Uncertain Reasoning and Data Mining

Master in Engineering

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

October 22, 2021

Course Information

Instructors:

- Marco Scutari (scutari@idsia.ch) for the Uncertainty Reasoning module;
- Dario Azzimonti (dario.azzimonti@idsia.ch) for the Data Mining module.

Teaching assistant: Rafael Cabañas (rcabanas@idsia.ch)

Every week on Thursday:

12:45–14:15	Lecture
14:15–14:30	Coffee Break
14:40–15:15	Lecture
15:15–15:30	Coffee Break
15:30–17:00	Problem class (with Rafael)
17:00–18:00	Individual work with tutoring, review, more solved exercises (optional).

IMPORTANT: let Rafael and Marco/Dario know in advance if you are planning to attend the hours in the afternoon, especially if you would like to have active tutoring.

Examinations

The final grade is determined by:

- The mid-term written test (42%) in week 10
  - on the material from the "Uncertain Reasoning" module;
  - problems, questions about theory.
- The final written test (25%) in late January
  - on the material from the "Data Mining" module;
  - problems, questions about theory.
- The first homework assignment (21%)
  - hands-on modelling using the techniques learned in the "Uncertain Reasoning" module;
  - starts mid-October, short report to be submitted in 6 weeks.
- The second homework assignment (12%)
  - hands-on data analysis using the techniques learned in the "Data Mining" module;
  - starts mid-December, short report to be submitted in 4 weeks.

Homework assignments must be submitted as PDFs by email.

Regulations

- Attendance is mandatory.
- Written tests are mandatory as well:
  - there is no provision to retake the test,
  - the written test will be replaced by an oral examination on Uncertain Reasoning, Data Mining or both;
  - except if you are undertaking compulsory military service.
- There is no provision to retake the mid-term test if you fail.
- Use of laptop computers is encouraged to try stuff as it is being explained and keep lectures interactive.
- Please check your @supsi.ch email address at least daily as it will be used for personal communications.

Overview

module: Uncertain Reasoning

- Focus is on Bayesian networks.
- What are they?
- Why do we need them?
- Are they popular, are they used in practice?
- All programming will be in R.

module: Data mining

- Focus is on Pattern classification.
- What is it?
- Why do we need it?
- Is it popular, used in practice?
- All programming will be in Python.

Why data mining together with Bayesian networks?

Modules and Lectures Plan

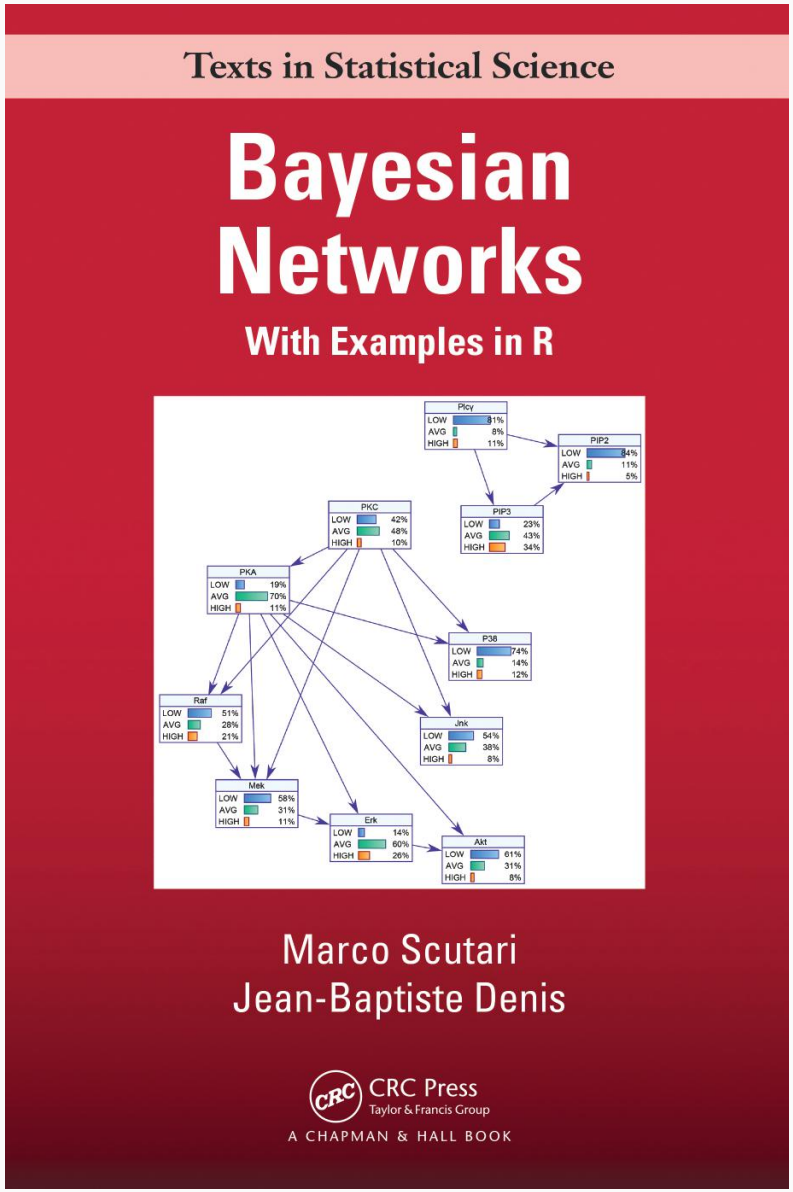
Week 1	Uncertain Reasoning	Probability Fundamentals
		Brief Introduction to R
Week 2	Uncertain Reasoning	Introduction to Machine Learning
Week 3	Uncertain Reasoning	Introduction to Bayesian Networks
Week 4	Uncertain Reasoning	Constructing a Bayesian Network
Week 5	Uncertain Reasoning	
Week 6	Uncertain Reasoning	
Week 7	Uncertain Reasoning	Inference
Week 8	Uncertain Reasoning	
Week 9	Uncertain Reasoning	Learning the Parameters
Week 10	Uncertain Reasoning	Mid-Term Written Test
Week 11	Data Mining	Introduction to Classification
		Decision Trees
Week 12	Data Mining	Model Evaluation
Week 13	Data Mining	Cross-Validation, k-Nearest Neighbours
Week 14	Data Mining	Naïve Bayes, Ensemble Methods Basics
Week 15	Data Mining	Advanced Topics

(The topic/week allocation is very tentative!)

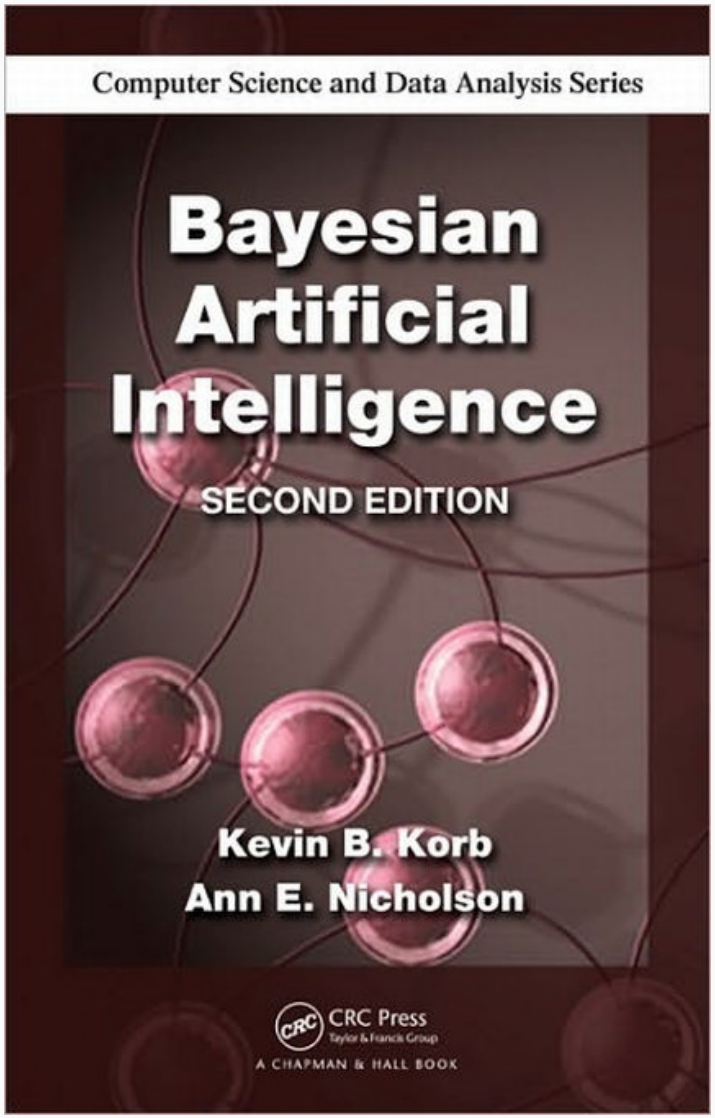
Reference Material

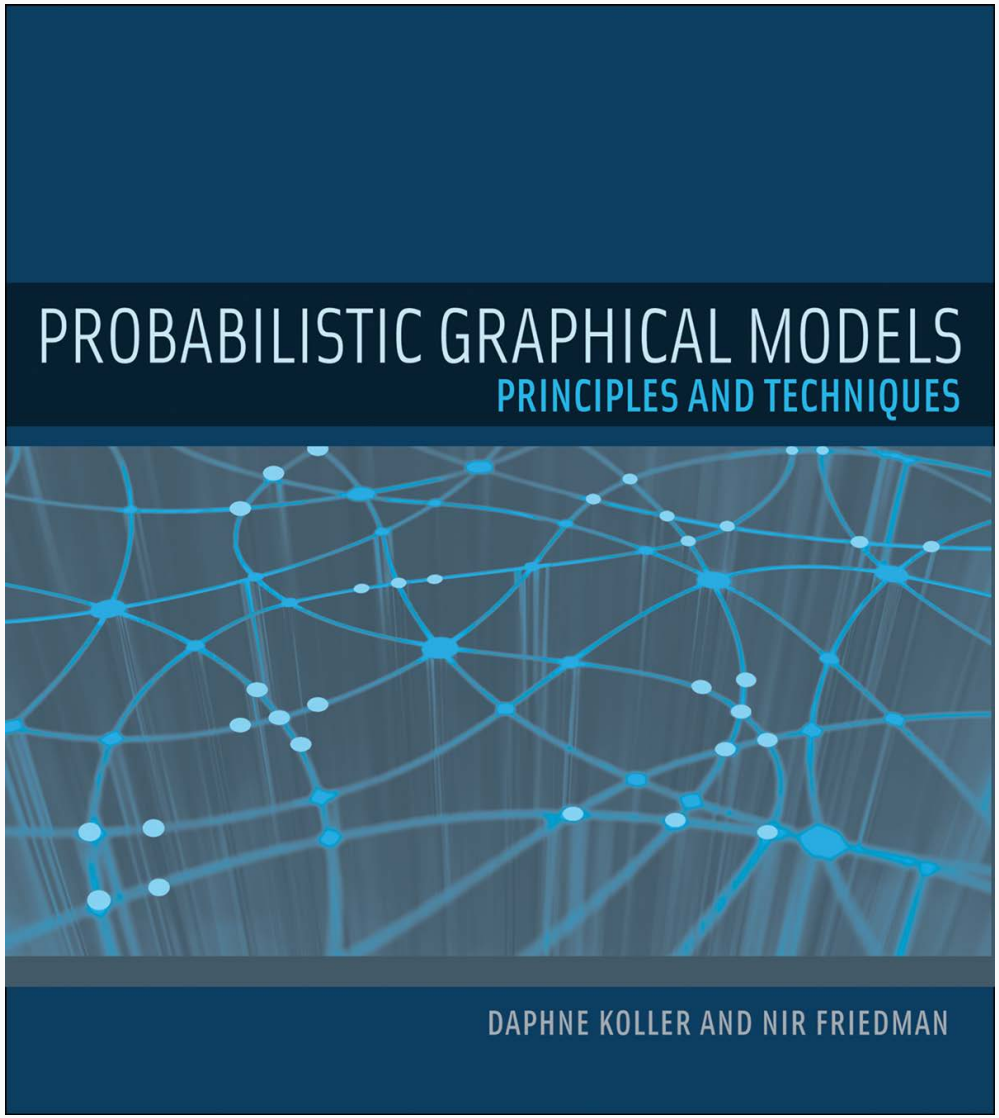
This is the reference textbook for Bayesian networks, from yours truly. It includes plenty of examples and code, going beyond the content of this module.

Introductory material on R including worksheets are available to work on in your own time courtesy of the “Statistical Programming” course I used to teach.



More Material on Bayesian Networks





Probability Fundamentals

Reasoning under Uncertainty

Reasoning under uncertainty means making rational decisions even when there is not enough information to prove that an action will work.

1. We have a set of events that may or may not happen and that are related to some phenomenon we are trying to model. (Will it rain tonight?)
2. We assign probabilities to those events, putting into numbers your knowledge. (It is 75% likely to rain tonight.)
3. We use those probabilities to answer any questions we may have on that phenomenon and to make a decision to take some action (or not) depending on the answer. (I should take an umbrella!)

But what is a probability?

Frequentist Probability

This is the classic definition from the 18th-19th century: probability is the relative frequency with which an event occurs over a large number of trials.

Gauss (1777–1855)

Laplace (1749–1827)

Bernoulli (1655–1705)

Frequentist Probability: a Polling Example

What is the probability that someone in the world reads books "every day or most days"?

Frequentist Probability: a Polling Example

The frequentist answer from the GfK poll is

$$\mathrm{P}(\text{reads every day or on most days}) = \frac{\text{number of people who read every day or on most days}}{\text{number of polled people}}$$

which gives

$$\mathrm{P}(\text{reads every day or on most days}) = \frac{6600}{22000} = 0.30$$

with the numbers in the previous slide.

If the number of people in the poll is large enough, we can take the probability to be a good approximation of the probability that someone reads books “every day or most days”.

Polling in R

1. We create a vector with the four reading habits covered in the poll.

```
reading.habits =
  c("every day or most days", "at least once a week",
    "at least once a month", "less than once a month")
```

2. We sample 22000 people.

```
poll = sample(reading.habits, size = 22000,
          prob = c(0.30, 0.29, 0.17, 0.24), replace = TRUE)
head(poll)
```

```
[1] "at least once a month"  "at least once a month"
[3] "every day or most days" "at least once a month"
[5] "less than once a month" "at least once a week"
```

Polling in R

3. We compute how many people in the poll have each specific reading habit (the absolute frequencies).

```
table(poll)
```

```
poll
 at least once a month   at least once a week
                  3691                   6320
every day or most days less than once a month
                  6671                   5318
```

4. We compute the corresponding relative frequencies.

```
table(poll) / sum(table(poll))
```

```
poll
 at least once a month   at least once a week
                 0.168                  0.287
every day or most days less than once a month
                 0.303                  0.242
```

Polling: Large Enough?

How many people should be polled to get an accurate probability? The larger the better! Ideally, we would ask every person in the world and get the exact value, but we can get pretty close with many fewer people than that.

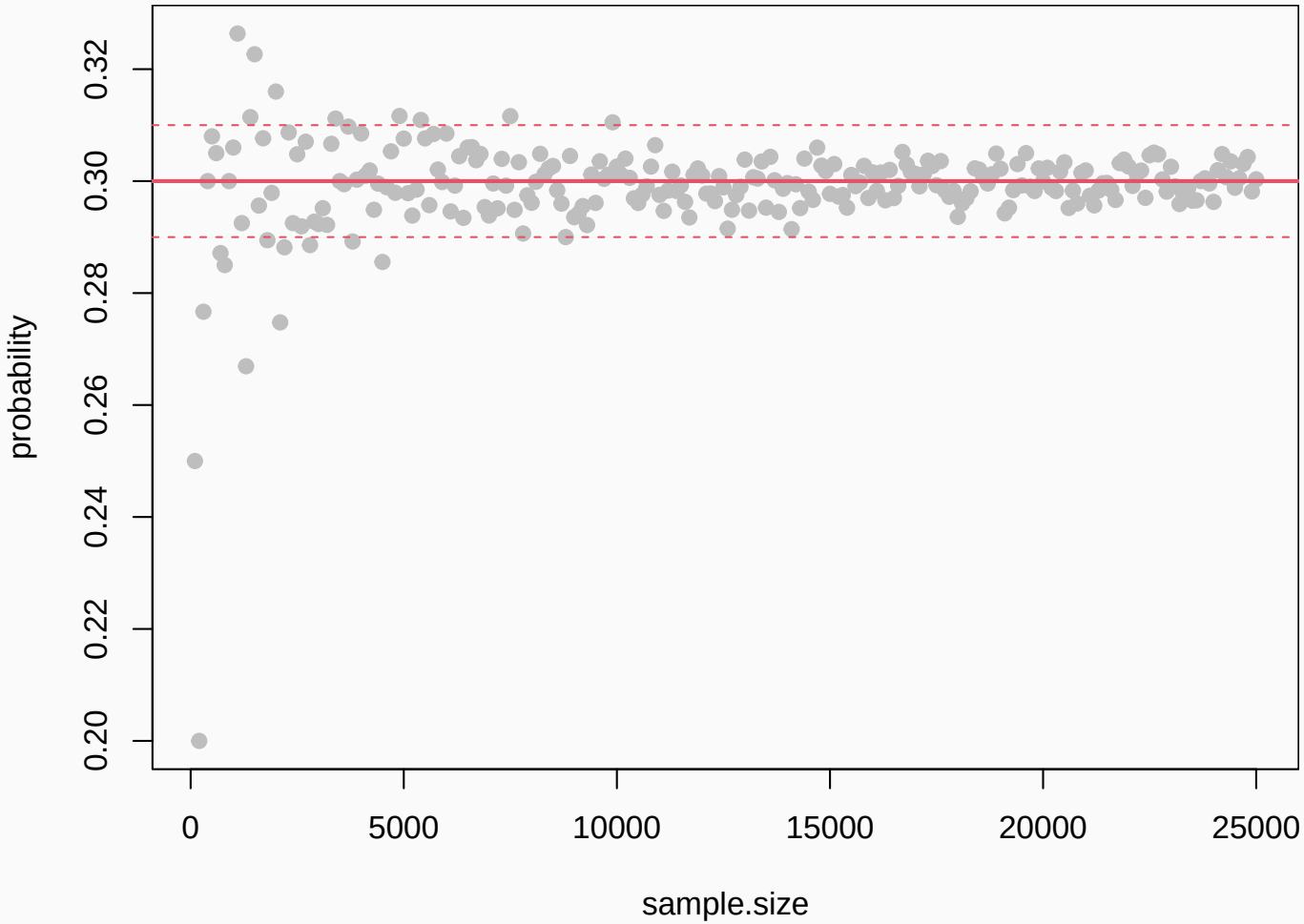
```
sample.size = seq(from = 100, to = 25000, by = 100)
probability = numeric(length(sample.size))
for (i in seq_along(sample.size)) {

  poll = sample(reading.habits, size = sample.size[i],
           prob = c(0.30, 0.29, 0.17, 0.24), replace = TRUE)
  probability[i] = prop.table(table(poll))["every day or most days"]

}#FOR
```

Polling: Large Enough?

```
par(mar = c(5, 5, 0, 0))
plot(probability ~ sample.size, pch = 19, col = "grey")
abline(h = 0.30 + c(-0.01, 0, +0.01), col = 2, lwd = c(1, 2, 1),
  lty = c(2, 1, 2))
```



Subjective Probability

This is a more modern definition from the 20th century: probability is a measure of the degree of belief of an individual assessing a particular phenomenon.

De Finetti (1906–1985)

Ramsey (1903–1930)

This idea that probability is a measure of our belief in an event happening is crucial to Bayesian statistics and modern data analysis. Bayesian statistics is founded on the idea that we update our beliefs as we observe the phenomenon we are modelling.

1. We have some belief about the probability of some events (our prior probabilities).
2. We observe some events, we do not observe others (our evidence).
3. We update our beliefs to incorporate the new evidence we have observed (our posterior probabilities)

In other words, prior + evidence = posterior.

Polling, Revisited

Switzerland is not listed in the GfK poll (shame!), so we decide to run our own poll of Swiss residents to find out the probability of people reading "every day or most days". Before we run the poll, we expect that the attitudes toward reading should be similar to those in Germany, France and Italy.

Hence our prior probabilities are:

	Germany	France	Italy	Average
"every day or most days"	0.25	0.27	0.30	0.2733
"at least once a week"	0.25	0.21	0.26	0.2400
"at least once a month"	0.18	0.19	0.19	0.1867
"less than once a month"	0.32	0.33	0.25	0.2300

Polling, Revisited in R

1. We store the prior probabilities computed from 7500 people interviewed in Germany, France and Italy in the GfK poll.

```
prior = c("every day or most days" = 0.2733,
          "at least once a week" = 0.24,
          "at least once a month" = 0.1867,
          "less than once a month" = 0.23)
prior
```

```
  every day or most days   at least once a week
                   0.273                   0.240
   at least once a month less than once a month
                   0.187                   0.230
```

2. We run our poll over 15000 Swiss residents.

```
evidence = c("every day or most days" = 4500,
             "at least once a week" = 3750,
             "at least once a month" = 4500,
             "less than once a month" = 2250)
sum(evidence)
```

```
  [1] 15000
```

Polling, Revisited in R

3. We compute the probabilities from the poll using their frequentist estimates.

```
prop.table(evidence)
```

```
  every day or most days   at least once a week
                    0.30                    0.25
   at least once a month less than once a month
                    0.30                    0.15
```

4. We combine the prior with the evidence, weighting them by the size of the respective polls, to obtain the posterior probabilities.

```
w = sum(evidence) / (sum(evidence) + sum(table(poll)))
posterior = w * prior + (1 - w) * prop.table(evidence)
posterior
```

```
  every day or most days   at least once a week
                   0.290                   0.246
   at least once a month less than once a month
                   0.258                   0.180
```

Evidence versus Prior: Strong Enough?

Intuitively, we would like that the more evidence we have the less our prior beliefs should matter: the more we directly observe a phenomenon, the less our previous expectations matter.

	prior	posterior	evidence
“every day or most days”	0.273	0.290	0.30
“at least once a week”	0.240	0.246	0.25
“at least once a month”	0.187	0.258	0.30
“less than once a month”	0.230	0.180	0.15

Hence, the larger our Swiss poll, the larger the weight w and the more the posterior will shift from the prior to the frequentist estimate. At some point we gather enough evidence that the posterior is no longer relevant at all.

Running Larger and Larger Polls

```
swiss.probabilities = prop.table(evidence)
poll.size = c(10, 20, 50, 100, 200, 500, 1000, 2000, 5000,
              10000, 20000, 50000, 100000, 8 * 10^6)
posterior = numeric(length(poll.size))

for (i in seq_along(poll.size)) {

  # run the poll.
  poll = sample(reading.habits, size = poll.size[i],
           prob = swiss.probabilities, replace = TRUE)

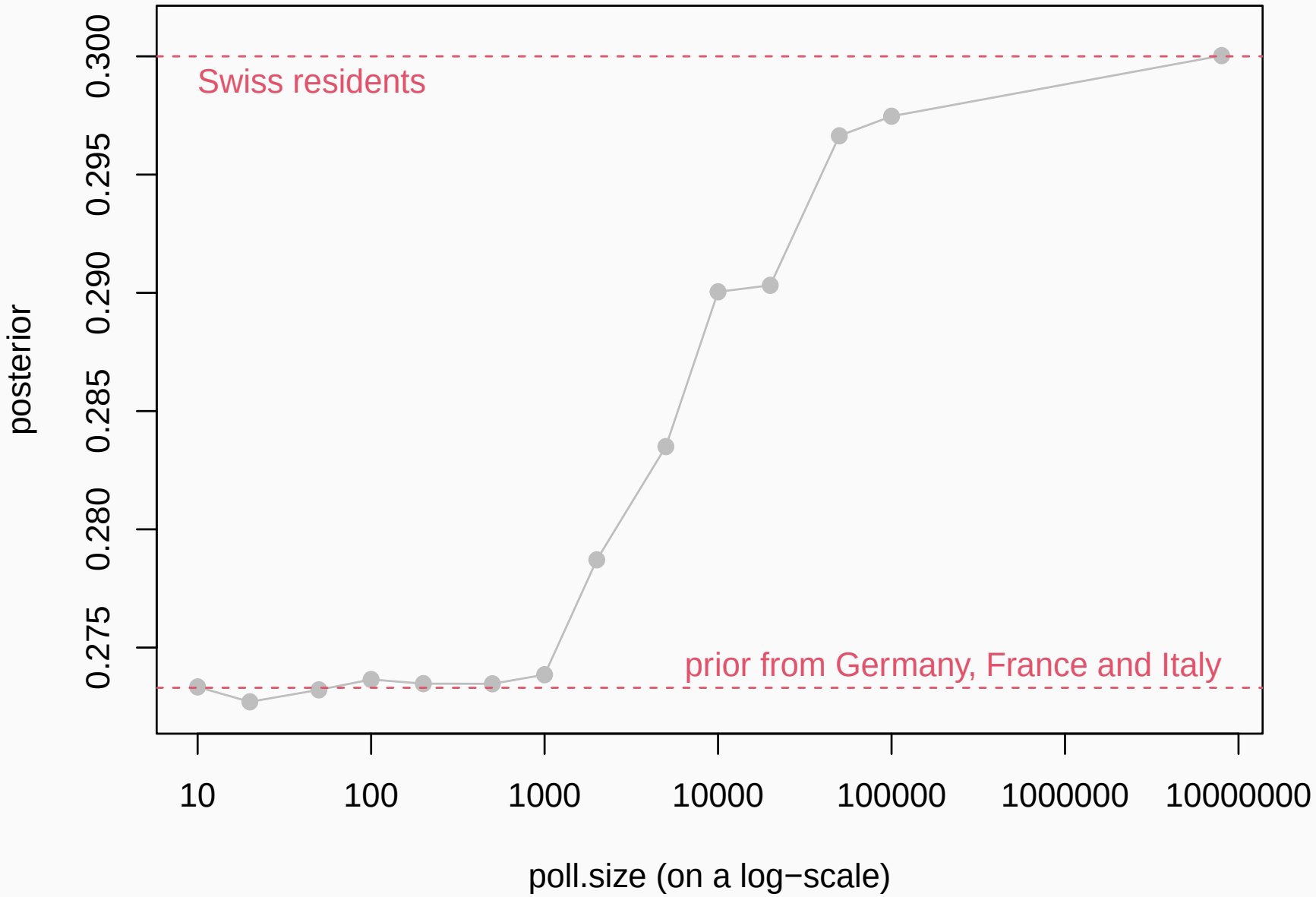
  # compute the frequentist probabilities from the evidence.
  evidence = table(poll)

  # compute the weight.
  w = 7500 / (7500 + poll.size[i])

  # combine prior and evidence into the posterior using the weight.
  posterior[i] = w * prior["every day or most days"] +
                   (1 - w) * prop.table(evidence)["every day or most days"]

}#FOR
```

Moving Away from the Prior



One more take on probability is that it encodes how uncertain we are about whether an event will happen or not. If we had perfect knowledge (in a "God is all-knowing" sense) of the phenomenon we are modelling, for any specific person we could say

$$\mathrm{P}(\text{“reads every day or most days”}) = 0 \qquad \text{or} \qquad = 1;$$

but we do not know each Swiss resident personally, so we do not have enough information to tell for sure which reading habits that person has.

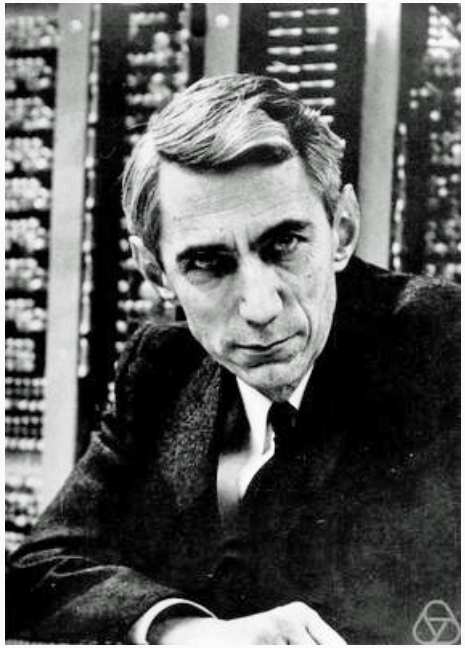
On the other hand, if we have no information at all we may say that as far as we know all reading habits are equally likely:

$$\mathrm{P}(\text{“reads every day or most days”}) =$$
$$\mathrm{P}(\text{“at least once a week”}) = \mathrm{P}(\text{“at least once a month”}) =$$
$$\mathrm{P}(\text{“less than once a month”}) = 0.25$$

The relationship between probability, information and uncertainty is the core of information theory and is centred on the entropy function:

$$\mathrm{H}(E_1, E_2, ...) = -\mathrm{P}(E_1)\log\mathrm{P}(E_1) - \mathrm{P}(E_2)\log\mathrm{P}(E_2) - ...$$

which is designed to:



Shannon
(1916–2001)

- be zero when there is no uncertainty;
- take its maximum value when there is maximum uncertainty, that is, when all events are equally probable;
- to be always positive.

Uncertainty in Poll Results

- No uncertainty: everybody has the same habits, say $\mathrm{P}(\text{“reads at least once a week”}) = 1$.

$$\mathrm{H}(\mathsf{Poll}) = -0\log 0 - 1\log 1 - 0\log 0 - 0\log 0 = 0$$

- Maximum uncertainty: all reading habits are equally probable.

$$\mathrm{H}(\mathsf{Poll}) = -0.25\log 0.25 \times 4 = 1.3863$$

- Somewhere in between: the results of the Swiss poll.

$$\mathrm{H}(\mathsf{Poll}) = -0.30\log 0.30 - 0.25\log 0.25 \\ - 0.30\log 0.30 - 0.15\log 0.15 = 1.3535$$

This association between probability and uncertainty is useful in choosing the prior distribution: since we have yet to observe the phenomenon we are modelling, we necessarily have an imperfect knowledge of what is going to happen.

In the worst case, we can choose the non-informative prior which assigns the same probability to every event (that is, every reading habit has probability 0.25). The downside is that it will take a much larger sample (that is, polling more people) to get the same posterior probabilities than with an informative prior (in which probabilities are closer to 0 and 1).

Informative vs Non-Informative Prior

```
prior = c("every day or most days" = 0.25,
          "at least once a week" = 0.25,
          "at least once a month" = 0.25,
          "less than once a month" = 0.25)
posterior2 = numeric(length(poll.size))

for (i in seq_along(poll.size)) {

  # run the poll.
  poll = sample(reading.habits, size = poll.size[i],
           prob = swiss.probabilities, replace = TRUE)

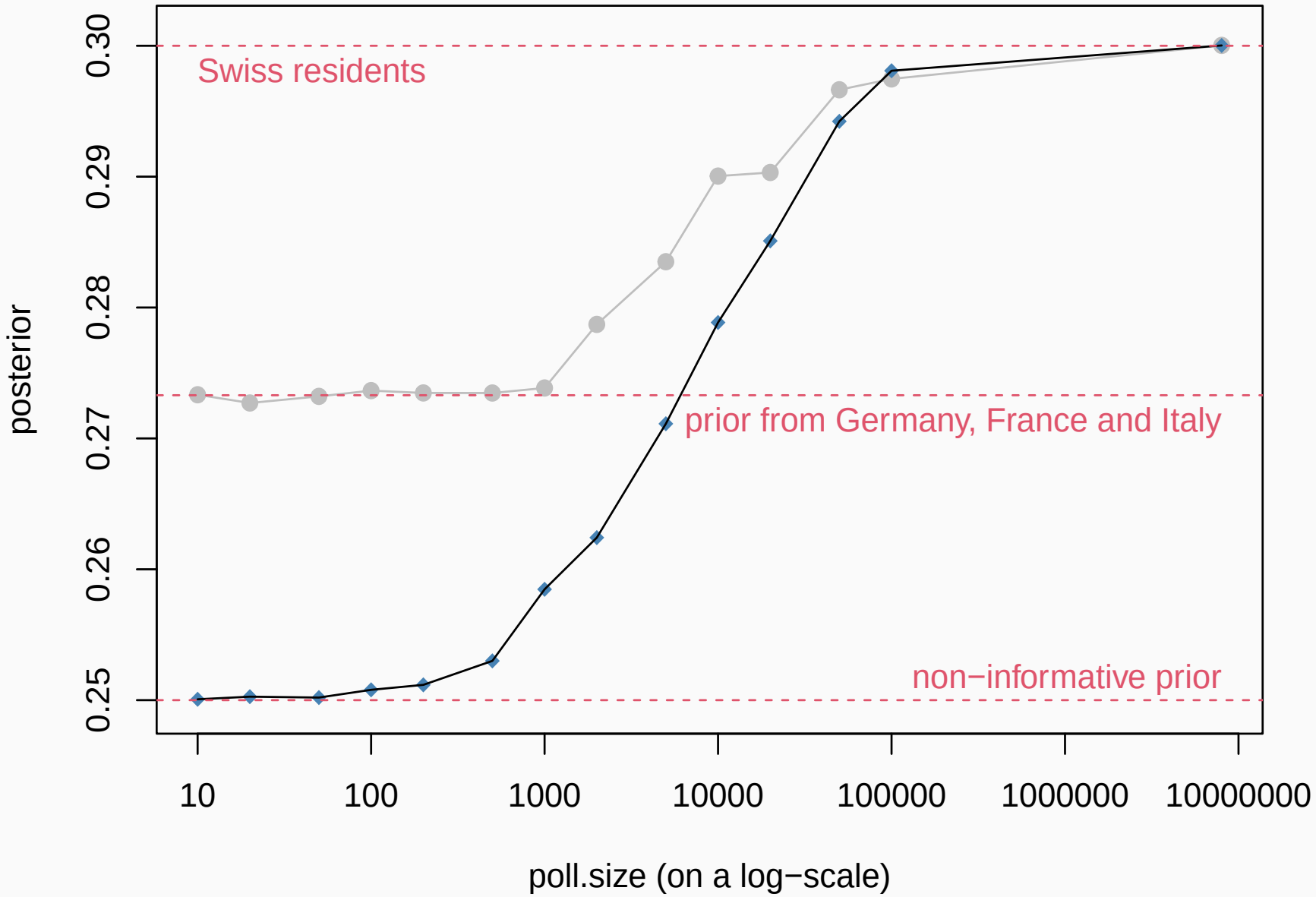
  # compute the frequentist probabilities from the evidence.
  evidence = table(poll)

  # compute the weight.
  w = 7500 / (7500 + poll.size[i])

  # combine prior and evidence into the posterior using the weight.
  posterior2[i] = w * prior["every day or most days"] +
                  (1 - w) * prop.table(evidence)["every day or most days"]

}#FOR
```

Moving Away from the Prior



A random variable X is a function that:

- starting from a sample space S of possible outcomes of a random phenomenon;
- it considers an event E that is a subset of the sample space;
- and assigns a probability to it.

In the GfK poll:

- the sample space is the set of the reading habits: “every day or most days”, “at least once a week”, “at least once a month”, “less than once a month”;
- the event is the reading habit of a Swiss resident;
- the function that assigns the probabilities is

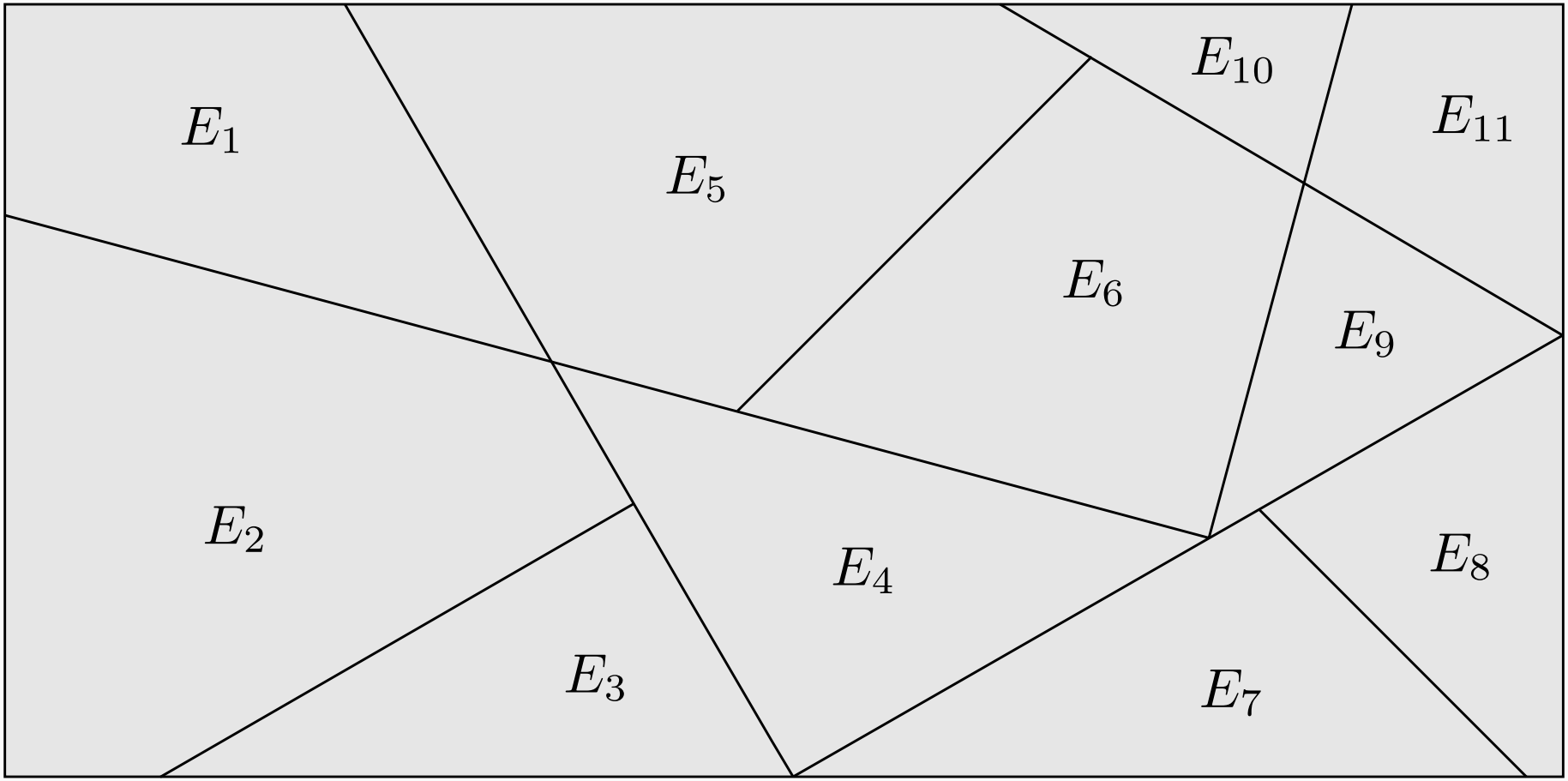
$$\mathrm{P}(X) = \begin{cases} 0.30 & \text{if reading “every day or most days”} \\ 0.25 & \text{if reading “at least once a week”} \\ 0.30 & \text{if reading “at least once a month”} \\ 0.15 & \text{if reading “less than once a month”} \end{cases}.$$

Basic Probability Axioms

Notation: by $\mathrm{P}(E)$ we mean $\mathrm{P}(X = E)$.

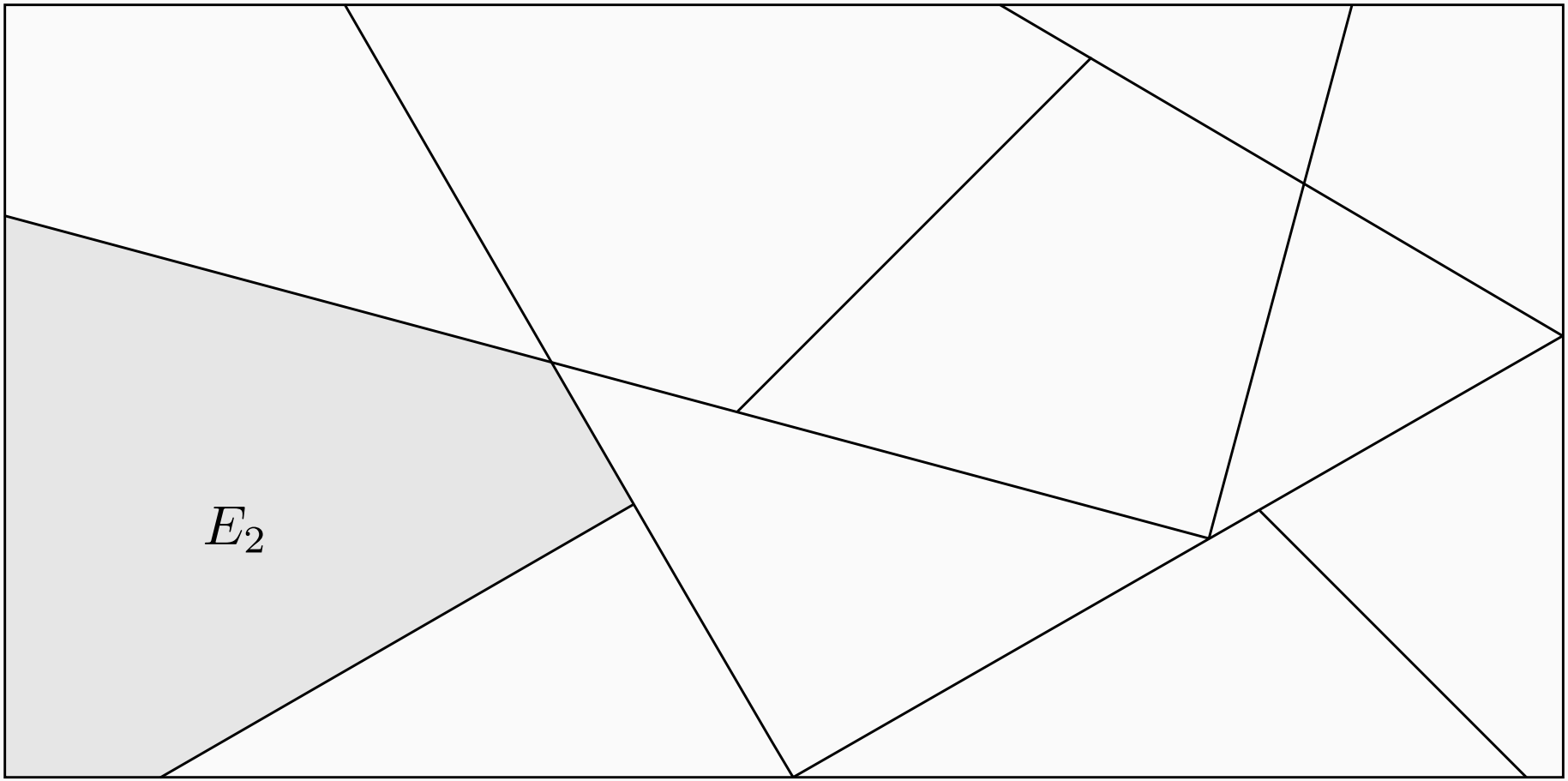
Mathematical notation	Meaning
$\mathrm{P}(S) = 1$	All possible events in the sample space combined have a probability of 1.
$\mathrm{P}(E) \in [0, 1]$	Each event has a probability between 0 and 1 (included).
$\mathrm{P}(E_1 \cup E_2) = \mathrm{P}(E_1) + \mathrm{P}(E_2)$ if $E_1 \cap E_2 = \varnothing$	Non-overlapping events have independent probabilities.
$\mathrm{P}(E_1 \cup E_2) = \mathrm{P}(E_1) + \mathrm{P}(E_2) - \mathrm{P}(E_1 \cap E_2)$ if $E_1 \cap E_2 \neq \varnothing$	Overlapping events share common probability, which should be taken into account.

Basic Probability Axiom #1



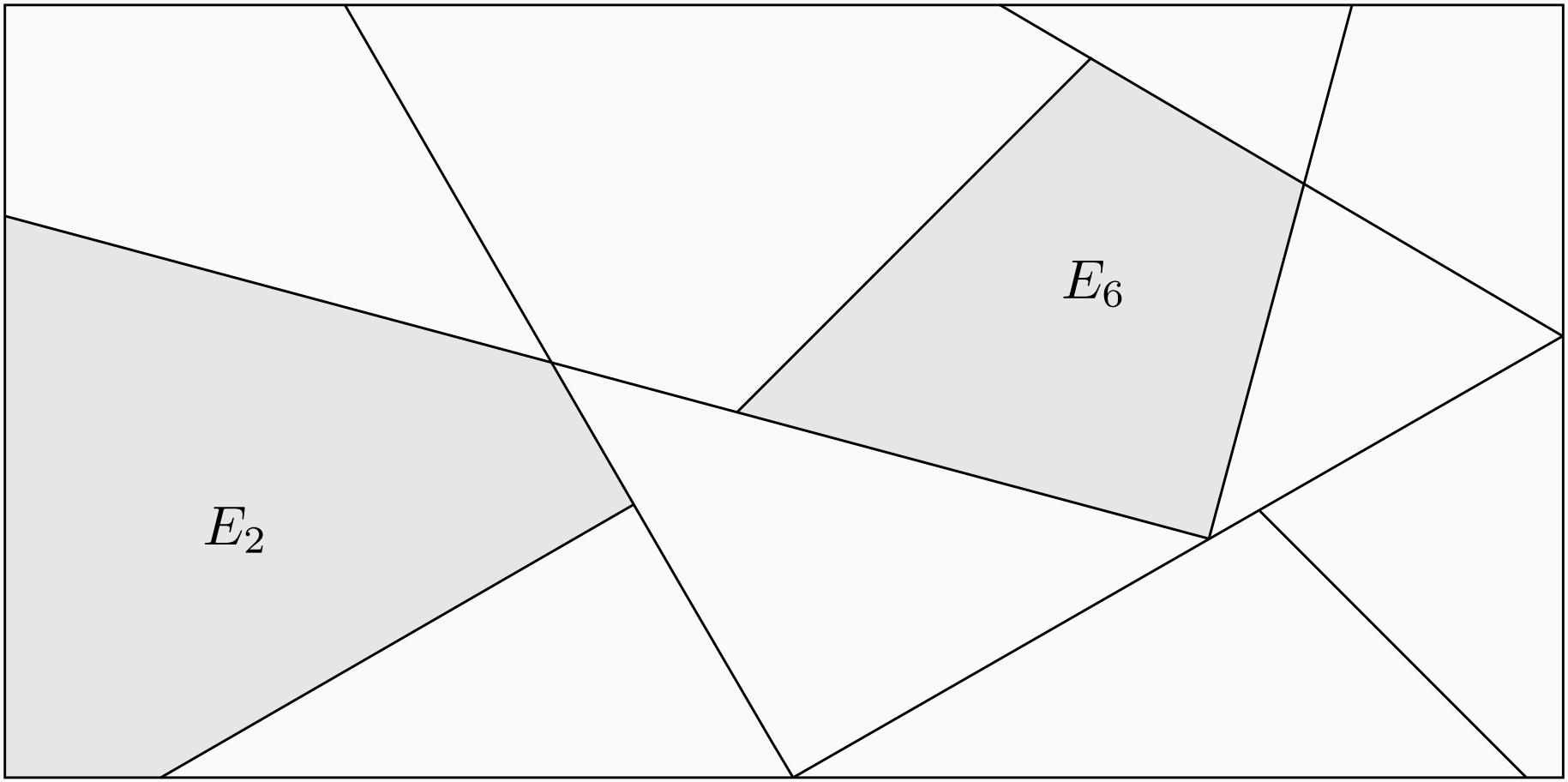
$$\mathrm{P}(S) = \mathrm{P}\left(\bigcup_{i=1}^{11} E_i\right) = 1.$$

Basic Probability Axiom #2



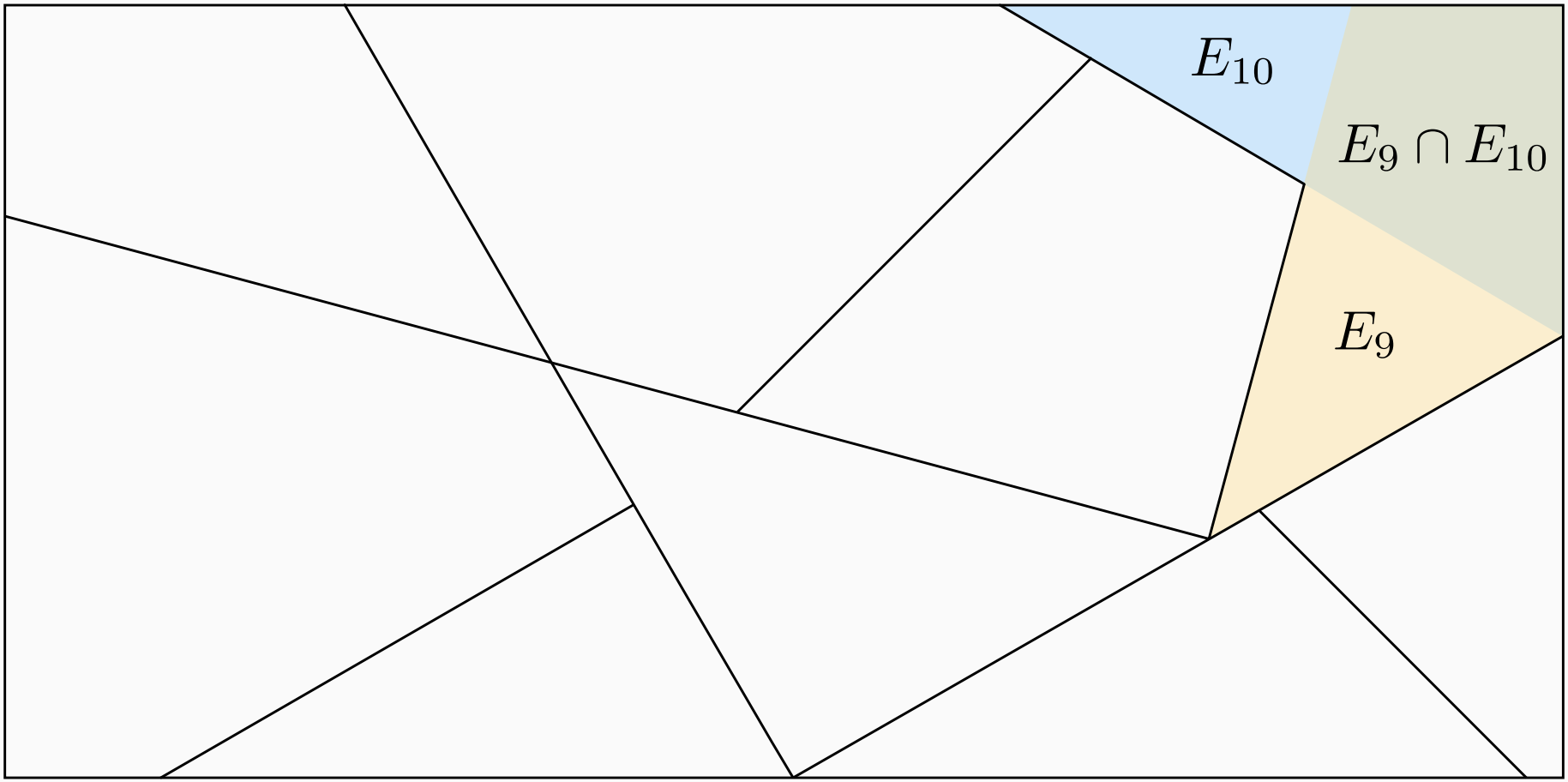
any $\mathrm{P}(E_i) \in [0, 1]$ because $\mathrm{P}(E_i) < 0$ would make it more-than-impossible, and because $\mathrm{P}(E_i) \leqslant \mathrm{P}(S) = 1$.

Basic Probability Axiom #3



$$\mathrm{P}(E_2 \cup E_6) = \mathrm{P}(E_2) + \mathrm{P}(E_6) \text{ if } E_2 \cap E_6 = \varnothing.$$

Basic Probability Axiom #4



$\mathrm{P}(E_9 \cup E_{10}) = \mathrm{P}(E_9) + \mathrm{P}(E_{10}) - \mathrm{P}(E_9 \cap E_{10})$ if $E_1 \cap E_2 \neq \varnothing$, because with $\cup$ we are asking about the probability of either E_9 or E_{10}; if both can happen we count the probability $E_9 \cap E_{10}$ twice.

Conditional Probabilities

Following up from axioms #3 and #4, we can define the concept of conditional probability: the probability of an event happening in a particular context. The notation for two events is

$$\mathrm{P}(E_1 \mid E_2) = \frac{\mathrm{P}(E_1 \cap E_2)}{\mathrm{P}(E_2)}$$

which can be read as "given that E_2 happened, what about the probability of E_1?". The notation is the same for random variables:

$$\mathrm{P}(X_1 \mid X_2) = \frac{\mathrm{P}(X_1 \cap X_2)}{\mathrm{P}(X_2)}$$

which means

$$\mathrm{P}(X_1 = E_i \mid X_2 = E_j) = \frac{\mathrm{P}(X_1 = E_i \cap X_2 = E_j)}{\mathrm{P}(X_2 = E_j)}$$

for any event E_i for X_1 and any event E_j for X_2.

More Conditional Probabilities

If we flip the definition of conditional probability around, we get

$$\mathrm{P}(X_1 \cap X_2) = \mathrm{P}(X_1 \mid X_2)\,\mathrm{P}(X_2)$$

which is commonly written as

$$\mathrm{P}(X_1, X_2) = \mathrm{P}(X_1 \mid X_2)\,\mathrm{P}(X_2)$$

meaning that the joint probability of X_1 and X_2 is equal to the probability of (some event for) X_1 given (another event for) X_2 and the probability of (the same event) for X_2.

For more than two variables, we can extend the formula to

$$\begin{aligned}\mathrm{P}(X_1, X_2, X_3) &= \mathrm{P}(X_1 \mid X_2, X_3)\,\mathrm{P}(X_2, X_3) \\ &= \mathrm{P}(X_1 \mid X_2, X_3)\,\mathrm{P}(X_2 \mid X_3)\,\mathrm{P}(X_3).\end{aligned}$$

Joint and Conditional Probabilities: An Example

In 1950 Doll and Hill investigated the link between smoking and lung cancer with data from 20 hospitals in London. For each of the 709 patients admitted, they recorded the smoking behaviour of a non-cancer patient at the same hospital of the same gender and within the same 5-year grouping on age.

```
counts = matrix(c(688, 21, 650, 59), nrow = 2, ncol = 2,
           dimnames = list("Smoker" = c("yes", "no"),
                           "Lung Cancer" = c("Case", "Control")))
counts
```

```
       Lung Cancer
Smoker Case Control
   yes  688     650
   no    21      59
```

(A smoker was defined as a person who had smoked at least one cigarette a day for at least a year.)

From Counts to Joint Probabilities

Firstly, we can estimate the joint probabilities of `Lung Cancer` and `Smoker` in a frequentist way using the relative frequencies.

```
joint.probs = prop.table(counts)
joint.probs
```

```
        Lung Cancer
Smoker    Case Control
   yes 0.4852  0.4584
   no  0.0148  0.0416
```

This is how we compute them manually:

Smoker	Lung Cancer	Count	divided by	Probability
yes	Case	688	/ 1418 =	0.485
no	Case	21	/ 1418 =	0.015
yes	Control	650	/ 1418 =	0.458
no	Control	59	/ 1418 =	0.042

From Joint to Conditional Probabilities

The conditional probabilities of Lung Cancer given Smoker are computed by normalising the rows of `joint.probs`.

```
conditional.probs = prop.table(joint.probs, margin = 1)
conditional.probs
```

```
        Lung Cancer
Smoker   Case Control
   yes 0.514   0.486
   no  0.263   0.738
```

Manually:

$$\mathrm{P}(\texttt{Lung Cancer} = \texttt{Case} \mid \texttt{Smoker} = \texttt{yes}) = 0.485/(0.485 + 0.458) = 0.514$$
$$\mathrm{P}(\texttt{Lung Cancer} = \texttt{Control} \mid \texttt{Smoker} = \texttt{yes}) = 0.458/(0.458 + 0.458) = 0.486$$
$$\mathrm{P}(\texttt{Lung Cancer} = \texttt{Case} \mid \texttt{Smoker} = \texttt{no}) = 0.015/(0.015 + 0.042) = 0.262$$
$$\mathrm{P}(\texttt{Lung Cancer} = \texttt{Control} \mid \texttt{Smoker} = \texttt{no}) = 0.042/(0.042 + 0.042) = 0.738$$

This is the probability of having lung cancer (or not) given that the patient was a smoker (or not).

From Joint to Conditional Probabilities

The conditional probabilities of Smoker given Lung Cancer are computed by normalising the columns of `joint.probs`.

```
conditional.probs = prop.table(joint.probs, margin = 2)
conditional.probs
```

```
        Lung Cancer
 Smoker    Case Control
    yes 0.9704  0.9168
    no  0.0296  0.0832
```

Manually:

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{yes} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case}) = 0.485/(0.485 + 0.015) = 0.97$$

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{no} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case}) = 0.015/(0.485 + 0.015) = 0.03$$

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{yes} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Control}) = 0.458/(0.458 + 0.042) = 0.917$$

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{no} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Control}) = 0.042/(0.458 + 0.042) = 0.083$$

This is the probability of being a smoker (or not) given that the patient has been admitted for lung cancer (or not).

As expected, conditional probabilities sum up to 1 along the columns (that is, the conditioning variable):

```
colSums(conditional.probs)
```

```
    Case Control
       1       1
```

If we write down the notation

$$\begin{aligned}\mathrm{P}(\mathsf{Smoker} = \mathsf{yes} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case}) + \\ \mathrm{P}(\mathsf{Smoker} = \mathsf{no} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case}) = \\ \mathrm{P}(\mathsf{Smoker} = \mathsf{yes} \cup \mathsf{no} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case})\end{aligned}$$

because of axiom #2 (yes and no are disjoint), and

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{yes} \cup \mathsf{no} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case}) = 1$$

because of axiom #1 (yes and no make up the whole sample space).

Marginals and Probability Tables

If we sum up on the rows or on the columns of `joint.probs` we get the marginal probabilities for Lung Cancer and Smoker.

```
marginal.smoker = margin.table(joint.probs, margin = 1)
marginal.smoker
```

```
Smoker
   yes     no
0.9436 0.0564
```

```
marginal.lung.cancer = margin.table(joint.probs, margin = 2)
marginal.lung.cancer
```

```
Lung Cancer
   Case Control
    0.5     0.5
```

Manually:

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{yes}) = \mathrm{P}(\mathsf{Smoker} = \mathsf{yes}, \mathsf{Lung\ \ Cancer} = \mathsf{Case}) +$$
$$\mathrm{P}(\mathsf{Smoker} = \mathsf{yes}, \mathsf{Lung\ \ Cancer} = \mathsf{Control}) = 0.485 + 0.458 = 0.944$$

$$\mathrm{P}(\mathsf{Smoker} = \mathsf{no}) = \mathrm{P}(\mathsf{Smoker} = \mathsf{no}, \mathsf{Lung\ \ Cancer} = \mathsf{Case}) +$$
$$\mathrm{P}(\mathsf{Smoker} = \mathsf{no}, \mathsf{Lung\ \ Cancer} = \mathsf{Control}) = 0.015 + 0.042 = 0.056$$

If we write down the notation

$$\mathrm{P}(\texttt{Smoker} = \texttt{yes}, \texttt{Lung Cancer} = \texttt{Case}) + \mathrm{P}(\texttt{Smoker} = \texttt{yes}, \texttt{Lung Cancer} = \texttt{Control}),$$

because of axiom #2 (Case and Control are disjoint) we have that

$$\mathrm{P}(\texttt{Smoker} = \texttt{yes}, \texttt{Lung Cancer} = \texttt{Case}) + \mathrm{P}(\texttt{Smoker} = \texttt{yes}, \texttt{Lung Cancer} = \texttt{Control}) = \mathrm{P}(\texttt{Smoker} = \texttt{yes}, \texttt{Lung Cancer} = \texttt{Case} \cup \texttt{Control}),$$

and because of axiom #1 (Case and Control make up the whole sample space)

$$\mathrm{P}(\texttt{Smoker} = \texttt{yes}, \texttt{Lung Cancer} = \texttt{Case} \cup \texttt{Control}) = \mathrm{P}(\texttt{Smoker} = \texttt{yes})$$

This works the other way round as well: we can compute joint probabilities from marginal and conditional probabilities.

```
matrix.with.marginals = matrix(rep(marginal.lung.cancer, 2), ncol = 2)
recomputed.joint.probs = conditional.probs * matrix.with.marginals
recomputed.joint.probs
```

```
        Lung Cancer
  Smoker   Case Control
     yes 0.4852  0.4584
     no  0.0148  0.0416
```

```
all.equal(joint.probs, recomputed.joint.probs)
```

```
  [1] TRUE
```

Manually:

$$\begin{aligned}
&\mathrm{P}(\mathsf{Smoker} = \mathsf{yes}, \mathsf{Lung\ \ Cancer} = \mathsf{Case}) \\
&\quad = \mathrm{P}(\mathsf{Smoker} = \mathsf{yes} \mid \mathsf{Lung\ \ Cancer} = \mathsf{Case})\, \mathrm{P}(\mathsf{Lung\ \ Cancer} = \mathsf{Case}) \\
&\quad = 0.97 \times 0.5 = 0.485
\end{aligned}$$

The general case of this formula is called the chain rule, in which we start with

$$\mathrm{P}(X_1, \ldots, X_N) = \mathrm{P}(X_1 \mid X_2, \ldots, X_N)\,\mathrm{P}(X_2, \ldots, X_N)$$

and then we follow on saying

$$\mathrm{P}(X_2, \ldots, X_N) = \mathrm{P}(X_2 \mid X_3, \ldots, X_N)\,\mathrm{P}(X_3, \ldots, X_N)$$

ans so on and so forth to get

$$\begin{aligned}\mathrm{P}(X_1, \ldots, X_N) = \mathrm{P}(X_1 \mid X_2, \ldots, X_N)\,\mathrm{P}(X_2 \mid X_3, \ldots, X_N)\times \\ \mathrm{P}(X_3 \mid X_4, \ldots, X_N) \times \ldots \times \mathrm{P}(X_{N-1} \mid X_N)\,\mathrm{P}(X_n)\end{aligned}$$

for any number N of variables.

Bayes' Rule

Another fundamental variation on the use of conditional probabilities: starting from

$$\mathrm{P}(X_1 \mid X_2) = \frac{\mathrm{P}(X_1, X_2)}{\mathrm{P}(X_2)}$$

we note that

$$\mathrm{P}(X_1, X_2) = \mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_1) \quad \text{as well as} \quad \mathrm{P}(X_1 \mid X_2)\,\mathrm{P}(X_2)$$

and if we substitute on the numerator you can write

$$\mathrm{P}(X_1 \mid X_2) = \frac{\mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_1)}{\mathrm{P}(X_2)}$$

which is known as Bayes' rule or Bayes' theorem.

Two events are independent if the occurrence of one does not affect the occurrence of the other:

$$\mathrm{P}(E_1 \cap E_2) = \mathrm{P}(E_1)\,\mathrm{P}(E_2);$$

and two random variables are independent if the respective events are independent

$$\mathrm{P}(X_1 = E_1 \cap X_2 = E_2) = \mathrm{P}(X_1 = E_1)\,\mathrm{P}(X_2 = E_2) \quad \text{for any } E_1, E_2.$$

Or, in short notation, $\mathrm{P}(X_1, X_2) = \mathrm{P}(X_1)\,\mathrm{P}(X_2)$.

But we also know that $\mathrm{P}(X_1, X_2) = \mathrm{P}(X_1 \mid X_2)\,\mathrm{P}(X_2)$, so if X_1 and X_2 are independent

$$\mathrm{P}(X_1 \mid X_2)\cancel{\mathrm{P}(X_2)} = \mathrm{P}(X_1)\cancel{\mathrm{P}(X_2)} \quad \text{giving} \quad \mathrm{P}(X_1 \mid X_2) = \mathrm{P}(X_1)$$

which makes sense because the conditioning variable does not tell us anything about the conditioned variable.

Conditional Independence

The concept of independence carries over to conditional probabilities; it just means that the occurrence of one event does not affect the occurrence of the other in the particular context given by the conditioning event:

$$\mathrm{P}(E_1 \cap E_2 \mid E_3) = \mathrm{P}(E_1 \mid E_3)\,\mathrm{P}(E_2 \mid E_3).$$

For random variables, this gives conditional independence as

$$\mathrm{P}(X_1, X_2 \mid X_3) = \mathrm{P}(X_1 \mid X_3)\,\mathrm{P}(X_2 \mid X_3);$$

if there are no conditioning variables (as in the previous slides) we often talk about marginal independence for clarity.

An Example: Radio vs Chemotherapy at Two Clinics

Consider now another example: we are comparing the effectiveness of radiotherapy and chemotherapy over two different clinics.

```
therapy = array(c(18, 12, 12, 8, 2, 8, 8, 32), dim = c(2, 2, 2),
            dimnames = list(Therapy = c("Radio", "Chemo"),
                            Response = c("Success", "Failure"),
                            Clinic = c("A", "B")))
therapy
```

```
, , Clinic = A

        Response
Therapy Success Failure
  Radio      18      12
  Chemo      12       8

, , Clinic = B

        Response
Therapy Success Failure
  Radio       2       8
  Chemo       8      32
```

An Example: Radio vs Chemotherapy at Two Clinics

First we need to transform a table of counts in a table of joint probabilities of Therapy, Response and Clinic.

```
therapy = prop.table(therapy)
therapy
```

```
, , Clinic = A

        Response
Therapy Success Failure
  Radio    0.18    0.12
  Chemo    0.12    0.08

, , Clinic = B

        Response
Therapy Success Failure
  Radio    0.02    0.08
  Chemo    0.08    0.32
```

Conditional Independence: at Clinic A

Let's start by looking at clinic A; if one of radiotherapy and chemotherapy is better than the other then Response should be dependent on Therapy conditional on `Clinic = A`.

```
conditional.A = prop.table(therapy[, , Clinic = "A"])
conditional.A
```

```
         Response
  Therapy Success Failure
    Radio    0.36    0.24
    Chemo    0.24    0.16
```

```
marginal.therapy = margin.table(conditional.A, margin = 1)
marginal.response = margin.table(conditional.A, margin = 2)
independent.A = prop.table(marginal.therapy %*% t(marginal.response))
independent.A
```

```
         Response
  Therapy Success Failure
    Radio    0.36    0.24
    Chemo    0.24    0.16
```

```
all.equal(conditional.A, independent.A)
```

```
  [1] TRUE
```

Conditional Independence: at Clinic A

Manually, we first compute the marginal probabilities in `marginal.therapy` and `marginal.response`.

$$
\begin{aligned}
&\mathrm{P}(\texttt{Therapy} = \texttt{Radio} \mid \texttt{Clinic} = \texttt{A}) \\
&\quad = \mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Success} \mid \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Failure} \mid \texttt{Clinic} = \texttt{A}) = 0.36 + 0.24 = 0.6 \\
&\mathrm{P}(\texttt{Therapy} = \texttt{Chemo} \mid \texttt{Clinic} = \texttt{A}) \\
&\quad = \mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Success} \mid \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Failure} \mid \texttt{Clinic} = \texttt{A}) = 0.24 + 0.16 = 0.4
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{P}(\texttt{Response} = \texttt{Success} \mid \texttt{Clinic} = \texttt{A}) \\
&\quad = \mathrm{P}(\texttt{Response} = \texttt{Success}, \texttt{Therapy} = \texttt{Radio} \mid \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Response} = \texttt{Success}, \texttt{Therapy} = \texttt{Chemo} \mid \texttt{Clinic} = \texttt{A}) = 0.36 + 0.24 = 0.6 \\
&\mathrm{P}(\texttt{Response} = \texttt{Failure} \mid \texttt{Clinic} = \texttt{A}) \\
&\quad = \mathrm{P}(\texttt{Response} = \texttt{Failure}, \texttt{Therapy} = \texttt{Radio} \mid \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Response} = \texttt{Failure}, \texttt{Therapy} = \texttt{Chemo} \mid \texttt{Clinic} = \texttt{A}) = 0.24 + 0.16 = 0.4
\end{aligned}
$$

Conditional Independence: at Clinic A

Then we construct $\mathrm{P}(\texttt{Therapy}, \texttt{Response} \mid \texttt{Clinic} = \texttt{A})$ as the product of the marginal probabilities $\mathrm{P}(\texttt{Therapy} \mid \texttt{Clinic} = \texttt{A})$ and $\mathrm{P}(\texttt{Response} \mid \texttt{Clinic} = \texttt{A})$ assuming they are independent.

	Success	Failure
Radio	$0.6 \times 0.6 = 0.36$	$0.6 \times 0.4 = 0.24$
Chemo	$0.4 \times 0.6 = 0.24$	$0.4 \times 0.4 = 0.16$

These probabilities are identical to those in `conditional.A`; hence we conclude Therapy and Response are conditionally independent given `Clinic = A`.

Conditional Independence: at Clinic B

The same turns out to be true given `Clinic = B`.

```
conditional.B = prop.table(therapy[, , Clinic = "B"])
conditional.B
```

```
         Response
  Therapy Success Failure
    Radio    0.04    0.16
    Chemo    0.16    0.64
```

```
marginal.therapy = margin.table(conditional.B, margin = 1)
marginal.response = margin.table(conditional.B, margin = 2)
independent.B = prop.table(marginal.therapy %*% t(marginal.response))
independent.B
```

```
         Response
  Therapy Success Failure
    Radio    0.04    0.16
    Chemo    0.16    0.64
```

```
all.equal(conditional.B, independent.B)
```

```
  [1] TRUE
```

Marginal Independence: Lumping Clinics Together

What do we find if we lump clinics A and B together, and look just at Therapy and Response?

```
marginal.AB = margin.table(therapy, margin = 1:2)
marginal.AB
```

```
         Response
Therapy Success Failure
  Radio     0.2     0.2
  Chemo     0.2     0.4
```

```
marginal.therapy = margin.table(marginal.AB, margin = 1)
marginal.response = margin.table(marginal.AB, margin = 2)
independent.AB = prop.table(marginal.therapy %*% t(marginal.response))
independent.AB
```

```
         Response
Therapy Success Failure
  Radio    0.16    0.24
  Chemo    0.24    0.36
```

```
all.equal(marginal.AB, independent.AB)
```

```
[1] "Mean relative difference: 0.16"
```

Marginal Independence: Lumping Clinics Together

Manually, we first compute the probabilities in `marginal.AB`.

$$
\begin{aligned}
&\mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Success}) \\
&\quad = \mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Success}, \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Success}, \texttt{Clinic} = \texttt{B}) = 0.18 + 0.02 = 0.2 \\
&\mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Failure}) \\
&\quad = \mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Failure}, \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Therapy} = \texttt{Radio}, \texttt{Response} = \texttt{Failure}, \texttt{Clinic} = \texttt{B}) = 0.12 + 0.08 = 0.2 \\
&\mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Success}) \\
&\quad = \mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Success}, \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Success}, \texttt{Clinic} = \texttt{B}) = 0.12 + 0.08 = 0.2 \\
&\mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Failure}) \\
&\quad = \mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Failure}, \texttt{Clinic} = \texttt{A}) + \\
&\qquad \mathrm{P}(\texttt{Therapy} = \texttt{Chemo}, \texttt{Response} = \texttt{Failure}, \texttt{Clinic} = \texttt{B}) = 0.08 + 0.32 = 0.4
\end{aligned}
$$

These probabilities are at the same time marginal with respect to `Clinic` (which we are removing) and joint for `Therapy` and `Response` (which we are measuring together).

Marginal Independence: Lumping Clinics Together

From the probabilities in `marginal.AB`, we can then compute the marginal probabilities for `Therapy` and `Response`;

$$\begin{aligned}
\mathrm{P}(\texttt{Therapy} = \texttt{Radio}) &= 0.2 + 0.2 = 0.4 \\
\mathrm{P}(\texttt{Therapy} = \texttt{Chemo}) &= 0.2 + 0.4 = 0.6 \\
\mathrm{P}(\texttt{Response} = \texttt{Success}) &= 0.2 + 0.2 = 0.4 \\
\mathrm{P}(\texttt{Response} = \texttt{Failure}) &= 0.4 + 0.2 = 0.6
\end{aligned}$$

and from the marginal probabilities we can compute the table of their products to get the joint probabilities under the assumption of independence.

	`Success`	`Failure`
`Radio`	$0.4 \times 0.4 = 0.16$	$0.4 \times 0.6 = 0.24$
`Chemo`	$0.6 \times 0.4 = 0.24$	$0.6 \times 0.6 = 0.36$

Marginal vs Conditional Independence

How can we interpret this difference?

1. Response is independent from Therapy in clinic A;
2. Response is independent from Therapy in clinic B;
3. Response is dependent from Therapy overall;
4. this suggests that: Response may be different in the two clinics?
5. or maybe: different Therapy are administered in differently in the two clinics?
6. or maybe: patients are different between the two clinics?

What we learned: conditional independence does not imply marginal independence; or vice versa for that matter.

Possible Explanations?

The probability of having Response = Success is higher at clinic A (odds are $0.3/0.2 = 1.5$ vs $0.1/0.4 = 0.25$ for clinic B).

```
margin.table(therapy, margin = c(2, 3))
```

```
          Clinic
Response    A   B
  Success 0.3 0.1
  Failure 0.2 0.4
```

But Therapy are also administered in different proportions in the two clinics; so if patients are different, or doctors have different skills, that may explain the marginal dependence.

```
margin.table(therapy, margin = c(1, 3))
```

```
        Clinic
Therapy   A   B
  Radio 0.3 0.1
  Chemo 0.2 0.4
```

Probability Calculus: a Summary

- Different interpretations of probability:
  - frequentist;
  - subjective;
  - information.
- Basic probability axioms.
- Joint, marginal and conditional probability.
- Marginal and conditional independence.
- Chain rule.
- Bayes' rule.

- `sample()` for random sampling.
- `table()` for counting and creating tables.
- `matrix()` and `array()` for 2D and higher-dimensional arrays.
- `prop.table()` to transform counts into relative frequencies.
- `margin.table()` to sum up array elements over one or more dimensions.
- `rowSums()` and `colSums()` as special cases to sum up over rows and columns in 2D arrays.

Summary and Remarks

- Probability is a key tool for modelling uncertainty.
- It has solid mathematical foundations, but it is governed by few simple axioms.
- Probability makes it possible to reason about multiple related variables and explain complex phenomena using marginal and conditional probability and independence.
- Probability itself has various interpretations, which are useful in framing problems in different ways.

Introduction to Machine Learning

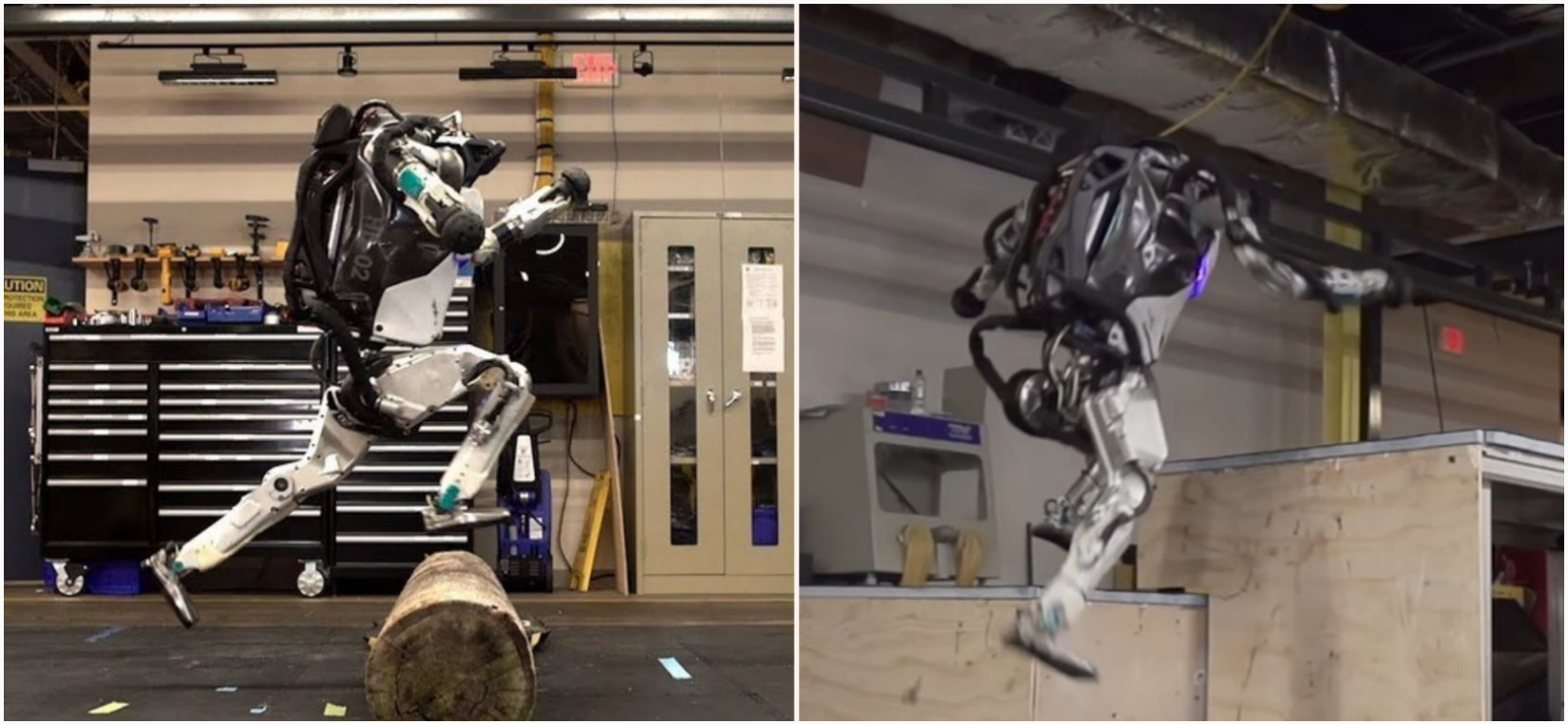
The Grand Vision

Machine learning studies the algorithms and statistical tools that allow computer systems to perform specific, well-defined tasks without explicit instructions. It is a sub-field of artificial intelligence.

Broadly speaking, in order to do this:

1. We need a working model of the world that describes the task and its context in a way a computer can understand.
2. We need a goal: how do we measure the performance of the model? Because that is what we optimise for; usually it is the ability to predict new events.
3. We encode our knowledge of the world drawing information from training data, experts or both; this is called learning.
4. The computer system uses the model as a proxy of reality and, as new inputs come in, to perform inference and decide if/how to perform the assigned task.

Robotics

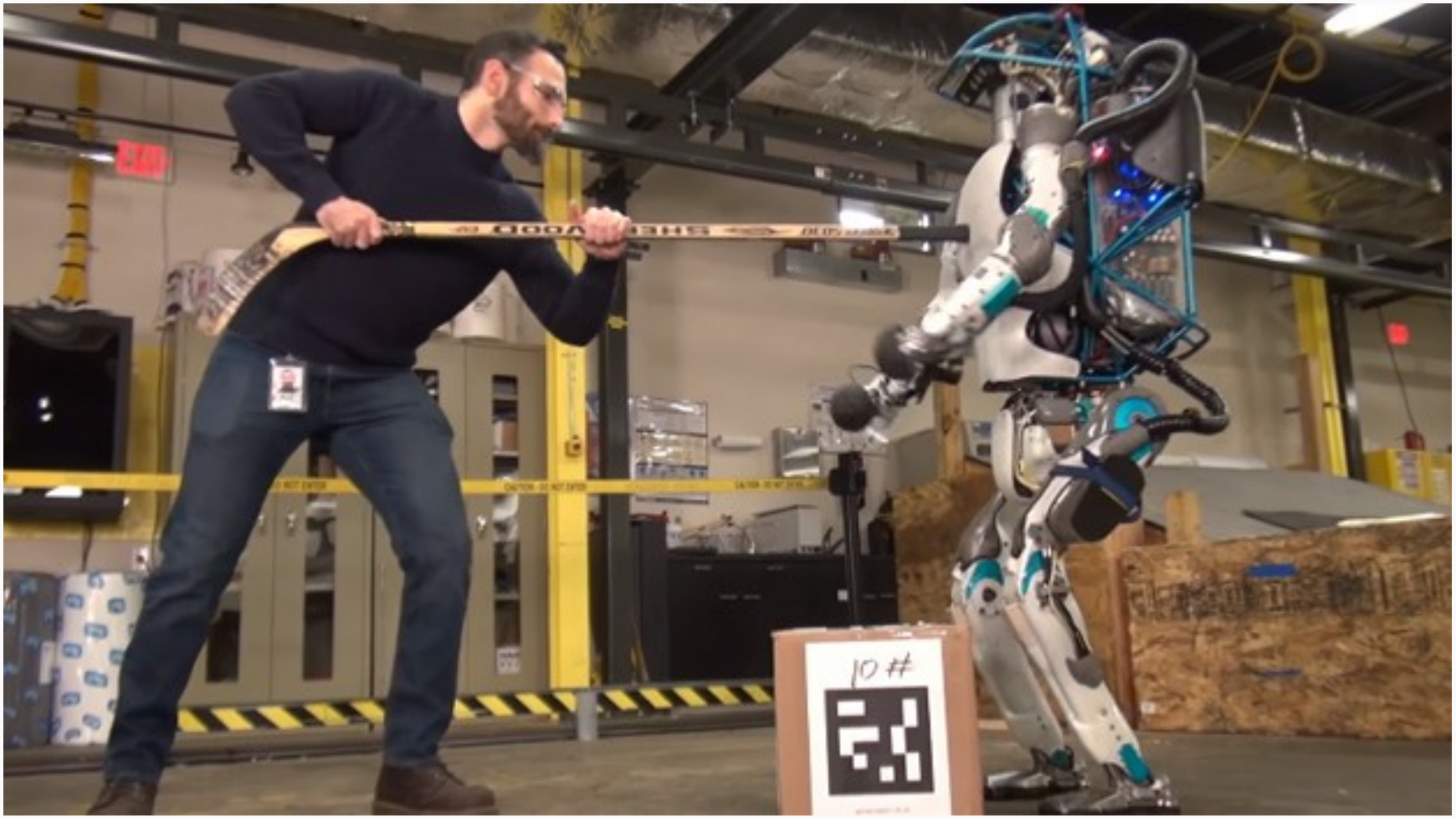


Boston Dynamics robots can walk, run, move around (or jump over!) obstacles and carry objects...

Robotics

... even with pesky humans interfering...

Robotics



... in violent ways!

Playing Games

DeepMind AlphaGo beating the best human Go player!

Diagnosing Diseases

The Unfortunate Reality

https://xkcd.com/1838

However, building machine learning applications is far from trivial and it is a craft as much as it is a science.

- It requires large amounts of data, which has to be collected keeping the goal of the model in mind.
- It is difficult to decide how to structure the model from a mathematical and probabilistic point of view
- It is difficult to evaluate and troubleshoot models of any real-world complexity.

Identify the Variables to Include in the Model

The first step in building a machine learning model is to choose which variables to include. Which aspects of/entities in the world do we need the model to represent for the computer to carry out the assigned task? This is known as feature selection.

- Each aspect of the world or entity is modelled with one random variable.
- We should use a small enough number of variables because if we have too many:
  - it is difficult it is to construct the model;
  - it is difficult to interpret and troubleshoot it;
  - the model requires too much computing power to learn and to run.
- We must choose which are the relevant events that make up the sample space of each variable, again taking care of not having too many.

Identify the Variables to Include in the Model

For instance, to play a game of Go we need to model each piece using its position on the board (which provides a regular grid to use for coordinates) and which player it belongs to.

In robotics applications:

- the position of the robot, and the positions of its hands relative to its body?
- the size and position of all the objects in the room?
- the size and position of the box to pick up?

All of these can be guessed from the information the robot gets from the sensors it is equipped with (camera, infrared, radar, etc.). Then there is the question of how to represent them (coordinates on a grid, and sizes in increments of 5cm? real numbers?).

In clinical applications we need additional machine learning to even figure out which variables we should include in our model...

Generative versus Discriminative Models

The second step is choosing which class of machine learning models to select from.

- Generative models: we have a set of variables $X_1, \ldots, X_N$ describing various components of a complex phenomenon, and we are interested in modelling that phenomenon in a mechanistic way. Hence, we want to show how the various parts interact with each other, and in order to do that we choose to model their joint probability $\mathrm{P}(X_1, \ldots, X_N)$.
- Discriminative models: we have one particular variable (say, X_1) that is closely tied with our model task, and a number of other variables ($X_2, \ldots, X_N$) which we believe can be used to explain it. We do not care about how the X_i are related to each other, so we just model $\mathrm{P}(X_1 \mid X_2, \ldots, X_N)$.

Generative versus Discriminative Models

Generative Models

Pros

More flexible in terms of what questions they can answer.

Cons

More complicated to formulate.
Require more data to learn.

Discriminative Models

Pros

Require less data to learn.
Better at prediction than generative models.

Cons

Exclusively focused on predicting the variable of interest.

The “Uncertainty Reasoning” covers generative models, the “Data Mining” module covers discriminative models.

Model Relationships Between Variables

How do we decide whether there is a relationships between variables? If we had perfect knowledge we could completely describe the world. But we never have perfect knowledge:

- in a game of Go, some moves are more likely than others but we do not know for sure what more our opponent will pick;
- in robotics we are limited by what the sensors can tell us;
- in clinical applications we are limited by what we can learn from patients.

Hence we use the language of probability, and we say that two variables are associated if the occurrence of an event in one variable affects the probability of an event occurring in another variable. In other words, they are associated if they are not independent, possibly given other variables.

An example: the Car Start Problem

"*In the morning, my car will not start. The start engine turns, but nothing happens. The battery is OK. The problem may be due to dirty spark plugs or the fuel may be stolen. I look at the fuel meter. It shows* $1/2$*, hence I expect the spark plugs to be dirty.*"

We need to formalise this kind of reasoning into a model that a computer can understand.

