Social cognition

The `Infer` operator is an ordinary WebPPL function, in the sense that it can occur anywhere that any other function can occur. In particular, we can construct an inference containing another inference inside of it: this represents hypothetical inference *about* a hypothetical inference. Nested queries are particularly useful in modeling social cognition: reasoning about another agent, who is herself reasoning.

Prelude: Thinking About Assembly Lines

Imagine a factory where the widget-maker makes a stream of widgets, and the widget-tester removes the faulty ones. You don't know what tolerance the widget tester is set to, and wish to infer it. We can represent this as:

```
x
```

```
// this machine makes a widget -- which we'll just represent with a real number:
```

```
var widgetMachine = Categorical({vs: [.2 , .3, .4, .5, .6, .7, .8 ],
```

```
                                  ps: [.05, .1, .2, .3, .2, .1, .05]})
```

```
var thresholdPrior = Categorical({vs: [.3, .4, .5, .6, .7],
```

```
                                   ps: [.1, .2, .4, .2, .1]})
```

```
var makeWidgetSeq = function(numWidgets, threshold) {
```

```
  if(numWidgets == 0) {
```

```
    return [];
```

```
  } else {
```

```
    var widget = sample(widgetMachine);
```

```
    return (widget > threshold ?
```

```
            [widget].concat(makeWidgetSeq(numWidgets - 1, threshold)) :
            makeWidgetSeq(numWidgets, threshold));
  }
}

var widgetDist = Infer({method: 'rejection', samples: 300}, function() {
  var threshold = sample(thresholdPrior);
  var goodWidgetSeq = makeWidgetSeq(3, threshold);
  condition(_.isEqual([.6, .7, .8], goodWidgetSeq))
  return [threshold].join("");
})

viz.auto(widgetDist)
```

run ▼

But notice that the definition of next-good-widget is exactly like the definition of rejection sampling! We can re-write this as a nested-query model:

```
// this machine makes a widget -- which we'll just represent with a real number:
var widgetMachine = Categorical({vs: [.2 , .3, .4, .5, .6, .7, .8 ],
                                 ps: [.05, .1, .2, .3, .2, .1, .05]})
var thresholdPrior = Categorical({vs: [.3, .4, .5, .6, .7],
                                  ps: [.1, .2, .4, .2, .1]})
var makeGoodWidgetSeq = function(numWidgets, threshold) {
```

```
  return Infer({method: 'enumerate'}, function() {
    var widgets = repeat(numWidgets, function() {return sample(widgetMachine)});
    condition(all(function(widget) {return widget > threshold}, widgets));
    return widgets;
  })
}

var widgetDist = Infer({method: 'enumerate'}, function() {
  var threshold = sample(thresholdPrior);
  var goodWidgetSeq = makeGoodWidgetSeq(3, threshold);
  // Now that this is a *distribution*, we can just look up the likelihood
  factor(goodWidgetSeq.score([.6, .7, .8]))
  return [threshold].join("");
})

viz.auto(widgetDist)
```

run ▼

Rather than thinking about the details inside the widget tester, we are now abstracting to represent that the machine correctly chooses a good widget (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) (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.2977&rep=rep1&type=pdf), building on earlier work by

Dennett (1989) (https://scholar.google.com/scholar?q="The%20Intentional%20Stance"), among others.) We can represent this in WebPPL as an inference over actions—an agent reasons about actions that lead to their goal being satisfied:

```
var chooseAction = function(goalSatisfied, transition, state) {
  return Infer(..., function() {
    var action = sample(actionPrior)
    condition(goalSatisfied(transition(state, action)))
    return action;
  })
}
```

The function `transition` describes the outcome of taking a particular action in a particular state, the predicate `goalSatisfied` determines whether or not a state accomplishes the goal, the input `state` represents the current state of the world. The distribution `actionPrior` used within `chooseAction` represents an a-priori tendency towards certain actions.

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:

```
var actionPrior = Categorical({vs: ['a', 'b'], ps: [.5, .5]})
```

```
var haveCookie = function(obj) {return obj == 'cookie'};
```

```
var vendingMachine = function(state, action) {
```

```
  return (action == 'a' ? 'bagel' :
```

```
          action == 'b' ? 'cookie' :
```

```
          'nothing');
```

```
}
```

```

