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

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

```
Dirichlet = OmegaExamples.Dirichlet
```

```
Dirichlet = OmegaExamples.Dirichlet
```

Human knowledge is organized hierarchically into levels of abstraction. For instance, the most common or basic-level categories (e.g. dog, car) can be thought of as abstractions across individuals, or more often across subordinate categories (e.g., poodle, Dalmatian, Labrador, and so on). Multiple basic-level categories in turn can be organized under superordinate categories: e.g., dog, cat, horse are all animals; car, truck, bus are all vehicles. Some of the deepest questions of cognitive development are: How does abstract knowledge influence learning of specific knowledge? How can abstract knowledge be learned? In this section we will see how such hierarchical knowledge can be modeled with hierarchical generative models: generative models with uncertainty at several levels, where lower levels depend on choices at higher levels.

Learning a Shared Prototype: Abstraction at the Basic Level

Hierarchical models allow us to capture the shared latent structure underlying observations of multiple related concepts, processes, or systems – to abstract out the elements in common to the different sub-concepts, and to filter away uninteresting or irrelevant differences. Perhaps the most familiar example of this problem occurs in learning about categories. Consider a child learning about a basic-level kind, such as dog or car. Each of these kinds has a prototype or set of characteristic features, and our question here is simply how that prototype is acquired.

The task is challenging because real-world categories are not homogeneous. A basic-level category like dog or car actually spans many different subtypes: e.g., poodle, Dalmatian, Labrador, and such, or sedan, coupe, convertible, wagon, and so on. The child observes examples of these sub-kinds or subordinate-level categories: a few poodles, one Dalmatian, three Labradors, etc. From this data she must infer what it means to be a dog in general, in addition to what each of these different kinds of dog is like. Knowledge about the prototype level includes understanding what it means to be a prototypical dog and what it means to be non-prototypical, but still a dog. This will involve understanding that dogs come in different breeds which share features between them, but also differ systematically as well.

As a simplification of this situation consider the following generative process. We will draw marbles out of several different bags. There are five marble colors. Each bag contains a certain mixture of colors. This generative process is represented in the following example:

```
colours =  [:black, :blue, :green, :orange, :red]
```

```
colours = [:black, :blue, :green, :orange, :red]
```

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

```
colour_probs(n, ω, i) = ((@uid, i...)~ Dirichlet(n, 1))(ω)
```

```
[2, 3, 5, 2, 1, 2, 5, 4, 5, 1]
```

```
randsample(ω -> (@~ Categorical(colour_probs(5, ω, 1)))(ω), 10)
```

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

```
make_bag(i, ω, colours, n = @uid) =
    (pget(colours) ∘ (n~ Categorical(colour_probs(length(colours), ω, i))))(ω)
```

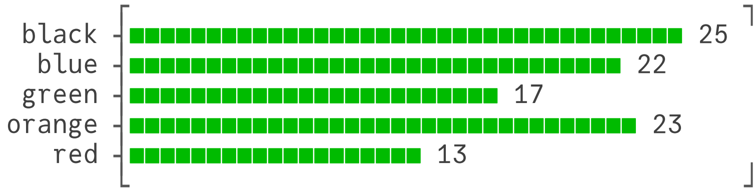
```
:green
```

```
randsample(ω -> make_bag(1, ω, colours))
```

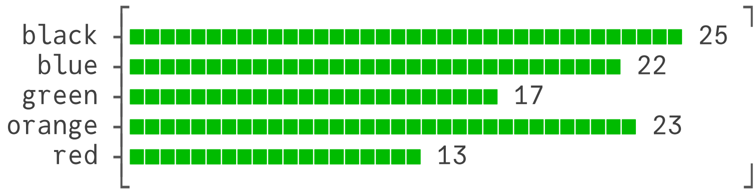
```
((-339660163576321312, 1), (384981856825085986, 2))
```

```
bagA, bagB = (@uid, 1), (@uid, 2)
```

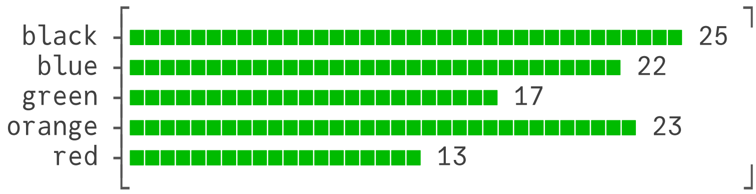
```
ωs = map(i -> defω(), 1:100);
```



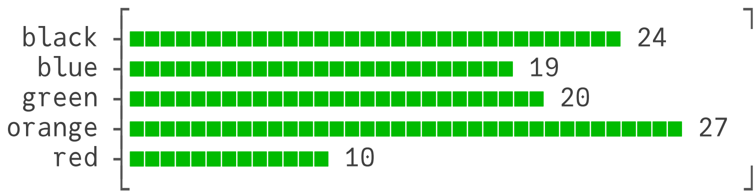

```
let
    bagA_samples = map(ω -> make_bag(bagA, ω, colours), ωs)
    viz(string.(bagA_samples))
end
```




```
let
    bagA_samples = map(ω -> make_bag(bagA, ω, colours), ωs)
    viz(string.(bagA_samples))
end
```




```
let
    bagA_samples = map(ω -> make_bag(bagA, ω, colours), ωs)
    viz(string.(bagA_samples))
end
```




```
let
    bagB_samples = map(ω -> make_bag(bagB, ω, colours), ωs)
    viz(string.(bagB_samples))
end
```

Here, notice that for the same `ω` and bag, `make_bag` returns the same value in every pass. As this examples shows, memoization is particularly useful when writing hierarchical models because it allows us to associate arbitrary random draws with categories across entire runs of the program. In this case it allows us to associate a particular mixture of marble colors with each bag. The mixture is drawn once, and then remains the same thereafter for that bag. Intuitively, you can see how each sample is sufficient to learn a lot about what that bag is like; there is typically a fair amount of similarity between the empirical color distributions in each of the four samples from `bagA`. In contrast, you should see a different distribution of samples from `bagB`.

