```
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`
```

Causal Dependence

Probabilistic programs encode knowledge about the world in the form of causal models, and it is useful to understand how their function relates to their structure by thinking about some of the intuitive properties of causal relations. Causal relations are local, modular, and directed.

- They are *modular* in the sense that any two arbitrary events in the world are most likely causally unrelated, or independent. If they are related, or dependent, the relation is only very weak and liable to be ignored in our mental models.
- Causal structure is *local* in the sense that many events that are related are not related directly: They are connected only through causal chains of several steps, a series of intermediate and more local dependencies.
- And the basic dependencies are directed: when we say that A causes B, it means something different than saying that B causes A. The *causal influence* flows only one way along a causal relation—we expect that manipulating the cause will change the effect, but not vice versa — but *information* can flow both ways—learning about either event will give us information about the other.

Let's examine this notion of "causal dependence" a little more carefully. What does it mean to believe that A depends causally on B? Viewing cognition through the lens of probabilistic programs, the most basic notions of causal dependence are in terms of the structure of the program and the flow of evaluation (or "control") in its execution. We say that expression A causally depends on expression B if it is necessary to evaluate B in order to evaluate A. (More precisely, expression A depends on expression B if it is ever necessary to evaluate B in order to evaluate A.) For instance, in this program `A` depends on `B` but not on `C` (the final expression depends on both `A` and `C`):

```
true
```

```
let
    C = @~ Bernoulli()
    B = @~ Bernoulli()
    A = ifelse.(B, (@~ Bernoulli(0.1)), (@~ Bernoulli(0.4)))
    randsample(A .| C)
end
```

Note that causal dependence order is weaker than a notion of ordering in time—one expression might happen to be evaluated before another in time (for instance `C` before `A`), but without the second expression requiring the first. (This notion of causal dependence is related to the notion of flow dependence in the programming language literature.)

For example, consider a simpler variant of our medical diagnosis scenario:

```
smokes = -4895573358191366609@Distributions.Bernoulli{Float64}(p=0.2)
```

```
smokes = @~ Bernoulli(0.2)
```

```
lung_disease =
|ₚ(&ₚ(-4895573358191366609@Distributions.Bernoulli{Float64}(p=0.2), 799298153830084005@Dist
```

```
lung_disease = (smokes .& (@~ Bernoulli(0.1))) .| (@~ Bernoulli(0.001))
```

```
cold = 4005626225151056973@Distributions.Bernoulli{Float64}(p=0.02)
```

```
cold = @~ Bernoulli(0.02)
```

```
cough =
|ₚ(&ₚ(4005626225151056973@Distributions.Bernoulli{Float64}(p=0.02), 8146438696559316210@Dis
```

```
cough = pw(|,
    (cold .& @~ Bernoulli()),
    (lung_disease .& @~ Bernoulli()),
    @~ Bernoulli(0.001)
)
```

```
fever =
|ₚ(&ₚ(4005626225151056973@Distributions.Bernoulli{Float64}(p=0.02), -3109735574544004429@Di
```

```
fever = (cold .& @~ Bernoulli()) .| @~ Bernoulli(0.01)
```

```
chest_pain =
|ₚ(&ₚ(|ₚ(&ₚ(-4895573358191366609@Distributions.Bernoulli{Float64}(p=0.2), 79929815383008400
```

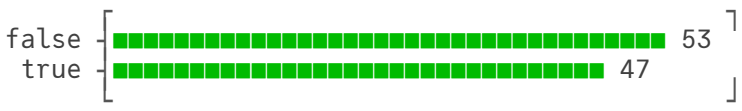
```
chest_pain = (lung_disease .& @~ Bernoulli(0.2)) .| @~ Bernoulli(0.01)
```

```
shortness_of_breath =
|ₚ(&ₚ(|ₚ(&ₚ(-4895573358191366609@Distributions.Bernoulli{Float64}(p=0.2), 79929815383008400
```

```
shortness_of_breath = (lung_disease .& @~ Bernoulli(0.2)) .| @~ Bernoulli(0.01)
```

```
cold_cond =
  Conditional(4005626225151056973@Distributions.Bernoulli{Float64}(p=0.02), |ₚ(&ₚ(400562622
```

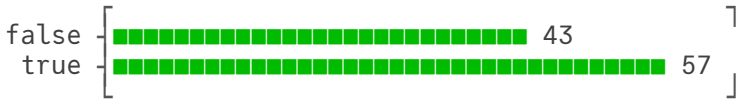
```
cold_cond = cold |ᶜ cough
```




```
viz(randsample(cold_cond, 100))
```

```
lung_disease_cond =
  Conditional(|ₚ(&ₚ(-4895573358191366609@Distributions.Bernoulli{Float64}(p=0.2), 7992981538
```

```
lung_disease_cond = lung_disease |ᶜ cough
```




```
viz(randsample(lung_disease_cond, 100))
```

Here, `cough` depends causally on both `lung_disease` and `cold`, while `fever` depends causally on `cold` but not `lung_disease`. We can see that `cough` depends causally on `smokes` but only indirectly: although `cough` does not call `smokes` directly, in order to evaluate whether a patient coughs, we first have to evaluate the expression `lung_disease` that must itself evaluate `smokes`.