```

```
var chooseAction = function(goalSatisfied, transition, state) {
```

```
  return Infer({method: 'enumerate'}, function() {
```

```
    var action = sample(actionPrior)
```

```
    condition(goalSatisfied(transition(state, action)))
```

```
    return action;

  })

}


viz.auto(chooseAction(haveCookie, vendingMachine, 'state'));
```

run ▼

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

```
///fold:

...

var vendingMachine = function(state, action) {

  return (action == 'a' ? categorical({vs: ['bagel', 'cookie'], ps: [.9, .1]}) :

          action == 'b' ? categorical({vs: ['bagel', 'cookie'], ps: [.1, .9]}) :

          'nothing');

}


var chooseAction = function(goalSatisfied, transition, state) {

  return Infer({method: 'enumerate'}, function() {

    var action = sample(actionPrior)

    condition(goalSatisfied(transition(state, action)))

    return action;

  })
```

```
}
```

```

```

```
viz.auto(chooseAction(haveCookie, vendingMachine, 'state'));
```

run ▼

为什么不是最优呢？

Technically, this method of making a choices is not optimal, but rather it is *soft-max* optimal (also known as following the "Boltzmann policy").

Goal Inference

Now imagine that we don't know Sally's goal (which food she wants), but we observe her pressing button b. We can use a query to infer her goal (this is sometimes called "inverse planning", since the outer query "inverts" the query inside `chooseAction`).

```

```

```
///fold:
```

```
...
```

```
var chooseAction = function(goalSatisfied, transition, state) {
```

```
  return Infer({method: 'enumerate'}, function() {
```

```
    var action = sample(actionPrior)
```

```
    condition(goalSatisfied(transition(state, action)))
```

```
    return action;
```

```
  })
```

```
}
```

```

```

```
var goalPosterior = Infer({method: 'enumerate'}, function() {
```

```
  var goal = categorical({vs: ['bagel', 'cookie'], ps: [.5, .5]})
```

```
  var goalSatisfied = function(outcome) {return outcome == goal};
```

```

```

```
  var actionDist = chooseAction(goalSatisfied, vendingMachine, 'state')

  factor(actionDist.score('b'));

  return goal;

})



viz.auto(goalPosterior);
```

run ▼

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

```
///fold:

...

var vendingMachine = function(state, action) {

  return (action == 'a' ? categorical({vs: ['bagel', 'cookie'], ps: [.9, .1]}) :

          action == 'b' ? categorical({vs: ['bagel', 'cookie'], ps: [.5, .5]}) :

          'nothing');

}



var goalPosterior = Infer({method: 'enumerate'}, function() {

  var goal = categorical({vs: ['bagel', 'cookie'], ps: [.5, .5]})

  var goalSatisfied = function(outcome) {return outcome == goal};

  var actionDist = chooseAction(goalSatisfied, vendingMachine, 'state')

  factor(actionDist.score('b'));
```

```
  return goal;
})

viz.auto(goalPosterior);
```

run ▼

Despite the fact that button b 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 a. The inner query takes these alternatives into account, adjusting the probability of the observed action based on alternative goals.

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 b several times, what will we expect her to want the next time?

```
///fold:
...
var vendingMachine = function(state, action) {
  return (action == 'a' ? categorical({vs: ['bagel', 'cookie'], ps: [.9, .1]}) :
          action == 'b' ? categorical({vs: ['bagel', 'cookie'], ps: [.1, .9]}) :
          'nothing');
}

var goalPosterior = Infer({method: 'MCMC', samples: 20000}, function() {
```

```
  var preference = uniform(0, 1);

  var goalPrior = function() {return flip(preference) ? 'bagel' : 'cookie'};

  var makeGoal = function(food) {return function(outcome) {return outcome == food}};

  condition((sample(chooseAction(makeGoal(goalPrior()), vendingMachine, 'state')) == 'b')

            (sample(chooseAction(makeGoal(goalPrior()), vendingMachine, 'state')) == 'b')

            (sample(chooseAction(makeGoal(goalPrior()), vendingMachine, 'state')) == 'b'))

  return goalPrior();

})



viz.auto(goalPosterior);
```

run ▼

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 *a* 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:

```

///fold:

...

var vendingMachine = function(state, action) {

  return (action == 'a' ? categorical({vs: ['bagel', 'cookie'], ps: [.1, .9]}) :

          action == 'b' ? categorical({vs: ['bagel', 'cookie'], ps: [.1, .9]}) :

          'nothing');

}
```

```
var goalPosterior = Infer({method: 'MCMC', samples: 50000}, function() {
  var preference = uniform(0, 1);
  var goalPrior = function() {return flip(preference) ? 'bagel' : 'cookie'};
  var makeGoal = function(food) {return function(outcome) {return outcome == food}};
  condition((sample(chooseAction(makeGoal(goalPrior()), vendingMachine, 'state')) == 'b')
            (sample(chooseAction(makeGoal(goalPrior()), vendingMachine, 'state')) == 'b')
            (sample(chooseAction(makeGoal(goalPrior()), vendingMachine, 'state')) == 'b'))
  return goalPrior();
})

viz.auto(goalPosterior);
```

run ▼

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) (https://scholar.google.com/scholar?q="Young%20Children%20Use%20Statistical%20Sampling%20to%20Infer%20the%20Prefe

