```julia
begin
    import Pkg
    # activate the shared project environment
    Pkg.activate(Base.current_project())
    using Omega, Distributions, UnicodePlots, OmegaExamples
end
```

```
Activating project at `~/Documents/GitHub/Omega.jl/OmegaExamples`
```

Cognition and conditioning

We have built up a tool set for constructing probabilistic generative models. These can represent knowledge about causal processes in the world: running one of these programs generates a particular outcome by sampling a "history" for that outcome. However, the power of a causal model lies in the flexible ways it can be used to reason about the world. In the last chapter we used generative models to reason about outcomes from initial conditions. Generative models also enable reasoning in other ways. For instance, if we have a generative model in which X is the output of a process that depends on Y (say `X = cool_function(Y)`) we may ask: "Assuming I have observed a certain X, what must Y have been?" That is we can reason backward from outcomes to initial conditions. More generally, we can make hypothetical assumptions and reason about the generative history: "assuming something, how did the generative model run?" In this section, we describe how a wide variety of such hypothetical inferences can be made from a single generative model by conditioning the model on an assumed or observed fact.

Much of cognition can be understood in terms of conditional inference. In its most basic form, causal attribution is conditional inference: given some observed effects, what were the likely causes? Predictions are conditional inferences in the opposite direction: given that I have observed some cause, what are its likely effects? These inferences can be described by conditioning a probabilistic program that expresses a causal model. The acquisition of that causal model, or learning, is also conditional inference at a higher level of abstraction: given our general knowledge of how causal relations operate in the world, and some observed events in which candidate causes and effects co-occur in various ways, what specific causal relations are likely to hold between these observed variables?

To see how the same concepts apply in a domain that is not usually thought of as causal, consider language. The core questions of interest in the study of natural language are all at heart conditional inference problems. Given beliefs about the structure of my language, and an observed sentence, what should I believe about the syntactic structure of that sentence? This is the parsing problem. The complementary problem of speech production is related: given the structure of my language (and beliefs about others' beliefs about that), and a particular thought I want to express, how should I encode the thought? Finally, the acquisition problem: given some data from a particular language, and perhaps general knowledge about universals of grammar, what should we believe about that language's structure? This problem is simultaneously the problem facing the linguist and the child trying to learn a language.

Parallel problems of conditional inference arise in visual perception, social cognition, and virtually every other domain of cognition. In visual perception, we observe an image or image sequence that is the result of rendering a three-dimensional physical scene onto our two-dimensional retinas. A probabilistic program can model both the physical processes at work in the world that produce natural scenes, and the imaging processes (the "graphics") that generate images from scenes. Perception can then be seen as conditioning this program on some observed output image and inferring the scenes most likely to have given rise to it.

When interacting with other people, we observe their actions, which result from a planning process, and often want to guess their desires, beliefs, emotions, or future actions. Planning can be modeled as a program that takes as input an agent's mental states (beliefs, desires, etc.) and produces action sequences—for a rational agent, these will be actions that are likely to produce the agent's desired states reliably and efficiently. A rational agent can plan their actions by conditional inference to infer what steps would be most likely to achieve their desired state. Action understanding, or interpreting an agent's observed behavior, can be expressed as conditioning a planning program (a "theory of mind") on observed actions to infer the mental states that most likely gave rise to those actions, and to predict how the agent is likely to act in the future.

Hypothetical Reasoning

Suppose that we know some fixed fact, and we wish to consider hypotheses about how a generative model could have given rise to that fact. In Omega, we can use conditional random variables to describe a distribution under some assumptions or conditions.

Consider the following simple generative model:

```
A = 2903940729156277861@Distributions.Bernoulli{Float64}(p=0.5)
```

```julia
A = @~ Bernoulli()
```

```
B = -8680774252460076235@Distributions.Bernoulli{Float64}(p=0.5)
```

```julia
B = @~ Bernoulli()
```

```
C = -3766641597178692317@Distributions.Bernoulli{Float64}(p=0.5)
```

```julia
C = @~ Bernoulli()
```

```
model =
+ₚ(+ₚ(2903940729156277861@Distributions.Bernoulli{Float64}(p=0.5), -8680774252460076235@Dis
```

```julia
model = A .+ B .+ C
```

```
                 ┌                                        ┐
   [0.0, 0.5) ┤■■■■■■■■■■■ 111
   [0.5, 1.0) ┤ 0
   [1.0, 1.5) ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 394
   [1.5, 2.0) ┤ 0
   [2.0, 2.5) ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 371
   [2.5, 3.0) ┤ 0
   [3.0, 3.5) ┤■■■■■■■■■■■■ 124
                 └                                        ┘
                               Frequency
```

```julia
viz(randsample(model, 1000))
```

