A SYMBOLIC GENERALIZATION OF PROBABILITY THEORY

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Adnan Y. Darwiche

January 1993

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Matthew L. Ginsberg
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Yoav Shoham

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Ross Shachter

Approved for the University Committee
on Graduate Studies:

Abstract

Three questions motivate much work in AI. How should an agent's state of belief be represented? How should an agent change its state of belief upon recording an observation? And what is a practical way for domain experts to convey their states of belief to agents?

Probability calculus provides answers to these questions: A state of belief should be (1) represented by a probability function over some language, (2) changed using probabilistic conditionalization, and (3) conveyed using a probabilistic causal network. Despite the popularity of these answers, domain experts have often complained about their commitment to numeric degrees of belief. In this thesis, I attempt to address this complaint by suggesting an abstract belief calculus that is not committed to numbers (nor to any specific set of degrees of belief) and yet has the key features of probability calculus. The abstract calculus has three components: (1) Abstract states of belief, (2) abstract conditionalization, and (3) abstract causal networks. The calculus is also equipped with an algorithm for computing degrees of belief, which corresponds to a popular algorithm in the probabilistic literature.

I present many concrete instances of the proposed abstract belief calculus. Some of these instances are well known, such as proposition, possibility, and probability calculi. But other instances are novel, such as objection calculus. I also show that objection calculus is closely related to clause management and diagnosis systems — which are influential in AI — and study the ramifications of this relation.

Acknowledgements

I am grateful to my advisor, Matthew Ginsberg, for motivating this thesis, for teaching me how to do and communicate research, and for being very critical. Thanks Matt!

I would also like to thank the other members of my thesis committee, Yoav Shoham and Ross Shachter.

The PRINCIPIA group at Stanford has been both stimulative and educative. Special thanks to Will Harvey, Don Geddis, Ari Jónsson, Scott Roy, and Narinder Singh. I would also like thank Gidi Avrahami, Aaron Goldberg, Alon Levy, and Adam Grove, who contributed in their own ways to my stay at Stanford.

I was introduced to AI while I was obtaining a Bachelor of Science in Civil Engineering. Since then, I have started a journey towards computer science that has been made possible by the help and support of Nabil Qaddumi, Amr Azzouz, Mervat Geith, Barbara Hayes–Roth, and, especially, Raymond Levitt. Thanks to you all.

I am deeply indebted to my brother Ahmad and my friend Nouria for their crucial support during my most difficult times at Stanford.

My parents, brothers, and sisters, have been giving meaning to my life. This thesis is dedicated to them.

To my wife, Jinan, I shall plainly say: Thanks for everything ...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The basic question underlying this thesis was motivated by two personal experiences. In this chapter, I introduce these experiences and the question they have motivated.

———

I was at the Knowledge Systems Laboratory at Stanford University during the academic year of 1989, taking part in an effort to build an intelligent agent for real–time monitoring and control. The agent was called GUARDIAN because its goal was to guard patients in Intensive Care Units (ICUs). My initial task was to maintain and improve a component of GUARDIAN called REACT. This component had the following responsibilities: possess a state of belief about a patient in the ICU, change this state of belief as more information became available, and react when the patient was thought to be in danger.

Given these responsibilities, it was clear to me that a number of questions needed to be addressed: How should REACT's state of belief be represented? How should REACT change its state of belief as it recorded new observations? And how should a domain expert convey her own state of belief to REACT?

Following up on work that was done by another student, I referred to the probabilistic literature for answers:

1. A state of belief is a probability function over a propositional language.

2. A state of belief is changed using probabilistic conditionalization.

3. A state of belief is conveyed using a probabilistic causal network.

These answers turned out to have a number of desirable features.

First, a probabilistic state of belief admits uncertain propositions, which is a celebrated feature in real–world applications. Classical logic, by comparison, allows only true and false propositions. Second, probabilistic conditionalization as a commitment for changing a state of belief leads to plausible patterns of belief change [Polya, 1954; Pearl, 1988]. Third, and probably most important, constructing a consistent probabilistic causal network is straightforward. An additional attraction of probabilistic causal networks is that they lay the foundation for a number of efficient algorithms for computing conditional and unconditional probabilities.

In spite of all the desirable features I just mentioned, I did face a major challenge when using probability calculus to support the functionality of REACT. Specifically, on more than one occasion, the domain expert refused to give me the probabilities I wanted: "I don't have these numbers" was a common reply! And even when I did manage to get them, other members of the project felt uncomfortable using some probabilities that REACT computed: "I don't want to base this decision on numbers," is one statement that I still remember vividly. This does not seem to have been an isolated experience. Other researchers have pointed to similar difficulties in using probability calculus and other numerical approaches:

> One difficulty is that while it is relatively easy to elicit tentative propositional rules from experts and from people in general, it is considerably harder to get the commitment to particular grades of certainty ... Worse still, individual informants frequently vary in their answers to a repeated question depending on the day of the week, their emotional state, the preceding questions, and other extraneous factors. [Doyle, 1990]

To be fair though, the people I worked with had mixed feelings about probability calculus. On the one hand, they were very impressed by its features that I have discussed above. On the other hand, they found the commitment to numbers to be a very high price for these features. But I did continue using probability calculus to support the functionality of REACT, not by choice, but for lack of a satisfactory alternative. Again, this does not seem to have been an isolated experience:

> Understandably, expert system designers have difficulty justifying their use of the numerical judgements in face of these indications of psychological and pragmatic unreality. Unfortunately, they have had to stick to their guns, since no satisfactory alternative has been apparent. [Doyle, 1990]

———

I joined the Ph.D. program at Stanford University in the fall of 1989. In the spring of 1990, I decided to write a thesis on probabilistic temporal reasoning under the supervision of Dr. Matthew Ginsberg. Although I was able to convince Dr. Ginsberg that "uncertain" temporal reasoning is important to AI, it was hard to convince him that "probabilistic" temporal reasoning was the way to go. His argument was simple, "The desirable features of probability calculus do not justify the commitment to numbers." I tried hard to change his mind, but with no success.

During my attempt to convince Dr. Ginsberg, I was puzzled by the following questions: If numbers were intrinsic to the features of probability calculus, then why was there no proof of this? And if they were not, then why was there no symbolic belief calculus that has the desirable features of probability calculus?

Having no answers to these questions, my determination to convince Dr. Ginsberg to adopt probability calculus was gradually replaced by an occupation with the following question: Is it possible to relax the commitment to numbers while retaining the desirable features of probability calculus? The answer to this question and its ramifications are what this thesis is about.

———

As I shall demonstrate, it is possible to relax the commitment to numbers without losing the key features responsible for the success of probability calculus.

The demonstration is constructive. I shall present in Chapters 2–5 a belief calculus that is not committed to numbers and yet has the key features of probability calculus. The calculus is abstract in the sense that it is not committed to either numeric or symbolic degrees of belief. It is comprehensive and has three major components: Abstract states of belief, abstract conditionalization, and abstract causal networks. These components subsume their probabilistic counterparts, and they are discussed in Chapters 2, 3, and 4.

Similar to their probabilistic counterparts, abstract causal networks lay the foundation for algorithms that compute conditional and unconditional abstract degrees of belief. Chapter 5 presents such an algorithm and Chapter 6 presents a Common Lisp implementation.

Beyond demonstrating that numbers are not strictly needed, the abstract calculus unifies a number of concrete calculi that exist in the uncertainty literature. This role shall be evident from instances of the calculus that are given in Chapters 2 and 3. All of these calculi, however, employ numeric degrees of belief. This observation has motivated the creation of objection calculus, which employs symbolic degrees of belief. Objection calculus is discussed in Chapter 7.

Being symbolic, objection calculus enhances the unifying role of the abstract calculus. Moreover, and probably more important, objection calculus is closely related to clause management and diagnosis systems, which are influential in AI. This relation and its ramifications are studied in Chapter 8.

The results in this thesis have sparked a number of important questions. I try to answer some of these questions in Chapter 9. Finally, I conclude the thesis in Chapter 10 by a summary of results, a review of some technical limitations, and a discussion on future work.

Chapter 2

Abstract States of Belief

In this chapter, I formalize the notion of abstract states of belief and show that some AI representations can be viewed as instances of abstract states of belief.

2.1 What is a state of belief?

In probability calculus, a state of belief is usually defined as a mapping from a language $\mathcal{L}$ into the interval $[0, 1]$. This representation is restrictive because of its commitment to numbers as degrees of belief. I adopt a more general representation of a state of belief that is not committed to numeric degrees of belief:

Assumption 2.1.1 *An state of belief is a mapping from a propositional language $\mathcal{L}$ into a set of quantities $\mathcal{S}$.*

In principle, requiring that $\mathcal{L}$ be a propositional language is not necessary. But relaxing this requirement leads to states of belief that are outside the scope of this thesis.

The quantities in $\mathcal{S}$ are called *degrees of support.* They are neither strictly numeric nor strictly symbolic. Degrees of support can be integers, rationals, and even logical sentences. A degree of support is a primitive concept that derives its meaning from the operations and relations that are defined on degrees of support.

Degrees of support could be positive or negative. Positive degrees of support quantify the support for a sentence, while negative degrees of support quantify the support against a sentence. Positive and negative degrees of support are dual: The positive support for a sentence is the negative support for its negation, and vice versa. I shall consider concrete states of belief that use positive degrees of support. I shall also consider concrete states of belief that use negative degrees of support.

There is a difference between degrees of support and degrees of belief. Although I did not introduce degrees of belief yet, we shall see later that the degree of belief in a sentence is determined by the degree of support for the sentence and the degree of support for the negation of that sentence.

I assume that degrees of support are useful in the following sense.

Definition 2.1.2 *A degree of support is* <u>*useful*</u> *precisely when it is attributed to a sentence by some state of belief.*

Notation a, b, c, and s denote degrees of support in $\mathcal{S}$.

2.2 Coherence and normality

The notion of a state of belief as embodied by Assumption 2.1.1 fails to capture some intuitions about "coherent" states of belief. For example, let the degrees of support be {*possible, impossible*}, and consider a state of belief that

- attributes *possible* to sentence A,
- attributes *impossible* to sentence B, and
- attributes *impossible* to sentence $A \vee B$.

Intuition says that this state of belief is incoherent because we expect the support for $A \vee B$ to be *possible* in this case. To exclude such incoherent states of belief, we need to impose more constraints on the notion of a state of belief. Below are some constraints that I have identified.

Axiom 1 *The support for A equals the support for B when A and B are logically equivalent.*

Axiom 2 *The support for $A \vee B$ is determined by the support for A and the support for B when A and B are logically disjoint.*

Axiom 3 *If A entails B, if B entails C, and if A has the same support as C, then B has the same support as A and C.*

Theorem 2.2.1 *Axiom 3 implies Axiom 1.*

It is common to compare the degree to which a sentence is supported by different states of belief. Such a comparison is meaningful only if different states of belief use the same support scale. Below are two constraints that unify the support scales used by different state of belief.

Axiom 4 *Unsatisfiable sentences have the same support across all states of belief.*

Axiom 5 *Valid sentences have the same support across all states of belief, which is different from the support for unsatisfiable sentences.*

A formal statement of Axioms 1–5 is given in Appendix C.

2.3 Defining abstract states of belief

Axioms 2–5 lead to a number of formal properties. First, they lead to the existence of an algebraic structure given by the following definition:

Definition 2.3.1 *A partial support structure is a pair* $\langle \mathcal{S}, \oplus \rangle$*, where*

1. $\mathcal{S}$ *is a set containing at least two elements.*

2. $\oplus$ *is a partial function from* $\mathcal{S} \times \mathcal{S}$ *to* $\mathcal{S}$ *that satisfies the following properties:*

   (X0) $a \oplus b = b \oplus a$.

   (X1) $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.

   (X2) If $(a \oplus b) \oplus c = a$*, then also* $a \oplus b = a$.

   (X3) There is a unique element $\mathbf{0}$ *in* $\mathcal{S}$ *that satisfies the following property: for all* a *in* $\mathcal{S}$, $a \oplus \mathbf{0} = a$.

   (X4) There is a unique element $\mathbf{1} \neq \mathbf{0}$ *in* $\mathcal{S}$ *that satisfies the following property: for all* a *in* $\mathcal{S}$*, there exists* b *in* $\mathcal{S}$*, such that* $a \oplus b = \mathbf{1}$.

The function $\oplus$ is called *support summation*, $\mathbf{0}$ is called *zero support*, and $\mathbf{1}$ is called *full support*. Table 1 lists some partial support structures.

Axioms 2–5 also lead to properties of states of belief that are embodied by the following definition:

	$\mathcal{S}$	$a \oplus b$	$\mathbf{0}$	$\mathbf{1}$
Proposition	$\{0, 1\}$	$\max(a, b)$	0	1
Probability	$[0, 1]$	$a + b$	0	1
Improbability	$[0, 1]$	$a + b - 1$	1	0
Possibility	$[0, 1]$	$\max(a, b)$	0	1
Impossibility	$\{0, 1, \ldots, \infty\}$	$\min(a, b)$	∞	0

Table 1: Examples of partial support structures.

Definition 2.3.2 *Let $\langle \mathcal{S}, \oplus \rangle$ be a partial support structure and $\mathcal{L}$ be a propositional language. A state of belief Φ with respect to $\langle \mathcal{S}, \oplus \rangle_{\mathcal{L}}$ is a mapping from $\mathcal{L}$ to $\mathcal{S}$ that satisfies the following properties:*

1. $\Phi(A) = \Phi(B)$ *when* $\models A \equiv B$.
2. $\Phi(A \vee B) = \Phi(A) \oplus \Phi(B)$ *when* $\models \neg(A \wedge B)$.
3. $\Phi(\mathbf{false}) = \mathbf{0}$.
4. $\Phi(\mathbf{true}) = \mathbf{1}$.

Definition 2.3.1 of a partial support structure and Definition 2.3.2 of a state of belief are consequences of four theorems. These theorems refer to the notion of a meaningful sum, which I did not introduce yet. Therefore, I shall introduce this notion first, and then state the theorems.

The definition of a partial support structure does not specify the domain of support summation—the definition only says that support summation is a partial function. The domain of support summation is not specified because the definition of a partial support structure is a consequence of Axioms 2–5, which do not entail the definition of support summation. This raises the question, "Where should support summation be defined?"

If a partial support structure is to be used for modeling states of belief as suggested by Definition 2.3.2, then $a \oplus b$ should be defined precisely when $a \oplus b$ is a meaningful sum. Let me first state what a meaningful sum is and then justify the previous statement.

Definition 2.3.3 *A sum over $\mathcal{S}$ is defined as follows:*

- $a \oplus b$ *is a sum over* $\mathcal{S}$ *if* a *and* b *are in* $\mathcal{S}$.
- $a \oplus b$ *is a sum over* $\mathcal{S}$ *if* a *and* b *are sums over* $\mathcal{S}$.

Definition 2.3.4 *A sum over $\mathcal{S}$ is meaningful precisely when there exists an intuitive state of belief that attributes the supports appearing in the sum to logically disjoint sentences.*[1]

[1]Intuitive states of belief are states of belief that Definition 2.3.2 attempts to formalize.

For example, suppose that the partial support structure is $\langle [0,1], + \rangle$ and degrees of support are frequencies. Then $.4 + .9$ is not a meaningful sum, because there is no state of belief that attributes $.4$ and $.9$ to logically disjoint sentences. However, $.2 + .3$ is a meaningful sum, because there is a state of belief that attributes $.2$ and $.3$ to logically disjoint sentences.

Depending on the domain of support summation, mappings from $\mathcal{L}$ into $\mathcal{S}$ may or may not qualify as states of belief. If $a \oplus b$ is meaningful but not defined, then some intuitive states of belief may not be captured by Definition 2.3.2. Moreover, if $a \oplus b$ is defined but not meaningful, then Definition 2.3.2 may admit mappings that do not correspond to intuitive states of belief. This is why $a \oplus b$ should be defined precisely when $a \oplus b$ is a meaningful sum.

The existence of a partial support structure and the properties of states of belief given by Definition 2.3.2 are consequences of the following theorems:

Theorem 2.3.5 *Assume Axiom 1. If Axiom 2 holds, then Properties (X0) and (X1) hold over meaningful sums.*

The above theorem proves that support summation is commutative and associative.

Theorem 2.3.6 *Assume Axioms 1–2. Axiom 3 holds precisely when Property (X2) holds over meaningful sums.*

The next two theorems prove the existence of zero and full supports, respectively.

Theorem 2.3.7 *Axiom 4 implies Property (X3) given Axioms 1–2.*

Theorem 2.3.8 *Axiom 5 implies Property (X4) given Axioms 1–3.*

2.4 Absolute attitudes

Abstract states of belief hold absolute and comparative attitudes towards sentences. I discuss absolute attitudes in this section and comparative attitudes in Section 2.5.

There are two types of absolute attitudes towards sentences. The first type has the form, "I support A to degree s," while the second type has the form "I believe A to degree d."

Definition 2.4.1 *The degree to which a state of belief Φ supports sentence A is $\Phi(A)$.*

The support for a sentence does not determine the support for its negation in general. For example, if degrees of support are {*possible, impossible*}, and if *possible* is the support for A, then the support for $\neg A$ could be either *possible* or *impossible.*

Definition 2.4.2 *The degree to which a state of belief Φ believes sentence A, written $\ddot{\Phi}(A)$, is $\langle \Phi(A), \Phi(\neg A)\rangle$.*

On the other hand, the belief in a sentence does determine the belief in its negation. Moreover, degrees of belief are closer to truth values in many–valued logics [Rosser and Turquette, 1952; Rescher, 1969; Ginsberg, 1988] than are degrees of support. For example, if degrees of support are {*possible, impossible*}, then

1. $\langle possible, impossible\rangle$ represents the truth value *true.*

2. $\langle impossible, possible\rangle$ represents the truth value *false.*

3. $\langle possible, possible\rangle$ represents the truth value *unknown.*

In general, every abstract state of belief has two extreme belief attitudes that correspond to truth and falsity. These are the attitudes of acceptance and rejection.

Definition 2.4.3 *A state of belief Φ accepts sentence A precisely when $\ddot{\Phi}(A) = \langle \mathbf{1}, \mathbf{0}\rangle$.*

Definition 2.4.4 *A state of belief Φ rejects sentence A precisely when $\ddot{\Phi}(A) = \langle \mathbf{0}, \mathbf{1}\rangle$.*

The correspondence between acceptance and truth and between rejection and falsity becomes evident when we discuss the process of observing in Chapter 3. As we shall see, acceptance and rejection are attitudes that can never be given up as a result of recording more observations.

A sentence is rejected precisely when it is supported to degree $\mathbf{0}$. And a sentence is accepted only if it is supported to degree $\mathbf{1}$. But if the sentence is supported to degree $\mathbf{1}$, it is not necessarily accepted. For example, when degrees of support are $\{$*possible, impossible*$\}$, a sentence and its negation could be *possible.* Here, both the sentence and its negation are supported to degree *possible,* but neither is accepted.

2.5 Comparative attitudes

Abstract states of belief hold two types of comparative attitudes towards sentences. The first type has the form, "I support A no more than I support B," while the second type has the form, "I believe A no more than I believe B."

Comparative attitudes are based on ordering degrees of support and belief. Degrees of support can be ordered using support summation. The intuition here is that the sum of two supports is at least as great as each of the summands.

Definition 2.5.1 *Support a is no greater than b, written $a \preceq_\oplus b$, precisely when there is a support c such that $a \oplus c = b$. The relation $\preceq_\oplus$ is called a support order.*

Table 2 lists some support orders. The following theorem shows that every support order is a partial order with minimal and maximal elements:

Theorem 2.5.2 *The relation $\preceq_\oplus$ is a partial support order under which* **0** *is minimal and* **1** *is maximal.*

Degrees of belief can also be ordered.

Definition 2.5.3 *Degree of belief $\langle s_1, s_2\rangle$ is no greater than degree of belief $\langle s_3, s_4\rangle$, written $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle s_3, s_4\rangle$, precisely when $s_1 \preceq_\oplus s_3$ and $s_2 \succeq_\oplus s_4$. The relation $\sqsubseteq_\oplus$ is called a belief order.*

The belief order is also a partial support order with minimal and maximal elements:

	$\langle \mathcal{S}, \oplus\rangle$	$a \preceq_\oplus b$
Proposition	$\langle\{0,1\}, \max\rangle$	$a \leq b$
Probability	$\langle[0,1], +\rangle$	$a \leq b$
Improbability	$\langle[0,1], (\lambda\ (a\ b)\ a+b-1)\rangle$	$b \leq a$
Possibility	$\langle[0,1], \max\rangle$	$a \leq b$
Impossibility	$\langle\{0,1,\ldots,\infty\}, \min\rangle$	$b \leq a$

Table 2: Examples of support orders.

Theorem 2.5.4 *The relation* $\sqsubseteq_\oplus$ *is a partial order under which* $\langle \mathbf{0}, \mathbf{1} \rangle$ *is minimal and* $\langle \mathbf{1}, \mathbf{0} \rangle$ *is maximal.*

Now that we have ordered degrees of support and belief, we can define the comparative attitudes held by abstract states of belief.

Definition 2.5.5 *Sentence A is <u>no more supported</u> than B by a state of belief Φ, written $A \preceq_\Phi B$, precisely when $\Phi(A) \preceq_\oplus \Phi(B)$.*

Definition 2.5.6 *Sentence A is <u>no more believed</u> than B by a state of belief Φ, written $A \sqsubseteq_\Phi B$, precisely when $\ddot{\Phi}(A) \sqsubseteq_\oplus \ddot{\Phi}(B)$.*

If two sentences are equally believed, then they are also equally supported. But the converse is not true. For example, when degrees of support are *possible, impossible,* a state of belief Φ may be such that $\Phi(\neg Bird) = impossible$ and $\Phi(Bird) = \Phi(Fly) = \Phi(\neg Fly) = possible$. Although *Bird* and *Fly* are equally supported by Φ, *Bird* is more believed than *Fly*.

Rejected sentences are always minimally supported, and accepted sentences are always maximally supported. But although minimally supported sentences are rejected, maximally supported sentences are not necessarily accepted. A sentence and its negation may be maximally supported at the same time, while neither of them may be accepted.

Rejected sentences are always minimally believed and accepted sentences are always maximally believed. The converse is true as shown by the following theorem:

Theorem 2.5.7 *The relation* $\sqsubseteq_\Phi$ *satisfies the following properties:*

1. $\sqsubseteq_\Phi$ *is a partial order.*

2. $A \models B$ *only if* $A \sqsubseteq_\Phi B$.

3. *For all A,* $\mathbf{false} \sqsubseteq_\Phi A \sqsubseteq_\Phi \mathbf{true}$.

4. *A is rejected precisely when A is minimal under* $\sqsubseteq_\Phi$.

5. *A is accepted precisely when A is maximal under* $\sqsubseteq_\Phi$.

2.6 Concrete states of belief

In this section, I present a number of concrete partial support structures that induce states of belief in propositional logic, probability calculus, and nonmonotonic logic based on preferential models [Kraus *et al.*, 1990]. In Chapter 7, I present a partial support structure that has symbolic degrees of support.

2.6.1 Propositional

Below is the simplest partial support structure.

Theorem 2.6.1 *The pair* $\langle \{0, 1\}, \max \rangle$ *is a partial support structure.* ∎

A state of belief with respect to this structure is called a *propositional state of belief.* A propositional state of belief Φ places a sentence A into one of three classes:

1. A is rejected: $\Phi(A) = 0$.
2. A is accepted: $\Phi(\neg A) = 0$.
3. A is undetermined: $\Phi(A) = 1$ and $\Phi(\neg A) = 1$.

This is the classification of a propositional logic database: A sentence A is entailed by the database (A is true), its negation $\neg A$ is entailed by the database (A is false), or neither is entailed by the database (A is unknown). Next I show formally that every propositional state of belief corresponds to a consistent database in propositional logic.

Definition 2.6.2 *A propositional state of belief* Φ *corresponds to database* Δ *in propositional logic precisely when for all* A*:* Φ *accepts* A *precisely when* Δ *entails* A.

The correspondence between propositional states of belief and consistent databases in propositional logic is stated by the following theorem:

Theorem 2.6.3 *For every consistent database* Δ *in propositional logic there is a propositional state of belief that corresponds to* Δ.

What can be represented by a database in propositional logic can be also represented by a propositional state of belief.

2.6.2 Probabilistic and improbabilistic

The following theorem shows that probabilistic states of belief are instances of abstract states of belief as defined in this chapter.

Theorem 2.6.4 *The pair* $\langle [0,1], + \rangle$ *is a partial support structure and its support order is* $\leq$.

A state of belief may attribute improbabilities, as opposed to probabilities, to sentences. The following theorem shows that these states are also instances of abstract states of belief.

Theorem 2.6.5 *The pair* $\langle [0,1], \lambda\ (a\ b)\ a+b-1 \rangle$ *is a partial support structure and its support order is* $\geq$.

2.6.3 Possibilistic and impossibilistic

A state of belief may attribute degrees of possibility to sentences. There is more than one choice of degrees of possibility; the following theorems identify two of these choices.

Theorem 2.6.6 *The pair* $\langle [0,1], \max \rangle$ *is a partial support structure and its support order is* $\leq$.

Theorem 2.6.7 *The pair* $\langle \{0, 1, \ldots, \infty\}, \max \rangle$ *is a partial support structure and its support order is* $\leq$. ∎

A state of belief may also attribute degrees of impossibility to sentences. Again, there is more than one choice of degrees of possibility, and the following theorems identify two of these choices.

Theorem 2.6.8 *The pair* $\langle [0,1], \min \rangle$ *is a partial support structure and its support order is* $\geq$.

Theorem 2.6.9 *The pair* $\langle \{0, 1, \ldots, \infty\}, \min \rangle$ *is a partial support structure and its support order is* $\geq$. ∎

A state of belief with respect to the structure $\langle\{0,1,\ldots,\infty\},\min\rangle$ is a variant of Spohn's natural conditional function [Spohn, 1987; Spohn, 1990] — the major difference being the existence of ∞, which Spohn does not allow. I therefore refer to states of belief with respect to this structure as *Spohnian states of belief.* These states place a sentence A into one of the following classes:

1. A is rejected: $\Phi(A)=\infty$.
2. A is accepted: $\Phi(\neg A)=\infty$.
3. A is rejected by default: $\Phi(A)\neq\infty>0$. $\Phi(A)$ is the default's strength.
4. A is accepted by default: $\Phi(\neg A)\neq\infty>0$. $\Phi(\neg A)$ is the default's strength.
5. A is undetermined: $\Phi(A)=0$ and $\Phi(\neg A)=0$.

This classification is more refined than the one given by propositional states of belief. Its value becomes more evident when I discuss belief change in Chapter 3 where sentences may change classes when new observations are recorded.

The notion of acceptance by default is central to the work on nonmonotonic reasoning. Indeed, I show next that each Spohnian state of belief corresponds to a database in nonmonotonic logic based on ranked preferential models (RPM) [Kraus *et al.*, 1990].

The nonmonotonic logic I am referring to is very much like classical propositional logic. We have a language $\mathcal{L}$, truth assignments W, and the usual meaning function $\mathcal{M}$ that maps each sentence to the truth assignments that satisfy it. The only additional construct that we have in RPM logic is a partitioning of the truth assignments W into a well ordered set of classes $\mu=W_0,W_1,\ldots$ The intuition is that truth assignments in W_i are preferred to those in W_j when $i<j$.

Given this preference over truth assignments, one can define the notions of preferred meaning and preferred entailment, both of which can be shown to subsume their classical counterparts.

Definition 2.6.10 *The preferred meaning of database Δ in RPM logic, written $\mathcal{M}_\mu(\Delta)$, is the set of the most preferred truth assignments in Δ's classical meaning.*

Definition 2.6.11 *Database Δ in RPM logic preferentially entails sentence A, written $\Delta \models_\mu A$, precisely when the preferred meaning of Δ is included in the classical meaning of A.*

The above entailment is nonmonotonic because, as we shall see in Chapter 3, it is possible that A is preferentially entailed by Δ but not by $\Delta \wedge B$.

I am now ready to state the correspondence result.

Definition 2.6.12 *A Spohnian state of belief Φ corresponds to database Δ in RPM logic precisely when the following holds: For all sentences A, Φ accepts A, or accepts A by default, precisely when Δ preferentially entails A.*

Theorem 2.6.13 *For every consistent database Δ in RPM logic there is a Spohnian state of belief that corresponds to Δ.*

According to Theorem 2.6.13, what can be represented by a database in RPM logic can also be represented by a Spohnian state of belief. The reader is referred elsewhere for a study of some other relations between Spohnian states of belief and databases in RPM logic [Hunter, 1991].

Chapter 3

Abstract Conditionalization

Now that we know what an abstract state of belief is, we may ask, How should an abstract state of belief change as a result of recording an observation? In this chapter, I address this question when the observation is a non–rejected sentence. This leads to the notion of abstract conditionalization, which generalizes probabilistic conditionalization and enjoys its most desirable properties.

3.1 Observing

The treatment in this chapter is based on the following assumption:

Assumption 3.1.1 *When a state of belief observes a non–rejected sentence it changes into another state of belief.*

We say that the new state of belief is the result of *conditionalizing* the initial state of belief on the observation. The basic goal of this chapter is to formalize this conditionalization process.

Definition 3.1.2 *Let Φ be a state of belief with respect to $\langle \mathcal{S}, \oplus \rangle_{\mathcal{L}}$. If A is a sentence in $\mathcal{L}$ that is not rejected by Φ, then a conditionalization of Φ on A, written Φ_A, is a state of belief with respect to $\langle \mathcal{S}, \oplus \rangle_{\mathcal{L}}$ that accepts A.*

According to this definition, the conditionalization of a state of belief on an observation is the state of belief resulting from accepting the observation. However, for a given state of belief and a given observation, there is usually more than one conditionalized state of belief. Some of these conditionalized states of belief result from plausible changes to the initial state of belief, but others do not. To exclude implausible changes to a state of belief, we need to impose more constrains on conditionalized states of belief. Below are some constraints that I have identified about states of belief as formalized by Definition 2.3.2 and conditionalized states of belief as formalized by Definition 3.1.2.

Axiom 6 *The support for A after observing $A \vee B$ is determined by the initial support for A and the initial support for $A \vee B$.*

Axiom 7 *Observing a non–rejected sentence retains all accepted sentences.*

Axiom 8 *Observing an accepted sentence leads to no change in a state of belief.*

Axiom 9 *When $A \vee B$ is equally supported by two states of belief, the observation of $A \vee B$ by each state does not introduce equality or order between the respective supports for A.*

Axiom 10 *The support for a sentence either increases or does not change after observing one of its logical consequences.*

Axiom 11 *If A's support after observing C equals its support after observing $B \wedge C$, then B's support after observing C equals its support after observing $A \wedge C$.*

A formal statement of Axioms 6–11 is given in Appendix D.

By accepting the above axioms, one is committed to (1) the existence of another function on degrees of support — let us denoted it by $\oslash$ — that has a number of properties given later, and (2) a definition of a conditionalized state of belief in terms of this function. The properties of $\oslash$ interact with the partial support structure $\langle \mathcal{S}, \oplus \rangle$ with respect to which a state of belief is defined. The resulting triple $\langle \mathcal{S}, \oplus, \oslash \rangle$ is called a *support structure* and is defined below.

	$\mathcal{S}$	$a \oplus b$	$a \oslash b$
Proposition	$\{0, 1\}$	$\max(a, b)$	$\min(a, b)$
Probability	$[0, 1]$	$a + b$	a/b
Improbability	$[0, 1]$	$a + b - 1$	$(a - b)/(1 - b)$
Possibility	$[0, 1]$	$\max(a, b)$	a/b
Impossibility	$\{0, 1, \ldots, \infty\}$	$\min(a, b)$	$a - b$

Table 3: Examples of support scaling.

Definition 3.1.3 *A support structure is a triple $\langle \mathcal{S}, \oplus, \oslash \rangle$, where*

1. *$\langle \mathcal{S}, \oplus \rangle$ is a partial support structure.*

2. *$\oslash$ is a partial function from $\mathcal{S} \times \mathcal{S}$ to $\mathcal{S}$ such that*

   (Y0) *$a \oslash b$ is defined if $a \preceq_{\oplus} b \neq \mathbf{0}$.*

3. *The function $\oslash$ satisfies the following properties:*

   (Y1) *$\mathbf{0} \oslash a = \mathbf{0}$ when $a \neq \mathbf{0}$.*

   (Y2) *$a \oslash \mathbf{1} = a$.*

   (Y3) *$a \oslash c = b \oslash c$ only if $a = b$ when $a, b \preceq_{\oplus} c \neq \mathbf{0}$.*

   (Y4) *$a \oslash b = c \oslash d$ only if $a \oslash c = b \oslash d$ when $a \preceq_{\oplus} b, c \preceq_{\oplus} d$ and $b, c, d \neq \mathbf{0}$.*

4. *The functions $\oslash$ and $\oplus$ satisfy the following properties:*

   (Y5) *$a \oslash b \succeq_{\oplus} a$ when $a \preceq_{\oplus} b \neq \mathbf{0}$.*

   (Y6a) *$a \oplus b \preceq_{\oplus} c$ only if $(a \oslash c) \oplus (b \oslash c)$ is defined when $c \neq \mathbf{0}$.*

   (Y7) *$(a \oplus b) \oslash c = (a \oslash c) \oplus (b \oslash c)$ when $a \oplus b \preceq_{\oplus} c \neq \mathbf{0}$.*

   (Y8) *$a \oslash c \preceq_{\oplus} b \oslash c$ only if $a \preceq_{\oplus} b$ when $a, b \preceq_{\oplus} c \neq \mathbf{0}$.*

The function $\oslash$ is called *support scaling*, and Table 3 provides some examples.

Let me iterate here that the existence of support scaling and its properties, as given by Definition 3.1.3, are consequences of Axioms 6–11.

In particular, the existence of support scaling is a consequence of Axiom 6 as suggested by the following theorem.

Theorem 3.1.4 *Axiom 6 holds precisely when*

$$\Phi_A(B) = \Phi(A \wedge B) \oslash \Phi(A) \textit{ when } \Phi(A) \neq \mathbf{0} \tag{1}$$

for some function $\oslash$.

The relation between the properties of support scaling and the axioms of belief change is given by a sequence of theorems.

Theorem 3.1.5 *Axiom 7 is equivalent to Property (Y1) given Axiom 6.*

Theorem 3.1.6 *Axiom 8 is equivalent to Property (Y2) given Axioms 6–7.*

Theorem 3.1.7 *Axiom 9 is equivalent to Properties (Y3) and (Y8) given Axioms 6–8.*

Theorem 3.1.8 *Axiom 10 is equivalent to Property (Y5) given Axioms 6–9.*

Theorem 3.1.9 *Axiom 11 is usefully equivalent to Property (Y4) given Axioms 6–10.*

Theorem 3.1.10 *Assumption 3.1.1, Axiom 6, and Axiom 7 imply Properties (Y0), (Y6a), and (Y7).*

According to Theorem 3.1.4, accepting Axiom 6commit us to the following definition of conditionalization.

Definition 3.1.11 *Let Φ be a state of belief with respect to $\langle \mathcal{S}, \oplus \rangle_{\mathcal{L}}$, and let A be a sentence in $\mathcal{L}$ that is not rejected by Φ. The conditionalization of Φ on A with respect to $\langle \mathcal{S}, \oplus, \oslash \rangle_{\mathcal{L}}$ is defined as follows:*

$$\Phi_A(B) = \Phi(A \wedge B) \oslash \Phi(A). \tag{2}$$

Support scaling satisfies a number of other properties that could be very useful in proving statements that describe how a state of belief changes. The following theorem lists some of these properties:

Theorem 3.1.12 *If $\langle \mathcal{S}, \oplus, \oslash \rangle$ is a support structure, then*

(Y10) $a \oslash a = \mathbf{1}$ *when* $a \neq \mathbf{0}$.

(Y11) $a \oslash (a \oslash b) = b$ *when* $a \preceq_{\oplus} b$ *and* $a \neq \mathbf{0}$.

(Y12) $a \oslash b = c$ *only if* $a \oslash c = b$ *when* $a \preceq_{\oplus} b \neq \mathbf{0}$, *and* $c \neq \mathbf{0}$.

(Y13) $a \preceq_{\oplus} b$ *only if* $a \oslash c \preceq_{\oplus} b \oslash c$ *when* $a \preceq_{\oplus} b \preceq_{\oplus} c \neq \mathbf{0}$ *and* $b \neq \mathbf{0}$.

(Y14) $(a \oslash c) \oslash (b \oslash c) = a \oslash b$ *when* $a \preceq_{\oplus} b \preceq_{\oplus} c \neq \mathbf{0}$, *and* $b \neq \mathbf{0}$.

(Y15) $(a \oslash b) \oslash (a \oslash c) = c \oslash b$ *when* $a \preceq_{\oplus} c \preceq_{\oplus} b \neq \mathbf{0}$, *and* $c \neq \mathbf{0}$.

For example, Property (Y14) can be used to prove the following result about conditionalization:

Theorem 3.1.13 *If $A \wedge B$ is not rejected by a state of belief Φ, then*

$$(\Phi_A)_B(C) = \Phi_{A \wedge B}(C).$$

This says that the order in which observations are recorded is not important.

3.2 Extracting states of belief

Chapter 4 is devoted to the problem of extracting a state of belief from a domain expert, which, as we shall see, is a constraint satisfaction problem. In particular, a domain expert gives us constraints about her state of belief from which we try to recover that state of belief. This process usually raises the following questions:

1. Is there a state of belief satisfying all the constraints? That is, are the constraints consistent?

2. Is there only one state of belief satisfying these constraints? That is, are the constraints complete?

The difficulty of these questions depends greatly on the support structure with respect to which states of belief are defined. Below are two classes of support structures that make these questions easier to answer.

Definition 3.2.1 *A support structure $\langle \mathcal{S}, \oplus, \oslash \rangle$ is* <u>*bijective*</u> *precisely when*

(Y9) *For all $b \neq \mathbf{0}$ and c, there is an a such that $a \preceq_{\oplus} b$ and $a \oslash b = c$.*

When extracting a state of belief from a domain expert, it is common to ask, What is your support for B and what is your support for A after you have observed B? If the expert's state of belief and its conditionalizations are with respect to a bijective support structure, then the answers will always be consistent.[1]

Let me now present another class of support structures that facilitates the extraction of a state of belief.

Definition 3.2.2 *A support structure $\langle \mathcal{S}, \oplus, \oslash \rangle$ is* <u>*distributive*</u> *precisely when*

(Y6b) *$(a \oslash c) \oplus (b \oslash c)$ is defined only if $a \oplus b \preceq_{\oplus} c$ when $a, b \preceq_{\oplus} c \neq \mathbf{0}$.*

The value of the above support structures is most evident given the following theorem.

Axiom 12 *Given a state of belief that is conditionalized on sentence C, there is nothing we can conclude about the support for C before it was observed.*

Theorem 3.2.3 *Axiom 12 is equivalent to Properties (Y9) and (Y6b).*

[1] I am assuming here that A and B are not logically disjoint.

3.3 Concrete conditionalizations

In this section, I provide some concrete support scalings, which give rise to conditionalization rules that can be used to (1) augment databases in classical logic, (2) conditionalize probabilistic states of belief, and (3) augment databases in RPM logic.

3.3.1 Augmenting databases in classical logic

As I mentioned in Chapter 2, the simplest partial support structure is $\langle\{0,1\},\max\rangle$. So let us look at a support scaling function for this structure.

Property (Y1) says that $0 \oslash 1$ should be 0, and Property (Y2) says that $1 \oslash 1$ should be 1. If we consider the definition of $a \oslash b$ only when $a \preceq_{\oplus} b \neq \mathbf{0}$, then there is only one scaling function with the previous properties.

Theorem 3.3.1 $\langle\{0,1\},\max,\min\rangle$ *is a bijective, distributive support structure.* ∎

Instantiating Equation 2 of Definition 3.1.11 with respect to the previous structure gives

$$\Phi_A(B) = \min(\Phi(A \wedge B), \Phi(A)).$$

And since we can only conditionalize on non–rejected sentences, it follows that $\Phi(A)$ must always be 1. We can then simplify the above equation to

$$\Phi_A(B) = \Phi(A \wedge B).$$

Conditionalizing a propositional state of belief is closely related to augmenting a database in propositional logic. In particular, if a propositional state of belief is viewed as a database in propositional logic, then conditionalizing a propositional state of belief on a sentence amounts to augmenting the corresponding database with that sentence. This is stated by the following theorem:

Theorem 3.3.2 *Let Δ be a consistent database in propositional logic, and let Φ be its corresponding propositional state of belief. If $\Delta \wedge A$ is also consistent, then its corresponding propositional state of belief is Φ_A.*

3.3.2 Bayes conditionalization

The following theorem provides a support scaling function for probabilistic states of belief.

Theorem 3.3.3 $\langle [0,1], +, / \rangle$ *is a bijective, distributive support structure.* ∎

Instantiating Equation 2 with respect to this structure gives us Bayes conditionalization:

$$\Phi_A(B) = \frac{\Phi(A \wedge B)}{\Phi(A)}.$$

The following theorem provides a support scaling function for improbabilistic states of belief.

Theorem 3.3.4 $\langle [0,1], \lambda\ (a\ b)\ a{+}b{-}1, \lambda\ (a\ b)\ (a{-}b)/(1{-}b) \rangle$ *is a bijective, distributive support structure.* ∎

Instantiating Equation 2 with respect to this structure gives

$$\Phi_A(B) = \frac{\Phi(A \wedge B) - \Phi(A)}{1 - \Phi(A)}.$$

3.3.3 Augmenting databases in RPM logic

The following theorem provides a support scaling function for impossibilistic states of belief.

Theorem 3.3.5 $\langle \{0, 1, \ldots, \infty\}, \min, - \rangle$ *is a bijective, distributive support structure.* ∎

Instantiating Equation 2 with respect to the previous structure gives

$$\Phi_A(B) = \Phi(A \wedge B) - \Phi(A).$$

Moreover, if a Spohnian state of belief is viewed as a database in RPM logic, then conditionalizing a Spohnian state of belief on a sentence amounts to augmenting the corresponding database with that sentence. This is stated by the following theorem:

Theorem 3.3.6 *Let Δ be a consistent database in RPM logic, and let Φ be its corresponding Spohnian state of belief. If the database $\Delta \wedge A$ is also consistent, then its corresponding Spohnian state of belief is Φ_A.*

The above theorem is important for the following reason: Preferential entailment is nonmonotonic in the sense that a sentence may be preferentially entailed by a database Δ but may not be preferentially entailed by an augmentation of Δ, say $\Delta \wedge A$.[2] This property causes an implementation problem in AI applications.

In particular, suppose that we have modeled the state of belief of some agent by a database Δ. A sentence that is preferentially entailed by Δ is considered a *default belief* of that agent. Moreover, some default beliefs must be retracted when new sentences are added to the agent's database because they may cease to be preferentially entailed by the augmented database. Note, however, that detecting default beliefs that need to be retracted has proven to be a real problem from an implementational viewpoint.

In the light of Definition 2.6.12, Theorem 3.3.6 is an important step towards implementing this detection. Specifically, if Φ is the Spohnian state of belief corresponding to database Δ, then a sentence B is *not* preferentially entailed by an augmented database $\Delta \wedge A$ unless $\Phi_A(\neg B) > 0$.

Chapter 5 is concerned with the computation of conditional supports in abstract states of belief, which, together with Definition 2.6.12 and Theorem 3.3.6, provides a complete implementation of detecting beliefs that need to be retracted.

———

The following theorem provides a support scaling function for possibilistic states of belief.

Theorem 3.3.7 $\langle [0,1], \max, / \rangle$ *is a distributive, bijective support structure.* ∎

Instantiating Equation 2 with respect to this structure gives us the following conditionalization rule:

$$\Phi_A(B) = \frac{\Phi(A \wedge B)}{\Phi(A)}.$$

[2]The technical reason for this is that preferential entailment is defined in terms of preferred meaning, which does not necessarily get smaller as the database gets bigger.

3.4 Conditional and unconditional supports

For a theory of states of belief to be useful in building artificial agents, the specification of a state of belief must be made intuitive enough so that a domain expert can naturally map a state of belief onto an artificial agent. This section discusses a function on degrees of support that helps in achieving this goal.

A basic observation about human reasoning, claimed by Bayesian philosophers, is that it is more intuitive for people to specify their support for a sentence A (e.g., "The grass is wet") conditioned on accepting a relevant sentence B (e.g., "It rained") than to specify their unconditional support for A. It is therefore natural for domain experts to specify their states of belief by providing conditional supports. This is indeed the approach taken by most probabilistic representations where a domain expert provides statements of the form "$P(A|B) = p$," which reads as "If I accept B, then my probabilistic support for A becomes p."

One should note, however, that conditional supports are most useful when they can tell us something about unconditional supports. For example, conditional probabilities can be easily mapped into unconditional probabilities: $P(A \wedge B) = P(A|B)P(B)$. It is then important to ask whether the previous equality is an instance of a more general one that holds for abstract states of belief. This question is answered positively by the following theorem, which states that for every support structure there is a function on degrees of support that plays the same role as that played by numeric multiplication in probability calculus:

Theorem 3.4.1 *Let $\langle \mathcal{S}, \oplus, \oslash \rangle$ be a support structure. There is a partial function f from $\mathcal{S} \times \mathcal{S}$ to $\mathcal{S}$ such that*

$$a \oslash b = c \text{ and } a \preceq_{\oplus} b \neq \mathbf{0} \text{ only if } f(c, b) = a. \blacksquare$$

This theorem says that support scaling has an inverse, but only over a subset of its domain. This subset consists of the pairs (a, b) such that $a \preceq_{\oplus} b \neq \mathbf{0}$. But why is the existence of an inverse restricted to this subset? The answer lies in Property (Y3), which says indirectly that support scaling has an inverse:

$$a \oslash c = b \oslash c \text{ only if } a = b.$$

	$\mathcal{S}$	$a \otimes b$
Proposition	$\{0,1\}$	$\min(a,b)$
Probability	$[0,1]$	$a \times b$
Improbability	$[0,1]$	$a+b-ab$
Possibility	$[0,1]$	$a \times b$
Impossibility	$\{0,1,\ldots,\infty\}$	$a+b$

Table 4: Examples of support unscaling.

