These are just two numbers. And if you make them bigger, you will get more samples. You will get a better estimate of your posterior distribution.

You write down some generative model, you write down what you want to know, and you write down what you actually know. And you do a random walk in the program evaluation space. Like Josh said, what's nice about this is that it's very, very, very, very general. This will work on any program, more or less, defined correctly.

You need to make some decisions, like how many samples you want to take, what the lag is, what the burn-in is. You can do all sorts of fanciness. You can do particle filtering. You could run several chains. You can do temperate annealing. You can do lots of different things that I just said and might not make a lot of sense. But the point is that this procedure could be made more or less fancy.

One of the problems with it is, it takes a while. Like Josh said, there's a lot of better algorithms. If you know what your representation is, and it's something like a feedforward neural network, you probably shouldn't do Metropolis Hastings on it. There's a lot of very fast things that you could do, like gradient descent, and you don't need to wait around for this thing to happen.

Let's see. So I think we have enough time to give you some examples of inference. Let's walk

through some coin testing examples, a bit of intuitive physics, and a little bit of social reasoning.

So suppose that I took a coin and I flipped it, and it came up heads. What do you think? Is this coin weird? No. It's OK to say no.

What if I flipped and it got heads five times in a row. Do people think it's weird? Raise your hand to the degree that it's weird. Is it weird that it's five?

If I flipped it 10 times and it came up heads, raise your hands to the degree that it's weird. 15 times in a row, heads? 20 times in a row, heads? OK, we more or less asymptoted somewhere between 10 and 15, which is exactly right.

And the point here is something like, we have a particular prior over what we think the weight of the coin is. We're pretty sure that the coin is not biased. We're pretty sure that the coin is supposed to be equal weighted.

But then we get more and more evidence, and we sort of figure out that, wait a minute, no, this might be a trick coin. This might be weird. And the point of the first example is to show you the basics of conditioning and inference and things like that using the coin example.

So what we would do is, we would take in-- and again, as I said, I'm slightly going to rush this. But I'll still try to explain it. We're going to define some observed data. Suppose our observed data is that we got five heads in a row. Is that five? Yes. It's five.

And now we're going to say, OK, we're going to define something. We're going to define an inference procedure. We're going to call it samples.

The way it's going to work is that it's going to give us 1,000 samples back. We said we're going to write mh-query, the number of samples, and now we're going to define a generative model. We're going to end up with the thing that we're actually interested in under a certain condition.

So what's our model for the world-- for this simple world? Let's say my prior on this being a fair coin is very high. One means I'm absolutely sure it's a fair coin. Zero means it's not a fair coin.

And we're going to put in a big prior on it being a fair coin. It's going to be 0.999. And then we're going to basically say, somewhere in the beginning, the way I think the world works is that you're going to pull up a new coin off the mint, and you're going to say, is it a fair coin or

not?

999 out of 1,000 are fair. One is not. So we're basically saying this. We're going to say, is it a fair coin? And this is going to come up true 999 times out of 1,000. And it's going to come up false one time out of 1,000. Because what we're basically doing here is just flipping a coin.

We're flipping a coin with a bias of this prior. So what we have here is this thing is going to come up this thing-- fair coin. It's going to come up without any knowledge, without any data, without seeing anything. Just, you took a coin off the mint. 999 times out of 1,000, you think it's going to be fair, without seeing any data.

Now you're going to create a coin. The coin is going to take in some weight. There's this procedure that you can flip and actually get heads or tails. And the way this coin is going to work is that, if it's fair, it's going to have a weight of 0.5. If it's unfair-- and this is a very simple example-- it's going to have a weight of 0.95.

So the fair coin comes up heads or tails equally likely. The unfair coin-- the trick coin-- comes up heads almost all the time. And again, you can define a different hypothesis. It doesn't really matter. But the point is, I defined some sort of coin.