The process described in model samples three numbers and adds them. The value of the final expression here is 0, 1, 2 or 3. A priori, each of the variables `A`, `B`, `C` has 0.5 probability of being 1 or 0. However, suppose that we know that the sum `model` is equal to 3. How does this change the space of possible values that variable `A` could have taken? `A` (and `B` and `C`) must be *equal* to 1 for this result to happen. We can see this in the following Omega inference, where we use `|ᶜ` to express the desired assumption (ie., to condition on random variables):

```
A_cnd =
  Conditional(2903940729156277861@Distributions.Bernoulli{Float64}(p=0.5), ==ₚ(+ₚ(+ₚ(2903940
```

```julia
A_cnd = A |ᶜ (model .== 3)
```

```
          ┌                                        ┐
   true ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 100
          └                                        ┘
```

```julia
viz(randsample(A_cnd, 100))
```

The output here describes appropriate beliefs about the likely value of `A`, conditioned on `model` being equal to 3.

Now suppose that we condition on `model` being greater than or equal to 2. Then `A` need not be 1, but it is more likely than not to be. The corresponding plot shows the appropriate distribution of beliefs for `A` conditioned on this new fact:

```
A_cnd_new =
 Conditional(2903940729156277861@Distributions.Bernoulli{Float64}(p=0.5), >=ₚ(+ₚ(+ₚ(2903940
```

```julia
A_cnd_new = A |ᶜ (model .>= 2)
```

```
false ┤■■■■■■■■■■ 20
 true ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 80
```

```julia
viz(randsample(A_cnd_new, 100))
```

Rejection Sampling

How can we imagine answering a hypothetical such as those above? We have already seen how to get a sample from a generative model. We can get conditional samples by sampling from the entire model, but only keeping the sample if the value passed to the condition is true. For instance, to sample from the above model "`A` given that `model` is greater than or equal to 2" we could:

```
A_ = -4324112551805911731@Distributions.Bernoulli{Float64}(p=0.5)
```

```julia
A_ = @~ Bernoulli()
```

```
B_ = -7818619052299339989@Distributions.Bernoulli{Float64}(p=0.5)
```

```julia
B_ = @~ Bernoulli()
```

```
C_ = -6331607035421094195@Distributions.Bernoulli{Float64}(p=0.5)
```

```julia
C_ = @~ Bernoulli()
```

```
D =
+ₚ(+ₚ(-4324112551805911731@Distributions.Bernoulli{Float64}(p=0.5), -7818619052299339989@Di
```

```julia
D = A_ .+ B_ .+ C_
```

```
take_sample (generic function with 1 method)
```

```julia
take_sample(ω) = (D(ω) >= 2) ? A_(ω) : take_sample(defω())
```

```
false ┤■■■■■■■■■■■■■ 249
 true ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 751
```

```julia
viz(randsample(take_sample, 1000))
```

Notice that we have used recursion to sample the model repeatedly until `D >= 2` is true, and we then return `A_`: we generate and test until the condition is satisfied. This process is known as *rejection sampling*.

Bayes' Rule

One of the most famous rules of probability is Bayes' rule, which states:

$$P(h \mid d) = \frac{P(d \mid h)\, P(h)}{P(d)}$$

It is first worth noting that this follows immediately from the definition of conditional probability:

$$P(h \mid d) = \frac{P(d,h)}{P(d)} = \frac{P(d,h)\, P(h)}{P(d)P(h)} = \frac{P(d \mid h)\, P(h)}{P(d)}$$

Next, we can ask what this rule means in terms of sampling processes. Consider the program:

```
observed_data = true
```
```julia
observed_data = true
```

```
prior = 4138349364125604845@Distributions.Bernoulli{Float64}(p=0.5)
```
```julia
prior = @~ Bernoulli()
```

```
likelihood (generic function with 1 method)
```
```julia
likelihood(h) = ifelse.(h, (@~ Bernoulli(0.9)), (@~ Bernoulli(0.1)))
```

```
posterior =
  Conditional(4138349364125604845@Distributions.Bernoulli{Float64}(p=0.5), ==ₚ(ifelseₚ(41383
```
```julia
posterior = prior |ᶜ (likelihood(prior) .== observed_data)
```

```
                                                          ┐
  false ┤■■■■ 103
   true ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 897
                                                          ┘
```
```julia
viz(randsample(posterior, 1000))
```

We have generated a value, the hypothesis, from some distribution called the `prior`, then used an observation function `likelihood`, the probability of such an observation function is usually called the likelihood. Finally we have returned the hypothesis, conditioned on the observation being equal to some observed data — this conditional distribution is called the posterior. This is a typical setup in which Bayes' rule is used.

