UTRECHT UNIVERSITY

GRADUATE SCHOOL OF NATURAL SCIENCES

ARTIFICIAL INTELLIGENCE

MASTER'S THESIS

---

# Markov Abstraction Equivalence Classes

---

*Combining the Markov Equivalence Class and Constructive $\tau$-Abstraction to restrict the number of Causal Models*

*Author:*
Geke Pals
4148606
gekehpals@gmail.com

*Supervisor:*
dr. Sander Beckers
*Second examiner:*
dr. Benjamin Rin

June 14, 2019

Utrecht University

## Abstract

The last decade, the influence and the use of causal models is growing in several scientific disciplines. Recently, Beckers & Halpern (2019) and Beckers, Eberhardt & Halpern (2019) developed an account of abstraction for causal models which makes it possible to go from a low-level causal model to a high-level causal model, including interventions on the low-level and high-level causal model. This thesis combines the theory of abstracting causal models with the theory of Markov Equivalence Classes to come to an account of Markov Abstraction Equivalence Classes. A Markov Abstraction Equivalence Class is a subset of a Markov Equivalence Class, generated by using the information of an abstraction to eliminate models from the Markov Equivalence Class. Markov Abstraction Equivalence Classes reduce the search space of causal search algorithms, which improves the performance of causal search algorithms. The `pcabs` algorithm is developed to put the theory of Markov Abstraction Equivalence Classes into practice. This thesis builds on the theory of causal models, causal search algorithms, Markov Equivalence Classes and constructive $\tau$-abstraction.

**Keywords:** causal models, causal search algorithms, PC algorithm, Markov Equivalence Class, constructive $\tau$-abstraction, Markov Abstraction Equivalence Class

# Table of Contents

# 1 | Introduction

Since the beginning of human thinking, the concept of causality has been discussed by philosophers, scientists and everyday people. The basic principle of causality is about as intuitive as it gets: a child can recognize that when she kicks a ball against the window, the window will break. This is a simple causal relation: *if* the ball hits the window, *then* the window breaks. Reasoning back, the child will realize that because she does not want the window to break, she should not kick the ball against it. As simple as this might seem, philosophers and scientists have had much discussion about what causality is exactly, and how one can use it in science. The heart of the discussion lies at the question of how to *prove* causality. I can reason that every time I kick the ball against the window, the window breaks, so the ball hitting the window *causes* the window to break. However, this is no guarantee for causality to be at play. It is also the case that every time the rooster crows, the sun rises, but we would not say that the crowing of the rooster *causes* the sun to rise. It turns out that events that follow up each other do not necessarily have a causal relation – in most cases, a third or multiple other variables are involved. This has led scientists to embrace the notorious statement "correlation does not imply causation" and the existence of ironical websites like `https://tylervigen.com/spurious-correlations`, in which the most bizarre correlations between two events show the impossibility of any causal relation between them. Popular culture plays with the "correlation does not imply causation"-phenomenon as well, for example in the recent movie Smallfoot (2018), where it is believed that hitting a gong is necessary each morning to bring the sun to rise, and in a comic by xkcd, shown in Figure 1.1.

Because of the difficulty of proving causality, scientists do not even have the scientific tools to express causality in cases when we know it is true. Consider the example in Pearl (2018), where we want to write down the causal

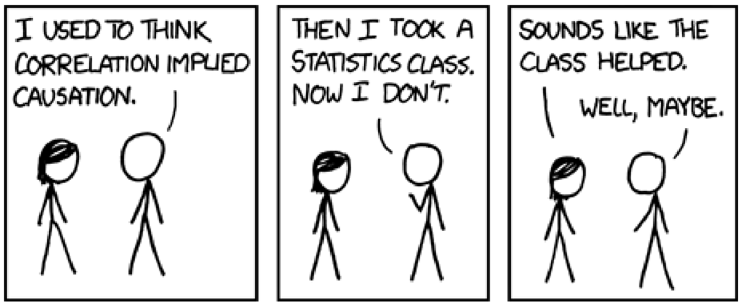

I USED TO THINK CORRELATION IMPLIED CAUSATION.
THEN I TOOK A STATISTICS CLASS. NOW I DON'T.
SOUNDS LIKE THE CLASS HELPED.
WELL, MAYBE.

Figure 1.1: A comic about the relation between correlation and causation by xkcd.

relationship between atmospheric pressure $P$ and the barometer reading $B$. We can write down this relationship in an equation like $B = kP$, where $k$ is some constant of proportionality: the value of the barometer is determined by the atmospheric pressure times a constant. We can use the rules of algebra to rewrite the statement as $P = B/k$, $k = B/P$ or as $B - kP = 0$. All the equations express the knowledge that if we know the value of two variables, we can determine the value of the third variable. However, none of the equations express our knowledge that it is the pressure that causes the value of the barometer. It is impossible to write down such a causal relationship in mathematical terms.

If we look at our own experiences, is the statement "correlation does not imply causation" consistent with how we reason? I am willing to believe that, if every time I eat an orange, a rash develops, the orange is the cause of the rash. Or look at the famous tennis player Rafael Nadal, who seems to believe that each of his 19 rituals is needed in order to win a tennis match. And I'm sure that you attribute your improved health to the medicine you took, instead of just the coincidence that they happened to correlate. All of this implies that in most cases, people are satisfied to accept correlated events as causally related events.

This omnipresence of thinking causally in everyday human thinking led some scientists to search for ways to involve causal thinking in scientific methods as well. A pioneer of this was Sewall Wright, who invented path analysis in the 1920s. This is a way to model dependent and independent variables graphically in order to answer causal questions. However, it took a long time before Wright's method got appreciated, apart from some coincidental duplications in social sciences and economics. In 1985, Judea Pearl brought Bayesian Networks into the world as a way to simplify computations with probability tables. It turned out that Bayesian Networks were a great tool to also model causality. These models were called Causal Models. The books by Spirtes, Glymour and Scheines (1993) and Pearl (2000) were influential in summarizing and explaining causal models, causal inference and causal discovery.

So what is the strength of causal inference? Causal models make it possible to model causality graphically: by pointing an arrow from variable $A$ to variable $B$, you express that $A$ causes $B$. An arrow missing from variable $A$ to variable $B$ means that $B$ is not caused by $A$. Causal inference makes it possible to reason with the causal relations expressed in causal models. You can reason about elementary causal questions ("is $B$ caused by $A$?"), you can reason about interventions ("what will happen if I *do* this?") and you can reason about counterfactuals ("what *would have* happened if I had done this?"). Some real questions asked in this category are "does smoking cause cancer?", "what will happen if we give this medicine to our patients?" and "what would have happened if we had funded a different diabetes treatment in Dutch insurances?".

The causal method is very useful and powerful. It allows us to formalize and objectively answer questions of causality, that for long have been impossible to answer in a scientific way. Nowadays, the causal method is used in a variety of scientific disciplines, like economics, social sciences, statistics, computer science and epidemiology. Causal inference allows us to form a bridge between human thinking and scientific thinking, by formalizing age-old causal questions. This shows the importance of causal inference for Artificial Intelligence: by formalizing a typical human-way of thinking, you allow this way of thinking to be used for artificial purposes too. Instead of an artificial intelligence-agent learning only from data, causal relations that are available in the data can be added to the agent's knowledge, which makes the range of possible questions to be asked and to be answered much wider. As Pearl (2018) describes in his Introduction:

> *"I believe that causal reasoning is essential for machines to communicate with us in our own language about policies, experiments, explanations, theories, regret, responsibility, free will, and obligations – and, eventually, to make their own moral decisions."* (Pearl 2018)

In this thesis, I will add a new insight to the theoretical foundation of causal modelling. Specifically, I combine the theory of abstracting causal models with the theory of Markov Equivalence Classes to come to an account of Markov Abstraction Equivalence Classes. An abstraction on a causal model is a way to go from a large, detailed causal model to a smaller, less detailed causal model, i.e. the smaller causal model is an *abstraction* of the larger model. A Markov Equivalence Class is a class of causal models that behave the same. Both terms will be explained extensively in this thesis.

This new account, Markov Abstraction Equivalence Classes, is accompanied by an algorithm called `pcabs`. The algorithm allows causal search algorithms to find a smaller set of possible causal models. Causal search algorithms are algorithms that are able to construct a causal model from data. Up until now, the norm for causal search algorithms is to find the Markov Equivalence Class of a causal model. By using the `pcabs` algorithm, causal search algorithms will be able to find the Markov Abstraction Equivalence Class, which is a subset of the Markov Equivalence Class. This will improve the performance of causal discovery algorithms.

The thesis is structured as follows. Chapter 2 serves as an introduction to causal models, consisting of an introduction to graphs, probability theory and Bayesian networks. Most terms and notations used in the rest of the thesis are explained in this chapter. Readers familiar with the subject of causal models are invited to skip this chapter.

Chapter 3 is an introduction to causal search algorithms, also called causal discovery algorithms. The important assumptions used in causal discovery are explained, such as the Causal Markov Condition and the Faithfulness Assumption. The three main approaches to causal discovery algorithms are discussed, and which algorithm belongs to which approach.

Chapter 4 discusses Markov Equivalence Classes, an important term in causal modelling and causal discovery algorithms. The definitions of Markov Equivalence and Markov Equivalence Classes are provided and the chapter ends with an example.

Chapter 5 discusses the PC Algorithm, one of the most used causal discovery algorithms. First, it is explained how algorithms like the PC algorithm are able to extract causal relations from data. Then, the reader is provided with the pseudocode of the PC algorithm, together with an extensive example of its use.

Chapter 6 discusses the phenomenon of abstracting causal models. Abstracting causal models is quite a technical subject. The specific type of abstraction called constructive $\tau$-abstraction is the only type of abstraction discussed in this thesis. The reader is provided with a definition and an example.

Subsequently, Chapter 7 is where the method and use of Markov Abstraction Equivalence Classes is presented. First, the notion of $\tau$-compatibility as a characteristic of a causal model is introduced. It is then argued in the Ontological Faithfulness Assumption that a causal model must be compatible with $\tau$ in order for a constructive $\tau$-abstraction to be possible. After an example of this, the definition of Markov $\tau$-Abstraction Equivalence and Markov Abstraction Equivalence Classes is given. The chapter ends with an explanation of why the Markov Abstraction Equivalence Class is useful.

Chapter 8 is focussed on the `pcabs` algorithm. This is the algorithm that puts the theory of Markov Abstraction Equivalence Classes into practice. First, `pcabs` is introduced conceptually, by a discussion of the four scenarios it faces. Following this is the pseudocode of `pcabs`. The chapter ends with the results of testing the `pcabs` algorithm.

In the appendix, the complete code of `pcabs` can be found. It is introduced with an example and explanation of how to use `pcabs` in practice.

# 2 | Causal Models

In this chapter I will provide an introduction to the basic concepts of causal modelling that will be used in the rest of the thesis. I will start with some elementary graph theory and probability theory. Next, I will describe Bayesian networks and d-separation. I will end the chapter with the definition of causal models and its characteristics.

## 2.1 Graphs

**Definition 2.1.1.** ***Graph*** A graph $G$ consists of a set of vertices (or nodes) $V$ and a set of edges (or links) $E$ that connect pairs of vertices. ■

The vertices in a graph represent the variables. There are two types of variables in a graph:

1. Exogenous variables, which are variables that are outside the control of the model;
2. Endogenous variables, which are variables that are determined by the values of the other variables in the model.

If you want to construct a simple model of how the type of weather influences the growth of the plants in your garden, you could use the variables $Weather$ and *Plant Growth*. The variable *Plant Growth* is completely determined by the variables in the model, namely by the variable $Weather$ (taken aside other variables that could influence plant growth, like the use of a fertilizer). *Plant Growth* is thus an endogenous variable. On the other hand, the variable $Weather$ is an exogenous variable, as its value – which could be rain, sunny, cloudy, etc. – is not determined in the model, but outside of the model, through factors like air pressure and the season. Its value is thus outside the control of the model.

Each edge in a graph can either be directed (a single arrowhead on the edge), undirected (no arrowhead) or bidirected (an arrowhead on each side of the edge). Two variables connected by an edge are called *adjacent*. If all edges in a graph are directed, we call the graph a *directed graph*. Figure 2.1 shows an example of a directed graph. If we remove all arrowheads in a graph, we are left with the *skeleton* of that graph. A *path* in a graph is a sequence of edges such that each edge starts with the node ending the preceding edge. Paths may go along or against the direction of the arrows. For example, in Figure 2.1, a possible path is $((W, Z), (Z, Y), (Y, X), (X, Z))$. If every edge in a path is directed from the first to the second node of the pair, we have a *directed path*. In Figure 2.1, the path $((W, Z), (Z, Y))$ is directed, but the path $((Z, Y), (Y, X))$ is not. If there exists a path between two nodes in a graph, then the two nodes are *connected*; else they are *disconnected*.

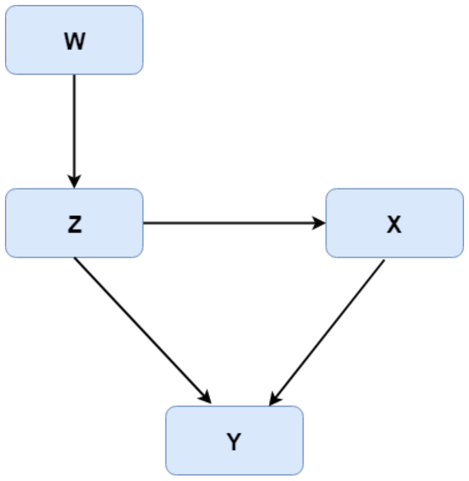


Figure 2.1: A directed acyclic graph (DAG).

A graph can contain *cycles*:

**Definition 2.1.2.** ***Cycle*** A cycle in a graph is a directed path from a node to itself. ■

A simple example of a cycle is the directed path $A \to B, B \to C, C \to A$. A graph that contains no cycles is called an *acyclic graph*. A graph that is both directed as well as acyclic is called a *directed acyclic graph* (DAG). Figure 2.1 is such a graph. These kinds of graph will be used most often in the discussion of causal graphs.

There are several kinds of relationships in a graph. The *parents* of a node are the direct ancestors of a node, i.e. the nodes at the other end of all incoming edges. The *children* of a node are the direct descendants of a node, i.e. the nodes at the other end of all outgoing edges. The *ascendants* of a node is the set of nodes that contains the parents of a node, the parents of the parents of a node, etc. In other words, all nodes that can reach the node of interest with a directed path. The *descendants* of a node is the set of nodes that contains the children of a node, the children of the children of a node, etc. In other words, all nodes that the node of interest can reach with a directed path. A *family* in a graph is the set of nodes containing a node and all its parents.

In Figure 2.1, variable $Y$ has two parents ($Z$ and $X$), no children, three ancestors ($Z$, $X$ and $W$) and no descendants. Variable $Z$ has one parent ($W$), two children ($Y$ and $X$), one ancestor ($W$) and two descendants ($Y$ and $X$). {W}, {Z, W}, {X, Z} and {Y, Z, X} are all the families in Figure 2.1.

A node in a directed graph is called a *root* if it has no parents, and a *sink* if it has no children. In Figure 2.1, variable $W$ is the root of the graph and variable $Y$ is the sink of the graph. Every DAG has at least one root and at least one sink. A connected DAG in which every node has at most one parent is called a *tree*, and a tree in which every node has at most one child is called a *chain.* A graph in which every pair of nodes in the graph is connected by an edge is called *complete.* A graph in which there are no edges is called *empty.*

## 2.2 Probability Theory

If you have a graph, you can add probabilities to it. In Figure 2.1, that would be to say that the edge from $W$ to $Z$ means that there is a 75% chance that after $W$, $Z$ follows. This is what is done in Bayesian Networks, upon which causal models are built. In this section I will shortly discuss some basic concepts in Probability Theory, in order to make the rest of the thesis understandable.

A probability is always the probability of *something*: of an event, a statement, a proposition, etc. In formal terms, we speak of $P(A)$: the probability of $A$. Probabilities are always expressed in a number from 0 to 1. A probability of 1 means that $A$ is certainly true, a probability of 0 means that $A$ is certainly false. We also speak of conditional probabilities $P(A \mid B)$. This denotes the probability of $A$, under the assumption that we already know $B$ with complete certainty. Think of it like this: we can determine the probability of a wet pavement $A$ ($P(A)$), but we can also determine the probability of a wet pavement given that we know $B$, namely that it has rained today ($P(A \mid B)$). This is all there is to conditional probabilities: you take into account that you have acquired some extra knowledge.

If it turns out that $P(A \mid B) = P(A)$, then we say that $A$ and $B$ are independent, because knowing $B$ does not change anything about the probability of $A$. It can also be the case that $P(A \mid B, C) = P(A \mid C)$, which means that $A$ and $B$ are conditionally independent given $C$: once we know $C$, learning $B$ does not change the probability of $A$. We calculate conditional probabilities as follows:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

Vice versa, when we want to calculate the probability of a joint event ($A \wedge B$), and when we know that $A$ and $B$ are <u>not</u> independent, we can do it as follows:

$$P(A \wedge B) = P(A \mid B) \cdot P(B)$$

If we know that $A$ and $B$ are in fact independent, the calculation of joint event $A \wedge B$ becomes easier:

$$P(A \wedge B) = P(A) \cdot P(B)$$

A famous theorem in Probability Theory is Bayes' Theorem. Bayes' Theorem makes it possible to calculate a conditional probability by turning the probability around: to calculate $P(A \mid B)$, you can use the probability of $P(B \mid A)$. The Theorem is as follows:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

Bayes' Theorem is useful when you do not have the exact information needed to calculate the conditional probabilities, but instead you know other probabilities related to it. For example, if you want to know the probability of having cancer given that you are a certain age, you can do so by using the probability of being a certain age given that you have cancer, the probability of having cancer and the probability of being a certain age. Formally:

$$P(Cancer \mid Age) = \frac{P(Age \mid Cancer) \cdot P(Cancer)}{P(Age)}$$

The probability of being a certain age given that you have cancer can easily be read off data of people having cancer, and likewise the probability of having cancer and the probability of being a certain age are easy to determine.

## 2.3 Bayesian Networks

Bayesian networks are graphs that represent conditional probabilities on a set of variables. They are used to make probabilistic distribution insightful and structured: instead of representing a probabilistic distribution with $n$ variables in a table with $2^n$ entries, a graph with $n$ nodes can do the same. This can be done with the use of *Markovian Parents*:

**Definition 2.3.1.** ***Markovian Parents*** Let $V = \{X_1, ..., X_n\}$ be an ordered set of variables, and let $P(v)$ be the joint probability distribution on these variables. A set of variables $PA_j$ is said to be the Markovian parents of $X_j$ if $PA_j$ is a minimal set of ascendants of $X_j$ that renders $X_j$ independent of all its other ascendants. In other words, $PA_j$ is any subset of $\{X_1, ..., X_{j-1}\}$ satisfying

$$P(x_j \mid pa_j) = P(x_j \mid x_1, ..., x_{j-1})$$

and such that no proper subset of $PA_j$ satisfies this. ■

This definition assigns to each variable $X_j$ a set of preceding variables $PA_j$ that are sufficient for determining the probability of $X_j$. This can be represented in a graph, in which from each node in the parent set $PA_j$, an edge is drawn to the child node $X_j$.

Figure 2.2 shows an example of a Bayesian network. The graph expresses that the variable *Season* is the parent of both *Sprinkler* and *Rain*, and that it is not a parent of the variables *Wet* and *Slippery.* The graph is quite intuitive: the current season influences the probability of rain and the probability of the sprinkler being on. Both rain and a sprinkler influence the probability of having a wet pavement, and a wet pavement influences the probability of a slippery pavement.

If we write out the probability distribution of the DAG in figure 2.2, we come to the following:

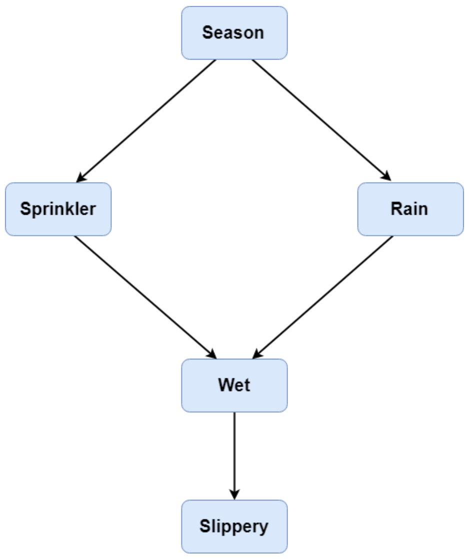


Figure 2.2: A Bayesian network with five variables.

$$P(Season, Sprinkler, Rain, Wet, Slippery) = P(Season) \cdot P(Sprinkler \mid Season) \cdot P(Rain \mid Season) \cdot P(Wet \mid Sprinkler, Rain) \cdot P(Slippery \mid Wet)$$

where each variable is only conditioned on its parent(s). This is a much smaller distribution than letting each variable condition on each other variable, which is the following probability distribution:

$$P(Season, Sprinkler, Rain, Wet, Slippery) = P(Season \mid Sprinkler, Rain, Wet, Slippery) \cdot P(Sprinkler \mid Season, Rain, Wet, Slippery) \cdot P(Rain \mid Season, Sprinkler, Wet, Sprinkler) \cdot P(Wet \mid Season, Sprinkler, Rain, Slippery) \cdot P(Slippery \mid SeasonSprinkler, Rain, Wet)$$

Looking at Figure 2.2 again, we can see and intuitively acknowledge that the season influences the probability that the pavement is wet: the probability of a wet pavement is much higher in autumn than in the summer. However, if we condition on the amount of rain ("given *Rain*"), learning about which season it is will not have any effect on the probability of a wet pavement, as the amount of rain is a much more direct indicator. We say that the variable *Wet* is *independent* of *Season* given *Rain*. You can read off the independence relation between *Season* and *Wet* from the graph in Figure 2.2. When using graphs, we say that two variables that are independent are *d-separated* from each other:

**Definition 2.3.2.** ***D-separation*** A path $p$ is said to be d-separated (or blocked) by a set of nodes $Z$ if and only if

1. $p$ contains a chain $I \rightarrow M \rightarrow J$ or a fork $I \leftarrow M \rightarrow J$ such that the middle node $M$ is in $Z$, or

2. $p$ contains an inverted fork (or collider) $I \rightarrow M \leftarrow J$ such that the middle node $M$ is *not* in $Z$ and such that no descendant of $M$ is in $Z$.