- What made we think of fuel and spark plugs?
- Why did we look at the fuel meter?
- Why had fuel meter reading an impact on our belief that spark plugs are dirty?

Car Start: Variables and Events

From the problem description, our task is to explain why the car will not start using a model comprising these four variables:

- `Fuel`;
- `Spark Plugs`;
- `Start`;
- `Fuel Meter`.

But how do we model the sample space in terms of events?

	Realistic	Pragmatic
`Fuel`	`0%–100%`	`Yes, No`
`Spark Plugs`	`Work, Fault`	`Work, Fault`
`Start`	`Yes, No`	`Yes, No`
`Fuel Meter`	`0%–100%`	`Empty, Half, Full`

Car Start: Which Variables Are Associated?

In a generative model, $\mathrm{P}(\texttt{Fuel}, \texttt{Spark Plugs}, \texttt{Fuel Meter}, \texttt{Start})$ is our probabilistic model in its most general form. It leads to a table with $2 \times 2 \times 2 \times 3 = 24$ probabilities; it would be difficult for us to choose accurate values for each of them.

We could simply use the chain rule to write the model as

$$\begin{aligned}
&\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) \\
&\quad = \mathrm{P}(\texttt{Start} \mid \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\qquad \mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\qquad \mathrm{P}(\texttt{Fuel} \mid \texttt{Spark Plugs})\, \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$

but that does not change the complexity model, even if it breaks it apart in smaller pieces.

To actually make it simpler we should ask: do we know which variables are associated with each other, and which are not?

Car Start: Expert Knowledge of Associations

In our (expert?) knowledge:

- `Start` is associated with `Fuel`: from the definition of independence

$$\mathrm{P}(\texttt{Start} = \texttt{Yes} \mid \texttt{Fuel} = \texttt{No}) = 0 \neq \mathrm{P}(\texttt{Start} = \texttt{Yes}) > 0$$

  so the two are not independent.

- Similarly, `Start` is associated with `Spark Plugs`.

- `Fuel Meter` is associated with `Fuel`, because the former is a transformation of the latter. It is also easy to see that

$$\mathrm{P}(\texttt{Fuel Meter} = \texttt{Full} \mid \texttt{Fuel} = \texttt{No}) = 0 \neq \mathrm{P}(\texttt{Fuel Meter} = \texttt{Full}) > 0.$$

Car Start: Expert Knowledge of Associations

- `Start` is not associated with `Fuel Meter` given `Fuel` for the same reason: if we know the amount of `Fuel`, whether `Fuel Meter` is `Empty`, `Half` or `Full` should not alter the probability that `Start` is `Yes` or `No`.
- `Fuel` is not associated with `Spark Plugs` because knowing whether `Fuel` is `Yes` or `No` should not alter the probability that `Spark Plugs` are `Work` or `Fault`.

We can take these considerations and use them to make the model simpler by removing the variables we do not need from the conditional probabilities.

Car Start: a Simpler Model

Hence we are left with:

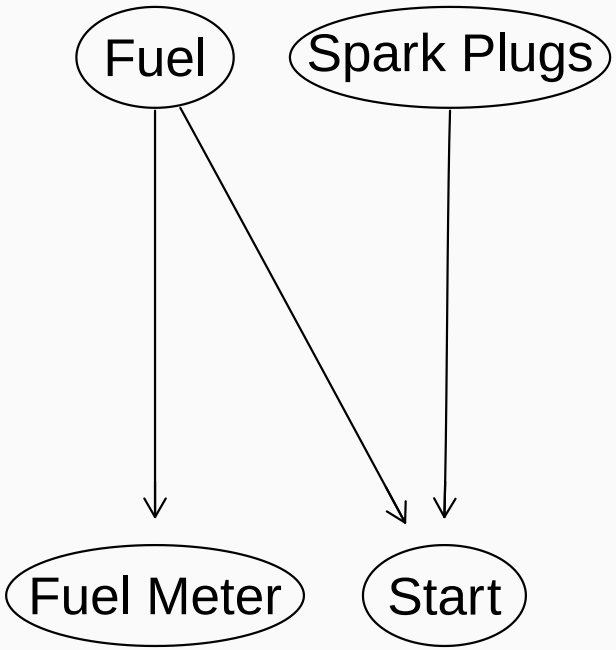
$$\begin{aligned}&\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs})\\ &\quad = \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs})\,\mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel})\times\\ &\qquad\qquad \mathrm{P}(\texttt{Fuel})\,\mathrm{P}(\texttt{Spark Plugs})\end{aligned}$$

If we represent this model as a graph, with

- variables as nodes and
- associations as arcs

we get a qualitative view of what our model looks like.

This representation is the key idea of Bayesian networks, which we will cover in the rest of the module.



Car Start: Probabilistic versus Causal Construction

In probability associations are symmetric; the derivation of Bayes' theorem makes it really clear that

$$\mathrm{P}(X_1 \mid X_2)\,\mathrm{P}(X_2) = \mathrm{P}(X_1, X_2) = \mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_1).$$

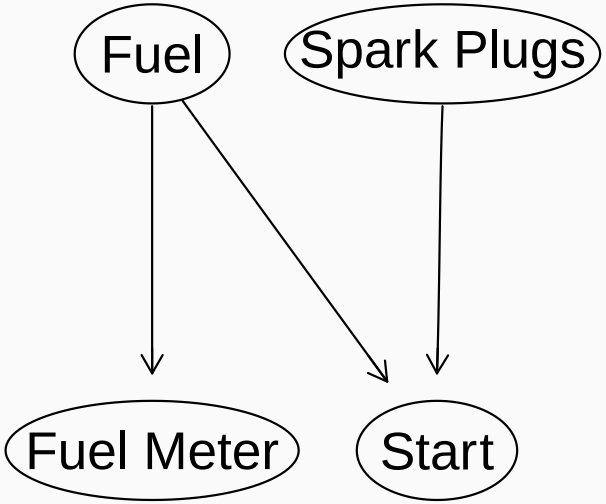
In order to write the conditional probabilities, we used common sense to choose the conditioning variables such that they affect the conditioned variables.

But what does that mean from a modelling point of view? It means we are giving arcs a causal interpretation and we choose arc directions to go from cause (nodes) to effect (nodes).

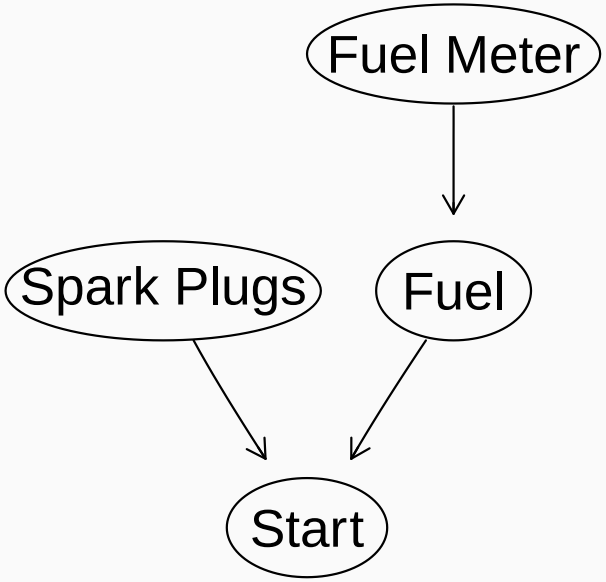
How do we do that?

Car Start: Playing with Arc Directions

$$\begin{aligned}
&\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) \\
&\quad = \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\qquad \mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel})\, \mathrm{P}(\texttt{Fuel}) \times \\
&\qquad \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



$$\begin{aligned}
&\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) \\
&= \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\quad \mathrm{P}(\texttt{Fuel Meter})\, \mathrm{P}(\texttt{Fuel} \mid \texttt{Fuel Meter}) \times \\
&\quad \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



Car Start: Playing with Arc Directions

The criterion to identify causes and effect is intervention. Consider:

- If we fill the tank with fuel, the fuel meter goes up.
- If we tamper with the fuel meter to make is say `Full`, the fuel tank does not magically refill itself.

Hence, `Fuel` is the cause and `Fuel Meter` is the effect and the most intuitive arc direction is $\texttt{Fuel} \rightarrow \texttt{Fuel Meter}$.

What the probability $\mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel})$ tells us is just that if the fuel meter says `Full` there probably is fuel in the tank, whereas if the fuel meter says `Empty` there may be no fuel in the tank (assuming the fuel meter works reliably).

Car Start: Learning Association from Data

What if we do not have expert knowledge of associations, so we are stuck at

$$\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel}, \texttt{Spark Plugs}) = ?$$

It is possible to learn associations from data to select a good model, which will be covered later in the module. The key idea is that:

1. we collect data (that is, sets of values for all the variables in the model);
2. we take different models and we compute the probability they give to the data;
3. we choose the model that gives the highest probability to the data, taking the complexity of the model into consideration.

Car Start: the Conditional Probabilities

Spark Plugs	
Work	Fault
?	?

Fuel	
Yes	No
?	?

Start

	Spark Plugs = Work	
	Fuel = Yes	Fuel = No
Yes	?	?
No	?	?

Fuel Meter

	Fuel = Yes	Fuel = No
Empty	?	?
Half	?	?
Full	?	?

	Spark Plugs = Fault	
	Fuel = Yes	Fuel = No
Yes	?	?
No	?	?

After we decide that this model is good to go, we need to fill in the values of all the conditional probabilities that are implied by the model.

The number of these probabilities gives the complexity of the model ($2 + 2 + 6 + 8 = 18 < 24$).

Car Start: the Conditional Probabilities

Again, we have two ways of doing that:

- ask someone with expert knowledge who can tell us which values to fill in;
- estimate the (conditional) probabilities from the data with one of the approaches we have seen earlier in the module.

Starting from expert knowledge, we can create the probability and conditional probability tables as follows.

1. Store the events in the sample space for each variable.

```
Fuel.lvl = c("Yes", "No")
Spark.Plugs.lvl = c("Work", "Fault")
Fuel.Meter.lvl = c("Full", "Half", "Empty")
Start.lvl = c("Yes", "No")
```

Car Start: Expert Probabilities in R

2. Create the probability tables labelling rows and columns for easy reference.

```
Fuel.probs = array(c(0.98, 0.02), dim = 2, dimnames = list(Fuel = Fuel.lvl))
Spark.Plugs.probs = array(c(0.96, 0.04), dim = 2,
                          dimnames = list(Spark.Plugs = Spark.Plugs.lvl))
Fuel.Meter.probs = array(c(0.39, 0.60, 0.01, 0.01, 0.01, 0.98), dim = c(3, 2),
                         dimnames = list(Fuel.Meter = Fuel.Meter.lvl,
                                         Fuel = Fuel.lvl))
Start.probs = array(c(0.99, 0.01, 0, 1, 0.01, 0.99, 0, 1), dim = c(2, 2, 2),
                    dimnames = list(Start = Start.lvl, Fuel = Fuel.lvl,
                                    Spark.Plugs = Spark.Plugs.lvl))
```

3. Store all probability tables together in a list.

```
expert.probabilities = list(
  Fuel = Fuel.probs,
  Spark.Plugs = Spark.Plugs.probs,
  Fuel.Meter = Fuel.Meter.probs,
  Start = Start.probs
)
```

Car Start: Expert Probabilities in R

The end result is this collection of expert probabilities, organised in tables:

```
expert.probabilities$Fuel
```

```
  Fuel
   Yes   No
  0.98 0.02
```

```
expert.probabilities$Spark.Plugs
```

```
  Spark.Plugs
   Work Fault
   0.96  0.04
```

```
expert.probabilities$Fuel.Meter
```

```
             Fuel
  Fuel.Meter  Yes   No
       Full  0.39 0.01
       Half  0.60 0.01
       Empty 0.01 0.98
```

```
expert.probabilities$Start
```

```
, , Spark.Plugs = Work

      Fuel
Start  Yes No
  Yes 0.99  0
  No  0.01  1

, , Spark.Plugs = Fault

      Fuel
Start  Yes No
  Yes 0.01  0
  No  0.99  1
```

How can we get the same thing from data, without using an expert?

Car Start: Probabilities from Data in R

Suppose we collect a number of data points for the four variables in the model.

```
dim(training.set)
```

```
[1] 500   4
```

```
head(training.set)
```

```
  Fuel Fuel.Meter Spark.Plugs Start
1  Yes       Full        Work   Yes
2  Yes       Half        Work   Yes
3  Yes       Full        Work   Yes
4  Yes       Half        Work   Yes
5  Yes       Half        Work   Yes
6  Yes       Half        Work   Yes
```

We can use these data to compute the frequentist probabilities we need to fill in the model using the corresponding relative frequencies, as we covered earlier in the module.

Car Start: Probabilities from Data in R

Again, we can compute the absolute frequencies (`*.counts`) and then the relative frequencies (`*.probs`) for each variable, making sure we get the right conditional probabilities from `prop.table()`.

```
Fuel.counts = table(training.set[, "Fuel"])
Fuel.probs = prop.table(Fuel.counts)
Spark.Plugs.counts = table(training.set[, "Spark.Plugs"])
Spark.Plugs.probs = prop.table(Spark.Plugs.counts)
Fuel.Meter.counts = table(training.set[, c("Fuel.Meter", "Fuel")])
Fuel.Meter.probs = prop.table(Fuel.Meter.counts, margin = 2)
Start.counts = table(training.set[, c("Start", "Fuel", "Spark.Plugs")])
Start.probs = prop.table(Start.counts, margin = 2:3)
```

We can then organise them in a list as we did for the expert probabilities.

```
probability.tables = list(
  Fuel = Fuel.probs,
  Spark.Plugs = Spark.Plugs.probs,
  Fuel.Meter = Fuel.Meter.probs,
  Start = Start.probs
)
```

Car Start: Probabilities from Data in R

The end result is this collection of frequentist probabilities, organised in the same way as the expert probabilities:

```
probability.tables$Fuel

   Yes    No
 0.988 0.012
probability.tables$Spark.Plugs

  Work Fault
 0.972 0.028
probability.tables$Fuel.Meter

            Fuel
Fuel.Meter     Yes      No
     Full  0.37854 0.00000
     Half  0.61538 0.00000
     Empty 0.00607 1.00000
```

```
probability.tables$Start

, , Spark.Plugs = Work

      Fuel
Start    Yes     No
  Yes 0.9833 0.0000
  No  0.0167 1.0000

, , Spark.Plugs = Fault

      Fuel
Start    Yes No
  Yes 0.0000
  No  1.0000
```

Each set of probabilities gives a different model for the car start problem.

Spot the Difference

- Expert probabilities are "nicer" in the sense that they are usually reasonably round numbers with 2-3 decimal places.
- Expert probabilities give probabilities greater than zero to events that are really rare, because the experts know these events are not impossible. (An example: `Fuel Meter` $=$ `Full` when `Fuel` $=$ `No`; fuel meters are very reliable but they definitely can break.)
- Frequentist probabilities are limited by the data we can collect; we may not actually observe rare events and then they get a probability of exactly zero. But they are not impossible!
- For this reason, frequentist probabilities are bad at representing small (≈ 0) and large (≈ 1) probabilities because they have a granularity of $1/$`nrow(training.set)` (here $1/500$).
- And if `nrow(training.set)` is small they are bad at representing any probability.

Training and Validation Data Sets

In machine learning we assess how well a model works by looking at how accurate it is in predicting new observations. To do this we need a data set that has not been previously used to define the model (such as computing frequentist probabilities); this second data set is called the validation data set. It is "new" in the sense that the model has hot seen these data before.

In contrast, the data set we use to define the model is called the training data set, because it is used to train the model.

```
head(validation.set)
```

```
  Fuel Fuel.Meter Spark.Plugs Start
1   No      Empty        Work    No
2  Yes       Full        Work   Yes
3  Yes       Half        Work   Yes
4  Yes       Half        Work   Yes
5  Yes       Half        Work   Yes
6  Yes       Full        Work   Yes
```

Probability of the Validation Data Set

If we take the first data point from the validation set, we have complete set of values for all the variables and we can feed those values to the model to compute their joint probability.

```
new.data = validation.set[1, ]
probability.tables$Fuel[new.data$Fuel]
```

```
    No
0.012
```

```
probability.tables$Spark.Plugs[new.data$Spark.Plugs]
```

```
  Work
0.972
```

```
probability.tables$Fuel.Meter[new.data$Fuel.Meter, new.data$Fuel]
```

```
[1] 1
```

```
probability.tables$Start[new.data$Start, new.data$Fuel, new.data$Spark.Plugs]
```

```
[1] 1
```

So, the probability of that data point is $0.012 \times 0.972 \times 1 \times 1 \approx 0.012$.

Probability of the Validation Data Set

If we iterate over all the validation set, and we multiply all the resulting probabilities together we obtain its predictive probability.

```
validation.set.probability = 1
for (i in seq(nrow(validation.set))) {

  val = validation.set[i, ]
  new.prob = probability.tables$Fuel[val$Fuel] *
             probability.tables$Spark.Plugs[val$Spark.Plugs] *
             probability.tables$Fuel.Meter[val$Fuel.Meter, val$Fuel] *
             probability.tables$Start[val$Start, val$Fuel, val$Spark.Plugs]

  validation.set.probability = validation.set.probability * new.prob

}#FOR
as.numeric(validation.set.probability)
```

```
[1] 7.1e-44
```

Why do we care about having a model with a good predictive probability?

Why the Probability of the Validation Data Set

The probability of the validation test is a measure of predictive accuracy, that is, the ability of the model to predict new events. The reason why we want a model that maximises it is as follows.

1. The four variables in the model can take 24 combinations of values.
2. Some combinations of their values will have higher probabilities (according to the model) than others.
3. In the validation sets, some combinations of values will appear more frequently than others.
4. If the machine learning model is a good working model of the world, it should assign high probability to combinations of values that appear more often.
5. Hence, we want a model that gives a high probability to the validation set as a whole.

The Log-Probability is What You Actually Want

Predictive accuracy is usually measured on a log-scale; on its natural scale it becomes too small very quickly when the validation set or the number of variables grow. It's already $\approx 10^{-43}$ for this simple model!

```
validation.log.probability = 0
for (i in seq(nrow(validation.set))) {

  val = validation.set[i, ]
  new.prob = probability.tables$Fuel[val$Fuel] *
             probability.tables$Spark.Plugs[val$Spark.Plugs] *
             probability.tables$Fuel.Meter[val$Fuel.Meter, val$Fuel] *
             probability.tables$Start[val$Start, val$Fuel, val$Spark.Plugs]

  validation.log.probability = validation.log.probability + log(new.prob)

}#FOR
as.numeric(validation.log.probability)
```

```
[1] -99.4
```

(The model with the expert probabilities gets -99.298, which is about the same.)

Car Start: Predicting Start

Sometimes we may also be interested in the accuracy of predicting just one variable given some other variables. For instance, we would like to predict `Start` from the other variables.

```
errors = 0
for (i in seq(nrow(validation.set))) {

  val = validation.set[i, ]
  base.prob = probability.tables$Fuel[val$Fuel] *
              probability.tables$Spark.Plugs[val$Spark.Plugs] *
              probability.tables$Fuel.Meter[val$Fuel.Meter, val$Fuel]
  Yes.prob = base.prob *
              probability.tables$Start["Yes", val$Fuel, val$Spark.Plugs]
  No.prob = base.prob *
              probability.tables$Start["No", val$Fuel, val$Spark.Plugs]

  if (ifelse(Yes.prob > No.prob, "Yes", "No") != val$Start)
    errors = errors + 1

}#FOR
errors / nrow(validation.set)
```

```
[1] 0.01
```

Car Start: Interrogating the Model

A more general way of using a model is to interrogate it: we have some evidence on some of the variables (that is, we assume we know their values), and we would like to know the the probability of some event.

For instance: say that $\texttt{Fuel Meter} = \texttt{Half}$. How does $\mathrm{P}(\texttt{Start} = \texttt{Yes})$ change after we introduce this evidence in the model?

Predicting `Start` from all the other variables is a particular case in which we have evidence on all the other variables.

Car Start: the Exhaustive (Dumb) Way

Using probability axiom #2, we can write

$$\mathrm{P}(\texttt{Start} = \texttt{Yes}) = \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}) + \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{No})$$

and then, recursively,

$$\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}) = \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}) + \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Fault})$$

$$\begin{aligned}&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}) \\ &= \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}, \texttt{Fuel.Meter} = \texttt{Full}) + \\ &\quad \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}, \texttt{Fuel.Meter} = \texttt{Half}) + \\ &\quad \mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel} = \texttt{Yes}, \texttt{Spark.Plugs} = \texttt{Work}, \texttt{Fuel.Meter} = \texttt{Empty})\end{aligned}$$

Car Start: the Exhaustive (Dumb) Way

In practice this means that, for small models, we can just go through all combinations of values of the other variables.

```
yes.prob = 0
for (FL in Fuel.lvl)
  for (SP in Spark.Plugs.lvl)
    for (FM in Fuel.Meter.lvl) {

      yes.prob = yes.prob +
                 expert.probabilities$Fuel[FL] *
                 expert.probabilities$Spark.Plugs[SP] *
                 expert.probabilities$Fuel.Meter[FM, FL] *
                 expert.probabilities$Start["Yes", FL, SP]

      }#FOR
as.numeric(yes.prob)
```

```
[1] 0.932
```

So $\mathrm{P}(\mathtt{Start} = \mathtt{Yes}) = 0.932$, and $\mathrm{P}(\mathtt{Start} = \mathtt{No}) = 1 - 0.932 = 0.068$.

Car Start: the Exhaustive (Dumb) Way

```
half.prob = yes.and.half.prob = 0
for (FL in Fuel.lvl)
  for (SP in Spark.Plugs.lvl) {

    yes.and.half.prob = yes.and.half.prob + expert.probabilities$Fuel[FL] *
                          expert.probabilities$Spark.Plugs[SP] *
                          expert.probabilities$Fuel.Meter["Half", FL] *
                          expert.probabilities$Start["Yes", FL, SP]

  }#FOR
for (FL in Fuel.lvl)
  for (SP in Spark.Plugs.lvl)
    for (ST in Start.lvl) {

      half.prob = half.prob + expert.probabilities$Fuel[FL] *
                    expert.probabilities$Spark.Plugs[SP] *
                    expert.probabilities$Fuel.Meter["Half", FL] *
                    expert.probabilities$Start[ST, FL, SP]

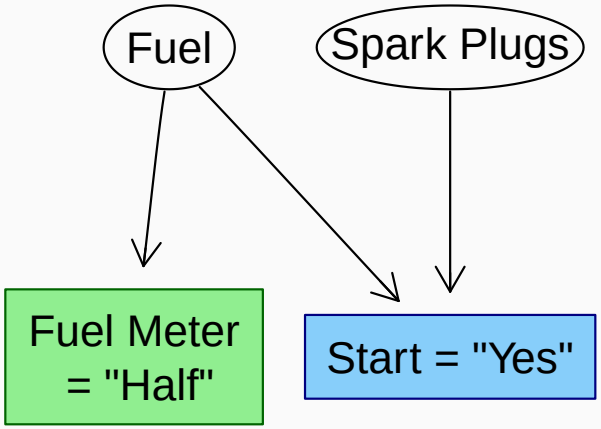
    }#FOR
as.numeric(yes.and.half.prob) / as.numeric(half.prob)
```

```
[1] 0.95
```

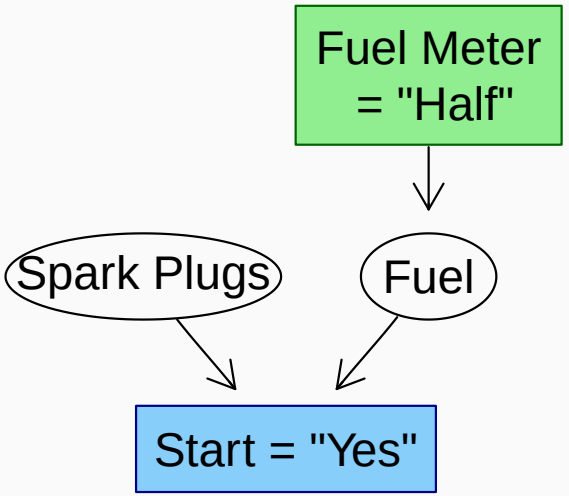
But is there a more efficient way of doing the same thing?

Car Start: the Principled (Probabilistic) Way

$$\begin{aligned}
&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel Meter} = \texttt{Half},\\
&\qquad \texttt{Fuel}, \texttt{Spark Plugs}) =\\
&= \mathrm{P}(\texttt{Start} = \texttt{Yes} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times\\
&\quad \mathrm{P}(\texttt{Fuel Meter} = \texttt{Half} \mid \texttt{Fuel}) \times\\
&\quad \mathrm{P}(\texttt{Fuel})\,\mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$

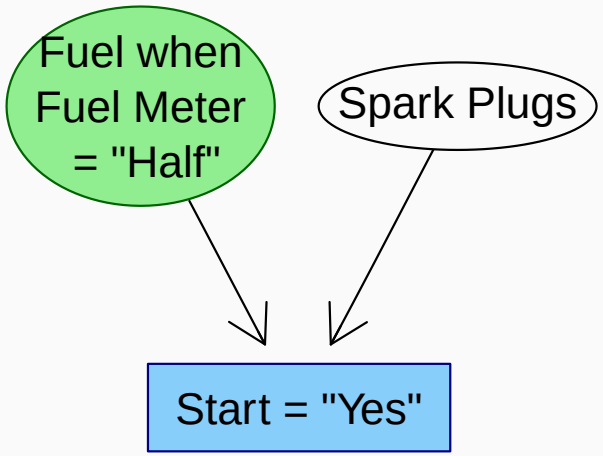


$$\begin{aligned}
&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel Meter} = \texttt{Half},\\
&\qquad \texttt{Fuel}, \texttt{Spark Plugs})\\
&= \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times\\
&\quad \mathrm{P}(\texttt{Fuel} \mid \texttt{Fuel Meter} = \texttt{Half}) \times\\
&\quad \frac{\mathrm{P}(\texttt{Fuel.Meter} = \texttt{Half})}{\cancel{\mathrm{P}(\texttt{Fuel})}} \times\\
&\quad \cancel{\mathrm{P}(\texttt{Fuel})}\,\mathrm{P}(\texttt{Spark Plugs})
\end{aligned}$$



Car Start: the Principled (Probabilistic) Way

$$
\begin{aligned}
&\mathrm{P}(\texttt{Start} = \texttt{Yes}, \texttt{Fuel}, \texttt{Spark Plugs} \mid \\
&\qquad \texttt{Fuel Meter} = \texttt{Half}) \\
&= \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs}) \times \\
&\qquad \mathrm{P}(\texttt{Fuel} \mid \texttt{Fuel Meter} = \texttt{Half}) \times \\
&\qquad \frac{\mathrm{P}(\texttt{Fuel.Meter} = \cancel{\texttt{Half}})}{\mathrm{P}(\cancel{\texttt{Fuel}}\texttt{.Meter} = \texttt{Half})} \times \\
&\qquad \mathrm{P}(\texttt{Spark Plugs})
\end{aligned}
$$



This leaves three variables, of which $\texttt{Start}$ is fixed to $\texttt{Yes}$: hence we have to consider $\mathrm{P}(\texttt{Start} = \texttt{Yes})$ under four scenarios:

$$
\begin{aligned}
&\texttt{Fuel} = \texttt{Yes} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Work} \\
&\texttt{Fuel} = \texttt{Yes} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Fault} \\
&\texttt{Fuel} = \texttt{No} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Work} \\
&\texttt{Fuel} = \texttt{No} \mid \texttt{Fuel Meter} = \texttt{Half}, \texttt{Spark Plugs} = \texttt{Fault}
\end{aligned}
$$

and sum the corresponding $\mathrm{P}(\texttt{Start} = \texttt{Yes} \mid \text{scenario})\,\mathrm{P}(\text{scenario})$.

Summary and Remarks

- Machine learning aims to make computer systems able to learn from and carry out tasks in the real world.
- Machine learning models represent a model of the world in a form useful to a computer, and use the language of probability to represent uncertainty.
- The focus of machine learning models is prediction, and by extension probabilistic reasoning and inference, so that the computer system can use the models to decide how to react to its environment.
- Generative models are better for reasoning, discriminative models are better for prediction; we focus on the former in this module.

Introduction to Bayesian Networks

Automating Modelling and Reasoning

Earlier in the course we have seen a simple machine learning model, which we have manually constructed with intuitive reasoning and manual applications on the probability axioms.

Now we want to automate the whole process, so that the computer system itself will (ideally) do all the work.

A model that promises to do this is Bayesian networks:

- they combine graphs and probability as we did earlier, but in a rigorous fashion;
- there are algorithms for automating reasoning that use the graphical part of the model to guide a computer system in applying the probability axioms to compute probabilities and predict events of interest;
- it is possible to learn them automatically from data.

Bayesian Networks in R: the bnlearn Package

The most comprehensive R package for working with Bayesian networks is **bnlearn**, which you should install by

```
install.packages("bnlearn")
```

The reference website for **bnlearn** is:

http://www.bnlearn.com

The list of functions in the package is available from:

```
help(package = "bnlearn")
```

The documentation of individual functions can be accessed with the ? operator after loading the package they are in:

```
library(bnlearn)
?empty.graph
```

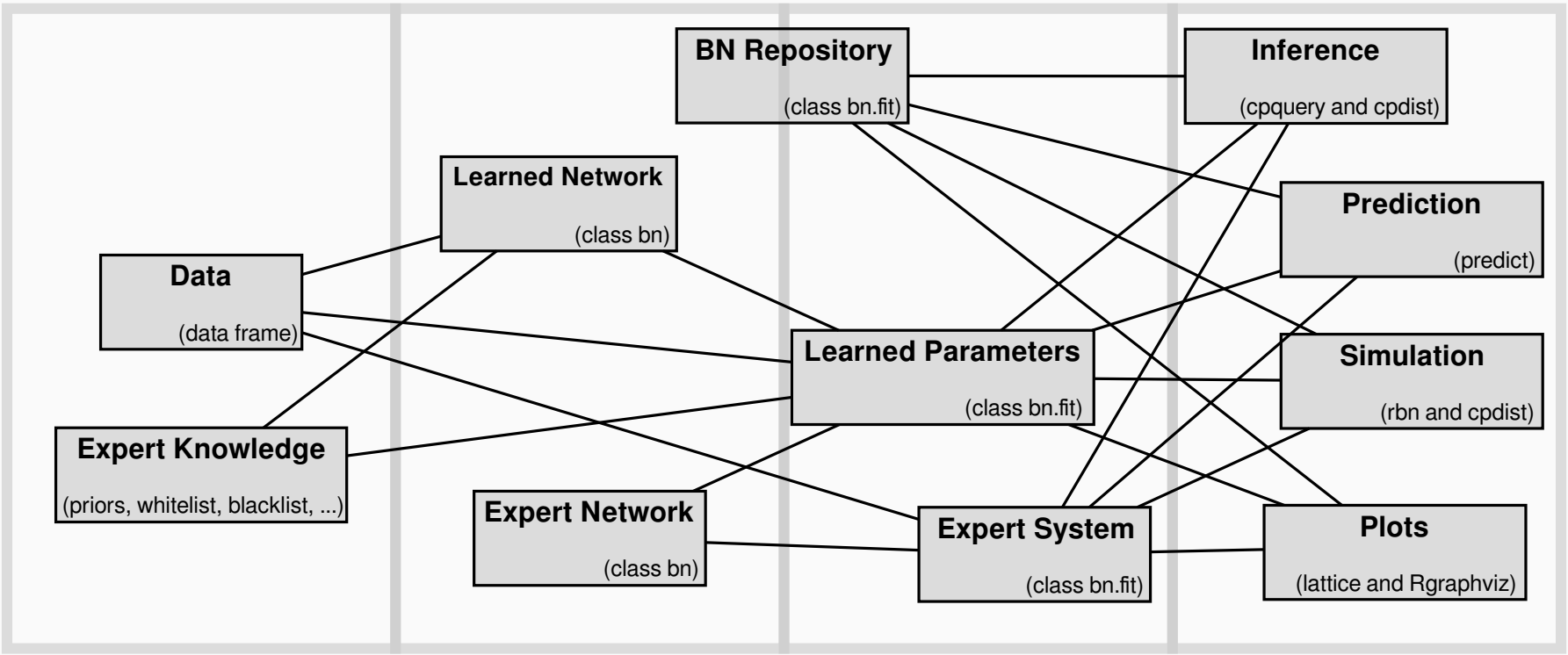
Bayesian Networks in R: Auxiliary Packages

There are many thousands of packages available for R: **bnlearn** uses a few for plotting (**lattice**, **Rgraphviz**) and to perform Bayesian network inference (**gRain**). You can install all of them with:

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("graph", "Rgraphviz", "RBGL"))
install.packages(c("gRain", "lattice"))
```

bnlearn will automatically load them as needed; and they can be loaded explicitly as well to use their functions directly.

The Architecture of bnlearn



A Graph and a Probability Distribution

Bayesian networks (BNs) are defined by:

- a network structure, a directed acyclic graph $\mathcal{G}$, in which each node corresponds to a random variable X_i;
- a global probability distribution over $\mathbf{X} = \{X_1, \ldots, X_N\}$ which can be factorised into smaller local probability distributions according to the arcs present in the graph.

The main role of the network structure is to express the conditional independence relationships among the variables in the model through graphical separation, thus specifying the factorisation of the global distribution:

$$\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \Pi_{X_i}; \Theta_{X_i}) \quad \text{where} \quad \Pi_{X_i} = \{\text{parents of } X_i\}.$$

Graphs

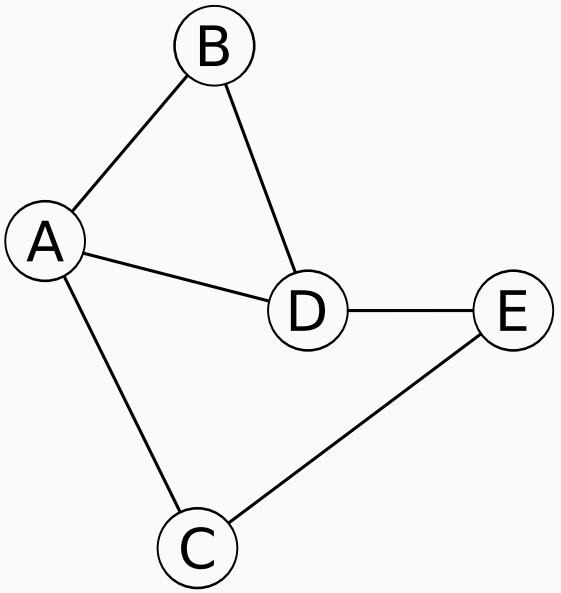
The first component of a BN is a graph. A graph $\mathcal{G}$ is a mathematical object with:

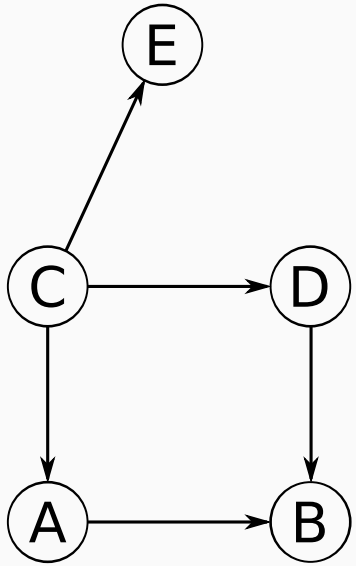
- a set of nodes;
- a set of arcs A which are identified by pairs for nodes.

Given the nodes, a graph is uniquely identified by the arc set. An arc can be:

- undirected if the arc has no direction, for instance $A - B$;
- directed if the arc has a specific direction, for instance $A \rightarrow B$.

The assumption is that there is at most one arc between each, pair of nodes.

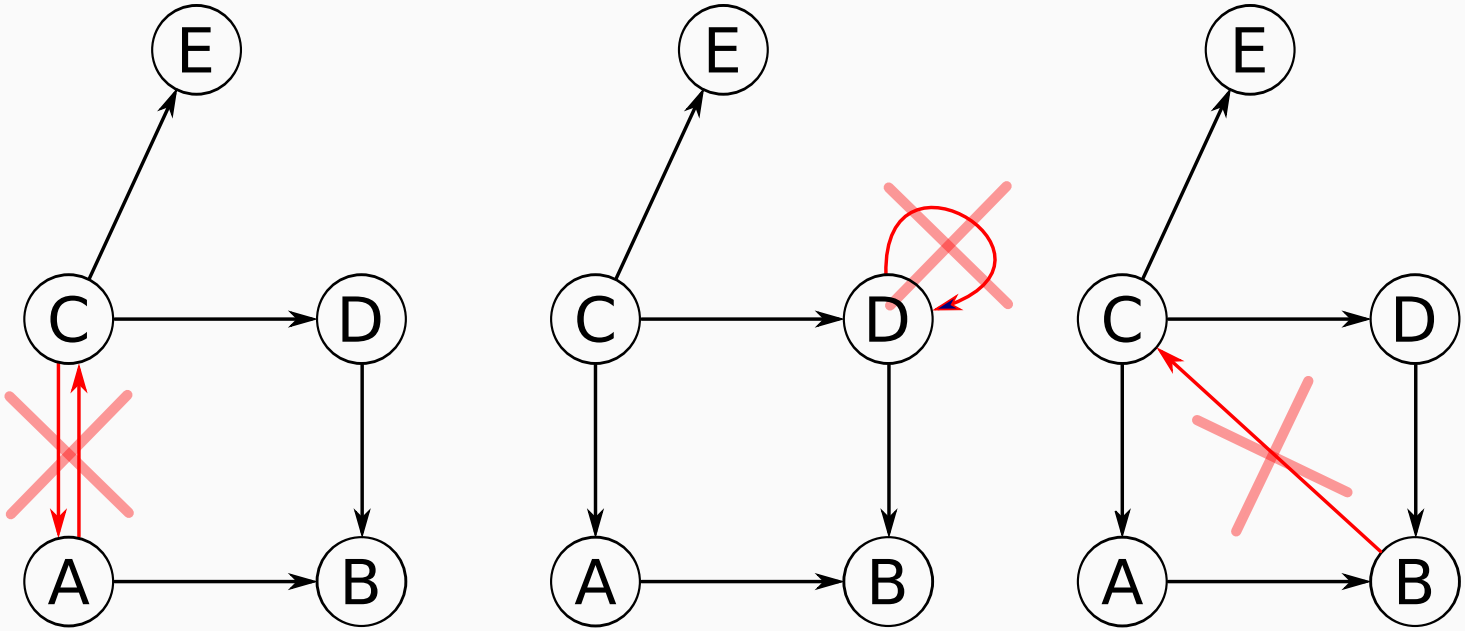




Directed Acyclic Graphs

BNs use a specific kind of graph called a directed acyclic graph (DAG), that:

- contains only directed arcs;
- does not contain any loop (an arc $D \rightarrow D$ from a node to itself);
- does not contain any cycle (a sequence of arcs like $B \rightarrow C \rightarrow D \rightarrow B$ that starts and ends in the same node).



Computer Representations of Graphs

Graphs are uniquely identified by their arcs, and the latter have three possible computer representations:

- arc set:

$$\{A \to B,\ D \to B,\ C \to D,\ C \to A,\ C \to E\};$$

- arc list: in which each node is associated with its children,

$$\{A = \{B\}, B = \{\varnothing\}, C = \{D, A, E\}, D = \{B\}, E = \{\varnothing\}\};$$

- adjacency matrix: in which parents are on the rows and children are on the columns,

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Creating Graphs

- Setting individual arcs.

```
dag = empty.graph(nodes = c("A", "B", "C", "D", "E"))
dag = set.arc(dag, from = "A", to = "B")
dag = set.arc(dag, from = "D", to = "B")
dag = set.arc(dag, from = "C", to = "D")
dag = set.arc(dag, from = "C", to = "A")
dag = set.arc(dag, from = "C", to = "E")
```

- Setting the whole arc set at once.

```
arc.set = matrix(c("A", "B",
                   "D", "B",
                   "C", "D",
                   "C", "A",
                   "C", "E"),
            byrow = TRUE, ncol = 2,
            dimnames = list(NULL, c("from", "to")))
arcs(dag) = arc.set
```

Creating Graphs

- Using the adjacency matrix representation of the arc set.

```
amat(dag) =
  matrix(c(0L, 1L, 0L, 0L, 0L,
           0L, 0L, 0L, 0L, 0L,
           1L, 0L, 0L, 1L, 1L,
           0L, 1L, 0L, 0L, 0L,
           0L, 0L, 0L, 0L, 0L),
    byrow = TRUE, nrow = 5, ncol = 5,
    dimnames = list(nodes(dag), nodes(dag)))
```

- Using the formula representation for the Bayesian network.

```
dag = model2network("[A|C][B|A:D][C][D|C][E|C]")
```

Acyclicity is enforced by all there functions by default:

```
set.arc(dag, from = "B", to = "C")
```

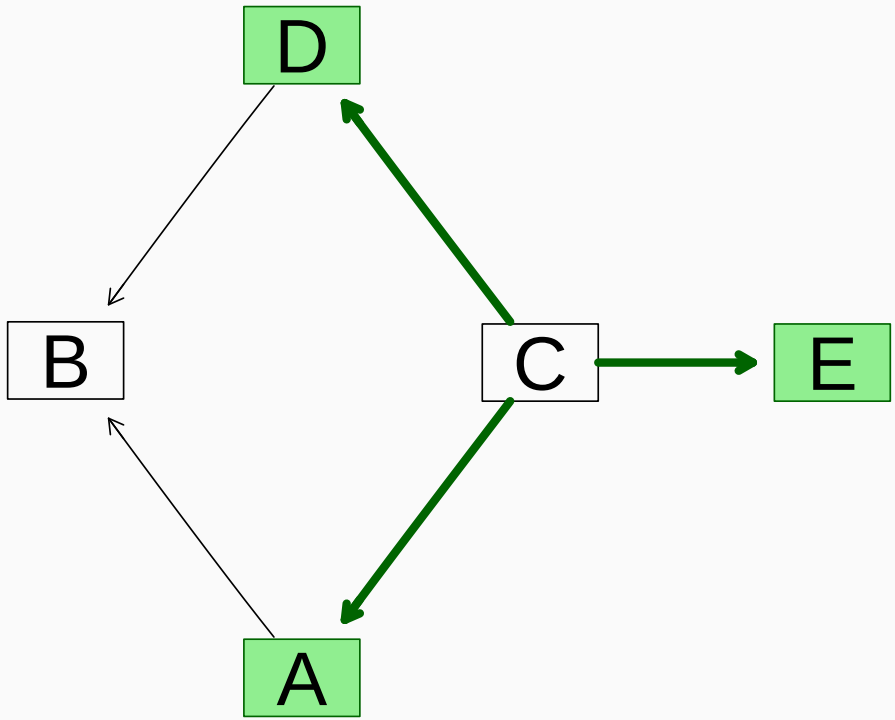
```
  Error in arc.operations(x = x, from = from, to = to, op = "set", check.cycles =
check.cycles, : the resulting graph contains cycles.
```

Plotting Graphs

The main plotting function in **bnlearn** is `graphviz.plot()` which is based on the **Rgraphviz** package and the **Graphviz** library by AT&T.

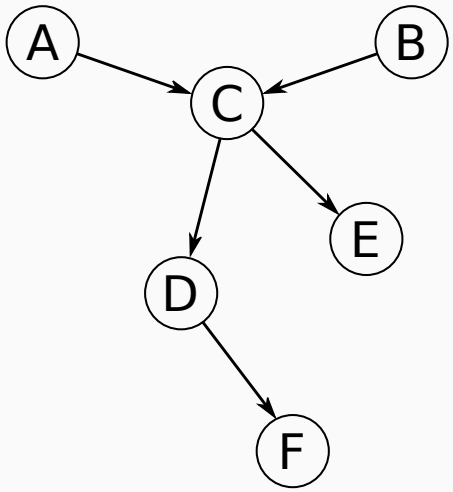
It produces interpretable graph layouts automatically and supports highlighting nodes and arcs.

```
graphviz.plot(dag, layout = "circo",
  shape = "rectangle",
  highlight =
    list(nodes = children(dag, "C"),
         arcs = outgoing.arcs(dag, "C"),
         fill = "palegreen2",
         col = "darkgreen", lwd = 5))
```



How the DAG Maps to the Probability Distribution

DAG



$\Longrightarrow$

Graphical separation

$$
\begin{aligned}
&A \perp\!\!\!\perp_G B \\
&A \perp\!\!\!\perp_G D \mid C \\
&B \perp\!\!\!\perp_G D \mid C \\
&A \perp\!\!\!\perp_G E \mid C \\
&B \perp\!\!\!\perp_G E \mid C \\
&D \perp\!\!\!\perp_G E \mid C \\
&C \perp\!\!\!\perp_G F \mid D \\
&\ldots
\end{aligned}
$$

$\Longrightarrow$

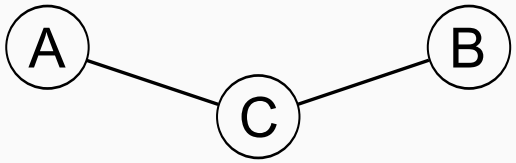
Probabilistic independence

$$
\begin{aligned}
&A \perp\!\!\!\perp_P B \\
&A \perp\!\!\!\perp_P D \mid C \\
&B \perp\!\!\!\perp_P D \mid C \\
&A \perp\!\!\!\perp_P E \mid C \\
&B \perp\!\!\!\perp_P E \mid C \\
&D \perp\!\!\!\perp_P E \mid C \\
&C \perp\!\!\!\perp_P F \mid D \\
&\ldots
\end{aligned}
$$

Formally, the DAG is an independence map of the probability distribution of $\mathbf{X}$, with graphical separation ($\perp\!\!\!\perp_G$) implying probabilistic independence ($\perp\!\!\!\perp_P$).

Graphical Separation in DAGs: Fundamental Connections

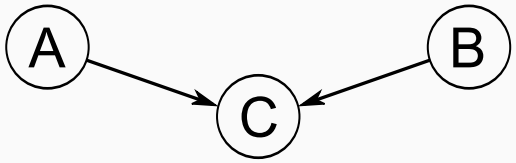
separation (undirected graphs)



$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

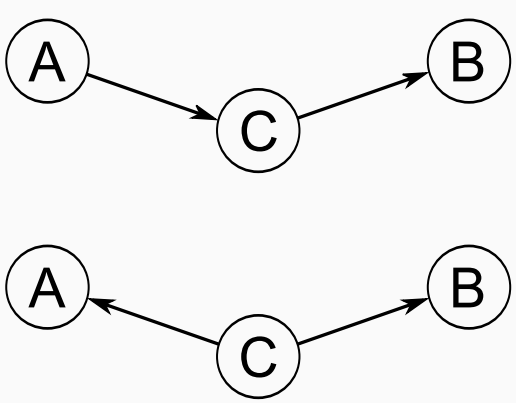
$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C})$$

d-separation (directed acyclic graphs)



$$\mathbf{A} \not\perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{C} \,|\, \mathbf{A}, \mathbf{B})\, \mathrm{P}(\mathbf{A})\, \mathrm{P}(\mathbf{B})$$



$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

$$\begin{aligned} \mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) &= \\ &= \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C} \,|\, \mathbf{A})\, \mathrm{P}(\mathbf{A}) \\ &= \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C}) \end{aligned}$$

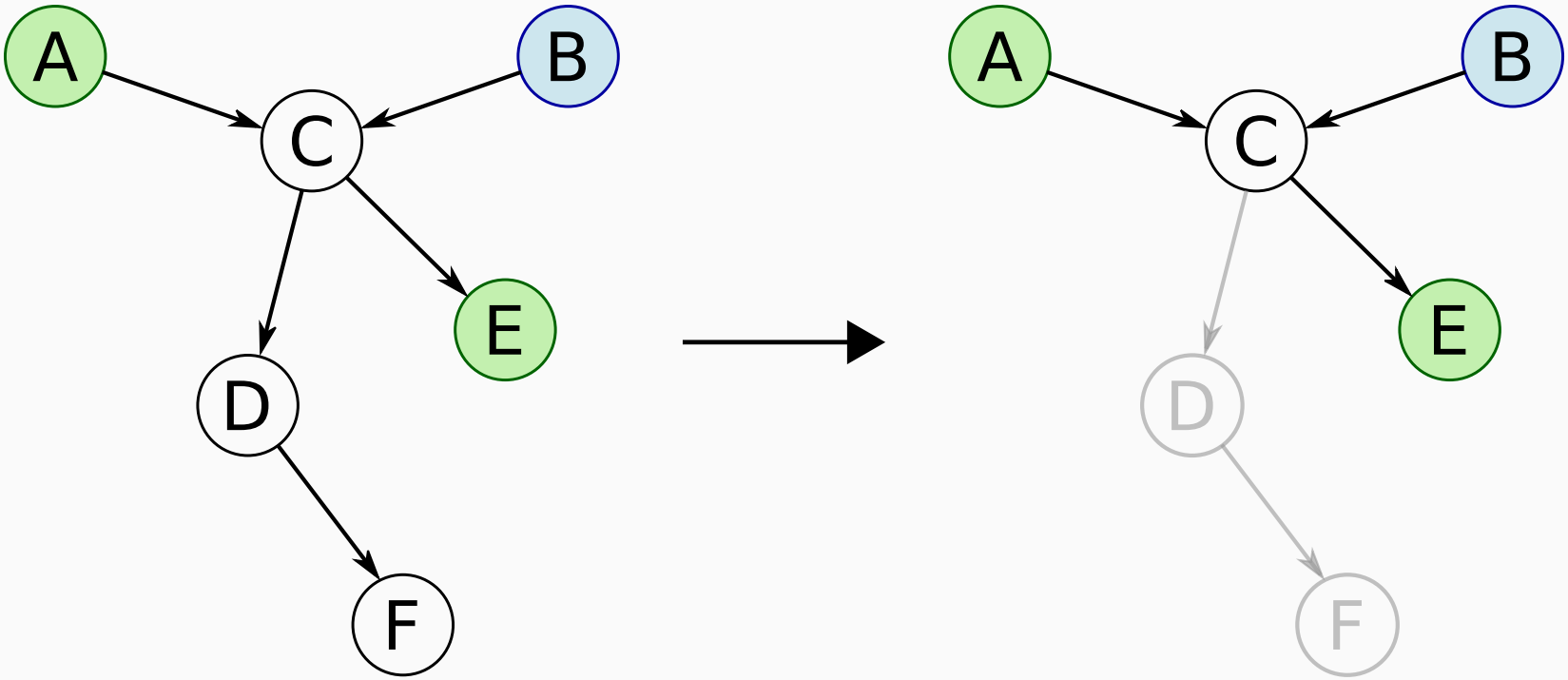
Graphical Separation in DAGs: General Case

Now, in the general case we can extend the patterns from the fundamental connections and apply them to every possible path between $\mathbf{A}$ and $\mathbf{B}$ for a given $\mathbf{C}$; this is how d-separation is defined.

> *If $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are three disjoint subsets of nodes in a directed acyclic graph $\mathcal{G}$, then $\mathbf{C}$ is said to d-separate $\mathbf{A}$ from $\mathbf{B}$, denoted $\mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \mid \mathbf{C}$, if along every path between a node in $\mathbf{A}$ and a node in $\mathbf{B}$ there is a node v satisfying one of the following two conditions:*
>
> 1. *v has converging edges (that is, there are two edges pointing to v from the adjacent nodes in the path) and none of v or its descendants (that is, the nodes that can be reached from v) are in $\mathbf{C}$.*
> 2. *v is in $\mathbf{C}$ and does not have converging edges.*

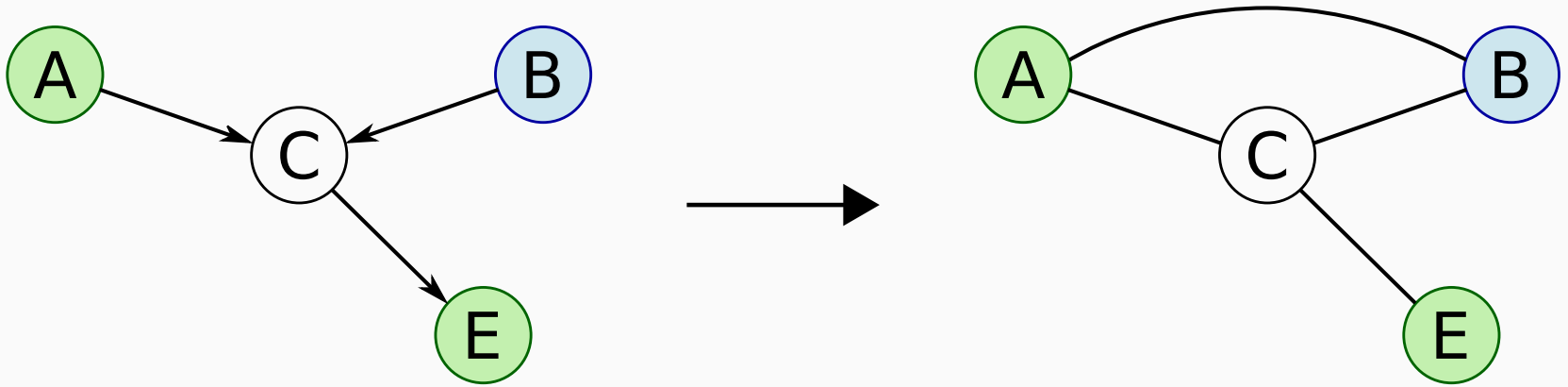
This definition clearly does not provide a computationally feasible approach to assess d-separation; but there are other ways.

A Simple Algorithm to Check D-Separation



Say we want to check whether A and E are d-separated by B. First, we can drop all the nodes that are not ancestors (that is, parents, parents' parents, etc.) of A, E and B since each node only depends on its parents.

A Simple Algorithm to Check D-Separation

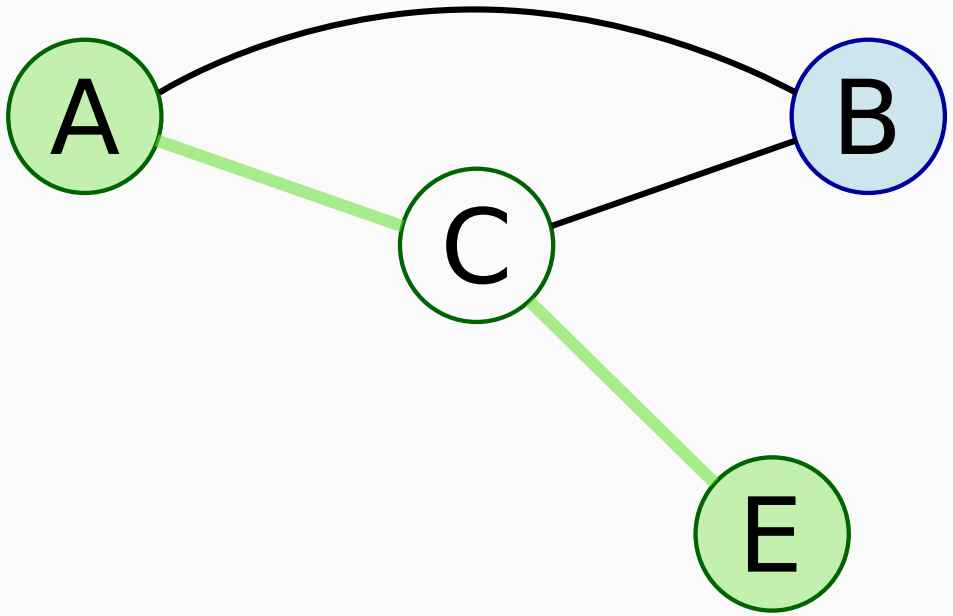


Transform the subgraph into its moral graph by

1. connecting all nodes that have one child in common; and
2. removing all arc directions to obtain an undirected graph.

This transformation has the double effect of making the dependence between parents explicit by "marrying" them and of allowing us to use the classic definition of graphical separation.

A Simple Algorithm to Check D-Separation



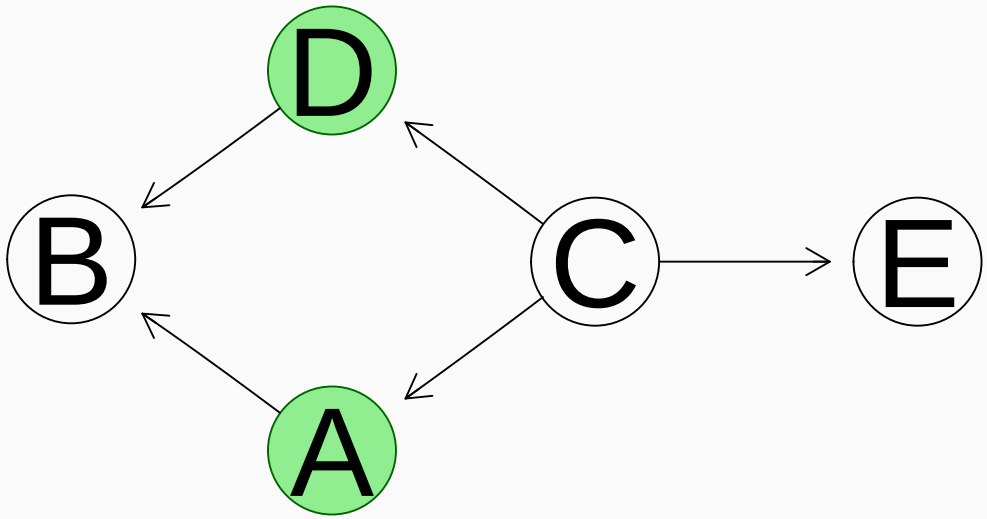
Finally, we can just perform a depth-first or breadth-first search and see if we can find an open path between A and E, that is, a path that is not blocked by B.

D-Separation Example: the DAG We Created Earlier

Say that we want to check whether A and D are d-separated by B in the DAG we created earlier, dag.

0. Take a look at the original DAG: A and D share a child (B) that is part of the d-separating set.

```
graphviz.plot(dag, layout = "circo",
  highlight = list(nodes = c("A", "D"), fill = "palegreen2",
          col = "darkgreen"))
```

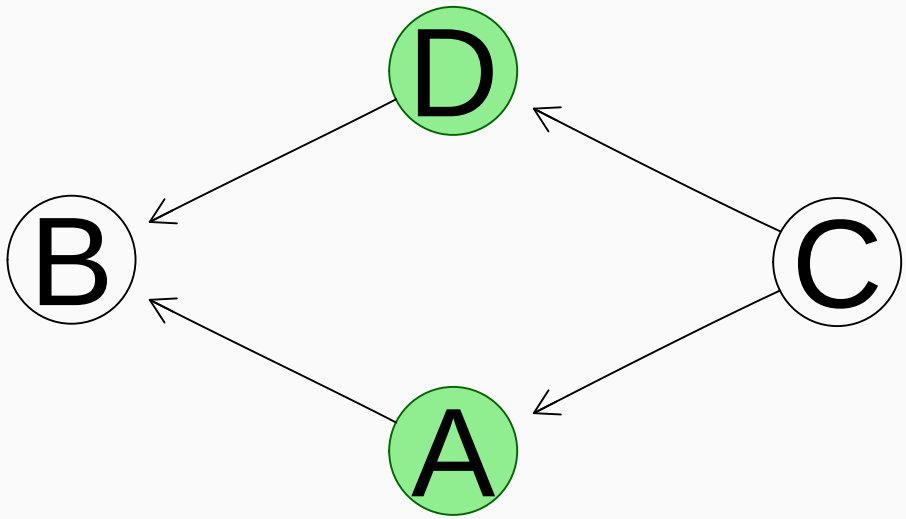


D-Separation Example: the DAG We Created Earlier

1. Drop all the nodes that are not ancestors of A, D, or B, and obviously keep A, D and B!

```
ancestorsA = ancestors(dag, node = "A")
ancestorsD = ancestors(dag, node = "D")
ancestorsB = ancestors(dag, node = "B")
step1 = subgraph(dag, nodes = unique(c("A", "D", "B",
                                ancestorsA,  ancestorsD, ancestorsB)))

graphviz.plot(step1, layout = "circo",
  highlight = list(nodes = c("A", "D"), fill = "palegreen2",
          col = "darkgreen"))
```

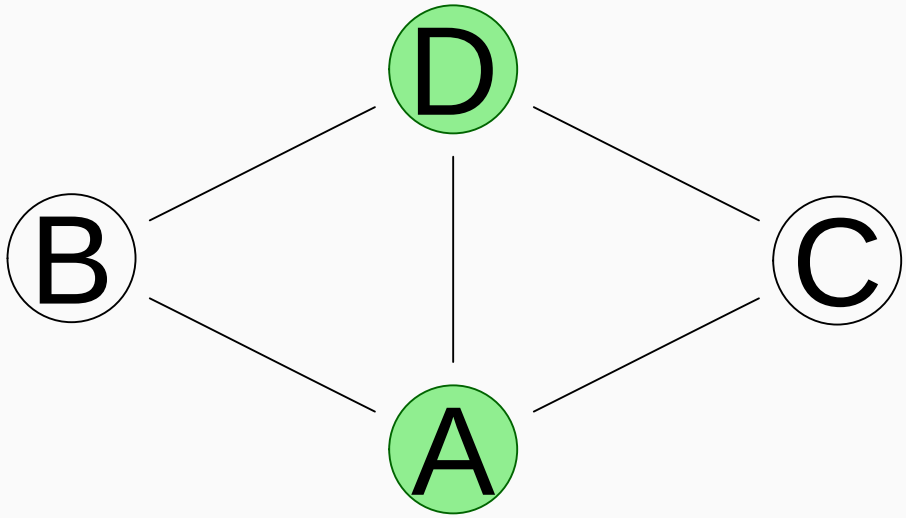


D-Separation Example: the DAG We Created Earlier

2. Transform the DAG `step1` into the moral graph `step2`.

```
step2 = moral(step1)

graphviz.plot(step2, layout = "circo",
  highlight = list(nodes = c("A", "D"), fill = "palegreen2",
          col = "darkgreen"))
```

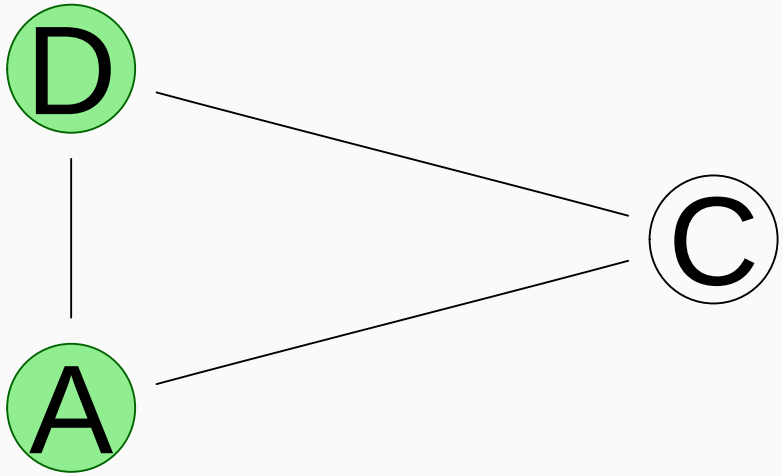


D-Separation Example: the DAG We Created Earlier

3. Remove the d-separating node to remove all the paths that pass through it.

```
step3 = subgraph(step2, nodes = setdiff(nodes(step2), "B"))

graphviz.plot(step3, layout = "circo",
  highlight = list(nodes = c("A", "D"), fill = "palegreen2",
          col = "darkgreen"))
```



D-Separation Example: the DAG We Created Earlier

4. Check whether there is still a path from A to D.

```
path.exists(step3, from = "A", to = "D")
  [1] TRUE
path.exists(step3, from = "D", to = "A")
  [1] TRUE
```

Since step3 is an undirected graphs, if there is a path from A to D there is a path from D to A. This means that d-separation is symmetric:

$$\mathsf{A} \not\perp\!\!\!\perp_G \mathsf{D} \mid \mathsf{B} \iff \mathsf{D} \not\perp\!\!\!\perp_G \mathsf{A} \mid \mathsf{B}$$

Which must be the case because in probability independence is also symmetric:

$$\mathrm{P}(\mathsf{A}, \mathsf{D} \mid \mathsf{B}) = \mathrm{P}(\mathsf{D}, \mathsf{A} \mid \mathsf{B}) \neq \mathrm{P}(\mathsf{A} \mid \mathsf{B})\,\mathrm{P}(\mathsf{D} \mid \mathsf{B}),$$

and d-separation implies probabilistic independence.

Checking D-Separation

Our conclusion is: there is a path between A and D in step3 that does not pass through/is not blocked by B, so A and D are not d-separated and are not conditionally independent.

```
dsep(dag, "A", "D", "B")
```

```
[1] FALSE
```

NOTE: d-separation does not necessarily require a separating set. Or, to put it in another way, the separating set can be empty.

```
dsep(dag, "A", "D")
```

```
[1] FALSE
```

In that case we are checking whether A and D are marginally independent because there is any path at all that connects them.

If we use d-separation as our definition of graphical separation, assuming that the DAG is an independence map leads to the general formulation of the decomposition of the global distribution $\mathrm{P}(\mathbf{X})$:

$$\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \Pi_{X_i})$$

into the local distributions for the X_i given their parents Π_{X_i}. If X_i has two or more parents it depends on their joint distribution, because each pair of parents forms a convergent connection centred on X_i and we cannot establish their independence. This decomposition is preferable to that obtained from the chain rule,

$$\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid X_{i+1}, \ldots, X_N)$$

because the conditioning sets are typically smaller.

The Local Markov Property

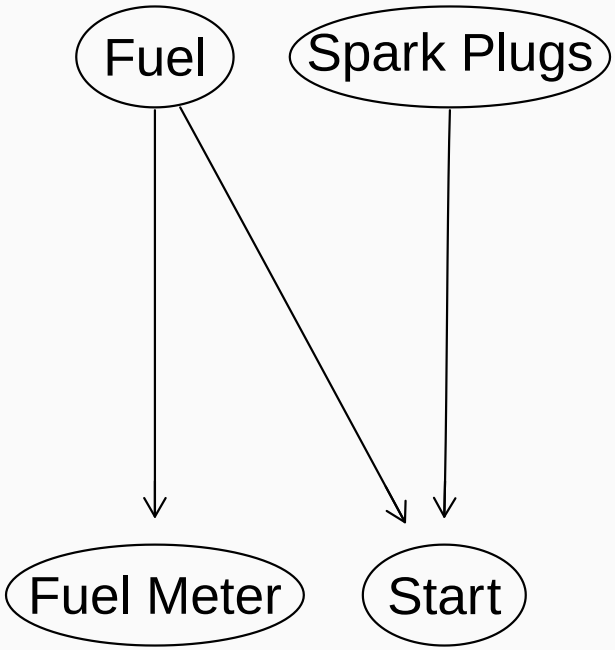
Another result along the same lines is called the local Markov property, which can be combined with the chain rule above to get the decomposition into local distributions.

> *Each node X_i is conditionally independent of its non-descendants (the nodes X_j for which there is no path from X_i to X_j) given its parents.*

Compared to the previous decomposition, it highlights the fact that parents are not completely independent from their children in the BN; a trivial application of Bayes' theorem to invert the direction of the conditioning shows how information on a child can change the distribution of the parent.

The Local Markov Property: Car Start

```
graphviz.plot(car.start,
  shape = "ellipse")
```



Printing the parents of each node:

```
for (node in nodes(car.start))
  cat(node, "has parents:",
    parents(car.start, node), "\n")
```

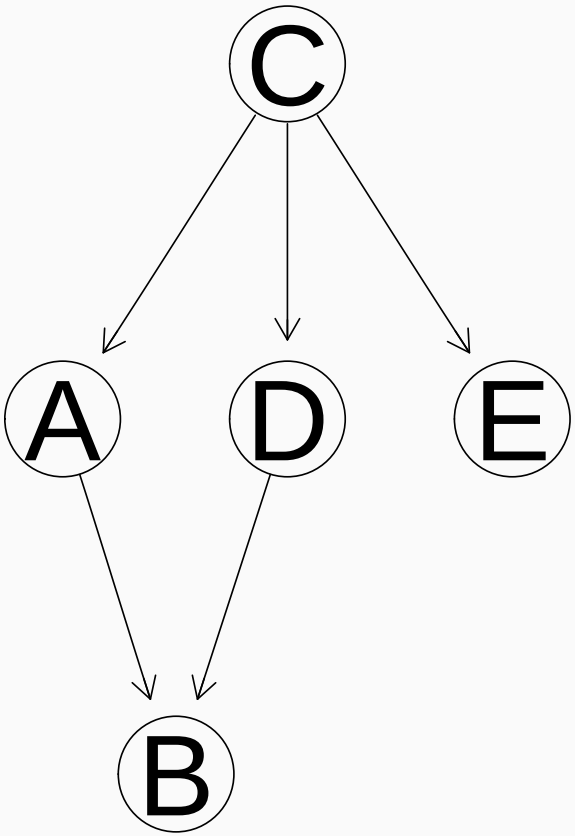
```
Fuel has parents:
Fuel Meter has parents: Fuel
Spark Plugs has parents:
Start has parents: Fuel Spark Plugs
```

The corresponding decomposition:

$$\begin{aligned}\mathrm{P}(\texttt{Start}, \texttt{Fuel Meter}, \texttt{Fuel},\\ \texttt{Spark Plugs}) =\\ \mathrm{P}(\texttt{Start} \mid \texttt{Fuel}, \texttt{Spark Plugs})\\ \mathrm{P}(\texttt{Fuel Meter} \mid \texttt{Fuel}) \times\\ \mathrm{P}(\texttt{Fuel})\,\mathrm{P}(\texttt{Spark Plugs})\end{aligned}$$

The Local Markov Property: the DAG We Created Earlier

```
graphviz.plot(dag)
```



Printing the parents of each node:

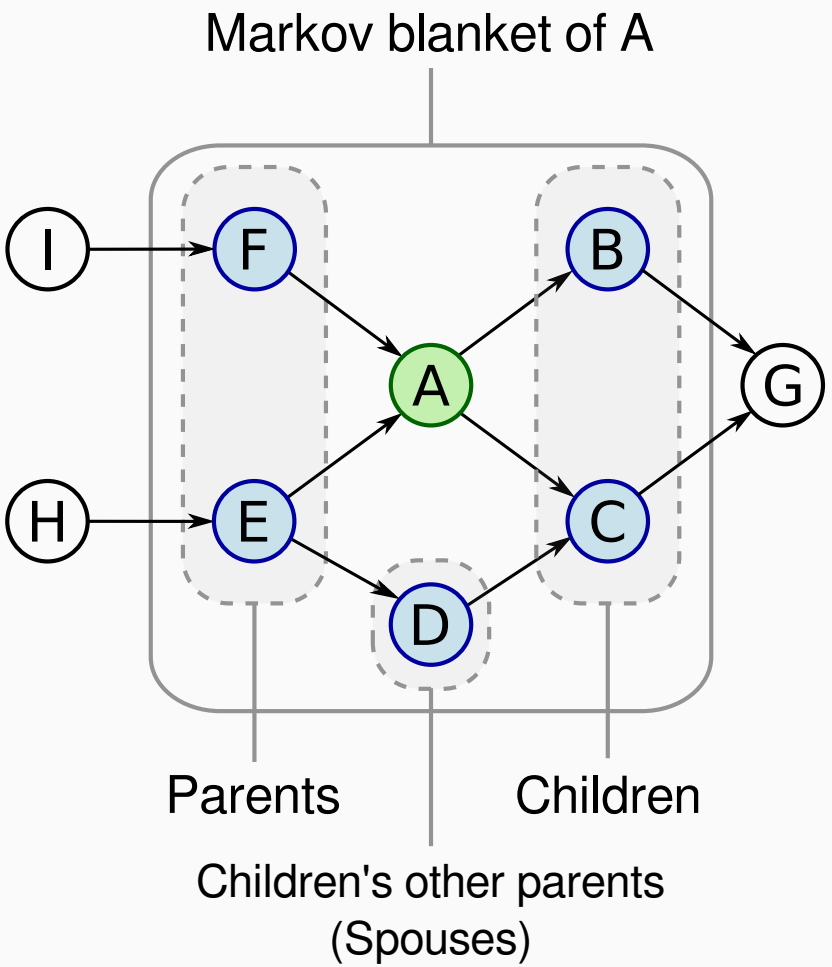
```
for (node in nodes(dag))
  cat(node, "has parents:",
    parents(dag, node), "\n")
```

```
A has parents: C
B has parents: A D
C has parents:
D has parents: C
E has parents: C
```

The corresponding decomposition:

$$\mathrm{P}(\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \mathsf{E}) = \mathrm{P}(\mathsf{A} \mid \mathsf{C})\, \mathrm{P}(\mathsf{B} \mid \mathsf{A}, \mathsf{D})\, \mathrm{P}(\mathsf{C})\, \mathrm{P}(\mathsf{D} \mid \mathsf{C})\, \mathrm{P}(\mathsf{E} \mid \mathsf{C})$$

Completely D-Separating: Markov Blankets



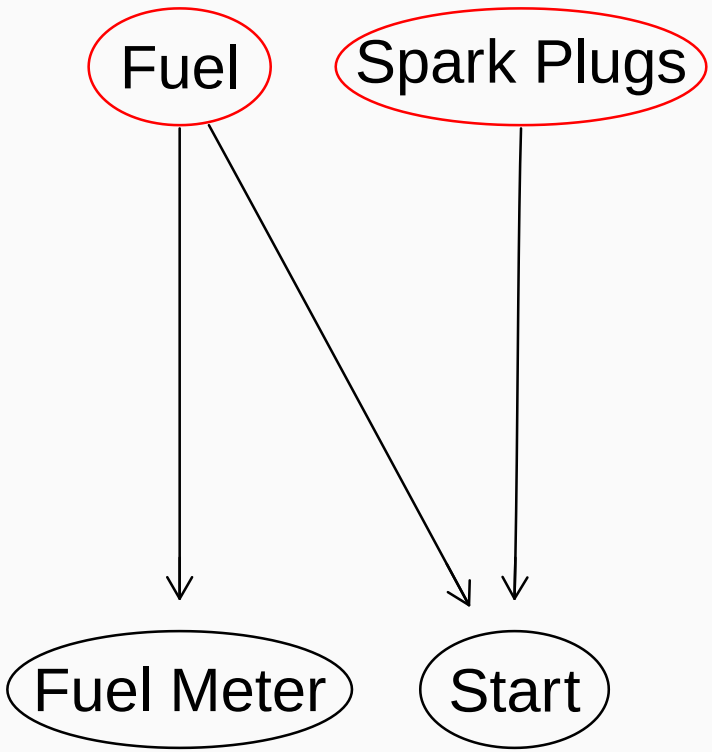
We can easily use the DAG to solve the feature selection problem. The set of nodes that graphically isolates a target node from the rest of the DAG is called its Markov blanket and includes:

- its parents;
- its children;
- other nodes sharing a child.

Since $\perp\!\!\!\perp_G$ implies $\perp\!\!\!\perp_P$, we can restrict ourselves to the Markov blanket to perform any kind of inference on the target node, and disregard the rest.

Markov Blanket: Car Start

```
mbS = mb(car.start, node = "Start")
graphviz.plot(car.start,
  shape = "ellipse",
  highlight = list(nodes = mbS))
```



Printing the parents, children and spouses of `Start`:

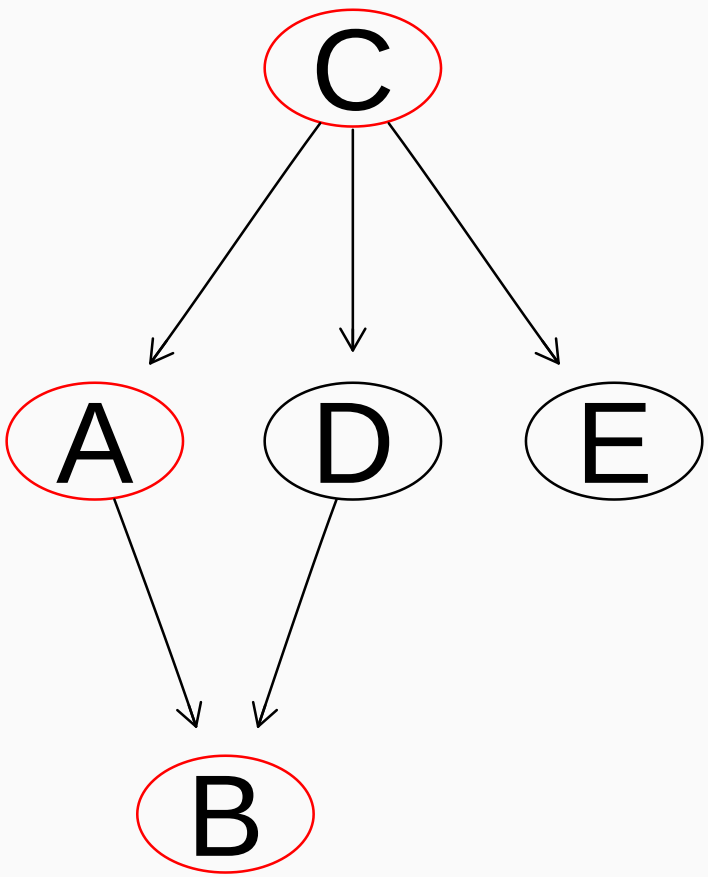
```
parents(car.start, node = "Start")
|  [1] "Fuel"         "Spark Plugs"
children(car.start, node = "Start")
|  character(0)
spouses(car.start, node = "Start")
|  character(0)
```

In one go:

```
mb(car.start, node = "Start")
|  [1] "Fuel"         "Spark Plugs"
```

Markov Blanket: The DAG we created earlier

```
mbD = mb(dag, node = "D")
graphviz.plot(dag,
  shape = "ellipse",
  highlight = list(nodes = mbD))
```



Printing the parents, children and spouses of `Start`:

```
parents(dag, node = "D")
|  [1] "C"
children(dag, node = "D")
|  [1] "B"
spouses(dag, node = "D")
|  [1] "A"
```

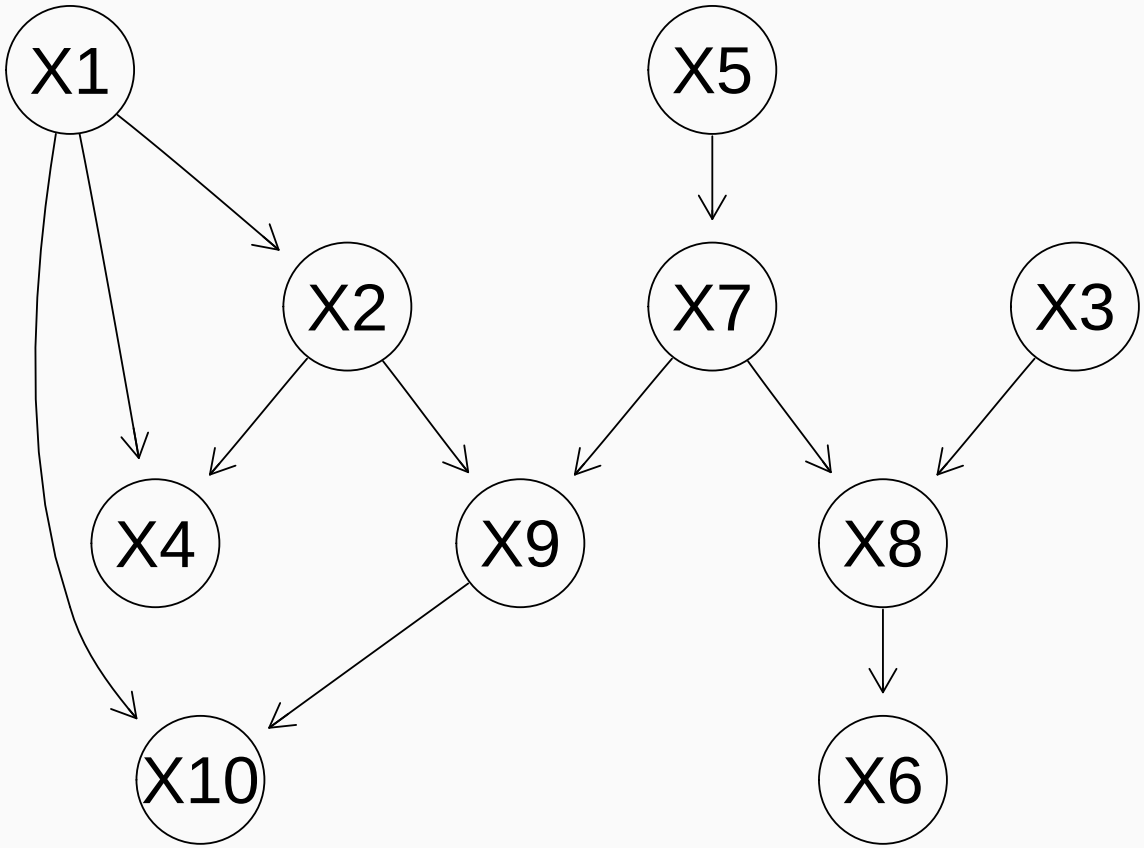
In one go:

```
mb(dag, node = "D")
|  [1] "A" "B" "C"
```

Different DAGs, Same Distribution

A DAG uniquely identifies a factorisation of $\mathrm{P}(\mathbf{X})$; the converse is not necessarily true. Consider this DAG:

```
large = model2network(paste0("[X1][X3][X5][X6|X8][X2|X1][X7|X5][X4|X1:X2]",
         "[X8|X3:X7][X9|X2:X7][X10|X1:X9]"))
graphviz.plot(large)
```



Different DAGs, Same Distribution

The decomposition into local distributions is:

$$\begin{aligned}&\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ &\quad \mathrm{P}(X_1)\,\mathrm{P}(X_3)\,\underbrace{\mathrm{P}(X_5)}_{X_5}\,\mathrm{P}(X_6 \mid X_8)\,\mathrm{P}(X_2 \mid X_1)\,\underbrace{\mathrm{P}(X_7 \mid X_5)}_{X_5 \to X_7} \\ &\quad \mathrm{P}(X_4 \mid X_1, X_2)\,\mathrm{P}(X_8 \mid X_3, X_7)\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

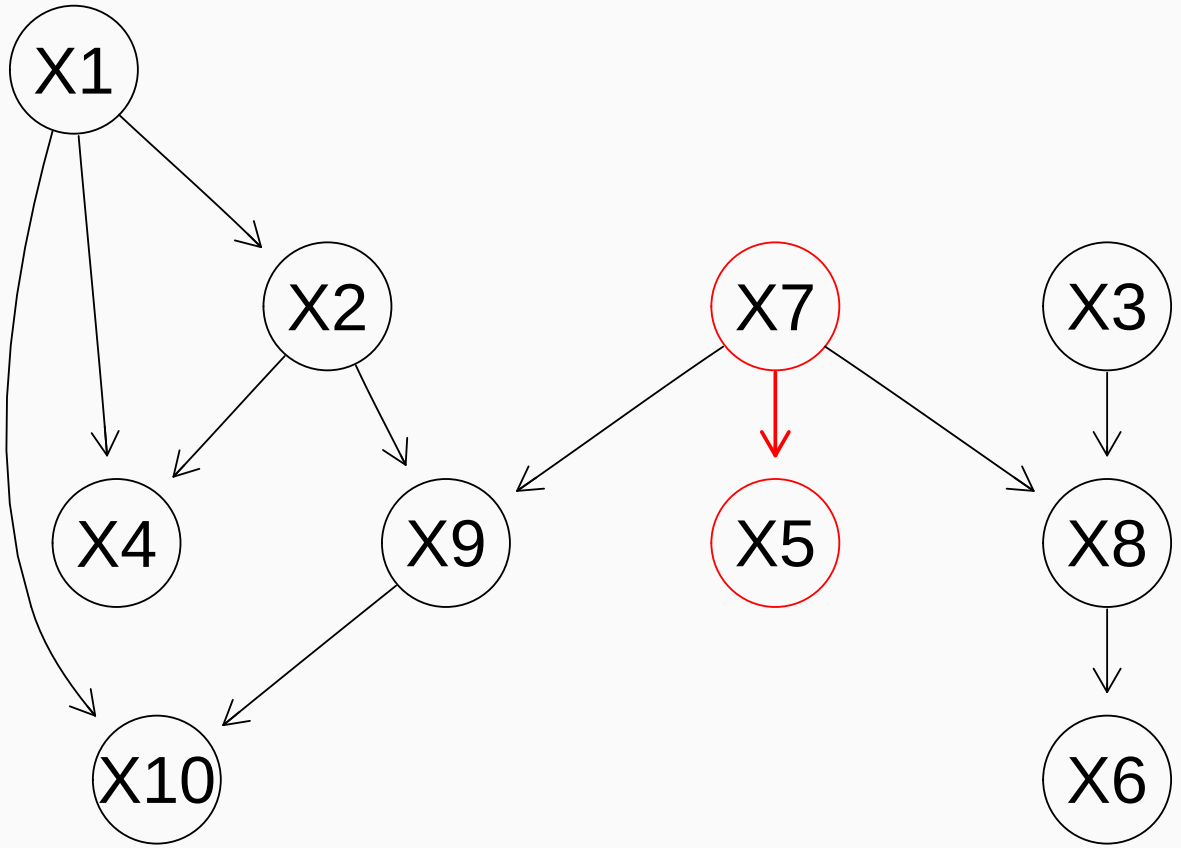
However, look at $X_5 \to X_7$: $\mathrm{P}(X_7 \mid X_5)\,\mathrm{P}(X_5) = \mathrm{P}(X_5 \mid X_7)\,\mathrm{P}(X_7)$ by Bayes' theorem. Then

$$\begin{aligned}&\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ &\quad \mathrm{P}(X_1)\,\mathrm{P}(X_3)\,\underbrace{\mathrm{P}(X_7)}_{X_7}\,\mathrm{P}(X_6 \mid X_8)\,\mathrm{P}(X_2 \mid X_1)\,\underbrace{\mathrm{P}(X_5 \mid X_7)}_{X_7 \to X_5} \\ &\quad \mathrm{P}(X_4 \mid X_1, X_2)\,\mathrm{P}(X_8 \mid X_3, X_7)\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

Different DAGs, Same Distribution

The DAG that gives this new, equivalent decomposition is:

```
large2 = reverse.arc(large, from = "X5", to = "X7")
graphviz.plot(large2, highlight = list(nodes = c("X5", "X7"),
                                       arcs = c("X7", "X5"), lwd = 2))
```



Next let's look at $X_8 \to X_6$.

$$\begin{aligned}\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) = \\ \mathrm{P}(X_1)\,\mathrm{P}(X_3)\,\mathrm{P}(X_7)\,\underbrace{\mathrm{P}(X_6 \mid X_8)}_{X_8 \to X_6}\,\mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_5 \mid X_7) \\ \mathrm{P}(X_4 \mid X_1, X_2)\,\underbrace{\mathrm{P}(X_8 \mid X_3, X_7)}_{X_8 \leftarrow X_3, X_8 \leftarrow X_7}\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).\end{aligned}$$

We cannot reverse the $X_8 \to X_6$ as we did with $X_5 \to X_7$ without changing the probability distribution. If we try, we get

$$\mathrm{P}(X_6 \mid X_8)\,\mathrm{P}(X_8 \mid X_3, X_7) = \mathrm{P}(X_8 \mid X_6)\,\mathrm{P}(X_6)\frac{\mathrm{P}(X_8 \mid X_3, X_7)}{\mathrm{P}(X_8)},$$

which does not simplify because X_8 has other parents (X_3, X_7).

Different DAGs, Same Distribution

Finally, let's look at X_1, X_2 and X_4.

$$\mathrm{P}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}) =$$
$$\underbrace{\mathrm{P}(X_1)}_{X_1} \mathrm{P}(X_3)\,\mathrm{P}(X_5)\,\mathrm{P}(X_6 \mid X_8)\,\underbrace{\mathrm{P}(X_2 \mid X_1)}_{X_1 \to X_2}\,\mathrm{P}(X_7 \mid X_5)$$
$$\underbrace{\mathrm{P}(X_4 \mid X_1, X_2)}_{X_1 \to X_4, X_2 \to X4}\,\mathrm{P}(X_8 \mid X_3, X_7)\,\mathrm{P}(X_9 \mid X_2, X_7)\,\mathrm{P}(X_{10} \mid X_1, X_9).$$

By Bayes' theorem we can say

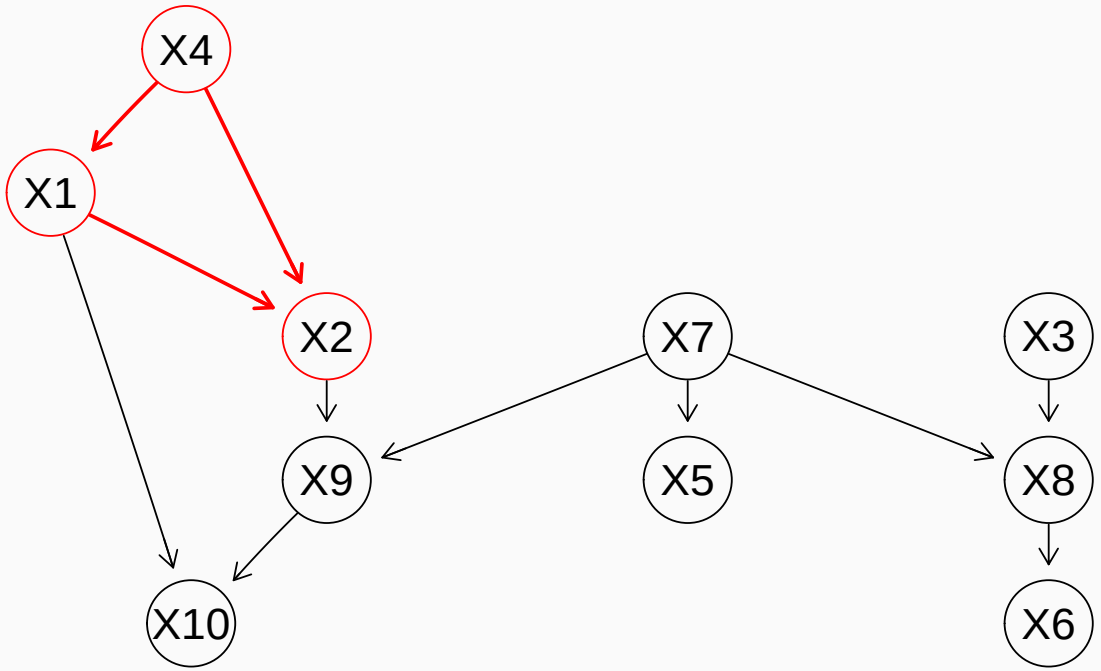
$$\mathrm{P}(X_1)\,\mathrm{P}(X_2 \mid X_1)\,\mathrm{P}(X_4 \mid X_1, X_2) = \mathrm{P}(X_1, X_2, X_4) =$$
$$\underbrace{\mathrm{P}(X_2)}_{X_2}\,\underbrace{\mathrm{P}(X_2 \mid X_4)}_{X_4 \to X_2}\,\underbrace{\mathrm{P}(X_1 \mid X_2, X_4)}_{X_1 \leftarrow X_2, X_1 \leftarrow X_4}$$

which gives us another DAG again.

Different DAGs, Same Distribution

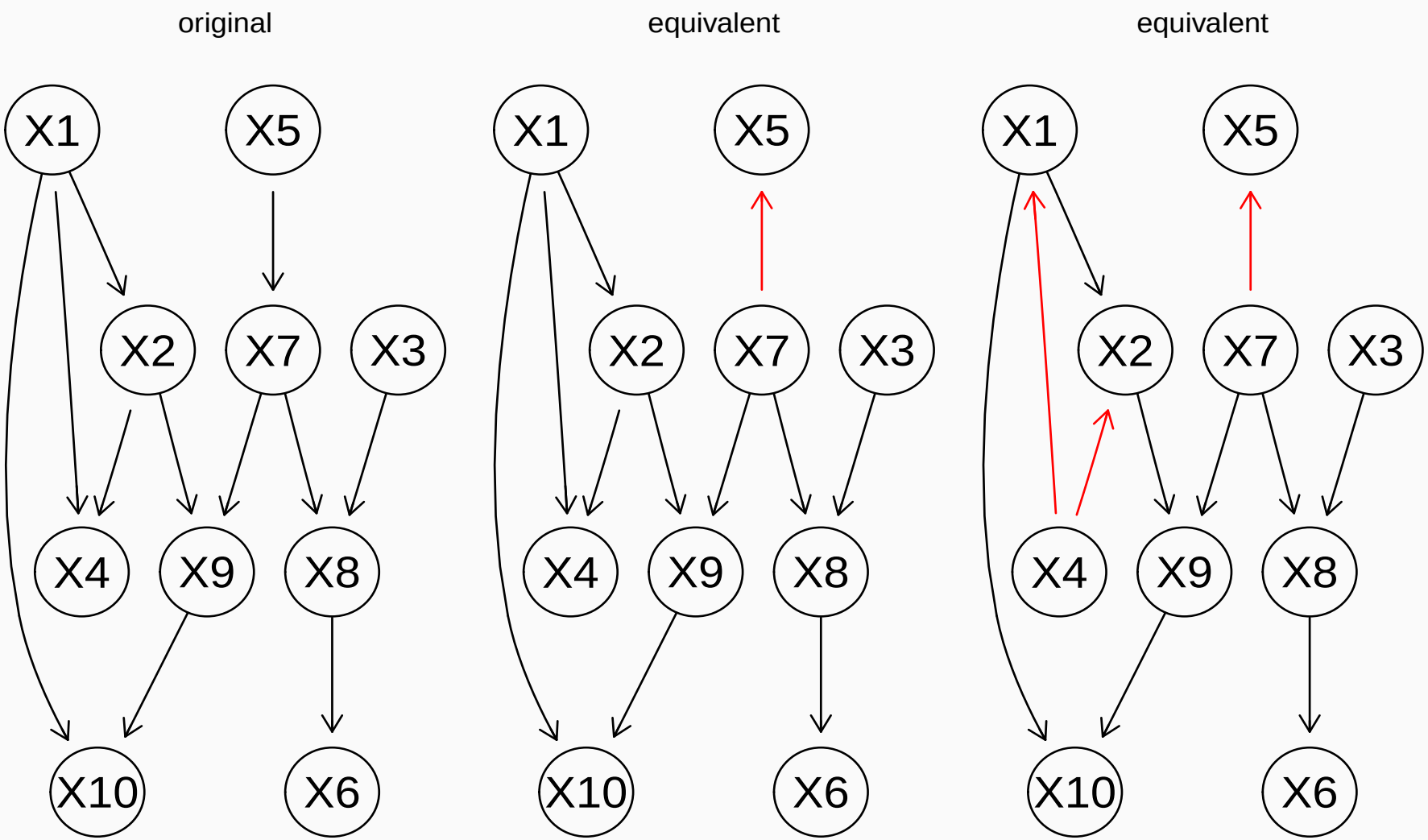
The DAG that gives this last equivalent decomposition is:

```
large3 = set.arc(large2, from = "X1", to = "X2")
large3 = set.arc(large3, from = "X4", to = "X2")
large3 = set.arc(large3, from = "X4", to = "X1")
graphviz.plot(large3, highlight =
  list(nodes = c("X1", "X2", "X4"),
       arcs = arcs(subgraph(large3, c("X1", "X2", "X4"))), lwd = 2))
```



Comparing These Different DAGs

```
par(mfrow = c(1, 3))
graphviz.compare(large, large2, large3,
  main = c("original", "equivalent", "equivalent"))
```



Different DAGs, Same Distribution: Equivalence Classes

To sum it up: we can reverse a number of arcs without changing the dependence structure of $\mathbf{X}$. Since the fundamental connections $A \to C \to B$ and $A \leftarrow C \to B$ are probabilistically equivalent, we can reverse the directions of their arcs as we like as long as we do not create any new v-structure ($A \to C \leftarrow B$, with no arc between A and B).

This means that we can group DAGs into equivalence classes that are uniquely identified by the underlying undirected graph and the v-structures. The directions of other arcs can be either:

- uniquely identifiable because one of the directions would introduce cycles or new v-structures (compelled arcs);
- completely undetermined.

The result is a completed partially directed graph (CPDAG).

What Are V-Structures, and What Are Not

It is important to note that even though $A \to C \leftarrow B$ is a convergent connection, it is not a v-structure if A and C are connected by $A \to B$ or $B \to A$. As a result, we are no longer able to identify which nodes are the parents in the connection.

For instance:

$$\underbrace{\mathrm{P}(A)\,\mathrm{P}(B \mid A)\,\mathrm{P}(C \mid A, B)}_{A \to C \leftarrow B,\, A \to B} = \mathrm{P}(A)\,\frac{\mathrm{P}(B, A)}{\mathrm{P}(A)}\,\frac{\mathrm{P}(C, A, B)}{\mathrm{P}(A, B)} =$$

$$= \mathrm{P}(A)\,\mathrm{P}(C, B \mid A) = \underbrace{\mathrm{P}(A)\,\mathrm{P}(B \mid C, A)\,\mathrm{P}(C \mid A)}_{C \to B \leftarrow A,\, A \to C}.$$

Therefore, the fact that the two parents in a convergent connection are not connected by an arc is crucial in the identification of the correct CPDAG.

Our DAG

From this description we can tell different groups of arcs apart:

```
directed.arcs(large)
```

```
      from to
 [1,] "X8" "X6"
 [2,] "X1" "X2"
 [3,] "X5" "X7"
 [4,] "X1" "X4"
 [5,] "X2" "X4"
 [6,] "X3" "X8"
 [7,] "X7" "X8"
 [8,] "X2" "X9"
 [9,] "X7" "X9"
[10,] "X1" "X10"
[11,] "X9" "X10"
```

```
undirected.arcs(large)
```

```
      from to
```

```
compelled.arcs(large)
```

```
      from to
[1,] "X1" "X10"
[2,] "X2" "X9"
[3,] "X3" "X8"
[4,] "X7" "X8"
[5,] "X7" "X9"
[6,] "X8" "X6"
[7,] "X9" "X10"
```

```
vstructs(large)
```

```
      X    Z     Y
[1,] "X1" "X10" "X9"
[2,] "X3" "X8"  "X7"
[3,] "X2" "X9"  "X7"
```

The Corresponding CPDAG

Which in the corresponding CPDAG become:

```
directed.arcs(cpdag(large))
```

```
      from to
 [1,] "X1" "X10"
 [2,] "X2" "X9"
 [3,] "X3" "X8"
 [4,] "X7" "X8"
 [5,] "X7" "X9"
 [6,] "X8" "X6"
 [7,] "X9" "X10"
```

```
undirected.arcs(cpdag(large))
```

```
      from to
 [1,] "X1" "X2"
 [2,] "X1" "X4"
 [3,] "X2" "X1"
 [4,] "X2" "X4"
 [5,] "X4" "X1"
 [6,] "X4" "X2"
 [7,] "X5" "X7"
 [8,] "X7" "X5"
```

```
compelled.arcs(cpdag(large))
```

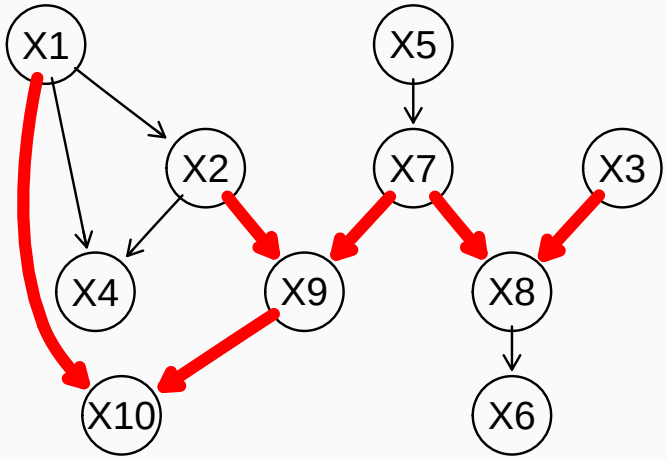
```
      from to
 [1,] "X1" "X10"
 [2,] "X2" "X9"
 [3,] "X3" "X8"
 [4,] "X7" "X8"
 [5,] "X7" "X9"
 [6,] "X8" "X6"
 [7,] "X9" "X10"
```

```
vstructs(cpdag(large))
```

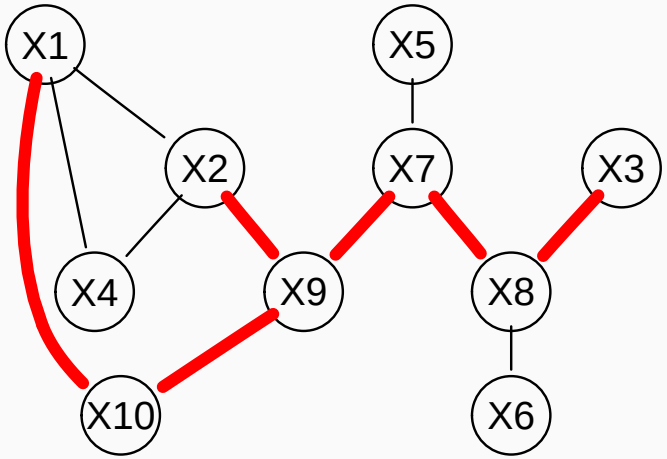
```
      X    Z     Y
 [1,] "X1" "X10" "X9"
 [2,] "X3" "X8"  "X7"
 [3,] "X2" "X9"  "X7"
```

DAG, CPDAG and Equivalent DAGs

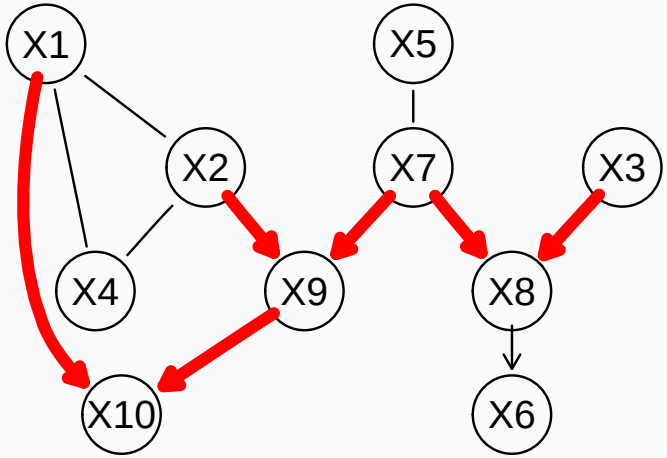
DAG



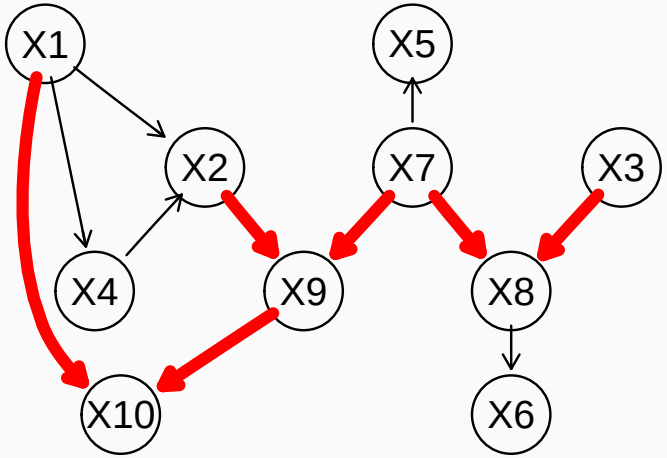
Skeleton



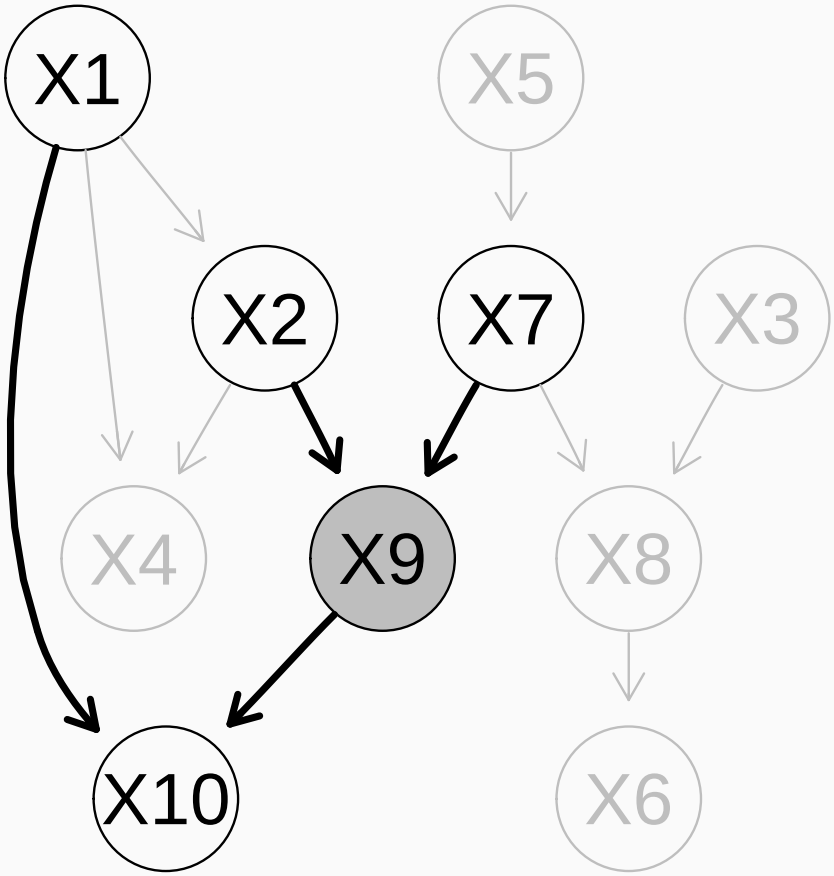
CPDAG

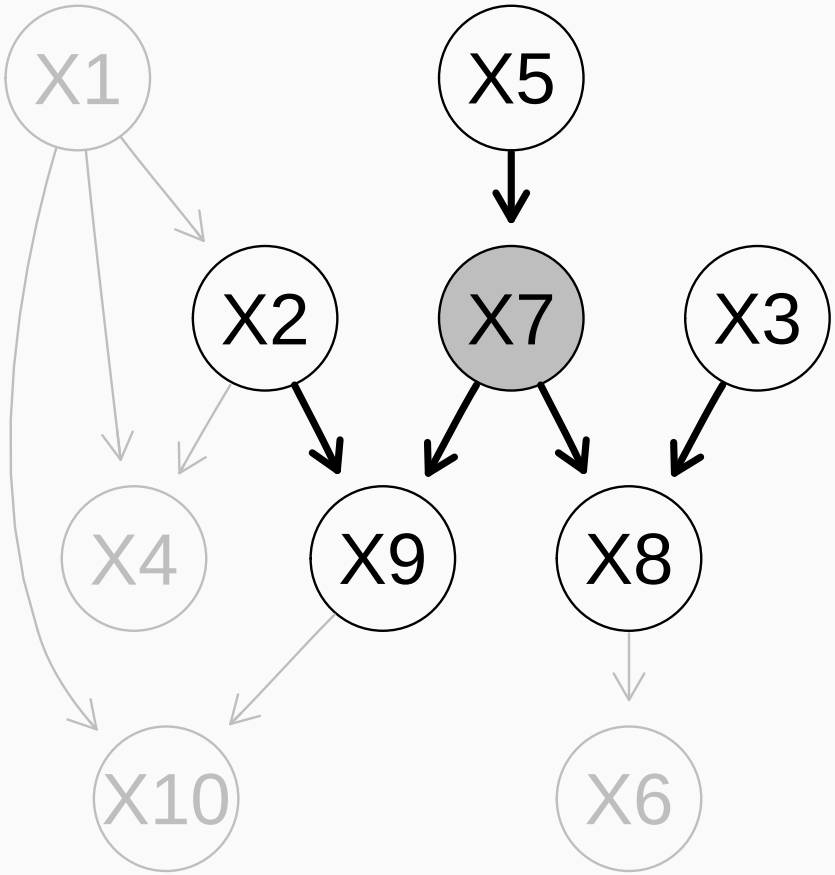


An Equivalent DAG



Two More Examples of Markov Blankets





Two More Examples of Markov Blankets

We can verify again that the Markov blanket (of X_9) contains the children, the parents and the spouses of the node it is centred on...

```
mb(large, node = "X9")
```

```
  [1] "X1"  "X10" "X2"  "X7"
```

```
par.X9 = parents(large, node = "X9")
ch.X9 = children(large, node = "X9")
sp.X9 = sapply(ch.X9, parents, x = large)
sp.X9 = sp.X9[sp.X9 != "X9"]
unique(c(par.X9, ch.X9, sp.X9))
```

```
  [1] "X2"  "X7"  "X10" "X1"
```

... and that it does not contain the node itself. Same for X_7.

```
mb(large, node = "X7")
```

```
  [1] "X2" "X3" "X5" "X8" "X9"
```

```
par.X7 = parents(large, node = "X7")
ch.X7 = children(large, node = "X7")
sp.X7 = sapply(ch.X7, parents, x = large)
sp.X7 = sp.X7[sp.X7 != "X7"]
unique(c(par.X7, ch.X7, sp.X7))
```

```
  [1] "X5" "X8" "X9" "X3" "X2"
```

Markov Blankets are Symmetric

We can also check that Markov blankets are symmetric: if A is in the Markov blanket of B, then B is in the Markov blanket of A.

```
sapply(nodes(large), function(node) node %in% mb(large, node = "X9"))
```

```
   X1   X10    X2    X3    X4    X5    X6    X7    X8    X9
 TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

```
sapply(nodes(large), function(node) "X9" %in% mb(large, node = node))
```

```
   X1   X10    X2    X3    X4    X5    X6    X7    X8    X9
 TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

This is a consequence of the fact that if A is a parent of B, then B is a child of A; and if A is a spouse of B, then B is a spouse of A.

R Programming: a Summary

- creating DAGs: `empty.graph()`, `set.arc()`, `drop.arc()`, `reverse.arc()`.
- model string representations: `modelstring()`, `model2network()`.
- nodes in a DAG: `nodes()`, `parents()`, `children()`, `spouses()`, `nbr()`, `mb()`.
- arcs in a DAG: `arcs()`, `path.exists()`, `dsep()`, `directed.arcs()`, `undirected.arcs()`, `compelled.arcs()`.
- DAG transformation: `subgraph()`, `moral()`, `cpdag()`.
- plotting: `graphviz.plot()`, `graphviz.compare()`.

Summary and Remarks

- BNs are a probabilistic model that use DAGs to make computations systematic in a rigorous way.
- BNs allow computer systems to perform automatically all the computations we did by hand at the beginning of the course.
- At the same time, BNs using DAGs means that they provide a qualitative, intuitive way to reason about complex phenomena.

Next:

- How do we construct a BN?
- How do we make a computer system answer questions using a BN?

Constructing a Bayesian Network

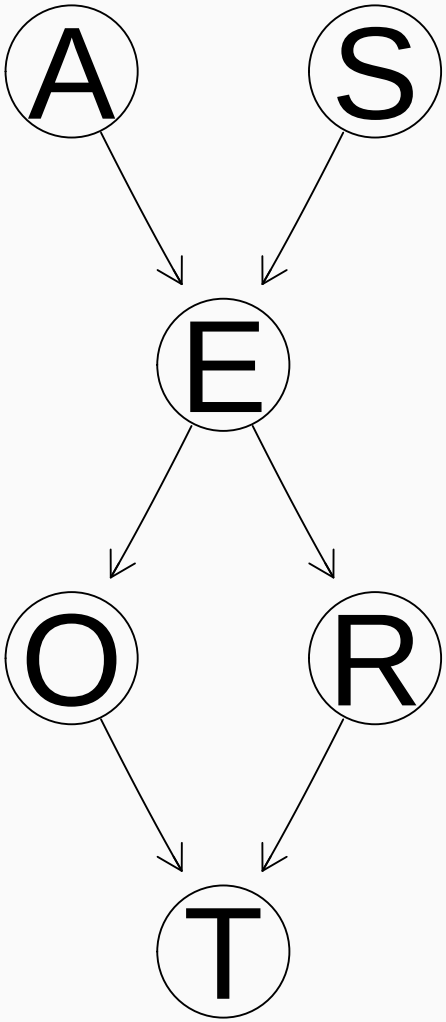
An Example: Train Use Survey

Consider a simple survey whose aim is to investigate the usage patterns of different means of transport, with a focus on cars and trains.

- Age (A): *young* for individuals below 30 years old, *adult* for individuals between 30 and 60 years old, and *old* for people older than 60.
- Sex (S): *male* or *female*.
- Education (E): *up to high school* or *university degree*.
- Occupation (O): *employee* or *self-employed*.
- Residence (R): the size of the city the individual lives in, recorded as either *small* or *big*.
- Travel (T): the means of transport favoured by the individual, recorded either as *car*, *train* or *other*.

The nature of the variables recorded in the survey suggests how they may be related with each other.

The Train Use Survey as a BN (v1)

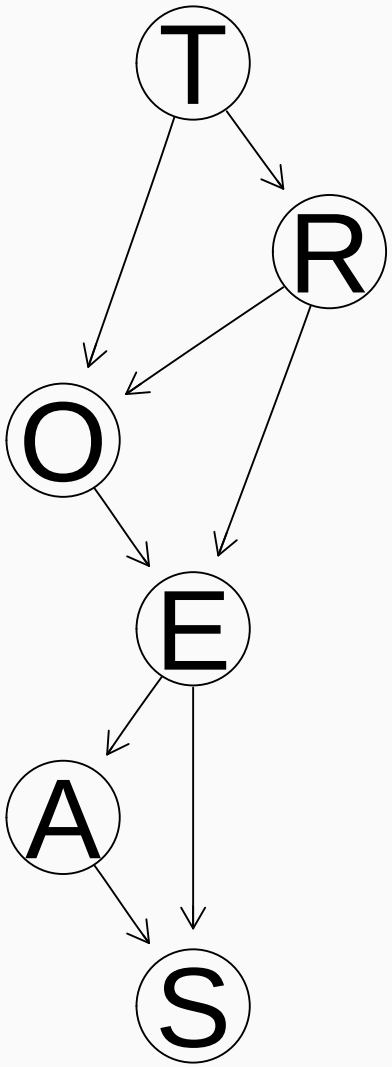


That is a prognostic view of the survey as a BN:

1. the blocks in the experimental design on top (e.g. stuff from the registry office);
2. the variables of interest in the middle (e.g. socio-economic indicators);
3. the object of the survey at the bottom (e.g. means of transport).

Variables that can be thought as "causes" are on above variables that can be considered their "effect", and confounders are on above everything else.

The Train Use Survey as a BN (v2)



That is a diagnostic view of the survey as a BN: it encodes the same dependence relationships as the prognostic view but is laid out to have "effects" on top and "causes" at the bottom.

Depending on the phenomenon and the goals of the survey, one may have a graph that makes more sense than the other; but they are equivalent for any subsequent inference.

Creating the Survey Network

Firstly, we create a DAG with one node for each variable in the survey and no arcs.

```
survey.dag = empty.graph(nodes = c("A", "S", "E", "O", "R", "T"))
```

This is the empty graph, because it has an empty arc set. The DAG is stored in an object of class bn.

```
survey.dag
```

```
  Random/Generated Bayesian network

  model:
   [A][S][E][O][R][T]
  nodes:                                6
  arcs:                                 0
    undirected arcs:                    0
    directed arcs:                      0
  average markov blanket size:          0.00
  average neighbourhood size:           0.00
  average branching factor:             0.00

  generation algorithm:                 Empty
```

The Guardian

Higher education

University gender gap at record high as 30,000 more women accepted

Ucas says young women a third more likely to go to university than men, and overall admissions are down on last year

Press Association

Sun 27 Aug 2017
19.01 EDT

▲ Students check their A-level results. Photograph: Andrew Matthews/PA

Expert Knowledge, Prior Information: Age

INDEPENDENT

Record numbers of teenagers going to university in England and Scotland, study finds

One in three youngsters in England and one in four in Scotland have been awarded university places this year

Alison Kershaw | Sunday 17 September 2017 19:19 |

Hike comes as overall numbers fall, due to fewer mature and EU students *(Getty)*

Creating the Survey Network

Now we can start adding the arcs that encode the direct dependencies between the variables in the survey.

- Age and Sex are not influenced by any of the other variables, hence we do not need any arcs pointing to either variable.
- Age has a direct influence on Education. The number of people attending universities has increased over the years: so younger people are more likely to have a university degree than older people.

```
survey.dag = set.arc(survey.dag, from = "A", to = "E")
```

- Sex also influences Education; the gender gap in university applications has been widening for many years, with women outnumbering and outperforming men.

```
survey.dag = set.arc(survey.dag, from = "S", to = "E")
```

The Telegraph

Lifestyle › Education & Careers

Britain's highest paying degrees, according to UK graduate salaries

Universities UK

The voice of universities

Graduates more likely to be in employment and earn more than non-graduates – new statistics

24 April 2018

The Guardian

North-south divide

London attracts one-quarter of graduates from UK universities

Centre for Cities report finds 24% of new graduates in 2014 and 2015 were working in capital within six months of finishing

Katie Allen

Sun 20 Nov 2016
19.01 EST

▲ Students at London South Bank University. The capital benefits from a brain drain from the north of England. Photograph: Dan Kitwood/Getty Images

Creating the Survey Network

- Education strongly influences Occupation because higher education levels help in accessing more prestigious professions.

```
survey.dag = set.arc(survey.dag, from = "E", to = "O")
```

- Education influences Residence as well because people often move to attend a particular university or to find a job that matches the skills they acquired in their studies.

```
survey.dag = set.arc(survey.dag, from = "E", to = "R")
```

This leaves only the arcs to the Travel variable.

Expert Knowledge, Prior Information: Travel

theguardian

News › UK news › Census

DATA BLOG

Facts are sacred

Car, bike, train, or walk: how people get to work mapped

The 2011 census reveals the main way people commute to work in 34,753 'output areas' across England and Wales, each of 1,500 people. Find out what happens where you live - which are the top areas for cycling, driving and walking? And why the local concentrations of each?

Travel by local authority

Click heading to sort table. Download this data

Place	Total people	Work at home, %	Tube, metro, light rail, tram, %	Train, %	Bus, coach, %	Taxi, %	Motorcycle etc %	Driving %	Passenger in car or van, %	Cycle, %	On foot, %
ENGLAND	38,881,374	3.5	2.6	3.5	4.9	0.3	0.5	36.9	3.3	1.9	6.9
NORTH EAST	1,924,206	2.2	1.5	0.7	5.6	0.5	0.3	36.9	4.2	1.1	6.4
NORTH WEST	5,184,216	2.8	0.4	1.7	5.2	0.5	0.4	39	3.8	1.4	6.8
YORKSHIRE AND THE HUMBER	3,875,219	2.9	0.3	1.5	5.3	0.4	0.4	38.4	4	1.6	7.4
EAST MIDLANDS	3,336,532	3.3	0.2	0.9	4	0.3	0.5	42.2	3.9	1.8	7.1
WEST MIDLANDS	4,067,119	3	0.2	1.6	4.8	0.3	0.4	40.6	3.8	1.2	6.2
EAST	4,245,544	3.8	0.8	4.8	2.5	0.3	0.5	41.4	3.4	2.4	6.8
LONDON	6,117,482	3.3	14.7	8.7	9.2	0.3	0.8	18.3	1.1	2.6	5.8
SOUTH EAST	6,274,341	4.5	0.2	5	3	0.3	0.6	41.3	3.2	2	7.4
SOUTH WEST	3,856,715	4.6	0.1	1	3.1	0.2	0.7	41.4	3.4	2.3	9

Creating the Survey Network

- Finally, the preferred means of transport are directly influenced by both Occupation and Residence. For Occupation, the reason is that a few jobs require periodic long-distance trips, while others require more frequent trips but on shorter distances.

```
survey.dag = set.arc(survey.dag, from = "O", to = "T")
```

- For Residence, the reason is that both commute time and distance are deciding factors in choosing between travelling by car or by train.

```
survey.dag = set.arc(survey.dag, from = "R", to = "T")
```

On the whole, the model formula (which resembles the corresponding probability notation) of the network is the following:

```
modelstring(survey.dag)
```

```
[1] "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

It is the same as that you would pass to `model2network()`.

The DAGs (and other kind of graphs) are stored in objects of class bn. Printing the object gives some information about the network structure.

```
survey.dag
```

```
   Random/Generated Bayesian network

   model:
    [A][S][E|A:S][O|E][R|E][T|O:R]
   nodes:                                  6
   arcs:                                   6
     undirected arcs:                      0
     directed arcs:                        6
   average markov blanket size:            2.67
   average neighbourhood size:             2.00
   average branching factor:               1.00

   generation algorithm:                   Empty
```

D-Separation and Markov Blankets

The `dsep()` and `mb()` functions can be used to show how d-separation and Markov blankets interact in practice. Firstly, node that a node is never part of its own Markov blanket.

```
mbE = mb(survey.dag, "E")
"E" %in% mbE
```

```
[1] FALSE
```

Secondly, note that the Markov blanket is minimal and that it makes all other nodes independent of the target node.

```
for (node in mbE)
  print(dsep(survey.dag, "E", node, setdiff(mbE, c("E", node))))
```

```
[1] FALSE
[1] FALSE
[1] FALSE
[1] FALSE
```

```
for (node in setdiff(nodes(survey.dag), c("E", mbE)))
  print(dsep(survey.dag, "E", node, mbE))
```

```
[1] TRUE
```

Moral Graphs and CPDAGs

There are functions to compute them:

```
moral(survey.dag)
```

```
cpdag(survey.dag)
```

And if we go back to the survey example, we find that all arcs are compelled and that the CPDAG is identical to the original DAG.

```
all.equal(cpdag(survey.dag), survey.dag)
```

```
[1] TRUE
```

```
compelled.arcs(survey.dag)
```

```
     from to
[1,] "A"  "E"
[2,] "S"  "E"
[3,] "E"  "O"
[4,] "E"  "R"
[5,] "O"  "T"
[6,] "R"  "T"
```

And we can observe that:

```
all.equal(compelled.arcs(survey.dag), directed.arcs(cpdag(survey.dag)))
```

```
[1] TRUE
```

Choosing the Parameters of the Survey Network

Having the structure saved in `survey.dag`, we must now think about the random variables to associate to the nodes. We must choose their sample spaces so that

- we can interpret events easily;
- the chosen events should allow us to express our questions of interest and should allow the model to express the answers to those questions;
- there are few enough events that we can specify their probabilities easily.

```
A.lv = c("young", "adult", "old")
S.lv = c("M", "F")
E.lv = c("high", "uni")
O.lv = c("emp", "self")
R.lv = c("small", "big")
T.lv = c("car", "train", "other")
```

Creating the Parameter Sets

The survey.dag encodes the DAG

```
modelstring(survey.dag)
  [1] "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

This implies the local probability distributions

$$\mathrm{P}(\mathsf{A}, \mathsf{S}, \mathsf{E}, \mathsf{O}, \mathsf{R}, \mathsf{T}) = \mathrm{P}(\mathsf{A})\,\mathrm{P}(\mathsf{S})\,\mathrm{P}(\mathsf{E} \mid \mathsf{A}, \mathsf{S})\,\mathrm{P}(\mathsf{O} \mid \mathsf{E})\,\mathrm{P}(\mathsf{R} \mid \mathsf{E})\,\mathrm{P}(\mathsf{T} \mid \mathsf{O}, \mathsf{R}),$$

and we must choose values for the probabilities for each distribution.

$$\mathrm{P}(\mathsf{A}) = \begin{cases} 0.30 & \text{for young} \\ 0.50 & \text{for adult} \\ 0.20 & \text{for old} \end{cases} \qquad \mathrm{P}(\mathsf{S}) = \begin{cases} 0.60 & \text{for M} \\ 0.40 & \text{for F} \end{cases}$$

```
A.prob = array(c(0.30, 0.50, 0.20), dim = 3, dimnames = list(A = A.lv))
S.prob = array(c(0.60, 0.40), dim = 2, dimnames = list(S = S.lv))
```

Creating the Parameter Sets

$$\mathrm{P}(\mathsf{E} \mid \mathsf{A} = \mathsf{young}, \mathsf{S} = \mathsf{M}) = \begin{cases} 0.75 & \text{for high} \\ 0.25 & \text{for uni} \end{cases}$$

$$\mathrm{P}(\mathsf{E} \mid \mathsf{A} = \mathsf{adult}, \mathsf{S} = \mathsf{M}) = \begin{cases} 0.72 & \text{for high} \\ 0.28 & \text{for uni} \end{cases}$$

$$\mathrm{P}(\mathsf{E} \mid \mathsf{A} = \mathsf{old}, \mathsf{S} = \mathsf{M}) = \begin{cases} 0.88 & \text{for high} \\ 0.12 & \text{for uni} \end{cases}$$

$$\mathrm{P}(\mathsf{E} \mid \mathsf{A} = \mathsf{young}, \mathsf{S} = \mathsf{F}) = \begin{cases} 0.64 & \text{for high} \\ 0.36 & \text{for uni} \end{cases}$$

$$\mathrm{P}(\mathsf{E} \mid \mathsf{A} = \mathsf{adult}, \mathsf{S} = \mathsf{F}) = \begin{cases} 0.70 & \text{for high} \\ 0.30 & \text{for uni} \end{cases}$$

$$\mathrm{P}(\mathsf{E} \mid \mathsf{A} = \mathsf{old}, \mathsf{S} = \mathsf{F}) = \begin{cases} 0.90 & \text{for high} \\ 0.10 & \text{for uni} \end{cases}$$

```
E.prob = array(c(0.75, 0.25, 0.72, 0.28, 0.88, 0.12, 0.64,
           0.36, 0.70, 0.30, 0.90, 0.10), dim = c(2, 3, 2),
           dimnames = list(E = E.lv, A = A.lv, S = S.lv))
```

Creating the Parameter Sets

$$\mathrm{P}(\mathtt{O} \mid \mathtt{E} = \mathtt{high}) = \begin{cases} 0.96 & \text{for } \mathtt{emp} \\ 0.04 & \text{for } \mathtt{self} \end{cases}$$

$$\mathrm{P}(\mathtt{O} \mid \mathtt{E} = \mathtt{uni}) = \begin{cases} 0.92 & \text{for } \mathtt{emp} \\ 0.08 & \text{for } \mathtt{self} \end{cases}$$

```
O.prob = array(c(0.96, 0.04, 0.92, 0.08), dim = c(2, 2),
               dimnames = list(O = O.lv, E = E.lv))
```

$$\mathrm{P}(\mathtt{R} \mid \mathtt{E} = \mathtt{high}) = \begin{cases} 0.25 & \text{for } \mathtt{small} \\ 0.75 & \text{for } \mathtt{big} \end{cases}$$

$$\mathrm{P}(\mathtt{R} \mid \mathtt{E} = \mathtt{uni}) = \begin{cases} 0.20 & \text{for } \mathtt{small} \\ 0.80 & \text{for } \mathtt{big} \end{cases}$$

```
R.prob = array(c(0.25, 0.75, 0.20, 0.80), dim = c(2, 2),
               dimnames = list(R = R.lv, E = E.lv))
```

Creating the Parameter Sets

$$\mathrm{P}(\mathsf{T} \mid \mathsf{O} = \mathsf{emp}, \mathsf{R} = \mathsf{small}) = \begin{cases} 0.48 & \text{for } \mathsf{car} \\ 0.42 & \text{for } \mathsf{train} \\ 0.10 & \text{for } \mathsf{other} \end{cases}$$

$$\mathrm{P}(\mathsf{T} \mid \mathsf{O} = \mathsf{self}, \mathsf{R} = \mathsf{small}) = \begin{cases} 0.56 & \text{for } \mathsf{car} \\ 0.36 & \text{for } \mathsf{train} \\ 0.08 & \text{for } \mathsf{other} \end{cases}$$

$$\mathrm{P}(\mathsf{T} \mid \mathsf{O} = \mathsf{emp}, \mathsf{R} = \mathsf{big}) = \begin{cases} 0.58 & \text{for } \mathsf{car} \\ 0.24 & \text{for } \mathsf{train} \\ 0.18 & \text{for } \mathsf{other} \end{cases}$$

$$\mathrm{P}(\mathsf{T} \mid \mathsf{O} = \mathsf{self}, \mathsf{R} = \mathsf{big}) = \begin{cases} 0.70 & \text{for } \mathsf{car} \\ 0.21 & \text{for } \mathsf{train} \\ 0.09 & \text{for } \mathsf{other} \end{cases}$$

```
T.prob = array(c(0.48, 0.42, 0.10, 0.56, 0.36, 0.08, 0.58,
           0.24, 0.18, 0.70, 0.21, 0.09), dim = c(3, 2, 2),
           dimnames = list(T = T.lv, O = O.lv, R = R.lv))
```

Creating the Parameter Sets

Having encoded all the probabilities, we can save them in a list and create a `bn.fit` object that contains both the DAG and the probabilities.

```
survey.cpt = list(A = A.prob, S = S.prob, E = E.prob, O = O.prob,
                  R = R.prob, T = T.prob)
survey.bn = custom.fit(survey.dag, survey.cpt)
```

The information about each variable:

```
survey.bn$A
```

```
  Parameters of node A (multinomial distribution)

Conditional probability table:
 A
young adult   old
  0.3   0.5   0.2
```

Creating the Parameter Sets

```
survey.bn$T
```

```
  Parameters of node T (multinomial distribution)

Conditional probability table:

, , R = small

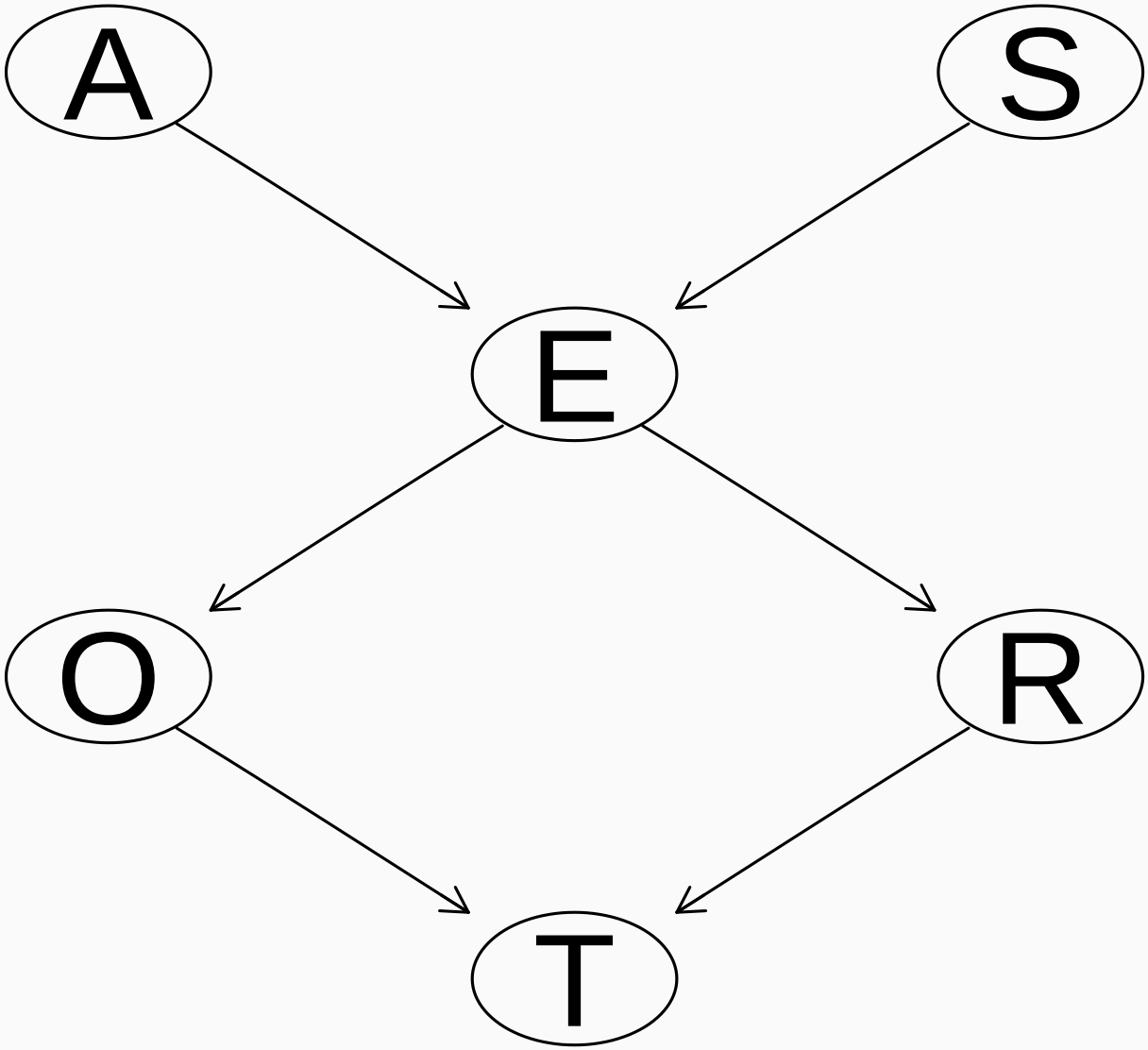
       O
T        emp self
  car   0.48 0.56
  train 0.42 0.36
  other 0.10 0.08

, , R = big

       O
T        emp self
  car   0.58 0.70
  train 0.24 0.21
  other 0.18 0.09
```

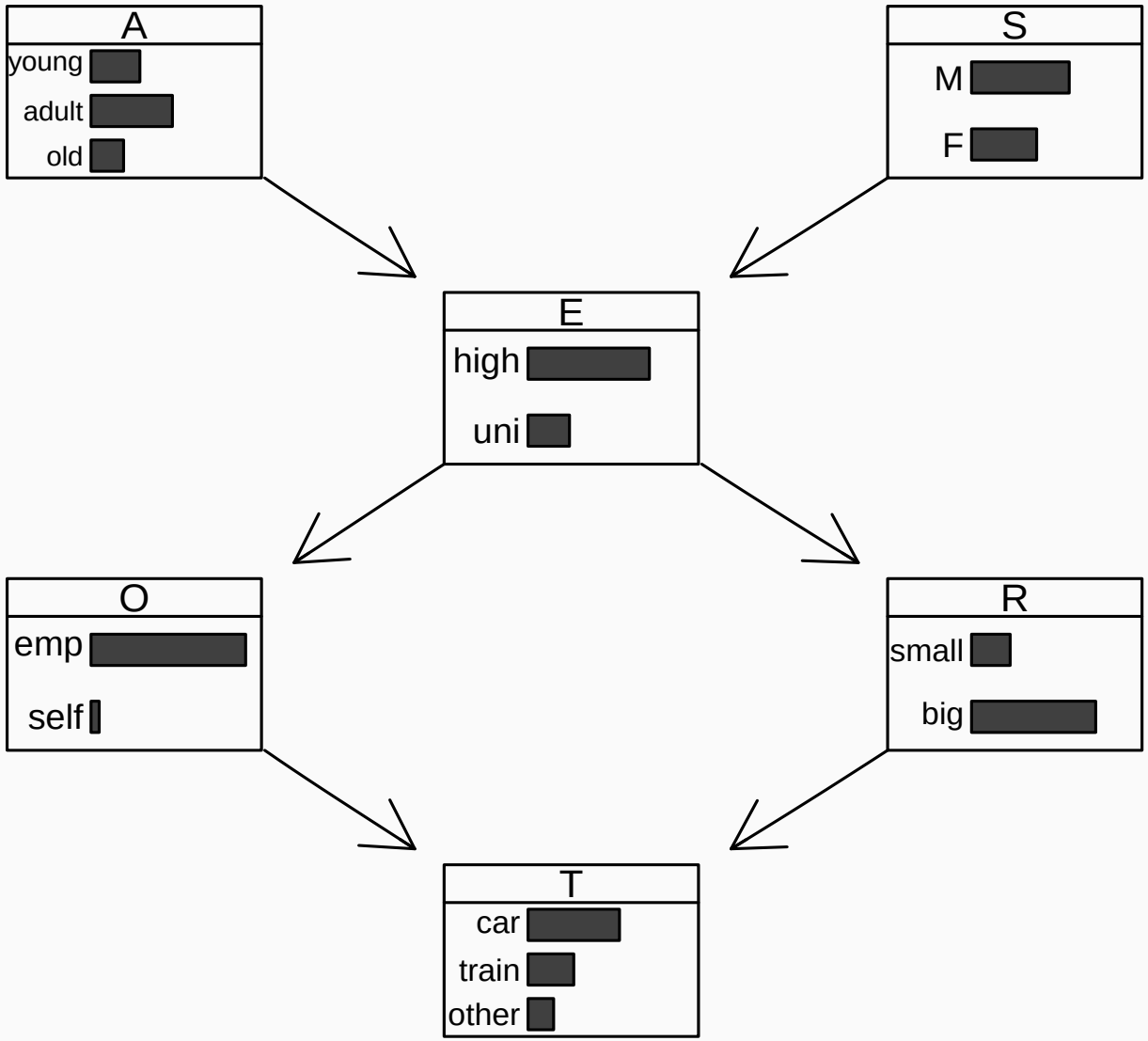
Plotting the Survey Bayesian Network

```
graphviz.plot(survey.bn, shape = "ellipse")
```



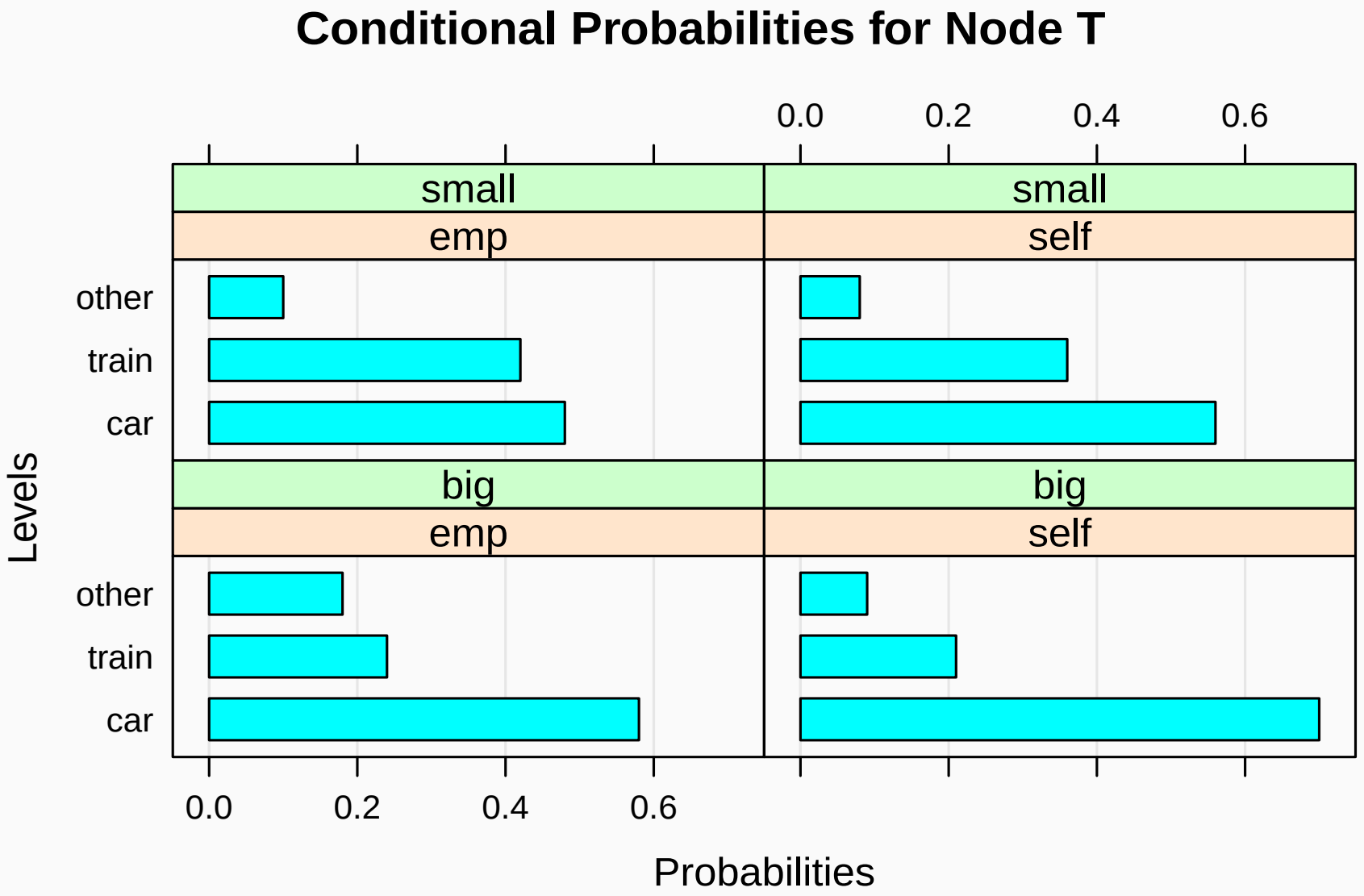
Plotting the Survey Bayesian Network

```
graphviz.chart(survey.bn)
```



Plotting the Survey Bayesian Network

```
bn.fit.barchart(survey.bn$T)
```



Case Study: Business Creation

Say we would like to create a model for the following theory from behavioural economics on business creation based on the attitudes perspective:

> *The intention to create a new business would depend on attitudinal evaluation, if someone considers that creating a new business is a positive thing, he or she will be more prone to carry out the target behaviour. Additionally, intentions also depend on normative beliefs. That is to say, intentions depend on the perceived social pressure related with a particular behaviour.*

Since no expert will be able to give us values for the probabilities in the BN or suggestions about what its structure looks like, we have collected some information using electronic questionnaires from 1542 university professors.

The Questionnaire

The questionnaire contained six sections:

1. demographic data;
2. questions directly related with entrepreneurship phenomena;
3. environment attitudes;
4. obstacles and facilitators;
5. an attitudinal scale;
6. comments and details.

To measure different aspect related with the entrepreneurial attitude we used scales about perceived obstacles, perceived facilitators, self-efficacy, locus of control, attitude towards business creation and normative beliefs. Scores in all scales were individually recorded using three levels of response.

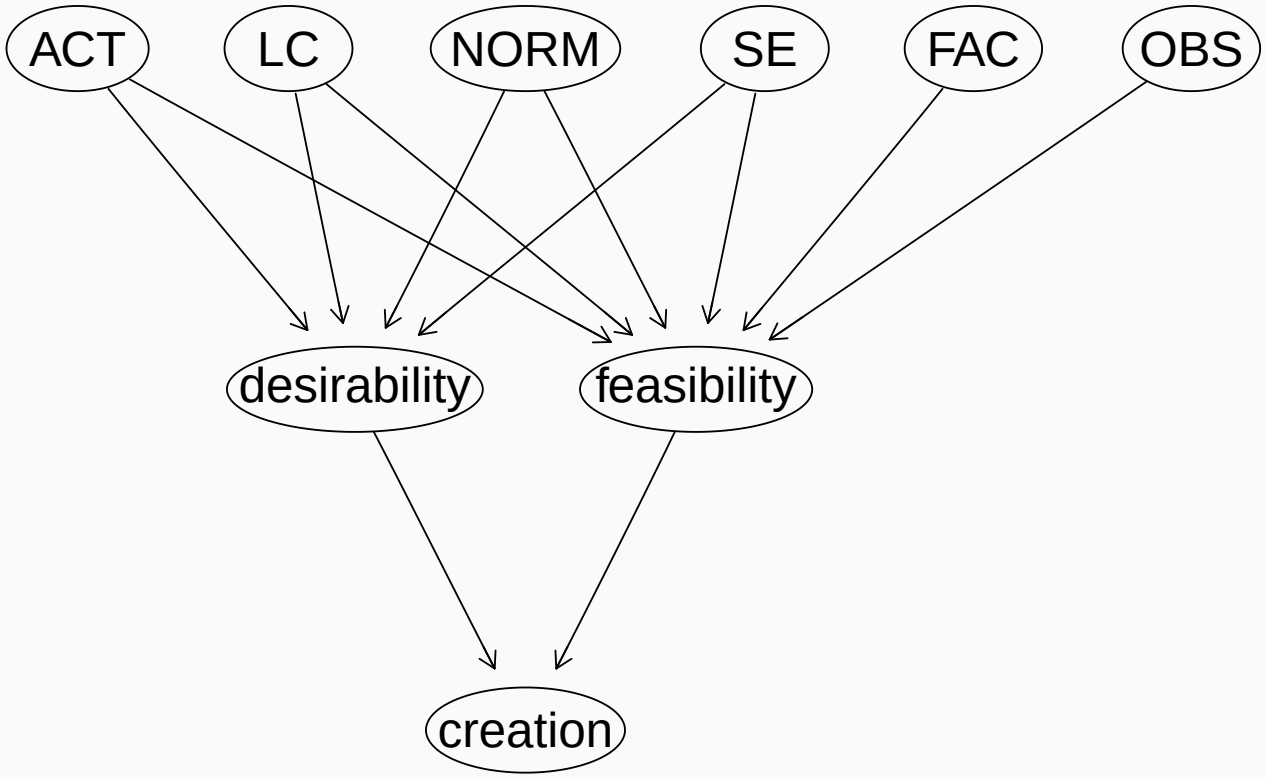
The Derived Scales

- perceived obstacles (OBS): “Having to work too many hours”, “Lack of experience”, “Ignorance of activity sector”, etc.
- perceived facilitators (FAC): “Have perceived a need in the market”, “The detection of a business opportunity” or “The availability of personal assets to invest”, etc.
- self-efficacy (SE), the perceived difficulty to actually carry out a specific behaviour: “Working under continuous stress, pressure and conflict”, “To form alliances or partnerships with other companies”, etc.
- locus of control (LC): “If you want, you can easily be an entrepreneur and starting your own business”, etc.
- attitude towards business creation (ACT): “To what extent do you believe that these elements are related with the creation of a new company?”, “To what extent do you like assume it?”, etc.
- normative beliefs (NORM): “Please, think in your family, closest friends and social environment and indicate the degree to which they are favourable to the idea that you create a company”, etc.

A First Try at BN Modelling

From the description of the behavioural theory model, we can venture to define a DAG for our BN model like the following:

```
progn = model2network(
  paste0("[creation|desirability:feasibility][desirability|LC:SE:ACT:NORM]",
         "[feasibility|LC:SE:ACT:NORM:FAC:OBS][LC][FAC][OBS][SE][ACT][NORM]"))
graphviz.plot(progn, shape = "ellipse")
```



Running Out of Samples

The problems start when we compare the number of probabilities in the BN with the number of questionnaires we collected. Each variable takes two to four values:

```
summary(inted)
```

```
   creation   desirability                feasibility
   Yes: 480   Yes:882      Very.little.feasible:378
   No :1062   No :660      A.little.feasible   :672
                           Feasible            :444
                           A.lot.feasible      : 48
        LC           FAC           OBS            SE
   High  :373   Low   :561   Low   :312   Medium:412
   Low   :544   High  :259   Medium:793   Low   :774
   Medium:625   Medium:722   High  :437   High  :356

       ACT           NORM
   Medium:724   High  :318
   Low   :226   Medium:452
   High  :592   Low   :772
```

Running Out of Samples

A cursory examination suggests that we do not have enough questionnaires (1542) compared to the number of probabilities in the BN (2288):

```
nparams(progn, inted)
  [1] 2288
nrow(inted)
  [1] 1542
```

This means that in practice we cannot possibly observe all the events we need to define the probability of. Some will not be observed even once! And then if we estimate their probability with their frequency in the sample of questionnaires then

$$\mathrm{P}(\text{some event we do not observe}) = \frac{\text{zero questionnaires}}{\text{1542 questionnaires}} = 0$$

which we are pretty sure it is not true.

How Do We Compute the Number of Probabilities?

How does `nparams()` compute the number of probabilities?

For each node:

1. find out its parents;

```
par = parents(progn, "desirability")
par
```

```
[1] "ACT"  "LC"   "NORM" "SE"
```

2. find out how many values each parent takes in the data;

```
par.values = rep(0, length(par))
for (p in seq_along(par))
  par.values[p] = nlevels(inted[, par[p]])
```

3. count how many combinations;

```
par.combn = prod(par.values)
```

4. look up how many values the node takes;

```
node.values = nlevels(inted[, "desirability"])
```

5. multiply these two numbers.

```
(node.values - 1) * par.combn
```

```
[1] 81
```

How Do We Compute the Number of Probabilities?

Consider $\mathrm{P}(\texttt{creation} \mid \texttt{feasibility}, \texttt{desirability})$.

$\texttt{desirability} = \texttt{yes}$

	Very.little	A.little	Feasible	A.lot
yes	?	?	?	?
no	?	?	?	?

$\texttt{desirability} = \texttt{no}$

	Very.little	A.little	Feasible	A.lot
yes	?	?	?	?
no	?	?	?	?

The table has $2 \times 4 \times 2$ cells, but since each column is a conditional distribution that sums up 1 one probability in each column is fixed to $1-$ the sum of the other probabilities in the column.

How Do We Compute the Number of Probabilities?

ACT	$3-1$		$=2$
LC	$3-1$		$=2$
NORM	$3-1$		$=2$
SE	$3-1$		$=2$
FAC	$3-1$		$=2$
OBS	$3-1$		$=2$
desirability	$2-1$	$\times 3^4$	$=81$
feasibility	$4-1$	$\times 3^6$	$=2187$
creation	$2-1$	$\times 2 \times 4$	$=8$
total			$=2288$

The problem is clearly in `feasibility`, but even if we reduced it to just 2 values we would still have too 729 probabilities to specify. That too many!

Diagnostic versus Prognostic Models

In `feasibility` $\approx 40\%$ of the probabilities are missing values and another $\approx 40\%$ is 0-1 distributions, which clearly is not ideal.

```
fitted.progn = bn.fit(progn, inted)
ldist = coef(fitted.progn$feasibility)
length(which(is.na(ldist))) / length(ldist)
```

```
[1] 0.396
```

```
length(which(ldist %in% c(0, 1))) / length(ldist)
```

```
[1] 0.397
```

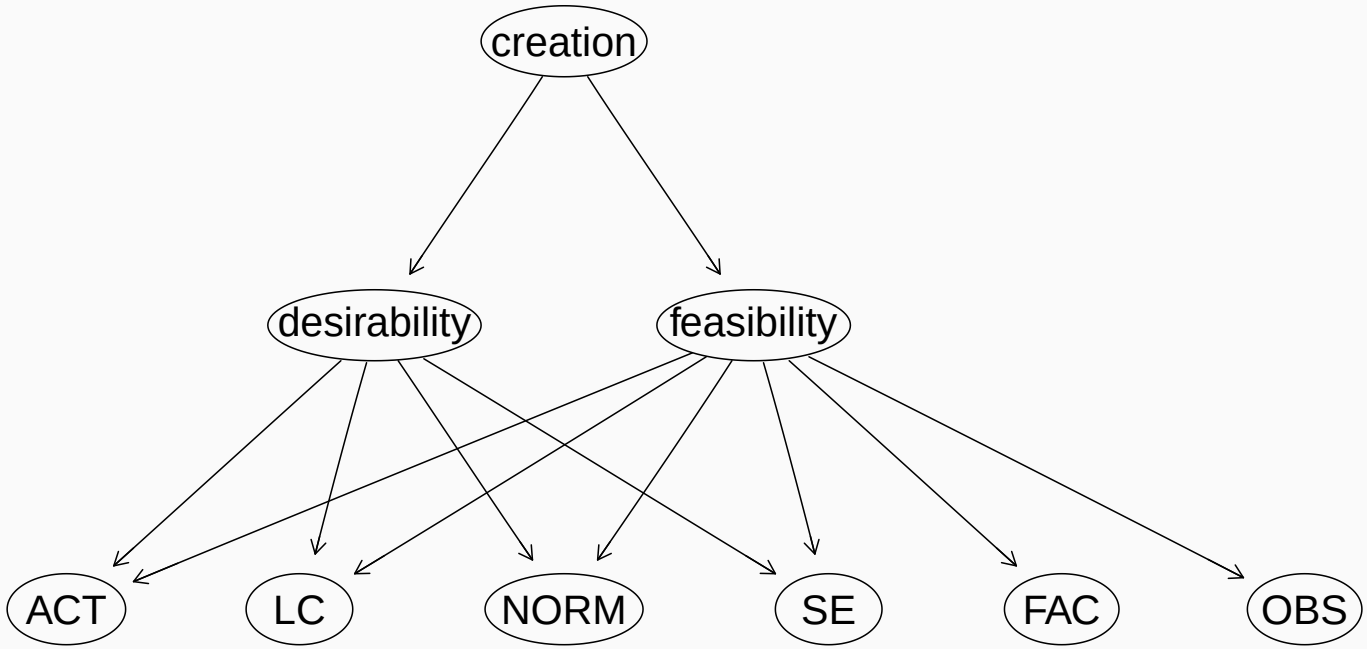
The only recourse is changing the DAG. `progn` was built as a prognostic model in which the causes are on the top and the effects are at the bottom of the DAG. We could also try the converse: a diagnostic models in which the effects are on the top and the causes are on the bottom.

A Diagnostic Model

```
diagn = model2network(
  paste("[creation][desirability|creation][feasibility|creation]",
        "[LC|desirability:feasibility][FAC|feasibility][OBS|feasibility]",
        "[SE|desirability:feasibility][ACT|desirability:feasibility]",
        "[NORM|desirability:feasibility]", sep = ""))
nparams(diagn, inted)
```

```
[1] 89
```

```
graphviz.plot(diagn, shape = "ellipse")
```



Developing the Model

The diagnostic BN has far fewer parameters, and we can estimate them with reasonable accuracy from the data.

```
fitted.diagn = bn.fit(diagn, inted)
```

The `bn.fit()` function creates a `bn.fit` object as `custom.fit` did earlier; but it learns the probabilities from the data instead of requiring them as expert knowledge.

```
fitted.diagn$desirability
```

```
  Parameters of node desirability (multinomial distribution)

Conditional probability table:

            creation
desirability   Yes    No
         Yes 0.727 0.502
         No  0.273 0.498
```

Job Creation, Goodness of Fit

We now have two models on the table. How do we decide which one is better? Recall that in machine learning we measure performance with the ability of predicting new events.

We can measure predictive accuracy by:

1. splitting the 1542 questionnaires into a training set (say, of 1156) and a validation set (say, of 386);
2. estimating the probabilities in the prognostic BN from the training set;
3. estimating the probabilities in the diagnostic BN from the training set;
4. computing the (log-)probability of the test set using the prognostic BN;
5. computing the (log-)probability of the test set using the diagnostic BN;
6. comparing the two and pick the BN with the highest one.

Job Creation, Goodness of Fit

1. ```
   training.samples = sample(nrow(inted), 1156)
   training.set = inted[training.samples, ]
   validation.set = inted[-training.samples, ]
   ```

2. ```
   progn.fit = bn.fit(progn, training.set)
   ```

3. ```
   diagn.fit = bn.fit(diagn, training.set)
   ```

4. ```
   logLik(progn.fit, validation.set)
   ```
   ```
   [1] -3170
   ```

5. ```
   logLik(diagn.fit, validation.set)
   ```
   ```
   [1] -3269
   ```

6. `progn.fit` wins!

But does that make sense? `progn.fit` was supposed to be the worse of the two BNs...
   ```

Job Creation, Goodness of Fit

An alternative measure of predictive accuracy is the probability of predicting `creation` correctly: that is key in proving our theory is valid.

```
predicted = predict(progn.fit, node = "creation", data = validation.set,
              method = "bayes-lw")
prop.table(table(predicted, validation.set[, "creation"]))
```

```
  predicted   Yes    No
        Yes 0.138 0.142
        No  0.154 0.565
```

```
predicted = predict(diagn.fit, node = "creation", data = validation.set,
              method = "bayes-lw")
prop.table(table(predicted, validation.set[, "creation"]))
```

```
  predicted   Yes    No
        Yes 0.145 0.130
        No  0.153 0.573
```

Predictive accuracy is similar for both models, so we can choose `diagn` which is much simpler.

A Better Way: Cross-Validation

You may ask: what if I randomly pick a "bad" training sample? After all, if we split the data just once we cannot be sure if we are getting a high/low predictive accuracy just by chance.

The answer to this question is cross-validation:

1. Split the data into k folds (usually 10) of equal size.
2. For each fold in turn:
   - 2.1 merge the other $k-1$ folds into a training set;
   - 2.2 take the chosen fold to be the validation set;
   - 2.3 learn the BN from the training set;
   - 2.4 estimate predictive accuracy on the validation set.
3. Take the average of all the k estimates of predictive accuracy.

PROS: more reliable measures of predictive accuracy.
CONS: it takes more time to run.

Cross-Validation and Predictive Accuracy

The function that implements cross-validation is `bn.cv()`, which requires a data set, a network, and a definition of what measure of predictive accuracy to use.

```
bn.cv(inted, progn, loss = "pred-lw", loss.args = list(target = "creation"))
```

```
  k-fold cross-validation for Bayesian networks

  target network structure:
   [ACT][FAC][LC][NORM][OBS][SE]
   [desirability|ACT:LC:NORM:SE]
   [feasibility|ACT:FAC:LC:NORM:OBS:SE]
   [creation|desirability:feasibility]
  number of folds:                         10
  loss function:
                      Classification Error (Posterior, disc.)
  training node:                           creation
  expected loss:                           0.265
```

Cross-Validation and Predictive Accuracy

```
bn.cv(inted, diagn, loss = "pred-lw", loss.args = list(target = "creation"))
```

```
  k-fold cross-validation for Bayesian networks

  target network structure:
   [creation][desirability|creation][feasibility|creation]
   [ACT|desirability:feasibility][FAC|feasibility]
   [LC|desirability:feasibility]
   [NORM|desirability:feasibility][OBS|feasibility]
   [SE|desirability:feasibility]
  number of folds:                         10
  loss function:
                      Classification Error (Posterior, disc.)
  training node:                           creation
  expected loss:                           0.274
```

The prediction error (that is, the proportion of questionnaires for which the prediction is wrong) is virtually identical for `progn` and `diagn`. Hence we still choose to user `diagn` for its simplicity.

An Example: Domotics

Suppose we have a house in which we want to install automatic windows that open and close depending on outside conditions:

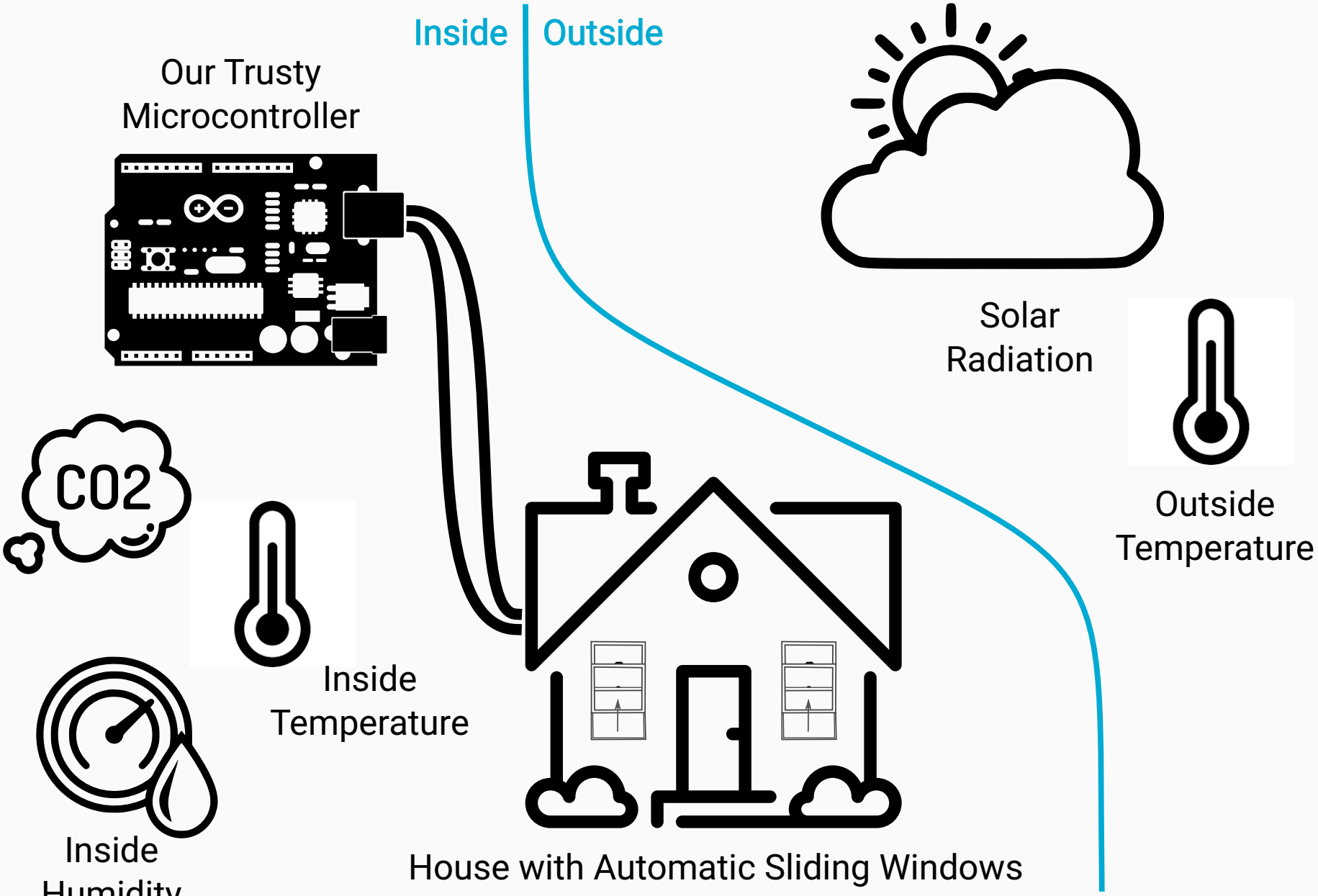
- temperature (`Tout`);
- sunlight (`Sun`);

to optimise inside conditions:

- CO_2 (`CO2`);
- temperature (`Tin`);
- humidity (`Hin`).

Our goal is to program the microcontroller using a BN that opens and closes the windows (`W`) to maintain optimal environment inside the house.

An Example: Domotics

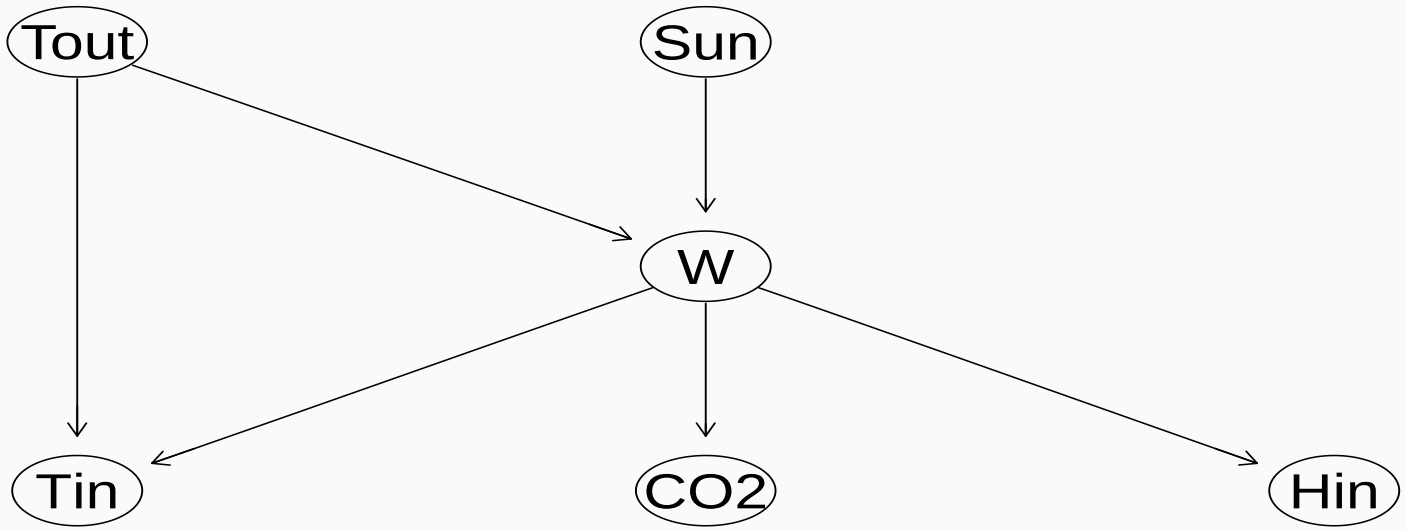


Defining the DAG

In keeping with the idea that we can use causal reasoning to define the DAG of a BN, we observe that:

- the outside temperature contributes to the inside temperature;
- the solar radiation and the outside temperature determine whether it is a good idea to open the windows;
- whether windows are open influences all the inside conditions.

```
win.dag = model2network("[Tout][Sun][W|Tout:Sun][Tin|Tout:W][Hin|W][CO2|W]")
graphviz.plot(win.dag, shape = "ellipse")
```



Defining the Sample Spaces

We should then decide which values each variable can take, with an eye towards keeping the BN simple (since we have to enter all the probabilities next!).

- `Tout` and `Tin`: `<18`, `18-24`, `>24`.
- `Sun`: `low` or `high`.
- `W`: `open` or `closed`.
- `CO2`: `low`, `medium`, `high`.
- `Hin`: `low`, `high`.

The DAG givens us the decomposition

$$\begin{aligned}\mathrm{P}(\texttt{Tout}, \texttt{Sun}, \texttt{W}, \texttt{Tin}, \texttt{CO2}, \texttt{Hin}) = \mathrm{P}(\texttt{Tout})\,\mathrm{P}(\texttt{Sun}) \times \\ \times\, \mathrm{P}(\texttt{W} \mid \texttt{Tout}, \texttt{Sun})\,\mathrm{P}(\texttt{Tin} \mid \texttt{Tout}, \texttt{W})\,\mathrm{P}(\texttt{CO2} \mid \texttt{W})\,\mathrm{P}(\texttt{Hin} \mid \texttt{W})\end{aligned}$$

which implies $2 + 1 + 6 + 12 + 4 + 2 = 37$ probabilities.

Specifying the Probabilities

The probabilities of the outside weather conditions can be read from any number of weather websites (or even Wikipedia!).

```
T.lv = c("<18", "18-24", ">24")
Sun.lv = c("low", "high")

Tout.prob = array(c(0.20, 0.70, 0.10), dim = 3, dimnames = list(Tout = T.lv))
Tout.prob
```

```
Tout
  <18 18-24   >24
  0.2   0.7   0.1
```

```
Sun.prob = array(c(0.70, 0.30), dim = 2, dimnames = list(Sun = Sun.lv))
Sun.prob
```

```
Sun
 low high
 0.7  0.3
```

(These would be sensible values for early summer in the UK, the effects of global warming notwithstanding.)

Specifying the Probabilities

If it is very cold or very hot outside, windows should be closed; and they should be closed if the sunlight is low and it's cold or if the sunlight is strong and the temperature is high.

```
W.lv = c("open", "closed")

W.prob = array(c(0.10, 0.90, 0.70, 0.30, 0.90, 0.10,
                 0.20, 0.80, 0.50, 0.50, 0.10, 0.90), dim = c(2, 3, 2),
           dimnames = list(W = W.lv, Tout = T.lv, Sun = Sun.lv))
W.prob
```

```
, , Sun = low

        Tout
W        <18 18-24 >24
  open   0.1   0.7 0.9
  closed 0.9   0.3 0.1

, , Sun = high

        Tout
W        <18 18-24 >24
  open   0.2   0.5 0.1
  closed 0.8   0.5 0.9
```

Specifying the Probabilities

The inside temperature should be close to the outside temperature if windows are open, or be hotter. If windows are closed then the inside might be hotter than the outside temperature.

```
Tin.prob = array(c(0.25, 0.70, 0.05, 0.20, 0.60, 0.20, 0, 0.20, 0.80,
                   0.08, 0.90, 0.02, 0.10, 0.70, 0.20, 0, 0.50, 0.50),
             dim = c(3, 3, 2),
             dimnames = list(Tin = T.lv, Tout = T.lv, W = W.lv))
Tin.prob
```

```
, , W = open

        Tout
Tin      <18 18-24 >24
  <18   0.25   0.2 0.0
  18-24 0.70   0.6 0.2
  >24   0.05   0.2 0.8

, , W = closed

        Tout
Tin      <18 18-24 >24
  <18   0.08   0.1 0.0
  18-24 0.90   0.7 0.5
  >24   0.02   0.2 0.5
```

Specifying the Probabilities

Finally both inside humidity and CO_2 improve if the windows are open and they let in some fresh air.

```
Hin.lv = Sun.lv = c("low", "high")
CO2.lv = c("low", "medium", "high")

CO2.prob = array(c(0.05, 0.30, 0.65, 0.20, 0.40, 0.40), dim = c(3, 2),
              dimnames = list(CO2 = CO2.lv, W = W.lv))
CO2.prob
```

```
          W
  CO2       open closed
    low     0.05    0.2
    medium 0.30    0.4
    high   0.65    0.4
```

```
Hin.prob = array(c(0.30, 0.70, 0.50, 0.50), dim = c(2, 2),
              dimnames = list(Hin = Hin.lv, W = W.lv))
Hin.prob
```

```
        W
  Hin     open closed
    low   0.3    0.5
    high  0.7    0.5
```

Building the BN from the DAG and the Probabilities

Finally, we put the DAG and the probabilities together to build the BN in a `bn.fit` object.

```
win.bn = custom.fit(win.dag, list(Tout = Tout.prob, Sun = Sun.prob,
           W = W.prob, Tin = Tin.prob, CO2 = CO2.prob, Hin = Hin.prob))
```

We defined this BN for the most part using common sense. We should now check that:

1. it describes what it is modelling in a realistic way;
2. it is not overly sensitive to small changes in the probabilities we specified.

These are fundamental sanity checks that ensure the BN is a well-behaved machine learning model.

Is it Realistic?

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == "18-24") & (Tin == ">24"))
prop.table(table(in.out))
```

```
in.out
  open closed
 0.625  0.375
```

If the temperature outside is between 18° and 24°, and the temperature inside is above 24°, we want the window to be open to lower the inside temperature.

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == "<18") & (Tin == ">24"))
prop.table(table(in.out))
```

```
in.out
  open closed
 0.321  0.679
```

However, if it is very cold outside we want the window to be open but less so because opening it would create a very uncomfortable draft!

Is it Realistic?

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == "18-24") & (Tin == "18-24"))
prop.table(table(in.out))
```

```
in.out
  open closed
 0.588  0.412
```

If both the inside and outside temperatures are comfortable, we want the window to be open more than we do not, just to keep the air fresh; but can keep it closed nearly 40% of the time.

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == ">24") & (Tin == "18-24"))
prop.table(table(in.out))
```

```
in.out
  open closed
 0.395  0.605
```

If the outside temperature is both hot and hotter than the inside temperature, keeping the windows closed helps keeping the heat out.

Is it Realistic?

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == "18-24") & (Tin == "<18"))
prop.table(table(in.out))
```

```
in.out
  open closed
 0.775  0.225
```

Finally, if the temperature outside is pleasant but it is cold inside, we may want to open the windows to make it more comfortable.

```
in.co2 = cpdist(win.bn, nodes = "W",
           evidence = (Hin == "high") & (CO2 == "high"))
prop.table(table(in.co2))
```

```
in.co2
  open closed
 0.728  0.272
```

If we have high humidity or CO_2 inside the house, we may want to open the windows and refresh the air.