Bayes' rule simply says that, in special situations where the model decomposes nicely into a part "before" the value to be returned (hypothesis) and a part "after" the value to be returned, then the conditional probability can be expressed simply in terms of the prior and likelihood components of the model. This is often a useful way to think about conditional inference in simple settings. However, we will see examples as we go along where Bayes' rule doesn't apply in a simple way, but the conditional distribution is equally well understood in other terms.

Conditions and Observations

A very common pattern is to condition directly on the value of a sample from some distribution. For instance here we try to recover a true number from a noisy observation of it:

```
true_X = 2415469375174439251@Distributions.Normal{Float64}(μ=0.0, σ=1.0)
```

```
true_X = @~ Normal(0, 1)
```

```
obs_X (generic function with 1 method)
```

```
obs_X(ω) = (0 ~ Normal(true_X(ω), 0.1))(ω)
```

Alternatively, a more convenient way to define `obs_X` is - `obs_X = @~ Normal(true_X, 0.1)` in Omega.

```
obs_X_ = 0@#1
```

```
obs_X_ = 0 ~ Normal(true_X, 0.1)
```

```
(-0.833968, -0.833968)
```

```
randsample((obs_X_, obs_X))
```

```
cnd_true_X =
  Conditional(2415469375174439251@Distributions.Normal{Float64}(μ=0.0, σ=1.0), false)
```

```
cnd_true_X = true_X |ᶜ (obs_X .== 0.2)
```

```
randsample(cnd_true_X)
```

You will note that when you run the above function, it never finishes. (Why? Think about what rejection sampling tries to do here...)

Example: Reasoning about Tug of War

Imagine a game of tug of war, where each person may be strong or weak, and may be lazy or not on each match. If a person is lazy they only pull with half their strength. The team that pulls hardest will win. We assume that strength is a continuous property of an individual, and that on any match, each person has a 1 in 3 chance of being lazy.

```
strength (generic function with 1 method)
```

```
strength(person) = person ~ truncated(Normal(1, 1), 0.1, Inf)
```

```
lazy (generic function with 1 method)
```

```
lazy(n, ω) = (n ~ Bernoulli(1/3))(ω)
```

```
pulling (generic function with 1 method)
```

```
pulling(n, person, ω) = lazy(n, ω) ? strength(person)(ω) / 2 : strength(person)(ω)
```

```
total_pulling (generic function with 1 method)
```

```julia
function total_pulling(n, team, ω)
    s = 0.
    for (i, person) in enumerate(team)
        s += pulling(n+i, person, ω)
    end
    return s
end
```

```
winner (generic function with 2 methods)
```

```julia
begin
    function winner(team1, team2, ω)
        if total_pulling(0, team1, ω) > total_pulling(length(team1), team2, ω)
            return team1
        else
            return team2
        end
    end
    winner(team1, team2) = ω -> winner(team1, team2, ω)
end
```

```
(bob = 2, tom = 4)
```

```julia
begin
    team1 = (alice = 1, bob = 2)
    team2 = (sue = 3, tom = 4)
    team1_ = (alice = 1, sue = 3)
    team2_ = (bob = 2, tom = 4)
end
```

```
((alice = 1, bob = 2), (alice = 1, sue = 3))
```

```julia
randsample((winner(team1, team2), winner(team1_, team2_)))
```

Notice that `strength` is a property of a person true across many matches, while lazy isn't. Each time you run this program, however, a new "random world" will be created: people's strengths will be randomly re-generated, then used in all the matches.