And now, I define some sort of hypothesized data. Well, the hypothesized data is just, I sample from this coin that I just made. And what I want to know is, is this a fair coin? Yes or no.

The last statement is conditioned on this sample data, this illusory data, this imagined data being equal to the observed data. So now, you have some sort of program, and you're trying to figure out, did this come out to be a fair coin or not? When I did this here, if I didn't condition on anything, then it, 999 times out of 1,000, should give me back, yes, this is a fair coin.

But I've now conditioned on some data. And the data is that it came up five times heads in a row. And if you do a histogram for that, then you'll find that it's still very likely to be a fair coin. Because the prior is so strong.

But now, we can change. We can change the data to add a few more heads here. And I think we're now more or less at 10. And it's starting to be like, well, is it a fair coin-- yes or no? Well, I'm like 60% sure that it's a fair coin now.

What if I flipped it-- I don't know-- like 20 times or something like that, and I came up with that? And it's basically saying that it's 100% not a fair coin. This is false. It's basically saying, is it a

fair coin? No. Even though the prior is strong-- even though if I just ran my generative model without any conditions, usually, it would be a fair coin, there is no way that I would run my generative model and sample it 20 times-- that coin flip-- and it would come up heads all the time, when the alternative is that it's going to come up heads.

So now, you can sort of play around with this coin. This is a nice example. And by the way, Josh, some version of this-- not much more complicated than that-- was a cognition paper a few years ago, where, basically, you gave people different sequences of coins, different sequences of numbers, and you started to see, where does it become weird? What hypothesis did they think is likely?

And all that they did was, they gave it a more interesting hypothesis space. Instead of saying it's either a fair coin or a coin that's 95% heads, they gave you a more general hypothesis space. Like, it could be a coin that comes mostly heads, mostly tails. Maybe it's a coin that does heads, tails, heads, tails, heads, tails, heads, tails. Or you can define the sort of alternative procedure, but that, you would change over here. The rest of this would stay more or less the same.

That's a very simple way of getting some hypothesis tested. Let's do some very simple, intuitive physics. Let's try something like this.

So in Church, you can basically animate physics forward. I guess I hadn't counted on, when it's a full screen, it sort of does that thing. But what you can do is, you can define, basically, a two-dimensional world where you say, listen, here's a two-dimensional world. It's this big. I'm going to add some shapes. I'm going to put the shapes in random locations. I'm going to set gravity to something. And then I'm going to run it forward. What happens?

AUDIENCE: Command minus.

TOMER ULLMAN: Sorry?

AUDIENCE: Command minus.

TOMER ULLMAN: Command minus for running? OK.

AUDIENCE: [INAUDIBLE]

TOMER ULLMAN: Oh, of course. Thank you. Trivial. Thank you. Is that better, everybody? Yeah. OK.

So I have this thing, and I'm going to hit simulate to try and see what happens. So I basically have these things. I guess, in this case, I didn't put any randomness on where they actually are. But you could easily imagine putting some randomness on where these blocks start out, where these blocks are. But the point is, this is just running it forward in physics.

Now what is that good for? Well, you could, for example, define a tower. A tower is just defining a bunch of blocks, one on top of the other. And now you can run it forward, see what happens.

What do you think? Is this going to fall or not? Yes? No? Let's see. And you can simulate that forward. It fell. OK. Very nice.

Now what can we do with that? And as I said, I'm going to zoom through this. What we can define is basically a bunch of towers like this. Each one of them is just saying, the blocks are like that. And all I'm going to do is slightly perturb them and see if they fall down. And I'm going to do that 1,000 times for each tower. And I'm going to do that for a bunch of towers. Some of them are stable. Some of them are not stable.

So you can write down some Church code, which is basically, this is my world. There's some ground. The ground is just a rectangle. Here's a tower-- a stable tower. The stable tower-- all that it means is that I'm creating some blocks in this particular order.

Here's an almost stable tower. It's blocks in this order. Here's an unstable tower. It's blocks in that order.

