```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`
```

Prelude: Thinking About Assembly Lines

Imagine a factory where the widget-maker is used to make widgets, but they are sometimes faulty. The tester tests them and lets through only the good ones. You don't know what tolerance the widget tester is set to, and wish to infer it. We can represent this as:

```
widget_machine_choice = Distributions.Categorical{Float64, Vector{Float64}}(
                        support: Base.OneTo(7)
                        p: [0.05, 0.1, 0.2, 0.3, 0.2, 0.1, 0.05]
                        )
```

```
widget_machine_choice = Categorical([.05, .1, .2, .3, .2, .1, .05])
```

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

```
widget_machine(i, ω) = [.2 , .3, .4, .5, .6, .7, .8 ][widget_machine_choice(i, ω)]
```

```
actual_weights = [0.6, 0.7, 0.8]
```

```
actual_weights = [.6, .7, .8]
```

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

```
function get_good_widget(i, ω)
    widget = (@uid, i) ~ widget_machine
    widget(ω) > tolerance(ω) ? widget(ω) : get_good_widget(i + 1, ω)
end
```

```
tolerance = -7006832023973978854@Distributions.Uniform{Float64}(a=0.3, b=0.7)
```

```
tolerance = @~ Uniform(0.3, 0.7)
```

```
actual_widgets = [0.6, 0.7, 0.8]
```

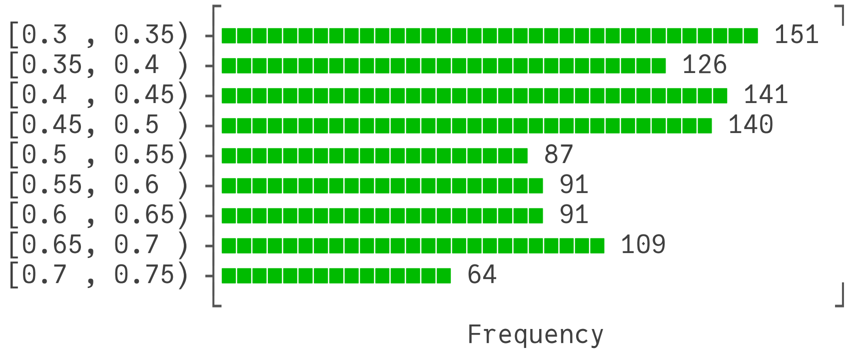
```
actual_widgets = [0.6, 0.7, 0.8]
```

```
random_widgets = Mv(1:3, get_good_widget (generic function with 1 method))
```

```
random_widgets = manynth(get_good_widget, 1:length(actual_widgets))
```

```
tolerance_ =
 [0.544882, 0.54326, 0.538705, 0.541332, 0.47757, 0.510969, 0.573191, 0.564022, 0.525892, 0.4
```

```
tolerance_ = randsample(tolerance |ᶜ pw(==, random_widgets, actual_widgets), 1000,
alg=MH)
```




```
1 viz(tolerance_)
```

But notice that the definition of `getGoodWidget` is exactly like the definition of rejection sampling! We can re-write this much more simply

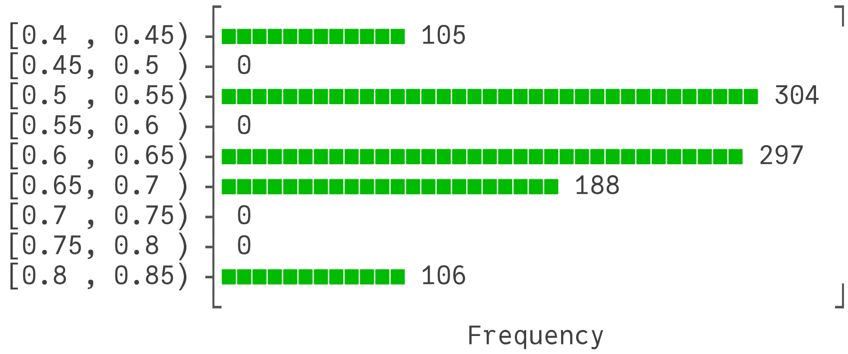
```
widget = -7846688463851221737@widget_machine
```

```
1 widget = @~ widget_machine
```

```
get_good_widget_simple =
  Conditional(-7846688463851221737@widget_machine, >ₚ(-7846688463851221737@widget_machine, -
```

```
1 get_good_widget_simple = widget |ᶜ (widget .> tolerance)
```