Note, however, that this property is conditional on $a, b \preceq_{\oplus} c \neq \mathbf{0}$. Therefore, we can prove the existence of an inverse only when this condition is met. In fact, the definition of a support structure requires support scaling to be defined only if this condition is met. This is to be expected because in the context of conditionalization, which has led to support scaling, it is meaningful to scale support a with respect to support b only when $a \preceq_{\oplus} b \neq \mathbf{0}$.

Definition 3.4.2 *Support unscaling is a partial function $\otimes$ from $\mathcal{S} \times \mathcal{S}$ to $\mathcal{S}$ such that*

$$c \otimes b \stackrel{def}{=} a \text{ precisely when } a \oslash b = c \text{ and } a \preceq_{\oplus} b \neq \mathbf{0}.$$

Table 4 lists some examples of support unscaling. Support unscaling is used in mapping conditional supports onto unconditional supports.

Corollary 3.4.3 *If the state of belief Φ does not reject A, then*

$$\Phi(A \wedge B) = \Phi_A(B) \otimes \Phi(A). \blacksquare \tag{3}$$

We have looked before at two classes of support structures that facilitate the extraction of states of belief. Each of these classes suggests an additional property that support scaling should satisfy. Given the relation between support scaling and unscaling, these two classes of support structures can be characterized by the properties that support unscaling needs to satisfy. The following two theorems identify these properties:

Theorem 3.4.4 *Let $\langle \mathcal{S}, \oplus, \oslash \rangle$ be a bijective support structure and let $\otimes$ be its support unscaling function. Then $a \otimes b$ is defined precisely when $b \neq \mathbf{0}$.*

This theorem says that if the support structure is bijective, then the following two statements are always consistent:

1. My support for B is $b \neq \mathbf{0}$.

2. My support for A given B is a.

In particular, the state of belief Φ such that $\Phi(A \wedge B) = a \otimes b$ and $\Phi(B) = b$ satisfies these statements. In general, however, the two statements are consistent only if $a \otimes b$ is defined. Therefore, the domain of support unscaling is crucial in the process of belief extraction. We shall see an example of this in Chapter 7.

Theorem 3.4.5 *Let $\langle \mathcal{S}, \oplus, \oslash \rangle$ be a distributive support structure and let $\otimes$ be its support unscaling function. Then*

$$(a \otimes c) \oplus (b \otimes c) \preceq_{\oplus} c \quad \textit{when } a \oplus b,\ a \otimes c \textit{ and } b \otimes c \textit{ are defined.}$$

Theorem 3.4.6 *The following properties hold for support scaling and unscaling:*

(Z1) $(a \oslash b) \otimes b = a$.

(Z2) $(a \otimes b) \oslash b = a$.

(Z3) $\mathbf{0} \otimes a = \mathbf{0}$.

(Z4) $a \otimes \mathbf{1} = a$.

(Z5) $a \otimes b \preceq_\oplus b$.

(Z6) $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ *when* $(a \otimes c) \oplus (b \otimes c) \preceq_\oplus c \neq \mathbf{0}$.

(Z7) $a \preceq_\oplus b$ *only if* $a \otimes c \preceq_\oplus b \otimes c$.

(Z8a) $a \otimes b = b \otimes a$.

(Z8b) $a \otimes b$ *is defined precisely when* $b \otimes a$ *is defined when* $a \neq \mathbf{0}$ *and* $b \neq \mathbf{0}$.

(Y16) $(a \oslash b) \oslash c = a \oslash (b \otimes c) = (a \oslash c) \oslash b$ *when* $a \preceq_\oplus b \otimes c \neq \mathbf{0}$.
(This is a property of support scaling, but its proof is based on the above results).

(Z9a) $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.

(Z9b) $(a \otimes b) \otimes c$ *is defined precisely when* $a \otimes (b \otimes c)$ *is defined.*

(Z10) $a \otimes b = a$ *precisely when* $a = \mathbf{0}$ *or* $b = \mathbf{1}$.

(Z11) $(a \otimes b) \oslash c = a \otimes (b \oslash c)$.

(Z12) $(a \otimes c) \oslash (b \otimes c) = a \oslash b$ *when* $a \otimes c \preceq_\oplus b \otimes c \neq \mathbf{0}$.

3.4.1 Concrete support unscalings

In this section, I will consider a number of support unscaling functions. But let me first stress that unscaling functions are in general not total functions. I will show the technical reason for this and then provide the intuition behind it.

By definition, $a \otimes \mathbf{0}$ is not defined for any a. And unless the function $\oslash b : \mathcal{S} \to \mathcal{S}$ is bijective, there will be other places where support unscaling $\otimes$ is not defined. Recall that the function $\oslash b$ is bijective precisely when for any support c, there is another support a such that $a \preceq_{\oplus} b$ and $a \oslash b = c$. Recall also that the formalization of Section 3.1 does not require the function $\oslash b$ to be bijective.[3]

Intuitively, it is very helpful to think of support unscaling in the context of Equation 3 of Definition 3.4.3, which can be viewed as defining the following inference rule:

> If ζ is some information about a state of belief Φ ($\Phi(A) = a$), and η is some information about a conditionalization of Φ ($\Phi_A(B) = b$), then we can infer more information about the state of belief Φ ($\Phi(A \wedge B) = b \otimes a$).

As it turns out, however, not all pieces of information (ζ, η) are consistent. That is, it is possible that ζ is true about a state of belief, but η is not true about any of its conditionalizations. These are precisely the places where support unscaling is not defined. In other words, cases where support unscaling is undefined correspond to illegitimate scenarios of belief change.

The following theorem provides a class of partial support structures that is known to induce states of belief with non–trivial illegitimate scenarios of belief change.

Theorem 3.4.7 *If the set of supports $\mathcal{S}$ is finite and has more than two elements, then there are supports a and $b \neq \mathbf{0}$ where $a \otimes b$ is not defined.*

In the remainder of this section, I provide the support unscaling functions of some support structures that I have introduced earlier.

Theorem 3.4.8 *Support unscaling of* $\langle \{0, 1\}, \max, \min \rangle$ *is* $\max$. ∎

Theorem 3.4.9 *Support unscaling of* $\langle [0, 1], +, / \rangle$ *is* $\times$. ∎

Theorem 3.4.10 *Support unscaling of* $\langle [0, 1], \lambda\,(a\ b)\ a + b - 1, \lambda\,(a\ b)\ (a - b)/(1 - b) \rangle$ *is* $\lambda\,(a\ b)\ a + b - ab$. ∎

[3] Axiom 9 requires the function $\oslash b$ be an injection only.

Theorem 3.4.11 *Support unscaling of* $\langle \{0, 1, \ldots, \infty\}, \min, - \rangle$ *is* $+$. ∎

Theorem 3.4.12 *Support unscaling of* $\langle [0, 1], \max, / \rangle$ *is* $\times$. ∎

3.4.2 Support scaling versus support unscaling

The way I have presented belief change seems to indicate that support scaling is the primitive concept, and support unscaling is just a convenient side effect of that concept. In fact, the existence of support unscaling is only a consequence of Axiom 9 discussed in Section 3.1. That is, had we not accepted Axiom 9, we would not have had a support unscaling function in general. But we would still have had a legitimate account of belief change that agrees partially with our intuition.

Surprisingly enough, it seems to be more customary in the literature to take a variant of support unscaling as the primitive concept in formalizing belief change than to choose support scaling. For example, in his work on Probabilistic Logic [Aleliunas, 1988], Aleliunas suggests a primitive operator called *product*, which is closely related to support unscaling. And in their work on valuation–based systems [Shenoy, 1989; Shenoy and Shafer, 1990], Shenoy and Shafer suggest a primitive operator called *combination* that is also related to support unscaling.

The choice between support scaling and unscaling as a primitive in formalizing belief change is a choice between one of the following two ways of asking questions:

1. Given an initial state of belief and an observation, what can we say about the state of belief that results from accepting this observation?

2. Given a new state of belief and the observation that has led to it, what can we say about the initial state of belief?

That is, the choice between support scaling and unscaling is a choice between predicting belief changes and explaining them. I find the first more intuitive as a tool for formalizing belief change. I believe it is more natural for people to predict how their beliefs would change as they obtain more information than to explain why their beliefs could have changed in a particular way.

3.5 Patterns of plausible reasoning

The ultimate objective of much work in AI—most notably on nonmonotonic logics—is to capture patterns of plausible reasoning in nonnumerical terms. George Polya (1887–1985) was one of the first mathematicians to attempt a formal characterization of qualitative human reasoning. Polya identified five main patterns of plausible reasoning and demonstrated that they can be formalized using probability theory [Polya, 1954, Chapter XV]. Pearl highlighted these patterns in his recent book [Pearl, 1988] and took them—along with other patterns such as nonmonotonicity, abduction, explaining–away, and the law of the hypothetical middle [Pearl, 1988, Page 19]—as evidence for the indispensability of probability theory in formalizing plausible reasoning. In his own words:

> We take for granted that probability calculus is unique in the way it handles context–dependent information and that no competing calculus exists that closely covers so many aspects of plausible reasoning [Pearl, 1988, Page 20].

I show in this section that four of Polya's patterns of plausible reasoning hold with respect to abstract states of belief and their conditionalizations. But first, I need to formally define certain terms that Polya used in stating his patterns:

- To "verify," or "prove," a proposition is to *accept* it.
- To "explode" a proposition is to *reject* it.
- The "credibility of," or "confidence in," a proposition is its *degree of support.*

Four of Polya's patterns of plausible reasoning follow. These patterns have the form, If something holds about a state of belief, then something else holds about its conditionalizations. The statement of each pattern is followed by a theorem that proves the pattern with respect to abstract states of belief and their conditionalizations.

Polya's first pattern:

Pattern 1 (Examining a consequence) *The verification of a consequence renders a conjecture more credible. [Polya, 1954, Page 120]*

Theorem 3.5.1 *If $A \supset B$ is accepted and B is not rejected by a state of belief Φ, then $\Phi_B(A) \succ_{\oplus} \Phi(A)$ unless $\Phi(A) = \mathbf{0}$ or $\Phi(B) = \mathbf{1}$.*

Polya's second pattern:

Pattern 2 (Examining a possible ground) *Our confidence in a conjecture can only diminish when a possible ground for the conjecture has been exploded. [Polya, 1954, Page 123]*

Theorem 3.5.2 *If $A \supset B$ is accepted and $\neg A$ is not rejected by a state of belief Φ, then $\Phi_{\neg A}(\neg B) \succ_{\oplus} \Phi(\neg B)$ unless $\Phi(\neg B) = \mathbf{0}$ or $\Phi(\neg A) = \mathbf{1}$.*

Polya's third pattern:

Pattern 3 (Examining a conflicting conjecture) *Our confidence in a conjecture can only increase when an incompatible rival conjecture has been exploded. [Polya, 1954, Page 124]*

Theorem 3.5.3 *If $A \wedge B$ is rejected and $\neg A$ is not rejected by a state of belief Φ, then $\Phi_{\neg A}(B) \succ_{\oplus} \Phi(B)$ unless $\Phi(B) = \mathbf{0}$ or $\Phi(\neg A) = \mathbf{1}$.*

Polya's fourth pattern:

Pattern 4 (Examining several consequences in succession) *The verification of a new consequence enhances our confidence in the conjecture, unless the new consequence is implied by formerly verified consequences [Polya, 1954, Page 125].*

The condition, "the new consequence is implied by formerly verified consequences," means that the conditional probability of the new consequence given the formerly verified consequences is one [Polya, 1954]. With respect to abstract states of belief and their conditionalizations, this condition becomes, "the new consequence is maximally supported given formerly verified consequences."

Theorem 3.5.4 *If $A \supset C_1, \ldots, A \supset C_n$ is accepted and $C_1 \wedge \ldots \wedge C_n$ is not rejected by a state of belief Φ, then $\Phi_{C_1 \wedge \ldots \wedge C_{n-1}}(A) \prec_{\oplus} \Phi_{C_1 \wedge \ldots \wedge C_n}(A)$ unless $\Phi_{C_1 \wedge \ldots \wedge C_{n-1}}(C_n) = \mathbf{1}$.*

Chapter 4

Independence and Belief Extraction

To specify an abstract state of belief over n primitive propositions, one needs to specify 2^n degrees of support. This is unrealistic and counterintuitive: Specifying a state of belief over 100 propositions should not require 1267650600228229401496703205376 degrees of support! In this chapter, I present a solution to this problem that is a direct generalization of a similar solution for specifying probabilistic states of belief.

4.1 The intuition

This chapter is based on a key observation in the probabilistic literature [Pearl, 1988].

Observation 4.1.1 *Assertions of the form, Observing A does not change the belief in B, reduces the exponential number of supports required to specify a state of belief.*

In particular, when an expert says that observing A does not change her belief in B, this statement can be viewed as a constraint on the expert's state of belief Φ:

$$\Phi_A(B) = \Phi(B) \text{ and } \Phi_A(\neg B) = \Phi(\neg B).$$

The more constraints of the above form, the lower the number of supports required to specify a state of belief. Indeed, it is possible to reduce the exponential number of required supports to a linear number.

For example, an expert might tell us that for every proposition i, observing propositions $1, \ldots, i-1$ does not change her belief in i. This information, together with the expert's belief in proposition i, specifies her state of belief completely. To see why, note that a state of belief over propositions $1, \ldots, n$ is determined by the support for each sentence of the form $[\neg]1 \wedge \ldots \wedge \ldots [\neg]n$, where $[\neg]$ means that the negation sign may or may not appear. The support for sentences of this form are computed as follows:

$$\begin{aligned}
&\Phi([\neg]1 \wedge \ldots \wedge \ldots [\neg]n) \\
&\quad = \Phi_{[\neg]1\wedge\ldots\wedge\ldots[\neg]n-1}([\neg]n) \otimes \Phi_{[\neg]1\wedge\ldots\wedge\ldots[\neg]n-2}([\neg]n{-}1) \otimes \ldots \otimes \Phi([\neg]1) \\
&\quad = \Phi([\neg]n) \otimes \Phi([\neg]n{-}1) \otimes \ldots \otimes \Phi([\neg]1).
\end{aligned}$$

That is, using Observation 4.1.1, the number of supports required to specify a state of belief over n propositions could be reduced from 2^n to $2n$, an exponential reduction.

If observing A does not change the belief in B, we say that A is *independent* from B. Observation 4.1.1 can then be rephrased as follows: Independence assertions reduce the exponential number of supports required to specify a state of belief.

What makes Observation 4.1.1 so powerful in extracting states of belief is that it sets the basis for a claim and a result to be discussed in Section 4.3. The claim concerns inducing independence assertions from the causal structure underlying a domain of interest. The result concerns the number of degrees of support required to complete the specification of a state of belief that is partially specified by a causal structure. The following section explores the notion of independence in more detail, which paves the way for Section 4.3.

4.2 Independence among propositions

Section 4.1 talked about the independence of one sentence from another. Another meaningful and very useful notion, as we shall see, is the independence of one set of propositions from another. The intuition here is that a set of propositions J is independent from another set I if observing any state of propositions J does not change the support for any state of propositions I. The notion of a state is defined more formally as follows.

Definition 4.2.1 *A* <u>*state*</u> *of a primitive proposition i, written $\underline{i}$, is either i or $\neg i$. A state of a set of primitive propositions I, written $\underline{I}$, is $\bigwedge_{i \in I} \underline{i}$.*

For example, D and $\neg D$ are the states of proposition D, while $D \wedge E$, $D \wedge \neg E$, $\neg D \wedge E$, and $\neg D \wedge \neg E$ are the states of propositions $\{D, E\}$.

Definition 4.2.2 *A state of belief Φ finds I independent from J, written $IN_{\Phi}(I, J)$, precisely when $\Phi_{\underline{J}}(\underline{I}) = \Phi(\underline{I})$ for all $\underline{I}$ and every $\underline{J}$ that is not rejected by Φ.*

Independence is nonmonotonic. It is possible for a state of belief to find one set of propositions independent from another, but then find it dependent after recording an observation. It is also possible for dependent propositions to become independent when more observations are recorded. Below are some examples.

1. *i is dependent on j initially, but it becomes independent from j once k is observed.* In Figure 1, the outputs of the AND gates are dependent on each other, but they become independent once input k is observed. This is an example of conditional independence.

2. *i is independent from j initially, but it becomes dependent on j once k is observed.* In Figure 1, the inputs to the XOR gate are independent from each other, but they become dependent once output k is observed. This is an example of conditional dependence.

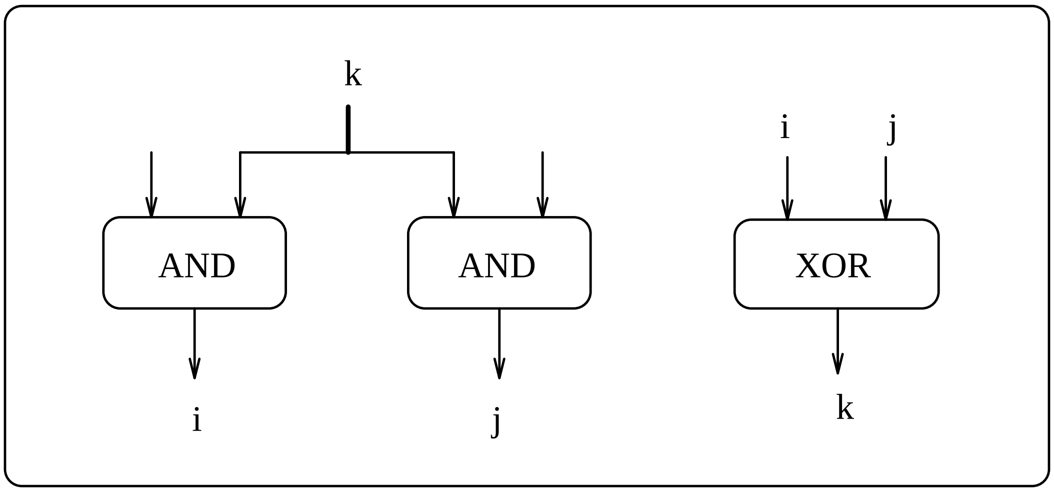

Figure 1: On the left, i is dependent on j initially, but it becomes independent from j once k is observed. On the right, i is independent from j initially, but it becomes dependent on j once k is observed.

In general, a set of propositions I is conditionally independent from another set J given a third set of propositions K if I is unconditionally independent from J once the state of propositions K is observed. Therefore, conditional independence could be defined in terms of unconditional independence as given below.

Definition 4.2.3 *A state of belief Φ finds I independent from J given K, written $IN_\Phi(I, K, J)$, precisely when $IN_{\Phi_{\underline{K}}}(I, J)$ for every $\underline{K}$ that is not rejected by Φ.*

It is more customary to define conditional independence without appealing to unconditional independence. Although I find Definition 4.2.3 more intuitive, I provide the following result for the sake of completeness.

Corollary 4.2.4 *A state of belief Φ finds I independent from J given K precisely when*

$$\Phi_{\underline{J}\wedge\underline{K}}(\underline{I}) = \Phi_{\underline{K}}(\underline{I})$$

for all $\underline{I}$ and every $\underline{J} \wedge \underline{K}$ that is not rejected by Φ. ∎

4.3 Extracting a state of belief

Observation 4.1.1 is powerful in extracting states of belief because of a claim that independence assertions are induced by the "causal structure" underlying the domain of interest. Let me first define the notion of a causal structure and then state the claim.

Definition 4.3.1 *A causal structure over primitive propositions N is a binary relation $DC \subseteq N \times N$, where i DC j reads as i directly causes j. We say that i causes j (or, j is an effect of i) precisely when*

1. *i directly causes j, or*

2. *i causes k and k directly causes j.*

In a causal structure, a proposition cannot cause itself.

It is common to represent a causal structure using a directed acyclic graph. For example, the causal structure

$$\begin{array}{rcl} \mathit{Rain} & \text{directly causes} & \mathit{Slippery\_Road} \\ \mathit{Rain} & \text{directly causes} & \mathit{Wet\_Grass} \\ \mathit{Sprinkler\_On} & \text{directly causes} & \mathit{Wet\_Grass} \end{array}$$

is depicted graphically in Figure 2. We simply create a node for each primitive proposition, and then add an arc from node i to node j precisely when i directly causes j. By definition of a causal structure, the resulting directed graph is guaranteed to be acyclic.

Below is a key claim that underlies the way independence assertions are induced in many practical applications.

Claim 4.3.2 *The expert who identifies a causal structure finds a proposition independent from its non–effects given its direct causes.*

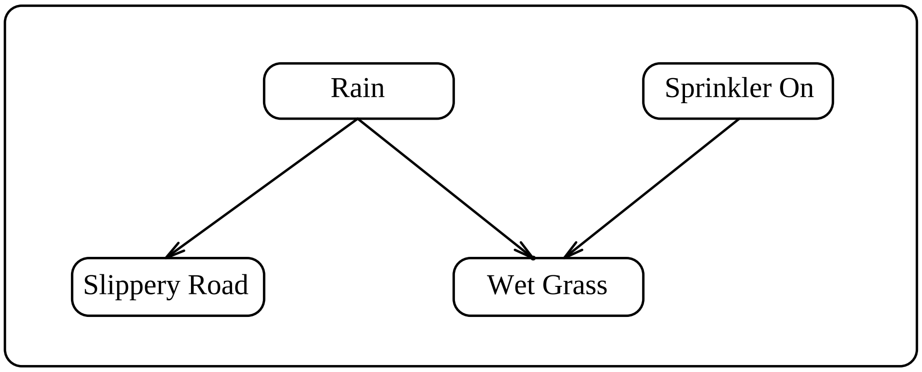

Figure 2: A graphical representation of a causal structure.

The intuition here is that information about the non–effects of a proposition is relevant to the proposition only because it conveys information about its direct causes. Therefore, the information becomes useless once the state of direct causes is known. For example, in Figure 2, information about *Slippery_Road* is relevant to *Wet_Grass* only because it conveys information about *Rain*. However, once the state of *Rain* is known, information about *Slippery_Road* is no longer relevant to *Wet_Grass*.

In practical applications, it is common to ask a domain expert to identify a causal structure, and then use Claim 4.3.2 to induce constraints on the state of belief held by the expert. The soundness of this practice hinges on the correctness of Claim 4.3.2, which must be relative to some formal definition of causation. However, due to the lack of a universally accepted definition of causation, we are in no position to provide a satisfactory proof of Claim 4.3.2. Nevertheless, the role played by Claim 4.3.2 in practical applications is not minor.

Given the discussion of Section 4.1, and Claim 4.3.2, an expert who identifies a causal structure is in fact supplying us with constraints on her state of belief. We have seen in Section 4.1 how these constraints reduce the exponential number of supports required to specify a state of belief. This section focuses on the supports needed to complete the specification of a state of belief.

One completes the specification of a state of belief by *quantifying* a causal structure. Quantification is the process in which an expert assesses her support for each primitive proposition given a particular state of its direct causes. For example, the node *Slippery_Road* in Figure 2 is quantified by providing four conditional supports:

1. The conditional support for *Slippery_Road* given *Rain*.

2. The conditional support for *Slippery_Road* given $\neg$*Rain*.

3. The conditional support for $\neg$*Slippery_Road* given *Rain*.

4. The conditional support for $\neg$*Slippery_Road* given $\neg$*Rain*.

These supports constitute the quantification of node *Slippery_Road*. The quantification of node *Wet_Grass*, however, consists of eight conditional supports because there are four possible states of the direct causes of *Wet_Grass*.

A quantified causal structure is called a *causal network* and is usually depicted graphically using two components:

- *A direct acyclic graph,* which specifies a causal structure.

- *A set of tables,* which contain node quantifications. Each node has a table associated with it. The table has two columns corresponding to the states of the node. It has one row for each state of the node's direct causes. Each entry in the table is a support for some state of the node conditioned on a particular state of its direct causes. The entries of each row must sum up to the full support.

Figure 3 depicts a causal network with respect to the support structure $\langle [0,1], +, / \rangle$. Figure 4 depicts another causal network but with respect to the support structure $\langle \{0, 1, \ldots, \infty\}, \min, - \rangle$. The causal networks in Figures 3 and 4 share the same causal structure, but they differ in the way they are quantified. We shall look at more methods of quantification in the following chapters.

The formal definition of a causal network is given below.

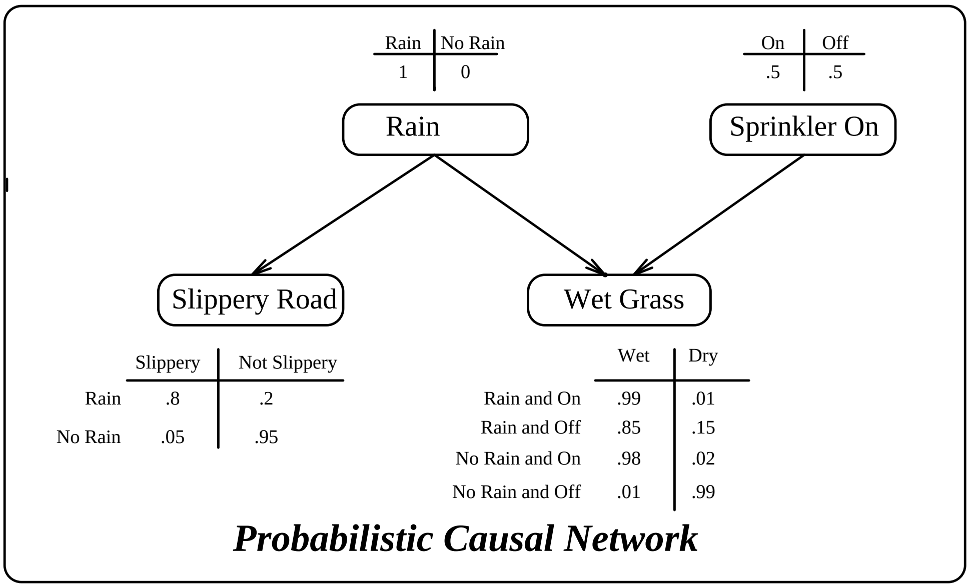

Figure 3: A probabilistic causal network. The top left entry of the bottom left table reads, The probability of *Rain* given *Slippery_Road* is .8.

Notation $i\diamond$ denotes the parents of node i. $\mathcal{L}(N)$ denotes a propositional language constructed from primitive propositions N.

Definition 4.3.3 *A <u>causal network</u> with respect to a support structure* $\langle \mathcal{S}, \oplus, \oslash \rangle$ *is a triple* $\langle N, \mathcal{G}, \mathcal{CS} \rangle$, *where*

- N *is a set of primitive propositions.*
- $\mathcal{G}$ *is a directed acyclic graph over* N.
- $\mathcal{CS}$ *is a partial function* $\mathcal{L}(N) \times \mathcal{L}(N) \to \mathcal{S}$ *such that*
  - $\mathcal{CS}_{\underline{i\diamond}}(\underline{i})$ *is defined, and*
  - $\mathcal{CS}_{\underline{i\diamond}}(i) \oplus \mathcal{CS}_{\underline{i\diamond}}(\neg i) = \mathbf{1}$,

  for every node i.

$\mathcal{CS}$ *is called a conditional support function.*

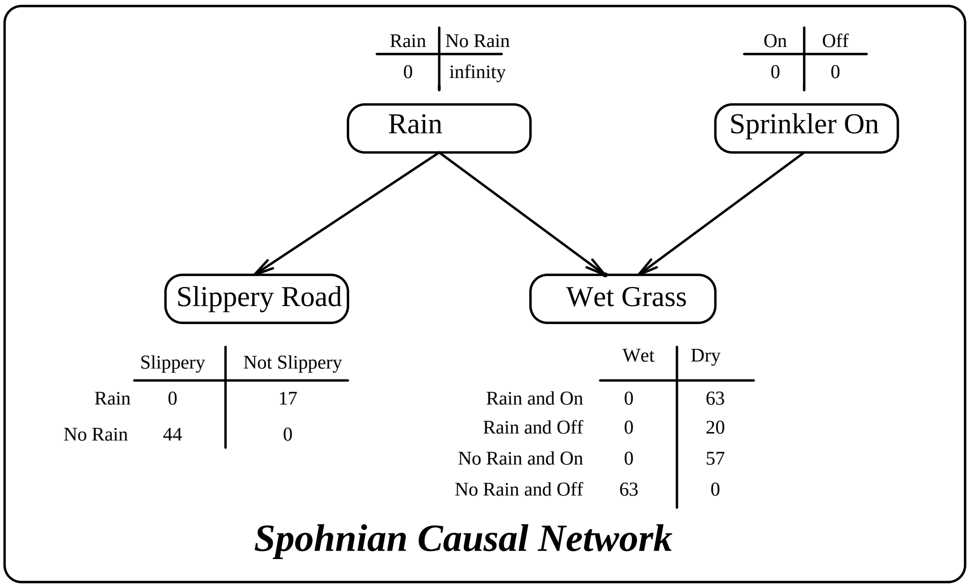

Figure 4: A Spohnian causal network. The top left entry of the bottom left table reads, The impossibility of *Rain* given *Slippery_Road* is 0.

A causal network consists of two sets of constraints on a state of belief. The first set is about independence assertions, while the second is about conditional supports. Since the goal of constructing a causal network is to specify a state of belief, it is most important to know whether a given state of belief satisfies the constraints imposed by a causal network.

Notation $i\triangleleft$ denotes the non–descendents of node i.

Definition 4.3.4 *A state of belief Φ over propositions N satisfies a causal network $\langle N, \mathcal{G}, \mathcal{CS}\rangle$ precisely when*

$$IN_{\Phi}(i, i\diamond, i\triangleleft \setminus i\diamond) \text{ and } \Phi_{\underline{i\diamond}}(\underline{i}) = \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \text{ for every node } i.$$

If we view a causal network as a set of constraints, then the definition of a causal network does not always guarantee the consistency of these constraints. Although we can find a state of belief that satisfies the independences asserted by a causal structure, it is not always possible to find a state of belief that satisfies the conditional supports quantifying the causal structure. However, causal networks with respect to a distributive and bijective support structure are always consistent.

Theorem 4.3.5 *An abstract causal network that is induced with respect to a distributive and bijective support structure is satisfied by exactly one state of belief.*

Probabilistic and Sphonian causal networks are induced with respect to distributive and bijective support structures.

4.4 More on independence

Let me summarize what I have done so far.

I started with the problem of having to provide too many degrees of support in order to specify a state of belief. I then observed that the number of supports can be reduced if one utilizes independence assertions. I followed this observation by the claim that independence assertions are induced by the causal structure of the domain of interest. I concluded, therefore, that by appealing to causal structures, one can reduce the number of degrees of support required to specify a state of belief.

In addition to this representational role, independence assertions play a major computational role, as we shall see in Chapter 5. But to utilize independence assertions computationally, we need to know more about their properties. Some of these properties are discussed in Section 4.4.1. We also need to retrieve the independences asserted by a causal network without having to reconstruct the state of belief satisfying the network. A retrieval method, which examines only the topology of a causal network, is discussed in Section 4.4.2.

4.4.1 Properties of independence

In this section, I discuss five properties of independence. The first four are known as the *graphoid* axioms [Pearl, 1988] and hold with respect to any support structure. The fifth property, however, is shown to hold with respect to a restricted class of support structures.

Independence 1 (Symmetry) *If I is independent from J, then J is also independent from I.*

Theorem 4.4.1 $IN_{\Phi}(I,K,J)$ *precisely when* $IN_{\Phi}(J,K,I)$.

By examining the proof of Theorem 4.4.1, we see that Symmetry of independence hinges on Property (Y4) of support scaling.

Independence 2 (Decomposition) *If I is independent from J, then it is also independent from any subset of J.*

Theorem 4.4.2 *If $IN_\Phi(I, K, J \cup L)$, then $IN_\Phi(I, K, J)$.*

To see why this property holds, recall that if I is independent from $J \cup L$, then $J \cup L$ is also independent from I. Hence, observing any sentence $\underline{I}$ does not change the support for any sentence $\underline{J} \wedge \underline{L}$. But this also means that observing any sentence $\underline{I}$ does not change the support for any sentence $\underline{L}$. This follows from

$$\Phi(\underline{L}) = \bigoplus_{\underline{J}} \Phi(\underline{J} \wedge \underline{L}).$$

Independence 3 (Weak Union) *If I is independent from $J \cup L$, then I is independent from J given L.*

Theorem 4.4.3 *If $IN_\Phi(I, K, J \cup L)$, then $IN_\Phi(I, K \cup J, L)$.*

If I is independent from $J \cup L$, then Decomposition and Symmetry tell us that J and L are also independent from I. Therefore, in the context of these properties, Weak Union says: If I is independent from J, then it remains independent given an independent L.

To see why Weak Union holds, suppose that observing some $\underline{L}$ makes I dependent on J. That is, observing $\underline{L}$ and then observing some $\underline{J}$ changes the support for some $\underline{I}$. Therefore, observing $\underline{L} \wedge \underline{J}$ changes the support for $\underline{I}$. But this is a contradiction because I is independent from $J \cup L$.

The above proof of Weak Union assumes that consecutive observations have the same effect on a state of belief as simultaneous observations. This assumption is satisfied by Definition 3.1.11: The states $\Phi_{A \wedge B}$ and $(\Phi_A)_B$ are equivalent if either is defined. This result is stated by Theorem 3.1.13.

Independence 4 (Contraction) *If I is independent from J, and if I is independent from L given J, then I is also independent from $J \cup L$.*

Theorem 4.4.4 *If $IN_\Phi(I, K, J)$ and $IN_\Phi(I, K \cup J, L)$, then $IN_\Phi(I, K, J \cup L)$.*

Contraction holds because simultaneous observations have the same effect on a state of belief as consecutive observations. That is, observing $\underline{J}$ followed by observing

$\underline{L}$ leads to the same state of belief as observing $\underline{J} \wedge \underline{L}$. The premise of Theorem 4.4.4 says that observing $\underline{J}$ does not change the support for $\underline{I}$ and neither does observing $\underline{L}$ consequently. It follows that observing $\underline{J} \wedge \underline{L}$ does not change the support for $\underline{I}$, which is the consequence of Theorem 4.4.4.

Together, Decomposition, Weak Union, and Contraction, say that

$$IN_\Phi(I, K, J) \text{ and } IN_\Phi(I, K \cup J, L) \text{ precisely when } IN_\Phi(I, K, J \cup L).$$

That is, I is independent from $J \cup L$ precisely when I is independent from J and I is independent from L given J.

Independence 5 (Intersection) *If I is independent from J given L, and if I is independent from L given J, then I is also independent from $J \cup L$.*

Theorem 4.4.5 *If the state of belief Φ is with respect to a distributive support structure, and if* **false** *is the only sentence rejected by Φ, then* $IN_\Phi(I, K \cup L, J)$ *and* $IN_\Phi(I, K \cup J, L)$ *only if* $IN_\Phi(I, K, J \cup L)$.

4.4.2 Retrieving independence assertions

There is a topological test on directed acyclic graphs, called *d–separation*, that tells us whether two sets of nodes are d–separated by a third set [Pearl, 1988]. The d–separation test could be viewed as a relation $IN_\mathcal{G} \subseteq N \times N \times N$ where $IN_\mathcal{G}(I, K, J)$ holds precisely when K d–separates I from J in the directed acyclic graph $\mathcal{G}$. The importance of d–separation stems from the following result [Verma, 1986]:

Theorem 4.4.6 *Let Φ be the state of belief satisfying a causal network $\langle N, \mathcal{G}, \mathcal{CS} \rangle$. If K d–separates I from J in the graph, then the state of belief Φ finds I independent from J given K. That is, $If IN_\mathcal{G}(I, K, J)$, then $IN_\Phi(I, K, J)$.*

That is, d–separation allows us to infer many of the independences in a state of belief by examining the topology of the corresponding causal network. These independences will be very useful in deriving an algorithm for computing degrees of support, which is presented in Chapter 5.

The following definition is required to state d–separation:

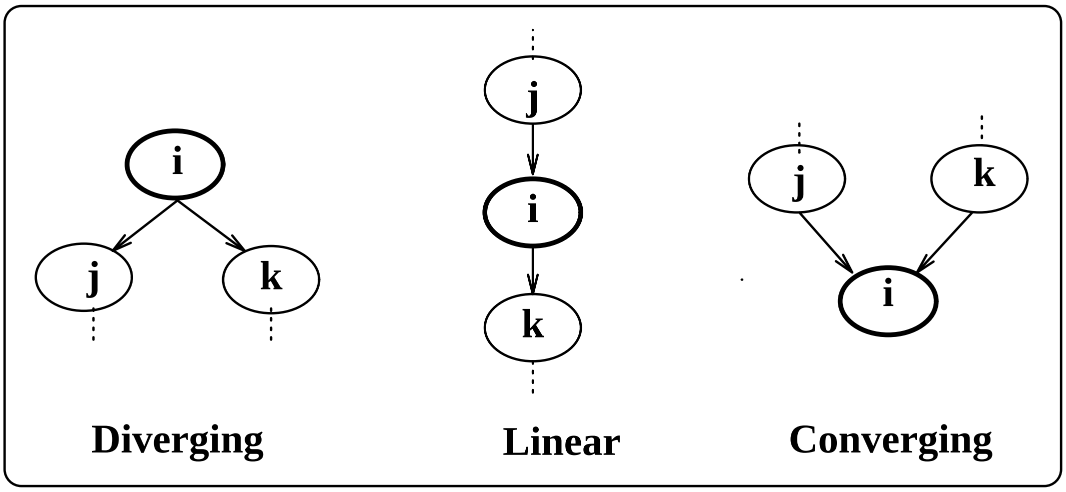

Figure 5: There are three types of intermediate nodes on a given path. The type of a node is determined by its relation to its neighbors. A node is diverging if both neighbors are children. A node is linear if one neighbor is a parent and the other is a child. A node is converging if both neighbors are parents.

Definition 4.4.7 *Let $\mathcal{G}$ be a directed acyclic graph and let I, J, and K be three disjoint sets of nodes in $\mathcal{G}$. A path between I and J is* <u>*K–active*</u> *precisely when its nodes satisfy the following conditions:*

1. *A converging node belongs to K or has a descendent in K.*

2. *A diverging or linear node is outside K.*

See Figure 5 for the definition of converging, diverging, and linear nodes.

Definition 4.4.8 ([Pearl, 1988]) *In a directed acyclic graph $\mathcal{G}$, nodes K d–separate I from J, written $IN_{\mathcal{G}}(I, K, J)$, precisely when there is no K–active path between I and J in $\mathcal{G}$.*

I conclude this chapter by the following observation: Applying the test of *d*–separation to a causal network is sound but not complete in terms of retrieving the independences in the corresponding state of belief. That is, some of the independences in a state of belief cannot be discovered by applying *d*–separation to the corresponding

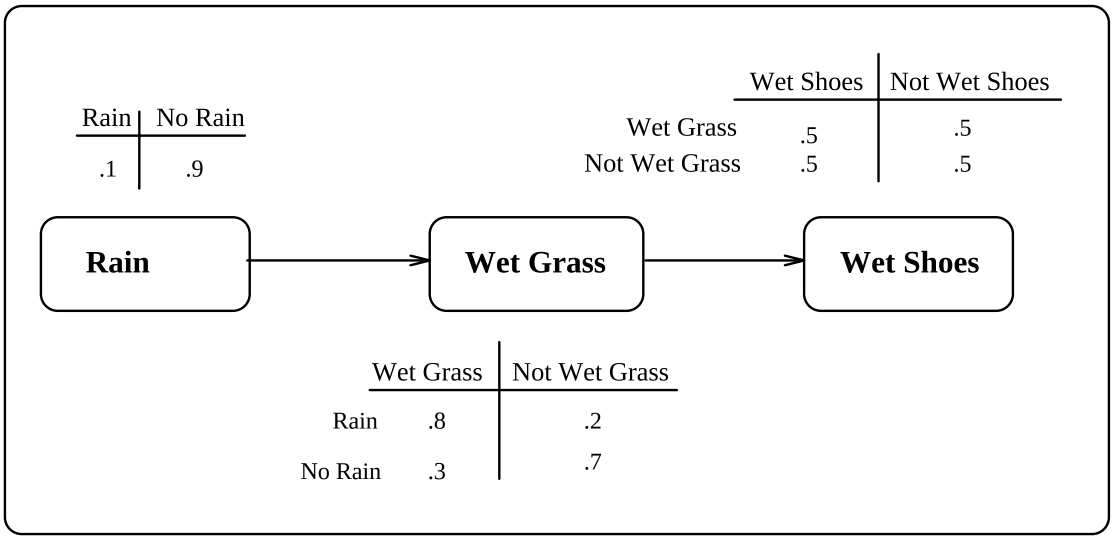

Figure 6: A probabilistic causal network.

causal network. The causal network given in Figure 6 is such an example. There, d–separation tells us that Wet_Grass is not d–separated from Wet_Shoes, which means that Wet_Grass is dependent on Wet_Shoes. However, the state of belief specified by this causal network finds Wet_Shoes independent from Wet_Grass.

Chapter 5

Independence and Belief Computation

In this chapter, I present an algorithm for computing the belief in every node of an abstract causal network. The algorithm is a direct generalization of the *polytree algorithm*, which is well–known in the probabilistic literature.

5.1 Introduction

Consider the causal network given in Figure 7, and suppose that we want to compute the belief in Node 0 given the observation $22 \wedge 30 \wedge \neg 39$. This is an instance of the computational problem I address in this chapter.

The probabilistic literature provides a number of algorithms for performing this computation with respect to probabilistic causal networks. For example, when the causal network is singly connected, the belief in a node can be computed using the popular polytree algorithm [Pearl, 1988; Peot and Shachter, 1991].[1] The probabilistic literature also provides a number of methods for extending the polytree algorithm to multiply connected networks.

In this chapter, I show that a direct generalization of the probabilistic polytree

[1]A network is singly connected if every two nodes are connected by at most one undirected path. The causal network given in Figure 7 is singly connected.

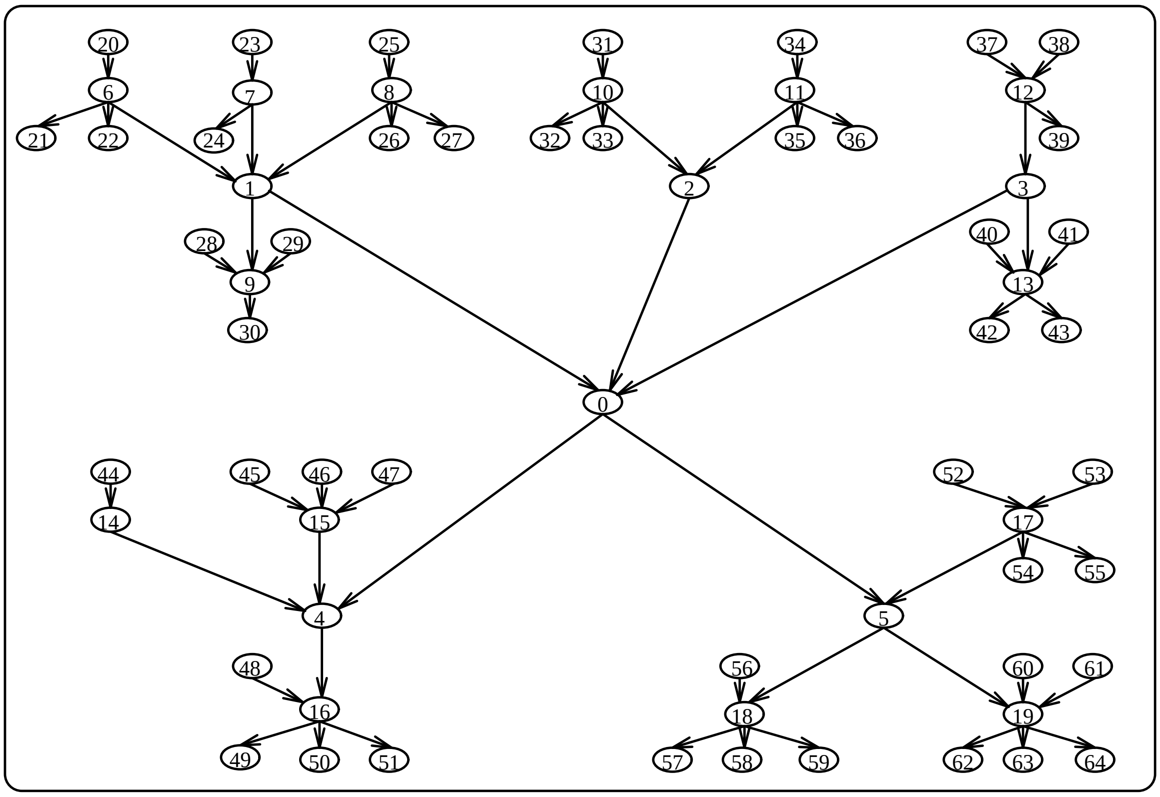

Figure 7: A singly connected causal network. There is at most one undirected path between any two nodes.

algorithm can be used to compute beliefs in abstract causal networks. I also show how the method of *conditioning,* which is well–known in the probabilistic literature, can be used to extend the abstract polytree algorithm to multiply connected causal networks.