A set $Z$ is said to d-separate $X$ from $Y$ if and only if $Z$ blocks every path from a node in $X$ to a node in $Y$. ■

So, in chains like $I \rightarrow M \rightarrow J$ and forks like $I \leftarrow M \rightarrow J$, variables $I$ and $J$ are marginally dependent, but become independent of each other once we condition on $M$ (i.e. know the value of $M$). So, if we already know the value of $M$, learning about $I$ has no effect on the probability of $J$: if we know about the amount of rain, which season it is will not affect the probability of a wet pavement. If we look at inverted forks like $I \rightarrow M \leftarrow J$, we say that $I$ and $J$ are marginally independent, but become dependent once we condition on $M$. We see such an inverted fork in Figure 2.2, with the variables $Sprinkler \rightarrow Wet \leftarrow Rain$. If we learn that the pavement is wet, *Sprinkler* and *Rain* become dependent: refuting one of these explanations ("the sprinkler is off") increases the probability of the other ("it probably rains").

## 2.4 Causal Models

When there is an arrow between two variables in a Bayesian network $A \rightarrow B$, we do not say that $A$ *causes* $B$. The arrow merely indicates the *influence* of the probability of $A$ on $B$. Looking back at Figure 2.2, it would be rather odd to say that the season causes the sprinkler to be on or off. Instead, the season has influence on the probability of the sprinkler being on or off. This changes for causal models. Causal models use a stronger notion of association between two variables, saying that $A$ is really a *cause* of $B$. If $A$ is the only variable that points at $B$, we also say that $A$ is the only cause of $B$ (given that specific model).

You can also think of causal models in terms of hypothetical experiments. An arrow from $A$ to $B$ means that if we vary the value of variable $A$ only, we would see a change in the probability of $B$. No arrow from $A$ to $B$ means that varying the value of $A$ has no effect on the value of $B$.

In order to define causal models properly, we first need to define the *signature*:

**Definition 2.4.1.** ***Signature*** A signature $S$ is a tuple $(U, V, R)$, where $U$ is a set of exogenous variables, $V$ is a set of endogenous variables, and $R$ is a function that associates with every variable $Y \in U \wedge V$ a nonempty set $R(Y)$ of possible values for $Y$, that is, the set of values over which $Y$ ranges. ■

**Definition 2.4.2.** ***Causal Model*** A causal model $M$ is a pair $(S, F)$, where $S$ is a signature and $F$ defines a function that associates with each endogenous variable $X$ a structural equation $F_X$, that gives the value of $X$ in terms of the values of the other variables. ■

A specific kind of causal model is a *probabilistic causal model.* In order to define it properly, we first need to define the *context*:

**Definition 2.4.3.** ***Context*** The context of a causal model is a specific setting $\vec{u}$ of values of the exogenous variables in the causal model. ■

**Definition 2.4.4.** ***Probabilistic Causal Model*** A probabilistic causal model $M = (S, F, I, Pr)$ is a causal model together with a probability $Pr$ on contexts.■

A causal model is represented by a set of variables and a set of equations, that show how each variable depends upon its parents. The set of equations induces a graph with parent- and child-relations: if the equation of variable $B$ contains the value of variable $A$, then we can conclude that variable $A$ must be a parent of variable $B$. This way, causal models use graphs as a tool to visualize the relationships between its variables.

In this thesis, it is assumed that each variable has an error term, which expresses the probability that certain variables in the equations are unobserved. Each variable equation has the following form:

$$x_i = f_i(pa_i, u_i) \qquad\qquad i = 1, ..., n$$

in which $pa_i$ is the set of immediate causes of $x_i$ (its parents), and where $u_i$ represents the error term.

A simple causal model $M$ with five variables can be represented as follows in five equations:

$$\begin{aligned}
X_1 &= u_1 \\
X_2 &= f_2(x_1, u_2) \\
X_3 &= f_3(x_1, u_3) \\
X_4 &= f_4(x_2, x_3, u_4) \\
X_5 &= f_5(x_3, u_5)
\end{aligned}$$

A specification of the functions in $M$ could be as follows:

$$\begin{aligned}
X_1 &= u_1 \\
X_2 &= 5x_1 + u_2 \\
X_3 &= 3x_1 + u_3 \\
X_4 &= 2x_2 + 6x_3 + u_4 \\
X_5 &= 3x_3 + u_5
\end{aligned}$$

This clearly shows how the graph of $M$ should be composed: variable $X_1$ does not have a parent, variable $X_2$ has variable $X_1$ as single parent, etc. The complete graph of $M$ is represented in figure 2.3. Figure 2.4 represents the graph of $M$ including the error terms $(E\_X_1, ..., E\_X_5)$ on each variable.

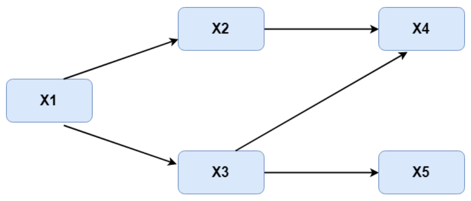


Figure 2.3: The complete graph of causal model $M$.

A difference between Bayesian networks and causal models is that Bayesian network can only tell the probability of an event, while causal models allow us to perform *interventions* on a model.

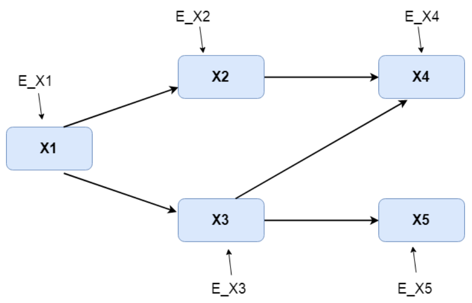


Figure 2.4: The graph of causal model $M$ including error terms.

**Definition 2.4.5.** ***Intervention*** Given a causal model $M$, an intervention on a variable $A$ in $M$ is to set the value of $A$ by a process that overrides the usual causal structure, without interfering with the causal processes determining the other variables. ■

An intervention is an external change to a model, like an order to turn off the sprinkler to preserve water. Interventions are represented by *do*-statements: $do(Sprinkler = Off)$. Note the difference between the statements "$do(Sprinkler = Off)$" and "$Sprinkler = Off$": the first is an action, led by external events like the order to preserve water, and the latter is a simple observation that it happens to be the case that the sprinkler is off.

We write an intervention as $X \leftarrow x$, which means that the value of variable $X$ has been set to $x$. We can also write it for a set of variables, denoted by $\vec{X}$, as follows: $\vec{X} \leftarrow \vec{x}$, which means that the values of the variables in $\vec{X}$ are set to $\vec{x}$. With an intervention, you actually create a new causal model, as every incoming edge on the intervened variable is removed. This new model is denoted as $M_{\vec{X} \leftarrow \vec{x}}$. The probability of the value of the variable following the intervention is set to 1, because the external change determines the value of the variable with certainty. Likewise, the corresponding equation changes, as the value of the variable does not depend on its parents anymore.

For example, if we intervene on causal model $M$ in Figure 2.3 by setting the value of variable $X_3$ to 11, we delete the edge from $X_1$ to $X_3$ and change the equation of $X_3$ from $X_3 = 3x_1 + u_3$ to $X_3 = 11$. The resulting graph is shown in Figure 2.5. The outgoing edges from the intervened variable remains the same, since $X_3$ is still a cause of variables $X_4$ and $X_5$, regardless of the intervention. Likewise, the rest of the equations in $M$ stay the same.

Given a signature $S = (U, V, R)$, a *primitive event* is a formula of the form

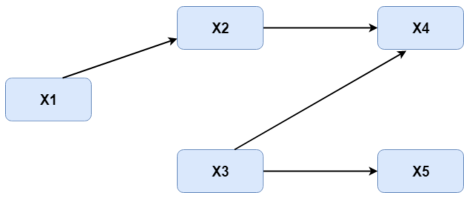


Figure 2.5: An intervention performed on causal model $M$: the arrow between $X_1$ and $X_3$ has been removed.

$X = x$, for $X \in V$ and $x \in R(X)$. In other words, this is the observation that variable $X$ has the value $x$. A causal formula over $S$ has the form $[Y_1 \leftarrow y_1, ..., Y_k \leftarrow y_k]\phi$, where

1. $\phi$ is a Boolean combination of primitive events,
2. $Y_1, ..., Y_k$ are distinct variables in $V$, and
3. $y_i \in R(Y_i)$

This formula can be abbreviated to $[\vec{Y} \leftarrow \vec{y}]\phi$. Intuitively, it means that $\phi$ would hold if $Y_i$ would be set to $y_i$, for $i = 1, ..., k$.

Finally, a causal formula $\psi$ is true or false in a causal model, given a certain context. A true causal formula $\psi$ in causal model $M$ given context $\vec{u}$ is denoted as $(M, \vec{u}) \vDash \psi$.

In chapter 6, we write $M(\vec{u})$ to denote the unique element of $R(V)$ such that $(M, \vec{u}) \vDash V = \vec{v}$. The same holds given an intervention $\vec{Y} \leftarrow \vec{y}$, where $M(\vec{u}, \vec{Y} \leftarrow \vec{y})$ denotes the unique element of $R(V)$ such that $(M, \vec{u}) \vDash [\vec{Y} \leftarrow \vec{y}](V = \vec{v})$.

# 3 | Causal Search Algorithms

Causal search algorithms, or causal discovery algorithms, are algorithms that examine data and find a causal model based on it. Causal search algorithms do not test certain hypotheses, so whether a certain model would be a good fit to the data, but instead they consider all possible models, test them and select the set of models consistent with the data.

There are several parts at play in causal discovery. First, there are several assumptions that every causal search algorithm makes, and some assumptions that are specific to certain algorithms. Next, there are three kinds of causal search algorithms. I will describe these in the following sections.

## 3.1 Assumptions in Causal Discovery

There are several assumptions that are held by causal search algorithms: some by all, some only by a few.

### Causal Markov Condition

All causal search algorithms assume that the "true graph", which is the graph to be found, satisfies the *Causal Markov Condition* (CMC). This means that every variable $X$ in the set of variables $V$ is independent of its non-effects, the variables that are not descendants of $X$, conditioned on its direct causes.

**Definition 3.1.1.** ***Causal Markov Condition*** Let $G$ be a causal graph with vertex set $V$ and the probability distribution $P$ over the vertices in $V$ generated by the causal structure represented by $G$. $G$ and $P$ satisfy the Causal Markov Condition if and only if for every $X$ in $V$, $X$ is independent of all the variables in $V$ that are not its parents and not its descendants, given the parents of $X$.

Formally:

$$X \perp\!\!\!\perp V \backslash (Descendants(X) \cup Parents(X)) \mid Parents(X).$$

■

If we look at the simple causal model in Figure 3.1, we can conclude on the basis of the CMC that $B$ must be independent of $C$ given $A$, because $C$ is not a descendant from $B$ and $A$ is the parent of $B$.

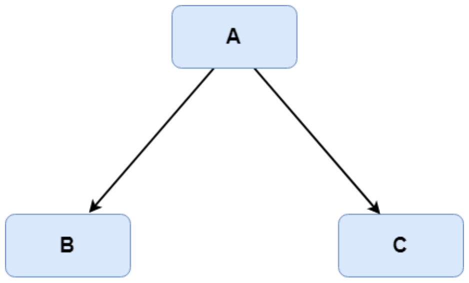


Figure 3.1: A simple causal model.

### Faithfulness Assumption

Most search algorithms also assume Faithfulness: the only independencies among the variables in $V$ are those entailed by the CMC.

**Definition 3.1.2.** ***Faithfulness*** A causal graph $G$ and a probability distribution $P$ over $G$ are faithful to one another if all and only the conditional independence relations true in $P$ are entailed by the Causal Markov Condition applied to $G$. ■

It is possible for the faithfulness condition not to hold, in the case when there are other independence relations besides the ones entailed by the CMC.

Together, CMC and Faithfulness imply a tight connection between the structure of the causal graph and the conditional independencies in the data. It is this connection that is useful for search algorithms. For example, if we were to look for the graph in Figure 3.1, we would first measure the variables $A$, $B$ and $C$. We can then find that $B$ and $C$ are dependent (denoted as $B \not\perp\!\!\!\perp C$), that $A$ and $C$ are dependent ($A \not\perp\!\!\!\perp C$), and that $A$ and $B$ are dependent ($A \not\perp\!\!\!\perp B$), while

$B$ and $C$ are independent given $A$ (denoted as $B \perp\!\!\!\perp C \mid A$). If we know this, and we assume CMC and Faithfulness, we can infer that one of the following three causal structures is the true one:

$$B \rightarrow A \rightarrow C$$
$$B \leftarrow A \leftarrow C$$
$$B \leftarrow A \rightarrow C$$

These three causal structures form a *Markov Equivalence Class*, which will be discussed in detail in the next chapter.

CMC and Faithfulness thus do not provide a unique causal structure in most cases, but instead provide a set of possible causal structures. These can be united in a *pattern*, which is a directed graph with some or only undirected edges. An undirected edge means that the arrow of the edge can be pointed in both ways. The pattern of the graph in Figure 3.1 would be the one depicted in Figure 3.2, as the edges $A - B$ and $A - C$ can be pointed both ways. We only know that the structure can not be $B \rightarrow A \leftarrow C$.

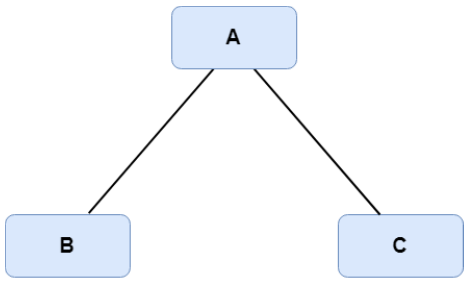


Figure 3.2: The pattern of the causal model in Figure 3.1.

### Linearity

Linearity is an assumption about the functional form of a causal relation. It requires that the value of a variable is determined by a linear sum of the values of its causes plus some error term. Graphs that are not linear can be graphs that involve some threshold, above or below which there is no, or a different causal effect, or graphs with a non-monotonic effect. The justification for linearity is that it is the most common relationship between variables, and that it is computationally simple: the values of the variables can easily be found through regression analysis. Linearity is an assumption adopted by some causal discovery algorithms.

For example, when we model the relationship between body weight and weeks of exercise, we can end up with quite different looking graphs, each belonging to a different graph form. A linear graph consists of a linear sum of the variables plus some error term. It would look as follows: the more weeks you exercise, the less you will weigh. The graph is depicted in Figure 3.3.

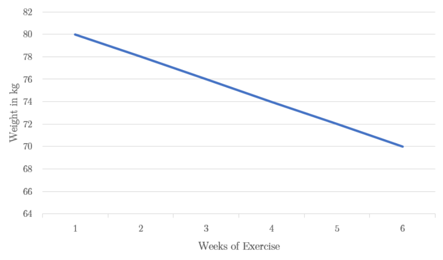


Figure 3.3: A linear relationship between *Weight* and *Weeks of Exercise*.

However, it could also be that at first you do not lose any weight, and then, in the third week of exercise, your weight suddenly drops a lot. We call this form of graph a threshold graph: once the threshold has been reached, the value changes. So, once the threshold of three weeks of exercise has been reached, your weight drops. Such a graph is depicted in Figure 3.4.

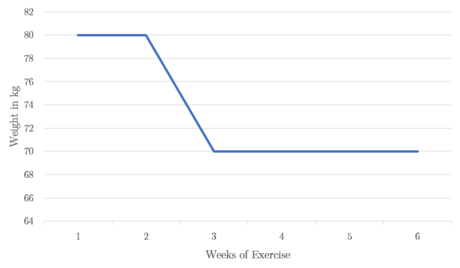


Figure 3.4: A threshold relationship between *Weight* and *Weeks of Exercise*.

Another way could be that when you start exercising, you first add some weight, due to your body functioning differently. Then, after about three weeks, your weight starts to drop again. We call this form of graph a non-monotonic graph, as there is no clear increasing or decreasing order of the graph. Such a graph is depicted in Figure 3.5.

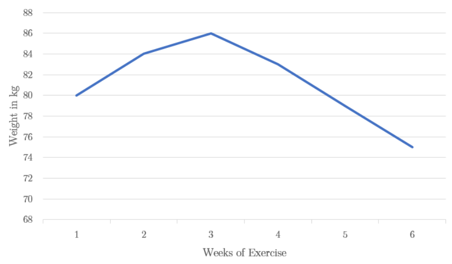


Figure 3.5: A non-monotonic relationship between *Weight* and *Weeks of Exercise*.

### Gaussianity

The linearity assumption is usually combined with the assumption that the error term of a variable has a Gaussian distribution, i.e. a bell-curve distribution. This has to do with the computational simplicity of it: combinations of Gaussian distributions together form joint Gaussian distributions as well. If you take all the individual error terms to represent the minor individual influences not accounted for in the model, then the error term for a large sample can be expected to be Gaussian as well.

### Causal Sufficiency

Causal sufficiency is a strong assumption to make. Causal sufficiency states that there are no unmeasured common causes, i.e. that the model is complete. So, if you have only two variables $X$ and $Y$ in your causal model, causal sufficiency states that there is no unmeasured common cause of $X$ and $Y$. This means that if a dependency between $X$ and $Y$ is observed, the only possible structures are the following:

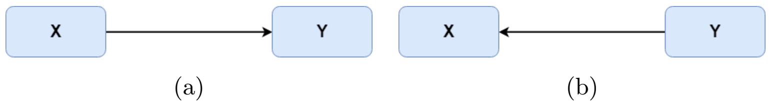


Figure 3.6: Possible structures between dependent variables $X$ and $Y$ with no unmeasured common cause.

However, if we drop the assumption of causal sufficiency, three more models become possible, in which $L$ represents an unobserved variable, as depicted in Figure 3.7.

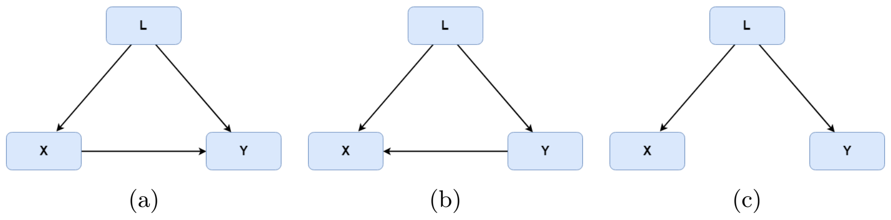


Figure 3.7: Possible structures between dependent variables $X$ and $Y$ with unmeasured common cause $L$.

In other words, the assumption of causal sufficiency makes the search space smaller, i.e. there are fewer possible models. Even though there is rarely a proper justification for causal sufficiency, it is necessary for causal discovery algorithms to adopt the assumption: you have to assume that the variables provided in the data are the only variables at play in the causal phenomenon, because otherwise it is as good as impossible to determine causal relationships between the variables.

## 3.2 Types of Causal Search Algorithms

There are three different approaches to causal discovery. I will briefly explain each of them.

### Constraint-based algorithms

Constraint-based algorithms focus directly on the connection between graphs and the implied independence facts. These algorithms search for the set of causal graphs that imply exactly the conditional independencies found in the data. This is done through a sequence of hypothesis tests. In the most extensive manner, this is done by generating every possible graph on a set of variables $V$, and testing the implied conditional independencies for each graph against the data. The graph with the best score is crowned the best graph. However, most algorithms use a heuristic to explore the space of possible DAG models in an efficient way, in order to prevent having to test each and every one of them. Algorithms that belong to the constraint-based algorithms are the SGS algorithm, the PC algorithm and Fast Causal Inference (FCI). The difference between the SGS algorithm and the PC algorithm is that the SGS algorithm tests every possible graph on a set of variables, while the PC algorithm thins out the set of possible graphs before performing hypothesis tests, by using $n$-order conditional independence relations. More information on this can be found in chapter 5, where we discuss the PC algorithm in detail.

**Score-based algorithms**

Score-based algorithms compare models on the basis of some score of model fit. The most common score is the Bayesian Information Criterion (BIC) score, which approximates the posterior probability of the model given the data. Scoring can be done on every possible model, but most algorithms use a heuristic to prevent having to score every possible model. For this, a "greedy" algorithm is typically used. Algorithms that belong to the score-based algorithms are Greedy Equivalence Search (GES) and Fast Greedy Equivalence Search (FGES). The main difference between constraint-based algorithms and score-based algorithms is that constraint-based algorithms turn out be much faster than score-based algorithms. However, recent simulation studies show that score-based algorithms are generally more accurate than constraint-based algorithms that make the same assumptions for datasets with small sample sizes.

**Causal search algorithms with semi-parametric assumptions**

As the name says, algorithms that fall into this category make some stronger assumptions to learn causal relationships more efficiently and in more detail. These algorithms do not rely on the Faithfulness assumption, but instead adopt some more specific assumptions. For example, when the data are generated by a linear mechanism but with non-Gaussian error terms, Linear Non-Guassian Model (LiNGaM) algorithms can find the causal structure using independent components analysis. The drawback of the algorithms in this approach of causal discovery is that the semi-parametric assumptions that are used are usually difficult to test. In addition, the algorithms usually require a much larger sample size compared to constraint-based and score-based algorithms. This is why these algorithms are less developed in comparison with constraint-based algorithms and score-based algorithms.

In this thesis, I will focus on the PC algorithm, which belongs to the class of constraint-based algorithms. A detailed discussion of the PC algorithm can be found in chapter 5.

# 4 | Markov Equivalence Classes

In this chapter I will describe the use of Markov Equivalence Classes. First, I will introduce Markov Equivalence, upon which the definition of Markov Equivalence Classes follows. The chapter ends with an extensive example.

Note that from now on in the thesis, it is assumed that the Causal Markov Condition (CMC) holds, which is explained in the previous chapter.