And now, what I'm going to do is, I'm going to run this tower many times, and I'm going to count up the number of times that it actually fell. And if you do that, you'll see that the stable tower didn't fall down. This is just saying, like, did it fall down-- false, true?

This one didn't fall down any of the time. This one fell down some of the time. This one fell down all the time.

It's a toy example. But it's actually a toy example of a very nice and interesting paper that came out very recently and shows something deep about intuitive physics. We can ask how hard it was to implement this thing. This is an implementation of liquid physics in not so much Church as webPPL. And what they were trying to do here is to sort of say, well, this is a bunch of water.

Physics is frozen right now. Imagine that this is a big glob of water. This is a cup over here. And this is some barrier. So if the water falls down on the barrier, it's going to go every which way.

So let's see. If we run it, one of the questions that we can ask here-- and this has sort of been an example-- what I'm trying to show you here is that this is an active area of research. Even though it's sort of like Church-- it's 2D physics. It's simple. But people have been doing what they've been porting in as a liquid physics thing into Church. That took them a little bit of time. You probably want to talk to people who know what they're doing in that. Save yourself some time and talk to people in a Goodman's group.

But what they did is, they ported it in a liquid physics implementation into Church. And they sort of said, OK, now we have some liquid physics and we can try to ask some questions. Like, suppose that this glob is going to fall down. We set it down over here and it's going to fall down. Where should we put this block in order to get as much of the liquid into that cup?

That's an interesting question. It shows something about intuitive physics of liquids and things like that, more than just objects. So the way that you would do that is, even in a simple world like this, you would basically say, fine, put this thing somewhere, randomly uniform, conditioned on getting as much water into this thing as possible. And then try to figure out where this block should go.

So you start out uniform, condition on as much water as possible. You'll get some posterior distribution of where to place this block. And just to show you what that looks like, let's try to-- so suppose that we actually tried to run this.

So in this case, you don't want the block to go there, for example, right? Because most of the water is going to slosh over there. What if we put it over there? It's a little bit better. That's not so great. You could run it many, many different times.

At this point, I hope most of you, even if you don't quite know what you're doing, you can see how you would go about writing the program to figure this out. You would write down the physics world, assume most of that is taken care of for you. All you need to do is sort of figure out where to place this block, put some uniform distribution in this area, condition on most of the water landing here, and then just sample, and figure out where this thing should be in the world. And that's pretty cool.

So here's another example. That was intuitive physics. Let's move on to intuitive psychology, which Josh was sort of getting out at the end of his lecture.

And here's a very, very simple question, which is something like, suppose that you see an agent-- this guy with googly eyes. Can people see him from way down there? There's a guy with googly eyes. It doesn't matter. It's me.

I'm right here. There's a banana over there. There's an apple over here. So there's the banana over there, apple over there, and I start walking over here. And now, someone asks you, why did Tomer go over there? And you say, well, I guess he wanted the banana. And you say, oh, but you don't have access to his goals. How do you know he wanted the banana? And you say, well, because he went to the banana. Well, you're just being circular.

How would you actually solve a task like this? It's sort of trivial. And you could solve it through something like cues. You could say something like, well, the thing that you approach is your goal.

But another thing that you could do is, you could say, well, I assume that the way Tomer works is that he has goals. I don't know what his goals are. But I assume he has goals. And I assume that he can plan to reach those goals in some sort of semi-efficient manner.

And he has some beliefs about the world. And he's going to carry out some sort of planning procedure in order to get to his goals. And if Tomer wanted the banana, the action he should take is to do this. It would be very unlikely for him to do this. If he wanted the apple, he would do that, not that.

So you could sort of use this planning procedure to set the knobs on your procedure. Think of it like a generative model. Your generative model is something that goes from goals or utilities and beliefs to something like actions. And you would define the goals.