We can use this to ask a variety of different questions. For instance, how likely is it that Bob is strong, given that he's been on a series of winning teams? (Note that we have added the helper function beat as in "team1 beat team2"; this just makes for more compact conditioning statements.)

```
beat (generic function with 1 method)
```

```julia
beat(team1, team2) = ω -> (winner(team1, team2)(ω) == team1)
```

```
cnd_str =
  Conditional(2@Truncated(Distributions.Normal{Float64}(μ=1.0, σ=1.0); lower=0.1, upper=Inf)
```

```julia
cnd_str = strength(team1.bob) |ᶜ beat(team1, team2)
```

```
1.332115845516819
```

```julia
randsample(cnd_str)
```

A model very similar to this was used in Gerstenberg and Goodman (2012) to predict human judgements about the strength of players in ping-pong tournaments. It achieved very accurate quantitative predictions without many free parameters.

We can form many complex queries from this simple model. We could ask how likely a team of Bob and Mary is to beat a team of Jim and Sue, given that Mary is at least as strong as sue, and Bob beat Jim in a previous direct match up:

```
(jim = 6, sue = 3)
```

```julia
begin
    team_A = (bob = 2, mary = 5)
    team_B = (jim = 6, sue = 3)
end
```

```
condition (generic function with 1 method)
```

```julia
condition(ω) =
(strength(team_A.mary)(ω) >= strength(team_B.sue)(ω)) & beat((bob=2,), (jim=6,))(ω)
```

```
cnd_beat =
  Conditional(#3 (generic function with 1 method), condition (generic function with 1 method
```

```julia
cnd_beat = beat(team_A, team_B) |ᶜ condition
```

```
true
```

```julia
randsample(cnd_beat)
```

Example: Causal Inference in Medical Diagnosis

This classic Bayesian inference task is a special case of conditioning. Kahneman and Tversky, and Gigerenzer and colleagues, have studied how people make simple judgments like the following:

The probability of breast cancer is 1% for a woman at 40 who participates in a routine screening. If a woman has breast cancer, the probability is 80% that she will have a positive mammography. If a woman does not have breast cancer, the probability is 9.6% that she will also have a positive mammography. A woman in this age group had a positive mammography in a routine screening. What is the probability that she actually has breast cancer?

What is your intuition? Many people without training in statistical inference judge the probability to be rather high, typically between 0.7 and 0.9. The correct answer is much lower, less than 0.1, as we can see by running the following:

```
breast_cancer = 3318624165727911215@Distributions.Bernoulli{Float64}(p=0.01)
```

```julia
breast_cancer = @~ Bernoulli(0.01)
```

```
positive_mammogram (generic function with 1 method)
```

```julia
positive_mammogram(ω) =
    breast_cancer(ω) ? (@~ Bernoulli(0.8))(ω) : (@~ Bernoulli(0.096))(ω)
```

```
breast_cancer_cond =
  Conditional(3318624165727911215@Distributions.Bernoulli{Float64}(p=0.01), positive_mammogr
```

```julia
breast_cancer_cond = breast_cancer |ᶜ positive_mammogram
```

```
     ┌                                       ┐
false┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 930
 true┤■■■ 70
     └                                       ┘
```

```
viz(randsample(breast_cancer_cond, 1000))
```

Tversky and Kahneman (1974) named this kind of judgment *error base rate neglect*, because in order to make the correct judgment, one must realize that the key contrast is between the *base rate* of the disease, 0.01 in this case, and the false alarm rate or probability of a positive mammogram given no breast cancer, 0.096. The false alarm rate (or FAR for short) seems low compared to the probability of a positive mammogram given breast cancer (the likelihood), but what matters is that it is almost ten times higher than the base rate of the disease. All three of these quantities are needed to compute the probability of having breast cancer given a positive mammogram using Bayes' rule for posterior conditional probability:

$$P(cancer \mid positive\, mammogram) = \frac{P(positive\, mammogram \mid cancer) \times P(cancer)}{P(positive\, mammogram)}$$

$$= \frac{0.8 \times 0.01}{0.8 \times 0.01 + 0.096 \times 0.99} = 0.078$$

Gigerenzer and Hoffrage (1995) showed that this kind of judgment can be made much more intuitive to untrained reasoners if the relevant probabilities are presented as "natural frequencies", or the sizes of subsets of relevant possible outcomes:

On average, ten out of every 1000 women at age 40 who come in for a routine screen have breast cancer. Eight out of those ten women will get a positive mammography. Of the 990 women without breast cancer, 95 will also get a positive mammography. We assembled a sample of 1000 women at age 40 who participated in a routine screening. How many of those who got a positive mammography do you expect to actually have breast cancer?