## 4.1 Markov Equivalence

When the use of DAGs (Directed Acyclic Graphs) became more popular in a wide variety of research fields, one problem arose: it turned out that it was quite common that two different causal models produce the same results and are equally predictive. When this is the case, these causal models are said to be *Markov equivalent*:

**Definition 4.1.1.** ***Markov Equivalence*** Let $M_1$ and $M_2$ be two DAGs with the same set of nodes $V$. $M_1$ and $M_2$ are Markov equivalent if for every three mutually disjoint subsets $A$, $B$, $C \subseteq V$, $A$ and $B$ are d-separated by $C$ in $M_1$ if and only if $A$ and $B$ are d-separated by $C$ in $M_2$. ■

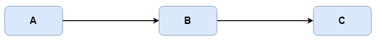
**Theorem 4.1.1.** Two DAGs are Markov equivalent if and only if they have the same independence statements.

**Theorem 4.1.2.** Two DAGs have the same independence statements if and only if they have the same adjacencies and the same colliders.

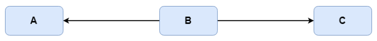
The proof of theorem 4.1.2 can be found in Verma and Pearl (1991).

Markov Equivalence can be shown with the two causal models in Figure 4.1a and 4.1b. Even though their structure differs, the probability distribution of both models turns out to be the same. The probability distribution of model 1 is $P(A)$, $P(B \mid A)$ and $P(C \mid B)$. The probability distribution of model 2 is $P(A \mid B)$, $P(B)$, and $P(C \mid B)$. To show that the two models are Markov equivalent, we apply the definition of conditional probability:

$$P(A)P(B \mid A) = P(AB) = P(B)P(A \mid B)$$



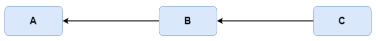
(a) Causal model 1.



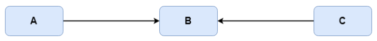
(b) Causal model 2.

Figure 4.1: Two Markov Equivalent causal models.

What this means is that the two models in Figure 4.1 have the same d-separation, namely $A \perp\!\!\!\perp C \mid B$ (A is d-separated from C by B). We call a statement like $A \perp\!\!\!\perp C \mid B$ an *independence statement*. When we look at causal models 3 and 4, shown in Figure 4.2a and 4.2b, we can derive their Markov equivalence with the models in Figure 4.1 through their independence statements: model 3 shows the independence statement $A \perp\!\!\!\perp C \mid B$, and is thus Markov equivalent with causal models 1 and 2, depicted in Figures 4.1a and 4.1b. Model 4 shows the independence statement $A \perp\!\!\!\perp C$, without conditioning on variable $B$. This is a different d-separation statement, which means that model 4 is not Markov equivalent with models 1, 2 and 3.



(a) Causal model 3.



(b) Causal model 4.

Figure 4.2: Two causal models that are not Markov Equivalent.

Markov equivalence thus comes down to having the same d-separations or independence statements. This can be shown by rewriting the probability distribution, which is what we did with the causal models in Figure 4.1 and 4.2. However, there is an easier way to check for similar independence statements, as described in Theorem 4.1.2: by checking for the same adjacencies and the same colliders between two models.

To check for this, you first have to check whether the two causal models have the same adjacencies. It is quite obvious that Markov equivalent models must have the same adjacencies: if model 1 has a link between $A$ and $B$, but model 2 does not have such a link, the two models cannot behave the same, and therefore not be Markov equivalent.

Second, you have to check whether the two causal models have the same colliders. We can see the importance of similar colliders when we look at the models in Figures 4.1 and 4.2. The difference between model 4 and the Markov equivalent models 1, 2 and 3 is that model 4 has two incoming edges at variable $B$. In models 1, 2 and 3, each variable has either one incoming edge, or no incoming edge at all. If we look back at the definition of d-separation (see chapter 2), the structure of two incoming edges is called a collider, or v-structure: with the collider $A \rightarrow B \leftarrow C$, variables $A$ and $C$ become dependent given $B$, while first being independent of each other. A collider behaves different than the other possible structures, which are chains and forks. We can therefore say that two graphs with the same colliders on the same three variables share the same independence statements. If three variables form a chain or a fork, the precise direction of the edges does not matter, as they have the same independence relation.

So, in order to check for Markov equivalence between two or more causal models, we can check for similar adjacencies and similar colliders between the models. Together, they ensure that the models have the same d-separations or independence statements, which makes them Markov equivalent.

## 4.2 Markov Equivalence Classes

With the definition of Markov equivalence, you can easily construct the *class* of Markov equivalent models.

**Definition 4.2.1.** ***Markov Equivalence Class*** The Markov Equivalence Class of a causal model $M$ is the class of all graphs that are Markov Equivalent with $M$. ■

In a Markov Equivalence Class, it is quite common that an edge can be directed multiple ways. If we look back at the Markov equivalent causal models in Figures 4.1 and 4.2, we see that the edge between $A$ and $B$ can be directed both $A \rightarrow B$ as well as $A \leftarrow B$. The direction of the edge between $A$ and $B$ is thus not determined in the Markov Equivalence Class of the causal models in Figures 4.1 and 4.2. We depict such an undetermined edge as $A - B$, without an arrow. As a consequence, this means that if there *is* a directed edge between two variables in a Markov Equivalence Class, then this edge is a determined edge, i.e. for all the causal models that are part of that Markov Equivalence Class, the edge is directed exactly that way. This is always the case for colliders in a causal model, as models that are Markov equivalent must have the same colliders.

The construction of the Markov Equivalence Class of a causal model consists of two steps:

1. First, the construction of the *rudimentary pattern* of the causal model. This is a partially directed graph, with all the arrowheads removed that do not form a collider.
2. Second, the construction of the *completed pattern*. In this step, you fill in the direction of arrowheads for the following edges:
   (a) Undirected edges with a variable on one side that already has an edge directed at it ($A - B \leftarrow C$) are directed the other way (as $A \leftarrow B \leftarrow C$), as not to point to this variable. This is to prevent creating a new collider that is not present in the causal model.
   (b) Undirected edges that can create a cycle are directed the other way, as not to create the cycle. This is because we are working with directed *acyclic* graphs.

   A more detailed version of how to direct the edges in a Markov Equivalence Class can be found in the PC algorithm, as described in Chapter 5.

Upon completing these steps, the completed pattern is the Markov Equivalence Class. However, note that a Markov Equivalence Class always asks for a right interpretation by its user. That is, the undirected edges can not be directed in a model in such a way that a new collider or a cycle is created. On the other hand, we know with 100% certainty that the directed edges must be directed as they are, in all models belonging to the Markov Equivalence Class.

### 4.2.1 Markov Equivalence Class Example

The following example shows the construction of the Markov Equivalence Class of causal model $M_1$, as depicted in Figure 4.3.

First, we create the rudimentary pattern by removing all arrowheads that do not form a collider. This means that only the two incoming edges on variable $D$ remain, which is shown in Figure 4.4.

What rests is filling in the direction of the edges as to not create a loop or a new collider. From the graph in Figure 4.5, no loops can be created, as the

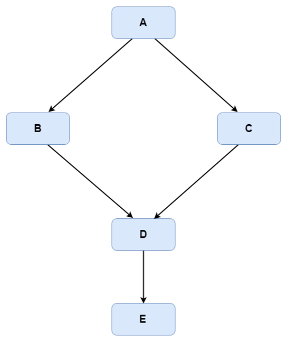


Figure 4.3: Causal model $M_1$.

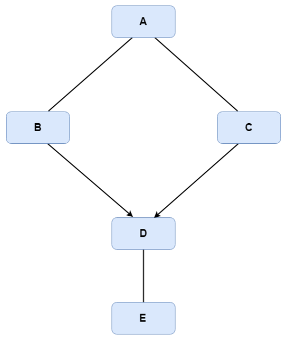


Figure 4.4: The rudimentary pattern of $M_1$.

collider on variable $D$ prevents this. However, a new collider can be created, when the edge $D - E$ is pointed to variable $D$ (as $D \leftarrow E$). This is not allowed, which means that the arrow must be pointed the other way ($D \rightarrow E$). The completed pattern is depicted in Figure 4.5.

This makes the Markov Equivalence Class of $M_1$. We see that the edges between $A - B$ and $A - C$ are not directed, which means that there are other causal models belonging to this Markov Equivalence Class, in which $A - B$ and $A - C$ are directed differently than in $M_1$. They are shown in Figure 4.6.

The specific causal model with the edges directed as $A \leftarrow B$ and $A \leftarrow C$ does not belong to the Markov Equivalence Class of $M_1$, because it contains a new collider on variable $A$, not included in the Markov Equivalence Class. We can thus say with certainty that the Markov Equivalence Class in Figure 4.5 consists only of the three causal models depicted in Figure 4.3 and Figure 4.6.

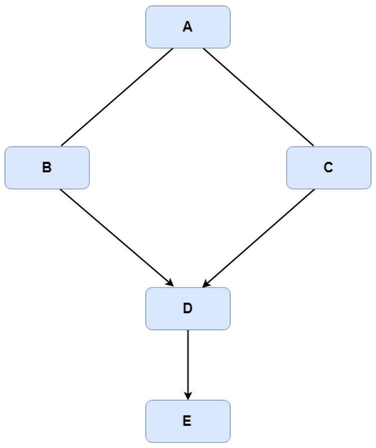


Figure 4.5: The completed pattern or the Markov Equivalence Class of $M_1$.

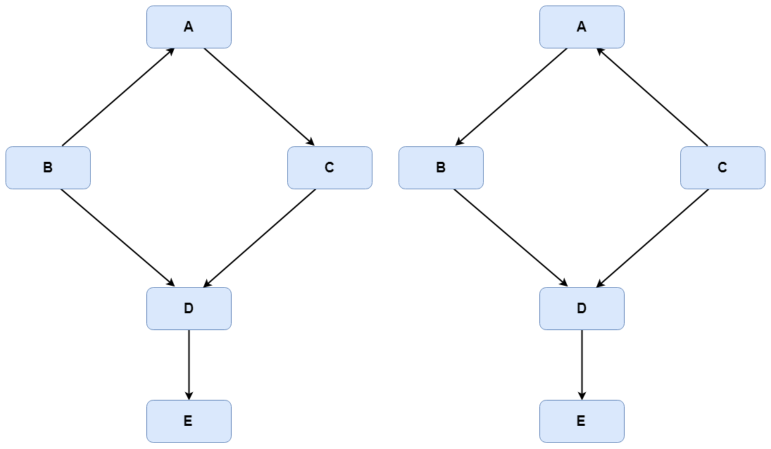


Figure 4.6: The other two causal models belonging to the Markov Equivalence Class of $M_1$.

# 5 | The PC Algorithm

The PC algorithm is one of the most well-known and most used causal search algorithms. It is also one of the oldest search algorithms: in 1991 Peter Spirtes and Clark Glymour wrote a paper in which they described the PC algorithm as the successor of the SGS algorithm. They named the algorithm after themselves: **P**eter and **C**lark. In this chapter, I will first describe how to find independence statements in data, which is used as the input for the PC algorithm. Then, I will describe the PC algorithm in pseudo code, followed by an extensive example of its use.

## 5.1 Independence facts from data

The PC algorithm constructs a causal model on the basis of data. However, as input, the PC algorithm uses "independence and conditional independence facts about the data", as quoted from Spirtes and Glymour (1991). What are these, and how do you find them?

(Conditional) independence facts are the independencies found in the data. A conditional independency is a d-separation: conditioned on some variable, two other variables are found to be independent. An independency does not need to be conditional, as it can also be the case that two variables have no correlation at all. These variables are then independent from each other, without the need to condition on another set of variables. We call this kind of independence a *zero order independency.* Conditional independencies that condition on just one variable are called *first order independencies*, conditional independencies that condition on a set of two variables are called *second order independencies*, etc.

Independencies are found using the correlation coefficient. This is a value that describes the correlation between two variables. It is a number between -1 and 1, where -1 means a perfect negative correlation (negative slope), and

1 means a perfect positive correlation (positive slope). You can compute the correlation coefficient $r$ using the following formula:

$$r = \frac{1}{n-1} \sum (\frac{x_i - \bar{x}}{s_x})(\frac{y_i - \bar{y}}{s_y})$$

where $n$ is the number of variables, $\bar{x}$ is the sample mean of the variables and $s_x$ is the sample standard deviation of the variables. It is also possible to compute the correlation coefficient in programming language R, using the linear regression function. You can compute the correlation coefficient on a set of variables, or on only two variables. We will use the latter, since we are looking for the correlation and possible independency between two variables. Each correlation coefficient has a p-value, which describes the significance of the correlation. Generally, a p-value above 0.05 is considered not significant; anything with a p-value under 0.05 is considered significant. There are two ways to discover an independency between two variables using the p-value of the correlation coefficient:

1. The p-value of the correlation coefficient between variables $A$ and $B$ is above 0.05, i.e. not significant. This means that there is no correlation between the two variables, and thus that they are independent from each other. We write this as $A \perp\!\!\!\perp B$.

2. Discovering a conditional independency consists of two steps:

   (a) First, you find that the correlation coefficient between variables $A$ and $C$ is below 0.05, which means that the correlation is significant.

   (b) However, when you control for another variable $B$, you find that the correlation between $A$ and $C$ is not significant anymore, i.e. the p-value increases above 0.05.

   This means that $A$ is independent from $C$ given $B$. We write this as $A \perp\!\!\!\perp C \mid B$.

When checking the independency between each pair of variables in the data, you generate a list of independence and conditional independence facts about the data. This is what the PC algorithm needs.

## 5.2 The PC Algorithm - pseudocode

The PC algorithm uses the list of independence facts as input. The facts are divided into sets of zero order independencies, first order independencies, second order independencies, etc. The following is the pseudocode of the PC algorithm:

1. Start with a complete undirected graph, i.e. every node is connected to every other node with an undirected edge.
2. Set $n = 0$
   (a) Check for $n = 0$ order independencies, i.e. independencies that are not conditional. For these independencies, remove the direct edges between the nodes.
3. Set $n+ = 1$
   (a) Check for all $n = 1$ order independencies. These are independencies of the sort "$X \perp\!\!\!\perp Z \mid Y$", which is an independency given one variable. For these independencies, remove the direct edges between the independent nodes. So, for $X \perp\!\!\!\perp Z \mid Y$, remove the edge from $X$ to $Z$. Add each conditional to the variable 'Sepset' of the two independent nodes. So, for $X \perp\!\!\!\perp Z \mid Y$, add $Y$ to $Sepset(X, Z)$.
4. Set $n+ = 1$
   (a) Check for $n = 2$ order independencies. These are independencies of the sort "$X \perp\!\!\!\perp Z \mid \{Y, W\}$", which is an independency given two variables. For these independencies, remove the direct edges between the independent nodes. So, for $X \perp\!\!\!\perp Z \mid \{Y, W\}$, remove the edge from $X$ to $Z$. Add each conditional to the variable 'Sepset' of the two independent nodes. So, for $X \perp\!\!\!\perp Z \mid \{Y, W\}$, add $Y$ and $W$ to $Sepset(X, Z)$.
5. Set $n = n + 1$
6. Keep doing this for higher values of $n$. Repeat until no more independency statements are left.
7. Directing edges part 1: for each triple of nodes $X, Y, Z$, such that the pair $X$ and $Y$ are adjacent and the pair $Y$ and $Z$ are adjacent, but $X$ and

$Z$ are not adjacent, iff $Y$ is not in $Sepset(X, Z)$ (i.e. $X$ and $Z$ are not independent conditioned on $Y$), orient $X - Y - Z$ as $X \rightarrow Y \leftarrow Z$.

8. Directing edges part 2: if $X \rightarrow Y$, $Y$ and $Z$ are adjacent, $X$ and $Z$ are not adjacent, and there is no arrowhead at $Y$, then orient $Y - Z$ as $Y \rightarrow Z$.
9. Directing edges part 3: if there is a directed path from $X$ to $Y$, and an edge between $X$ and $Y$, then orient $X - Y$ as $X \rightarrow Y$.
10. Repeat steps 8 and 9 until no more edges can be oriented.
11. End. This produces the final pattern.

In many ways, the PC algorithm shows similarities with Markov Equivalence Classes. First, the output of the PC algorithm is a pattern. This is a partially directed graph, in which undirected edges stand for a conditional dependence between the two variables they connect, however uncertain about the direction of the dependence. An undirected edge between variable $X$ and $Y$ thus means that $X$ could be a cause of $Y$, or $Y$ could be a cause of $X$. It is only known that there *is* a conditional dependence between $X$ and $Y$. On the other hand, a directed edge between two variables means that the direction of the conditional dependence is certain. This is similar to Markov Equivalence Classes: in Markov Equivalence Classes, an undirected edge also means that the direction of dependency could be both ways. In fact, the pattern that the PC algorithm produces *is* a Markov Equivalence Class: the pattern stands for a class of possible causal models compatible with the data.

A second similarity between the PC algorithm and Markov Equivalence Classes is the way the final pattern is produced, specifically steps 7 through 10, in which the edges are directed. In step 7, the algorithm looks for colliders. As shown in Chapter 4, all colliders in a causal model must be included in the Markov Equivalence Class or final pattern, as they represent a different independence statement in comparison with chains and forks. In step 8, the algorithm makes sure that no other colliders are created, as all colliders in the causal model have already been found in step 7.

In short, the similarities between the PC algorithm and Markov Equivalence Classes are found in the output of the PC algorithm, which simply is a Markov Equivalence Class, and in the steps to produce the output.

### 5.2.1 PC Algorithm Example

Let's look at an example of the working of the PC algorithm. For this, we will trace the discovery of the causal model depicted in Figure 5.1, called the "true graph".

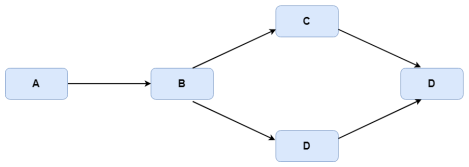


Figure 5.1: The true graph.

The PC algorithm starts with the complete undirected graph, thus connecting all variables in the model with all the other variables. This is shown in Figure 5.2.

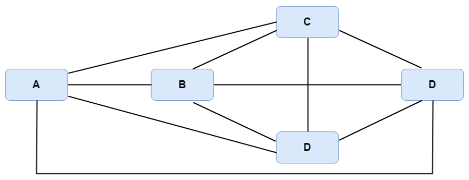


Figure 5.2: The complete undirected graph of Figure 5.1.

The next step is checking for $n$-order independencies. In the data belonging to this causal model, there are no zero-order independencies, i.e. no unconditional independencies. There are some first-order independencies however:

$$A \perp\!\!\!\perp C \mid B$$
$$A \perp\!\!\!\perp E \mid B$$
$$A \perp\!\!\!\perp D \mid B$$
$$C \perp\!\!\!\perp D \mid B$$

This means that the edges between $A$ and $C$, $A$ and $E$, $A$ and $D$, and $C$ and $D$ must be removed. This gives the graph depicted in Figure 5.3.

Next, there is only one second order independency:

$$B \perp\!\!\!\perp E \mid \{C, D\}$$

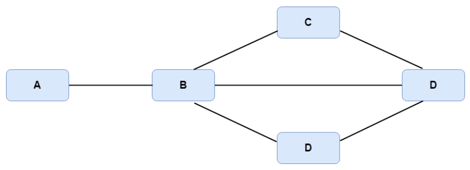


Figure 5.3: The complete undirected graph of Figure 5.1 with the first-order independencies removed.

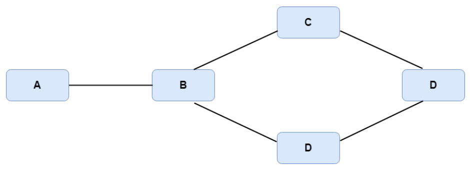


Figure 5.4: The complete undirected graph of Figure 5.1 with the first-order and second-order independencies removed.

This means that we need to remove the edge between $B$ and $E$. There are no further $n$-order independencies. The graph in Figure 5.4 is thus the skeleton of the causal model given the data. The variables $Sepset()$ are filled in these steps as well. They are as follows:

$$\begin{aligned} Sepset(A, C) &= \{B\} \\ Sepset(A, E) &= \{B\} \\ Sepset(A, D) &= \{B\} \\ Sepset(C, D) &= \{B\} \\ Sepset(B, E) &= \{C, D\} \end{aligned}$$

Next, we can start directing the edges. To find the colliders in the graph, we need to check all triples of nodes $X$, $Y$, $Z$ such that the pair $X$, $Y$ and the pair $Y$, $Z$ are adjacent, but $X$, $Z$ are not adjacent in the graph. They can simply be read off the skeleton depicted in Figure 5.4. The triples of nodes are as follows:

$$A - B - C$$
$$A - B - D$$
$$B - C - E$$
$$B - D - E$$
$$C - E - D$$
$$C - B - D$$

We can direct the triple $X$, $Y$, $Z$ as $X \rightarrow Y \leftarrow$ only if $Y$ is not in $Sepset(X, Z)$. The only triple for which this holds is $C - E - D$, as $E$ is not in $Sepset(C, D)$. We thus know that there is a collider on $E$, as depicted in Figure 5.5.

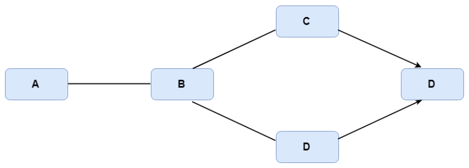


Figure 5.5: The discovered graph of Figure 5.1 with the PC Algorithm.

Lastly, we need to check whether there are any other edges we can direct. First, can we direct an edge as not to produce a new collider? Since there are no other edges at the collider variable $E$, this is not the case. Second, is there a directed path between two variables for which the edge between those two variables is not directed yet? This is also not the case. The graph produced in Figure 5.5 is thus the final pattern produced by the PC algorithm. Likewise, the graph depicted in Figure 5.5 is also the Markov Equivalence Class of the causal model shown in Figure 5.1.

# 6 | Abstracting Causal Models

In this chapter I will discuss the notion of abstraction in causal models. First, I will explain the intuition behind abstracting causal models. Then, I will discuss constructive $\tau$-abstraction, which is the type of abstraction considered in this thesis. The chapter ends with an example of constructive $\tau$-abstraction. All definitions in this chapter are based on Beckers & Halpern (2019) and Beckers, Eberhardt & Halpern (2019).

