Causal Models: Guide to `CausalQueries`

Macartan Humphreys and Alan Jacobs

2023-08-31

Contents

Preface

Map

This guide is supplementary material for our book. *Integrated Inferences* (Humphreys and Jacobs, 2023).

- The first part of the guide provides a brief motivation of causal models.
- The second part describes how the package works and how to use it.
- The third part illustrates applications of the package for defining and learning from a set of canonical causal models.
- The short last part has a notation guide.

Credits

The approach used in `CausalQueries` is a generalization of the "biqq" models described in "Mixing Methods: A Bayesian Approach" (Humphreys and Jacobs, 2015). The conceptual extension makes use of work on probabilistic causal models described in Pearl's *Causality* (Pearl, 2009). The approach to generating a generic stan function that can take data from arbitrary models was developed in key contributions by Jasper Cooper and Georgiy Syunyaev. Lily Medina did magical work pulling it all together and developing approaches to characterizing confounding and defining estimands. Clara Bicalho helped figure out a nice syntax for causal statements. Julio Solis made many key contributions figuring out how to simplify the specification of priors. Merlin Heidemanns figure out the `rstantools` integration and made myriad code improvements. Till Tietz revamped the package and improved every part of it.

Part I

Causal Models

Chapter 1

What and why

The `CausalQueries` package is designed to make it easy to *build*, *update*, and *query* causal models defined over binary variables.

The causal models we use are of the form described by Pearl (2009), with Bayesian updating on the causal models similar to that described by Cowell et al. (1999). Though drawing heavily on the Pearlian framework, the approach specifies parameters using potential outcomes (specifically, using principal stratificaton (Frangakis and Rubin, 2002)) and uses the `stan` framework (Carpenter et al., 2017) to implement updating.

We will illustrate how to use these models for a number of inferential tasks that are sometimes difficult to implement:

- **Bayesian process tracing**: How to figure out what to believe about a case-level causal claim given one or more pieces of evidence about a case (Bennett and Checkel, 2015).
- **Mixed methods**: How to combine within-case ("qualitative") evidence with cross-case ("quantitative") evidence to make causal claims about cases or groups (Humphreys and Jacobs, 2015).
- **Counterfactual reasoning**: How to estimate the probability that an outcome is due to a cause and other counterfactual queries (effects of causes, probability of sufficiency, explanation) (Tian and Pearl, 2000).
- **Inference in the absence of causal identification**: What inferences can you draw when causal quantities are not identified? How can you figure out *whether* a causal quantity is identified (Manski, 1995)?

- **Extrapolation, data fusion**: How to draw inferences from mixtures of observational and experimental data? How to draw out-of-sample inferences given a theory of how one place differs from another (Bareinboim and Pearl, 2016)?

The functions in the package allow these different kinds of questions to be answered using the same three basic steps — `make_model`, `update_model`, and `query_model` — without recourse to specific estimators for specific estimands.

The approach, however, requires thinking about causal inference differently from how many in the experimental tradition are used to.

1.1 Two approaches to inference

We contrast the two approaches to causal inference using the simplest problem: the analysis of data from a two-arm experimental trial to determine the average effect of a treatment on an outcome.

In the canonical experimental trial, a treatment, X, is randomly assigned and outcomes, Y, are measured in both treatment and control groups. The usual interest is in understanding the average effect of X on Y (which we will assume are both binary).

The classic approach to answering this question is to take the difference between outcomes in treatment and control groups as an estimate of this average treatment effect. Estimates of uncertainty about this answer can also be generated using information on variation in the treatment and control groups.

It is also possible, however, to answer this question—along with a rich variety of other questions—using a *causal models* approach (Pearl, 2009).

For intuition for how a causal model approach works, say instead of simply taking the differences between treatment and control one were to:

1. construct a simple model in which (a) X is randomly assigned, with 50% probability and (b) we specify some prior beliefs about how Y responds to X, which could be positively, negatively, or not at all, and possibly different for each unit

Table 1.1: Inferences on the ATE from differences in means

	Estimate	Std. Error	t value	Pr(>|t|)	CI Lower	CI Upper	DF
X	0.484	0.028	17.48	0	0.43	0.538	996.4

2. update beliefs about how Y responds to X given the data on X and Y

Though very simple, this $X \rightarrow Y$ model adds a great deal of structure to the situation It is, in fact, more of a model than you need if all you want to do is estimate average treatment effects. But it is nevertheless enough of a model to let you estimate quantities—such as causal attribution—that you could not estimate without a model, even given random assignment.

1.2 Recovering the ATE with Difference in Means

To illustrate, imagine that *in actuality* (but unknown to us) X shifts Y from 0 to 1 in 50% of cases while in the rest Y is 0 or 1 regardless.

We imagine we have access to some data in which treatment, X, has been randomly assigned:

```
# Fabricate data using fabricatr
set.seed(17022020)
data  <- fabricate(N = 1000, X = complete_ra(N), Y = 1*(runif(N, -1, 3)/2 < X))
```

The classic experimental approach to causal inference is to estimate the effect of X on Y using differences in means: taking the difference between the average outcome in treatment and the average outcome in control. Thanks to randomization, in expectation that difference is equal to the average of the differences in outcomes that units would exhibit if they were in treatment versus if they were in control—that is, the causal effect.

```
difference_in_means(Y~X, data = data)
```

This approach gets us an accurate and precise answer, and it's simply done (here using a function from the `estimatr` package).

Table 1.2: Inferences on the ATE from updated model

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	posteriors	FALSE	0.4802	0.0283	0.4242	0.5335

1.3 Recovering the ATE with a Causal Model

The model-based approach takes a few more lines of code and is implemented in `CausalQueries` as follows.

First we define a model, like this:

```
model <- make_model("X -> Y")
```

Implicit in the definition of the model is a set of parameters and priors over these parameters. We discuss these in much more detail later, but for now we will just say that priors are uniform over possible causal relations.

Second, we update the model, like this:

```
model <- update_model(model, data)
```

Third, we query the model like this:

```
query_model(model, using  = "posteriors", query = "Y[X=1] - Y[X=0]")
```

We see that the answers we get from the differences-in-means approach and the causal-model approach are about the same, as one would hope.

1.4 Going further

In principle, however, the causal models approach can let you do more. The third section of this guide is full of examples, but for a simple one consider the following: say, instead of wanting to know the average effect of X on Y you wanted to know, "What is the probability that X caused Y in a case in which $X = Y = 1$?"

This is a harder question. Differences-in-means is an estimation strategy tied to a particular estimand, and it does not provide a good answer to this question. However, with a causal model in hand, *we can ask whatever causal*

Table 1.3: Causes of effects estimand

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	X==1 & Y==1	posteriors	FALSE	0.7936	0.0941	0.6296	0.9663

question we like, and get an answer plus estimates of uncertainty around the answer.

Here is the answer we get:

```
query_model(model,
            using = "posteriors",
            query = "Y[X=0] == 0",
            given = "X==1 & Y==1")
```

In this case we are asking *for those cases in which* $X = 1$ *and* $Y = 1$, what are the chances that Y would have been 0 if X were 0?

Note, however, that while the model gave a precise answer to the ATE question, the answer for the causes-of-effects estimand is not precise. Moreover, more data won't reduce the uncertainty substantially. The reason is that this estimand is not identified.

This then is a situation in which we can ask a question about a quantity that is not identified and still learn a lot. We will encounter numerous examples like this as we explore different causal models.

Part II

The Package

Chapter 2

Installation

You can install the package from Rstudio via:

```
install.packages("CausalQueries")
```

The package has some dependencies, the most important of which is `rstan`. Make sure you have `rstan` installed and working properly before doing anything else. Note that at installation `CausalQueries` compiles and stores a `stan` model so if you have problems with `stan` you won't be able to install `CausalQueries`.

- See the rstan getting started guide for general installation procedures
- Check that rstan is working by running one of the simple `stan` examples (see for example `?stan`)
- For windows users you might have to configure according to (these instructions](https://github.com/stan-dev/rstan/wiki/Installing-RStan-from-source-on-Windows#configuration)
- if all is working you should be able to run a simple model of the form:

```
make_model("X->Y") |> update_model()
```

Chapter 3

Defining models

3.1 Getting going

Model definition involves:

- Defining a causal structure (required)
- Indicating causal type restrictions (optional)
- Indicating possible unobserved confounding (optional)
- Providing priors and parameters (required, but defaults are provided)

We discuss these in turn.

3.2 Causal structure

A simple model is defined in one step using a `dagitty` syntax in which the structure of the model is provided as a statement.

For instance:

```
model <- make_model("X -> M -> Y <- X")
```

The statement (in quotes) provides the names of nodes. An arrow ("`->`" or "`<-`") connecting nodes indicates that one node is a potential cause of another (that is, whether a given node is a "parent" or "child" of another; see section 12.1.1).

Formally a statement like this is interpreted as:

1. Functional equations:

- $Y = f(M, X, \theta^Y)$
- $M = f(X, \theta^M)$
- $X = \theta^X$

2. Distributions on shocks:

- $\Pr(\theta^i = \theta^i_k) = \lambda^i_k$

3. Independence assumptions:

- $\theta_i \perp\!\!\!\perp \theta_j, i \neq j$

where function f maps from the set of possible values of the parents of i to values of node i given θ^i. Units with the same value on θ^i react in the same way to the parents of i. Indeed in this discrete world we think of θ^i as fully dictating the functional form of f, indicating what outcome is observed on i for any value of i's parents.

In addition, it is also possible to indicate "unobserved confounding", that is, the presence of an unobserved variable that might influence observed variables. In this case condition 3 above is relaxed. We describe how this is done in greater detail in section 3.4.

For instance:

```
model <- make_model("X -> Y <- W -> X; X <-> Y")
```

Note that different segments of the graph can be included in the same statement, separated by semicolons. There are many ways to write the same model. Here is the same model written once using a three part statement and once as a chain (with the same node appearing possibly more than once).

```
model <- make_model("W -> X; W -> Y; X -> Y")
model <- make_model("X -> Y <- W -> X")
```

Once defined, a model can be graphed (we use the `dagitty` package for this):

```
model <- make_model("X -> Y <- W -> X; X <-> Y")
plot(model)
```

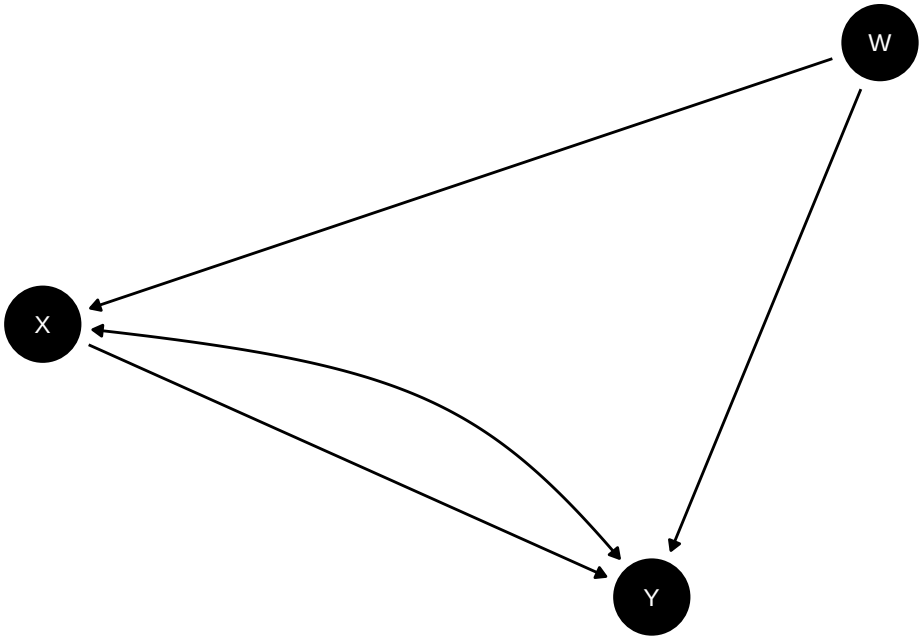


Figure 3.1: A plotted model. Curved double headed arrows indicate unobserved confounding

This is useful to check that you have written the structure down correctly.

When a model is defined, a set of objects are generated. These are the key quantities that are used for all inference. We discuss each in turn. (See the notation guide—section 12—for definitions and code pointers).

3.2.1 Nodal types

Two units have the same *nodal type* at node Y, θ^Y, if their outcome at Y responds in the same ways to parents of Y.

A node with k parents has 2^{2^k} nodal types. The reason is that with k parents, there are 2^k possible values of the parents and so 2^{2^k} ways to respond to these possible parental values. As a convention we say that a node with no parents has two nodal types (0 or 1).

When a model is created the full set of "nodal types" is identified. These are stored in the model. The subscripts become very long and hard to parse for more complex models so the model object also includes a guide to interpreting nodal type values. You can see them like this.

```
make_model("X -> Y")$nodal_types
```

```
$X
[1] "0" "1"

$Y
[1] "00" "10" "01" "11"

attr(,"interpret")
attr(,"interpret")$X
  node position display interpretation
1    X       NA      X0           X = 0
2    X       NA      X1           X = 1

attr(,"interpret")$Y
  node position display interpretation
1    Y        1   Y[*]*       Y | X = 0
2    Y        2   Y*[*]       Y | X = 1
```

Note that we use θ^j to indicate nodal types because for qualitative analysis the nodal types are often the parameters of interest.

3.2.2 Causal types

Causal types are collections of nodal types. Two units are of the same *causal type* if they have the same nodal type at every node. For example in a $X \rightarrow M \rightarrow Y$ model, $\theta = (\theta^X_0, \theta^M_{01}, \theta^Y_{10})$ is a type that has $X = 0$, M responds positively to X, and Y responds positively to M.

When a model is created, the full set of causal types is identified. These are stored in the model object:

```
make_model("A -> B")$causal_types
```

```
        A  B
A0.B00 0 00
A1.B00 1 00
A0.B10 0 10
A1.B10 1 10
A0.B01 0 01
A1.B01 1 01
A0.B11 0 11
A1.B11 1 11
```

A model with n_j nodal types at node j has $\prod_j n_j$ causal types. Thus the set of causal types can be large. In the model $(X \rightarrow M \rightarrow Y \leftarrow X)$ there are $2 \times 4 \times 16 = 128$ causal types.

Knowledge of a causal type tells you what values a unit would take, on all nodes, absent an intervention. For example for a model $X \rightarrow M \rightarrow Y$ a type $\theta = (\theta^X_0, \theta^M_{01}, \theta^Y_{10})$ would imply data $(X = 0, M = 0, Y = 1)$. (The converse of this, of course, is the key to updating: observation of data $(X = 0, M = 0, Y = 1)$ result in more weight placed on θ^X_0, θ^M_{01}, and θ^Y_{10}).)

3.2.3 Parameters dataframe

When a model is created, `CausalQueries` attaches a "parameters dataframe" which keeps track of model parameters, which belong together in a family, and how they relate to causal types. This becomes especially important

for more complex models with confounding that might involve more complicated mappings between parameters and nodal types. In the case with no confounding the nodal types *are* the parameters; in cases with confounding you generally have more parameters than nodal types.

For instance:

```
make_model("X->Y")$parameters_df %>%
  kable()
```

param_names	node	gen	param_set	nodal_type	given	param_value	priors
X.0	X	1	X	0		0.50	1
X.1	X	1	X	1		0.50	1
Y.00	Y	2	Y	00		0.25	1
Y.10	Y	2	Y	10		0.25	1
Y.01	Y	2	Y	01		0.25	1
Y.11	Y	2	Y	11		0.25	1

Each row in the dataframe corresponds to a single parameter.

The columns of the parameters data frame are understood as follows:

- `param_names` gives the name of the parameter, in shorthand. For instance the parameter $\lambda^X_0 = \Pr(\theta^X = \theta^X_0)$ has `par_name X.0`. See section 12 for a summary of notation.
- `param_value` gives the (possibly default) parameter values. These are probabilities.

- `param_set` indicates which parameters group together to form a simplex. The parameters in a set have parameter values that sum to 1. In this example $\lambda^X_0 + \lambda^X_1 = 1$.
- `node` indicates the node associated with the parameter. For parameter `\lambda^X_0` this is X.
- `nodal_type` indicates the nodal types associated with the parameter.
- `gen` indicates the place in the partial causal ordering (generation) of the node associated with the parameter
- `priors` gives (possibly default) Dirichlet priors arguments for parameters in a set. Values of 1 (.5) for all parameters in a set implies uniform (Jeffrey's) priors over this set.

Below we will see examples where the parameter dataframe helps keep track

of parameters that are created when confounding is added to a model.

3.2.4 Parameter matrix

The parameters dataframe keeps track of parameter values and priors for parameters but it does not provide a mapping between parameters and the probability of causal types.

The parameter matrix (P matrix) is added to the model to provide this mapping. The P matrix has a row for each parameter and a column for each causal type. For instance:

```
make_model("X->Y") %>% get_parameter_matrix %>%
  kable
```

	X0.Y00	X1.Y00	X0.Y10	X1.Y10	X0.Y01	X1.Y01	X0.Y11	X1.Y11
X.0	1	0	1	0	1	0	1	0
X.1	0	1	0	1	0	1	0	1
Y.00	1	1	0	0	0	0	0	0
Y.10	0	0	1	1	0	0	0	0
Y.01	0	0	0	0	1	1	0	0
Y.11	0	0	0	0	0	0	1	1

The probability of a causal type is given by the product of the parameters values for parameters whose row in the P matrix contains a 1.

Below we will see examples where the P matrix helps keep track of parameters that are created when confounding is added to a model.

3.3 Setting restrictions

When a model is defined, the complete set of possible causal relations are identified. This set can be very large.

Sometimes for theoretical or practical reasons it is useful to constrain the set of types. In `CausalQueries` this is done at the level of nodal types, with restrictions on causal types following from restrictions on nodal types.

For instance to impose an assumption that Y is not decreasing in X we generate a restricted model as follows:

```
model <- make_model("X->Y") %>% set_restrictions("Y[X=1] < Y[X=0]")
```

or:

```
model <- make_model("X->Y") %>% set_restrictions(decreasing("X", "Y"))
```

Viewing the resulting parameter matrix we see that both the set of parameters and the set of causal types are now restricted:

```
get_parameter_matrix(model)
```

```
Rows are parameters, grouped in parameter sets

Columns are causal types

Cell entries indicate whether a parameter probability is used
in the calculation of causal type probability

     X0.Y00 X1.Y00 X0.Y01 X1.Y01 X0.Y11 X1.Y11
X.0       1      0      1      0      1      0
X.1       0      1      0      1      0      1
Y.00      1      1      0      0      0      0
Y.01      0      0      1      1      0      0
Y.11      0      0      0      0      1      1


 param_set  (P)
```

Here and in general, setting restrictions typically involves using causal syntax; see Section 12.2 for a guide the syntax used by `CausalQueries`.

Note:

- Restrictions have to operate on nodal types: restrictions on *levels* of endogenous nodes aren't allowed. This, for example, will fail: `make_model("X->Y") %>% set_restrictions(statement = "(Y == 1)")`. The reason is that it requests a correlated restriction on nodal types for `X` and `Y` which involves undeclared confounding.

- Restrictions implicitly assume fixed values for *all* parents of a node. For instance: `make_model("A -> B <- C") %>% set_restrictions("(B[C=1]==1)")` is interpreted as shorthand for the restriction `"B[C = 1, A = 0]==1 | B[C = 1, A = 1]==1"`.
- To place restrictions on multiple nodes at the same time, provide these as a vector of restrictions. This is not permitted: `set_restrictions("Y[X=1]==1 & X==1")`, since it requests correlated restrictions. This however is allowed: `set_restrictions(c("Y[X=1]==1", "X==1"))`.

- Use the `keep` argument to indicate whether nodal types should be dropped (default) or retained.
- Restrictions can be set using nodal type labels. `make_model("S -> C -> Y <- R <- X; X -> C -> R") %>% set_restrictions(labels = list(C = "C1000", R = "R0001", Y = "Y0001"), keep = TRUE)`
- Wild cards can be used in nodal type labels: `make_model("X->Y") %>% set_restrictions(labels = list(Y = "Y?0"))`

3.4 Allowing confounding

(Unobserved) confounding between two nodes arises when the nodal types for the nodes are not independently distributed.

In the $X \rightarrow Y$ graph, for instance, there are 2 nodal types for X and 4 for Y. There are thus 8 joint nodal types (or causal types):

		θ^X		
		0	1	Sum
θ^Y	00	$\Pr(\theta^X_0, \theta^Y_{00})$	$\Pr(\theta^X_1, \theta^Y_{00})$	$\Pr(\theta^Y_{00})$
	10	$\Pr(\theta^X_0, \theta^Y_{10})$	$\Pr(\theta^X_1, \theta^Y_{10})$	$\Pr(\theta^Y_{10})$
	01	$\Pr(\theta^X_0, \theta^Y_{01})$	$\Pr(\theta^X_1, \theta^Y_{01})$	$\Pr(\theta^Y_{01})$
	11	$\Pr(\theta^X_0, \theta^Y_{11})$	$\Pr(\theta^X_1, \theta^Y_{11})$	$\Pr(\theta^Y_{11})$
	Sum	$\Pr(\theta^X_0)$	$\Pr(\theta^X_1)$	1

This table has 8 interior elements and so an unconstrained joint distribution

would have 7 degrees of freedom. A no confounding assumption means that $\Pr(\theta^X|\theta^Y) = \Pr(\theta^X)$, or $\Pr(\theta^X, \theta^Y) = \Pr(\theta^X)\Pr(\theta^Y)$. In this case we just put a distribution on the marginals and there would be 3 degrees of freedom for Y and 1 for X, totaling 4 rather than 7.

`set_confounds` lets you relax this assumption by increasing the number of parameters characterizing the joint distribution. Using the fact that $\Pr(A, B) = \Pr(A)\Pr(B|A)$ new parameters are introduced to capture $\Pr(B|A = a)$ rather than simply $\Pr(B)$.

The simplest way to allow for confounding is by adding a bidirected edge, such as via: `set_confound(model, list("X <-> Y"))`. In this case the descendant node has a distribution conditional on the value of the ancestor node. To wit:

```
confounded <- make_model("X->Y") %>%
  set_confound("X <-> Y")

confounded$parameters_df %>% kable
```

param_names	node	gen	param_set	nodal_type	given	param_value	priors
X.0	X	1	X	0		0.50	1
X.1	X	1	X	1		0.50	1
Y.00_X.0	Y	2	Y.X.0	00	X.0	0.25	1
Y.10_X.0	Y	2	Y.X.0	10	X.0	0.25	1
Y.01_X.0	Y	2	Y.X.0	01	X.0	0.25	1
Y.11_X.0	Y	2	Y.X.0	11	X.0	0.25	1
Y.00_X.1	Y	2	Y.X.1	00	X.1	0.25	1
Y.10_X.1	Y	2	Y.X.1	10	X.1	0.25	1
Y.01_X.1	Y	2	Y.X.1	01	X.1	0.25	1
Y.11_X.1	Y	2	Y.X.1	11	X.1	0.25	1

We see here that there are now two parameter families for parameters associated with the node Y. Each family captures the conditional distribution of Y's nodal types, given X. For instance the parameter `Y01_X.1` can be interpreted as $\Pr(\theta^Y = \theta^Y_{01}|X = 1)$.