`randsample` uses rejection sampling by default, but we could also explicitly specify it by using `alg` keyword as given below:




```
1 viz(randsample(get_good_widget_simple, 1000, alg = RejectionSample))
```

We are now abstracting the tester machine, rather than thinking about the details inside the widget tester. We represent only that the machine correctly gives a widget above tolerance (by some means).

Social Cognition

How can we capture our intuitive theory of other people? Central to our understanding is the principle of rationality: an agent tends to choose actions that she expects to lead to outcomes that satisfy her goals. (This is a slight restatement of the principle as discussed in Baker et al. (2009), building on earlier work by Dennett (1989), among others.) We can represent this in Omega as an inference over actions—an agent reasons about actions that lead to their goal being satisfied.

For instance, imagine that Sally walks up to a vending machine wishing to have a cookie. Imagine also that we know the mapping between buttons (potential actions) and foods (outcomes). We can then predict Sally's action:

```
action_prior =
Distributions.Categorical{Float64, Vector{Float64}}(support=Base.OneTo(2), p=[0.5, 0.5])
```

```julia
action_prior = Categorical([0.5, 0.5])
```

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

```julia
function vending_machine(action)
    if action == 1
        return :bagel
    elseif action == 2
        return :cookie
    end
end
```

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

```julia
function choose_action(goal_state, transition)
    a = @~ action_prior
    a |ᶜ (Variable(transition ∘ a) .== goal_state)
end
```

```
sally_cookie_samples =
 ["2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", "2"
```

```julia
sally_cookie_samples =
    string.(randsample(choose_action(:cookie, vending_machine), 1000))
```

```
     ┌                                        ┐
   2 ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 1 000
     └                                        ┘
```

```julia
viz(sally_cookie_samples)
```

We see, unsurprisingly, that if Sally wants a cookie, she will always press button 2. In a world that is not quite so deterministic Sally's actions will be more stochastic:

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

```julia
function vending_machine_stochastic(ω, action)
    choices = [:bagel, :cookie]
    if action == 1
        choices[(@~ Categorical([0.9, 0.1]))(ω)]
    elseif action == 2
        choices[(@~ Categorical([0.1, 0.9]))(ω)]
    end
end
```

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

```julia
function choose_action_stochastic(goal, transition, i)
    a = i ~ action_prior
    a |ᶜ (Variable(ω -> transition(ω, a(ω))) .== goal)
end
```

```
action_samples =
 [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2,    more ,2, 2, 1, 2, 2, 2, 2, 2, 2, 2
```

```julia
action_samples =
    randsample(choose_action_stochastic(:cookie, vending_machine_stochastic, 0), 1000)
```

```
     ┌                                        ┐
   1 ┤■■■■ 103
   2 ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 897
     └                                        ┘
```

```julia
viz(string.(action_samples))
```

Inferring Goals

Now imagine that we don't know Sally's goal (which food she wants), but we observe her pressing button 2. We can infer her goal (this is sometimes called "inverse planning") as follows:

```
goal =
ᵛOmegaExamples.var"#7#8"{Vector{Symbol}}([:bagel, :cookie]) ∘ -5058250424038994267@Distribu
```

```julia
goal = Variable(pget([:bagel, :cookie]) ∘ @~ Categorical([.5, .5]))
```

```
action_dist = ᵛ#7
```

```julia
action_dist =
    Variable(ω -> choose_action_stochastic(goal(ω), vending_machine_stochastic, 1)(ω))
```

```
goal_posterior =
 Conditional(ᵛOmegaExamples.var"#7#8"{Vector{Symbol}}([:bagel, :cookie]) ∘ -505825042403899
```

```julia
goal_posterior = goal |ᶜ (action_dist .== 2)
```

```
goal_post_samples =
 [:cookie, :bagel, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :c
```

```julia
goal_post_samples = randsample(goal_posterior, 1000)
```

```
  bagel -[■■■■ 104
 cookie -[■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 896
```

```
viz(goal_post_samples)
```