## 6.1 The idea of abstraction

There are many problems that we analyse at different levels of detail. For example, political scientist $A$ might analyse a national election by looking at all votes that have been cast, in order to find a pattern. On the other hand, political scientist $B$ might analyse the same election by looking at groups of votes that tend to behave the same: for example, by bundling the votes of well-educated people living in cities in one group, by bundling the votes of the middle class living in the countryside in another group, etc. Political scientist $B$ thus analyses the votes on the level of voting groups. These are different levels of understanding the same phenomenon, namely, the election. We can say that political scientist $B$ looks at a higher, or more abstract, level of detail, by bundling votes in groups. The single votes, analysed by political scientist $A$, are a representation of the election at a more detailed level, which we call the *micro-level*, or low-level. The groups of votes, analysed by political scientist $B$, are a representation of the election at a less detailed level, which we call the *macro-level*, or high-level. In general, the macro-level is an abstraction of the micro-level.

If we are interested in the causal relationships between variables, we can model various levels of abstraction in causal models. In an abstraction between causal models, you look at a causal phenomenon at two different levels of detail: the detailed level is modelled in the micro-level model, the less detailed level is modelled in the macro-level model. However, you have to be careful that the abstraction to the high-level model preserves the causal relationships in the micro-level model. For example, if you cluster variables $X$, $Y$ and $Z$ into a single variable with the value $X+Y+Z$, you do not want that different settings $(x, y, z)$ and $(x', y', z')$ such that $x+y+z = x'+y'+z'$ lead to different outcomes in the high-level model.

It is inherent to abstractions that you lose some information: the single value $X + Y + Z$ contains less information than the three values $X, Y, Z$ apart. A good abstraction only loses the inessential information of the low-level model. However, the question then remains what this inessential information is. To some extent, this will always be in the eye of the beholder. However, a good definition of abstraction will guide the user in making the right decisions. This will become clear in the next sections.

## 6.2 Constructive $\tau$-Abstraction

In this section, I will explain and define constructive $\tau$-abstraction, which is the only type of abstraction considered in this thesis. For this, I will follow Beckers & Halpern (2019) and Beckers, Eberhardt & Halpern (2019). However, a remark is in place. In Beckers & Halpern (2019), the definition of constructive $\tau$-abstraction is brought about by first discussing other kinds of abstraction, like $\tau$-abstraction, that are used in the definition of constructive $\tau$-abstraction. In this thesis, I will not do this, as the other kinds of abstraction are not relevant for my purpose. I will explain all elements that are needed for the definition of constructive $\tau$- abstraction. However, as the definition will not be entirely identical to the definition in Beckers & Halpern (2019), we consider it a special kind of constructive $\tau$-abstraction. The definition remains conceptually similar.

An abstraction on a micro-level model is guided by the $\tau$-function, which takes care of several things needed for an abstraction. First, the $\tau$-function abstracts the state space of the set of low-level endogenous variables to the state space of the set of high-level endogenous variables. We only focus on abstracting the endogenous variables here, because they make up the structure of a causal model and are therefore the variables to be discovered by causal search algorithms. If we look back at the definition of causal models in Chapter 2, we see that the set of endogenous variables is signified by $V$, and that $R$ is the function that associates with every variable the set of possible values, i.e. the state space of a variable: $R(V)$. The $\tau$-function that abstracts the endogenous variables then looks as follows:

$$\tau : R_L(V_L) \rightarrow R_H(V_H)$$

where the subscript $_L$ stands for the low-level causal model and the subscript $_H$ stands for the high-level causal model. $R_L(V_L)$ thus stands for the state space of the low-level variables. We adopt this notation in the rest of the thesis.

The $\tau$-function means that $\tau$ abstracts the endogenous variables of the low-level model to variables in the high-level model, together with the function that defines the set of values for each variable.

The type of abstraction that we consider here is a *constructive* abstraction, which is when the low-level variables are clustered in such a way that the clusters form the variables of the high-level model. Intuitively, this means that a variable in the high-level model captures the effect of a set of variables in the low-level model. For the $\tau$-function, this means that $\tau$ can only abstract a cluster of low-level variables to a single high-level variable. When this is the case, we say that $\tau$ is *constructive.*

**Definition 6.2.1.** ***Constructive*** $\boldsymbol{\tau}$**.** If $V_H = \{Y_1, ..., Y_n\}$, then $\tau : R_L(V_L) \rightarrow R_H(V_H)$ is constructive if $\tau$ is surjective, there exists a partition $P = \{\vec{Z}_1, ..., \vec{Z}_{n+1}\}$ of $V_L$, where $\vec{Z}_1, ..., \vec{Z}_n$ are nonempty, and mappings $\tau_i : R(\vec{Z}_i) \rightarrow R(Y_i)$ for $i = 1, ..., n$ such that $\tau = (\tau_1, ..., \tau_n)$; that is, $\tau(\vec{v}_L) = \tau_1(\vec{z}_1) \cdot ... \cdot \tau_n(\vec{z}_n)$, where $\vec{z}_i$ is the projection of $\vec{v}_L$ onto the variables in $\vec{Z}_i$, and $\cdot$ is the concatenation operator on sequences. ■

For each cluster of low-level variables, a mapping $\tau_i$ is thus defined to map the cluster to a high-level variable $Y_i$. The partition $P$ consists of $n + 1$ sets of variables, of which only $n$ sets are abstracted to the high-level model. This means that the variables that are placed in set $n + 1$ are the variables that are not abstracted to the high-level model, i.e. they are the marginalized variables.

The reason we demand $\tau$ to be surjective is that as we are going from the low-level model to the high-level model, we are taking away details from the low-level model. However, we still want to make sure that every high-level state has at least one low-level state.

As we use $\tau$ to abstract variables in the low-level model to variables in the high-level model, we also want to use $\tau$ to abstract the interventions in the low-level model to interventions in the high-level model. However, it turns out that not all interventions in the low-level model can be abstracted to an intervention in the high-level model. This is the case because as each variable in the low-level model is mapped to a $\vec{Z}_i$ in $P$, we can only abstract interventions that are performed on all variables in $\vec{Z}_i$. The following example will show why this is the case.

Let $X_1$, $X_2$, $X_3$ be variables in low-level model $M_L$, that are abstracted to variable $Y_1$ in high-level model $M_H$. All variables in $V_L$ and $V_H$ are binary. The value of $Y_1$ is the sum modulo 2 of the values of $X_1$, $X_2$ and $X_3$. If we look at the intervention $(X_1 \leftarrow 1)$ on $M_L$, we can ask what the corresponding intervention is on $M_H$. This means we have to pick a value for $Y_1$ that matches the intervention $(X_1 \leftarrow 1)$ on the low-level. If we pick $Y_1 = 1$, this value would turn out to be wrong if $X_2 = 1$ and $X_3 = 0$, as the sum modulo 2 of $X_1$, $X_2$ and $X_3$ makes 0. If on the other hand we pick $Y_1 = 0$, this value would turn out to be wrong if $X_2 = 1$ and $X_3 = 1$, as the sum modulo 2 makes 1. There is thus no intervention on the high-level variable $Y_1$ that matches with the low-level intervention $(X_1 \leftarrow 1)$, as the value of $Y_1$ depends on all three low-level variables $X_1$, $X_2$, $X_3$. This is why we can only abstract interventions that are performed on all variables mapped to a certain $\vec{Z}_i$. An example of such an intervention is the intervention $\vec{X} \leftarrow \vec{x}$ on $M_L$, where $\vec{X} = (X_1, X_2, X_3)$ and $\vec{x} = (1, 0, 1)$. This is abstracted to the high-level intervention $Y_1 \leftarrow 0$.

We call the interventions on the low-level variables for which this is the case, i.e. interventions that are performed on all variables mapped to a certain $\vec{Y}_i$, *allowed interventions.* This means that the set of interventions on the low-level model is restricted to the set of allowed interventions by the following conditions:

$\tau(\vec{X} \leftarrow \vec{x}) = \vec{Y} \leftarrow \vec{y}$ if and only if there exists some $a$ and $b$ in $1, ..., n$ such that:

- $\vec{X} \leftarrow \vec{x}$ can be written as $\vec{Z}_a \leftarrow \vec{z}_a, ..., \vec{Z}_b \leftarrow \vec{z}_b, \vec{Z}_c \leftarrow \vec{z}_c$ where $c = n + 1$ or $\vec{Z}_c$ is empty;
- $\vec{Y} \leftarrow \vec{y}$ can be written as $Y_a \leftarrow y_a, ..., Y_b \leftarrow y_b$;
- $\tau_a(\vec{z}_a) = y_a, ..., \tau_b(\vec{z}_b) = y_b$.

We can see how these restrictions follow from the definition of the constructive $\tau$ that abstracts the endogenous variables. The conditions say that all low-level interventions must be interventions on all variables in a cluster $\vec{Z}_i$ in $P$ and the $\tau$-function $\tau_i$ abstracts the low-level cluster-intervention value $\vec{z}_i$ to a high-level intervention value $y_i$. When a low-level intervention meets these requirements, we say that the intervention is allowed. For interventions on the high-level model, we do not need to define the set of allowed interventions, as every intervention in a high-level model will have a corresponding intervention in the low-level model. The topic of allowed interventions has been discussed in more detail in other papers, like Rubenstein et al. (2017) and Beckers & Halpern (2019).

We perform an abstraction on a Probabilistic Causal Model, which is a causal model with a probability distribution $Pr$ on contexts (see Chapter 2). A probability distribution on the context, or exogenous variables, of a causal model indirectly determines the probability distribution on the endogenous variables $R(V)$ as well, as the values of the exogenous variables determine the values of the endogenous variables. Concretely, this means

$$Pr(\vec{v}) = Pr(\{\vec{u} : M(\vec{u}) = \vec{v}\}).$$

The $M(\vec{u})$ notation is explained in chapter 2, just as the $M(\vec{u}, \vec{X} \leftarrow \vec{x})$ notation.

This also holds for any specific intervention, as each intervention $\vec{X} \leftarrow \vec{x}$ induces a probability $Pr^{\vec{X} \leftarrow \vec{x}}$ on $R(V)$ as follows:

$$Pr^{\vec{X} \leftarrow \vec{x}}(\vec{v}) = Pr(\{\vec{u} : M(\vec{u}, \vec{X} \leftarrow \vec{x}) = \vec{v}\}).$$

In the rest of the thesis, we will view $Pr$ as a distribution on both $R(U)$ and $R(V)$. The context should make clear which we intend.

Lastly, $\tau(Pr)$ "pushes up" the low-level distribution to a high-level distribution, as follows:

$$Pr(\vec{v}_H) = Pr(\{\vec{v}_L : \tau(\vec{v}_L) = \vec{v}_H\})$$

With all the elements explained, we can now say that for an abstraction, we require that for all allowed interventions $\vec{X} \leftarrow \vec{x}$ on the low-level model, we have that $\tau(Pr_L^{\vec{X} \leftarrow \vec{x}}) = Pr_H^{\tau(\vec{X} \leftarrow \vec{x})}$. This means that if you start at the low-level intervention $\vec{X} \leftarrow \vec{x}$ and abstract this to the high-level model, you can follow two different routes to end up at the same place.

First, you can follow the route where the low-level intervention $\vec{X} \leftarrow \vec{x}$ changes the probability distribution on the low-level variables, as denoted by $Pr_L^{\vec{X}\leftarrow\vec{x}}$. This distribution on the low-level variables can be abstracted to a high-level distribution by applying the $\tau$-function, denoted by $\tau(Pr_L^{\vec{X}\leftarrow\vec{x}})$. In this route, you first apply the intervention on the low-level model, and then you abstract the distribution on the low-level variables with the $\tau$-function.

On the other hand, the second route abstracts the low-level intervention $\vec{X} \leftarrow \vec{x}$ to a high-level intervention by applying the $\tau$-function, as denoted by $\tau(\vec{X} \leftarrow \vec{x})$. This intervention changes the probability distribution on the high-level variables, as denoted by $Pr_H^{\tau(\vec{X}\leftarrow\vec{x})}$. In this route, you first abstract the low-level intervention to a high-level intervention, and then you apply the intervention to the high-level variables.

The two routes are shown visually in Figure 6.3.

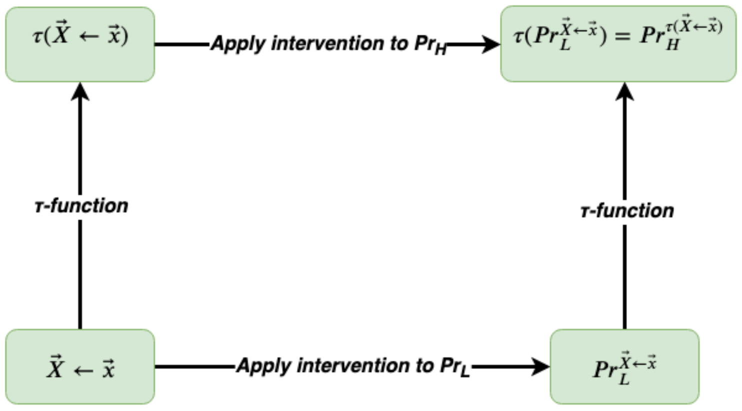

$$\begin{tikzcd}
\tau(\vec{X} \leftarrow \vec{x}) \arrow[r, "\text{Apply intervention to } Pr_H"] & \tau(Pr_L^{\vec{X}\leftarrow\vec{x}}) = Pr_H^{\tau(\vec{X}\leftarrow\vec{x})} \\
\vec{X} \leftarrow \vec{x} \arrow[u, "\tau\text{-function}"] \arrow[r, "\text{Apply intervention to } Pr_L"] & Pr_L^{\vec{X}\leftarrow\vec{x}} \arrow[u, "\tau\text{-function}"]
\end{tikzcd}$$

Figure 6.1: The two routes to $\tau(Pr_L^{\vec{X}\leftarrow\vec{x}}) = Pr_H^{\tau(\vec{X}\leftarrow\vec{x})}$.

Now, we can properly define *Constructive $\tau$-Abstraction*:

**Definition 6.2.2.** ***Constructive $\tau$-Abstraction*** High-level causal model $M_H$ is a constructive $\tau$-abstraction of low-level causal model $M_L$ if $\tau$ is constructive, $|R_L(U_L)| \geq |R_H(U_H)|$ and for all allowed interventions $\vec{X} \leftarrow \vec{x}$ on $M_L$, we have that $\tau(Pr_L^{\vec{X}\leftarrow\vec{x}}) = Pr_H^{\tau(\vec{X}\leftarrow\vec{x})}$. ■

In a constructive $\tau$-abstraction, we require that $|R_L(U_L)| \geq |R_H(U_H)|$ holds. This means that $U_L$, which denotes the possible contexts of the low-level model, has to be larger than or equal to $U_H$, the context of the high-level model. This is also quite intuitive: as the low-level model is more detailed and consists of more variables than the high-level model, it follows that its context will also be larger than or equal to the context of the high-level model.

### 6.2.1 Constructive $\tau$-Abstraction Example

Let's take a look at a somewhat concrete example. In the example, we want to model a causal phenomenon in two ways: as a low-level causal model consisting of a detailed view of the causal phenomenon, and as a more abstract model of the causal phenomenon. The former is the micro-level model, and the latter is the macro-level model. The micro-level model $M_L$ is depicted in Figure 6.2.

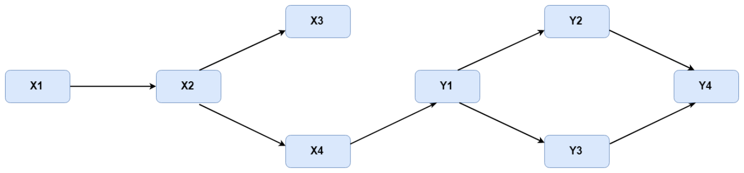


Figure 6.2: The micro-level causal model $M_L$.

$M_L$ consists of 8 variables, $(X_1, X_2, X_3, X_4, Y_1, Y_2, Y_3, Y_4)$. Figure 6.2 shows which variables are a cause of other variables. So, variable $X_1$ causes variable $X_2$, variable $X_2$ causes variables $X_3$ and $X_4$, etc. We can abstract this model by dividing the 8 variables into two groups, namely group $X$ and $Y$. This is macro-model $M_H$, depicted in Figure 6.3.

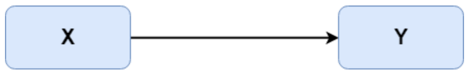


Figure 6.3: The macro-level causal model $M_H$.

We want to give the constructive $\tau$-abstraction between the micro- and macro-level model. This means that we have to define partition $P$ of the low-level variables:

$$P = \{\vec{Z}_1, \vec{Z}_2, \vec{Z}_3\} = \{(X_1, X_2, X_3, X_4), (Y_1, Y_2, Y_3, Y_4), \emptyset\}$$

$\vec{Z}_{n+1}$ is empty, as all low-level variables are abstracted to a high-level variable. In the rest of this thesis, we will not specifically define $\vec{Z}_{n+1}$ when $\vec{Z}_{n+1}$ is empty. Partition $P$ is abstracted with $\tau$ to the high-level model $M_L$, as follows:

$$\tau(\vec{Z}_1, \vec{Z}_2) = (X, Y),$$

$$\text{where } \tau_1 = R(\vec{Z}_1) \to R(Y_1) \text{ and } \tau_2 = R(\vec{Z}_2) \to R(Y_2).$$

This means that the low-level variables $X_1$, $X_2$, $X_3$ and $X_4$ are mapped to $\vec{Z}_1$ and are abstracted to variable $X$, and that variables $Y_1$, $Y_2$, $Y_3$ and $Y_4$ are mapped to $\vec{Z}_2$ and are abstracted to variable $Y$ in the high-level model, and it has to be the case that $\tau_1$ and $\tau_2$ are surjective.

As was just discussed, the allowed interventions on the low-level are interventions on the group of variables in $\vec{Z}_i$ of $P$. The allowed interventions on the low-level model $I_L$ are as follows:

$$I_L = \{\vec{Z}_1 \leftarrow \vec{z}_1, \vec{Z}_2 \leftarrow \vec{z}_2 : \vec{z}_1 \in R(\vec{Z}_1), \vec{z}_2 \in R(\vec{Z}_2)\}$$

On the high-level, all interventions are allowed.

# 7 | Markov Abstraction Equivalence Class

In this chapter, I will combine the insights of Markov Equivalence Classes and constructive $\tau$-abstraction on a low-level causal model. The idea of the theory in this chapter is how you can use the knowledge that there exists a constructive $\tau$-abstraction to determine how a low-level causal model should look.

It is important to note that in this chapter I will only focus on the causal structure of causal models, and not on the accompanying equations. The causal structure consists of the variables and the edges of a causal model, including the directions of the edges. I only focus on the causal structure because the theory I present in this chapter serves as an expansion of the PC algorithm, and the PC algorithm also solely focusses on the causal structure to be found, as we saw in Chapter 5. The reason for this is that the search for equations is a regression problem, which is not what the PC algorithm and this thesis are concerned with.

## 7.1 $\tau$-Compatibility and the Ontological Faithfulness Assumption

As we saw in the previous chapter, a constructive $\tau$-abstraction allows us to define a macro-level causal model as the abstraction of its micro-level causal model. In a constructive $\tau$-abstraction, we define a partition on the low-level model, which represents the cluster of variables in the low-level model that form the variables of the high-level model. In other words, a cluster of variables in the low-level consisting of variables $X_1$, $X_2$, $X_3$ and $X_4$ is abstracted to the high-level model as one variable $X$.

When the high-level model consists of two or more variables, meaning the low-level model consists of two or more clusters, the edges between the clusters on the low-level model face an obvious condition: there can not exist a cycle on the low-level clusters, because this would result in a cycle on the high-level. When this is the case, we say that the low-level causal model is *compatible with* $\tau$ or $\tau$*-compatible*:

**Definition 7.1.1.** $\tau$***-Compatibility*** Given a causal model $M_L$ with a constructive $\tau$ with partition $P = \{\vec{Z}_1, ..., \vec{Z}_{n+1}\}$ of $V_L$, we say that $M_L$ is $\tau$-compatible, or compatible with $\tau$, if there does not exist a cycle of the form $\vec{Z}_i \to \vec{Z}_j, ..., \vec{Z}_k \to \vec{Z}_i$, where $\vec{Z}_i \to \vec{Z}_j$ means that there exists an edge from $X_a \in \vec{Z}_i$ to $X_b \in \vec{Z}_j$ for some $a$ and $b$. ■

If we only look at the structure of a causal model, we can say that if a causal model $M_L$ is compatible with $\tau$, then we can construct a causal model $M_H$ which is a constructive $\tau$-abstraction of $M_L$. So, as far as the structure goes, if a causal model is $\tau$-compatible, then a constructive $\tau$-abstraction is possible. This is the case, because if we see in $M_L$ that a variable $X_4 \in \vec{Z}_2$ is causally influenced by a variable $X_2 \in \vec{Z}_1$ through an edge from $X_2$ to $X_4$, then we abstract this influence to $M_H$ by directing the edge between high-level variables $Y_1$ and $Y_2$ likewise. This is shown visually in Figure 7.1.

Intuitively, one might think that the implication also runs in the other direction: if a constructive $\tau$-abstraction is possible, then $M_L$ is $\tau$-compatible, because otherwise there would be a cycle in the high-level model. However, this does not always hold: there are cases when a constructive $\tau$-abstraction is possible and $M_L$ is not $\tau$-compatible, i.e. there a exists a low-level cycle of the form $\vec{Z}_i \to \vec{Z}_j, ..., \vec{Z}_k \to \vec{Z}_i$ which does not result in a cycle in the high-level causal model. To show when this is the case, we have to take the causal equations into account. Take a look at the following example.

Take low-level model $M_L$ with six variables, all ranging over $\mathbb{N}$: $X_1$, $X_2$, $Y_1$, $Y_2$, $A$, $B$. Two equations of $M_L$ are important:

$$Y_1 = X_1 + A$$
$$X_2 = Y_2 \mod 2 + 2 \cdot B$$

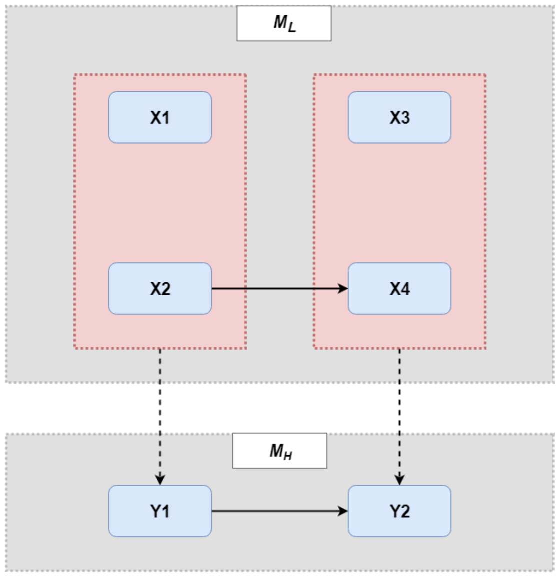


Figure 7.1: The edge from $X_2 \in \vec{Z}_1$ to $X_4 \in \vec{Z}_2$ in $M_L$ is abstracted to $M_H$ as an edge from $Y_1$ to $Y_2$.