Now let's explore how this model learns about the contents of different bags. We represent the results of learning in terms of the posterior predictive distribution for each bag: a single hypothetical draw from the bag. We will also draw a sample from the posterior predictive distribution on a new bag, for which we have had no observations.

```
obs =
  Dict(2 ⇒ [:blue, :green, :blue, :blue, :blue, :red], 3 ⇒ [:blue, :orange], 1 ⇒ [:blue, :bl
```

```
obs = Dict(1 => [:blue, :blue, :black, :blue, :blue, :blue],
           2 => [:blue, :green, :blue, :blue, :blue, :red],
           3 => [:blue, :orange])
```

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

```
rand_make_bag(i, ω, colours, k) = map(n -> make_bag(i, ω, colours, n), 1:k)
```

```
obs_fn =
&ₚ(==ₚ(ᵛ#21, [:blue, :blue, :black, :blue, :blue, :blue]), ==ₚ(ᵛ#22, [:blue, :green, :blue,
```

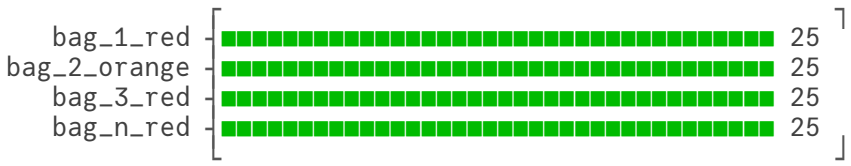
```
obs_fn = pw(&,
    pw(==, Variable(ω -> rand_make_bag(1, ω, colours, length(obs[1]))),  obs[1]),
    pw(==, Variable(ω -> rand_make_bag(2, ω, colours, length(obs[2]))), obs[2]),
    pw(==, Variable(ω -> rand_make_bag(3, ω, colours, length(obs[3]))), obs[3])
)
```

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

```
bags(ω, colours) = (
    bag_1 = make_bag(1, ω, colours),
    bag_2 = make_bag(2, ω, colours),
    bag_3 = make_bag(3, ω, colours),
    bag_n = make_bag(@uid, ω, colours)
)
```

```
predictives =
  Conditional(#27 (generic function with 1 method), &ₚ(==ₚ(ᵛ#21, [:blue, :blue, :black, :blu
```

```
predictives = (ω -> bags(ω, colours)) |ᶜ obs_fn
```




```
viz_marginals(randsample(predictives, 25, alg=MH))
```

In all cases there is a fair amount of residual uncertainty about what other colors might be seen. Nothing significant is learned about the new bag as it has no observations. This generative model describes the prototypical mixture in each bag, but it does not attempt learn a common higher-order prototype. It is like learning separate prototypes for subordinate classes *poodle*, *Dalmatian*, and *Labrador*, without learning a prototype for the higher-level kind *dog*.

Let us introduce another level of abstraction: a global prototype that provides a prior on the specific mixtures of each bag.

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

```
ϕ(colours) = @~ OmegaExamples.Dirichlet(length(colours), 1)
```

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

```
prototype(ϕ, ω) = ϕ(ω) .* 5
```

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

```
colour_probs_global(i, ω) = (i~ OmegaExamples.Dirichlet(prototype(ϕ(colours), ω)))(ω)
```

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

```
make_bag_global(i, ω, colours, n = @uid) =
    (pget(colours) ∘ (n~ Categorical(colour_probs_global(i, ω))))(ω)
```

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

```
rand_make_bag_gl(i, ω, colours, k) = map(n -> make_bag_global(i, ω, colours, n), 1:k)
```

```
obs_fn_global =
&ₚ(==ₚ(ᵛ#29, [:blue, :blue, :black, :blue, :blue, :blue]), ==ₚ(ᵛ#30, [:blue, :green, :blue,
```

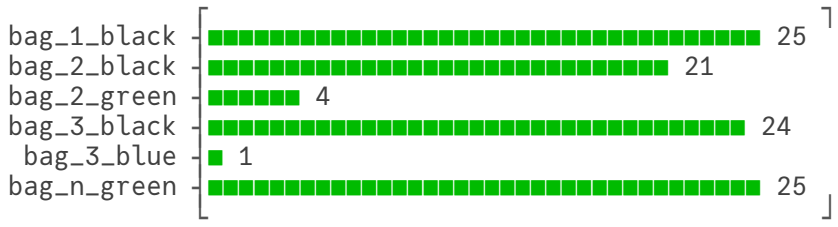
```
obs_fn_global = pw(&,
    pw(==, Variable(ω -> rand_make_bag_gl(1, ω, colours, length(obs[1]))), obs[1]),
    pw(==, Variable(ω -> rand_make_bag_gl(2, ω, colours, length(obs[2]))), obs[2]),
    pw(==, Variable(ω -> rand_make_bag_gl(3, ω, colours, length(obs[3]))), obs[3])
)
```

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

```
bags_global(ω, colours) = (
    bag_1 = make_bag_global(1, ω, colours),
    bag_2 = make_bag_global(2, ω, colours),
    bag_3 = make_bag_global(3, ω, colours),
    bag_n = make_bag_global(@uid, ω, colours)
)
```

```
predictives_global =
  Conditional(#35 (generic function with 1 method), &ₚ(==ₚ(ᵛ#29, [:blue, :blue, :black, :blu
```

```
predictives_global = (ω -> bags_global(ω, colours)) |ᶜ obs_fn_global
```




```
viz_marginals(randsample(predictives_global, 25, alg=MH))
```

Compared with inferences in the previous example, this extra level of abstraction enables faster learning: more confidence in what each bag is like based on the same observed sample. This is because all of the observed samples suggest a common prototype structure, with most of its weight on `blue` and the rest of the weight spread uniformly among the remaining colours. In particular, we now make strong inferences for bag **3** that blue is likely but orange isn't – quite different from the earlier case without a shared global prototype.

Statisticians sometimes refer to this phenomenon of inference in hierarchical models as “sharing of statistical strength”: it is as if the sample we observe for each bag also provides a weaker indirect sample relevant to the other bags. In machine learning and cognitive science this phenomenon is often called *transfer learning*. Intuitively, knowing something about bags in general allows the learner to transfer knowledge gained from draws from one bag to other bags. This example is analogous to seeing several examples of different subtypes of dogs and learning what features are in common to the more abstract basic-level dog prototype, independent of the more idiosyncratic features of particular dog subtypes.

Learning about shared structure at a higher level of abstraction also supports inferences about new bags without observing any examples from that bag: a hypothetical new bag could produce *any* colour, but is likely to have more blue marbles than any other colour. We can imagine hypothetical, previously unseen, new subtypes of dogs that share the basic features of dogs with more familiar kinds but may differ in some idiosyncratic ways.