Now one can practically read off the answer from the problem formulation: 8 out of 103 ($95 + 8$) women in this situation will have breast cancer.

Gigerenzer (along with Cosmides, Tooby and other colleagues) has argued that this formulation is easier because of evolutionary and computational considerations: human minds have evolved to count and compare natural frequencies of discrete events in the world, not to add, multiply and divide decimal probabilities. But this argument alone cannot account for the very broad human capacity for causal reasoning. We routinely make inferences for which we haven't stored up sufficient frequencies of events observed *in the world*. (And often for which no one has told us the relevant frequencies, although perhaps we have been told about degrees of causal strength or base rates in the form of probabilities or other linguistic encoding).

However, the basic idea that the mind is good at manipulating frequencies of situations, but bad at arithmetic on continuous probability values, can be extended to cope with novel situations if the frequencies that are manipulated can be frequencies of *imagined* situations.

Recall that probabilistic programs explicitly give instructions for sampling imagined situations, and only implicitly specify probability distributions. If human inference is similar to an Omega inference then it would readily create and manipulate imagined situations, and this could explain both why the frequency framing of Bayesian probability judgment is natural to people and how people cope with rarer and more novel situations.

Selecting just the 106 hypothetical cases of women with a positive mammogram, and computing the fraction of those who also have breast cancer (7/106), corresponds exactly to rejection sampling (used in `randsample`). Thus, we have used the causal representation in the above program to manufacture frequencies which can be used to arrive at the inference that relatively few women with positive mammograms actually have breast cancer.

Yet unlike rejection sampler, people are quite bad at reasoning in this scenario. Why? One answer is that people don't represent their knowledge in quite the form of this simple program. Indeed, Krynski and Tenenbaum (2007) have argued that human statistical judgment is fundamentally based on conditioning more explicit causal models: they suggested that "base rate neglect" and other judgment errors may occur when people are given statistical information that cannot be easily mapped to the parameters of the causal models they intuitively adopt to describe the situation. In the above example, they suggested that the notion of a false alarm rate is not intuitive to many people—particularly when the false alarm rate is ten times higher than the base rate of the disease that the test is intended to diagnose! They showed that "base rate neglect" could be eliminated by reformulating the breast cancer problem in terms of more intuitive causal models. For example, consider their version of the breast cancer problem (the exact numbers and wording differed slightly):

1% of women at age 40 who participate in a routine screening will have breast cancer. Of those with breast cancer, 80% will receive a positive mammogram. 20% of women at age 40 who participate in a routine screening will have a benign cyst. Of those with a benign cyst, 50% will receive a positive mammogram due to unusually dense tissue of the cyst. All others will receive a negative mammogram. Suppose that a woman in this age group has a positive mammography in a routine screening. What is the probability that she actually has breast cancer?

This question is easy for people to answer—empirically, just as easy as the frequency-based formulation given above. We may conjecture this is because the relevant frequencies can be computed from a simple inference on the following more intuitive causal model:

```
breast_cancer_ = 1605837118518757O9@Distributions.Bernoulli{Float64}(p=0.01)
```

```julia
breast_cancer_ = @~ Bernoulli(0.01)
```

```
benign_cyst = 3795508743981037182@Distributions.Bernoulli{Float64}(p=0.2)
```

```julia
benign_cyst = @~ Bernoulli(0.2)
```

```
positive_mammogram_ =
|ₚ(&ₚ(1605837118518757O9@Distributions.Bernoulli{Float64}(p=0.01), 6135021911737329257@Dist
```

```julia
positive_mammogram_ =
        (breast_cancer_ .& @~ Bernoulli(0.8)) .| (benign_cyst .& @~ Bernoulli())
```

```
breast_cancer_cond_ =
  Conditional(1605837118518757O9@Distributions.Bernoulli{Float64}(p=0.01), |ₚ(&ₚ(160583711851
```

```
breast_cancer_cond_ = breast_cancer_ |ᶜ positive_mammogram_
```

```
      ┌                                        ┐
false ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 931
 true ┤■■■ 69
      └                                        ┘
```

```
viz(randsample(breast_cancer_cond_, 1000))
```

Because this causal model is more intuitive to people, they can imagine the appropriate situations, despite having been given percentages rather than frequencies. What makes this causal model more intuitive than the one above with an explicitly specified false alarm rate? Essentially we have replaced probabilistic dependencies on the "non-occurrence" of events (e.g., the dependence of a positive mammogram on *not* having breast cancer) with dependencies on explicitly specified alternative causes for observed effects (e.g., the dependence of a positive mammogram on having a benign cyst).