Now let's imagine a more ambiguous case: button 2 is "broken" and will (uniformly) randomly result in a food from the machine. If we see Sally press button 2, what goal is she most likely to have?

vending_machine_broken (generic function with 1 method)

```
function vending_machine_broken(ω, action)
    choices = [:bagel, :cookie]
    if action == 1
        choices[((@uid, 1) ~ Categorical([0.9, 0.1]))(ω)]
    elseif action == 2
        choices[((@uid, 2) ~ Categorical([0.5, 0.5]))(ω)]
    end
end
```

action_dist_broken = ᵛ#9

```
action_dist_broken =
    Variable(ω ->
            choose_action_stochastic(goal, vending_machine_broken, 1)(ω))
```

goal_posterior_broken =
 Conditional(ᵛOmegaExamples.var"#7#8"{Vector{Symbol}}([:bagel, :cookie]) ∘ -505825042403899

```
goal_posterior_broken = goal |ᶜ (action_dist_broken .== 2)
```

goal_post_broken_samples =
 [:bagel, :bagel, :cookie, :cookie, :cookie, :cookie, :bagel, :bagel, :bagel, :cookie, :bagel

```
goal_post_broken_samples = randsample(goal_posterior_broken, 100)
```

```
  bagel -[■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 50
 cookie -[■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 50
```

```
viz(goal_post_broken_samples)
```

Despite the fact that button 2 is equally likely to result in either bagel or cookie, we have inferred that Sally probably wants a cookie. This is a result of the inference implicitly taking into account the counterfactual alternatives: if Sally had wanted a bagel, she would have likely pressed button 1. The inner query takes these alternatives into account, adjusting the probability of the observed action based on alternative goals.

Inferring preferences

If we have some prior knowledge about Sally's preferences (which goals she is likely to have) we can incorporate this immediately into the prior over goals (which above was uniform).

A more interesting situation is when we believe that Sally has some preferences, but we don't know what they are. We capture this by adding a higher level prior (a uniform) over preferences. Using this we can learn about Sally's preferences from her actions: after seeing Sally press button 2 several times, what will we expect her to want the next time?

```
preference = 9384230667355559508@Distributions.Uniform{Float64}(a=0.0, b=1.0)
```

```julia
preference = @~ Uniform(0 , 1)
```

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

```julia
goal_prior(i, ω) = ifelse.(((@uid, i) ~ Bernoulli(preference)), :bagel, :cookie)(ω)
```

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

```julia
action_dist_has_preference(i, ω) =
    choose_action_stochastic(goal_prior(i, ω), vending_machine_stochastic,
    :has_preference)(ω)
```

```
random_actions =  Mv(1:3, action_dist_has_preference (generic function with 1 method))
```

```julia
random_actions = manynth(action_dist_has_preference, 1:3)
```

```
goal_posterior_samples =
 [:cookie, :cookie, :cookie, :cookie, :bagel, :cookie, :cookie, :bagel, :cookie, :cookie, :co
```

```julia
goal_posterior_samples =
        randsample((@~ goal_prior) |ᶜ pw(==, random_actions, [2, 2, 2]), 100)
```

```
         ┌                                        ┐
   bagel ┤■■■■■■■■■■■■■■■■ 29
  cookie ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 71
         └                                        ┘
```

```julia
viz(goal_posterior_samples)
```

Try varying the amount and kind of evidence. For instance, if Sally one time says "I want a cookie" (so you have directly observed her goal that time) how much evidence does that give you about her preferences, relative to observing her actions?