To see exactly how the parameters map to causal types we can view the parameter matrix:

```
get_parameter_matrix(confounded) %>% kable
```

	X0.Y00	X1.Y00	X0.Y10	X1.Y10	X0.Y01	X1.Y01	X0.Y11	X1.Y11
X.0	1	0	1	0	1	0	1	0
X.1	0	1	0	1	0	1	0	1
Y.00_X.0	1	0	0	0	0	0	0	0
Y.10_X.0	0	0	1	0	0	0	0	0
Y.01_X.0	0	0	0	0	1	0	0	0
Y.11_X.0	0	0	0	0	0	0	1	0
Y.00_X.1	0	1	0	0	0	0	0	0
Y.10_X.1	0	0	0	1	0	0	0	0
Y.01_X.1	0	0	0	0	0	1	0	0
Y.11_X.1	0	0	0	0	0	0	0	1

Importantly, the P matrix works as before, despite confounding. We can assess the probability of causal types by multiplying the probabilities of the constituent parameters.

Note:

- Ordering of conditioning can also be controlled however via `set_confound(model, list(X = "Y"))` in which case X is given a distribution conditional on nodal types of Y.
- More specific confounding statements are also possible using causal syntax.
  - A statement of the form `list(X = "Y[X=1]==1")` can be interpreted as: "Allow X to have a distinct conditional distribution when Y has types that involve $Y(do(X = 1)) = 1$." In this case, nodal types for Y would continue to have 3 degrees of freedom. But there would be parameters assigning the probability of X when $\theta^Y = \theta^Y_{01}$ or $\theta^Y = \theta^Y_{11}$ and other parameters for residual cases. Thus 6 degrees of freedom in all.
  - Similarly a statement of the form `list(Y = "X==1")` can be interpreted as: "Allow Y to have a distinct conditional distribution when X=1." In this case there would be two distributions over nodal types for Y, producing 2*3 = 6 degrees of freedom. Nodal types for X would continue to have 1 degree of freedom. Thus 7 degrees of freedom in all, corresponding to a fully unconstrained

joint distribution.

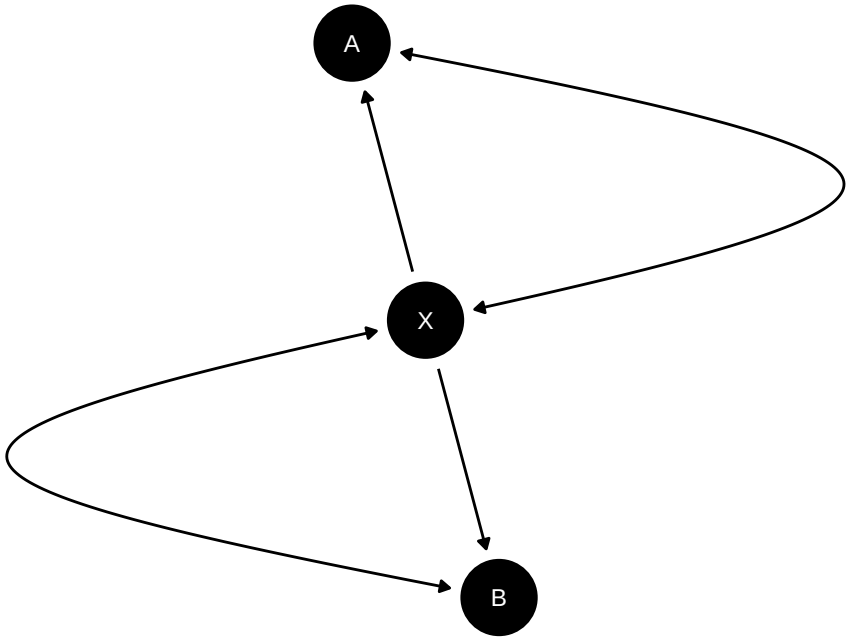
- Unlike nodal restrictions, a confounding relation can involve multiple nodal types simultaneously. For instance `make_model("X -> M -> Y") %>% set_confound(list(X = "Y[X=1] > Y[X=0]"))` allows for a parameter that renders X more or less likely depending on whether X has a positive effect on Y whether it runs through a positive or a negative effect on M.

- The parameters needed to capture confounding relations depend on the direction of causal arrows. For example compare:

  - `make_model("A -> W <- B ; A <-> W; B <-> W")$parameters_df %>% dim` In this case we can decompose shocks on A, B, W via: $\Pr(\theta^A, \theta^B, \theta^W) = \Pr(\theta^W|\theta^A, \theta^A)\Pr(\theta^A)\Pr(\theta^B)$, and we have 68 parameters.
  - `make_model("A <- W -> B ; A <-> W; B <-> W")$parameters_df %>% dim` In this case we have $\Pr(\theta^A, \theta^B, \theta^W) = \Pr(\theta^A|\theta^W)\Pr(\theta^B|\theta^W)\Pr(\theta^W)$ and just has just 18 parameters.

When confounding is added to a model, a dataframe, `confounds_df` is created and added to the model, recording which variables involve confounding. This is then used for plotting:

```
make_model("A <- X -> B; A <-> X; B <-> X") %>% plot()
```



Sometimes the `confounds_df` can highlight nonobvious confounding relations:

```
model <- make_model("X -> M -> Y") %>%
  set_confound(list(X = "Y[X=1] > Y[X=0]"))
model$confounds_df
```

```
NULL
```

In this example, the confounding statement implies confounding between X and M even though M is not included explicitly in the confound statement (the reason is that X can have a positive effect on Y by having a positive effect on M and this in turn having a positive effect on Y *or* by having a negative effect on M and this in turn having a negative effect on Y).

3.5 Setting Priors

Priors on model parameters can be added to the parameters dataframe. The priors are interpreted as alpha arguments for a Dirichlet distribution. They

can be seen using `get_priors`.

```
make_model("X->Y") %>% get_priors
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
   1    1    1    1    1    1
```

Here the priors have not been specified and so they default to 1, which corresponds to uniform priors.

Alternatively you could set jeffreys priors like this:

```
make_model("X->Y") %>% set_priors(distribution = "jeffreys") %>% get_priors
```

```
no specific parameters to alter values for specified. Altering all parameters.
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
 0.5  0.5  0.5  0.5  0.5  0.5
```

3.5.1 Custom priors

Custom priors are most simply specified by being added as a vector of numbers using `set_priors`. For instance:

```
make_model("X->Y") %>% set_priors(1:6) %>% get_priors
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
   1    2    3    4    5    6
```

The priors here should be interpreted as indicating:

- $\alpha_X = (1, 2)$, which implies a distribution over $(\lambda^X_0, \lambda^X_1)$ centered on $(1/3, 2/3)$.
- $\alpha_Y = (3, 4, 5, 6)$, which implies a distribution over $(\lambda^Y_{00}, \lambda^Y_{10}, \lambda^Y_{01}\lambda^Y_{11})$ centered on $(3/18, 4/18, 5/18, 6/18)$.

For larger models it can be hard to provide priors as a vector of numbers and so `set_priors` can allow for more targeted modifications of the parameter vector. For instance:

```
make_model("X->Y") %>%
  set_priors(statement = "Y[X=1] > Y[X=0]", alphas = 3) %>%
  get_priors
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
   1    1    1    1    3    1
```

See `?set_priors` and `?make_priors` for many examples.

3.5.2 Prior warnings

"Flat" priors over parameters in a parameter family put equal weight on each nodal type, but this in turn can translate into strong assumptions on causal quantities of interest.

For instance in an $X \rightarrow Y$ model model in which negative effects are ruled out, the average causal effect implied by "flat" priors is 1/3. This can be seen by querying the model:

```
make_model("X -> Y") %>%
  set_restrictions(decreasing("X", "Y")) %>%
  query_model("Y[X=1] - Y[X=0]", n_draws = 10000) %>%
  kable
```

Query	Given	Using	Case.estimand	mean
Q 1	-	parameters	FALSE	0.3333

More subtly the *structure* of a model, coupled with flat priors, has substantive importance for priors on causal quantities. For instance with flat priors, priors on the probability that X has a positive effect on Y in the model $X \rightarrow Y$ is centered on 1/4. But priors on the probability that X has a positive effect on Y in the model $X \rightarrow M \rightarrow Y$ is centered on 1/8.

Again, you can use `query_model` to figure out what flat (or other) priors over parameters imply for priors over causal quantities:

```
make_model("X -> Y") %>%
  query_model("Y[X=1] > Y[X=0]", n_draws = 10000) %>%
  kable
```

Query	Given	Using	Case.estimand	mean
Q 1	-	parameters	FALSE	0.25

```
make_model("X -> M -> Y") %>%
  query_model("Y[X=1] > Y[X=0]", n_draws = 10000) %>%
  kable
```

Query	Given	Using	Case.estimand	mean
Q 1	-	parameters	FALSE	0.125

Caution regarding priors is particularly important when models are not identified, as is the case for many of the models considered here. In such cases, for some quantities, the marginal posterior distribution can be the same as the marginal prior distribution (Poirier, 1998).

The key point here is to make sure you do not fall into a trap of thinking that "uninformative" priors make no commitments regarding the values of causal quantities of interest. They do, and the implications of flat priors for causal quantities can depend on the structure of the model. Moreover for some inferences from causal models the priors can matter a lot even if you have a lot of data. In such cases it can be helpful to know what priors on parameters imply for priors on causal quantities of interest (by using `query_model`) and to assess how much conclusions depend on priors (by comparing results across models that vary in their priors).

3.6 Setting Parameters

By default, models have a vector of parameter values included in the `parameters_df` dataframe. These are useful for generating data, or for situations, such as process tracing, when one wants to make inferences about causal types (θ), given case level data, under the assumption that the model is known.

Consider the causal model below. It has two parameter sets, X and Y, with six nodal types, two corresponding to X and four corresponding to Y. The key feature of the parameters is that they must sum to 1 within each parameter set.

```
make_model("X->Y") %>% get_parameters()
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
0.50 0.50 0.25 0.25 0.25 0.25
```

Setting parameters can be done using a similar syntax as `set_priors`. The main difference is that when a given value is altered the entire set must still always sum to 1. The example below illustrates a change in the value of the parameter Y in the case it is increasing in X. Here nodal type `Y.Y01` is set

to be .5, while the other nodal types of this parameter set were renormalized so that the parameters in the set still sum to one.

```
make_model("X->Y") %>%
  set_parameters(statement = "Y[X=1] > Y[X=0]", parameters = .5) %>%
  get_parameters
```

```
   X.0    X.1   Y.00   Y.10   Y.01   Y.11
0.5000 0.5000 0.1667 0.1667 0.5000 0.1667
```

Alternatively, instead of setting a particular new value for a parameter you can set a value that then gets renormalized along with all other values. In the example below, if we begin with vector (.25, .25, .25, .25) and request a value of `Y.Y01` of .5, *without* requesting a renormalization of other variables, then we get a vector (.2, .2, .4, .2) which is itself a renormalization of (.25, .25, .5, .25).

```
make_model("X->Y") %>%
  set_parameters(statement = "Y[X=1] > Y[X=0]", parameters = .5, normalize=FALSE) %>%
  get_parameters
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
 0.5  0.5  0.2  0.2  0.4  0.2
```

This normalization behavior can mean that you can control parameters better if they are set in a single step rather than in multiple steps, compare:

```
make_model('X -> Y') %>%
  make_parameters(statement = c('Y[X=1]<Y[X=0] | Y[X=1]>Y[X=0]'), parameters = c(1/2,
```

```
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
0.50 0.50 0.25 0.50 0.00 0.25
```

Here the .6 in the second vector arises because a two step process is requested (`statement` is of length 2) and the vector first becomes (1/6, 1/2, 1/6, 1/6) and then becomes (.2, .6, 0, .2) which is a renormalization of (1/6, 1/2, 0, 1/6).

If in doubt, check parameter values.

See `? set_parameters` for some handy ways to set parameters either manually (`define`) or using `prior_mean`, `prior_draw`, `posterior_mean`,

`posterior_draw`.

Chapter 4

Updating models with `stan`

When we generate a model we often impose a lot of assumptions on nature of causal relations. This includes "structure" regarding what relates to what but also the nature of those relations—how strong the effect of a given variable is and how it interacts with others, for example. The latter features are captured by parameters whose values, fortunately, can be data based.

The approach used by the `CausalQueries` package to updating parameter values given observed data uses `stan` and involves the following elements:

- Dirichlet priors over parameters, λ (which, in cases without confounding, correspond to nodal types)
- A mapping from parameters to event probabilities, w
- A likelihood function that assumes events are distributed according to a multinomial distribution given event probabilities.

We provide further details below.

4.1 Data for `stan`

We use a generic `stan` model that works for all binary causal models. Rather than writing a new `stan` model for each causal model we send `stan` details of each particular causal model as data inputs.

In particular we provide a set of matrices that `stan` tailor itself to particular models: the parameter matrix (P) tells `stan` how many parameters there

are, and how they map into causal types; an ambiguity matrix A tells `stan` how causal types map into data types; and an event matrix E relates data types into patterns of observed data (in cases where there are incomplete observations).

The internal function `prep_stan_data` prepares data for `stan`. You generally don't need to use this manually, but we show here a sample of what it produces as input for `stan`.

We provide `prep_stan_data` with data in compact form (listing "data events").

```
model <- make_model("X->Y")

data  <- data.frame(X = c(0, 1, 1, NA), Y = c(0, 1, 0, 1))

compact_data <-  collapse_data(data, model)

kable(compact_data)
```

event	strategy	count
X0Y0	XY	1
X1Y0	XY	1
X0Y1	XY	0
X1Y1	XY	1
Y0	Y	0
Y1	Y	1

Note that NAs are interpreted as data not having been sought. So in this case the interpretation is that there are two data strategies: data on Y and X was sought in three cases; data on Y only was sought in just one case.

`prep_stan_data` then returns a list of objects that `stan` expects to receive. These include indicators to figure out where a parameter set starts (`l_starts`, `l_ends`) and ends and where a data strategy starts and ends (`strategy_starts`, `strategy_ends`), as well as the matrices described above.

```
CausalQueries:::prep_stan_data(model, compact_data)
```

```
$parmap
```

```
     X0Y0 X1Y0 X0Y1 X1Y1
X.0     1    0    1    0
X.1     0    1    0    1
Y.00    1    1    0    0
Y.10    0    1    1    0
Y.01    1    0    0    1
Y.11    0    0    1    1
attr(,"map")
     X0Y0 X1Y0 X0Y1 X1Y1
X0Y0    1    0    0    0
X1Y0    0    1    0    0
X0Y1    0    0    1    0
X1Y1    0    0    0    1

$map
     X0Y0 X1Y0 X0Y1 X1Y1
X0Y0    1    0    0    0
X1Y0    0    1    0    0
X0Y1    0    0    1    0
X1Y1    0    0    0    1

$n_paths
[1] 4

$n_params
[1] 6

$n_param_sets
[1] 2

$n_param_each
X Y
2 4

$l_starts
X Y
1 3
```

```
$l_ends
X Y
2 6

$node_starts
X Y
1 3

$node_ends
X Y
2 6

$n_nodes
[1] 2

$lambdas_prior
 X.0  X.1 Y.00 Y.10 Y.01 Y.11
   1    1    1    1    1    1

$n_data
[1] 4

$n_events
[1] 6

$n_strategies
[1] 2

$strategy_starts
[1] 1 5

$strategy_ends
[1] 4 6

$keep_transformed
[1] 1

$E
```

```
     X0Y0 X1Y0 X0Y1 X1Y1
X0Y0    1    0    0    0
X1Y0    0    1    0    0
X0Y1    0    0    1    0
X1Y1    0    0    0    1
Y0      1    1    0    0
Y1      0    0    1    1

$Y
[1] 1 1 0 1 0 1

$P

Rows are parameters, grouped in parameter sets

Columns are causal types

Cell entries indicate whether a parameter probability is used
in the calculation of causal type probability

     X0.Y00 X1.Y00 X0.Y10 X1.Y10 X0.Y01 X1.Y01 X0.Y11
X.0       1      0      1      0      1      0      1
X.1       0      1      0      1      0      1      0
Y.00      1      1      0      0      0      0      0
Y.10      0      0      1      1      0      0      0
Y.01      0      0      0      0      1      1      0
Y.11      0      0      0      0      0      0      1
     X1.Y11
X.0       0
X.1       1
Y.00      0
Y.10      0
Y.01      0
Y.11      1


 param_set  (P)
```

```
$n_types
[1] 8
```

4.2 stan code

Below we show the stan code. This starts off with a block saying what input data is to be expected. Then there is a characterization of parameters and the transformed parameters. Then the likelihoods and priors are provided. stan takes it from there and generates a posterior distribution.

```
functions{
  row_vector col_sums(matrix X) {
    row_vector[cols(X)] s ;
    s = rep_row_vector(1, rows(X)) * X ;
    return s ;
  }
}
data {
int<lower=1> n_params;
int<lower=1> n_paths;
int<lower=1> n_types;
int<lower=1> n_param_sets;
int<lower=1> n_nodes;
int<lower=1> n_param_each[n_param_sets];
int<lower=1> n_data;
int<lower=1> n_events;
int<lower=1> n_strategies;
int<lower=0, upper=1> keep_transformed;
vector<lower=0>[n_params] lambdas_prior;
int<lower=1> l_starts[n_param_sets];
int<lower=1> l_ends[n_param_sets];
int<lower=1> node_starts[n_nodes];
int<lower=1> node_ends[n_nodes];
int<lower=1> strategy_starts[n_strategies];
int<lower=1> strategy_ends[n_strategies];
matrix[n_params, n_types] P;
matrix[n_params, n_paths] parmap;
```

```
matrix[n_paths, n_data] map;
matrix<lower=0,upper=1>[n_events,n_data] E;
int<lower=0> Y[n_events];
}
parameters {
vector<lower=0>[n_params - n_param_sets] gamma;
}
transformed parameters {
vector<lower=0>[n_params] lambdas;
vector<lower=1>[n_param_sets] sum_gammas;
matrix[n_params, n_paths] parlam;
matrix[n_nodes, n_paths] parlam2;
vector<lower=0, upper=1>[n_paths] w_0;
vector<lower=0, upper=1>[n_data] w;
vector[n_events] w_full;
// Cases in which a parameter set has only one value need special handling
// they have no gamma components and sum_gamma needs to be made manually
for (i in 1:n_param_sets) {
if (l_starts[i] >= l_ends[i]) {
  sum_gammas[i] = 1;
  // syntax here to return unity as a vector
  lambdas[l_starts[i]] = lambdas_prior[1]/lambdas_prior[1];
  }
else if (l_starts[i] < l_ends[i]) {
  sum_gammas[i] =
  1 + sum(gamma[(l_starts[i] - (i-1)):(l_ends[i] - i)]);
  lambdas[l_starts[i]:l_ends[i]] =
  append_row(1, gamma[(l_starts[i] - (i-1)):(l_ends[i] - i)]) / sum_gammas[i];
  }
  }
// Mapping from parameters to data types
parlam  = rep_matrix(lambdas, n_paths) .* parmap; // (usual case): [n_par * n_data] *
// Sum probability over nodes on each path
for (i in 1:n_nodes) {
 parlam2[i,] = col_sums(parlam[(node_starts[i]):(node_ends[i]),]);
 }
// then take product  to get probability of data type on path
for (i in 1:n_paths) {
```

```
  w_0[i] = prod(parlam2[,i]);
 }
 // last (if confounding): map to n_data columns instead of n_paths
 w = map'*w_0;
 w = w / sum(w);
 w_full = E * w;
}
model {
// Dirichlet distributions (earlier versions used gamma)
for (i in 1:n_param_sets) {
  target += dirichlet_lpdf(lambdas[l_starts[i]:l_ends[i]]  | lambdas_prior[l_s
  target += -n_param_each[i] * log(sum_gammas[i]);
 }
// Multinomials
for (i in 1:n_strategies) {
  target += multinomial_lpmf(
  Y[strategy_starts[i]:strategy_ends[i]] | w_full[strategy_starts[i]:strategy_
 }
}
// Option to export distribution of causal types
// Note if clause used here to effectively turn off this block if not required
generated quantities{
vector[n_types] prob_of_types;
if (keep_transformed == 1){
for (i in 1:n_types) {
   prob_of_types[i] = prod(P[, i].*lambdas + 1 - P[,i]);
}}
 if (keep_transformed == 0){
    prob_of_types = rep_vector(1, n_types);
 }
}
```

The `stan` model works as follows (technical!):

- We are interested in "sets" of parameters. For example in the $X \to Y$ model we have two parameter sets (`param_sets`). The first is $\lambda^X \in \{\lambda^X_0, \lambda^X_1\}$ whose elements give the probability that X is 0 or 1. These two probabilities sum to one. The second parameter set is $\lambda^Y \in \{\lambda^Y_{00}, \lambda^Y_{10}, \lambda^Y_{01}\lambda^Y_{11}\}$. These are also probabilities and their values sum to

one. Note in all that we have 6 parameters but just $1 + 3 = 4$ degrees of freedom.

- We would like to express priors over these parameters using multiple Dirichlet distributions (two in this case). In practice because we are dealing with multiple simplices of varying length, it is easier to express priors over gamma distributions with a unit scale parameter and shape parameter corresponding to the Dirichlet priors, α. We make use of the fact that $\lambda_0^X \sim Gamma(\alpha_0^X, 1)$ and $\lambda_1^X \sim Gamma(\alpha_1^X, 1)$ then $\frac{1}{\lambda_0^X + \lambda_1^X}(\lambda_0^X, \lambda_1^X) \sim Dirichlet(\alpha_0^X, \alpha_1^X)$. For a discussion of implementation of this approach in `stan` see https://discourse.mc-stan.org/t/ragged-array-of-simplexes/1382.

- For any candidate parameter vector λ we calculate the probability of *causal* types (`prob_of_types`) by taking, for each type i, the product of the probabilities of all parameters (λ_j) that appear in column i of the parameter matrix P. Thus the probability of a (X_0, Y_{00}) case is just $\lambda_0^X \times \lambda_{00}^Y$. The implementations in `stan` uses `prob_of_types_[i]` $= \prod_j \left(P_{j,i}\lambda_j + (1 - P_{j,i})\right)$: this multiplies the probability of all parameters involved in the causal type (and substitutes 1s for parameters that are not). (`P` and `not_P` (1-P) are provided as data to `stan`).

- The probability of data types, `w`, is given by summing up the probabilities of all causal types that produce a given data type. For example, the probability of a $X = 0, Y = 0$ case, w_{00} is $\lambda_0^X \times \lambda_{00}^Y + \lambda_0^X \times \lambda_{01}^Y$. The ambiguity matrix A is provided to `stan` to indicate which probabilities need to be summed.

- In the case of incomplete data we first identify the set of "data strategies", where a collection of a data strategy might be of the form "gather data on X and M, but not Y, for n_1 cases and gather data on X and Y, but not M, for n_2 cases. The probability of an observed event, within a data strategy, is given by summing the probabilities of the types that could give rise to the incomplete data. For example X is observed, but Y is not, then the probability of $X = 0, Y = \text{NA}$ is $w_{00} + w_{01}$. The matrix E is passed to `stan` to figure out which event probabilities need to be combined for events with missing data.

- The probability of a dataset is then given by a multinomial distribution with these event probabilities (or, in the case of incomplete data, the

product of multinomials, one for each data strategy). Justification for this approach relies on the likelihood principle and is discussed in Chapter 6.

4.3 Implementation

To update a CausalQueries model with data use:

```
update_model(model, data)
```

where the data argument is a dataset containing some or all of the nodes in the model.