Is it Realistic?

```
in.co2 = cpdist(win.bn, nodes = "W", evidence = (Hin == "high") & (CO2 == "low"))
prop.table(table(in.co2))
```

```
in.co2
  open closed
 0.266  0.734
```

```
in.co2 = cpdist(win.bn, nodes = "W", evidence = (Hin == "low") & (CO2 == "high"))
prop.table(table(in.co2))
```

```
in.co2
  open closed
 0.551  0.449
```

If CO_2 is high, opening the window may help; less so if the only problem is humidity.

```
in.co2 = cpdist(win.bn, nodes = "W", evidence = (Hin == "low") & (CO2 == "low"))
prop.table(table(in.co2))
```

```
in.co2
  open closed
  0.15   0.85
```

If both humidity and CO_2 are low, we are fine keeping the windows closed.

Is it Realistic?

```
all = cpdist(win.bn, nodes = "W",
        evidence = (Hin == "low") & (CO2 == "low") &
                                (Tin == "18-24") & (Tout == ">24"), n = 10^6)
prop.table(table(all))
```

```
  all
    open closed
  0.0939 0.9061
```

If, on top of having low humidity and CO_2, the outside temperature is very hot we definitely want to keep the windows closed!

```
all = cpdist(win.bn, nodes = "W",
        evidence = (Hin == "high") & (CO2 == "high") &
                                (Tin == "18-24") & (Tout == ">24"), n = 10^6)
prop.table(table(all))
```

```
  all
    open closed
   0.639  0.361
```

Whereas if they are both high we may want to open the window even if it is hot outside.

Is it Realistic?

We could keep going like this for a while more, querying the model and checking that we get sensible answers. The general idea is:

1. Not all possible events have a corresponding probability in the probability tables associated with the various nodes; but
2. we should make sure that (many of) those events are assigned realistic probabilities.
3. We can query the BN and check that it makes the right choices in different scenarios, but giving high probabilities to what we consider the right decisions to make.
4. If that is not the case, we can go back to the drawing board and rethink the structure and the probabilities we created the BN from.

Sensitivity Analysis

The general idea behind sensitivity analysis is that:

1. There is always some uncertainty in the information that we use to create a machine learning model because:
   - if we ask different experts, we may get different probabilities and different DAGs for the same model;
   - if we learn the model from data, collecting data again and re-learning the model will give us a somewhat different model because the data will be somewhat different.
2. Somewhat different models introduce uncertainty in the decisions and the conditional probabilities we get in response to queries, because models will not be numerically identical.
3. Ideally, small changes in the model should produce small changes in the decisions the model makes. In real-world applications things tend to change smoothly, so a small change in one variable should only produce a small change in a few other variables.

Sensitivity Analysis

Say that we make a small change to the relationship between inside and outside temperatures because we got it wrong the first time.

Tin.prob

```
, , W = open

        Tout
Tin       <18 18-24 >24
  <18    0.25   0.2 0.0
  18-24 0.70   0.6 0.2
  >24    0.05   0.2 0.8

, , W = closed

        Tout
Tin       <18 18-24 >24
  <18    0.08   0.1 0.0
  18-24 0.90   0.7 0.5
  >24    0.02   0.2 0.5
```

→

Tin.prob

```
, , W = open

        Tout
Tin       <18 18-24  >24
  <18    0.25   0.2 0.05
  18-24 0.70   0.6 0.25
  >24    0.05   0.2 0.70

, , W = closed

        Tout
Tin       <18 18-24 >24
  <18    0.08   0.1 0.0
  18-24 0.90   0.7 0.5
  >24    0.02   0.2 0.5
```

Sensitivity Analysis

Having made a small change to Tin, we create the updated BN

```
win.bn2 = custom.fit(win.dag, list(Tout = Tout.prob, Sun = Sun.prob,
           W = W.prob, Tin = Tin.prob, CO2 = CO2.prob, Hin = Hin.prob))
```

re-run all the queries and compare the results.

before	after	difference	decision
0.669, 0.331	0.668, 0.332	0.001	same
0.25, 0.75	0.286, 0.714	0.036	same
0.604, 0.396	0.61, 0.39	0.005	same
0.421, 0.579	0.546, 0.454	0.125	not
0.772, 0.228	0.79, 0.21	0.018	same
0.729, 0.271	0.711, 0.289	0.018	same
0.27, 0.73	0.325, 0.675	0.055	same
0.528, 0.472	0.52, 0.48	0.009	same
0.147, 0.853	0.158, 0.842	0.012	same
0.089, 0.911	0.127, 0.873	0.038	same
0.637, 0.363	0.689, 0.311	0.052	same

We changed the probabilities `Tin` by moving a 10% probability mass in a single conditional distribution; it is fairly large change. Even so, the largest change in the probabilities returned by the queries is 12.5%, about the same amount.

The decision of whether to open the windows is the same before and after we change `Tin`, with the only exception of

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == ">24") & (Tin == "18-24"))
prop.table(table(in.out))
```

which is understandable since the evidence is exactly the conditional distribution we changed.

What does it mean? It means that the BN is robust to perturbations up 10% in `Tin`/`Tout`.

How Sensitive is Sensitive?

```
in.out = cpdist(win.bn, nodes = "W",
           evidence = (Tout == ">24") & (Tin == "18-24"), n = 10^6)
orig = prop.table(table(in.out))

diff = seq(from = 0.01, to = 0.20, by = 0.01)
query.prob = rep(0, length(diff))

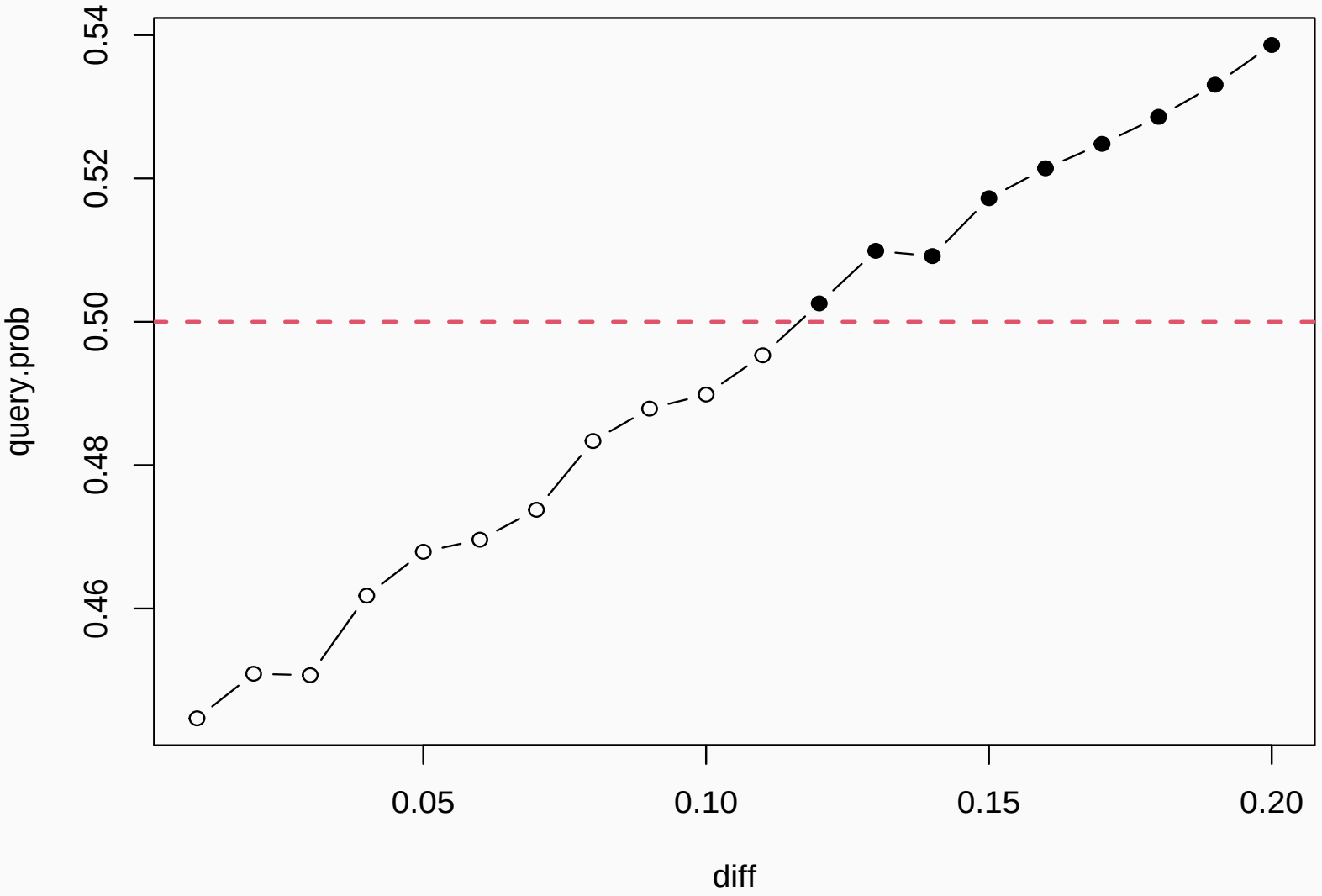
for (i in seq_along(diff)) {

  new.bn = win.bn
  cpt = coef(new.bn$Tin)
  cpt[, ">24" , "open"] = cpt[, ">24" , "open"] +
                          c(diff[i]/2, diff[i]/2, -diff[i])
  new.bn$Tin = cpt

  in.out = cpdist(new.bn, nodes = "W",
             evidence = (Tout == ">24") & (Tin == "18-24"), num = 10^6)
  query.prob[i] = prop.table(table(in.out))[1]

}#FOR
```

How Sensitive is Sensitive?



It's a matter of degree...

Revisiting Domotics: a Dynamic Model

The BN we run on our microcontroller interact continuously with the surrounding environment: as it registers some inside temperature, humidity and CO_2, it may decide to open or close the window. After some time all of these adjust and move closer to the outside weather conditions; at that point the BN may decide to close or open the windows depending on how the situation is at that time.

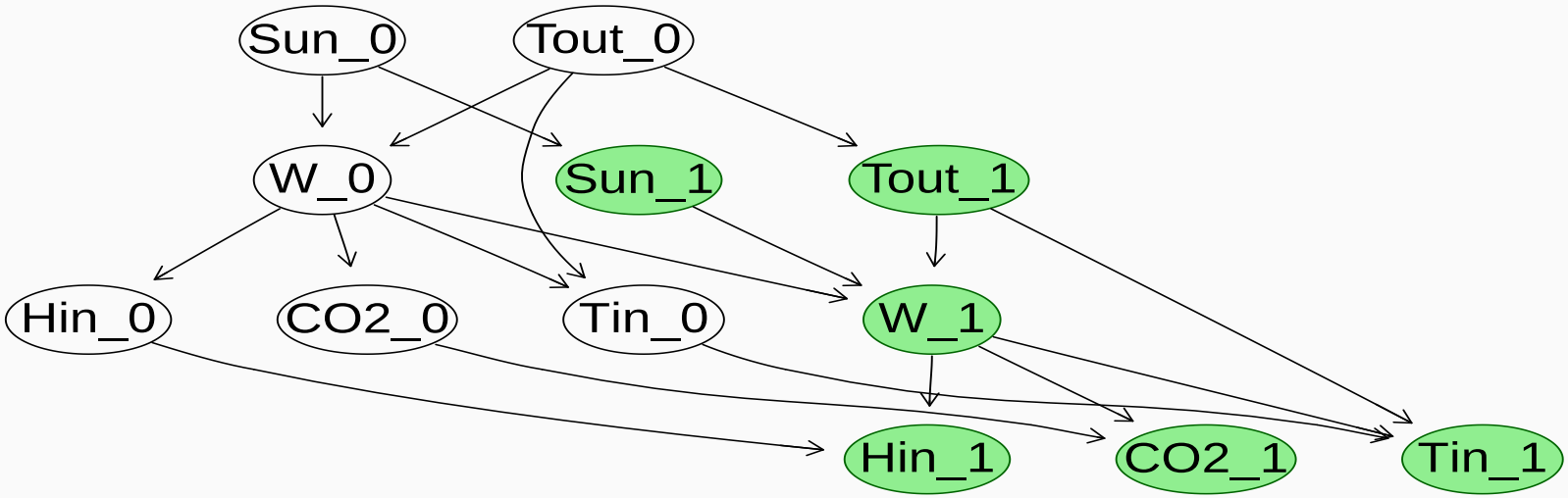
Framing the model in this way suggests that we want a dynamic BN that explicitly models the passage of time and how the variables interact at different time points.

A BN that incorporates time is called a dynamic BN (as opposed to a static BN that does not).

Revisiting Domotics: a Dynamic Model

A naive approach to construct such a BN would be to duplicate all nodes and link them with arcs across time.

```
win.dyn = model2network(
  paste0("[Tout_0][Sun_0][W_0|Tout_0:Sun_0][Tin_0|Tout_0:W_0][Hin_0|W_0]",
    "[CO2_0|W_0][Tout_1|Tout_0][Sun_1|Sun_0][W_1|W_0:Tout_1:Sun_1]",
    "[Tin_1|Tin_0:Tout_1:W_1][Hin_1|Hin_0:W_1][CO2_1|CO2_0:W_1]"))
nodes.t0 = grep("_0$", nodes(win.dyn), value = TRUE)
nodes.t1 = grep("_1$", nodes(win.dyn), value = TRUE)
graphviz.plot(win.dyn, shape = "ellipse", groups = list(nodes.t0, nodes.t1),
  highlight = list(nodes = nodes.t1, fill = "palegreen2", col = "darkgreen"))
```



Do We Really Need All These Arcs?

We probably do not need `W_0` $\rightarrow$ `W_1`; implying that we do not care if the windows have been open or closed since the previous time point when deciding to open or close them in the current time point.

```
win.dyn = drop.arc(win.dyn, from = "W_0", to = "W_1")
```

The other arcs all make sense in various ways. But they also make the BN much more complex. All the new `*_1` variables require new tables of probabilities, which contain 2-3 times as many probabilities:

$$37 + \underbrace{2 \times 1}_{\texttt{Sun\_1}} + \underbrace{3 \times 2}_{\texttt{Tout\_1}} + \underbrace{3 \times 2 \times 2 \times 1}_{\texttt{W\_1}} + \underbrace{2 \times 2 \times 1}_{\texttt{Hin\_1}} + \underbrace{2 \times 3 \times 2}_{\texttt{CO2\_1}} + \underbrace{3 \times 2 \times 3 \times 2}_{\texttt{Tin\_1}} = 110.$$

Do We Really Need All These Arcs?

We would like to use this dynamic BN to open and close the windows.

Why do we want to do that? Because in the first time point the air inside the house is hot/humid/stuffy, and if we open the windows the air will get better by the second time point.

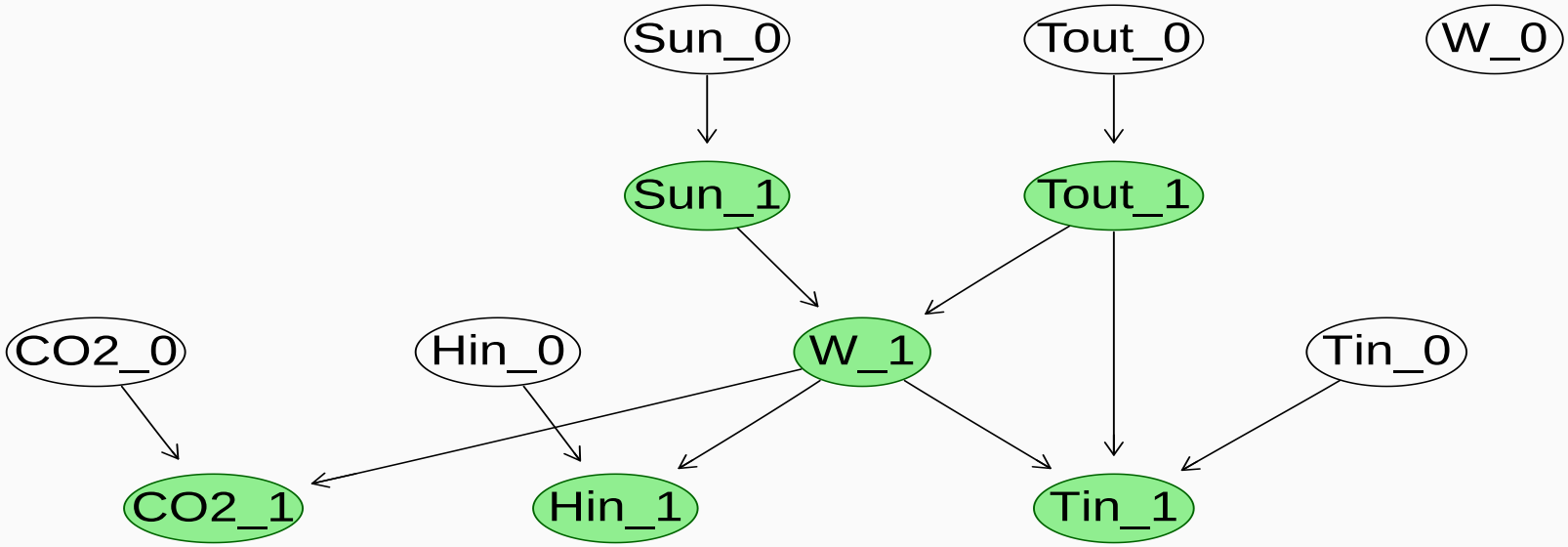
Hence we can assume that we know all the values of the variables in the first time point from the sensors attached to the microcontroller, and we can just concentrate on modelling the dependencies between the variables in the first time point and the variables in the second time point.

(If we condition on all variables in the first time point, we do not really care about how they interact with each other!)

A Simpler Dynamic BN

This design choice reduces the number of arcs quite a bit:

```
win.dyn = model2network(paste0("[Tout_0][Sun_0][W_0][Tin_0][Hin_0][CO2_0]",
    "[Tout_1|Tout_0][Sun_1|Sun_0][W_1|Tout_1:Sun_1][Tin_1|Tin_0:Tout_1:W_1]",
    "[Hin_1|Hin_0:W_1][CO2_1|CO2_0:W_1]"))
graphviz.plot(win.dyn, shape = "ellipse", highlight = list(nodes = nodes.t1,
  fill = "palegreen2", col = "darkgreen"))
```

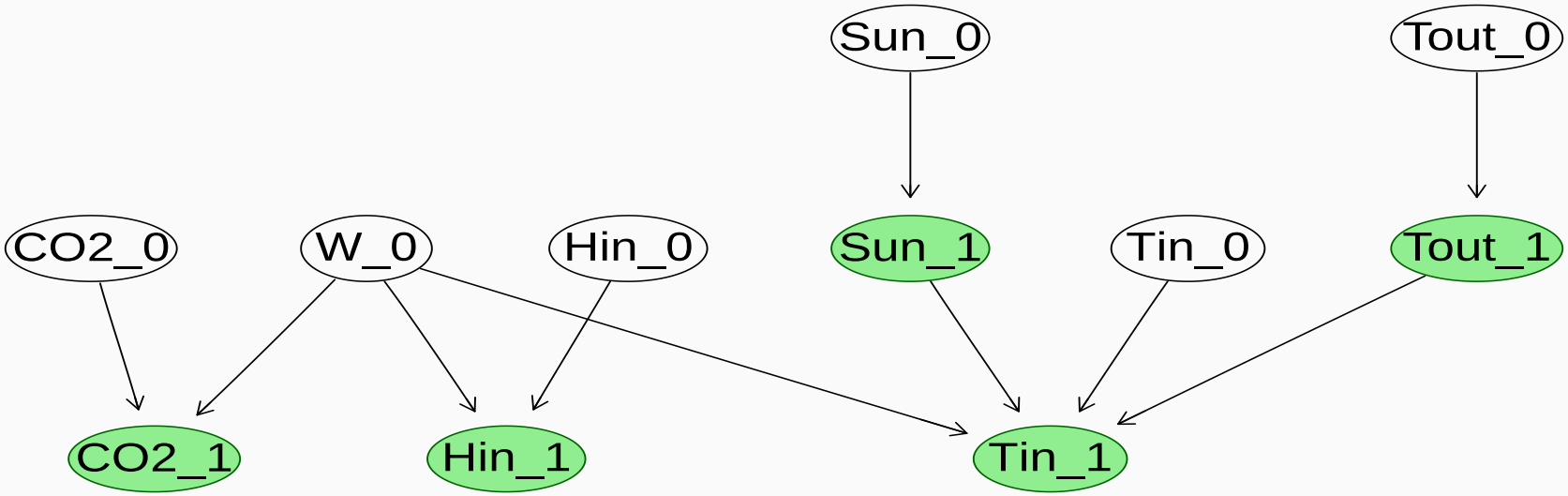


But having `W_0` as an isolated node makes no sense...

Redefining the Meaning of W_0 / W_1

So, we just drop W_1 and keep W_0; opening the windows in the first time point affects CO2_1, Hin_1, Tin_1 instead of CO2_0, Hin_0, Tin_0.

```
win.dyn = model2network(paste0("[Tout_0][Sun_0][W_0][Tin_0][Hin_0][CO2_0]",
    "[Tout_1|Tout_0][Sun_1|Sun_0][Tin_1|Tin_0:Tout_1:W_0:Sun_1]",
    "[Hin_1|Hin_0:W_0][CO2_1|CO2_0:W_0]"))
nodes.t1 = grep("_1$", nodes(win.dyn), value = TRUE)
graphviz.plot(win.dyn, shape = "ellipse", highlight = list(nodes = nodes.t1,
  fill = "palegreen2", col = "darkgreen"))
```



Given this DAG, the probabilities we need to specify are:

1. the initial probabilities of the states of each variable at time 0 (the `*_0` nodes);
2. the transition probabilities controlling the states of each variable at time 1 given the parents (which are at both time 0 and time 1).

Overall, the number of probabilities of this latest incarnation of the BN is:

$$\underbrace{2}_{\texttt{CO2\_0}} + \underbrace{2 \times 3 \times 2}_{\texttt{CO2\_1}} + \underbrace{1}_{\texttt{W\_0}} + \underbrace{1}_{\texttt{Hin\_0}} + \underbrace{1 \times 2 \times 2}_{\texttt{Hin\_1}} + \underbrace{1}_{\texttt{Sun\_0}} + \underbrace{1 \times 2}_{\texttt{Sun\_1}} + \underbrace{2}_{\texttt{Tin\_0}} + \underbrace{2 \times 3 \times 3 \times 3}_{\texttt{Tin\_1}} + \underbrace{2}_{\texttt{Tout\_0}} + \underbrace{2 \times 3}_{\texttt{Tout\_1}} = 87$$

Transitions: Outside Temperature and Sunlight

Assuming that time 0 and time 1 are close-ish in time, we assign high probabilities to the outside temperature and sunlight to be about the same while allowing them to change.

```
Tout_0.prob = array(c(0.20, 0.70, 0.10), dim = 3,
                dimnames = list(Tout_0 = T.lv))
Tout_1.prob = array(c(0.80, 0.19, 0.01, 0.10, 0.80, 0.10, 0.01, 0.19, 0.80),
                dim = c(3, 3), dimnames = list(Tout_1 = T.lv, Tout_0 = T.lv))
Tout_1.prob
```

```
         Tout_0
  Tout_1   <18 18-24  >24
    <18   0.80   0.1 0.01
    18-24 0.19   0.8 0.19
    >24   0.01   0.1 0.80
```

```
Sun_0.prob = array(c(0.70, 0.30), dim = 2, dimnames = list(Sun_0 = Sun.lv))
Sun_1.prob = array(c(0.70, 0.30, 0.30, 0.70), dim = c(2, 2),
               dimnames = list(Sun_1 = Sun.lv, Sun_0 = Sun.lv))
Sun_1.prob
```

```
        Sun_0
  Sun_1  low high
    low  0.7  0.3
    high 0.3  0.7
```

Transitions: CO_2

```
CO2_0.prob = array(c(0.15, 0.50, 0.35), dim = 3,
                   dimnames = list(CO2_0 = CO2.lv))
CO2_1.prob = array(c(0.99, 0.01, 0, 0.60, 0.40, 0, 0.30, 0.69, 0.01,
                     0.80, 0.20, 0, 0.10, 0.70, 0.20, 0.01, 0.14, 0.85),
                   dim = c(3, 3, 2),
                   dimnames = list(CO2_1 = CO2.lv, CO2_0 = CO2.lv, W_0 = W.lv))
```

```
CO2_1.prob
```

```
, , W_0 = open

        CO2_0
CO2_1     low medium high
  low    0.99    0.6 0.30
  medium 0.01    0.4 0.69
  high   0.00    0.0 0.01

, , W_0 = closed

        CO2_0
CO2_1    low medium high
  low    0.8    0.1 0.01
  medium 0.2    0.7 0.14
  high   0.0    0.2 0.85
```

If the windows are open, CO_2 should go down; if they are closed it should go up.

In either case, CO_2 may stay the same with a reasonably high probability.

Transitions: Humidity

```
Hin_0.prob = array(c(0.50, 0.50), dim = 2, dimnames = list(Hin_0 = Hin.lv))
Hin_1.prob = array(c(0.90, 0.10, 0.30, 0.70, 0.70, 0.30, 0.01, 0.99),
                   dim = c(2, 2, 2),
                   dimnames = list(Hin_1 = Hin.lv, Hin_0 = Hin.lv, W_0 = W.lv))
```

```
Hin_1.prob
```

```
, , W_0 = open

       Hin_0
Hin_1   low high
  low   0.9  0.3
  high  0.1  0.7

, , W_0 = closed

       Hin_0
Hin_1   low high
  low   0.7 0.01
  high  0.3 0.99
```

If the windows are open, humidity inside the house may stay low or go down (if high).

If the windows are closed, humidity it should go up or stay high.

In either case, humidity may stay the same as well.

Transitions: Inside Temperature

When we talk about large tables of probabilities being unwieldy or unfeasible to fill, this is what we mean:

```
Tin_0.prob = array(c(0.10, 0.85, 0.05), dim = 3, dimnames = list(Tin_0 = T.lv))
Tin_1.prob = array(c(0.90, 0.10, 0, 0.50, 0.50, 0, 0.20, 0.70, 0.10,
                     0.90, 0.10, 0, 0.10, 0.90, 0, 0, 0.20, 0.80,
                     0.85, 0.15, 0, 0.45, 0.55, 0, 0.10, 0.75, 0.15,
                     0.85, 0.15, 0, 0.05, 0.90, 0.05, 0, 0.05, 0.95,
                     0.10, 0.90, 0, 0.10, 0.80, 0.10, 0, 0.50, 0.50,
                     0.50, 0.50, 0, 0, 0.90, 0.10, 0, 0.10, 0.90,
                     0.05, 0.90, 0.05, 0.05, 0.80, 0.15, 0, 0.45, 0.55,
                     0.45, 0.50, 0.05, 0, 0.85, 0.15, 0, 0.05, 0.95,
                     0.10, 0.70, 0.20, 0, 0.90, 0.10, 0, 0.50, 0.50,
                     0.05, 0.65, 0.30, 0, 0.50, 0.50, 0, 0.20, 0.80,
                     0.05, 0.70, 0.25, 0, 0.85, 0.15, 0, 0.45, 0.55,
                     0, 0.65, 0.35, 0, 0.45, 0.55, 0, 0.15, 0.85),
                   dim = c(3, 3, 2, 2, 3),
                   dimnames = list(Tin_1 = T.lv, Tin_0 = T.lv, W_0 = W.lv,
                                   Sun_1 = Sun.lv, Tout_1 = T.lv))
```

The overall pattern is that if windows are open, inside temperature will move towards the outside temperature; but if windows are closed, temperature can only go up.

Transitions: Inside Temperature

```
Tin_1.prob[, , "open", "low", ]
```

```
, , Tout_1 = <18

       Tin_0
Tin_1   <18 18-24 >24
  <18   0.9   0.5 0.2
  18-24 0.1   0.5 0.7
  >24   0.0   0.0 0.1

, , Tout_1 = 18-24

       Tin_0
Tin_1   <18 18-24 >24
  <18   0.1   0.1 0.0
  18-24 0.9   0.8 0.5
  >24   0.0   0.1 0.5

, , Tout_1 = >24

       Tin_0
Tin_1   <18 18-24 >24
  <18   0.1   0.0 0.0
  18-24 0.7   0.9 0.5
  >24   0.2   0.1 0.5
```

```
Tin_1.prob[, , "open", "high", ]
```

```
, , Tout_1 = <18

       Tin_0
Tin_1    <18 18-24  >24
  <18   0.85  0.45 0.10
  18-24 0.15  0.55 0.75
  >24   0.00  0.00 0.15

, , Tout_1 = 18-24

       Tin_0
Tin_1    <18 18-24  >24
  <18   0.05  0.05 0.00
  18-24 0.90  0.80 0.45
  >24   0.05  0.15 0.55

, , Tout_1 = >24

       Tin_0
Tin_1    <18 18-24  >24
  <18   0.05  0.00 0.00
  18-24 0.70  0.85 0.45
  >24   0.25  0.15 0.55
```

Transitions: Inside Temperature

```
Tin_1.prob[, , "closed", "low", ]
```

```
, , Tout_1 = <18

       Tin_0
Tin_1   <18 18-24 >24
  <18   0.9   0.1 0.0
  18-24 0.1   0.9 0.2
  >24   0.0   0.0 0.8

, , Tout_1 = 18-24

       Tin_0
Tin_1   <18 18-24 >24
  <18   0.5   0.0 0.0
  18-24 0.5   0.9 0.1
  >24   0.0   0.1 0.9

, , Tout_1 = >24

       Tin_0
Tin_1    <18 18-24 >24
  <18   0.05   0.0 0.0
  18-24 0.65   0.5 0.2
  >24   0.30   0.5 0.8
```

```
Tin_1.prob[, , "closed", "high", ]
```

```
, , Tout_1 = <18

       Tin_0
Tin_1    <18 18-24  >24
  <18   0.85  0.05 0.00
  18-24 0.15  0.90 0.05
  >24   0.00  0.05 0.95

, , Tout_1 = 18-24

       Tin_0
Tin_1    <18 18-24  >24
  <18   0.45  0.00 0.00
  18-24 0.50  0.85 0.05
  >24   0.05  0.15 0.95

, , Tout_1 = >24

       Tin_0
Tin_1    <18 18-24  >24
  <18   0.00  0.00 0.00
  18-24 0.65  0.45 0.15
  >24   0.35  0.55 0.85
```

Building the New Dynamic BN

The last probability distribution we need is that of `W_0`, which does not really matter since in using the model we will always fix this variable to open or `closed`.

```
W_0.prob = array(c(0.5, 0.5), dim = 2, dimnames = list(W_0 = W.lv))
```

Finally, we build the BN by combining the DAG and the probability distributions using `custom.fit()`.

```
win.dbn = custom.fit(win.dyn, list(Tout_0 = Tout_0.prob, Tout_1 = Tout_1.prob,
            Sun_0 = Sun_0.prob, Sun_1 = Sun_1.prob, W_0 = W_0.prob,
            Tin_0 = Tin_0.prob, Tin_1 = Tin_1.prob, CO2_0 = CO2_0.prob,
            CO2_1 = CO2_1.prob, Hin_0 = Hin_0.prob, Hin_1 = Hin_1.prob))
```

Now, we should check that BN in `win.dbn` provides sensible answers when queried before putting it to use on the microcontroller.

Sanity Checking the Dynamic BN

First, let's check what the dynamic BN tells us about `Tin_1` from what we know at time 0. If the room is cold, and outside is warm, the room will warm up.

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
            evidence = (Tin_0 == "<18") & (Tout_0 == "18-24"))
prop.table(table(in.out))
```

```
in.out
   <18  18-24    >24
0.3291 0.6154 0.0556
```

If the room is cold and outside is hot, it is more likely to warm up and can possibly become hot.

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
            evidence = (Tin_0 == "<18") & (Tout_0 == ">24"))
prop.table(table(in.out))
```

```
in.out
   <18  18-24    >24
0.0547 0.7188 0.2266
```

Sanity Checking the Dynamic BN

The converse also holds: if the room is hot and outside is cold, there is about 50% chance that it will get cooler.

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
             evidence = (Tin_0 == ">24") & (Tout_0 == "<18"))
prop.table(table(in.out))
```

```
in.out
   <18  18-24    >24
0.0784 0.3922 0.5294
```

Finally, if the inside and outside temperature are both temperate then there is no change with high probability.

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
             evidence = (Tin_0 == "18-24") & (Tout_0 == "18-24"))
prop.table(table(in.out))
```

```
in.out
   <18  18-24    >24
0.0631 0.8051 0.1318
```

Sanity Checking the Dynamic BN

Contrast the effect of opening the window with keeping it closed: `Tin_1` is much more likely to converge to the outside temperature if the window is open. Which makes sense.

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
          evidence = (Tin_0 == "<18") & (Tout_0 == "18-24") & (W_0 == "open"))
prop.table(table(in.out))
```

```
  in.out
     <18   18-24    >24
  0.1499 0.8098 0.0403
```

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
        evidence = (Tin_0 == "<18") & (Tout_0 == "18-24") & (W_0 == "closed"))
prop.table(table(in.out))
```

```
  in.out
     <18   18-24    >24
  0.4721 0.4721 0.0557
```

This is also true if outside is cooler than inside...

Sanity Checking the Dynamic BN

... if we open the window the room is more likely to cool down than if the window is closed.

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
           evidence = (Tin_0 == ">24") & (Tout_0 == "18-24") & (W_0 == "open"))
prop.table(table(in.out))
```

```
  in.out
     <18   18-24    >24
  0.0318 0.4904 0.4777
```

```
in.out = cpdist(win.dbn, nodes = "Tin_1",
         evidence = (Tin_0 == ">24") & (Tout_0 == "18-24") & (W_0 == "closed"))
prop.table(table(in.out))
```

```
  in.out
    <18 18-24   >24
  0.000 0.124 0.876
```

Is the same true for humidity and CO_2?

Sanity Checking the Dynamic BN

```
prop.table(table(cpdist(win.dbn, nodes = "Hin_1",
                  evidence = (Hin_0 == "low") & (W_0 == "open"))))
```

```
   low  high
 0.892 0.108
```

```
prop.table(table(cpdist(win.dbn, nodes = "Hin_1",
                  evidence = (Hin_0 == "low") & (W_0 == "closed"))))
```

```
   low  high
 0.706 0.294
```

```
prop.table(table(cpdist(win.dbn, nodes = "Hin_1",
                  evidence = (Hin_0 == "high") & (W_0 == "open"))))
```

```
   low  high
 0.299 0.701
```

```
prop.table(table(cpdist(win.dbn, nodes = "Hin_1",
                  evidence = (Hin_0 == "high") & (W_0 == "closed"))))
```

```
    low   high
 0.0104 0.9896
```

The probability of `low` is always higher if the windows are closed.

Sanity Checking the Dynamic BN

```
prop.table(table(cpdist(win.dbn, nodes = "CO2_1",
                  evidence = (CO2_0 == "low") & (W_0 == "open"))))
```

```
    low medium   high
 0.9802 0.0198 0.0000
```

```
prop.table(table(cpdist(win.dbn, nodes = "CO2_1",
                  evidence = (CO2_0 == "low") & (W_0 == "closed"))))
```

```
    low medium   high
  0.785  0.215  0.000
```

```
prop.table(table(cpdist(win.dbn, nodes = "CO2_1",
                  evidence = (CO2_0 == "high") & (W_0 == "open"))))
```

```
     low  medium    high
 0.29375 0.69650 0.00975
```

```
prop.table(table(cpdist(win.dbn, nodes = "CO2_1",
                  evidence = (CO2_0 == "high") & (W_0 == "closed"))))
```

```
    low medium   high
 0.0161 0.1348 0.8491
```

Same for CO_2.

Sanity Checking: Accidental Dependencies?

We expect humidity and CO_2 to be independent, and they actually are since they are d-separated in the dynamic BN. (The small differences in the probabilities are simulation noise.)

```
hin.co2 = rbn(win.dbn, 1000)[, c("Hin_0", "CO2_0")]
prop.table(table(hin.co2))
```

```
        CO2_0
 Hin_0     low medium  high
   low   0.080  0.252 0.194
   high  0.076  0.222 0.176
```

```
dsep(win.dbn, "Hin_0", "CO2_0")
```

```
 [1] TRUE
```

Same for humidity and inside temperature.

```
hin.tin = rbn(win.dbn, 1000)[, c("Hin_0", "Tin_0")]
prop.table(table(hin.tin))
```

```
        Tin_0
 Hin_0     <18 18-24   >24
   low   0.046 0.418 0.016
   high  0.058 0.439 0.023
```

```
dsep(win.dbn, "Hin_0", "Tin_0")
```

```
 [1] TRUE
```

Sanity Checking: Accidental Dependencies?

However, while Hin_0 and CO2_0 are independent, it turns out that Hin_1 and CO2_1 are not!

```
hin.co2 = rbn(win.dbn, 1000)[, c("Hin_1", "CO2_1")]
prop.table(table(hin.co2))
```

```
        CO2_1
 Hin_1     low medium  high
   low   0.203  0.207 0.061
   high  0.183  0.227 0.119
```

```
dsep(win.dbn, "Hin_1", "CO2_1")
```

```
 [1] FALSE
```

This is not unexpected: they have W_0 as a common parent. Hence unless we condition on W_0, there is an open path between Hin_1 and CO2_1 going through W_0 which makes the nodes not graphically separated.

This makes sense intuitively: if the windows are open, both humidity and CO_2 go down, and if they are closed both go up.

Sanity Checking: Accidental Dependencies?

If we condition on `W_0`, `Hin_1` and `CO2_1` become graphically separated and therefore they are independent by definition.

```
dsep(win.dbn, "Hin_1", "CO2_1", "W_0")
```

```
[1] TRUE
```

This again makes sense intuitively: if we do not know whether the windows are open or not, we observe that humidity and CO_2 go up and down simultaneously and therefore we may think they influence each other. But if we know that the windows are open, then we realise that that is not the case and that both humidity and CO_2 go down because fresh air is coming in. And if the windows are closed they build up.

Extending a Dynamic BN Over More Time Points

Recall that the task we are fitting this dynamic model for is to put it on a microcontroller to automatically open and close the windows. We likely would want to open them if it is too hot inside the house, or if CO_2 is too high so the air is stuffy; and to close them if it is too cold.

A the most basic level, this means that we must predict inside temperature and CO_2 for time 1 given what we know at time 0. Like this:

```
pred.values = cpdist(win.dbn, nodes = c("Tin_1", "CO2_1"), method = "lw",
                evidence = list(CO2_0 = "high", Hin_0 = "high", Sun_0 = "high",
                                Tin_0 = "18-24", Tout_0 = "<18", W_0 = "open"))
values.prob = prop.table(table(pred.values))
values.prob
```

```
         CO2_1
  Tin_1      low medium   high
    <18   0.1225 0.2575 0.0028
    18-24 0.1780 0.4037 0.0064
    >24   0.0089 0.0200 0.0002
```

Extending a Dynamic BN Over More Time Points

Having the window open, the dynamic BN tells us that:

- The best result would be `Tin_1` = `18-24` and `CO2_1` = `low`, which is predicted to happen with probability 0.178.
- The most likely outcome is `Tin_1` = `18-24` and `CO2_1` = `medium` with probability 0.404.
- The probability of the house getting too hot or for CO_2 to become too high are negligible (0.029 and 0.009 respectively).
- The house can possibly get too cold with probability 0.383, and we do not want that to happen.

But how can we know when to close the windows? The dynamic BN can only predict one time point forward; it only models the transition between a generic time point 0 and the following time point 1.

Extending a Dynamic BN Over More Time Points

But what if we assume that the same dynamic BN is a valid model for the transition between time 1 and time 2? Then we can:

- take the values of all the variables of time 0;
- compute the probabilities of all the variables at time 1;
- assess whether it is a good time to close the windows; if not
- create a copy of the dynamic BN relabelling "time 0" as "time 1" and "time 1" as "time 2";
- take the marginal probabilities for the the variables at time 1 and feed them into the new dynamic BN;
- compute the the probabilities of all the variables at time 2.

This procedure can be repeated as many times as needed, to obtain probabilities for events at any point in the future.

Manual Two-Times Dynamic BN

- Create a copy of the dynamic BN relabelling “time 0” as “time 1” and “time 1” as “time 2”;

```
win.dbn2 = win.dbn
nodes(win.dbn)
```

```
  [1] "CO2_0"  "CO2_1"  "Hin_0"  "Hin_1"  "Sun_0"  "Sun_1"
  [7] "Tin_0"  "Tin_1"  "Tout_0" "Tout_1" "W_0"
```

```
nodes(win.dbn2) = gsub("_1", "_2", nodes(win.dbn))
nodes(win.dbn2) = gsub("_0", "_1", nodes(win.dbn2))
nodes(win.dbn2)
```

```
  [1] "CO2_1"  "CO2_2"  "Hin_1"  "Hin_2"  "Sun_1"  "Sun_2"
  [7] "Tin_1"  "Tin_2"  "Tout_1" "Tout_2" "W_1"
```

- take the marginal probabilities for the the variables at time 1;

```
pred.values = cpdist(win.dbn, nodes = nodes(win.dbn), method = "lw",
                evidence = list(CO2_0 = "high", Hin_0 = "high",
                                Sun_0 = "high", Tin_0 = "18-24",
                                Tout_0 = "<18", W_0 = "open"))
CO2_1.new = prop.table(table(pred.values[, "CO2_1"]))
Hin_1.new = prop.table(table(pred.values[, "Hin_1"]))
Sun_1.new = prop.table(table(pred.values[, "Sun_1"]))
Tin_1.new = prop.table(table(pred.values[, "Tin_1"]))
Tout_1.new = prop.table(table(pred.values[, "Tout_1"]))
```

Manual Two-Times Dynamic BN

- feed them into the new dynamic BN;

```
win.dbn2$CO2_1 = CO2_1.new
win.dbn2$Hin_1 = Hin_1.new
win.dbn2$Sun_1 = Sun_1.new
win.dbn2$Tin_1 = Tin_1.new
win.dbn2$Tout_1 = Tout_1.new
```

- compute the joint probability of `Tin_2` and `CO2_2`.

```
pred.values = rbn(win.dbn2, 5000)[, c("Tin_2", "CO2_2")]
values.prob2 = prop.table(table(pred.values))
values.prob2
```

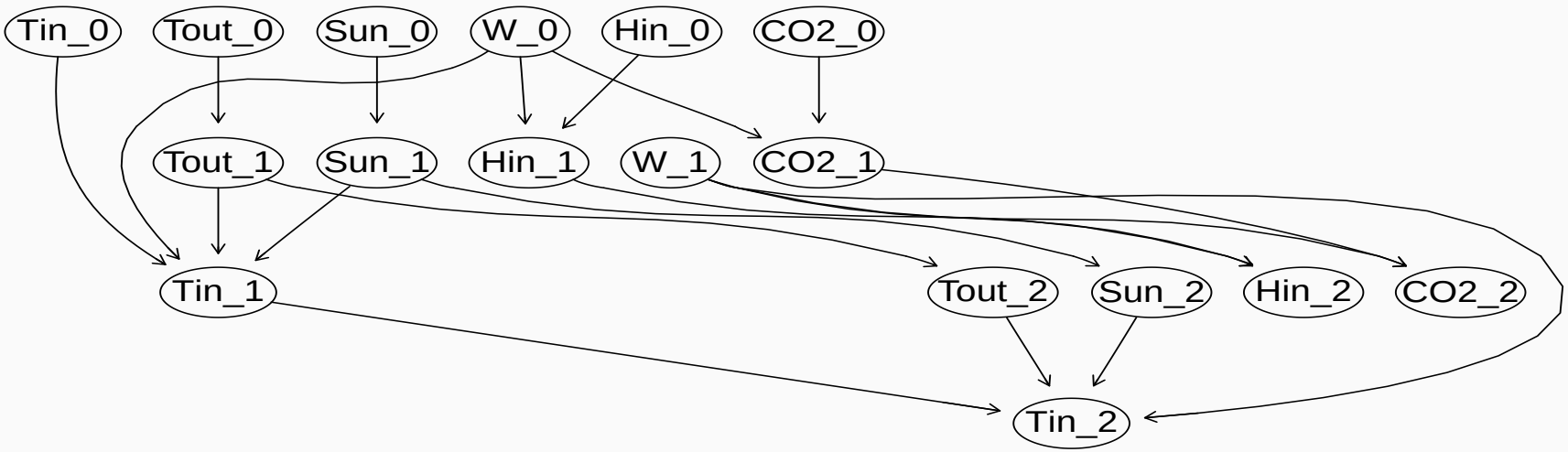
```
         CO2_2
Tin_2        low medium   high
  <18    0.2044 0.1378 0.0204
  18-24  0.2860 0.2512 0.0464
  >24    0.0246 0.0234 0.0058
```

What can we see? The probability of `18-24` and `medium` is much lower at time 2 compared to time (0.251 down from 0.404), while the probability of low CO_2 is higher (0.515 up from 0.309). The effect of the window being kept open...

Systematic Two-Times Dynamic BN

A more systematic approach is to build a dynamic BN spanning all of times 0, 1, and 2.

```
win.dyn2 = model2network(paste0("[Tout_0][Sun_0][W_0][Tin_0][Hin_0][CO2_0]",
             "[Tout_1|Tout_0][Sun_1|Sun_0][Tin_1|Tin_0:Tout_1:W_0:Sun_1]",
             "[Hin_1|Hin_0:W_0][CO2_1|CO2_0:W_0][Tout_2|Tout_1][Sun_2|Sun_1]",
             "[Tin_2|Tin_1:Tout_2:W_1:Sun_2][Hin_2|Hin_1:W_1]",
             "[CO2_2|CO2_1:W_1][W_1]"))
nodes.t0 = grep("_0$", nodes(win.dyn2), value = TRUE)
nodes.t1 = grep("_1$", nodes(win.dyn2), value = TRUE)
nodes.t2 = grep("_2$", nodes(win.dyn2), value = TRUE)
graphviz.plot(win.dyn2, groups = list(nodes.t0, nodes.t1, nodes.t2),
  shape = "ellipse")
```



Systematic Two-Times Dynamic BN

Each variable is replicated in time 2; so we need to copy the probability tables as they are, just changing the variable names.

```
Tin_2.prob = Tin_1.prob
names(dimnames(Tin_2.prob)) = c("Tin_2", "Tin_1", "W_1", "Sun_2", "Tout_2")
Tout_2.prob = Tout_1.prob
names(dimnames(Tout_2.prob)) = c("Tout_2", parents(win.dyn2, "Tout_2"))
Sun_2.prob = Sun_1.prob
names(dimnames(Sun_2.prob)) = c("Sun_2", parents(win.dyn2, "Sun_2"))
Hin_2.prob = Hin_1.prob
names(dimnames(Hin_2.prob)) = c("Hin_2", parents(win.dyn2, "Hin_2"))
CO2_2.prob = CO2_1.prob
names(dimnames(CO2_2.prob)) = c("CO2_2", parents(win.dyn2, "CO2_2"))
win.bn2 = custom.fit(win.dyn2, list(Tout_0 = Tout_0.prob, Tout_1 = Tout_1.prob,
            Tout_2 = Tout_2.prob, Sun_0 = Sun_0.prob, Sun_1 = Sun_1.prob,
            Sun_2 = Sun_2.prob, W_0 = W_0.prob, W_1 = W_0.prob,
            Tin_0 = Tin_0.prob, Tin_1 = Tin_1.prob, Tin_2 = Tin_2.prob,
            CO2_0 = CO2_0.prob, CO2_1 = CO2_1.prob, CO2_2 = CO2_2.prob,
            Hin_0 = Hin_0.prob, Hin_1 = Hin_1.prob, Hin_2 = Hin_2.prob))
```

Comparing Predicted Probabilities

Since we now have all of times 0, 1 and 2 in a single BN, we can perform a query about time 2 conditioning on the evidence we observe at time 0 without resorting to the iterative procedure we used earlier.

```
pred.values = cpdist(win.bn2, nodes = c("Tin_2", "CO2_2"), method = "lw",
                evidence = list(CO2_0 = "high", Hin_0 = "high", Sun_0 = "high",
                                Tin_0 = "18-24", Tout_0 = "<18", W_0 = "open"))
prop.table(table(pred.values))
```

```
        CO2_2
Tin_2       low medium   high
  <18   0.2120 0.1492 0.0252
  18-24 0.2701 0.2404 0.0442
  >24   0.0264 0.0272 0.0053
```

Same as before, modulo some simulation variability:

```
values.prob2
```

```
        CO2_2
Tin_2       low medium   high
  <18   0.2044 0.1378 0.0204
  18-24 0.2860 0.2512 0.0464
  >24   0.0246 0.0234 0.0058
```

R Programming: a Summary

- `model2network()` and `empty.graph()` + `set.arc()` to create the graph of a Bayesian network.
- `custom.fit()` to create the networks from expert knowledge.
- `bn.fit()` to estimate the probabilities from data.
- `bn.cv()` to tests predictive accuracy using cross-validation.
- `nparams()` to count the parameters; `nnodes()` gives the number of nodes and `narcs()` gives the number of arcs.
- `cpdist()`, `cpquery()` and `predict()` to use the networks for inference, either with queries or prediction. (More on that in the next lectures.)

Summary and Remarks

- Bayesian networks can be used to model the most disparate problems.
- The information we need to construct a Bayesian network can come from experts, from data or both; if it comes from data we must make sure we have enough to estimate the probabilities well.
- Bayesian networks can be dynamic and model phenomena that evolve over time.
- In any case, we must sanity check them before putting them to use:
  - making sure they are not sensitive to small changes in the probabilities for various variables, and
  - using queries we know the answer of to check the answers we get are sensible.

Inference

Events, Evidence and Queries

A BN represents a working model of the world that a computer can understand; but how does a computer system use it to help and perform its assigned task?

We ask questions, and the computer system performs probabilistic inference to answer them and decide what to do in the process.

Questions that can be asked are called queries and are typically about an event of interest given some evidence. The evidence is the input to the computer system ("Someone with a high-school degree.") and the event is the output ("A man driving a car."). This is often called belief update: we observe some evidence and we update our beliefs before taking action.

Events, Evidence and Queries

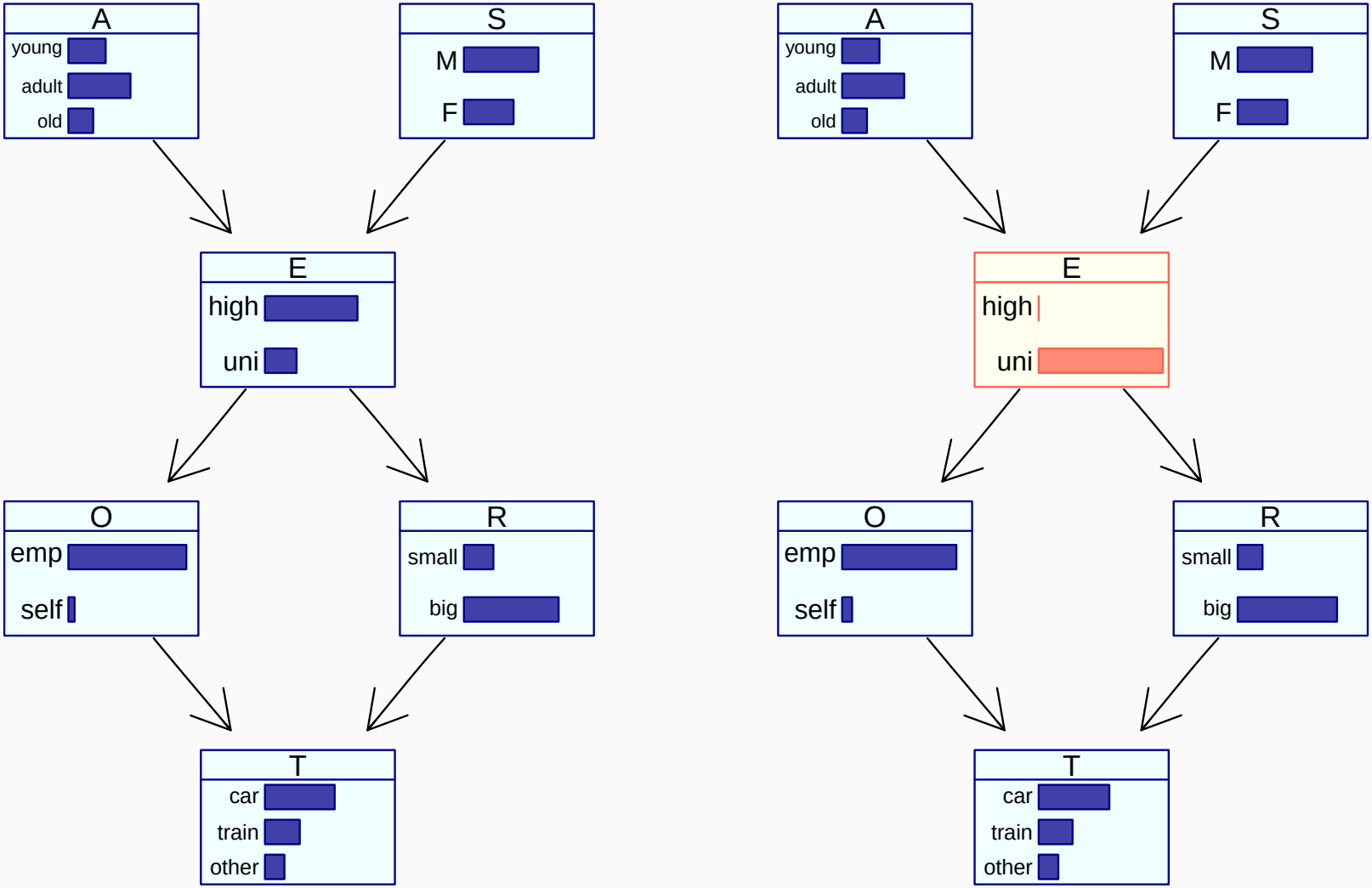
The two most common queries are

- conditional probability queries ("What is the probability that someone with high-school degree is a man driving a car?"); and
- most probable explanation queries ("What is the most probable sex and mode of transportation for someone with a high-school degree?")

In both cases the evidence is hard evidence: we set some variables to particular values. Then the computer system checks how the probabilities of other variables change and provides an answer to the query.

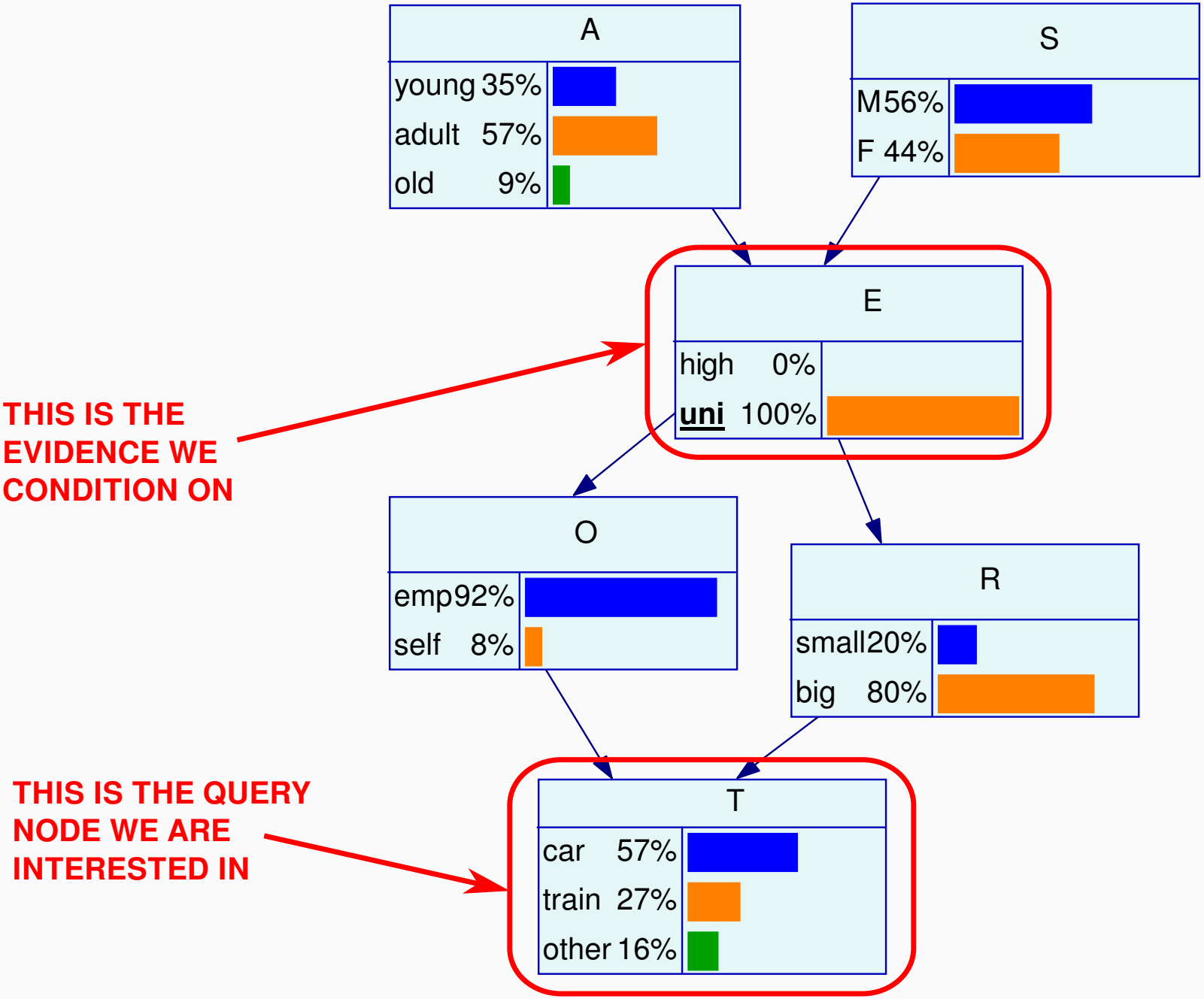
No more manual probability computations...

The Effects of Conditioning on Hard Evidence

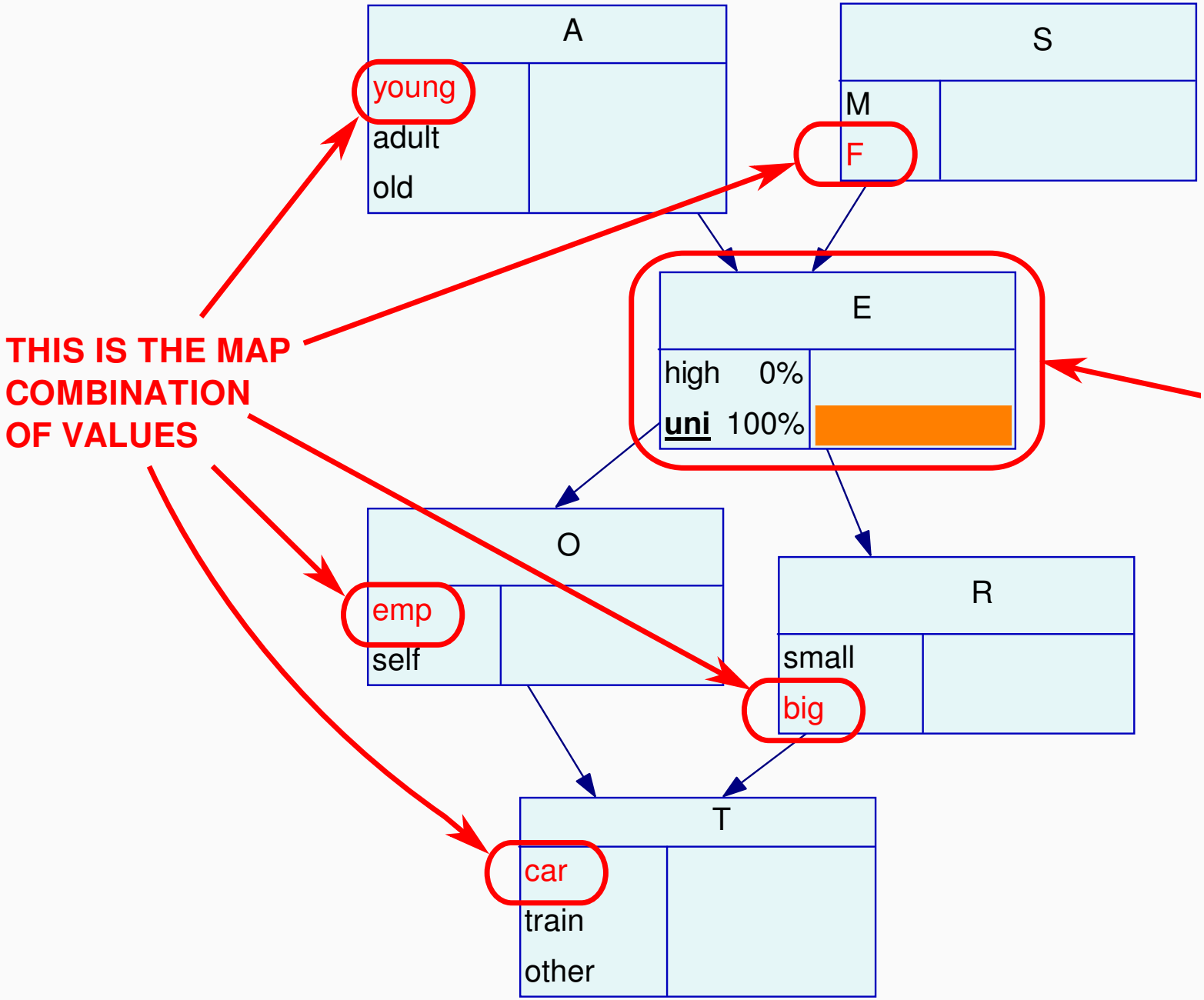


The original survey BN (left), and the posterior BN with hard evidence on Education (right).

Conditional Probability Queries in Pictures



Maximum a Posteriori Queries in Pictures



Exact and Approximate Inference

There are two approaches to answer queries using BNs.

Exact algorithms use the DAG of a BN to schedule and perform repeated applications of Bayes theorem and the probability axioms on the probabilities in the model. In other words, the computer system uses the DAG to perform all the math we did by hand in earlier lectures.

The two best known are

- variable elimination; and
- belief updates based on junction trees.

PROS: they return exact values for the probabilities of interest.
CONS: they do not scale well when BNs have many nodes and many arcs.

Exact and Approximate Inference

Approximate algorithms use the BN as a model of the world in a very literal sense. In the real world to answer some question in a scientific, rigorous way we would perform an experiment and observe the outcome; approximate algorithms imitate this process by generating random observations from the BN, thus running a simulated experiment that approximates reality.

The two best known are

- logic sampling; and
- likelihood weighting.

PROS: they scale really well when BNs have many nodes and many arcs.
CONS: they return approximate, estimated values for the probabilities of interest.

The Logic Sampling Algorithm

INPUT: a BN, evidence E and query event Q.

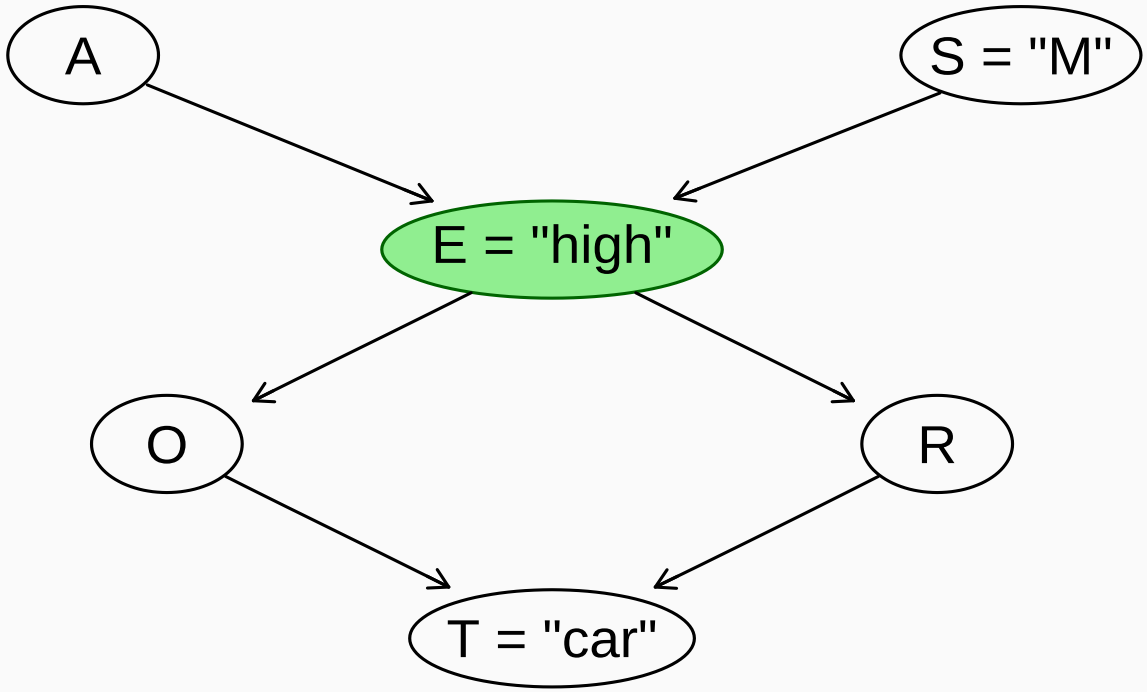
1. Order the variables in $\mathbf{X}$ according to the topological ordering in the DAG (from top to bottom), so that parents come before children.
2. Set $n_E = 0$ and $n_{E,Q} = 0$.
3. For a suitably large number of samples $\mathbf{x}$:
   - 3.1 generate a random value from each $X_i \mid \Pi_{X_i}$ taking advantage of the fact that, thanks to the topological ordering, by the time we are considering X_i we have already generated the values of all its parents Π_{X_i};
   - 3.2 if $\mathbf{x}$ includes E, set $n_E = n_E + 1$;
   - 3.3 if $\mathbf{x}$ includes both Q and E, set $n_{E,Q} = n_{E,Q} + 1$.
4. The answer to the query is the estimated probability $n_{E,Q}/n_E$.

A Survey Example

Consider:

- the evidence: someone whose Education (`E`) level is a high school diploma (`high`)...
- the event: ... is a man (`S` is equal to `M`) uses a `car` as a means of Transportation (`T`).

We will answer this query using the different inference algorithms.



Stepping Through Logic Sampling

First, we sample from the BN with `rbn()`, which takes a `bn.fit` object and the number of random samples to generate as arguments.

```
particles = rbn(bn, 10^6)
head(particles, n = 5)
```

```
       A    E   O   R S     T
1   old high emp big M train
2   old high emp big M   car
3 adult high emp big F   car
4   old high emp big M other
5 young high emp big M   car
```

The samples have the correct types and format as derived from the BN, and they are stored in a data frame that has the same structure as that of the data that were used to learn the BN (if any).

Stepping Through Logic Sampling

Then we count how many of those samples that match the evidence E to estimate $\mathrm{P}(E)$.

```
partE = particles[(particles[, "E"] == "high"), ]
nE = nrow(partE)
```

We also count how many of those samples that match the evidence E and the query event Q to estimate $\mathrm{P}(Q, E)$.

```
partEQ =
  partE[(partE[, "S"] == "M") & (partE[, "T"] == "car"), ]
nEQ = nrow(partEQ)
```

Finally, we estimate

$$\mathrm{P}(Q \mid E) = \frac{\mathrm{P}(Q, E)}{\mathrm{P}(E)}.$$

```
nEQ/nE
```

```
[1] 0.343
```

The cpquery() Function

These steps are implemented in cpquery(), with the obvious arguments:

- event is Q;
- evidence is E;
- method is "ls" for logic sampling (the default);
- n is the number of random samples.

```
cpquery(bn, event = (S == "M") & (T == "car"),
          evidence = (E == "high"), method = "ls", n = 10^6)
```

```
[1] 0.343
```

Both event end evidence are expressions that are evaluated on the random samples much like subset() would, so they must evaluate to a vector of TRUE and FALSE values (hence & and not &&).

More Advanced Queries with cpquery()

Specifying the arguments requires some care, but the result is an extremely flexible framework to compute the probability of arbitrary combinations of events.

As an example of a more complex query, we can compute

$$\mathrm{P}(\mathtt{S} = \mathtt{M}, \mathtt{T} = \mathtt{car} \mid \{\mathtt{A} = \mathtt{young}, \mathtt{E} = \mathtt{uni}\} \cup \{\mathtt{A} = \mathtt{adult}\}),$$

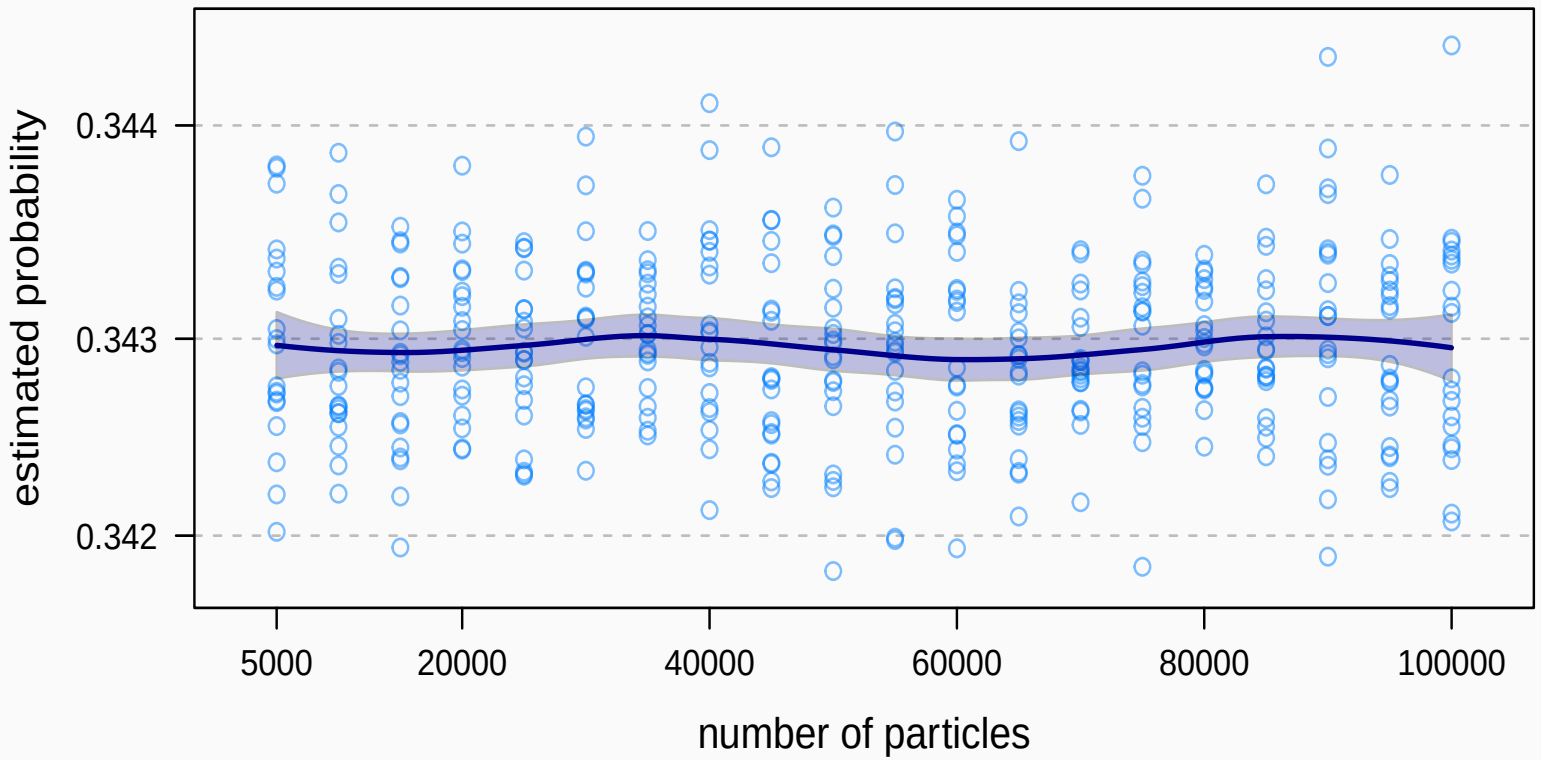
the probability of a man travelling by car given that his Age is young and his Education is uni or that he is an adult, regardless of his Education. That would be:

```
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = ((A == "young") & (E == "uni")) | (A == "adult"))
```

```
[1] 0.338
```

Stepping Through Logic Sampling

```
nparticles = seq(from = 5 * 10^3, to = 10^5, by = 5 * 10^3)
prob = matrix(0, nrow = length(nparticles), ncol = 20)
for (i in seq_along(nparticles))
  for (j in 1:20)
    prob[i, j] = cpquery(bn, event = (S == "M") & (T == "car"),
                  evidence = (E == "high"), method = "ls", n = 10^6)
```



The Limits of Logic Sampling

Notice anything in the figure in the previous slide?

- Logic sampling is obviously affected by sampling variability: every time we run it we get a different estimate of the probability that is the answer to our query because the random samples we generate will be different.
- Sampling variability decreases with the number of samples we generate, but it never goes to zero; there is always some uncertainty around the exact value we estimate (here 0.343 ± 0.001).
- Remember that we essentially discard all random samples that do not match the evidence we condition on, so if the evidence has low probability we are throwing out almost all samples we generate.

The Likelihood Weighting Algorithm

An improvement over logic sampling, designed to solve this problem, is the likelihood weighting algorithm. Unlike logic sampling, all the random samples generated by likelihood weighting include the evidence E by design.

1. Order the variables in $\mathbf{X}$ according to the topological ordering in the DAG (from top to bottom), so that parents come before children.
2. Set $w_E = 0$ and $w_{E,Q} = 0$.
3. For a suitably large number of samples $\mathbf{x}$:
   - 3.1 generate a random value from each $X_i \mid \Pi_{X_i}$ and fix the relevant variables to the values specified by the evidence E.
   - 3.2 compute the weight $w_{\mathbf{x}} = \mathrm{P}(E)$.
   - 3.3 set $w_E = w_E + w_{\mathbf{x}}$;
   - 3.4 if $\mathbf{x}$ includes Q , set $w_{E,Q} = w_{E,Q} + w_{\mathbf{x}}$.
4. The answer to the query is the estimated probability $w_{E,Q}/w_E$.

Stepping Through Likelihood Weighting

We do not want to sample from the original BN, but from the BN in which all the nodes covered by E are fixed. This network is called the mutilated network.

Compare:

```
coef(bn$E)
```

```
, , S = M

      A
E      young adult  old
  high  0.75  0.72 0.88
  uni   0.25  0.28 0.12

, , S = F

      A
E      young adult  old
  high  0.64  0.70 0.90
  uni   0.36  0.30 0.10
```

```
parents(bn, "E")
```

```
[1] "A" "S"
```

```
mutbn = mutilated(bn, list(E = "high"))
coef(mutbn$E)
```

```
high  uni
   1    0
```

```
parents(mutbn, "E")
```

```
character(0)
```

No parents, and the value is that in the evidence with probability equal to 1.

Stepping Through Likelihood Weighting

Simply sampling from `mutbn` is not a correct way of answering our query! A simple empirical check tells us that the naive estimate we could draw from `mutbn` is wrong, since it does not match the exact value we got earlier.

```
particles = rbn(mutbn, 10^6)
partE = particles[(particles[, "E"] == "high"), ]
partEQ = partE[(particles[, "S"] == "M") &
               (particles[, "T"] == "car"), ]
nrow(partEQ) / nrow(partE)
```

```
[1] 0.336
```

That is because `nrow(partE)` is identical to `nrow(particles)` by construction, so the conditional probability is not computed correctly. What we get is:

$$\mathrm{P}(Q, E) = \frac{n_{E,Q}}{n} \neq \mathrm{P}(Q \mid E).$$

Stepping Through Likelihood Weighting

The weights adjust for the fact that we are sampling from the mutilated BN instead of the original BN. The weights are just the likelihood components associated with the nodes we are conditioning on (E in this case):

```
w = logLik(bn, particles, nodes = "E", by.sample = TRUE)
wEQ = sum(exp(w[(particles[, "S"] == "M") &
                (particles[, "T"] == "car")]))
wE = sum(exp(w))
wEQ/wE
```

```
[1] 0.343
```

NOTE: the likelihood of an observation has the same mathematical expression as its probability, so for practical purposes here it is just $\mathrm{P}(E)$. `logLik()` returns $\log \mathrm{P}(E)$ in the code above.

Stepping Through Likelihood Weighting

More conveniently, we can perform likelihood weighting with `cpquery` by setting `method = "lw"` and specifying the evidence as a named list with one element for each node we are conditioning on.

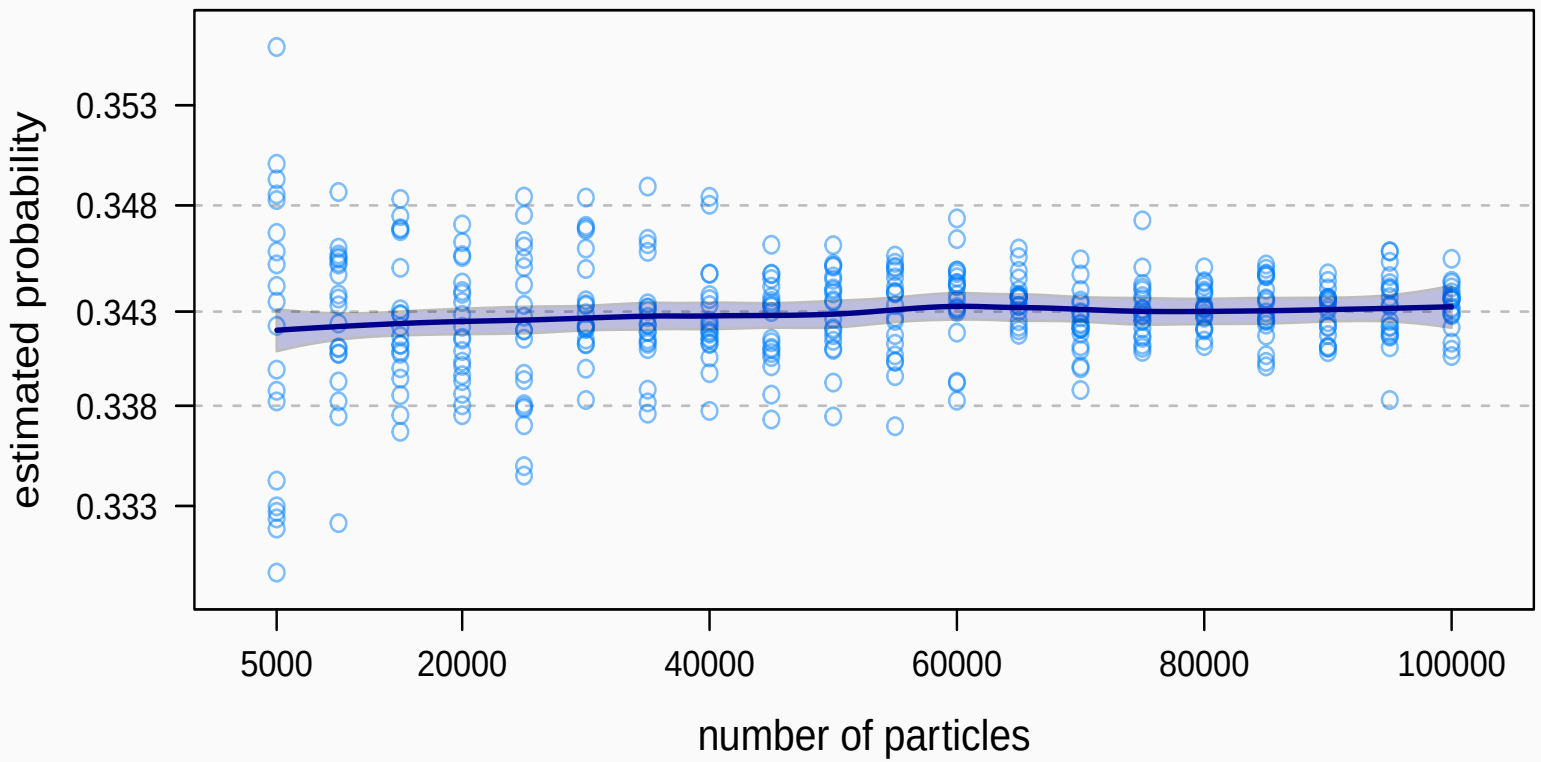
```
cpquery(bn, event = (S == "M") & (T == "car"),
          evidence = list(E = "high"), method = "lw", n = 5 * 10^4)
```

```
[1] 0.343
```

The estimate we obtain is still very precise with small numbers of random samples, as was the case for logic sampling, but the variability of the estimated probabilities is actually larger. There is no guarantee that likelihood weighting will always have lower variance than logic sampling.

Stepping Through Likelihood Weighting

```
nparticles = seq(from = 5 * 10^3, to = 10^5, by = 5 * 10^3)
prob = matrix(0, nrow = length(nparticles), ncol = 20)
for (i in seq_along(nparticles))
  for (j in 1:20)
    prob[i, j] = cpquery(bn, event = (S == "M") & (T == "car"),
                   evidence = list(E = "high"), method = "lw",
                   n = nparticles[i])
```



Then Why Use Likelihood Weighting?

Logic sampling will be computationally inefficient and very inaccurate if $\mathrm{P}(E)$ is small because most random samples will be discarded without contributing to the estimation of $\mathrm{P}(Q \mid E)$.

```
extreme.dag = model2network("[A][B|A]")
A.prob = array(c(0.999999, 0.000001), dim = 2,
           dimnames = list(A = c("a1", "a2")))
B.prob = array(c(0.5, 0.5, 0.75, 0.25), dim = c(2, 2),
           dimnames = list(B = c("b1", "b2"), A = c("a1", "a2")))
extreme.bn = custom.fit(extreme.dag, list(A = A.prob, B = B.prob))
cpquery(extreme.bn, event = (B == "b2"), evidence = (A == "a2"),
  method = "ls", n = 10^6)
```

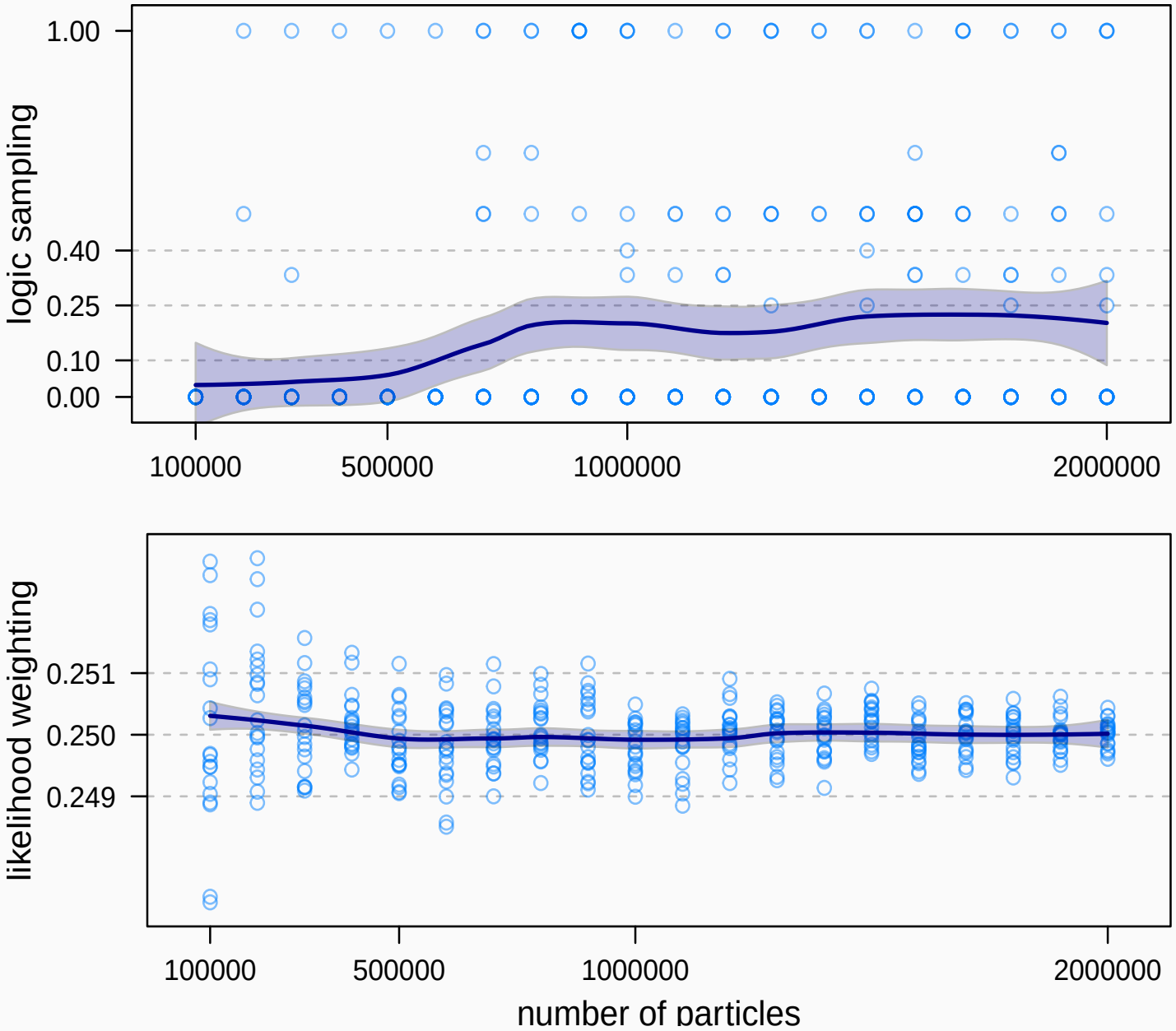
```
[1] 0.333
```

This simply does not happen with likelihood weighting.

```
cpquery(extreme.bn, event = (B == "b2"), evidence = list(A = "a2"),
  method = "lw", n = 5 * 10^3)
```

```
[1] 0.249
```

A Comparison for Different Numbers of Random Samples



Extensions of Likelihood Weighting

The `event` is still a general expression, which means it is possible to describe complex events. However, likelihood weighting relies on the fact that the `evidence` is fixed to a single value to compute the weights. In **bnlearn** this assumption is relaxed: the `event` can take more than one value for each variable. All combinations of values are given the same probability so as not to alter the weights.

```
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = list(A = c("young", "adult")), method = "lw", n = 10^6)
```

```
[1] 0.337
```

```
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = list(A = "young"), method = "lw", n = 10^6) * 0.5 +
cpquery(bn, event = (S == "M") & (T == "car"),
  evidence = list(A = "adult"), method = "lw", n = 10^6) * 0.5
```

```
[1] 0.337
```

Sampling and Conditioning

Last but not least, we can also use `cpdist()` to generate random samples conditional on some evidence E. Likelihood weighting works best, and attaches the weights to the samples (for use in later analyses).

```
cpdist(bn, nodes = c("S", "T"), evidence = list(A = "adult"),
  method = "lw", n = 5)
```

```
  S   T
1 M car
2 F car
3 F car
4 M car
5 F car
```

Logic sampling works less well because it often returns far fewer observations than requested.

```
cpdist(bn, nodes = c("S", "T"), evidence = (A == "young"),
  method = "ls", n = 5)
```

```
  S   T
1 M car
2 F car
```

The Junction Tree Algorithm

1. **Moralise:** create the moral graph of the BN $\mathcal{B}$.
2. **Triangulate:** break every cycle spanning 4 or more nodes into sub-cycles of exactly 3 nodes by adding arcs to the moral graph, thus obtaining a triangulated graph.
3. **Cliques:** identify the cliques $C_1, \ldots, C_k$ of the triangulated graph, i.e., maximal subsets of nodes in which each element is adjacent to all the others.
4. **Junction Tree:** create a tree in which each clique is a node, and adjacent cliques are linked by arcs. The tree must satisfy the running intersection property: if a node belongs to two cliques C_i and C_j, it must be also included in all the cliques in the (unique) path that connects C_i and C_j.
5. **Parameters:** use the parameters of the local distributions of $\mathcal{B}$ to compute the parameter sets of the compound nodes of the junction tree.