The Blessing of Abstraction

Now let's investigate the relative learning speeds at different levels of abstraction. Suppose that we have a number of bags that all have identical prototypes: they mix red and blue in proportion $\mathbf{2:1}$. But the learner doesn't know this. She observes only one ball from each of N bags. What can she learn about an individual bag versus the population as a whole as the number of bags changes? We plot learning curves: the mean squared error (MSE) of the prototype from the true prototype for the specific level (the first bag) and the general level (global prototype) as a function of the number of observed data points. We normalize by the MSE of the first observation (from the first bag), to focus on the effects of diverse data. (Note that these MSE quantities are directly comparable because they are each derived from a Dirichlet distribution of the same size – this is often not the case in hierarchical models.)

```
c =  [:red, :blue]
```

```
c = [:red, :blue]
```

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

```
ϕ_(colours) = @~ OmegaExamples.Dirichlet(length(colours), 1)
```

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

```
bag_probs(i, ω) = (i~ OmegaExamples.Dirichlet(prototype(ϕ_(c), ω)))(ω)
```

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

```
make_bag_(i, ω, colours, n = @uid) =
    (pget(colours) ∘ (n~ Categorical(bag_probs(i, ω))))(ω)
```

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

```
obs_fn_(ω, data) = all(map(k -> make_bag_(k, ω, c, k) == data[k], 1:length(data)))
```

```
bag1_posterior (generic function with 1 method)
```

```
bag1_posterior(data) = (ω -> bag_probs(1, ω)) |ᶜ (ω -> obs_fn_(ω, data))
```

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

```
ϕ_posterior(data) = ϕ_(c) |ᶜ (ω -> obs_fn_(ω, data))
```

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

```
posterior(data) =
    ω -> (bag1 = bag1_posterior(data)(ω), global_ = ϕ_posterior(data)(ω))
```

```
data =  [:red, :red, :blue, :red, :red, :blue, :red, :red, :blue, :red, :red, :blue]
```

```
data = [:red, :red, :blue, :red, :red, :blue, :red, :red, :blue, :red, :red, :blue]
```

```
num_obs =  [1, 3, 6, 9, 12]
```

```
num_obs = [1, 3, 6, 9, 12]
```

```
posteriors =
  Vector{@NamedTuple{bag1::Vector{Float64}, global_::Vector{Float64}}}[
     1:    [(bag1 = Float64[0.780732, 0.219268], global_ = Float64[0.922529, 0.0774711]), (bag1
     2:    [(bag1 = [   more], global_ = [   more]), (bag1 = [   more], global_ = [   more]), (bag1
     3:    [(bag1 = [   more], global_ = [   more]), (bag1 = [   more], global_ = [   more]), (bag1
     4:    [(bag1 = [   more], global_ = [   more]), (bag1 = [   more], global_ = [   more]), (bag1
     5:    [(bag1 = [   more], global_ = [   more]), (bag1 = [   more], global_ = [   more]), (bag1
]
```

```julia
posteriors = map(i -> randsample(posterior(data[1:i]), 10, alg=MH), num_obs)
```

```
[0.0420838, 0.00189515, 0.0972078, 0.199716, 0.218552]
```

```julia
begin
    bag1_error =
    [mean((first.(getfield.(p, :bag1)) .- 0.66) .^ 2, dims=1)[1] for p in posteriors]
    global_error = [mean((first.(getfield.(p, :global_)) .- 0.66) .^ 2, dims=1)[1]
    for p in posteriors]
end
```

```
                ┌                                        ┐
   [0.1 , 0.15) ┤■■■■■■■■■■■■■■■■■■■ 1
   [0.15, 0.2 ) ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 2
   [0.2 , 0.25) ┤■■■■■■■■■■■■■■■■■■■ 1
   [0.25, 0.3 ) ┤■■■■■■■■■■■■■■■■■■■ 1
                └                                        ┘
                               Frequency
```

```julia
viz(bag1_error)
```

```
                ┌                                        ┐
   [0.0 , 0.05) ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 2
   [0.05, 0.1 ) ┤■■■■■■■■■■■■■■■■■■■ 1
   [0.1 , 0.15) ┤ 0
   [0.15, 0.2 ) ┤■■■■■■■■■■■■■■■■■■■ 1
   [0.2 , 0.25) ┤■■■■■■■■■■■■■■■■■■■ 1
                └                                        ┘
                               Frequency
```

```julia
viz(global_error)
```

What we see is that learning is faster at the general level than the specific level—that is that the error in the estimated prototype drops faster in the general than the specific plots. We also see that there is continued learning at the specific level, even though we see no additional samples from the first bag after the first; this is because the evolving knowledge at the general level further constrains the inferences at the specific level. Going back to our familiar categorization example, this suggests that a child could be quite confident in the prototype of "dog" while having little idea of the prototype for any specific kind of dog—learning more quickly at the abstract level than the specific level, but then using this abstract knowledge to constrain expectations about specific dogs. This dynamic depends crucially on the fact that we get very diverse evidence: let's change the above example to observe the same N examples, but coming from a single bag (instead of N bags).

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

```julia
obs_fn_same(ω, data) = all(map(k -> make_bag_(1, ω, c) == data[k], 1:length(data)))
```

```
bag1_posterior_same (generic function with 1 method)
```

```
bag1_posterior_same(data) = (ω -> bag_probs(1, ω)) |ᶜ (ω -> obs_fn_same(ω, data))
```

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

```
ϕ_posterior_same(data) = ϕ_(c) |ᶜ (ω -> obs_fn_same(ω, data))
```

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

```
posterior_same(data) =
    ω -> (bag1 = bag1_posterior_same(data)(ω), global_ = ϕ_posterior_same(data)(ω))
```

```
# posteriors_same =
        # map(i -> randsample(posterior_same(data[1:i]), 10), num_obs)
```

```
# plot
```

We now see that learning for this bag is quick, while global learning (and transfer) is slow.

In machine learning one often talks of the curse of dimensionality. The curse of dimensionality refers to the fact that as the number of parameters of a model increases (i.e. the dimensionality of the model increases), the size of the hypothesis space increases exponentially. This increase in the size of the hypothesis space leads to two related problems. The first is that the amount of data required to estimate model parameters (called the "sample complexity") increases rapidly as the dimensionality of the hypothesis space increases. The second is that the amount of computational work needed to search the hypothesis space also rapidly increases. Thus, increasing model complexity by adding parameters can result in serious problems for inference.

In contrast, we have seen that adding additional levels of abstraction (and hence additional parameters) in a probabilistic model can sometimes make it possible to learn *more* with *fewer* observations. This happens because learning at the abstract level can be quicker than learning at the specific level. Because this ameliorates the curse of dimensionality, we refer to these effects as the **blessing of abstraction**.

In general, the blessing of abstraction can be surprising because our intuitions often suggest that adding more hierarchical levels to a model increases the model's complexity. More complex models should make learning harder, rather than easier. On the other hand, it has long been understood in cognitive science that learning is made easier by the addition of *constraints* on possible hypothesis. For instance, proponents of universal grammar have long argued for a highly constrained linguistic system on the basis of learnability. Hierarchical Bayesian models can be seen as a way of introducing soft, probabilistic constraints on hypotheses that allow for the transfer of knowledge between different kinds of observations.

Learning Overhypotheses: Abstraction at the Superordinate Level

Hierarchical models also allow us to capture a more abstract and even more important "learning to learn" phenomenon, sometimes called learning *overhypotheses*. Consider how a child learns about living creatures (an example we adapt from the psychologists Liz Shipley and Rob Goldstone). We learn about specific kinds of animals – dogs, cats, horses, and more exotic creatures like elephants,

ants, spiders, sparrows, eagles, dolphins, goldfish, snakes, worms, centipedes – from examples of each kind. These examples tell us what each kind is like: Dogs bark, have four legs, a tail. Cats meow, have four legs and a tail. Horses neigh, have four legs and a tail. Ants make no sound, have six legs, no tail. Robins and eagles both have two legs, wings, and a tail; robins sing while eagles cry. Dolphins have fins, a tail, and no legs; likewise for goldfish. Centipedes have a hundred legs, no tail and make no sound. And so on. Each of these generalizations or prototypes may be inferred from seeing several examples of the species.