Other `stan` arguments can be passed to `update_data`, in particular:

- `iter` sets the number of iterations and ultimately the number of draws in the posterior
- `chains` sets the number of chains; doing multiple chains in parallel speeds things up
- lots of other options via `?rstan::stan`

If you have multiple cores you can do parallel processing by including this line before running `CausalQueries`:

```
options(mc.cores = parallel::detectCores())
```

The `stan` output from a simple model looks like this:

```
Inference for Stan model: simplexes.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

                  mean se_mean     sd   2.5%    25%
gamma[1]          3.13    0.13   5.65   0.24   0.85
gamma[2]         11.71    5.15 181.05   0.02   0.27
gamma[3]         14.96    5.85 213.49   0.05   0.50
gamma[4]          7.37    2.86 151.51   0.04   0.36
lambdas[1]        0.40    0.00   0.20   0.06   0.24
lambdas[2]        0.60    0.00   0.20   0.20   0.46
lambdas[3]        0.25    0.00   0.17   0.01   0.12
```

lambdas[4]	0.21	0.00	0.16	0.01	0.08
lambdas[5]	0.31	0.00	0.19	0.02	0.16
lambdas[6]	0.23	0.00	0.16	0.01	0.10
sum_gammas[1]	4.13	0.13	5.65	1.24	1.85
sum_gammas[2]	35.04	12.63	441.32	1.61	2.74
parlam[1,1]	0.40	0.00	0.20	0.06	0.24
parlam[1,2]	0.00	NaN	0.00	0.00	0.00
parlam[1,3]	0.40	0.00	0.20	0.06	0.24
parlam[1,4]	0.00	NaN	0.00	0.00	0.00
parlam[2,1]	0.00	NaN	0.00	0.00	0.00
parlam[2,2]	0.60	0.00	0.20	0.20	0.46
parlam[2,3]	0.00	NaN	0.00	0.00	0.00
parlam[2,4]	0.60	0.00	0.20	0.20	0.46
parlam[3,1]	0.25	0.00	0.17	0.01	0.12
parlam[3,2]	0.25	0.00	0.17	0.01	0.12
parlam[3,3]	0.00	NaN	0.00	0.00	0.00
parlam[3,4]	0.00	NaN	0.00	0.00	0.00
parlam[4,1]	0.00	NaN	0.00	0.00	0.00
parlam[4,2]	0.21	0.00	0.16	0.01	0.08
parlam[4,3]	0.21	0.00	0.16	0.01	0.08
parlam[4,4]	0.00	NaN	0.00	0.00	0.00
parlam[5,1]	0.31	0.00	0.19	0.02	0.16
parlam[5,2]	0.00	NaN	0.00	0.00	0.00
parlam[5,3]	0.00	NaN	0.00	0.00	0.00
parlam[5,4]	0.31	0.00	0.19	0.02	0.16
parlam[6,1]	0.00	NaN	0.00	0.00	0.00
parlam[6,2]	0.00	NaN	0.00	0.00	0.00
parlam[6,3]	0.23	0.00	0.16	0.01	0.10
parlam[6,4]	0.23	0.00	0.16	0.01	0.10
parlam2[1,1]	0.40	0.00	0.20	0.06	0.24
parlam2[1,2]	0.60	0.00	0.20	0.20	0.46
parlam2[1,3]	0.40	0.00	0.20	0.06	0.24
parlam2[1,4]	0.60	0.00	0.20	0.20	0.46
parlam2[2,1]	0.56	0.00	0.20	0.18	0.42
parlam2[2,2]	0.46	0.00	0.18	0.13	0.33
parlam2[2,3]	0.44	0.00	0.20	0.10	0.29
parlam2[2,4]	0.54	0.00	0.18	0.20	0.41
w_0[1]	0.22	0.00	0.14	0.03	0.11

w_0[2]	0.27	0.00	0.14	0.06	0.16
w_0[3]	0.18	0.00	0.13	0.01	0.08
w_0[4]	0.33	0.00	0.16	0.06	0.21
w[1]	0.22	0.00	0.14	0.03	0.11
w[2]	0.27	0.00	0.14	0.06	0.16
w[3]	0.18	0.00	0.13	0.01	0.08
w[4]	0.33	0.00	0.16	0.06	0.21
w_full[1]	0.22	0.00	0.14	0.03	0.11
w_full[2]	0.27	0.00	0.14	0.06	0.16
w_full[3]	0.18	0.00	0.13	0.01	0.08
w_full[4]	0.33	0.00	0.16	0.06	0.21
w_full[5]	0.49	0.00	0.14	0.22	0.40
w_full[6]	0.51	0.00	0.14	0.24	0.41
prob_of_types[1]	0.10	0.00	0.09	0.00	0.03
prob_of_types[2]	0.15	0.00	0.12	0.01	0.06
prob_of_types[3]	0.08	0.00	0.09	0.00	0.02
prob_of_types[4]	0.12	0.00	0.10	0.00	0.04
prob_of_types[5]	0.12	0.00	0.10	0.00	0.04
prob_of_types[6]	0.19	0.00	0.14	0.01	0.08
prob_of_types[7]	0.09	0.00	0.09	0.00	0.03
prob_of_types[8]	0.14	0.00	0.11	0.01	0.05
lp__	-10.33	0.04	1.62	-14.43	-11.14

	50%	75%	97.5%	n_eff	Rhat
gamma[1]	1.58	3.10	16.77	1879	1
gamma[2]	0.75	2.13	24.62	1237	1
gamma[3]	1.22	3.29	36.89	1331	1
gamma[4]	0.88	2.14	22.14	2813	1
lambdas[1]	0.39	0.54	0.80	2417	1
lambdas[2]	0.61	0.76	0.94	2417	1
lambdas[3]	0.23	0.36	0.62	1893	1
lambdas[4]	0.17	0.30	0.58	3827	1
lambdas[5]	0.29	0.44	0.72	3565	1
lambdas[6]	0.20	0.34	0.61	3555	1
sum_gammas[1]	2.58	4.10	17.77	1879	1
sum_gammas[2]	4.38	8.55	85.77	1220	1
parlam[1,1]	0.39	0.54	0.80	2417	1
parlam[1,2]	0.00	0.00	0.00	NaN	NaN
parlam[1,3]	0.39	0.54	0.80	2417	1

parlam[1,4]	0.00	0.00	0.00	NaN	NaN
parlam[2,1]	0.00	0.00	0.00	NaN	NaN
parlam[2,2]	0.61	0.76	0.94	2417	1
parlam[2,3]	0.00	0.00	0.00	NaN	NaN
parlam[2,4]	0.61	0.76	0.94	2417	1
parlam[3,1]	0.23	0.36	0.62	1893	1
parlam[3,2]	0.23	0.36	0.62	1893	1
parlam[3,3]	0.00	0.00	0.00	NaN	NaN
parlam[3,4]	0.00	0.00	0.00	NaN	NaN
parlam[4,1]	0.00	0.00	0.00	NaN	NaN
parlam[4,2]	0.17	0.30	0.58	3827	1
parlam[4,3]	0.17	0.30	0.58	3827	1
parlam[4,4]	0.00	0.00	0.00	NaN	NaN
parlam[5,1]	0.29	0.44	0.72	3565	1
parlam[5,2]	0.00	0.00	0.00	NaN	NaN
parlam[5,3]	0.00	0.00	0.00	NaN	NaN
parlam[5,4]	0.29	0.44	0.72	3565	1
parlam[6,1]	0.00	0.00	0.00	NaN	NaN
parlam[6,2]	0.00	0.00	0.00	NaN	NaN
parlam[6,3]	0.20	0.34	0.61	3555	1
parlam[6,4]	0.20	0.34	0.61	3555	1
parlam2[1,1]	0.39	0.54	0.80	2417	1
parlam2[1,2]	0.61	0.76	0.94	2417	1
parlam2[1,3]	0.39	0.54	0.80	2417	1
parlam2[1,4]	0.61	0.76	0.94	2417	1
parlam2[2,1]	0.58	0.71	0.90	2800	1
parlam2[2,2]	0.46	0.59	0.80	3231	1
parlam2[2,3]	0.42	0.58	0.82	2800	1
parlam2[2,4]	0.54	0.67	0.87	3231	1
w_0[1]	0.20	0.30	0.54	2805	1
w_0[2]	0.26	0.37	0.58	2807	1
w_0[3]	0.15	0.25	0.50	2485	1
w_0[4]	0.31	0.43	0.68	2868	1
w[1]	0.20	0.30	0.54	2805	1
w[2]	0.26	0.37	0.58	2807	1
w[3]	0.15	0.25	0.50	2485	1
w[4]	0.31	0.43	0.68	2868	1
w_full[1]	0.20	0.30	0.54	2805	1

```
w_full[2]          0.26  0.37  0.58  2807    1
w_full[3]          0.15  0.25  0.50  2485    1
w_full[4]          0.31  0.43  0.68  2868    1
w_full[5]          0.50  0.59  0.76  2433    1
w_full[6]          0.50  0.60  0.78  2433    1
prob_of_types[1]   0.08  0.14  0.34  2209    1
prob_of_types[2]   0.12  0.22  0.44  2116    1
prob_of_types[3]   0.06  0.12  0.32  3056    1
prob_of_types[4]   0.09  0.17  0.38  3350    1
prob_of_types[5]   0.09  0.17  0.38  2671    1
prob_of_types[6]   0.16  0.27  0.53  3259    1
prob_of_types[7]   0.07  0.13  0.33  2829    1
prob_of_types[8]   0.11  0.20  0.41  3744    1
lp__              -9.98 -9.14 -8.31  1351    1

Samples were drawn using NUTS(diag_e) at Thu Aug 31 04:20:06 2023.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Note the parameters include the gamma parameters plus transformed parameters, λ, which are our parameters of interest and which `CausalQueries` then interprets as possible row probabilities for the P matrix.

4.4 Extensions

4.4.1 Arbitrary parameters

Although the package provides helpers to generate mappings from parameters to causal types via nodal types, it is possible to dispense with the nodal types altogether and provide a direct mapping from parameters to causal types.

For this you need to manually provide a `P` matrix and a corresponding `parameters_df`. As an example here is a model with complete confounding and parameters that correspond to causal types directly.

```
model <- make_model("X->Y")

model$P <- diag(8)
```

```
colnames(model$P) <- rownames(model$causal_types)

model$parameters_df <- data.frame(
  param_names = paste0("x",1:8),
  param_set = 1,
  priors = 1,
  parameters = 1/8)

# Update fully confounded model on strongly correlated data
model <- make_model("X->Y")
data <- make_data(make_model("X->Y"), n = 100, parameters = c(.5, .5, .1,.1,.7,.1))

fully_confounded <- update_model(model, data)
```

4.4.2 Non binary data

In principle the `stan` model could be extended to handle non binary data. Though a limitation of the current package there is no structural reason why nodes should be constrained to be dichotomous. The set of nodal and causal types however expands even more rapidly in the case of non binary data. .

Chapter 5

Querying models

Models can be queried using the `query_distribution` and `query_model` functions. The difference between these functions is that `query_distribution` examines a single query and returns a full distribution of draws from the distribution of the estimand (prior or posterior); `query_model` takes a collection of queries and returns a dataframe with summary statistics on the queries.

The simplest queries ask about causal estimands given particular parameter values and case level data. Here is one surprising result of this form:

5.1 Case level queries

The `query_model` function takes causal queries and conditions (`given`) and specifies the parameters to be used. The result is a dataframe which can be displayed as a table.

For a case level query we can make the query *given* a particular parameter vector, as below:

```
make_model("X-> M -> Y <- X") %>%

  set_restrictions(c(decreasing("X", "M"),
                     decreasing("M", "Y"),
                     decreasing("X", "Y"))) %>%
```

Table 5.1: In a monotonic model with flat priors, knowledge that $M = 1$ *reduces* confidence that $X = 1$ caused $Y = 1$

Query	Given	Using	Case.estimand	mean
Q 1	X==1 & Y==1	parameters	FALSE	0.6154
Q 1	X==1 & Y==1 & M==1	parameters	FALSE	0.6000
Q 1	X==1 & Y==1 & M==0	parameters	FALSE	0.6667

```
query_model(queries = "Y[X=1]> Y[X=0]",
            given = c("X==1 & Y==1",
                      "X==1 & Y==1 & M==1",
                      "X==1 & Y==1 & M==0"),
            using = c("parameters")) %>%

kable(
  caption = "In a monotonic model with flat priors, knowledge
  that $M=1$ *reduces* confidence that $X=1$ caused $Y=1$")
```

This example shows how inferences change given additional data on M in a monotonic $X \rightarrow M \rightarrow Y \leftarrow X$ model. Surprisingly observing $M = 1$ *reduces* beliefs that X caused Y, the reason being that perhaps M and not X was responsible for $Y = 1$.

5.2 Posterior queries

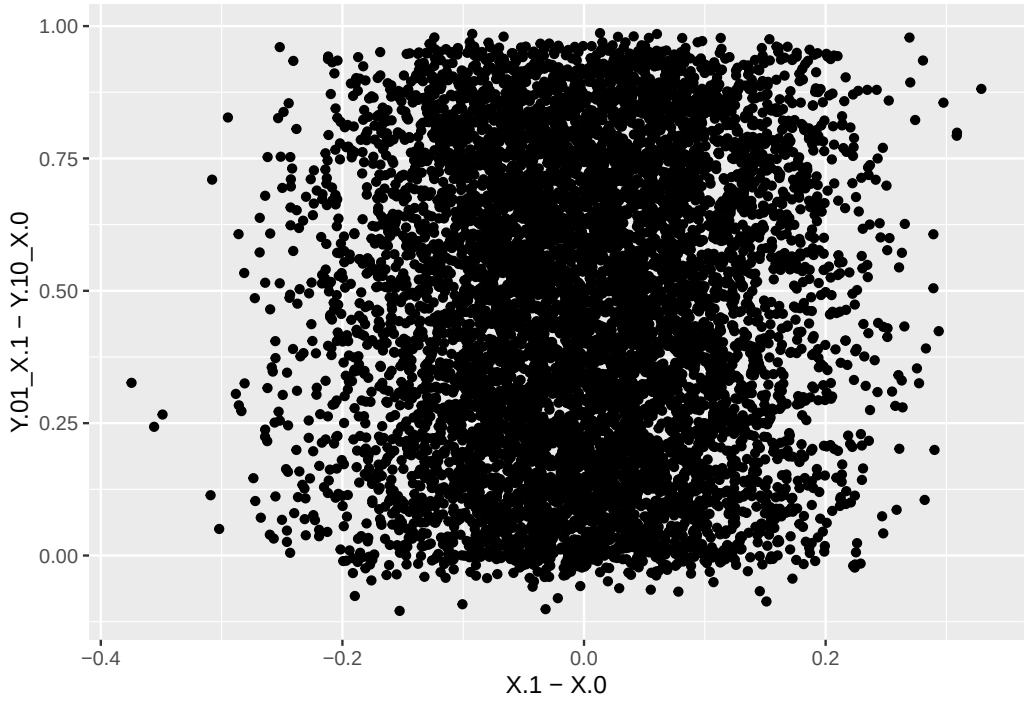
Queries can also draw directly from the posterior distribution provided by `stan`. In this next example we illustrate the joint distribution of the posterior over causal effects, drawing directly from the posterior dataframe generated by `update_model`:

```
data  <- fabricate(N = 100, X = complete_ra(N), Y = X)

model <- make_model("X -> Y; X <-> Y") %>%
  update_model(data, iter  = 4000)

model$posterior_distribution %>%
```

```
  data.frame() %>%
  ggplot(aes(X.1 - X.0, Y.01_X.1 - Y.10_X.0)) +
  geom_point()
```



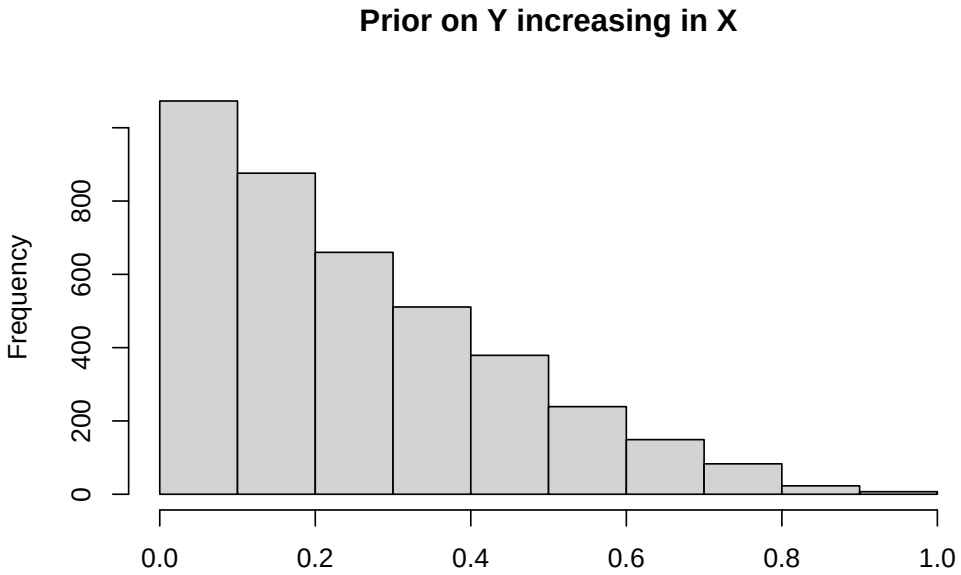
We see that beliefs about the size of the overall effect are related to beliefs that X is assigned differently when there is a positive effect.

5.3 Query distribution

`query_distribution` works similarly except that the query is over an estimand. For instance:

```
make_model("X -> Y") %>%
  query_distribution(increasing("X", "Y"), using = "priors") %>%
 hist(main = "Prior on Y increasing in X")
```

```
## Prior distribution added to model
```



.

5.4 Token and general causation

Note that in all these cases we use the same technology to make case level and population inferences. Indeed the case level query is just a conditional population query. As an illustration of this imagine we have a model of the form $X \rightarrow M \rightarrow Y$ and are interested in whether X caused Y in a case in which $M = 1$. We answer the question by asking "what would be the probability that X caused Y in a case in which $X = M = Y = 1$?" (line 3 below). This speculative answer is the same answer as we would get were we to ask the same question having updated our model with knowledge that in a particular case, indeed, $X = M = Y = 1$. See below:

```
model <- make_model("X->M->Y") %>%
  set_restrictions(c(decreasing("X", "M"), decreasing("M", "Y"))) %>%
  update_model(data = data.frame(X = 1, M = 1, Y = 1), iter = 8000)

query_model(
            model,
```

Table 5.2: Posteriors equal priors for a query that conditions on data used to form the posterior

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.hi
Q 1	X==1 & Y==1	priors	FALSE	0.2091	0.2042	0.0026	0.74
Q 1	X==1 & Y==1	posteriors	FALSE	0.2226	0.2102	0.0027	0.75
Q 1	X==1 & Y==1 & M==1	priors	FALSE	0.2485	0.2174	0.0046	0.78
Q 1	X==1 & Y==1 & M==1	posteriors	FALSE	0.2500	0.2197	0.0037	0.78

```
            query = "Y[X=1]> Y[X=0]",
            given = c("X==1 & Y==1", "X==1 & Y==1 & M==1"),
            using = c("priors", "posteriors"),
            expand_grid = TRUE)
```

We see the conditional inference is the same using the prior and the posterior distributions.

5.5 Complex queries

The Billy Suzy bottle breaking example illustrates complex queries. See Section 7.2.

Part III

Applications

Chapter 6

Basic Models

6.1 The ladder of causation in an $X \rightarrow Y$ model

We first introduce a simple X causes Y model with no confounding and use this to illustrate the "ladder of causation" (Pearl and Mackenzie, 2018).

The model is written:

```
model <- make_model("X -> Y")
```

We will assume a "true" distribution over parameters. Let's assume that the true effect of 0.5, but that this is not known to researchers. The .5 effect comes from the difference between the share of units with a positive effect (.6) and those with a negative effect (.1). (We say share but we may as well think in terms of the probability that a given unit is of one or other type.)

```
model <-
  set_parameters(model, node = "Y", parameters = c(.2, .1, .6, .1))

kable(t(get_parameters(model)))
```

X.0	X.1	Y.00	Y.10	Y.01	Y.11
0.5	0.5	0.2	0.1	0.6	0.1

We can now simulate data using the model:

```
data <- make_data(model, n = 10000)
```

With a model and data in hand we update the model thus:

```
updated <- update_model(model, data)
```

From the updated model we can draw posterior inferences over estimands of interest.

We will imagine three estimands, corresponding to Pearl's "ladder of causation."

- At the first level we are interested in the distribution of some node, perhaps given the value of another node. This question is answerable from observational data.
- At the second level we are interested in treatment effects: how changing one node changes another. This question is answerable from experimental data.
- At the third level we are interested in counterfactual statements: how would things have been different if some features of the world were different from what they are? Answering this question requires a causal model.

Here are the results:

```
results <- query_model(
 updated,
 query = list("Y | X=1" = "Y==1",
              ATE = "Y[X=1] - Y[X=0]",
              PC  = "Y[X=1] > Y[X=0]"),
 given = c("X==1", TRUE, "X==1 & Y==1"),
 using = "posteriors")
```

Query (rung)	Query	Given	Using	Case.estimand	mean	sd	con
1 Association	Y | X=1	X==1	posteriors	FALSE	0.69	0.01	
2 Intervention	ATE	-	posteriors	FALSE	0.49	0.01	
3 Imagining	PC	X==1 & Y==1	posteriors	FALSE	0.85	0.09	

We see from the posterior variance on PC that we have the greatest difficulty with the third rung. In fact the PC is not identified (the distribution does not

tighten even with very large N). For more intuition we graph the posteriors:

```
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```

```
## Warning: Removed 4 rows containing missing values
## (`geom_bar()`).
```

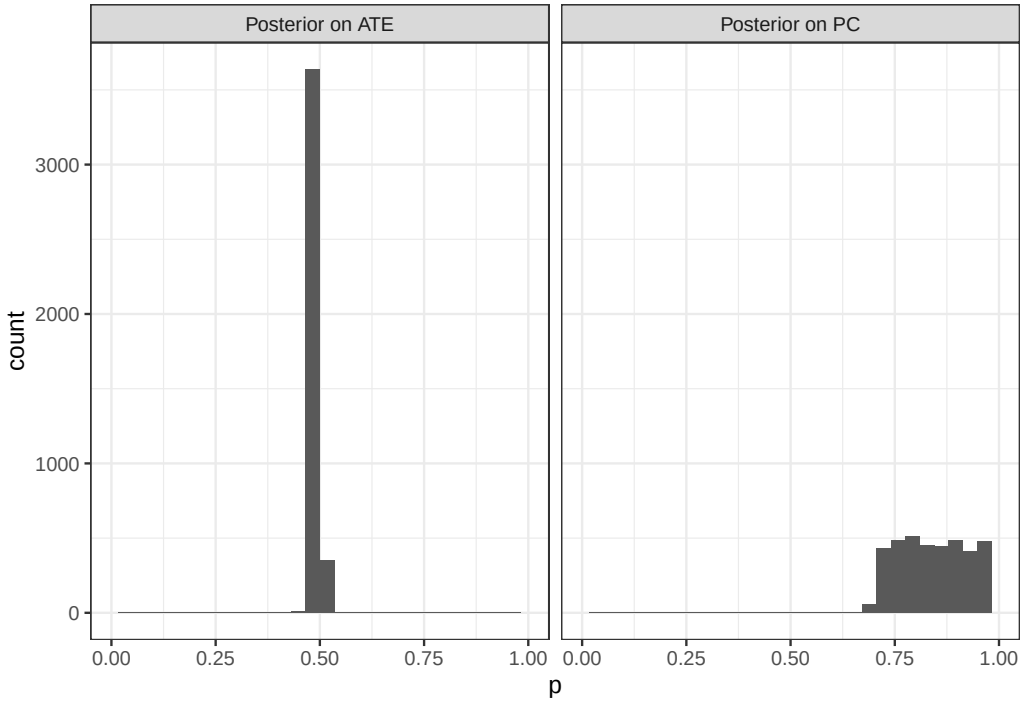


Figure 6.1: ATE is identified, PC is not identified but has informative bounds

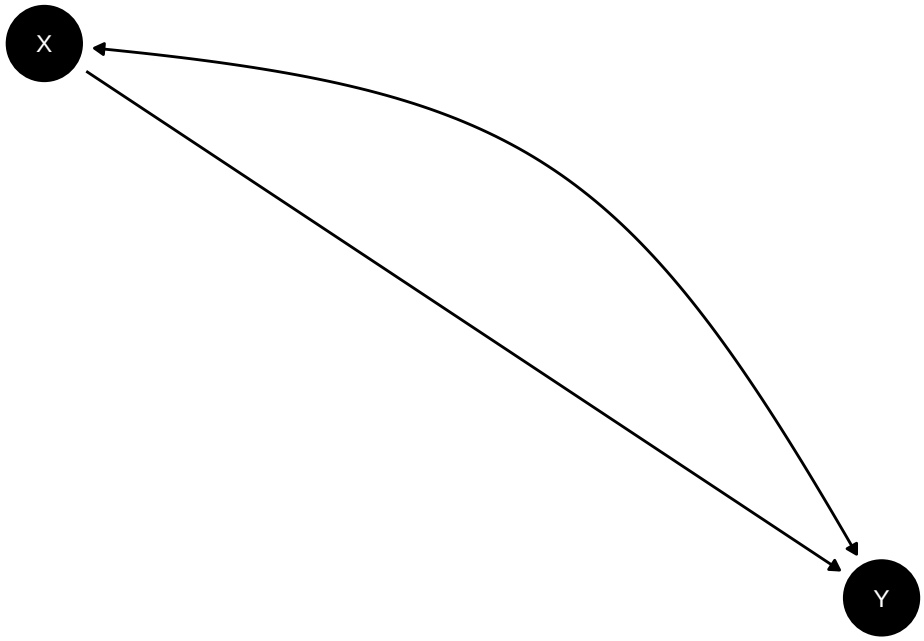
We find that they do not converge but they do place positive mass in the right range. Within this range, the shape of the posterior depends on the priors only.