Creating the Moral Graph

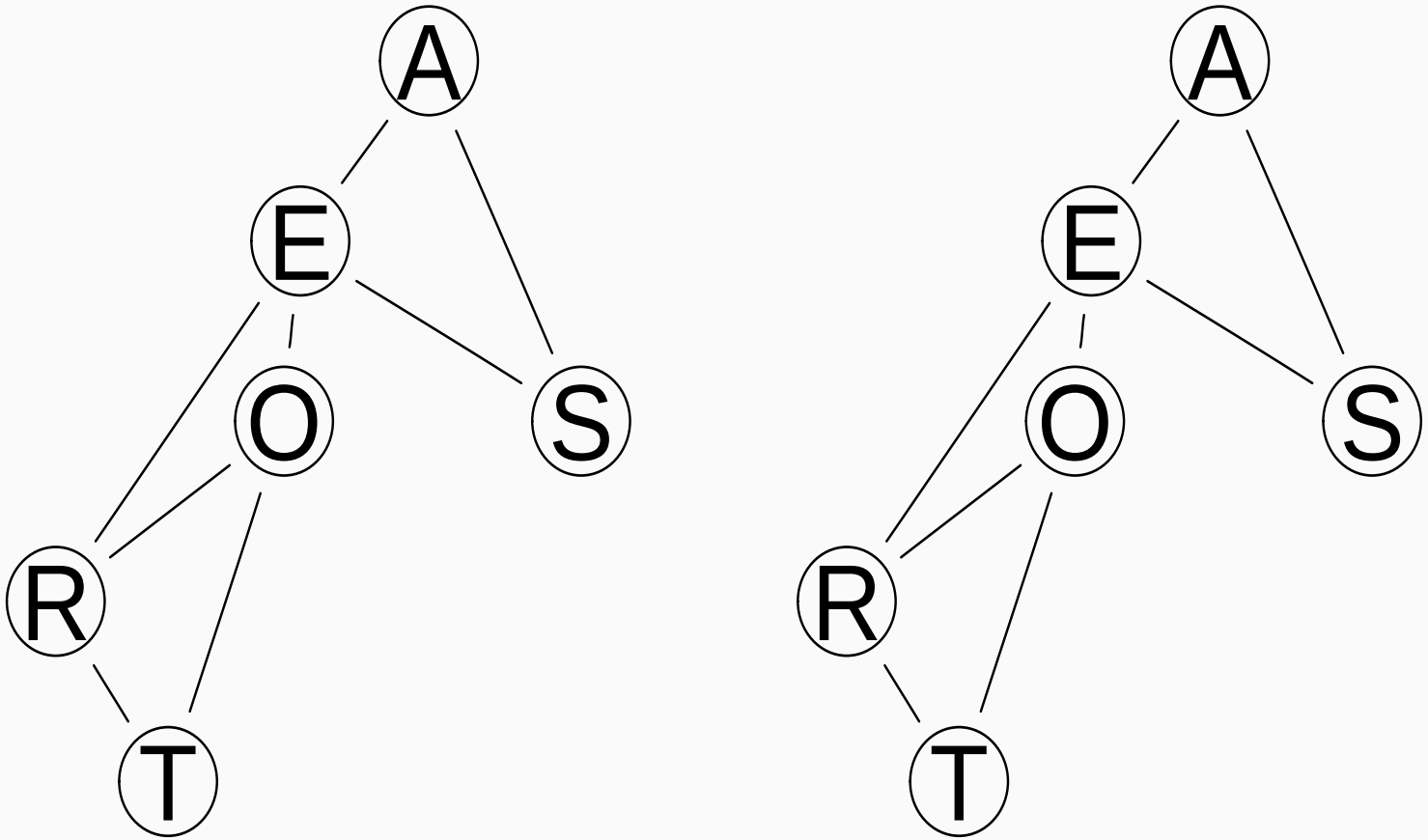
We saw how to create a moral graph earlier when introducing d-separation:

```
survey.dag = model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
survey.moral = moral(survey.dag)
```

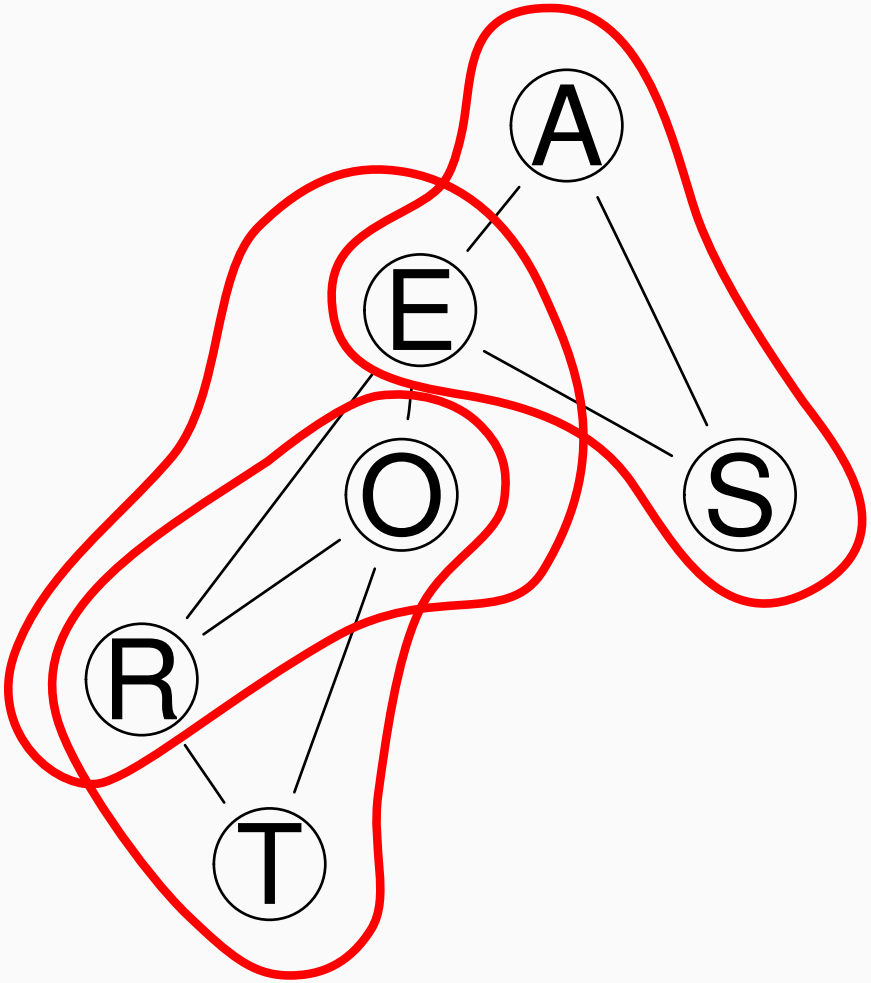
NOTE: different DAGs can express the same set of dependencies, and therefore will have the same moral graph. This in turn means exact inference by means of junction trees will return the same results for conditional probability and maximum a posteriori queries. They are probabilistically indistinguishable.

Different DAGs, Same Moral Graph

```
survey.dag1 = model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
survey.dag2 = model2network("[A|E][S|A:E][E|O:R][O|R:T][R|T][T]")
par(mfrow = c(1, 2))
graphviz.plot(moral(survey.dag1))
graphviz.plot(moral(survey.dag2))
```



Finding the Cliques



The moral graph is already triangulated, and we can see three cliques:

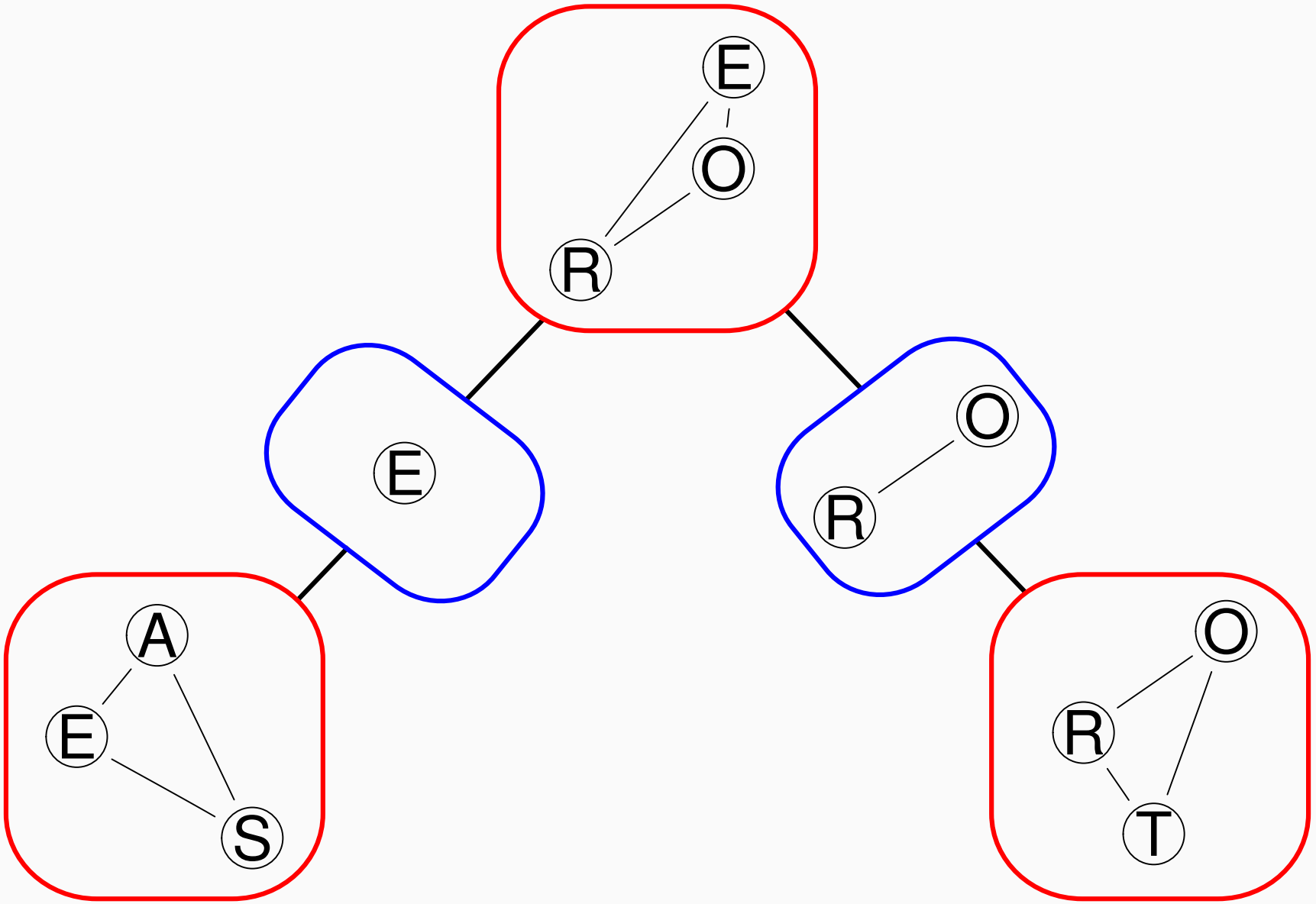
$$C_1 = \{A, E, S\}$$
$$C_2 = \{E, O, R\}$$
$$C_3 = \{O, R, T\}$$

with separators:

$$S_{12} = \{E\}$$
$$S_{23} = \{O, R\}$$

which we can use to build the junction tree.

Building the Junction Tree



In this example on the survey BN, the parameters for the cliques are:

$$\Theta_{C_1} = \mathrm{P}(A, E, S) = \mathrm{P}(A)\,\mathrm{P}(S)\,\mathrm{P}(E \mid A, S)$$
$$\Theta_{C_2} = \mathrm{P}(E, O, R) = \mathrm{P}(O \mid E)\,\mathrm{P}(R \mid E)\,\mathrm{P}(E)$$
$$\Theta_{C_3} = \mathrm{P}(O, R, T) = \mathrm{P}(T \mid O, R)\,\mathrm{P}(O), \mathrm{P}(R)$$

and those for the separators are:

$$\Theta_{S_{12}} = \mathrm{P}(E)$$
$$\Theta_{S_{23}} = \mathrm{P}(O, R)$$

All can be readily computed from the local distributions in the BN.

Estimating the Parameters

```
C1 = coef(bn$E)
for (a in A.lv)
  for (s in S.lv)
    C1[, a, s] = C1[, A = a, S = s] * coef(bn$A)[a] * coef(bn$S)[s]
C1
```

```
, , S = M

      A
E       young  adult    old
  high 0.1350 0.2160 0.1056
  uni  0.0450 0.0840 0.0144

, , S = F

      A
E       young  adult    old
  high 0.0768 0.1400 0.0720
  uni  0.0432 0.0600 0.0080
```

```
S12 = margin.table(C1, 1)
S12
```

```
E
 high   uni
0.745 0.255
```

Estimating the Parameters

```
C2 = array(0, dim = c(2, 2, 2), dimnames = list(O = O.lv, R = R.lv, E = E.lv))
for (o in O.lv)
  for (r in R.lv)
    for (e in E.lv)
      C2[o, r, e] = coef(bn$O)[o, e] * coef(bn$R)[r, e] * S12[e]
C2
```

```
, , E = high

      R
O        small    big
  emp  0.17890 0.5367
  self 0.00745 0.0224

, , E = uni

      R
O        small    big
  emp  0.04685 0.1874
  self 0.00407 0.0163
```

Estimating the Parameters

```
S23 = margin.table(C2, 1:2)
S23
```

```
        R
O         small    big
  emp   0.2257 0.7241
  self 0.0115 0.0387
```

```
C3 = coef(bn$T)
for (t in T.lv)
  for (o in O.lv)
    for (r in R.lv)
      C3[t, o, r] = C3[t, o, r] *
                    S23[o, r]
```

```
C3
```

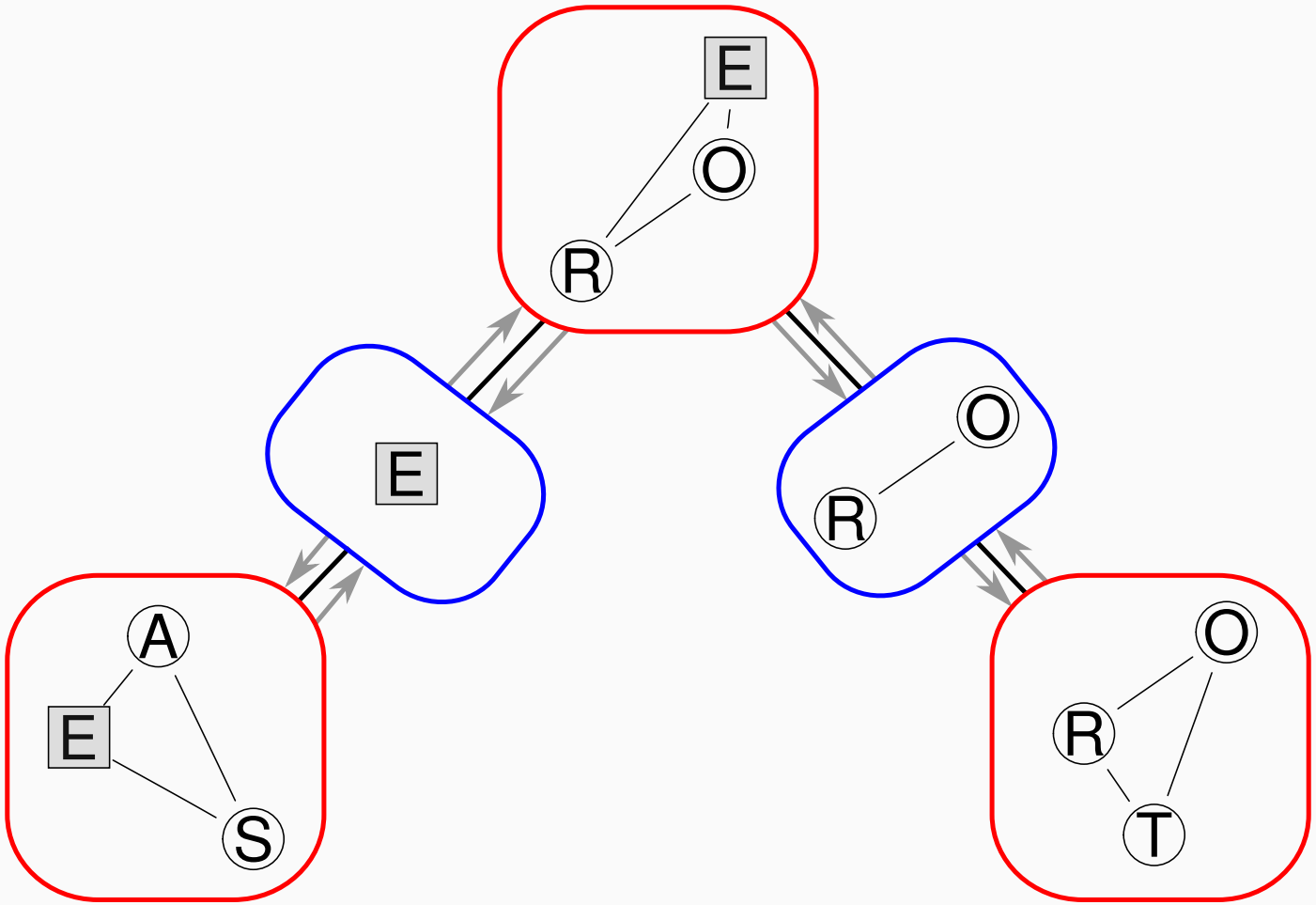
```
, , R = small

       O
T             emp     self
  car    0.108356 0.006455
  train 0.094812 0.004150
  other 0.022574 0.000922

, , R = big

       O
T             emp     self
  car    0.419963 0.027059
  train 0.173778 0.008118
  other 0.130333 0.003479
```

Belief Propagation and Message Passing



Say we set Education to “high school”; we can change it directly in S_{12}, but then we need to propagate the changes to C_1 and C_2; and from C_2 to S_{23} and to C_3. This is called belief propagation by message passing.

Belief Propagation and Message Passing

```
new.S12 = S12
new.S12["high"] = 1
new.S12["uni"] = 0
new.S12
```

```
 high  uni
    1    0
```

```
new.C1 = C1
for (e in E.lv)
  for (a in A.lv)
    for (s in S.lv)
      new.C1[e, a, s] =
        C1[e, a, s] / S12[e] *
                    new.S12[e]
```

```
new.C1
```

```
, , S = M

       A
E        young  adult    old
  high 0.1811 0.2898 0.1417
  uni  0.0000 0.0000 0.0000

, , S = F

       A
E        young  adult    old
  high 0.1030 0.1878 0.0966
  uni  0.0000 0.0000 0.0000
```

```
margin.table(new.C1, 1)
```

```
E
high  uni
   1    0
```

`margin.table(new.C1)` and `new.S12` match as expected.

Belief Propagation and Message Passing

```
new.C2 = C2
for (o in O.lv)
  for (r in R.lv)
    for (e in E.lv)
      new.C2[o, r, e] =
        C2[o, r, e] / S12[e] *
                    new.S12[e]
new.C2
```

```
, , E = high

      R
O       small  big
  emp    0.24 0.72
  self   0.01 0.03

, , E = uni

      R
O       small big
  emp       0   0
  self      0   0
```

```
new.S23 = margin.table(new.C2, 1:2)
new.S23
```

```
        R
 O       small  big
   emp    0.24 0.72
   self   0.01 0.03
```

```
new.C3 = C3
for (t in T.lv)
  for (o in O.lv)
    for (r in R.lv)
      new.C3[t, o, r] =
        C3[t, o, r] / S23[o, r] *
                    new.S23[o, r]
```

Which completes the first iteration of belief propagation.

In more complex graphs and more complex queries we may need more than one iteration, but for this relatively simple network the belief propagation is complete.

Computing $\mathrm{P}(\mathtt{S} = \mathtt{M}, \mathtt{T} = \mathtt{car})$ at this point can be done easily by:

```
T = margin.table(new.C3, 1)
S = margin.table(new.C1, 3)
as.numeric(S["M"] * T["car"])
```

```
[1] 0.343
```

because Sex and Transportation are in different cliques and are separated by Education, and therefore independent.

gRain: Exact Inference with Junction Trees

Junction trees and belief propagation are implemented in the **gRain** package. In order to answer our query, we convert the BN from **bnlearn** to its equivalent in **gRain** with `as.grain()` and we construct the junction tree with `compile()`.

```
library(gRain)
junction = compile(as.grain(bn))
```

Then we set the evidence on the node, fixing it to "high school" with probability 1 with `setEvidence()`.

```
jedu = setEvidence(junction, nodes = "E", states = "high")
```

And after that, we can perform our conditional probability query with `querygrain()`, which also takes care of the belief propagation.

```
SxT.cpt = querygrain(jedu, nodes = c("S", "T"), type = "joint")
```

Joint and Marginal Conditional Probabilities

The result of our query is the joint distribution of Sex and Travel given that Education is “high school”.

SxT.cpt

```
    T
S      car train  other
  M 0.343 0.174 0.0962
  F 0.217 0.110 0.0609
```

Similarly, we can use `querygrain()` compute the marginal distributions of Sex and Travel conditional on Education.

```
querygrain(jedu, nodes = c("S", "T"), type = "marginal")
```

```
$S
S
    M     F
0.613 0.387

$T
T
  car train other
0.559 0.283 0.157
```

Interestingly, we can also compute the conditional distribution of Sex given Travel (still conditioning on Education being “high school”), which turns out to be:

```
querygrain(jedu, nodes = c("S", "T"), type = "conditional")
```

```
    T
S      car train other
  M 0.613 0.613 0.613
  F 0.387 0.387 0.387
```

This makes sense in the light of d-separation, which implies conditional independence.

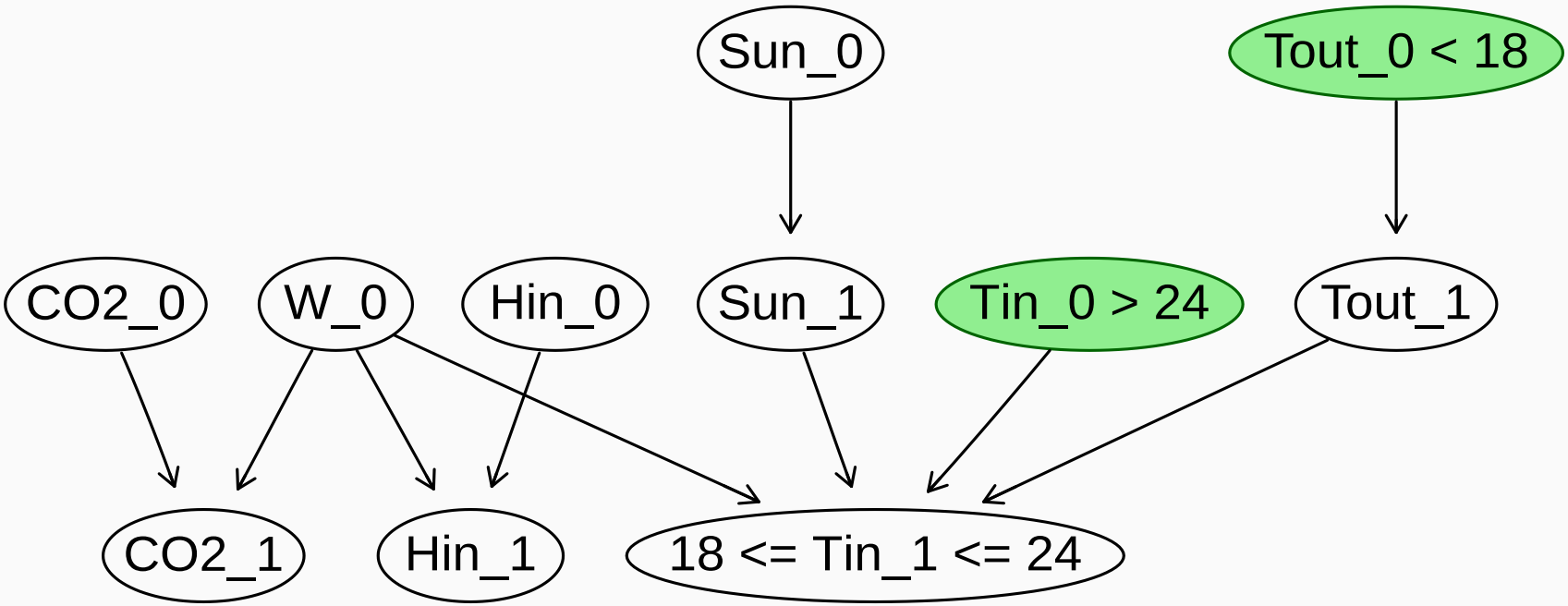
```
dsep(bn, x = "S", y = "T", z = "E")
```

```
[1] TRUE
```

A Domotics Example

Consider one of the queries we used to assess the dynamic BN:

- the evidence is that, at time zero, the inside temperature (`Tin_0`) is higher than 24 degrees and the outside temperature (`Tout_0`) is lower than 18 degrees;
- the event is that the inside temperature at time 1 (`Tin_1`) is between 18 and 24 degrees.



Generating Random Observations from the BN

Logic sampling involves generating random observations from the BN, to simulate a real-world experiment in which we would collect new data. How is it done?

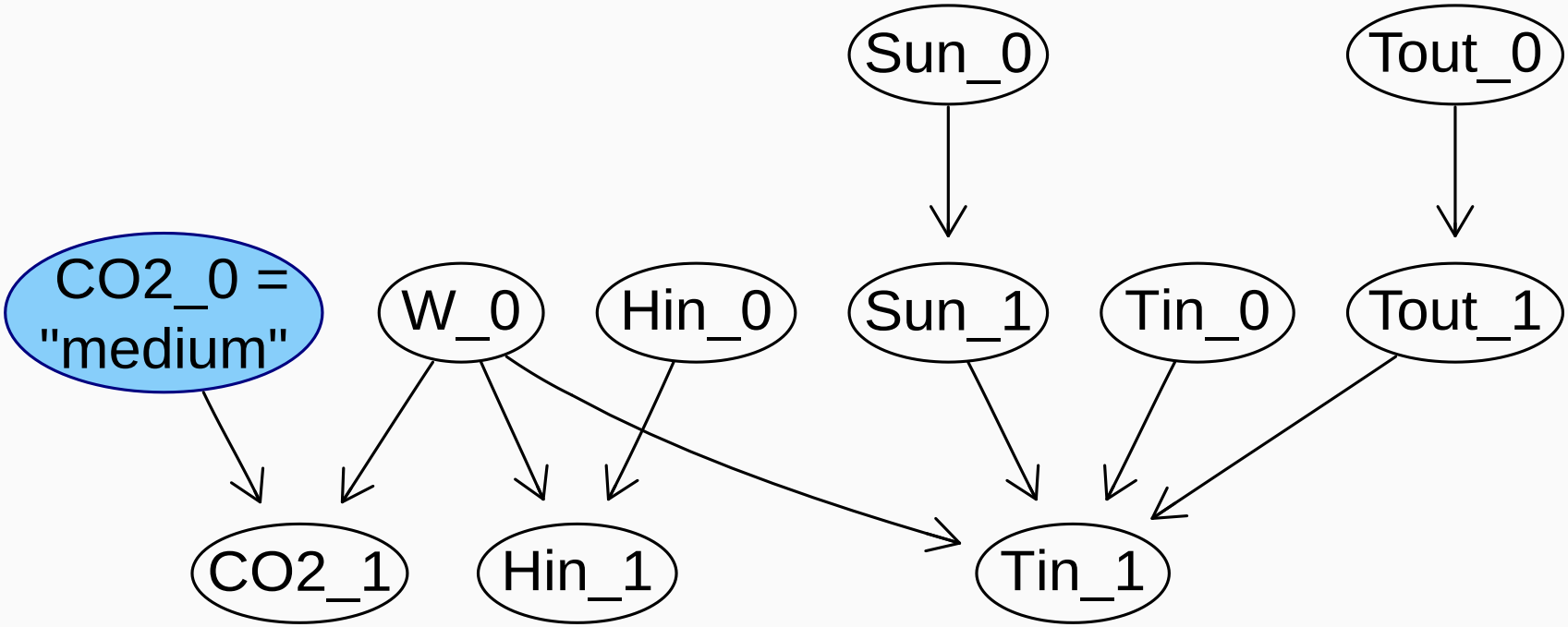
We start from a completely empty set of values for the variables.

CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
?	?	?	?	?	?

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

Then we start generating values for the variables starting from the top of the BN and going down.

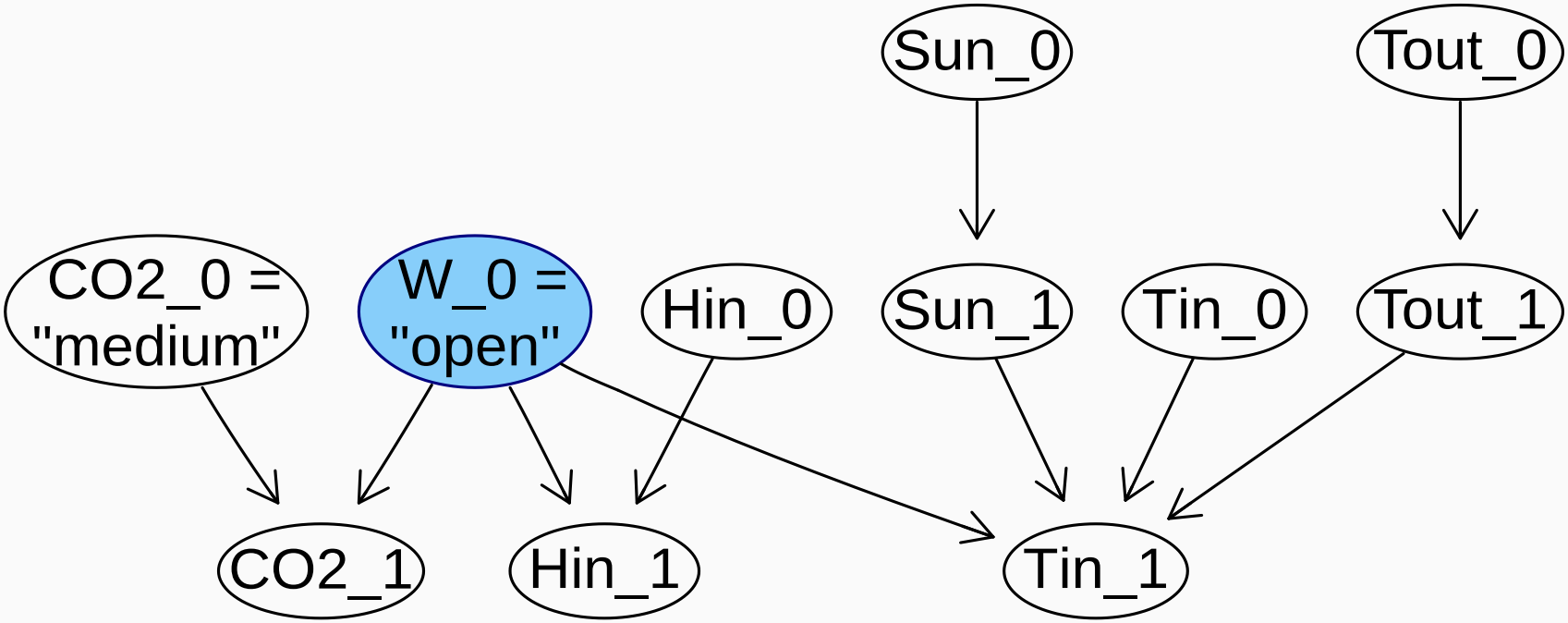
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	?	?	?	?	?

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

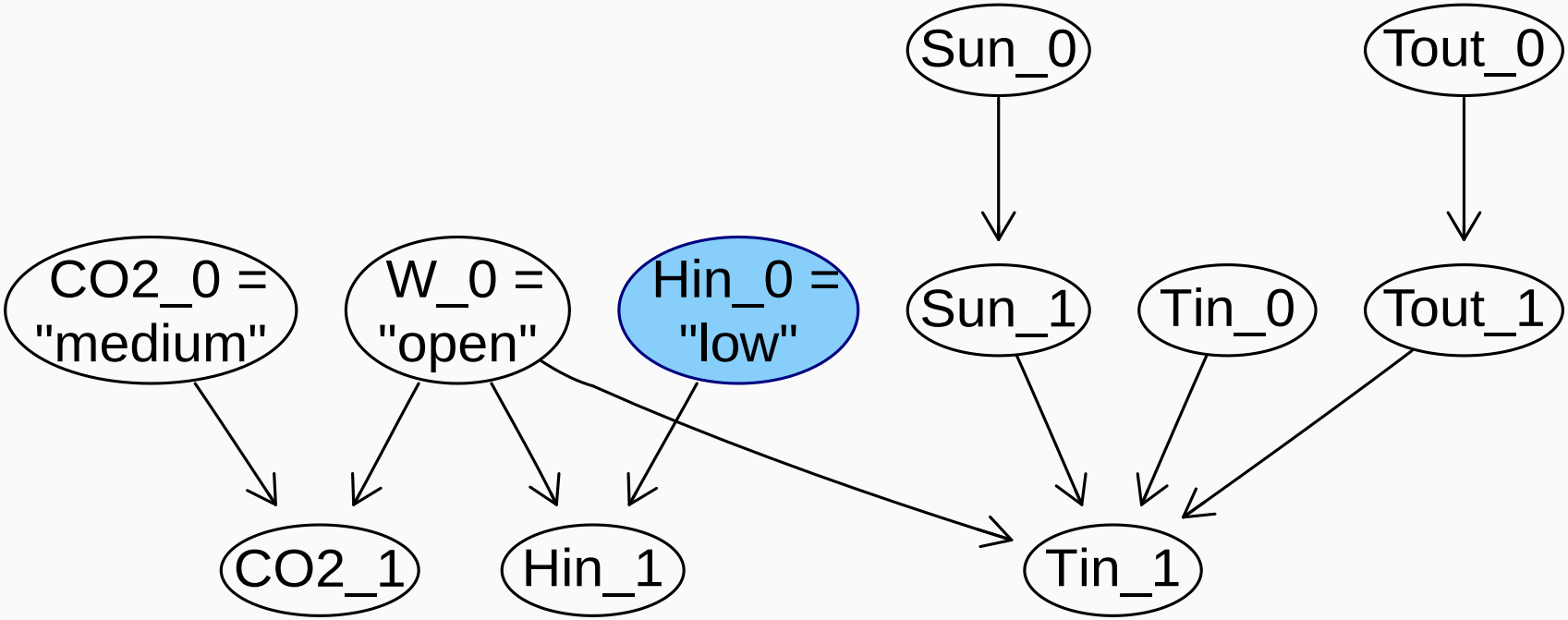
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	?	?	?	?

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

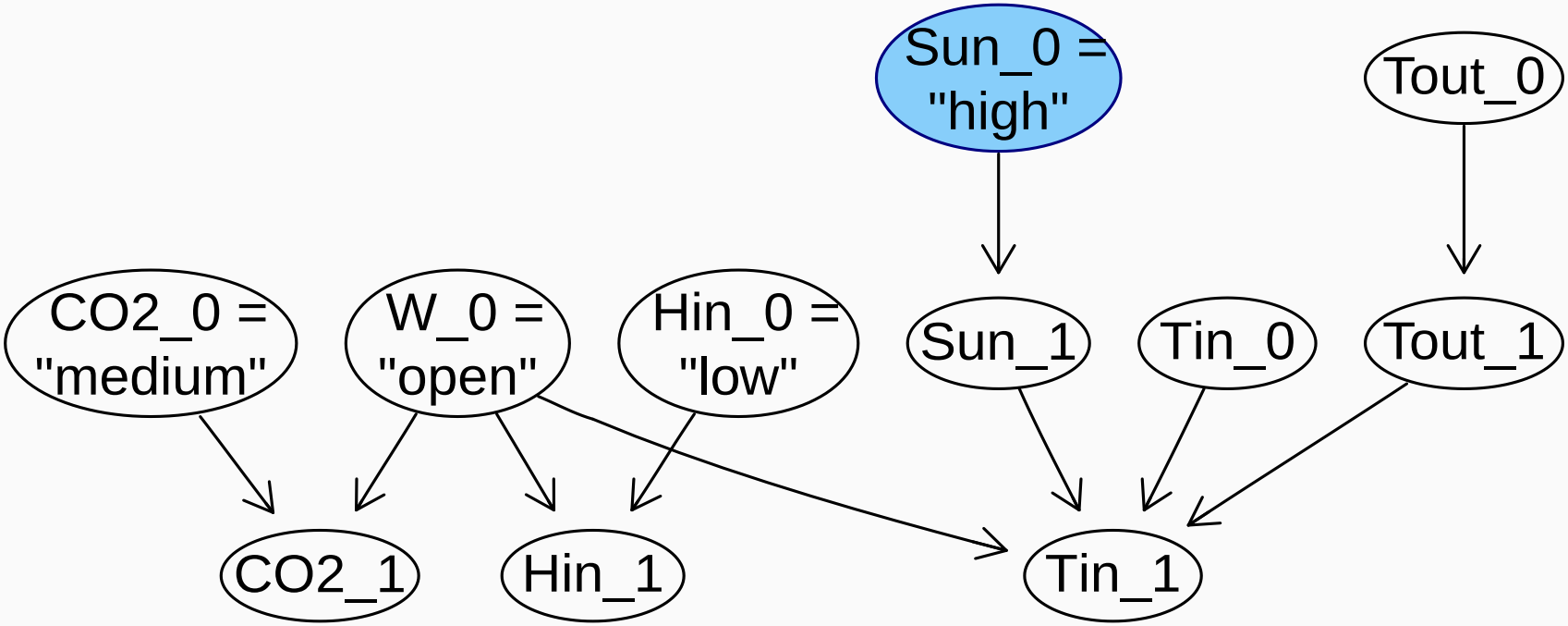
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	?	?	?

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

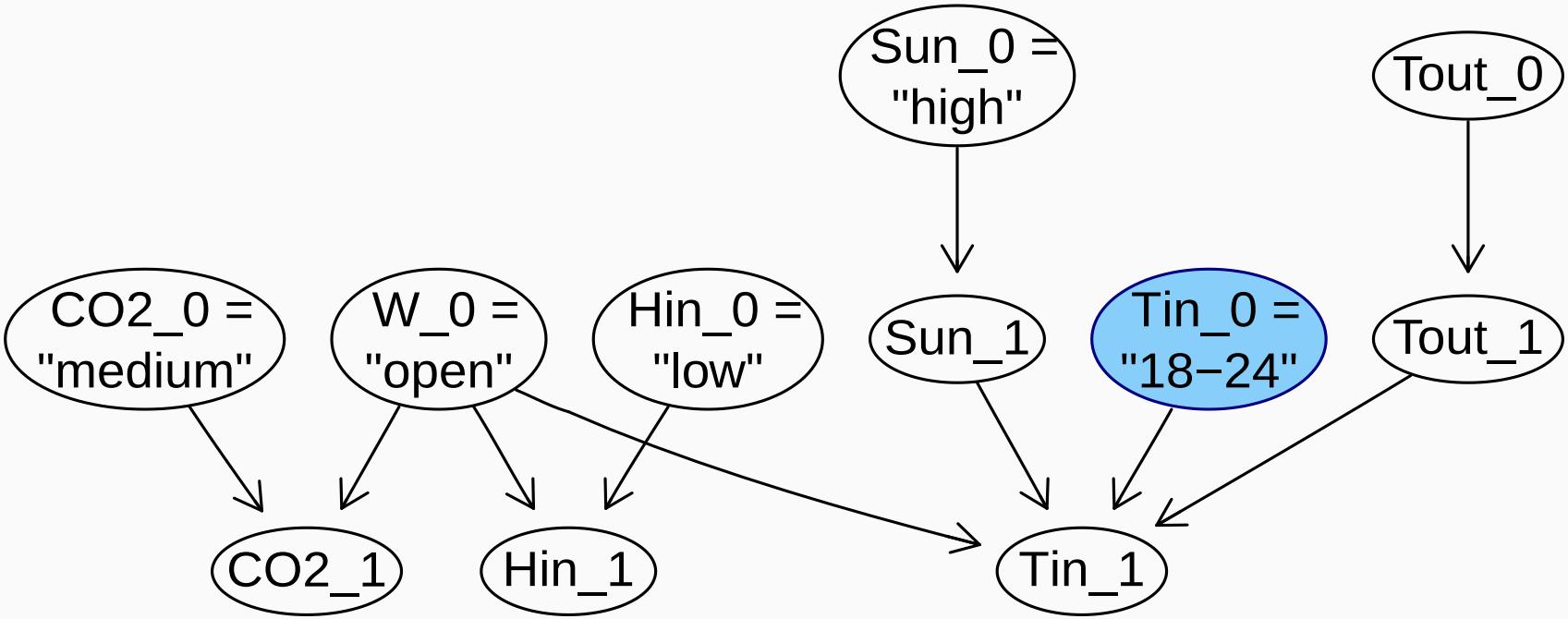
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	?	?

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

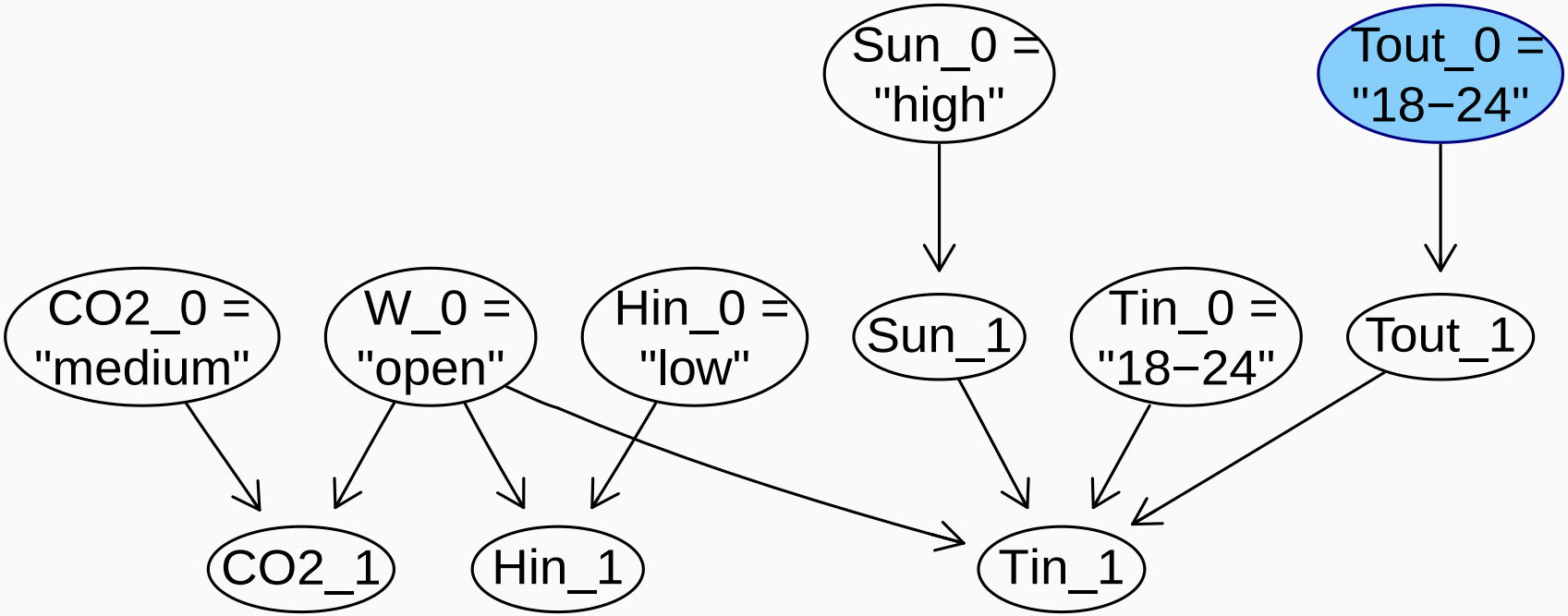
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	?

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

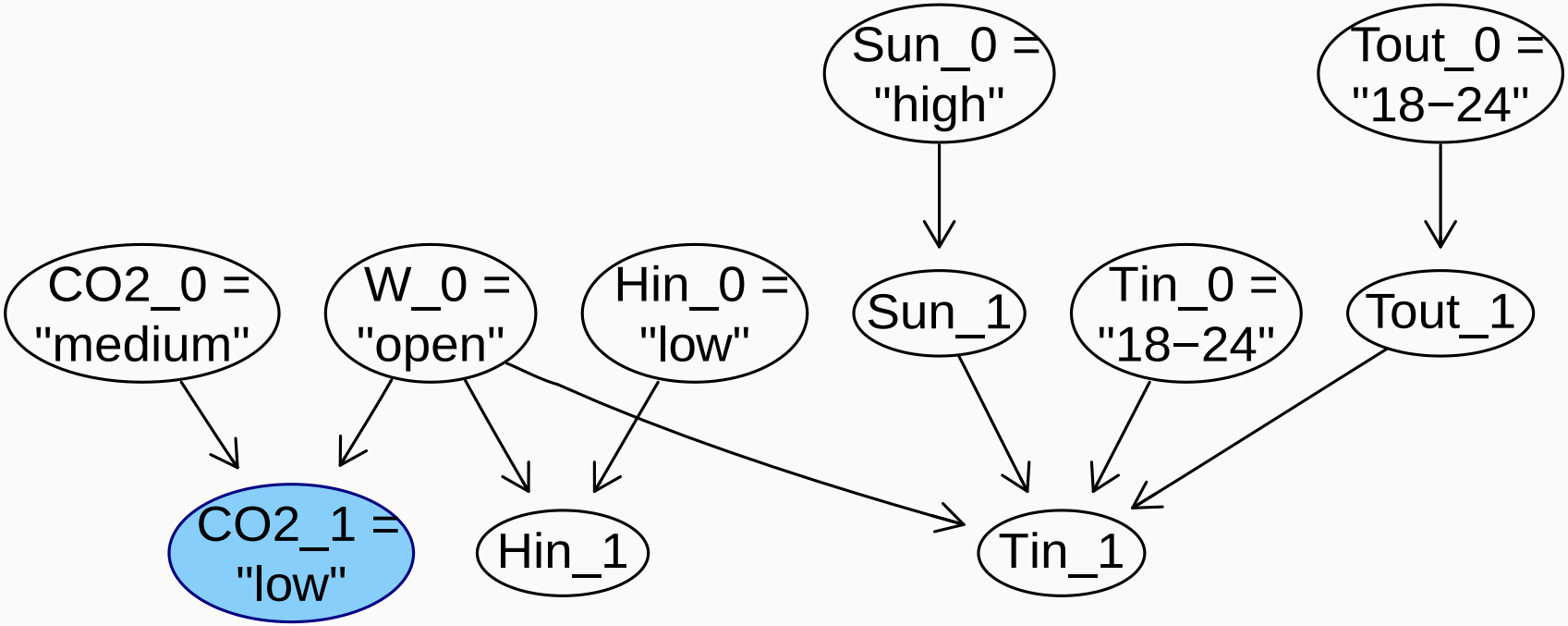
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	18-24

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
?	?	?	?	?

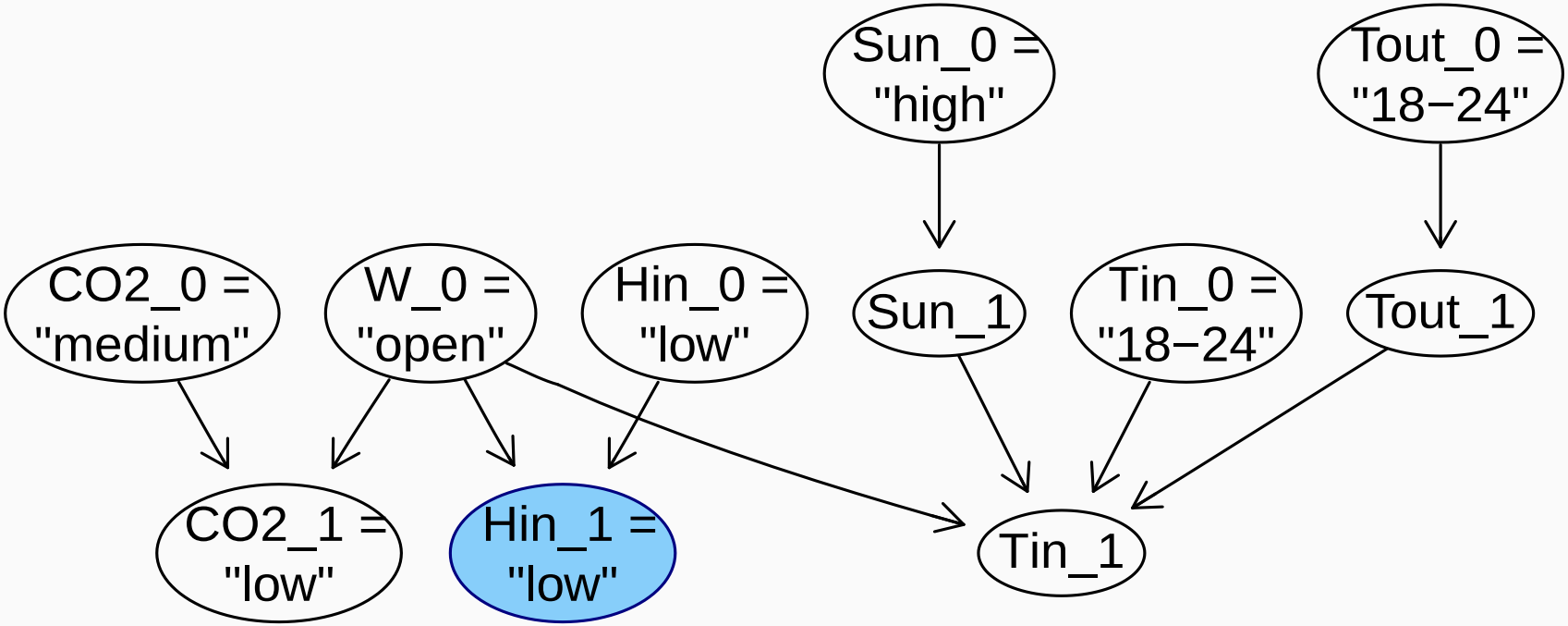
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	18-24

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
low	?	?	?	?

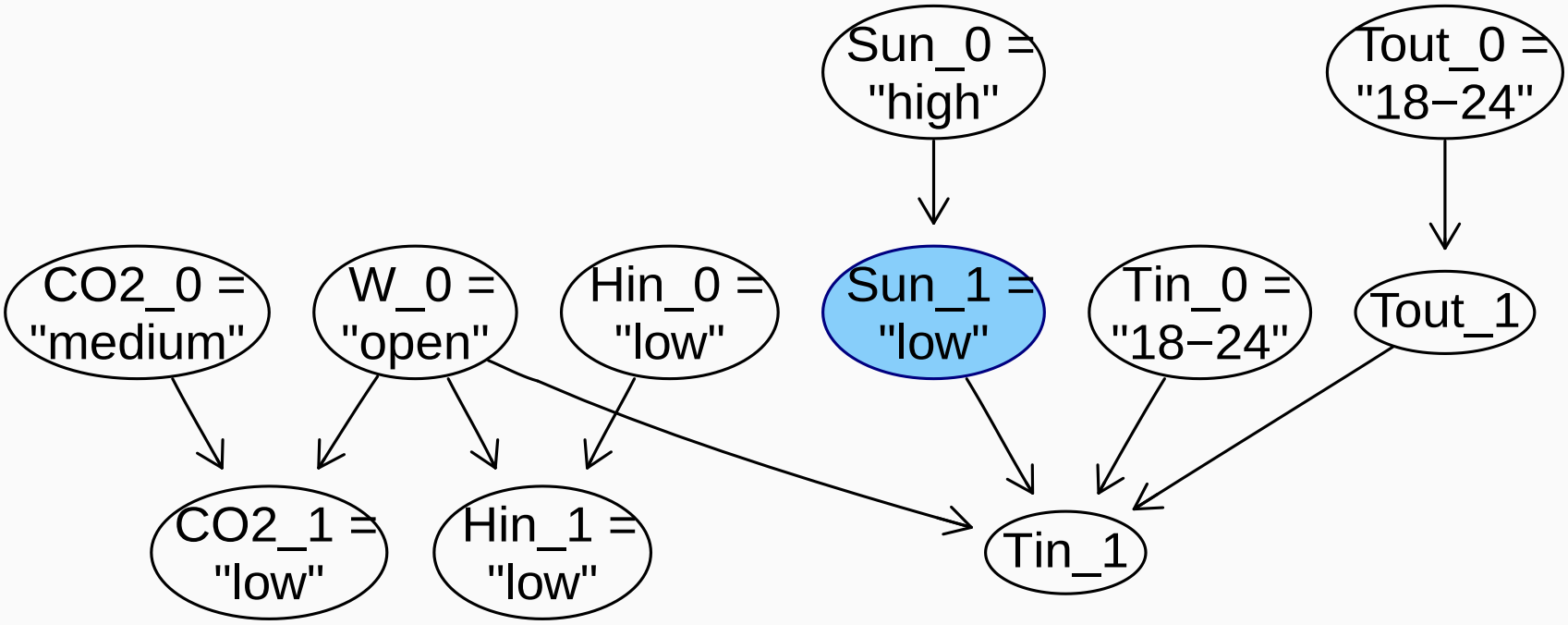
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	18-24

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
low	low	?	?	?

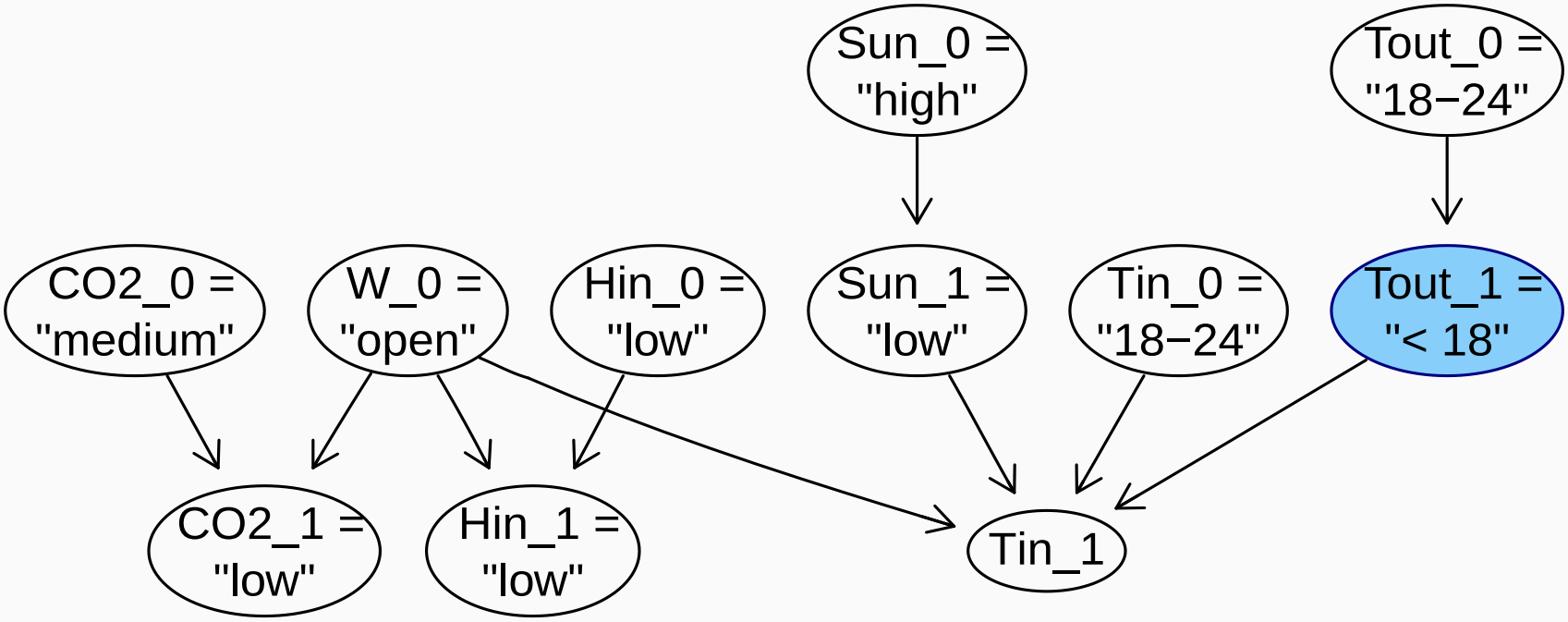
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	18-24

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
low	low	low	?	?

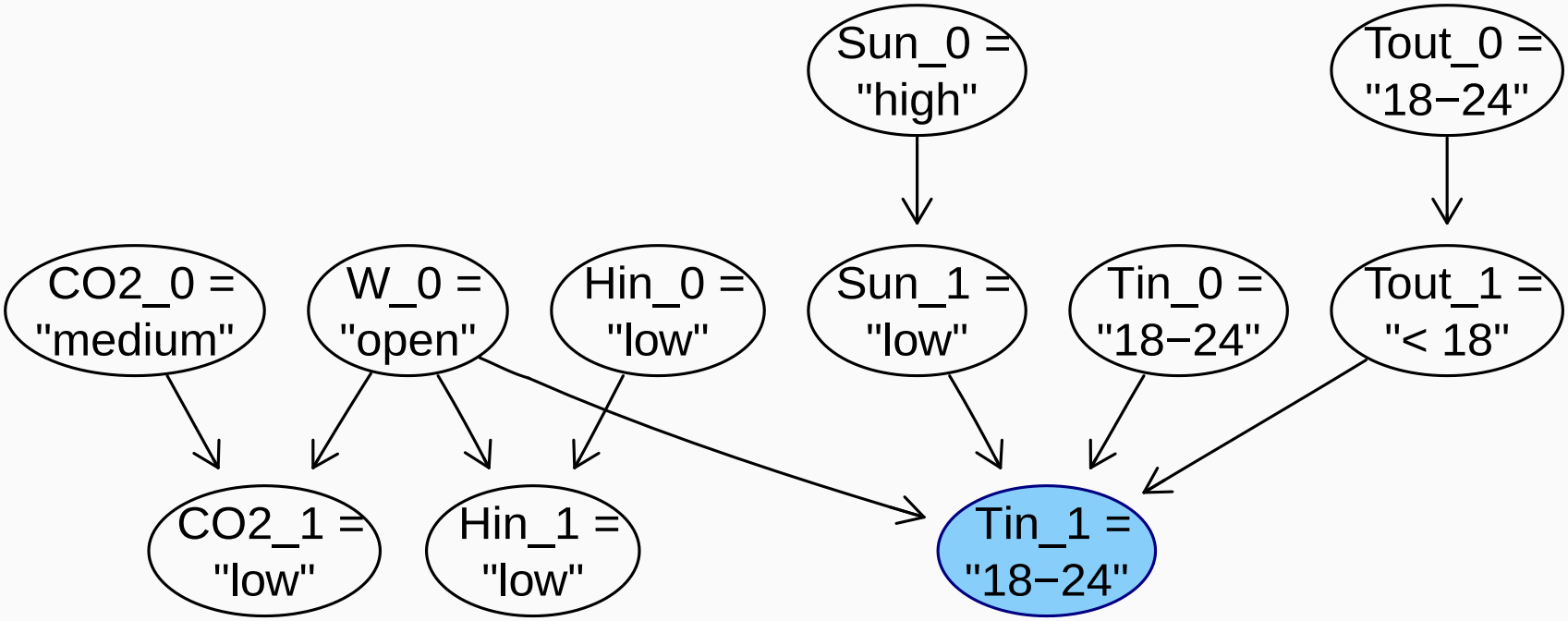
Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	18-24

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
low	low	low	?	< 18

Generating Random Observations from the BN



CO2_0	W_0	Hin_0	Sun_0	Tin_0	Tout_0
medium	open	low	high	18-24	18-24

CO2_1	Hin_1	Sun_1	Tin_1	Tout_1
low	low	low	18-24	< 18

This process goes under the name of random sampling, and it can be conditional or unconditional.

`rbn()` repeats the steps in the previous slides for each random sample it generates from the BN. It performs unconditional sampling, in the sense that we use the probabilities in the BN without fixing the values of any of the nodes.

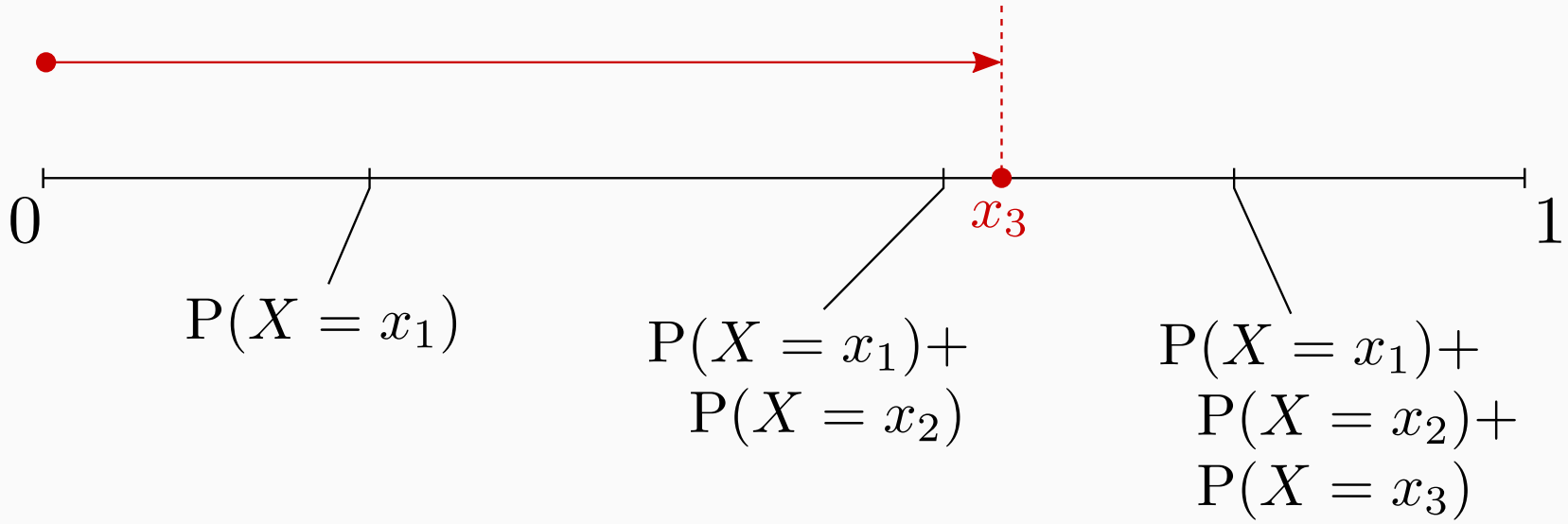
For logic sampling, `cpdist()` generates random samples with `rbn()` and then discards those in which the variables in the evidence do not take the values they are expected to. Hence it performs conditional sampling, in the sense that it only returns a sample conditional on the fact that the evidence is observed.

`cpquery()` then takes the samples returned by `cpdist()` and computes the conditional probability for the query by the relative frequency of the event we are looking for.

An Aside: Random Sampling from a Discrete Variable

How do we sample the values of a discrete variable at random?

1. generate a random number between 0 and 1;
2. sort the values of the variables;
3. for each value of the variable, in order:
   - 3.1 if the random number is smaller than the probability of this value and of those that precede it, choose it;
   - 3.2 else move to the next value.



Stepping Through Logic Sampling

Computing the conditional probability with logic sampling involves the following steps:

1. generate the random samples;

```
particles = rbn(win.dbn, 10^6)
head(particles, n = 5)
```

```
    CO2_0  CO2_1 Hin_0 Hin_1 Sun_0 Sun_1 Tin_0 Tin_1 Tout_0 Tout_1    W_0
1 medium    low   low   low   low  high 18-24 18-24  18-24  18-24   open
2    low    low  high  high   low   low 18-24 18-24  18-24    >24   open
3    low    low  high   low   low   low 18-24 18-24  18-24  18-24   open
4   high   high   low  high  high  high 18-24 18-24  18-24  18-24 closed
5 medium medium   low   low   low  high   <18 18-24  18-24    >24   open
```

2. count how many match the evidence;

```
partE = particles[(particles[, "Tin_0"] == ">24") &
                  (particles[, "Tout_0"] == "<18"), ]
nE = nrow(partE)
nE
```

```
[1] 9821
```

Stepping Through Logic Sampling

3. count how many also match the event;

```
partEQ = partE[(partE[, "Tin_1"] == "18-24"), ]
nEQ = nrow(partEQ)
nEQ
```

```
[1] 3932
```

4. estimate the conditional probability.

```
nEQ/nE
```

```
[1] 0.4
```

The estimated conditional probability of 0.4 is about the same as that returned by `cpquery()`; it is not identical because `cpquery()` generates a second set of random samples, so the frequencies nE and nEQ will not be exactly the same as before.

```
cpquery(win.dbn, event = (Tin_1 == "18-24"),
  evidence = (Tin_0 == ">24") & (Tout_0 == "<18"), n = 10^6)
```

```
[1] 0.406
```

Even though we generated 10^6 random samples, we use only the 9821 samples that match the evidence to compute the conditional probability. If the evidence is a rare event, that is a problem!

Stepping Through Likelihood Weighting

Computing the conditional probability with likelihood weighting involves the following steps:

1. mutilate the BN;

```
mutbn = mutilated(win.dbn, list(Tin_0 = ">24", Tout_0 = "<18"))
mutbn$Tin_0
```

```
  Parameters of node Tin_0 (multinomial distribution)

Conditional probability table:
   <18 18-24   >24
    0     0     1
```

```
mutbn$Tout_0
```

```
  Parameters of node Tout_0 (multinomial distribution)

Conditional probability table:
   <18 18-24   >24
    1     0     0
```

Stepping Through Likelihood Weighting

2. generate the random samples;

```
particles = rbn(mutbn, 10^6)
```

3. compute the weights for the random samples;

```
w = logLik(win.dbn, particles, nodes = c("Tin_0", "Tout_0"), by.sample = TRUE)
```

4. sum up the weights of the random samples that match the evidence, and of all the random samples;

```
wEQ = sum(exp(w[particles[, "Tin_1"] == "18-24"]))
wE = sum(exp(w))
```

5. estimate the conditional probability.

```
wEQ/wE
```

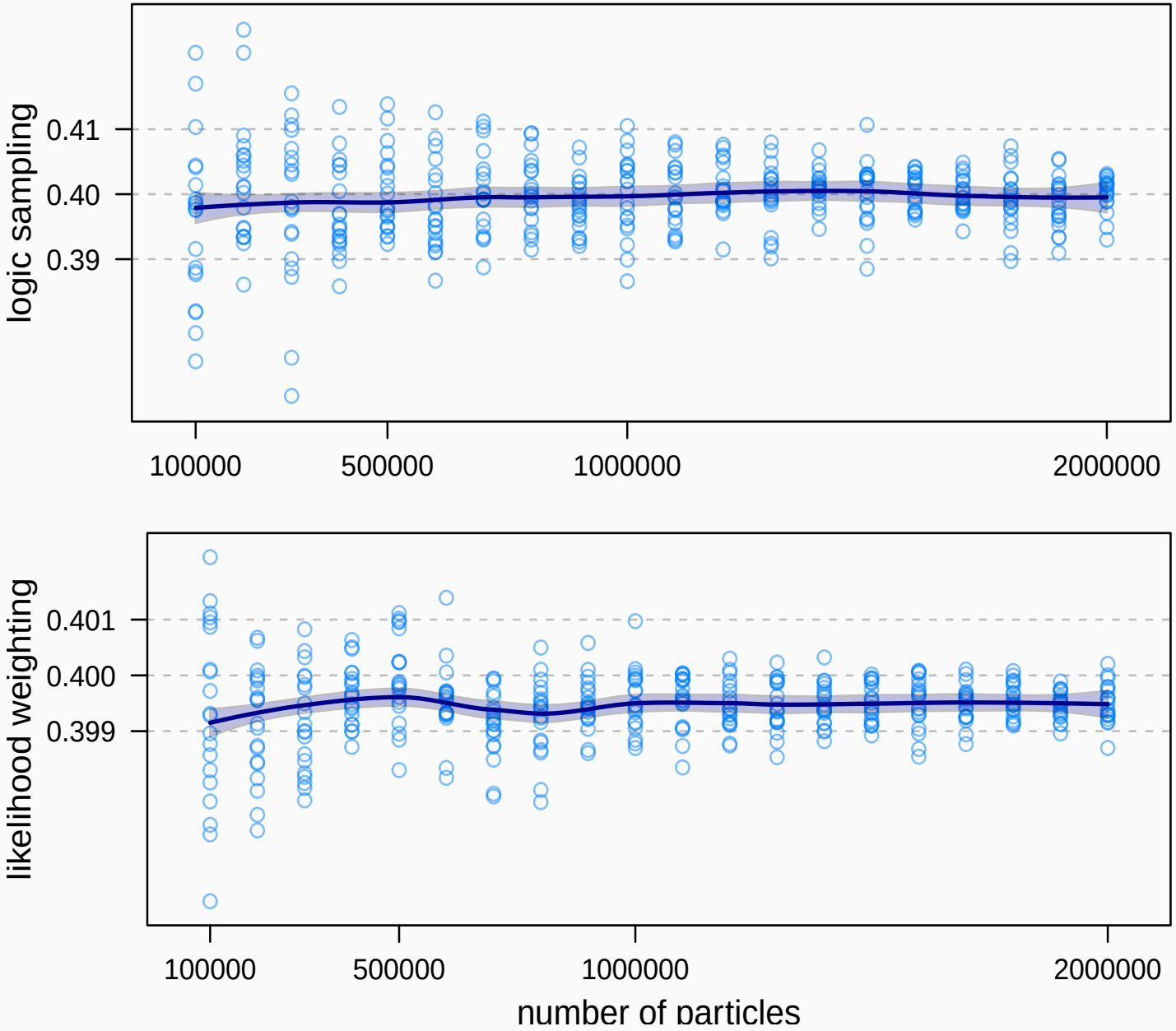
```
[1] 0.399
```

The estimated probability is again very close to that returned by `cpquery()`.

```
cpquery(win.dbn, event = (Tin_1 == "18-24"),
          evidence = list(Tin_0 = ">24", Tout_0 = "<18"),
          n = 10^6, method = "lw")
```

```
[1] 0.399
```

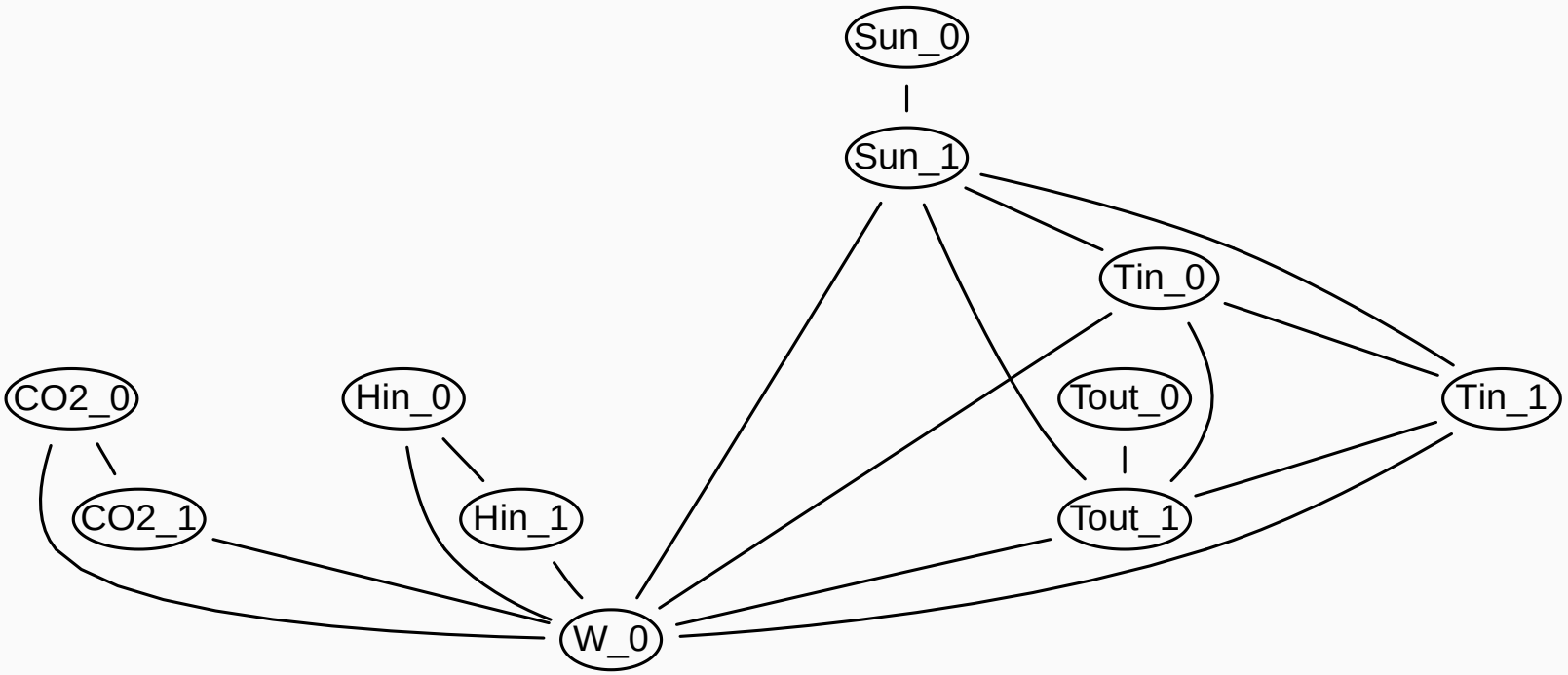
For Different Numbers of Random Samples?



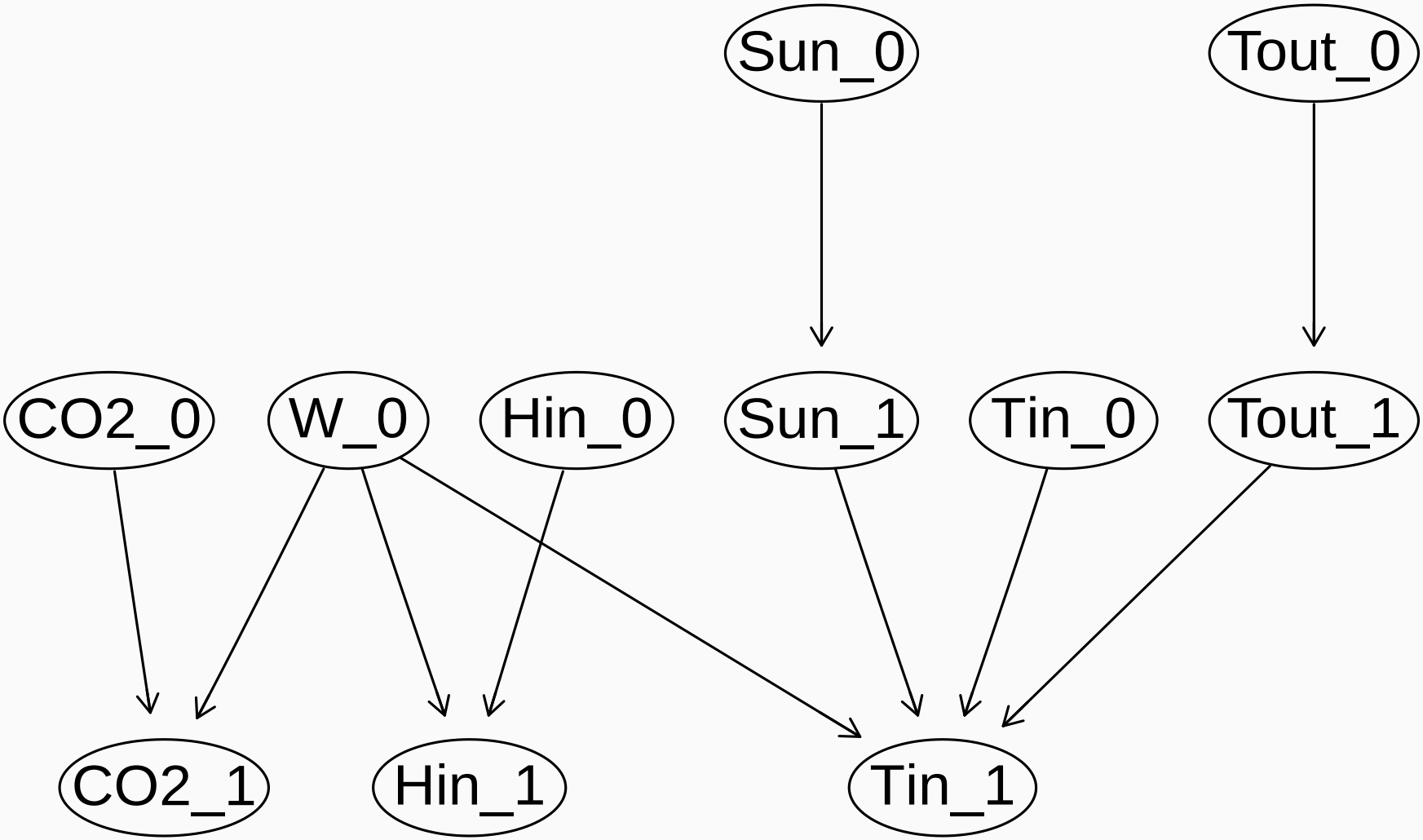
The Junction Tree Algorithm

The first step in the junction tree algorithm is to construct the moral graph from the BN by removing arc directions and joining parents that share a common child.