The abstract polytree algorithm is introduced in Section 5.2 and given in Section 5.3. Control flow in this algorithm is discussed in Section 5.4, while its computational complexity is discussed in Section 5.5. The method of conditioning is discussed in Section 5.6. In Chapter 6, I present a Common Lisp implementation of the abstract polytree algorithm.

5.2 Introducing the abstract polytree algorithm

The probabilistic polytree algorithm computes a pair of probabilities,

$$\langle Pr(i \mid \delta), Pr(\neg i \mid \delta)\rangle,$$

for each node i in a causal network. Here, Pr is the probabilistic state of belief satisfying the given causal network, and δ is an observed state of some nodes in the network [Pearl, 1988]. A modification to this algorithm, suggested by Peot and Shachter, computes a different pair of probabilities,

$$\langle Pr(i \wedge \delta), Pr(\neg i \wedge \delta)\rangle,$$

for each node in the network [Peot and Shachter, 1991]. The first pair of probabilities can be obtained easily from the second pair because of the following equalities:

$$\begin{aligned} Pr(\delta) &= Pr(i \wedge \delta) + Pr(\neg i \wedge \delta), \\ Pr(\underline{i} \mid \delta) &= Pr(\underline{i} \wedge \delta)/Pr(\delta). \end{aligned}$$

Although both algorithms can be used to compute the conditional probability of some node given an observation, the modified polytree algorithm is preferred if the probability of the observation is desired. I shall, therefore, generalize the modified polytree algorithm.

5.2.1 Breaking down the computation

The input to the abstract polytree algorithm is

- $\langle N, \mathcal{G}, \mathcal{CS}\rangle$, an abstract causal network, and
- δ, a state of some nodes in N.

The output of the algorithm is a pair of supports for each node i,

- $\langle \Phi(i \wedge \delta), \Phi(\neg i \wedge \delta)\rangle$, which is denoted by BL_i, where Φ is the state of belief satisfying the causal network $\langle N, \mathcal{G}, \mathcal{CS}\rangle$.

When node i has n parents and m children, the abstract polytree algorithm breaks down the computation of the pair BL_i into $n+m$ computations involving $n+m$ subnetworks. Each of the first n subnetworks consists of nodes connected to node i via an incoming arc, while each of the following m subnetworks consists of nodes connected to node i via an outgoing arc. Since the causal network is singly connected, these subnetworks are guaranteed to be disjoint. For example, the computation of BL_0 is broken down into five computations involving the five subnetworks in Figure 8.

I show next the elements of this breakdown of computation in three stages.

The first stage

In the first stage, the computation of the pair BL_i is broken down into two computations:

- $\langle \Phi(i \wedge \delta_{i\triangleleft}), \Phi(\neg i \wedge \delta_{i\triangleleft})\rangle$, which is denoted by π_i.
- $\langle \Phi_i(\delta_{i\triangleright}), \Phi_{\neg i}(\delta_{i\triangleright})\rangle$, which is denoted by λ_i.

Here, $\delta_{i\triangleleft}$ is the observation about non–descendents of node i, and $\delta_{i\triangleright}$ is the observation about descendents of node i. When node i is not observed, the observation δ is equivalent to the conjunction of observations $\delta_{i\triangleleft}$ and $\delta_{i\triangleright}$.

For example, the computation of BL_0 in Figure 8 is broken down into two computations:

- $\langle \Phi(0 \wedge \delta_{0\triangleleft}), \Phi(\neg 0 \wedge \delta_{0\triangleleft})\rangle$, denoted by π_0.
- $\langle \Phi_0(\delta_{0\triangleright}), \Phi_{\neg 0}(\delta_{0\triangleright})\rangle$, denoted by λ_0.

Here, $\delta_{0\triangleleft}$ is $22 \wedge 30 \wedge \neg 39$ and $\delta_{0\triangleright}$ is $50 \wedge 54$.

The second stage

In the second stage, the computation of the pair π_i is broken down into a number of computations, each of which is associated with a parent j of node i:

- $\langle \Phi(j \wedge \delta_{i\triangleleft j}), \Phi(\neg j \wedge \delta_{i\triangleleft j})\rangle$, which is denoted by $\pi_{j.i}$.

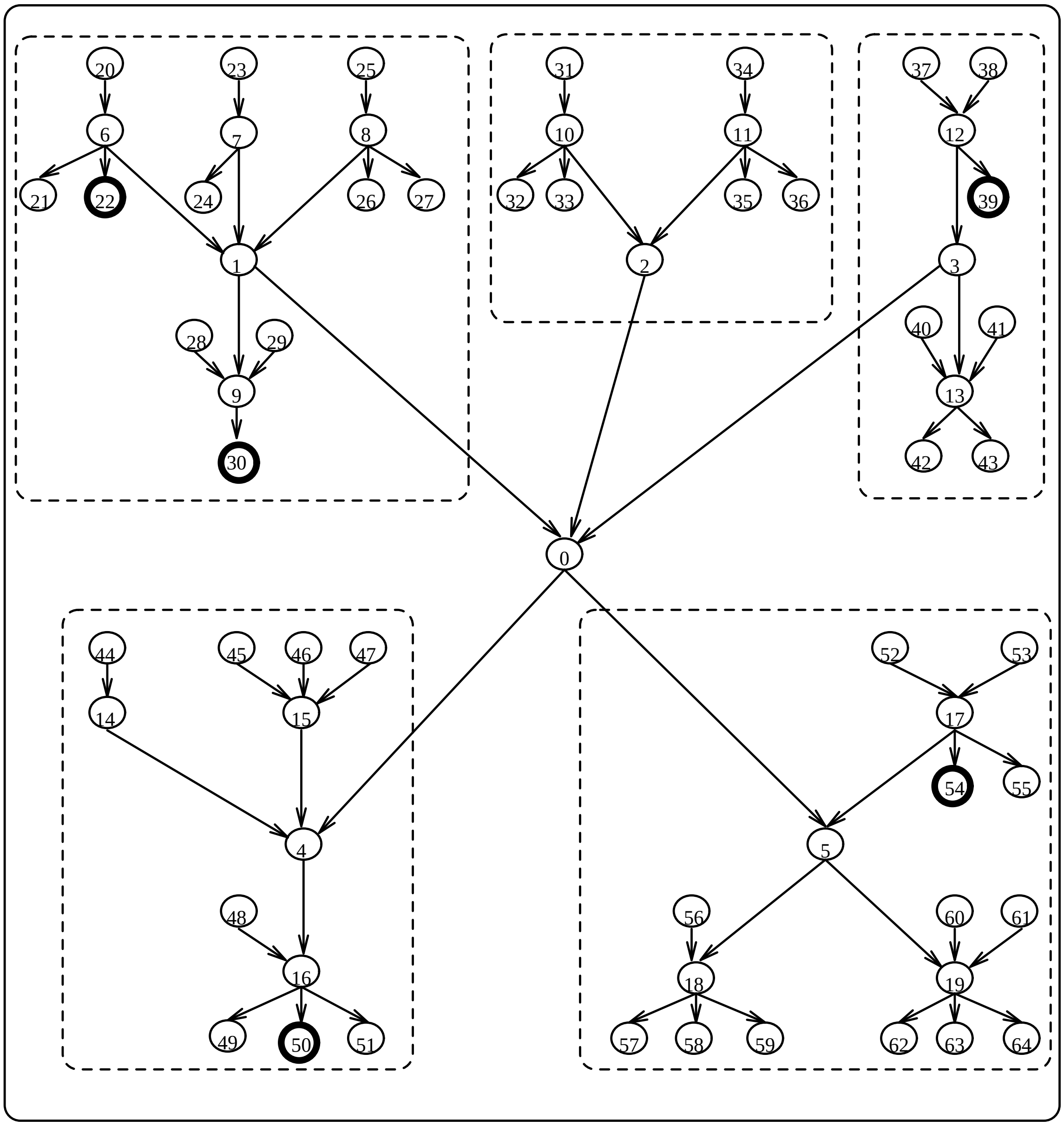

Figure 8: The computation breakdown for belief in Node 0 given the observation $22 \wedge 30 \wedge \neg 39 \wedge 50 \wedge 54$.

Here, $\delta_{i \triangleleft j}$ is the observation about nodes connected to node i via the incoming arc $j \rightarrow i$.

For example, the computation of π_0 in Figure 8 is broken down into three computations:

- $\langle \Phi(1 \wedge \delta_{0 \triangleleft 1}), \Phi(\neg 1 \wedge \delta_{0 \triangleleft 1}) \rangle$, denoted by $\pi_{1.0}$.
- $\langle \Phi(2 \wedge \delta_{0 \triangleleft 2}), \Phi(\neg 2 \wedge \delta_{0 \triangleleft 2}) \rangle$, denoted by $\pi_{2.0}$.
- $\langle \Phi(3 \wedge \delta_{0 \triangleleft 3}), \Phi(\neg 3 \wedge \delta_{0 \triangleleft 3}) \rangle$, denoted by $\pi_{3.0}$.

Here, $\delta_{0 \triangleleft 1}$ is $22 \wedge 30$, $\delta_{0 \triangleleft 2}$ is **true** and $\delta_{0 \triangleleft 3}$ is $\neg 39$.

The third stage

In the third stage, the computation of the pair λ_i is broken down into a number of computations, each of which is associated with a child k of node i:

- $\langle \Phi_i(\delta_{i \triangleright k}), \Phi_{\neg i}(\delta_{i \triangleright k}) \rangle$, which is denoted by $\lambda_{k.i}$.

Here, $\delta_{i \triangleright k}$ is the observation about nodes connected to node i via the outgoing arc $i \rightarrow k$.

For example, the computation of λ_0 in Figure 8 is broken down into two computations:

- $\langle \Phi_0(\delta_{0 \triangleright 4}), \Phi_{\neg 0}(\delta_{0 \triangleright 4}) \rangle$, denoted by $\lambda_{4.0}$.
- $\langle \Phi_0(\delta_{0 \triangleright 5}), \Phi_{\neg 0}(\delta_{0 \triangleright 5}) \rangle$, denoted by $\lambda_{5.0}$.

Here, $\delta_{0 \triangleright 4}$ is 50 and $\delta_{0 \triangleright 5}$ is 54.

Justifying the breakdown

The justification for the above computational breakdowns is based on:

1. An assumption that only leaf nodes having single parents could be observed in a causal network. As we shall see in Section 5.2.3, this assumption does not compromise the generality of the abstract polytree algorithm: Every observation about a node in a causal network can be simulated by an observation about an auxiliary leaf node having only one parent.

2. The independences asserted by a causal network. These independences are discussed in Section 5.2.4.

5.2.2 The message–passing paradigm

When each node i in a causal network stores its conditional belief given every state of its parents, that is, $\langle \mathcal{CS}_{\underline{i\diamond}}(i), \mathcal{CS}_{\underline{i\diamond}}(\neg i)\rangle$, the abstract polytree algorithm can be viewed as a process involving the following steps:

1. Node j computes the pair $\pi_{j.i}$ and sends it as a message to its child i.
2. Node k computes the pair $\lambda_{k.i}$ and sends it as a message to its parent i.
3. Node i combines the messages $\pi_{j.i}$ it receives from its parents to yield the pair π_i.
4. Node i combines the messages $\lambda_{k.i}$ it receives from its children to yield the pair λ_i.
5. Node i combines the pairs π_i and λ_i to yield the pair BL_i.

Given this view, the abstract polytree algorithm is specified by answering the following questions:

1. How should node i compute the message it must send to a parent?
2. How should node i compute the message it must send to a child?
3. How should node i combine the messages it receives from its parents?
4. How should node i combine the messages it receives from its children?
5. How should node i combine the pairs π_i and λ_i to yield the pair BL_i?
6. When should node i send a message to a parent?
7. When should node i send a message to a child?

The first five questions are answered in Section 5.3, while the last two questions are answered in Section 5.4. Before I move to these sections, I elaborate on the basic justifications for the abstract polytree algorithm in the next two sections.

5.2.3 Simulating observations

The abstract polytree algorithm assumes that observations are available only about leaf nodes having single parents. This assumption simplifies the algorithm, but does not affect its generality. I explain why in this section.

Given a state of belief Φ over propositions N, we can simulate the observation of proposition i that belongs to N by the observation of an additional proposition k that does not belong to N. In particular, let us create a new state of belief Ψ over propositions $N \cup \{k\}$ such that

- $\Psi(\underline{N \setminus \{i\}} \wedge i \wedge k) = \Phi(\underline{N \setminus \{i\}} \wedge i)$.
- $\Psi(\underline{N \setminus \{i\}} \wedge \neg i \wedge \neg k) = \Phi(\underline{N \setminus \{i\}} \wedge \neg i)$.
- $\Psi(\underline{N \setminus \{i\}} \wedge i \wedge \neg k) = \mathbf{0}$.
- $\Psi(\underline{N \setminus \{i\}} \wedge \neg i \wedge k) = \mathbf{0}$.

The new state of belief has three important properties. First, it agrees with the old state of belief on any sentence of the form $\underline{N}$. Second, it accepts the sentence $i \equiv k$. Third, it finds proposition k independent from propositions $N \setminus \{i\}$ given proposition i. Hence, accepting or rejecting proposition k in the new state of belief has the same effect as accepting or rejecting proposition i.

Therefore, we can extend a causal network that represents a state of belief, such as Φ, to a causal network that represents a new state of belief, such as Ψ, by doing the following:

1. Creating a new node k,
2. making node k a child of node i, and
3. quantifying the causal connection from node i to node k as follows:

$$\begin{aligned}
\mathcal{CS}_i(k) &= \mathbf{1} \\
\mathcal{CS}_i(\neg k) &= \mathbf{0} \\
\mathcal{CS}_{\neg i}(k) &= \mathbf{0} \\
\mathcal{CS}_{\neg i}(\neg k) &= \mathbf{1}.
\end{aligned}$$

The newly created node k has the property of being a leaf node with only a single parent i. Therefore, we can simulate an observation about any node in a causal network by an observation about a newly created leaf node that has a single parent. For example, observation 0 in the causal network of Figure 7 can be simulated by observation 65 after Node 65 is created as given in Figure 9. Similarly, Nodes 66 and 67 are added to simulate observations about Nodes 45 and 14, respectively.

The abstract polytree algorithm refers to four classes of nodes:

Root nodes, which have no parents.

Normal nodes, which have at least one parent and at least one child.

Observed leaf nodes, which have no children and are observed.

Non–observed leaf nodes, which have no children but are not observed.

Let me repeat here that observed leaf nodes are added to a causal network in order to simulate observations about other nodes. Moreover, since observed nodes are always leaf nodes having single parents, root, normal and leaf nodes having multiple parents cannot be observed.

In Figure 9, Node 20 is root, Node 0 is normal, Nodes 50 and 65 are observed leafs, and Node 21 is a non–observed leaf.

5.2.4 The independences of singly connected networks

The abstract polytree algorithm is based on four independences that are asserted by every singly connected network. (Recall, a network is singly connected if it has at most one undirected path between any two nodes — the network given in Figure 7 is an example.) I state these independence assertions now to use them later in deriving the abstract polytree algorithm.

Theorem 5.2.1 *Let I be some non–descendents of node i, J be some descendents of node i, and K be a set of nodes disjoint from I and J. If K includes i, then I is independent from J given K.*

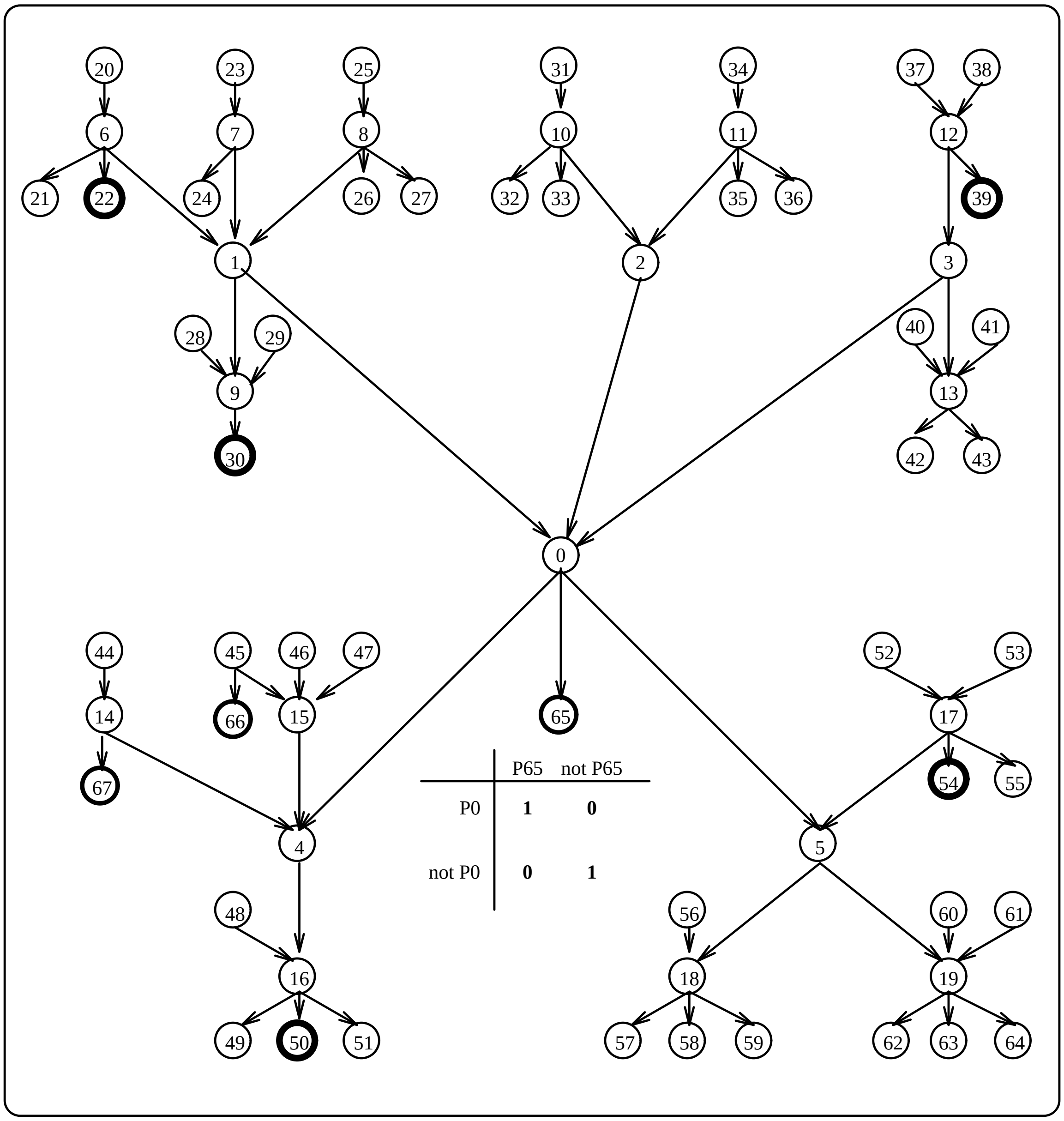

Figure 9: A singly connected network with an observation, $22 \wedge 30 \wedge \neg 39 \wedge 50 \wedge 54 \wedge 65 \wedge 66 \wedge 67$. Nodes 22, 30, 39, 50, 54, 65, 66, and 67 are added to the network to simulate observations about nodes 6, 9, 12, 16, 17, 0, 45 and 14, respectively.

In Figure 9, Nodes {22, 30, 39} are independent from Nodes {50, 54, 65, 66, 67} given Node 0. As we shall see in Section 5.3.1, Theorem 5.2.1 is the basis for breaking down the computation of the pair BL_i into the computation of the pairs π_i and λ_i.

Theorem 5.2.2 *Let I and J be sets of nodes connected to node i via disjoint outgoing arcs. Then I is independent from J given i.*

In Figure 9, Nodes {50, 66, 67} are independent from Nodes {54, 65} given Node 0. As we shall see in Section 5.3.2, Theorem 5.2.2 is the basis for breaking down the computation of the pair λ_i into the computation of the pairs $\lambda_{k.i}$.

Theorem 5.2.3 *Let K be some parents of node i, and let J be set of nodes disjoint from K and connected to node i via nodes in K. Then i is independent from J given K.*

In Figure 9, Nodes {22, 30, 39} are independent from Node 0 given Nodes {1, 2, 3}.

Theorem 5.2.4 *Let I and J be sets of nodes connected to node i via disjoint incoming arcs. Then I is independent from J.*

In Figure 9, Nodes {1, 22, 30} are independent from Nodes {3, 39}. As we shall see in Section 5.3.3, Theorems 5.2.3 and 5.2.4 are the basis for breaking down the computation of the pair π_i into the computation of the pairs $\pi_{j.i}$.

The independence assertions discussed in this section have uses that go beyond what I have mentioned here. In particular, these assertions can be used to show that the message sent by a node to a neighbor can be computed form the messages received by the node from other neighbors. This is discussed in more detail in Section 5.3.

In the next section, I discuss some notational conventions that are useful for Section 5.3.

5.2.5 Manipulating pairs of support

The abstract polytree algorithm maintains a number of support pairs with each node i. The first component of each pair is related to proposition i and is called the i–component of that pair. The second component is related to the negation of proposition i and is called the $\neg i$–component of that pair. These components are accessed as follows:

$$\begin{aligned}\langle a, b\rangle(i) &\stackrel{def}{=} a\\ \langle a, b\rangle(\neg i) &\stackrel{def}{=} b.\end{aligned}$$

The following operations are also defined on pairs:

$$\begin{aligned}\langle a, b\rangle \otimes \langle c, d\rangle &\stackrel{def}{=} \langle a \otimes c, b \otimes d\rangle\\ \langle a, b\rangle \oslash \langle c, d\rangle &\stackrel{def}{=} \langle a \oslash c, b \oslash d\rangle\\ c \otimes \langle a, b\rangle &\stackrel{def}{=} \langle c \otimes a, c \otimes b\rangle\\ \langle a, b\rangle \oslash c &\stackrel{def}{=} \langle a \oslash c, b \oslash c\rangle.\end{aligned}$$

5.3 The abstract polytree algorithm

I present the abstract polytree algorithm in this section by showing how to compute and combine the messages exchanged between nodes in a causal network. I also give concrete examples of message computation and combination with respect to computing the belief in Node 0, which appears in Figure 9, given the observation

$$\delta = 22 \wedge 30 \wedge \neg 39 \wedge 50 \wedge 54 \wedge 65 \wedge 66 \wedge 67.$$

5.3.1 Belief

The pair BL_i is called the *belief* in node i. The following theorem shows how to compute this pair.

Theorem 5.3.1 *Let i be a non–observed node in a singly connected causal network.*

$$\begin{array}{rll} \textit{If} & BL_i & \stackrel{def}{=} \langle \Phi(i \wedge \delta), \Phi(\neg i \wedge \delta) \rangle, \\ & \pi_i & \stackrel{def}{=} \langle \Phi(i \wedge \delta_{i\triangleleft}), \Phi(\neg i \wedge \delta_{i\triangleleft}) \rangle, \\ & \lambda_i & \stackrel{def}{=} \langle \Phi_i(\delta_{i\triangleright}), \Phi_{\neg i}(\delta_{i\triangleright}) \rangle, \\ \textit{then} & BL_i & = \pi_i \otimes \lambda_i. \end{array}$$

According to this theorem, the belief in Node 0 is given by

$$\underbrace{\langle \Phi(0 \wedge \delta_{0\triangleleft}), \Phi(\neg 0 \wedge \delta_{0\triangleleft}) \rangle}_{\pi_0} \langle \Phi_0(\delta_{0\triangleright}), \Phi_{\neg 0}(\delta_{0\triangleright}) \rangle_{\lambda_0},$$

where

$$\begin{aligned} \delta_{0\triangleleft} &= 22 \wedge 30 \wedge \neg 39, \\ \delta_{0\triangleright} &= 50 \wedge 54 \wedge 65 \wedge 66 \wedge 67. \end{aligned}$$

Computing the belief in Node 0 can be viewed as the aggregate of two computations. First, the computation of the pair π_0, which depends only on nodes connected to Node 0 via incoming arcs. And second, the computation of the pair λ_0, which depends only on nodes that are connected to Node 0 via outgoing arcs. These sets of nodes are shown in Figure 10.

In the following sections, I show how to compute the pairs π_0 and λ_0, which completes the computation of belief in Node 0.

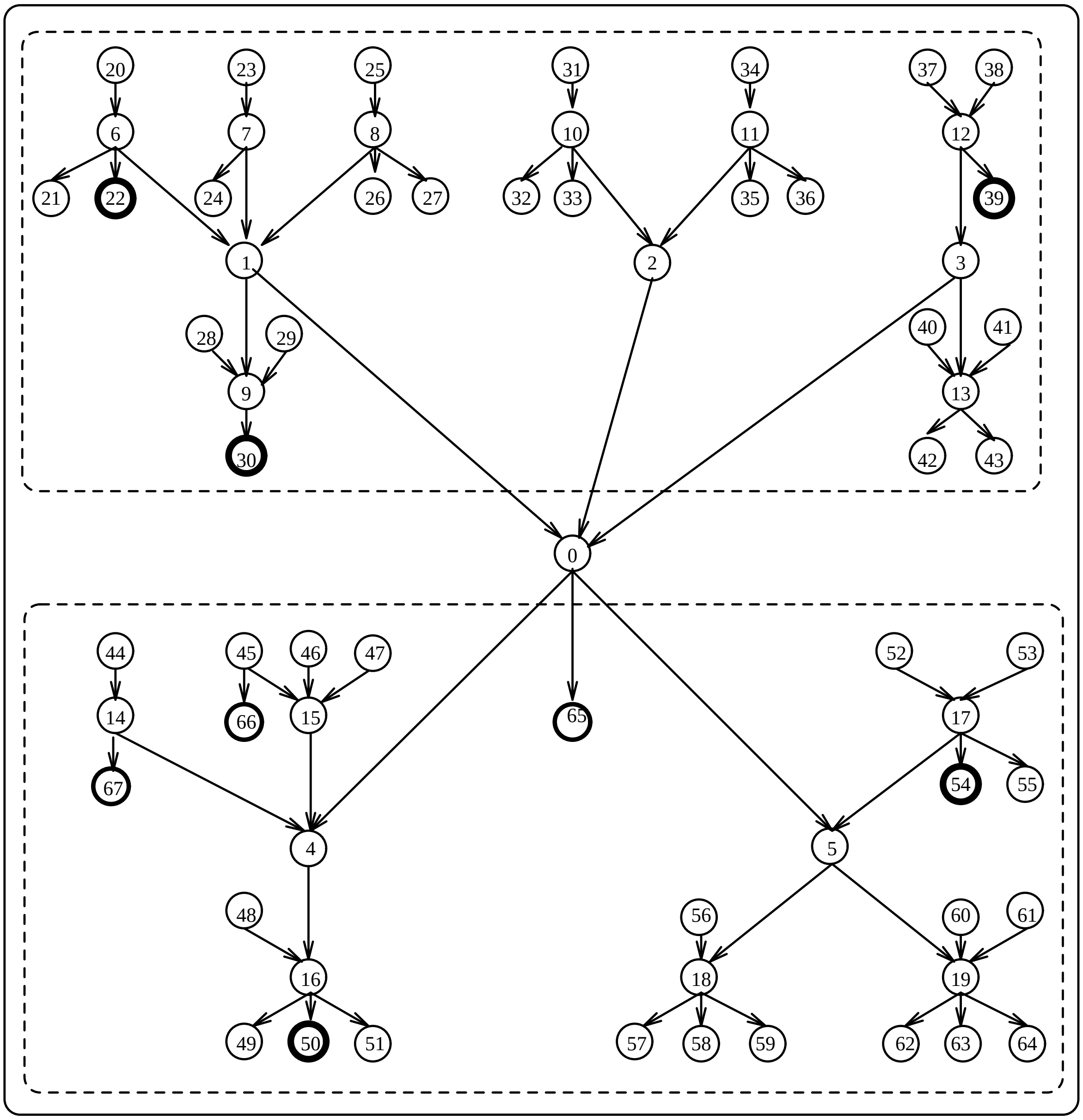

Figure 10: Computing the belief in Node 0 can be viewed as the aggregate of two computations involving the subnetworks shown above.

5.3.2 Diagnostic support

The pair λ_i is called the *diagnostic support* for node i. It is computed as follows.

Notation $i\circ$ denotes the children of node i.

Theorem 5.3.2 *Let i be a non–observed node in a singly connected network.*

$$\begin{array}{lll} \textit{If} & \lambda_{k.i} & \stackrel{def}{=} \langle \Phi_i(\delta_{i\triangleright k}), \Phi_{\neg i}(\delta_{i\triangleright k})\rangle, \\ \textit{then} & \lambda_i & = \bigotimes_{k \in i\circ} \lambda_{k.i}. \end{array}$$

According to Theorem 5.3.2, computing diagnostic support for a node can be broken down into a number of computations, each of which is associated with a child of that node. In Section 5.3.4, I show how to perform each of these subcomputations.

The diagnostic support for Node 0 equals

$$\underbrace{\langle \Phi_0(\delta_{0\triangleright 4}), \Phi_{\neg 0}(\delta_{0\triangleright 4})\rangle}_{\lambda_{4.0}} \otimes \underbrace{\langle \Phi_0(\delta_{0\triangleright 65}), \Phi_{\neg 0}(\delta_{0\triangleright 65})\rangle}_{\lambda_{65.0}} \otimes \underbrace{\langle \Phi_0(\delta_{0\triangleright 5}), \Phi_{\neg 0}(\delta_{0\triangleright 5})\rangle}_{\lambda_{5.0}},$$

where

$$\begin{aligned} \delta_{0\triangleright 4} &= 50 \wedge 66 \wedge 67, \\ \delta_{0\triangleright 65} &= 65, \\ \delta_{0\triangleright 5} &= 54. \end{aligned}$$

Computing the diagnostic support for Node 0 can then be viewed as the aggregate of three computations. First, the computation of the pair $\lambda_{4.0}$, which depends only on Nodes {4,14-16,44-51}. Next, the computation of the pair $\lambda_{65.0}$, which depends only on Node 65. Finally, the computation of the pair $\lambda_{5.0}$, which depends only on Nodes {5,17-19,52-64}. These sets of nodes are shown in Figure 11.

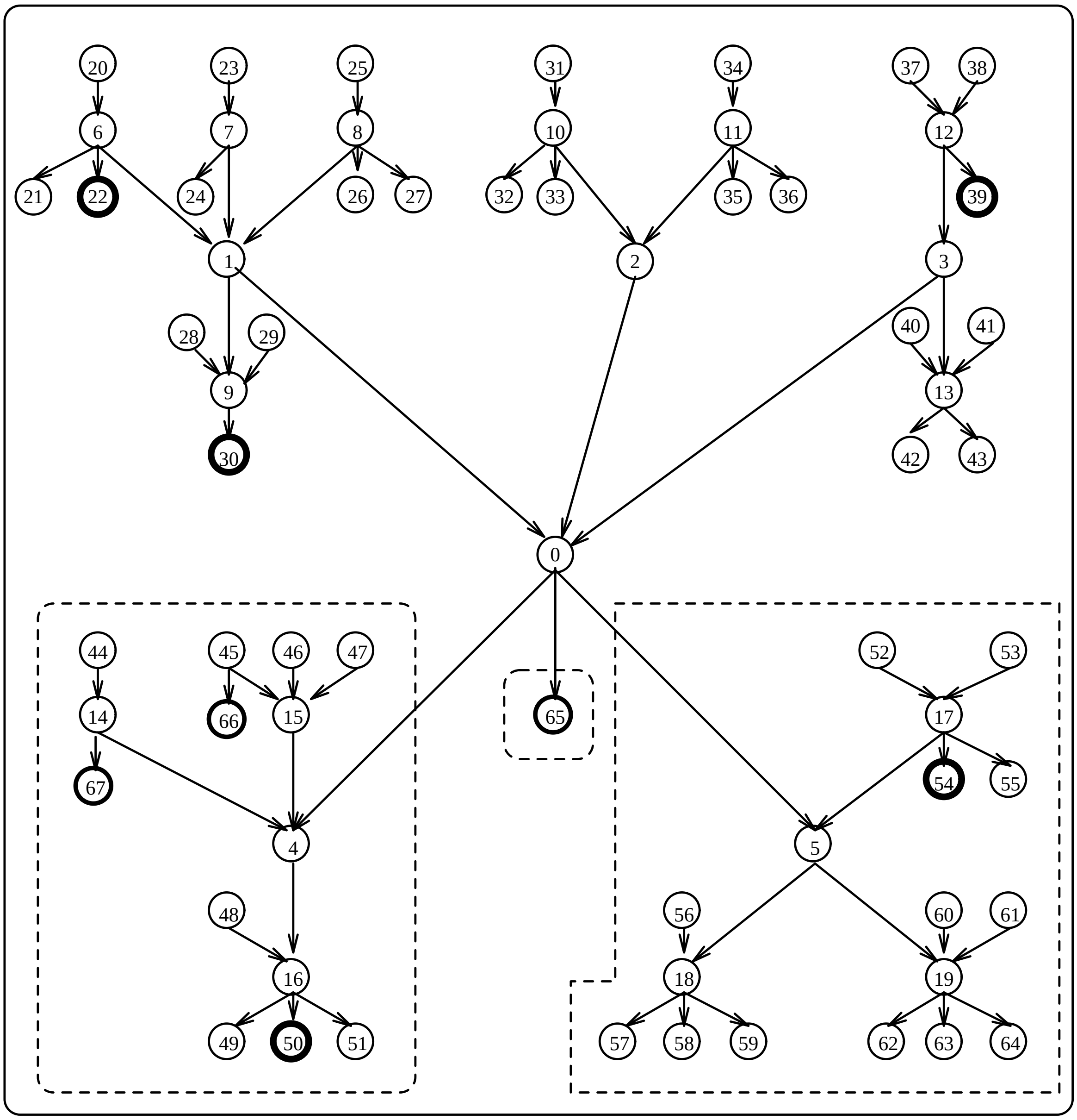

Figure 11: Computing diagnostic support for Node 0 can be viewed as the aggregate of three computations. The first computes the belief in observations about nodes in the left subnetwork given Node 0. The second computes the belief in observations about nodes in the middle subnetwork given Node 0. The third computes the belief in observations about nodes in the right subnetwork given Node 0.

5.3.3 Causal support

The pair π_i is called the *causal support* for node i. It is computed as follows.

Theorem 5.3.3 *Let i be a non–observed node in a singly connected network.*

$$\begin{array}{lll} \textit{If} & \pi_{j.i} & \stackrel{def}{=} \langle \Phi(j \wedge \delta_{i \triangleleft j}), \Phi(\neg j \wedge \delta_{i \triangleleft j}) \rangle, \\ \textit{then} & \pi_i(\underline{i}) & = \bigoplus_{\underline{i\diamond}} \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond} \models \underline{j}} \pi_{j.i}(\underline{j}). \end{array}$$

According to this theorem, computing the causal support for a node can be broken down into a number of computations, each of which is associated with a parent of the node. In Section 5.3.5, I show how to perform each of these subcomputations.

Computing the causal support for Node 0 can be viewed as the aggregate of three computations. First, the computation of the pair $\pi_{1.0}$, which depends only on Nodes {1,6-9,20-30}. Next, the computation of the pair $\pi_{2.0}$, which depends only on Nodes {2,10-11,31-36}. Finally, the computation of the pair $\pi_{3.0}$, which depend only on Nodes {3,12-13,37-43}. These sets of nodes are shown in Figure 12.

5.3.4 Diagnostic Support to a parent

The pair $\lambda_{i.j}$ is called the diagnostic support form node i to its parent j. It is computed as follows.

Notation $i\diamond j$ denotes the parents of node i excluding parent j.

Theorem 5.3.4 *Let i be a non–observed node in a singly connected network. Then*

$$\lambda_{i.j}(\underline{j}) = \bigoplus_{\underline{i}} \lambda_i(\underline{i}) \otimes \bigoplus_{\underline{i\diamond j}} \mathcal{CS}_{\underline{j} \wedge \underline{i\diamond j}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond j} \models \underline{l}} \pi_{l.i}(\underline{l}).$$

Moreover, if i is an observed node, then

$$\lambda_{i.j} = \begin{cases} \langle \mathbf{1}, \mathbf{0} \rangle, & \textit{if } \delta \models i; \\ \langle \mathbf{0}, \mathbf{1} \rangle, & \textit{if } \delta \models \neg i. \end{cases}$$

According to this theorem, when node i is not observed, the message it sends to its parent j is the result of combining the messages that node i receives from all its neighbors, except node j.

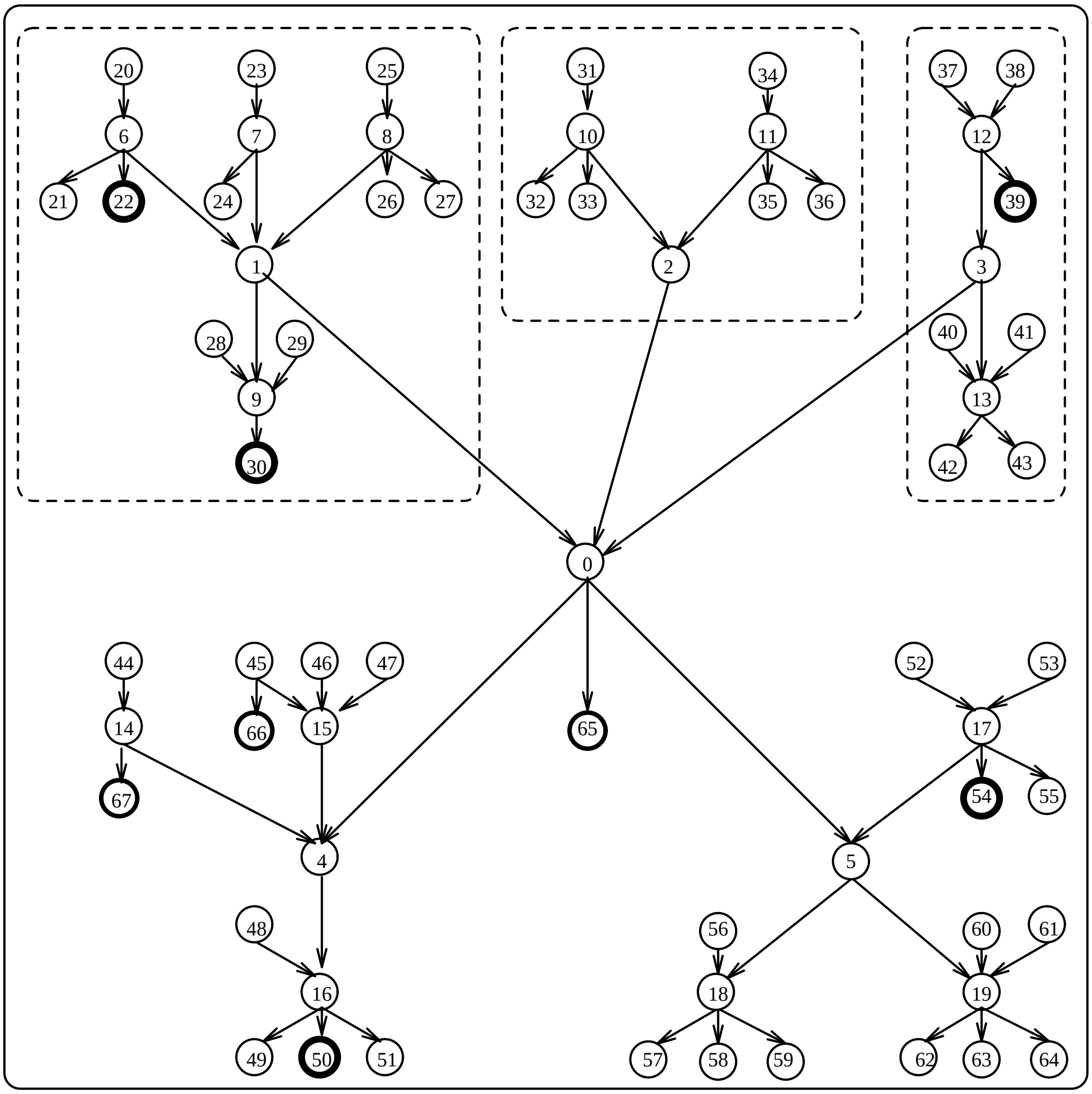

Figure 12: Computing the causal support for Node 0 can be viewed as the aggregate of three computations involving the subnetworks shown above.

5.3.5 Causal support to a child

The pair $\pi_{i.k}$ is called the support from node i to its child k. It is computed as follows.
Notation $i{\circ}k$ denotes the children of node i excluding child k.

Theorem 5.3.5 *Let i be a non–observed node in a singly connected network. Then*

$$\pi_{i.k} = \pi_i \otimes \bigotimes_{l \in i \circ k} \lambda_{l.i}.$$

According to this theorem, the message that node i sends to its child k is the result of combining the messages that node i receives from all its neighbors, except node k.

5.3.6 Summary of the abstract polytree algorithm

Following is a summary of the computations that each class of nodes performs. All computations are based on node i with parent j and child k.

Root nodes

Causal support:

$$\pi_i = \langle \mathcal{CS}_{\mathbf{true}}(i), \mathcal{CS}_{\mathbf{true}}(\neg i) \rangle.$$

Diagnostic support: If support from every child is available, then

$$\lambda_i = \bigotimes_{k \in i \circ} \lambda_{k.i}. \tag{4}$$

Belief: If diagnostic support is available, then

$$BL_i = \pi_i \otimes \lambda_i. \tag{5}$$

Support to a child:

$$\pi_{i.k} = \pi_i \otimes \bigotimes_{l \in i \circ k} \lambda_{l.i}. \tag{6}$$

Support to a parent: Not applicable.

Normal nodes

Causal support: If support from every parent is available, then

$$\pi_i(\underline{i}) = \bigoplus_{\underline{i\diamond}} \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond}\models \underline{j}} \pi_{j.i}(\underline{j}). \tag{7}$$

Diagnostic support: Use Equation 4.

Belief: Use Equation 5.

Support to a child: Use Equation 6.

Support to a parent: If diagnostic support is available and causal support from every parent except j is also available, then

$$\lambda_{i.j}(\underline{j}) = \bigoplus_{\underline{i}} \lambda_i(\underline{i}) \otimes \bigoplus_{\underline{i\diamond j}} \mathcal{CS}_{\underline{j}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond j}\models \underline{l}} \pi_{l.i}(\underline{l}). \tag{8}$$

Non–observed leaf nodes

Causal support: Use Equation 7.

Diagnostic support: $\lambda_i = \langle \mathbf{1}, \mathbf{1} \rangle$.

Belief: Use Equation 5.

Support to a child: Not applicable.

Support to a parent: Use Equation 8.

Observed leaf nodes

Causal support: Not applicable.

Diagnostic support: Not applicable.

Belief: Available directly from the observation.

Support to a child: Not applicable.

Support to a parent:

$$\lambda_{i.j} = \begin{cases} \langle \mathbf{1}, \mathbf{0} \rangle, & \text{if } \delta \models i; \\ \langle \mathbf{0}, \mathbf{1} \rangle, & \text{if } \delta \models \neg i. \end{cases} \tag{9}$$

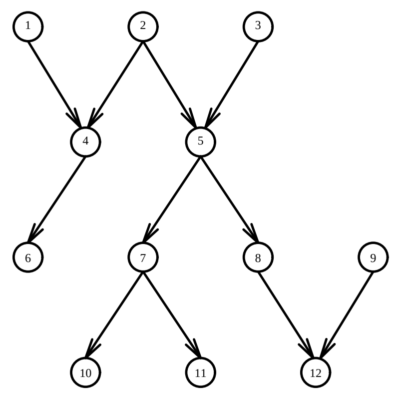

Figure 13: A singly connected causal network.

5.4 Control flow in singly connected networks

Now that we know how to compute messages, we need to know the order in which to propagate them. In the following two sections, I discuss two propagation schemes that are well–known in the probabilistic literature.

5.4.1 Backward propagation

In this propagation scheme, one is interested in computing the belief in a particular node. Backward propagation starts when such a node requests messages from its neighbors in order to compute its belief. This sparks a chain reaction in which every node that needs to send a message to a neighbor requests messages from all other neighbors.

Consider the causal network depicted in Figure 13 as an example, and suppose that we want to compute the belief in Node 5. This computation requires the messages

$$\lambda_{2.5}, \lambda_{3.5}, \pi_{7.5}, \pi_{8.5},$$

which are depicted in Figure 14(a).