6.2 X causes Y, with unmodelled confounding

The first model assumed that X was as-if randomly assigned, but we do not need to make such strong assumptions.

An X causes Y model with confounding can be written:

```
model <- make_model("X -> Y; X <-> Y")
plot(model)
```



If we look at the parameter matrix implied by this model we see that it has more parameters than nodal types, reflecting the joint assignment probabilities of θ_X and θ_Y. Here we have parameters for $\Pr(\theta_X = x)$ and $\Pr(\theta_Y|\theta_X = x)$, which allow us to represent $\Pr(\theta_X, \theta_Y)$ via $\Pr(\theta_X = x)\Pr(\theta_Y|\theta_X = x)$.

With the possibility of any type of confounding, the best we can do is place "Manski bounds" on the average causal effect.

To see this, let's plot a histogram of our posterior on average causal effects, given lots of data. We will assume here that in truth there is no confounding, but that that is not known to researchers.

```
data5000 <- make_data(
    model, n = 5000,
    parameters = c(.5, .5, .25, .0, .5, .25, .25, 0, .5, .25))
```

Table 6.1: Parameter matrix for X causes Y model with arbitrary confounding

	X0.Y00	X1.Y00	X0.Y10	X1.Y10	X0.Y01	X1.Y01	X0.Y11	X1.Y11
X.0	1	0	1	0	1	0	1	0
X.1	0	1	0	1	0	1	0	1
Y.00_X.0	1	0	0	0	0	0	0	0
Y.10_X.0	0	0	1	0	0	0	0	0
Y.01_X.0	0	0	0	0	1	0	0	0
Y.11_X.0	0	0	0	0	0	0	1	0
Y.00_X.1	0	1	0	0	0	0	0	0
Y.10_X.1	0	0	0	1	0	0	0	0
Y.01_X.1	0	0	0	0	0	1	0	0
Y.11_X.1	0	0	0	0	0	0	0	1

```
data100 <- data5000[sample(5000, 100), ]
```

The key thing here is that the posterior on the ATE has shifted, as it should, but it is not tight, even with large data. In fact the distribution of the posterior covers one unit of the range between -1 and 1.

6.3 X causes Y, with confounding modeled

Say now we have a theory that the relationship between X and Y is confounded by possibly unobserved variable C. Although C is unobserved we can still include it in the model and observe the confounding it generates by estimating the model on data generated by the model (but without benefiting from observing C). We will assume that it is know that X does not have a negative effet on X. In addition we will assume that both C and X have a positive effect on Y—though this is not known (and so this is built into the model parameters but not into the priors).

```
model <- make_model("C -> X -> Y <- C") |>

  set_restrictions("(X[C=1] < X[C=0])") |>
```

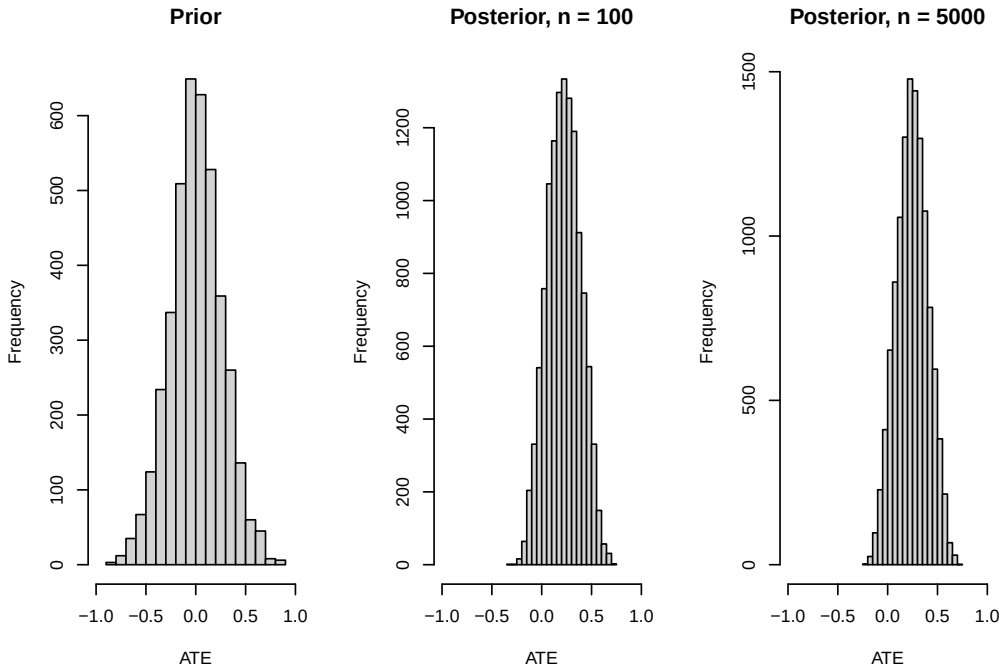


Figure 6.2: Modest gains from additional data when ATE is not identified

```
  set_parameters(
            statement = c("(Y[X=1] < Y[X=0]) | (Y[C=1] < Y[C=0])"),
            parameters = 0)
```

The ATE estimand in this case is given by:

Query	Given	Using	Case.estimand	mean
ATE	-	parameters	FALSE	0.3333

A regression based approach won't fare very well here without data on C. It would yield a precise but incorrect estimate.

```
data <- make_data(model, n = 1000)

estimatr::lm_robust(Y~X, data = data) |>
  tidy() |>
  kable(digits = 2)
```

term	estimate	std.error	statistic	p.value	conf.low	conf.high	df	outcome
(Intercept)	0.26	0.02	13.45	0	0.22	0.30	998	Y
X	0.47	0.03	16.80	0	0.41	0.52	998	Y

In contrast, the Bayesian estimate takes account of the fact that we are missing data on C.

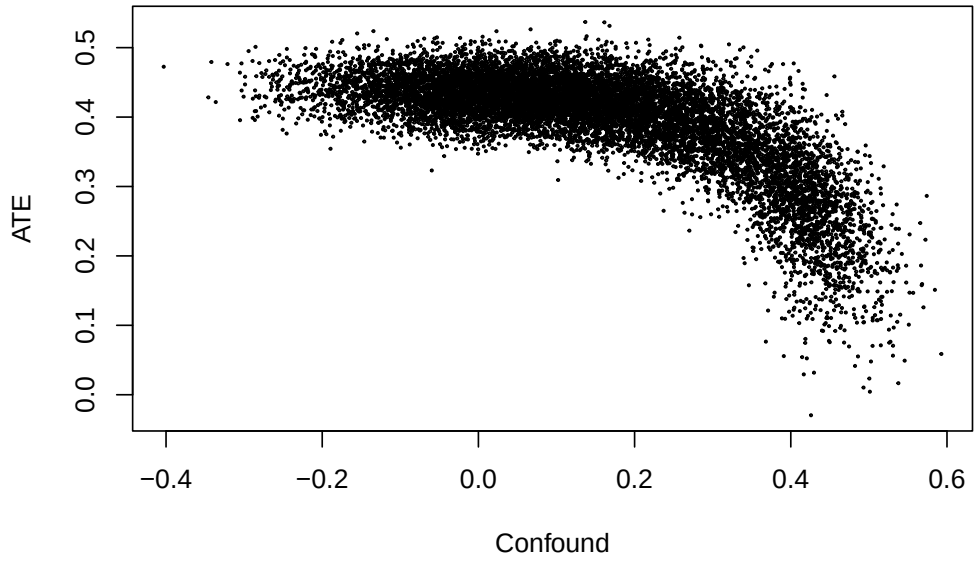
Our posteriors over the effect of X on Y and the effect of the unobserved confounder (C) on Y have a joint distribution with negative covariance.

To illustrate we update on the same data (note that although relationship between C and Y is resrticted in the parameters it is not restricted in the priors). We then plot the joint posterior over our estimand and a measure of confounding (we will use the effect of C on Y, since we have built in already that C matters for X).

```
updated  <- update_model(model, select(data, X, Y))

ate <-
  query_distribution(updated, "c(Y[X=1] - Y[X=0])", using = "posteriors")

confound <-
  query_distribution(updated, "c(Y[C=1] - Y[C=0])", using = "posteriors")
```



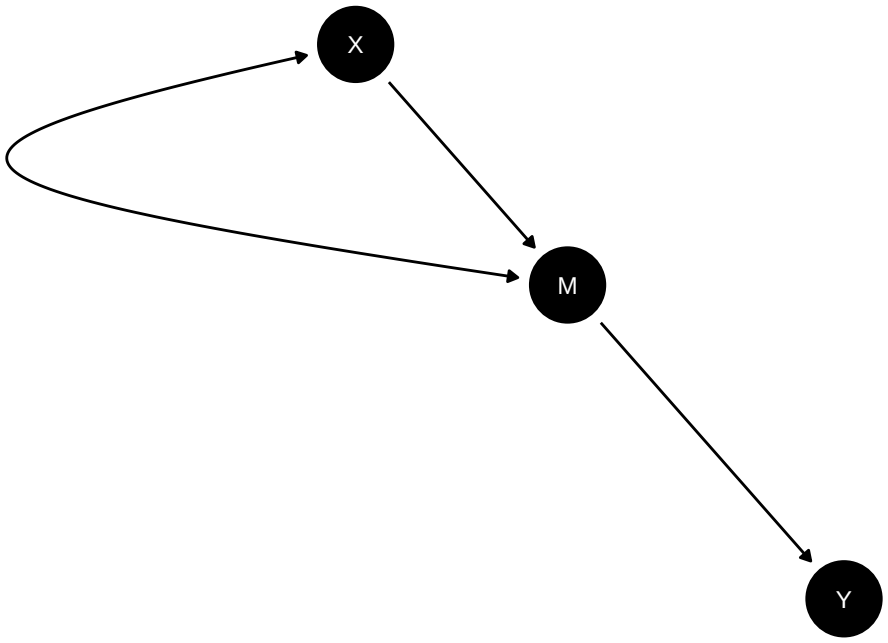
The strong negative correlation shows that when we update we contemplate possibilities in which there is a strong effect and negative confounding, or a weak or even negative effect and positive confounding. If we knew the extent of confounding we would have tighter posteriors on the estimand, but our ignorance regarding the nature of confounding keeps the posterior variance on the estimand large.

6.4 Simple mediation model

We define a simple mediation model and illustrate learning about whether $X = 1$ caused $Y = 1$ from observations of M.

```
model <- make_model("X -> M -> Y; X <-> M") |>
         set_parameters(node = "M", given = "X.0", parameters = c(.2, 0, .8, 0
         set_parameters(node = "M", given = "X.1", parameters = c(.2, .8, 0, 0
         set_parameters(node = "Y", parameters = c(.2, 0, .8, 0))
```

```
plot(model)
```



Data and estimation:

```
data <- make_data(model, n = 1000, using = "parameters")
```

```
updated <- update_model(model, data)
```

```
result <- query_model(
    updated,
    queries = list(COE = "c(Y[X=1] > Y[X=0])"),
    given = c("X==1 & Y==1", "X==1 & Y==1 & M==0", "X==1 & Y==1 & M==1"),
    using = "posteriors")
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.hi
COE	X==1 & Y==1	posteriors	FALSE	0.2441	0.1794	0.0140	0.66
COE	X==1 & Y==1 & M==0	posteriors	FALSE	0.2484	0.2124	0.0059	0.76
COE	X==1 & Y==1 & M==1	posteriors	FALSE	0.2386	0.2138	0.0029	0.76

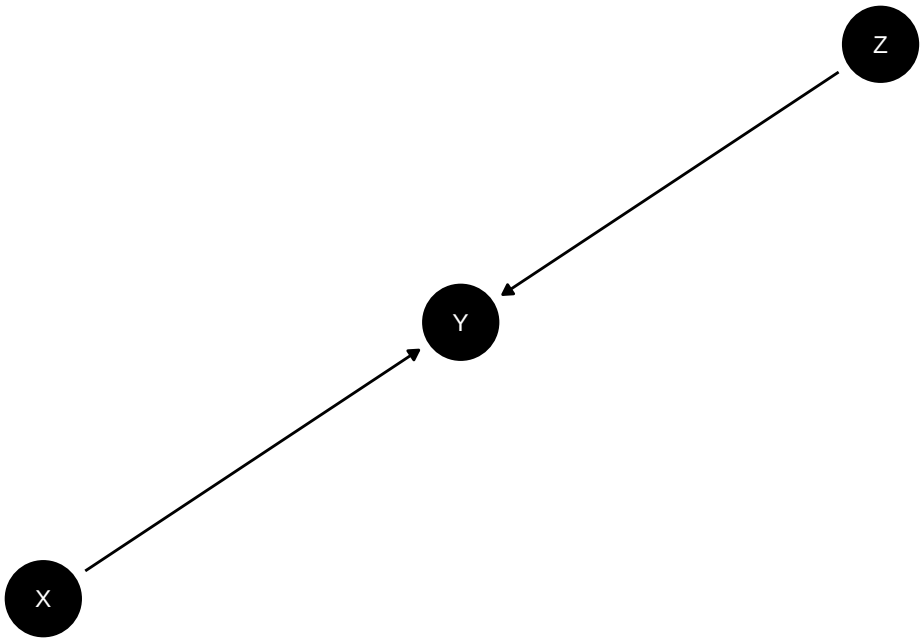
Note that observation of $M = 0$ results in a near 0 posterior that X caused Y,

while observation of $M = 1$ has only a modest positive effect. The mediator thus provides what qualitative scholars call a "hoop" test for the proposition that X caused Y.

6.5 Simple moderator model

We define a simple model with a moderator and illustrate how updating about COE is possible using the value of a moderator as a clue.

```
model <- make_model("X -> Y; Z -> Y")
plot(model)
```




```
data <- make_data(
    model, n = 1000,
    parameters = c(.5, .5, .5, .5,
                   .01, .01, .01, .01, .01, .01, .01, .01,
                   .01, .85, .01, .01, .01, .01, .01, .01))
```

```
posterior <- update_model(model, data)
```

```
result <- query_model(
    updated,
    queries = list(COE = "Y[X=1] > Y[X=0]"),
    given = list("X==1 & Y==1", "X==1 & Y==1 & Z==0", "X==1 & Y==1 & Z==1"),
    using = "posteriors")
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
COE	X==1 & Y==1	posteriors	FALSE	0.89	0.03	0.83	0.94
COE	X==1 & Y==1 & Z==0	posteriors	FALSE	0.42	0.16	0.14	0.74
COE	X==1 & Y==1 & Z==1	posteriors	FALSE	0.93	0.02	0.89	0.97

Knowledge of the moderator provides sharp updating in both directions, depending on what is found.

Chapter 7

Explanation

7.1 Tightening bounds on causes of effects using an unobserved covariate

"Explanation" can sometimes be thought of assessing whether an outcome was due to a cause: "Y because X." We saw examples showing the difficulty of identifying the "probability of causation" (whether X caused Y in a case) above (See for example Figure 6.1).

Knowledge of moderators and mediators can help however. In a particularly striking result, Dawid (2011) shows that knowledge derived from moderators can help *even when the moderator is not observed for the case in question.*

We illustrate with a simple example in which data is drawn from a process in which X has a positive effect on Y when $C = 1$ and a negative effect otherwise. We will assume X is as-if randomized, though C is not:

```
model <- make_model("X -> Y <- C; Y <-> C")

data <-
  model |>
  set_restrictions("(Y[X=1, C=1] > Y[X=0, C=1]) &
                    (Y[X=1, C=0] < Y[X=0, C=0])", keep = TRUE) |>
  make_data(n= 200)
```

Table 7.1: Probability X caused Y from model updated without data on C

Query	Given	Using	Case.estimand	mean	sd	conf.low	co
X caused Y	X==1 & Y==1	posteriors	FALSE	0.5153	0.1108	0.2964	

Table 7.2: Probability X caused Y from model updated using data on C

Query	Given	Using	Case.estimand	mean	sd	conf.low	co
X caused Y	X==1 & Y==1	posteriors	FALSE	0.8706	0.0415	0.7803	

These restrictions, coupled with flat priors, produce the following priors (i) on the effect of X on Y and (ii) that X caused Y in a case with $X = Y = 1$:

Query	Given	Using	Case.estimand	mean
ATE	All	parameters	FALSE	0.0
PC	X==1 & Y==1	parameters	FALSE	0.5

We now compare inferences on the PC (for a case where we have no data on C) using one model that has been updated using data on C and one that has not:

```
update_model(model, select(data, X, Y)) |>

  query_model(query = "Y[X=1]>Y[X=0]", given = "X==1 & Y==1",
              using = "posteriors")

update_model(model, data) |>

    query_model(query = "Y[X=1]>Y[X=0]", given = "X==1 & Y==1",
              using = "posteriors")
```

We see tight gains even though C is not observed. The remarkable result arises because although C is not observed in the case at hand, the model that has been updated with knowledge of C lets us figure out that the average effect of 0 is due to strong heterogeneity of effects. Indeed $X = Y = 1$ only arises when $C = 1$, in which case X causes Y. Thus observing $X = Y = 1$ lets us infer that $C = 1$ and so in this case X causes Y.

7.2 Actual Causation: Billy and Suzy's moderator and mediation model

A classic problem in the philosophy of causation examines a story in which Billy and Suzy throw stones at a bottle (Hall, 2004). Both are deadly shots but Suzy's stone hits the bottle first. Had it not, Billy's surely would have. Can we say that Suzy's throw caused the bottle to break if it would have broken even if she hadn't thrown?

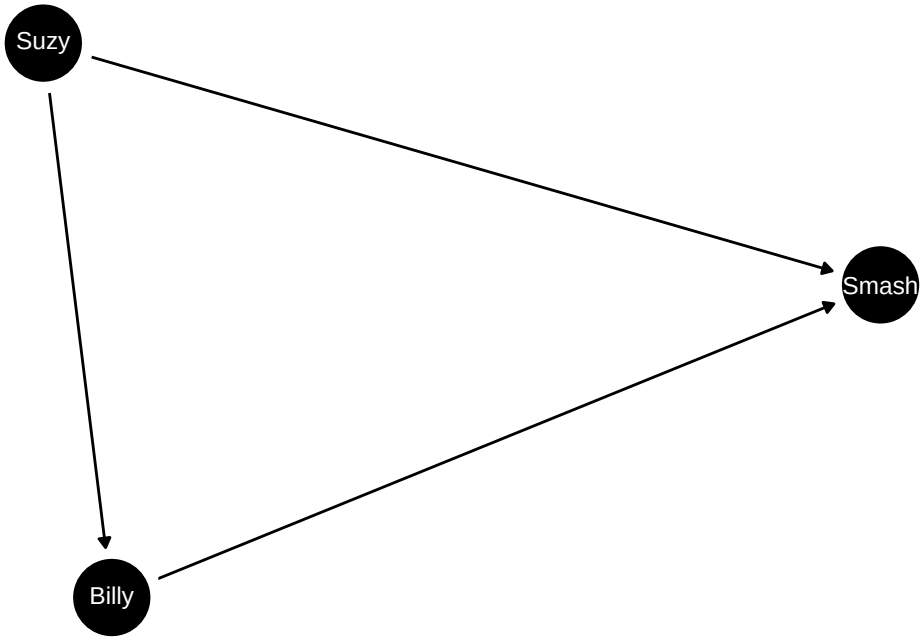
We model a simple version of the Billy and Suzy stone throwing game as a causal model with moderation and mediation in three nodes.

```
model <- make_model("Suzy -> Billy -> Smash <- Suzy") |>
         set_restrictions(c(

           # If Suzy throws the bottle breaks
           "(Smash[Suzy=1]==0)",

           # The bottle won't break by itself
           "(Smash[Billy=0, Suzy = 0]==1)",

           # Suzy's throw doesn't *encourage* Billy to throw
           "Billy[Suzy=1]>Billy[Suzy=0]"))
plot(model)
```



Here "Suzy" means Suzy throws, "Billy": means Billy throws—which he might not do if Suzy throws—and "Smash" means the bottle gets smashed.

The version here is a somewhat less deterministic version of the classic account. Suzy is still an ace shot but now she may or may not throw and Billy may or may not respond positively to Suzy and if he does respond he may or may not be successful. With all these possibilities we have twelve unit causal types instead of 1.

We have two estimands of interest: counterfactual causation and actual causation. Conditional on Suzy throwing and the bottle breaking, would the bottle not have broken had Suzy not thrown her stone? That's counterfactual causation. The actual causation asks the same question but *conditioning* on the fact that Billy did or did not throw *his* stone—which we know could itself be due to Suzy throwing her stone. If so then we might think of an "active path" from Suzy's throw to the smashing, even though had she not thrown the bottle might have smashed anyhow.