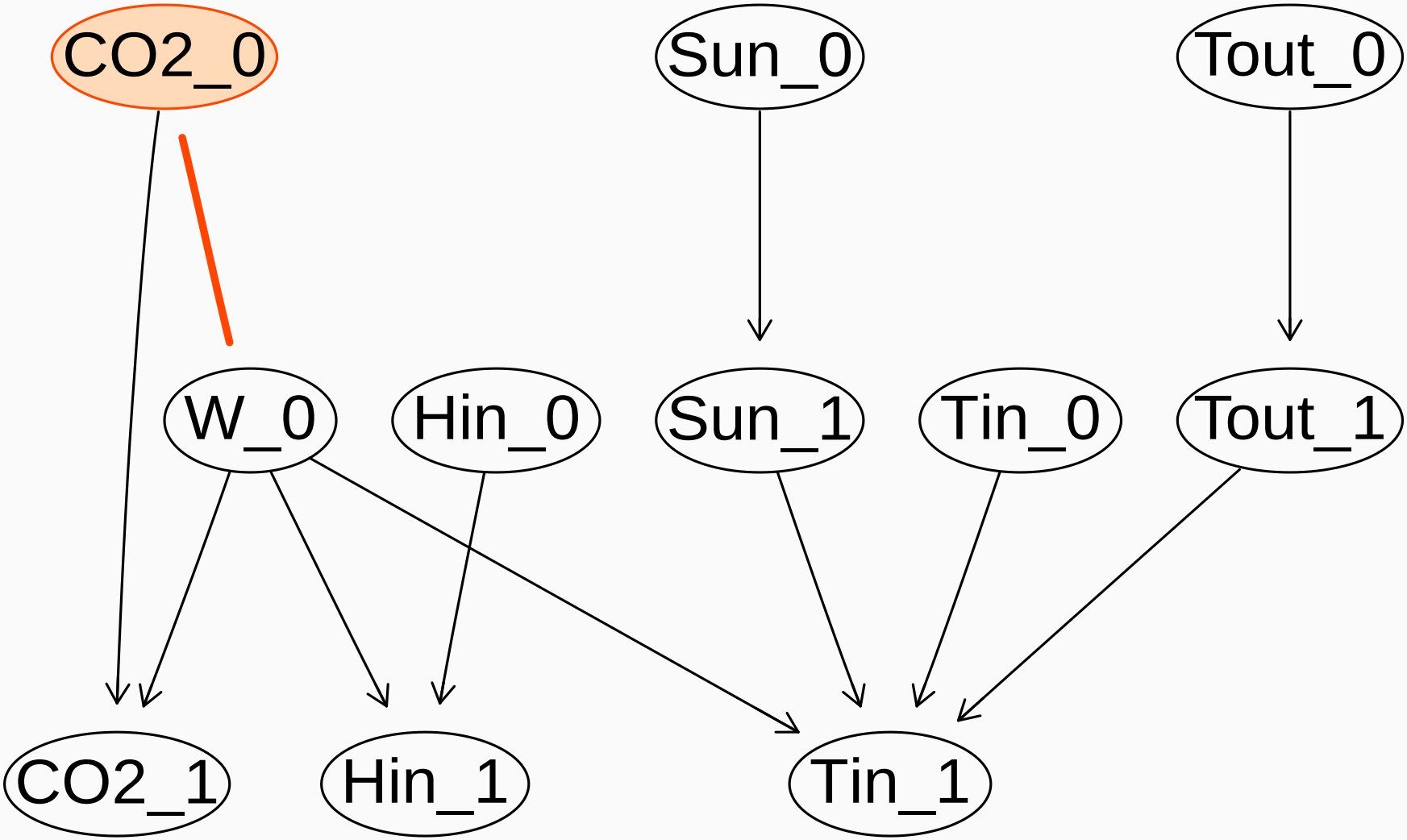
```
moral.dyn = moral(win.dyn)
graphviz.plot(moral.dyn, shape = "ellipse")
```



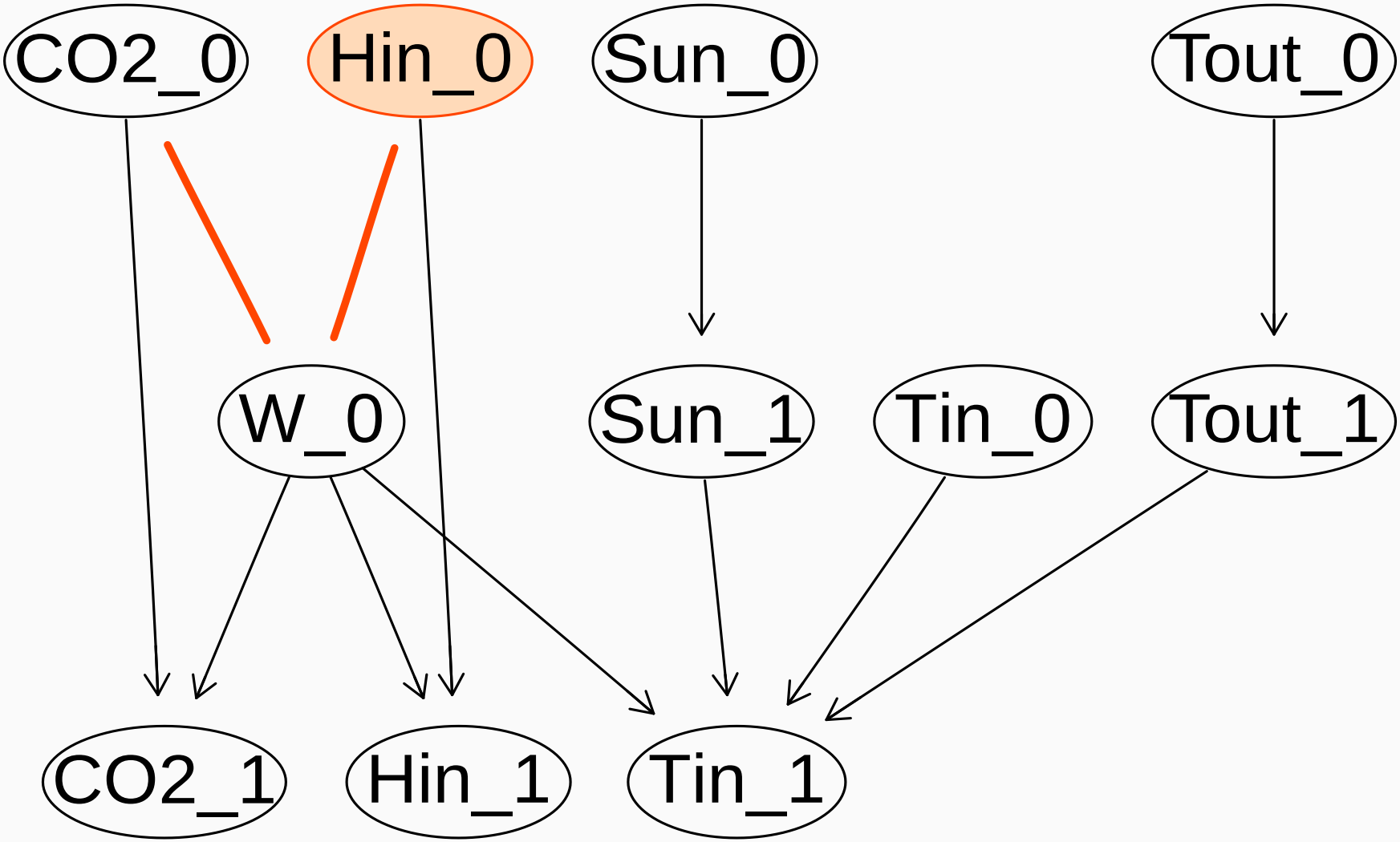
Constructing the Moral Graph



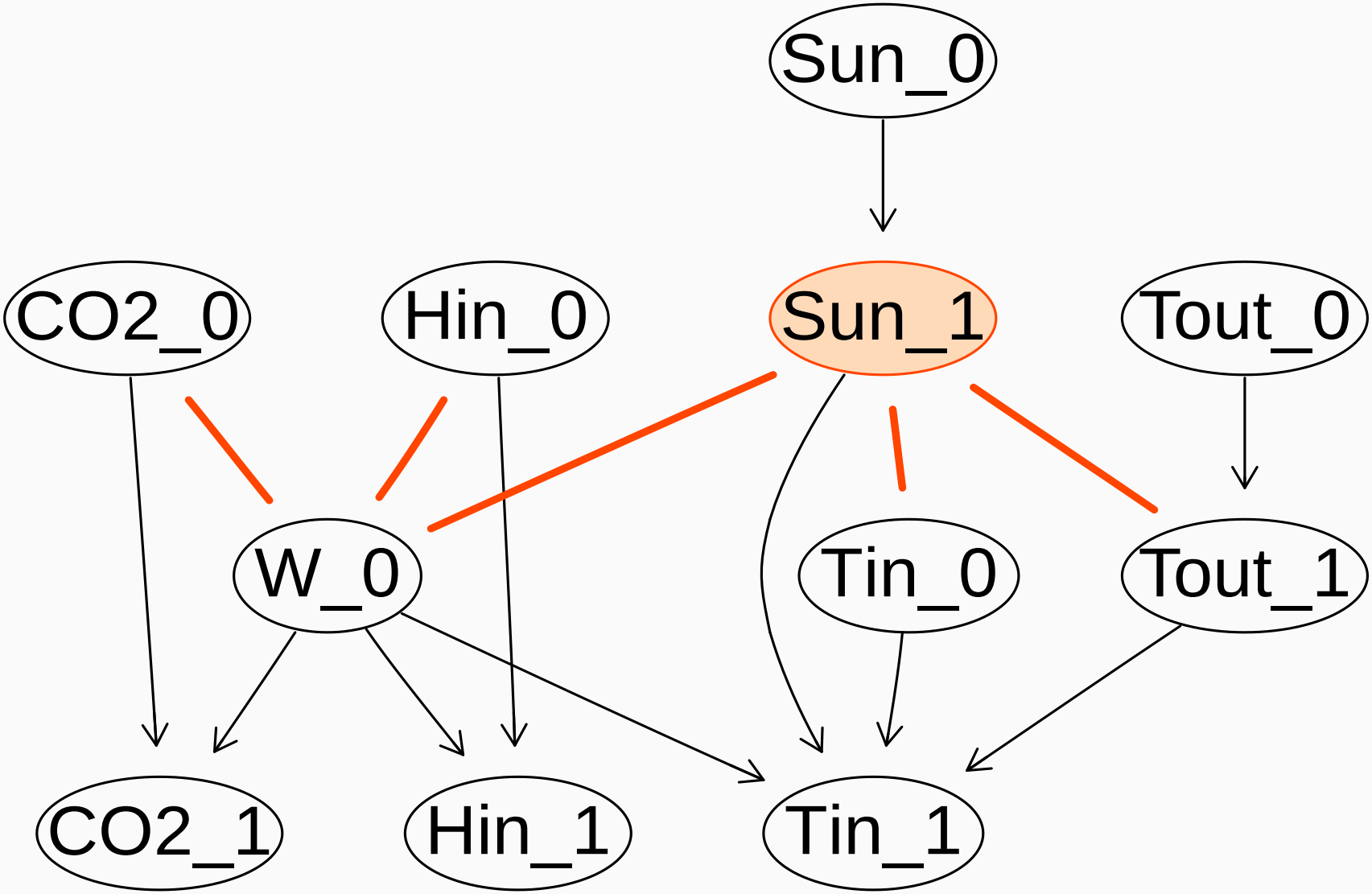
Constructing the Moral Graph



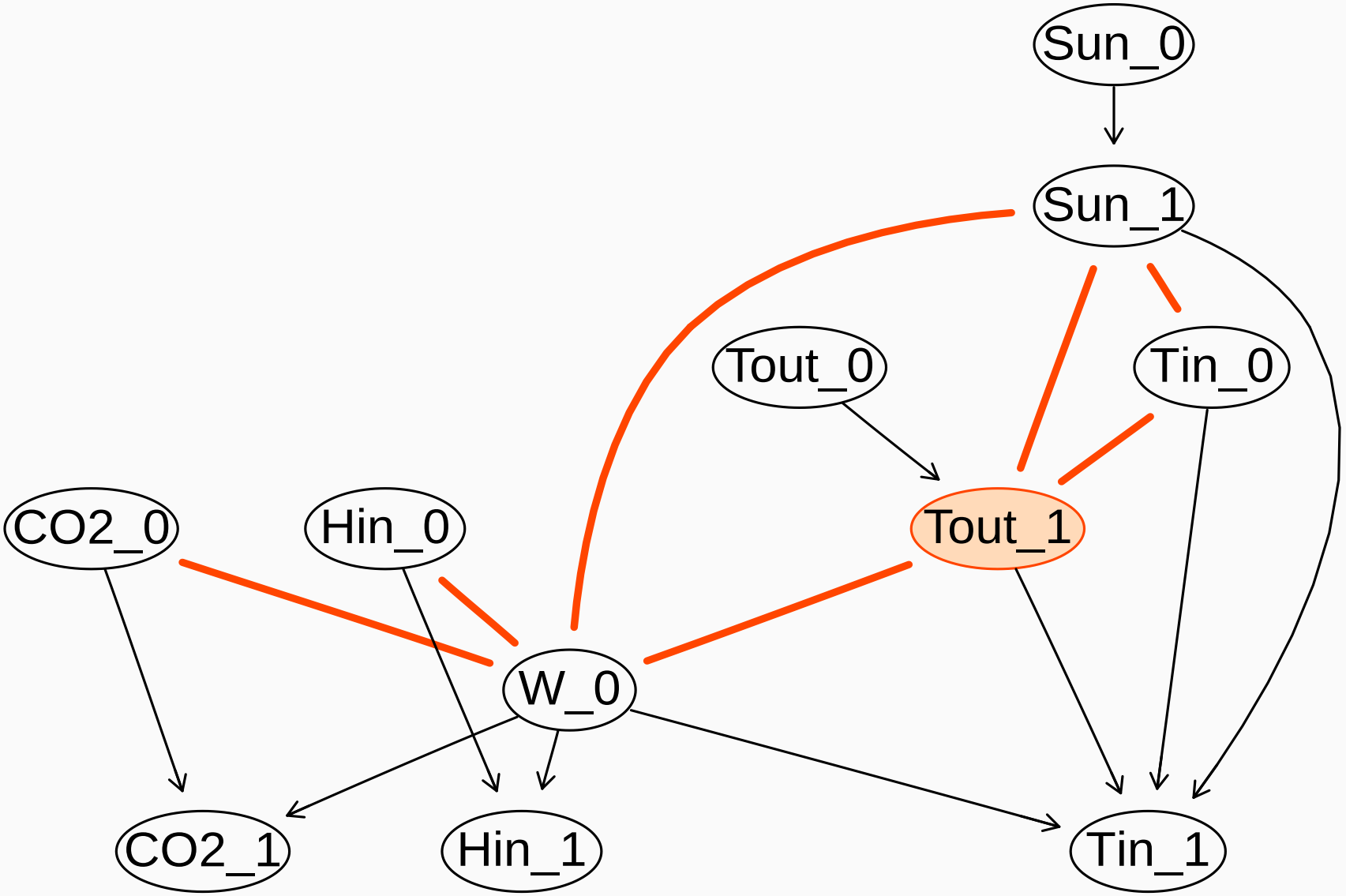
Constructing the Moral Graph



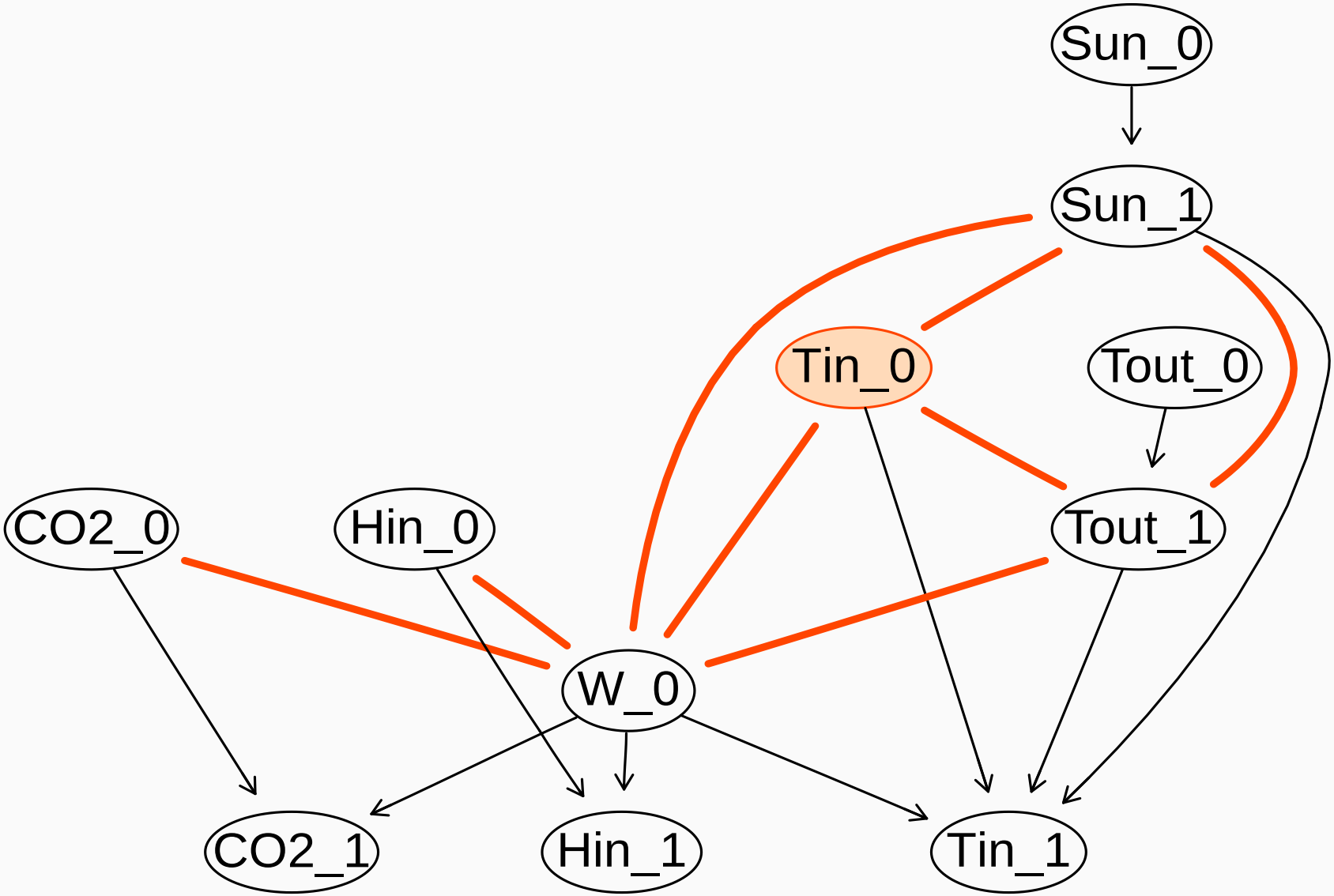
Constructing the Moral Graph



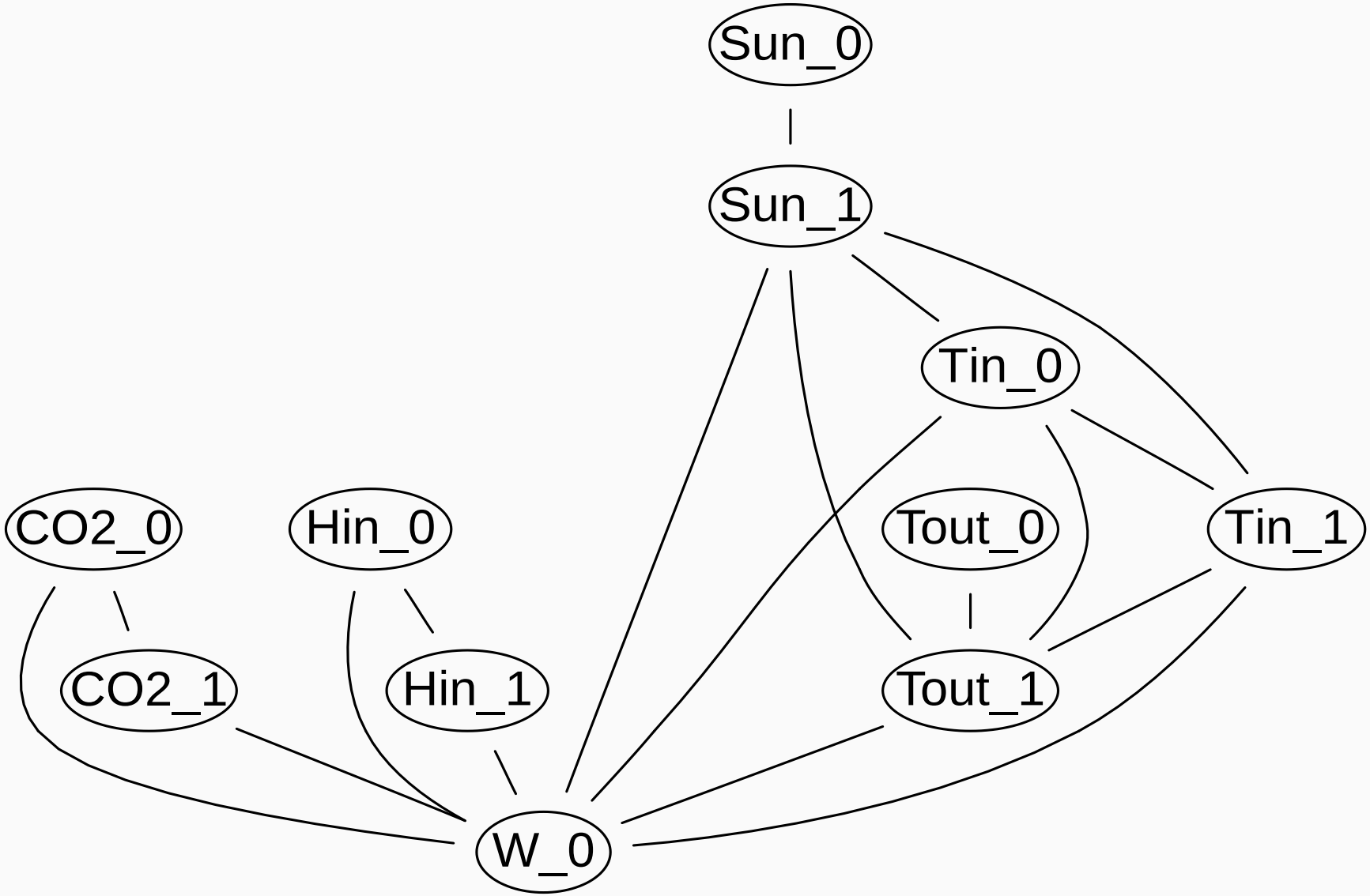
Constructing the Moral Graph



Constructing the Moral Graph

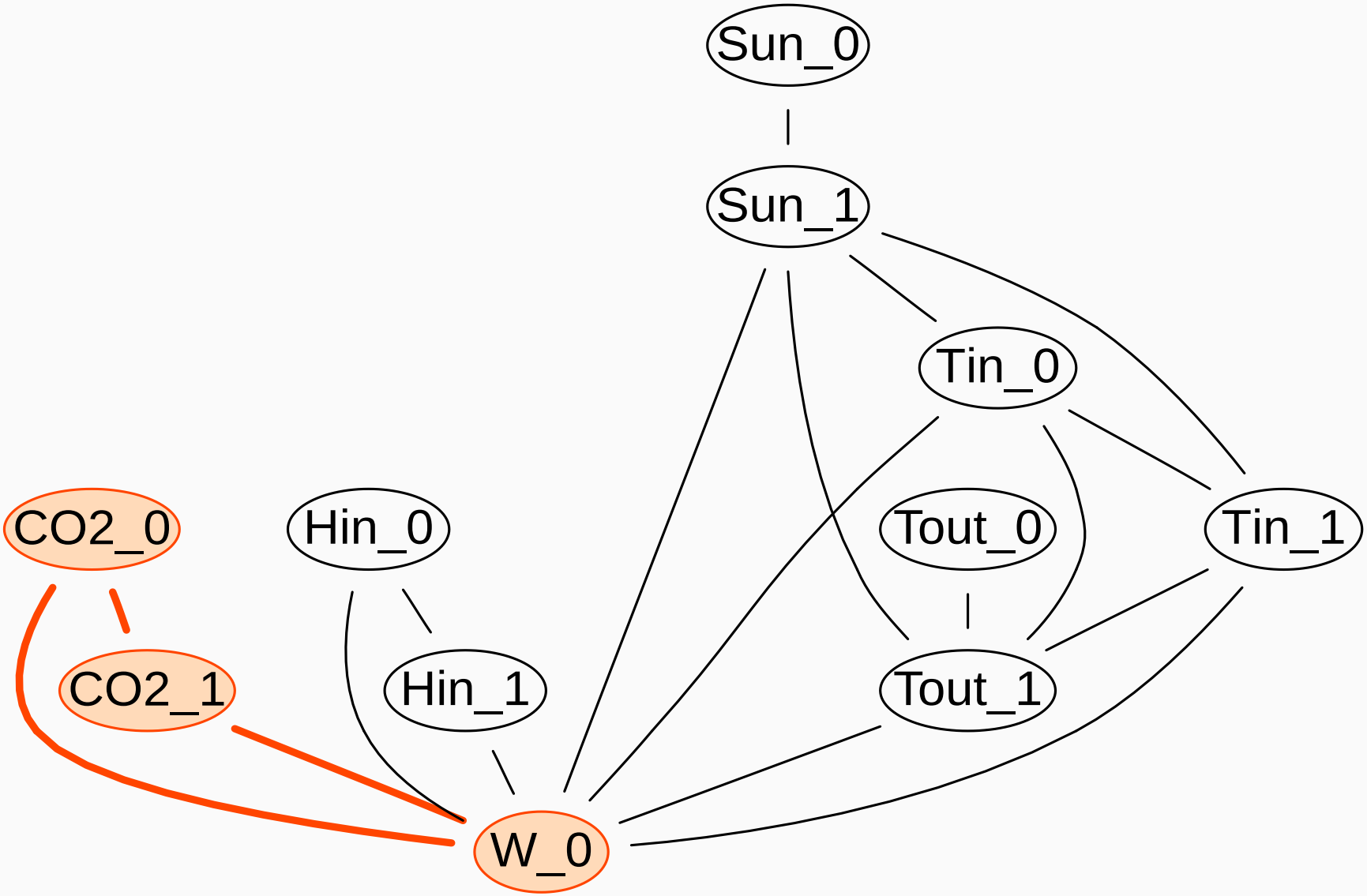


Constructing the Moral Graph

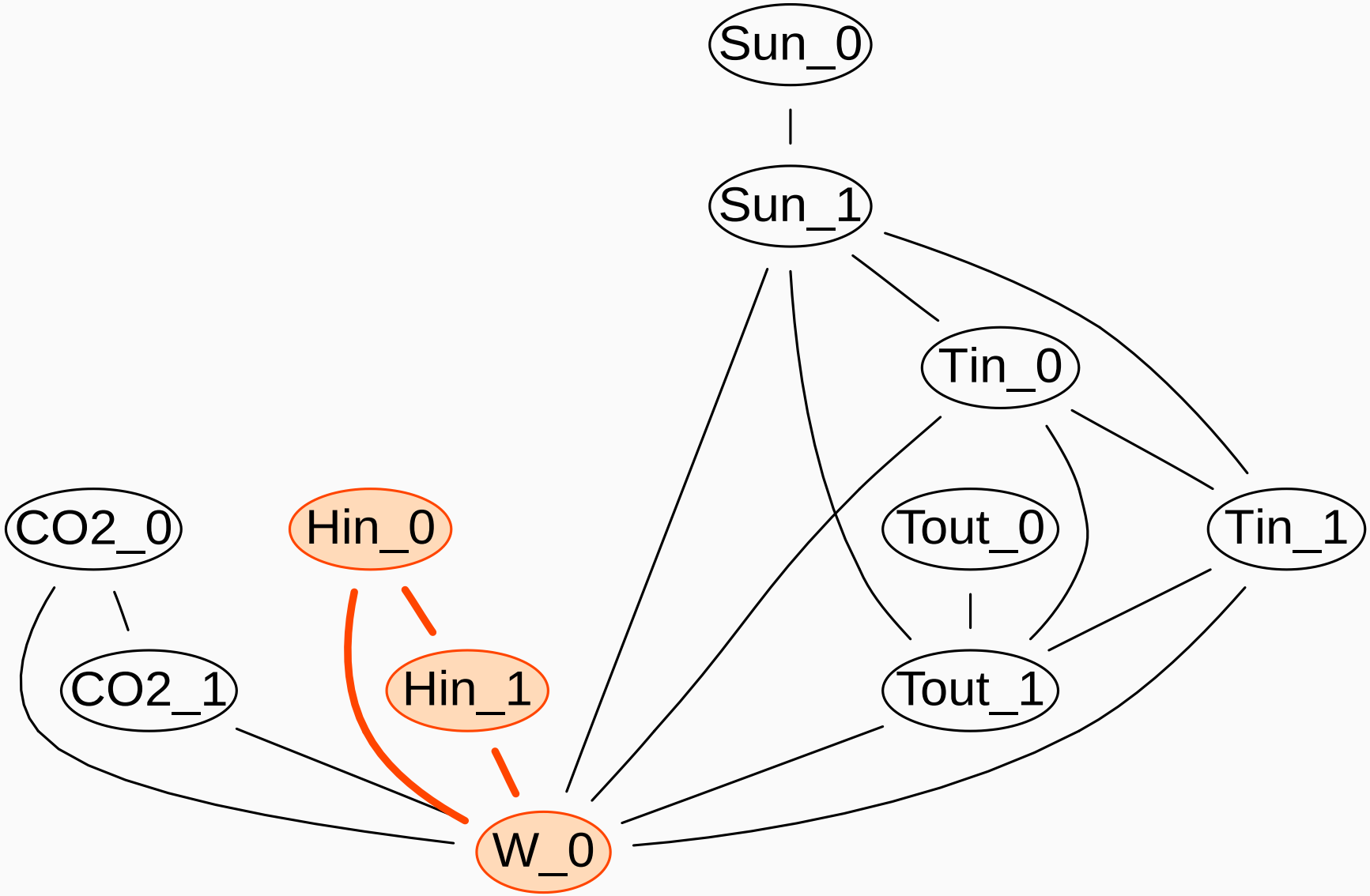


Identifying the Cliques

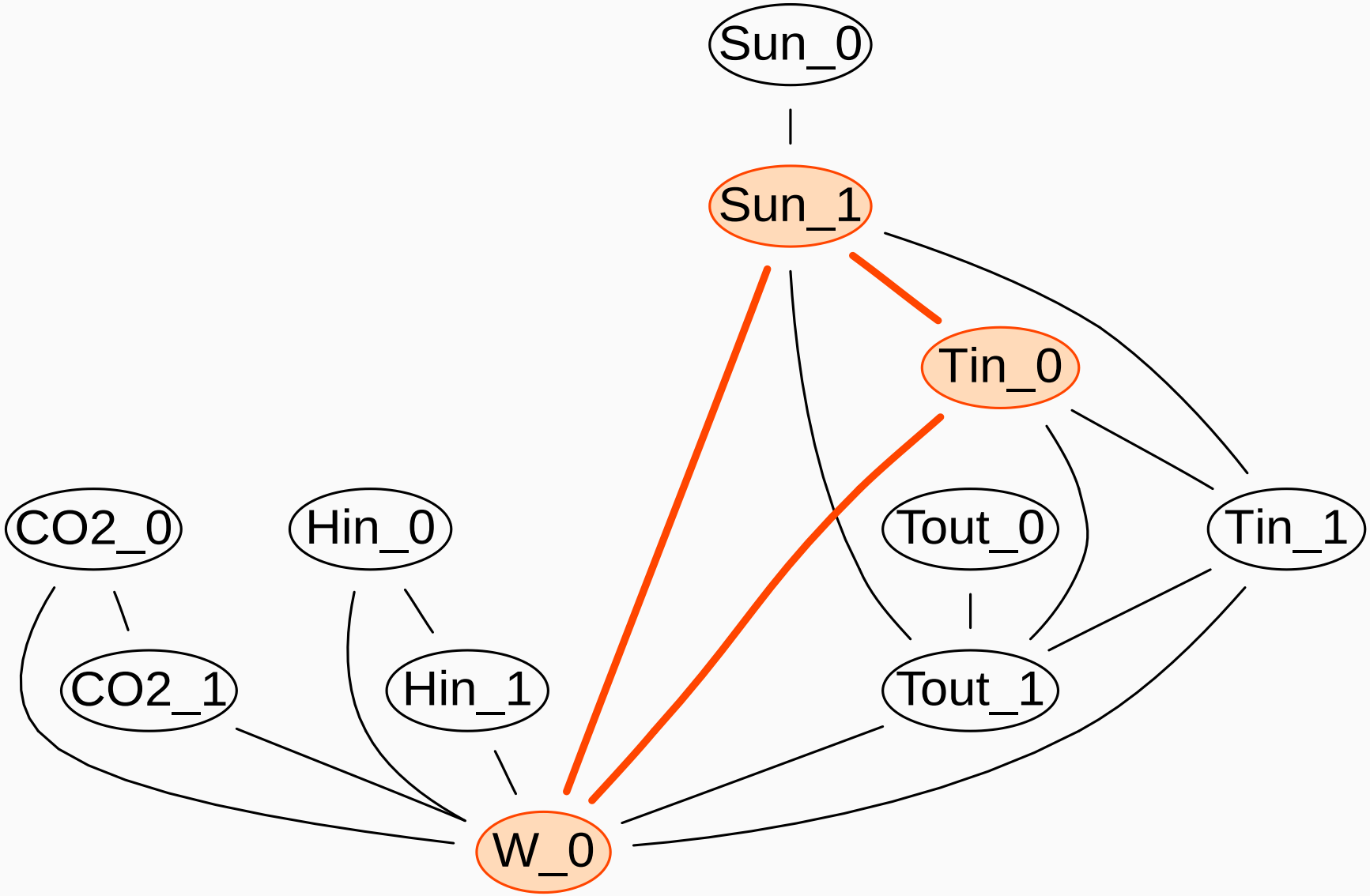
The second step is to identify the cliques in the moral graph.



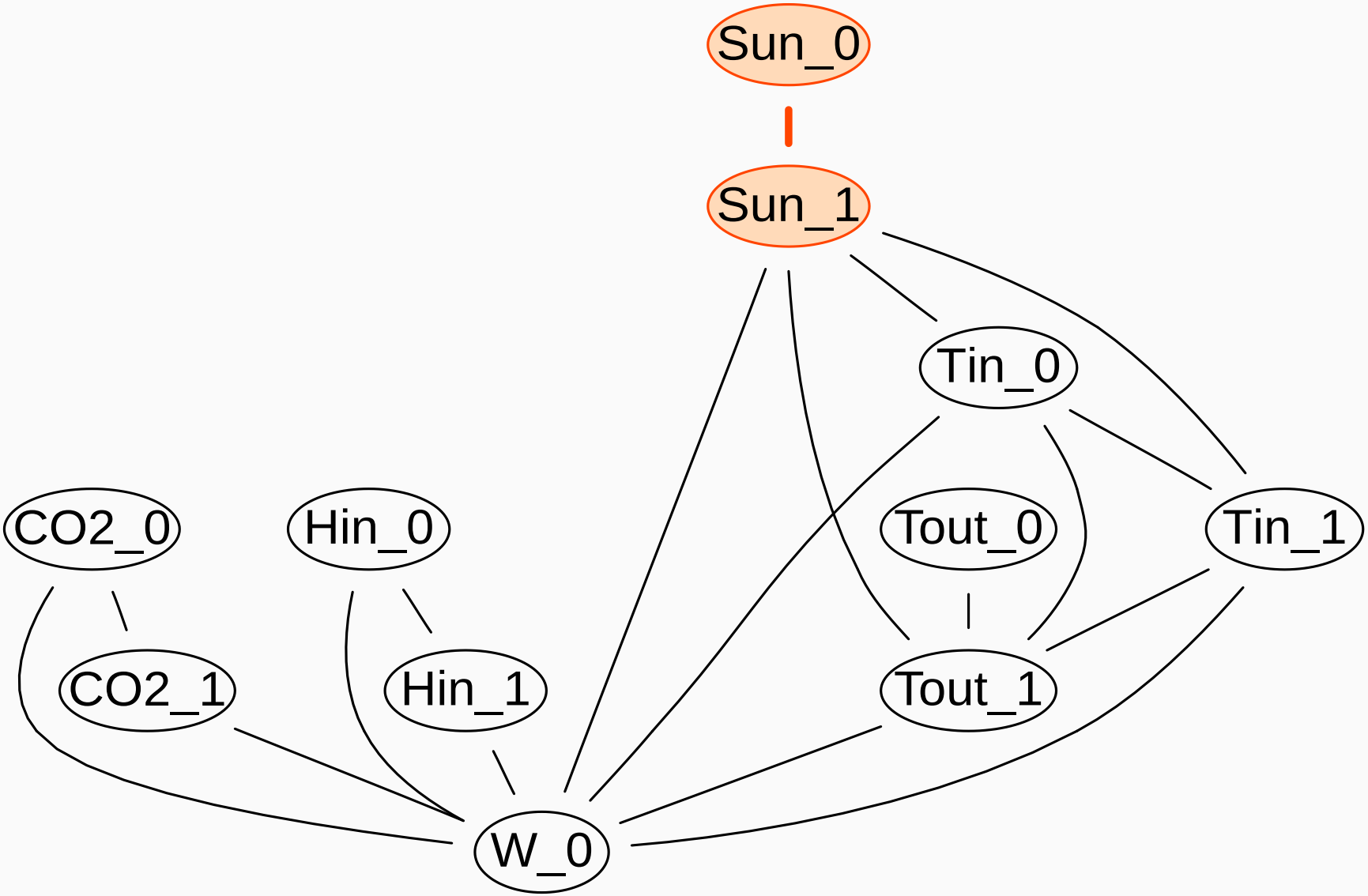
Identifying the Cliques



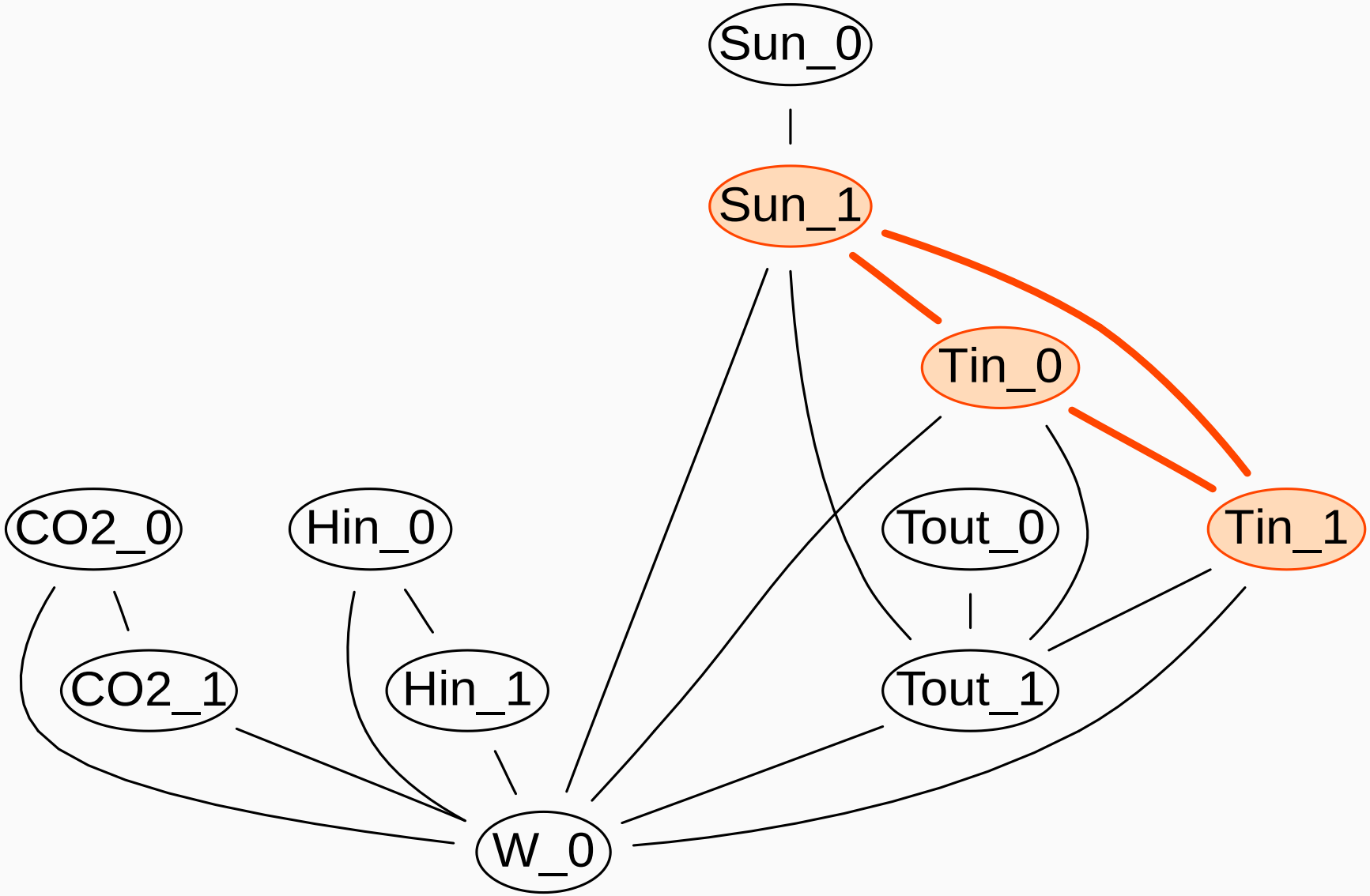
Identifying the Cliques



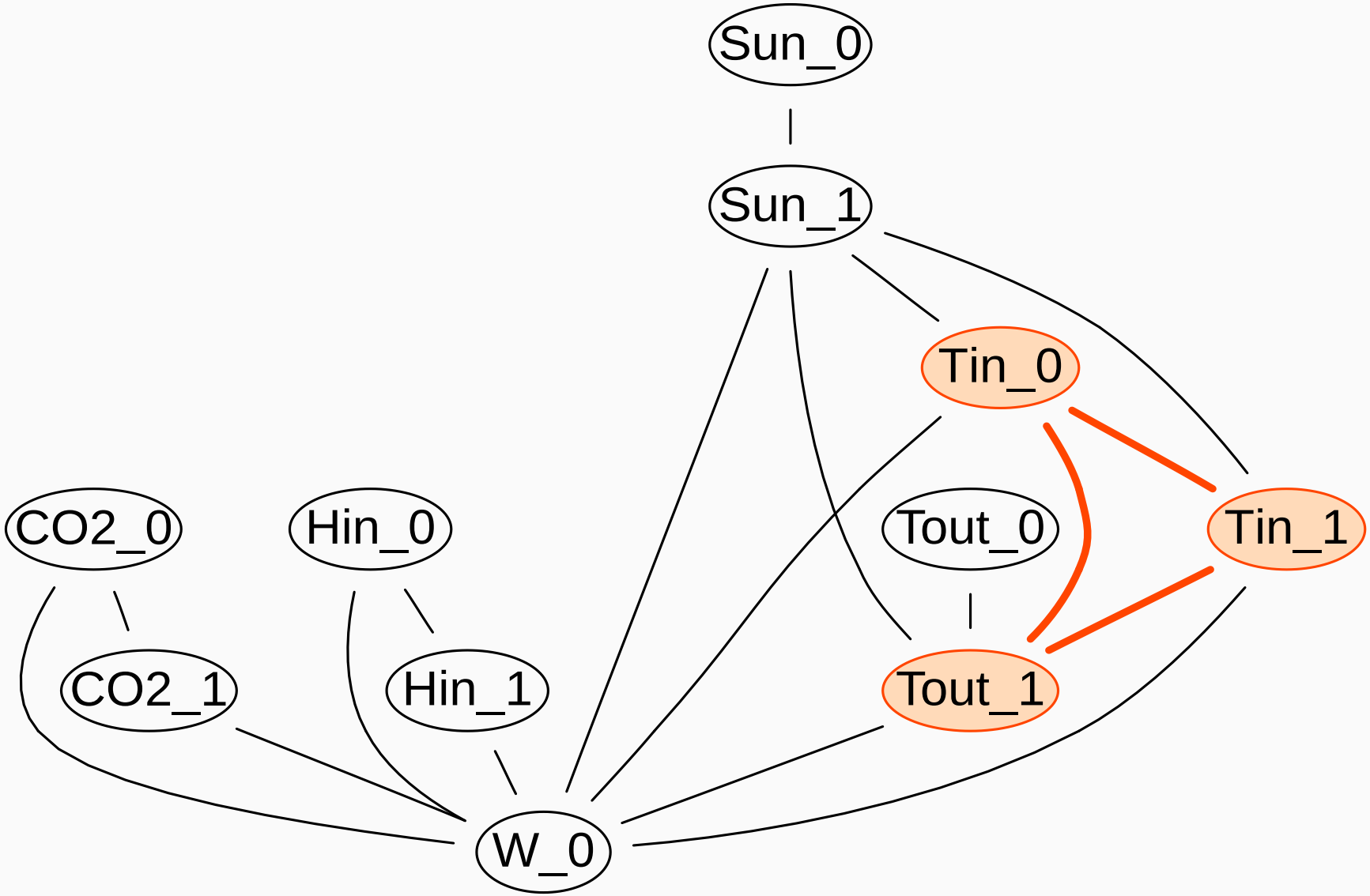
Identifying the Cliques



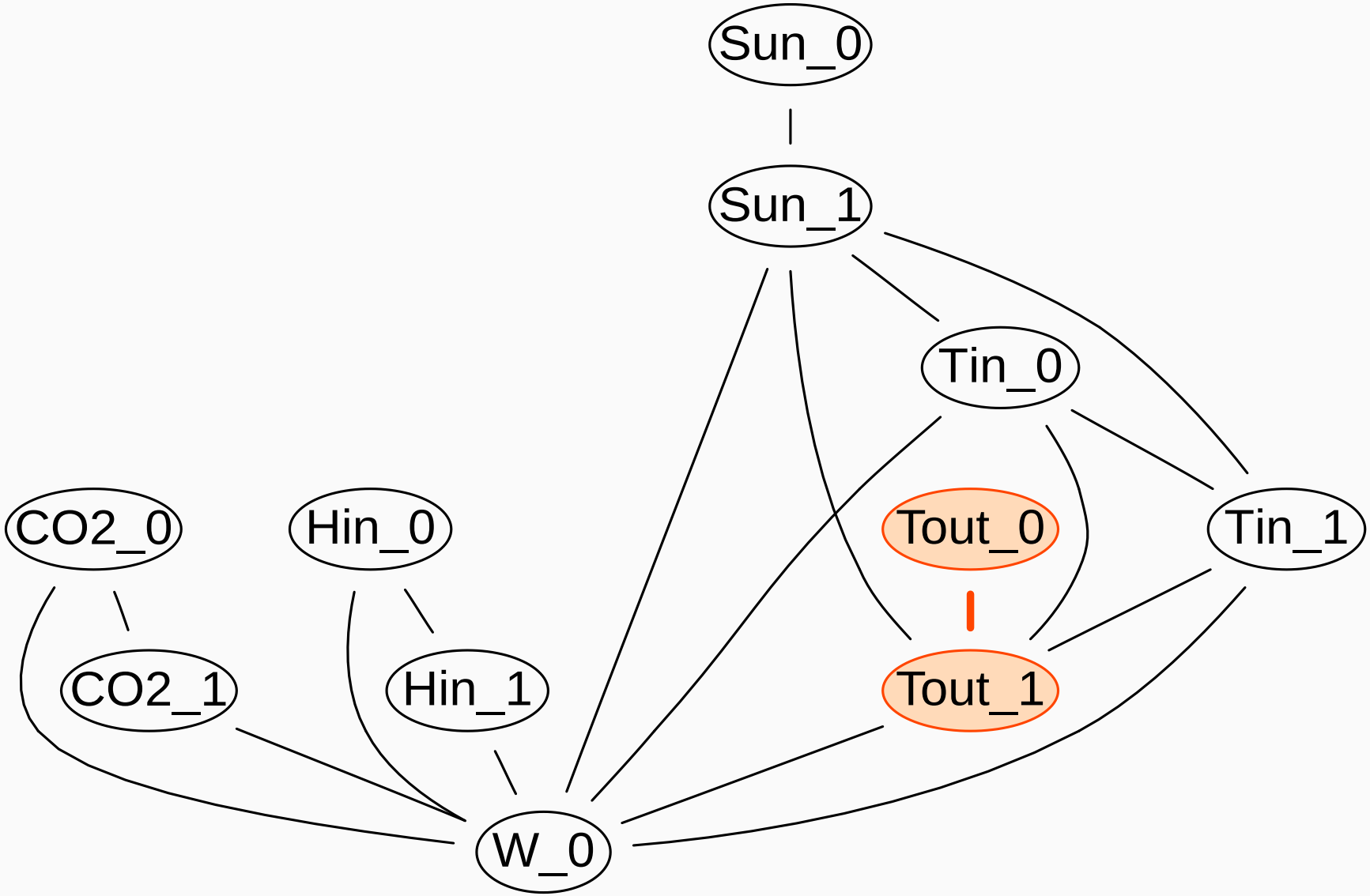
Identifying the Cliques



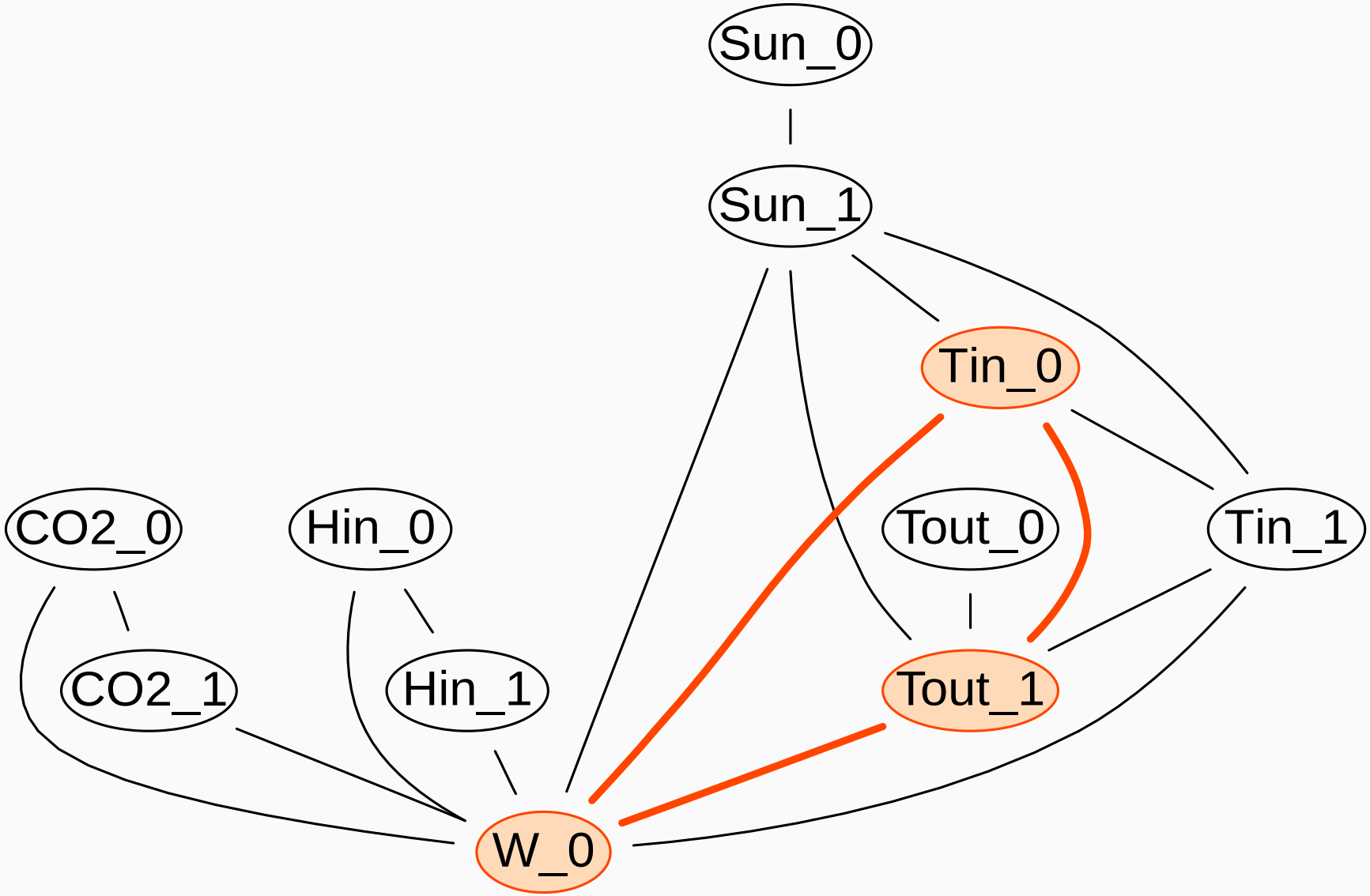
Identifying the Cliques



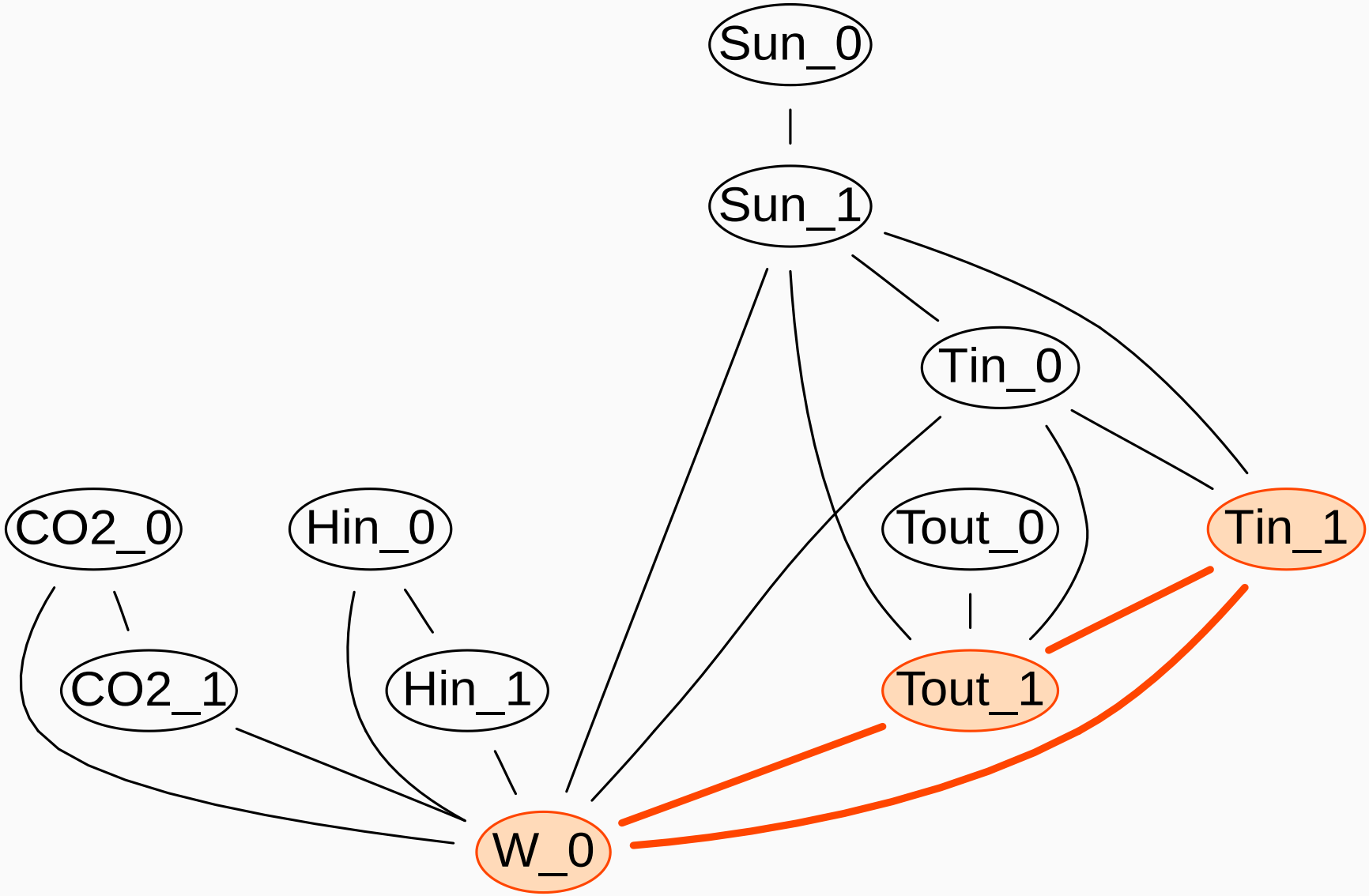
Identifying the Cliques



Identifying the Cliques



Identifying the Cliques



Identifying the Cliques: the Complete Set

The complete list of the cliques is:

$$C_1 = \{\texttt{W\_0}, \texttt{CO2\_0}, \texttt{CO2\_1}\}$$
$$C_2 = \{\texttt{W\_0}, \texttt{Hin\_0}, \texttt{Hin\_1}\}$$
$$C_3 = \{\texttt{W\_0}, \texttt{Sun\_1}, \texttt{Tin\_0}\}$$
$$C_4 = \{\texttt{Sun\_0}, \texttt{Sun\_1}\}$$
$$C_5 = \{\texttt{Sun\_1}, \texttt{Tin\_0}, \texttt{Tin\_1}\}$$
$$C_6 = \{\texttt{Tin\_0}, \texttt{Tin\_1}, \texttt{Tout\_1}\}$$
$$C_7 = \{\texttt{Tout\_0}, \texttt{Tout\_1}\}$$
$$C_8 = \{\texttt{W\_0}, \texttt{Tout\_1}, \texttt{Tin\_0}\}$$
$$C_9 = \{\texttt{W\_0}, \texttt{Tout\_1}, \texttt{Tin\_1}\}$$

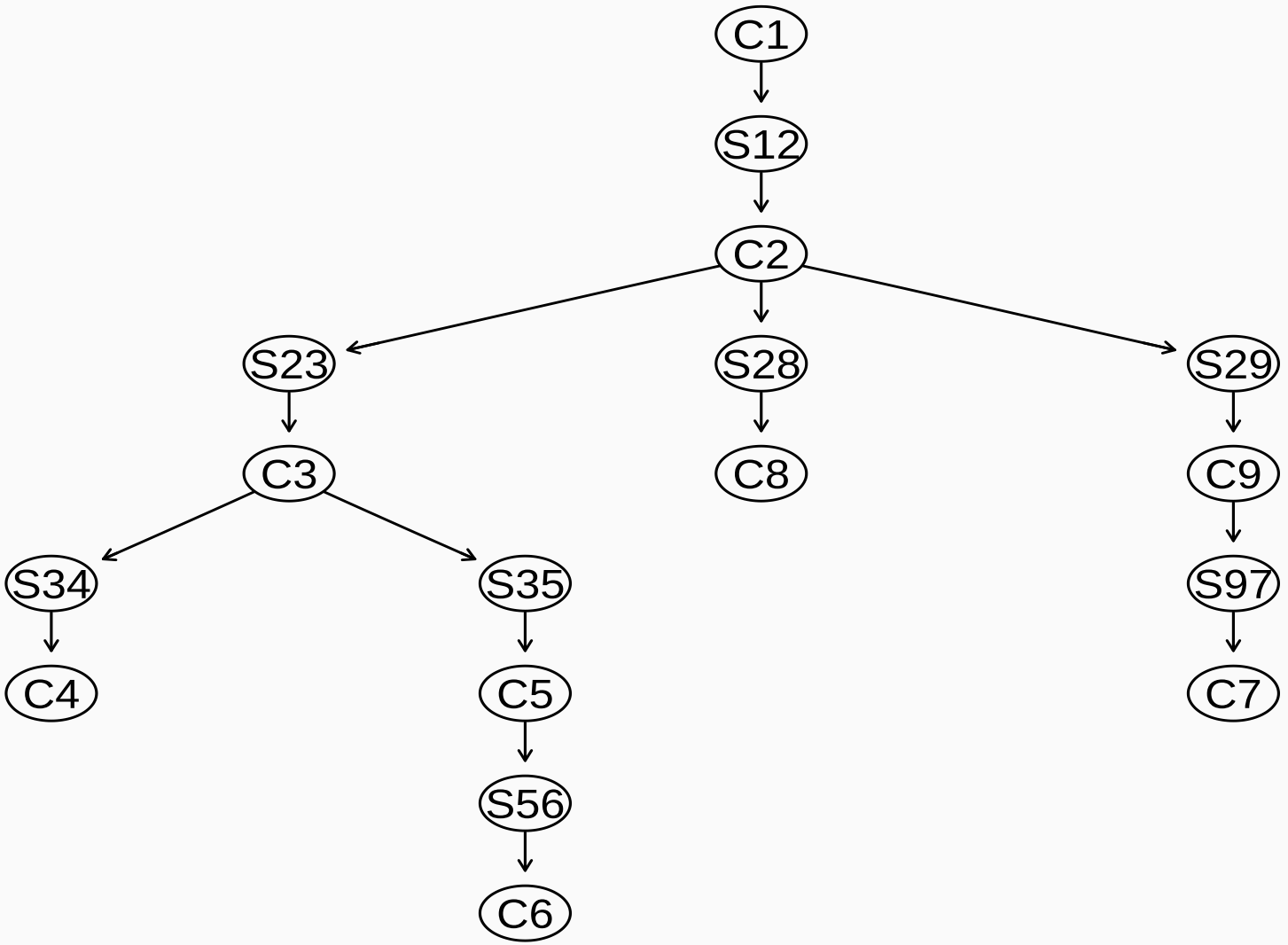
$$S_{12} = \{\texttt{W\_0}\}$$
$$S_{23} = \{\texttt{W\_0}\}$$
$$S_{34} = \{\texttt{Sun\_1}\}$$
$$S_{35} = \{\texttt{Sun\_1}, \texttt{Tin\_0}\}$$
$$S_{56} = \{\texttt{Tin\_0}, \texttt{Tin\_1}\}$$
$$S_{97} = \{\texttt{Tout\_1}\}$$
$$S_{28} = \{\texttt{W\_0}\}$$
$$S_{29} = \{\texttt{W\_0}\}$$

Now we have to join them to build the junction tree.

The Many Invariants of Junction Trees

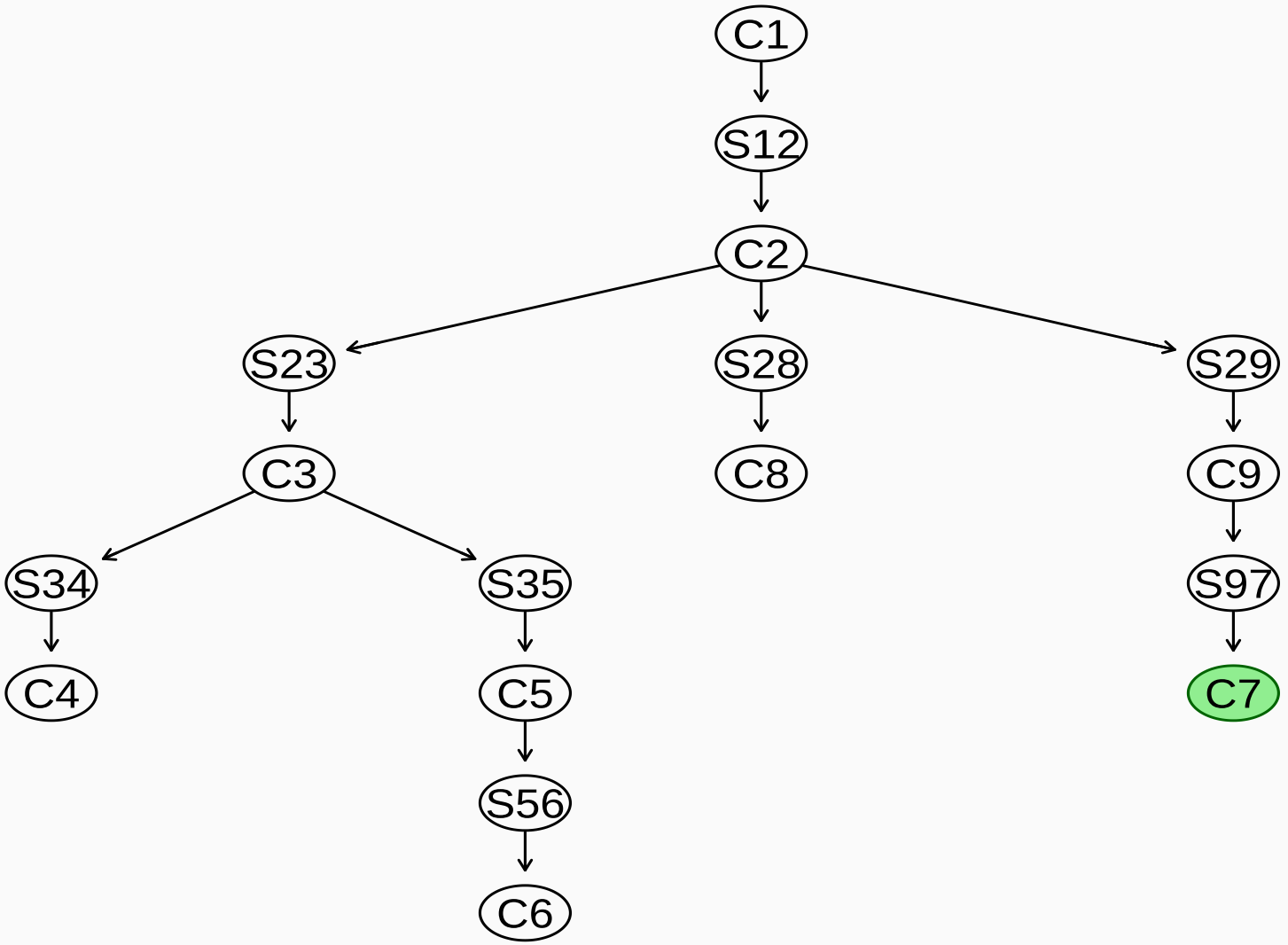
- Each clique is typically adjacent to many other cliques; if we just build an undirected graph including all possible separators it will not be a tree. We include only enough separators to make it a tree; which ones we choose does not influence the results of the queries.
- Wider trees may be preferable to deeper trees, because we can do propagation in parallel on different branches.
- The junction trees is an undirected tree; we can pick an arbitrary node as the root and add arc directions, but they do not matter in determining the sequence of steps to propagate the evidence.
- There typically are multiple valid sequences of steps in evidence propagation; all of them lead to the same results and require the same computations (obviously in different orders).

Building the Junction Tree



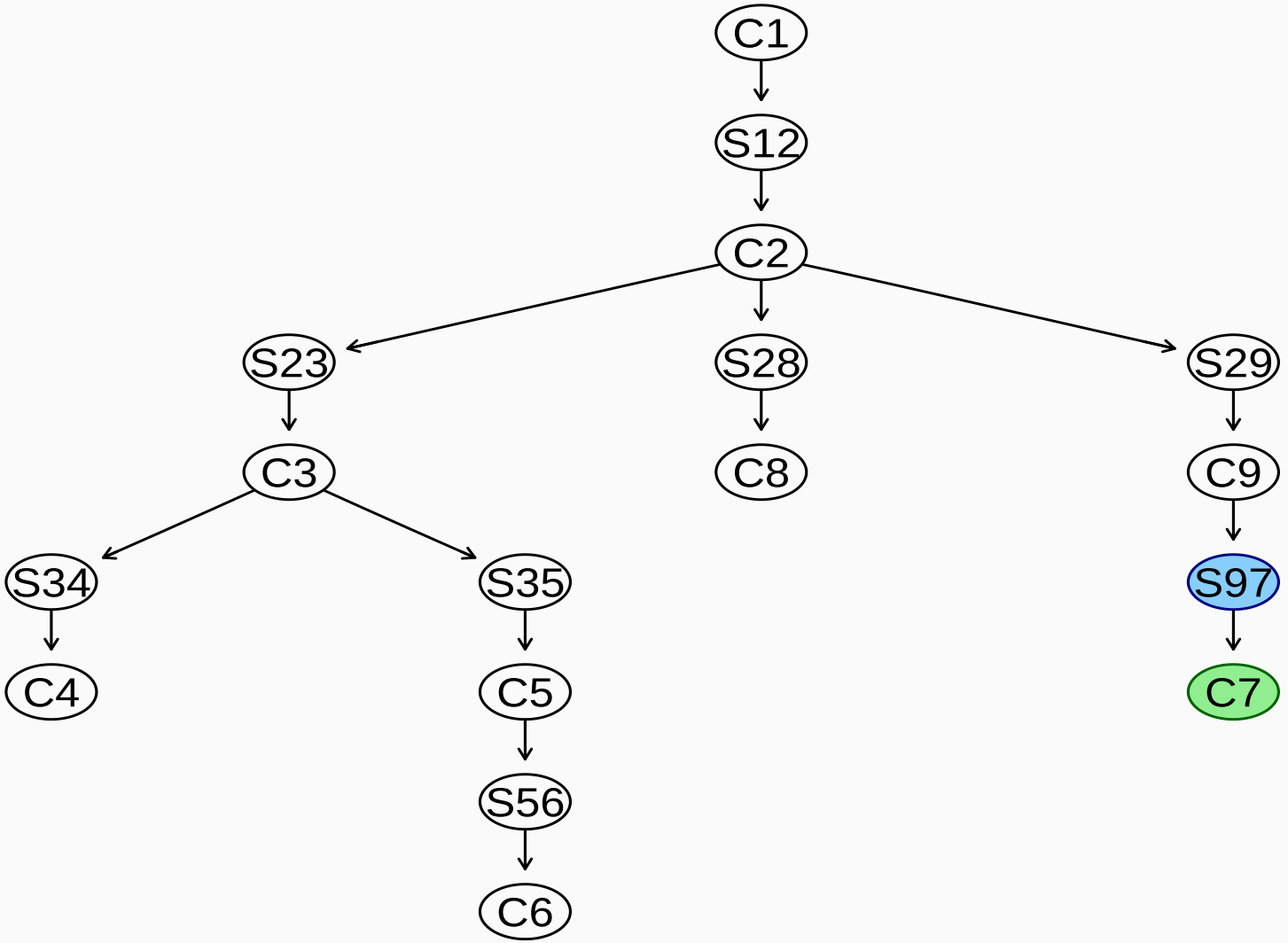
(Arc directions are arbitrary, any node can be the root node.)

Set the Evidence



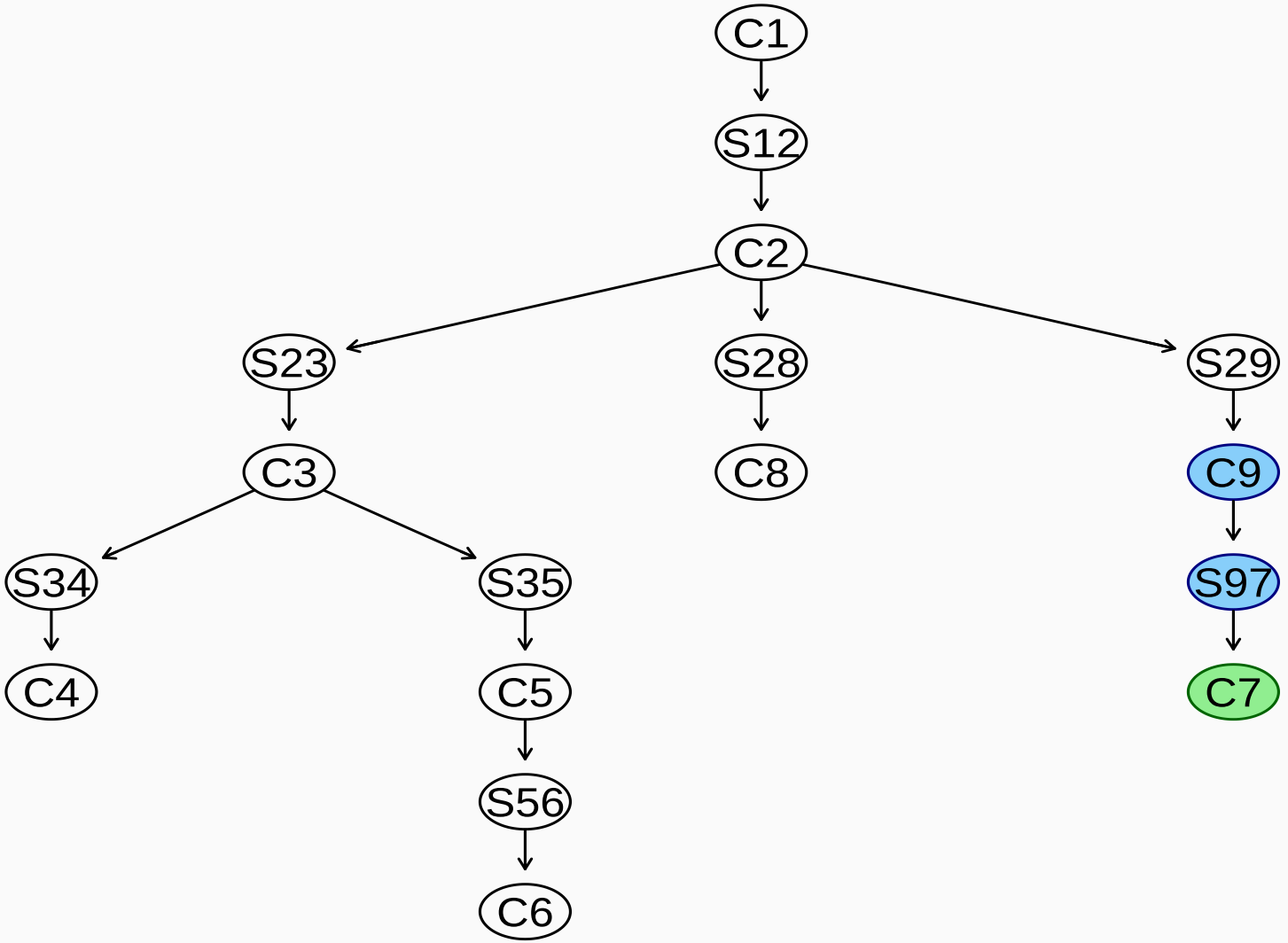
Set the first piece of evidence (`Tout_0` is equal to <18) in C_7.

Propagate the Evidence



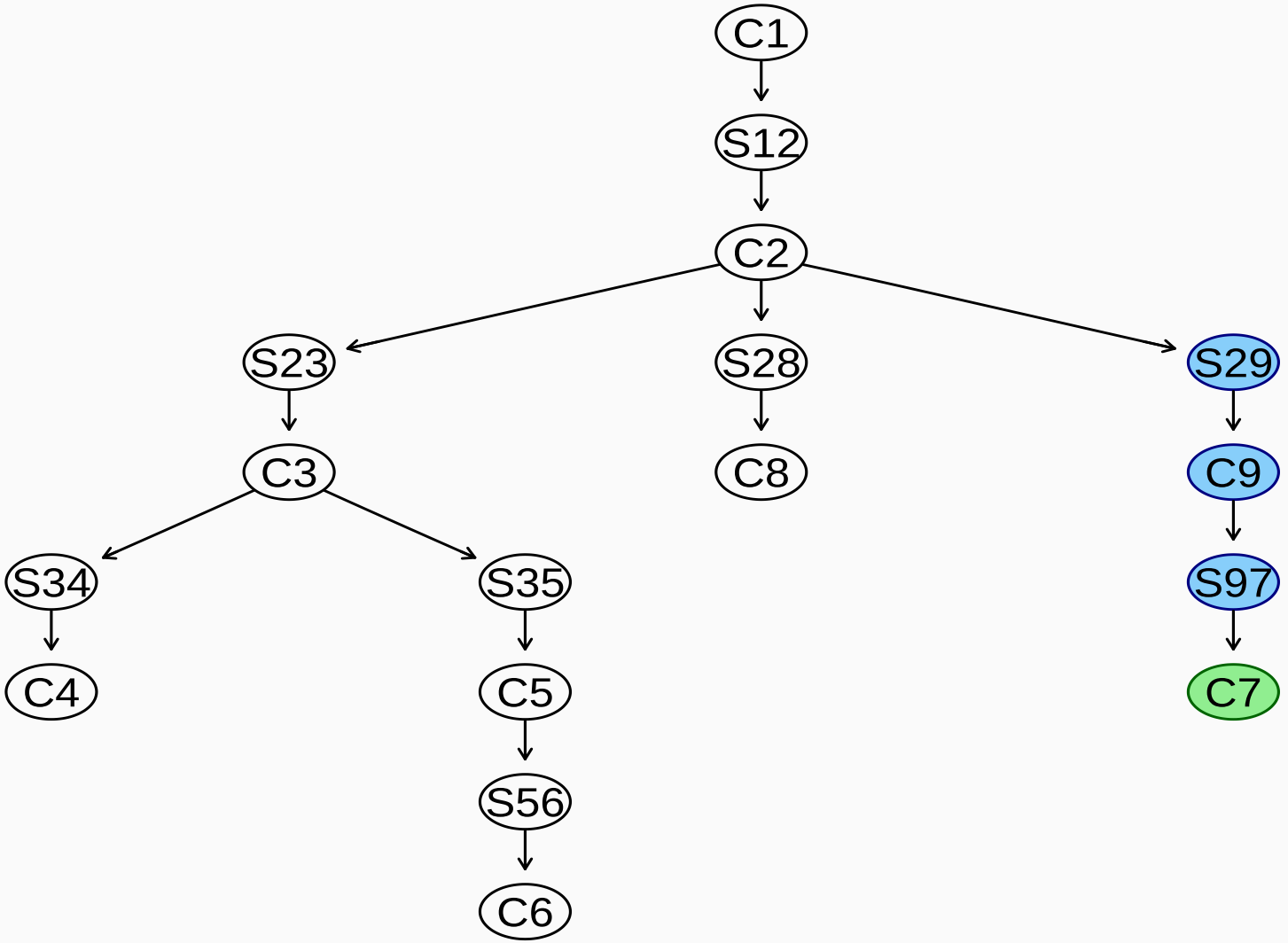
Update the marginal distribution of `Tout_1` in S_{97} from the joint distribution of `Tout_0`, `Tout_1` in C_7.

Propagate the Evidence



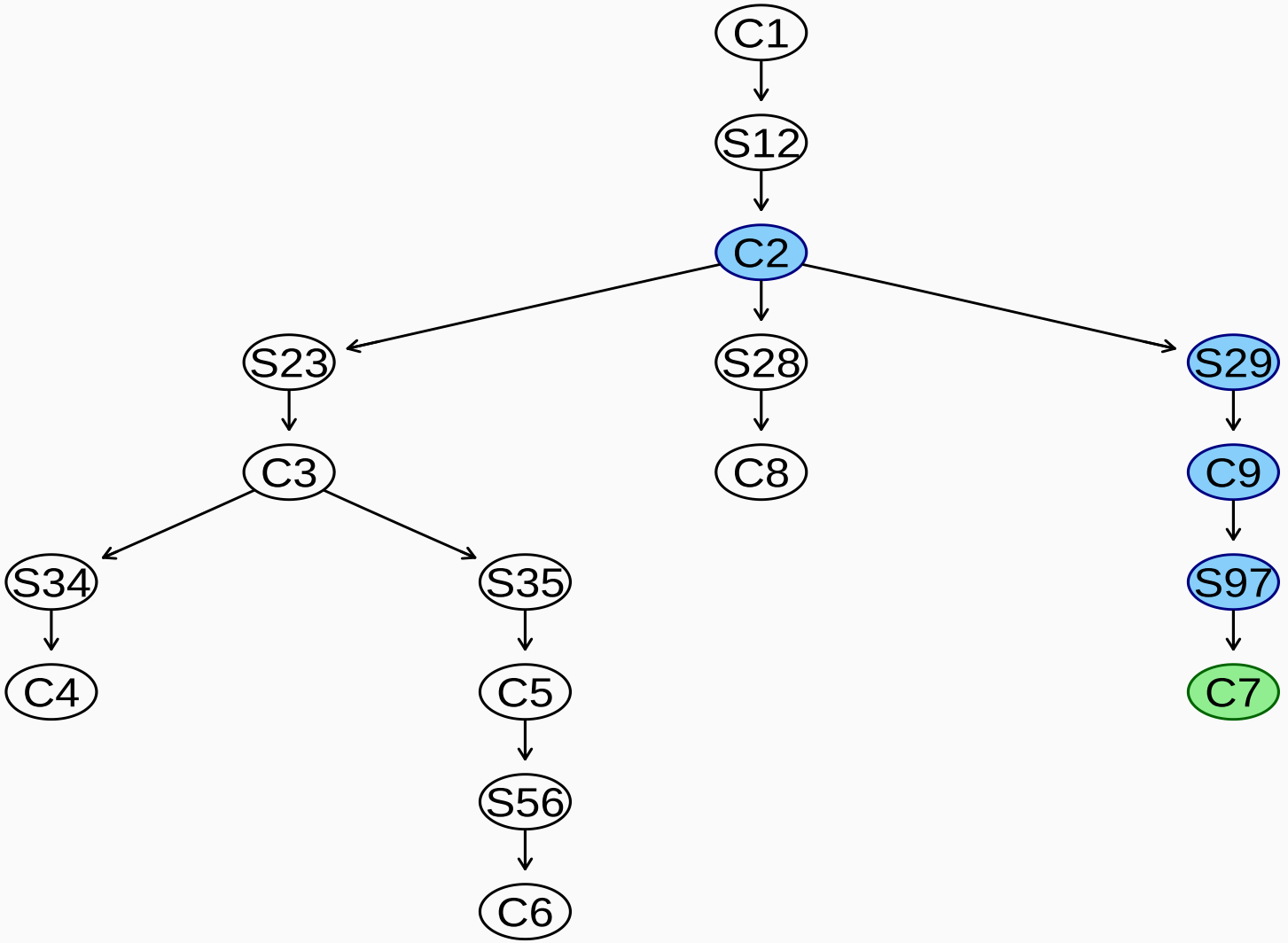
Update the joint distribution of `W_0`, `Tout_1`, `Tin_1` in C_9 from the marginal distribution of `Tout_1` in S_{97}.

Propagate the Evidence



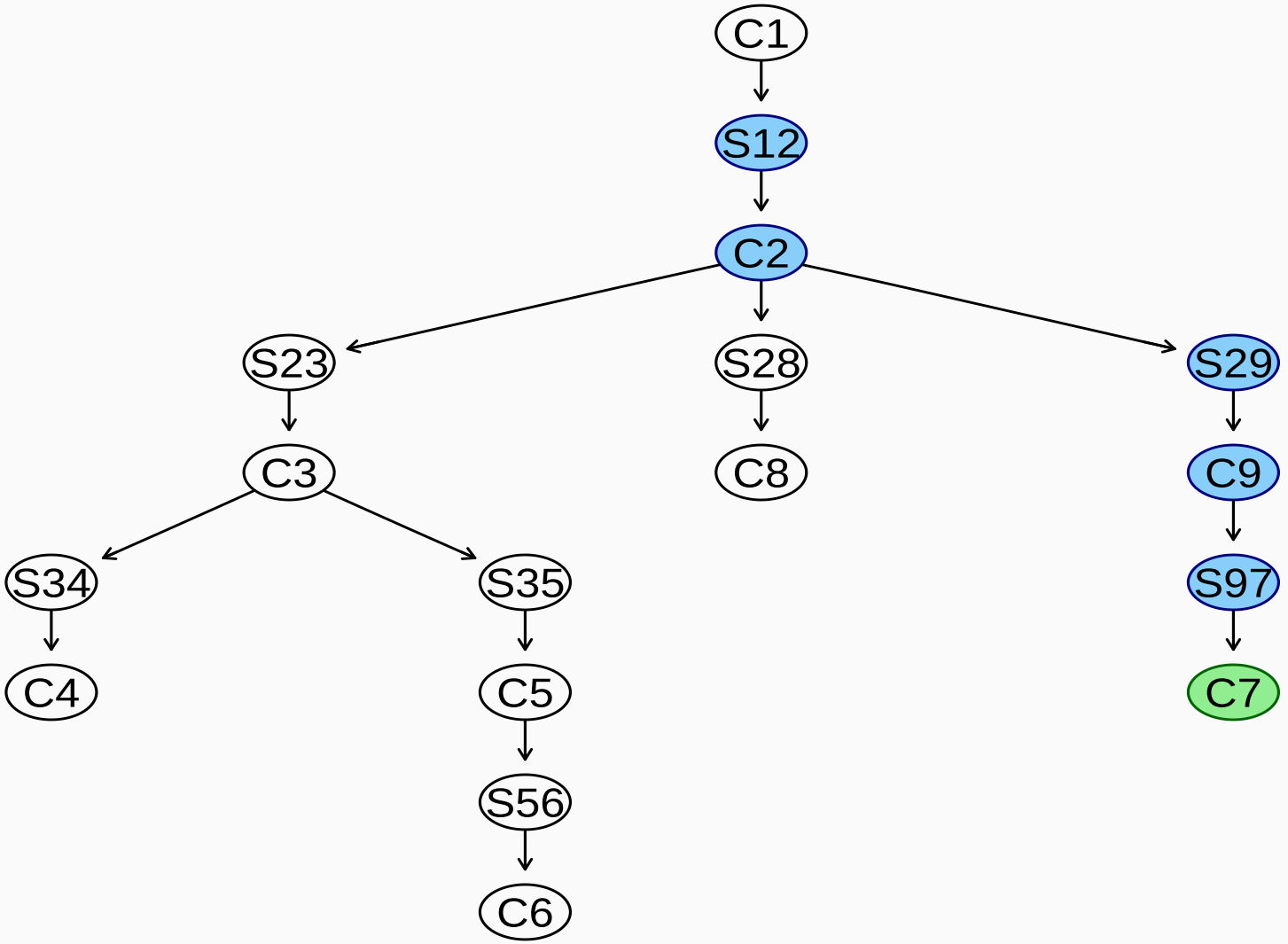
Update the marginal distribution of `W_0` in S_{29} from the joint distribution of `W_0`, `Tout_1`, `Tin_1` in C_9.

Propagate the Evidence



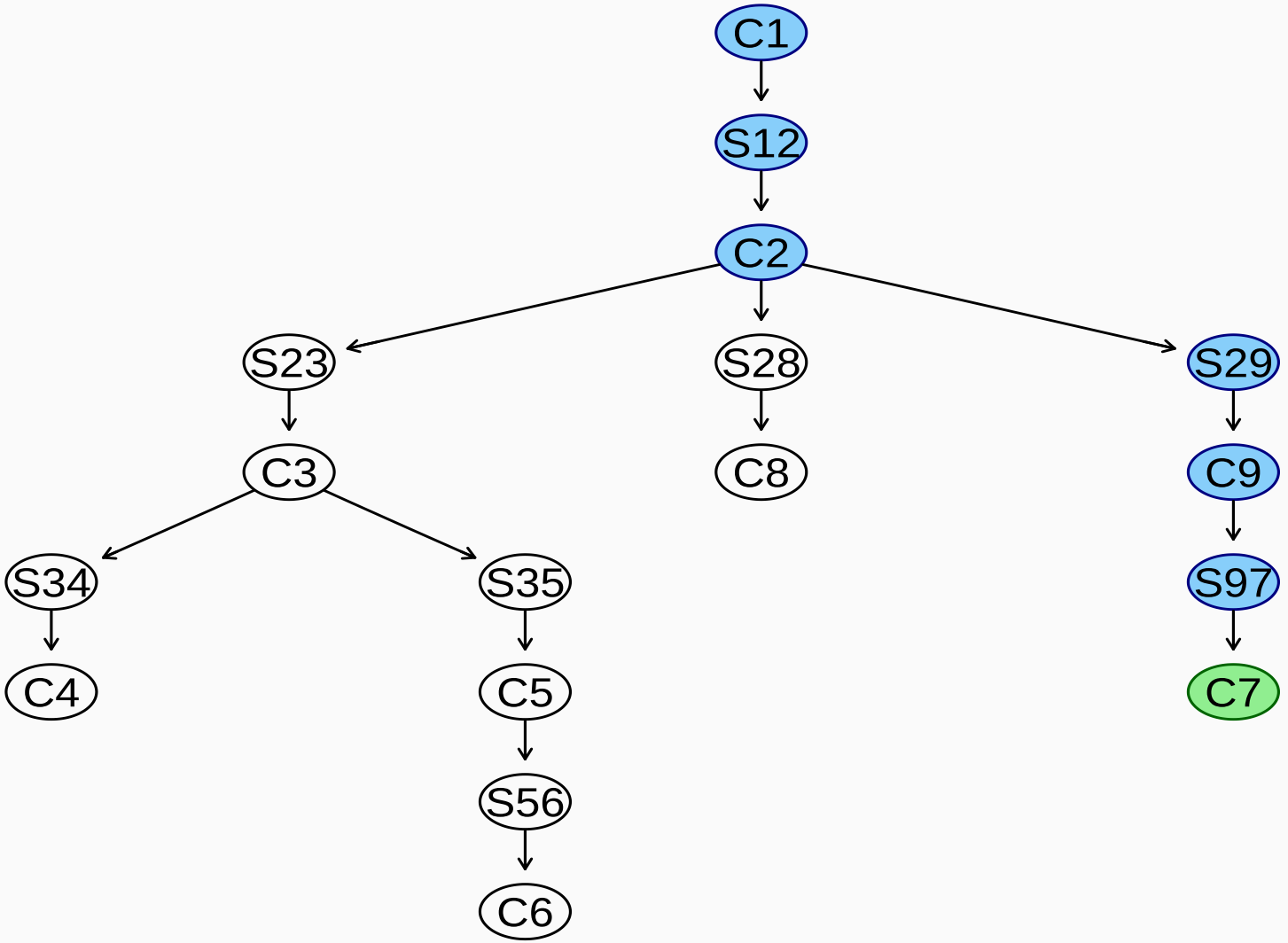
Update the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2 from the marginal distribution of `W_0` in S_{29}.

Propagate the Evidence



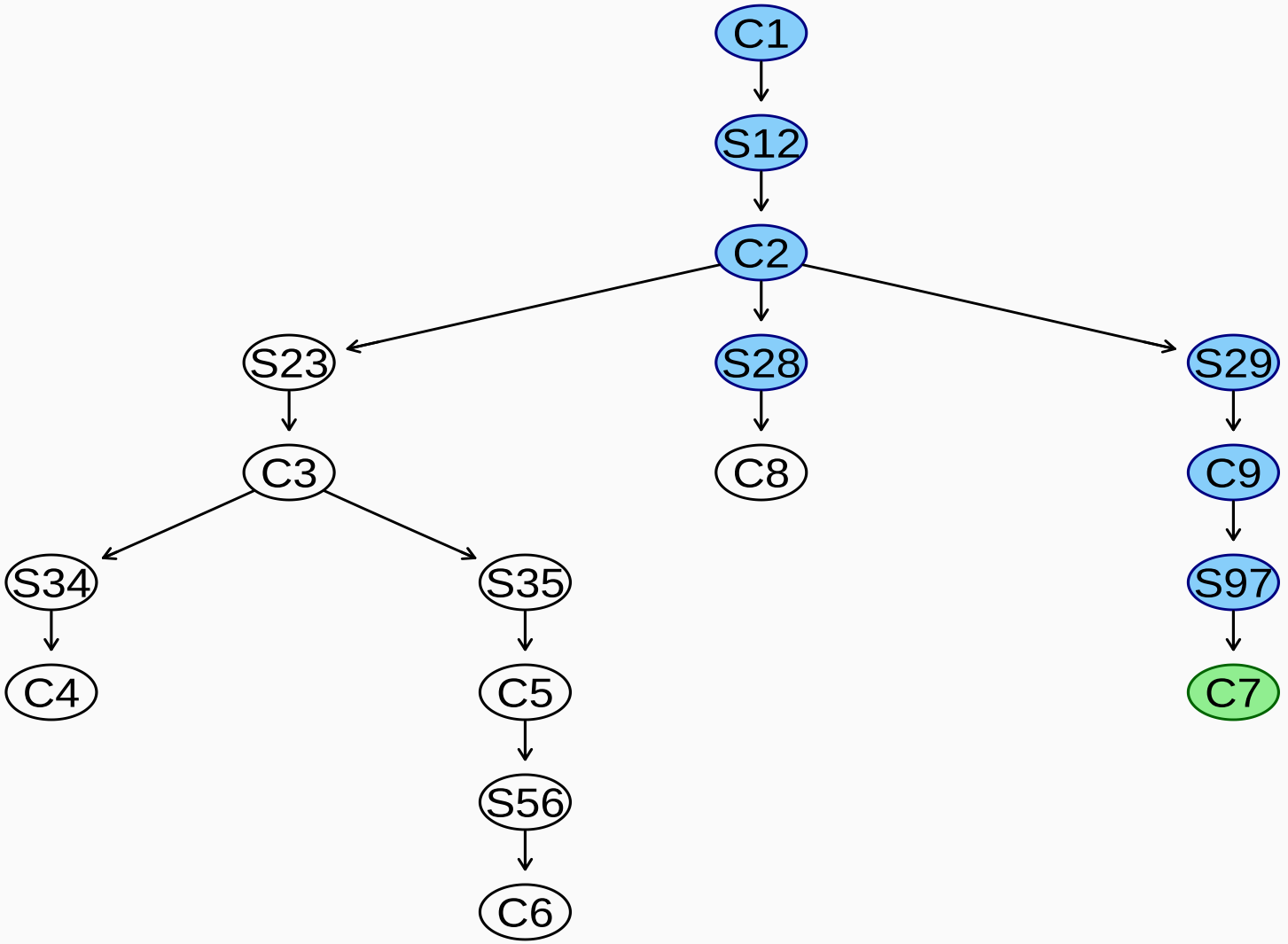
Update the marginal distribution of `W_0` in S_{12} from the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2.