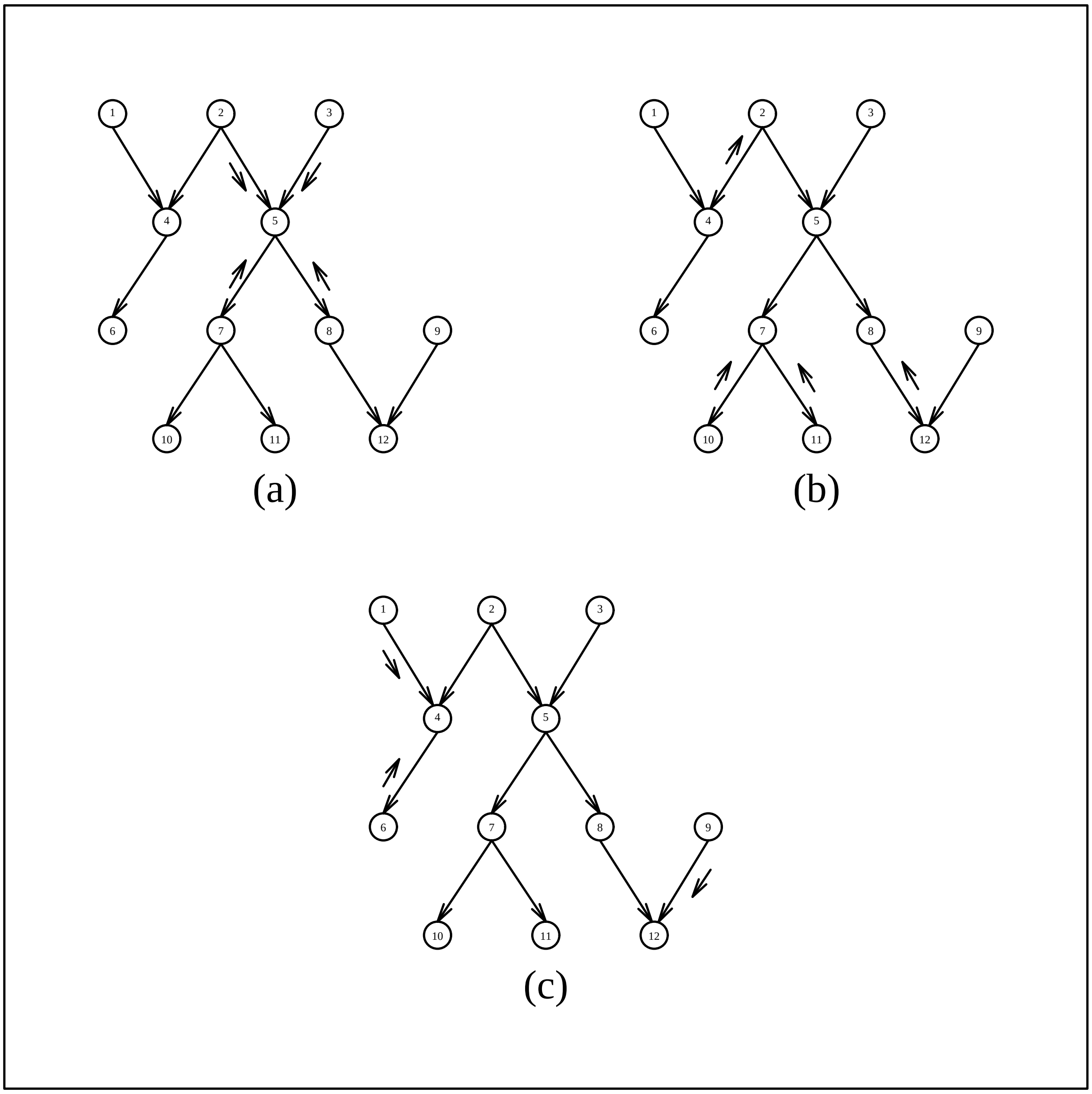

Figure 14: The stages of backward propagation for computing the belief in Node 5.

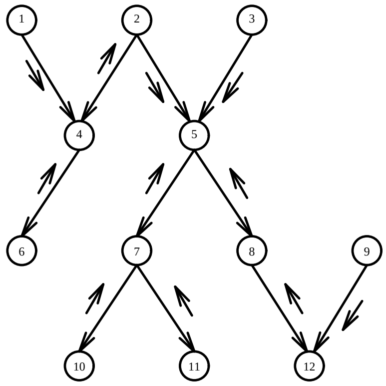

Figure 15: The messages exchanged in backward propagation for computing the belief in Node 5.

Node 3 is ready to send Message $\lambda_{3.5}$, but

- Node 2 needs Message $\pi_{4.2}$ in order to compute Message $\lambda_{2.5}$.
- Node 7 needs Messages $\pi_{10.7}$ and $\pi_{11.7}$ in order to compute Message $\pi_{7.5}$.
- Node 8 needs Message $\pi_{12.8}$ in order to compute Message $\pi_{8.5}$.

The required messages are depicted in Figure 14(b). Nodes 10 and 11 are ready to send Messages $\pi_{10.7}$ and $\pi_{11.7}$, but

- Node 4 needs Messages $\pi_{1.4}$ and $\lambda_{6.4}$ in order to compute Message $\pi_{4.2}$.
- Node 12 needs Message $\pi_{9.12}$ in order to compute Message $\pi_{12.8}$.

The required messages are depicted in Figure 14(c). Nodes 1, 6, and 9 are ready to compute Messages $\pi_{1.4}$, $\lambda_{6.4}$, and $\pi_{9.12}$, respectively. Figure 15 depicts all the intra–node messages needed for computing the belief in Node 5.

Below are some observations about the backward propagation shown in Figure 15:

1. There are 11 messages exchanged between nodes, which is the number of arcs in the causal network.

2. Only one message is exchanged across each arc and it heads towards Node 5.

3. The propagation takes place in three stages, which is the length of the path between Node 5 and the furthest boundary (root or leaf) node.

I now formalize backward propagation and then prove that it terminates.

The following definition says that backward propagation starts when the node of interest requests messages from all its neighbors. This sparks a chain reaction in which a node requests messages from its neighbors whenever one of these neighbors requests a message from it.

Definition 5.4.1 (Backward Propagation) *Let CN be a singly connected causal network and let i be a node in CN. Backward propagation with respect to $\langle CN, i\rangle$ is a sequence of non–empty sets $S_1, S_2, \ldots$, where*

- *S_1 consists of the messages directed to node i.*

- *S_n contains a message from node j to node k if S_{n-1} contains a message from node k to a node other than j.*

The sets $S_1, S_2, \ldots$ are called propagation states.

The backward propagation in Figure 14 with respect to Node 5 is

$$\begin{aligned} S_1 &= \{\pi_{2.5}, \pi_{3.5}, \lambda_{7.5}, \lambda_{8.5}\} \\ S_2 &= \{\lambda_{4.2}, \lambda_{10.7}, \lambda_{11.7}, \lambda_{12.8}\} \\ S_3 &= \{\pi_{1.4}, \pi_{9.12}, \lambda_{6.4}\}. \end{aligned}$$

Theorem 5.4.2 *Let CN be a singly connected causal network and let i be a node in CN. The number of states in backward propagation with respect to $\langle CN, i\rangle$ is no more than the length of the longest path between node i and any other node in CN.*

5.4.2 Forward propagation

In forward propagation, one is interested in computing the belief in every node of a causal network. One way to compute these beliefs is to use backward propagation on each node of the network. The number of messages exchanged in this computation would be the number of nodes in the network multiplied by the number of arcs.

In this section, I describe a propagation scheme for computing the belief of every node of a causal network, which involves only twice the number of messages exchanged in backward propagation. Again, this propagation scheme is well–known in the probabilistic literature.

Forward propagation starts when every node having a single neighbor sends a message to this neighbor. This sparks a chain reaction in which each node sends a message to a neighbor after it has received messages from all other neighbors.

Consider the causal network in Figure 13 as an example. Two classes of nodes can send messages to their neighbors immediately. These are root nodes with a single child and leaf nodes with a single parent. In the causal network of Figure 13, Nodes 1, 3, and 9 fall into the first class, and Nodes 6, 10, and 11 fall into the second. The messages sent by these nodes to their neighbors are depicted in Figure 16(a).

Once these messages are sent, a chain reaction is started:

1. Nodes 4, 5, and 12 send the messages depicted in Figure 16(b).
2. Nodes 2 and 8 follow by sending the messages depicted in Figure 16(c).
3. This enables Node 5 to send the four messages depicted in Figure 16(d).
4. Nodes 2, 7, and 8 follow by sending the messages depicted in Figure 16(e).
5. Finally, Nodes 4 and 12 send the messages depicted in Figure 16(f).

Figure 17 depicts the messages sent during all the previous stages. Every node has all the information it needs to compute its belief.

The following observations are about the forward propagation in Figure 16:

1. There are 22 exchanged messages, which is twice the number of arcs in the causal network.

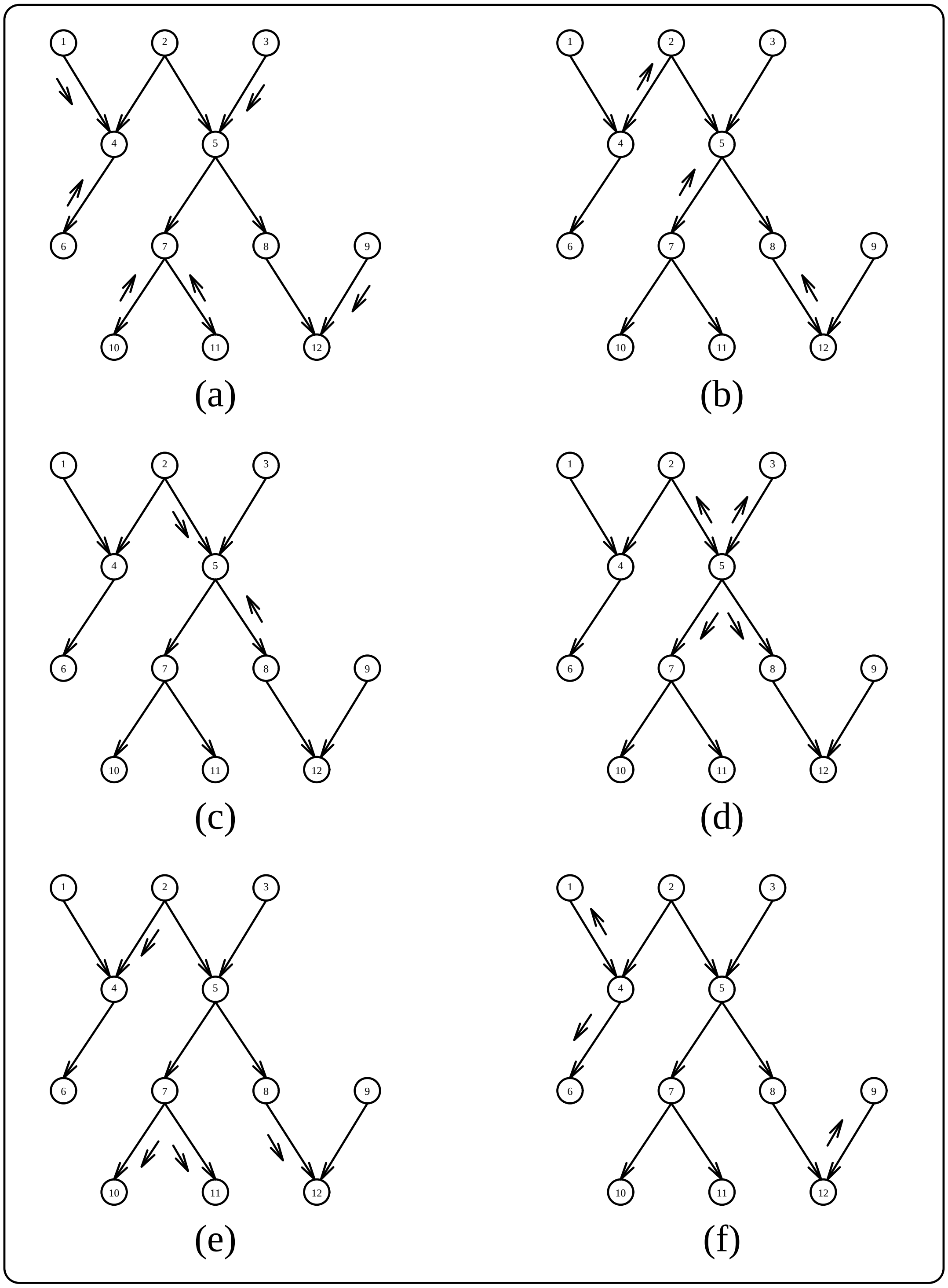

Figure 16: The stages of forward propagation.

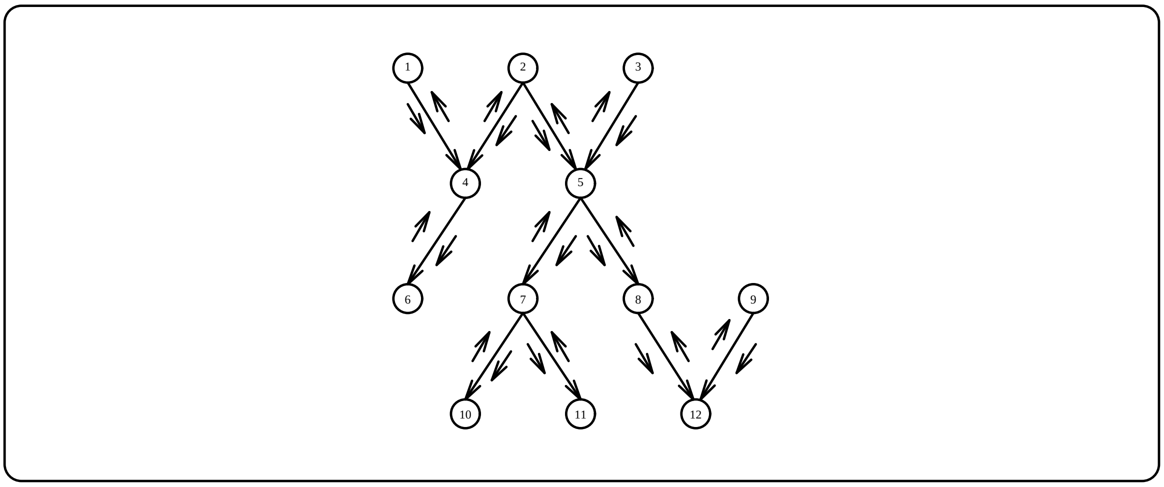

Figure 17: The messages exchanged in forward propagation.

2. Only two messages are exchanged across an arc, one in each direction.

3. The propagation takes place in six stages, which is the length of a longest path in the network.

I now formalize forward propagation and then prove that it terminates.

The following definition says that forward propagation starts when every node having a single neighbor sends a message to its neighbor. This sparks a chain reaction in which a node sends a message to a neighbor once it has received messages from all other neighbors.

Definition 5.4.3 (Forward Propagation) *Let CN be a singly connected causal network. Forward propagation with respect to CN is a sequence of non–empty sets* $S_1, S_2, \ldots$, *where*

- S_1 *contains the messages that could be sent by nodes with single neighbors.*

- S_n *contains a message from node* j *to node* k *if* S_{n-1} *contains all the messages from neighbors of node* j*, other than* k*, to node* j*.*

The forward propagation with respect to the causal network of Figure 16(g) is

$$\begin{aligned}
S_1 &= \{\pi_{1.4}, \pi_{3.5}, \pi_{9.12}, \lambda_{6.4}, \lambda_{10.7}, \lambda_{11.7}\} \\
S_2 &= \{\lambda_{4.2}, \lambda_{7.5}, \lambda_{12.8}\} \\
S_3 &= \{\pi_{2.5}, \lambda_{8.5}\} \\
S_4 &= \{\pi_{5.7}, \pi_{5.8}, \lambda_{5.2}, \lambda_{5.3}\} \\
S_5 &= \{\pi_{2.4}, \pi_{7.10}, \pi_{7.11}, \pi_{8.12}\} \\
S_6 &= \{\pi_{4.6}, \lambda_{4.1}, \lambda_{12.9}\}.
\end{aligned}$$

The following theorem says that forward propagation must terminate.

Theorem 5.4.4 *The number of states in forward propagation with respect to a singly connected causal network CN is no more than the length of a longest path in CN.*

5.5 Computational complexity

In this section, I count the number of support operations performed by a normal node during forward propagation. A normal node with n parents and m children performs the following computations:

1. Causal support. This involves $n2^{n+1}$ operations.
2. Diagnostic support. This involves $2m - 2$ operations.
3. Belief. This involves 2 operations.
4. Messages for all children. This involves $2m$ operations.
5. Messages for all parents. This involves $(n-1)n2^{n+1} + 4n$ operations.

Adding up all the above costs gives $n^2 2^{n+1} + 4(n+m)$. This expression is exponential in the number of parents and linear in the number of children.

Let me summarize what else we know about the number of messages involved in forward propagation:

- The number of total messages exchanged is twice the number of arcs in the network.
- The propagation of these messages takes place at a number of stages that equals the length of a longest path in the network.

The number of arcs in a singly connected network is always one less than the number of nodes. Therefore, computing the belief in every node in a singly connected network is linear in the number of nodes or arcs, exponential in the number of parents per node, and linear in the number of children per node.

In Chapter 6, I present a Common Lisp implementation of the abstract polytree algorithm and report a number of experiments on networks of up to 15000 nodes.

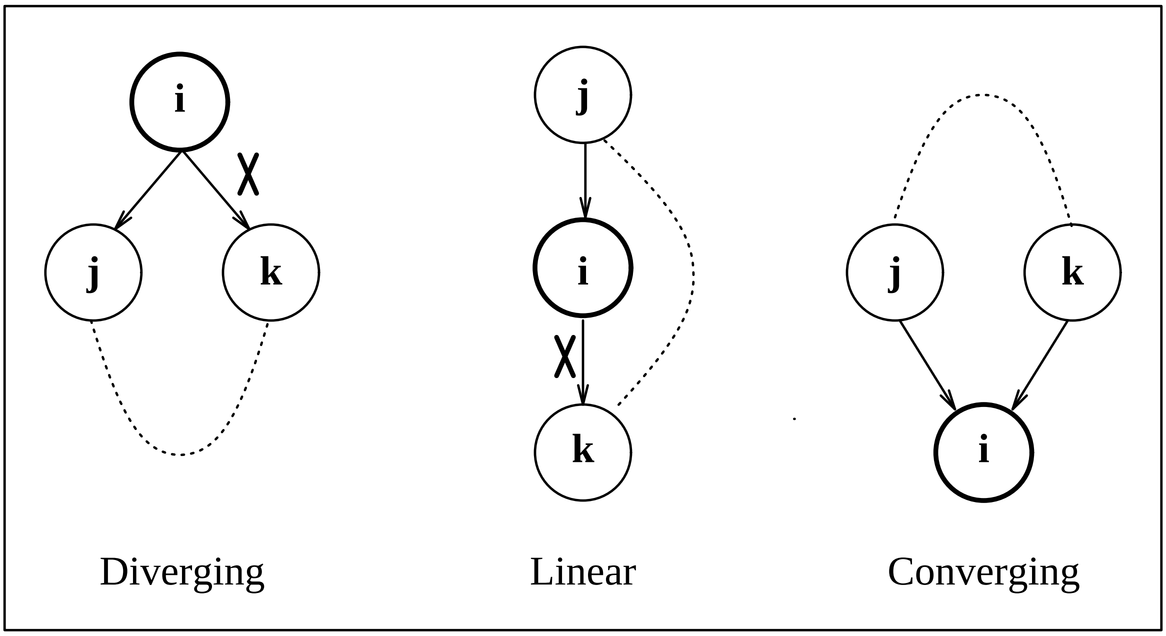

Figure 18: Nodes in a loop could be diverging, linear, or converging. Observing a diverging or a linear node i and deleting Arc $i \rightarrow k$ breaks the loop.

5.6 Multiply connected networks

The abstract polytree algorithm is based on independences that are not necessarily asserted by multiply connected networks. Therefore, the abstract polytree algorithm is not applicable to multiply connected networks in general.

However, the independences justifying the abstract polytree algorithm become asserted by a multiply connected network if a selected set of its nodes are observed. This set of nodes is called the *loop–cutset.* When the loop–cutset is observed, the abstract polytree algorithm becomes applicable. But even when the loop–cutset is not observed, the abstract polytree algorithm can still be made applicable to multiply connected network. There are two steps involved here.

The first step is to identify the loop–cutset, which consists of a node from each loop (undirected cycle) in the causal network. The node chosen from each loop must be either diverging or linear according to Figure 18. The reason for this choice is that after observing such a node, deleting one of its outgoing arcs does not change the independences asserted by the network. Consider, for example, the diverging node i in the left loop of Figure 18. Deleting the arc $i \rightarrow k$ eliminates the loop without changing the independences asserted by the network. To see why, recall that the test of d–separation detects an independence only if it cannot find certain O–active paths, where O is the set of observed nodes (see Definition 4.4.8). When node i belongs to O, no path that includes the arc $i \rightarrow k$ can be an O–active path (see Definition 4.4.7). Therefore, deleting the arc $i \rightarrow k$ does not affect the test of d–separation when node i is observed.

After identifying the loop–cutset LC and deleting arcs as suggested above, the multiply connected network becomes singly connected. Next, to justify arc deletion, we assume a certain state $\underline{LC}$ of the loop–cutset, and apply the polytree algorithm to compute the belief

$$\langle \Phi(i \wedge \delta \wedge LC), \Phi(\neg i \wedge \delta \wedge LC) \rangle$$

in every node of the singly connected network. If we do this for every possible state of the loop–cutset, we obtain the beliefs we want:

$$\langle \Phi(i \wedge \delta), \Phi(\neg i \wedge \delta) \rangle = \bigoplus_{\underline{LC}} \langle \Phi(i \wedge \delta \wedge \underline{LC}), \Phi(\neg i \wedge \delta \wedge \underline{LC}) \rangle.$$

This technique is called the method of *conditioning* [Horvitz *et al.*, 1989; Pearl, 1988; Suermondt and Cooper, 1988; Peot and Shachter, 1991]. When the loop–cutset has n nodes, the method of conditioning requires 2^n applications of the abstract polytree algorithm. This should not be surprising, however, because the computation of belief in probabilistic causal networks is known to be NP–hard [Cooper, 1990].

Chapter 6

Implementing the Abstract Polytree Algorithm

In this chapter, I present a Common Lisp program that uses forward propagation to compute the belief in every node of a singly connected causal network.

6.1 Introduction

The probabilistic polytree algorithm is based on the following message–passing abstraction. Each node i in a causal network can be viewed as a processor that has the following properties:

1. It knows the conditional belief in node i given its parents.

2. It is responsible for computing the belief in node i.

3. It is responsible for sending messages to the neighbors of node i.

This abstraction is useful for describing the algorithm and for implementing it. I shall adopt the same abstraction in implementing a forward propagation version of the abstract polytree algorithm.

A number of components are needed to realize this implementation:

- A representation of support structures. The code for this representation is given in Section 6.2.
- A representation of causal networks. The code for this representation is given in Section 6.3.
- A component to compute causal support, diagnostic support, and belief in a node given the messages it receives from neighbors. The component also computes the message that a node sends to each neighbor. The code for this component is given in Section 6.4.
- A component to decide when a node should perform a particular computation or send a message. The code for this component is given in Section 6.5.
- A component to create causal networks, declare observations, and activate forward propagation. The code for this component is given in Section 6.6.
- A component that represents concrete support structures. The code for this component is given in Section 6.7.

Section 6.8 reports a number of forward propagations in causal networks that are generated randomly and contain between 100 and 15000 nodes.

The Lisp program I shall present is referred to as CNETS.

6.2 Support structures

CNETS represents a support structure as a tuple of seven elements:

$$\langle \oplus, \otimes, \oslash, \mathbf{0}, \mathbf{1}, \preceq_{\oplus}, =_{\oplus} \rangle.$$

Although one needs only the summation and scaling functions to define a support structure, CNETS assumes that all components are provided. Moreover, as shown below, the representation of a support structure does not explicate the set of supports.

```
(defstruct support-structure
  support-summation
  support-unscaling
  support-scaling
  zero-support
  full-support
  support<=
  support=)
```

The operations of a support structure are defined next.

The following function returns a $\oplus$ b:

```
(defun support-summation (a b &optional (ss *CURRENT-SS*))
  (funcall (support-structure-support-summation ss) a b))
```

The following function returns a $\otimes$ b:

```
(defun support-unscaling (a b &optional (ss *CURRENT-SS*))
  (funcall (support-structure-support-unscaling ss) a b))
```

The following function returns a $\oslash$ b:

```
(defun support-scaling (a b &optional (ss *CURRENT-SS*))
  (funcall (support-structure-support-scaling ss) a b))
```

The following function returns $\mathbf{0}$:

```
(defun zero-support (&optional (ss *CURRENT-SS*))
  (support-structure-zero-support ss))
```

The following function returns **1**:

```
(defun full-support (&optional (ss *CURRENT-SS*))
  (support-structure-full-support ss))
```

The following function tests for `a` $\preceq_{\oplus}$ `b`:

```
(defun support<= (a b &optional (ss *CURRENT-SS*))
  (funcall (support-structure-support<= ss) a b))
```

The following function tests for `a` $=_{\oplus}$ `b`:

```
(defun support= (a b &optional (ss *CURRENT-SS*))
  (funcall (support-structure-support= ss) a b))
```

6.3 Nodes and networks

CNETS represents causal networks and nodes as CLOS (Common Lisp Object System) objects. This section provides the definitions of these objects and some basic operations on them.

A causal network is an object with three slots:

1. **name** is an identifier of the network.

2. **nodes** is a list of the nodes in the network.

3. **support-structure** is the structure with respect to which the network is quantified.

```
(defvar *CURRENT-NETWORK* nil)
(defclass network ()
  ((name :initarg :name :reader network-name)
   (nodes :initform nil :accessor network-nodes)
   (support-structure :initform *probability-support-structure*
                      :accessor network-support-structure)))
```

A node is an object with identifier **name**:

```
(defclass node ()
    ((name :initarg :name :reader node-name)))
```

There are two subclasses of the class of nodes, the subclass of observed nodes and the subclass of non–observed nodes. Observed nodes are not part of the causal network defined by a CNETS user but are added by CNETS to simulate observations. An observed node has two slots:

1. **parent** is the node about which the observation is recorded.

2. **observation** is set to 0 if **parent** is observed to be false, and is set to 1 if **parent** is observed to be true.

```
(defclass observed-node (node)
  ((parent :initarg :parent :accessor node-parent)
   (observation :initarg :observation :accessor node-observation)))
```

A non–observed node has ten slots. If the node is denoted by N, and has n parents and m children, then

1. `parents` is a list of parent nodes. In the list $(P_{n-1}P_{n-2}\dots P_0)$, parent P_i has position i. That is, in counting positions, one starts from the far right and moves to the left.

2. `children` is a list of children nodes. In the list $(C_{m-1}C_{m-2}\dots C_0)$, child C_i has position i. That is, in counting positions, one starts from the far right and moves to the left.

3. `parent-messages` is a list of pairs. The pair at position i in `parent-messages` represents the message from parent P_i to node N:
$$\langle \pi_{P_i.N}(\neg P_i), \pi_{P_i.N}(P_i)\rangle .$$

4. `child-messages` is a list of pairs. The pair at position i in `child-messages` represents the message from child C_i to node N:
$$\langle \lambda_{C_i.N}(\neg N), \lambda_{C_i.N}(N)\rangle .$$

5. `cond-supports` is a list of 2^n pairs. Each pair represents a conditional belief
$$\alpha = \left\langle \mathcal{CS}_{\underline{N\diamond}}(\neg N), \mathcal{CS}_{\underline{N\diamond}}(N)\right\rangle ,$$
where $\underline{N\diamond}$ is a state of nodes $N\diamond$, the parents of node N. The state $\underline{N\diamond}$ is determined by the position of the conditional belief α in the list `cond-supports`. In particular, let $b_{n-1}b_{n-2}\dots b_0$ be the binary representation of the position of α in the list `cond-supports`, and let $(P_{n-1}P_{n-2}\dots P_0)$ be the parents of node N. If bit b_i equals 1, then parent P_i is true; otherwise, parent P_i is false.

6. `pstatus` is a bit vector $b_{n-1}b_{n-2}\dots b_0$ such that b_i equals 0 precisely when the parent at position i in `parents` has sent a message to node N.

7. `cstatus` is a bit vector $b_{m-1}b_{m-2}\dots b_0$ such that b_i equals 0 precisely when the child at position i in `children` has sent a message to node N.

8. `causal-support` is a pair representing $\langle \pi_N(\neg N), \pi_N(N) \rangle$.

9. `diagnostic-support` is a pair representing $\langle \lambda_N(\neg N), \lambda_N(N) \rangle$.

10. `belief` is a pair representing $\langle BL_N(\neg N), BL_N(N) \rangle$.

```
(defclass non-observed-node (node)
  ((parents :initform nil :accessor node-parents)
   (children :initform nil :accessor node-children)
   (parent-messages :initform nil :accessor node-parent-messages)
   (child-messages :initform nil :accessor node-child-messages)
   (cond-supports :initform nil :accessor node-cond-supports)
   (pstatus :initform nil :accessor node-pstatus)
   (cstatus :initform nil :accessor node-cstatus)
   (causal-support :initform nil :accessor node-causal-support)
   (diagnostic-support :initform nil :accessor node-diagnostic-support)
   (belief :initform nil :accessor node-belief)))
```

The function `add-node` inserts `node` in `network`:

```
(defmacro add-node (node &optional (network *CURRENT-NETWORK*))
  `(push ,node (network-nodes ,network)))
```

The function `get-node` retrieves a node with identifier `name` from `network`.

```
(defmacro get-node (name &optional (network *CURRENT-NETWORK*))
  `(find-if #'(lambda (n) (equal ,name (node-name n)))
    (network-nodes ,network)))
```

6.3.1 Operations on parents, children, and messages

There are two basic operations on these lists. The first is to retrieve a parent, a child, a parent–message, or a child–message, at a given position in a list. This is achieved by the function `relt`, which is like the function `elt` except that it counts positions from right to left.

```
(defmacro relt (list position)
  `(elt ,list (1- (- (length ,list) ,position))))
```

For example, `(relt '(a b c d e) 1)` is `d`.

The second set of operations find the position of a node with respect to one of its neighbors. The function call `(c-index c p)` returns the position of child `c` with respect to its parent. The function call `(p-index p c)` returns the position of parent `p` with respect to its child `c`.

```
(defmacro rposition (e list) `(position e (reverse list)))
(defmacro c-index (c p) `(rposition ,c (node-children ,p)))
(defmacro p-index (p c) `(rposition ,p (node-parents ,c)))
```

6.3.2 Operations on bit vectors

There are two basic operations on these vectors. The function call `(clear-bit bv bi)` clears the bit at position `bi` in vector `bv`. This function assumes that bit `bi` is already set.

```
(defmacro clear-bit (bv bi)
  `(decf ,bv (expt 2 ,bi)))
```

This function is used to keep track of neighbors that send messages: when a parent sends a message, its corresponding bit in `pstatus` is cleared, and when a child sends a message, its corresponding bit in `cstatus` is cleared.

The function call `(bit-index bv)` assumes that vector `bv` has only one bit set. The call returns the position of the set bit in this case.

```
(defmacro bit-index (bv)
  `(round (log ,bv 2)))
```

This function is used to retrieve the position of the only parent or child that did not yet send a message. Both functions will be put into use in Section 6.5, which deals with controlling the flow of messages in forward propagation.

6.3.3 Operations on pairs of support

The following are standard operations on pairs:

```
(defun unscale-pairs (pair1 pair2)
  (mapcar #'support-unscaling pair1 pair2))

(defun normalize-pair (pair)
  (let ((sum (apply #'support-summation pair)))
    (loop for s in pair
collect (support-scaling s sum))))
```

6.3.4 Propagation initiators

Certain classes of nodes are of special interest because they get forward propagation started. These are nodes that have only a single neighbor. They initiate propagation by sending messages to their neighbors. The following functions identify these nodes:

The function `parentless` returns `t` if node `n` has no parents. It returns `nil` otherwise. The function `childless` works similarly.

```
(defmethod parentless ((n non-observed-node))
  (null (node-parents n)))

(defmethod childless ((n non-observed-node))
  (null (node-children n)))
```

The function `single-child-parentless` returns the child of node `n` if `n` has no parents and only a single child. It returns `nil` otherwise. The function `single-parent-childless` works similarly.

```
(defmethod single-child-parentless ((n non-observed-node))
  (let ((children (node-children n)))
    (when (and (parentless n)
       (= 1 (length children)))
      (relt children 0))))
(defmethod single-child-parentless ((n observed-node)) nil)
```

```
(defmethod single-parent-childless ((n non-observed-node))
  (let ((parents (node-parents n)))
    (when (and (childless n)
	       (= 1 (length parents)))
      (relt parents 0))))
(defmethod single-parent-childless ((n observed-node))
  (node-parent n))
```

6.4 Node computations

Each node in a causal network computes five pairs of support:

1. Causal support.
2. Diagnostic support.
3. Belief.
4. Support to each child.
5. Support to each parent.

Section 6.4.2 provides one function for each computation. Each function is passed the parameters required by the corresponding computation as suggested by the equations given in Section 5.3.6 of Chapter 5. The next section provides a construct that is basic to the functions in Section 6.4.2.

6.4.1 Operating over

The function `operate-over` computes the following expression:

$$\mathtt{OP}_{\substack{0\le i<\mathbf{bound}\\ \mathtt{condition}(i)}}\ \mathtt{expression}(i),$$

where `OP` is a commutative and associative operation that has an identity element `identity`. If `condition` is `nil`, then `operate-over` computes

$$\mathtt{OP}_{0\le i<\mathbf{bound}}\ \mathtt{expression}(i).$$

```
(defun operate-over
    (OP identity bound expression &optional (condition nil))
  (reduce OP
  (loop for state from 0 below bound
      when (or (not condition) (funcall condition state))
      collect (funcall expression state))
  :initial-value identity))
```

The function `sum-over` computes the following expression:

$$\bigoplus_{\substack{0 \leq i < \texttt{bound} \\ \texttt{condition}(i)}} \texttt{expression}(i),$$

where the value of `expression`(i) is a degree of support.

```
(defmacro sum-over (bound expression &optional (condition nil))
  `(operate-over #'support-summation
                 (zero-support)
                 ,bound
                 ,expression
                 ,condition))
```

The function `unscale-over` computes the following expression:

$$\bigotimes_{\substack{0 \leq i < \texttt{bound} \\ \texttt{condition}(i)}} \texttt{expression}(i),$$

where the value of `expression`(i) is a degree of support.

```
(defmacro unscale-over (bound expression &optional (condition nil))
  `(operate-over #'support-unscaling
                 (full-support)
                 ,bound
                 ,expression
                 ,condition))
```

The function `unscale-pairs-over` computes the following expression:

$$\bigotimes_{\substack{0 \leq i < \texttt{bound} \\ \texttt{condition}(i)}} \texttt{expression}(i),$$

where the value of `expression`(i) is a pair of supports.

```
(defmacro unscale-pairs-over (bound expression &optional (condition nil))
  `(operate-over #'unscale-pairs
                 (list (full-support) (full-support))
                 ,bound
                 ,expression
                 ,condition))
```

6.4.2 The computation functions

The function `causal-support` is a direct implementation of Equation 7 of Section 5.3.6.

```
(defun causal-support (parents# p-messages cond-supports)
  (flet ((state-support (i-state)
          (sum-over (expt 2 parents#)
            #'(lambda (p-state)
                (support-unscaling
                 (elt (elt cond-supports p-state) i-state)
                 (unscale-over parents#
                  #'(lambda (j-index)
                      (elt (relt p-messages j-index)
                           (bit-state p-state j-index)))))))))
    (list (state-support 0) (state-support 1))))
(defmacro bit-state (bv bi)
  `(ldb (byte 1 ,bi) ,bv))
```

The function call `(bit-state p-state j-index)` returns the state of bit `j-index` in the bit vector `p-state`.

The function `diagnostic-support` is a direct implementation of Equation 4 of Section 5.3.6.

```
(defun diagnostic-support (children-no c-messages)
  (unscale-pairs-over children-no
   #'(lambda (k-index) (relt c-messages k-index))))
```

The function `belief` is a direct implementation of Equation 5 of Section 5.3.6.

```
(defun belief (causal-support diagnostic-support)
  (normalize-pair (unscale-pairs causal-support diagnostic-support)))
```

The function `support-to-child` is a direct implementation of Equation 6 of Section 5.3.6.

```
(defun support-to-child (children-no c-index c-messages causal-support)
  (unscale-pairs
   causal-support
   (unscale-pairs-over children-no
    #'(lambda (k-index) (relt c-messages k-index))
    #'(lambda (k-index) (not (equal k-index c-index))))))
```

The function **support-to-parent** is a direct implementation of Equation 8 of Section 5.3.6.

```
(defun support-to-parent
    (parents# j-index p-messages diagnostic-support cond-supports)
  (flet ((state-support (j-state)
           (sum-over 2
            #'(lambda (i-state)
                (support-unscaling
                 (elt diagnostic-support i-state)
                 (sum-over (expt 2 parents#)
                  #'(lambda (state)
                      (support-unscaling
                       (elt (elt cond-supports state) i-state)
                       (unscale-over parents#
                        #'(lambda (k-index)
                            (elt (relt p-messages k-index)
                                 (bit-state state k-index)))
                        #'(lambda (k-index)
                            (not (= k-index j-index))))))
                  #'(lambda (state)
                      (consistent-bit state j-index j-state))))))))
    (list (state-support 0) (state-support 1))))
(defmacro consistent-bit (bv bi bs)
  `(equal (bit-state ,bv ,bi) ,bs))
```

The function call `(consistent-bit state j-index j-state)` checks whether bit `j-index` has the state `j-state` in the bit vector `state`.

6.4.3 The interface to computation functions

This section contains functions that interface with the functions given in the previous section. The first three functions respond to messages that request a computation to be performed. The functions find the data needed to perform this computation, perform the computation, and then save the result:

```
(defmethod compute-belief (n)
  (setf (node-belief n)
    (belief (node-causal-support n)
    (node-diagnostic-support n))))

(defmethod compute-causal-support (n)
  (setf (node-causal-support n)
    (causal-support (length (node-parents n))
    (node-parent-messages n)
    (node-cond-supports n))))

(defmethod compute-diagnostic-support (n)
  (setf (node-diagnostic-support n)
    (diagnostic-support (length (node-children n))
(node-child-messages n))))
```

The next two functions respond to messages that order a node to send a message. The functions find the data needed to send the message and then send it.

```
(defmethod send-support-to-child (p c)
  (setf (relt (node-parent-messages c) (p-index p c))
    (support-to-child (length (node-children p))
      (c-index c p)
      (node-child-messages p)
      (node-causal-support p))))

(defmethod send-support-to-parent ((c non-observed-node) p)
  (setf (relt (node-child-messages p) (c-index c p))
    (support-to-parent (length (node-parents c))
       (p-index p c)
       (node-parent-messages c)
       (node-diagnostic-support c)
       (node-cond-supports c))))
```

Finally, the next function is a direct implementation of Equation 9 of Section 5.3.6.

```
(defmethod send-support-to-parent ((c observed-node) p)
  (setf (relt (node-child-messages p) (c-index c p))
    (case (node-observation c)
      ((0) (list (full-support) (zero-support)))
      ((1) (list (zero-support) (full-support))))))
```

6.5 Control flow

A node with n parents and m children is ready to send a message to a neighbor only if it has received messages from all other neighbors. Therefore, a node with $n+m$ neighbors that has sent its first message must also have received $n+m-1$ messages. Once the node has received its final message, it becomes ready to send the remaining $n+m-1$ messages. Therefore, each node sends its messages in two bursts only, one message in the first and $n+m-1$ in the second.

The states of the slots `pstatus` and `cstatus` of the object class `node` are enough to answer the following questions:

1. When and to whom should a node send a message?

2. When should a node compute its belief, causal, and diagnostic supports?

If a node `c` receives a message from parent `p`, it updates the value of `pstatus` by clearing the bit corresponding to `p` in `pstatus`. Moreover, if all bits in `pstatus` are clear, then `c` has received messages from every parent and is ready to compute its causal support. Now, depending on the state of `pstatus` and `cstatus`, the node can take one of three actions:

1. If only one bit is set in `pstatus` and all bits are clear in `cstatus`, then the node has received $n+m-1$ messages and is ready to send its first message. The receiver of the message is the neighbor that did not yet send a message. This happens to be a parent and its position in `parents` is given by `(bit-index pstatus)`.

2. If only one bit is set in `cstatus` and all bits are clear in `pstatus`, then the node has received $n+m-1$ messages and is ready to send its first message. The receiver of the message is the neighbor that did not yet send a message. This happens to be a child and its position in `children` is given by `(bit-index cstatus)`.

3. If all bits are clear in `pstatus` and all bits are clear in `cstatus`, then the node has received its last message and is ready to send messages to $n+m-1$

neighbors: $n-1$ parents and m children. The neighbor that does not get a message is the parent p.

The above logic is captured by the following function, which is called directly after node p sends a message to node c:

```
(defmethod send-support-to-child :after (p c)
  (let* ((parents (node-parents c))
         (children (node-children c))
         (pstatus (clear-bit (node-pstatus c) (p-index p c)))
         (cstatus (node-cstatus c))
         (pstatus-count (logcount pstatus))
         (cstatus-count (logcount cstatus)))
    (when (= pstatus-count 0) (compute-causal-support c))
    (cond ((and (= pstatus-count 1) (= cstatus-count 0))
           (send-support-to-parent c (relt parents (bit-index pstatus))))
          ((and (= pstatus-count 0) (= cstatus-count 1))
           (let ((cc (relt children (bit-index cstatus))))
             (unless (typep cc 'observed-node)
               (send-support-to-child c cc))))
          ((and (= pstatus-count 0) (= cstatus-count 0))
           (compute-belief c)
           (loop for cc in children
               unless (typep cc 'observed-node)
               do (send-support-to-child c cc))
           (loop for pc in parents
               unless (equal pc p)
               do (send-support-to-parent c pc))))))
```

The following function is called directly after node `c` sends a message to its parent `p`. The function implements a logic that is similar to the one implemented by `send-support-to-child` given above.

```
(defmethod send-support-to-parent :after ((c node) p)
  (let* ((parents (node-parents p))
         (children (node-children p))
         (pstatus (node-pstatus p))
         (cstatus (clear-bit (node-cstatus p) (c-index c p)))
         (pstatus-count (logcount pstatus))
         (cstatus-count (logcount cstatus)))
    (when (= cstatus-count 0) (compute-diagnostic-support p))
    (cond ((and (= pstatus-count 1) (= cstatus-count 0))
            (send-support-to-parent p (relt parents (bit-index pstatus))))
          ((and (= pstatus-count 0) (= cstatus-count 1))
            (let ((pc (relt children (bit-index cstatus))))
              (unless (typep pc 'observed-node)
                (send-support-to-child p pc))))
          ((and (= pstatus-count 0) (= cstatus-count 0))
            (compute-belief p)
            (loop for pc in children
                unless (or (typep pc 'observed-node) (equal pc c))
                do (send-support-to-child p pc))
            (loop for pp in parents do (send-support-to-parent p pp))))))
```

6.6 Interface

The user of CNETS does not need to know about the code presented so far. From the user point of view, the following operations need to be supported by CNETS:

1. Create a causal network.

2. Declare an observation.

3. Activate forward propagation.

The following sections provide functions to support these operations. These functions are all the user is expected to know in order to use CNETS for computing beliefs.

6.6.1 Creating a network

To create a causal network, one needs to do the following:

1. Create an empty network.

2. Add nodes to the network.

3. Declare the parents of each node and provide the node's conditional support function.

4. Declare observations.

The function **make-network** creates and returns a causal network with identifier **name**:

```
(defun make-network (name)
  (make-instance 'network :name name))
```

The function **make-node** creates and returns a node with identifier **name**. It also adds the node to **network**.

```
(defun make-node (name &optional (network *CURRENT-NETWORK*))
  (add-node (make-instance 'non-observed-node :name name) network))
```

The function **make-parents** declares that a node with identifier **name** has parents **p-names**, which is a list of node identifiers. It also declares that **cond-supports** is the conditional support function for the node with identifier **name**. The parameters **cond-supports** and **p-names** are closely related because **cond-supports** is interpreted with respect to the order of parents in the list **p-names**.

```
(defun make-parents
    (name p-names cond-supports &optional (network *CURRENT-NETWORK*))
  (let ((node (get-node name network)))
    (setf (node-parents node)
      (loop for p-name in p-names collect (get-node p-name network)))
    (setf (node-cond-supports node) cond-supports)
    (loop for p in (node-parents node) do (push node (node-children p)))))
```

The function **name-observation** declares that the node with identifier **name** is observed to have state **obs**, which is either 0 (false) or 1 (true).

```
(defun make-observation (name obs &optional (network *CURRENT-NETWORK*))
  (let* ((node (get-node name network))
 (obs-name (list 'obs name))
 (obs-node (make-instance 'observed-node
     :name obs-name
     :observation obs
     :parent node)))
    (push obs-node (node-children node))
    (add-node obs-node network)))
```

6.6.2 Propagating messages

The function `activate-network` activates forward propagation in `network`. When this function returns, every node will have computed its belief. The function also prints some information about the time spent in forward propagation:

```
(defun activate-network (&optional (network *CURRENT-NETWORK*))
  (setf *CURRENT-SS* (network-support-structure network))
  (initialize network)
  (time (loop for n in (network-nodes network)
    do (let ((p (single-parent-childless n))
     (c (single-child-parentless n)))
 (when p (send-support-to-parent n p))
 (when c (send-support-to-child n c))))))
```

The function `activate-network` does the following:

1. Sets the current support structure with respect to which the network is quantified.
2. Initializes the network (more on this later on).
3. Identifies single–neighbor nodes and asks them to send messages to their neighbors.

This starts a chain reaction of exchanging messages that ends when the number of messages exchanged is twice the number of arcs in the network.

One initializes a network by initializing its nodes:

```
(defmethod initialize ((nt network))
  (loop for n in (network-nodes nt) do (initialize n)))
```

The initialization of an observed node is trivial.

```
(defmethod initialize ((n observed-node)))
```

A non–observed node is initialized as follows:

1. All the bits in `pstatus` are set to indicate that the node did not receive any message from parents.

2. All the bits in `cstatus` are set to indicate that the node did not receive any message from children.

3. A list of n empty messages (`nil`) is created and stored at `parents-messages`.

4. A list of m empty messages (`nil`) is created and stored at `children-messages`.

5. If a node has no parents, it is asked to compute its causal support; otherwise, the causal support is set to `nil`.

6. If a node has no children, it is asked to compute its diagnostic support; otherwise, the diagnostic support is set to `nil`.