We haven't made the notion of "direct" causal dependence precise: do we want to say that `cough` depends directly on `cold`, or only directly on the expression `(cold &ₚ @~ Bernoulli()) |ₚ ...`? This can be resolved in several ways that all result in similar intuitions. For instance, we could first re-write the program into a form where each intermediate expression is named (called A-normal form) and then say direct dependence is when one expression immediately includes the name of another.

There are several special situations that are worth mentioning. In some cases, whether expression A requires expression B will depend on the value of some third expression C. For example, here is a particular way of writing a noisy-AND relationship:

```
true
```

```
let
    C = @~ Bernoulli()
    B = @~ Bernoulli()
    A = ifelse.(C, ifelse.(B, (@~ Bernoulli(0.85)), false), false)
    randsample(A)
end
```

`A` always requires `C`, but only evaluates `B` if `C` returns true. Under the above definition of causal dependence `A` depends on `B` (as well as `C`). However, one could imagine a more fine-grained notion of causal dependence that would be useful here: we could say that `A` depends causally on `B` only in certain contexts (just those where `C` happens to return true and thus `A` calls `B`).

Another nuance is that an expression that occurs inside a function body may get evaluated several times in a program execution. In such cases it is useful to speak of causal dependence between specific evaluations of two expressions. (However, note that if a specific evaluation of `A` depends on a specific evaluation of `B`, then any other specific evaluation of `A` will depend on some specific evaluation of `B`)

Detecting Dependence Through Intervention

The causal dependence structure is not always immediately clear from examining a program, particularly where there are complex functions calls. Another way to detect (or according to some philosophers, such as Jim Woodward, to *define*) causal dependence is more operational, in terms of "difference making": If we manipulate A, does B tend to change? By *manipulate* here we don't mean an assumption in the sense of conditioning. Instead we mean actually edit, or *intervene on*, the program in order to make an expression have a particular value independent of its (former) causes. If setting A to different values in this way changes the distribution of values of B, then B causally depends on A.

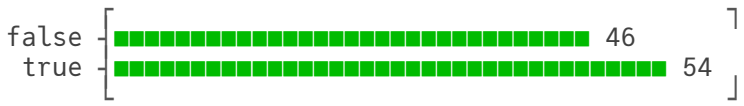
This method is known in the causal Bayesian network literature as the "do operator" or graph surgery (Pearl, 1988). It is also the basis for interesting theories of counterfactual reasoning by Pearl and colleagues (Halpern, Hitchcock and others).

For example, in the above example of medical diagnosis, we now give our hypothetical patient a cold — say, by exposing him to a strong cocktail of cold viruses. We should not model this as an

observation (e.g. by conditioning on having a cold), because we have taken direct action to change the normal causal structure. Instead, we implement intervention by directly editing the random variables:

```
cough_intervened =
|ₚ(&ₚ(4005626225151056973@Distributions.Bernoulli{Float64}(p=0.02), 8146438696559316210@Dis
```

```
cough_intervened = cough |ᵈ (cold => true)
```




```
viz(randsample(cough_intervened, 100))
```

You should see that the distribution on `cough` changes: coughing becomes more likely if we know that a patient has been given a cold by external intervention. But the reverse is not true: Try forcing the patient to have a cough (e.g., with some unusual drug or by exposure to some cough-inducing dust) by writing `cough => true` instead of `cold => true`: the distribution on `cold` is unaffected. We have captured a familiar fact: treating the symptoms of a disease directly doesn't cure the disease (taking cough medicine doesn't make your cold go away), but treating the disease *does* relieve the symptoms.

Verify in the program above that the method of manipulation works also to identify causal relations that are only indirect: for example, force a patient to smoke and show that it increases their probability of coughing, but not vice versa.

If we are given a program representing a causal model, and the model is simple enough, it is straightforward to read off causal dependencies from the program code itself. However, the notion of causation as difference-making may be easier to compute in much larger, more complex models—and it does not require an analysis of the program code. As long as we can modify (or imagine modifying) the definitions in the program and can run the resulting model, we can compute whether two events or functions are causally related by the difference-making criterion.

Statistical Dependence

One often hears the warning, "correlation does not imply causation". By "correlation" we mean a different kind of dependence between events or functions—statistical dependence. We say that A and B are statistically dependent, if learning information about A tells us something about B, and vice versa. Statistical dependence is a symmetric relation between events referring to how information flows between them when we observe or reason about them. (If conditioning on A changes B, then conditioning on B also changes A) The fact that we need to be warned against confusing statistical and causal dependence suggests they are related, and indeed, they are. In general, if A causes B, then A and B will be statistically dependent. (One might even say the two notions are "causally related", in the sense that causal dependencies give rise to statistical dependencies.)