In the above preference inference, it is extremely important that sally *could have* taken a different action if she had a different preference (i.e. she could have pressed button 1 if she preferred to have a bagel). In the program below we have set up a situation in which both actions lead to cookie most of the time:

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

```julia
function vending_machine_cookie(ω, action)
    choices = [:bagel, :cookie]
    if action == 1
        choices[(@~ Categorical([0.1, 0.9]))(ω)]
    elseif action == 2
        choices[(@~ Categorical([0.1, 0.9]))(ω)]
    end
end
```

```
preference_cookie = Distributions.Uniform{Float64}(a=0.0, b=1.0)
```

```julia
preference_cookie = Uniform(0, 1)
```

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

```
goal_prior_cookie(i, ω) = ifelse.((((@uid, i) ~ Bernoulli(i~preference_cookie)), :bagel, :cookie)(ω)
```

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

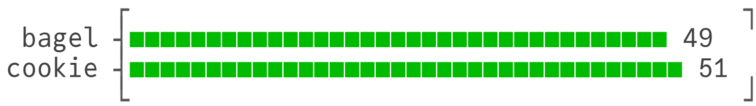
```
action_dist_cookie(i, ω) =
    choose_action_stochastic(goal_prior_cookie(i, ω), vending_machine_cookie, :cookie)(ω)
```

```
random_actions_cookie =  Mv(1:3, action_dist_cookie (generic function with 1 method))
```

```
random_actions_cookie = manynth(action_dist_cookie, 1:3)
```

```
goal_posterior_cookie_samples =
 [:cookie, :cookie, :bagel, :cookie, :bagel, :bagel, :cookie, :bagel, :bagel, :bagel, :cookie
```

```
goal_posterior_cookie_samples =
        randsample((@~goal_prior_cookie) |ᶜ pw(==, random_actions_cookie, [2, 2, 2]),
        100)
```




```
viz(goal_posterior_cookie_samples)
```

Now we can draw no conclusion about Sally's preferences. Try varying the machine probabilities, how does the preference inference change? This effect, that the strength of a preference inference depends on the context of alternative actions, has been demonstrated in young infants by Kushnir et al. (2010).

Inferring what they know

In the above models of goal and preference inference, we have assumed that the structure of the world (the operation of the vending machine) was common knowledge—they were non-random constructs used by both the agent (Sally) selecting actions and the observer interpreting these actions. What if we (the observer) don't know how exactly the vending machine works, but think that Sally knows how it works? We can capture this by placing uncertainty on the vending machine inside the overall query but "outside" of Sally's inference:

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

```
function vending_machine_know(ω, action)
    choices = [:bagel, :cookie]
    if action in [1, 2]
        c = ((:know, action) ~ Uniform(0, 1))(ω)
        choices[(:know ~ Categorical([c, 1 - c]))(ω)]
    end
end
```

```
goal_know =
ᵛOmegaExamples.var"#7#8"{Vector{Symbol}}([:bagel, :cookie]) ∘ -684314054380975331@Distribut
```

```
goal_know = Variable(pget([:bagel, :cookie]) ∘ @~ Categorical([.5, .5]))
```

```
action_dist_know = ᵛ#11
```

```julia
action_dist_know = Variable(ω -> choose_action_stochastic(goal_know(ω),
vending_machine_know, :know)(ω))
```

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

```julia
buttons(ω::Ω) =
    (button_1 = vending_machine_know(ω, 1), button_2 = vending_machine_know(ω, 2))
```

```
buttons_posterior =
  Conditional(buttons (generic function with 1 method), &ₚ(==ₚ(ᵛ#11, 2), ==ₚ(ᵛOmegaExamples.ᵛ
```

```julia
buttons_posterior = buttons |ᶜ .&((action_dist_know .== 2), (goal .== :cookie))
```

```
buttons_joint_samples =
  [(button_1 = :bagel, button_2 = :bagel), (button_1 = :bagel, button_2 = :cookie), (button_1 =
```

```julia
buttons_joint_samples = randsample(buttons_posterior, 100)
```

```
                ┌                                          ┐
  button_1_bagel ┤■■■■■■■■■■■ 24
 button_1_cookie ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 76
  button_2_bagel ┤■■■■■■■■■■■■■ 28
 button_2_cookie ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 72
                └                                          ┘
```

```julia
viz_marginals(buttons_joint_samples)
```