7. The belief of the node is set to `nil`.

```
(defmethod initialize ((n non-observed-node))
  (let ((parents# (length (node-parents n)))
        (children# (length (node-children n))))
    (setf (node-pstatus n) (1- (expt 2 parents#)))
    (setf (node-cstatus n) (1- (expt 2 children#)))
    (setf (node-parent-messages n)
      (make-list parents# :initial-element nil))
    (setf (node-child-messages n)
      (make-list children# :initial-element nil))
    (if (parentless n)
(compute-causal-support n)
      (setf (node-causal-support n) nil))
    (if (childless n)
(compute-diagnostic-support n)
      (setf (node-diagnostic-support n) nil))
    (setf (node-belief n) nil)))
```

6.7 Concrete support structures

In this section, three support structures are defined.

The first is the support structure of propositional calculus, which has only two degrees of support `nil` and `t`, which represent the zero and full supports, respectively.

```
(defparameter *binary-support-structure*
    (make-support-structure
     :support-summation #'(lambda (x y) (or x y))
     :support-unscaling #'(lambda (x y) (and x y))
     :support-scaling #'(lambda (x ?) (declare (ignore ?)) x)
     :zero-support nil
     :full-support t
     :support<= #'(lambda (x y) (unless (and x (not y))))
     :support= #'(lambda (x y) (equal x y))))
```

The second is the support structure of probability calculus, which has the degrees of support $[0, 1]$.

```
(defparameter *probability-support-structure*
    (make-support-structure
     :support-summation #'+
     :support-unscaling #'*
     :support-scaling #'/
     :zero-support 0
     :full-support 1
     :support<= #'<=
     :support= #'=))
(defvar *CURRENT-SS* *probability-support-structure*)
```

The third and final is the support structure of impossibility calculus, which has the degrees of support $\{0, 1, \ldots, \infty\}$. Here, ∞ is represented by `#.EXCL::*INFINITY-DOUBLE*`, which behaves like infinity in Allegro CL.[1]

[1] The variable `.EXCL::*INFINITY-DOUBLE*` is not part of Common Lisp, which does not have an explicit representation of infinity.

```
(defparameter *disbelief-support-structure*
    (make-support-structure
     :support-summation #'min
     :support-unscaling #'+
     :support-scaling #'-
     :zero-support #.EXCL::*INFINITY-DOUBLE*
     :full-support 0
     :support<= #'<=
     :support= #'=))
```

6.8 Experiments

This section contains a number of experiments involving CNETS. The presented networks are generated randomly by a program that accepts the following as input:

1. The maximal number of parents per node.
2. The maximal number of children per node.
3. The radius of the network.

The number of arcs in a singly connected network equals the number of nodes minus one. Therefore, it is not unreasonable to capture the size of a network by the following parameters:

1. The number of nodes, which determines the number of messages exchanged in forward propagation.
2. The radius of the network, which determines the number of stages in forward propagation.
3. The number of parents per node.

The number of children per node is ignored because it is dominated by the number of parents as suggested in Section 5.5 of Chapter 5.

Table 5 depicts a number of networks and the time it took to complete forward propagation using CNETS. The networks have between 100 and 15000 nodes. Two computational times are reported: user CPU time without garbage collection and real time. All these experiments are with respect to a probabilistic support structure.

The following observations are about Table 5:

1. The CPU time per node is constant for a given number of parents per node as suggested by Figures 19, 20, and 21.
2. The CPU time per node grows exponentially in the number of parents per node as suggested by Figure 22.

Run#	Nodes#	Radius	Maximum parents#	Time (msec) CPU	Real	Time (msec)/Node CPU
1	100	3	3	749	1267	7.5
2	164	3	4	1550	2490	9.5
3	294	3	5	3484	4473	11.9
4	402	3	6	9784	12601	24.3
5	463	3	7	20801	25756	44.9
6	348	3	8	38367	42004	110.3
7	347	4	3	2300	3462	6.6
8	218	4	4	2167	3815	9.9
9	443	4	5	6100	7314	13.8
10	2140	4	6	50066	62327	23.4
11	3707	4	7	160167	197802	43.2
12	797	4	8	82433	88823	103.4
13	1104	5	3	7334	10557	6.6
14	1834	5	4	16383	22592	8.9
15	3120	5	5	42666	56491	13.7
16	4250	5	6	104933	134615	24.7
17	15072	5	7	689567	954235	45.8
18	10469	5	8	954916	1195478	91.2

Table 5: Experiments using CNETS on randomly created probabilistic causal networks.

3. It took approximately one second to compute the belief in every node of a network that has 100 nodes, with 3 parents per node at most.

4. It took approximately 15 minutes to compute the belief in every node of a network that has 15070 nodes, with 7 parents per node at most.

These observations support the formal analysis of Chapter 5.

Table 6 compares the performance of CNETS with the performance of IDEAL, a system for Influence Diagram Evaluation and Analysis in Lisp [Srinivas and Breese, 1992]. The table shows CNETS to be at least two times faster than IDEAL on five randomly created causal networks.

Run#	Nodes#	Radius	Maximum parents#	CNETS CPU Time (sec)	IDEAL CPU Time (sec)	IDEAL / CNETS
1	100	3	3	.7	1.8	2.6
2	158	3	8	13.8	35.7	2.6
3	353	4	3	2.6	6.4	2.5
4	1281	4	8	107.6	278	2.6
5	978	5	3	6.9	19.3	2.8

Table 6: Experiments using CNETS and IDEAL on randomly created probabilistic causal networks. Each node has at most three children.

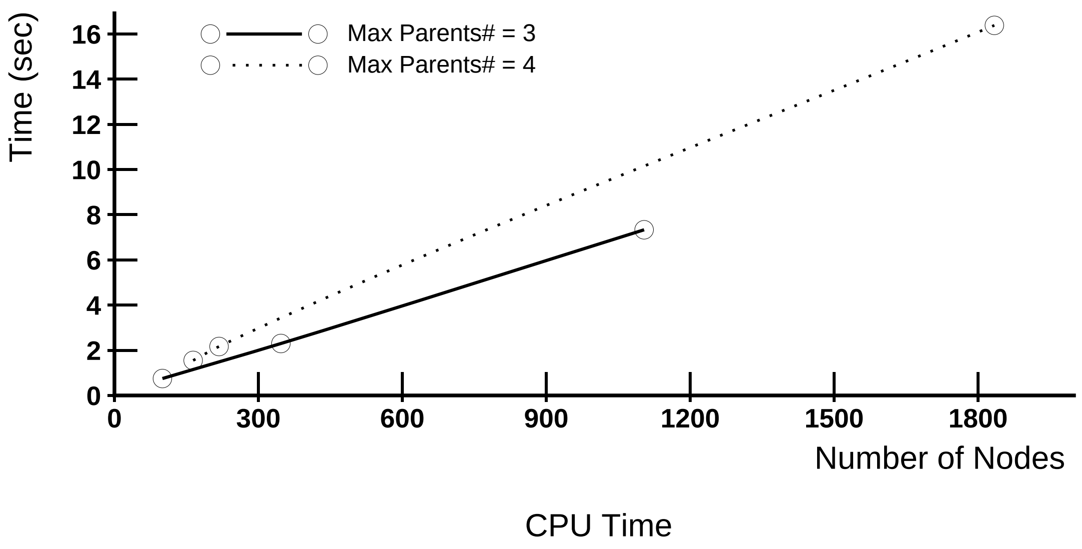

Figure 19: Computation time in singly connected networks of up to 1800 nodes.

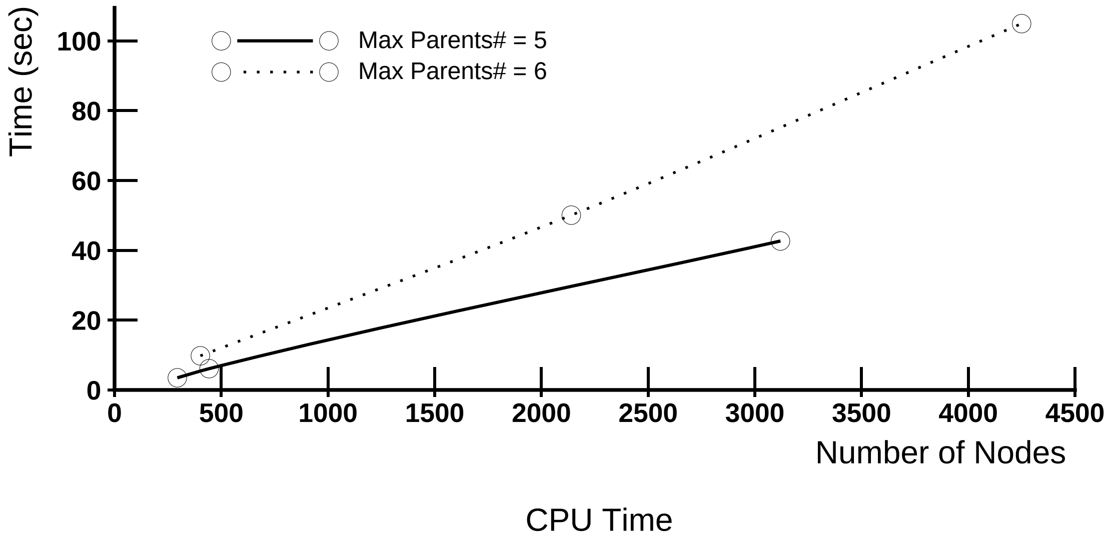

Figure 20: Computation time in singly connected networks of up to 4500 nodes.

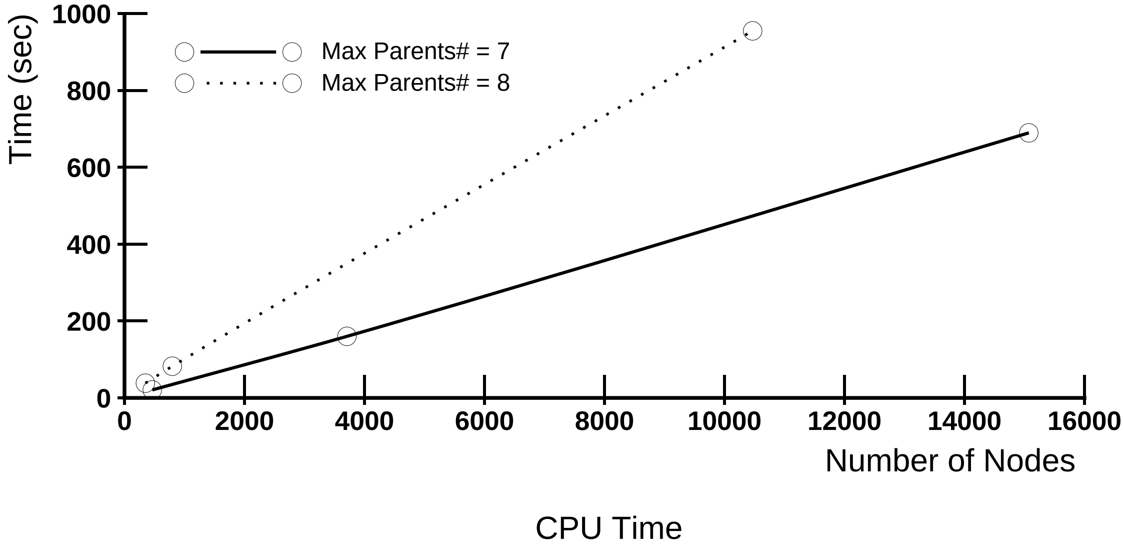

Figure 21: Computation time in singly connected networks of up to 15000 nodes.

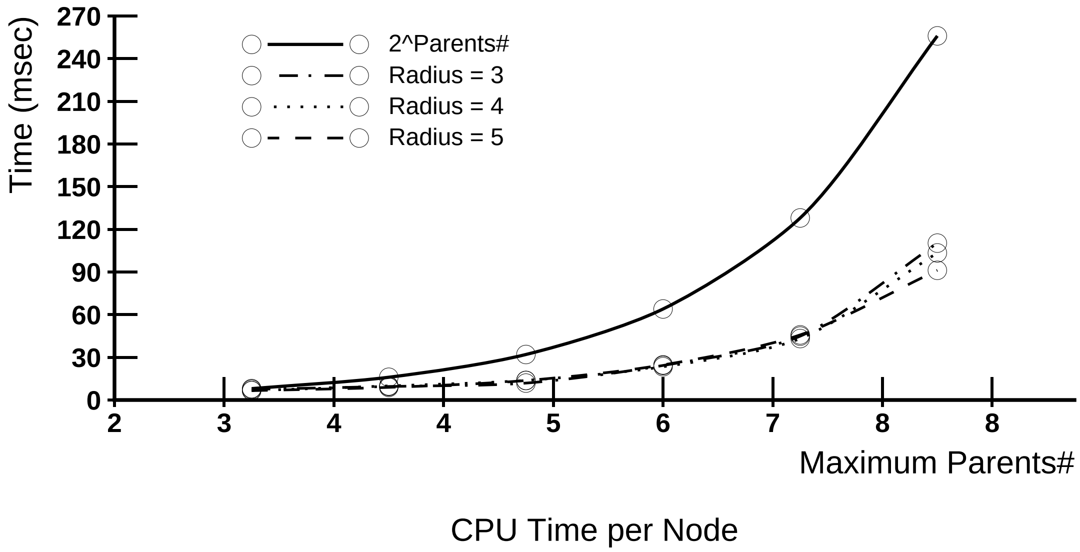

Figure 22: Computation time in singly connected networks of up to eight parents per node.

Chapter 7

Objection Calculus

One way to quantify our support for a sentence is to state the most general reason for rejecting it. The most general reason for rejecting a sentence is called the *objection* to that sentence. When degrees of support are taken to be objections, the resulting support calculus is called *objection calculus.* In this chapter, I introduce objection calculus and discuss its central notions: objection–based states of belief, their conditionalizations, and objection–based causal networks.

7.1 Introduction

Abstract states of belief, their conditionalizations, and abstract causal networks were motivated by the need to relax the commitment to numbers while retaining the key features of probability calculus. The main theoretical value of these notions is that they relax the commitment to numbers. But their practical value is most appreciated when they are instantiated with respect to concrete, non–numeric, and intuitive degrees of support. Such instantiations give birth to concrete calculi for reasoning under uncertainty that could be alternatives to probability calculus in some applications.

The abstract framework developed in Chapters 2–6 mechanizes the construction of concrete calculi for reasoning under uncertainty. Each concrete calculus is characterized by a support structure $\langle \mathcal{S}, \oplus, \oslash \rangle$, which consists of degrees of support $\mathcal{S}$, support summation $\oplus$, and support scaling $\oslash$. Therefore, a calculus is constructed by defining a support structure.

The process of defining a concrete support calculus usually involves the following steps:

1. Characterizing the degrees of support $\mathcal{S}$ by appealing to one's intuition about how to quantify the support for a sentence.

2. Choosing an intuitive definition of support summation $\oplus$, and then verifying the choice by showing that $\langle \mathcal{S}, \oplus \rangle$ is a partial support structure.

3. Choosing an intuitive definition of support scaling $\oslash$, and then verifying the choice by showing that $\langle \mathcal{S}, \oplus, \oslash \rangle$ is a support structure.

Defining a support structure gives birth to a concrete calculus for reasoning under uncertainty that has the following basic elements: a definition of a state of belief, a definition of conditionalization, and a definition of a causal network.

In this chapter, I introduce a concrete calculus, called objection calculus, which results from taking degrees of support to be objections. The objection to a sentence is the most general reason for rejecting that sentence.

Objections are more than just quantifiers of support. They are also reasons for rejecting. This aspect of objections gives them a role outside uncertainty applications. We shall see in Chapter 8 that objections play a central role in diagnosis applications.

The construction of objection calculus in this chapter follows the procedure outlined above. I start by characterizing the set of objections in Section 7.2. I then provide the definition of objection summation in Section 7.3, where I also discuss the consequent notion of objection–based state of belief. In Section 7.4, I provide the definition of objection scaling and discuss objection–based conditionalization. Objection–based independence and causal networks are discussed in Sections 7.5 and 7.6.

7.2 Objections

The objection to a sentence is the most general reason for rejecting that sentence.

I take objections to be sentences in a propositional language, denoted by $\mathcal{O}$, called the *objection language.* A sentence to which an objection is attributed is called an *objectionee.* I take objectionees to be sentences in a propositional language, denoted by $\mathcal{L}$, called the *objectionee language.* Thus, an objection–based state of belief maps each objectionee in $\mathcal{L}$ to an objection in $\mathcal{O}$.

I assume that the holder of an objection–based state of belief is embedded in a world that decides the truth and falsity of objections and objectionees. Although the holder can observe objectionees, I assume that she cannot observe objections. The reason for this assumption is technical. According to the formalization of abstract states of belief in Chapter 2, objections are quantities. And from the viewpoint of this formalization, it is not meaningful to observe quantities. For the same reason, I assume that the primitive propositions of the objectionee language are disjoint from those of the objection language.

But how can objections play the role of quantities?

The basic intuition here is that an objectionee is rejected if its objection holds in the world. Therefore, the strongest objections are valid sentences in $\mathcal{O}$ because their objectionees are rejected in any state of the world. Moreover, the weakest objections are unsatisfiable sentences in $\mathcal{O}$, because their objectionees are not rejected in any state of the world. Between these two extremes, there are objections that are neither valid nor unsatisfiable. For these objections, the rejection of objectionees depends on the state of the world embedding an objection–based state of belief. Therefore, assuming that every state of the world is equally likely, the rejectability of objectionees is quantified by the logical strength of their objections. For example, an objectionee is no more rejectable than another objectionee if its objection logically entails the objection to the other objectionee. This is how objections play the role of quantities.

7.3 Objection summation

After choosing the degrees of support for a support calculus, we must decide on how to sum these supports. The question to ask is the following:

> If α is the support for sentence A, if β is the support for sentence B, and if A and B are logically disjoint, then what would be the support for $A \vee B$?

According to the formalization of abstract states of belief, the support for $A \vee B$ should be $\alpha \oplus \beta$. Therefore, the answer to the previous question defines support summation.

For a given pair of supports (α, β), it is not always possible to find a state of belief that attributes α and β to logically disjoint sentences. For example, if degrees of supports are frequencies in the interval $[0, 1]$, then there is no state of belief that attributes the frequencies .6 and .9 to logically disjoint sentences. Therefore, before we ask the above question, we need to identify the pairs of supports about which it is meaningful to ask the question. Identifying these pairs amounts to defining the domain of support summation.

In objection calculus, the domain of objection summation is the Cartesian product $\mathcal{O} \times \mathcal{O}$, because objections to logically disjoint objectionees are not related unless the objectionees are also logically exhaustive. In this case, the objections must contradict each other because, otherwise, the state of belief may reject the objectionees simultaneously. For example, consider the exhaustive objectionees, "Tweety flies" and "Tweety does not fly," and their respective objections, "Tweety is wingless or sick," and "Tweety is a bird." The two objections here do not contradict each other, which is a problem. If "Tweety is a wingless or sick bird" holds in the world, then "Tweety flies" and "Tweety does not fly" are both rejected.

To define objection summation, we ask the following:

> If α is the most general reason for rejecting A, if β is the most general reason for rejecting B, and if A and B are logically disjoint, then what is the most general reason for rejecting $A \vee B$?

The answer is clearly $\alpha \wedge \beta$, because $A \vee B$ should be rejected precisely when A is rejected and B is rejected. Therefore, objection summation is logical conjunction. The following theorem verifies this definition:

Theorem 7.3.1 *The pair $\langle \mathcal{O}, \wedge \rangle$ is a partial support structure.*

We know from Chapter 2 that every support summation function induces a partial order on degrees of support. Objection summation induces the following order:

Definition 7.3.2 *Objection α is no greater than β precisely when there is an objection γ such that $\alpha \wedge \gamma \equiv \beta$.*

Therefore, objection α is no greater than β precisely when $\beta \models \alpha$. Moreover, objection α equals objection β precisely when $\alpha \equiv \beta$. Unsatisfiable sentences in $\mathcal{O}$ are the minimal objections, and valid sentences are the maximal objections.

By defining objection summation, we obtain a number of results. First, we obtain a formal definition of objection–based states of belief, which is given in Section 7.3.1. Second, we obtain a spectrum of attitudes that these states hold towards sentences, which is discussed in Section 7.3.2. Finally, we obtain a measure of the ignorance of objection–based states of belief, which is discussed in Section 7.3.3.

7.3.1 States of belief

The formalization in Chapter 2 provides a definition of a state of belief with respect to every partial support structure. In objection calculus, the definition is the following:

Definition 7.3.3 *An objection–based state of belief Φ with respect to $(\mathcal{L}, \mathcal{O})$ is a mapping from $\mathcal{L}$ to $\mathcal{O}$ satisfying the following conditions:*

1. $\Phi(A) = \Phi(B)$ *if* $\models A \equiv B$.
2. $\Phi(A \vee B) = \Phi(A) \wedge \Phi(B)$ *if* $\models \neg(A \wedge B)$.
3. $\Phi(\mathbf{false}) = \mathbf{true}$.
4. $\Phi(\mathbf{true}) = \mathbf{false}$.

This definition imposes the following conditions on objection–based states of belief:

1. Equivalent sentences should be rejected under equivalent conditions. In probability calculus, for example, the corresponding condition is that equivalent sentences should have equal probabilities.

2. A disjunction of disjoint sentences should be rejected precisely when the disjuncts are rejected. In probability calculus, the probability of a disjunction of disjoint sentences is the summation of the probabilities of the disjuncts.

3. An unsatisfiable sentence should be rejected. In probability calculus, an unsatisfiable sentence should have probability zero.

4. A valid sentence should not be rejected. In probability calculus, a valid sentence should have probability one.

Objection summation is idempotent, that is, $\alpha \wedge \alpha = \alpha$. As a consequence of this property, the objection to $A \vee B$ is the objection to A conjoined with the objection to B, even when A and B are not logically disjoint. This facilitates the computation of objections.

7.3.2 Attitudes

Objection–based states of belief hold absolute or relative attitudes towards sentences. There are two classes of absolute attitudes. The first has the form, "I reject A under α," or "I accept A under α," and is defined as follows:

Definition 7.3.4 *An objection–based state of belief Φ <u>rejects A under α</u> precisely when $\alpha \models \Phi(A)$. Moreover, Φ <u>accepts A under α</u> precisely when it rejects $\neg A$ under α.*

For example, if my objection to "Tweety flies" is "Tweety is an elephant," then I reject "Tweety flies" and accept "Tweety does not fly" under "Tweety is a white elephant."

The second class of absolute attitudes has the form, "I can reject A under α," or "I can accept A under α," and is defined as follows:

Definition 7.3.5 *An objection–based state of belief Φ can reject A under α precisely when $\alpha \not\models \neg\Phi(A)$. Moreover, Φ can accept A under α precisely when it can reject $\neg A$ under α.*

For example, if the objection to "Tweety flies" is "Tweety is wingless or sick," then "Tweety flies" can be rejected under "Tweety is not wingless," but it cannot be rejected under "Tweety is neither wingless nor sick."

Objection–based states of belief could hold relative attitudes of the form, "I find A no more rejectable than B," or "I find A no more acceptable than B." These attitudes are defined as follows.

Definition 7.3.6 *An objection–based state of belief Φ finds A no more rejectable than B precisely when $\Phi(A) \models \Phi(B)$. Moreover, Φ finds A no more acceptable than B precisely when it finds $\neg A$ is no more rejectable than $\neg B$.*

Therefore, sentence A is no more rejectable than B precisely when A is rejected only if B is rejected. Similarly, A is no more acceptable than B precisely when A is accepted only if B is accepted.

The formalization of abstract states of belief suggests definitions of the attitudes of rejection and acceptance. Viewing objection–based states of belief as abstract states of belief, we have the following equivalences:

Theorem 7.3.7 *An objection–based state of belief rejects A according to Definition 2.4.4 precisely when it rejects A under* **true** *according to Definition 7.3.4. Moreover, it accepts A according to Definition 2.4.3 precisely when it accepts A under* **true** *according to Definition 7.3.4.* ∎

Therefore, rejection and acceptance of sentences by abstract states of belief are the extreme attitudes held by objection–based states of belief.

The formalization of abstract states of belief also suggests definitions of the orders no–more–supported and no–more–believed. Viewing objection–based states of belief as abstract states of belief, we have the following equivalences:

$\mathcal{L}$	$\mathcal{O}$	$\mathcal{L}$	$\mathcal{O}$
$bird \wedge fly$	$\neg normal$	$bird$	**false**
$bird \wedge \neg fly$	$normal$	$\neg bird$	**false**
$\neg bird \wedge fly$	**true**	fly	$\neg normal$
$\neg bird \wedge \neg fly$	**false**	$\neg fly$	**false**

Table 7: An objection–based state of belief. Here, *bird* is equally acceptable to *fly* because the objection to $\neg bird$ is equivalent to the objection to $\neg fly$. Note, however, that *fly* is more rejectable than *bird* because the objection to *bird* strictly entails the objection to *fly*.

Theorem 7.3.8 *An objection–based state of belief* Φ *supports* A *no more than it supports* B *(*$A \preceq_\Phi B$*) according to Definition 2.5.5 precisely when it finds* B *no more rejectable than* A*.* ∎

Theorem 7.3.9 *An objection–based state of belief* Φ *believes* A *no more than it believes* B *(*$A \sqsubseteq_\Phi B$*) according to Definition 2.5.6 precisely when it finds* B *no more rejectable than* A *and finds* A *no more acceptable than* B*.* ∎

The two conditions in this theorem may seem redundant, but they are not. For example, A and B might be equally acceptable, but B might be more rejectable than A (see Table 7).

7.3.3 Ignorance

By definition of a state of belief, the objection to a sentence and that to its negation are constrained as follows:

$$\Phi(A) \wedge \Phi(\neg A) = \mathbf{false}.$$

This constraint says that no objection–based state of belief could reject a sentence and its negation in some state of the world.

Although an objection–based state of belief never rejects a sentence and its negation in some state of the world, it may also reject neither (and, hence, accept neither). As we shall see next, such states of the world are used to indicate the ignorance of a state of belief towards the sentence.

The states of the world in which a state of belief Φ accepts either A or $\neg A$ are those that satisfy the objection $\Phi(\neg A) \vee \Phi(A)$. Under the objection $\Phi(\neg A)$, the state of belief accepts A, and under the objection $\Phi(A)$, it accepts $\neg A$. Therefore, it accepts either A or $\neg A$ in any state of the world that satisfies $\Phi(\neg A) \vee \Phi(A)$. Hence, the state of belief Φ accepts neither A nor $\neg A$ in any state of the world that satisfies $\neg\Phi(\neg A) \wedge \neg\Phi(A)$. When the state of belief is embedded in one of these states, we say that it is ignorant about sentence A.

Definition 7.3.10 *The degree to which an objection–based state of belief Φ is ignorant about sentence A is given by*

$$IG_{\Phi}(A) \stackrel{def}{=} \neg\Phi(\neg A) \wedge \neg\Phi(A).$$

Theorem 7.3.11 $IG_{\Phi}(A) = IG_{\Phi}(\neg A)$. ∎

The degree of ignorance about a sentence characterizes the states of the world in which neither the sentence nor its negation are accepted.

If neither a sentence nor its negation is accepted in any state of the world, then the degree of ignorance about the sentence is **true**. This is the maximal degree of ignorance and we say that the state of belief is maximally ignorant about the sentence in this case.

Theorem 7.3.12 *An objection–based state of belief is maximally ignorant about sentence A precisely when $\Phi(A) = \Phi(\neg A) =$ **false**.* ∎

If either the sentence or its negation is accepted in each state of the world, then the degree of ignorance about the sentence is **false**. This is the minimal degree of ignorance and we say that the state of belief is minimally ignorant about the sentence in this case.

Theorem 7.3.13 *An objection–based state of belief is minimally ignorant about sentence A precisely when $\Phi(A) = \neg\Phi(\neg A)$.* ∎

Objection–based states of belief hold absolute and relative ignorance attitudes towards sentences. The absolute attitudes have the form, "I am ignorant about A under α," and are defined as follows:

Definition 7.3.14 *An objection–based state of belief Φ is* <u>*ignorant about A under α*</u> *precisely when $\alpha \models IG_{\Phi}(A)$.*

If a state of belief is ignorant about sentence A under **true**, then it is maximally ignorant about A. Moreover, if it is ignorant about sentence A only under **false**, then it is minimally ignorant about A.

The relative ignorance attitudes have the form, "I am no more ignorant about A than about B," and are defined as follows:

Definition 7.3.15 *An objection–based state of belief Φ is* <u>*no more ignorant*</u> *about A than about B precisely when $IG_{\Phi}(A) \models IG_{\Phi}(B)$.*

This says that a state of belief is no more ignorant about A than about B precisely when it is ignorant about A only if it is ignorant about B.

7.4 Objection scaling

The last step in constructing a calculus for reasoning under uncertainty is defining the support scaling function $\oslash$. The question to ask is the following:

> If α is the support for A, if β is the support for B, and if A entails B, then what will be the support for A after observing B?

According to Chapter 3, the support for A after observing B should be $\alpha \oslash \beta$. Therefore, the answer to the previous question defines support scaling. This question, however, is meaningful only if α is no greater than β, because A entails B. Otherwise, there will be no state of belief that satisfies the premise of the question.

To define objection scaling, we ask the following question:

> If α is the most general reason for rejecting A, if β is the most general reason for rejecting B, and if A entails B, then what is the most general reason for rejecting A after observing B?

The answer to this question is based on the following:

- The weakest sentence under which one rejects A is the conjunction of
  - The weakest sentence under which one rejects $A \wedge B$.
  - The weakest sentence under which one rejects $A \wedge \neg B$.

  That is, $\Phi(A) = \Phi(A \wedge B) \wedge \Phi(A \wedge \neg B)$.

- The weakest sentence under which one rejects $A \wedge B$ is the disjunction of
  - The weakest sentence under which one rejects B.
  - The weakest sentence under which one rejects $A \wedge B$ but cannot reject B.

  That is, $\Phi(A \wedge B) = \Phi(B) \vee (\Phi(A \wedge B) \wedge \neg\Phi(B))$.

- After observing B,
  - The weakest sentence under which one rejects $A \wedge \neg B$ becomes **true**.
  - The weakest sentence under which one rejects B becomes **false**.

  That is, $\Phi_B(A \wedge \neg B) = \mathbf{true}$ and $\Phi_B(B) = \mathbf{false}$.

Therefore, the weakest sentence under which one rejects A after observing B is the weakest sentence under which one rejects $A \wedge B$ but cannot reject B. That is, $\alpha \wedge \neg\beta$, which says that objection scaling is logical falsification $\wedge\neg$.

Although this seems to be an intuitive definition of objection scaling, it does not conform to the properties of support scaling suggested in Chapter 3. As it turns out, this definition violates Property (Y1), $\mathbf{0} \oslash a = \mathbf{0}$, which says that observing a non–rejected sentence retains all accepted sentences. If we modify the previous definition of objection scaling to account for Property (Y1), we obtain a definition of objection scaling that is given and verified by the following theorem:

Theorem 7.4.1 *The triple $\langle \mathcal{O}, \wedge, \angle \rangle$ is a distributive, but non–bijective, support structure, where*

$$\alpha \angle \beta \stackrel{def}{=} \begin{cases} \mathbf{true}, & \textit{if } \alpha = \mathbf{true}; \\ \alpha \wedge \neg\beta, & \textit{otherwise.} \end{cases}$$

By defining objection scaling, we obtain a number of results. First, a definition of conditionalized states of belief, which is given in Section 7.4.1. Next, the notion of objection–based independence, which is discussed in Section 7.5. Finally, the notion of objection–based causal networks, which is discussed in Section 7.6.

7.4.1 Conditionalized states of belief

Chapter 3 provides a definition of conditionalized abstract states of belief. Objection calculus inherits this definition, which is given below.

Definition 7.4.2 *Let Φ be an objection–based state of belief that does not reject B. The conditionalization of Φ on B is defined as follows:*

$$\Phi_B(A) = \Phi(A \wedge B) \angle \Phi(B).$$

The following theorem is a key to the intuition behind conditional objections:

Theorem 7.4.3 *When a state of belief Φ in objection calculus does not reject $A \wedge B$, the conditional objection to A given B is the weakest sentence under which Φ rejects $A \wedge B$ but cannot reject B.* ∎

Assessing a sentence under which $A \wedge B$ is rejected is usually easy, but assessing the weakest such sentence is usually a challenging task. Moreover, ensuring that B cannot be rejected under such a weakest sentence requires assessing the weakest sentence under which B is rejected. This makes the assessment of conditional objections not always a natural task. This issue is discussed further in Section 7.4.3.

7.4.2 Objection unscaling

Since every support scaling has a support unscaling, one should ask, What is objection unscaling? The answer to this question is suggested by Figure 23, which shows the relation between the objection to a conjunction $A \wedge B$ and the objection to one of the conjuncts B.

According to Figure 23, when $A \wedge B$ is not rejected, the objection to $A \wedge B$ can be computed by disjoining the conditional objection to A given B with the objection to B. This is verified by the following theorem:

Theorem 7.4.4 *Support unscaling of $\langle \mathcal{O}, \wedge, \angle \rangle$ is $\sqcup$, where*

$$\alpha \sqcup \beta \stackrel{def}{=} \alpha \vee \beta \text{ precisely when } (\alpha = \mathbf{true} \text{ or } \alpha \wedge \beta = \mathbf{false}) \text{ and } \beta \neq \mathbf{true}.$$

Corollary 7.4.5 *If Φ is an objection–based state of belief that does not reject B, then*

$$\Phi(A \wedge B) \quad = \quad \Phi_B(A) \sqcup \Phi(B). \blacksquare$$

Corollary 7.4.5 shows that a conditional objection is constrained by the objection to its condition. In particular, when the conditional objection is not **true**, it must be disjoint from the objection to its condition. Therefore, the following two statements are not necessarily consistent in objection calculus:

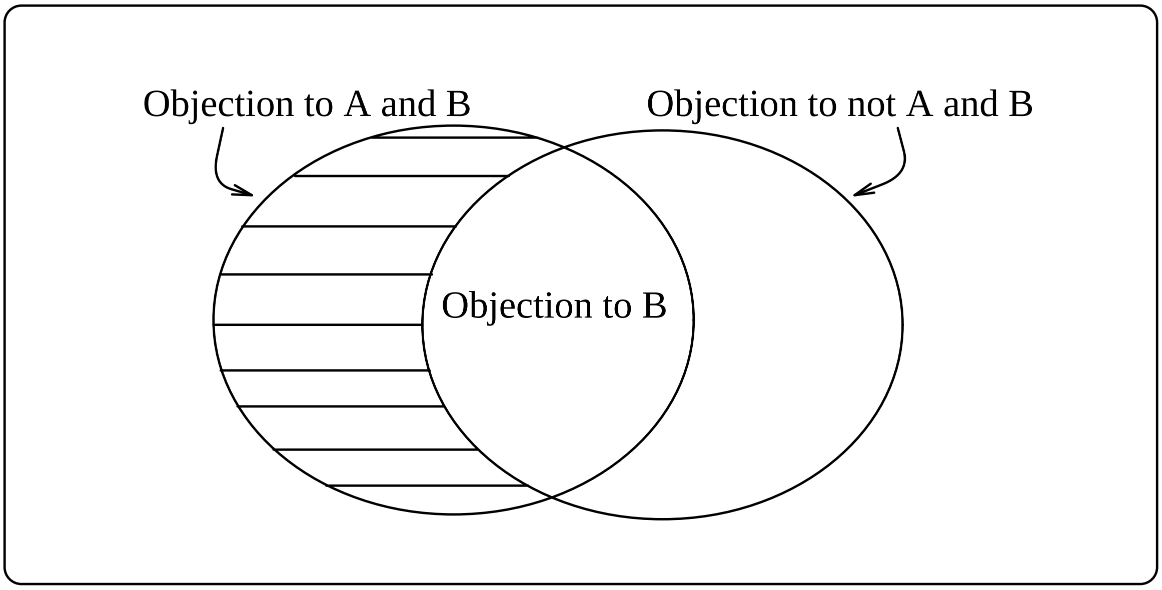

Figure 23: The circle on the left represents the objection to $A \wedge B$, and the one on the right represents the objection to $\neg A \wedge B$. The intersection of the two circles is the objection to B . Therefore, the shaded area represents $\Phi(A \wedge B) \wedge \neg\Phi(B)$, which is the conditional objection to A given B when $A \wedge B$ is not rejected. In this case, the conditional objection to A given B is logically disjoint from the objection to B .

1. The objection to B is $\beta \neq \mathbf{true}$.

2. The objection to A given B is α.

These two statements are consistent only if $\alpha \sqcup \beta$ is defined. Otherwise, there would be no objection–based state of belief that satisfies these statements. This is contrary to probability calculus, where the following two statements are always consistent:

1. The probability of B is $q \neq 0$.

2. The probability of A given B is p.

In probability calculus, one can always construct a probabilistic state of belief that satisfies the statements above.

7.4.3 Sufficient objections

Assessing a condition that leads to the rejection of a conjunction $A \wedge B$ is relatively easy, but assessing the weakest such condition is often a challenging task. Therefore, assessing objections is not always trivial.

In probability calculus, a related problem is solved by appealing to conditional probabilities. In particular, instead of assessing the probability of a conjunction $A \wedge B$ directly, one can assess the conditional probability of A given B and multiply it by the probability of B:

$$Pr(A \wedge B) = Pr(A \mid B) \times Pr(B).$$

This indirect assessment of probabilities is useful because assessing the conditional probability of A given B is usually easier than assessing the probability of the conjunction $A \wedge B$.

In objection calculus, the objection to a conjunction $A \wedge B$ can be also be assessed indirectly. One can assess the conditional objection to A given B and disjoin it with the objection to B:

$$\Phi(A \wedge B) = \Phi_B(A) \sqcup \Phi(B).$$

But, as it turns out, assessing the conditional objection to A given B is usually as hard as assessing the objection to $A \wedge B$. Assessing conditional objections is not always a natural task because one must ensure the consistency of a conditional objection with the objection to its condition. This defeats the expected role of conditional objections, namely, to make the assessment of objections easier.

Fortunately though, there is an alternative solution to making the assessment of objections easier. The solution hinges on the notion of sufficient objections. A sufficient objection to A given B is an objection to $A \wedge B$ that when disjoined with the objection to B becomes the objection to $A \wedge B$.

Definition 7.4.6 *A sufficient objection to A given B, written $\breve{\Phi}_B(A)$, is any sentence α such that $\Phi(A \wedge B) = \alpha \vee \Phi(B)$.*

Notation $\breve{\Phi}_B(A) := \alpha$ means that α is a sufficient objection to A given B.

Assessing a sufficient objection $\breve{\Phi}_B(A)$ is usually easier than assessing the objection $\Phi(A \wedge B)$. But why? The following theorem suggests the answer.

Theorem 7.4.7 *Every sufficient objection $\breve{\Phi}_B(A)$ satisfies the following:*

$$\neg\Phi(B) \models \Phi(A \wedge B) \equiv \breve{\Phi}_B(A). \blacksquare$$

According to Theorem 7.4.7, a sufficient objection to A given B is equivalent to the objection to $A \wedge B$ when the objection to B is assumed to be false. Therefore, to assess a sufficient objection to A given B, one asks, What would be the objection to $A \wedge B$ in the absence of an objection to B?

Consider the circuit in Figure 24 as an example. Let the objectionee language be over primitive propositions P_0, P_1, P_2, P_3, P_4, which assert the state of wires in the circuit. And let the objection language be over primitive propositions $ok(X)$, $ok(Y)$, $ok(Z)$, which assert the statuses of gates in the circuit. To assess a sufficient objection to $\neg P_4$ given $P_3 \wedge P_2$, one asks, What would be the objection to $\neg P_4 \wedge P_3 \wedge P_2$ in the absence of an objection to $P_3 \wedge P_2$? The answer would usually be $ok(Z)$ in this case.

Although there is only one conditional objection to A given B, there is usually more than one sufficient objection to A given B. For example, the conditional objection $\Phi_B(A)$ and the objection $\Phi(A \wedge B)$ are both sufficient objections to A given B. However, people rarely provide these objections when asked for a sufficient objection.

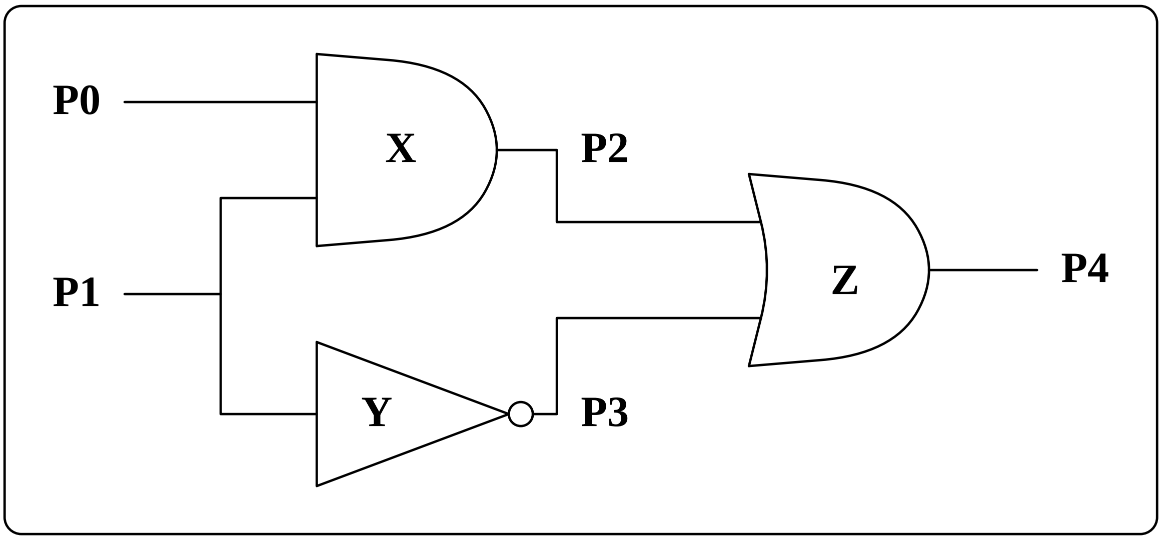

Figure 24: A digital circuit.

Consider again the circuit in Figure 24. When asked for a sufficient objection to $\neg P_4$ given $P_3 \wedge P_2$, people usually give $ok(Z)$. Curiously enough, $ok(Z)$ is not the conditional objection to $\neg P_4$ given $P_3 \wedge P_2$:

- The objection to $\neg P_4 \wedge P_3 \wedge P_2$ is $ok(Z) \vee (ok(X) \wedge ok(Y))$.
- The objection to $P_3 \wedge P_2$ is $ok(X) \wedge ok(Y)$.
- The conditional objection to $\neg P_4$ given $P_3 \wedge P_2$ is $ok(Z) \wedge (\neg ok(X) \vee \neg ok(Y))$.

7.5 Objection–based independence

Chapter 4 provides a definition of independence in abstract states of belief. Objection calculus inherits this definition, which is discussed in Section 7.5.1. However, there is a weaker notion of independence in objection calculus, which seems to be more appropriate for certain applications. The notion of independence is discussed in Section 7.5.2.

7.5.1 Strong independence

In probability calculus, the notion of independence is closely related to the notion of probability change. In particular, we say that a set of propositions I is independent from J given K if the conditional probability of $\underline{I}$ given $\underline{K}$ equals the conditional probability of $\underline{I}$ given $\underline{J} \wedge \underline{K}$. That is, once $\underline{K}$ is observed, the probability of $\underline{I}$ does not change when $\underline{J}$ is observed.

In Chapter 4, this notion of independence was generalized to abstract states of belief. Since states of belief in objection calculus are instances of abstract states of belief, they inherit this definition.

Definition 7.5.1 *A state of belief Φ finds I <u>strongly independent</u> from J given K, written $SIN_{\Phi}(I, K, J)$, precisely when the conditional objection to $\underline{I}$ given $\underline{J} \wedge \underline{K}$ is equivalent to the conditional objection to $\underline{I}$ given $\underline{K}$.*

I refer to this as "strong" independence because objection calculus has a weaker notion of independence. Interestingly enough, weak independence in objection calculus seems to be more appropriate for certain applications. The definition of weak independence is given in the next section. In the rest of this section, I identify an application for which strong independence seems inappropriate.