But we also learn about what kinds of creatures are like in *general*. It seems that certain kinds of properties of animals are characteristic of a particular kind: either every individual of a kind has this property, or none of them have it. Characteristic properties include number of legs, having a tail or not, and making some kind of sound. If one individual in a species has four legs, or six or two or eight or a hundred legs, essentially all individuals in that species have that same number of legs (barring injury, birth defect or some other catastrophe). Other kinds of properties don't pattern in such a characteristic way. Consider external color. Some kinds of animals are homogeneous in coloration, such as dolphins, elephants, sparrows. Others are quite heterogeneous in coloration: dogs, cats, goldfish, snakes. Still others are intermediate, with one or a few typical color patterns: horses, ants, eagles, worms.

This abstract knowledge about what animal kinds are like can be extremely useful in learning about new kinds of animals. Just one example of a new kind may suffice to infer the prototype or characteristic features of that kind: seeing a spider for the first time, and observing that it has eight legs, no tail and makes no sound, it is a good bet that other spiders will also have eight legs, no tail and make no sound. The specific coloration of the spider, however, is not necessarily going to generalize to other spiders. Although a basic statistics class might tell you that only by seeing many instances of a kind can we learn with confidence what features are constant or variable across that kind, both intuitively and empirically in children's cognitive development it seems that this "one-shot learning" is more the norm. How can this work? Hierarchical models show us how to formalize the abstract knowledge that enables one-shot learning, and the means by which that abstract knowledge is itself acquired (Kemp et al., 2007).

We can study a simple version of this phenomenon by modifying our bags of marbles example, articulating more structure to the hierarchical model as follows. We now have two higher-level parameters: ϕ describes the expected proportions of marble colors across bags of marbles, while α, a real number, describes the strength of the learned prior – how strongly we expect any newly encountered bag to conform to the distribution for the population prototype ϕ. For instance, suppose that we observe that bag `1` consists of all blue marbles, `2` consists of all green marbles, `3` all red, and so on. This doesn't tell us to expect a particular color in future bags, but it does suggest that bags are very regular—that all bags consist of marbles of only one color.

```
colors =  [:black, :blue, :green, :orange, :red]
```

```
1 colors = [:black, :blue, :green, :orange, :red]
```

```
observed_data =
  Dict(4 ⇒ [:orange], 2 ⇒ [:green, :green, :green, :green, :green, :green], 3 ⇒ [:red, :red,
```

```
observed_data = Dict(
    1 => [:blue, :blue, :blue, :blue, :blue, :blue],
    2 => [:green, :green, :green, :green, :green, :green],
    3 => [:red, :red, :red, :red, :red, :red],
    4 => [:orange]
    )
```

```
ϕ_s =
-8959152054995468020@OmegaExamples.Dirichlet{Vector{Float64}}([1.0, 1.0, 1.0, 1.0, 1.0])
```

```
ϕ_s = @~ Dirichlet(length(colours), 1)
```

```
α = 7293942453871372192@Distributions.Gamma{Float64}(α=2.0, θ=2.0)
```

```
α = @~ Gamma(2, 2)
```

```
prototype_s =
Base.Broadcast.BroadcastFunction(*)ₚ(-8959152054995468020@OmegaExamples.Dirichlet{Vector{Fl
```

```
prototype_s = pw(.*, ϕ_s, α)
```

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

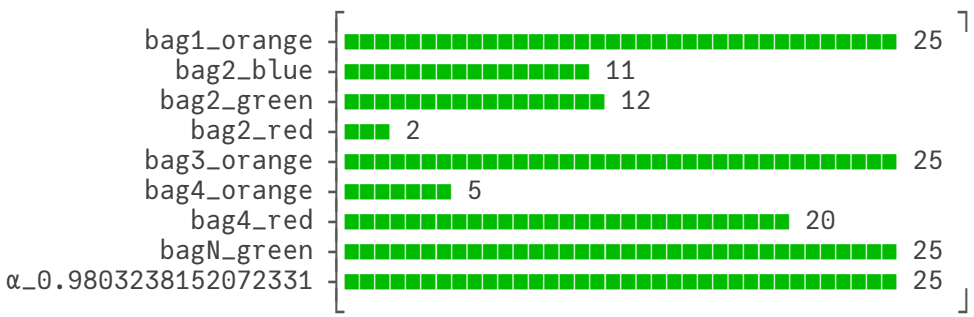
```
function make_bag_s(i, ω, k = @uid)
    color_probs = ((k, i) ~ Dirichlet(prototype_s(ω)))(ω)
    (pget(colors) ∘ ((k, i) ~ Categorical(color_probs)))(ω)
end
```

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

```
obs_fn_s(ω) = all(map(k -> manynth((i, ω) -> make_bag_s(i, ω, k),
1:length(observed_data[k]))(ω) == observed_data[k], 1:length(observed_data)))
```

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

```
predictives_s(ω) = (
    bag1 = make_bag_s(1, ω),
    bag2 = make_bag_s(2, ω),
    bag3 = make_bag_s(3, ω),
    bag4 = make_bag_s(4, ω),
    bagN = make_bag_s(@uid, ω),
    α = α(ω)
)
```




```
viz_marginals(randsample(predictives_s |ᶜ obs_fn, 25, alg=MH))
```

Consider the fourth bag, for which only one marble has been observed (orange): we see a very strong posterior predictive distribution focused on orange – a "one-shot" generalization. This posterior is

much stronger than the single observation for that bag can justify on its own. Instead, it reflects the learned overhypothesis that bags tend to be uniform in color.

To see that this is real one-shot learning, contrast with the predictive distribution for a new bag with no observations: `bagN` gives a mostly flat distribution. Little has been learned in the hierarchical model about the specific colors represented in the overall population; rather we have learned the abstract property that bags of marbles tend to be uniform in color. Hence, a single observation from a new bag is enough to make strong predictions about that bag even though little could be said prior to seeing the first observation.

We have also generated the posterior distribution on `α`, representing how strongly the prototype distribution captured in `ϕ_s`, constrains each individual bag – how much each individual bag is expected to look like the prototype of the population. You should see that the inferred values of alpha are typically significantly less than 1. This means roughly that the learned prototype in phi should exert less influence on prototype estimation for a new bag than a single observation. Hence the first observation we make for a new bag mostly determines a strong inference about what that bag is like.