Now imagine a vending machine that has only one button, but it can be pressed many times. We don't know what the machine will do in response to a given button sequence. We do know that pressing more buttons is less a priori likely.

```
action_prior_one_button =
-3199665172235308988@Distributions.Categorical{Float64, Vector{Float64}}(support=Base.OneTo
```

```julia
action_prior_one_button = @~ Categorical([0.7, 0.2, 0.1])
```

```
goal_one_button =
ᵛOmegaExamples.var"#7#8"{Vector{Symbol}}([:bagel, :cookie]) ∘ 7268292349891251283@Distribut
```

```julia
goal_one_button = Variable(pget([:bagel, :cookie]) ∘ @~ Categorical([.5, .5]))
```

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

```julia
function vending_machine_one_button(ω, action)
    choices = [:bagel, :cookie]
    if action in [1, 2, 3]
        c =  ((@uid, action) ~ Uniform(0, 1))(ω)
        choices[(@~ Categorical([c, 1 - c]))(ω)]
    end
end
```

```
action_dist_one_button = ᵛ#13
```

```julia
action_dist_one_button = Variable(ω -> choose_action_stochastic(goal_one_button(ω),
vending_machine_one_button, :one_button)(ω))
```

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

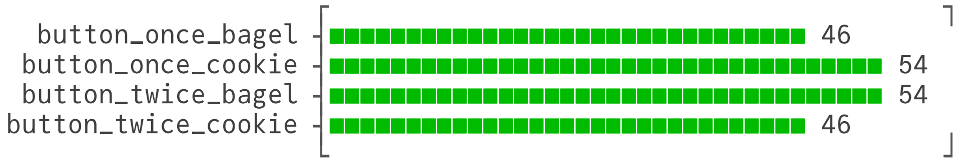
```julia
buttons_(ω::Ω) =
    (button_once = vending_machine_one_button(ω, 1), button_twice =
    vending_machine_one_button(ω, 2))
```

```
buttons_posterior_ =
  Conditional(buttons_ (generic function with 1 method), &ₚ(==ₚ(ᵛ#13, 1), ==ₚ(ᵛOmegaExamples
```

```
buttons_posterior_ =
    buttons_ |ᶜ .&((action_dist_one_button .== 1), (goal .== :cookie))
```

```
buttons_joint_samples_ =
  [(button_once = :cookie, button_twice = :bagel), (button_once = :cookie, button_twice = :cooki
```

```
buttons_joint_samples_ = randsample(buttons_posterior_, 100)
```




```
viz_marginals(buttons_joint_samples_)
```

Joint inference about knowledge and goals

In social cognition, we often make joint inferences about two kinds of mental states: agents' beliefs about the world and their desires, goals or preferences. We can see an example of such a joint inference in the vending machine scenario. Suppose we condition on two observations: that Sally presses the button twice, and that this results in a cookie. Then, assuming that she knows how the machine works, we jointly infer that she wanted a cookie, that pressing the button twice is likely to give a cookie, and that pressing the button once is unlikely to give a cookie.

```
goal_kg =
ᵛOmegaExamples.var"#7#8"{Vector{Symbol}}([:bagel, :cookie]) ∘ -5708447257567624O8@Distribut
```

```
goal_kg = Variable(pget([:bagel, :cookie]) ∘ @~ Categorical([.5, .5]))
```

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

```
function vending_machine_kg(ω, action)
    choices = [:bagel, :cookie]
    if action in [1, 2, 3]
        c = ((:kg, action) ~ Uniform(0, 1))(ω)
        choices[(@~ Categorical([c, 1 - c]))(ω)]
    end
end
```

```
action_dist_kg = ᵛ#15
```

```
action_dist_kg =
    Variable(ω -> choose_action_stochastic(goal_kg(ω), vending_machine_kg, :kg)(ω))
```