Let's say you don't know what my goals are, so you place some distribution over them. And then you get to see my actions. And you basically try to say, well, in program space, what would be the setting of the goals of Tomer, such that it would have produced the observed actions. And if you write down a model for that, you'll find that if I set the goal for Tomer as banana, I'll get the observed action, which is, he walked towards the banana.

Similarly, for belief, if you know something like, you know there are two boxes here. You don't know what's inside them. You know it's either a banana or an apple. And you know that I really

love bananas and I hate apples. And you see me walking towards this box. You can infer that, ah, he probably thought that was a banana inside, or he knew there was a banana inside.

And again, if you had some sort of planning procedure, you would say, OK, it would make sense for me to set his belief to be banana, because the outcome of that, if I run the model forward with those settings, would be for him to walk in that direction.

Now let me show you just one example of what that sort of model would look like. So this would be under intuitive psychology. Those of you who are interested in sort of inference over inference and, how do agents reason about other agents, or goal inference or things like that, you might want to take a look at this section.

And this is sort of super simple. There's no probabilities, exactly, in the sense of, it's going to be either this goal or that goal. It's obviously something that can be modified if you want to. Does everyone more or less see what's going on here? Let me make that a bit bigger.

What I tried to write down is a model in which someone went for an apple, and you're trying to figure out, why did he go for that? Yes. Sorry. It really should be over here.

Now, let me start out, actually, with something like planning. Let's write down the forward model before we do inference. Before we do inference, let's write down how we think the world works.

The thing that I said-- the way the world works is that Tomer has some goals and some beliefs. And given his goals, he'll take some action to achieve his goals. Let's write down that part. That's the forward part. If we can write down the forward part, the inference part comes for free. We just put that in an mh-query and say, what's the goal that made the observed thing happen?

So what we would do is, we would write down something like, what action should Tomer take? Choose an action. It's a procedure. It's a procedure that takes on a particular goal. Here, it's a particular condition that I can satisfy. But it could be a utility. It could be anything.

But it's, what action should I take, given a particular goal, given how I think the world works? That's a transition function. If I take this action, what will happen? I need to know that if I go left, from your perspective, I'll get to the banana. That's the transition function for the world. And I need some initial state.

And now, I sample some action. I do an action at random. I either go left or go right. I sample it from some action prior. Suppose it's completely uniform. Define action. It's simple action prior. And suppose my prior is go left, go right, with equal probability.

Everyone with me so far? We're trying to get a procedure that will give us an action. What action should I take? Imagine you took an action. It doesn't matter which one.

Now, what action did you end up with conditioned on that action getting you to your goal? This is a rejection query. I'm trying to sample an action conditioned on that action getting me to my goal.

So let's say I sample the action. I hypothesize that I go that way. Did I satisfy my goal? No. I ended up with the apple.

Do it again. Run it again. Run it again. Now I sample the action "go here." Did I end up with a banana? Yes. OK.

So what action did I take? I went, from your perspective, left. Return that.

We've just written down a procedure for planning. And it can be made much more complex than that in a few short steps. By complex, I don't mean that it's hard for you to follow. I mean that it can take in multiple worlds, multiple steps, utilities, probabilities, things like that. And it will spit out a sequence of actions for you to go from x to y to get to your goal.

And it's written as planning as inference. Now, there are many different types of planning procedures. You could write down Markov decision planning processes. You could write down rapid random trees. I'm just throwing out names there for those of you who are interested in these things. There's lots of ways of doing planning.

This is one particular way of doing planning. You could have done many different ways. But the point is that we assume you can even sort of wrap this up in something. Like I, as the observer-- I as you-- don't need to know exactly how Tomer works. I just need to know that there is some procedure such that if I put into it a goal, and somehow, how the world works, it will spit out some rational action.

It's preferable that I have some idea of how it works. It doesn't need to be the right one. It doesn't need to be the one that I actually use. But you need to have some sort of sense that I am planning somehow. This is one way to plan.