Our results:

```
actual_cause <- query_model(model, using = "priors",
  queries = c(
    Counterfactual = "Smash[Suzy = 1] > Smash[Suzy = 0]",
    Actual = "Smash[Suzy = 1, Billy = Billy[Suzy = 1] ] >
              Smash[Suzy = 0, Billy = Billy[Suzy = 1]]"),
  given = c("Suzy==1 & Smash==1", "Suzy==1 & Smash==1 & Billy==0", "Suzy==1 & Smash==
  expand_grid = TRUE
  )
```

Query	Given	Using	Case.estimand	mean	sd	co
Counterfactual	Suzy==1 & Smash==1	priors	FALSE	0.6662	0.2326	
Counterfactual	Suzy==1 & Smash==1 & Billy==0	priors	FALSE	0.7521	0.2149	
Counterfactual	Suzy==1 & Smash==1 & Billy==1	priors	FALSE	0.4972	0.2843	
Actual	Suzy==1 & Smash==1	priors	FALSE	0.8301	0.1677	
Actual	Suzy==1 & Smash==1 & Billy==0	priors	FALSE	1.0000	0.0000	
Actual	Suzy==1 & Smash==1 & Billy==1	priors	FALSE	0.4972	0.2843	

Our inferences, *without even observing* Billy's throw distinguish between Suzy being a counterfactual cause and an actual cause. We think it likely that Suzy's throw was an actual cause of the outcome though we are less sure that it was a counterfactual causes. Observing Billy's throw strengthens our inferences. If Billy didn't throw then we are sure Suzy's throw was the actual cause, though we are still in doubt about whether her throw was a counterfactual cause (since Billy might have thrown if she hadn't).

Note that if we observed Suzy *not* throwing then we would learn *more* about whether she would be a counterfactual cause since we would have learned more about whether Billy reacts to her and also about whether Billy is a good shot.

7.3 Diagnosis: Inferring a cause from symptoms

Sometimes we want to know whether a particular condition was present that could have caused an observed outcome. This is the stuff of medical diagnosis: on observing symptoms, is the sickness due to A or to B?

We imagine cases in which we do not get to observe the putative cause

Table 7.3: Inferences when Suzy does *not* throw

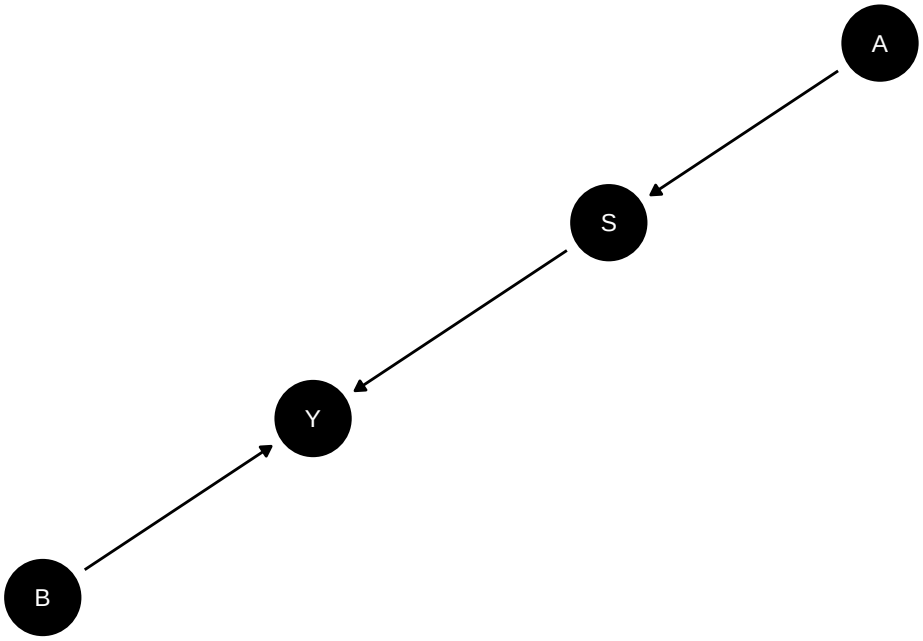
Query	Given	Using	Case.estimand	mean	
Counterfactual	Suzy==0 & Billy==0	priors	FALSE	1.0000	0.0
Counterfactual	Suzy==0 & Billy==1	priors	FALSE	0.5065	0.2
Counterfactual	Suzy==0 & Billy==1 & Smash==1	priors	FALSE	0.0000	0.0
Actual	Suzy==0 & Billy==0	priors	FALSE	1.0000	0.0
Actual	Suzy==0 & Billy==1	priors	FALSE	0.7512	0.2
Actual	Suzy==0 & Billy==1 & Smash==1	priors	FALSE	0.4984	0.2

directly and we want to infer both whether the putative cause was present and whether it caused the outcome. This requires stating a query on both an effect and the level of an unobserved node.

An illustration:

```
model <- make_model("A -> S -> Y <- B") |>
  set_restrictions(c("(S[A=1]< S[A=0])",
                     "(Y[S=1]<Y[S=0])",
                     "(Y[S = 0, B = 0]== 1)"))

plot(model)
```




```
query_model(model,
       queries = list(A="(Y[A=1] > Y[A =0]) & A==1", B="(Y[B=1] > Y[B =0]) & B==1"),
       given = list("Y==1",  "Y==1 & S==1"), using = "priors",
       expand_grid = TRUE) |> kable()
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
A	Y==1	priors	FALSE	0.1949	0.1978	0.0023	0.7133
A	Y==1 & S==1	priors	FALSE	0.2512	0.2271	0.0033	0.7886
B	Y==1	priors	FALSE	0.4435	0.2610	0.0228	0.9150
B	Y==1 & S==1	priors	FALSE	0.2889	0.2207	0.0086	0.7799

In this example there are two possible causes of interest, A and B. With flat priors the B path starts as clearly more probable. Observation of symptom S, which is a consequence of A, however raises the chances that the outcome is due to A and lowers the chances that it is due to B.

Chapter 8

Process tracing

8.1 What to infer from what

The simplest application of the `CausalQueries` package is to figure out what inferences to make about a case upon observing within-case data, given a model. One might observe many pieces of evidence and have to figure out how to update from these jointly.

In *Integrated Inferences* we explore an inequality-democratization model where for a case with low inequality and democratization (say) one is interested in whether the democratization was due to the low inequality. In the simple model, inequality can give rise to popular mobilization which in turn forces democratization; or alternatively, inequality could prevent democratization by generating a threat from elites. In addition other forces, such as international pressure, could give rise to democratization. The question is: how do we update on our beliefs that low inequality caused democratization when we observe mobilization or international pressure?

```
model <- make_model("I -> M -> D <- I; P -> D") |>
  set_restrictions(c(
    "(M[I=1] < M[I=0])",
    "(D[I=1] > D[I=0]) | (D[M=1] < D[M=0]) | (D[P=1] < D[P=0])"))
```

We can read inferences directly from `query_model`:

```
query_model(model,
            query = list(`I = 0 caused D = 1` = "D[I=1] != D[I=0]"),
            using = "parameters",
            given = c("I==0 & D==1",
                      "I==0 & D==1 & M==0",
                      "I==0 & D==1 & M==1",
                      "I==0 & D==1 & P==0",
                      "I==0 & D==1 & P==1",
                      "I==0 & D==1 & M == 0 & P==0",
                      "I==0 & D==1 & M == 1 & P==0",
                      "I==0 & D==1 & M == 0 & P==1",
                      "I==0 & D==1 & M == 1 & P==1")) |> kable()
```

Query	Given	Using	Case.estimand	
I = 0 caused D = 1	I==0 & D==1	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & M==0	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & M==1	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & P==0	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & P==1	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & M == 0 & P==0	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & M == 1 & P==0	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & M == 0 & P==1	parameters	FALSE	0
I = 0 caused D = 1	I==0 & D==1 & M == 1 & P==1	parameters	FALSE	0

We see in this example that learning about a rival cause—the moderator P (international pressure)—induces larger changes in beliefs than learning about the mediator, M (mobilization). The two clues substitute for each other marginally.

The importance of different clues depends however on what one wants to explain. In the next analysis, we see that if we want to know if inequality explained democratization, learning that $M = 0$ has a large impact on beliefs.

Query	Given	Using	Case.estimand	mean
I = 1 caused D = 1	I==1 & D==1	parameters	FALSE	0.1277
I = 1 caused D = 1	I==1 & D==1 & M==0	parameters	FALSE	0.0000
I = 1 caused D = 1	I==1 & D==1 & M==1	parameters	FALSE	0.1500
I = 1 caused D = 1	I==1 & D==1 & P==0	parameters	FALSE	0.2308
I = 1 caused D = 1	I==1 & D==1 & P==1	parameters	FALSE	0.0882

Note that inferences are taken here based on the model made by `make_model`, without any updating of the model using data. In this sense the approach simply makes the model used for process tracing explicit, but it does not justify. It is possible however to first update a model using data from many cases and then use the updated model to draw inferences about a single case.

8.2 Probative value and *d*-separation

Observation of a node (a "clue") is potentially informative for a query when it is *not* *d*-separated[1] from query-relevant nodes (See *Integrated Inferences*, Ch 6).

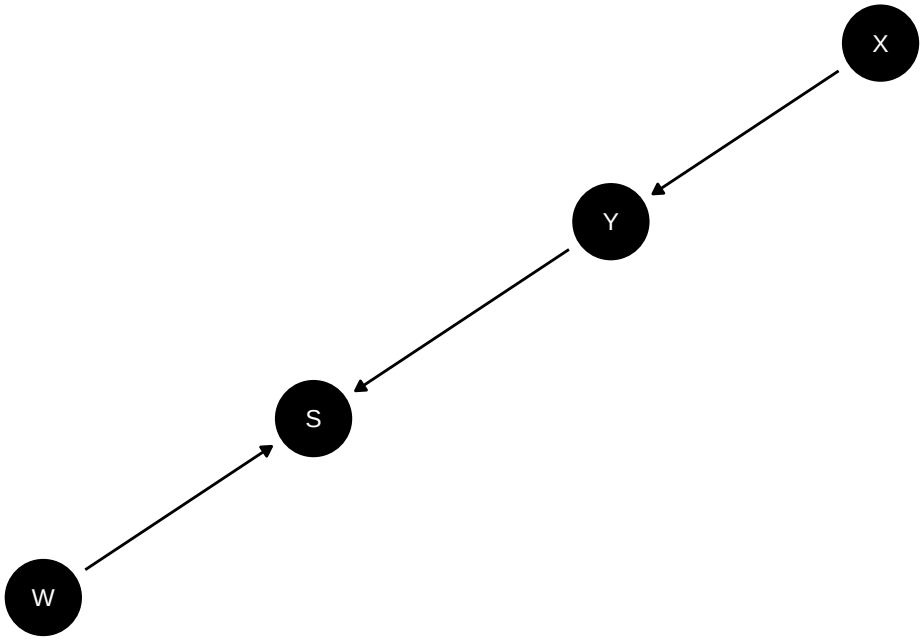
An implication of this is that the observation of some nodes may render other nodes more or less informative. From the graph alone you can sometimes tell when additional data will be uninformative for a query.

To wit:

```
model <- make_model("X -> Y -> S <- W") |>
  set_restrictions(complements("Y", "W", "S"), keep = TRUE)

plot(model)
```

[1] *d*-separation is a key idea in the study of directed acyclic graphs; for an introduction see *d*-separation without tears.




```
query_model(model,
            query = "Y[X=1] > Y[X=0]",
            using = "parameters",
            given = c("X==1",
                      "X==1 & W==1",
                      "X==1 & S==1",
                      "X==1 & S==1 & W==1",
                      "X==1 & Y==1",
                      "X==1 & W==1 & S==1 & Y==1")) |> kable()
```

In this example W is not informative for the X causes Y query (a query about θ^Y, a parent of Y), when Y and S are unobserved (Row 1 = Row 3). It becomes informative, however, when S, a symptom of Y, is observed (Row 3 $\neq$ Row 4). But when Y is observed neither S nor W are informative (Row 5 = Row 6).

The reason is that W is d-separated from θ^Y when Y and S are unobserved. But S is a "collider" for Y and W and so W *becomes* informative about Y once S is observed, and hence of θ^Y (so long as Y is unobserved). When Y

Table 8.1: Whether a clue is informative or not depends on what else has been observed: in particular whether the clue is d-separated from the query.

Query	Given	Using	Case.estimand	mean
1	X==1	parameters	FALSE	0.25
2	X==1 & W==1	parameters	FALSE	0.25
3	X==1 & S==1	parameters	FALSE	0.25
4	X==1 & S==1 & W==1	parameters	FALSE	0.40
5	X==1 & Y==1	parameters	FALSE	0.50
6	X==1 & W==1 & S==1 & Y==1	parameters	FALSE	0.50

is observed however now S and W become d-separated from θ^Y and neither is informative.

8.3 Foundations for Van Evera's tests

Students of process tracing often refer to a set of classical "qualitative tests" that are used to link within-case evidence to inferences around specific (often case-level) hypotheses. The four classical tests as described by Collier (2011) and drawing on Van Evera (1997) are "smoking gun" tests, "hoop" tests, "doubly decisive" tests, and "straw-in-the-wind" tests. A hoop test is one which, if failed, bodes especially badly for a claim; a smoking gun test is one that bodes well for a hypothesis if passed; a doubly decisive test is strongly conclusive no matter what is found, and a straw-in-the-wind test is suggestive, though not conclusive, either way.

In some treatments (such as Humphreys and Jacobs (2015)) formalization involves specifying a prior that a hypothesis is true and an independent set of beliefs about the probability of seeing some data if the hypothesis is true and if it is false. Then updating proceeds using Bayes' rule.

This simple approach suffers from two related weaknesses however: first, there is no good reason to expect these probabilities to be independent; second, there is nothing in the set-up to indicate how beliefs around the probative value of clues can be established or justified.

Both of these problems are easily resolved if the problem is articulated using fully specified causal models.

Many different causal models might justify Van Evera's tests. We illustrate using one in which the requisite background knowledge to justify the tests can be derived from a factorial experiment and in which one treatment serves as a clue for the effect of another.

For the illustration we first make use of a function that generates data from a model with a constrained set of types for Y and a given prior distribution over clue K.

```
van_evera_data <- function(y_types, k_types)

  make_model("X -> Y <- K") |>

  set_restrictions(labels = list(Y = y_types), keep = TRUE) |>

  set_parameters(param_type = "define", node = "K", parameters = c(1 - k_types

  make_data(n = 1000)
```

We then use a function that draws inferences, given different values of a clue K, from a model that has been updated using available data. Note that the model that is updated has no constraints on Y, has flat beliefs over the distribution of K, and imposes no assumption that K is informative for how Y reacts to X.

```
van_evera_inference <- function(data)

  make_model("X -> Y <- K") |>

  update_model(data = data) |>

  query_model(query = "Y[X=1] > Y[X=0]",
              given = c(TRUE, "K==0", "K==1"),
              using = "posteriors")
```

We can now generate posterior beliefs, given K, for different types of tests where the tests are now justified by different types of data, coupled with a common prior causal model.

Results:

Table 8.2: Doubly decisive test

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	posteriors	FALSE	0.5091	0.0155	0.4793	0.5390
Q 1	K==0	posteriors	FALSE	0.0094	0.0052	0.0023	0.0222
Q 1	K==1	posteriors	FALSE	0.9763	0.0074	0.9591	0.9886

Table 8.3: Hoop test

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	posteriors	FALSE	0.4605	0.0224	0.4174	0.5033
Q 1	K==0	posteriors	FALSE	0.0367	0.0231	0.0069	0.0933
Q 1	K==1	posteriors	FALSE	0.5122	0.0243	0.4646	0.5594

```
doubly_decisive <- van_evera_data("0001", .5) |> van_evera_inference()

hoop            <- van_evera_data(c("0001", "0101"), .9) |> van_evera_inference()

smoking_gun     <- van_evera_data(c("0001", "0011"), .1) |> van_evera_inference()

straw_in_wind   <- van_evera_data(c("0001", "0101", "0011"), .5) |> van_evera_infere
```

We see that these tests all behave as expected. Importantly, however, the approach to thinking about the tests is quite different to that described in Collier (2011) or Humphreys and Jacobs (2015). Rather than having a belief about the probative value of a clue, and a prior over a hypothesis, inferences are drawn directly from a causal model that relates a clue to possible causal effects. Critically, with this approach, the inferences made from observing

Table 8.4: Smoking gun test

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	posteriors	FALSE	0.5240	0.0226	0.4799	0.5692
Q 1	K==0	posteriors	FALSE	0.4801	0.0243	0.4323	0.5283
Q 1	K==1	posteriors	FALSE	0.8991	0.0371	0.8132	0.9566

Table 8.5: Straw in the wind test

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	posteriors	FALSE	0.4919	0.0216	0.4492	0.5340
Q 1	K==0	posteriors	FALSE	0.3075	0.0285	0.2544	0.3649
Q 1	K==1	posteriors	FALSE	0.6828	0.0301	0.6224	0.7404

clues can be justified by reference to a more fundamental, agnostic model, that has been updated in light of data. The updated model yields both a prior over the proposition, belief about probative values, and guidance for what conclusions to draw given knowledge of K.

8.4 Clue selection: clues at the center of chains can be more informative

Model querying can also be used to assess which types of clues are more informative among a set of informative clues. Consider a chain linking X to Y via M_1, M_2, M_3. To keep things simple let's assume that the chain is monotonic: no node in the chain has a negative effect on the next node in the chain.

Which clue is most informative for the proposition that X caused Y in a case with $X = Y = 1$?

In all case we will conclude that X did not cause Y if we see a 0 along the chain (since a 1 can not cause a 0). But what do we conclude if we see a 1?

```
model <- make_model("X -> M1 -> M2 -> M3 -> Y") |>
  set_restrictions(labels = list(M1 = "10", M2 = "10", M3 = "10", Y = "10"))
```

In imposing monotonicity and using default parameter values we are assuming that the effect of each node on the next node is 1/3. What does this imply for our query? We get the answer using `query_model`.

```
query_model(model,
            query = "Y[X=1] > Y[X=0]",
            given = c("X==1 & Y==1", "X==1 & Y==1 & M1==1", "X==1 & Y==1 & M2=
                      "X==1 & Y==1 & M3==1", "X==1 & Y==1 & M1==1 & M2==1 & M3
```

```
            using= "parameters") |> kable()
```

Query	Given	Using	Case.estimand	mean
Q 1	X==1 & Y==1	parameters	FALSE	0.0244
Q 1	X==1 & Y==1 & M1==1	parameters	FALSE	0.0357
Q 1	X==1 & Y==1 & M2==1	parameters	FALSE	0.0400
Q 1	X==1 & Y==1 & M3==1	parameters	FALSE	0.0357
Q 1	X==1 & Y==1 & M1==1 & M2==1 & M3==1	parameters	FALSE	0.0625

A couple of features are worth noting. First without any data our beliefs that X caused Y are quite low. This is due to the fact that even though the ATE at each step is reasonably large, the ATE over the whole chain is small, only $(1/3)^4$) (incidentally, a beautiful number: 0.01234568).

Second we learn from which nodes we learn the most. We update most strongly from positive evidence on the middle mediator. One can also show that not only is there greater updating higher if a positive outcome is seen on the middle mediator, but the *expected* reduction in posterior variance is also greater (expected reduction in posterior variance takes account of the probability of observing different outcomes, which can also be calculated from the model given available data.)[2]

Last, while we update most strongly when we observe positive evidence on all steps, even that does not produce a large posterior probability that $X = 1$ caused $Y = 1$. Positive evidence on a causal chain is often not very informative. Explanations for this are in Dawid et al. (2019).

[2]These quantities can be calculated by the `CQtools` package, still in alpha, via: `CQtools::expected_learning(model, "Y[X=1] > Y[X=0]", given = "X==1 & Y==1", strategy = "M2")`

Chapter 9

Identification

9.1 Illustration of the backdoor criterion

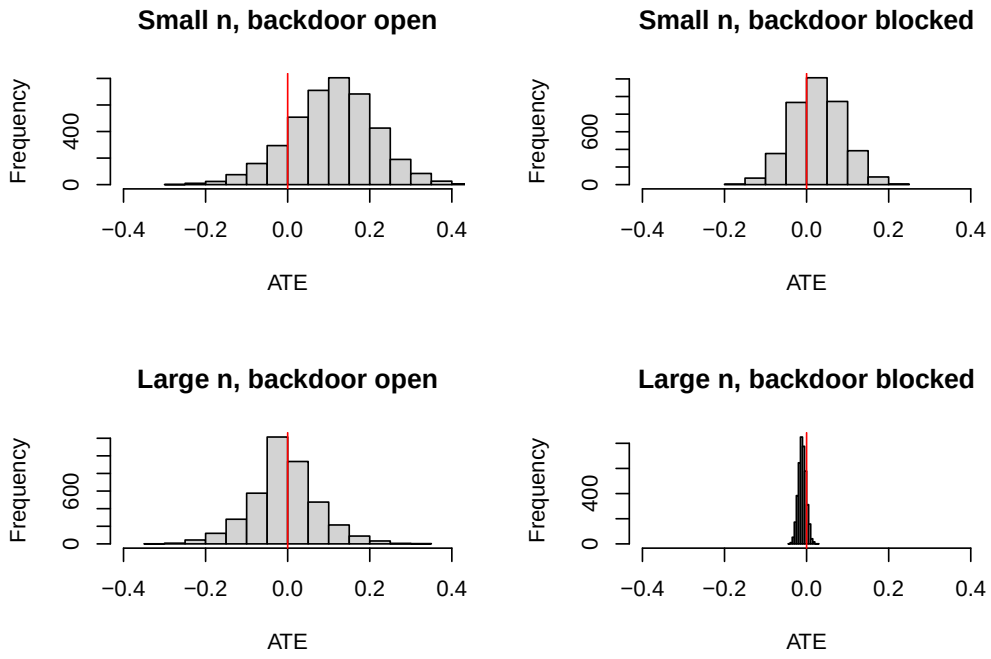
Perhaps the most common approach to identifying causal effects in observational research is to condition on possible confounders. The "backdoor" criterion for identifying an effect of X on Y involves finding a set of nodes to condition on that collectively block all "backdoor paths" between X and Y. The intuition is that if these paths are blocked, then any systematic correlation between X and Y reflects the effect of X on Y.

To illustrate the backdoor criterion we want to show that estimates of the effect of X on Y are identified if we have data on a node that blocks a backdoor path—C—but not otherwise. With `CausalQueries` models however, rather than conditioning on C we simply include data on C in our model and update as usual.