Epistemic States

In the above models of goal and preference inference, we have assumed that the structure of the world (both the operation of the vending machine and the, irrelevant, initial state) were 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 however it works Sally knows? We can capture this by placing uncertainty on the vending machine, inside the overall query but "outside" of Sally's inference:

```
///fold:
```

```
...

var makeVendingMachine = function(aEffects, bEffects) {

  return function(state, action) {

    return (action == 'a' ? categorical({vs: ['bagel', 'cookie'], ps: aEffects}) :

            action == 'b' ? categorical({vs: ['bagel', 'cookie'], ps: bEffects}) :

            'nothing');

  }

};



var goalPosterior = Infer({method: 'MCMC', samples: 50000}, function() {

  var aEffects = dirichlet(Vector([1,1]))

  var bEffects = dirichlet(Vector([1,1]));



  var vendingMachine = makeVendingMachine(aEffects, bEffects);



  var goal = categorical({vs: ['bagel', 'cookie'], ps: [.5, .5]})

  var goalSatisfied = function(outcome) {return outcome == goal};



  condition(goal == 'cookie' &&

            sample(chooseAction(goalSatisfied, vendingMachine, 'state')) == 'b');

  return T.get(bEffects, 1);
```

```
})
```

```

```

```
print("probability of action 'b' giving cookie")
```

```
viz.auto(goalPosterior);
```

run ▼

Here we have conditioned on Sally wanting the cookie and Sally choosing to press button b. Thus, we have no *direct* evidence of the effects of pressing the buttons on the machine. What happens if you condition instead on the action and outcome, but not the intentional choice of this outcome (that is, change the condition to `(vendingMachine('state', 'b') == 'cookie')`)?

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.

```

```

```
///fold:
```

```
...
```

```
var actionPrior = function() {
```

```
  return categorical({vs: ['a', 'aa', 'aaa'], ps:[0.7, 0.2, 0.1] })
```

```
}
```

```

```

```
var goalPosterior = Infer({method: 'rejection', samples: 5000}, function() {
```

```
  var buttonsToOutcomeProbs = {'a': T.toScalars(dirichlet(ones([2,1]))),
```

```
                               'aa': T.toScalars(dirichlet(ones([2,1]))),
```

```
                               'aaa': T.toScalars(dirichlet(ones([2,1])))}
```

```
  var vendingMachine = function(state, action) {
```

```
    return categorical({vs: ['bagel', 'cookie'], ps: buttonsToOutcomeProbs[action]
```

```

```

```
  }

  var goal = categorical({vs: ['bagel', 'cookie'], ps: [.5, .5]})

  var goalSatisfied = function(outcome) {return outcome == goal}

  var chosenAction = sample(chooseAction(goalSatisfied, vendingMachine, 'state'))

  condition(goal == 'cookie' && chosenAction == 'a')

  return {once: buttonsToOutcomeProbs['a'][1],
          twice: buttonsToOutcomeProbs['aa'][1]}
})

print("probability of actions giving a cookie")

viz.marginals(goalPosterior)
```

run ▼

Compare the inferences that result if Sally presses the button twice to those if she only presses the button once. Why can we draw much stronger inferences about the machine when Sally chooses to press the button twice? When Sally does press the button twice, she could have done the "easier" (or rather, a priori more likely) action of pressing the button just once. Since she doesn't, a single press must have been unlikely to result in a cookie. This is an example of the *principle of efficiency*—all other things being equal, an agent will take the actions that require least effort (and hence, when an agent expends more effort all other things must not be equal). Here, Sally has three possible actions but simpler actions are a priori more likely.