And now, this is just showing you. I put some uniform prior on the action prior. I either go left or right. The transition function of the world is such that if you go left, you get an apple. If you go right, you get a banana. So that's from my perspective, I guess. If you do anything else, you get nothing.

And then you sort of just say, my goal is, did I get to the apple, let say, or did I get to the banana? I put that into the choose action and I'll end up going left. Because my goal was to get to the apple. The apple was on the left. I'm going to choose an action in order to go to the left. This whole thing is just to show you that, in fact, this works. If you sample it forward, it will give you the right action.

You can now wrap up this whole thing in something that does goal inference, that doesn't know that my goal was this, that puts a uniform prayer on this thing, and then runs forward many, many different samples and comes to the conclusion that it must have been the apple, because he went left. And again, this example is fully written out for you over here, as well as the belief inference.

This is an example of implicature. How many of you know what implicature means, like Gricean implicature? It's the sort of thing where someone tells me, hey, are you going to the party tonight? And I say, I'm washing my hair. Or you say something like, how good of a lecturer was John? And I say, well, he was speaking English.

I'm not exactly telling you he was a bad lecturer. But if he was a good one. I would say it. The fact that I didn't say-- the fact that I chose to say something else-- implies that he probably wasn't a good lecturer. And this sort of happens a lot.

And that it works is that-- and this happens a lot in language games, social games, reasoning about reasoning-- I'm the speaker. You're the listener. I have some model of you. You have some model of me.

I know that you know that I would have said he was a good lecturer. I know that you know that. If I wanted to, I would have said that. And I'm not. So I know that you know that I know that. And it works out such that you realize that he's not a good lecturer.

And that sounds sort of convoluted. But it's actually not that bad. And I want to show you an example of how that works.

And this particular example is based on the "some not all" example. So this is the sort of thing like, I'm a TA in a class and someone asks me, how did the students do on the test? And I say, some of the students passed the test. Do you think I mean, all the students passed the test? No.

Now, why not? Because some, in a sense, also means all. All the students is true if some is something like a logical thing that means greater than five or greater than zero or greater than one-- whatever it is. It can include all. But if it was all, I would have said all. And if it was one, I would have said one.

So people are likely to infer from me saying some, that I probably mean-- if there's 100 students and I say some, they can give you a distribution over what I mean by that. And that distribution depends on the alternatives-- the alternative words I could have used, which they know I didn't. But they know I could have.

There's a nice example of scaler implicature and how it would work in probmods. What I want to show you is a slightly different example, which is, again, the London bombing example. But the way it would work is something like this.

So here's the background for implicature. I came up with this yesterday. I'm not quite sure it'll work. But we'll see.

Imagine that things work like this. The city of London is being bombed. Again, sorry for the slightly dire things. The city of London is being bombed, and there are three places it could be bombed. Again, it's this uniform square.

It could be bombed in the blue part-- anywhere here. If it's bombed there in the blue part, I would say it was bombed outside London. That means outside London. It doesn't include these things. It's just outside London.

If it was bombed anywhere in this red square-- so imagine that this square is something like-- I don't know-- zero to two, and over here, it's zero to one. Anywhere in the red square is called London. If a bomb fell there, I would say, a bomb during the blitz dropped on London. If someone asked me, where did the bomb fall, I would say, in London. But that includes this whole thing.

It also includes Big Ben. Big Ben is in London. Maybe some of you can see what I'm getting at.

So if it fell on Big Ben, I could say it fell on Big Ben. I can say it fell in London, because that's also true.

If it fell here, I would say it fell in London. If it fell here, I would say it fell outside London. Now, there's a general, and a staff sergeant walks up to him, and he says, a bomb fell on London during the blitz. And the general says, where did it fall? And he says, it fell in London. And the general says, OK.

Then he looks outside his window and he says, good god, it hit Big Ben! And he says, yes, I said it fell in London. That's very weird. We don't expect people to do that. And Gricean implicature says we shouldn't do that.