This means that there is an edge $X_1 \to Y_1$ and an edge $Y_2 \to X_2$. The variables $A$ and $B$ serve to create a collider at $Y_1$ and $X_2$. $M_L$ is depicted in Figure 7.2.

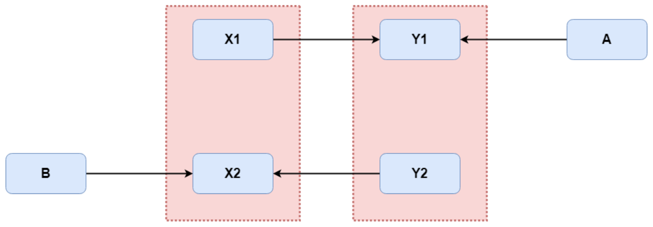


Figure 7.2: Causal model $M_L$, with the clusters $\vec{Z}_1$ and $\vec{Z}_2$ defined.

$M_H$ is the constructive $\tau$-abstraction of $M_L$. The variables $X_1$ and $X_2$ are grouped together in $\vec{Z}_1$, which is mapped to the high-level variable $X$ through the $\tau$-mapping $\tau_1 : X = X_1 + X_2/2$, where the division is a Euclidean division (i.e. $7/2 = 3$). The variables $Y_1$ and $Y_2$ are grouped together in $\vec{Z}_2$, which is mapped to the high-level variable $Y$ through the $\tau$-mapping $\tau_2 : Y = Y_1 + Y_2$. The clusters $\vec{Z}_1$ and $\vec{Z}_2$ can also be seen in Figure 7.2. We see that there exists a cycle on the variables in $M_L$ that are grouped together, namely a cycle between $\vec{Z}_1$ and $\vec{Z}_2$. However, in this case this does not pose a problem for the high-level causal model $M_H$: the influence of $Y_2$ on $X_2$ is not relevant for $M_H$, because for each $B$ and $Y_2$ it holds that

$$(Y_2 \mod 2 + 2 \cdot B)/2 = B$$

Together with the $\tau$-mapping $\tau_1 : X = X_1 + X_2/2$, we can conclude that $X$ does not depend on $Y_2$, which means that there should not be an edge from $Y$ to $X$ on the high-level. Instead, $M_H$ has the edge $X \to Y$.

This counterexample shows that a constructive $\tau$-abstraction is still possible, even though a cycle exists between $\vec{Z}_1$ and $\vec{Z}_2$ on the low-level. Because the causal equations show that the influence of one of the directions of the cycle is irrelevant for the high-level variables, we can formulate $M_H$ without cycles.

However, I argue that it is reasonable to adopt the view that when a constructive $\tau$-abstraction is possible, there can not exist a cycle of the form $\vec{Z}_i \to \vec{Z}_j, ..., \vec{Z}_k \to \vec{Z}_i$, i.e. $M_L$ must be $\tau$-compatible. If we accept two simple assumptions, this automatically follows:

1. The variables that are grouped together to a $\vec{Z}_i$ in $P$ in the low-level model $M_L$ describe the characteristics of the same causal event, as they would not be grouped together if this was not the case. Following this, we consider the variables that are grouped together to a $\vec{Z}_i$ in $P$ to be one causal event.

2. An arrow between two variables means that there is some form of causation from one variable to the other variable. If there is a cycle on a set of variables, this means that ultimately a variable causes itself. In other words, we speak of self-causation (*causa sui*). I will not go into the philosophical discussion of self-causation, but, as is generally accepted, we consider self-causation to be impossible.

So, because self-causation of a causal event is impossible, and because the variables in $M_L$ that are grouped together to a $\vec{Z}_i$ in $P$ are considered one causal event, self-causation of a $\vec{Z}_i$ is impossible, which means that there can not exist a cycle of the form $\vec{Z}_i \to \vec{Z}_j, ..., \vec{Z}_k \to \vec{Z}_i$. We summarize these assumptions as one assumption, the *Ontological Faithfulness Assumption*:

***Ontological Faithfulness Assumption*** Given a causal model $M_L$ with a constructive $\tau$ with partition $P = \{\vec{Z}_1, ..., \vec{Z}_{n+1}\}$ of $V_L$, if there exists a causal model $M_H$ which is a constructive $\tau$-abstraction of $M_L$, then $M_L$ is compatible with $\tau$. ■

We adopt the Ontological Faithfulness Assumption in the rest of this thesis. This means that if a causal model is compatible with $\tau$, then a constructive $\tau$-abstraction is possible, and if a constructive $\tau$-abstraction is possible, then the causal model is compatible with $\tau$.

### 7.1.1 Ontological Faithfulness Assumption Example

Let's take a look at an example. Let $M_L$ be a causal model with variables $X_1$, $X_2$, $X_3$, $Y_1$, $Y_2$, $Y_3$, $W_1$ and $W_2$, as depicted in Figure 7.3. The partition $P$ of $M_L$ is as follows:

$$P = \{\vec{Z}_1, \vec{Z}_2, \vec{Z}_3\} = \{(X_1, X_2, X_3), (Y_1, Y_2, Y_3), (W_1, W_2)\}$$

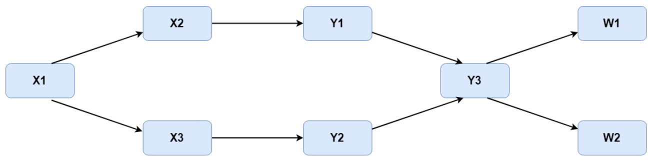


Figure 7.3: Low-level causal model $M_L$.

Let $M_H$ be the high-level abstraction of $M_L$, with the following $\tau$-mapping:

$$\tau(\vec{Z}_1, \vec{Z}_2, \vec{Z}_3) = (X, Y, W)$$

In other words, low-level variables $X_1$, $X_2$ and $X_3$ are clustered in the high-level variable $X$, low-level variables $Y_1$, $Y_2$ and $Y_3$ are clustered in the high-level variable $Y$, and low-level variables $W_1$ and $W_2$ are clustered in the high-level variable $W$. High-level model $M_H$ is depicted in Figure 7.4.

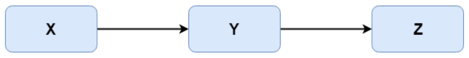


Figure 7.4: High-level causal model $M_H$.

We will check if the Ontological Faithfulness Assumption holds for low-level causal model $M_L$, i.e. if $M_L$ is compatible with $\tau$. We consider this in two steps. First, we will only consider the edges between each pair of clusters ($\vec{Z}_i$, $\vec{Z}_j$). The edges between a pair of clusters must either be undirected, or be directed equally, because otherwise a cycle would exist on $\vec{Z}_i$ and $\vec{Z}_j$. Second, we will check whether a cycle exists considering all clusters $\vec{Z}_1, ..., \vec{Z}_n$. If there is a directed edge between a pair ($\vec{Z}_i$, $\vec{Z}_j$), we consider this to be the direction of all edges between $\vec{Z}_i$ and $\vec{Z}_j$. When we do this for all pairs of clusters, we can check whether a cycle is created on the entire model. The steps will now be explained in detail.

First, we look at each pair of clusters. The edges between a pair must either be undirected, or directed equally. We will first consider the pair ($\vec{Z}_1$, $\vec{Z}_2$). The edges between them are $(X_2, Y_1)$ and $(X_3, Y_2)$. In Figure 7.3, we see that $(X_2, Y_1)$ is directed as $X_2 \rightarrow Y_1$, and $(X_3, Y_2)$ is directed as $X_3 \rightarrow Y_2$. This means that in both cases, the edge is directed as $\vec{Z}_1 \rightarrow \vec{Z}_2$. The edges between $\vec{Z}_1$ and $\vec{Z}_2$ are thus directed equally, and meet the Ontological Faithfulness Assumption.

The second pair we consider is the pair ($\vec{Z}_2$, $\vec{Z}_3$). The edges between them are $(Y_3, W_1)$ and $(Y_3, W_2)$. In Figure 7.3, we see that $(Y_3, W_1)$ is directed as $Y_3 \rightarrow W_1$, and $(Y_3, W_2)$ is directed as $Y_3 \rightarrow W_2$. This means that in both cases, the edge is directed as $\vec{Z}_2 \rightarrow \vec{Z}_3$. The edges between $\vec{Z}_2$ and $\vec{Z}_3$ are thus also directed equally, and meet the Ontological Faithfulness Assumption.

The last pair of clusters to consider is the pair ($\vec{Z}_1$, $\vec{Z}_3$). However, there are no edges between $\vec{Z}_1$ and $\vec{Z}_3$, so there is nothing to check.

The next step is to check whether a cycle exists on all clusters $\vec{Z}_1, \vec{Z}_2, \vec{Z}_3$. As there exists a directed edge between $\vec{Z}_1$ and $\vec{Z}_2$, we consider this to be the direction of the edge between the clusters $\vec{Z}_1$ and $\vec{Z}_2$: $\vec{Z}_1 \rightarrow \vec{Z}_2$. There also exists a directed edge between $\vec{Z}_2$ and $\vec{Z}_3$, so we consider this to be the direction of the edge between the clusters $\vec{Z}_2$ and $\vec{Z}_3$: $\vec{Z}_2 \rightarrow \vec{Z}_3$. There are no edges between $\vec{Z}_1$ and $\vec{Z}_3$. The entire model thus looks as follows:

$\vec{Z}_1 \rightarrow \vec{Z}_2 \rightarrow \vec{Z}_3$.

We can easily see that this is not a cycle. Therefore, the edges between all cycles also meet the Ontological Faithfulness Assumption.

As we have proven that there does not exist a cycle between any pair of clusters in $M_L$, and that there does not exist a cycle on all clusters $\vec{Z}_1, ..., \vec{Z}_3$ in $M_L$, we know with certainty that $M_L$ is compatible with $\tau$, i.e. the Ontological Faithfulness Assumption holds for $M_L$.

To give an example of when the Ontological Faithfulness Assumption does not hold, we apply a small change to $M_L$, as depicted in Figure 7.5. Here, the edges between $\vec{Z}_1$ and $\vec{Z}_2$ are directed as $X_2 \to Y_1$ and $X_3 \leftarrow Y_2$. The edges between $\vec{Z}_1$ and $\vec{Z}_2$ are thus not directed equally, which results in a cycle between $\vec{Z}_1$ and $\vec{Z}_2$. This means that $M_L$ is not compatible with $\tau$ and that the Ontological Faithfulness Assumption does not hold. As a consequence, a constructive $\tau$-abstraction is not possible on this model.

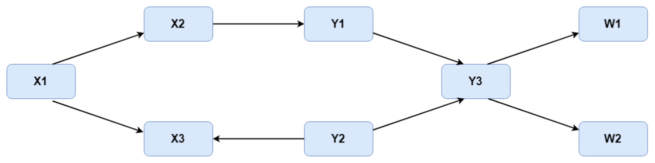


Figure 7.5: A small change of $M_L$ for which the Ontological Faithfulness Assumption does not hold, which makes a constructive $\tau$-abstraction not possible.

## 7.2 Markov Abstraction Equivalence Class

When we have acquired data for a certain causal model, we can find its Markov Equivalence Class by running the PC algorithm. When we also know that there exists a constructive $\tau$-abstraction of this model, and we know which variables are grouped together in the partition $P$ to be abstracted, we can use this information to restrict the number of possible models taken from the Markov Equivalence Class by tossing out the models of which no constructive $\tau$-abstraction exists. For this, we first have to define Markov $\tau$-Abstraction Equivalence between two models:

**Definition 7.2.1.** ***Markov $\tau$-Abstraction Equivalence*** Two DAGs are Markov $\tau$-Abstraction Equivalent if they are Markov Equivalent and compatible with $\tau$. ■

As we have established in Section 7.1 with the Ontological Faithfulness Assumption, if a causal model is compatible with $\tau$, then there exists a constructive $\tau$-abstraction. So, the consequence of both models that are Markov $\tau$-Abstraction Equivalent being compatible with $\tau$ is that for both models there exists a constructive $\tau$-abstraction.

With the definition of Markov $\tau$-Abstraction Equivalence, you can easily construct the *class* of Markov $\tau$-Abstraction equivalent models:

**Definition 7.2.2.** ***Markov $\tau$-Abstraction Equivalence Class*** The Markov $\tau$-Abstraction Equivalence Class $C_{(\tau,L)}$ of a causal model $M_L$ with a constructive $\tau$-abstraction is the class of all causal models that are Markov $\tau$-Abstraction Equivalent with $M_L$. ■

If the function $\tau$ is clear from the context, we refer to the Markov $\tau$-Abstraction Equivalence Class as the Markov Abstraction Equivalence Class.

This definition thus requires that all causal models in the Markov Abstraction Equivalence Class $C_{(\tau,L)}$ are compatible with $\tau$, which means, following the Ontological Faithfulness Assumption, that on all models a constructive $\tau$-abstraction exists.

The Markov Abstraction Equivalence Class is pictured in the same way as the Markov Equivalence Class: an undirected edge means that the edge can be directed both ways, depending on the specific model, and a directed edge means that for all causal models in the Markov Abstraction Equivalence Class, the edge is directed exactly that way. Because all models in the Markov Abstraction Equivalence are compatible with $\tau$, we know that all edges between a pair $\vec{Z}_i$ and $\vec{Z}_j$ in $P$ must be directed equally or be undirected, as otherwise a cycle would exist between the clusters. Consequently, we know that in the structure of the Markov Abstraction Equivalence Class all edges between clusters must either be directed equally or be undirected.

Because $C_{(\tau,L)}$ is a subset of the Markov Equivalence Class $C_L$, all models in $C_{(\tau,L)}$ must be Markov equivalent to all models in $C_L$. This means that $C_{(\tau,L)}$ must have the same colliders as $C_L$.

The characteristics of the Markov Abstraction Equivalence Class can determine the direction of edges in $C_{(\tau,L)}$ that were not directed in $C_L$. This means that $C_{(\tau,L)}$ contains fewer causal models in comparison to $C_L$.

Note that a Markov Abstraction Equivalence Class also asks its user to interpret the undirected edges in the right way, just like the Markov Equivalence Class does. That is, in a specific model the undirected edges can not be directed in such a way that a new collider or a cycle is created (conditions of the Markov Equivalence Class), and in such a way that a cycle exists on $\vec{Z}_i \rightarrow \vec{Z}_j, ..., \vec{Z}_k \rightarrow \vec{Z}_i$, which makes the specific model not $\tau$-compatible (condition of the Markov Abstraction Equivalence Class).

### 7.2.1 Markov Abstraction Equivalence Class Example

The following example will illustrate the construction of the Markov Abstraction Equivalence Class. Let $M_L$ be a causal model with variables $X_1, X_2, X_3, X_4, X_5$, $W_1, W_2, W_3, W_4, W_5$, as depicted in Figure 7.6.

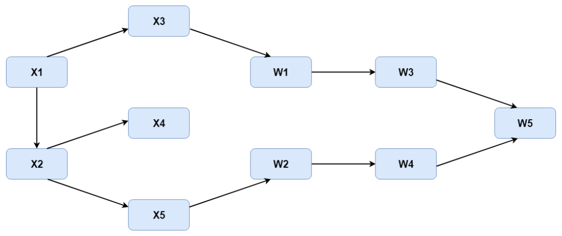


Figure 7.6: Causal model $M_L$.

The Markov Equivalence Class $C_L$ of $M_L$ is the undirected graph of $M_L$, with the only directed edges being the two incoming edges on $W_5$. The Markov Equivalence Class of $M_L$ is depicted in Figure 7.7.

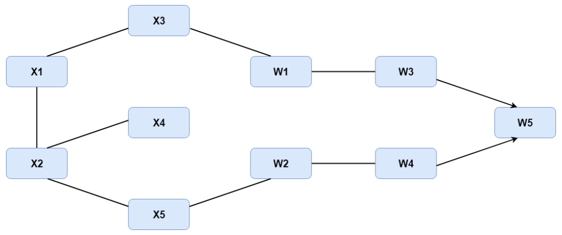


Figure 7.7: The Markov Equivalence Class $C_L$ of $M_L$.

The partition $P$ on $V_L$ is as follows:

$$P = \{\vec{Z}_1, \vec{Z}_2\} = \{(X_1, X_2, X_3, X_4, X_5), (W_1, W_2, W_3, W_4, W_5)\}$$

Let $M_H$ be the constructive $\tau$-abstraction of $M_L$, with the following $\tau$-mapping:

$$\tau(\vec{Z}_1, \vec{Z}_2) = (X, W)$$

Given that we know that $M_H$ is an abstraction of $M_L$, we can construct the Markov Abstraction Equivalence Class $C_{(\tau,L)}$ of $M_L$. We know that in $C_{(\tau,L)}$, all models have to be Markov $\tau$-Abstraction Equivalent, i.e. all models must be compatible with $\tau$. When we check for this, we can determine the direction of certain edges that were undirected in $C_L$.

As we can remember from the Ontological Faithfulness Assumption Example (Section 7.1.1), checking whether $M_L$ is compatible with $\tau$ happens in two steps: (1) checking whether the edges between each pair of clusters ($\vec{Z}_i$, $\vec{Z}_j$) are either undirected, or directed equally; (2) checking whether a cycle exists on all clusters $\vec{Z}_1, ..., \vec{Z}_n$.

As there is only one pair of clusters ($\vec{Z}_1$, $\vec{Z}_2$) in $M_L$, we only have to check this pair in the first step. The edges between them are $(X_3, W_1)$ and $(X_5, W_2)$. They are undirected in $C_L$. However, because $M_L$ must be compatible with $\tau$, we know that these edges must be directed equally. We can actually determine what way the edges must be directed, when we consider both directions and check whether one of the directions causes a new collider or cycle in $C_{(\tau,L)}$. A new collider in $C_{(\tau,L)}$ is not allowed, as the definition of Markov $\tau$-Abstraction Equivalence states that all models in $C_{(\tau,L)}$ must be Markov Equivalent. If it is the case that one direction causes a new collider or cycle, then we know that that direction is not allowed, and that the edges should be directed the other way.

First, we try directing both edges with a right arrow, as $X_3 \rightarrow W_1$ and $X_5 \rightarrow W_2$. We notice that the edges $(W_1, W_3)$ and $(W_2, W_4)$ must be directed with a right arrow as well, as $W_1 \rightarrow W_3$ and $W_2 \rightarrow W_4$, in order to prevent the creation of a new collider. This does not cause any new colliders, which means that directing the edges between $\vec{Z}_1$ and $\vec{Z}_2$ with a right arrow is allowed.

Second, we try directing both edges with a left arrow, as $X_3 \leftarrow W_1$ and $X_5 \leftarrow W_2$. We notice that the edges $(X_1, X_3)$ and $(X_2, X_5)$ must be directed with a left arrow as well, as $X_1 \leftarrow X_3$ and $X_2 \leftarrow X_5$, in order to prevent the creation of a new collider. However, upon doing so, we also notice that the edge $(X_1, X_2)$ can not be directed as not to create a new collider. This means that directing the edges between $\vec{Z}_1$ and $\vec{Z}_2$ with a left arrow causes a new collider in $C_{(\tau,L)}$, which makes it not Markov equivalent with the causal models in $C_L$ anymore. Therefore, directing the edges between $\vec{Z}_1$ and $\vec{Z}_2$ with a left arrow is not allowed.

Because directing the edges between $\vec{Z}_1$ and $\vec{Z}_2$ with a left arrow is not allowed, we conclude that the edges between $\vec{Z}_1$ and $\vec{Z}_2$ must be directed with a right arrow. Following this, the edges $(W_1, W_3)$ and $(W_2, W_4)$ must be directed with a right arrow as well.

What rests is checking whether a cycle exists on all clusters $\vec{Z}_1, \vec{Z}_2$. As we have just established the direction of the edges between $\vec{Z}_1$ and $\vec{Z}_2$, and $\vec{Z}_1$ and $\vec{Z}_2$ are the only two clusters of $M_L$, the entire model looks as follows:

$\vec{Z}_1 \rightarrow \vec{Z}_2$

It is clear that this is not a cycle. Therefore, $M_L$ is compatible with $\tau$.

We have now established that the Markov Abstraction Equivalence Class $C_{(\tau,L)}$ of $M_L$ consists of the Markov Equivalence Class $C_L$ of $M_L$, with four extra edges directed: $X_3 \rightarrow W_1$, $X_5 \rightarrow W_2$, $W_1 \rightarrow W_3$ and $W_2 \rightarrow W_4$. $C_{(\tau,L)}$ is depicted in Figure 7.8.

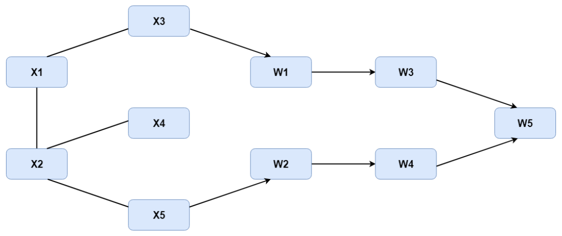


Figure 7.8: The Markov Abstraction Equivalence Class $C_{(\tau,L)}$ of $M_L$.

## 7.3 Justifying the Use of the Markov Abstraction Equivalence Class

The Markov Abstraction Equivalence Class is a class of low-level causal models that are Markov Equivalent and compatible with $\tau$, i.e. causal models on which a constructive $\tau$-abstraction exists. But why do we use information about a constructive $\tau$-abstraction, i.e. information that there exists a high-level causal model, on the low-level causal model? Why do we not focus our attention on the high-level causal model immediately?

When we have acquired data for a causal model, we can construct the Markov Equivalence Class of the causal model by using algorithms like the PC Algorithm. However, when we know that a constructive $\tau$-abstraction exists on this causal model, and we know $\tau$, we can also transform the data by $\tau$ and construct the high-level causal model right away by applying the PC algorithm on this data, without bothering about constructing the low-level causal model first.