Now we change the `observed_data` to

```
observed_data = Dict(
    1 => [:blue, :red, :green, :black, :red, :blue],
    2 => [:green, :red, :blue, :black, :blue, :green],
    3 => [:red, :green, :blue, :blue, :black, :green],
    4 => [:orange]
    )
```

and observe the marginals.

Intuitively, the observations for bags one, two and three should now suggest a very different overhypothesis: that marble color, instead of being homogeneous within bags but variable across bags, is instead variable within bags to about the same degree that it varies in the population as a whole. We can see this inference represented via two coupled effects. First, the inferred value of `α` is now significantly *greater* than **1**, asserting that the population distribution as a whole, `ϕ_s`, now exerts a strong constraint on what any individual bag looks like. Second, for a new 'bag4' which has been observed only once, with a single orange marble, that draw is now no longer very influential on the color distribution we expect to see from that bag; the broad distribution in `ϕ_s` exerts a much stronger influence than the single observation.

Example: The Shape Bias

One well studied overhypothesis in cognitive development is the 'shape bias': the inductive bias which develops by 24 months and which is the preference to generalize a novel label for some object to other objects of the same shape, rather than say the same color or texture. Studies by Smith and colleagues (Smith et al., 2002) have shown that this bias can be learned with very little data. They trained 17 month old children, over eight weeks, on four pairs of novel objects where the objects in each pair had the same shape but differed in color and texture and were consistently given the same novel name. First order generalization was tested by showing children an object from one of the four trained categories and asking them to choose another such object from three choice objects that

matched the shown object in exactly one feature. Children preferred the shape match. Second order generalization was also tested by showing children an object from a novel category and again children preferred the choice object which matched in shape. Smith and colleagues further found an increase in real-world vocabulary as a result of this training such that children who had been trained began to use more object names. Children had thus presumably learned something like 'shape is homogeneous within object categories' and were able to apply this inductive bias to word learning outside the lab.

We now consider a model of learning the shape bias which uses the compound Dirichlet-Discrete model that we have been discussing in the context of bags of marbles. This model for the shape bias is from (Kemp et al., 2007). Rather than bags of marbles we now have object categories and rather than observing marbles we now observe the features of an object (e.g. its shape, color, and texture) drawn from one of the object categories. Suppose that a feature from each dimension of an object is generated independently of the other dimensions and there are separate values of alpha and phi for each dimension. Importantly, one needs to allow for more values along each dimension than appear in the training data so as to be able to generalize to novel shapes, colors, etc. To test the model we can feed it training data to allow it to learn the values for the alphas and phis corresponding to each dimension. We can then give it a single instance of some new category and then ask what the probability is that the various choice objects also come from the same new category. The code below shows a model for the shape bias, conditioned on the same training data used in the Smith et al experiment. We can then ask both for draws from some category which we've seen before, and from some new category which we've seen a single instance of. One small difference from the previous models we've seen for the example case is that the alpha hyperparameter is now drawn from an exponential distribution with inverse mean 1, rather than a Gamma distribution. This is simply for consistency with the model given in the Kemp et al (2007) paper.

```
attributes =  [:shape, :color, :texture, :size]
```

```
1 attributes = [:shape, :color, :texture, :size]
```

```
values =
 (shape = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], color = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], texture
```

```
1 values = (shape = collect(0:10),
2     color = collect(0:10),
3     texture = collect(0:10),
4     size = collect(0:10))
```

```
obs_data =
 [(cat = 1, shape = 1, color = 1, texture = 1, size = 1), (cat = 1, shape = 1, color = 2, texture
```

```
1  obs_data = [
2      (cat = 1, shape = 1, color = 1, texture = 1, size = 1),
3      (cat = 1, shape = 1, color = 2, texture = 2, size = 2),
4      (cat = 2, shape = 2, color = 3, texture = 3, size = 1),
5      (cat = 2, shape = 2, color = 4, texture = 4, size = 2),
6      (cat = 3, shape = 3, color = 5, texture = 5, size = 1),
7      (cat = 3, shape = 3, color = 6, texture = 6, size = 2),
8      (cat = 4, shape = 4, color = 7, texture = 7, size = 1),
9      (cat = 4, shape = 4, color = 8, texture = 8, size = 2),
10     (cat = 5, shape = 5, color = 9, texture = 9, size = 1)
11 ]
```

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

```
function prototype_attr(i, ω, attr)
    ϕ = (@~ Dirichlet(length(values[attr]), 1))(ω)
    α = (i~ Exponential(1))(ω)
    return α.*ϕ
end
```

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

```
function make_attr_dist(i, ω, attr)
    probs = (@~ Dirichlet(prototype_attr(i, ω, attr)))(ω)
    return (i~ Categorical(probs))(ω) - 1
end
```

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

```
obs_fn_attr(ω) =
    all([make_attr_dist(d.cat, ω, attr) == d[attr] for attr in attributes for d in
    obs_data])
```

```
cat_5_shape (generic function with 1 method)
```

```
cat_5_shape(ω) = make_attr_dist(5, ω, :shape)
```

```
cat_5_color (generic function with 1 method)
```

```
cat_5_color(ω) = make_attr_dist(5, ω, :color)
```

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

```
cat_N_shape(ω) = make_attr_dist(6, ω, :shape)
```

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

```
cat_N_color(ω) = make_attr_dist(6, ω, :color)
```