```
model <- make_model("C -> X -> Y <- C")  |>
         set_restrictions("(Y[C=1]<Y[C=0])")


# Four types of data: Large, small, door open, door closed
N <- 10000
df_closed_door_large <- make_data(model, n = N)
df_open_door_large   <- mutate(df_closed_door_large, C = NA)
df_closed_door_small <- df_closed_door_large[sample(N, 200), ]
```

```
df_open_door_small   <- df_open_door_large[sample(N, 200), ]
```



We see that with small n (200 units), closing the backdoor (by including data on C) produces a tighter distribution on the ATE. With large N (10,000 units) the distribution around the estimand collapses when the backdoor is closed but not when it is open.

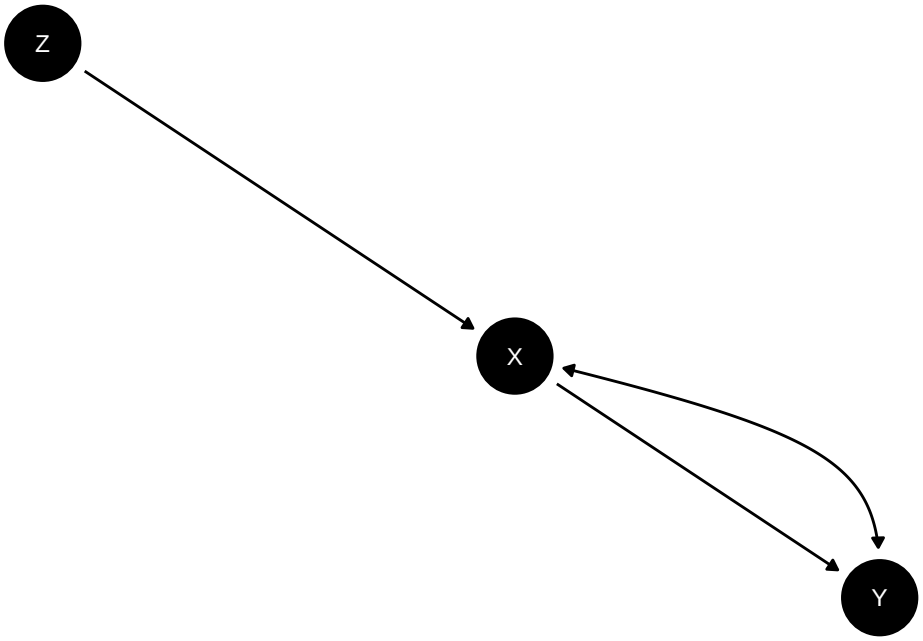
9.2 Identification: Instruments

We illustrate how you can learn about whether $X = 1$ caused $Y = 1$ by taking advantage of an "instrument," Z.

We start with a model that builds in the instrumental variables exclusion restriction (no unobserved confounding between Z and Y, no paths between Z and Y except through X) but does not include a monotonicity restriction (no negative effect of Z on X).

```
model <- make_model("Z -> X -> Y; X <-> Y")

plot(model)
```




```
result <- query_model(
    updated,
    queries = list(ATE = "c(Y[X=1] - Y[X=0])"),
    given = list(TRUE, "X[Z=1] > X[Z=0]",  "X==0",  "X==1"),
    using = "posteriors")
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ATE	-	posteriors	FALSE	0.4784	0.1073	0.2604	0.6316
ATE	X[Z=1] > X[Z=0]	posteriors	FALSE	0.5813	0.0504	0.4765	0.6736
ATE	X==0	posteriors	FALSE	0.4851	0.1309	0.2155	0.6646
ATE	X==1	posteriors	FALSE	0.4716	0.1344	0.1980	0.6545

We calculate the average causal effect (a) for all (b) for the compliers and (c) conditional on values of M.

We see here that the effects are strongest for the "compliers"—units for whom X responds positively to Z; in addition they are stronger for the treated than for the untreated. Moreover we see that the posterior variance on the complier average effect is low. If our model also imposed a monotonicity assumption then it would be lower still.

```
model <- make_model("Z -> X -> Y; X <-> Y")  |>
         set_restrictions(decreasing("Z", "X"))
```

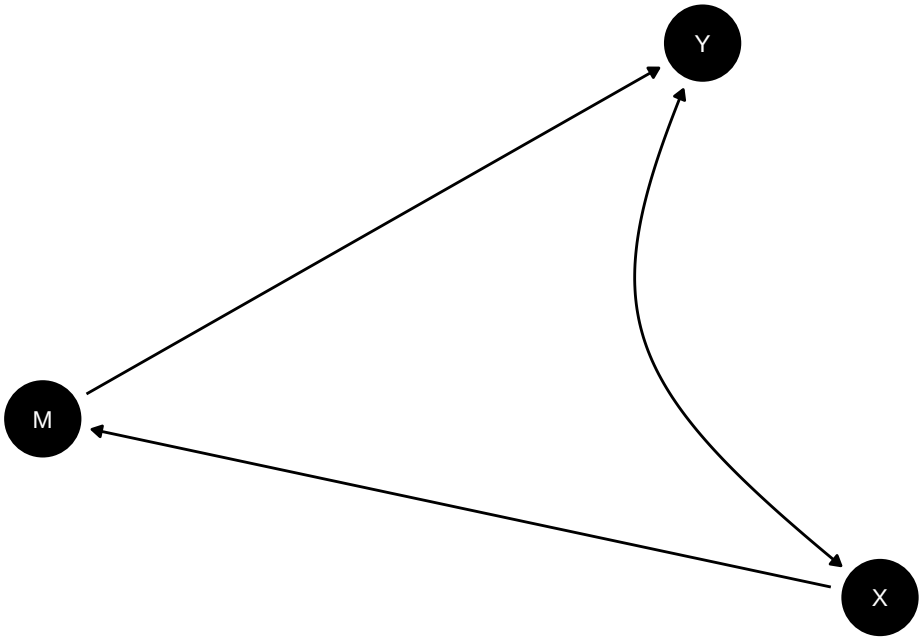
Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.hi
ATE	-	posteriors	FALSE	0.6055	0.0113	0.5838	0.6
ATE	X[Z=1] > X[Z=0]	posteriors	FALSE	0.6055	0.0113	0.5838	0.6
ATE	X==0	posteriors	FALSE	0.6055	0.0113	0.5838	0.6
ATE	X==1	posteriors	FALSE	0.6055	0.0113	0.5838	0.6

9.3 Identification through the frontdoor

A less well known approach to identification uses information on the causal path from X to Y. Consider the following model:

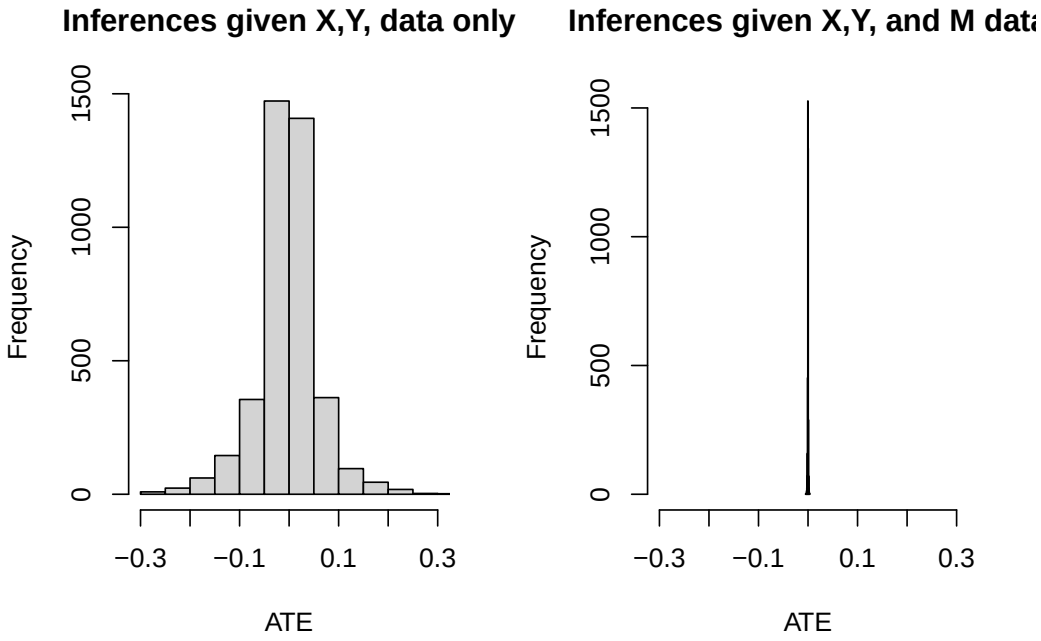
```
frontdoor <- make_model("X -> M -> Y; X <-> Y")

plot(frontdoor)
```



Although in both the instrumental variables (IV) setup and the frontdoor setup we are trying to deal with confounding between X and Y, the two differ in that in the IV set up we make use of a variable that is prior to X whereas in the frontdoor model we make use of a variable between X and Y. In both cases we need other exclusion restrictions: here we see that there is no unobserved confounding between X and M or between M and Y. Importantly too there is no direct path from X to Y, only the path that runs through M.

Below we plot posterior distributions given observations on 2000 units, with and without data on M:



The spike on the right confirms that we have identification.

9.4 Simple sample selection bias

Say we are interested in assessing the share of Republicans in a population but Republicans are (possible) systematically likely to be absent from our sample. What inferences can we make given our sample?

We will assume that we know when we have missing data, though of course we do not know the value of the missing data.

To tackle the problem we will include sample selection into our model:

```
model <- make_model("R -> S") |>
  set_parameters(node = "R", parameters = c(2/3,1/3)) |>
  set_parameters(node = "S", parameters = c(1/3, 0, 1/3, 1/3))

data <- make_data(model, n = 1000) |>
        mutate(R = ifelse(S==0, NA, R ))
```

From this data and model, the priors and posteriors for population and sample quantities are:

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	parameters	FALSE	0.3333		0.3333	0.3333
Q 1	-	priors	FALSE	0.4975	0.2908	0.0236	0.9741
Q 1	-	posteriors	FALSE	0.5043	0.1391	0.2553	0.7352
Q 1	S==1	parameters	FALSE	0.5000		0.5000	0.5000
Q 1	S==1	priors	FALSE	0.4967	0.3071	0.0186	0.9820
Q 1	S==1	posteriors	FALSE	0.4941	0.0234	0.4478	0.5398

For the population average effect we tightened our posteriors relative to the priors, though credibility intervals remain wide, even with large data, reflecting our uncertainty about the nature of selection. Our posteriors on the sample mean are accurate and tight.

Importantly we would not do so well if our data did not indicate that we had missingness.

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	parameters	FALSE	0.3333		0.3333	0.3333
Q 1	-	posteriors	FALSE	0.5096	0.0074	0.4954	0.5246
Q 1	S==1	parameters	FALSE	0.5000		0.5000	0.5000
Q 1	S==1	posteriors	FALSE	0.5096	0.0074	0.4955	0.5245

We naively conclude that all cases are sampled and that population effects are the same as sample effects. The problem here arises because the causal model does not encompass the data gathering process.

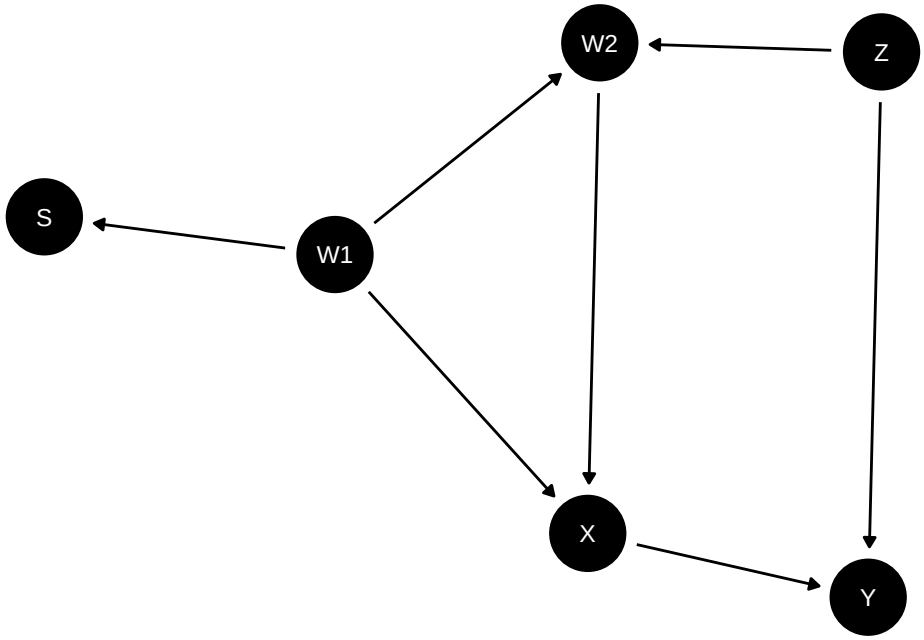
9.5 Addressing both sample selection bias and confounding

Consider the following model from Bareinboim and Pearl (2016) (their Figure 4C). The key feature is that data is only seen for units with $S = 1$ (S for sampling).

In this model the relationship between X and Y is confounded. Two strategies work to address confounding: controlling for either Z or for $W1$ *and* $W2$ works. But only the first strategy addresses the sample selection prob-

lem properly. The reason is that Z is independent of S and so variation in Z is not affected by selection on S.

```
selection <- make_model("X <- W1 -> W2 -> X -> Y <- Z -> W2; W1 -> S")
```



To keep the parameter and type space small we also impose a set of restrictions: S is non decreasing in W_1, X is not decreasing in either $W1$ or $W2$, Y is not decreasing Z or X and X affects Y only if $Z = 1$. $W_2 = 1$ if and only if both $W_1 = 1$ and $Z = 1$. These all reduce the problem to one with 18 nodal types and 288 causal types.

Worth noting that in this model although selection is related to patterns of confounding, it is not related to causal effects: the effect of X on Y is not different from units that are or are not selected.

Given these priors we will assume a true (unknown) data generating process with no effect of X on Y, in which W_1 arises with a $1/3$ probability but has a strong positive effect on selection into the sample when it does arise.

The estimand values given the true parameters and priors for this model are as shown below.

Table 9.1: Estimand values

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Q 1	-	parameters	FALSE	0.000		0.0000	0.0000
Q 1	-	priors	FALSE	0.336	0.1541	0.0839	0.6725

This confirms a zero true effect, though priors are dispersed, centered on a positive effect.

We can see the inference challenge from observational data using regression analysis with and without conditioning on Z and W_1, W_2.

Dependent variable:

Y

(1)

(2)

(3)

X

0.081***

0.080***

-0.006

(0.008)

(0.016)

(0.008)

X:W1_norm

0.161***

(0.031)

X:W2_norm

0.098***

(0.033)

X:Z_norm

-0.047***

(0.016)

Observations

14,947

14,947

14,947

R2

0.007

0.021

0.117

Adjusted R2

0.007

0.020

0.117

Residual Std. Error

0.498 (df = 14945)

0.495 (df = 14941)

0.470 (df = 14943)

F Statistic

99.020*** (df = 1; 14945)

62.770*** (df = 5; 14941)

661.200*** (df = 3; 14943)

Note:

p<0.1; ***p<0.05;*** p<0.01

Naive analysis is far off; but even after conditioning on W_1, W_2 we still wrongly infer a positive effect.

Bayesian inferences given different data strategies are shown below:

data	mean	sd
X,Y	0.0714	0.0144
X,Y, W1, W2	0.0127	0.0061
X, Y, Z	0.0133	0.0054

We see the best performance is achieved for the model with data on Z—in this case the mean posterior estimate is closest to the truth–0–and the standard deviation is lowest also. However the gains in choosing Z over $W1, W2$ are not as striking as in the regression estimates since knowledge of the model structure protects us from error.

9.6 Learning from a collider!

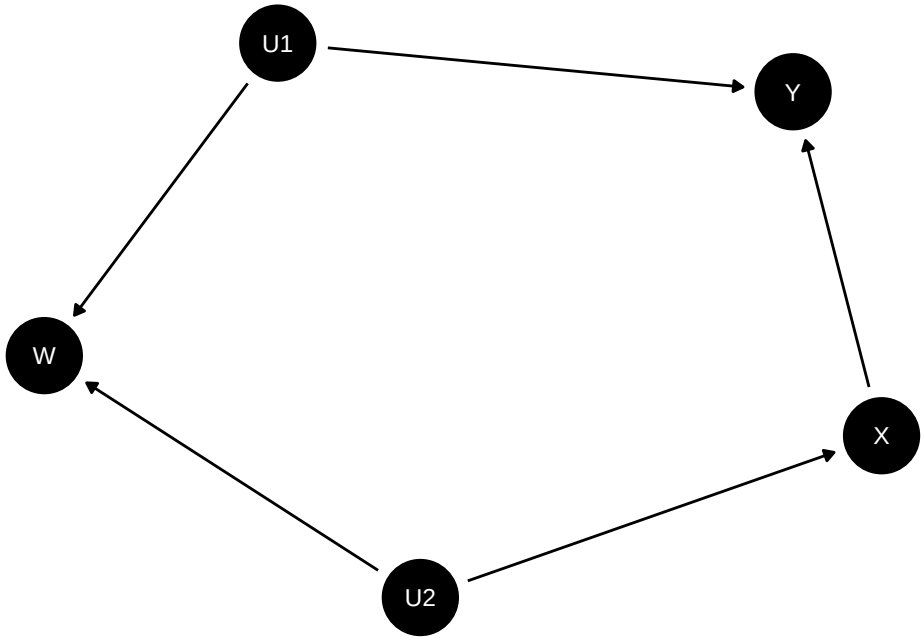
Conditioning on a collider can be a bad idea as it can introduce a correlation between variables that might not have existed otherwise (Elwert and Winship, 2014). But that doesn't mean colliders should be ignored in analysis altogether. For a Bayesian, knowledge of the value of a collider can still be informative.

Pearl describes a model similar to the following as a case for which controlling for covariate W induces bias in the estimation of the effect of X on Y, which could otherwise be estimated without bias using simple differences in means.

```
model <- make_model("X -> Y <- U1 -> W <- U2 -> X") |>

  set_restrictions(labels = list(Y = c("0001", "1111"), W = "0001"), keep = TRUE) |>
  set_restrictions("(X[U2=1]<X[U2=0])") |>
  set_parameters(node = "U1",  parameters = c(1/4, 3/4)) |>
  set_parameters(node = "Y", parameters = c(2/3, 1/3))

plot(model)
```




```
data <- make_data(model,
                  n = 25000,
                  vars = c("W", "X", "Y"),
                  using = "parameters")
```

The effect of X on Y is .5 but average effects as well as the probability of causation, are different for units with $W = 0$ and $W = 1$ (this, even though W does not affect Y):

Query	Given	Using	Case.estimand	mean
Y(1)-Y(0)	-	parameters	FALSE	0.5000
Y(1)-Y(0)	W==0	parameters	FALSE	0.4000
Y(1)-Y(0)	W==1	parameters	FALSE	0.6667
Y(1)-Y(0)	X==1 & Y==1	parameters	FALSE	0.6000
Y(1)-Y(0)	X==1 & Y==1 & W==0	parameters	FALSE	0.5000
Y(1)-Y(0)	X==1 & Y==1 & W==1	parameters	FALSE	0.6667

These are the quantities we seek to recover. The ATE can be gotten fairly precisely in a simple regression. But controlling for W introduces bias in the

estimation of this effect (whether done using a simple control or an interactive model):

Dependent variable:

Y

(1)

(2)

(3)

X

0.498***

0.453***

0.458***

(0.005)

(0.005)

(0.005)

W

0.187***

(0.006)

W_norm

-0.008

(0.008)

X:W_norm

0.346***

(0.011)

Constant

0.334***

0.285***

0.333***

(0.004)

(0.004)

(0.004)

Observations

25,000

25,000

25,000

R2

0.255

0.286

0.313

Adjusted R2

0.255

0.286

0.313

Residual Std. Error

0.426 (df = 24998)

0.417 (df = 24997)

0.409 (df = 24996)

F Statistic

8,539.000*** (df = 1; 24998)

5,013.000*** (df = 2; 24997)

3,795.000*** (df = 3; 24996)

Note:

Table 9.2: Collider excluded from model

Query	Given	Using	Case.estimand	mean	sd	conf.low	co
Y(1)-Y(0)	-	posteriors	FALSE	0.4904	0.0054	0.4797	
Y(1)-Y(0)	W==0	posteriors	FALSE	0.3608	0.1086	0.0795	
Y(1)-Y(0)	W==1	posteriors	FALSE	0.6620	0.0042	0.6535	
Y(1)-Y(0)	X==1 & Y==1	posteriors	FALSE	0.5920	0.0054	0.5813	
Y(1)-Y(0)	X==1 & Y==1 & W==0	posteriors	FALSE	0.4181	0.1603	0.0364	
Y(1)-Y(0)	X==1 & Y==1 & W==1	posteriors	FALSE	0.6620	0.0042	0.6535	

Table 9.3: Collider included in model

Query	Given	Using	Case.estimand	mean	sd	conf.low	co
Y(1)-Y(0)	-	posteriors	FALSE	0.4905	0.0054	0.4795	
Y(1)-Y(0)	W==0	posteriors	FALSE	0.3878	0.0067	0.3741	
Y(1)-Y(0)	W==1	posteriors	FALSE	0.6621	0.0042	0.6540	
Y(1)-Y(0)	X==1 & Y==1	posteriors	FALSE	0.5921	0.0053	0.5817	
Y(1)-Y(0)	X==1 & Y==1 & W==0	posteriors	FALSE	0.4874	0.0079	0.4712	
Y(1)-Y(0)	X==1 & Y==1 & W==1	posteriors	FALSE	0.6621	0.0042	0.6540	

p<0.1; ***p<0.05;*** p<0.01

How does the Bayesian model do, with and without data on W?

Without W we have:

Thus we estimate the treatment effect well. What's more we can estimate the probability of causation when $W = 1$ accurately, even though we have not observed W. The reason is that if $W = 1$ then, given the model restrictions, we know that both $U_1 = 1$ and $U_2 = 1$ which is enough. We are not sure however what to infer when $W = 0$ since this could be due to either $U_1 = 0$ or $U_2 = 0$.

When we incorporate data on W our posteriors are:

We see including the collider does not induce error in estimation of the ATE, even though it does in a regression framework. Where we do well before we continue to do well. However the new information lets us improve our model and, in particular, we see that we now get a good and tight estimate for the

probability that $X = 1$ caused $Y = 1$ in a case where $W = 0$.

In short, though conditioning on a collider induces error in a regression framework; including the collider as data for updating our causal model doesn't hurt us and can help us.

Chapter 10

Mixing methods

In Humphreys and Jacobs (2015) we describe an approach to mixed methods in which within-case inference from a small set of cases are combined with cross sectional data from many cases to form integrated inferences. Reconceived of as a process of updating causal models, the distinction between within-case and cross case data becomes difficult to maintain—both are, after all, just nodes on a model. However the basic procedure can still be implemented, with, in this case, a rooted justification for *why* within case information is informative for estimands of interest.

10.1 Using within case data to help with identification

Here is a model in which a little within-case data adds a lot of leverage to assessing estimands of interest that cannot be estimated confidently with X, Y data alone. In this model a front door type criterion is half satisfied. We assume $X \rightarrow M \rightarrow Y$ but we allow confounding to take the form of X less likely in cases where $Y = 1$ regardless of X.

With simple X, Y data, even on many cases (20,000 here), we cannot get precise estimates and we greatly underestimate the effect of X on Y; with full data on M in many cases we do very well.