Propagate the Evidence



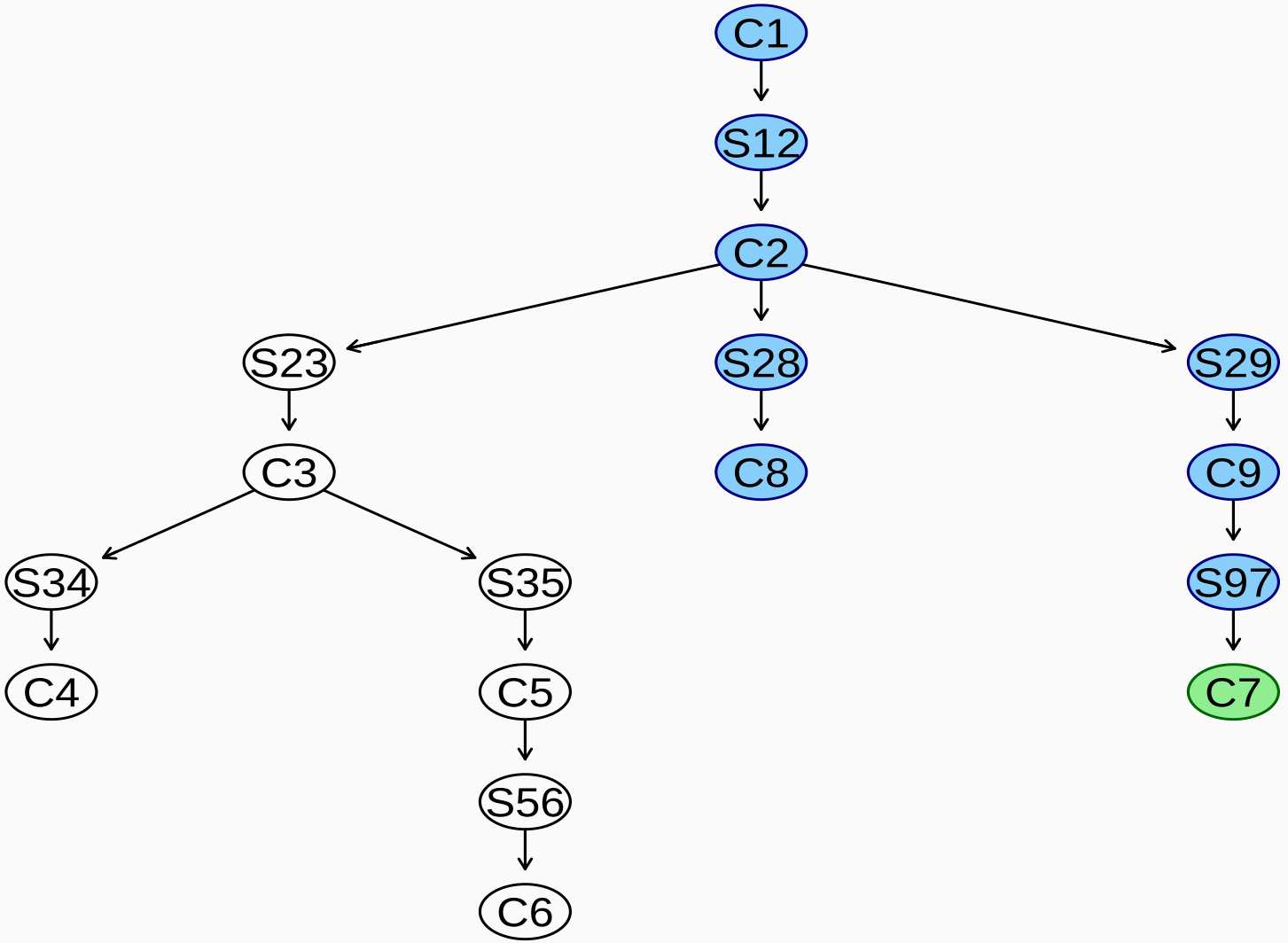
Update the joint distribution of `W_0`, `CO2_0`, `CO2_1` in C_1 from the marginal distribution of `W_0` in S_{12}.

Propagate the Evidence



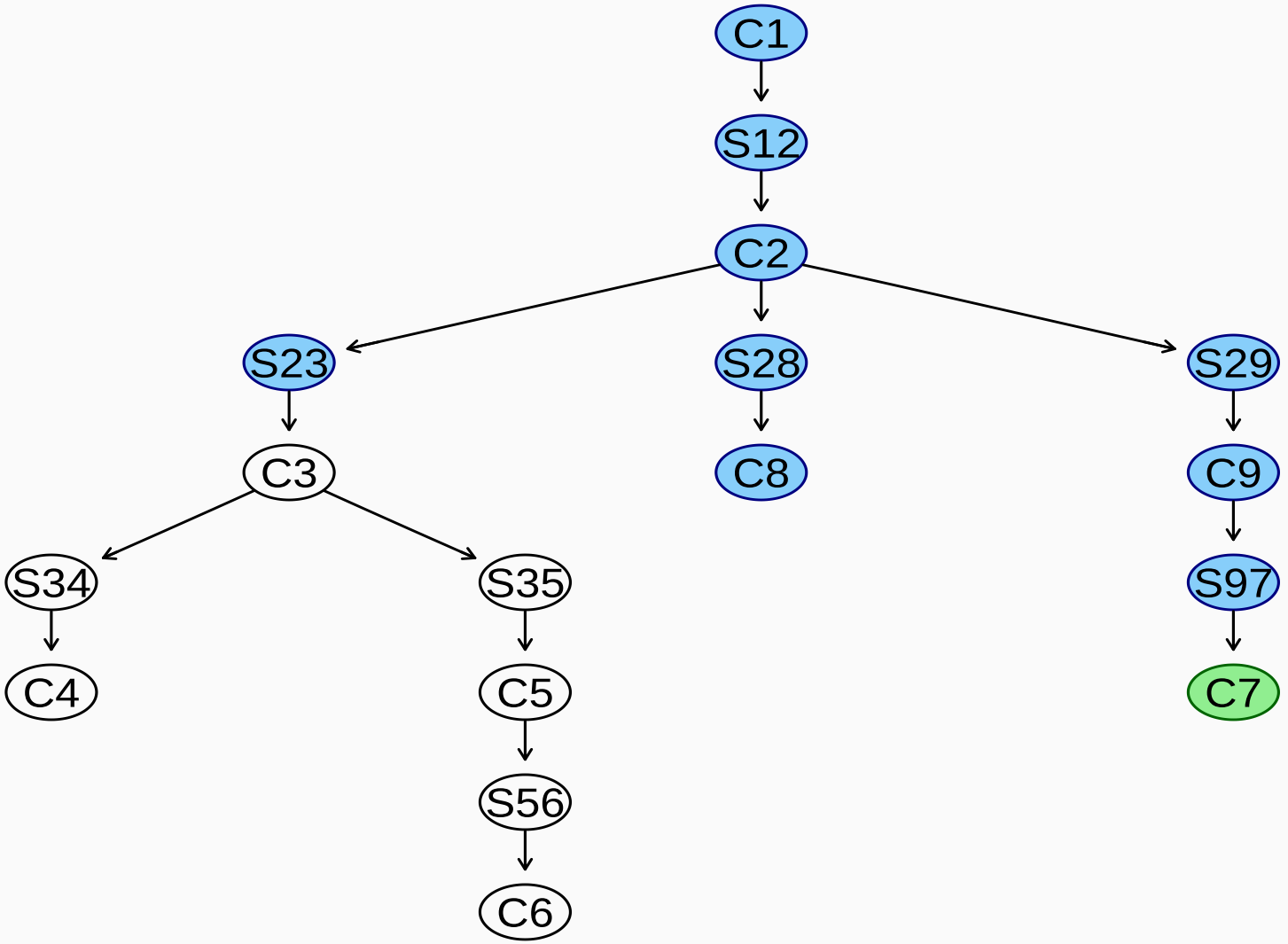
Update the marginal distribution of `W_0` in S_{28} from the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2.

Propagate the Evidence



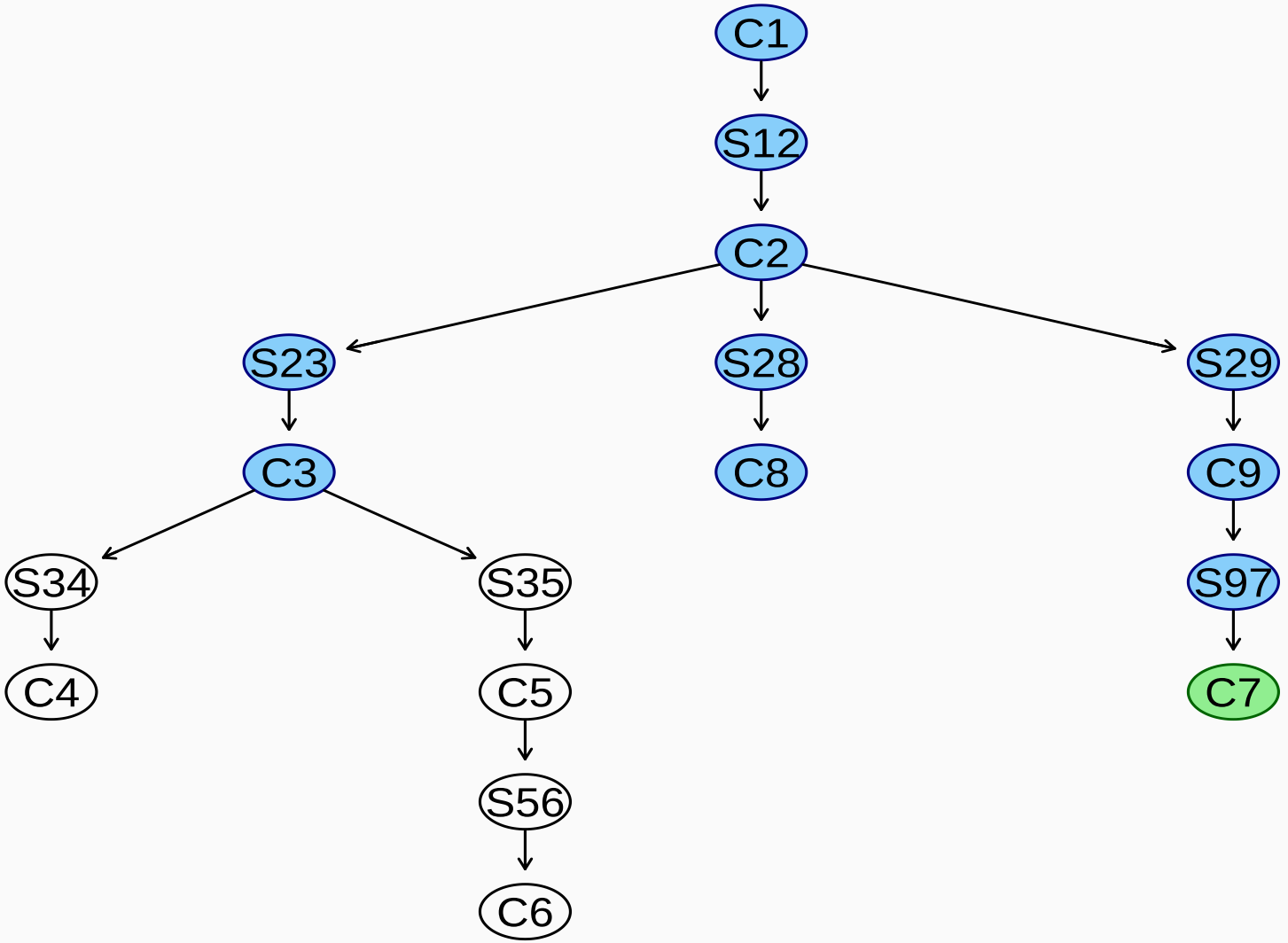
Update the joint distribution of `W_0`, `Tout_1`, `Tin_0` in C_8 from the marginal distribution of `W_0` in S_{28}.

Propagate the Evidence



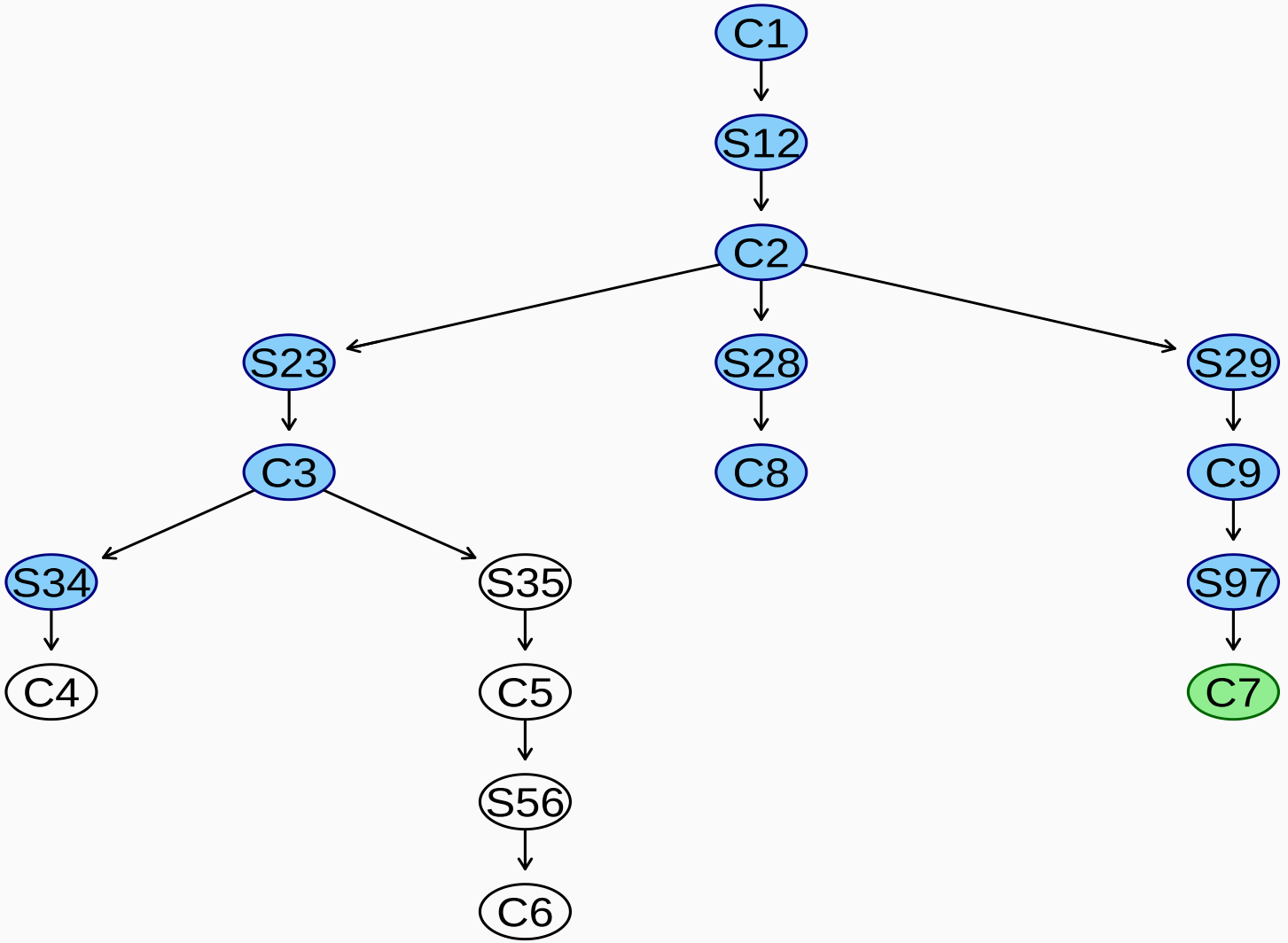
Update the marginal distribution of `W_0` in S_{23} from the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2.

Propagate the Evidence



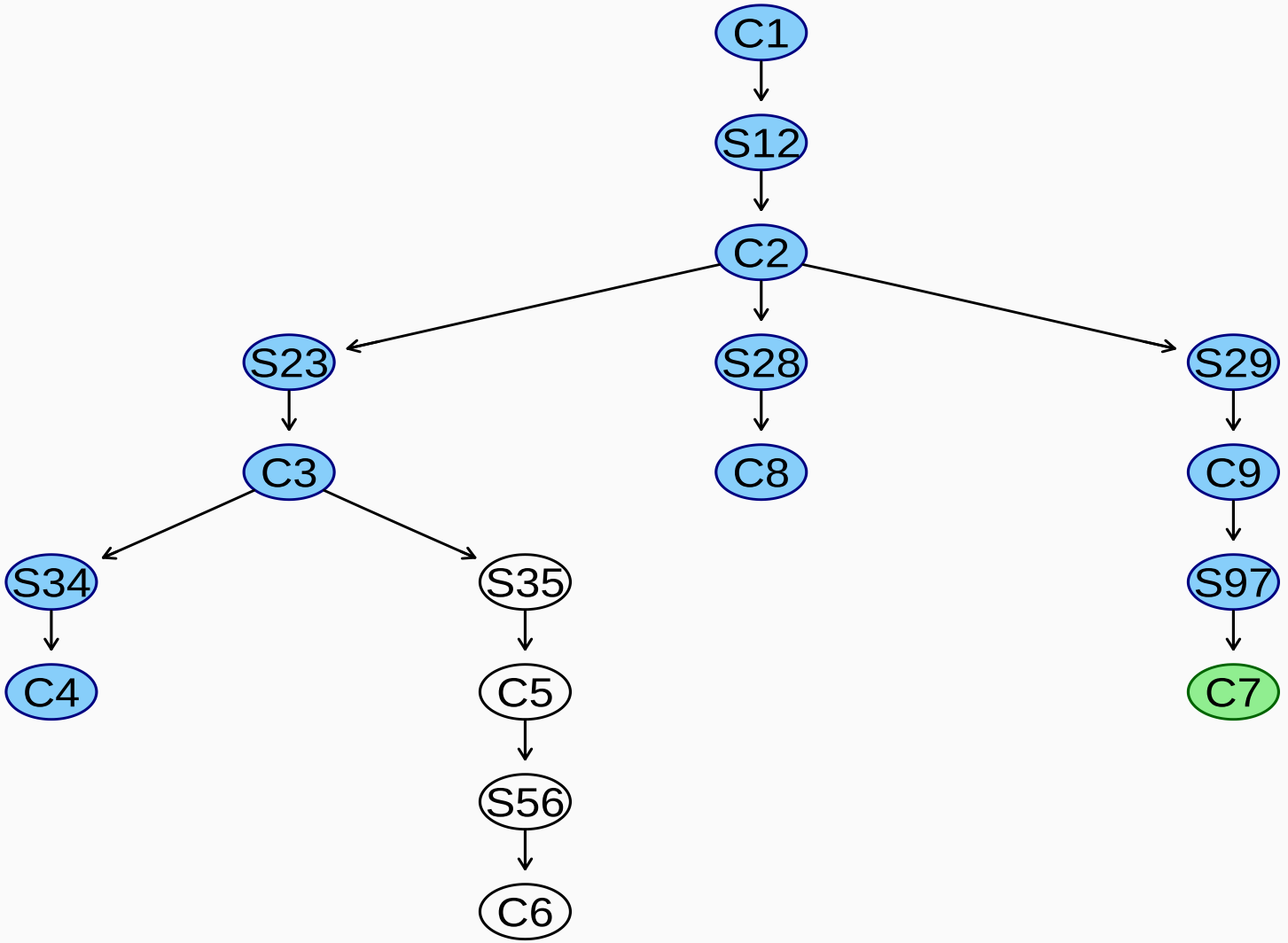
Update the joint distribution of `W_0`, `Sun_1`, `Tin_0` in C_3 from the marginal distribution of `W_0` in S_{23}.

Propagate the Evidence



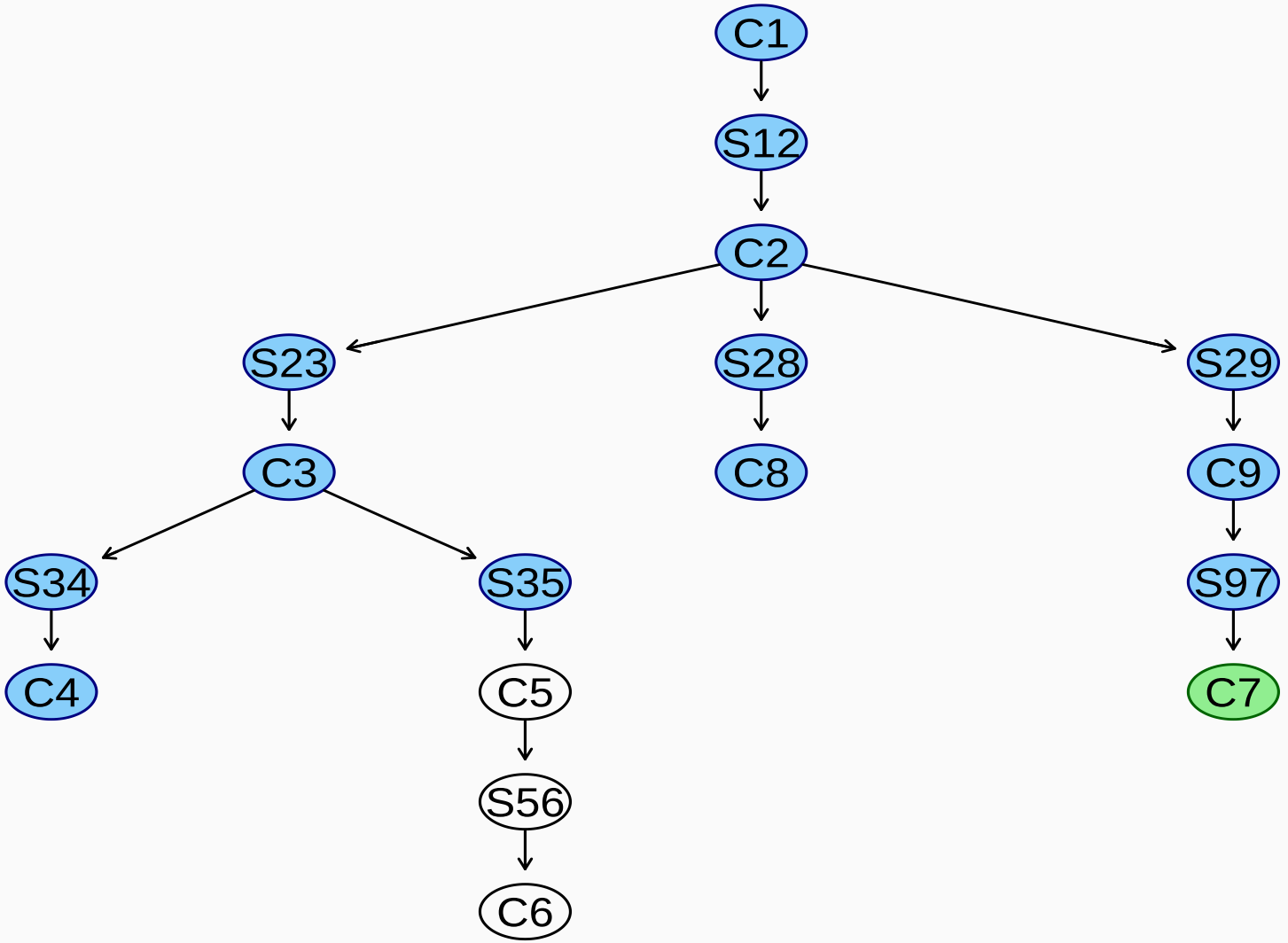
Update the marginal distribution of Sun_1 in S_{34} from the joint distribution of W_0, Sun_1, Tin_0 in C_3.

Propagate the Evidence



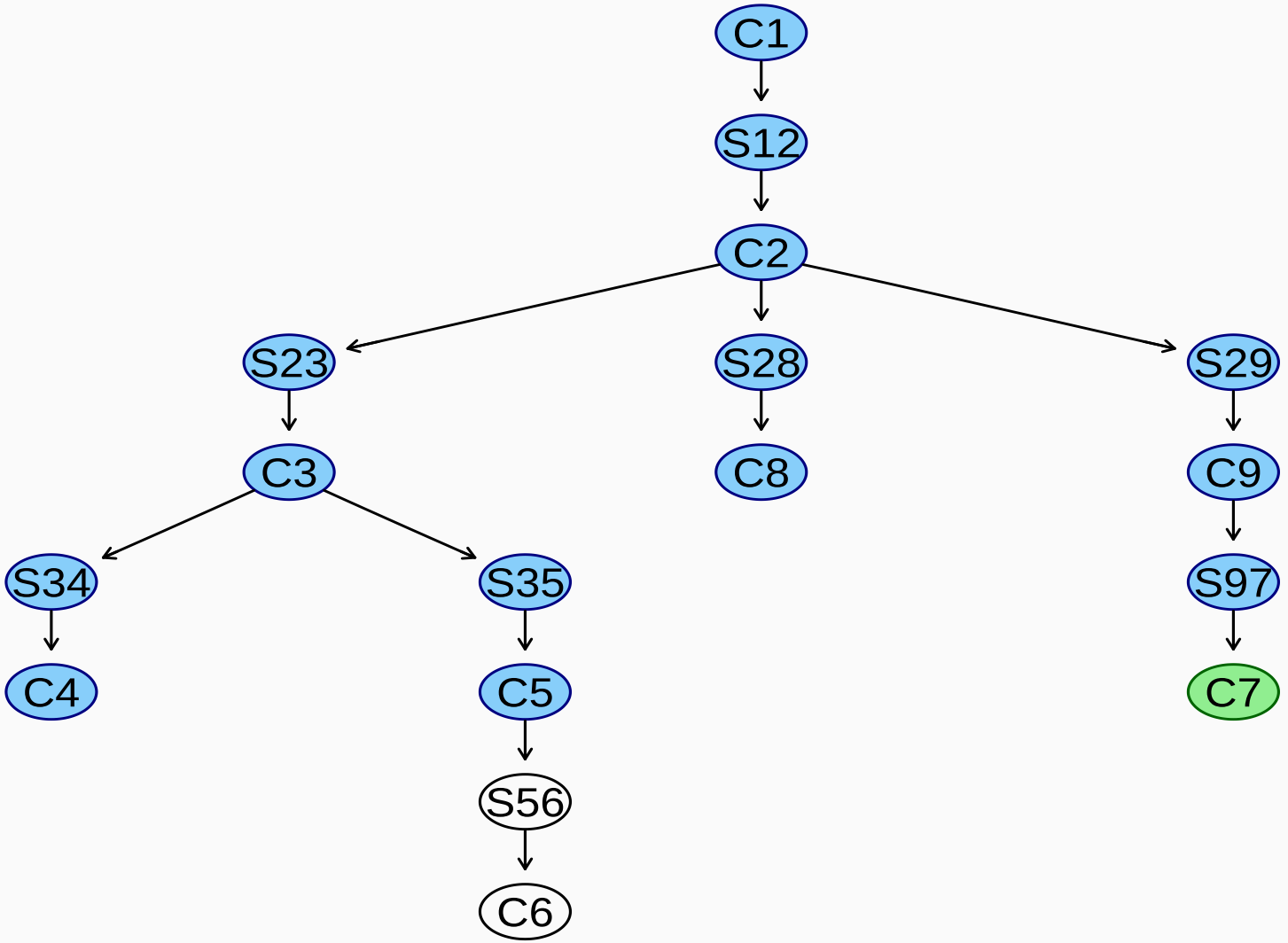
Update the joint distribution of Sun_0, Sun_1 in C_4 from the marginal distribution of Sun_1 in S_{34}.

Propagate the Evidence



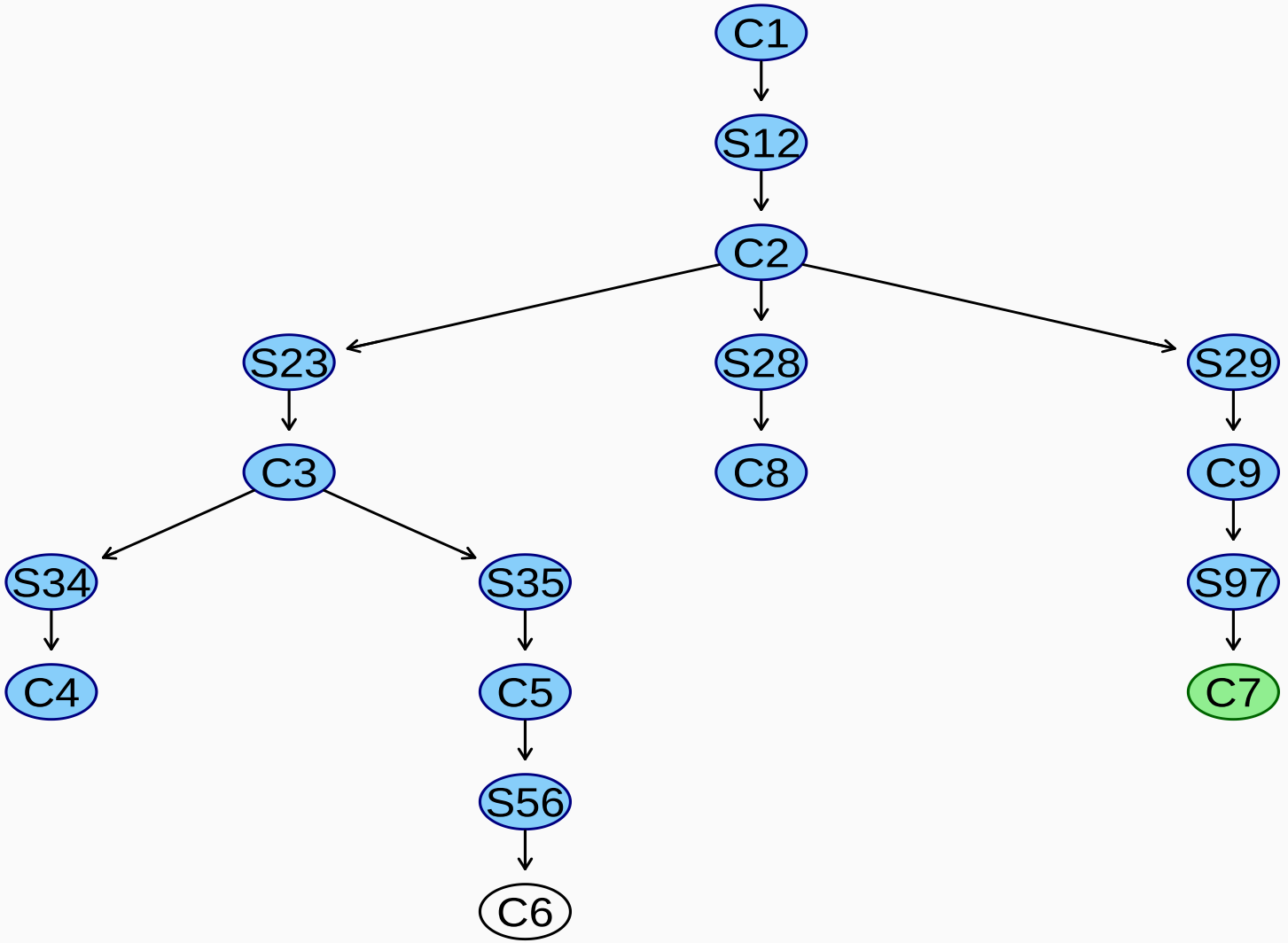
Update the marginal distribution of Sun_1, Tin_0 in S_{35} from the joint distribution of W_0, Sun_1, Tin_0 in C_3.

Propagate the Evidence



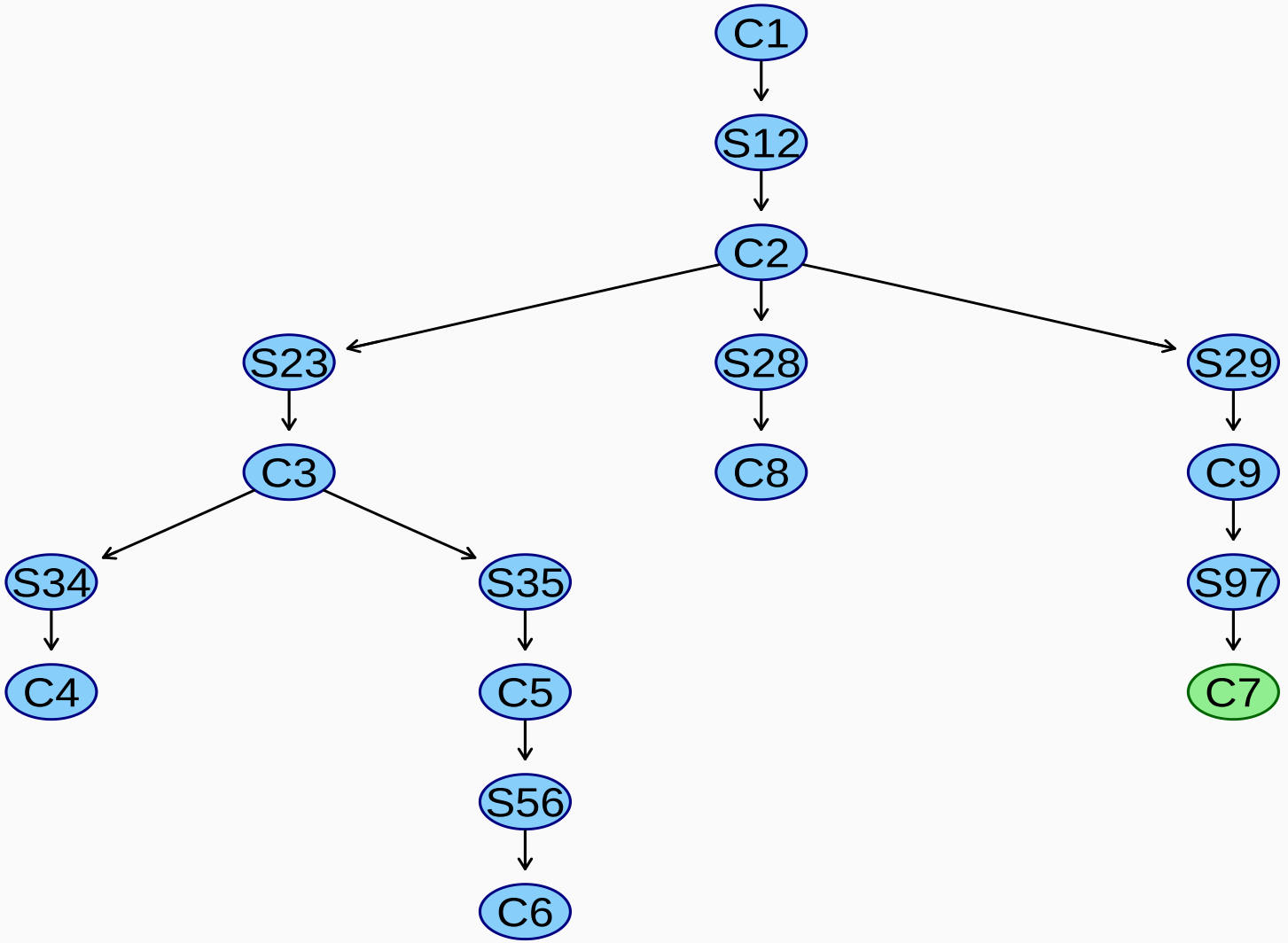
Update the joint distribution of Sun_1, Tin_0, Tin_1 in C_5 from the marginal distribution of Sun_1, Tin_0 in S_{35}.

Propagate the Evidence



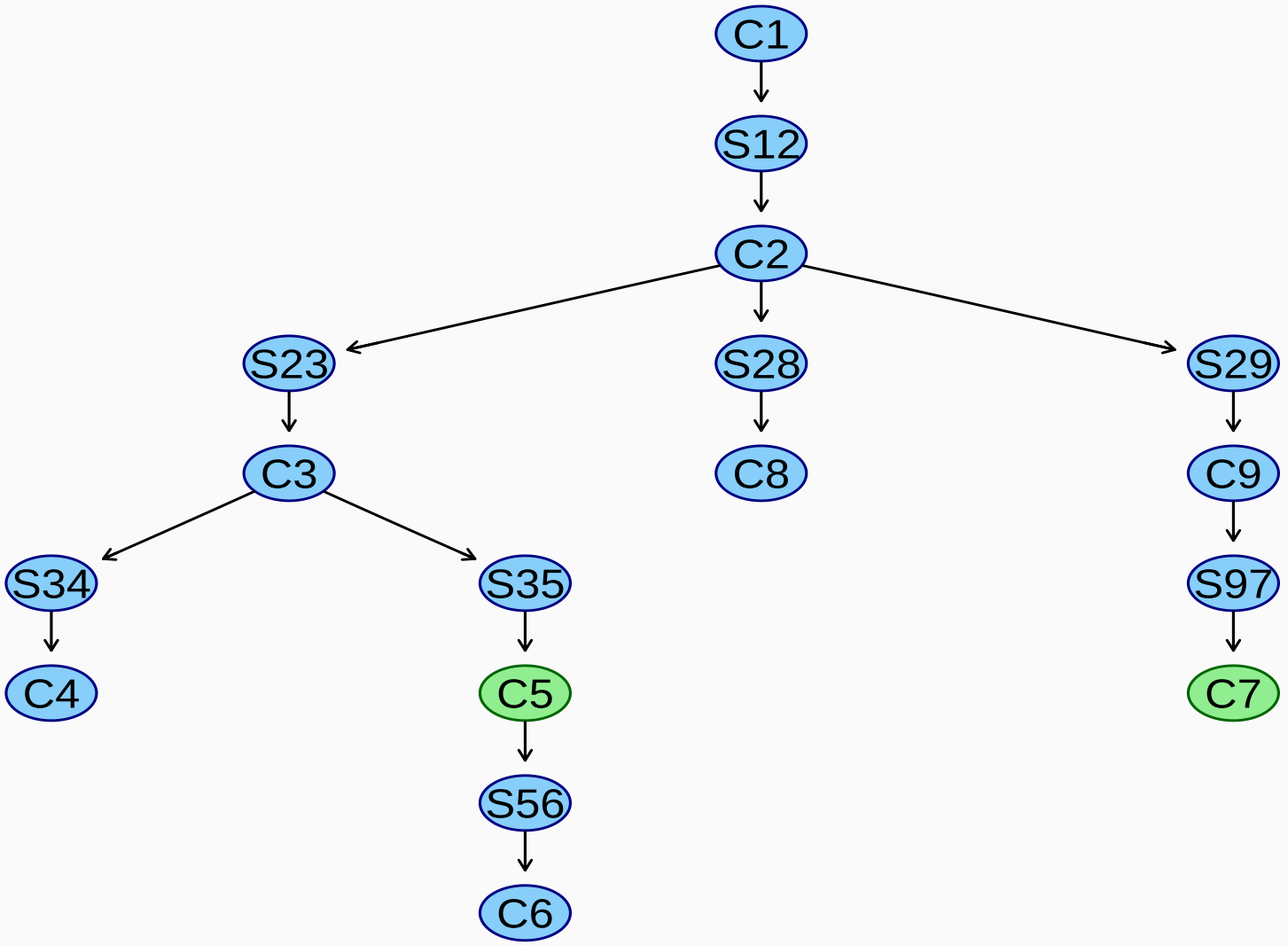
Update the marginal distribution of `Tin_0`, `Tin_1` in S_{56} from the joint distribution of Sun_1, `Tin_0`, `Tin_1` in C_5.

Propagate the Evidence



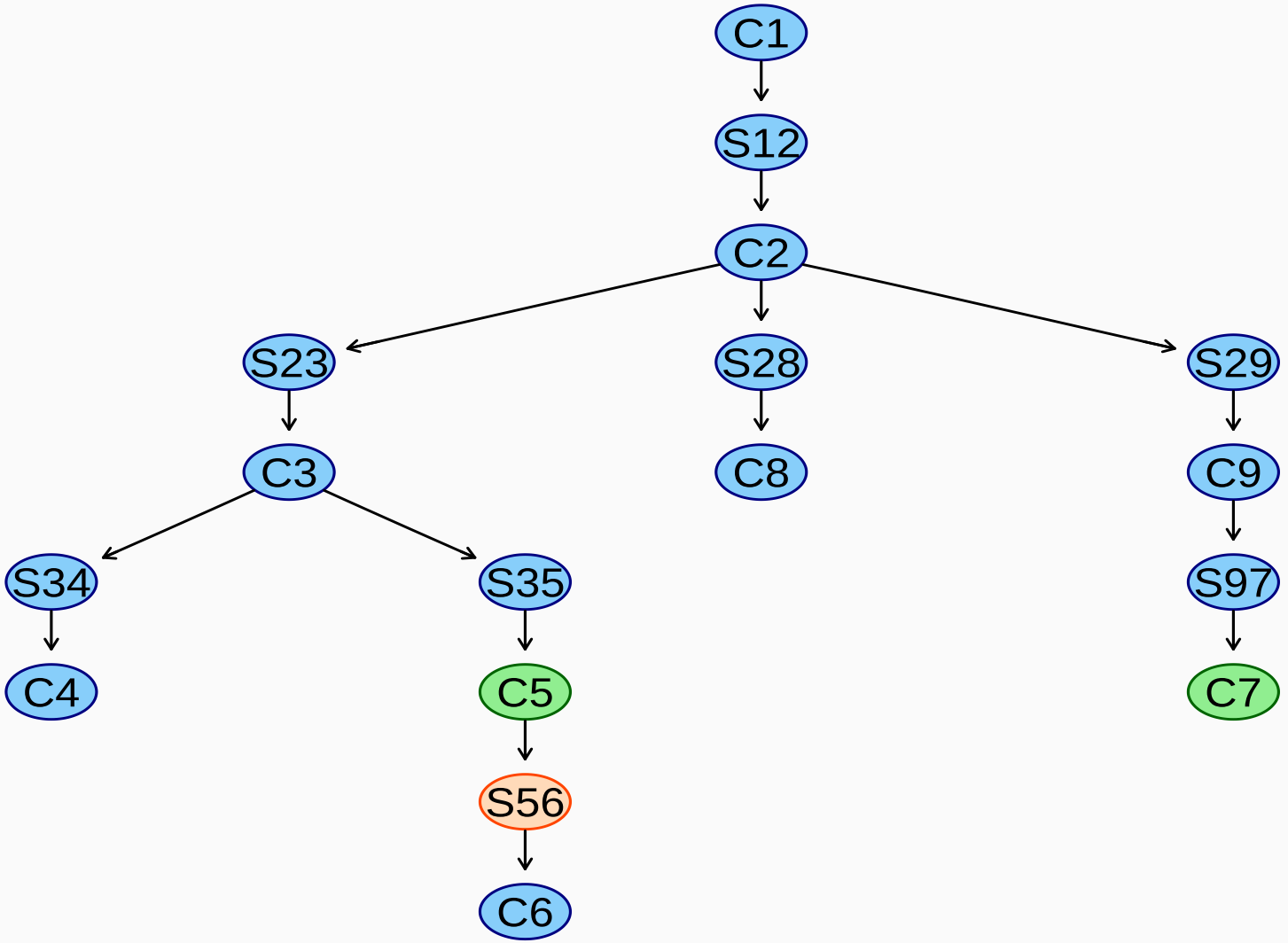
Update the joint distribution of `Tin_0`, `Tin_1`, `Tout_1` in C_6 from the marginal distribution of `Tin_0`, `Tin_1` in S_{56}.

Set the Evidence



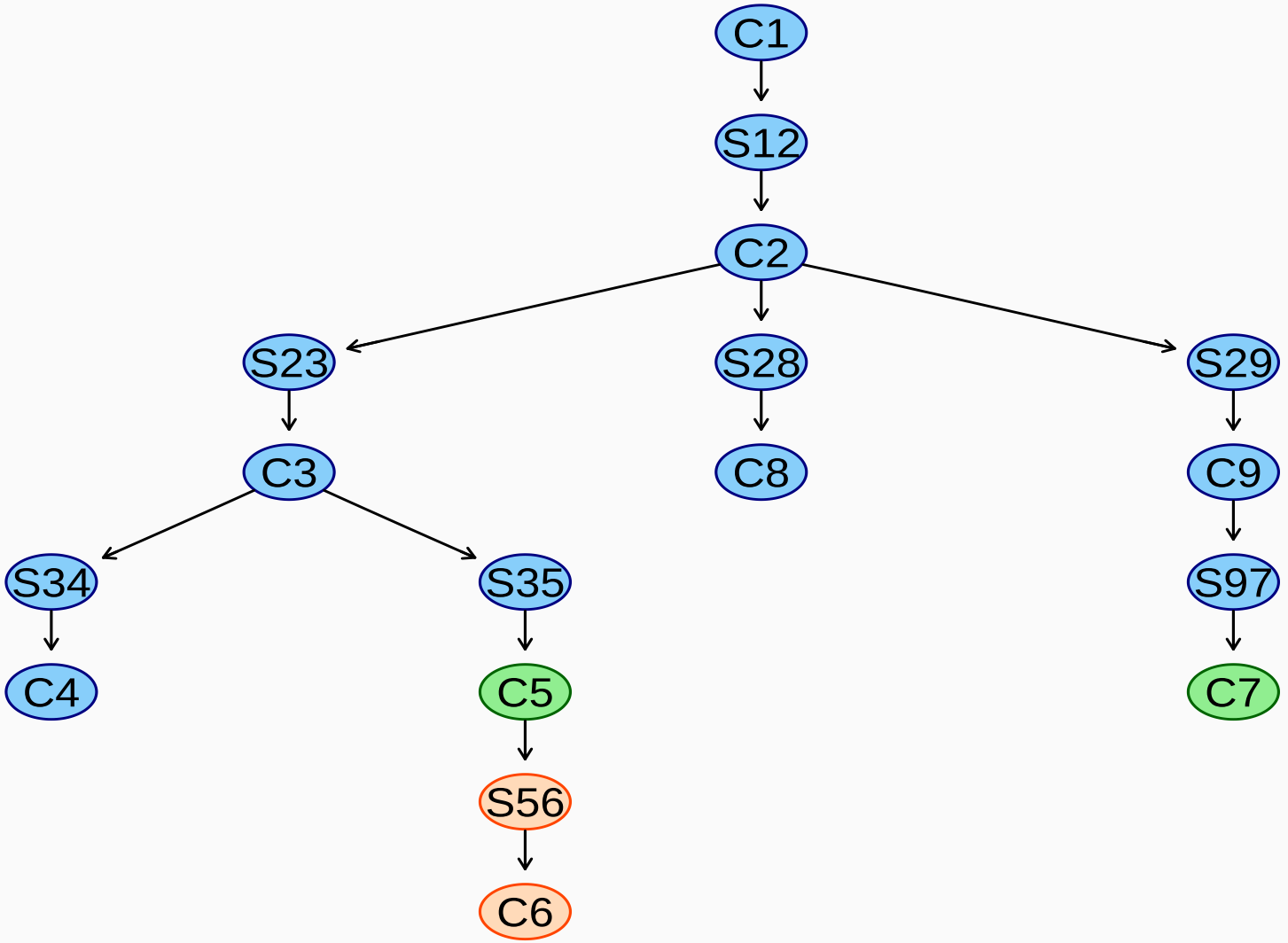
Set the second piece of evidence (`Tin_0` is equal to `18-24`) in C_5.

Propagate the Evidence



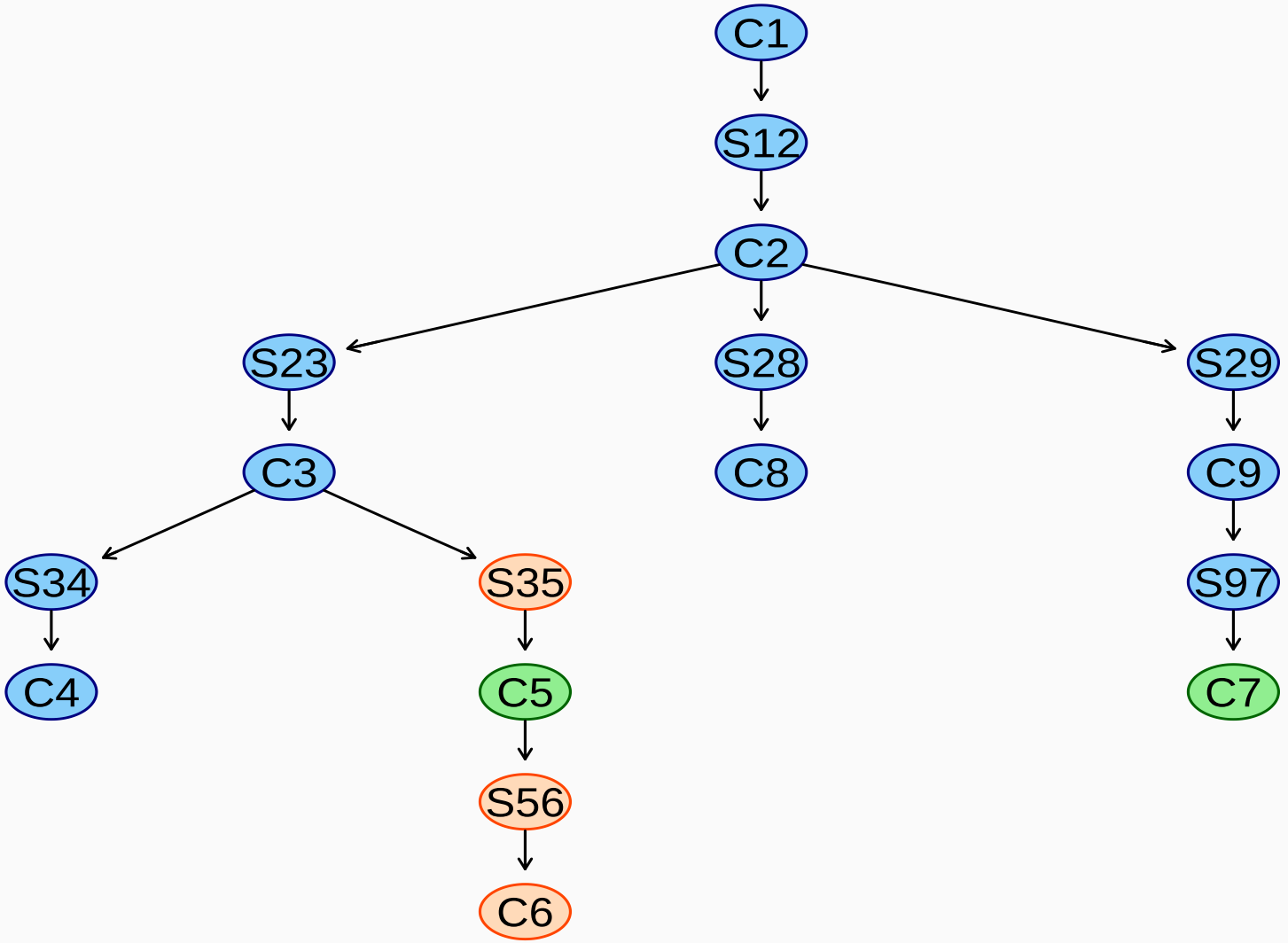
Update the marginal distribution of `Tin_0`, `Tin_1` in S_{56} from the joint distribution of Sun_1, `Tin_0`, `Tin_1` in C_5.

Propagate the Evidence



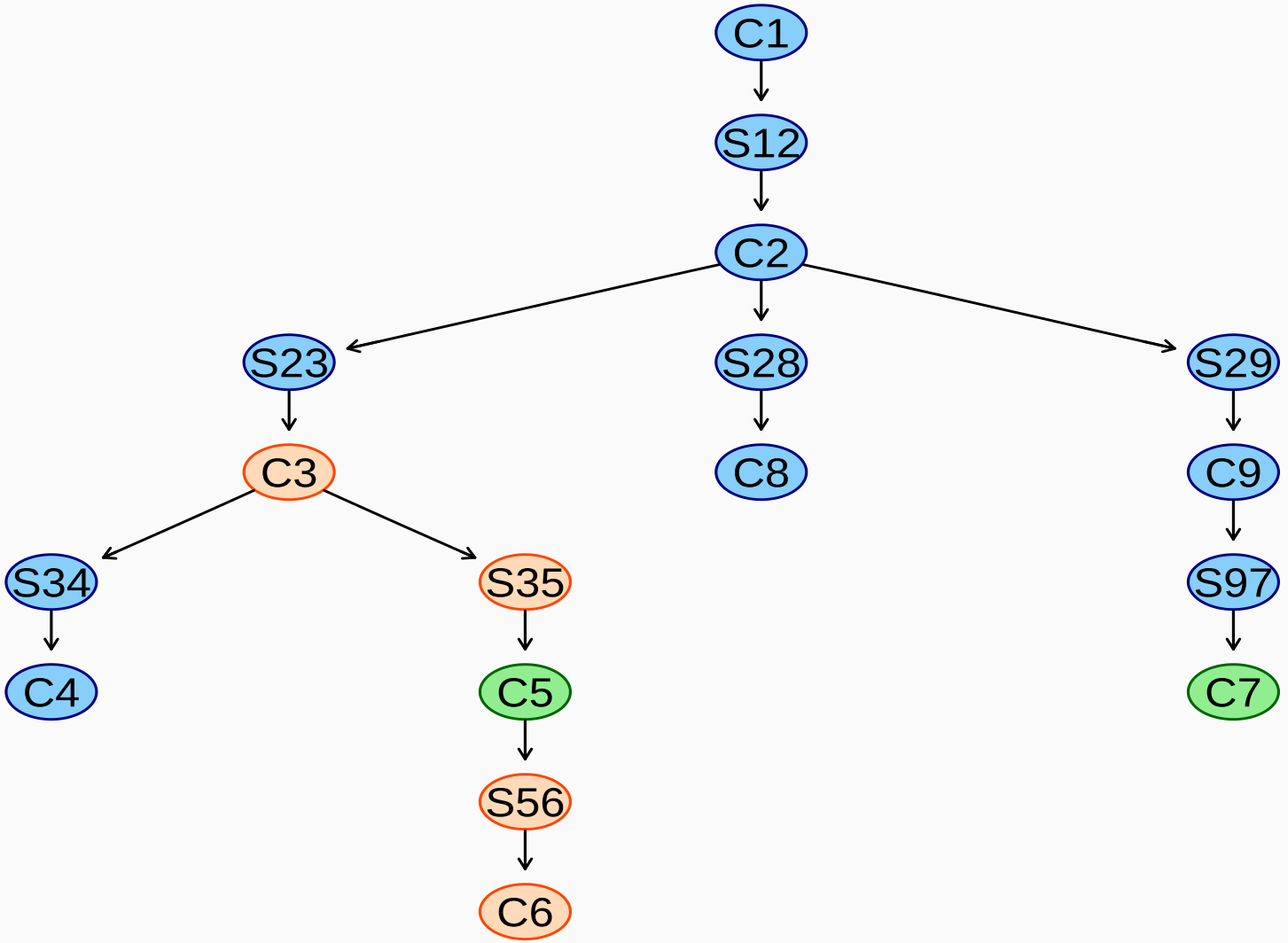
Update the joint distribution of `Tin_0`, `Tin_1`, `Tout_1` in C_6 from the marginal distribution of `Tin_0`, `Tin_1` in S_{56}.

Propagate the Evidence



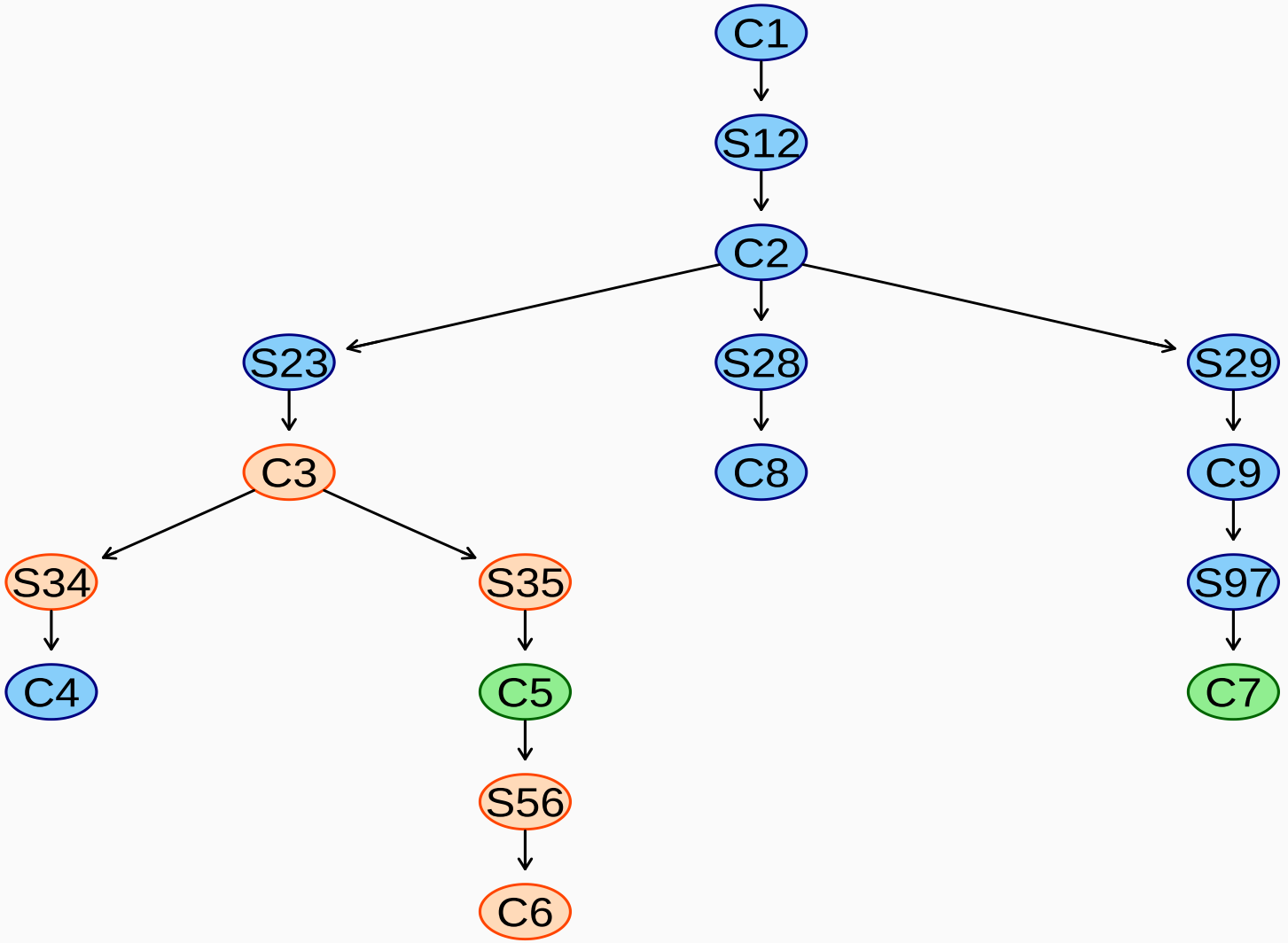
Update the marginal distribution of of Sun_1, Tin_0 in S_{35} from the joint distribution of Sun_1, Tin_0, Tin_1 in C_5.

Propagate the Evidence



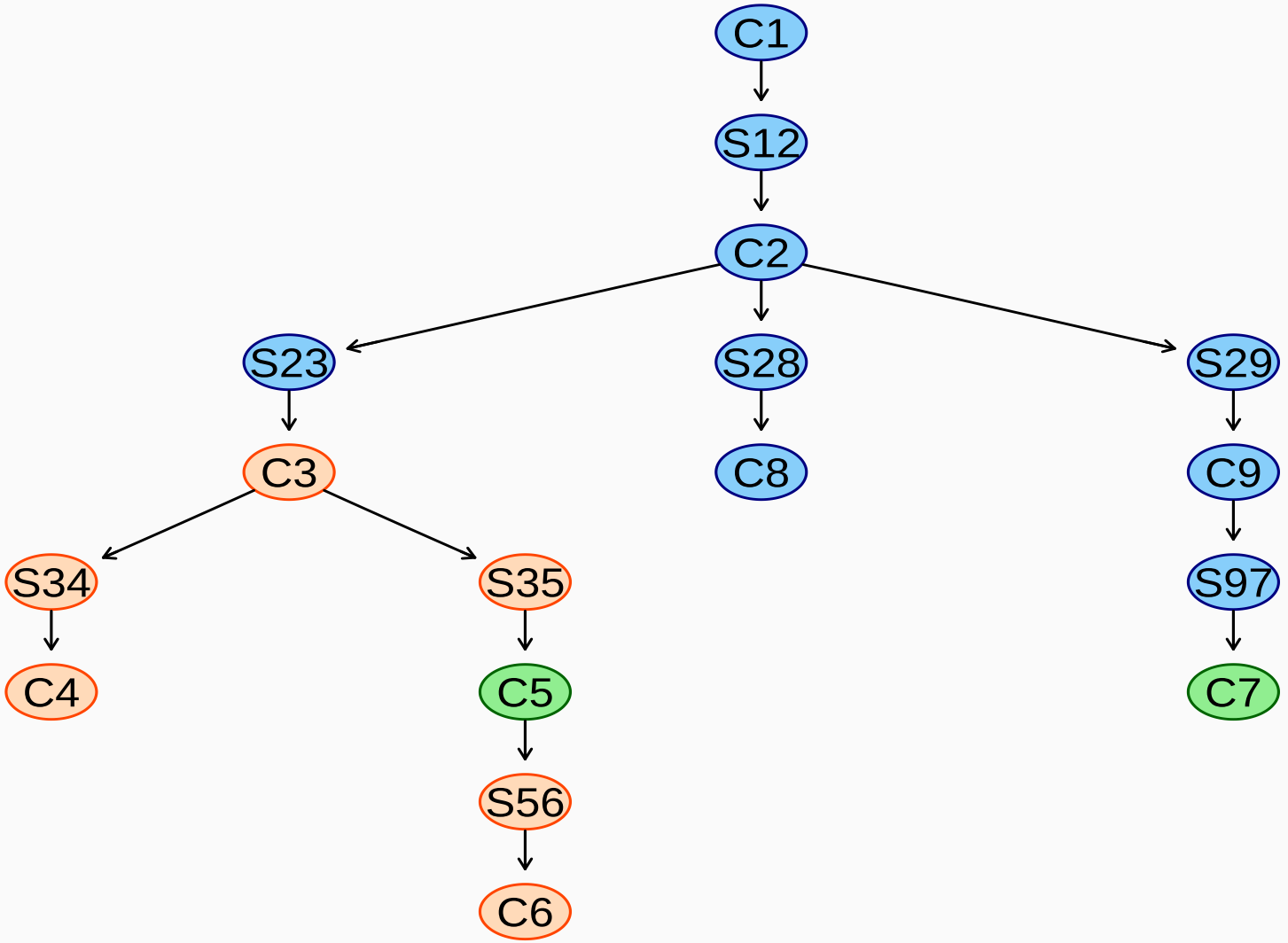
Update the joint distribution of `W_0`, `Sun_1`, `Tin_0` in C_3 from the marginal distribution of of `Sun_1`, `Tin_0` in S_{35}.

Propagate the Evidence



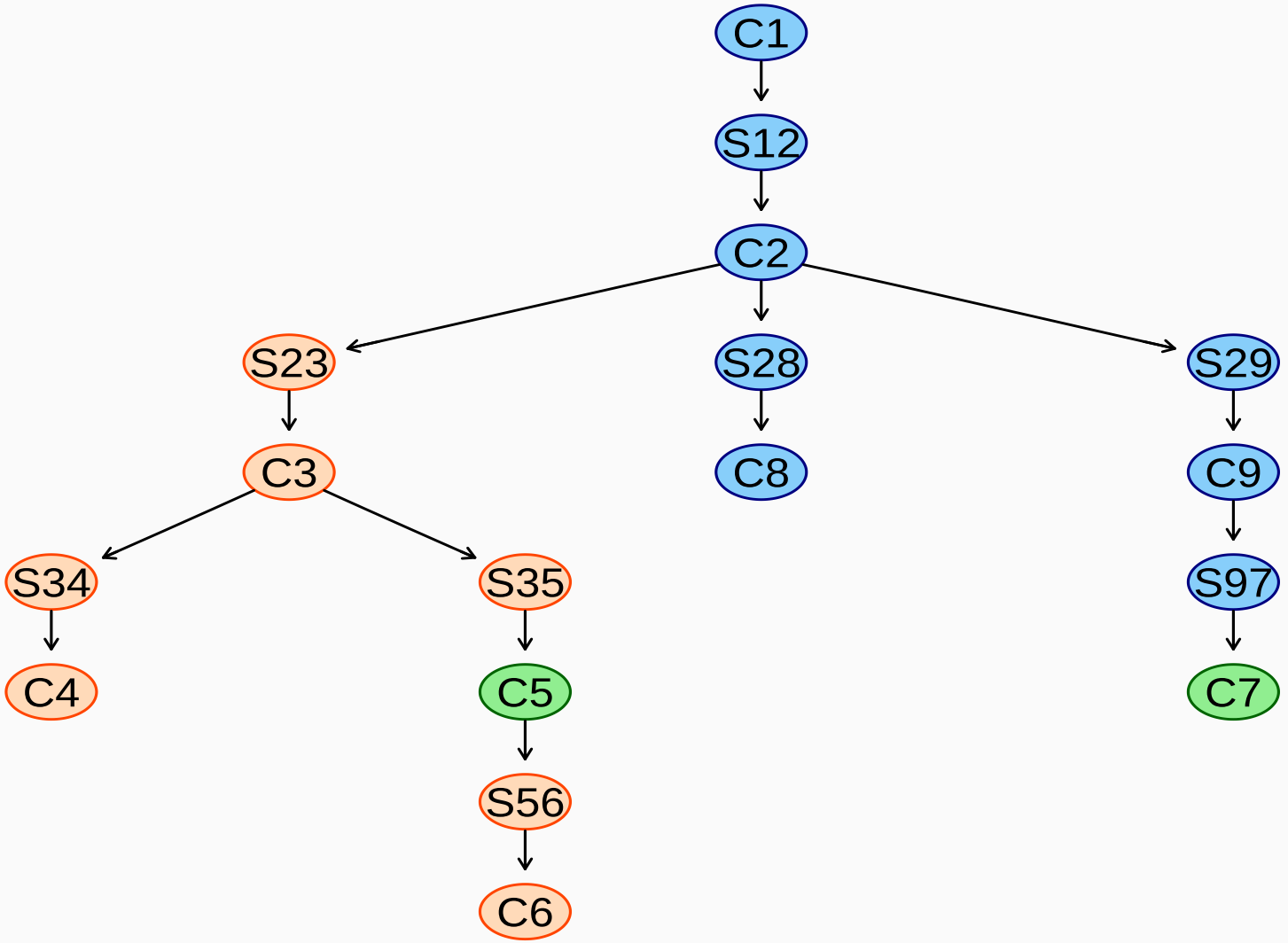
Update the marginal distribution of Sun_1 in S_{34} from the joint distribution of W_0, Sun_1, Tin_0 in C_3.

Propagate the Evidence



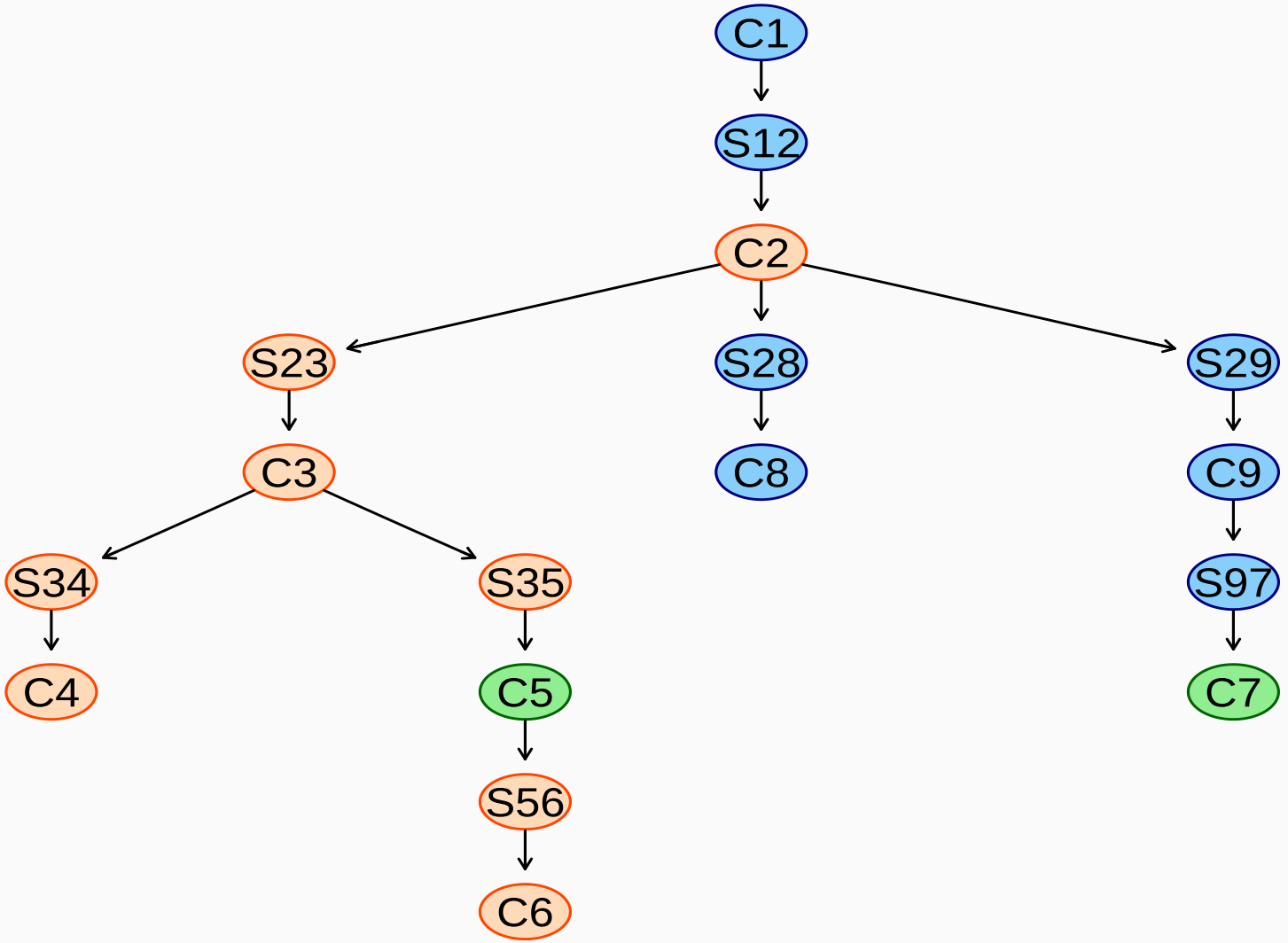
Update the joint distribution of Sun_0, Sun_1 in C_4 from the marginal distribution of Sun_1 in S_{34}.

Propagate the Evidence



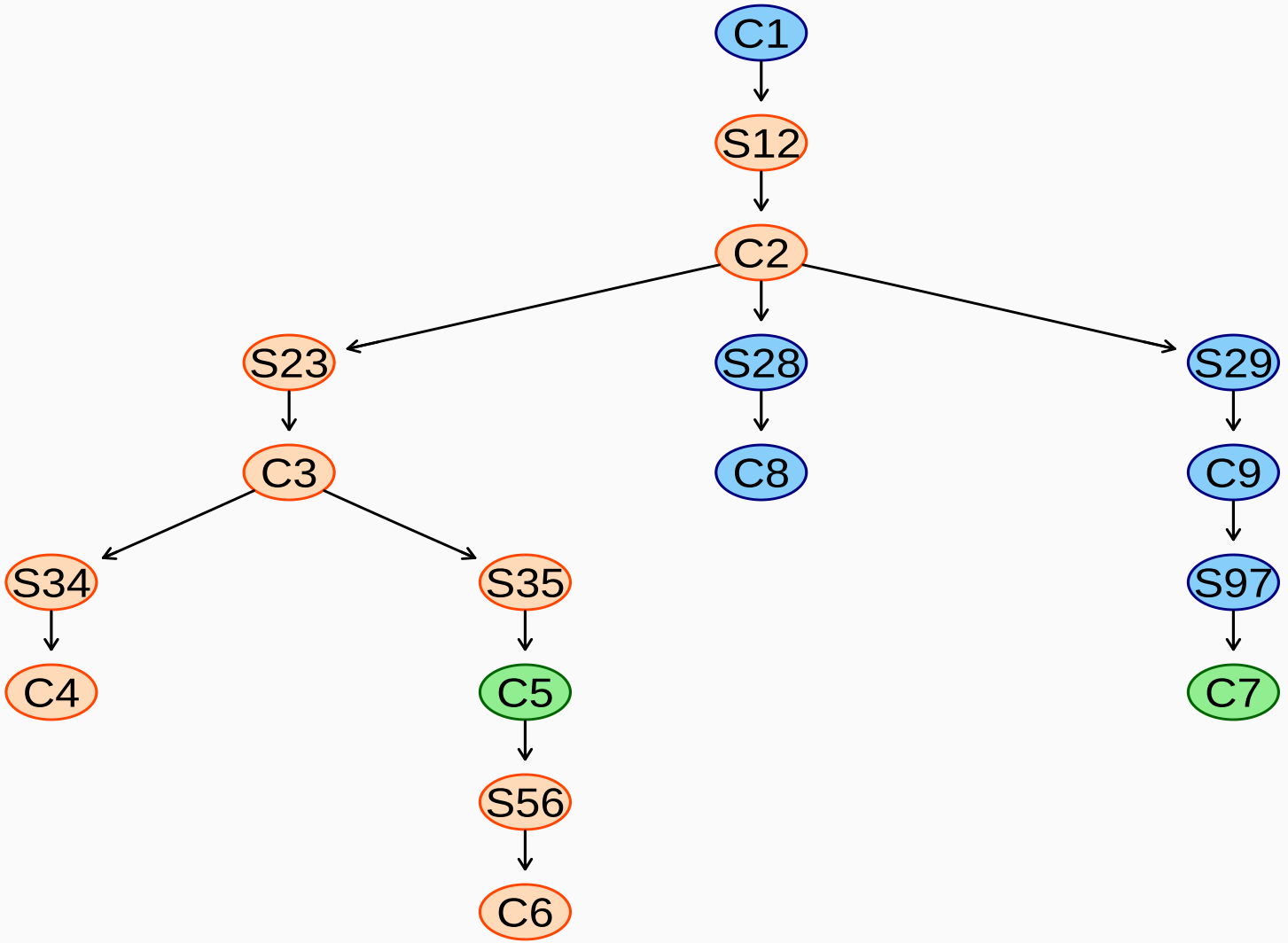
Update the marginal distribution of `W_0` in S_{23} from the joint distribution of `W_0`, `Sun_1`, `Tin_0` in C_3.

Propagate the Evidence



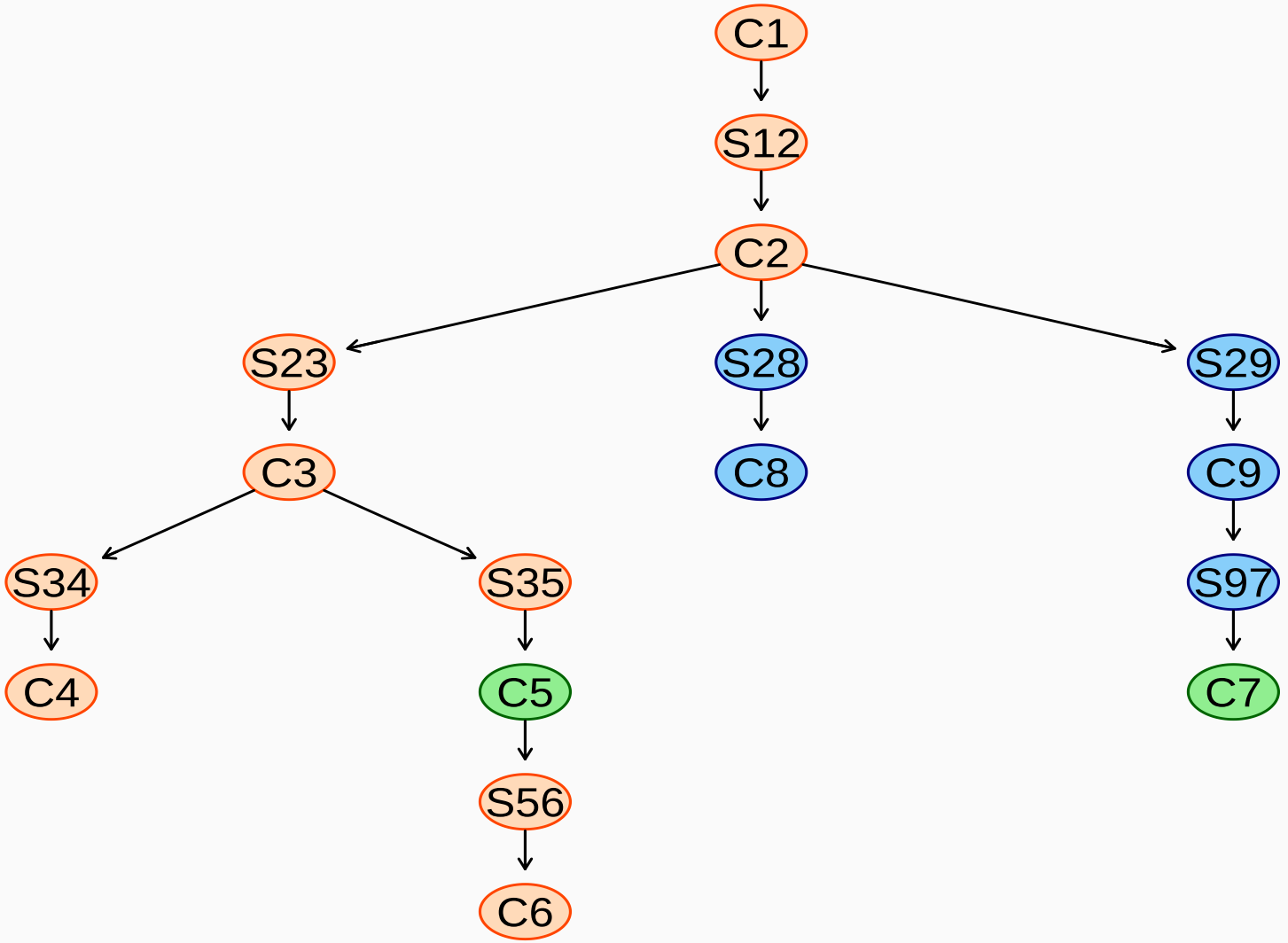
Update the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2 from the marginal distribution of `W_0` in S_{23}.

Propagate the Evidence



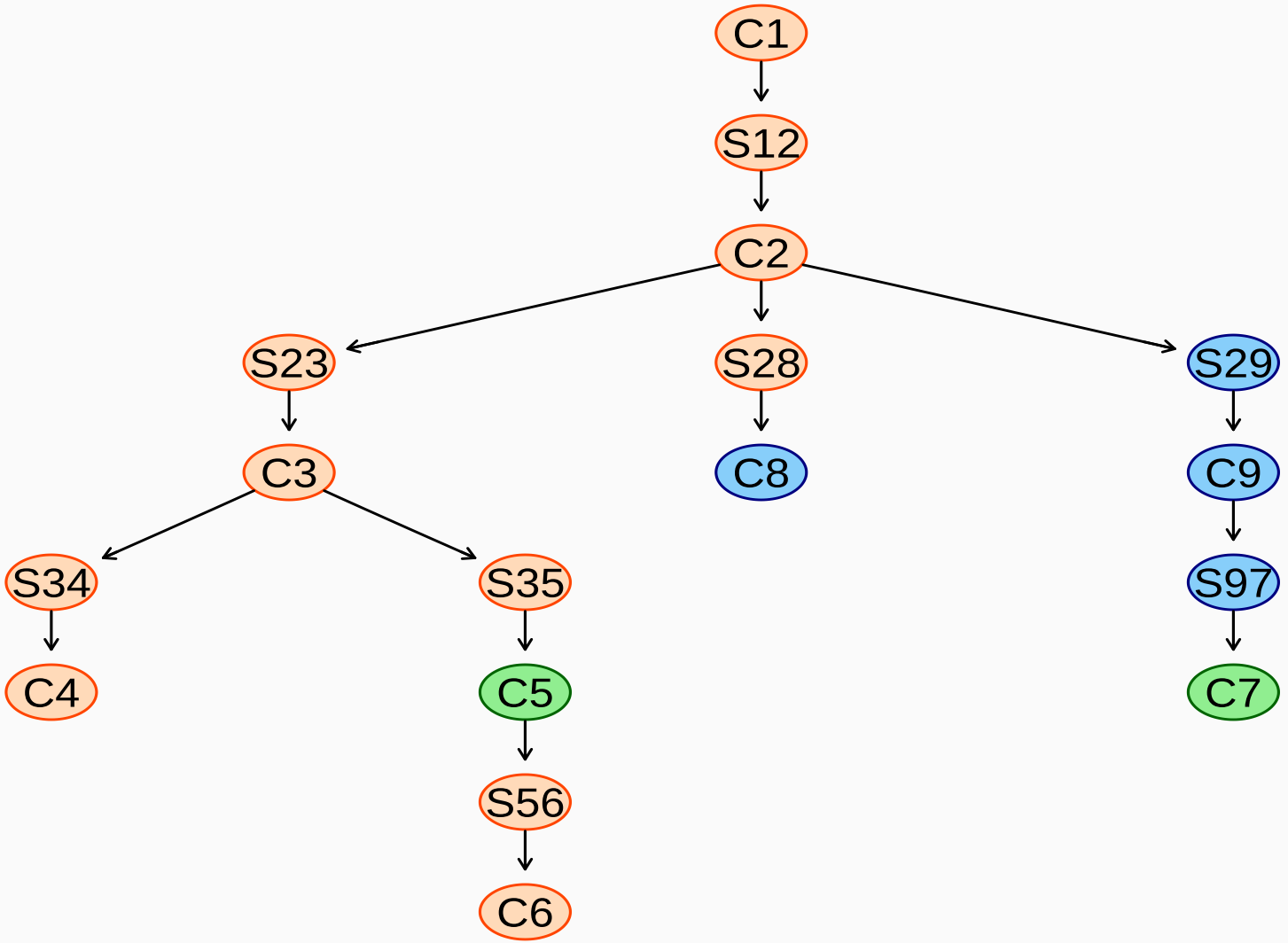
Update the marginal distribution of `W_0` in S_{12} from the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2.

Propagate the Evidence



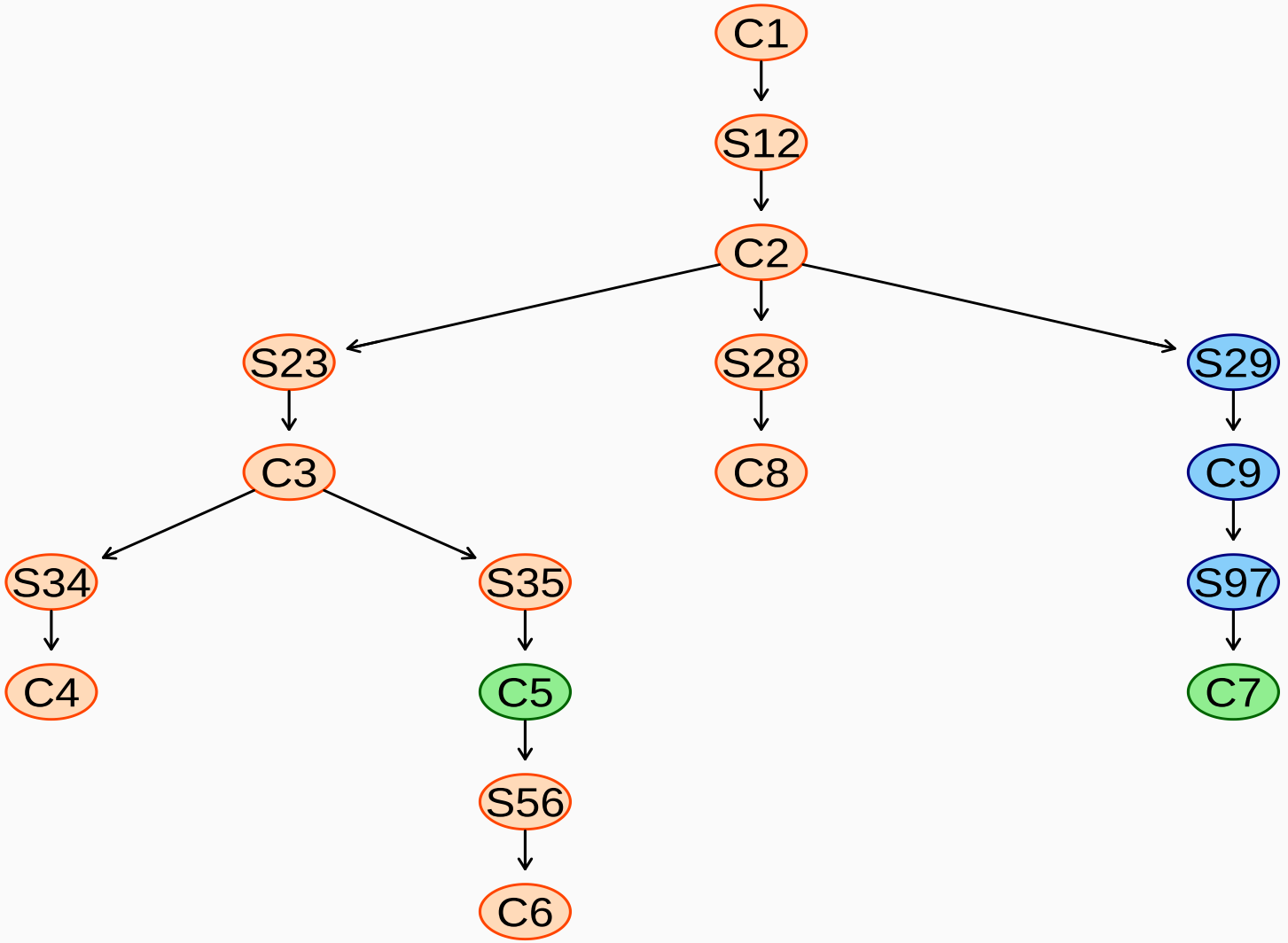
Update the joint distribution of `W_0`, `CO2_0`, `CO2_1` in C_1 from the marginal distribution of `W_0` in S_{12}.