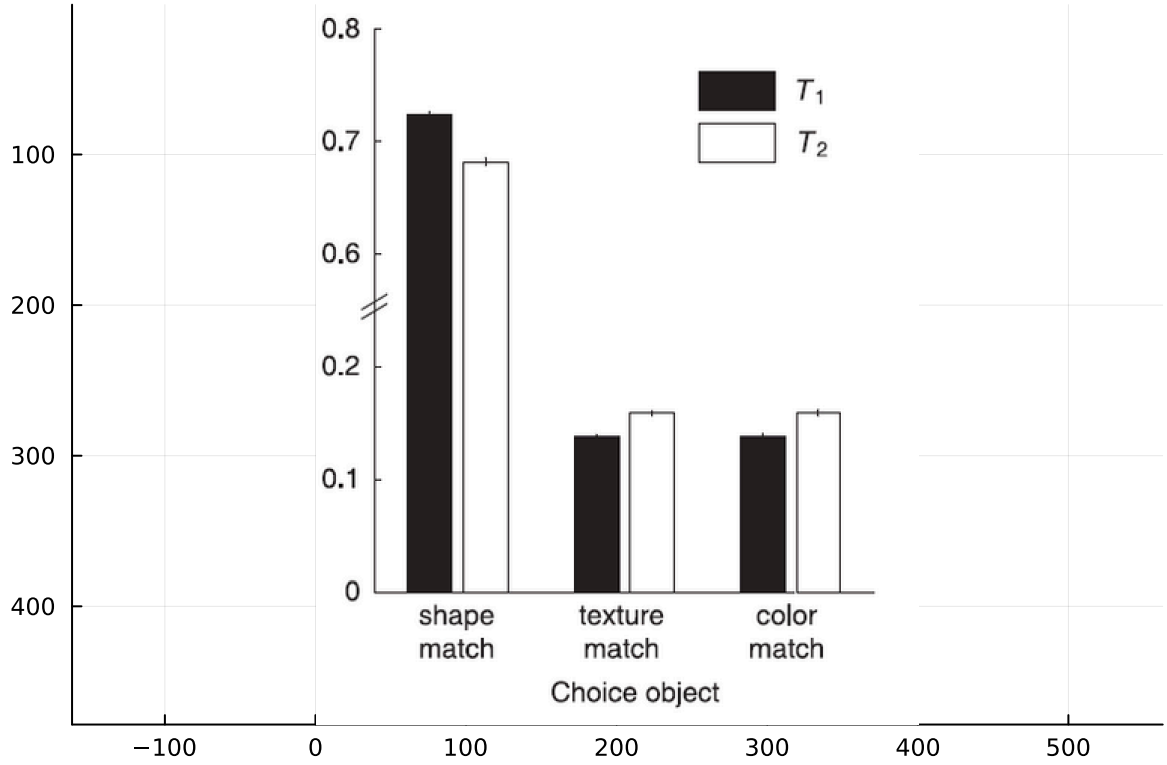
```
predictives_attr = #65 (generic function with 1 method)
```

```
predictives_attr = @joint cat_5_shape cat_5_color cat_N_shape cat_N_color
```

```
predictives_attr_samples =
  [(cat_5_shape = 5, cat_5_color = 5, cat_N_shape = 5, cat_N_color = 5), (cat_5_shape = 5, cat_5_c
```

```
predictives_attr_samples = randsample(predictives_attr |ᶜ obs_fn_attr, 25, alg=MH)
```

The program above gives us draws from some novel category for which we've seen a single instance. In the experiments with children, they had to choose one of three choice objects which varied according to the dimension they matched the example object from the category. We show below model predictions (from Kemp et al (2007)) for performance on the shape bias task which show the probabilities (normalized) that the choice object belongs to the same category as the test exemplar. The model predictions reproduce the general pattern of the experimental results of Smith et al in that shape matches are preferred in both the first and second order generalization case, and more strong in the first order generalization case. The model also helps to explain the childrens' vocabulary growth in that it shows how the shape bias can be generally learned, as seen by the differing values learned for the various alpha parameters, and so used outside the lab.



The model can be extended to learn to apply the shape bias only to the relevant ontological kinds, for example to object categories but not to substance categories. The Kemp et al (2007) paper discusses such an extension to the model which learns the hyperparameters separately for each kind and further learns what categories belong to each kind and how many kinds there are. This involves the use of a non-parametric prior, called the Chinese Restaurant Process, which will be discussed in the section on non-parametric models.

Example: Beliefs about Homogeneity and Generalization

In a 1983 paper, Nisbett and colleagues (Nisbett et al., 1983) examined how, and under what conditions, people made use of statistical heuristics when reasoning. One question they considered was how and when people generalized from a few instances. They showed that to what extent people generalise depends on beliefs about the homogeneity of the group that the object falls in with respect to the property they are being asked to generalize about. In one study, they asked subjects the following question:

```
Imagine that you are an explorer who has landed on a little known island in the Sou
theastern Pacific. You encounter several new animals, people, and objects. You observ
e the properties of your “samples” and you need to make guesses about how common thes
e properties would be in other animals, people, or objects of the same type.
```

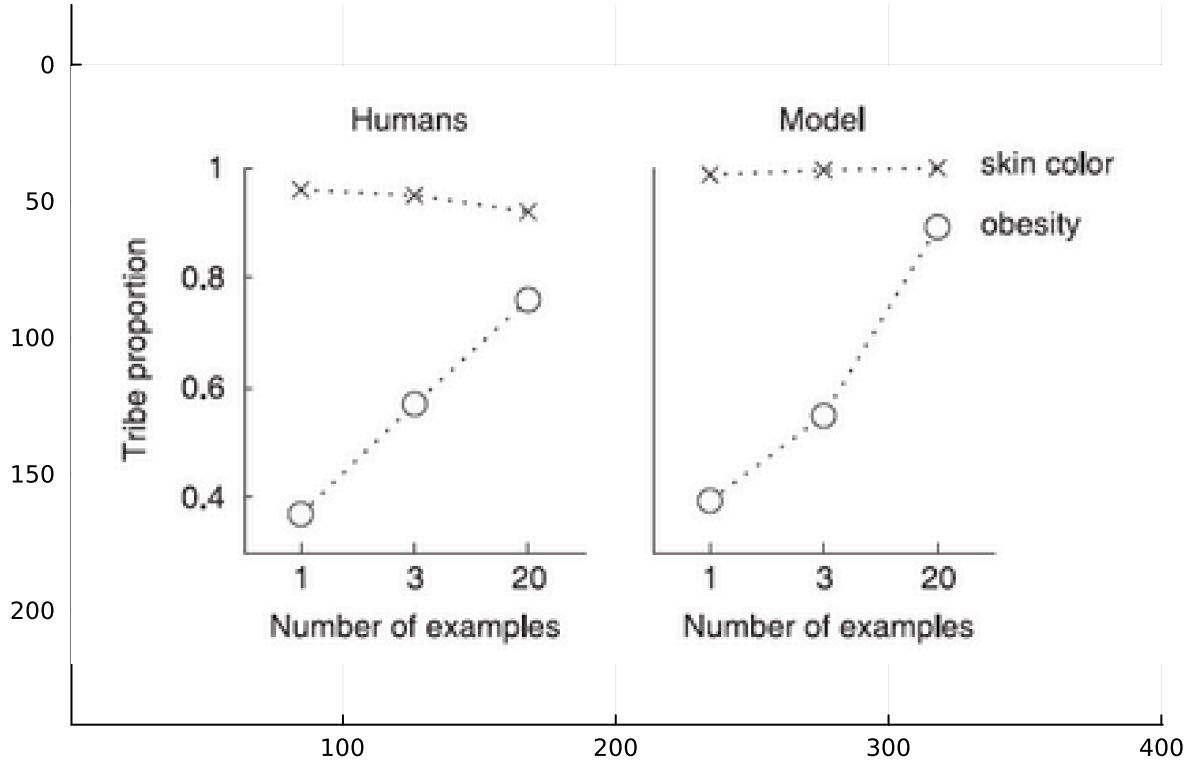
The number of encountered instances of an object were varied (one, three, or twenty instances) as well as the type and property of the objects. For example:

```
Suppose you encounter a native, who is a member of a tribe he calls the Barratos. H
e is obese. What percent of the male Barratos do you expect to be obese?
```

and

```
Suppose the Barratos man is brown in color. What percent of male Barratos do you ex
pect to be brown (as opposed to red, yellow, black or white)?
```

Results for two questions of the experiment are shown below. The results accord both with the beliefs of the experimenters about how heterogeneous different groups would be, and subjects stated reasons for generalizing in the way they did for the different instances (which were coded for beliefs about how homogeneous objects are with respect to some property).