But we *also* do well even with quite partial data on M, e.g. if we have data

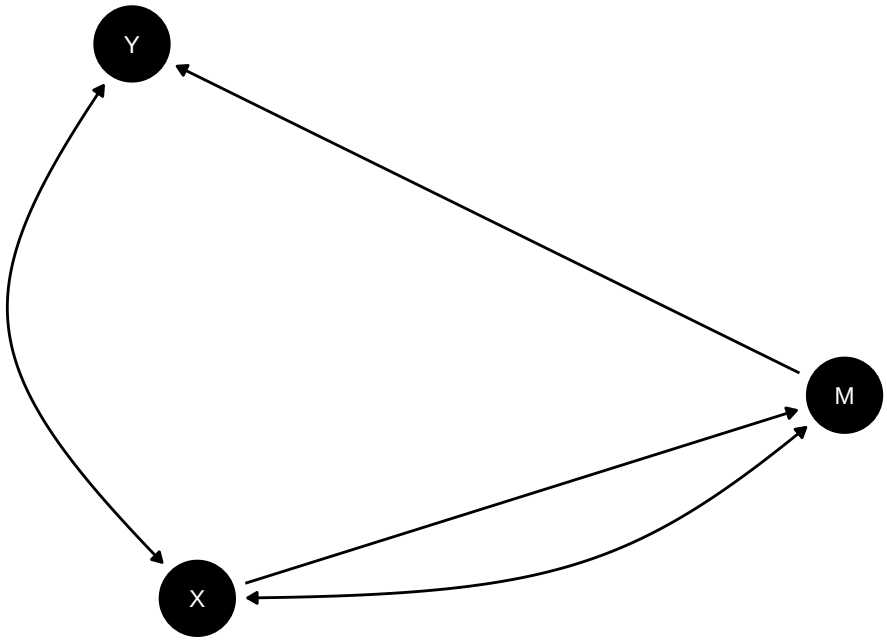
for 100 out of 20,000 cases.

Thus a little within-case data helps us make sense of the X, Y data we have.

The model:

```
model <-
  make_model("X -> M -> Y; M <-> X; X <-> Y")

plot(model)
```



Parameters: We imagine a true model with a treatment effect of .25 but positive confounding ($X = 1$ more likely in cases in which $Y = 1$, regardless of X.

```
model <-
  make_model("X -> M -> Y;  X <-> Y")

model <- model |>
  set_parameters(
       node = c("X", "M", "Y"),
```

```
        parameters = c(0.75, 0.25,
          .25, 0, .5, .25,
          .25, 0, .5, .25,
          .25, 0, .5, .25))

query_model(model, list(ATE = "Y[X=1] - Y[X=0]"), using = "parameters")  |> kable()
```

Query	Given	Using	Case.estimand	mean
ATE	-	parameters	FALSE	0.25

We use the model to simulate different types of data we might have access to thus:

```
n <- 20000

full_data <-  make_data(model, n, using = "parameters")

XY_data <- full_data |> select(X, Y)

some_data  <- full_data |>  mutate(M = ifelse((1:n) %in% sample(1:n, 100), M, NA))
```

Naive analysis:

```
summary(lm(Y~X, data = full_data))$coef |>
  kable(digits = 2)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.38	0.00	95.25	0
X	0.25	0.01	31.32	0

Note that the estimated ATE is too high – over twice what it should be.

In contrast as seen in the next graph the `CausalQueries` estimate using X, Y data only is low – close to our prior at 0 and estimated with wide posterior variance.

With full data on X, M and Y we get a tight estimate on the ATE, though our estimate of the nature of confounding is not identified—though it has much tighter bounds than before. With partial data on M (100 cases out of 20000) we do nearly as well. A small amount of data is enough to narrow bounds on confounding and improve our estimates of the ATE considerably.

```
# X, Y data only
updated_XY      <- update_model(model, XY_data)

# Full Data
updated_full    <- update_model(model, full_data)

# Partial Data
updated_partial <- update_model(model, some_data)
```

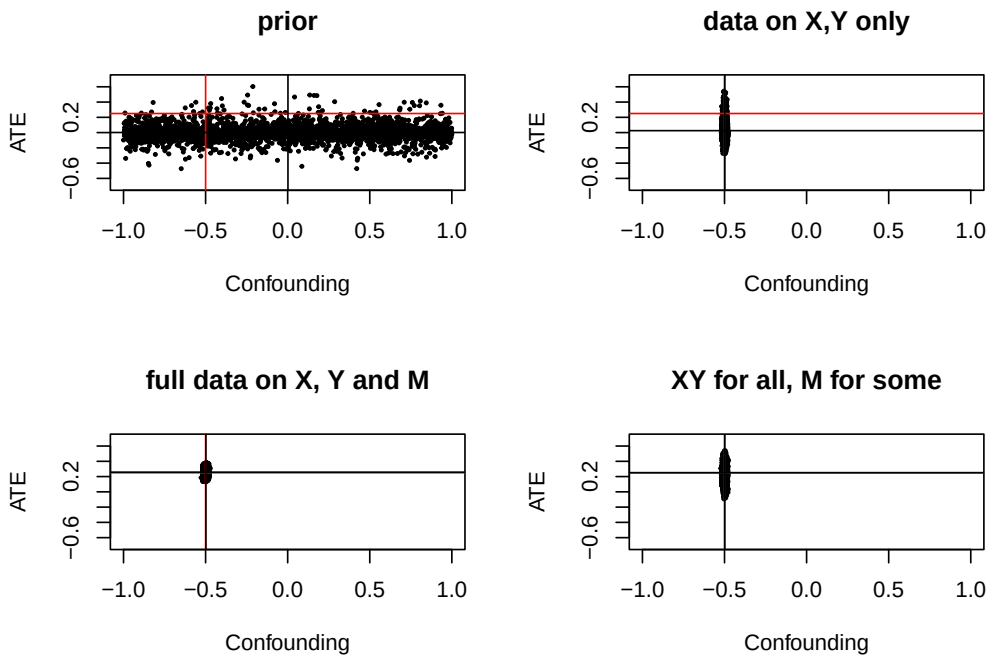


Figure 10.1: Red lines denote estimand values, black lines show posterior means. Full data allows narrowing of posterior variance on confounding and tight esimates of treatment effects. But even limited data on M gets us quite far (bottom right panel).

10.2 Distinguishing paths

Here is another example of mixing methods where a little within-case data goes a long way. We imagine that available data makes us very confident that X causes Y but we want to know about channels. In such cases, with high confidence about overall effects, and *absent unobserved confounding*, a little data on mediators might be highly informative.

```
model <- make_model("X -> M1 -> Y <- M2 <- X") |>
  set_restrictions(c(decreasing("X", "M1"),
                     decreasing("M1", "Y"),
                     decreasing("X", "M2"),
                     decreasing("M2", "Y"))) |>
  set_parameters(node = "M1", parameters = c(0,1,0), normalize=FALSE)  |>
  set_parameters(node = "M2", parameters = c(.5, 0,.5), normalize=FALSE)  |>
  set_parameters(statement = "(Y[M1=1] == Y[M1=0])", parameters = 0)

plot(model)
```

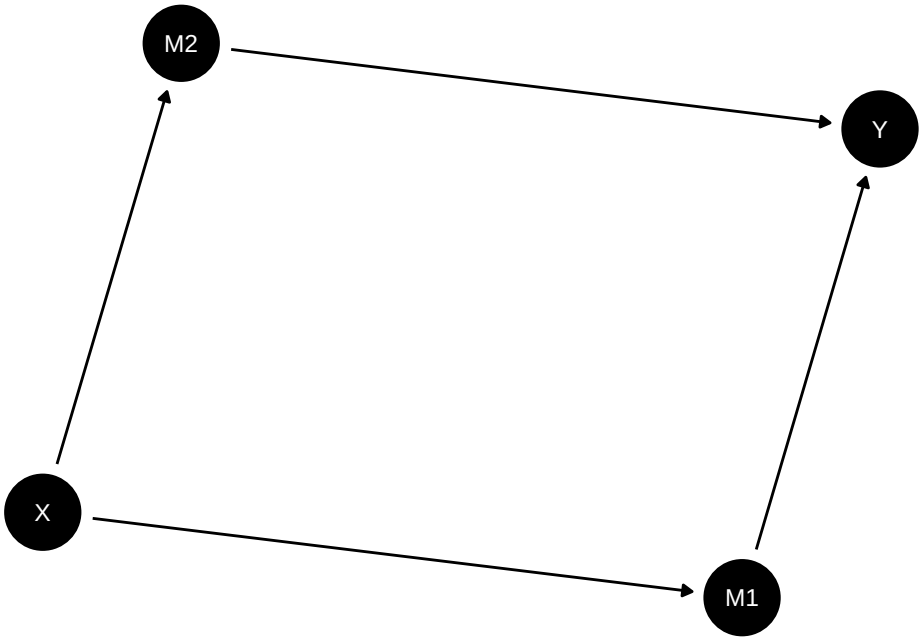


Table 10.1: Priors and parameters

Query	Using	Case.estimand	mean	sd	conf.low	conf.high
ate	parameters	FALSE	1.0000		1.0000	1.0000
via_M1	parameters	FALSE	1.0000		1.0000	1.0000
via_M2	parameters	FALSE	0.0000		0.0000	0.0000
ate	priors	FALSE	0.2230	0.1396	0.0300	0.5470
via_M1	priors	FALSE	0.1131	0.1024	0.0034	0.3807
via_M2	priors	FALSE	0.1098	0.1005	0.0031	0.3696

Table 10.2: Inferences with 1000 observations; data on X, Y, only

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ate	-	posteriors	FALSE	0.9880	0.0049	0.9767	0.9955
via_M1	-	posteriors	FALSE	0.4905	0.2214	0.0518	0.9242
via_M2	-	posteriors	FALSE	0.4980	0.2213	0.0654	0.9362

We imagine that in truth X always causes Y and it does so via $M1$, though this is not known *ex ante*:

```
Q1 <- query_model(model,
  queries = list(ate = te("X", "Y"),
                 via_M1 = "(M1[X=1]>M1[X=0]) & (Y[M1=1]>Y[M1=0])",
                 via_M2 = "(M2[X=1]>M2[X=0]) & (Y[M2=1]>Y[M2=0])"),
  using = c("parameters", "priors"),
  expand_grid = TRUE)
```

Now suppose we have access to large X, Y, data.

We infer that that X certainly causes Y. But we are unsure about channels.

However, this changes dramatically with data on M_1 and M_2. Here we assume data on only 20 cases.

```
query_model(updated_mixed,
  queries = list(ate = te("X", "Y"),
                 via_M1 = "(M1[X=1]>M1[X=0]) & (Y[M1=1]>Y[M1=0])",
                 via_M2 = "(M2[X=1]>M2[X=0]) & (Y[M2=1]>Y[M2=0])"),
  using = c("posteriors"),
```

Table 10.3: Inferences with 1000 observations for X, Y, 20 observations for M1, M2

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ate	-	posteriors	FALSE	0.9862	0.0050	0.9750	0.9941
via_M1	-	posteriors	FALSE	0.9850	0.0057	0.9717	0.9937
via_M2	-	posteriors	FALSE	0.0012	0.0019	0.0000	0.0060

```
  expand_grid = TRUE)  |> kable(caption = "Inferences with 1000 observations for X, Y
```

The data now convinces us that X must work only through one channel. Thus, again, small within case data can dramatically alter conclusions from large N data when that data has little discriminatory power for the estimands of interest.

10.3 Nothing from nothing

Many of the models we have looked at—especially for process tracing—have a lot of structure, viz:

- conditional independence assumptions
- no confounding assumptions, and
- monotonicity assumptions, or other restrictions

What happens if you have none of these? Can access to observational data render clues meaningful for inferences on causal effects?

We show the scope for learning from a mediator for a "good case"—a world in which in fact (though unknown *ex ante* to the researcher):

- X causes Y through M
- X is a necessary condition for M and M is a sufficient condition for Y – and so Y is monotonic in X and
- there is no confounding

Here is the data:

```
data <- make_model("X -> M -> Y") |>
```

Table 10.4: Data contains strong correlations.

	X	M	Y
X	1.00	0.81	0.61
M	0.81	1.00	0.77
Y	0.61	0.77	1.00

```
        make_data(n = 2000,
                  parameters = c(.5, .5, .2, 0, .8, 0, 0, 0, .8, .2))
```

We imagine inferences are made starting from two types of model. In both we allow all possible links and we impose no restrictions on nodal types. Even though there are only three nodes, this model has 128 causal types ($2 \times 4 \times 16$). In addition:

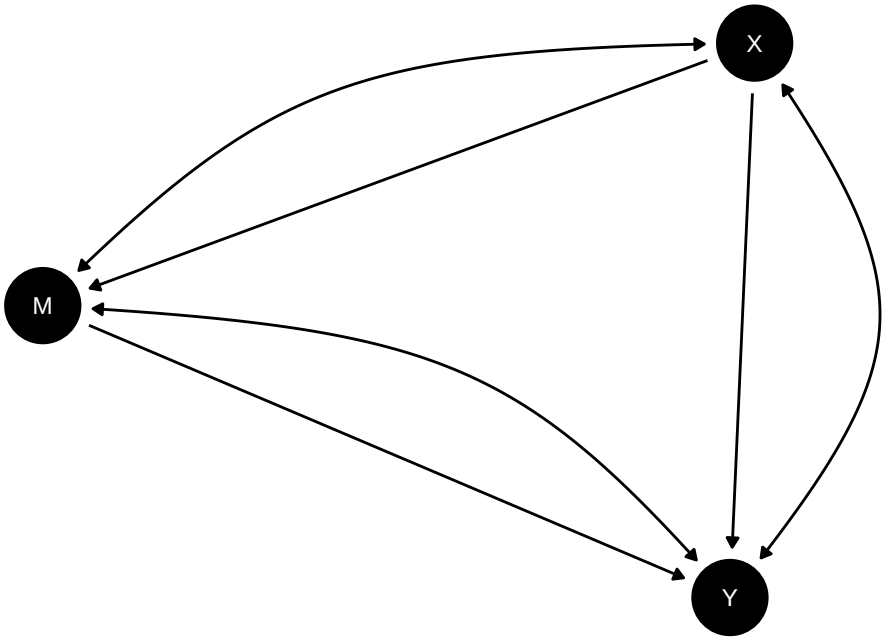
- In `model_1` we allow confounding between all pairs of nodes. This results in 127 free parameters.

- In `model_2` we assume that X is known to be randomized. There are now only 64 free parameters.

Here are the models:

```
model_1 <-
  make_model("X -> M -> Y <- X; X <-> M; M <->Y; X <-> Y")

model_2 <-
  make_model("X -> M -> Y <- X; M <->Y")
```


X
M
Y

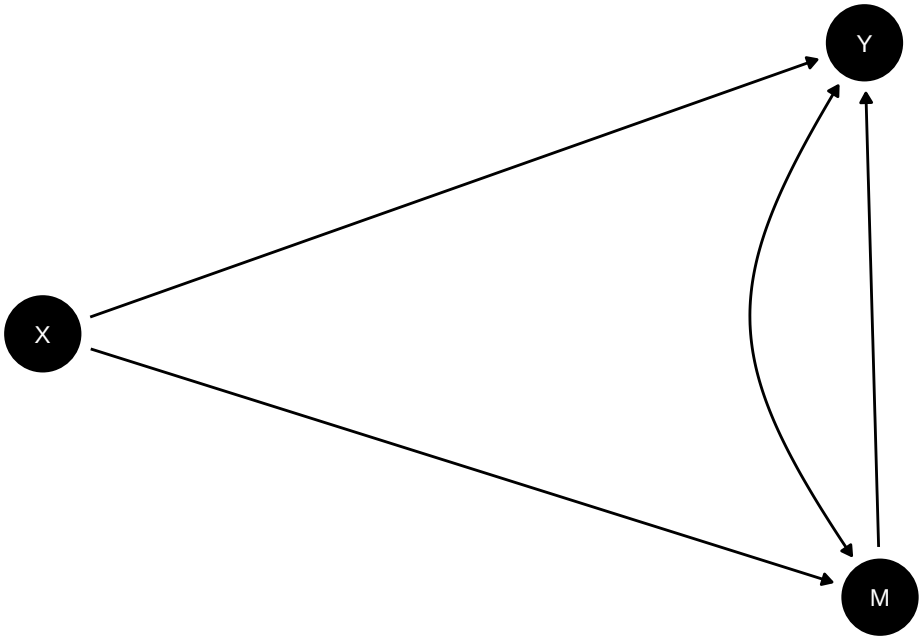

Y
X
M

Table 10.5: Can observation of large N data render mediator M informative for case level inference? Observational data.

Query	Given	Using	Case.estimand	mean	sd	conf.low
Q 1	X==1 & Y==1	posteriors	FALSE	0.4994	0.1581	0.1980
Q 1	X==1 & M==1 & Y==1	posteriors	FALSE	0.4993	0.1667	0.1815
Q 1	X==1 & M==0 & Y==1	posteriors	FALSE	0.5005	0.1380	0.2405
Q 1	X==1 & Y==1	priors	FALSE	0.5016	0.1042	0.2962
Q 1	X==1 & M==1 & Y==1	priors	FALSE	0.5018	0.1361	0.2374
Q 1	X==1 & M==0 & Y==1	priors	FALSE	0.5017	0.1346	0.2407

Table 10.6: Can observation of large N data render mediator M informative for case level inference? X randomized.

Query	Given	Using	Case.estimand	mean	sd	conf.low
Q 1	X==1 & Y==1	posteriors	FALSE	0.8519	0.0297	0.7959
Q 1	X==1 & M==1 & Y==1	posteriors	FALSE	0.8692	0.0320	0.8080
Q 1	X==1 & M==0 & Y==1	posteriors	FALSE	0.5294	0.1618	0.2134
Q 1	X==1 & Y==1	priors	FALSE	0.5019	0.1061	0.2953
Q 1	X==1 & M==1 & Y==1	priors	FALSE	0.4996	0.1350	0.2443
Q 1	X==1 & M==0 & Y==1	priors	FALSE	0.5032	0.1366	0.2432

After updating we query the models to see how inferences depend on M like this:

```
model_1 <- update_model(model_1, data, iter = 6000)

query_model(model_1,
            queries = "Y[X=1] > Y[X=0]",
            given = c("X==1 & Y==1", "X==1 & M==1 & Y==1"),
            using = c("priors", "posteriors"),
            expand_grid = TRUE)
```

We find that even with an auspicious monotonic data generating process in which M is a total mediator, M gives no traction on causal inference in Model 1 but it gives considerable leverage in Model 2: M is informative, especially if $M = 0$, when X is known to be randomized, but it provides essentially no

guidance if it is not.

Chapter 11

External validity and inference aggregation

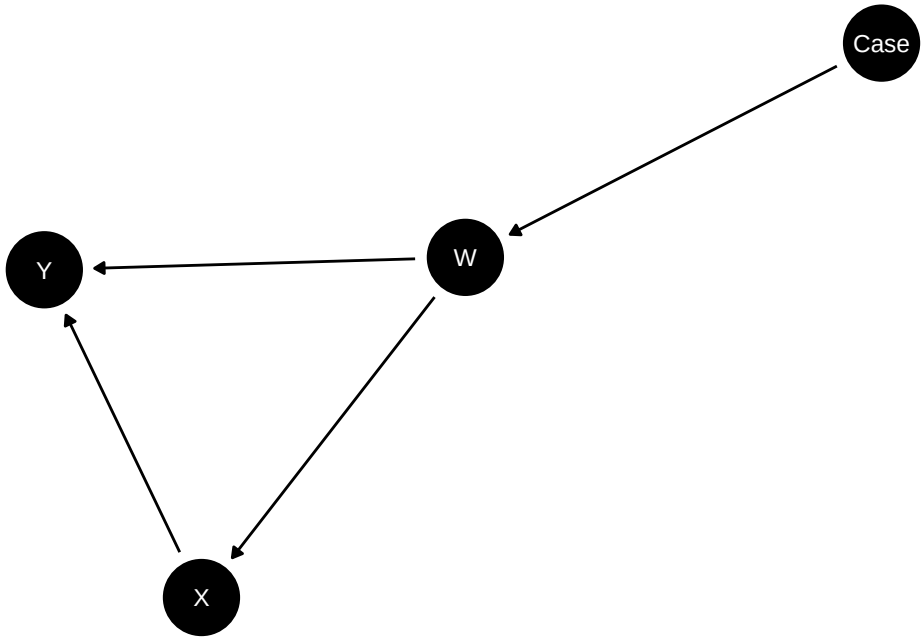
11.1 Transportation of findings across contexts

Say we study the effect of X on Y in case 0 (a country, for instance) and want to make inferences to case 1 (another country). Our problem however is that effects are heterogeneous and features that differ across units may be related both to treatment assignment, outcomes, and selection into the sample. This is the problem studied by Pearl and Bareinboim (2014). In particular Pearl and Bareinboim (2014) show for which nodes data is needed in order to "licence" external claims, given a model.

We illustrate with a simple model in which a confounder has a different distribution in a study site and a target site.

```
model <- make_model("Case -> W  -> X -> Y <- W") |>
  set_restrictions("W[Case = 1] < W[Case = 0]") |>
  set_parameters(statement = "X[W=1]>X[W=0]", parameters = 1/2)|>
  set_parameters(statement = complements("W", "X", "Y"), parameters = .17) |>
  set_parameters(statement = decreasing("X", "Y"), parameters = 0)

plot(model)
```



We start by checking some basic quantities in the priors and the posteriors, we will then see how we do with data.

```
query_model(model,
            queries = list(Incidence = "W==1",
                           ATE = "Y[X=1] - Y[X=0]",
                           CATE = "Y[X=1, W=1] - Y[X=0, W=1]"),
            given = c("Case==0", "Case==1"),
            using = c("priors", "parameters"), expand_grid = TRUE) |> kable()
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Incidence	Case==0	priors	FALSE	0.3321	0.2365	0.0130	0.8435
Incidence	Case==0	parameters	FALSE	0.3333		0.3333	0.3333
Incidence	Case==1	priors	FALSE	0.6710	0.2344	0.1745	0.9868
Incidence	Case==1	parameters	FALSE	0.6667		0.6667	0.6667
ATE	Case==0	priors	FALSE	-0.0006	0.1401	-0.2792	0.2771
ATE	Case==0	parameters	FALSE	0.3333		0.3333	0.3333
ATE	Case==1	priors	FALSE	-0.0024	0.1372	-0.2725	0.2639
ATE	Case==1	parameters	FALSE	0.5727		0.5727	0.5727
CATE	Case==0	priors	FALSE	-0.0012	0.1689	-0.3368	0.3329
CATE	Case==0	parameters	FALSE	0.8121		0.8121	0.8121
CATE	Case==1	priors	FALSE	-0.0012	0.1689	-0.3368	0.3329
CATE	Case==1	parameters	FALSE	0.8121		0.8121	0.8121

We see that the incidence of W as well as the ATE of X on Y is larger in case 1 than in case 0 (in parameters, though not in priors). However the effect of X on Y conditional on W is the same in both places.

We now update the model *using data on X and Y only from one case* (case 1) and data on W from both and check inferences on the other.

The function `make_data` lets us generate data like this by specifying a multistage data strategy:

```
data <- make_data(model, n = 1000,
                  vars = list(c("Case", "W"), c("X", "Y")),
                  probs = c(1,1),
                  subsets = c(TRUE, "Case ==1"))

transport <- update_model(model, data)

query_model(transport,
            queries = list(Incidence = "W==1",
                           ATE = "Y[X=1] - Y[X=0]",
                           CATE = "Y[X=1, W=1] - Y[X=0, W=1]"),
            given = c("Case==0", "Case==1"),
            using = c("posteriors", "parameters"), expand_grid = TRUE)
```

We do well in recovering the (different) effects both in the location we study and the one in which we do not. In essence querying the model for the out of

Table 11.1: Extrapolation when two sites differ on W and W is observable in both sites

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Incidence	Case==0	posteriors	FALSE	0.3318	0.0066	0.3188	0.3448
Incidence	Case==0	parameters	FALSE	0.3333		0.3333	0.3333
Incidence	Case==1	posteriors	FALSE	0.6769	0.0069	0.6632	0.6902
Incidence	Case==1	parameters	FALSE	0.6667		0.6667	0.6667
ATE	Case==0	posteriors	FALSE	0.3427	0.0115	0.3199	0.3649
ATE	Case==0	parameters	FALSE	0.3333		0.3333	0.3333
ATE	Case==1	posteriors	FALSE	0.5772	0.0091	0.5591	0.5943
ATE	Case==1	parameters	FALSE	0.5727		0.5727	0.5727
CATE	Case==0	posteriors	FALSE	0.7967	0.0089	0.7784	0.8140
CATE	Case==0	parameters	FALSE	0.8121		0.8121	0.8121
CATE	Case==1	posteriors	FALSE	0.7967	0.0089	0.7784	0.8140
CATE	Case==1	parameters	FALSE	0.8121		0.8121	0.8121

sample case requests a type of post stratification. We get the right answer, though as always this depends on the model being correct.