Propagate the Evidence



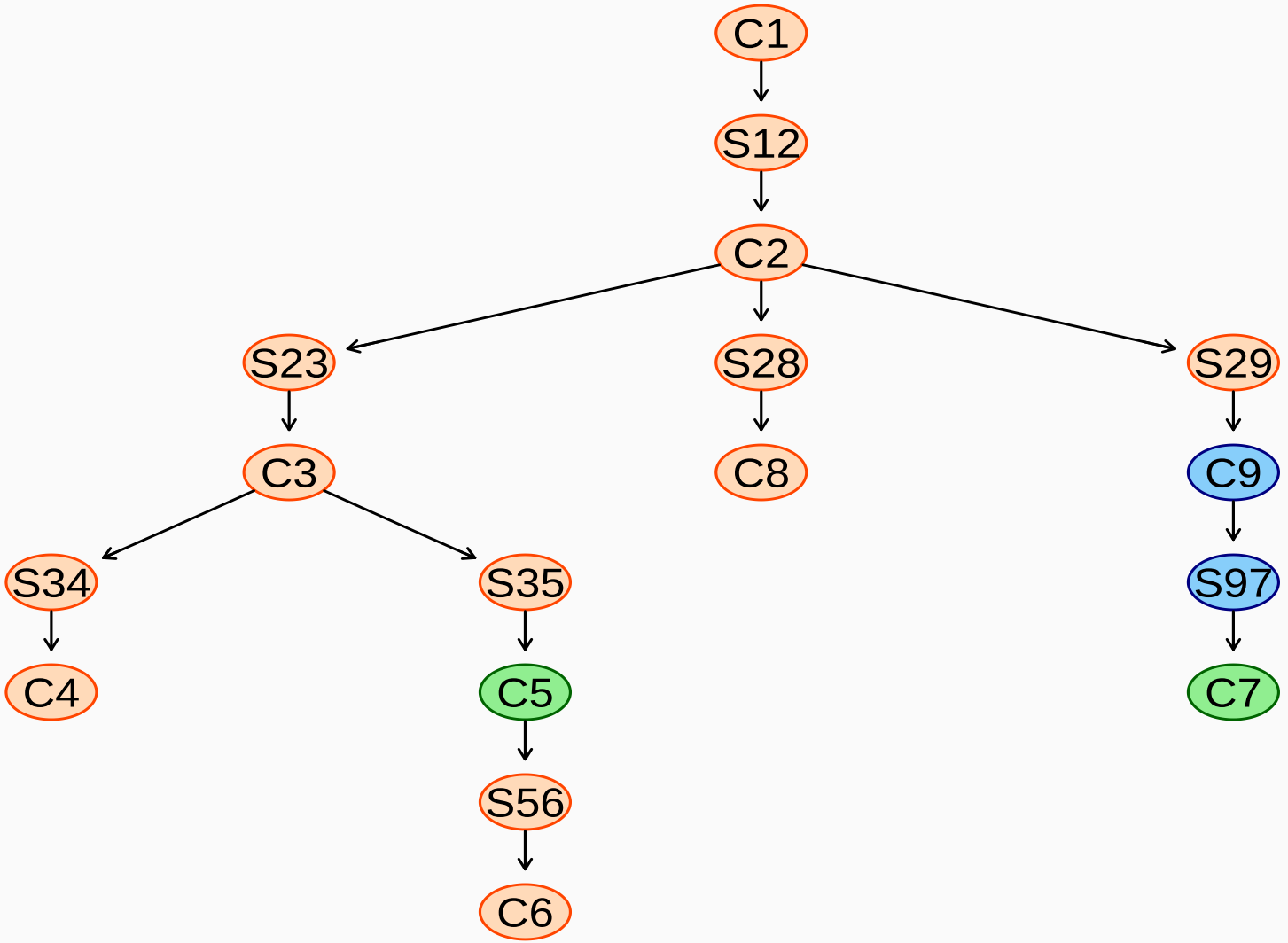
Update the marginal distribution of `W_0` in S_{28} from the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2.

Propagate the Evidence



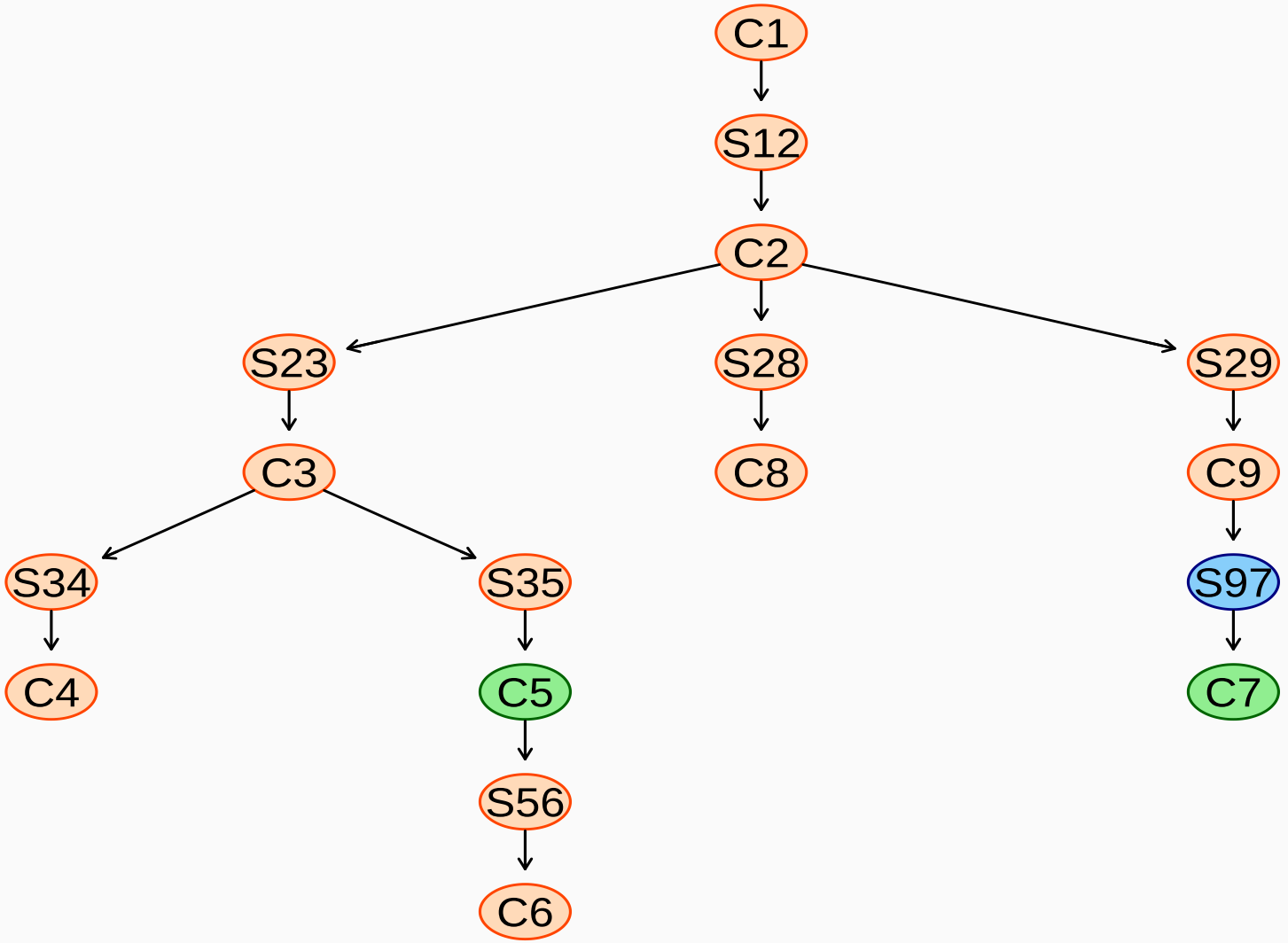
Update the joint distribution of `W_0`, `Tout_1`, `Tin_0` in C_8 from the marginal distribution of `W_0` in S_{28}.

Propagate the Evidence



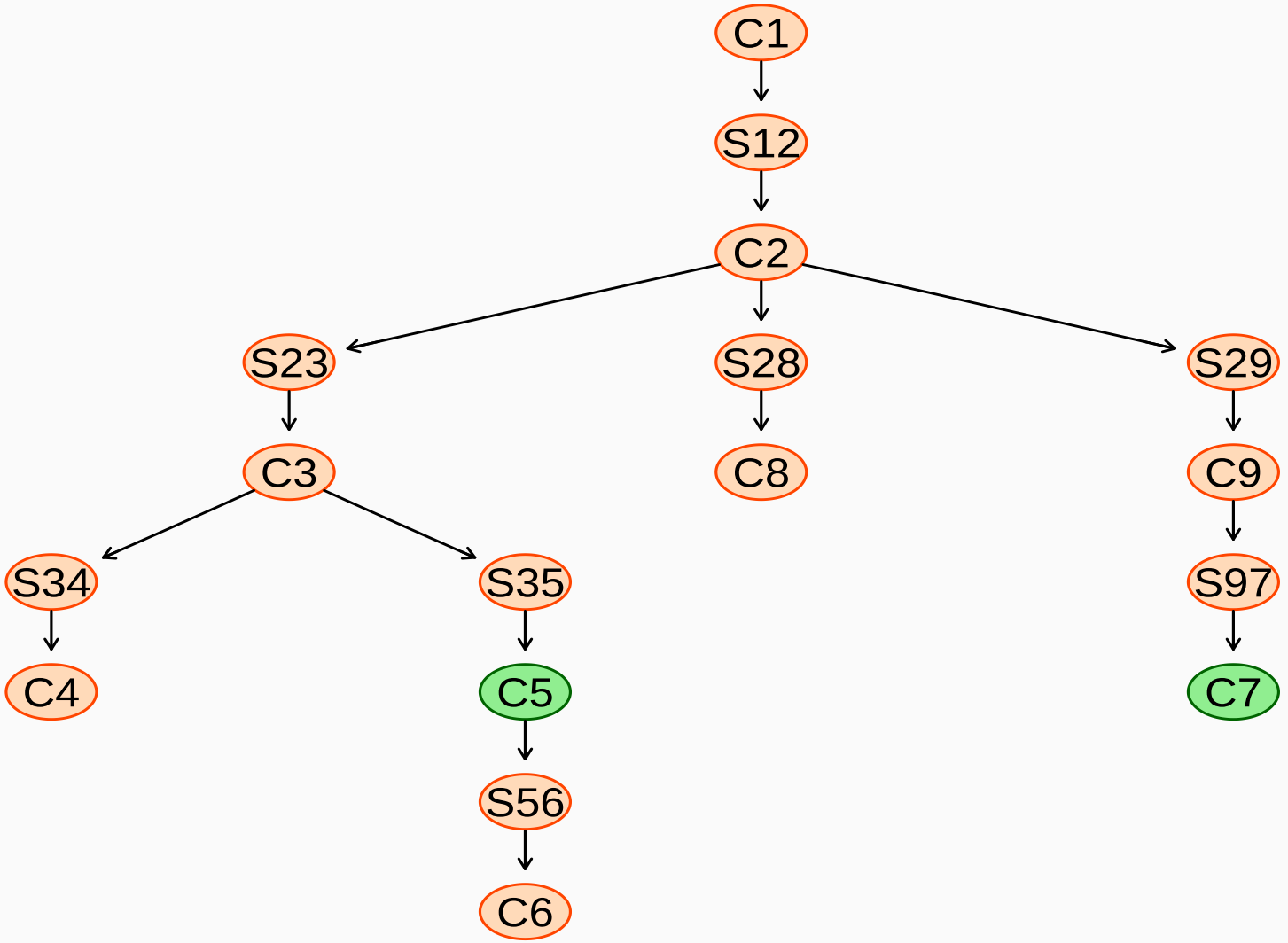
Update the marginal distribution of `W_0` in S_{29} from the joint distribution of `W_0`, `Hin_0`, `Hin_1` in C_2.

Propagate the Evidence



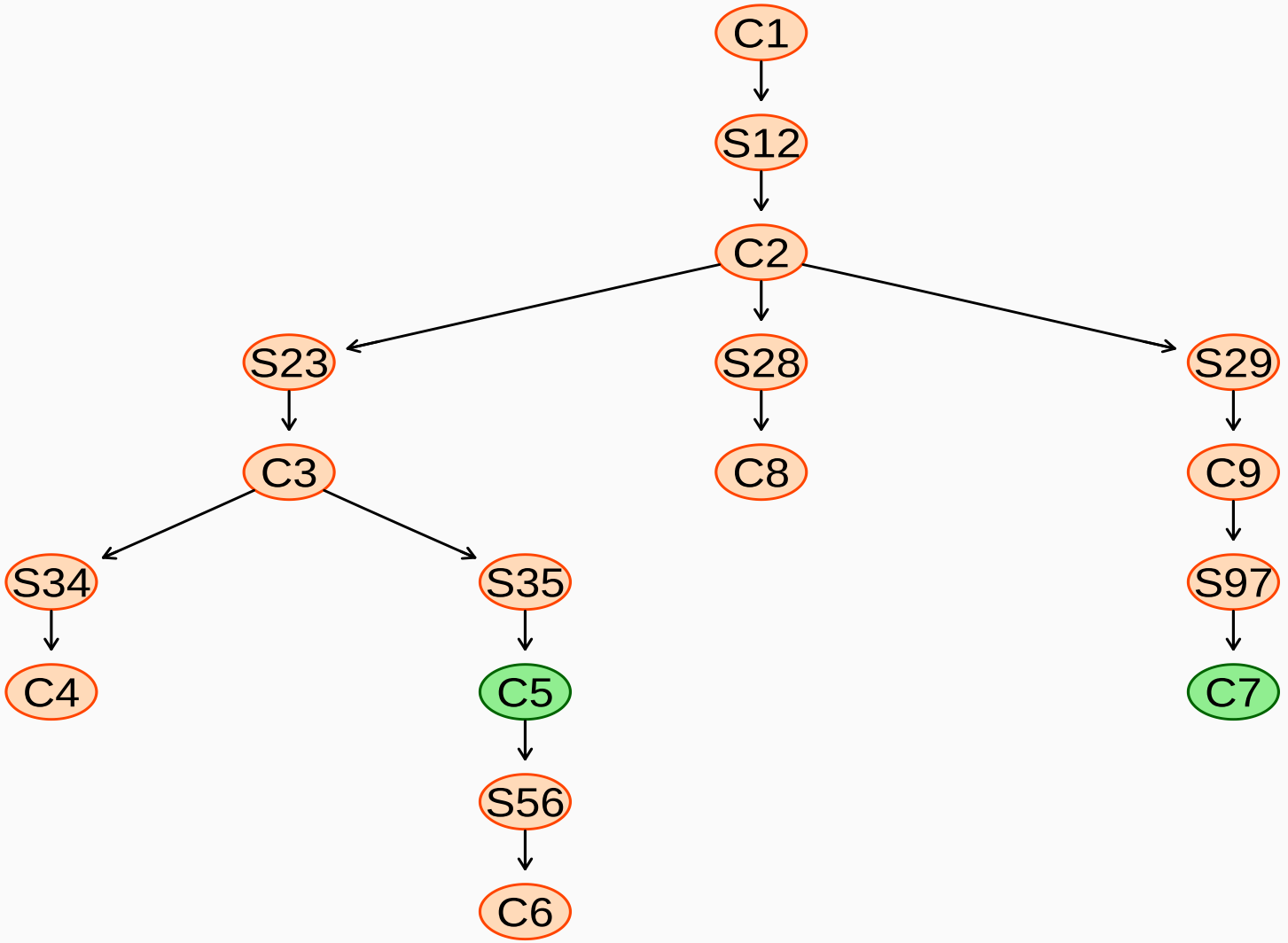
Update the joint distribution of `W_0`, `Tout_1`, `Tin_1` in C_9 from the marginal distribution of `W_0` in S_{29}.

Propagate the Evidence



Update the marginal distribution of `Tout_1` in S_{97} from the joint distribution of `W_0`, `Tout_1`, `Tin_1` in S_{29}.

Propagate the Evidence



Update the joint distribution of `W_0`, `CO2_0`, `CO2_1` in C_7 from the marginal distribution of `Tout_1` in S_{97}.

Junction Trees with gRain

All this is done for us by the function in the **gRain** package:

1. build the junction tree;

```
junction = compile(as.grain(win.dbn))
```

2. set the evidence and propagate it;

```
updated = setEvidence(junction, nodes = c("Tin_0", "Tout_0"),
             states = c(">24", "<18"))
```

3. compute the conditional probability.

```
querygrain(updated, nodes = "Tin_1", type = "joint")
```

```
Tin_1
   <18  18-24    >24
0.0632 0.3995 0.5373
```

The probability of 0.4 agrees with the approximate inference we performed earlier.

- `as.grain()` to export a fitted BN from **bnlearn** to **gRain**.
- `setEvidence()` and `querygrain()` in **gRain** to perform exact inference with junction trees.
- `rbn()` to generate random samples from a BN.
- `cpdist()` to generate random samples from a BN conditional on some evidence.
- `cpquery()` to perform approximate inference with logic sampling and likelihood weighting.

Summary and Remarks

1. Models in machine learning must be able to decide whether to perform particular actions given evidence on the surrounding environment.
2. The basis of these decisions are the predictions and the conditional probabilities computed after incorporating evidence into the model.
3. In the context of BNs computing these probability is called inference.
4. There are two classes of algorithms to perform inference: approximate and exact algorithms.
5. Approximate algorithms are based on generating random samples to simulate real-world experiments.
6. Exact algorithms automate the mathematical steps we would perform to manipulate the probabilities in the model.

Learning the Parameters

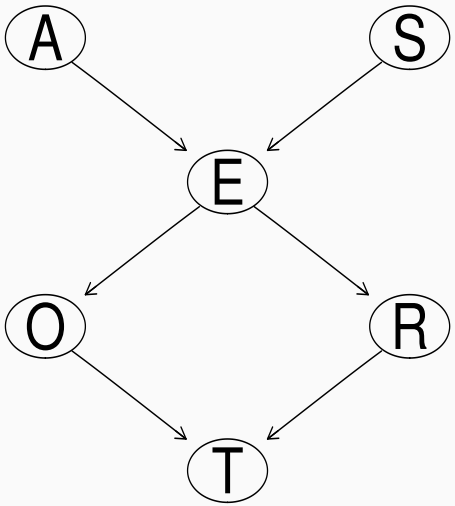
Learning Probabilities from Data

What we have:

- a BN structure, that is, a directed acyclic graph;
- a data set covering the variables in the BN.

What we want:

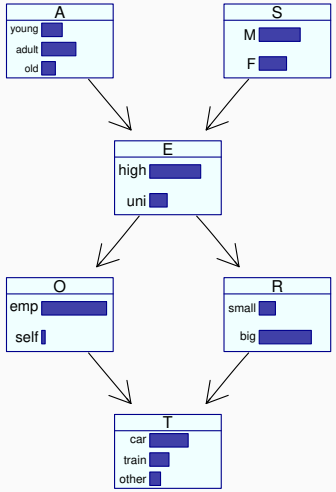
- the probabilities we need to fill the conditional probability tables for all the nodes.



+

	A	R	E	O	S	T
1	adult	big	high	emp	F	car
2	adult	small	uni	emp	M	car
3	adult	big	uni	emp	F	train
4	adult	big	high	emp	M	car
5	adult	big	high	emp	M	car
6	adult	small	high	emp	F	train

=



Network Structure, Data and Parameters

For each node X_i:

1. We identify the parents Π_{X_i} of the node X_i from the network structure.
2. We collect the variables X_i, Π_{X_i} from the data.
3. We construct a table of counts n_{ikj} for each value k of X_i for each possible configuration of the values of Π_{X_i}.
4. We estimate the conditional probabilities $p_{ik|j} = \mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j)$ from the n_{ikj}.

Two ways of estimating probabilities are:

- frequentist probability estimates; and
- Bayesian probability estimates.

Frequentist Probability Estimates

In the frequentist paradigm, probability is the relative frequency with which an event occurs over a large number of observations. Hence:

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{\mathrm{P}(X_i = x_k, \Pi_{X_i} = \pi_j)}{\mathrm{P}(\Pi_{X_i} = \pi_j)}$$

$$\mathrm{P}(X_i = x_k, \Pi_{X_i} = \pi_j) = \frac{n_{ikj}}{n}$$

$$\mathrm{P}(\Pi_{X_i} = \pi_j) = \frac{n_{i+j}}{n} = \frac{\sum_k n_{ikj}}{n}$$

which means

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{n_{ikj}}{\cancel{n}} \frac{\cancel{n}}{\sum_k n_{ikj}} = \frac{n_{ikj}}{\sum_k n_{ikj}}.$$

In tabular form:

		Π_{X_i}			
		π_1	π_2	$\cdots$	π_{q_i}
X_i	x_1	n_{i11}	n_{i12}	$\cdots$	n_{i1q_i}
	x_2	n_{i21}	n_{i22}	$\cdots$	n_{i2q_i}
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	x_{r_i}	n_{ir_i1}	n_{ir_i2}	$\cdots$	$n_{ir_iq_i}$
		n_{i+1}	n_{i+2}	$\cdots$	n_{i+q_i}

with $n_{i+1} = \sum_k n_{ik1}$, $n_{i+2} = \sum_k n_{ik2}$, etc. leading to

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{n_{ikj}}{n_{i+j}} = \frac{\text{cell for } x_k \text{ and } \pi_j}{\text{column total for } \pi_j}.$$

Frequentist Probability Estimates with Low Sample Sizes

Frequentist probability estimates only work well if lots of data are available. One reason is that their granularity (or their precision, if you like) depends on n_{i+k}. Given that

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{n_{ikj}}{n_{i+k}}$$

for a given n_{i+k} the possible estimates will necessarily come out of the set

$$\left\{ \frac{0}{n_{i+k}}, \quad \frac{1}{n_{i+k}}, \quad \frac{2}{n_{i+k}}, \quad \ldots, \quad \frac{n_{i+k}}{n_{i+k}} \right\}$$

with a possible estimation error due to this granularity of $\frac{1}{2n_{i+k}}$.

Consider: if the true value of the probability you are trying to estimate is 0.343 and you only have $n_{i+k} = 50$, with a $1/50 = 0.02$ granularity the closest you can get is 0.34.

Frequentist Probability Estimates with Low Sample Sizes

Another reason is that if $n_{i+k} = 0$, because the configuration π_k of the parents Π_{X_i} is not observed in the data, then

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{0}{0}$$

which gives you a NaN instead of a usable parameter estimate. If $n_{i+k} = 1$,

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{0 \text{ or } 1}{1}$$

which is not ideal because the true probability we are trying is not likely to be 0 (the event is impossible) or 1 (the event is certain).

More in general, if the sample size is small rare events are not likely to be observed and thus be deemed impossible by the model.

Frequentist Probability Estimates in `bn.fit()`

The function that performs parameter learning in **bnlearn** is `bn.fit()`, which takes a network structure (as a `bn` object) and a data set (as a `data.frame` with each column coded as a `factor`).

```
survey.data = read.table("survey.txt", stringsAsFactors = TRUE)
survey.fitted = bn.fit(survey.dag, survey.data, method = "mle")
```

Like `custom.fit()`, it returns a `bn.fit` object containing the complete BN with all the (conditional) probability tables. The `method` argument defaults to `mle` so it can be omitted when estimating frequentist probabilities.

Bayesian Probability Estimates

In the Bayesian paradigm, probabilities are a measure of our belief that an event will happen (or not) and are computed as

prior belief + evidence from data = posterior belief

by weighting prior belief appropriately against the evidence we collect from the data. From a mathematical point of view:

- the evidence from the data are the counts n_{ikj};
- we can imagine we have a second set of counts α_{ikj} from an imaginary sample that was collected previously and that is the source of our prior belief.
- the imaginary sample size determines how much weight we give to our prior beliefs compared to the information coming from the data.

In tabular form:

		Π_{X_i}			
		π_1	π_2	$\cdots$	π_{q_i}
X_i	x_1	$\alpha_{i11} + n_{i11}$	$\alpha_{i12} + n_{i12}$	$\cdots$	$\alpha_{i1q_i} + n_{i1q_i}$
	x_2	$\alpha_{i21} + n_{i21}$	$\alpha_{i22} + n_{i22}$	$\cdots$	$\alpha_{i2q_i} + n_{i2q_i}$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	x_{r_i}	$\alpha_{ir_i1} + n_{ir_i1}$	$\alpha_{ir_i2} + n_{ir_i2}$	$\cdots$	$\alpha_{ir_iq_i} + n_{ir_iq_i}$
		$\alpha_{i+1} + n_{i+1}$	$\alpha_{i+2} + n_{i+2}$	$\cdots$	$\alpha_{i+q_i} + n_{i+q_i}$

leading to the posterior probability

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{\alpha_{ikj} + n_{ikj}}{\alpha_{i+j} + n_{i+j}}.$$

The problem is now how to choose the α_{ikj}. There are as many prior probabilities as there are probabilities in the model!

If we do not have enough prior information to make a meaningful choice, we can always choose the uniform distribution in which

$$\alpha_{ikj} = \frac{\alpha}{r_i q_i} \qquad \text{for all } j \text{ and } k \text{ in each } X_i.$$

- This is the distribution that has the maximum possible entropy, which implies that it carries the least amount of information. All events are equally likely because we do not know any better.
- The imaginary sample size is trivially $\alpha = \sum_k \sum_j \alpha_{ikj}$ for all X_i.

Choosing the Imaginary Sample Size

This leaves only a single number to choose: α. How large it is determines how much weight we give to our prior beliefs compared to the information contained in the data:

$$
\begin{aligned}
\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) &= \frac{\alpha_{ikj} + n_{ikj}}{\alpha_{i+j} + n_{i+j}} \\
&= \underbrace{\frac{\alpha_{ikj}}{\alpha_{i+j} + n_{i+j}}}_{\text{prior}} + \underbrace{\frac{n_{ikj}}{\alpha_{i+j} + n_{i+j}}}_{\text{data}} \\
&= \underbrace{\frac{\alpha_{i+j}}{\alpha_{i+j} + n_{i+j}}}_{\text{weight}} \cdot \underbrace{\frac{\alpha_{ikj}}{\alpha_{i+j}}}_{\text{prior}} + \underbrace{\frac{n_{i+j}}{\alpha_{i+j} + n_{i+j}}}_{\text{1 - weight}} \cdot \underbrace{\frac{n_{ikj}}{n_{i+j}}}_{\text{data}} \\
&= w \frac{\alpha_{ikj}}{\alpha_{i+j}} + (1 - w) \frac{n_{ikj}}{n_{i+j}}, w \in (0, 1).
\end{aligned}
$$

Choosing the Imaginary Sample Size

The expression of the posterior probability

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = w \frac{\alpha_{ikj}}{\alpha_{i+j}} + (1 - w)\frac{n_{ikj}}{n_{i+j}}$$

is made up by:

- the frequentist probability estimate $\frac{n_{ikj}}{n_{i+j}}$, which is based on the information present in the data;
- the corresponding prior probability, which in the case of the uniform prior simplifies to

$$\frac{\alpha_{ikj}}{\alpha_{i+j}} = \frac{\alpha}{r_i q_i} \cdot \frac{r_i}{\alpha} = \frac{1}{q_i} \qquad \text{for all probabilities.}$$

- a weight $w = \frac{\alpha_{i+j}}{\alpha_{i+j}+n_{i+j}}$ that is determined by the ratio of the imaginary sample size and the sample size of the data.

Imaginary Sample Size in Pictures

- If the sample size is large compared to the data sample size, the posterior probabilities will be close to the uniform distribution.
- If the data sample size is large compared to the imaginary sample size, the posterior probabilities will be close to the frequentist estimates.

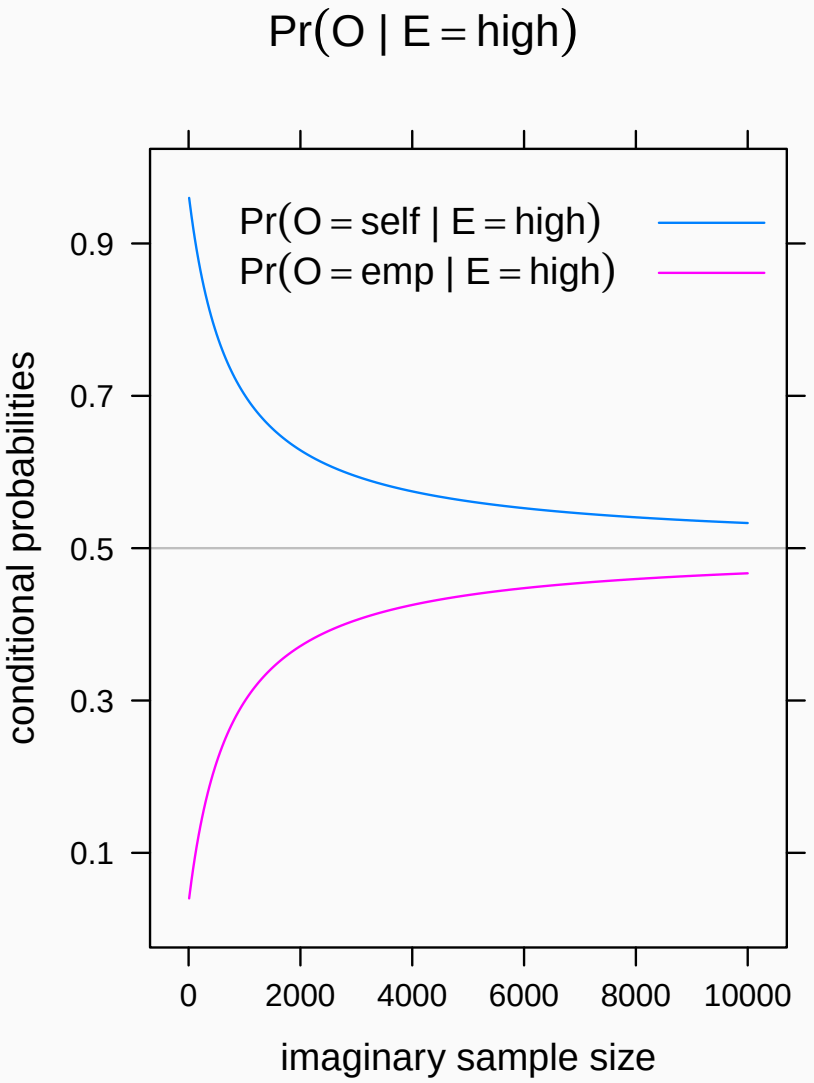
```
iss = seq(from = 10, to = 10000, by = 10)
trace = data.frame(
 ISS = iss,
 P1 = numeric(length(iss)),
 P2 = numeric(length(iss)),
 P3 = numeric(length(iss)),
 P4 = numeric(length(iss))
)

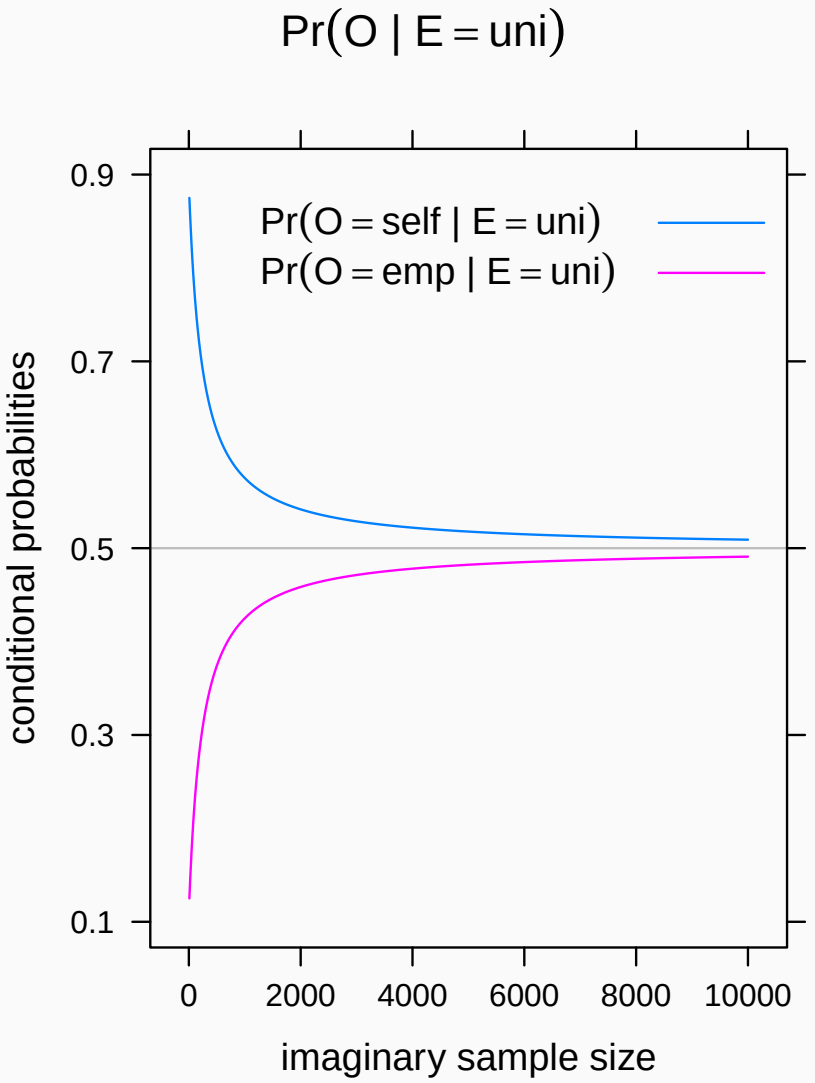
for (i in seq_along(iss)) {

  fitted = bn.fit(survey.dag, survey.data, method = "bayes", iss = iss[i])
  trace[i, c("P1", "P2", "P3", "P4")] = as.vector(coef(fitted$O))

}#FOR
```

Imaginary Sample Size in Pictures





Bayesian Probability Estimates in bn.fit()

The function to use is again `bn.fit()`, but this time with the argument `method` set to `bayes` to obtain posterior probability estimates.

```
survey.data = read.table("survey.txt", stringsAsFactors = TRUE)
survey.fitted = bn.fit(survey.dag, survey.data, method = "bayes", iss = 1)
```

The optional argument `iss` specifies the imaginary sample size, and it defaults to `1` (which is the smallest value that makes sense as a sample size).

No More NaNs with Bayesian Probability Estimates

An advantage of Bayesian probabilities is that it is impossible to obtain NaN estimates. If $n_{i+k} = 0$ then

$$w = \frac{\alpha_{i+j}}{\alpha_{i+j} + 0} = 1$$

and

$$\mathrm{P}(X_i = x_k \mid \Pi_{X_i} = \pi_j) = \frac{\alpha_{ikj}}{\alpha_{i+j}} = \frac{1}{q_i}.$$

If we have no information from the data because we do not have any observation for a particular conditional probability distribution, we fall back to the uniform distribution provided by the prior. In the case of frequentist probabilities we can only give up since there is no prior and we can only rely on the (non-existent) data.

Incomplete Data

How do we handle data with missing values? They arise in many fields for example:

- Faulty sensor readings.
- Values that have been intentionally omitted, such as sensitive (HIV status) or embarrassing (IQ) information in questionnaires.
- Some variables are unobservable in some situations.

The intuitive answer to this question would be to just throw out the data with the missing values and keep the data that are completely observed.

However, this only works if the data is missing completely at random. If that is not the case, we can get probability estimates that are markedly biased.

Classes of Missing Data

There are three classes of missing data:

- Missing completely at random (MCAR): there is no relationship between the data being missing and any values, observed or missing. Those missing data points are a random subset of the data.
- Missing at Random (MAR): there is a systematic relationship between the propensity of values to be missing and the observed data, but not the missing data.
- Missing Not at Random (MNAR): there is a relationship between the propensity of a value to be missing and its values.

MNAR is non-ignorable because the missing data mechanism itself has to be modelled manually (why the data are missing and what the likely values are). MCAR and MAR are both considered ignorable because we don't have to include any information about the missing data itself when we deal with the missing data.

Representing the Missingness Mechanism

In the context of BNs, each variable has a local distribution $X_i \sim \mathrm{P}(X_i \mid \Pi_{X_i})$ if the data are complete. If X_i has missing data, in the MCAR case

$$X_i \sim \begin{cases} \mathrm{P}(X_i \mid \Pi_{X_i}) & \text{for observed data } X_i^{(O)} \\ \mathrm{P}(X_i \mid \Pi_{X_i}) & \text{for missing data } X_i^{(M)}. \end{cases}$$

The same happens in the MAR case, since the missingness depends on Π_{X_i}. On the other hand, in the MNAR case

$$X_i \sim \begin{cases} \mathrm{P}(X_i^{(O)} \mid \Pi_{X_i}, M) & \text{for observed data } X_i^{(O)} \\ \mathrm{P}(X_i^{(M)} \mid \Pi_{X_i}, M) & \text{for missing data } X_i^{(M)} \end{cases}$$

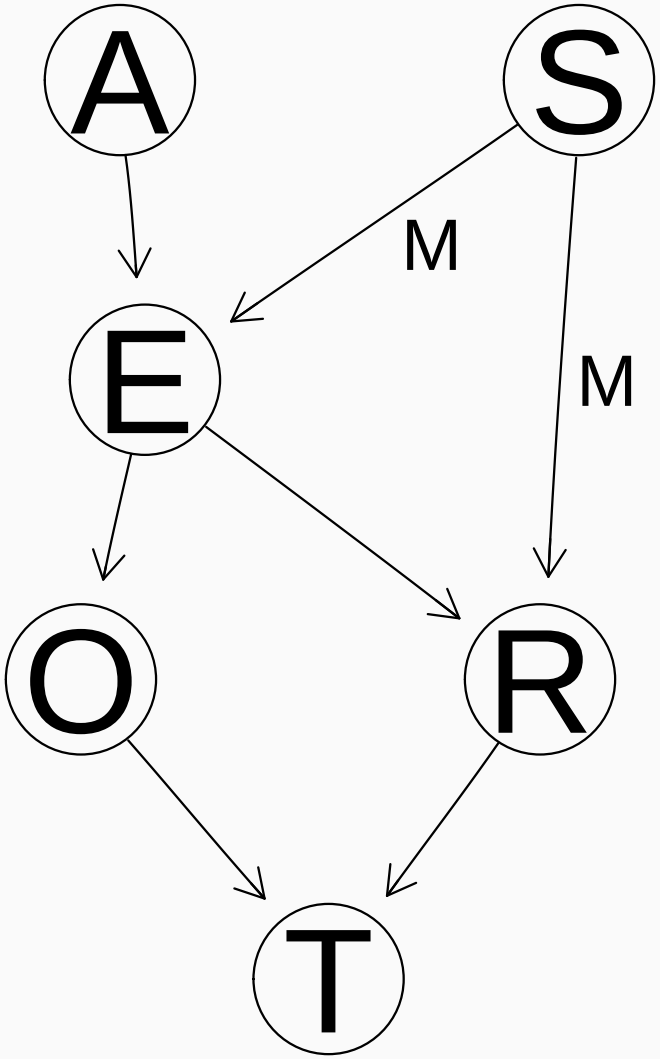
where M is the missingness mechanism. M is non-ignorable because we cannot estimate the local distribution of X_i properly without knowing the missing values in the first place.

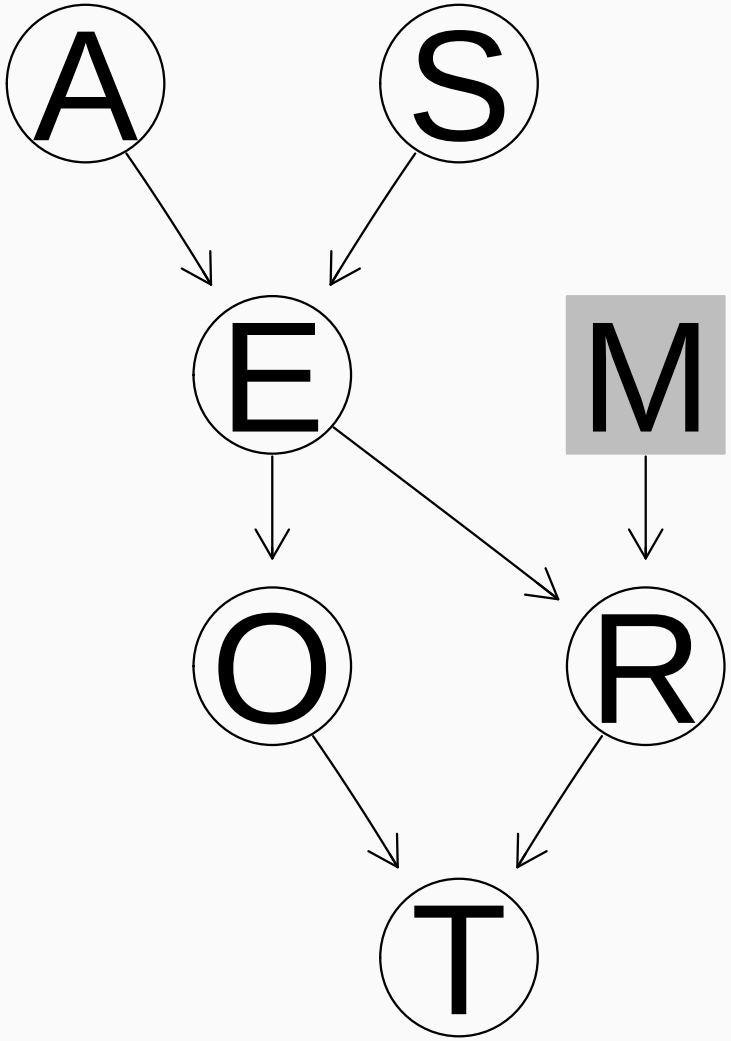
Examples with the Train Use Survey

Since the survey data are collected through a questionnaire, there will be a positive non-response rate for various questions and for the whole questionnaire.

- A MCAR situation may arise when questionnaires are lost in the post – the missingness does not depend on the characteristics of the individual.
- A MAR situation may arise if women refuse to answer some questions in the questionnaire in rates significant higher than men – that is fine since S is observed.
- A MNAR situation may arise if all people in a specific big city do not answer or people of certain social groups do not answer all or part of the questionnaire – we need to introduce M to identify the non-responders.

Examples with the Train Use Survey



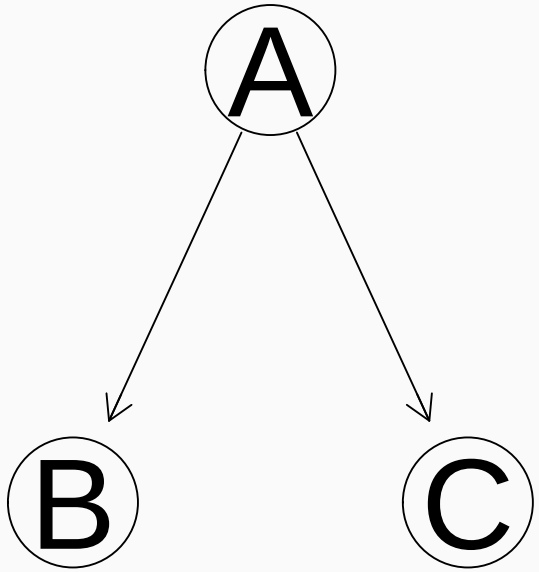


Manual Approach: a Numeric Example

Consider the following simple numeric example.

```
inc = data.frame(
  A = c(NA, "a1", "a1", "a1", NA),
  B = c("b1", "b2", "b1", "b1", "b2"),
  C = c("c1", "c1", NA, "c2", NA)
)
inc
```

```
      A  B    C
1 <NA> b1   c1
2   a1 b2   c1
3   a1 b1 <NA>
4   a1 b1   c2
5 <NA> b2 <NA>
```



If the data were complete, we could estimate the conditional probabilities in the BN simply by counting the frequencies of the various configurations of values.

Manual Approach: a Numeric Example

Frequentist probability estimates would look like:

$$\mathrm{P}(\mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{a1}}}{n}$$

$$\mathrm{P}(\mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{A} = \mathtt{a1})$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{b1,a1}}}{n_{\mathtt{a1}}}$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b2} \mid \mathtt{A} = \mathtt{a1}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a1})$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{b1,a2}}}{n_{\mathtt{a2}}}$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b2} \mid \mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2})$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c1} \mid \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{c1,a2}}}{n_{\mathtt{a2}}}$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c2} \mid \mathtt{A} = \mathtt{a1}) = 1 - \mathrm{P}(\mathtt{C} = \mathtt{c1} \mid \mathtt{A} = \mathtt{a1})$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c1} \mid \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{c1,a2}}}{n_{\mathtt{a2}}}$$

$$\mathrm{P}(\mathtt{C} = \mathtt{c2} \mid \mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{C} = \mathtt{c1} \mid \mathtt{A} = \mathtt{a2})$$

Manual Approach: a Numeric Example

Looking at A first, the count we need is

$$n_{\mathtt{a1}} = \mathbb{1}(\text{1st observation is a1}) + \mathbb{1}(\text{2nd observation is a1}) + \mathbb{1}(\text{3rd observation is a1}) + \mathbb{1}(\text{4th observation is a1}) + \mathbb{1}(\text{5th observation is a1}),$$

where $\mathbb{1}()$ is equal to 1 if its argument is true and 0 otherwise.

For the 2nd, 3rd and 4th observations we know that A = a1, so we can write

$$n_{\mathtt{a1}} = \mathbb{1}(\text{1st observation is a1}) + 1 + 1 + 1 + \mathbb{1}(\text{5th observation is a1}).$$

From a different perspective, what we are saying is that we know that those observations take value a1 with probability 1.

Manual Approach: a Numeric Example

If we take this new perspective further, we can then write:

$$n_{\texttt{a1}} = \mathrm{P}(\text{1st observation is a1}) + \mathrm{P}(\text{2nd observation is a1}) + \mathrm{P}(\text{3rd observation is a1}) + \mathrm{P}(\text{4th observation is a1}) + \mathrm{P}(\text{5th observation is a1}).$$

If we had no missing values, each of those probabilities would be equal to either 0 or 1 and tallying them up would give us the usual empirical frequency $n_{\texttt{a1}}$.

But since have missing values, all we can do is to say

$$n_{\texttt{a1}} = \mathrm{P}(\text{1st observation is a1}) + 1 + 1 + 1 + \mathrm{P}(\text{5th observation is a1}).$$

This is easier to work with, because now we can use the axioms of probability to borrow information from the other variables to fill the missing values.

Manual Approach: a Numeric Example

Consider that our BN can be written as

$$\mathrm{P}(\mathtt{A}, \mathtt{B}, \mathtt{C}) = \mathrm{P}(\mathtt{A})\,\mathrm{P}(\mathtt{B} \mid \mathtt{A})\,\mathrm{P}(\mathtt{C} \mid \mathtt{A}).$$

We cannot use $\mathrm{P}(\mathtt{A})$ to fill in the missing values because from the data we would guess $\mathrm{P}(\mathtt{A} = \mathtt{a1}) = 1$ and $\mathrm{P}(\mathtt{A} = \mathtt{a2}) = 0$ since a2 is never observed; but that is not desirable if we assume a2 can actually happen.

What we can do is to reverse the dependencies in the model to get

$$\mathrm{P}(\mathtt{A})\,\mathrm{P}(\mathtt{B} \mid \mathtt{A})\,\mathrm{P}(\mathtt{C} \mid \mathtt{A}) = \mathrm{P}(\mathtt{A})\,\mathrm{P}(\mathtt{B}, \mathtt{C} \mid \mathtt{A}) = \mathrm{P}(\mathtt{A} \mid \mathtt{B}, \mathtt{C})\,\mathrm{P}(\mathtt{B}, \mathtt{C})$$

an use $\mathrm{P}(\mathtt{A} \mid \mathtt{B}, \mathtt{C})$ instead of $\mathrm{P}(\mathtt{A})$ to borrow the information from the other variables.

Manual Approach: a Numeric Example

This gets us to

$$n_{\mathsf{a1}}^{(0)} = \mathrm{P}_0(\mathsf{A} = \mathsf{a1} \mid \mathsf{B} = \mathsf{b1}, \mathsf{C} = \mathsf{c1}) + 1 + 1 + 1 + \mathrm{P}_0(\mathsf{A} = \mathsf{a1} \mid \mathsf{B} = \mathsf{b1})$$

and if we assume as a starting point that

$$\mathrm{P}_0(\mathsf{A}) = \begin{cases} 0.5 & \text{for a1} \\ 0.5 & \text{for a2} \end{cases} \qquad \mathrm{P}_0(\mathsf{A} \mid \mathsf{B}, \mathsf{C}) = \begin{cases} 0.5 & \text{for a1 for all B, C} \\ 0.5 & \text{for a2 for all B, C} \end{cases}$$

$$\mathrm{P}_0(\mathsf{A} \mid \mathsf{B}) = \begin{cases} 0.5 & \text{for a1 given b1} \\ 0.5 & \text{for a2 given b2} \end{cases}$$

we can now compute

$$n_{\mathsf{a1}}^{(0)} = \frac{1}{2} + 1 + 1 + 1 + \frac{1}{2} = 4$$

as an initial estimate for n_{a1}.

Manual Approach: a Numeric Example

Replacing $n_{\mathtt{a1}}$ with $n_{\mathtt{a1}}^{(0)}$, we can estimate

$$\mathrm{P}_1(\mathtt{A} = \mathtt{a1}) = \frac{4}{5} = 0.8, \qquad \mathrm{P}_1(\mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{A} = \mathtt{a1}) = 0.2.$$

Moving to B, what we need to estimate is

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a1}) = \frac{n_{\mathtt{b1,a1}}}{n_{\mathtt{a1}}} = \frac{n_{\mathtt{b1,a1}}}{4}$$

where

$$\begin{aligned} n_{\mathtt{b1,a1}} &= \mathrm{P}(\text{1st observation is } \mathtt{b1}, \mathtt{a1}) + \mathrm{P}(\text{2nd observation is } \mathtt{b1}, \mathtt{a1}) + \\ &\quad \mathrm{P}(\text{3rd observation is } \mathtt{b1}, \mathtt{a1}) + \mathrm{P}(\text{4th observation is } \mathtt{b1}, \mathtt{a1}) + \\ &\quad \mathrm{P}(\text{5th observation is } \mathtt{b1}, \mathtt{a1}) \\ &= \mathrm{P}(\text{1st observation is } \mathtt{b1}, \mathtt{a1}) + 0 + 1 + 1 + 0 \end{aligned}$$

since only for the first observation B = b1 we do not know the value of A.

Manual Approach: a Numeric Example

We are working with the joint frequencies of A and B; hence we choose as starting probabilities the joint uniform

$$\mathrm{P}_0(\mathtt{B}, \mathtt{A} \mid \mathtt{C}) = \begin{cases} 0.25 & \text{for b1, a1 for both c1, c2} \\ 0.25 & \text{for b1, a2 for both c1, c2} \\ 0.25 & \text{for b2, a1 for both c1, c2} \\ 0.25 & \text{for b2, a2 for both c1, c2} \end{cases}$$

so that the probability of the first observation is

$$\begin{aligned} &\mathrm{P}(\text{1st observation is b1, a1}) \\ &= \mathrm{P}_0(\mathtt{A} = \mathtt{a1} \mid \mathtt{B} = \mathtt{b1}, \mathtt{C} = \mathtt{c1}) \\ &= \frac{\mathrm{P}_0(\mathtt{A} = \mathtt{a1}, \mathtt{B} = \mathtt{b1} \mid \mathtt{C} = \mathtt{c1})}{\mathrm{P}_0(\mathtt{B} = \mathtt{b1} \mid \mathtt{C} = \mathtt{c1})} \\ &= \frac{\mathrm{P}_0(\mathtt{A} = \mathtt{a1}, \mathtt{B} = \mathtt{b1} \mid \mathtt{C} = \mathtt{c1})}{\mathrm{P}_0(\mathtt{A} = \mathtt{a1}, \mathtt{B} = \mathtt{b1} \mid \mathtt{C} = \mathtt{c1}) + \mathrm{P}_0(\mathtt{A} = \mathtt{a2}, \mathtt{B} = \mathtt{b1} \mid \mathtt{C} = \mathtt{c1})} \\ &= \frac{0.25}{0.25 + 0.25} = 0.5. \end{aligned}$$

Manual Approach: a Numeric Example

We can now compute

$$n^{(0)}_{\mathtt{b1},\mathtt{a1}} = 0.5 + 0 + 1 + 1 + 0 = 2.5$$

and in turn

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a1}) = \frac{n^{(0)}_{\mathtt{b1},\mathtt{a1}}}{n^{(0)}_{\mathtt{a1}}} = \frac{2.5}{4} = 0.625$$

$$\mathrm{P}(\mathtt{B} = \mathtt{b2} \mid \mathtt{A} = \mathtt{a1}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a1}) = 0.375$$

which is the first of the two conditional distributions of B.

The second of the two conditional distributions of B is computed in the same way, starting from

$$\mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2}) = \frac{n_{\mathtt{b1},\mathtt{a2}}}{n_{\mathtt{a2}}} = \frac{n_{\mathtt{b1},\mathtt{a2}}}{n - n_{\mathtt{a1}}} = \frac{n_{\mathtt{b1},\mathtt{a2}}}{1}.$$

Manual Approach: a Numeric Example

Using the same $\mathrm{P}_0(\mathtt{A} \mid \mathtt{B}, \mathtt{C})$ as before,

$$\begin{aligned} n_{\mathtt{b1},\mathtt{a2}} &= \mathrm{P}(\text{1st observation is } \mathtt{b1}, \mathtt{a2}) + \mathrm{P}(\text{2nd observation is } \mathtt{b1}, \mathtt{a2}) + \\ &\quad \mathrm{P}(\text{3rd observation is } \mathtt{b1}, \mathtt{a2}) + \mathrm{P}(\text{4th observation is } \mathtt{b1}, \mathtt{a2}) + \\ &\quad \mathrm{P}(\text{5th observation is } \mathtt{b1}, \mathtt{a2}) \\ &= \mathrm{P}(\text{1st observation is } \mathtt{b1}, \mathtt{a2}) + 0 + 0 + 0 + 0, \end{aligned}$$

leading to $n^{(0)}_{\mathtt{b1},\mathtt{a2}} = 0.5 + 0 + 0 + 0 + 0 = 0.5$.

If we replace $n^{(0)}_{\mathtt{b1},\mathtt{a2}}$ in the expression above we get

$$\mathrm{P}_1(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2}) = \frac{n^{(0)}_{\mathtt{b1},\mathtt{a2}}}{n - n^{(0)}_{\mathtt{a1}}} = \frac{0.5}{5 - 4} = 0.5,$$

$$\mathrm{P}_1(\mathtt{B} = \mathtt{b2} \mid \mathtt{A} = \mathtt{a2}) = 1 - \mathrm{P}(\mathtt{B} = \mathtt{b1} \mid \mathtt{A} = \mathtt{a2}) = 0.5.$$

This makes complete sense: we never observe the combination of values b1, a2 so there is no information to learn from the data.

Manual Approach: a Numeric Example

As for C, what we need to estimate is

$$\mathrm{P}(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a1}) = \frac{n_{\mathsf{c1,a1}}}{n_{\mathsf{a1}}}$$

where we know that $n_{\mathsf{a1}}^{(0)} = 4$ from before and

$$\begin{aligned} n_{\mathsf{c1,a1}} = {} & \mathrm{P}(\text{1st observation is c1, a1}) + \mathrm{P}(\text{2nd observation is c1, a1}) + \\ & \mathrm{P}(\text{3rd observation is c1, a1}) + \mathrm{P}(\text{4th observation is c1, a1}) + \\ & \mathrm{P}(\text{5th observation is c1, a1}). \end{aligned}$$

The starting probabilities are the joint uniform distribution over C and A

$$\mathrm{P}_0(\mathsf{C}, \mathsf{A} \mid \mathsf{B}) = \begin{cases} 0.25 & \text{for c1, a1 given b1, b2} \\ 0.25 & \text{for c1, a2 given b1, b2} \\ 0.25 & \text{for c2, a1 given b1, b2} \\ 0.25 & \text{for c2, a2 given b1, b2} \end{cases}.$$

Manual Approach: a Numeric Example

Considering the partial observations we have for A and C, we can rewrite the above as:

$$n^{(0)}_{\mathsf{c1},\mathsf{a1}} = \mathrm{P}_0(\mathsf{A} = \mathsf{a1} \mid \mathsf{C} = \mathsf{c1}, \mathsf{B} = \mathsf{b1}) + 1 + \mathrm{P}_0(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a1}, \mathsf{B} = \mathsf{b1}) + 0 + \mathrm{P}_0(\mathsf{C} = \mathsf{c1}, \mathsf{A} = \mathsf{a1} \mid \mathsf{B} = \mathsf{b2}).$$

By the axioms of probability,

$$\mathrm{P}_0(\mathsf{A} \mid \mathsf{C}, \mathsf{B}) = \frac{\mathrm{P}_0(\mathsf{C}, \mathsf{A} \mid \mathsf{B})}{\mathrm{P}_0(\mathsf{C} \mid \mathsf{B})} = \frac{\mathrm{P}_0(\mathsf{C}, \mathsf{A} \mid \mathsf{B})}{\mathrm{P}_0(\mathsf{C}, \mathsf{A} = \mathsf{a1} \mid \mathsf{B}) + \mathrm{P}_0(\mathsf{C}, \mathsf{A} = \mathsf{a2} \mid \mathsf{B})} = \frac{0.25}{0.25 + 0.25} = 0.5$$

$$\mathrm{P}_0(\mathsf{C} \mid \mathsf{A}, \mathsf{B}) = \frac{\mathrm{P}_0(\mathsf{C}, \mathsf{A} \mid \mathsf{B})}{\mathrm{P}_0(\mathsf{A} \mid \mathsf{B})} = \frac{\mathrm{P}_0(\mathsf{A}, \mathsf{C} \mid \mathsf{B})}{\mathrm{P}_0(\mathsf{C} = \mathsf{c1}, \mathsf{A} \mid \mathsf{B}) + \mathrm{P}_0(\mathsf{C} = \mathsf{c2}, \mathsf{A} \mid \mathsf{B})} = \frac{0.25}{0.25 + 0.25} = 0.5$$

As a result, $n^{(0)}_{\mathsf{c1,a1}} = 0.5 + 1 + 0.5 + 0 + 0.25 = 2.25$ and

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a1}) = \frac{n^{(0)}_{\mathsf{c1,a1}}}{n^{(0)}_{\mathsf{a1}}} = \frac{2.25}{4} = 0.56,$$

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c2} \mid \mathsf{A} = \mathsf{a1}) = 1 - \mathrm{P}(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a1}) = 0.44.$$

The second conditional distribution of C,

$$\mathrm{P}(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a2}) = \frac{n_{\mathsf{c1,a2}}}{n_{\mathsf{a2}}},$$

requires $n_{\mathsf{a2}} = 1$ from before and

$$\begin{aligned} n_{\mathsf{c1,a2}} &= \mathrm{P}(\text{1st observation is c1, a2}) + \mathrm{P}(\text{2nd observation is c1, a2}) + \\ &\quad \mathrm{P}(\text{3rd observation is c1, a2}) + \mathrm{P}(\text{4th observation is c1, a2}) + \\ &\quad \mathrm{P}(\text{5th observation is c1, a2}) \\ &= \mathrm{P}_0(\mathsf{A} = \mathsf{a2} \mid \mathsf{C} = \mathsf{c1}, \mathsf{B} = \mathsf{b1}) + 0 + 0 + 0 + \\ &\quad \mathrm{P}_0(\mathsf{C} = \mathsf{c1}, \mathsf{A} = \mathsf{a2} \mid \mathsf{B} = \mathsf{b2}) = 0.5 + 0 + 0 + 0 + 0.25 = 0.75. \end{aligned}$$

Manual Approach: Summary

This gives the last conditional probability distribution:

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a2}) = \frac{n^{(0)}_{\mathsf{c1},\mathsf{a2}}}{n^{(0)}_{\mathsf{a2}}} = \frac{0.75}{1} = 0.75,$$

$$\mathrm{P}_1(\mathsf{C} = \mathsf{c2} \mid \mathsf{A} = \mathsf{a2}) = 1 - \mathrm{P}(\mathsf{C} = \mathsf{c1} \mid \mathsf{A} = \mathsf{a2}) = 0.25.$$

What did we do?

1. We could not estimate the conditional probabilities due to the missing values in the data.
2. We assumed all distributions were uniform as a starting point.
3. In the frequencies we needed to estimate the conditional probabilities, we replaced the missing values with the probabilities of observing corresponding values.
4. We computed the frequencies, and used them to compute the conditional probabilities in the BN.

Manual Approach: Summary

This is what we started from: uniform distributions everywhere.

```
dag = model2network("[A][B|A][C|A]")
A.prob = array(c(0.5, 0.5), dim = 2, dimnames = list(A = c("a1", "a2")))
B.prob = array(c(0.5, 0.5, 0.5, 0.5), dim = c(2, 2),
           dimnames = list(B = c("b1", "b2"), A = c("a1", "a2")))
C.prob = array(c(0.5, 0.5, 0.5, 0.5), dim = c(2, 2),
           dimnames = list(C = c("c1", "c2"), A = c("a1", "a2")))
bn.start = custom.fit(dag, list(A = A.prob, B = B.prob, C = C.prob))
```

```
coef(bn.start$A)
```

```
  A
 a1  a2
0.5 0.5
```

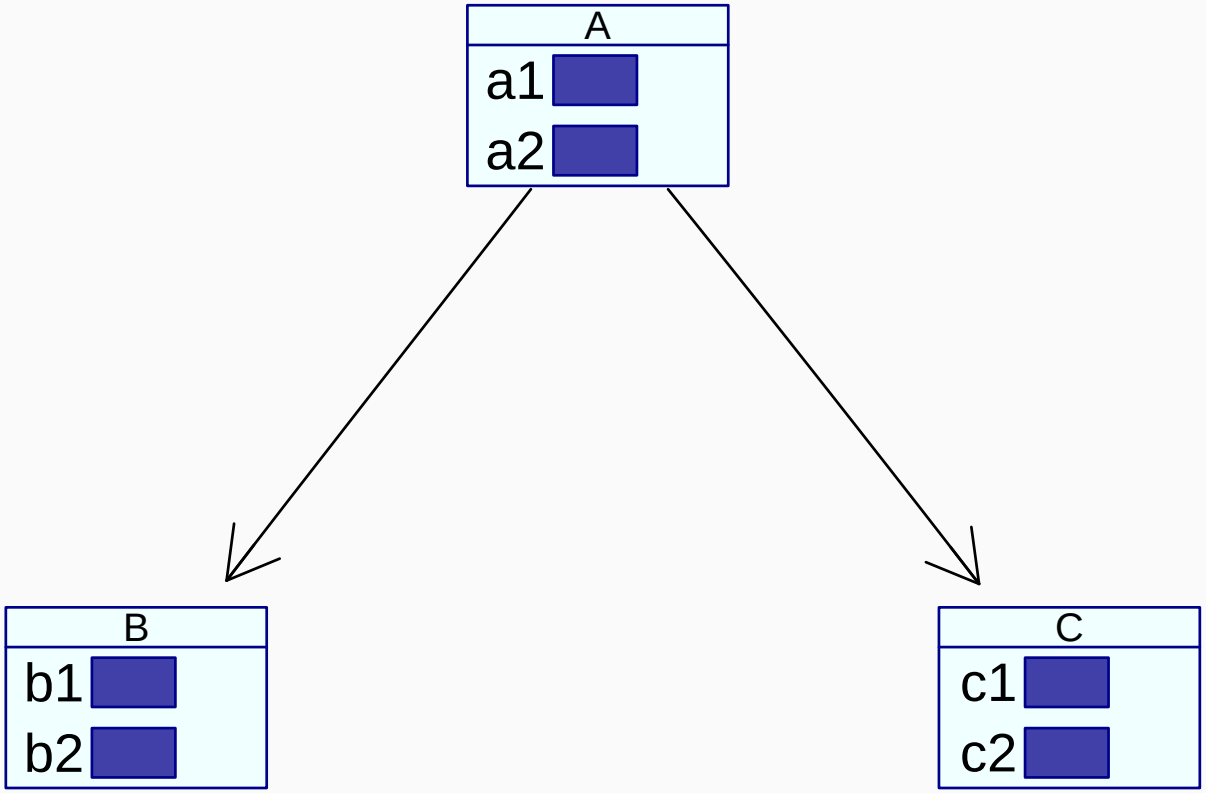
```
coef(bn.start$B)
```

```
     A
B     a1  a2
  b1 0.5 0.5
  b2 0.5 0.5
```

```
coef(bn.start$C)
```

```
     A
C     a1  a2
  c1 0.5 0.5
  c2 0.5 0.5
```

Manual Approach: Summary



Manual Approach: Summary

This is what we get at the end: updated distributions for all variables.

```
dag = model2network("[A][B|A][C|A]")
A.prob = array(c(0.8, 0.2), dim = 2, dimnames = list(A = c("a1", "a2")))
B.prob = array(c(0.625, 0.375, 0.5, 0.5), dim = c(2, 2),
           dimnames = list(B = c("b1", "b2"), A = c("a1", "a2")))
C.prob = array(c(0.56, 0.44, 0.75, 0.25), dim = c(2, 2),
           dimnames = list(C = c("c1", "c2"), A = c("a1", "a2")))
bn.step1 = custom.fit(dag, list(A = A.prob, B = B.prob, C = C.prob))
```

```
coef(bn.step1$A)
```

```
  A
  a1  a2
0.8 0.2
```

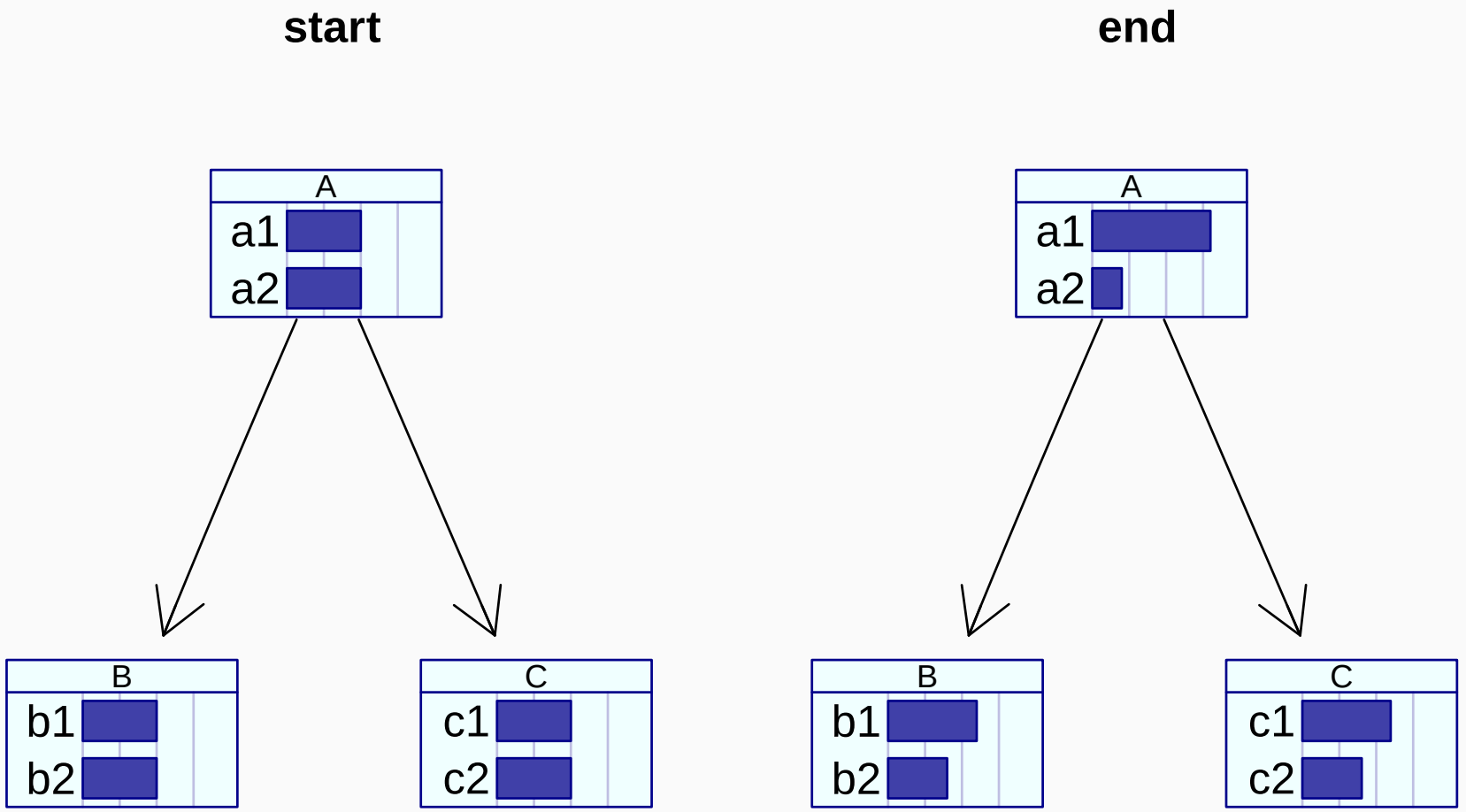
```
coef(bn.step1$B)
```

```
     A
B       a1    a2
  b1 0.625 0.500
  b2 0.375 0.500
```

```
coef(bn.step1$C)
```

```
     A
C       a1   a2
  c1 0.56 0.75
  c2 0.44 0.25
```

Manual Approach: Summary



The Expectation-Maximisation Algorithm

Iteratively performing the steps above leads to the Expectation-Maximisation (EM) algorithm, summarised below.

1. Assume some distribution for all the required probability distributions, such as the uniform distribution.
2. Iterate until convergence:
   - 2.1 E-step: compute the tables of expected frequencies by replacing the missing values with their probabilities given the observed values for each observation.
   - 2.2 M-step: compute the probability distributions as if the expected frequencies were the real frequencies.

The algorithm is guaranteed to converge to the best possible estimates of the conditional probabilities, defined as those that give the highest probability to the data. In practice, we can choose to stop when consecutive iterations do not change first few significant digits of the probabilities.

In the context of the numeric example above:

E-step	M-step
$\mathrm{P}_0(\mathtt{A}), \mathrm{P}_0(\mathtt{B} \mid \mathtt{A}), \mathrm{P}_0(\mathtt{C} \mid \mathtt{A})$	$n^{(0)}_{\mathtt{a1}}, n^{(0)}_{\mathtt{b1,a1}}, n^{(0)}_{\mathtt{b1,a2}}, n^{(0)}_{\mathtt{c1,a1}}, n^{(0)}_{\mathtt{c1,a2}}$
$\mathrm{P}_1(\mathtt{A}), \mathrm{P}_1(\mathtt{B} \mid \mathtt{A}), \mathrm{P}_1(\mathtt{C} \mid \mathtt{A})$	$n^{(1)}_{\mathtt{a1}}, n^{(1)}_{\mathtt{b1,a1}}, n^{(1)}_{\mathtt{b1,a2}}, n^{(1)}_{\mathtt{c1,a1}}, n^{(1)}_{\mathtt{c1,a2}}$
$\mathrm{P}_2(\mathtt{A}), \mathrm{P}_2(\mathtt{B} \mid \mathtt{A}), \mathrm{P}_2(\mathtt{C} \mid \mathtt{A})$	$n^{(2)}_{\mathtt{a1}}, n^{(2)}_{\mathtt{b1,a1}}, n^{(2)}_{\mathtt{b1,a2}}, n^{(2)}_{\mathtt{c1,a1}}, n^{(2)}_{\mathtt{c1,a2}}$
$\mathrm{P}_3(\mathtt{A}), \mathrm{P}_3(\mathtt{B} \mid \mathtt{A}), \mathrm{P}_3(\mathtt{C} \mid \mathtt{A})$	$n^{(3)}_{\mathtt{a1}}, n^{(3)}_{\mathtt{b1,a1}}, n^{(3)}_{\mathtt{b1,a2}}, n^{(3)}_{\mathtt{c1,a1}}, n^{(3)}_{\mathtt{c1,a2}}$

If, say, $\mathrm{P}_3(\mathtt{A}) \approx \mathrm{P}_2(\mathtt{A}), \mathrm{P}_3(\mathtt{B} \mid \mathtt{A}) \approx \mathrm{P}_2(\mathtt{B} \mid \mathtt{A}), \mathrm{P}_3(\mathtt{C} \mid \mathtt{A}) \approx \mathrm{P}_2(\mathtt{C} \mid \mathtt{A})$, we can declare that EM has converged and stop.

Code: Computing $n_{\mathsf{a1}}^{(0)}$

```
grain.start = as.grain(bn.start)
nA = array(c(0, 0), dim = 2, dimnames = list(A = c("a1", "a2")))

for (i in seq(nrow(inc))) {

  if (is.na(inc[i, "A"])) {

    nA = nA + querygrain(grain.start, node = "A",
                 evidence = list(B = inc[i, "B"], C = inc[i, "C"]))$A

  }#THEN
  else {

    nA[inc[i, "A"]] = nA[inc[i, "A"]] + 1

  }#ELSE

}#FOR

nA
```

```
A
a1 a2
 4  1
```

Code: Computing $n^{(0)}_{b1,a1}$ and $n^{(0)}_{b1,a2}$

```
nBA = array(rep(0, 4), dim = c(2, 2),
        dimnames = list(B = c("b1", "b2"), A = c("a1", "a2")))

for (i in seq(nrow(inc))) {

  if (is.na(inc[i, "A"]) || is.na(inc[i, "B"])) {

    nBA = nBA + querygrain(grain.start, node = c("B", "A"), type = "joint",
                  evidence = list(C = inc[i, "C"]))

  }#THEN
  else {

    nBA[inc[i, "B"], inc[i, "A"]] = nBA[inc[i, "B"], inc[i, "A"]] + 1

  }#ELSE

}#FOR

nBA
```

```
    A
B     a1  a2
  b1 2.5 0.5
  b2 1.5 0.5
```

Code: Computing $n^{(0)}_{c1,a1}$ and $n^{(0)}_{c1,a2}$

```
nCA = array(rep(0, 4), dim = c(2, 2),
        dimnames = list(C = c("c1", "c2"), A = c("a1", "a2")))

for (i in seq(nrow(inc))) {

  if (is.na(inc[i, "A"]) && is.na(inc[i, "C"])) {
    nCA = nCA + querygrain(grain.start, node = c("C", "A"), type = "joint",
                  evidence = list(B = inc[i, "B"]))

  }#THEN
  else if (is.na(inc[i, "A"]) && !is.na(inc[i, "C"])) {

    p =  querygrain(grain.start, node = "A",
           evidence = list(B = inc[i, "B"], C = inc[i, "C"]))$A
    nCA[inc[i, "C"], ] = nCA[inc[i, "C"], ] + p

  }#THEN

}#FOR
```

Code: Computing $n^{(0)}_{c1,a1}$ and $n^{(0)}_{c1,a2}$

```
for (i in seq(nrow(inc))) {

  if (!is.na(inc[i, "A"]) && is.na(inc[i, "C"])) {

    p = querygrain(grain.start, node = "C",
          evidence = list(B = inc[i, "B"], A = inc[i, "A"]))$C
    nCA[, inc[i, "A"]] = nCA[, inc[i, "A"]] + p

  }#THEN
  else if (!is.na(inc[i, "A"]) && !is.na(inc[i, "C"])) {

    nCA[inc[i, "C"], inc[i, "A"]] = nCA[inc[i, "C"], inc[i, "A"]] + 1

  }#ELSE

}#FOR
```

nCA

```
    A
C      a1   a2
  c1 2.25 0.75
  c2 1.75 0.25
```

Code: Computing $\mathrm{P}_1(\cdot \mid \cdot)$

```
bn.step1 = bn.start
bn.step1$A = prop.table(nA)
bn.step1$B = prop.table(nBA, margin = 2)
bn.step1$C = prop.table(nCA, margin = 2)
coef(bn.step1$A)
```

```
  A
   a1  a2
  0.8 0.2
```

```
coef(bn.step1$B)
```

```
     A
  B       a1    a2
    b1 0.625 0.500
    b2 0.375 0.500
```

```
coef(bn.step1$C)
```

```
     A
  C       a1    a2
    c1 0.562 0.750
    c2 0.438 0.250
```

```
grain.step1 = as.grain(bn.step1)
```

Code: Computing $n_{a1}^{(1)}$

```
nA = array(c(0, 0), dim = 2, dimnames = list(A = c("a1", "a2")))

for (i in seq(nrow(inc))) {

  if (is.na(inc[i, "A"])) {

    nA = nA + querygrain(grain.step1, node = "A",
                  evidence = list(B = inc[i, "B"], C = inc[i, "C"]))$A

  }#THEN
  else {

    nA[inc[i, "A"]] = nA[inc[i, "A"]] + 1

  }#ELSE

}#FOR

nA
```

```
A
   a1    a2
4.539 0.461
```

Code: Computing $n^{(0)}_{b1,a1}$ and $n^{(0)}_{b1,a2}$

```
nBA = array(rep(0, 4), dim = c(2, 2),
        dimnames = list(B = c("b1", "b2"), A = c("a1", "a2")))

for (i in seq(nrow(inc))) {

  if (is.na(inc[i, "A"]) || is.na(inc[i, "B"])) {

    nBA = nBA + querygrain(grain.step1, node = c("B", "A"), type = "joint",
                  evidence = list(C = inc[i, "C"]))

  }#THEN
  else {

    nBA[inc[i, "B"], inc[i, "A"]] = nBA[inc[i, "B"], inc[i, "A"]] + 1

  }#ELSE

}#FOR

nBA
```

```
    A
B       a1    a2
  b1 2.97 0.581
  b2 1.23 0.225
```

Code: Computing $n^{(1)}_{c1,a1}$ and $n^{(1)}_{c1,a2}$

```
nCA = array(rep(0, 4), dim = c(2, 2),
        dimnames = list(C = c("c1", "c2"), A = c("a1", "a2")))

for (i in seq(nrow(inc))) {

  if (is.na(inc[i, "A"]) && is.na(inc[i, "C"])) {
    nCA = nCA + querygrain(grain.step1, node = c("C", "A"), type = "joint",
                  evidence = list(B = inc[i, "B"]))

  }#THEN
  else if (is.na(inc[i, "A"]) && !is.na(inc[i, "C"])) {

    p =  querygrain(grain.step1, node = "A",
           evidence = list(B = inc[i, "B"], C = inc[i, "C"]))$A
    nCA[inc[i, "C"], ] = nCA[inc[i, "C"], ] + p

  }#THEN

}#FOR
```

Code: Computing $n^{(1)}_{c1,a1}$ and $n^{(1)}_{c1,a2}$

```
for (i in seq(nrow(inc))) {

  if (!is.na(inc[i, "A"]) && is.na(inc[i, "C"])) {

    p = querygrain(grain.step1, node = "C",
          evidence = list(B = inc[i, "B"], A = inc[i, "A"]))$C
    nCA[, inc[i, "A"]] = nCA[, inc[i, "A"]] + p

  }#THEN
  else if (!is.na(inc[i, "A"]) && !is.na(inc[i, "C"])) {

    nCA[inc[i, "C"], inc[i, "A"]] = nCA[inc[i, "C"], inc[i, "A"]] + 1

  }#ELSE

}#FOR

nCA
```

```
   A
C     a1     a2
  c1 2.77 0.5387
  c2 1.62 0.0625
```

Code: Computing $\mathrm{P}_2(\cdot \mid \cdot)$

```r
bn.step2 = bn.start
bn.step2$A = prop.table(nA)
bn.step2$B = prop.table(nBA, margin = 2)
bn.step2$C = prop.table(nCA, margin = 2)
coef(bn.step2$A)
```

```
  A
     a1     a2
0.9079 0.0921
```

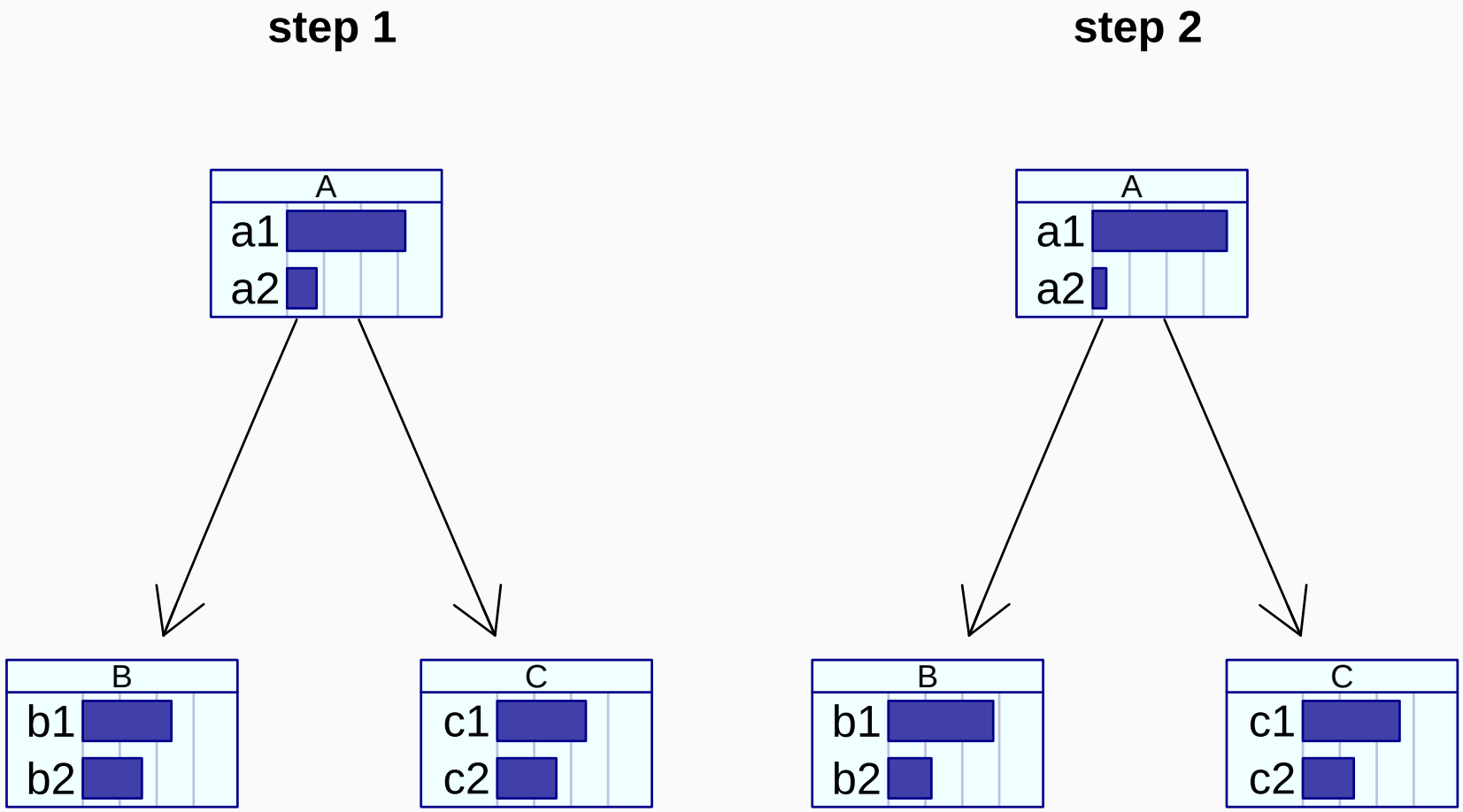
```r
coef(bn.step2$B)
```

```
     A
B       a1    a2
  b1 0.708 0.721
  b2 0.292 0.279
```

```r
coef(bn.step2$C)
```

```
     A
C       a1    a2
  c1 0.631 0.896
  c2 0.369 0.104
```

Code: Comparing $P_1(\cdot \mid \cdot)$ and $P_2(\cdot \mid \cdot)$



Summary and Remarks

- Learning the parameters of a Bayesian network is a direct application of probability theory.
- There are two main paradigms: frequentist and Bayesian.
- The Bayesian approach performs better with small samples and always produced well-formed estimates without `NaN`s.
- The EM algorithm makes it possible to use incomplete data containing missing values to learn the parameters of a BN.

Thanks!

Any questions?