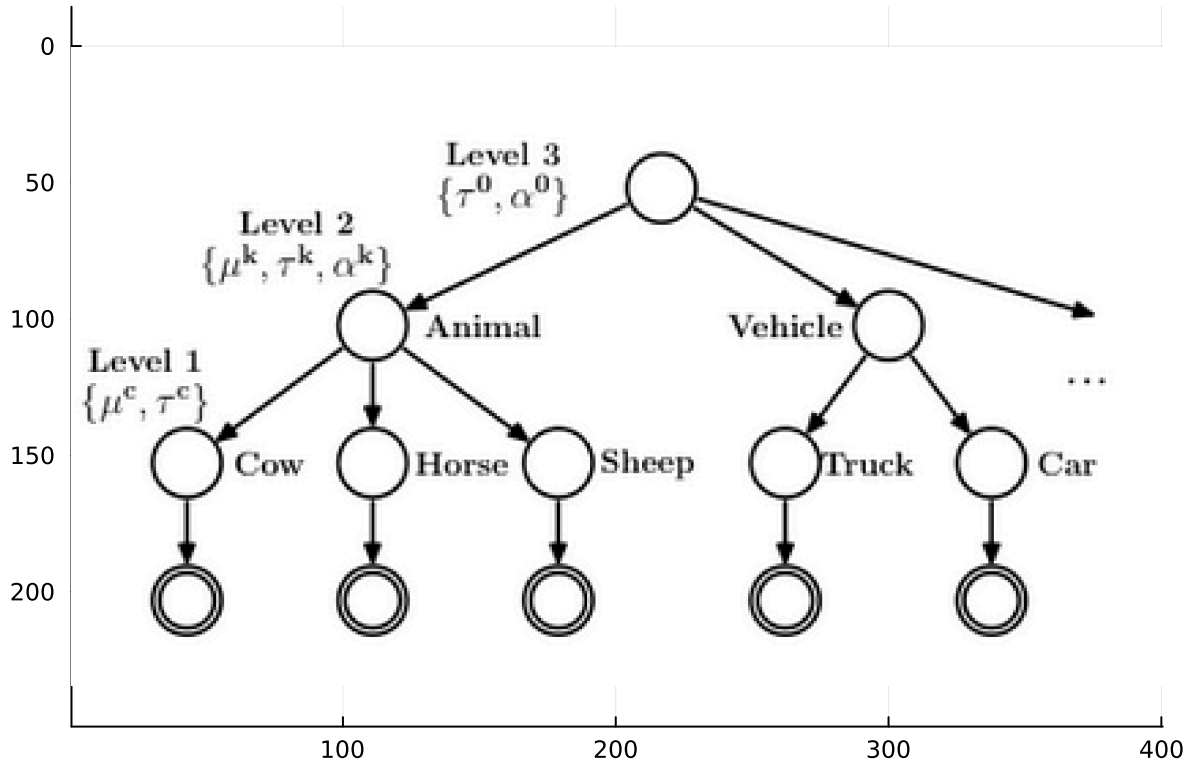


Again, we can use the compound Dirichlet-multinomial model we have been working with throughout to model this task, following Kemp et al (2007). In the context of the question about members of the Barratos tribe, replace bags of marbles with tribes and the color of marbles with skin color, or the property of being obese. Observing data such that skin color is consistent within tribes but varies between tribes will cause a low value of the alpha corresponding to skin color to be learned, and so seeing a single example from some new tribe will result in a sharply peaked predictive posterior distribution for the new tribe. Conversely, given data that obesity varies within a tribe the model will learn a higher value of the alpha corresponding to obesity and so will not generalize nearly as much from a single instance from a new tribe. Note that again it's essential to have learning at the level of hyperparameters in order to capture this phenomenon. It is only by being able to learn appropriate values of the hyperparameters from observing a number of previous tribes that the model behaves reasonably when given a single observation from a new tribe.

Example: One-shot learning of visual categories

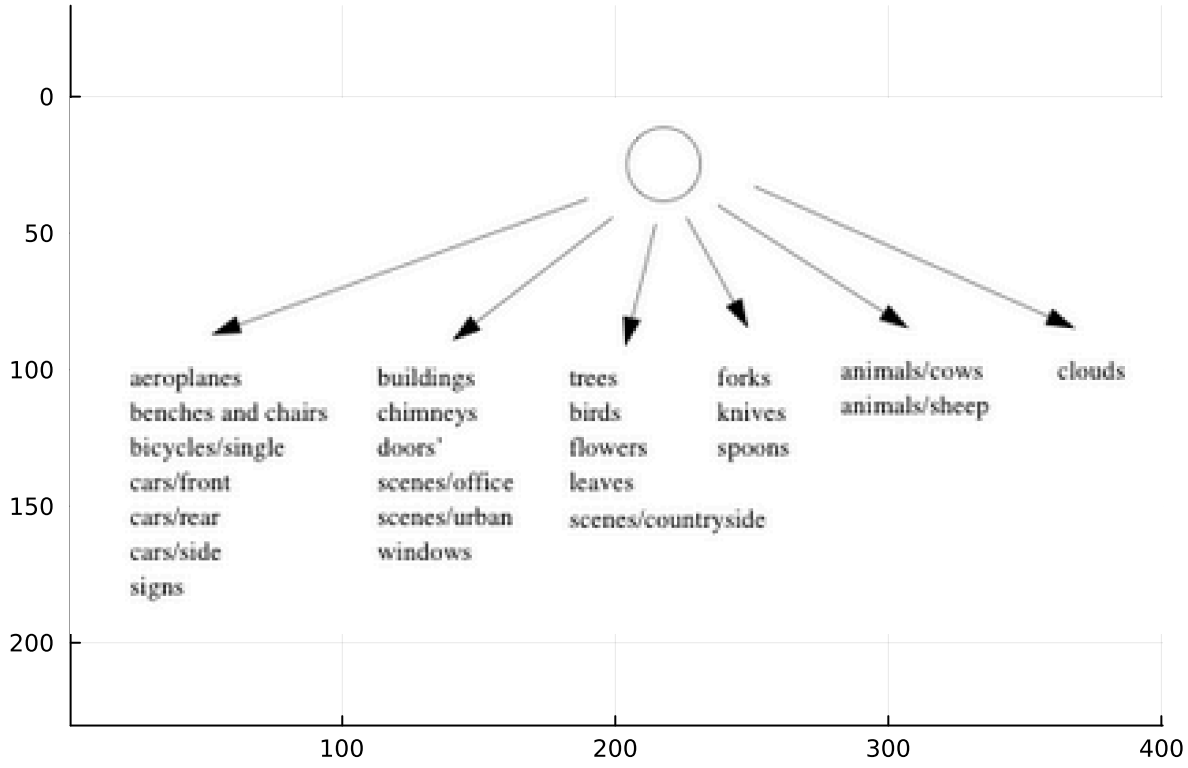
Humans are able to categorize objects (in a space with a huge number of dimensions) after seeing just one example of a new category. For example, after seeing a single wildebeest people are able to identify other wildebeest, perhaps by drawing on their knowledge of other animals. The model in

Salakhutdinov et al (Salakhutdinov et al., 2010) uses abstract knowledge learned from other categories as a prior on the mean and covariance matrix of new categories.