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

```
knowledge_and_goals(ω::Ω) = (goal = goal_kg(ω),
          one_press_result = vending_machine_kg(ω, 1),
          two_press_result = vending_machine_kg(ω, 2),
          one_press_cookie_prob = 1 - ((:kg, 1) ~ Uniform(0, 1))(ω))
```

kg_posterior =
```
Conditional(knowledge_and_goals (generic function with 1 method), &ₚ(==ₚ(ᵛ#17, Base.RefVal
```

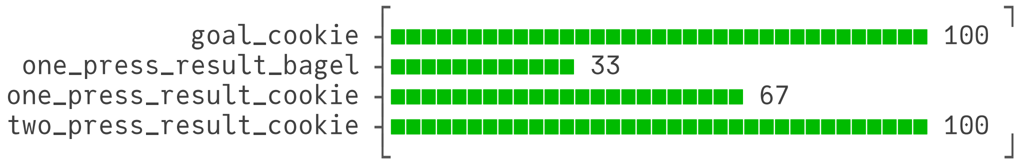
```
kg_posterior =
    knowledge_and_goals |ᶜ pw(&, (Variable(ω -> vending_machine_kg(ω, 2)) .==
    :cookie), (action_dist_kg .== 2))
```

kg_samples =
```
[(goal = :cookie, one_press_result = :cookie, two_press_result = :cookie, one_press_cookie_prol
```

```
kg_samples = randsample(kg_posterior, 100)
```

kg_ =
```
[(goal = :cookie, one_press_result = :cookie, two_press_result = :cookie), (goal = :cookie, on
```

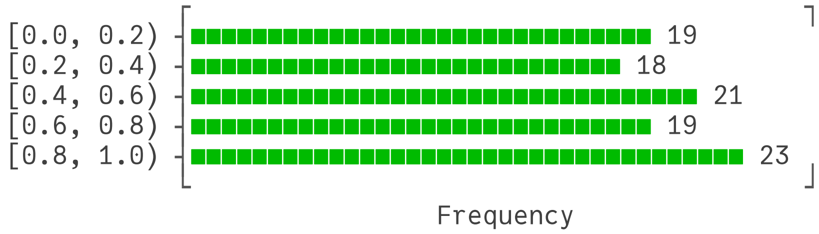
```
kg_ = map(b -> Base.structdiff(b, (one_press_cookie_prob = b.one_press_cookie_prob,
)), kg_samples)
```




```
viz_marginals(kg_)
```

kg_one_press_cookie_prob =
```
[0.745929, 0.864057, 0.489846, 0.927995, 0.294315, 0.948009, 0.851168, 0.350462, 0.3814, 0.0
```

```
kg_one_press_cookie_prob = map(b -> b.one_press_cookie_prob, kg_samples)
```




```
viz(kg_one_press_cookie_prob)
```

Inferring whether they know

Let's imagine that we (the observer) know that the vending machine actually tends to return a bagel for button 1 and a cookie for button 2. But we don't know if Sally knows this! Instead we see Sally announce that she wants a cookie, but pushes button 1. How can we determine, from her actions, whether Sally is knowledgeable or ignorant? We hypothesize that if she is ignorant, Sally chooses according to a random vending machine. We can then infer her knowledge state:

b_probs = `[0.9, 0.1]`

```
b_probs = [0.9, 0.1]
```

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

```julia
function true_vending_machine(ω, action)
    choices = [:bagel, :cookie]
    c = b_probs[action]
    choices[(@~ Categorical([c, 1 - c]))(ω)]
end
```

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

```julia
function random_machine(ω, action)
    choices = [:bagel, :cookie]
    choices[(@~ Categorical([0.5, 0.5]))(ω)]
end
```

```
knows = 8678448019359285034@Distributions.Bernoulli{Float64}(p=0.5)
```

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

```
s =
 [false, false, false, false, false, false, false, false, false, false, false, false, false, f
```

```julia
s = randsample(knows |ᶜ (ω -> (choose_action_stochastic(:cookie, knows(ω) ?
true_vending_machine : random_machine, :know)(ω) == 1) & (true_vending_machine(ω, 1)
== :bagel)), 100)
```

```
false ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 100
```

```julia
viz(s)
```

This is a very simple example, but it illustrates how we can represent a difference in knowledge between the observer and the observed agent by simply using different world models (the vending machines) for explaining the action (in `choose_action_stochastic`) and for explaining the outcome (in `|ᶜ`).