We can do this for the low-level causal model $M_L$ and its high-level causal model $M_H$ of the example in Section 7.2.1. $M_L$ has 10 variables that are divided into two clusters $\vec{Z}_1$ and $\vec{Z}_2$ (see section 7.2.1 for more details). $M_H$ has two variables, $X$ and $W$, that are mapped to by $\vec{Z}_1$ and $\vec{Z}_2$. With this information, we can transform the data of $M_L$, by merging the data of $X_1, X_2, X_3, X_4$ and $X_5$ into data for the high-level variable $X$ and by merging the data of $W_1, W_2, W_3, W_4$ and $W_5$ into data for the high-level variable $W$. We have now transformed the low-level data into high-level data, on which we can apply the PC algorithm. When we do this, we get the following result: the Markov Equivalence Class of $M_H$, with an undirected edge between $X$ and $W$, as depicted in Figure 7.9. The PC algorithm is unable to find the direction of the edge between the two variables, as there is too little information.

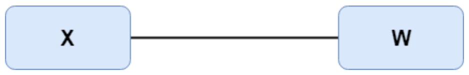


Figure 7.9: The Markov Equivalence Class of $M_H$, constructed by transforming the low-level data to high-level data and applying the PC algorithm.

To compare, we follow the other route, by first constructing the low-level causal model $M_L$ by applying the PC algorithm, constructing the Markov Abstraction Equivalence Class $C_{(\tau,L)}$ of $M_L$, and by performing a constructive $\tau$-abstraction on $C_{(\tau,L)}$. In section 7.2.1, we can see the Markov Equivalence Class of $M_L$ (Figure 7.7) and the Markov Abstraction Equivalence Class $C_{(\tau,L)}$ of $M_L$ (Figure 7.8). Notice that all edges between $\vec{Z}_1$ and $\vec{Z}_2$ are directed in $C_{(\tau,L)}$ with a right arrow. When performing a constructive $\tau$-abstraction on $C_{(\tau,L)}$, we can take this information about the direction of the edges between the clusters and apply it to $M_H$. This means that we know that the edge between $X$ and $W$ in $M_H$ must be directed as $X \rightarrow W$. The high-level causal model $M_H$ as a result of following this route is shown in Figure 7.10.

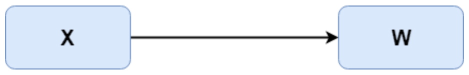


Figure 7.10: The Markov Equivalence Class of $M_H$, constructed by applying the constructive $\tau$-abstraction on the Markov Abstraction Equivalence Class of $M_L$.

The two routes just described are depicted in Figure 7.11. The first route I described is the 'red route': you transform the low-level data with $\tau$ to high-level data, on which you apply the PC algorithm. This gives the Markov Equivalence Class (MEC) of $M_H$. The second route I described is the 'blue route': you obtain the MEC of $M_L$ by applying the PC algorithm on the low-level data, and with the Ontological Faithfulness Assumption you construct the Markov Abstraction Equivalence Class (MAEC) of $M_L$. Abstracting this with $\tau$ gives the MEC of $M_H$. For ease, I will refer to these routes as the red route and the blue route.

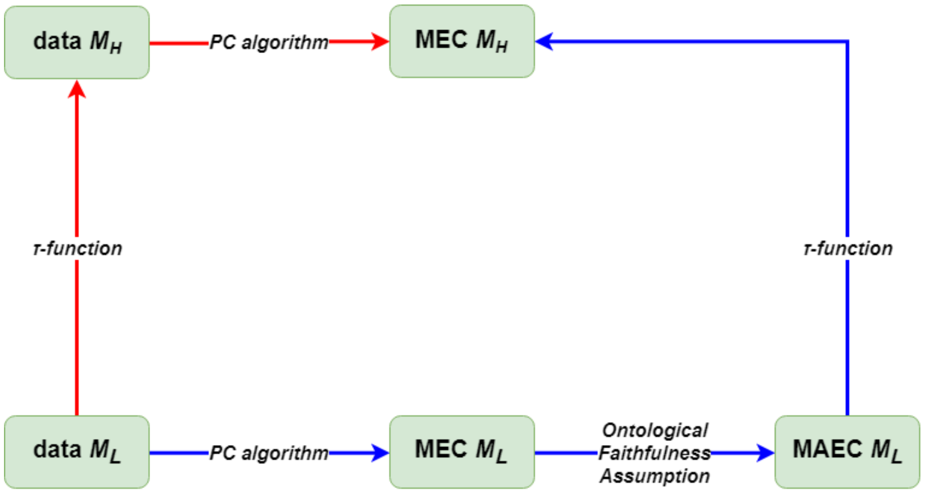


Figure 7.11: The two routes to get to the Markov Equivalence Class of high-level causal model $M_H$.

So what does this say? By following the blue route, we have obtained more information about the structure of $M_H$ in comparison with the red route. One would expect this, because the low-level causal model contains more variables than the high-level causal model. This means that the red route works with less information than the blue route. In addition, the blue route gains information with the construction of the Markov Abstraction Equivalence Class. The Markov Abstraction Equivalence Class thus not only adds information about the structure of the low-level Markov Abstraction Equivalence Class, but it also helps in finding the structure of the high-level Markov Equivalence Class.

In general, the Markov Abstraction Equivalence Class adds information in two ways:

1. It adds information about the structure of the high-level Markov Equivalence Class;
2. It adds information about the structure of the low-level Markov Abstraction Equivalence Class.

Together, this provides the justification of the focus on the low-level causal model and the construction of the Markov Abstraction Equivalence Class: it does not only add information about the structure of the low-level causal model, but it also adds information about the structure of the high-level causal model.

# 8 | The `pcabs` Algorithm

In this chapter, I will present the algorithm **'pcabs'**, which puts the theory of Markov Abstraction Equivalence Classes into practice. I will first discuss the algorithm conceptually, after which I will give the pseudocode of `pcabs`. The chapter ends by testing the performance of `pcabs`.

## 8.1 `pcabs` conceptually - The Four Scenarios

The `pcabs` algorithm puts the theory of Markov Abstraction Equivalence Classes into practice, by adding the information of a constructive $\tau$-abstraction to a Markov Equivalence Class, in order to determine the direction of certain edges in the Markov Equivalence Class. The algorithm knows the direction of certain edges by removing the causal models that:

1. are not compatible with $\tau$, i.e. the causal models that have a cycle on $\vec{Z}_i \to \vec{Z}_j, ..., \vec{Z}_k$;
2. are not Markov Equivalent with the models in the Markov Equivalence Class. These are the models for which directing all edges between a pair of clusters creates a new collider.

The output of the `pcabs` algorithm is exactly the Markov Abstraction Equivalence Class. The algorithm removes all models that are not compatible with $\tau$, which means that all models that are left are compatible with $\tau$. Consequently, on all models, a constructive $\tau$-abstraction is possible. The algorithm also ensures that all models that are left are Markov Equivalent. Together, this makes exactly the Markov Abstraction Equivalence Class.

The `pcabs` algorithm uses the code of the package **'pcalg'**, which is the PC Algorithm, and extends it. More precisely, the code of `pcalg` for constructing the pattern or Markov Equivalence Class from data is used, which is the PC-function and its helping functions. `pcabs` should be used after the pattern has been found by `pcalg`. This pattern is the Markov Equivalence Class, which we make more complete by using the given partition of the constructive $\tau$-abstraction. With the partition, `pcabs` will find Markov Abstraction Equivalence Class, by removing the causal models that are not compatible with $\tau$ and that are not Markov Equivalent with the models in the Markov Equivalence Class. You add the partition of a constructive $\tau$-abstraction in `pcabs` by defining the low-level clusters that will be abstracted to the high-level variables. You thus do not need to define the $\tau$-function, but only the partition, because we focus only on the causal structure. For example, on the basis of the Markov Equivalence Class in Figure 7.7, you can add the two clusters $(X_1, X_2, X_3, X_4, X_5)$ and $(W_1, W_2, W_3, W_4, W_5)$. For each pair of clusters, the code faces four scenarios:

Scenario 1 There are no edges between the pair of clusters. This means that no edges have to be directed.

Scenario 2 All edges between the pair of clusters are undirected. This is the case for the Markov Equivalence Class in Figure 7.7. Because we know that the edges between a pair of clusters must be directed equally, the algorithm tries both directions of all edges, and checks whether one of the directions leads to a new v-structure (collider). If so, this means that that direction of the edges is forbidden, as a new v-structure means that the models are not Markov Equivalent with the Markov Equivalence Class anymore. If one of the directions turns out to be forbidden and the other direction is possible, the other direction of the edges automatically is the right direction. The algorithm then checks if any other edges can be directed based on this, as to not produce any new v-structures or cycles. In the case that both directions of the edges are allowed, the abstraction offers no new information, and the edges remain undirected.

Scenario 3 There is one directed edge between pair of clusters. This means that the other edges between the pair of clusters need to be directed that same way, in order to prevent a cycle on the clusters. The algorithm finds out what way the already directed edge is directed, and directs the other edges equally. If this leads to new v-structures, the constructive $\tau$-abstraction turns out to be impossible, and the algorithm returns the original graph with an error message. If the direction of the edges is possible, the algorithm checks if any other edges can be directed based on this, again not producing any new v-structures or cycles.

Scenario 4 There are multiple directed edges between pair of clusters. The algorithm first has to check whether all directed edges are directed the same way. If this is not the case, the constructive $\tau$-abstraction turns out to be impossible, because the model is not compatible with $\tau$. In that case, the algorithm outputs an error message and the original graph. Otherwise, if all edges are pointed in the same direction, the algorithm has to check whether there are any undirected edges left. These have to be directed the same way as the other edges. The algorithm checks whether this leads to new v-structures, which makes the constructive $\tau$-abstraction impossible. If not, the algorithm checks if any other edges can be directed based on the abstraction, again not producing any new v-structures or cycles.

## 8.2 Pseudocode `pcabs`

I will now give the pseudocode of `pcabs`. As input, `pcabs` needs a pattern, as produced by the PC algorithm, and a specification of the partition of the low-level variables in a list. The algorithm consists of two parts:

1. The **leading part**, which looks at the existing edges between all pairs $\vec{Z}_i$ and $\vec{Z}_j$ in $P$, and decides which pairs are directed first. This matters, because directing the edges between a pair can cause other edges to be directed as well, including edges between other pairs in $P$. In this algorithm, the pairs with one or more directed edges are directed first, as the direction of their edges is determined already. Lastly, the pairs with only undirected edges are directed.
2. The **directing part**, which takes as input one pair of clusters $\vec{Z}_i$ and $\vec{Z}_j$ and their connecting edges from the leading part. This part of the algorithm then determines in what direction the edges should be directed, according to the four scenarios just described. The directing part gives back the updated pattern to the leading part.

First, the pseudocode of the **leading part** is as follows:

1. By using a nested loop, loop through each pair ($\vec{Z}_i$, $\vec{Z}_j$) in $P$:
   (a) Find the edges that connect $\vec{Z}_i$ and $\vec{Z}_j$. Store these edges in the variable `absList`.
   (b) If the length of `absList` is bigger than 0, we know that pair $\vec{Z}_i$ and $\vec{Z}_j$ are directly connected. This means that this pair needs to be sent to the directing part. If the length of `absList` is 0, scenario 1 comes into play: we know that pair $\vec{Z}_i$ and $\vec{Z}_j$ are not directly connected, and can thus be forgotten for the algorithm.
   (c) Check the directed edges between pair $\vec{Z}_i$ and $\vec{Z}_j$. Store the directed edges in the variable `dir_list`.
   (d) If `dir_list` is not empty, we know that there are directed edges between $\vec{Z}_i$ and $\vec{Z}_j$. Because we first direct the pairs with directed edges, this pair is sent directly to the directing part (the `add_abstraction` function), which sends back the updated pattern.
   (e) If `dir_list` is empty, we know that there are no directed edges between $\vec{Z}_i$ and $\vec{Z}_j$. This means that we want to direct the edges between this pair lastly. The `absList` of this pair is added to the list `undirected_list`.
2. When all the pairs have been checked, the pairs in `undirected_list` are sent to the directing part, which again sends back the updated pattern. If it turns out that one or multiple pairs can not be directed (i.e. both directions are possible), these pairs are checked again until only those pairs that can not be directed remain. These pairs then remain undirected.
3. Lastly, we need to check if a cycle exists on the clusters in the final pattern. We do so with the `check_highlevel_cycles` function, which generates the high-level model with the directed edges of the final pattern. If there exists a cycle on this, `check_highlevel_cycles` returns `TRUE`. The final pattern is then not compatible with $\tau$, so we return the original pattern with an error message. If `check_highlevel_cycles` returns `FALSE`, the final pattern is compatible with $\tau$ and the final pattern is returned.

The directing part receives the pattern and the edges between a pair $\vec{Z}_i$ and $\vec{Z}_j$ from the leading part. The variables of $\vec{Z}_i$ that are directly connected to $\vec{Z}_j$ are stored in the variable `abs_group1`. The variables of $\vec{Z}_j$ that are directed connected to $\vec{Z}_i$ are stored in the variable `abs_group2`. `abs_group1` and `abs_group2` follow the same order, so the first variable in `abs_group1` is connected to the first variable in `abs_group2`, etc.

The pseudocode of the **directing part** is as follows:

1. Check the directed edges between `abs_group1` and `abs_group2` on the pattern given by the PC algorithm. The directed edges are stored in the variable `dir_list`.

2. Check the number of v-structures, or colliders. This number is stored in the variable `v_structures`.

3. If the length of variable `dir_list` is 0, i.e. there are no directed edges between the two groups to be abstracted, scenario 2 comes into play:

    (a) Point all edges between `abs_group1` and `abs_group2` as *abs_ group1* $\rightarrow$ *abs_ group2*. Store this graph in variable `pdag1`.

    (b) Check whether `pdag1` causes a cycle or new v-structures not stored in the original pattern, by checking the number of v-structures and comparing this with the variable `v_structures`. We do so with the function `is_direction_possible()`. `is_direction_possible()` also checks whether other necessarily directed edges, according to the rules of the Markov Equivalence Class, causes a cycle or new v-structures. If `is_direction_possible()` returns `FALSE`, we know that pointing the edges as *abs_ group1* $\rightarrow$ *abs_ group2* is not possible. Store `dir1 = TRUE` if the direction *abs_ group1* $\rightarrow$ *abs_ group2* is possible, otherwise `dir1 = FALSE`.

    (c) Point all edges between `abs_group1` and `abs_group2` as *group1* $\leftarrow$ *group2*. Store this graph in variable `pdag2`.

    (d) Check `pdag2` with the function `is_direction_possible()`. If `is_direction_possible()` returns `FALSE`, we know that pointing the edges as *abs_ group1* $\leftarrow$ *abs_ group2* is not possible. Store `dir2 = TRUE` if the direction *group1* $\leftarrow$ *group2* is possible, otherwise `dir2 = FALSE`.

    (e) If `dir1` and `dir2` are both `TRUE`:

        i. We know that the two directions of the edges are both possible. This means that we have not gained any new information. The original pattern is returned.

(f) If `dir1 = TRUE` and `dir2 = FALSE`:

i. We know that directing the edges as *abs_group1* $\rightarrow$ *abs_group2* is the only possible direction. Check if this direction causes other edges to be directed as well, according to the rules of the Markov Equivalence Class. We do so with the function `apply_mec_rules()`. Direct these edges.

ii. Output the pattern with the edges directed as *abs_group1* $\rightarrow$ *abs_group2*, with the other necessary directed edges.

(g) If `dir1 = FALSE` and `dir2 = TRUE`:

i. We know that directing the edges as *abs_group1* $\leftarrow$ *abs_group2* is the only possible direction. Direct other edges according to the function `apply_mec_rules()`.

ii. Output the pattern with the edges directed as *abs_group1* $\leftarrow$ *abs_group2*, with the other necessary directed edges.

(h) If `dir1 = FALSE` and `dir2 = FALSE`:

i. We know that both directions of the edges are not possible. This means that an abstraction on the two groups is not possible. Return the original pattern with an error message.

4. If the length of variable `dir_list` is 2, there is exactly one directed edge between the two groups to be abstracted. This is the case because `dir_list` stores both the receiving as well as the outgoing variable of a directed edge. Scenario 3 comes into play:

(a) Check if the direction of this edge is *abs_group1* $\rightarrow$ *abs_group2*. If this is the case, store `group1_check = TRUE`. Else, store `group1_check = FALSE`.

(b) If `group1_check = FALSE`, change the `abs_group1` and `abs_group2` variables as follows: `q = abs_group1`, `abs_group1 = abs_group2`, `abs_group2 = q`. This is necessary for the next step.

(c) Point all edges between `abs_group1` and `abs_group2` as *abs_group1* $\rightarrow$ *abs_group2*. This is always right, as `abs_group1` and `abs_group2` might have been exchanged in the previous step.

(d) Check these directed edges with the function `is_direction_possible()`. If `is_direction_possible()` returns `FALSE`, we know that the given abstraction on the two groups is not possible. Return the original pattern with an error message. If `is_direction_possible()` returns `TRUE`, continue to the next step.

(e) With the edges directed as *abs_ group1* $\rightarrow$ *abs_ group2*, direct other edges according to the function `apply_mec_rules()`.

(f) Output this graph, with the edges directed as *abs_ group1* $\rightarrow$ *abs_ group2*, and with the other necessary edges.

5. If the length of variable `dir_list` is bigger than 2, i.e. there are multiple directed edges between the two groups to be abstracted, scenario 4 comes into play:

   (a) Put all variables (receiving and outgoing) connected to the directed edges in two groups: one group with all the variables that are in `abs_group1`, and one group with all the variables that are in `abs_group2`. Store the variables of `abs_group1` in the list `leftrighlist$left` and the variables of `abs_group2` in the list `leftrightlist$right`.

   (b) Check if the variables in `leftrightlist$left` have outgoing directed edges. If this is the case, store `result = "left"`. If `leftrightlist$left` has ingoing directed edges, store `result = "right"`. If it turns out that `leftrightlist$left` has both outgoing as well as ingoing directed edges, store `result = "error"`. This means that the given abstraction on the two groups is not possible. Return the original pattern with an error message.

   (c) If `result == "left"`:

      i. Go through all the variables in `abs_group1`. If a variable is not in `leftrightlist$left`, i.e. it does not have a directed edge, make the undirected edge(s) from this variable be an outgoing directed edge. Do so for all variables in `abs_group1`.

      ii. Check these directed edges with the function `is_direction_possible()`. If `is_direction_possible()` returns `FALSE`, we know that the given abstraction on the two groups is not possible. Return the original pattern with an error message. If `is_direction_possible()` returns `TRUE`, continue to the next step.

      iii. With the edges directed as *abs_ group1* $\rightarrow$ *abs_ group2*, direct other edges according to the function `apply_mec_rules()`.

iv. Output this graph.

(d) If `result = "right"`:

i. Go through all the variables in `abs_group1`. If a variable is not in `leftrightlist$right`, i.e. it does not have a directed edge, make the undirected edge(s) from this variable be an ingoing directed edge. Do so for all variables in `abs_group1`.

ii. Analogous to steps ii, iii and iv in (c).

6. End of the algorithm.

This algorithm, under the name `pcabs`, produces exactly the Markov Abstraction Equivalence Class. The algorithm removes all causal models from the Markov Equivalence Class that are not $\tau$-compatible and that are not Markov Equivalent with the models in the Markov Equivalence Class. By removing these models, certain directions of edges are determined that were undirected in the Markov Equivalence Class. This way, the Markov Abstraction Equivalence Class contains more information in comparison with the Markov Equivalence Class.

## 8.3 Testing the `pcabs` algorithm

In order to test the performance of the `pcabs` algorithm, we have fed the algorithm several existing and fictional causal models. The results are summarized in Table 8.1, which shows the difference in the amount of directed edges between the Markov Equivalence Class (MEC) and Markov Abstraction Equivalence Class (MAEC), as well as the difference in computation time. The computation time of `pcabs` is the sum of the CPU time of `pcalg` and `pcabs`, as `pcabs` needs the output of `pcalg`. I will explain each example shortly. All computations were run on a computer with an Intel Core i7 Processor running at 2.20 GHz using 4 GB of RAM, running on Windows 8.1.

**Example 1**

The first example is an existing example of Spirtes & Scheines (2000), to show which edges the PC algorithm is able to direct. The Markov Equivalence Class, as formed by the PC algorithm, has 2 of 5 edges directed, as depicted in Figure 8.1a. The constructive $\tau$-abstraction with $\vec{Z}_1 = (X_1, X_2)$ and $\vec{Z}_2 = (X_3, X_4, X_5)$

| Causal Model | Number of low-level Variables | Number of high-level Variables | Number of Edges | Directed Edges in MEC | Directed Edges in MAEC | CPU time `pcalg` | CPU time `pcabs` |
|---|---|---|---|---|---|---|---|
| Example 1 | 5 | 2 | 5 | 2 | 4 | 0.33 sec | 0.35 sec |
| Example 2 | 11 | 3 | 11 | 2 | 9 | 2.10 sec | 2.13 sec |
| Example 3 | 7 | 3 | 8 | 3 | 8 | 0.98 sec | 1.00 sec |
| Example 4 | 30 | 2 | 31 | 11 | 31 | 7.31 sec | 7.34 sec |
| Example 5 | 18 | 2 | 17 | 9 | 15 | 2.38 sec | 2.42 sec |

Table 8.1: The results of running the `pcabs` algorithm on five causal models.

gives the information to direct two more edges, as depicted in Figure 8.1b. The amount of directed edges is thus doubled with use of the `pcabs` algorithm.