Suppose, first that the model is given an assignment of objects to basic categories and basic categories to superordinate categories. Objects are represented as draws from a multivariate Gaussian and the mean and covariance of each basic category is determined by hyperparameters attached to the corresponding superordinate category. The parameters of the superordinate categories are all drawn from a common set of hyperparameters.

The model in the Salakhutdinov et al (2010) paper is not actually given the assignment of objects to categories and basic categories to superordinate categories, but rather learns this from the data by putting a non-parametric prior over the tree of object and category assignments.



Example: X-Bar Theory

(This example comes from an unpublished manuscript by O'Donnell, Goodman, and Katzir)

One of the central problems in generative linguistics has been to account for the ease and rapidity with which children are able to acquire their language from noisy, incomplete, and sparse data. One suggestion for how this can happen is that the space of possible natural languages varies *parametrically*. The idea is that there are a number of higher-order constraints on structure that massively reduce the complexity of the learning problem. Each constraint is the result of a parameter taking on one of a small set of values. (This is known as "principles and parameters" theory.) The child needs only see enough data to set these parameters and the details of construction-specific structure will then generalize across the rest of the constructions of their language.

One example is the theory of headedness and X-bar phrase structure (Chomsky, 1970). X-bar theory provides a hierarchical model for phrase structure. All phrases follow the same basic *template*:

$$XP \longrightarrow Spec\ X\prime\ X\prime \longrightarrow X\ Comp$$

Where X is a lexical (or functional) category such as N (noun), V (verb), etc. X-bar theory proposes that all phrase types have the same basic "internal geometry"; They have a *head* – a word of category X. They also have a specifier ($Spec$) and a complement ($Comp$), the complement is more closely associated with the head than the specifier. The set of categories that can appear as complements and specifiers for a particular category of head is usually thought to be specified by universal grammar (but may also vary parametrically).

An important way in which languages vary is the order in which heads appear with respect to their complements (and specifiers). Within a language there tends to be a dominant order, often with exceptions for some category types. For instance, English is primarily a head-initial language. In verb

phrases, for example, the direct object (complement noun phrase) of a verb appears to the right of the head. However, there are exceptional cases such as the order of (simple) adjective and nouns: adjectives appear before the noun rather than after it (although more complex complement types such as relative clauses appear after the noun).

The fact that languages show consistency in head directionality could be of great advantage to the learner; after encountering a relatively small number of phrase types and instances the learner of a consistent language can learn the dominant head direction in their language, transferring this knowledge to new phrase types. The fact that within many languages there are exceptions suggests that this generalization cannot be deterministic, however, and, furthermore means that a learning approach will have to be robust to within-language variability. Here is a highly simplified model of X-Bar structure:

```
categories =  ["D", "N", "T", "V", "A", "Adv"]
```

```
# the "grammar": a set of phrase categories, and an associating of the complement to each head category:
categories = ["D", "N", "T", "V", "A", "Adv"]
```

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

```
head_to_comp(head) = head == "D" ? "N" :
          head == "T" ? "V" :
          head == "N" ? "A" :
          head == "V" ? "Adv" :
          head == "A" ? nothing :
          head == "Adv" ? nothing :
          error()
```

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

```
function make_phrase_dist(i, ω, head_to_phrase)
    head = (i~ UniformDraw(categories))(ω)
    if isnothing(head_to_comp(head))
        return [head]
    else
        if (i~ Bernoulli(head_to_phrase[head]))(ω)
            return [head_to_comp(head), head]
        else
            [head, head_to_comp(head)]
        end
    end
end
```

```
data_lang =  ["D", "N"]
```

```
data_lang = ["D", "N"]
```

```
language_dir = 5165389365478928634@Distributions.Beta{Float64}(α=1.0, β=1.0)
```

```
language_dir = @~ Beta(1, 1)
```

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

```
function head_to_phrase(ω)
    h(i, ω) = (i ~ Dirichlet([language_dir(ω), 1 - language_dir(ω)]))(ω)
    Dict(zip(categories, map(i -> h(i, ω)[2], 1:length(categories))))
end
```

```
# uses factor in the code below (compares vector of strings with their 'score'),
which is not present in Omega
```

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

```
lang(ω) = (@~ Bernoulli(head_to_phrase(ω)["N"]))(ω) ? "N second" : "N first"
```

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

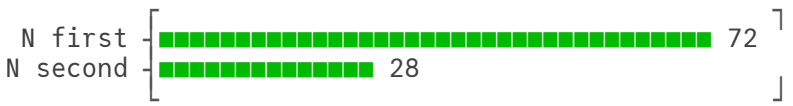
```
cond(ω) = make_phrase_dist(@uid, ω, head_to_phrase(ω)) == data_lang
```

```
posterior_lang =
  Conditional(lang (generic function with 1 method), cond (generic function with 1 method))
```

```
posterior_lang = lang |ᶜ cond
```

```
samples_lang =
  ["N second", "N first", "N first", "N first", "N first", "N first", "N second", "N first", '
```

```
samples_lang = randsample(posterior_lang, 100)
```




```
viz(samples_lang)
```

First, try increasing the number of copies of `['D', 'N']` observed. What happens? Now, try changing the data to `[['D', 'N'], ['T', 'V'], ['V', 'Adv']]`. What happens if you condition on additional instance of `['V', 'Adv']`? How about `['Adv', 'V']`?

What we see in this example is a simple probabilistic model capturing a version of the "principles and parameters" theory. Because it is probabilistic, systematic inferences will be drawn despite exceptional sentences or even phrase types. More importantly, due to the blessing of abstraction, the overall headedness of the language can be inferred from very little data—before the learner is very confident in the headedness of individual phrase types.

Thoughts on Hierarchical Models

We have just seen several examples of *hierarchical Bayesian models*: generative models in which there are several levels of latent random choices that affect the observed data. In particular a hierarchical model is usually one in which there is a branching structure in the dependence diagram, such that the "deepest" choices affect all the data, but they only do so through a set of more shallow choices which each affect some of the data, and so on.

Hierarchical model structures give rise to a number of important learning phenomena: transfer learning (or learning-to-learn), the blessing of abstraction, and learning curves with fairly abrupt

transitions. This makes them important for understanding human learning, as well as useful for creating artificial intelligence that makes the best use of available data.