Inferring what they believe

Above we assumed that if Sally is ignorant, she chooses based on a random machine. This is both not flexible enough and too strong an assumption. Indeed, Sally may have all kinds of specific (and potentially false) beliefs about vending machines. To capture this, we can represent Sally's beliefs as a separate randomly chosen vending machine: by passing this into Sally's `choose_action_stochastic` we indicate these are Sally's beliefs, by putting this inside the outer Infer we represent the observer reasoning about Sally's beliefs:

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

```julia
sally_belief(i, ω) = ((@uid, i) ~ Uniform(0, 1))(ω)
```

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

```julia
function sally_machine(ω, action)
    choices = [:bagel, :cookie]
    c = (action ~ sally_belief)(ω)
    choices[(@~ Categorical([c, 1 - c]))(ω)]
end
```

```
s_ =
  [:cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :cookie, :
```

```
s_ = randsample((ω -> sally_machine(ω, 1)) |ᶜ (ω ->
(choose_action_stochastic(:cookie, sally_machine, :believe)(ω) == 1) &
(true_vending_machine(ω, 1) == :bagel)), 100)
```

```
cookie ┤■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 100
```

```
viz(s_)
```

In the developmental psychology literature, the ability to represent and reason about other people's *false beliefs* has been extensively investigated as a hallmark of human Theory of Mind.

Emotion and other mental states

So far we have explored reasoning about others' goals, preferences, knowledge, and beliefs. It is commonplace to discuss other's actions in terms of many other mental states as well! We might explain an unexpected slip in terms of wandering attention, a short-sighted choice in terms of temptation, a violent reaction in terms of anger, a purposeless embellishment in terms of joy. Each of these has a potential role to play in an elaborated scientific theory of how humans represent other's minds.

Communication and Language

A Communication Game

Imagine playing the following two-player game. On each round the "teacher" pulls a die from a bag of weighted dice, and has to communicate to the "learner" which die it is (both players are familiar with the dice and their weights). However, the teacher may only communicate by giving the learner examples: showing them faces of the die.

We can formalize the inference of the teacher in choosing the examples to give by assuming that the goal of the teacher is to successfully teach the hypothesis – that is, to choose examples such that the learner will infer the intended hypothesis:

To make this concrete, assume that there are two dice, A and B, which each have three sides (red, green, blue) that have weights. Which hypothesis will the learner infer if the teacher shows the green side?

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

```julia
begin
    function die_to_probs(die::Int64)
        if die == 1
            return @~ Categorical([0., 0.2, 0.8])
        elseif die == 2
            return @~ Categorical([0.1, 0.3, 0.6])
        end
    end
end
```

```
side_prior =
OmegaExamples.var"#7#8"{Vector{Symbol}}([:red, :green, :blue]) ∘ 8710490788825742538@Distri
```

```julia
side_prior = pget([:red, :green, :blue]) ∘ @~ Categorical([1/3, 1/3, 1/3])
```

```
die_prior =
8980200737310221347@Distributions.Categorical{Float64, Vector{Float64}}(support=Base.OneTo(
```

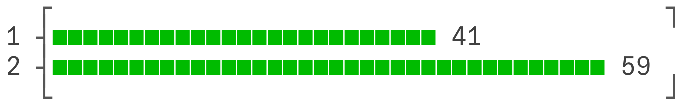
```julia
die_prior = @~ Categorical([0.5, 0.5])
```