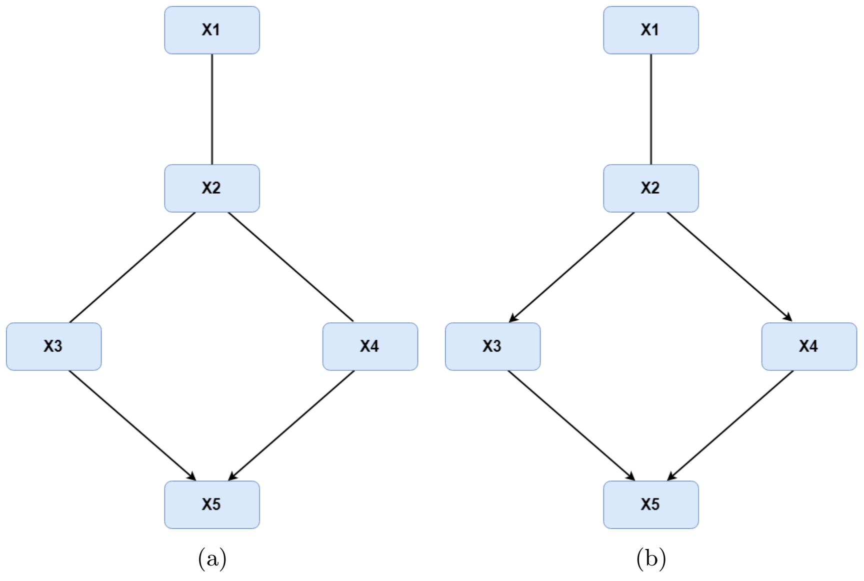


Figure 8.1: The Markov Equivalence Class of Example 1 with two directed edges (a) and the Markov Abstraction Equivalence Class of Example 1 with four directed edges (b).

**Example 2**

Example 2 is a fictional example, consisting of 11 variables. It shows the explosive amount of extra information that a constructive $\tau$-abstraction can give. The Markov Equivalence Class has 2 of 11 edges directed, as depicted in Figure 8.2a. The constructive $\tau$-abstraction with $\vec{Z}_1 = (X_1, X_2, X_3)$, $\vec{Z}_2 = (X_4, X_5, X_6)$ and

$\vec{Z}_3 = (X_7, X_8, X_9, X_{10}, X_{11})$ gives the information to direct 7 more edges, as depicted in Figure 8.2b. The amount of directed edges is thus enlarged from 18% to 82% by using the `pcabs` algorithm.

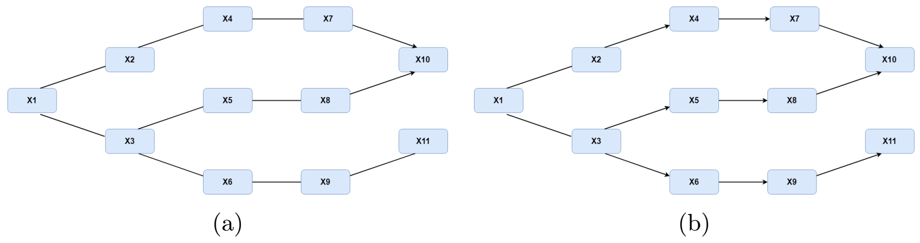


Figure 8.2: The Markov Equivalence Class of Example 2 with two directed edges (a) and the Markov Abstraction Equivalence Class of Example 2 with nine directed edges (b).

**Example 3**

The third example is an existing example, found in Shalizi (2019), as depicted in Figure 8.3. The Markov Equivalence Class of this causal model has 3 of 8 edges directed, as depicted in Figure 8.4a. The constructive $\tau$-abstraction with $\vec{Z}_1 = (X_1)$, $\vec{Z}_2 = (X_2, X_3, X_4, X_5, X_6)$ and $\vec{Z}_3 = (X_7)$ gives the information to direct all 8 edges of the causal model, as depicted in Figure 8.4b. This partition makes sense, given what we know of the causal model: the variables *Frequency of toothbrushing*, *Gum disease*, *Inflammatory immune respone*, *Frequency of exercise* and *Amount of fat and red meat in diet* are grouped together because they are all characteristics of health consciousness that influence the chance of a heart disease. So, the information added by the abstraction gives a 100% score of directed edges.

**Example 4**

Example 4 is a fictional example, to show how the `pcabs` algorithm works with a large causal model. The causal model consists of 31 edges, of which 11 are directed in the Markov Equivalence Class, as depicted in Figure 8.5. If we add just two abstraction groups to the Markov Equivalence Class, consisting of $\vec{Z}_1 = (X_1, X_2, X_3, X_4)$ and $\vec{Z}_2 = (X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_{16}, X_{17}, X_{18}, X_{19}, X_{20}, X_{21}, X_{22}, X_{23}, X_{24}, X_{25}, X_{26}, X_{27}, X_{28}, X_{29}, X_{30})$, we are able to direct all 31 edges of the causal model, as depicted in Figure 8.6. This shows how useful the information of only two abstraction groups can be.

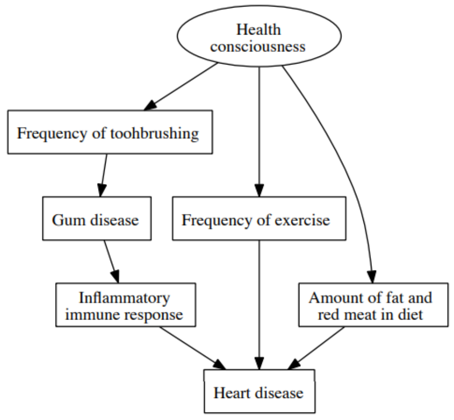

Health consciousness
Frequency of toohbrushing
Gum disease
Frequency of exercise
Inflammatory immune response
Amount of fat and red meat in diet
Heart disease

Figure 8.3: The Causal Model of Example 3 (figure taken from Shalizi (2019), p. 482.)

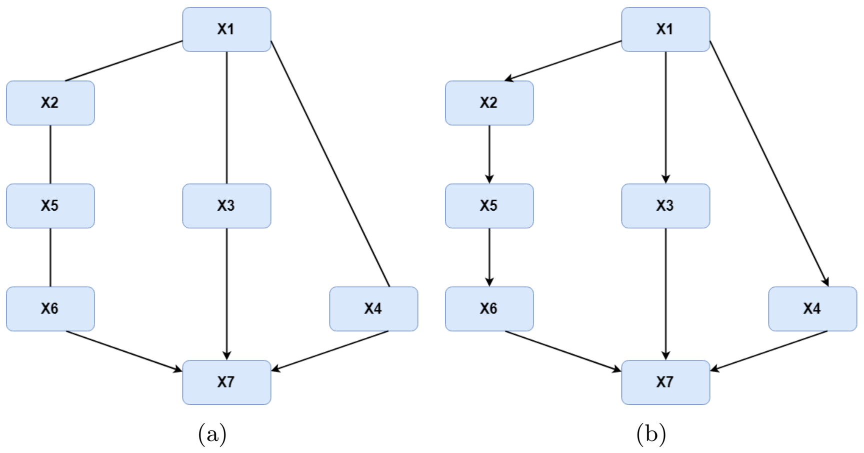


Figure 8.4: The Markov Equivalence Class of Example 3 with three directed edges (a) and the Markov Abstraction Equivalence Class of Example 3 with eight directed edges (b).

**Example 5**

The final example is a fictional example, to show which edges are not influenced by an abstraction. The causal model consists of 17 edges, of which 9 are directed in the Markov Equivalence Class, as depicted in Figure 8.7. If we add two abstraction groups to the Markov Equivalence Class, consisting of $\vec{Z}_1 = (X_1, X_2)$ and $\vec{Z}_2 = (X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14},$

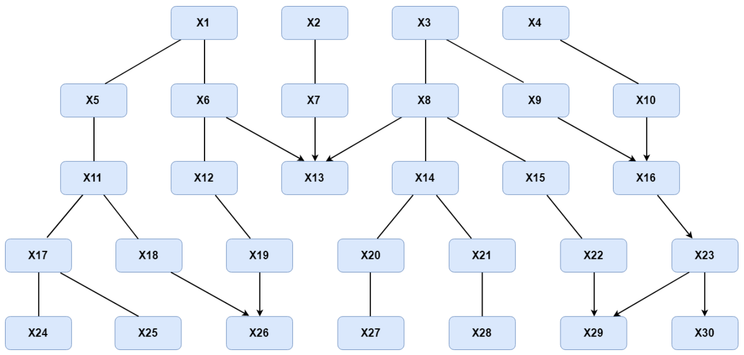


Figure 8.5: The Markov Equivalence Class of Example 4 with 11 directed edges.

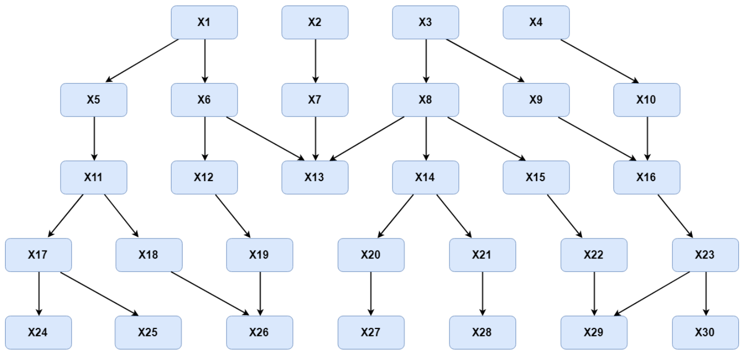


Figure 8.6: The Markov Abstraction Equivalence Class of Example 4 with 31 directed edges.

$X_{15}, X_{16}, X_{17}, X_{18}$), we are able to direct 6 more edges, as depicted in Figure 8.8. It is remarkable which edges are *not* directed in the Markov Abstraction Equivalence Class, namely the edges $(X_6, X_{10})$ and $(X_9, X_{14})$. The variables $X_6$ and $X_9$ are the only two variables that do not have $X_1$ or $X_2$ as their ascendants, and that are not directly connected to a v-structure, like variable $X_3$ is. The abstraction group $\vec{Z}_1$ thus influences all other undirected edges, i.e. all its descendants.

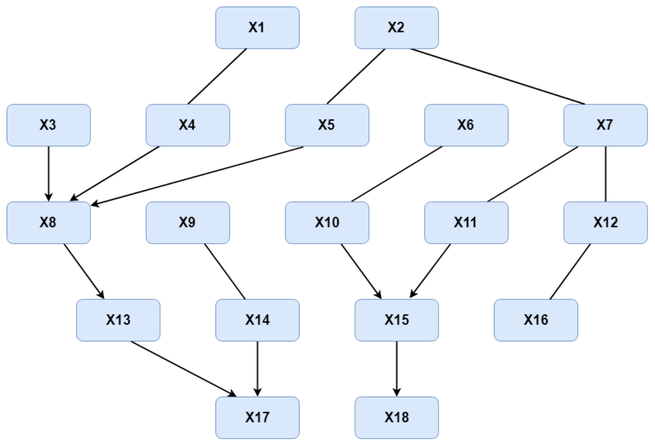


Figure 8.7: The Markov Equivalence Class of Example 5 with 9 directed edges.

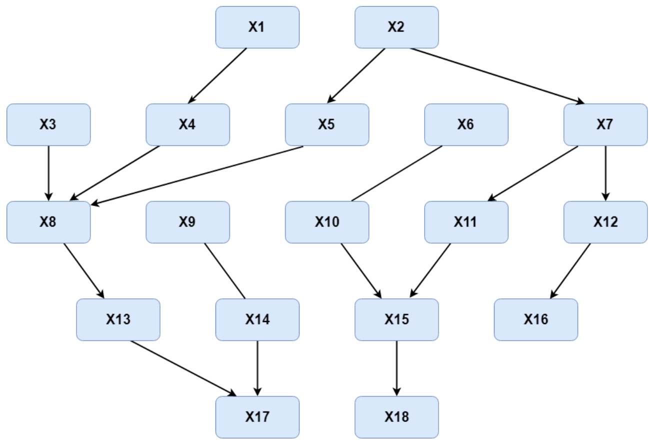


Figure 8.8: The Markov Abstraction Equivalence Class of Example 5 with 15 directed edges.

# 9 | Conclusion

In this thesis, I have introduced an account of Markov Abstraction Equivalence Classes. The Markov Abstraction Equivalence Class is a subset of the Markov Equivalence Class, which is constructed by using the fact that we have information about a constructive $\tau$-abstraction. The motivation to develop this account comes from studying the theories underlying it, and specifically by studying the constructive $\tau$-abstraction.

In Chapter 2, I have laid out the foundation of causal models, which consists of the theory of graphs, probability theory and Bayesian Networks. In this chapter I also introduced the terms and notations used in the rest of the thesis. In chapter 3, I introduced causal search algorithms, which are algorithms that structure causal models on the basis of data. Causal search algorithms are known for adopting several assumptions, of which the best known is the Causal Markov Assumption. I have also adopted this assumption in my thesis. In this chapter I also introduced constraint-based algorithms as a type of causal search algorithms, to which the PC algorithm belongs. In Chapter 4, I introduced Markov Equivalence and Markov Equivalence Classes. Two causal models are Markov Equivalent if they have the same d-separations, or the same independence statements. All causal models that are Markov Equivalent with each other are part of the same Markov Equivalence Class. The Markov Equivalence Class is an important tool in causal modelling, because it encompasses the causal models that behave the same and produce the same results, and because they are the result of causal search algorithms in the form of patterns. In Chapter 5, I discussed the details of the PC algorithm. The PC algorithm is one of the most frequently used causal search algorithms, and the main causal search algorithm I have chosen for this thesis. The pseudocode of the PC algorithm shows the similarities it has with Markov Equivalence Classes. These similarities also become clear in the output of the PC algorithm, which is the Markov Equivalence Class of the true graph. In Chapter 6, I dove into the more technical subject of abstracting causal models. Abstraction is a powerful tool, which enables you to create a smaller, less detailed, but more comprehensible causal model that is an abstraction of a bigger, more detailed causal model. In this thesis, I have only considered constructive $\tau$-abstraction, as this is the notion of abstraction that is most often used in practice. Constructive $\tau$-abstraction maps variables from a low-level causal model to a high-level causal model.

In Chapter 7, I described the Markov Abstraction Equivalence Class. First, I defined $\tau$-compatibility and argued in the Ontological Faithfulness Assumption that a causal model must be $\tau$-compatible for a constructive $\tau$-abstraction to be possible. This paved the way to introduce Markov $\tau$-Abstraction Equivalence and Markov Abstraction Equivalence Class. The Markov Abstraction Equivalence Class can be seen as a subset of the Markov Equivalence Class, with the causal models removed from the Markov Equivalence Class for which there does not exist a constructive $\tau$-abstraction. The Markov Abstraction Equivalence Class is put into practice by the algorithm `pcabs` as an addition to the PC algorithm, explained in Chapter 8. `pcabs` finds the Markov Abstraction Equivalence Class of a Markov Equivalence Class. The performance of `pcabs` shows how much information is added by the algorithm and how more detailed causal models are generated. The results of testing `pcabs` also show a fast CPU time.

A disadvantage of Markov Abstraction Equivalence Classes and the `pcabs` algorithm is that it can only be used for the specific case in which it is known that an abstraction on a causal model exists, and it is known how the variables in the low-level causal model are grouped together. Because the theory on abstracting causal models is still recent, there are not many causal models to which a constructive $\tau$-abstraction is applied. This also explains why it was hard to find real-life examples to use `pcabs` on. As the theory of abstracting causal models becomes more widespread and is used more often, the theory on Markov Abstraction Equivalence Classes and the `pcabs` algorithm will also become more useful.

Concretely, this thesis has attributed to the theoretical foundation of causal models. By adding the new account of Markov Abstraction Equivalence Classes, causal search algorithms are able to find a smaller set of possible causal models based on the Markov Equivalence Class and the partition of a constructive $\tau$-abstraction. By restricting the set of causal models, you increase the value of your data, as the Markov Abstraction Equivalence Class provides a more detailed account in comparison with the Markov Equivalence Class. This is crucial in an era where data is used more and more often. As the importance of data grows, the best use of this data also becomes more important. So, even though this thesis is quite theoretical, it contributes to a concrete goal: a better and more efficient use of the data at hand.

# 10 | Bibliography

Andersson, S. A., D. Madigan, and M. D. Perlman. 1997. A Characterization of Markov Equivalence Classes for Acyclic Digraphs. *The Annals of Statistics* 25(2), 505-541.

Beckers, S., F. Eberhardt, and J. Y. Halpern. 2019. Approximate Causal Abstraction. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence.*

Beckers, S. and J. Y. Halpern. 2019. Abstracting Causal Models. In *Proc. Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19).*

Chalupka K., P. Perona, and F. Eberhardt. 2016. Multi-Level Cause-Effect Systems. In *Proc. 19th Int. Conf. on Artificial Intelligence and Statistic (AISTATS 2016)*, 361-369.

Eberhardt, F. 2009. Introduction to the Epistemology of Causation. *Philosophy Compass* 4(6) 913-925.

He, Y., J. Jia, and B. Yu. 2015. Counting and Exploring Sizes of Markov Equivalence Classes of Directed Acyclic Graphs. *Journal of Machine Learning Research* 16, 2589-2609.

Hitchcock, Christopher, "Causal Models", in *Stanford Encyclopedia of Philosophy* (`https://plato.stanford.edu/entries/causal-models/`).

Malinsky D. and D. Dansk. 2017. Causal discovery algorithms: A practical guide. *Philosophy Compass* 13, 1-11.

Mooij, J. M., J. Peters, D. Janzing, J. Zscheischler, and B. Schölkopf. 2016. Distinguishing Cause from Effect Using Observational Data: Methods and Benchmarks. *Journal of Machine Learning Research* 17, 1-102.

Pearl, J. 1985. Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning. *Seventh Annual Conference of the Cognitive Science Society (August 1985).*

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference.* New York: Cambridge University Press.

Pearl, J. and D. Mackenzie. 2018. *The Book of Why.* New York: Basic Books.

Rubenstein, P. K., S. Weichwald, S. Bongers, J. M. Mooij, D. Janzing, M. Grosse-Wentrup, M, and B. Schölkopf. 2017. Causal Consistency of Structural Equation Models. In *Proc. 33rd Conference on Uncertainty in Artificial Intelligence (UAI 2017).*

Shalizi, C. H. 2019. *Advanced Data Analysis from an Elementary Point of View.* Unpublished draft.

Spirtes, P. and C. Glymour. 1991. An Algorithm for Fast Recovery of Sparse Causal Graphs. *Social Science Computer Review* 9(1), 62-72.

Spirtes, P. and R. Scheines. 2000. *Causation, Prediction, and Search.* Cambridge: The MIT Press.

Verma, T. S. and J. Pearl. 1991. Equivalence and Synthesis of Causal Models. *Uncertainty in Artificial Intelligence* 6, 255-268.

Woodward, J. 2016. The problem of variable choice. *Synthese* 193, 1047-1072.

Yu, K., J. Li and L. Liu. 2016. A Review on Algorithms for Constraint-based Causal Discovery. 1-17.

# 11 | Appendix

In this appendix, I describe and provide the code of `pcabs`. The theory behind `pcabs` is explained in Chapter 7 and 8, so I will only focus on the use of `pcabs` here. In the first section, I describe a simple example of how to create the Markov Abstraction Equivalence Class from a causal model with a constructive $\tau$-abstraction, by using `pcabs`. In the second section, the complete code of `pcabs` is provided. Each function is introduced with a short description about what it does, what the input is and what the output is for the function. Some functions are also supplemented with comments to denote what is happening. All code is written in `R`. Each comment starts with a hashtag (#). The code of `pcabs` can also be found on `https://github.com/gekepals/pcabs`.

## 11.1 How to use pcabs

The following code uses `pcabs` to create the Markov Abstraction Equivalence Class, as discussed in Chapter 8. At the start of your project, you need to make sure that you have added the `pcalg` and `pcabs` package to your project, as both will be used:

```
library(pcalg)
library(pcabs)
```

After this, we simulate the data of the true graph, for which there exists a constructive $\tau$-abstraction. In other words, this is the low-level causal model $M_L$. In this case, we create a simple model consisting of 5 variables $X_1$, $X_2$, $X_3$, $X_4$ and $X_5$, as depicted in Figure 11.1. You may recognize this example as the first example in Section 8.3.

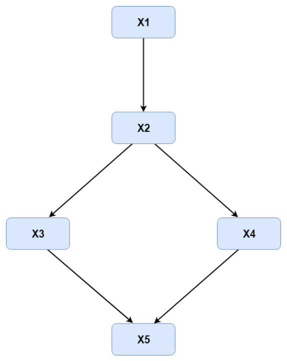


Figure 11.1: Causal model $M_L$.

```
x1 <- rnorm(4000)
x2 <- x1 + rnorm(4000)
x3 <- x2 + rnorm(4000)
x4 <- x2 + rnorm(4000)
x5 <- x3 + x4 + rnorm(4000)
dat <- cbind(x1, x2, x3, x4, x5)
```

As input for `pcabs`, we need a tabular representation of the Markov Equivalence Class of $M_L$, which is depicted in Figure 11.2. We call this the `pdag` of $M_L$. This table stores which variables are connected with each other. The `pdag` of $M_L$ looks as follows:

| | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ |
|---|---|---|---|---|---|
| $X_1$ | 0 | 1 | 0 | 0 | 0 |
| $X_2$ | 1 | 0 | 1 | 1 | 0 |
| $X_3$ | 0 | 1 | 0 | 0 | 1 |
| $X_4$ | 0 | 1 | 0 | 0 | 1 |
| $X_5$ | 0 | 0 | 0 | 0 | 0 |

The row of each variable denotes with which variable it is connected. So, we see that $X_1$ is only connected with $X_2$. In the row of $X_2$, we see that $X_2$ is also connected with $X_1$. This means that the edge between $X_1$ and $X_2$ is undirected. This is right if we look at the Markov Equivalence Class of $M_L$ (Figure 11.2). On the other hand, we see that $X_3$ is connected with $X_5$, but that $X_5$ is not connected with $X_3$. This means that there is an incoming edge from $X_3$ to $X_5$:

$X_3 \to X_5$. This is also right if we look at the Markov Equivalence Class of $M_L$.

We create the `pdag` of $M_L$ by first creating the skeleton of $M_L$ with the `pcalg` package. The skeleton of $M_L$ is the graph of $M_L$ without any directions of edges. From the skeleton, the `pdag` of $M_L$ is created with the `find_pattern` function from `pcabs`.

```
labels <- colnames(dat)
n <- nrow(dat)

skel <- skeleton(suffStat = list(C = cor(dat), n=n),
                 indepTest = gaussCItest, alpha = 0.01,
                 labels = labels, verbose = TRUE)

pdag <- find_pattern(skel)
```

We can also plot this `pdag`, to see the Markov Equivalence Class of $M_L$. This is depicted in Figure 11.2.

```
mec <- as(pdag, "graphNEL")
plot(mec)
```

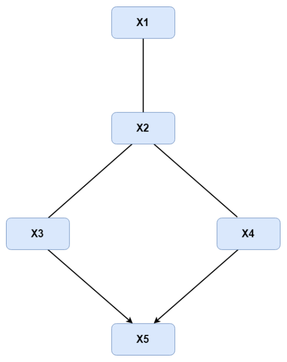


Figure 11.2: The Markov Equivalence Class of $M_L$.