In these examples we have seen two important assumptions combining to allow us to infer something about the world from the indirect evidence of an agents actions. The first assumption is the principle of rational action, the second is an assumption of *knowledgeability*—we assumed that Sally knows how the machine works, though we don't. Thus inference about inference, can be a powerful way to learn what others already know, by observing their actions. (This example was inspired by Goodman et al. (2009)

(https://scholar.google.com/scholar?q="Cause%20and%20intent%3A%20Social%20reasoning%20in%20causal%20learning"))

Joint inference about beliefs and desires

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.

```
///fold:
...
var actionPrior = function() {
  return categorical({vs: ['a', 'aa', 'aaa'], ps:[0.7, 0.2, 0.1] })
}

var goalPosterior = Infer({method: 'rejection', samples: 5000}, function() {
  var buttonsToOutcomeProbs = {'a': T.toScalars(dirichlet(ones([2,1]))),
                               'aa': T.toScalars(dirichlet(ones([2,1]))),
                               'aaa': T.toScalars(dirichlet(ones([2,1])))}

  var vendingMachine = function(state, action) {
    return categorical({vs: ['bagel', 'cookie'], ps: buttonsToOutcomeProbs[action]
  }
```

```
  var goal = categorical({vs: ['bagel', 'cookie'], ps: [.5, .5]})

  var goalSatisfied = function(outcome) {return outcome == goal}

  var chosenAction = sample(chooseAction(goalSatisfied, vendingMachine, 'state'))


  condition(chosenAction == 'aa')

  condition(vendingMachine('state', 'aa') == 'cookie')


  return {goal: goal,

          once: buttonsToOutcomeProbs['a'][1],

          twice: buttonsToOutcomeProbs['aa'][1]};

})


print("probability of actions giving a cookie")

viz.marginals(goalPosterior)
```

run ▼

Notice the U-shaped distribution for the effect of pressing the button just once. Without any direct evidence about what happens when the button is pressed just once, we can infer that it probably won't give a cookie—because her goal is likely to have been a cookie but she didn't press the button just once—but there is a small chance that her goal was actually not to get a cookie, in which case pressing the button once could result in a cookie. This very complex (and hard to describe!) inference comes naturally from joint inference of goals and knowledge.

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 (throughout this section we simplify the code by specializing to the situation at hand, rather than using the more general `chooseAction` function introduced above):

```
var teacher = function(die) {
  return Infer(..., function() {
    var side = sample(sidePrior);
    condition(learner(side) == die)
    return side;
  })
}
```

The goal of the learner is to infer the correct hypothesis, given that the teacher chose to give these examples:

```
var learner = function(side) {
  return Infer(..., function() {
    var die = sample(diePrior);
    condition(teacher(die) == side)
    return die;
  })
}
```

This pair of mutually recursive functions represents a teacher choosing examples or a learner inferring a hypothesis, each thinking about the other. However, notice that this recursion will never halt—it will be an unending chain of "I think that you think that I think that...". To avoid this infinite recursion say that eventually the learner will just assume that the teacher rolled the die and showed the side that came up (rather than reasoning about the teacher choosing a side):

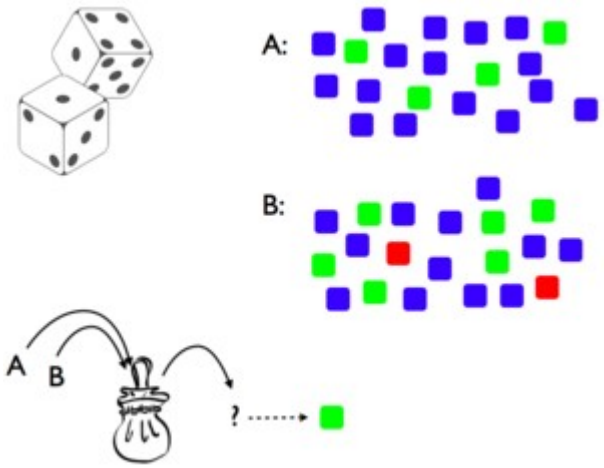
```
var teacher = function(die, depth) {
  return Infer(..., function() {
    var side = sample(sidePrior);
    condition(learner(side, depth) == die)
    return side
  })
}

var learner = function(side, depth) {
  return Infer(..., function() {
    var die = sample(diePrior);
    condition(depth == 0 ?
              side == roll(die) :
              side == teacher(die, depth - 1))
    return die
  })
}
```

看看你在第几层……

To make this concrete, assume that there are two dice, A and B, which each have three

sides (red, green, blue) that have weights like so:



Which hypothesis will the learner infer if the teacher shows the green side?

```
var dieToProbs = function(die) {
  return (die == 'A' ? [0, .2, .8] :
          die == 'B' ? [.1, .3, .6] :
          'uhoh')
}

var sidePrior = Categorical({vs: ['red', 'green', 'blue'], ps: [1/3, 1/3, 1/3]})
var diePrior = Categorical({vs: ['A', 'B'], ps: [1/2, 1/2]})
var roll = function(die) {return categorical({vs: ['red', 'green', 'blue'], ps: dieToProbs(die

var teacher = function(die, depth) {
  return Infer({method: 'enumerate'}, function() {
    var side = sample(sidePrior);
    condition(sample(learner(side, depth)) == die)
```

```
    return side
  })
}

var learner = function(side, depth) {
  return Infer({method: 'enumerate'}, function() {
    var die = sample(diePrior);
    condition(depth == 0 ?
              side == roll(die) :
              side == sample(teacher(die, depth - 1)))
    return die
  })
}

viz.auto(learner('green', 3))
```

run ▼

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 B most of the time (about 60% of the time). This is the same as using the "strong sampling" assumption: the learner infers B because B is more likely to have landed on side 2. However, if we increase the recursion depth we find this reverses: the learner infers B only about 40% of the time. Now die A becomes the better inference, because "if the teacher had meant to communicate B, they would have shown the red side because that can never come from A."

这怎么做到的……感觉就像变魔术

This model, has been proposed by Shafto et al. (2012) (https://langcog.stanford.edu/papers/SGF-perspectives2012.pdf) 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 are truth-functional: that each sentence corresponds to a function from states of the world to true/false. mapping is onto

As above, the speaker chooses what to say in order to lead the listener to infer the correct state:

```
var speaker = function(state) {
  return Infer(..., function() {
    var words = sample(sentencePrior);
    condition(state == listener(words));
    return words;
  });
};
```

The listener does an inference of the state of the world given that the speaker chose to say what they did:

```
var listener = function(words) {
  return Infer(..., function() {
    var state = sample(statePrior);
    condition(words == speaker(state));
    return state;
  });
};
```

However this suffers from two flaws: the recursion never halts, and the literal meaning has not been used. We slightly modify the listener function such that the listener either assumes that the literal meaning of the sentence is true, or figures out what the speaker must have meant given that they chose to say what they said:

```
var listener = function(words) {
  return Infer(..., function() {
    var state = sample(statePrior);
    condition(flip(literalProb) ?
              words(state) :
              words == speaker(state))
    return state;
  });
};
```

这样比给定想到第几层更科学

Here the probability `literalProb` controls the expected depth of recursion. Another ways to bound the depth of recursion is with an explicit depth argument (which is decremented on each recursion).

We have used a standard, truth-functional, formulation for the meaning of a sentence:

each sentence specifies a (deterministic) predicate on world states. Thus the literal meaning of a sentence specifies the worlds in which the sentence is satisfied. That is the literal meaning of a sentence is the sort of thing one can condition on, transforming a prior over worlds into a posterior. Here's another way of motivating this view: meanings are belief update operations, and since the right way to update beliefs coded as distributions is conditioning, meanings are conditioning statements. Of course, the deterministic predicates can be immediately (without changing any other code) relaxed to probabilistic truth functions, that assign a probability to each world. This might be useful if we want to allow exceptions.

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):

```
var allSprouted = function(state) {return state == 3}

var someSprouted = function(state) {return state > 0}

var noneSprouted = function(state) {return state == 0}

var meaning = function(words) {

  return (words == 'all' ? allSprouted :

          words == 'some' ? someSprouted :

          words == 'none' ? noneSprouted :

          console.error("unknown words"))

}

var statePrior = Categorical({vs: [0,1,2,3],

                              ps: [1/4, 1/4, 1/4, 1/4]})

var sentencePrior = Categorical({vs: ["all", "some", "none"],
```

```
                                ps: [1/3, 1/3, 1/3]})
```

```

```

```
var speaker = function(state, depth) {
```

```
  return Infer({method: 'enumerate'}, function() {
```

```
    var words = sample(sentencePrior)
```

```
    condition(state == sample(listener(words, depth)))
```

```
    return words
```

```
  })
```

```
};
```

```

```

```
var listener = function(words, depth) {
```

```
  return Infer({method: 'enumerate'}, function() {
```

```
    var state = sample(statePrior);
```

```
    var wordsMeaning = meaning(words)
```

```
    condition(depth == 0 ? wordsMeaning(state) :
```

```
              _.isEqual(words, sample(speaker(state, depth - 1))))
```

```
    return state;
```

```
  })
```

```
};
```

```

```

```
viz.auto(listener("some", 1))
```

run ▼

We see that if the listener hears “some” the probability of three out of three is low, even though the basic meaning of “some” is equally consistent with 3/3, 1/3, and 2/3. This is called the “some but not all” implicature.

Reading & Discussion: Readings (/readings/social-cognition.html)

Test your knowledge: Exercises (/exercises/social-cognition.html)

Next chapter: 20. Appendix - JavaScript basics (/chapters/appendix-js-basics.html)