But Grice said it in a way that's like, you should give the maximal amount of helpful information and not hold out other things. How would that fall out of a particular model? Well, the way it would fall out is something like this. And again, I'll show you the code for that.

It's something like, there's the speaker. The speaker is the staff sergeant. He could choose one of three words-- outside London, in London, dropped on Big Ben. Let's say Ben, London, outside-- something like that. He could choose one of three.

Now, the fact that he decided to say London could include Big Ben. But if it was Ben, he could have also said Ben. He didn't, which implies that it probably fell over here.

So the last model that I wanted to show you was exactly that. You sort of say, listen, there's some prior. The bombs sort of fall anywhere. And there's some distance here. And let's say you start out with just random gibberish. You can say either Ben or outside or inside. It doesn't matter. Regardless of what the world actually was, this is just defining what each one of these words mean.

To hit London means to be inside that small square I said. To hit Ben means to be inside that smaller square inside London. To hit outside means to hit outside. That's what they mean. It just gives you back a true or false on a particular point in a two-dimensional space. Is it true or is it false that this happened?

Now, you have a speaker and a listener model. And the way that the speaker works is that he has a particular state in mind. Like, he's looking at the state of the world. The bomb fell here. And he needs to communicate something. And he's reasoning about the listener to a particular depth.

What he's going to do is he's going to choose a word randomly, because our prior is random. Kind of like before, we chose an action at random, and we saw if it worked, he's going to use a word at random. And he's going to choose the word such that it's going to cause the right state in the listener. So he needs a model of the listener.

What's the listener? The listener is someone who takes in a word and tries to figure out the state. He doesn't know what happened. Where did the bomb fall? Someone gives him a word. And he's trying to figure out the state.

So he's drawing a state from the prior. And this prior is, it could be anywhere. It could be anywhere in the square. Where did it actually fall? Well, it fell here, let's say, given that I got this particular word.

But this word was generated by a speaker, which I need a model of. So there's this model of the speaker understanding the listener, and a model of the listener understanding the speaker up to a particular depth. And they need to bottom out at some point.

And it's not that hard. I mean, it takes some time to wrap your head around it. But it's written about eight lines of code. And that's why I said that Church-- remember that caveat I gave you of, it's not a toy language. It's under development. But it's actually been doing some pretty interesting stuff. This is one of those things. These sort of language games are really hard to write in many other models-- really hard to write.

And here, it's kind of trivial. You can sort of see where to play with it. I came up with this example yesterday. And I asked Andreas, which is my go-to guy for these things, and he thought it was interesting.

So you would run that. And let's say that the speaker said it hit London. What should the listener understand. Where is this distribution over it? And I've sort of just sampled from it. I've sampled from his distribution over where he thinks it fell.

And I just did a few samples, but you can sort of notice the suspicious gap over here. And if I sample 100 points, you'll notice that it's going to be in London-- so between one and one. This might take it a minute.

But what you'll end up seeing is that it's probably anywhere in London. If someone said to you, it fell in London, then you infer that it's anywhere in London except Big Ben. Because if it was

Big Ben, you would have said Big Ben.

It's not perfect. I mean, there are some samples that get there. It's actually some shifting distribution. But yeah. If you took some heat map of this thing, what you would end up with is that there's some sort of suspicious emptiness over here. In this case, there's also a bit of an emptiness over there. But in the limit, you'll get that.

So we did plan B, which was to zoom through a few things very quickly. I'm sure you guys didn't fully grok the details. And that's OK. What I wanted to do with plan B was to give you a taste of what is possible and how you would go about writing models.

The important things to remember here is that probabilistic programs are great tools for capturing lots of rich structure-- anything from physics to psychology to language games to grammar to vision. Church is a particularly useful language for teaching yourselves about these things. There's a lot of different models that you can play with on probmods.org. You can write down a generative model very easily to describe how you think the world works, and then you can put that in an inference engine and try to figure out what you actually saw. OK. So thank you.

[APPLAUSE]