Diagnosing statistical dependence by conditioning is similar to diagnosing causal dependence through intervention. We condition on various values of the possible statistical dependent, here A, and see whether it changes the distribution on the target, here B:

```
A = -7010358001857595108@Distributions.Bernoulli{Float64}(p=0.5)
```

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

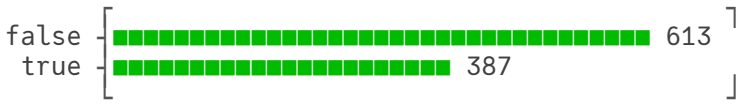
```
C = 4340529198860691160@Distributions.Bernoulli{Float64}(p=0.5)
```

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

```
B = ifelse_p(4340529198860691160@Distributions.Bernoulli{Float64}(p=0.5), -7276244943751792723@D
```

```
B = ifelse.(C, (@~ Bernoulli(0.1)), (@~ Bernoulli(0.4)))
```


```
viz(randsample((B |ᶜ (C .== true)), 1000))
```




```
viz(randsample((B |ᶜ (C .== false)), 1000))
```

Because the two distributions on B (when we have different information about C) are different, we can conclude that B statistically depends on B. Do the same procedure for testing if C statistically depends on B. How is this similar (and different) from the causal dependence between these two? As an exercise, make a version of the above medical example to test the statistical dependence between cough and cold. Verify that statistical dependence holds symmetrically for events that are connected by an indirect causal chain, such as smokes and coughs.

Correlation is not just a symmetrized version of causality. Two events may be statistically dependent even if there is no causal chain running between them, as long as they have a common cause (direct or indirect). Here is an example of statistical dependence generated by a common cause:

```
X = -9116344323510361582@Distributions.Bernoulli{Float64}(p=0.5)
```

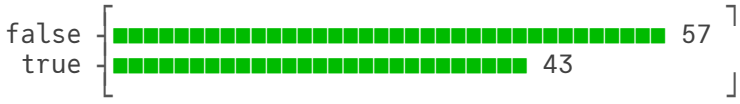
```
X  = @~ Bernoulli()
```

```
Y = ifelse_p(-9116344323510361582@Distributions.Bernoulli{Float64}(p=0.5), 8420916776214848160@D
```

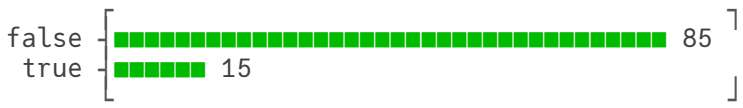
```
Y = ifelse.(X, (@~ Bernoulli()), (@~ Bernoulli(0.9)))
```

```
Z = ifelse_p(-9116344323510361582@Distributions.Bernoulli{Float64}(p=0.5), 6287078598729496171@D
```

```
Z = ifelse.(X, (@~ Bernoulli(0.1)), (@~ Bernoulli(0.4)))
```




```
viz(randsample((Z |ᶜ (Y .== true)), 100))
```




```
viz(randsample((Z |ᶜ (Y .== false)), 100))
```

Situations like this are extremely common. In the medical example above, `cough` and `fever` are not causally dependent but they are statistically dependent because they both depend on `cold`; likewise for `chest_pain` and `shortness_of_breath` which both depend on `lung_disease`. Here we can read off these facts from the program definitions, but more generally all of these relations can be diagnosed by reasoning using `|ᶜ`.

Successful learning and reasoning with causal models typically depend on exploiting the close coupling between causation and correlation. Causal relations are typically unobservable, while correlations are observable from data. Noticing patterns of correlation is thus often the beginning of causal learning, or discovering what causes what. On the other hand, with a causal model already in place, reasoning about the statistical dependencies implied by the model allows us to predict many aspects of the world not directly observed from those aspects we do observe.

Graphical Notations for Dependence

Graphical models are an extremely important idea in modern machine learning: a graphical diagram is used to represent the direct dependence structure between random choices in a probabilistic model. A special case are *Bayesian networks*, in which there is a node for each random variable (an expression in our terms) and a link between two nodes if there is a direct conditional dependence between them (a direct causal dependence in our terms). The sets of nodes and links define a *directed acyclic graph* (hence the term graphical model), a data structure over which many efficient algorithms can be defined. Each node has a *conditional probability table* (CPT), which represents the probability distribution of that node, given values of its parents. The joint probability distribution over random variables is given by the product of the conditional distributions for each variable in the graph.

Simple generative models will have a corresponding graphical model that captures all of the dependencies (and independencies) of the model, without capturing the precise form of these functions. The CPTs provide a less compact representation of the conditional probabilities compared to Omega programs, which express the *structural causal model* (SCM) of the models).

More complicated generative models, which can be expressed as probabilistic programs, often don't have a graphical model (or rather they have many approximations, none of which captures all independencies). Recursive models generally give rise to ambiguous (or loopy) Bayes nets.