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

```julia
roll(die, ω) = (pget([:red, :green, :blue]) ∘ die_to_probs(die))(ω)
```

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

```
begin
    function teacher(die, depth)
        return side_prior |ᶜ (Variable(ω -> learner(side_prior(ω), depth)(ω)) .== die)
    end
    function learner(side, depth)
        if (depth == 0)
            return die_prior |ᶜ (Variable(ω -> roll(die_prior(ω), ω)) .== side)
        else
            return die_prior |ᶜ (Variable(ω -> teacher(die_prior(ω), depth - 1)(ω))
            .== side)
        end
    end
end
```




```
viz(string.(randsample(learner(:green, 3), 100)))
```

If we run this with recursion depth 0—that is a learner that does probabilistic inference without thinking about the teacher thinking—we find the learner infers hypothesis 2 most of the time (about 60% of the time). This is the same as using the "strong sampling" assumption: the learner infers 2 because 2 is more likely to have landed on side 2. However, if we increase the recursion depth we find this reverses: the learner infers 2 only about 40% of the time. Now die 1 becomes the better inference, because "if the teacher had meant to communicate 2, they would have shown the red side because that can never come from 1."

This model has been proposed by Shafto et al. (2012) as a model of natural pedagogy. They describe several experimental tests of this model in the setting of simple "teaching games," showing that people make inferences as above when they think the examples come from a helpful teacher, but not otherwise.

Communicating with Words

Unlike the situation above, in which concrete examples were given from teacher to student, words in natural language denote more abstract concepts. However, we can use almost the same setup to reason about speakers and listeners communicating with words, if we assume that sentences have literal meanings, which anchor sentences to possible worlds. We assume for simplicity that the meaning of sentences is truth-functional: that each sentence corresponds to a function from states of the world to true/false.

Example: Scalar Implicature

Let us imagine a situation in which there are three plants which may or may not have sprouted. We imagine that there are three sentences that the speaker could say, "All of the plants have sprouted", "Some of the plants have sprouted", or "None of the plants have sprouted". For simplicity we represent the worlds by the number of sprouted plants (0, 1, 2, or 3) and take a uniform prior over worlds. Using the above representation for communicating with words (with an explicit depth argument):

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

```julia
all_sprouted(state) = state == 3
```

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

```julia
some_sprouted(state) = state > 0
```

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

```julia
none_sprouted(state) = state == 0
```

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

```julia
function meaning(words)
    if words == "all"
        return all_sprouted
    elseif words == "some"
        return some_sprouted
    elseif words == "none"
        return none_sprouted
    end
    @assert true "Unknown words"
end
```

```
state_prior =
-ₚ(-7712787724570764098@Distributions.Categorical{Float64, Vector{Float64}}(support=Base.On
```

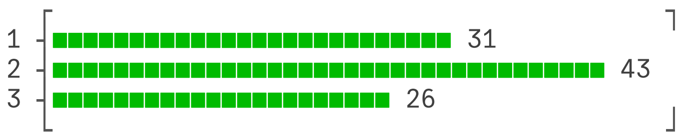
```julia
state_prior = (@~ Categorical([0.25, 0.25, 0.25, 0.25])) .- 1
```

```
sentence_prior =
OmegaExamples.var"#7#8"{Vector{String}}(["all", "some", "none"]) ∘ -5293383370984010930@Dis
```

```julia
sentence_prior = pget(["all", "some", "none"]) ∘ @~ Categorical([1/3, 1/3, 1/3])
```

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

```julia
begin
    function speaker(state, depth)
        condition = Variable(ω -> listener(sentence_prior(ω), depth)(ω)) .== state
        return sentence_prior |ᶜ condition
    end
    function listener(words, depth)
        if depth == 0
            condition = Variable(ω -> meaning(words)(state_prior(ω)))
        else
            condition = Variable(ω -> speaker(state_prior(ω), depth - 1)(ω)) .== words
        end
        state_prior |ᶜ condition
    end
end
```




```julia
viz(string.(randsample(listener("some", 1), 100)))
```