Basics

Probabilistic programming is about programming with measures. Probability distributions *are* measures (e.g. functions from subsets of something called a Borel algebra to the positive real numbers which some mathematical properties which I won't get into).

So the whole point of probabilistic programming is to make it easier for users to specify measures, and do some common operations on them.

A density (like a Normal distribution) is related to measures - a density is an object which described how a measure is related to another measure (a so-called *base measure*). For the Normal density, it's a density with respect to something called the Lebesgue measure on the real numbers. (More mathematical: a density is the Radon-Nikodym of a measure $\partial\mu$ with respect to another measure $\partial\nu$.)

In probabilistic + Bayesian reasoning, a model describes a measure but we normally just talk about the density.

So a model like:

```
(burglary)          (disabled)
          \        /
           (alarm)
```

describes a distribution `P(alarm, burglary, disabled)` which a modeler would normally decompose `P(alarm | burglary, disabled)P(burglary)P(disabled)`. Then the modeler would construct a density by making some assumptions about each of these components (e.g. that `P(alarm | burglary, disabled)` is Bernoulli (which produces a "density mass function" aka a Radon-Nikodym derivative with respect to the base counting measure (which assigns to each set $A \in N$ the size of the set)).

Cool, so you don't really have to think about any of that when using one of these frameworks. Really only language designers / compiler writers must think about this stuff.

You can just write:

```
@gen function burglary_model()::Nothing
    burglary ~ Bernoulli(0.05)
    disabled ~ Bernoulli(0.03)
    !disabled ? alarm ~ Bernoulli(burglary ? 0.94 : 0.01) : alarm ~ Bernoulli(0.01)
    call ~ Bernoulli(alarm ? 0.70 : 0.05)
    return nothing
end
```

This describes a joint prior `P(:burglary, :disabled, :alarm, :call)`.

Intermediate

Why is this paradigm useful? Why is computing *anything* useful? The basic answer is that we can map models of the world onto objects which we can manipulate (there *is* a category theoretic perspective on this, obv - for those programming nerds).

In this case, our objects are probabilistic measures - what are the manipulations? Measure preserving transformations (e.g. morphisms in the category of measures). There's a few which we care about - but I'll only talk about one.

Disintegration

Distintegration is a model for conditioning. E.g. if we observe `:alarm = true` and we want to compute the distribution `P(:burglary | :alarm = true)` from the prior distribution `P(:burglary, :disabled, :alarm, :call)`.

Please see this wonderful article for disintegration.

This is how Bayes' theorem arises - Bayes' theorem represents an inference methodology (when you think about mathematical transformations that represent the operation of inference) - but it can also be derived straight from the math of disintegration.

Bayes' rule/theorem

$$P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

(I'm not going to go into the measure theoretic treatment - but it's basically the conditional density is the Radon-Nikodym derivative of the restriction of a measure with respect to a base measure).

Bayes also represents an inference methodology. If we imagine `B = obs`, Bayes tells us how to update the *prior* `P(A)` with the observation `B = obs` to arrive at the *posterior* `P(A | B)`. It represents an aesthetically elegant way to update your model in light of new knowledge/observations.

But there's a problem...

Conditioning is hard

At the bottom (denominator) of the RHS of Bayes is an integral

$$P(B) = \int_A P(A,B)\ dA$$

Integrals are difficult - they are not easy to solve. For restricted classes of models, you can compute this analytically (such models possess a property called

conjugacy). Of course, we don't want to live in the restricted space because sometimes we can't model what we want in that space.

This is the fundamental issue. All the myriad of different Bayesian frameworks / inference algorithms take different perspectives on solving this issue: how do we get around this integral?

Here are a couple of philosophies:

0. Compute the integral exactly or approximately using analytic properties of `P(A, B)` → (Exact enumeration/variable elimination and belief propagation)
1. Compute the integral by randomly walking in a high-dimensional space → (Markov chain Monte Carlo)
2. Compute the integral by transforming it into an optimization problem with an approximate solution → (Variational inference)
3. Factor the integral by making assumptions about `P(A, B)` and then use (0), (1), or (2) → (Graphical models)

(0, 1, 2) really represent concrete inference methodologies, (3) is really about modeling assumptions.

> Note the interplay! Modeling assumptions can reduce the difficulty of performing inference - more effective inference methodologies can expand the expressiveness of modeling, etc.