Consider a state of belief Φ about the digital circuit in Figure 25, where the objection language $\mathcal{O}$ is constructed from the following primitive propositions:

$$\begin{aligned} ok(X) &= \text{"Inverter } X \text{ is functioning normally,"} \\ ok(Y) &= \text{"Inverter } Y \text{ is functioning normally,"} \end{aligned}$$

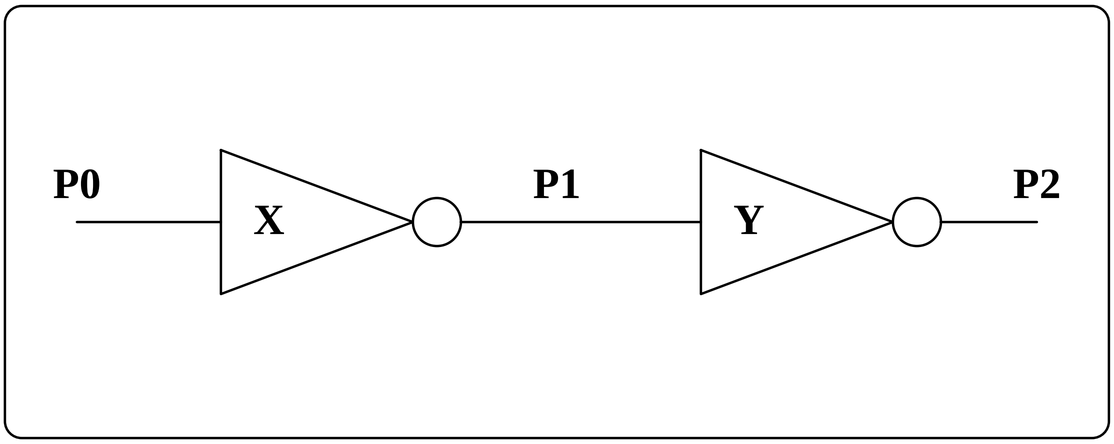

Figure 25: A digital circuit.

and the objectionee language $\mathcal{L}$ is constructed from the following primitive propositions:

$$\begin{aligned} P_0 &= \text{“Wire } P_0 \text{ is on,”} \\ P_1 &= \text{“Wire } P_1 \text{ is on,”} \\ P_2 &= \text{“Wire } P_2 \text{ is on.”} \end{aligned}$$

We expect the state of belief Φ to be such that

- The objection to $\neg P_0 \wedge P_1$ is **false**.
- The objection to $P_0 \wedge P_1$ is $ok(X)$.
- The objection to $\neg P_0 \wedge P_1 \wedge P_2$ is $ok(Y)$.
- The objection to $P_0 \wedge P_1 \wedge P_2$ is $ok(X) \vee ok(Y)$.

Moreover, we expect the state of belief Φ to find P_2 “intuitively” independent from P_0 given P_1. As we shall see, however, P_2 is not strongly independent from P_0 given P_1. In particular, the conditional objection to P_2 given $\neg P_0 \wedge P_1$ is not equivalent to the conditional objection to P_2 given $P_0 \wedge P_1$. After observing $\neg P_0 \wedge P_1$, the objection to P_2 becomes

$$\begin{aligned}\Phi_{\neg P_0 \wedge P_1}(P_2) &= \Phi(\neg P_0 \wedge P_1 \wedge P_2) \angle \Phi(\neg P_0 \wedge P_1) \\ &= ok(Y) \angle \mathbf{false} \\ &= ok(Y).\end{aligned}$$

Therefore, $ok(Y)$ is the weakest condition under which Φ rejects $\neg P_0 \wedge P_1 \wedge P_2$ but cannot reject $\neg P_0 \wedge P_1$. But after observing $P_0 \wedge P_1$, the objection to P_2 becomes

$$\begin{aligned}\Phi_{P_0 \wedge P_1}(P_2) &= \Phi(P_0 \wedge P_1 \wedge P_2) \angle \Phi(P_0 \wedge P_1) \\ &= (ok(X) \vee ok(Y)) \angle ok(X) \\ &= ok(Y) \wedge \neg ok(X).\end{aligned}$$

Therefore, $ok(Y) \wedge \neg ok(X)$ is the weakest condition under which Φ rejects $P_0 \wedge P_1 \wedge P_2$ but cannot reject $P_0 \wedge P_1$.

This example demonstrates that strong independence in objection calculus does not correspond to "independence" as we know it in digital circuits. But what does? This question is answered in the next section.

7.5.2 Weak independence

Strong independence in objection calculus appeals to conditional objections. The weaker notion of independence appeals to sufficient objections. Specifically, if propositions I are weakly independent from K given J, then assessing a sufficient objection to $\underline{I}$ given $\underline{J} \wedge \underline{K}$ does not depend on the state $\underline{J}$. The formal definition is given below.

Definition 7.5.2 *A state of belief Φ finds I weakly independent from J given K, written $WIN_\Phi(I, K, J)$, precisely when every sufficient objection to $\underline{I}$ given $\underline{K}$ is also a sufficient objection to $\underline{I}$ given $\underline{J} \wedge \underline{K}$.*

Corollary 7.5.3 *If $WIN_\Phi(I, K, J)$, then $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \check{\Phi}_{\underline{K}}(\underline{I}) \vee \Phi(\underline{J} \wedge \underline{K})$.*

The following is a characterization of weak independence that does not mention sufficient objections:

Theorem 7.5.4 *$WIN_\Phi(I,K,J)$ precisely when $\Phi(\underline{I}\wedge\underline{J}\wedge\underline{K}) = \Phi(\underline{I}\wedge\underline{K})\vee\Phi(\underline{J}\wedge\underline{K})$.*

Considering the digital circuit in Figure 25, one can show that

$$\Phi(\underline{P_2}\wedge\underline{P_1}\wedge\underline{P_0}) = \Phi(\underline{P_2}\wedge\underline{P_1})\vee\Phi(\underline{P_1}\wedge\underline{P_0}).$$

For example,

$$\begin{aligned}\Phi(P_2\wedge P_1\wedge P_0) &= ok(Y)\vee ok(X),\\ \Phi(P_2\wedge P_1) &= ok(Y),\\ \Phi(P_1\wedge P_0) &= ok(X).\end{aligned}$$

Therefore, one can show that P_2 is weakly independent from P_0 given P_1.

Weak independence satisfies a number of properties that have major consequences. For example, the following property shows that sufficient objections can be decomposed into simpler sufficient objections in the presence of weak independence:

Theorem 7.5.5 *If $WIN_\Phi(I,K,J)$, then $\breve{\Phi}_{\underline{K}}(\underline{I}\wedge\underline{J}) := \breve{\Phi}_{\underline{K}}(\underline{I})\vee\breve{\Phi}_{\underline{K}}(\underline{J})$.*

Among the most important properties of weak independence are the graphoid axioms [Pearl, 1988].

Theorem 7.5.6 (Symmetry) *$WIN_\Phi(I,K,J)$ precisely when $WIN_\Phi(J,K,I)$.* ∎

Theorem 7.5.7 (Decomposition) *If $WIN_\Phi(I,K,J\cup L)$, then $WIN_\Phi(I,K,J)$.*

Theorem 7.5.8 (Weak Union) *If $WIN_\Phi(I,K,J\cup L)$, then $WIN_\Phi(I,K\cup J,L)$.*

Theorem 7.5.9 (Contraction)

$$\textit{If } WIN_\Phi(I,K,J) \textit{ and } WIN_\Phi(I,K\cup J,L), \textit{ then } WIN_\Phi(I,K,J\cup L).$$

Together, Decomposition, Weak Union, and Contraction give

$$WIN_\Phi(I,K,J) \text{ and } WIN_\Phi(I,K\cup J,L) \text{ precisely when } WIN_\Phi(I,K,J\cup L).$$

7.6 Objection–based causal networks

Chapter 4 provides a definition of abstract causal networks. Objection calculus inherits this definition, which is discussed in Section 7.6.1. However, objection calculus has a weaker notion of causal networks, which seems to be more appropriate for certain applications. This notion is discussed in Section 7.6.2.

7.6.1 Strong causal networks

When degrees of support are objections, I refer to an abstract causal network as a strong objection–based (sob) causal network. The topology of a sob causal network encodes strong independence assertions, and its tables contain conditional objections. Below is the formal definition.

Definition 7.6.1 *A sob causal network is a tuple* $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{CO} \rangle$*, where*

- $\mathcal{L}$ *and* $\mathcal{O}$ *are propositional languages over disjoint primitive propositions.*
- $\mathcal{G}$ *is a directed acyclic graph over the primitive propositions of* $\mathcal{L}$*.*
- $\mathcal{CO}$ *is a partial function* $\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{O}$ *such that*
  - $\mathcal{CO}_{\underline{i\diamond}}(\underline{i})$ *is defined, and*
  - $\bigwedge_{\underline{i}} \mathcal{CO}_{\underline{i\diamond}}(\underline{i}) = \mathbf{false}$ *for every primitive proposition* i *in* $\mathcal{L}$*.*

The function $\mathcal{CO}$ *is called a conditional objection function.*

Definition 7.6.2 *A state of belief* Φ *satisfies a sob causal network* $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{CO} \rangle$ *precisely when*

$$SIN_{\Phi}(i, i\diamond, i\triangleleft) \text{ and } \Phi(\underline{i\diamond}) \neq \mathbf{true} \text{ only if } \Phi_{\underline{i\diamond}}(\underline{i}) = \mathcal{CO}_{\underline{i\diamond}}(\underline{i}).$$

Sob causal networks are based on strong independence and on conditional objections. As I mentioned earlier, conditional objections are not easy to assess, which makes the quantification of a sob causal network unnatural. Moreover, strong independence is not appropriate for certain applications, which makes sob causal networks inappropriate for these applications.

It is also possible to construct a sob causal network that is not satisfied by any state of belief. This should not be surprising because the definition of a sob causal network does not guarantee a conditional objection $\mathcal{CO}_{\underline{i}\diamond}$ to be consistent with the objection to its condition $\underline{i\diamond}$.

A solution to all these problems is given in the next section.

7.6.2 Weak causal networks

Weak objection–based (wob) causal networks have the same syntax as sob causal networks, but their semantics are different. The topology of a wob causal network encodes weak independence assertions, and its tables contain sufficient objections. Below is the formal definition.

Definition 7.6.3 *A* *<u>wob causal network</u>* *is a tuple* $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$, *where*

- $\mathcal{L}$ *and* $\mathcal{O}$ *are propositional languages over disjoint primitive propositions.*
- $\mathcal{G}$ *is a directed acyclic graph over the primitive propositions of* $\mathcal{L}$.
- $\mathcal{SO}$ *is a partial function* $\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{O}$ *such that*
  - $\mathcal{SO}_{\underline{i\diamond}}(\underline{i})$ *is defined, and*
  - $\bigwedge_{\underline{i}} \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) = \mathbf{false}$ *for every primitive proposition* i *in* $\mathcal{L}$.

The function $\mathcal{SO}$ *is called a* *<u>sufficient objection function.</u>*

Figure 26 depicts a wob causal network. This network could also be a sob causal network, but the given quantification makes it inconsistent. For example, if one viewed Figure 26 as a sob causal network, the objection to $A \wedge B$ would be $ok(X)$, and the conditional objection to C given $A \wedge B$ would be $ok(Y)$. These two objections are inconsistent.

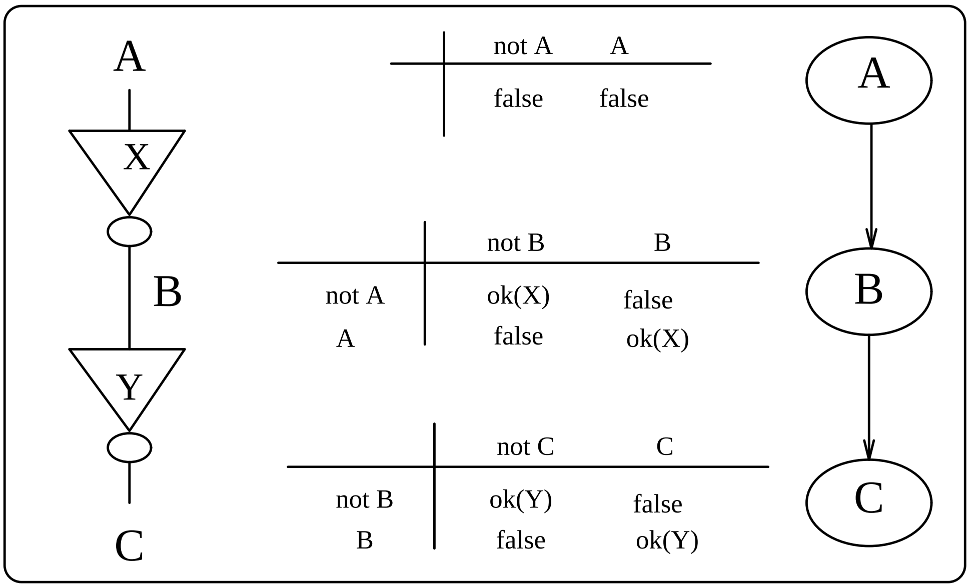

Figure 26: A digital circuit and its corresponding wob causal network.

Definition 7.6.4 *A state of belief Φ satisfies a wob causal network $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ precisely when*

$$WIN_{\Phi}(i, i\diamond, i\triangleleft) \text{ and } \breve{\Phi}(\underline{i} \wedge \underline{i\diamond}) := \mathcal{SO}_{\underline{i\diamond}}(\underline{i}).$$

Unlike sob causal networks, every wob causal network is consistent.

Theorem 7.6.5 *Every wob causal network is satisfied by exactly one objection–based state of belief.*

Moreover, wob causal networks are rich enough to represent any state of belief.

Theorem 7.6.6 *Every objection–based state of belief satisfies some wob causal network.*

Many of the weak independences that hold in a wob causal network can be retrieved by applying d–separation to the topology of the network.

Theorem 7.6.7 *Let Φ be the state of belief satisfying a wob causal network $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$. If $IN_{\mathcal{G}}(I, K, J)$, then $WIN_{\Phi}(I, K, J)$.*

7.7 The wob algorithm

In this section, I present an algorithm for the following computation. Given

- $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$, a wob causal network, and
- δ, an observation about some nodes in $\mathcal{G}$,

compute the following pair of objections for every node i in $\mathcal{G}$:

$$BL_i \stackrel{def}{=} \langle \Phi(i \wedge \delta), \Phi(\neg i \wedge \delta) \rangle.$$

The pair BL_i contains much information. For example, it can be used to compute:

1. The objection to the observation δ: $\Phi(i \wedge \delta) \wedge \Phi(\neg i \wedge \delta)$.
2. The conditional objection to $\underline{i}$ given δ: $\Phi(\underline{i} \wedge \delta) \angle \Phi(\delta)$.
3. The objection to $\underline{i}$: $\Phi(i \wedge \delta)$ when $\delta =$ **true**.

The algorithm I am about to present assumes that the causal network is singly connected. But similar to the generalized polytree algorithm, it can be extended to handle multiply connected networks.

The algorithm also assumes that the observation δ is about leaf nodes only, but this assumption does not affect the generality of the algorithm. I show in Chapter 5 that an observation about any node can be simulated by another observation about a leaf auxiliary node.

The algorithm is based on breaking down the computation of the pair BL_i into a number of smaller computations that are performed by the neighbors of node i. The result of each computation is passed on to node i as a message:

- The message sent by parent j to node i is denoted by $\mu_{j.i}$.
- The message sent by child k to node i is denoted by $\nu_{k.i}$.

The messages that node i receives from its parents are combined to form a pair of objections denoted by μ_i, while the messages that node i receives from its children are combined to form a pair of objections denoted by ν_i. The target pair, BL_i is the result of combining the pairs μ_i and ν_i.

Now that I have outlined the algorithm, let me explain what the passed messages are, how they are computed, and how they are combined.

7.7.1 Messages from parents

The message that node i receives from its parent j is defined as follows:

$$\mu_{j.i} \stackrel{def}{=} \langle \Phi(j \wedge \delta_{i \lhd j}), \Phi(\neg j \wedge \delta_{i \lhd j}) \rangle.$$

The messages that node i receives from its parents combine to yield the pair,

$$\mu_i \stackrel{def}{=} \langle \Phi(i \wedge \delta_{i \lhd}), \Phi(\neg i \wedge \delta_{i \lhd}) \rangle.$$

These messages are combined as follows:

Theorem 7.7.1 $\mu_i(\underline{i}) = \bigwedge_{\underline{i\diamond}} \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \bigvee_{\underline{i\diamond} \models \underline{j}} \mu_{j.i}(\underline{j})$.

Parent j computes the message that it sends to node i as follows:

Theorem 7.7.2 $\mu_{j.i} = \mu_j \vee \bigvee_{l \in j \circ i} \nu_{l.j}$.

That is, the message $\mu_{j.i}$ is the result of combining all the messages that node j receives from its neighbors, except node i.

7.7.2 Messages from children

The message that node i receives from its child k is defined as follows:

$$\nu_{k.i} \stackrel{def}{=} \langle \breve{\Phi}_i(\delta_{i \rhd k}), \breve{\Phi}_{\neg i}(\delta_{i \rhd k}) \rangle.$$

The messages that node i receives from its children combine to yield the pair,

$$\nu_i \stackrel{def}{=} \langle \breve{\Phi}_i(\delta_{i \rhd}), \breve{\Phi}_{\neg i}(\delta_{i \rhd}) \rangle.$$

These messages are combined as follows:

Theorem 7.7.3 $\nu_i = \bigvee_{k \in io} \nu_{k.i}$.

Child k computes the message that it sends to node i as follows:

Theorem 7.7.4 *If node k is not observed, then*

$$\nu_{k.i}(\underline{i}) = \bigwedge_{\underline{k}} \nu_k(\underline{k}) \vee \bigwedge_{\underline{k \diamond i}} \mathcal{SO}_{\underline{k \diamond i} \wedge \underline{i}}(\underline{k}) \vee \bigvee_{\underline{k \diamond i} \models \underline{l}} \mu_{l.k}(\underline{l}).$$

But if node k is observed, then

$$\nu_{k.i} = \begin{cases} \langle \textbf{false}, \textbf{true} \rangle, & \textit{if } \delta \models k; \\ \langle \textbf{true}, \textbf{false} \rangle, & \textit{if } \delta \models \neg k. \end{cases}$$

When node k is observed, the observation δ decides how the message $\nu_{k.i}$ is computed. But when k is not observed, the message $\nu_{k.i}$ is the result of combining all the messages that node k receives from its neighbors, except node i.

———

After node i has received all messages from its parents, it uses them to compute the pair μ_i. Similarly, node i uses the messages it has received from its children to compute the pair ν_i. The target pair BL_i is computed by combining the pairs μ_i and ν_i:

Theorem 7.7.5 $BL_i = \mu_i \vee \nu_i$.

7.7.3 Computational complexity

The algorithm I have given involves passing $2n$ messages, where n is the number of arcs in the network. The computation performed by each node is exponential in the number of its parents, but linear in the number of its children. These complexities also apply to the probabilistic version of this algorithm. Note, however, that although it is reasonable to assume that numeric addition and multiplication are operations that take constant time, it does not seem reasonable to assume that logical conjunction and disjunction—which in objection calculus play the role played by addition and

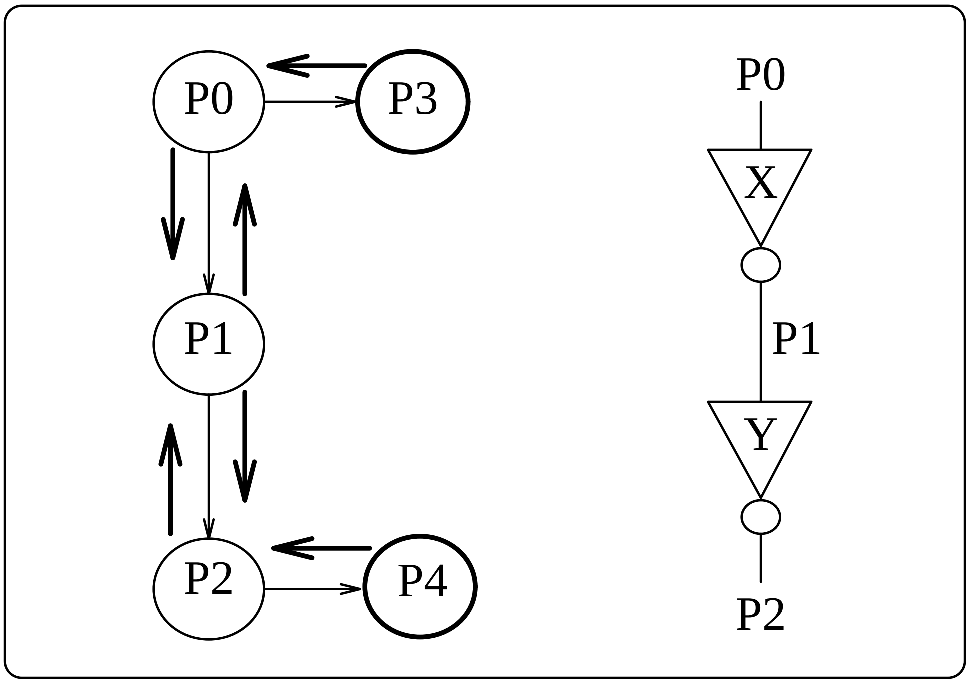

Figure 27: Messages exchanged in a forward propagation.

multiplication in probability calculus—are operations that take constant time. In fact, when objections are represented using disjunctive normal forms, experimental results show that in causal networks with a few hundred nodes, about 99% of the algorithm time is spent in conjoining and disjoining objections. Moreover, this time seems to vary significantly depending on the number and nature of available observations.

7.7.4 An example

Consider Figure 26, which depicts a wob causal network. Given the observation

$$\delta = P_3 \wedge \neg P_4,$$

let us compute the belief in every node of the network using forward propagation. This computation requires the messages,

$$\nu_{3.0}, \mu_{0.1}, \mu_{1.2}, \nu_{4.2}, \nu_{2.1}, \nu_{1.0},$$

which are depicted in Figure 27. These messages are computed as follows:

$$\nu_{3.0} = \langle \mathbf{false}, \mathbf{true} \rangle$$

$$\begin{aligned}
\mu_{0.1} &= \mu_0 \vee \nu_{3.0} \\
&= \langle \mathbf{false}, \mathbf{false} \rangle \vee \langle \mathbf{false}, \mathbf{true} \rangle, \text{ because Node 0 is root} \\
&= \langle \mathbf{false}, \mathbf{true} \rangle.
\end{aligned}$$

$$\begin{aligned}
\mu_{1.2} &= \mu_1 \\
&= \langle [\mathcal{SO}_{P_0}(P_1) \vee \mu_{0.1}(P_0)] \wedge [\mathcal{SO}_{\neg P_0}(P_1) \vee \mu_{0.1}(\neg P_0)], \\
&\quad\; [\mathcal{SO}_{P_0}(\neg P_1) \vee \mu_{0.1}(P_0)] \wedge [\mathcal{SO}_{\neg P_0}(\neg P_1) \vee \mu_{0.1}(\neg P_0)] \rangle \\
&= \langle [ok(X) \vee \mathbf{false}] \wedge [\mathbf{false} \vee \mathbf{true}], \\
&\quad\; [\mathbf{false} \vee \mathbf{false}] \wedge [ok(X) \vee \mathbf{true}] \rangle \\
&= \langle ok(X), \mathbf{false} \rangle.
\end{aligned}$$

$$\nu_{4.2} = \langle \mathbf{true}, \mathbf{false} \rangle.$$

$$\begin{aligned}
\nu_{2.1} &= \langle [\nu_2(P_2) \vee \mathcal{SO}_{P_1}(P_2)] \wedge [\nu_2(\neg P_2) \vee \mathcal{SO}_{P_1}(\neg P_2)], \\
&\quad\; [\nu_2(P_2) \vee \mathcal{SO}_{\neg P_1}(P_2)] \wedge [\nu_2(\neg P_2) \vee \mathcal{SO}_{\neg P_1}(\neg P_2)] \rangle, \\
&= \langle [\mathbf{true} \vee ok(Y)] \wedge [\mathbf{false} \vee \mathbf{false}], \\
&\quad\; [\mathbf{true} \vee \mathbf{false}] \wedge [\mathbf{false} \vee ok(Y)] \rangle, \text{ because } \nu_2 = \nu_{4.2} \\
&= \langle \mathbf{false}, ok(Y) \rangle.
\end{aligned}$$

$$\begin{aligned}
\nu_{1.0} &= \langle [\nu_1(P_1) \vee \mathcal{SO}_{P_0}(P_1)] \wedge [\nu_1(\neg P_1) \vee \mathcal{SO}_{P_0}(\neg P_1)], \\
&\quad\; [\nu_1(P_1) \vee \mathcal{SO}_{\neg P_0}(P_1)] \wedge [\nu_1(\neg P_1) \vee \mathcal{SO}_{\neg P_0}(\neg P_1)] \rangle \\
&= \langle [\mathbf{false} \vee ok(X)] \wedge [ok(Y) \vee \mathbf{false}], \\
&\quad\; [\mathbf{false} \vee \mathbf{false}] \wedge [ok(Y) \vee ok(X)] \rangle, \text{ because } \nu_1 = \nu_{2.1} \\
&= \langle ok(X) \wedge ok(Y), \mathbf{false} \rangle.
\end{aligned}$$

We must now compute the pair ν_i for each node i:

$$\begin{aligned}
\nu_0 &= \nu_{3.0} \vee \nu_{1.0} \\
&= \langle \mathbf{false}, \mathbf{true} \rangle \vee \langle ok(X) \wedge ok(Y), \mathbf{false} \rangle \\
&= \langle ok(X) \wedge ok(Y), \mathbf{true} \rangle. \\
\nu_1 &= \nu_{2.1} = \langle \mathbf{false}, ok(Y) \rangle. \\
\nu_2 &= \nu_{4.2} = \langle \mathbf{true}, \mathbf{false} \rangle.
\end{aligned}$$

And compute the pair μ_i for each node i:

$$\begin{aligned}
\mu_0 &= \langle \mathbf{false}, \mathbf{false} \rangle. \\
\mu_1 &= \langle \mathbf{false}, ok(X) \rangle. \\
\mu_2 &= \langle [\mathcal{SO}_{P_1}(P_2) \vee \mu_{1.2}(P_1)] \wedge [\mathcal{SO}_{\neg P_1}(P_2) \vee \mu_{1.2}(\neg P_1)], \\
&\quad [\mathcal{SO}_{P_1}(\neg P_2) \vee \mu_{1.2}(P_1)] \wedge [\mathcal{SO}_{\neg P_1}(\neg P_2) \vee \mu_{1.2}(\neg P_1)] \rangle \\
&= \langle [ok(Y) \vee ok(X)] \wedge [\mathbf{false} \vee \mathbf{false}], \\
&\quad [\mathbf{false} \vee ok(X)] \wedge [ok(Y) \vee \mathbf{false}] \rangle \\
&= \langle \mathbf{false}, ok(X) \wedge ok(Y) \rangle.
\end{aligned}$$

Finally, we compute the pair BL_i for every node i:

$$\begin{aligned}
BL_0 &= \langle \mathbf{false}, \mathbf{false} \rangle \vee \langle ok(X) \wedge ok(Y), \mathbf{true} \rangle \\
&= \langle ok(X) \wedge ok(Y), \mathbf{true} \rangle.
\end{aligned}$$

$$\begin{aligned}
BL_1 &= \langle ok(X), \mathbf{false} \rangle \vee \langle \mathbf{false}, ok(Y) \rangle \\
&= \langle ok(X), ok(Y) \rangle.
\end{aligned}$$

$$\begin{aligned}
BL_2 &= \langle \mathbf{false}, ok(X) \wedge ok(Y) \rangle \vee \langle \mathbf{true}, \mathbf{false} \rangle \\
&= \langle \mathbf{true}, ok(X) \wedge ok(Y) \rangle.
\end{aligned}$$

7.8 Justification and consequence calculi

In objection calculus, one quantifies the support for a sentence by assessing the objection to the sentence; that is, the most general reason for rejecting it.

The following question arises usually in connection with objection calculus, Could one quantify the support for a sentence by assessing the *justification* for the sentence; that is, the most general reason for accepting it?

The answer is yes, but the resulting calculus, *justification calculus,* is not an instance of the abstract calculus formalized in Chapters 2 and 3. Let me explain why.

When a sentence is rejected, its negation is accepted. Therefore, the objection to a sentence is the justification for its negation. Given this connection, it follows that the justification for the disjunction $A \vee B$, when A and B are logically disjoint, cannot be computed from the justification for A and the justification for B. This violates one of the principles underlying abstract states of belief:

- The support for $A \vee B$ is a function of the support for A and the support for B, when A and B are logically disjoint.

Instead, we have the following in justification calculus:

- The justification for $A \wedge B$ is a function of the justification for A and the justification for B.

To see why this holds in justification calculus, let $J(A)$ denote the justification for A, and $\Phi(A)$ denote the objection to A. Using the connection between justifications and objections, we have

$$\begin{aligned} J(A \wedge B) &= \Phi(\neg A \vee \neg B) \\ &= \Phi(\neg A) \wedge \Phi(\neg B) \\ &= J(A) \wedge J(B). \end{aligned}$$

———

Another calculus that is closely related to objection calculus is *consequence calculus,* in which one quantifies the support for a sentence by assessing the consequence of the sentence; that is, the most specific conclusion of accepting the sentence. Consequence calculus is an instance of the abstract calculus formalized in Chapters 2 and 3. Objection, justification, and consequence calculi are duals of one another. Their duality is given by the following equivalent statements:

1. α is the objection to A.
2. α is the justification for $\neg A$.
3. $\neg\alpha$ is the consequence of A.

A similar duality exists among impossibility, possibility, and necessity calculi [Dubois and Prade, 1988].

Chapter 8

Diagnosis using Objection Calculus

In this chapter, I explore the application of objection calculus to diagnosing faults in physical systems. In particular, I show how to describe the behavior of a physical system using a wob causal network, and how to use the wob algorithm to compute diagnoses of observations about the system.

8.1 Introduction

Objections are closely related to two influential notions in AI: *labels* and *diagnoses*.

Computing labels is the job of a clause management system [Reiter and de Kleer, 1987]. In section 8.2, I discuss clause management systems and define labels formally. In section 8.3, I study the relation between objections and labels.

Computing diagnoses is the job of a diagnosis system [de Kleer *et al.*, 1992]. In Section 8.4, I discuss diagnosis systems and define diagnoses formally. In Section 8.5, I study the relation between objections and diagnoses.

Finally, in Section 8.6, I show how wob causal networks can be used to describe the behavior of physical systems, and how the wob algorithm of Chapter 7 can be used to compute diagnoses; I also provide some examples.

8.2 Clause management systems

The basic task of a clause management system is to help a reasoner answer the following general question: Given a particular database, under what condition can I derive a particular sentence? The condition is usually required to be most general and to be phrased using a particular language. The following example provides some intuitive justification for this requirement.

Consider the circuit depicted in Figure 28. Let $\mathcal{L}'$ be a propositional language over primitive propositions A, B, C, D, E, F, which assert the state of wires in the circuit. And let $\mathcal{O}'$ be a propositional language over primitive propositions $ok(X)$, $ok(Y)$, $ok(Z)$, which assert the statuses of gates in the circuit. The behavior of the circuit can be described using statements of the form

$$gate\_input \wedge ok(gate) \supset gate\_output.$$

Let Δ' denote the conjunction of all such statements:

$$\begin{aligned}
\Delta' \equiv\ & (A \wedge ok(X) \supset \neg D) \wedge \\
& (\neg A \wedge ok(X) \supset D) \wedge \\
& (B \wedge C \wedge ok(Y) \supset E) \wedge \\
& (B \wedge \neg C \wedge ok(Y) \supset \neg E) \wedge \\
& \vdots
\end{aligned}$$

If a reasoner observes $\delta' = \neg A \wedge B \wedge C$ about the circuit, then it might ask a clause management system the following question: Given the database $\Delta' \wedge \delta'$, under what condition can I derive the sentence F? The most general such condition phrased using the language $\mathcal{O}'$ is $(ok(X) \vee ok(Y)) \wedge ok(Z)$. This condition is called the $\mathcal{O}'$–label for F with respect to $\Delta' \wedge \delta'$.

Definition 8.2.1 *The* <u>*$\mathcal{O}$–label*</u> *for sentence A with respect to database Δ, written $Label(A, \Delta, \mathcal{O})$, is the weakest sentence in language $\mathcal{O}$ that when conjoined with Δ entails A.*

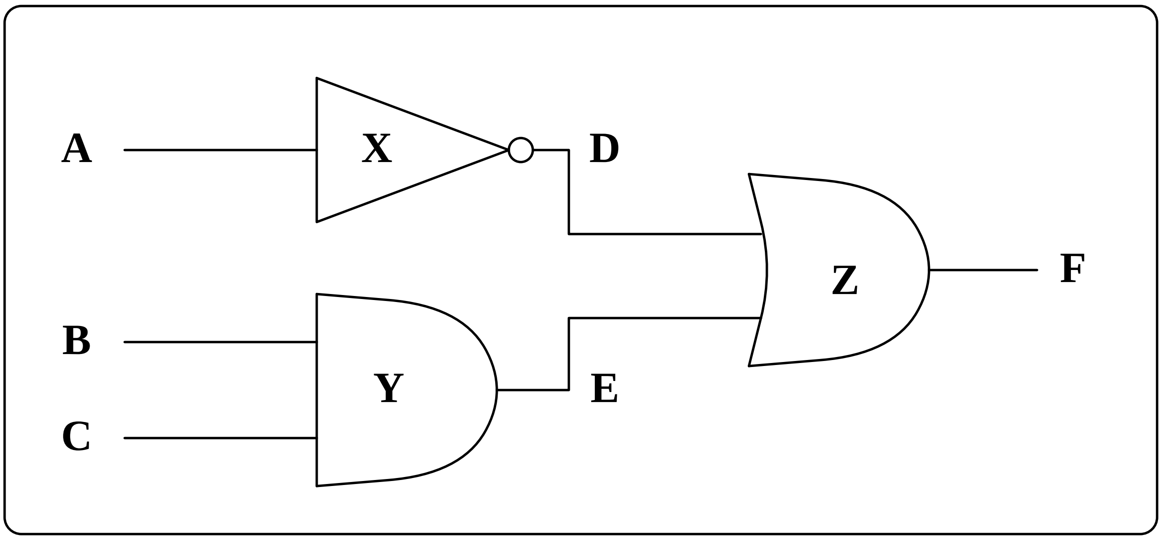

Figure 28: A digital circuit.

Reiter and de Kleer [Reiter and de Kleer, 1987] define labels differently. To state and analyze their definition, I need a number of notions. These are the notions of conjunctive clause, disjunctive clause, prime implicant, prime implicate, and minimal support clause.

Definition 8.2.2 *A* <u>*conjunctive clause*</u> *is a conjunction of literals.*

Definition 8.2.3 *An* <u>*implicant*</u> *for A is a conjunctive clause that entails A. A* <u>*prime implicant*</u> *for A is a weakest implicant for A.*

Definition 8.2.4 *A* <u>*disjunctive clause*</u> *is a disjunction of literals.*

Definition 8.2.5 *An* <u>*implicate*</u> *of A is a disjunctive clause that is entailed by A. A* <u>*prime implicate*</u> *of A is a strongest implicate of A.*

Definition 8.2.6 ([Reiter and de Kleer, 1987]) *A* <u>*support*</u> *for sentence A with respect to database Δ is an implicate of $\Delta \wedge \neg A$ that is not an implicate of Δ. A* <u>*minimal support*</u> *for sentence A with respect to database Δ is a strongest support for A with respect to Δ.*

Reiter and de Kleer define labels as follows:

Definition 8.2.7 ([Reiter and de Kleer, 1987]) *The $\mathcal{O}$–label for sentence A with respect to database Δ is the set of all conjunctive clauses I, such that I belongs to language $\mathcal{O}$ and $\neg I$ is a minimal supports for A with respect to Δ.*[1]

At first glance, Definition 8.2.7 looks quite different from Definition 8.2.1. But a closer look shows that for a certain class of databases the difference between the two definitions is only syntactic. In particular, the $\mathcal{O}$–label for A with respect to Δ according to Definition 8.2.7 corresponds to the prime implicants for $Label(A, \Delta, \mathcal{O})$ that are consistent with Δ. This correspondence is a corollary of the following theorem:

Theorem 8.2.8 *I is a prime implicant for $Label(A, \Delta, \mathcal{O})$ and is consistent with Δ precisely when $\neg I$ belongs to $\mathcal{O}$ and is a minimal support for A with respect to Δ.*

The difference between Definition 8.2.7 and Definition 8.2.1 of labels is only syntactic provided the database is non–committal in the following sense:

Definition 8.2.9 *A database Δ is <u>non–committal</u> with respect to language $\mathcal{O}$ precisely when it does not entail any invalid sentence in $\mathcal{O}$.*

For example, the database $\Delta' \wedge \delta'$ given earlier with respect to the circuit of Figure 28 is non–committal with respect to the language $\mathcal{O}'$. However, the database $\Delta' \wedge \delta' \wedge D$ is committal because it entails $\neg ok(X)$, which belongs to $\mathcal{O}'$.

When a database Δ is non–committal with respect to language $\mathcal{O}$, every prime implicant for $Label(A, \Delta, \mathcal{O})$ is consistent with Δ. In this case, the label for A with respect to Δ according to Definition 8.2.7 corresponds to the set of prime implicants for $Label(A, \Delta, \mathcal{O})$.

Theorem 8.2.10 *Let Δ be a non–committal database with respect to language $\mathcal{O}$. The $\mathcal{O}$–label for A with respect to Δ according to Reiter and de Kleer is the set of prime implicants for $Label(A, \Delta, \mathcal{O})$.*

[1] Reiter and de Kleer refer to the primitive propositions of the language $\mathcal{O}$ as *assumptions*.

In this chapter, I consider only databases that are non–committal with respect to some language $\mathcal{O}$ of interest. But this does not affect the generality of the basic results given in this chapter. In fact, databases in diagnosis applications are usually non–committal with respect to some language $\mathcal{O}$. In these applications, the database Δ is usually a description of a system behavior, and the language $\mathcal{O}$ is about the statuses of the system components. Since behavioral system descriptions do not usually imply anything about the statuses of the system components, databases in diagnosis applications are usually non–committal.

8.3 The relation between objections and labels

There is a correspondence between non–committal databases and states of belief in objection calculus. For each state of belief Φ, we can construct a non–committal database Δ^Φ, where there is a one–to–one correspondence between the objections of Φ and the labels of Δ^Φ. Furthermore, for each non–committal database Δ, we can construct a state of belief Φ^Δ, where there is a one–to–one correspondence between the labels of Δ and the objections of Φ^Δ. These results are stated by the following theorems, which assume that Φ is a state of belief with respect to $(\mathcal{L}, \mathcal{O})$.

First, the database corresponding to state of belief Φ is the conjunction of all statements of the form, The objection to A implies $\neg A$.

Definition 8.3.1 *The database corresponding to state of belief Φ is*

$$\Delta^\Phi \stackrel{def}{=} \bigwedge_{A \in \mathcal{L}} \Phi(A) \supset \neg A.$$

The following theorem shows that the database corresponding to a state of belief is non–committal.

Theorem 8.3.2 *The database Δ^Φ is non–committal with respect to language $\mathcal{O}$.*

The label for sentence A with respect to database Δ^Φ is the objection attributed by state of belief Φ to sentence $\neg A$.

Theorem 8.3.3 *If A is a sentence in language $\mathcal{L}$, then $\Phi(\neg A) \equiv Label(A, \Delta^\Phi, \mathcal{O})$.*

We can also construct a state of belief that corresponds to any non–committal database.

Theorem 8.3.4 *The mapping $\Psi^\Delta : \mathcal{L} \to \mathcal{O}$ such that $\Psi^\Delta(A) \equiv Label(\neg A, \Delta, \mathcal{O})$ is an objection–based state of belief.*

8.3.1 The computational value of weak independence

Since every state of belief in objection calculus corresponds to some non–committal database, every weak independence assertion about a state of belief Φ must also be an assertion about the database Δ^{Φ}. But what is this assertion? The following theorem answers this question:

Theorem 8.3.5 *A state of belief Φ finds propositions I weakly independent from J given K precisely when*

$$\mathit{Label}(\neg\underline{I} \vee \neg\underline{J} \vee \neg\underline{K}, \Delta^{\Phi}, \mathcal{O}) \equiv \mathit{Label}(\neg\underline{I} \vee \neg\underline{K}, \Delta^{\Phi}, \mathcal{O}) \vee \mathit{Label}(\neg\underline{J} \vee \neg\underline{K}, \Delta^{\Phi}, \mathcal{O}). \blacksquare$$

Let me explain the intuition behind this theorem. Suppose that

$$\mathit{Label}(\neg\underline{I} \vee \neg\underline{K}, \Delta^{\Phi}, \mathcal{O}) \equiv \alpha \quad \text{and} \quad \mathit{Label}(\neg\underline{J} \vee \neg\underline{K}, \Delta^{\Phi}, \mathcal{O}) \equiv \beta.$$

That is, α and β are the weakest sentences in $\mathcal{O}$ such that

$$\Delta^{\Phi} \wedge \alpha \models \neg\underline{I} \vee \neg\underline{K} \quad \text{and} \quad \Delta^{\Phi} \wedge \beta \models \neg\underline{J} \vee \neg\underline{K}. \tag{10}$$

Logically, we can deduce the following from (10):

$$\Delta^{\Phi} \wedge (\alpha \vee \beta) \models \neg\underline{I} \vee \neg\underline{J} \vee \neg\underline{K}. \tag{11}$$

But logically, we cannot deduce that $\alpha \vee \beta$ is the weakest sentence in $\mathcal{O}$ that satisfies (11). Whether or not this holds depends on the nature of the database Δ^{Φ}. Therefore, deciding whether $\alpha \vee \beta$ is the $\mathcal{O}$–label for $\neg\underline{I} \vee \neg\underline{J} \vee \neg\underline{K}$ involves an examination of the database Δ^{Φ}.

Now, if the state of belief Φ finds I weakly independent from J given K, then $\alpha \vee \beta$ is indeed the weakest sentence in $\mathcal{O}$ that satisfies (11). In this case, we can conclude that $\alpha \vee \beta$ is the $\mathcal{O}$–label for $\neg\underline{I} \vee \neg\underline{J} \vee \neg\underline{K}$ without further examination of the database Δ^{Φ}. Avoiding this examination is the computational value of weak independence.

Consider the circuit depicted in Figure 28 as an example. We have,

$$\mathit{Label}(D, \Delta' \wedge \delta', \mathcal{O}') \quad \equiv \quad ok(X),$$

$$\begin{aligned} Label(E, \Delta' \wedge \delta', \mathcal{O}') &\equiv ok(Y), \\ Label(D \vee \neg F, \Delta' \wedge \delta', \mathcal{O}') &\equiv ok(X), \\ Label(E \vee \neg F, \Delta' \wedge \delta', \mathcal{O}') &\equiv ok(Y). \end{aligned}$$

Since proposition E is weakly independent from D, we have

$$\begin{aligned} Label(D \vee E, \Delta' \wedge \delta', \mathcal{O}') &\equiv Label(D, \Delta' \wedge \delta', \mathcal{O}') \vee Label(E, \Delta' \wedge \delta', \mathcal{O}') \\ &\equiv ok(X) \vee ok(Y). \end{aligned}$$

However, E is not weakly independent from D given F. This is verified by

$$\begin{aligned} & Label(D \vee E \vee \neg F, \Delta' \wedge \delta', \mathcal{O}') \\ \equiv \;& ok(X) \vee ok(Y) \vee ok(Z) \\ \not\equiv \;& Label(D \vee \neg F, \Delta' \wedge \delta', \mathcal{O}') \vee Label(E \vee \neg F, \Delta' \wedge \delta', \mathcal{O}'). \end{aligned}$$

8.3.2 The logical meaning of wob causal networks

We know from Chapter 7 that each wob causal network corresponds to a state of belief. Since each state of belief corresponds to a non–committal database, then each wob causal network must correspond to a non–committal database. But what is this database? The following theorem answers this question:

Theorem 8.3.6 *If $CN = \langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is a wob causal network, and if Φ is the state of belief satisfying CN, then*

$$\Delta^{\Phi} \equiv \bigwedge_i \underline{i_{\diamond}} \wedge \mathcal{SO}_{\underline{i_{\diamond}}}(\underline{i}) \supset \neg \underline{i}.$$

According to Theorem 8.3.6, the database corresponding to a wob causal network is the conjunction of statements of the form

$$state\_of\_parents \wedge sufficient\_objection \supset \neg state\_of\_node.$$

Also according to Theorem 8.3.6, the database corresponding to a causal network does not depend on the causal structure of the network. That is, given the tables of a causal network, the independences asserted by the causal structure are logically redundant. This is both surprising and consequential! It is surprising because it suggests that a causal structure has no representational value. And it is consequential because it suggests that the independences asserted by a causal structure must be correct. This means that Theorem 8.3.6 can be used to prove that Claim 4.3.2 — of Chapter 4 — is true with respect to objection calculus.

Although a causal structure is representationally redundant, we have seen in Section 8.3.1 that its computational value should not be underestimated.

8.4 Diagnosis systems

The basic task of a diagnosis system is to help a reasoner answer the following general question: Given a particular database, what can I conclude after recording a particular observation? The conclusion is usually required to be most specific and phrased using a particular language. The following example provides some intuitive justification for this requirement.

Consider the circuit depicted in Figure 28, and $\mathcal{L}', \mathcal{O}'$ and Δ' be as given in Section 8.2. If a reasoner observes $\delta' = \neg A \wedge B \wedge C \wedge \neg F$ about the circuit, it might ask a diagnosis system the following question: Given the database Δ', what can I conclude after recording the observation δ'? The most specific conclusion that is phrased using the language $\mathcal{O}'$ is $(\neg ok(X) \wedge \neg ok(Y)) \vee \neg ok(Z)$. This conclusion is called the $\mathcal{O}'$–diagnosis for δ' with respect to Δ'.

Definition 8.4.1 *The* $\underline{\mathcal{O}\text{–diagnosis}}$ *of* δ *with respect to* Δ*, written* $\mathit{Diagnosis}(\delta, \Delta, \mathcal{O})$*, is the strongest sentence in* $\mathcal{O}$ *that is entailed by* $\Delta \wedge \delta$*.*

The AI literature contains many notions of diagnosis. For example, de Kleer et al. [de Kleer *et al.*, 1992] discuss eight such notions, the most influential of which are minimal, prime, and kernel diagnoses. They conclude their discussion with the following remark:

> The notions of minimal and prime diagnosis are inadequate to characterize diagnoses generally. We argue that the notion of kernel diagnosis which designates some components as normal, others abnormal, and the reminder as being either, is a better way to characterize diagnoses. [de Kleer *et al.*, 1992, Page 221]

As we shall see next, the prime implicants for $\mathit{Diagnosis}(\delta, \Delta, \mathcal{O})$ correspond to kernel diagnoses as defined by de Kleer et al.