Next, we want to define the partition $P$ of $M_L$. We do so by first defining each $\vec{Z}_i$ of $P$ in a list. Then, we add all $\vec{Z}_i$ together in one list `abs_groups`.

```
a <- list("x1", "x2")
b <- list("x3", "x4", "x5")
abs_groups <- list(a, b)
```

This is all the information that `pcabs` needs. With the `pdag` of $M_L$ and the definition of the partition $P$ of $M_L$, `pcabs` is able to direct certain edges that were not directed in the Markov Equivalence Class of $M_L$. The result of `pcabs` is the Markov Abstraction Equivalence Class. We also plot the Markov Abstraction Equivalence Class. This is depicted in Figure 11.3.

```
pcabs_class <- lead_abs(pdag, abs_groups)

plot_pcabs_class <- as(pcabs_class, "graphNEL")
plot(plot_pcabs_class)
```

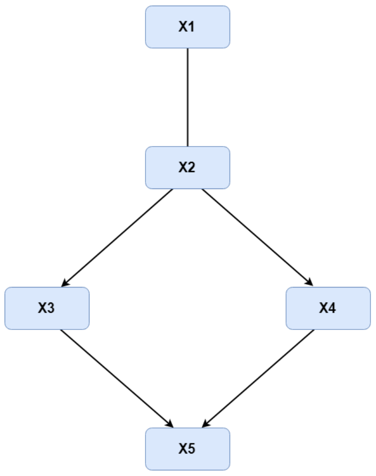


Figure 11.3: The Markov Abstraction Equivalence Class of $M_L$.

## 11.2 pcabs code

```

# The code is divided in two parts:
# 1. The two main functions lead_abs() and add_
    abstraction()
# 2. The helping functions

######


library(pcalg)


#### THE TWO MAIN FUNCTIONS ####


## Leading function ("the leading part")
## takes as input the pdag and all the groups (all Zs)
## loops over the groups to decide which are pairs (i.e.
    which have connected edges)
## decides the order of directing edges between pairs
## first, the pairs are directed that have 1 or more
    directed edges
## last, all pairs that have no directed edges are
    directed
## all pairs are sent to the add_abstraction function to
    direct the edges
## output: the new pdag, with all directed edges in it
lead_abs <- function(pdag, abs_groups){
  original_pdag <- pdag
  undirected_list <- list()
  for(i in 1:length(abs_groups)){
    cat("\nworking on abs_group ", i)
    for(z in i:length(abs_groups)){
      if((z+1) <= length(abs_groups)){
        absList <- find_edge_between_groups(pdag, abs_groups[[i]], abs_groups[[z+1]])
        if(length(absList$abs_group1) > 0){
          cat("\nfound connecting edges")
          group1 <- noquote(gsub("[^0-9]","", absList$abs_group1))
          group2 <- noquote(gsub("[^0-9]","", absList$abs_group2))
          dir_list <- check_directed_edges(pdag, group1, group2)
          if(!is.null(dir_list)){
            cat("\ndirected edges found between groups")
            pdag <- add_abstraction(pdag, absList$abs_group1, absList$abs_group2)
          }
          else {
            cat("\nno directed edges found between groups")
            undirected_list <- append(undirected_list, absList)
          }
        }
      }
    }
  }
  if(length(undirected_list) > 0){
    cat("\nstart with undirected list")
    print(length(undirected_list))
    pdag_changed <- FALSE
    repeat{
      for(i in seq(1, length(undirected_list), by=2)){
        new_pdag <- add_abstraction(pdag, undirected_list[i]$abs_group1, undirected_list[i+1]$abs_group2)
        if(all(new_pdag != "both_directions_possible")){
          pdag_changed <- TRUE
          pdag <- new_pdag
          undirected_list[i] <- NULL
          undirected_list[i] <- NULL
        }
      }
      if(length(undirected_list) == 0 || pdag_changed ==
         FALSE)
        break
      else
        pdag_changed <- FALSE
    }
  }
  # check for cycles on the highlevel!
  cycle_check <- check_highlevel_cycles(pdag, abs_groups)
  if(cycle_check){
    cat("\nERROR: directing the edges creates a cycle on
        the high-level. The original pdag is returned")
    return(original_pdag)
  }
  else {
    cat("\nno cycles found on the high-level. Directing
        the edges as such has been approved.")
    return(pdag)
  }
}


## Directing leading function ("the directing part")
## directs the edges between a pair of clusters
## there are 3 options when an abstraction is added:
## 1) there is no directed edge in the abstraction. This
    means both directions must be tried
## 2) there is 1 directed edge in the abstraction. This
    means all edges must be pointed that way
## 3) there are more than 1 directed edges in the
    abstraction. This means that first, it must
```

```
## be checked whether they all point in the same direction. Then, it must be checked if there
## are undirected edges left. If so, they must point in the same direction as well.
## input: a pair of clusters (abs_group1 and abs_group2)
## output: the updated pdag with directed edges

add_abstraction <- function(pdag, abs_group1, abs_group2){
  abs_group1 <- as.numeric(noquote(gsub("[^0-9]","", abs_group1)))
  abs_group2 <- as.numeric(noquote(gsub("[^0-9]","", abs_group2)))

  dir_list <- check_directed_edges(pdag, abs_group1, abs_group2)
  v_structures <- number_v_nodes(pdag)

  if (length(dir_list) == 0){
    #this means there are no directed edges between the pair
    #we try both directions of the edges

    ## pointing the one way
    pdag1 <- change_direction(pdag, abs_group1, abs_group2)
    dir1 <- is_direction_possible(pdag1, v_structures)

    ## pointing the other way
    pdag2 <- change_direction(pdag, abs_group2, abs_group1)
    dir2 <- is_direction_possible(pdag2, v_structures)

    if(dir1 && dir2){
      #this means that both directions are possible
      #we return the message "both_directions_possible"
```

```
        return("both_directions_possible")
      }
      if(dir1 && !dir2){
        #this means that the direction abs_group1 --> abs_group2 is the only possible direction
        pdag <- apply_mec_rules(pdag1)
        return(pdag)
      }
      if(!dir1 && dir2){
        #this means that the direction abs_group1 <- abs_group2 is the only possible direction
        pdag <- apply_mec_rules(pdag2)
        return(pdag)
      }
      if(!dir1 && !dir2){
        #this means that both directions are not possible
        #the original pdag is returned, without any directed edges
        return(pdag)
      }

    }
    else if (length(dir_list) == 2){
      #this means that there is one directed edge between the pair of clusters
      #we convert the other edges the same way
      group1_check <- check_direction(pdag, dir_list, abs_group1)
      if(!(group1_check)){
        q <- abs_group1
        abs_group1 <- abs_group2
        abs_group2 <- q
      }
      pdag <- change_direction(pdag, abs_group1, abs_group2)
      check <- is_direction_possible(pdag, v_structures)
      if(check){
        pdag <- apply_mec_rules(pdag)
        return(pdag)
      }
      else{
        cat("ERROR: directing the edges is not possible.
            Check if something is wrong with the graph.")
        return(pdag)
      }

    }
    else {
      #this means that multiple directed edges are found
          between the pair of clusters
      #we need to check if they are directed the same way,
          and direct the other edges
      leftrightlist <- list_dir_edges(dir_list)

      result <- match_dir_edges(leftrightlist$left,
          leftrightlist$right, abs_group1)

      if(result == "error"){
        cat("ERROR: the edges are not directed the same way
            . An abstraction is impossible!")
        return(pdag)
      }
      else if(result == "left"){
        #the directed edges point the same way
        for(i in abs_group1){
          if (!(i %in% leftrightlist$left)){
            #not all edges are directed yet
            group1_check <- check_direction(pdag, dir_list,
                abs_group1)
            if(!(group1_check)){
              q <- abs_group1
              abs_group1 <- abs_group2
              abs_group2 <- q
            }
            pdag <- change_direction(pdag, abs_group1, abs_group2)
            check <- is_direction_possible(pdag, v_structures)
            if(check){
              pdag <- apply_mec_rules(pdag)
              return(pdag)
            }
            else{
              cat("ERROR: directing the edges is not possible. Check if something is wrong with the graph.")
              return(pdag)
            }
          }
          else{
            #all edges are directed already. No extra edges need to be directed.
            return(pdag)
          }
        }
      }
      else if(result == "right"){
        #the directed edges point the same way
        for(i in abs_group1){
          if (!(i %in% leftrightlist$right)){
            #not all edges are directed yet
            group1_check <- check_direction(pdag, dir_list, abs_group1)
            if(!(group1_check)){
              q <- abs_group1
              abs_group1 <- abs_group2
              abs_group2 <- q
            }
            pdag <- change_direction(pdag, abs_group1, abs_group2)
            check <- is_direction_possible(pdag, v_structures)
            if(check){
              pdag <- apply_mec_rules(pdag)
              return(pdag)
            }
            else{
              cat("ERROR: directing the edges is not possible. Check if something is wrong with the graph.")
              return(pdag)
            }
          }
          else{
            #all edges are directed already. No extra edges need to be directed.
            return(pdag)
          }
        }
      }
    }
  }


  #### HELP FUNCTIONS ####


  ## NOTE: code from pcalg package
  ## function to produce the pdag from the skeleton
  ## the pdag is a tabular representation of the mec
  ## input parameter: skeleton (from pcalg)
  ## output: pdag of mec
  find_pattern <- function(skel){
    res <- skel
    g <- as(skel, "matrix")
    p <- as.numeric(dim(g)[1])
    pdag <- g
    ind <- which(g == 1, arr.ind = TRUE)
    for (i in seq_len(nrow(ind))) {
      x <- ind[i, 1]
      y <- ind[i, 2]
      allz <- setdiff(which(g[y, ] == 1), x)
      for (z in allz) {
        if (g[x, z] == 0 && !(y %in% skel@sepset[[x]][[z]] ||
                                  y %in% skel@sepset[[z]][[x]])) {
          pdag[x,y] <- pdag[z,y] <- 1
          pdag[y,x] <- pdag[y,z] <- 0
        }
      }
    }
    return(pdag)
  }


  ## function to find the edges between two groups of variables
  ## input: the pdag, the first group and the second group
  ## output: two lists in one list absList:
  ## absList$abs_group1 = in order, the variables that are connected to group_2
  ## absList$abs_group2 = in order, the variables that are connected to group_1
  ## so both lists are in order and define which variable is connected to which
  find_edge_between_groups <- function(pdag, group1, group2){
    t = 0
    abs_group1 <- list()
```

```r
  abs_group2 <- list()
  for(i in group1){
    conn_list = find_conn_edges(pdag, i)
    for(z in conn_list){
      if(z %in% group2){
        t = t+1
        abs_group1[t] <- i
        abs_group2[t] <- z
      }
    }
  }
  absList <- list("abs_group1" = abs_group1, "abs_group2"
      = abs_group2)
  return(absList)
}


## function to find the variables connected to a certain
    variable
## input: the pdag, and the variable for which you want
    to find the connected edges
## for example: x2 is connected to x1, x4, x5
## output: list of the connected edges
find_conn_edges <- function(pdag, row){
  conn_list = list()
  t = 0
  z = 0
  for(i in pdag[row,]){
    t = t+1
    if(i == 1){
      z = z+1
      conn_list[z] = colnames(pdag)[t]
    }
  }
  return(conn_list)
}
```

```r


## function to check for directed edges between two groups of variables
## input: the two groups of variables
## output: the list of directed edges (if any) in one list
## if no directed edges are found, return is NULL
check_directed_edges <- function(pdag, abs_group1, abs_group2) {
  ind <- which((pdag == 1 & t(pdag) == 0), arr.ind = TRUE)
  dir_edge <- FALSE
  ind_no <- list()
  t <- 1

  for (i in seq_len(nrow(ind))){
    if(ind[i,1] %in% abs_group1){
      if(ind[i,2] %in% abs_group2){
        dir_edge = TRUE
        ind_no[[t]] <- ind[i,1]
        t <- t+1
        ind_no[[t]] <- ind[i,2]
        t <- t+1
      }
    }
    else if (ind[i,2] %in% abs_group1){
      if(ind[i,1] %in% abs_group2){
        dir_edge = TRUE
        ind_no[[t]] <- ind[i,1]
        t <- t+1
        ind_no[[t]] <- ind[i,2]
        t <- t+1
      }
    }
  }

  if(dir_edge){
    return(ind_no)
  }
  else{
    return(NULL)
  }
}


## function to check the number of v-structures in a graph
## input: the pdag
## output: the number of v-structures
## this is useful for comparing the first MEC with the second MEC with abstraction:
## if the number of v-structures is not equal, the given abstraction is not valid
number_v_nodes <- function(pdag) {
  ind <- which((pdag == 1 & t(pdag) == 0), arr.ind = TRUE)
  v_nodes <- ind[duplicated(ind[,2]),2]
  no_v_nodes <- length(v_nodes)
  return(no_v_nodes)
}


## function that changes all undirected edges into directed edges
## the edges are directed as abs_group1 --> abs_group2
## input: the pdag and the two abstraction groups
## output: the directed pdag
change_direction <- function(pdag, abs_group1, abs_group2){
  for(i in seq_along(abs_group1)){
    a <- abs_group1[i]
    b <- abs_group2[i]

    if(pdag[a,b] == 1 & pdag[b,a] == 1){
      pdag[b,a] <- 0
    }
  }
  return(pdag)
}


## this function checks whether a certain direction of the edges is possible
## it first check whether the direction alone causes any more v-structures
## then, it applies the MEC-Rules of the pcalg algorithm, and checks whether
## this causes any more v-structures
## if both don't cause more v-structures, the direction is allowed
## input: the pdag and the number of v-structures in the original MEC
## output: TRUE if direction is allowed, FALSE if direction is not allowed
is_direction_possible <- function(pdag, v_structures){
  possible_direction <- TRUE
  pdag_v <- number_v_nodes(pdag)
  if(v_structures != pdag_v){
    possible_direction <- FALSE
  }
  if(possible_direction){
    ## point other edges according to MEC-rules
    pdag <- apply_mec_rules(pdag)
    pdag_v <- number_v_nodes(pdag)
    if(v_structures != pdag_v){
      possible_direction <- FALSE
    }
  }
  return(possible_direction)
}


## NOTE: code from pcalg pacakge
## function that applies the 3 rules of PC algorithm
## input parameter: pdag
## output: alternated pdag according to the rules
## this function checks if edges must be directed
    according to the rules of mec
apply_mec_rules <- function(pdag){
  g <- as(skel, "matrix")
  p <- as.numeric(dim(g)[1])

  old_pdag <- matrix(0, p, p)

  ## Rule 1
  while (!all(old_pdag == pdag)) {
    old_pdag <- pdag
    ind <- which((pdag == 1 & t(pdag) == 0), arr.ind =
       TRUE)
    for (i in seq_len(nrow(ind))) {
      a <- ind[i, 1]
      b <- ind[i, 2]
      indC <- which((pdag[b, ] ==1 & pdag[, b] ==1) &
                    (pdag[a, ] == 0 & pdag[, a] ==0))
      if (length(indC) > 0) {
        pdag[b, indC] <- 1
        pdag[indC, b] <- 0
      }
    }

    ## Rule 2
    ind <- which((pdag == 1 & t(pdag) == 1), arr.ind =
       TRUE)
```

```r
    for (i in seq_len(nrow(ind))) {
      a <- ind[i, 1]
      b <- ind[i, 2]
      indC <- which((pdag[a, ] == 1 & pdag[, a] == 0) &
                      (pdag[, b] == 1 & pdag[b, ] == 0))
      if (length(indC) > 0) {
        pdag[a, b] <- 1
        pdag[b, a] <- 0
      }
    }

    ## Rule 3
    ind <- which((pdag == 1 & t(pdag) ==1), arr.ind = TRUE)
    for (i in seq_len(nrow(ind))) {
      a <- ind[i, 1]
      b <- ind[i, 2]
      indC <- which((pdag[a, ] == 1 & pdag[, a] == 1) &
                      (pdag[, b] == 1 & pdag[b, ] == 0))
      if (length(indC) >= 2) {
        g2 <- pdag[indC, indC]
        if (length(g2) <= 1) {
          g2 <- 0
        }
        else {
          diag(g2) <- rep(1, length(indC))
        }
        if (any(g2 == 0)) {
          pdag[a, b] <- 1
          pdag[b, a] <- 0
        }
      }
    }
  }
  return(pdag)
}
```

```r


## function to check the direction of the variables in dir_list
## assumptions: the variables in the abs_groups are directed equally,
## and there is at least 1 directed edge in abs_group1
## (this has been checked before the function is called)
## input: the pdag, the dir_list (from check_directed_edges function) and abs_group1
## if the direction is abs_group1 --> abs_group2, return group1 = TRUE
## if the direction is abs_group1 <-- abs_group2, return group1 = FALSE
check_direction <- function(pdag, dir_list, abs_group1){
  a <- dir_list[[1]]
  b <- dir_list[[2]]
  if(pdag[a,b] == 1 & pdag[b,a] == 0){
    #this means the edge is directed as a -> b
    if (a %in% abs_group1){
      return(group1 <- TRUE)
    }
    else{
      return(group1 <- FALSE)
    }
  }
  if(pdag[b,a] == 1 & pdag[a,b] == 0){
    #this means the edge is directed as a <- b
  }
  if (b %in% abs_group1){
    return(group1 <- FALSE)
  }
  else{
    return(group1 <- TRUE)
  }
}
```

```
## function to put the results of check_directed_edges
    function into two lists:
## one list with all the nodes found on the left side,
    one with nodes on the right side
## the functions thus splits up the list found in check_
    directed_edges
## input: dir_list (from check_directed_edges function)
## output: two lists: left_list and right_list
list_dir_edges <- function(dir_list){
  left_list <- list()
  right_list <- list()
  l <- 1
  r <- 1
  for (i in seq(1, length(dir_list), 2)){
    left_list[l] <- dir_list[[i]]
    l <- l+1
  }
  for (i in seq(2, length(dir_list), 2)){
    right_list[r] <- dir_list[[i]]
    r <- r+1
  }
  leftright_list <- list("left" = left_list, "right" =
     right_list)
  return(leftright_list)
}


## function to check if variables in leftlist are in
    either absgroup1 or absgroup2
## this is necessary, otherwise there has been made a
    mistake, in the given absgroups
## input: leftlist and rightlist (from list_dir_edges
    function), and abs_group1
## if there is a mistake, then leftlist is both in
```

```
    absgroup1 and absgroup2
## the function then returns "error"
## else, the function returns "left" when leftlist is in
    absgroup1
##        this means an abstraction of absgroup1 -->
    absgroup2
## the function return "right" when leftlist is in
    absgroup2
##        this means an abstraction of absgroup1 <--
    absgroup2
match_dir_edges <- function(leftlist, rightlist,
    absgroup1){
  result <- "error"
  group1 <- FALSE
  group2 <- FALSE
  for (i in leftlist){
    if(i %in% absgroup1){
      group1 <- TRUE
    }
    else{
      group2 <- TRUE
    }
  }

  if(group1 && group2){
    return(result)
  }
  if(group1){
    for(i in rightlist){
      if(i %in% absgroup1){
        group1 <- FALSE
      }
      else{
        group2 <- TRUE
      }
    }
    if(group1 && group2){
      ##in this case, leftlist is in absgroup1, rightlist
          is in absgroup2
      result <- "left"
      return(result)
    }
  }
  else if(group2){
    for(i in rightlist){
      if(i %in% absgroup1){
        group1 <- TRUE
      }
      else{
        group2 <- FALSE
      }
    }
    if(group1 && group2){
      ##in this case, leftlist is in absgroup2, rightlist
          is in absgroup1
      result <- "right"
      return(result)
    }
    else{
      return(result)
    }
  }
}


## function to check whether there are cycles on the high
    -level
## input: low-level pdag and abs_groups
## output: value of cycle_check:
## if TRUE: a cycle exists on the high-level
## if FALSE: no cycle exists on the high-level
check_highlevel_cycles <- function(pdag, abs_groups){
  highlevel_pdag <- construct_highlevel(pdag, abs_groups)
  cycle_check <- find_cycle(highlevel_pdag)
  return(cycle_check)
}


## function to construct the high-level pdag
## that is: the table in which the edges between the
    clusters are defined
## input: the low-level pdag and abs_groups
## output: the high-level pdag
construct_highlevel <- function(pdag, abs_groups){
  p <- length(abs_groups)
  highlevel_pdag <- matrix(0, p, p)
  for(i in 1:length(abs_groups)){
    for(z in 1:length(abs_groups)){
      absList <- find_edge_between_groups(pdag, abs_
          groups[[i]], abs_groups[[z]])
      if(length(absList$abs_group1) > 0){
        group1 <- noquote(gsub("[^0-9]","", absList$abs_
            group1))
        group2 <- noquote(gsub("[^0-9]","", absList$abs_
            group2))
        dir_list <- check_directed_edges(pdag, group1,
            group2)
        if(!is.null(dir_list)){
          outgoing_edge <- check_direction(pdag, dir_list
              , group1)
          if(outgoing_edge){
            highlevel_pdag[i, z] <- 1
          }
        }
      }
    }
  }
  return(highlevel_pdag)
```

```r
}


## function to check whether there is a cycle in the model
## input: pdag
## output: cycle value which is TRUE if a cycle is found, FALSE otherwise
find_cycle <- function(pdag){
  cycle <- FALSE
  ind <- which((pdag == 1 & t(pdag) == 0), arr.ind = TRUE)
  for (i in seq_len(nrow(ind))) {
    a <- ind[i, 1]
    b <- ind[i, 2]
    c <- which((ind[,1]==b))

    while(length(c) > 0){
      b <- ind[c,2]
      if(a %in% b){
        break
      }
      c <- which((ind[,1]==b))
    }
    if(a %in% b){
      cycle <- TRUE
    }
  }
  return(cycle)
}
```