Had we attempted to make the extrapolation without data on W in country 1 we would get it wrong. In that case however we would also report greater posterior variance. The posterior variance here captures the fact that we know things could be different in country 1, but we don't know in what way they are different. Note that we get the CATE right since in the model this is assumed to be the same across cases.

11.2 Combining observational and experimental data

An interesting weakness of experimental studies is that, by dealing so effectively with self selection into treatment, they limit our ability to learn about self selection. Often however we want to know what causal effects would be specifically for people that would take up a treatment in non experimental settings. This kind of problem is studied for example by Knox et al. (2019).

Table 11.2: Extrapolation when two sites differ on W and W is not observable in target country.

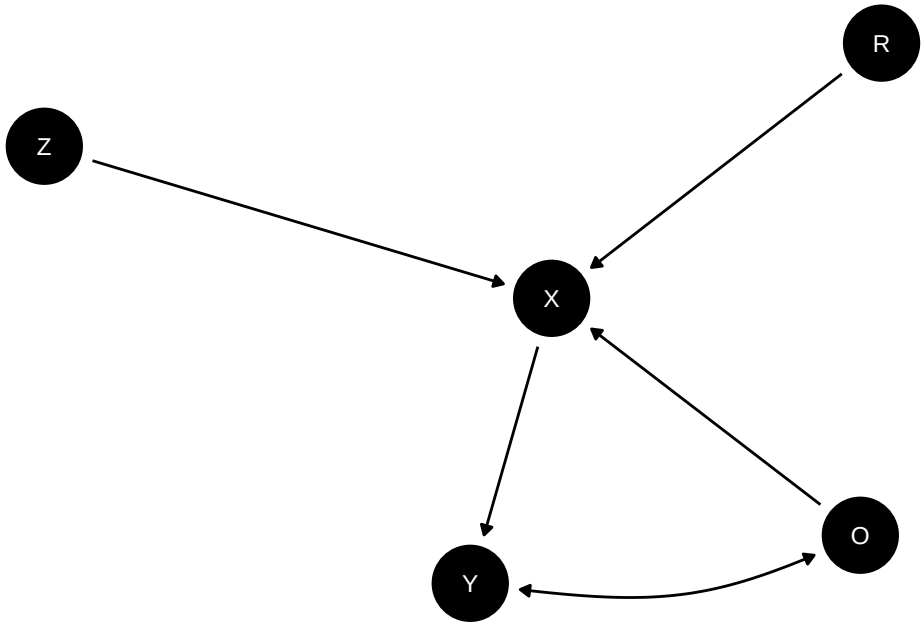
Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
Incidence	Case==0	posteriors	FALSE	0.3318	0.0067	0.3191	0.3450
Incidence	Case==0	parameters	FALSE	0.3333		0.3333	0.3333
Incidence	Case==1	posteriors	FALSE	0.6685	0.0065	0.6556	0.6810
Incidence	Case==1	parameters	FALSE	0.6667		0.6667	0.6667
ATE	Case==0	posteriors	FALSE	0.3400	0.0113	0.3178	0.3615
ATE	Case==0	parameters	FALSE	0.3333		0.3333	0.3333
ATE	Case==1	posteriors	FALSE	0.5800	0.0089	0.5618	0.5972
ATE	Case==1	parameters	FALSE	0.5727		0.5727	0.5727
CATE	Case==0	posteriors	FALSE	0.8163	0.0086	0.7990	0.8325
CATE	Case==0	parameters	FALSE	0.8121		0.8121	0.8121
CATE	Case==1	posteriors	FALSE	0.8163	0.0086	0.7990	0.8325
CATE	Case==1	parameters	FALSE	0.8121		0.8121	0.8121

A causal model can encompass both experimental and observational data and let you answer this kind of question. To illustrate, imagine that node R indicates whether a unit was assigned to be randomly assigned to treatment assignment ($X = Z$ if $R = 1$) or took on its observational value ($X = O$ if $R = 0$). We assume the exclusion restriction that entering the experimental sample is not related to Y other than through assignment of X.

```
model <- make_model("R -> X -> Y; O -> X <- Z; O <-> Y") |>

    set_restrictions("(X[R=1, Z=0]!=0) | (X[R=1, Z=1]!=1) | (X[R=0, O=0]!=0) | (X[R=0

plot(model)
```



The parameter matrix has just one type for X since X really operates here as a kind of switch, inheriting the value of Z or O depending on R. Parameters allow for complete confounding between O and Y but Z and Y are unconfounded.

We imagine parameter values in which there is a true .2 effect of X on Y. However the effect is positive (.6) for cases in which $X = 1$ under observational assignment but negative (-.2) for cases in which $X = 0$ under observational assignment.

```
model <- model |>
    set_parameters(node = "Y", given = "0.0", parameters = c(.8, .2,  0,  0)) |>
    set_parameters(node = "Y", given = "0.1", parameters = c( 0,  0, .6, .4))
```

To parse this expression: we allow different parameter values for the four possible nodal types for Y when $O = 0$ and when $O = 1$. When $O = 0$ we have $(\lambda_{00} = .8, \lambda_{10} = .2, \lambda_{01} = 0, \lambda_{11} = 0)$ which implies a negative treatment effect and many $Y = 0$ observations. When $O = 1$ we have $(\lambda_{00} = 0, \lambda_{10} = 0, \lambda_{01} = .6, \lambda_{11} = .4)$ which implies a positive treatment

Table 11.3: estimands

Query	Given	Using	Case.estimand	mean
ATE	-	parameters	FALSE	0.2
ATE	R==0	parameters	FALSE	0.2
ATE	R==1	parameters	FALSE	0.2

Table 11.4: priors

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ATE	-	priors	FALSE	0.003	0.2579	-0.5155	0.5266
ATE	R==0	priors	FALSE	0.003	0.2579	-0.5155	0.5266
ATE	R==1	priors	FALSE	0.003	0.2579	-0.5155	0.5266

effect and many $Y = 1$ observations.

The estimands:

The priors:

Data:

```
data <- make_data(model, n = 800)
```

The true effect is .2 but naive analysis on the observational data would yield a strongly upwardly biased estimate.

```
estimatr::difference_in_means(Y~X, data = filter(data, R==0))
```

The `CausalQueries` estimates are:

Table 11.5: Inferences on the ATE from differences in means

	Estimate	Std. Error	t value	Pr(>|t|)	CI Lower	CI Upper	DF
X	0.808	0.029	27.57	0	0.75	0.865	181

```
posterior <- update_model(model, data)
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ATE	-	posteriors	FALSE	0.176	0.031	0.1142	0.2367
ATE	R==0	posteriors	FALSE	0.176	0.031	0.1142	0.2367
ATE	R==1	posteriors	FALSE	0.176	0.031	0.1142	0.2367

Much better.

This model used both the experimental and the observational data. It is interesting to ask whether the observational data improved the estimates from the experimental data or did everything depend on the experimental data?

To see, lets do updating using experimental data only:

```
updated_no_0 <- update_model(model, dplyr::filter(data, R==1))
```

Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ATE	-	posteriors	FALSE	0.1535	0.0414	0.0698	0.2325
ATE	R==0	posteriors	FALSE	0.1535	0.0414	0.0698	0.2325
ATE	R==1	posteriors	FALSE	0.1535	0.0414	0.0698	0.2325

In this case we get a tightening of posterior variance and a more accurate result when we use the observational data but the gains are relatively small. They would be smaller still if we had more data, in which case inferences from the experimental data would be more accurate still.

In both cases the estimates for the average effect in the randomized and the observationally assigned group are the same. This is how it should be since these are, afterall, randomly assigned into these groups.

Heterogeneity in this model lies between those that are in treatment and those that are in control *in the observational* sample. We learn nothing about this heterogeneity from the experimental data alone but we learn a lot from the mixed model, picking up the strong self selection into treatment in the observational group:

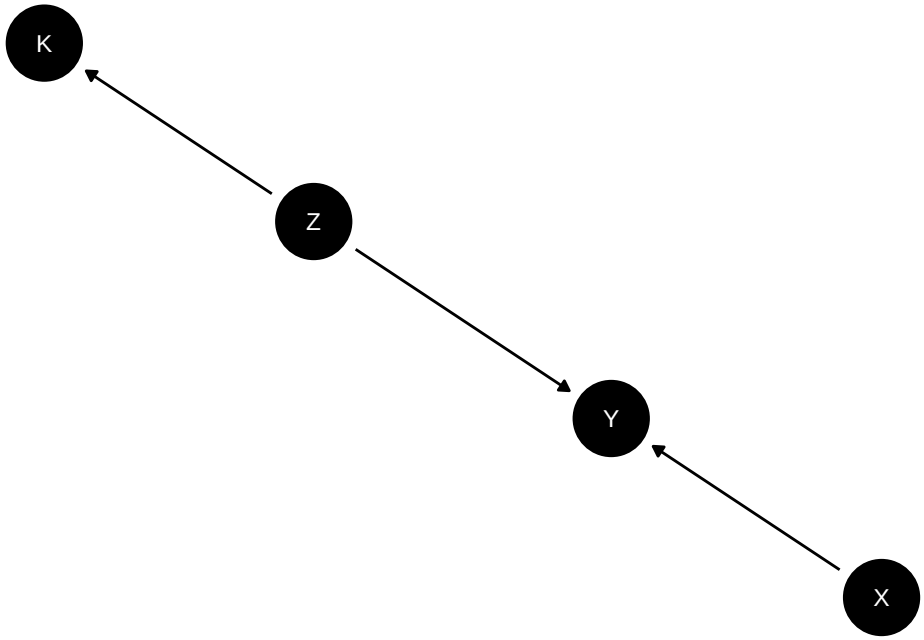
Query	Given	Using	Case.estimand	mean	sd	conf.low	conf.high
ATE	R==1 & X==0	posteriors	FALSE	0.1760	0.0310	0.1142	0.2367
ATE	R==1 & X==1	posteriors	FALSE	0.1760	0.0310	0.1142	0.2367
ATE	R==0 & X==0	posteriors	FALSE	-0.2148	0.0271	-0.2664	-0.1622
ATE	R==0 & X==1	posteriors	FALSE	0.6079	0.0497	0.5074	0.7049

11.3 A jigsaw puzzle: Learning across populations

Consider a situation in which we believe the same model holds in multiple sites but in which learning about the model requires combining data about different parts of the model from multiple studies.

```
model <-
  make_model("X -> Y <- Z -> K") |>
  set_parameters(
    statement = "(Y[X=1, Z = 1] > Y[X=0, Z = 1])", parameters = .24) |>
  set_parameters(
    statement = "(K[Z = 1] > K[Z = 0])", parameters = .85)


plot(model)
```



We imagine we have access to three types of data;

1. Study 1 is a factorial study examining the joint effects of X and Z on Y, K is not observed
2. Study 2 is an RCT looking at the relation between Z and K. X and Y are not observed.
3. Study 3 is an experiment looking at the effects of X on Y, ancillary data on context, K is collected but Z is not observed

```
df <- make_data(model, 300, using = "parameters") |>

      mutate(study = rep(1:3, each = 100),
             K = ifelse(study == 1, NA, K),
             X = ifelse(study == 2, NA, X),
             Y = ifelse(study == 2, NA, Y),
             Z = ifelse(study == 3, NA, Z)
             )
```

Tables 11.6 - 11.8 show conditional inferences for the probability that X caused Y in $X = Y = 1$ cases conditional on K for each study, analyzed

Table 11.6: Clue is uninformative in Study 1

Given	Case.estimand	mean	sd	conf.low	conf.high
X == 1 & Y == 1 & K == 1	FALSE	0.4963	0.1588	0.1999	0.8024
X == 1 & Y == 1 & K == 0	FALSE	0.4972	0.1544	0.2082	0.7914

Table 11.7: Clue is also uninformative in Study 2 (factorial)

Given	Case.estimand	mean	sd	conf.low	conf.high
X == 1 & Y == 1 & K == 1	FALSE	0.5698	0.1295	0.3047	0.8135
X == 1 & Y == 1 & K == 0	FALSE	0.5681	0.1292	0.3046	0.8108

individually

In no case is K informative. In study 1 data on K is not available, in study 2 it is available but researchers do not know, quantitatively, how it relates to Z. In the third study the Z, K relationship is well understood but the joint relation between Z, X, and Y is not understood.

Table 11.9 shows the inferences when the data are combined with joint updating across all parameters.

Here fuller understanding of the model lets researchers use information on K to update on values for Z and in turn update on the likely effects of X on Y. Rows 3-6 highlight that the updating works through inferences on Z and there if Z is known, as in Study 2, there are no additional gains from knowledge of K.

The collection of studies collectively allow for inferences that are not possible from any one study.

Table 11.8: Clue is also uninformative in Study 3 (experiment studying K)

Given	Case.estimand	mean	sd	conf.low	conf.high
X == 1 & Y == 1 & K == 1	FALSE	0.4984	0.1611	0.1975	0.8062
X == 1 & Y == 1 & K == 0	FALSE	0.5000	0.1624	0.1840	0.8100

Table 11.9: Clue is informative after combining studies linking K to Z and Z to Y

Given	Case.estimand	mean	sd	conf.low	con
X == 1 & Y == 1 & K == 1	FALSE	0.8395	0.0587	0.7105	0
X == 1 & Y == 1 & K == 0	FALSE	0.4627	0.0759	0.3098	0
X == 1 & Y == 1 & K == 1 & Z == 1	FALSE	0.8864	0.0560	0.7511	0
X == 1 & Y == 1 & K == 0 & Z == 1	FALSE	0.8864	0.0560	0.7511	0
X == 1 & Y == 1 & K == 1 & Z == 0	FALSE	0.4535	0.0774	0.2985	0
X == 1 & Y == 1 & K == 0 & Z == 0	FALSE	0.4535	0.0774	0.2985	0

Part IV

Notation

Chapter 12

Notation and syntax

12.1 Notation

A guide to key notation used in the `CausalQueries` package:

term	symbol	meaning
nodal type	θ^X	The way that a node responds to the values of its parents. Example: θ^Y_{10}, written `Y10`: Y takes the value 1 if $X = 0$ and 0 if $X = 1$. For interpretation of syntax, see `make_model("X->Y") %>% get_nodal_types`
causal type	θ	A causal type is a concatenation of nodal types, one for each node. Example: $(\theta^X_0, \theta^Y_{00})$, written `X0.Y00`, is a type that has $X = 0$ and $Y = 0$ no matter what the value of X. See: `make_model("X->Y") %>% get_causal_types`

term	symbol	meaning
parameter	λ	An unknown quantity of interest that generates the probability of causal types. In models without confounding parameters are the probabilities of nodal types. In models with confounding, parameters are the conditional probabilities of causal types. Example: `X.0`= $\lambda_0^X = \Pr(\theta^X = \theta_0^X)$ See `make_model("X->Y") %>% get_parameters`
data event type		A possible set of values on all nodes (including, possibly, NAs). Example: `X0Y1` $= (X = 0, Y = 1)$
event probability	w	The probability of a data event type. Example: $w_{00} = \Pr(X = 0, Y = 1)$
Dirichlet priors	alpha, α	Non negative numbers used to characterize a prior distribution over a simplex. The implied mean is the normalized vector $\mu = \alpha / \sum(\alpha)$ and the variance is $\mu(1-\mu)/(1+\sum \alpha)$.
parameter matrix	P	A matrix of 0s and 1s that maps from parameters (rows) to causal types (columns). Example: see `make_model("X->Y") %>% get_parameter_matrix`
ambiguities matrix	A	A matrix of 0s and 1s that maps from causal types (rows) to data types (columns). Example: see `make_model("X->Y") %>% get_ambiguities_matrix`
families matrix	E	A matrix of 0s and 1s that maps from partial data events to complete data events. Example: see `make_model("X->Y") %>% get_data_families`
data strategy	S	A plan indicating for how many nodes different types of data will be gathered. See `? make_data`

12.1.1 Parents, children, and all that

The causal models analyzed by `CausalQueries` all involve *directed* edges between nodes, with cycles over nodes precluded. In turn this implies a partial ordering over nodes which motivates some useful terminology:

- X is a *parent* of Y if a change in X sometimes induces a change in Y even when all other nodes are fixed. On the graph, there's an arrow from X to Y.
- Y is a *child* of X if a change in X sometimes induces a change in Y even when all other nodes are fixed. On the graph, there's an arrow from X to Y.
- A is an *ancestor* of B by analogy: a parent is an ancestor and any parent of an ancestor is an ancestor. On the graph there is a chain of arrows pointing in one direction going from A to B. Similarly for *descendant*.

You should find that the package complains if you try to specify a cyclical graph.

12.2 Causal syntax

Both model definition and model querying requires a simple way to make arbitrary causal statements.

- You can query **observational quantities**. For instance:
  - `make_model("X->Y") %>% get_query_types("Y==1")` Figures out the types that produce $Y = 1$ absent any interventions.
- You can query **experimental quantities**. For instance:
  - `make_model("X->Y") %>% get_query_types("Y[X=1]==1")` figures out the types that produce $Y = 1$ when X is set to 1.
  - `make_model("X->M->Y") %>% get_query_types("Y[X=1]>Y[X=0]")` figures out the types that have a positive causal effect.
- You can make queries with **complex counterfactuals**. For instance:
  - `make_model("X->M->Y") %>% get_query_types("Y[M=M[X=0], X=1]==1")` looks for the types for which $Y = 1$ when $X = 1$ and M is held constant at the value it would take if X were 0.
- You can use **wild cards** and AND or OR operators. For instance:
  - `make_model("X->Y") %>% get_query_types("(Y[X = .]==1)",`

`join_by = "|"`) figures out the causal types for which $Y = 1$ for some value of X.
  - `make_model("X->Y") %>% get_query_types("(Y[X = .]==1)", join_by = "&")` figures out the causal types for which $Y = 1$ for *all* values of X.
  - Note that the use of "." as a wild card also requires placing the causal statement in parentheses, as in these examples.
- You can make **conditional queries**. For instance, conditioning on observational or counterfactual quantities:
  - `make_model("X->Y") %>% query_model("Y[X = 1] > Y[X = 0]", subset = "X==1 & Y==1")` asks what is the probability that X has a positive effect on Y given $X = Y = 1$.
  - `make_model("X->M->Y") %>% query_model("Y[X = 1] != Y[X = 0]", subsets = "M[X=1]==M[X=0]")` asks what is the probability that X matters for Y given X doesn't matter for M.

We provide a few helpers for common causal statements:

- `increasing("A", "B")` produces the statement `"B[A=1] > B[A=0]"`
- `decreasing("A", "B")` produces the statement `"B[A=1] < B[A=0]"`
- `interacts("A", "B", "C")` produces the statement `"((C[A =1, B = 1]) - (C[A = 0, B = 1])) != ((C[A =1, B = 0]) - (C[A = 0, B = 0]))"`
- `complements("A", "B", "C")` produces the statement `"((C[A =1, B = 1]) - (C[A = 0, B = 1])) > ((C[A =1, B = 0]) - (C[A = 0, B = 0]))"`
- `substitutes("A", "B", "C")` produces the statement `"((C[A =1, B = 1]) - (C[A = 0, B = 1])) < ((C[A =1, B = 0]) - (C[A = 0, B = 0]))"`

These helpers can be used for setting restrictions, setting confounds, defining priors or parameter values, or querying models.

For instance:

```
get_query_types(model = make_model("A -> B <- C"),
         query = substitutes("A", "C", "B"))
```

```
Causal types satisfying query's condition(s)
```

```
 query =  ((B[A = 1, C = 1]) - (B[A = 0, C = 1])) < ((B[A = 1, C = 0]) - (B[A = 0, C

A0.C0.B0100  A1.C0.B0100
A0.C1.B0100  A1.C1.B0100
A0.C0.B0010  A1.C0.B0010
A0.C1.B0010  A1.C1.B0010
A0.C0.B0110  A1.C0.B0110
A0.C1.B0110  A1.C1.B0110
A0.C0.B1110  A1.C0.B1110
A0.C1.B1110  A1.C1.B1110
A0.C0.B0111  A1.C0.B0111
A0.C1.B0111  A1.C1.B0111


 Number of causal types that meet condition(s) =  20
 Total number of causal types in model =  64
```

Bibliography

Bareinboim, E. and Pearl, J. (2016). Causal inference and the data-fusion problem. *Proceedings of the National Academy of Sciences*, 113(27):7345–7352.

Bennett, A. and Checkel, J. T. (2015). *Process tracing*. Cambridge University Press.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).

Collier, D. (2011). Understanding process tracing. *PS: Political Science & Politics*, 44(04):823–830.

Cowell, R. G., Dawid, P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic networks and expert systems*. Springer.

Dawid, A. P. (2011). The role of scientific and statistical evidence in assessing causality. *Perspectives on causation*, pages 133–147.

Dawid, P., Humphreys, M., and Musio, M. (2019). Bounding causes of effects with mediators. *arXiv preprint arXiv:1907.00399*.

Elwert, F. and Winship, C. (2014). Endogenous selection bias: The problem of conditioning on a collider variable. *Annual review of sociology*, 40:31–53.

Frangakis, C. E. and Rubin, D. B. (2002). Principal stratification in causal inference. *Biometrics*, 58(1):21–29.

Hall, N. (2004). Two concepts of causation. *Causation and counterfactuals*, pages 225–276.

Humphreys, M. and Jacobs, A. M. (2015). Mixing methods: A bayesian approach. *American Political Science Review*, 109(04):653–673.

Humphreys, M. and Jacobs, A. M. (2023). *Integrated Inferences*. Cambridge University Press.

Knox, D., Yamamoto, T., Baum, M. A., and Berinsky, A. J. (2019). Design, identification, and sensitivity analysis for patient preference trials. *Journal of the American Statistical Association*, pages 1–27.

Manski, C. F. (1995). *Identification problems in the social sciences*. Harvard University Press.

Pearl, J. (2009). *Causality*. Cambridge university press.

Pearl, J. and Bareinboim, E. (2014). External validity: From do-calculus to transportability across populations. *Statistical Science*, 29(4):579–595.

Pearl, J. and Mackenzie, D. (2018). *The book of why: the new science of cause and effect*. Basic Books.

Poirier, D. J. (1998). Revising beliefs in nonidentified models. *Econometric Theory*, 14(4):483–509.

Tian, J. and Pearl, J. (2000). Probabilities of causation: Bounds and identification. *Annals of Mathematics and Artificial Intelligence*, 28(1-4):287–313.

Van Evera, S. (1997). *Guide to methods for students of political science*. Cornell University Press, Ithaca, NY.