Kernel diagnoses are defined with respect to a *system*, which is a triple $(\delta, \Delta, \mathcal{O})$, where[2]

[2]This definition is a propositional version of a definition used by de Kleer et al.

- δ, the *system observation,* is a propositional sentence.
- Δ, the *system description,* is also a propositional sentence.
- $\mathcal{O}$, the *system component language,* is a propositional language over primitive propositions of the form $ok(C)$, where C is a component of the system.

Definition 8.4.2 *The kernel diagnoses of system $(\delta, \Delta, \mathcal{O})$ are the prime implicants for the conjunction of all prime implicates of $\Delta \wedge \delta$ that belong to $\mathcal{O}$.*

The relation between kernel diagnoses and the notion of diagnosis as given by Definition 8.4.1 is a corollary of the following theorem:

Theorem 8.4.3 *The strongest sentence in $\mathcal{O}$ that is entailed by $\Delta \wedge \delta$ is equivalent to the conjunction of all prime implicates of $\Delta \wedge \delta$ that belong to $\mathcal{O}$.*

Corollary 8.4.4 *The prime implicants for $Diagnosis(\delta, \Delta, \mathcal{O})$ are the kernel diagnoses of the system $(\delta, \Delta, \mathcal{O})$.* ∎

8.5 The relation between objections and kernel diagnoses

There is a one–to–one correspondence between the objections of a state of belief Φ and the diagnoses of its corresponding database Δ^{Φ}.

Theorem 8.5.1 *If Φ is a state of belief with respect to $(\mathcal{L}, \mathcal{O})$, and if δ is a sentence in $\mathcal{L}$, then*

$$\neg\Phi(\delta) \equiv \mathit{Diagnosis}(\delta, \Delta^{\Phi}, \mathcal{O}).$$

According to Theorem 8.5.1, diagnoses are negated objections. This is intuitive: if $\Phi(\delta)$ is the most general reason for rejecting δ, then $\neg\Phi(\delta)$ is the most specific conclusion of accepting δ.

A corollary of Theorem 8.5.1 shows that kernel diagnoses correspond to the prime implicants for negated objections.

Corollary 8.5.2 *Let Φ be a state of belief with respect to $(\mathcal{L}, \mathcal{O})$, and let δ be a sentence in $\mathcal{L}$. The prime implicants for $\neg\Phi(\delta)$ are the kernel diagnoses of the system $(\delta, \Delta^{\Phi}, \mathcal{O})$.* ∎

This corollary is the key to computing kernel diagnoses using the wob algorithm of Chapter 7: Given a wob causal network that describes a physical system, and given an observation δ about the system,

1. the wob algorithm is used to compute the objection $\Phi(\delta)$ to the observation δ,
2. the computed objection is then negated, and
3. the prime implicants for the negated objection $\neg\Phi(\delta)$ are computed.

These are the kernel diagnoses of the observation δ.

8.6 Diagnosis using wob causal networks

In this section, I sketch the steps involved in diagnosing system faults using wob causal networks. I also provide three examples of diagnosing faults in digital circuits.

The first step in constructing a wob causal network $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is defining the propositional language $\mathcal{O}$. To define this language, one must identify the system components needed in order to describe the system behavior. The statuses of these components constitute the primitive propositions of the language $\mathcal{O}$. In particular, primitive propositions in $\mathcal{O}$ usually have the form $ok(C)$, where C is a system component. Alternatively, these primitive propositions can have the form $ab(C)$, where C is a system component. The two choices are dual. In digital circuits, for example, the components are usually the individual gates from which the circuit is composed.

The second step in constructing a wob causal network is defining the propositional language $\mathcal{L}$. To define this language, one must identify the system aspects—other than the statuses of its components—that are needed to describe the system behavior. These aspects constitute the primitive propositions of the language $\mathcal{L}$. For example, the aspects of digital circuits are usually the states of wires that connect the individual gates.

The third step in constructing a wob causal network is constructing the causal structure $\mathcal{G}$, which is a direct acyclic graph. The nodes of this graph must be the primitive propositions in the language $\mathcal{L}$, that is, the system aspects. The arcs of this graph must respect the following principle:

> Assuming that no system component is faulty, observing the state of aspects $i\diamond$ of the system, which are the parents of aspect i, should be enough to determine the state of aspect i.

A causal structure that respects this principle is said to be *quantifiable,* which brings us to the last step in constructing a wob causal network: quantifying its causal structure.

The purpose of quantifying a causal structure is to voice objections to *local behaviors* of the system. A local behavior is a pair $(\underline{i}, \underline{i\diamond})$, where $\underline{i}$ is a state of aspect i and $\underline{i\diamond}$ is a state of its parent aspects $i\diamond$. The local behavior $(\underline{i}, \underline{i\diamond})$ means that aspect

i has state $\underline{i}$ when aspects $i\diamond$ have state $\underline{i\diamond}$. In quantifying a causal structure, one provides a sufficient objection to state $\underline{i}$ given state $\underline{i\diamond}$ for each local behavior $(\underline{i\diamond}, \underline{i})$. These sufficient objections fill the tables of a wob causal network and, consequently, specify the sufficient objection function $\mathcal{SO}$.

To summarize, constructing a wob causal network $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ involves:

1. Identifying the system components and constructing the language $\mathcal{O}$.

2. Identifying the system aspects and constructing the language $\mathcal{L}$.

3. Constructing the causal structure $\mathcal{G}$.

4. Quantifying the structure $\mathcal{G}$ by providing the sufficient objection function $\mathcal{SO}$.

By performing these steps, one becomes committed to a state of belief $\Phi : \mathcal{L} \rightarrow \mathcal{O}$ about the system under consideration. Moreover, the constructed causal network and its corresponding state of belief become the basis for diagnosing faults of the described system.

For example, suppose that we observe δ about the described system. Then, given Theorem 8.5.1, the diagnosis of δ is the negated objection $\neg\Phi(\delta)$. This diagnosis can be computed using the algorithm given at the end of Chapter 7 when the observation δ is a conjunction of literals. In particular, the algorithm computes the pair

$$\langle \Phi(i \wedge \delta), \Phi(\neg i \wedge \delta) \rangle$$

for each node i in the causal network. But conjoining the elements of any such pair gives the objection to the observation δ,

$$\Phi(\delta) = \Phi(i \wedge \delta) \wedge \Phi(\neg i \wedge \delta).$$

Therefore, the diagnosis of δ is given by $\neg\Phi(i \wedge \delta) \vee \neg\Phi(\neg i \wedge \delta)$.

8.6.1 The first example

Figure 29 depicts a wob causal network that describes the digital circuit of Figure 28. The given quantification assumes the following:

- If a gate is OK, then it behaves normally.
- If a gate is faulty, then it may behave normally or abnormally.

The causal structure of a digital circuit is usually reflected by its wiring structure: Assuming that no gate is faulty, observing the input to a gate is enough to determine its output.

Now, suppose that we observe $\delta = \neg A \wedge B \wedge C \wedge \neg F$ about the circuit of Figure 28. Using the wob algorithm, we compute

$$BL_E = \langle ok(Z), ok(Y) \vee (ok(X) \wedge ok(Z)) \rangle.$$

Hence, $\neg\Phi(\delta) = (\neg ok(Y) \vee \neg ok(Z)) \wedge (\neg ok(X) \vee \neg ok(Z))$ is the diagnosis of the observation δ.

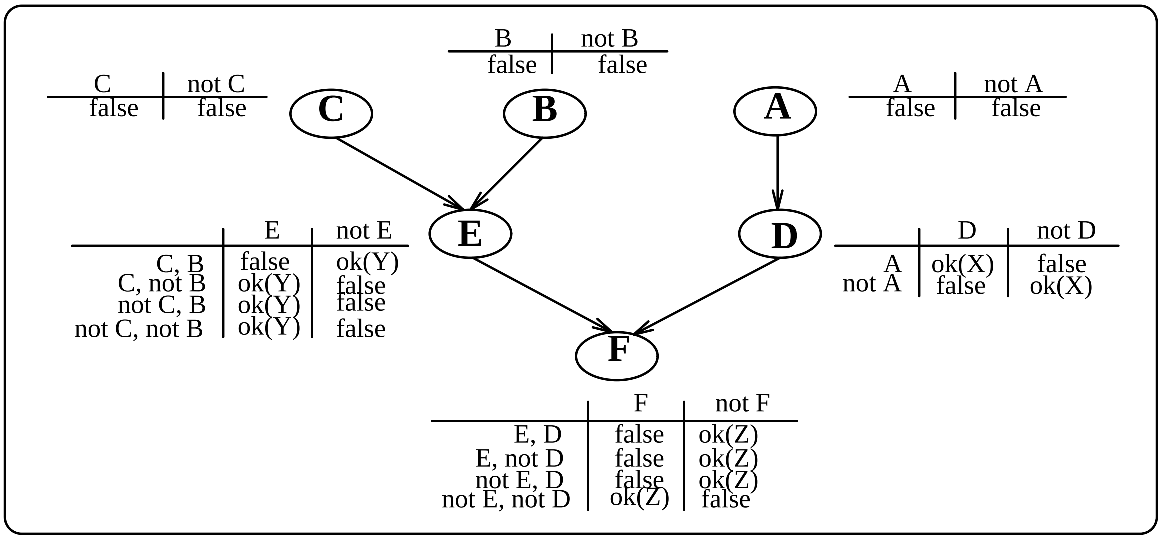

Figure 29: A wob causal network for the digital circuit of Figure 28.

8.6.2 The second example

Figure 30 depicts a digital circuit and the wob causal network describing it. The given quantification assumes the following:

- If an inverter is OK, then it behaves normally.
- If an inverter is faulty, then it shorts its output to its input.

Now, suppose that we observe $\delta = A \wedge \neg C$ about the circuit of Figure 30. Using the wob algorithm, we compute

$$BL_B = \langle ok(X) \vee \neg ok(Y), \neg ok(X) \vee ok(Y) \rangle.$$

Hence,

$$\neg\Phi(\delta) = (\neg ok(X) \wedge ok(Y)) \vee (ok(X) \wedge \neg ok(Y))$$

is the diagnosis of the observation δ.

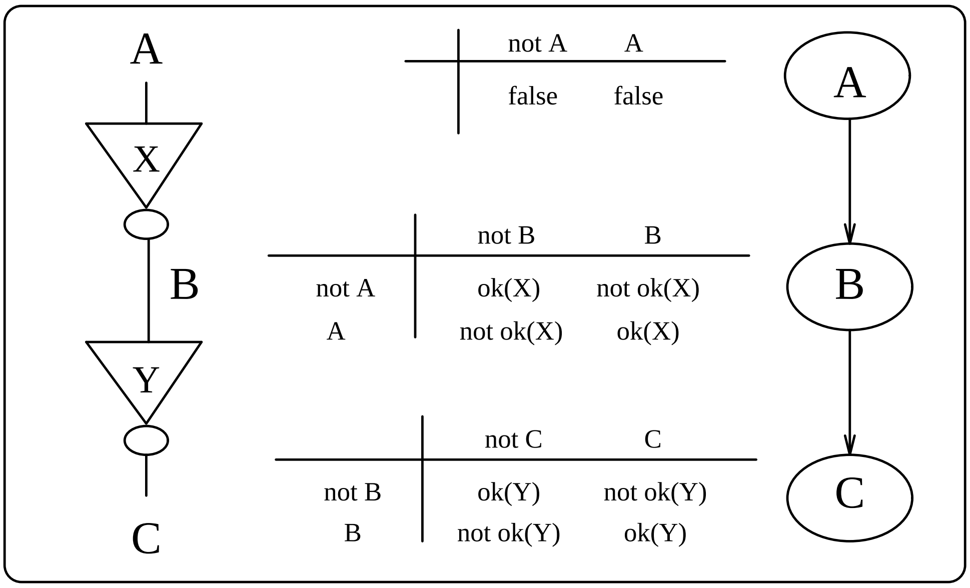

Figure 30: A digital circuit and its corresponding wob causal network.

8.6.3 The third example

Figure 31 depicts a digital circuit and the wob causal network describing it. The given quantification assumes the following:

- If an inverter is OK, then it behaves normally.
- If an inverter is faulty, then it shorts its output to its input, <u>or gets stuck at 0.</u>

Now, suppose that we observe $\delta = A \wedge \neg C$ about the circuit of Figure 31. Using the wob algorithm, we compute

$$BL_B = \langle ok(X), ok(Y) \rangle.$$

Hence,

$$\neg\Phi(\delta) = \neg ok(X) \vee \neg ok(Y)$$

is the diagnosis of the observation δ.

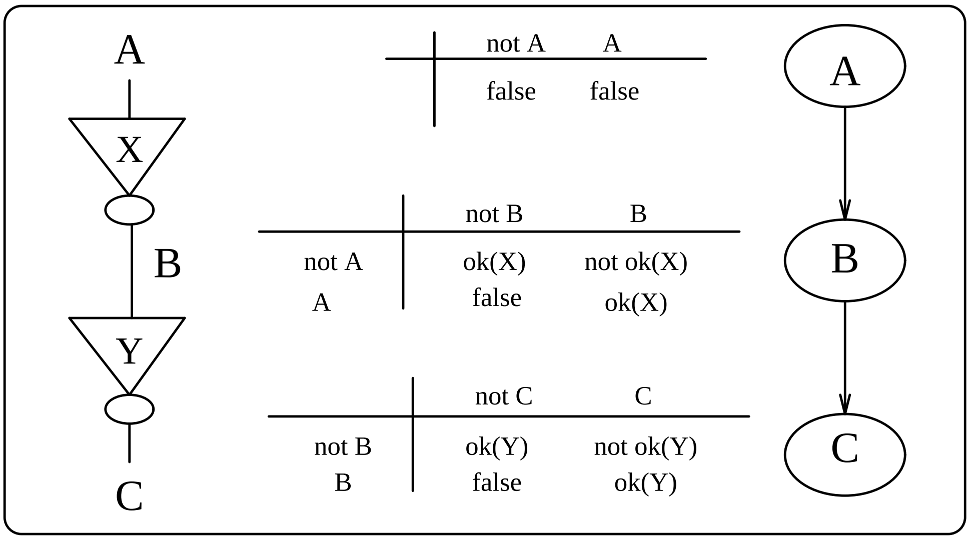

Figure 31: A digital circuit and its corresponding wob causal network.

Chapter 9

Commonly Asked Questions

In this chapter, I answer a number of questions about the relation between the work I presented here and other work in the AI, philosophical, and probabilistic literatures. In each of the following sections, I state one of these questions and answer it.

Notation ABC stands for the Abstract Belief Calculus of Chapters 2 and 3.

9.1 Many–valued logic

Question: Can one view ABC *as a many–valued logic?*

The answer is no. Let me first explain what a many–valued logic is, and then show why ABC cannot be viewed as such.

A many–valued logic is usually characterized by a choice of truth values and a choice of truth value functions F_i. Truth value functions are used as follows: If the truth values $t_1, \ldots, t_n$ are attributed to sentences $\alpha_1, \ldots, \alpha_n$, then the truth value $F_i(t_1, \ldots, t_n)$ is attributed to the sentence $C_i(\alpha_1, \ldots, \alpha_n)$, where C_i is a logical connective [Rosser and Turquette, 1952]. Therefore, in a many–valued logic, the truth value of a sentence is determined by the truth values of its constituent sentences. This is called the "truth functionality" of many–valued logic.

If we view degrees of support as truth values, then ABC is not truth functional. We know, for example, that the support for a conjunction is not determined by the supports for its conjuncts. Therefore, ABC is not a many–valued logic. If it were, then it would not subsume probability calculus:

> Most of the Polish logicians ... regarded the non–truth–functional character of probability assignments as a decisive obstacle against viewing a probabilistic system as a many–valued logic. K. Ajdukiewicz, S. Mazurkiewicz, and A. Tarski were of this mind. [Rescher, 1969, Page 14]

This also explains the basic difference between ABC and some existing many–valued logics in the AI literature [Bonissone, 1987].

Ginsberg proposes a many–valued logic that deviates from the tradition of being completely truth functional [Ginsberg, 1988]. But he still assumes that the truth value of a sentence determines the truth value of its negation. Ginsberg also assumes that the truth functions corresponding to the disjunction and conjunction connectives are idempotent, which precludes probabilities as truth values.

9.2 Fuzzy logic

Question: What is the relation between ABC *and fuzzy logic?*

Before I answer this question, I shall digress on a fundamental notion in fuzzy set theory on which I later base my answer. It is the notion of a linguistic variable [Zadeh, 1975]. Roughly speaking, a linguistic variable consists of the following elements:

- V, a variable.
- U, the exact values of the variable V.
- G, the linguistic values of the variable V.
  These are linguistic descriptions of the value of V.
- M, a meaning function.
  This function maps each linguistic value in G into a fuzzy set on U.

Consider for example the linguistic variable consisting of the variable *Age,* the exact values $[0, 100]$, and the linguistic values *old, very old, more or less old,* and so on. Figure 32 depicts the meaning of two linguistic values of the variable *Age.*

Armed with the concept of a linguistic variable, one can take almost any formalism that deals with variables and construct a fuzzy version of this formalism in which

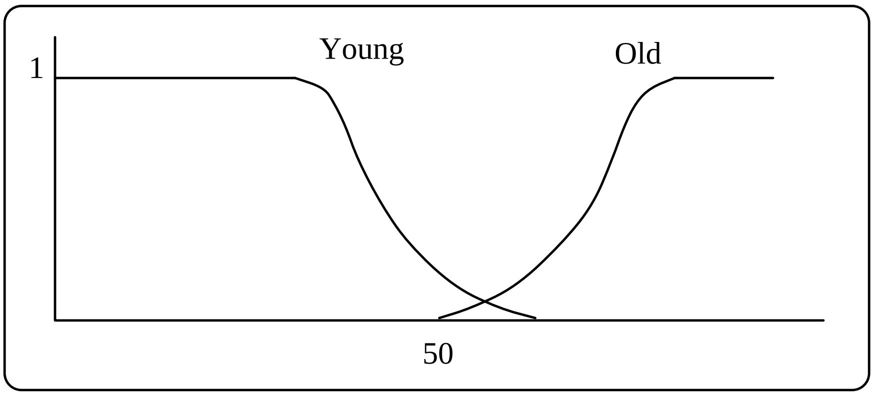

Figure 32: The meanings of two linguistic values.

variables can take on linguistic values. This process is called "fuzzification."[1] In particular, consider a many–valued logic with truth values $\mathcal{S}$. The truth value of a sentence A, $\mathit{Truth\_Value}(A)$, is a variable with an exact value in $\mathcal{S}$. One can fuzzify this logic by allowing the variable $\mathit{Truth\_Value}(A)$ to take on linguistic values. That is, for each sentence A, one can introduce a linguistic variable with the following elements:

- $\mathit{Truth\_Value}(A)$, the variable.
- $\mathcal{S}$, the exact values of $\mathit{Truth\_Value}(A)$.
- *True, very true, almost false, ...,* the linguistic values of $\mathit{Truth\_Value}(A)$.
- A meaning function that maps each linguistic value into a fuzzy set on $\mathcal{S}$.

The fuzzy logic proposed by Zadeh is constructed this way [Zadeh, 1975].

Now back to the question that motivated the above discussion: What is the relation between ABC and fuzzy logic? As it turns out, any concrete instance of ABC can be fuzzified in the same way that a many–valued logic can be fuzzified.

Consider the fuzzification of probability calculus for example. For each sentence A, one can introduces a linguistic variable with the following elements:

- $Pr(A)$, the variable.
- $[0, 1]$, the exact values of $Pr(A)$.
- *Likely, very likely, almost unlikely, ...,* the linguistic values of $Pr(A)$.
- A meaning function that maps each linguistic value into a fuzzy set on $[0, 1]$.

This fuzzification of probability calculus was introduced by Zadeh [Zadeh, 1976], but was not completely developed. I have recently constructed fuzzy probability calculus as an instance of a weaker version of ABC by constructing its support structure [Darwiche, 1993]. Other fuzzy instances of ABC can be constructed similarly.

[1]Fuzzification is useful because people most often use linguistic values as opposed to exact ones. For example, when asked about the age of a person, one is more likely to hear something like *very old* as opposed to eighty–five.

9.3 The Dempster–Shafer theory

Question: Does ABC *subsume the Dempster–Shafer belief functions?*

The answer is no. Let me first state what a belief function is and then explain my answer. A belief function with respect to a propositional language $\mathcal{L}$ is a mapping from $\mathcal{L}$ to the interval $[0,1]$ satisfying the following properties [Shafer, 1976]:

- $Bel(\mathbf{false}) = 0$.
- $Bel(\mathbf{true}) = 1$.
- $Bel(A) = Bel(B)$ *when* $\models A \equiv B$.
- $Bel(\bigvee_{i=1}^{k} A_i) \geq \sum_{\emptyset \neq I \subseteq \{1,\ldots,k\}} (-1)^{|I|+1} Bel(\bigwedge_{i \in I} A_i)$.

Each belief function is associated with a plausibility function Pl, where

$$Pl(A) = 1 - Bel(\neg A).$$

Belief functions violate a basic property of abstract states of belief:

- The support for a disjunction $A \vee B$ is determined by the support for A and the support for B when A and B are logically disjoint.

The definition of a belief function does not entail this property. Therefore, belief functions are not instances of abstract states of belief.

This is a limitation of ABC. The interpretation of belief functions, however, is too controversial for us to know precisely the nature of this limitation. Nonetheless, there is a recent interpretation of belief functions that is formal and intuitive enough to deserve attention here [Halpern and Fagin, 1990; Fagin and Halpern, 1989]. Specifically, a belief function Bel can be viewed as representing all probabilistic states of belief Pr that satisfy the following property:

$$Bel(A) \leq \Pr(A) \leq Pl(A).$$

That is, belief and plausibility are lower and upper bounds on probability.

A major attraction of this view is that belief functions relax the commitment to point probabilities, which are usually hard to assess. In particular, instead of having to attribute a point probability to each sentence, one attributes a probability interval $[Bel(A), Pl(A)]$. Given this view, and given that belief functions are not instances of abstract states of belief, one is then tempted to conclude that degrees of support in ABC cannot be probability intervals. However, this conclusion is not correct. For example, degrees of support in fuzzy probability calculus are fuzzy sets on the interval $[0,1]$. Such fuzzy sets are rich enough to represent probability intervals [Darwiche, 1993]. In particular, the interval $[a,b]$ is represented by the following fuzzy set f:

$$f(p) = \begin{cases} 1, & \text{if } p \text{ belongs to } [a,b]; \\ 0, & \text{otherwise.} \end{cases}$$

9.4 Valuation–based systems

Question: Parkash P. Shenoy and Glenn Shafer have a recent paper titled, "Axioms for Probability and Belief–Function Propagation," where they suggest an abstract framework and axioms under which exact local computation of probabilities is possible. How does your abstract framework relate to theirs?

Let me first describe the essence of their framework and then discuss the relation between the two frameworks. Shenoy and Shafer suggest an abstract model of computation, as opposed to an abstract belief calculus [Shenoy and Shafer, 1990; Shenoy, 1989]. Their model of a computation can be roughly described thus:

- Global information is represented as a collection of local pieces of information. These pieces are called *valuations.*

- Each valuation $\mathcal{V}$ provides information about some variables $\mathcal{H}$. The valuation $\mathcal{V}$ is said to be on variables $\mathcal{H}$ in this case.

- A valuation $\mathcal{V}_1$ on variables $\mathcal{H}_1$ and another valuation $\mathcal{V}_2$ on variables $\mathcal{H}_2$ can be aggregated to produce a valuation on the variables $\mathcal{H}_1 \cup \mathcal{H}_2$. This resulting valuation is called the combination of $\mathcal{V}_1$ and $\mathcal{V}_2$ and is denoted by $\mathcal{V}_1 \overline{\otimes} \mathcal{V}_2$.[2]

- A valuation $\mathcal{V}$ that provides information about variables $\mathcal{H}_1$ can be restricted to provide information about only a subset $\mathcal{H}_2$ of these variables. The result of this restriction is a smaller valuation on the variables $\mathcal{H}_2$. This valuation is called the *marginalization* of $\mathcal{V}$ to $\mathcal{H}_2$ and is denoted by $\mathcal{V}^{\downarrow \mathcal{H}_2}$.

- A *computation* is a process of finding out what some global information $\mathcal{V}$ has to say about a set of variables $\mathcal{H}$. That is, given a collection of valuations $\mathcal{V}_1, \mathcal{V}_2, \ldots$, a computation is a process in which one combines these valuations to produce the global valuation $\mathcal{V}$ and then restricts this valuation to the variables $\mathcal{H}$. Formally, the computation is $(\mathcal{V}_1 \overline{\otimes} \mathcal{V}_2 \overline{\otimes} \ldots)^{\downarrow \mathcal{H}}$.

[2]Shenoy and Shafer use the symbol $\otimes$ to denote combination. I use the symbol $\overline{\otimes}$ instead to avoid confusion with support unscaling in ABC.

These are the basic concepts in the framework of Shenoy and Shafer [Shenoy and Shafer, 1990]. Their basic result is an algorithm for performing the above computation under certain conditions. The conditions concern the variables about which valuations provide information, the properties of combination, and the properties of marginalization. Shenoy and Shafer have also shown that computation in a number of concrete belief calculi can be viewed as a process of combining and marginalizing valuations.

Now that I have briefly described the framework of Shenoy and Shafer, let me explain how it relates to my framework.

Although both frameworks "abstract away" from numbers, the scope and style of their abstractions are different. I am abstracting away from numbers because I am concerned with a belief calculus that is not committed to numbers and yet has the key features of probability calculus—the computational features being only one example. Shenoy and Shafer are abstracting away from numbers because, apparently, they are concerned with a model of computation that is not committed to numbers. More concisely, I am suggesting an abstract belief calculus, while Shenoy and Shafer are suggesting an abstract model of computation.

The difference between these types of abstractions is usually blurred when emphasizing their common concrete instances. But the difference is highlighted when considering the following questions:

1. *What are the consequences of showing that a concrete belief calculus is an instance of one of the abstractions?*
   If a belief calculus is shown to be an instance of ABC, then we conclude, for example, that states of belief in this calculus satisfy the properties given in Chapter 2 and their conditionalizations satisfy the properties given in Chapter 3. Most of these properties cannot even be phrased within the framework of Shenoy and Shafer. Polya's patterns of plausible reasoning are one example. To phrase these patterns, one needs the notion of a degree of belief and the notion of an ordering on degrees of belief. These notions are absent from the framework of Shenoy and Shafer.

2. *To what extent does each abstraction mechanize and guide the creation of concrete belief calculi?*
   We have seen how ABC has guided, for example, the creation of objection calculus in Chapter 7. The framework of Shenoy and Shafer appears too weak to guide such a creation. (This weakness, however, becomes a virtue when the framework is viewed as an abstract model of computation.)

The other difference between my framework and that of Shenoy and Shafer is the style of abstraction. Specifically, the basic notions that constitute my framework and the properties of these notions follow from a set of axioms. For example, the existence of support summation, scaling, and unscaling and the properties of these operations are all shown to be consequences of axioms about states of belief and their conditionalizations. I am not aware of a similar justification for the basic notions underlying the framework of Shenoy and Shafer.

9.5 Abstract theories of probability

Question: Abstracting away from numeric probability is an old tradition. There is a long history of such attempts in the probabilistic literature, and this history is reviewed in a book by Terrence Fine. How does your work relate to these attempts?

There have been many attempts indeed to abstract away from numeric probability [Fine, 1973]. I am aware of three classes of such attempts, which I discuss below.

9.5.1 Modal probability

In the first class of attempts, statements of the form "The probability of A is p" are replaced by statements of the form "A is probable," which are called unconditional modal statements [Walley, 1973]. There are also conditional modal statements, which have the form "A is probable given B." Below are two major questions asked with respect to modal probability.

1. *What axioms should the notion "probable" satisfy?*
   Some suggested axioms are [Walley, 1973]

   - **false** is not probable.
   - Either A is probable or $\neg A$ is probable.
   - If A is probable, and if A entails B, then B is probable.

2. *What is the relation between modal probability and numeric probability?*
   This question is usually addressed by appealing to the notion of *agreement,* which is inspired by the theory of measurement [Krantz *et al.*, 1971]. For example, a modal–probability state of belief is said to agree with a numeric–probability state of belief if A is probable precisely when the probability of A is no less than $1/2$.

9.5.2 Comparative probability

In the second class of attempts, statements of the form "The probability of A is p" are replaced by statements of the form "A is no more probable than B," which are called unconditional comparative statements [Walley, 1973]. There are also conditional comparative statements, which have the form "A given B is no more probable than C given D." Below are two major questions asked with respect to comparative probability.

1. *What axioms should the notion "no more probable than" satisfy?*
   Some suggested axioms are [Koopman, 1940; Fine, 1973; Walley, 1973]
   - **false** is no more probable than any A.
   - If A is no more probable than B, then $\neg B$ is no more probable than $\neg A$.
   - If A given B is no more probable than C given D, and if C given D is no more probable than E given F, then A given B is no more probable than E given F.
2. *What is the relation between comparative probability and numeric probability?*
   This question is again addressed by appealing to the notion of agreement. For example, a comparative–probability state of belief is said to agree with a numeric–probability state of belief if A is no more probable than B precisely when the probability of A is no greater than the probability of B.

The notion of agreement seems to be central in the study of modal and comparative probability. In fact, many of the axioms suggested for these notions seem to have been motivated by agreement results.

9.5.3 Quantitative probability

In the third class of attempts, statements of the form "The probability of A is p," where p is a number in $[0, 1]$, are replaced by statements of the form "The probability of A is a," where a is not necessarily a number. The only attempt in this class that

I am aware of (beyond mine) is that of Romas Aleliunas [Aleliunas, 1988]. In the remainder of this section, I discuss the work of Aleliunas in some detail.

Aleliunas suggests what he calls a "normative theory of belief," which explains how a body of evidence affects one's degree of belief in a possible hypothesis. According to Aleliunas, a probabilistic logic is a scheme for relating a body of evidence to a potential hypothesis in a rational way. Degrees of belief, or probabilities, are assigned to the possible relationships between hypotheses and pieces of evidence, where the relationships are called conditionals. The expression "$f(P|Q)$" is used to denote the conditional probability of P given Q as given by the probability assignment f, where P and Q are sentences in some language $\mathcal{L}$. If the set of probabilities is $\mathcal{P}$, then the goal of a probabilistic logic, according to Aleliunas, is to identify the characteristics of a family $\mathcal{F}$ of functions from $\mathcal{L} \times \mathcal{L}$ to $\mathcal{P}$. Here, $\mathcal{F}$ is the set of permissible probability assignments from which a rational agent is permitted to choose.

Aleliunas provides a number of axioms to constrain the set $\mathcal{F}$, which he divides into three groups: (1) axioms about the domain and range of the probability assignment f in $\mathcal{F}$, (2) axioms stating consistency constraints that each individual f in $\mathcal{F}$ must obey, and (3) axioms about the set $\mathcal{F}$. The axioms are given below.

1. *Axioms about the domain and range of each f in $\mathcal{F}$.*

   (a) The set of probabilities $\mathcal{P}$ is a partially ordered set, where the ordering relation is $\leq$.

   (b) The sentences in $\mathcal{L}$ are finite propositional sentences over a countable set of primitive propositions.

   (c) If $P \equiv X$ and $Q \equiv Y$, then $f(P|Q) = f(X|Y)$.

2. *Axioms that hold for all f in $\mathcal{F}$, and for any P, Q, R in $\mathcal{L}$.* (From here on, $f(P|\mathbf{true})$ is abbreviated by $f(P)$.

   (a) $f(P|\mathbf{false}) = f(P|P)$.

   (b) $f(P \wedge Q|Q) = f(P|Q) \leq f(Q|Q)$.

   (c) For any $g \in \mathcal{F}$, $f(P|P) = g(P|P)$.

(d) There is a monotone non–increasing total function, i, from $\mathcal{P}$ into $\mathcal{P}$ such that $f(\neg P|Q) = i(f(P|Q))$.

(e) There is an order–preserving total function, h, from $\mathcal{P} \times \mathcal{P}$ into $\mathcal{P}$ such that $f(P \wedge Q|R) = h(f(P|Q \wedge R), f(Q|R))$. Moreover, if $f(P \wedge Q|R) = \mathbf{0}$, then either $f(P|Q \wedge R) = \mathbf{0}$, or $f(Q|R) = \mathbf{0}$, where $\mathbf{0}$ is defined to be $f(\neg P|P)$.

(f) If $f(P|R) \leq f(P|\neg R)$, then $f(P|R) \leq f(P) \leq f(P|\neg R)$.

3. *Axioms about the set $\mathcal{F}$.*
For any distinct primitive propositions A, B and C in $\mathcal{L}$, and for any probabilities a, b and c in $\mathcal{P}$, there is f in $\mathcal{F}$ such that

(a) $f(A) = a$, $f(B|A) = b$, and $f(C|A \wedge B) = c$.

(b) $f(A|B) = f(A|\neg B) = a$ and $f(B|A) = f(B|\neg A) = b$.

(c) $f(A) = a$ and $f(A \wedge B) = b$ whenever $b \leq a$.

Below are some basic differences between the above axioms and those underlying ABC. First, Aleliunas does not require the existence of a function h^* such that for any f, P, Q, and R,

$$f(P \vee Q|R) = h^*(f(P|R), f(Q|R)) \text{ when } \models \neg(P \wedge Q).$$

The function h^* would correspond to support summation in ABC if it existed. Therefore, Aleliunas does not require a probability summation function. Moreover, since probability summation h^* does not exist, the partial order on the set of probabilities had to be assumed, as opposed to being derived as in ABC.

Second, Aleliunas remarks that his axioms imply the existence of a probability r that satisfies $a = h(r, b)$ whenever $a \leq b$. But he also remarks that r need not be unique. Since h corresponds to support unscaling in ABC, this means that Aleliunas does not require the existence of a unique support scaling function. Therefore, the conditionalization of a state of belief is not well defined.

Third, Aleliunas requires the function h to be total. In ABC, however, support unscaling in not necessarily a total function. The significance of this issue is discussed at length in Section 3.4.1.

Fourth and finally, Axiom (2d) of Aleliunas does not hold in ABC. That is, the support for a sentence does not determine the support for the negation of that sentence in general. If this axiom held in ABC, then ABC would not have subsumed possibility and objection calculi.

Chapter 10

Concluding Remarks

The basic result of this thesis has been that the commitment to numbers is not needed for obtaining the key features of probability calculus: Multiple degrees of belief, conditionalization, independence, causal networks, and local computation. In this chapter, I shall briefly summarize this basic result and then discuss some questions that remain to be answered.

10.1 Summary of the thesis

In Chapter 1, I pointed out that many people have mixed feelings about probability calculus. On the one hand, people are usually impressed by the key features of this calculus. On the other hand, they find the commitment to numbers a very high price to pay for these features.

In Chapter 2, I formalized the notion of an abstract state of belief, which is an attribution of abstract degrees of belief to propositional sentences. Abstract states of belief subsume their probabilistic counterparts but are not committed to either numeric or symbolic degrees of belief.

In Chapter 3, I formalized the notion of abstract conditionalization, which is a process for changing an abstract state of belief to accommodate an observation. Abstract conditionalization subsumes its probabilistic counterpart and leads to some celebrated patterns of plausible reasoning.

In Chapter 4, I formalized the notion of an abstract causal network, which is a graphical construct used in communicating an abstract state of belief. Abstract causal networks subsume probabilistic causal networks.

In Chapter 5, I presented the abstract polytree algorithm for computing beliefs in abstract causal networks, and, in Chapter 6, I implemented the algorithm using Common Lisp. This algorithm subsumes its probabilistic counterpart.

Chapters 2–6, therefore, comprise a comprehensive, abstract belief calculus that looks very much like probability calculus but is not committed to numbers.

In Chapter 7, I presented objection calculus, which is a symbolic instance of the abstract belief calculus. Objection calculus concretely demonstrates that multiple degrees of belief, conditionalization, independence, causal networks, and local computation are not features exclusive to probability calculus.

In Chapter 8, I explored some practical ramifications of the suggested theory of uncertain states of belief. In particular, I showed that clause management and diagnosis systems can be viewed as instances of the abstract belief calculus. This made the tools of the abstract calculus available to clause management and diagnosis applications, and I gave examples of this availability.

10.2 Technical limitations

I have made some assumptions while developing the theory of uncertain states of belief. Some of these assumptions were intended to simplify the presentation and can be relaxed in a straightforward manner. Other assumptions are more involved, and can be relaxed only if more research is conducted. In this section, I discuss some of the assumptions I have made.

Primitive propositions are binary. I assumed that primitive propositions can have only two values, *true* and *false.* In the probabilistic literature, however, primitive propositions are usually multi–valued. For example, *Todays_Weather* is a multi–valued primitive proposition that may have the values *Cloudy, Rainy,* and *Sunny.* My assumption here can be relaxed without losing any of the established results. This is true because every state of belief over multi–valued propositions can be represented by a state of belief over binary propositions.

The domain of a state of belief is a propositional language. This assumption can be relaxed in principle: An abstract state of belief can be an attribution of degrees of support to first–order sentences. Relaxing this assumption, however, leads to representational complexity. In particular, a state of belief can no longer be characterized by the degrees of support that it attributes to complete sentences—a property that is assumed by a number of proofs. Moreover, it is not clear how to extend the definition of abstract causal networks when this assumption is relaxed without compromising the usefulness of such networks.

Objections cannot be observed. As I mentioned in Chapter 7, this assumption is there because objections are degrees of support. In the theory of uncertain states of belief, it is not meaningful to observe quantities. Relaxing this assumption takes us, therefore, beyond the realm of the current theory. Moreover, relaxing this assumption undermines some of the results that I established in Chapters 7 and 8. For example, the database corresponding to an objection–based state of belief would no longer be non–committal in general (see Theorem 8.3.2).

10.3 Current and future work

Below are some research areas that, I believe, deserve immediate attention.

Uncertain observations The conditionalization on uncertain observations — that is, changing a state of belief such that a sentence becomes supported to some degree — needs to be explored. The uncertainty literature contains a number of proposals for conditionalizing concrete states of belief on uncertain observations. One can adopt some of these proposals to conditionalize abstract states of belief on uncertain observations. But most interesting will be characterizing these proposals in the same way I have characterized states of belief and their conditionalizations in Chapters 2 and 3. That is, identifying and formalizing properties of belief change that are equivalent to the proposals under consideration.

Sufficient supports We have seen in objection calculus, for example, that sufficient objections play an important role in extracting objections. In fact, sufficient objections are more important than conditional objections in this regard. The question is, Can we define the notion of sufficient supports in the abstract calculus? I believe so. And will this notion be as important as the notion of sufficient objections? Again, I believe so. This line of questioning about sufficient supports may also lead to a weak notion of independence in the abstract calculus.

Theoretical questions Finally, there are some theoretical questions whose answers would enhance our understanding of the current theory of uncertain states of belief. For example, What additional properties of belief change would guarantee the uniqueness of support scaling? Are the properties of belief change independent? Can they be reduced, simplified, or even beautified?

Appendix A

Propositional Logic

Abstract states of belief are among the most fundamental notions in this thesis. An abstract state of belief maps a standard propositional language into a set containing degrees of belief. By a standard propositional language I mean a language with the following syntax and semantics.

Syntax

The *propositional language* $\mathcal{L}$ with respect to *primitive propositions* $P_1, \ldots, P_n$ is constructed as follows:

1. $P_i \in \mathcal{L}$.
2. If A is in $\mathcal{L}$, then so is $\neg A$.
3. If A and B are in $\mathcal{L}$, then so is $A \vee B$.

Other logical connectives are defined as usual:

$$\begin{aligned} A \wedge B &\stackrel{def}{=} \neg(\neg A \vee \neg B) \\ A \supset B &\stackrel{def}{=} \neg A \vee B \\ A \equiv B &\stackrel{def}{=} (A \wedge B) \vee (\neg A \wedge \neg B). \end{aligned}$$

Semantics

A *truth assignment* with respect to primitive propositions $P_1, \ldots, P_n$ is a mapping w from $\{P_1, \ldots, P_n\}$ to $\{$*true, false*$\}$. The notion of *satisfaction* by a truth assignment is defined as follows:

1. $w \models P_i$ if $w(P_i) = \mathit{true}$.

2. $w \models \neg A$ if $w \not\models A$.

3. $w \models A \vee B$ if $w \models A$ or $w \models B$.

The satisfaction relation induces a *meaning function* $\mathcal{M}$ on sentences, where the meaning of a sentence is the set of truth assignments that satisfy it. That is, $\mathcal{M}(A)$ contains all truth assignments that satisfy A.

Complete sentences

A *complete sentence* over a set of primitive propositions I is a sentence of the form

$$\bigwedge_{i \in I} L_i,$$

where L_i is the literal i or $\neg i$. For example, if $I = \{A, B\}$, then

$$A \wedge B, \;\; A \wedge \neg B, \;\; \neg A \wedge B, \;\; \neg A \wedge \neg B,$$

are all the complete sentences over I. In general, there are 2^n complete sentences over a set of n propositions.

A complete sentence over primitive propositions I is denoted by $\underline{I}$. By definition, **true** is the complete sentence over the empty set of propositions, $\underline{\emptyset}$.

A complete sentence over primitive propositions I is also called a *state* of propositions I because it fixes the truth of each proposition in I. There is a one–to–one correspondence between complete sentences over a set of primitive propositions and truth assignments over the same set of propositions.

Logical falsification

The *falsification* of sentence A given sentence B is the sentence $A \wedge \neg B$. The notion of falsification plays an important role in Chapter 7. Its importance stems from the following characterization of falsification:

$$(B \models \alpha \equiv \textbf{false} \text{ and } \neg B \models \alpha \equiv A) \text{ precisely when } \models \alpha \equiv A \wedge \neg B.$$

That is, the falsification of A given B is **false** if B holds and A if B does not hold.

I adopt the following terminologies and definitions:

- A is *unsatisfiable,* written $\models \neg A$, if $\mathcal{M}(A)$ is the empty set.
- A is *satisfiable* if $\mathcal{M}(A)$ is not the empty set.
- A is *valid,* written $\models A$, if $\mathcal{M}(A)$ is the set of all truth assignments.
- **true** denotes a valid sentence.
- **false** denotes an unsatisfiable sentence.
- A and B are *equal* if $\models A \equiv B$.
- A and B are *logically disjoint* if $\models \neg(A \wedge B)$.
- A and B are *logically exhaustive* if $\models A \vee B$.
- A *entails,* or *implies,* B, written $A \models B$, if $\mathcal{M}(A) \subseteq \mathcal{M}(B)$.
- A is *stronger* than B precisely when A entails B.
- A is *weaker* than B precisely when B entails A.
- A *world* is a truth assignment.
- A *database* is a sentence.
- Database Δ is *consistent* if $\mathcal{M}(\Delta)$ is not the empty set.
- Database Δ *entails* sentence A if $\mathcal{M}(\Delta) \subseteq \mathcal{M}(A)$.

Appendix B

Notational Conventions

$\|N\|$	The cardinality of set N.
$\mathcal{L}(N)$	A propositional language with respect to primitive propositions N.
$\underline{I}$	A complete sentence over (or state of) primitive propositions I.
$j \rightarrow k$	Node j is a parent of node k.
$i\diamond$	Parents of node i.
$i\diamond j$	Parents of node i excluding parent j.
$i\circ$	Children of node i.
$i\circ k$	Children of node i excluding child k.
$i\triangleright$	Descendants of node i.
$i \triangleright k$	Descendants of node i that are connected to it via arc $i \rightarrow k$.
$i \triangleright \overline{k}$	Descendants of node i that are connected to it via any arc except $i \rightarrow k$.
$i\triangleleft$	Non–descendants of node i.
$i \triangleleft j$	Non–descendants of node i that are connected to it via arc $j \rightarrow i$.
$i \triangleleft \overline{j}$	Non–descendants of node i that are connected to it via any arc except $j \rightarrow i$.

Appendix C

Proofs of Chapter 2

Below are the formal statements of Axioms 1–5. Each state of belief $\Phi : \mathcal{L} \to \mathcal{S}$ satisfies the following:

Axiom 1 $\Phi(A) = \Phi(B)$ when $\models A \equiv B$.

Axiom 2 $\Phi(A \vee B) = \Phi(A) \oplus \Phi(B)$ when $\models \neg(A \wedge B)$, where $\oplus$ is some function.

Axiom 3 $\Phi(A) = \Phi(C)$ only if $\Phi(A) = \Phi(B)$ when $A \models B \models C$.

Axiom 4 $\Phi(\mathbf{false}) = \mathbf{0}$, where $\mathbf{0}$ is some degree of support.

Axiom 5 $\Phi(\mathbf{true}) = \mathbf{1}$, where $\mathbf{1}$ is some degree of support and $\mathbf{1} \neq \mathbf{0}$.

Axiom 1 says that syntax does not matter but meaning does. As a result of this, in the following proofs and in others, I view a state of belief as a mapping from 2^W into $\mathcal{S}$, where W is the set of all truth assignments of the language $\mathcal{L}$. That is, I make no distinction between a sentence and its meaning, although I occasionally refer to the meaning as a proposition.

Proof of Theorem 2.2.1

Using Axiom 3 and taking C to be A, we get

$$A \models B \models A \text{ only if } \Phi(A) = \Phi(B).$$

Therefore,

$$\models A \equiv B \text{ only if } \Phi(A) = \Phi(B). \blacksquare$$

Proof of Theorem 2.3.5 Assume Axioms 1 and 2.