A causal model framed in this way can scale up to significantly more complex situations. Recall our more elaborate medical diagnosis network from the previous section, which was also framed in this way using noisy-logical functions to describe the dependence of symptoms on disease:

```
-6610178191136775313@Distributions.Bernoulli{Float64}(p=0.1)
```

```
begin
    lung_cancer = @~ Bernoulli(0.01)
    TB = @~ Bernoulli(0.005)
    stomach_flu = @~ Bernoulli(0.1)
    cold = @~ Bernoulli(0.2)
    other = @~ Bernoulli(0.1)
end
```

```
cough =
|ₚ(&ₚ(8278565657186476366@Distributions.Bernoulli{Float64}(p=0.2), 2904482214463932625@Dist
```

```
cough = pw(|,
        (cold .& @~ Bernoulli()),
        (lung_cancer .& @~ Bernoulli(0.3)),
        (TB .& @~ Bernoulli(0.7)),
        (other .& @~ Bernoulli(0.01))
)
```

```
fever =
|ₚ(&ₚ(8278565657186476366@Distributions.Bernoulli{Float64}(p=0.2), 2084361662601316793@Dist
```

```
fever = pw(|,
    (cold .& @~ Bernoulli(0.3)),
    (stomach_flu .& @~ Bernoulli()),
    (TB .& @~ Bernoulli(0.1)),
    (other .& @~ Bernoulli(0.01))
)
```

```
chest_pain =
|ₚ(&ₚ(6195411251123093114@Distributions.Bernoulli{Float64}(p=0.01), -6829807023931004829@Di
```

```
chest_pain = pw(|,
    (lung_cancer .& @~ Bernoulli()),
    (TB .& @~ Bernoulli()),
    (other .& @~ Bernoulli(0.01))
)
```

```
shortness_of_breath =
|ₚ(&ₚ(6195411251123093114@Distributions.Bernoulli{Float64}(p=0.01), -5291495935792232557@Di
```

```
shortness_of_breath = pw(|,
    (lung_cancer .& @~ Bernoulli()),
    (TB .& @~ Bernoulli(0.2)),
    (other .& @~ Bernoulli(0.01))
)
```

```
lung_cancer_cond =
  Conditional(6195411251123093114@Distributions.Bernoulli{Float64}(p=0.01), &ₚ(|ₚ(&ₚ(8278565
```

```
lung_cancer_cond = lung_cancer |ᶜ pw(&, cough, chest_pain, shortness_of_breath)
```

```
TB_cond =
  Conditional(-3276005235140423010@Distributions.Bernoulli{Float64}(p=0.005), &ₚ(|ₚ(&ₚ(82785
```

```
TB_cond = TB |ᶜ pw(&, cough, chest_pain, shortness_of_breath)
```

```
         ┌                                        ┐
   false ┤■■■■■■■■■■■■■■■■ 284
    true ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 716
         └                                        ┘
```

```
viz(randsample(lung_cancer_cond, 1000))
```

```
         ┌                                        ┐
   false ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 717
    true ┤■■■■■■■■■■■■■■■■ 283
         └                                        ┘
```

```
viz(randsample(TB_cond, 1000))
```

You can use this model to infer conditional probabilities for any subset of diseases conditioned on any pattern of symptoms. Try varying the symptoms in the conditioning set or the diseases in the inference, and see how the model's inferences compare with your intuitions. For example, what happens to inferences about lung cancer and TB in the above model if you remove chest pain and shortness of breath as symptoms? (Why? Consider the alternative explanations.) More generally, we can condition on any set of events – any combination of symptoms and diseases – and query any others. We can also condition on the negation of an event: how does the probability of lung cancer (versus TB) change if we observe that the patient does not have a fever (i.e. condition on `!ₚ(fever)`), does not have a cough, or does not have either symptom?

As we discussed above, Omega thus effectively encodes the answers to a very large number of possible questions in a very compact form. In the program above, there are $3^9 = 19683$ possible simple conditions corresponding to conjunctions of events or their negations (because the program has 9 stochastic Boolean-valued functions, each of which can be observed true, observed false, or not observed). Then for each of those conditions there are a roughly comparable number of queries, corresponding to all the possible conjunctions of variables that can be in the return value expression.

This makes the total number of simple questions encoded on the order of 100 million. We are beginning to see the sense in which probabilistic programming provides the foundations for constructing a *language of thought*: a finite system of knowledge that compactly and efficiently supports an infinite number of inference and decision tasks.

Expressing our knowledge as a probabilistic program of this form also makes it easy to add in new relevant knowledge we may acquire, without altering or interfering with what we already know. For instance, suppose we decide to consider behavioral and demographic factors that might contribute causally to whether a patient has a given disease:

```
|ₚ(-8165672222873044650@Distributions.Bernoulli{Float64}(p=0.2), &ₚ(5322811306742226054@Dis
```

```
begin
    works_in_hospital = @~ Bernoulli(0.01)
    smokes = @~ Bernoulli(0.2)
    lung_cancer_ = (@~ Bernoulli(0.01)) .| (smokes .& @~ Bernoulli(0.02))
    TB_ = (@~ Bernoulli(0.005)) .| (works_in_hospital .& @~ Bernoulli(0.01))
    cold_ = (@~ Bernoulli(0.2)) .| (works_in_hospital .& @~ Bernoulli(0.25))
end
```

```
cough_ =
|ₚ(&ₚ(|ₚ(-8165672222873044650@Distributions.Bernoulli{Float64}(p=0.2), &ₚ(532281130674222‍6
```

```
cough_ = pw(|,
    (cold_ .& @~ Bernoulli()),
    (lung_cancer_ .& @~ Bernoulli(0.3)),
    (TB_ .& @~ Bernoulli(0.7)),
    (other .& @~ Bernoulli(0.01))
)
```

```
fever_ =
|ₚ(&ₚ(|ₚ(-8165672222873044650@Distributions.Bernoulli{Float64}(p=0.2), &ₚ(532281130674222‍6
```

```
fever_ = pw(|,
    (cold_ .& @~ Bernoulli(0.3)),
    (stomach_flu .& @~ Bernoulli()),
    (TB_ .& @~ Bernoulli(0.1)),
    (other .& @~ Bernoulli(0.01))
)
```

```
chest_pain_ =
|ₚ(&ₚ(|ₚ(8922206377335554953@Distributions.Bernoulli{Float64}(p=0.01), &ₚ(-129589811332921‍0
```

```
chest_pain_ = pw(|,
    (lung_cancer_ .& @~ Bernoulli()),
    (TB_ .& @~ Bernoulli()),
    (other .& @~ Bernoulli(0.01))
)
```

```
shortness_of_breath_ =
|ₚ(&ₚ(|ₚ(8922206377335554953@Distributions.Bernoulli{Float64}(p=0.01), &ₚ(-129589811332921‍0
```

```
shortness_of_breath_ = pw(|,
    (lung_cancer_ .& @~ Bernoulli()),
    (TB_ .& @~ Bernoulli(0.2)),
    (other .& @~ Bernoulli(0.01))
)
```

```
lung_cancer_cond_ =
  Conditional(|ₚ(8922206377335554953@Distributions.Bernoulli{Float64}(p=0.01), &ₚ(-129589811
```

```
lung_cancer_cond_ = lung_cancer_ |ᶜ pw(&, cough_, chest_pain_, shortness_of_breath_)
```

```
TB_cond_ =
  Conditional(|ₚ(1362227118880248443@Distributions.Bernoulli{Float64}(p=0.005), &ₚ(532281130
```

```
TB_cond_ = TB_ |ᶜ pw(&, cough_, chest_pain_, shortness_of_breath_)
```

```
         ┌                                        ┐
   false ┤■■■■■■■■■■■ 236
    true ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 764
         └                                        ┘
```

```
viz(randsample(lung_cancer_cond_, 1000))
```

```
         ┌                                        ┐
   false ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 772
    true ┤■■■■■■■■■■ 228
         └                                        ┘
```

```
viz(randsample(TB_cond_, 1000))
```

Under this model, a patient with coughing, chest pain and shortness of breath is likely to have either lung cancer or TB. Modify the above code to see how these conditional inferences shift if you also know that the patient smokes or works in a hospital (where they could be exposed to various infections, including many worse infections than the typical person encounters). More generally, the causal structure of knowledge representation in a probabilistic program allows us to model intuitive theories that can grow in complexity continually over a lifetime, adding new knowledge without bound.