(X0): Let a and b be two supports where $a \oplus b$ is meaningful. There must exist a "state of belief" Φ, and logically disjoint propositions A and B, such that $\Phi(A) = a$ and $\Phi(B) = b$. It follows that $\Phi(A \cup B) = a \oplus b$ and $\Phi(B \cup A) = b \oplus a$. Since $(A \cup B) = (B \cup A)$, we have $a \oplus b = b \oplus a$.

(X1): Let a, b and c be three supports where $(a \oplus b) \oplus c$ is meaningful. There must exist a "state of belief" Φ, and logically disjoint propositions A, B and C, such that $\Phi(A) = a, \Phi(B) = b$ and $\Phi(C) = c$. It follows that $\Phi((A \cup B) \cup C) = (a \oplus b) \oplus c$ and $\Phi(A \cup (B \cup C)) = a \oplus (b \oplus c)$. Since $((A \cup B) \cup C) = (A \cup (B \cup C))$, we have $(a \oplus b) \oplus c = a \oplus (b \oplus c)$. $\blacksquare$

Proof of Theorem 2.3.6 Assume Axioms 1 and 2.

Assume Axiom 3. Let a, b, and c be three supports such that $a \oplus b \oplus c$ is meaningful and $a \oplus b \oplus c = a$. There must exist a "state of belief" Φ, and logically disjoint propositions A, B and C, such that $\Phi(A) = a, \Phi(B) = b$ and $\Phi(C) = c$. It follows that $\Phi(A) = a = \Phi(A \cup B \cup C)$. Since $A \subseteq A \cup B \subseteq A \cup B \cup C$, we must have $\Phi(A \cup B) = a$. Therefore, $a \oplus b = a$.

Assume (X2). Suppose that $A \subseteq B \subseteq C$ and $\Phi(A) = \Phi(C)$. There must exist B' and C' such that $B = A \cup B'$, and $C = B \cup C'$. Moreover, $\Phi(C) = \Phi(A \cup B' \cup C') = \Phi(A) \oplus \Phi(B') \oplus \Phi(C')$. By supposition, we have $\Phi(A) \oplus \Phi(B') \oplus \Phi(C') = \Phi(A)$, and by assumption, we have $\Phi(A) \oplus \Phi(B') = \Phi(A)$. But $\Phi(A) \oplus \Phi(B') = \Phi(A \cup B') = \Phi(B)$. Therefore, $\Phi(A) = \Phi(B)$. $\blacksquare$

Proof of Theorem 2.3.7 Assume Axioms 1, 4, and 2.

Let a be any support, and choose a "state of belief" Φ and proposition A, such that $\Phi(A) = a$. It follows that $a = \Phi(A) = \Phi(A \cup \emptyset) = \Phi(A) \oplus \Phi(\emptyset) = a \oplus \mathbf{0}$. Therefore, $\mathbf{0}$ is an identity element for support summation. Now, suppose there was another identity element $\mathbf{0}'$. Then $\mathbf{0} = \mathbf{0} \oplus \mathbf{0}' = \mathbf{0}'$, and the identity element $\mathbf{0}$ is unique. ∎

Proof of Theorem 2.3.8 Assume Axioms 1, 5, 2, and 3.

Let a be any support and choose a "state of belief" Φ and a proposition A such that $\Phi(A) = a$. We have that $\Phi(A) \oplus \Phi(\overline{A}) = \Phi(A \cup \overline{A})$, and therefore $a \oplus \Phi(\overline{A}) = \mathbf{1}$. Now, suppose there was another support $\mathbf{1}'$ such that for any support a there is a support b and $a \oplus b = \mathbf{1}'$. Then there exists a support c where $\mathbf{1} \oplus c = \mathbf{1}'$. Moreover, there should exist a support d where $\mathbf{1}' \oplus d = \mathbf{1}$. That is, $(\mathbf{1} \oplus c) \oplus d = \mathbf{1}$ and $\mathbf{1} \oplus c = \mathbf{1}$. Therefore, $\mathbf{1} = (\mathbf{1} \oplus c) = \mathbf{1}'$ and $\mathbf{1}$ is unique. ∎

Proof of Theorem 2.5.2

- *Reflexive.* From $a \oplus \mathbf{0} = a$, we conclude that $a \preceq_{\oplus} a$.

- *Antisymmetric.* Suppose $a \preceq_{\oplus} b$ and $b \preceq_{\oplus} a$. Then

$$\begin{array}{rcll} \exists c:\ a \oplus c & = & b & \text{by def of } \preceq_{\oplus} \\ \exists d:\ b \oplus d & = & a & \text{by def of } \preceq_{\oplus} \\ (a \oplus c) \oplus d & = & a & \text{by substituting for } b \\ a \oplus c & = & a & \text{by (X2)} \\ b & = & a. & \end{array}$$

- *Transitive.* Suppose $a \preceq_{\oplus} b$ and $b \preceq_{\oplus} c$. Then

$$\begin{array}{rcll} \exists d:\ a \oplus d & = & b & \text{by def of } \preceq_{\oplus} \\ \exists e:\ b \oplus e & = & c & \text{by def of } \preceq_{\oplus} \\ (a \oplus d) \oplus e & = & c & \text{by substituting for } b \\ a \oplus (d \oplus e) & = & c & \text{by (X1)} \\ a & \preceq_{\oplus} & c & \text{by def of } \preceq_{\oplus}. \end{array}$$

- $\mathbf{0} \preceq_{\oplus} a$ for all $a \in \mathcal{S}$.

  For all a, we have $\mathbf{0} \oplus a = a$. Thus, $\mathbf{0} \preceq_{\oplus} a$.

- $a \preceq_{\oplus} \mathbf{1}$ for all $a \in \mathcal{S}$.

  For all a, there is b, $a \oplus b = \mathbf{1}$. Thus, $a \preceq_{\oplus} \mathbf{1}$. ∎

Proof of Theorem 2.5.4

- *Reflexive.* From $s_1 \preceq_\oplus s_1$ and $s_2 \preceq_\oplus s_2$, we conclude that $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle s_1, s_2\rangle$.
- *Antisymmetric.* Suppose $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle s_3, s_4\rangle$ and $\langle s_3, s_4\rangle \sqsubseteq_\oplus \langle s_1, s_2\rangle$. Then $s_1 \preceq_\oplus s_3$, $s_4 \preceq_\oplus s_2$, $s_3 \preceq_\oplus s_1$, and $s_2 \preceq_\oplus s_4$. By antisymmetry of $\preceq_\oplus$, we have $s_1 = s_3$ and $s_2 = s_4$. Therefore, $\langle s_3, s_4\rangle = \langle s_1, s_2\rangle$.
- *Transitive.* Suppose $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle s_3, s_4\rangle$ and $\langle s_3, s_4\rangle \sqsubseteq_\oplus \langle s_5, s_6\rangle$. Then $s_1 \preceq_\oplus s_3 \preceq_\oplus s_5$ and $s_6 \preceq_\oplus s_4 \preceq_\oplus s_2$. By transitivity of $\preceq_\oplus$, we have $s_1 \preceq_\oplus s_5$ and $s_6 \preceq_\oplus s_2$. Therefore, $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle s_5, s_6\rangle$.
- $\langle \mathbf{0}, \mathbf{1}\rangle \sqsubseteq_\oplus \langle s_1, s_2\rangle$ for every degree of belief $\langle s_1, s_2\rangle$.

  For all degrees of support s_1 and s_2, we have $\mathbf{0} \preceq_\oplus s_1$ and $s_2 \preceq_\oplus \mathbf{1}$. Thus, $\langle \mathbf{0}, \mathbf{1}\rangle \sqsubseteq_\oplus \langle s_1, s_2\rangle$.
- $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle \mathbf{1}, \mathbf{0}\rangle$ for every degree of belief $\langle s_1, s_2\rangle$.

  For all degrees of support s_1 and s_2, we have $s_1 \preceq_\oplus \mathbf{1}$ and $\mathbf{0} \preceq_\oplus s_2$. Thus, $\langle s_1, s_2\rangle \sqsubseteq_\oplus \langle \mathbf{1}, \mathbf{0}\rangle$.

Proof of Theorem 2.5.7

1. $\sqsubseteq_\Phi$ is a partial ordering:

   - *Reflexive:* By reflexivity of $\preceq_\Phi$, we have $A \preceq_\Phi A$ and $\neg A \preceq_\Phi \neg A$. Hence $A \sqsubseteq_\Phi A$.
   - *Transitive:* Suppose that $A \sqsubseteq_\Phi B$ and $B \sqsubseteq_\Phi C$. That is, $A \preceq_\Phi B$, $\neg B \preceq_\Phi \neg A$, $B \preceq_\Phi C$, $\neg C \preceq_\Phi \neg B$. By transitivity of $\preceq_\Phi$, we have $A \preceq_\Phi C$ and $\neg C \preceq_\Phi \neg A$. Hence $A \sqsubseteq_\Phi C$.
   - *Antisymmetric:* Suppose that $A \sqsubseteq_\Phi B$ and $B \sqsubseteq_\Phi A$. That is, $A \preceq_\Phi B$, $\neg B \preceq_\Phi \neg A$, $B \preceq_\Phi A$, $\neg C \preceq_\Phi \neg A$. By antisymmetry of $\preceq_\Phi$, we have $\Phi(A) =_\oplus \Phi(C)$ and $\Phi(\neg C) =_\oplus \Phi(\neg A)$. Hence $A =_\Phi C$.

2. $A \models B$ only if $A \sqsubseteq_\Phi B$: Suppose $A \models B$. Then $C = \neg A \wedge B$ is such that $\models \neg(A \wedge C)$ and $\models (A \vee C) \equiv B$. It follows that $\Phi(A) \oplus \Phi(C) = \Phi(B)$ and $A \preceq_\Phi B$. Since $\neg B \models \neg A$, then $C = \neg A \wedge B$ is such that $\models \neg(\neg B \wedge C)$ and $\models (\neg B \vee C) \equiv \neg A$. It follows that $\Phi(\neg B) \oplus \Phi(C) = \Phi(\neg A)$ and $\neg B \preceq_\Phi \neg A$.

3. $\mathbf{false} \sqsubseteq_\Phi A \sqsubseteq_\Phi \mathbf{true}$: Follows from $\mathbf{false} \models A \models \mathbf{true}$.

4. A is minimal under $\sqsubseteq_\Phi$ if and only if A is rejected: Suppose that A is minimal under $\sqsubseteq_\Phi$. Then $A \sqsubseteq_\Phi \mathbf{false}$ and $A \preceq_\Phi \mathbf{false}$. That is, $\Phi(A) \preceq_\oplus \Phi(\mathbf{false}) = \mathbf{0}$. Hence $\Phi(A) = \mathbf{0}$ and A is rejected. Now suppose that $\Phi(A) = \mathbf{0}$. Then $\Phi(\neg A) = \mathbf{1}$, $\Phi(A) \preceq_\oplus \Phi(\mathbf{false})$, $\Phi(\neg A) \preceq_\oplus \Phi(\mathbf{true})$, and $A \sqsubseteq_\Phi \mathbf{false}$. Hence A is minimal under $\sqsubseteq_\Phi$.

5. A is maximal under $\sqsubseteq_\Phi$ if and only if A is accepted: Suppose that A is maximal under $\sqsubseteq_\Phi$. Then $\mathbf{true} \sqsubseteq_\Phi A$, $\mathbf{true} \preceq_\Phi A$, and $\neg A \preceq_\Phi \mathbf{false}$. That is, $\Phi(\neg A) \preceq_\oplus \Phi(\mathbf{false}) = \mathbf{0}$, $\Phi(\neg A) = \mathbf{0}$, A is rejected, and A is accepted. Now suppose that A is accepted by Φ. Then $\Phi(\neg A) = \mathbf{0}$ and $\Phi(A) = \mathbf{1}$. That is, $\Phi(\neg A) \preceq_\oplus \Phi(\mathbf{false})$, $\Phi(\mathbf{true}) \preceq_\oplus \Phi(A)$, $\Phi(\mathbf{true}) \sqsubseteq_\Phi \Phi(A)$. Hence A is maximally supported. ∎

Proof of Theorem 2.6.3

Theorem 2.6.3 is a special case of Theorem 2.6.13, which is proved later. States of belief with respect to $\langle\{0,\infty\},\min\rangle_{\mathcal{L}}$ are a special case of Spohnian states of belief, and are isomorphic to propositional states of belief. That is, a state of belief Φ with respect to $\langle\{0,\infty\},\min\rangle_{\mathcal{L}}$ accepts A precisely when $\Phi(\neg A) = \infty$, which is precisely when $\Phi(\neg A) \neq 0$. ∎

Proof of Theorem 2.6.4 Numeric addition is commutative and associative. If $a+b+c=a$ then $b+c=0$ and $b=0$, since $a,b,c \in [0,1]$. Therefore, $a+b=a$. $a \preceq_{\oplus} b$ iff there is $c \in [0,1]$ such that $a+c=b$, that is, iff $a \leq b$. For all $a \in [0,1]$, $a+0=a$ and $a+(1-a)=1$ — 0 and 1 are the only members in [0,1] that satisfy this property. ∎

Proof of Theorem 2.6.5 The function $(\lambda\,(a\;b)\,a+b-1)$ is clearly commutative and associative. If $(a+b-1)+c-1=a$ then $b+c=2$ and $b=1$, since $a,b,c \in [0,1]$. Therefore, $a+b-1=a$. $a \preceq_{\oplus} b$ iff there is $c \in [0,1]$ such that $a+c-1=b$, that is, iff $a \geq b$. For all $a \in [0,1]$, $a+1-1=a$ and $a+(1-a)-1=0$ — 1 and 0 are the only members in [0,1] that satisfy this property. ∎

Proof of Theorem 2.6.6 The function max is commutative and associative. If $\max(\max(a,b),c)=a$, then $a \geq b$ and $a \geq c$. And if $\max(a,b) \neq a$, then $b > a$. Thus, we cannot have $\max(\max(a,b),c)=a$ and $\max(a,b) \neq a$. $a \preceq_{\oplus} b$ iff there is c such that $\max(a,c)=b$, that is, iff $a \leq b$. For all $a \in [0,1]$, $\max(a,0)=a$ and $\max(a,1)=1$— 1 and 0 are the only members in [0,1] that satisfy these properties. ∎

Proof of Theorem 2.6.8 The function min is commutative and associative. If $\min(\min(a,b),c)=a$, then $a \leq b$ and $a \leq c$. And if $\min(a,b) \neq a$, then $b < a$. Thus, we cannot have $\min(\min(a,b),c)=a$ and $\min(a,b) \neq a$. $a \preceq_{\oplus} b$ iff there is c such that $\min(a,c)=b$, that is, iff $a \geq b$. For all $a \in [0,1]$, $\min(a,1)=a$ and $\min(a,0)=0$— 0 and 1 are the only members in [0,1] that satisfy these properties. ∎

Proof of Theorem 2.6.13

Let $\mathcal{M}_\mu(\Delta) = \mathcal{M}(\Delta) \cap W_k$ be the preferred meaning of Δ. Note that k is the least integer such that $\mathcal{M}(\Delta) \cap W_k$ is not empty. Now, consider the following Spohnian state of belief:

$$\Phi^\Delta(w) = \begin{cases} i-k, & \text{if } w \in \mathcal{M}(\Delta) \cap W_i; \\ \infty, & \text{otherwise.} \end{cases}$$

By definition, $\Phi^\Delta(w) = 0$ if and only if w is in $\mathcal{M}_\mu(\Delta)$. I will now show that for all A, Δ preferentially entails A precisely when Φ^Δ accepts A, or accepts A by default ($\Phi^\Delta(\overline{A}) \neq 0$).

- Suppose that Δ preferentially entails A. It follows that $\mathcal{M}_\mu(\Delta) \subseteq A$, and there is no world w in $\overline{A}$ such that $\Phi^\Delta(w) = 0$. Therefore, $\Phi^\Delta(\overline{A}) = \min_{w \in \overline{A}} \Phi^\Delta(w)$ cannot be 0. Hence $\Phi^\Delta(\overline{A}) > 0$ and Φ either accepts A, or accepts it by default.

- Suppose that $\Phi^\Delta(\overline{A}) = \min_{w \in \overline{A}} \Phi^\Delta(w) \neq 0$. Then for all w in $\overline{A}$, $\Phi(w) \neq 0$. Also, if $\Phi(w) = 0$, then $w \in A$. Hence $\mathcal{M}_\mu(\Delta) \subseteq A$ and Δ preferentially entails A. ∎

Appendix D

Proofs of Chapter 3

Below are the formal statements of Axioms 6–12.

Axiom 6 $\Phi_{A\vee B}(A) = \Phi(A) \oslash \Phi(A \vee B)$ for some function $\oslash$.

Axiom 7 $\Phi_A(B) = \mathbf{0}$ when $\Phi(A) \neq \mathbf{0}$ and $\Phi(B) = \mathbf{0}$.

Axiom 8 $\Phi_A = \Phi$ when $\Phi(A) = \mathbf{1}$.

Axiom 9 $\Phi(A) \neq (\not\preceq_\oplus)\ \Phi'(A)$ only if $\Phi_{A\vee B}(A) \neq (\not\preceq_\oplus)\ \Phi'_{A\vee B}(A)$ when $\Phi(A \vee B) = \Phi'(A \vee B)$.

Axiom 10 $\Phi(A) \preceq_\oplus \Phi_{A\vee B}(A)$ when $\Phi(A \vee B) \neq \mathbf{0}$.

Axiom 11 $\Phi_{B\wedge C}(A) = \Phi_C(A)$ only if $\Phi_{A\wedge C}(B) = \Phi_C(B)$ when $\Phi(A \wedge B \wedge C) \neq \mathbf{0}$.

Axiom 12 For all Φ_C and $c \neq \mathbf{0}$, there is Φ such that $\Phi(C) = c$.

Three lemmas for Theorem 3.1.4

Lemma D.0.1 If $e \oplus f = \mathbf{0}$, then $e = \mathbf{0}$.

Proof Assume $e \oplus f = \mathbf{0}$. Then $e \oplus \mathbf{0} \oplus f = \mathbf{0}$, $e \oplus \mathbf{0} = \mathbf{0}$, and $e = \mathbf{0}$. ∎

Lemma D.0.2 If W is not rejected by Φ, then $\Phi_W(w) = \mathbf{0}$ for all $w \notin W$.

Proof By Definition 3.1.2, we have $\Phi_W(\overline{W}) = \mathbf{0}$. Therefore, $\oplus_{w \in \overline{W}} \Phi(w) = \mathbf{0}$. And by Lemma D.0.1, we have that $\Phi(w) = \mathbf{0}$ for all $w \in \overline{W}$. ∎

Lemma D.0.3 If W is not rejected by Φ, then $\Phi_W(W) = \mathbf{1}$.

Proof From $\Phi_W(\overline{W}) = \mathbf{0}$ and $\Phi_W(W) \oplus \Phi_W(\overline{W}) = \mathbf{1}$, we have $\Phi_W(W) \oplus \mathbf{0} = \mathbf{1}$. Therefore, $\Phi_W(W) = \mathbf{1}$. ∎

Proof of Theorem 3.1.4

$\Longrightarrow$ Assume Axiom 6. We have

$$\Phi_A(B) = \Phi_A(B \cap A) \oplus \Phi_A(B \cap \neg A).$$

Given Axiom 6, and since $A \cap B \subseteq A$, we have

$$\Phi_A(B \cap A) = \Phi(B \cap A) \oslash \Phi(A),$$

for some function $\oslash : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$. Moreover, by Lemma D.0.2, we have

$$\begin{aligned} \Phi_A(B \cap \neg A) &= \bigoplus_{w \in B \cap \neg A} \Phi_A(w) \\ &= \bigoplus_{w \in B \cap \neg A} \mathbf{0} \\ &= \mathbf{0}. \end{aligned}$$

Therefore,

$$\begin{aligned} \Phi_A(B) &= \Phi_A(B \cap A) \oplus \mathbf{0} \\ &= \Phi_A(B \cap A) \\ &= \Phi(B \cap A) \oslash \Phi(A). \end{aligned}$$

$\Longleftarrow$ Assume Equation 1. Then

$$\begin{aligned} \Phi_{A \cup B}(A) &= \Phi(A \cap (A \cup B)) \oslash \Phi(A \cup B) \\ &= \Phi(A) \oslash \Phi(A \cup B). \blacksquare \end{aligned}$$

Proof of Theorem 3.1.5 Assume Axiom 6.

$\Longrightarrow$ Assume Axiom 7. For any $a \neq \mathbf{0}$, choose Φ as given below, where $a \oplus a' = \mathbf{1}$.

w_i	Φ	$\Phi_{\{w_1,w_2\}}$
w_1	$\mathbf{0}$	$\mathbf{0} \oslash a$
w_2	a	$a \oslash a$
w_3	a'	$\mathbf{0}$

Note that $\Phi(w_1) = \mathbf{0}$ and $\Phi(\{w_1, w_2\}) \neq \mathbf{0}$. Since proposition $\{w_2, w_3\}$ is accepted by Φ, $\{w_2, w_3\}$ must also be accepted by $\Phi_{\{w_1,w_2\}}$: $\Phi_{\{w_1,w_2\}}(w_1) = \mathbf{0}$. Therefore $\mathbf{0} \oslash a = \mathbf{0}$.

$\Longleftarrow$ Assume (Y1). Suppose A is accepted and B is not rejected by Φ. Then $\Phi(\overline{A}) = \mathbf{0}$ and $\Phi(B) \neq \mathbf{0}$. Also, $\Phi(B \cap \overline{A}) = \mathbf{0}$ by Lemma D.0.1. We have

$$\begin{aligned}
\Phi_B(\overline{A}) &= \Phi(B \cap \overline{A}) \oslash \Phi(B) \\
&= \mathbf{0} \oslash \Phi(B) \\
&= \mathbf{0}.
\end{aligned}$$

That is, A is accepted byΦ_B. ∎

Proof of Theorem 3.1.6 Assume Axioms 6–7.

$\Longrightarrow$ Assume Axiom 8. For any a, choose Φ as given below, where $a \oplus a' = \mathbf{1}$.

w_i	Φ	$\Phi_{\{w_1,w_2\}}$
w_1	a	$a \oslash \mathbf{1}$
w_2	a'	$a' \oslash \mathbf{1}$

Since $\{w_1, w_2\}$ is accepted by Φ, we have $\Phi_{\{w_1,w_2\}}(w_1) = \Phi(w_1)$. Therefore $a \oslash \mathbf{1} = a$.

$\Longleftarrow$ Assume (Y2). Suppose B is accepted by Φ. Then $\Phi(\overline{B}) = \mathbf{0}$ and $\Phi(B) = \mathbf{1}$. Moreover,

$$\begin{aligned}
\Phi(A) &= \Phi(B \cap A) \oplus \Phi(\overline{B} \cap A) \\
&= \Phi(B \cap A) \oplus \mathbf{0} \\
&= \Phi(B \cap A) \\
&= \Phi(B \cap A) \oslash \mathbf{1} \\
&= \Phi(B \cap A) \oslash \Phi(B) \\
&= \Phi_B(A). \blacksquare
\end{aligned}$$

Proof of Theorem 3.1.7 Assume Axioms 6–8.

$\Longrightarrow$ Assume Axiom 9. Let a, b, c be three supports such that $a, b \preceq_{\oplus} c \neq \mathbf{0}$ and $a \neq (\not\preceq_{\oplus})\ b$. We want to show that $a \oslash c \neq (\not\preceq_{\oplus})\ b \oslash c$.

Choose Φ, and Φ' as given below, where $a \oplus a' = b \oplus b' = c$ and $c \oplus c' = \mathbf{1}$.

w_i	Φ	$\Phi_{\{w_1,w_2\}}$
w_1	a	$a \oslash c$
w_2	a'	$a' \oslash c$
w_3	c'	$\mathbf{0}$
w_4	$\mathbf{0}$	$\mathbf{0}$

w_i	Φ'	$\Phi'_{\{w_1,w_2\}}$
w_1	b	$b \oslash c$
w_2	b'	$b' \oslash c$
w_3	c'	$\mathbf{0}$
w_4	$\mathbf{0}$	$\mathbf{0}$

Note that $\Phi(\{w_1, w_2\}) = \Phi'(\{w_1, w_2\}) = c$, and $\Phi(w_1) = a \neq (\not\preceq_{\oplus})\ b = \Phi'(w_1)$. By Axiom 9, we have $\Phi_{\{w_1,w_2\}}(w_1) \neq (\not\preceq_{\oplus})\ \Phi'_{\{w_1,w_2\}}(w_1)$ and $a \oslash c \neq (\not\preceq_{\oplus})\ b \oslash c$. That is, if $a \neq (\not\preceq_{\oplus})\ b$ then $a \oslash c \neq (\not\preceq_{\oplus})\ b \oslash c$ which is equivalent to $a \oslash c = (\preceq_{\oplus})\ b \oslash c$ only if $a = (\preceq_{\oplus})\ b$.

$\Longleftarrow$ Assume (Y3) and (Y8). Suppose there exists Φ and Φ' such that $\Phi(A \cup B) = \Phi'(A \cup B)$ and $\Phi(A) \neq (\not\preceq_{\oplus})\ \Phi'(A)$. Then $\Phi_{A\cup B}(A) = \Phi(A) \oslash \Phi(A \cup B)$ and $\Phi'_{A\cup B}(A) = \Phi'(A) \oslash \Phi(A \cup B)$. Therefore, $\Phi_B(A) \neq (\not\preceq_{\oplus})\ \Phi'_B(A)$. ∎

Proof of Theorem 3.1.8 Assume Axioms 6–9.

$\Longrightarrow$ Assume Axiom 10. Let a and b be such that $a \preceq_{\oplus} b \neq \mathbf{0}$. Choose Φ as given below, where $a \oplus a' = b$ and $b \oplus b' = \mathbf{1}$. We have $\Phi_{\{w_1,w_2\}}(w_1) \succeq_{\oplus} \Phi(w_1)$. Therefore $a \oslash b \succeq_{\oplus} a$.

w_i	Φ	$\Phi_{\{w_1,w_2\}}$
w_1	a	$a \oslash b$
w_2	a'	$a' \oslash b$
w_3	b'	$\mathbf{0}$

$\Longleftarrow$ Assume (Y5). Suppose $A \subseteq B$. Then $\Phi_B(A) = \Phi(A) \oslash \Phi(B) \succeq_{\oplus} \Phi(A)$. ∎

Proof of Theorem 3.1.9 Assume Axioms 6–10.

$\Longrightarrow$ Assume Axiom 11. Suppose that $a \oslash b = c \oslash d$, and let Φ be a state of belief such that $\Phi(A \cap B \cap C) = a$, $\Phi(A \cap C) = c$, $\Phi(B \cap C) = b$ and $\Phi(C) = d$.[1] It follows that

$$\begin{aligned}
\Phi_{B \cap C}(A) &= \Phi(A \cap B \cap C) \oslash \Phi(B \cap C) \\
&= a \oslash b \\
&= c \oslash d \\
&= \Phi(A \cap C) \oslash \Phi(C) \\
&= \Phi_C(A).
\end{aligned}$$

Given Axiom 11, we should also have $\Phi_{A \cap C}(B) = \Phi_C(B)$. This leads to

$$\Phi(A \cap B \cap C) \oslash \Phi(A \cap C) = \Phi(B \cap C) \oslash \Phi(C)$$

$$a \oslash c = b \oslash d.$$

Therefore $a \oslash b = c \oslash d$ only if $a \oslash c = b \oslash d$.

$\Longleftarrow$ Assume (Y4). If $\Phi_{B \cap C}(A) = \Phi_C(A)$, then

$$\Phi(A \cap B \cap C) \oslash \Phi(B \cap C) = \Phi(A \cap C) \oslash \Phi(C)$$

$$\Phi(A \cap B \cap C) \oslash \Phi(A \cap C) = \Phi(B \cap C) \oslash \Phi(C).$$

Therefore, $\Phi_{A \cap C}(B) = \Phi_C(B)$. ∎

[1]The proof assumes the existence of the state Φ, and this is why I say "usefully equivalent." Note, however, that if $\oplus$ is either idempotent or has an inverse, then Φ can be shown to exist.

Proof of Theorem 3.1.10 Assume Axioms 6 and 7.

$\Longrightarrow$ (Y0): Assume $a \preceq_{\oplus} b$ and $b \neq \mathbf{0}$. There must exist c and d such that $a \oplus c = b$ and $b \oplus d = \mathbf{1}$. Construct a state of belief Φ as given below.

w_i	Φ	$\Phi_{\{w_1,w_2\}}$
w_1	a	$a \oslash b$
w_2	c	$c \oslash b$
w_3	d	$\mathbf{0}$

Since $\Phi(\{w_1, w_2\}) \neq \mathbf{0}$, $\Phi_{\{w_1,w_2\}}$ must exist. Given Axiom 6, we have

$$\begin{aligned} \Phi_{\{w_1,w_2\}}(w_1) &= \Phi(w_1) \oslash \Phi(\{w_1, w_2\}) \\ &= a \oslash b. \end{aligned}$$

Therefore, $a \oslash b$ must be defined. ∎

$\Longrightarrow$ (Y6a): Assume $a \oplus b \preceq_{\oplus} c \neq \mathbf{0}$. There must exist d and e such that $a \oplus b \oplus d = c$ and $c \oplus e = \mathbf{1}$. Choose Φ as given below.

w_i	Φ	$\Phi_{\{w_1,w_2,w_3\}}$
w_1	a	$a \oslash c$
w_2	b	$b \oslash c$
w_3	d	$d \oslash c$
w_4	e	$\mathbf{0}$

Since $\Phi(\{w_1, w_2, w_3\})$ exists, then $(a \oslash c) \oplus (b \oslash c)$ must be defined.

$\Longrightarrow$ (Y7): Choose Φ and Φ' as given below, where d and e are such that $a \oplus b \oplus d = c$ and $c \oplus e = \mathbf{1}$.

w_i	Φ	$\Phi_{\{w_1,w_2,w_3\}}$
w_1	a	$a \oslash c$
w_2	b	$b \oslash c$
w_3	d	$d \oslash c$
w_4	e	$\mathbf{0}$

w_i	Φ'	$\Phi'_{\{w_1,w_2,w_3\}}$
w_1	$a \oplus b$	$(a \oplus b) \oslash c$
w_2	$\mathbf{0}$	$\mathbf{0}$
w_3	d	$d \oslash c$
w_4	e	$\mathbf{0}$

Observe that:

$$\Phi(\{w_1, w_2\}) = \Phi'(\{w_1, w_2\}) = a \oplus b$$

$$\Phi(\{w_1, w_2, w_3\}) = \Phi'(\{w_1, w_2, w_3\}) = a \oplus b \oplus d = c \neq \mathbf{0}.$$

Therefore, by Axiom 6, we have

$$\Phi_{\{w_1,w_2,w_3\}}(\{w_1, w_2\}) = \Phi'_{\{w_1,w_2,w_3\}}(\{w_1, w_2\})$$

$$(a \oslash c) \oplus (b \oslash c) = [(a \oplus b) \oslash c] \oplus \mathbf{0}$$

$$(a \oslash c) \oplus (b \oslash c) = (a \oplus b) \oslash c. \blacksquare$$

Proof of Theorem 3.1.12

(Y10) $\underline{a \oslash a = \mathbf{1}.}$

Given (Y4) and $a \oslash \mathbf{1} = a \oslash \mathbf{1}$, we have $a \oslash a = \mathbf{1} \oslash \mathbf{1}$. And given (Y2), we have $a \oslash a = \mathbf{1}$. ∎

(Y11) $\underline{a \oslash (a \oslash b) = b.}$

Assume $a \preceq_{\oplus} b \neq \mathbf{0}$. We have that $a \preceq_{\oplus} a \oslash b$ by (Y4).

$$\begin{array}{rcll} a \oslash b & = & (a \oslash b) \oslash \mathbf{1} & \text{by (Y2)} \\ a \oslash (a \oslash b) & = & b \oslash \mathbf{1} & \text{by (Y4)} \\ a \oslash (a \oslash b) & = & b & \text{by (Y2).} \end{array}$$

(Y12) $\underline{a \oslash b = c \text{ only if } a \oslash c = b.}$

Assume $a \preceq_{\oplus} b \neq \mathbf{0}$, and $c \neq \mathbf{0}$. We have that $a \preceq_{\oplus} a \oslash b$ by (Y4). Assuming that $a \oslash b = c$, we also have that $a \preceq_{\oplus} c$.

$$\begin{array}{rcll} a \oslash b & = & c \oslash \mathbf{1} & \text{assumption} \\ a \oslash c & = & b \oslash \mathbf{1} & \text{by (Y4)} \\ a \oslash c & = & b & \text{by (Y2).} \end{array}$$

(Y13) $\underline{a \preceq_{\oplus} b \text{ only if } a \oslash c \preceq_{\oplus} b \oslash c}$. Assume $a \preceq_{\oplus} b \preceq_{\oplus} c \neq \mathbf{0}$ and $b \neq \mathbf{0}$.

$$\begin{array}{rcll} a & \preceq_{\oplus} & b & \text{assumption} \\ \exists a' : a \oplus a' & = & b & \text{by definition of } \preceq_{\oplus} \\ (a \oplus a') \oslash c & = & b \oslash c & \text{by definition of } \oslash c \\ (a \oslash c) \oplus (a' \oslash c) & = & b \oslash c & \text{by (Y7)} \\ a \oslash c & \preceq_{\oplus} & b \oslash c & \text{by definition of } \preceq_{\oplus}. \end{array}$$

(Y14) $\underline{(a \oslash c) \oslash (b \oslash c) = a \oslash b.}$

Assume $a \preceq_{\oplus} b \preceq_{\oplus} c \neq \mathbf{0}$, and $b \neq \mathbf{0}$. We also have $(a \oslash c) \preceq_{\oplus} (b \oslash c)$ by (Y13).

$$\begin{array}{rcll} a \oslash (a \oslash c) & = & b \oslash (b \oslash c) & \text{by (Y11)} \\ a \oslash b & = & (a \oslash c) \oslash (b \oslash c) & \text{by (Y4).} \end{array}$$

(Y15) $\underline{(a \oslash b) \oslash (a \oslash c) = c \oslash b}$.

Assume $a \preceq_{\oplus} c \preceq_{\oplus} b \neq \mathbf{0}$, and $c \neq \mathbf{0}$.

$$\begin{aligned} (a \oslash b) \oslash (c \oslash b) &= a \oslash c \quad \text{by (Y14)} \\ (a \oslash b) \oslash (a \oslash c) &= c \oslash b \quad \text{by (Y12).} \ \blacksquare \end{aligned}$$

Proof of Theorem 3.1.13

$$\begin{aligned} [\Phi_A]_B(C) &= \Phi_A(B \cap C) \oslash \Phi_A(B) \\ &= [\Phi(A \cap B \cap C) \oslash \Phi(A)] \oslash [\Phi(A \cap B) \oslash \Phi(A)] \\ &= \Phi(A \cap B \cap C) \oslash \Phi(A \cap B) \text{ by (Y14)} \\ &= \Phi_{A \cap B}(C). \ \blacksquare \end{aligned}$$

Proof of Theorem 3.2.3

$\Longrightarrow$ Assume Axiom 12.

(Y9) Suppose that we are given $b \neq \mathbf{0}$ and c. We want to find a such that $a \preceq_{\oplus} b$ and $a \oslash b = c$. Let Φ_B be a state of belief such that $\Phi_B(C) = c$. By Axiom 12, there must exist Φ such that $\Phi(B) = b$. Moreover,

$$\begin{aligned} \Phi_B(C) &= \Phi(B \wedge C) \oslash \Phi(B) \\ c &= \Phi(B \wedge C) \oslash b. \end{aligned}$$

Therefore, if we take a to be $\Phi(B \wedge C)$, then $a \preceq_{\oplus} b$ and $a \oslash b = c$.

(Y6b) Suppose that $a, b \preceq_{\oplus} c \neq \mathbf{0}$, and $a \oslash c \oplus b \oslash c$ is defined. We want to show that $a \oplus b \preceq_{\oplus} c$. Let A, B, Φ_C be such that $\models \neg(A \wedge B)$, $\Phi_C(A) = a \oslash c$ and $\Phi_C(B) = b \oslash c$. By Axiom 12, there must exist Φ such that $\Phi(C) = c$. Moreover,

$$\begin{aligned} \Phi(A \wedge C) \oslash \Phi(C) &= a \oslash c \\ \Phi(B \wedge C) \oslash \Phi(C) &= b \oslash c \\ \Phi(A \wedge C) &= a \\ \Phi(B \wedge C) &= b. \end{aligned}$$

Since $\Phi(A \wedge C) \oplus \Phi(B \wedge C) \preceq_{\oplus} \Phi(C)$, we also have $a \oplus b \preceq_{\oplus} c$.

$\Longleftarrow$ Assume (Y9) and (Y6b).

Given Φ_C and $c \neq \mathbf{0}$, we need to construct a state of belief Ψ such that $\Psi(C) = c$ and $\Psi_C = \Phi_C$.

By (Y9), there is an a for each b such that $a \preceq_\oplus c$ and $a \oslash c = b$. Let us denote this a by $b \otimes c$. Note that

$$\Phi_C(w) \otimes c \preceq_\oplus c,$$

and

$$\bigoplus_{w \in C} (\Phi_C(w) \otimes c) \oslash c = \bigoplus_{w \in C} \Phi_C(w),$$

which is defined. Therefore, by (Y6b),

$$\left(\bigoplus_{w \in C} \Phi_C(w) \otimes c \right) \preceq_\oplus c.$$

Now, construct Ψ such that $\Psi(w) = \Phi_C(w) \otimes c$ when $w \in C$. And can complete the definition of Ψ such that $\Psi(\neg C) \oplus \Psi(C) = \mathbf{1}$. We have,

$$\begin{aligned}
\Psi(C) \oslash c &= \left(\bigoplus_{w \in C} \Phi_C(w) \otimes c \right) \oslash c \\
&= \bigoplus_{w \in C} (\Phi_C(w) \otimes c) \oslash c \\
&= \bigoplus_{w \in C} \Phi_C(w) \\
&= \mathbf{1}.
\end{aligned}$$

Hence, $\Psi(C) = c$.

When $w \notin C$, we have $\Psi_C(w) = \Phi_C(w) = \mathbf{0}$. But when $w \in C$, we have

$$\begin{aligned}
\Psi_C(w) &= \Psi(w) \oslash \Psi(C) \\
&= (\Phi_C(w) \otimes c) \oslash c \\
&= \Phi_C(w).
\end{aligned}$$

Therefore, $\Psi_C = \Phi_C$. ∎

Proof of Theorem 3.3.2

The proof of this theorem is a special case of the proof of Theorem 3.3.6. States of belief with respect to $\langle\{0,\infty\},\min\rangle_{\mathcal{L}}$ are a special case of Spohnian states of belief, and are isomorphic to propositional states of belief. That is, a state of belief Φ with respect to $\langle\{0,\infty\},\min\rangle_{\mathcal{L}}$ accepts A precisely when $\Phi(\neg A)=\infty$, which is precisely when $\Phi(\neg A)\neq 0$. ∎

Proof of Theorem 3.3.6

We want to show that if Φ^{Δ} corresponds to Δ, then $\left(\Phi^{\Delta}\right)_A$ corresponds to $\Delta\cup\{A\}$. That is, we want to show that if

$$\Phi^{\Delta}(w)=\begin{cases} i-j, & \text{if } w\in\mathcal{M}(\Delta)\cap W_i;\\ \infty, & \text{otherwise,}\end{cases}$$

then

$$\left(\Phi^{\Delta}\right)_A(w)=\begin{cases} i-k, & \text{if } w\in\mathcal{M}(\Delta\cup\{A\})\cap W_i;\\ \infty, & \text{otherwise,}\end{cases}$$

where $\mathcal{M}(\Delta)\cap W_j$ and $\mathcal{M}(\Delta\cup\{A\})\cap W_k$ are the preferred meanings of Δ and $\Delta\cup\{A\}$, respectively.

We need the following result to carry out the proof:

$$\begin{aligned}
\Phi^{\Delta}(A) &= \min_{w\in A}\Phi^{\Delta}(w)\\
&= \min_{w\in A}\begin{cases} i-j, & \text{if } w\in\mathcal{M}(\Delta)\cap W_i;\\ \infty, & \text{otherwise,}\end{cases}\\
&= \min_{w\in A}\begin{cases} i-j, & \text{if } w\in\mathcal{M}(\Delta)\cap A\cap W_i;\\ \infty, & \text{otherwise,}\end{cases}\\
&= \min_{w\in A}\begin{cases} i-j, & \text{if } w\in\mathcal{M}(\Delta\cup\{A\})\cap W_i;\\ \infty, & \text{otherwise,}\end{cases}\\
&= k-j.
\end{aligned}$$

Recall that k is the least integer such that $\mathcal{M}(\Delta\cup\{A\})\cap W_k$ is not empty. This follows from $\mathcal{M}(\Delta\cup\{A\})\cap W_k$ being the preferred meaning of $\mathcal{M}(\Delta\cup\{A\})$.

Now suppose that

$$\Phi^{\Delta}(w) = \begin{cases} i - j, & \text{if } w \in \mathcal{M}(\Delta) \cap W_i; \\ \infty, & \text{otherwise,} \end{cases}$$

and let us expand $\left(\Phi^{\Delta}\right)_A(w)$. We have

$$\begin{aligned} \left(\Phi^{\Delta}\right)_A(w) &= \Phi^{\Delta}(\{w\} \cap A) - \Phi^{\Delta}(A) \\ &= \begin{cases} \Phi^{\Delta}(w) - \Phi^{\Delta}(A), & \text{if } w \in A; \\ \infty, & \text{otherwise.} \end{cases} \\ &= \begin{cases} \Phi^{\Delta}(w) - (k - j), & \text{if } w \in A; \\ \infty, & \text{otherwise.} \end{cases} \\ &= \begin{cases} (i - j) - (k - j), & \text{if } w \in A \cap \mathcal{M}(\Delta) \cap W_i; \\ \infty, & \text{otherwise.} \end{cases} \\ &= \begin{cases} i - k, & \text{if } w \in \mathcal{M}(\Delta \cup \{A\}) \cap W_i; \\ \infty, & \text{otherwise.} \end{cases} \end{aligned}$$ ∎

Proof of Theorem 3.4.4

Given (Y9), for all a and $b \neq \mathbf{0}$, there is c such that $c \preceq_{\oplus} b$ and $c \oslash b = a$. This makes $a \otimes b$ defined and equal c. ∎

Proof of Theorem 3.4.5

Given (Y6b), we have: $a \preceq_{\oplus} c$, $b \preceq_{\oplus} c$, and $(a \oslash c) \oplus (b \oslash c)$ is defined only if $a \oplus b \preceq_{\oplus} c$. Therefore, $(a \otimes c) \preceq_{\oplus} c$, $(b \otimes c) \preceq_{\oplus} c$, and $((a \otimes c) \oslash c) \oplus ((b \otimes c) \oslash c)$ is defined only if $(a \otimes c) \oplus (b \otimes c) \preceq_{\oplus} c$. That is, $((a \otimes c) \oslash c) \oplus ((b \otimes c) \oslash c)$ is defined only if $(a \otimes c) \oplus (b \otimes c) \preceq_{\oplus} c$. Further simplification gives: $a \oplus b$ is defined only if $(a \otimes c) \oplus (b \otimes c) \preceq_{\oplus} c$. ∎

Proof of Theorem 3.4.6

(Z1) $(a \oslash b) \otimes b = a$. Suppose that $a \oslash b$ is defined and equals c. By definition of $\otimes$, we have that $c \otimes b = a$, that is, $(a \oslash b) \otimes b = a$.

(Z2) $(a \otimes b) \oslash b = a$. Suppose that $a \otimes b$ is defined and equals c. By definition of $\otimes$, we have that $c \oslash b = a$, that is, $(a \otimes b) \oslash b = a$.

(Z3) $\mathbf{0} \otimes a = \mathbf{0}$.

$$\begin{array}{rcll} \mathbf{0} & \preceq_{\oplus} & a & \text{by def of } \preceq_{\oplus} \\ \mathbf{0} \oslash a & = & \mathbf{0} & \text{by (Y1)} \\ \mathbf{0} \otimes a & = & \mathbf{0} & \text{by definition of } \otimes. \end{array}$$

(Z4) $a \otimes \mathbf{1} = a$.

$$\begin{array}{rcll} a & \preceq_{\oplus} & \mathbf{1} & \text{by def of } \preceq_{\oplus} \\ a \oslash \mathbf{1} & = & a & \text{by (Y2)} \\ a \otimes \mathbf{1} & = & a & \text{by definition of } \otimes. \end{array}$$

(Z5) $a \otimes b \preceq_{\oplus} b$. Follows easily from the definition of $\otimes$.

(Z6) $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.

Assume $(a \otimes c) \oplus (b \otimes c) \preceq_{\oplus} c$ and $c \neq \mathbf{0}$.

$$\begin{array}{rcll} (a \otimes c) \oplus (b \otimes c) & = & X & \\ [(a \otimes c) \oplus (b \otimes c)] \oslash c & = & X \oslash c & \text{by definition of } \oslash \\ a \oplus b & = & X \oslash c & \text{by (Y7) and (Z2)} \\ (a \oplus b) \otimes c & = & X & \text{by definition of } \otimes. \end{array}$$

(Z7) $a \preceq_{\oplus} b$ only if $a \otimes c \preceq_{\oplus} b \otimes c$.

$$\begin{array}{rcll} a & \preceq_{\oplus} & b & \text{assumption} \\ (a \otimes c) \oslash c & \preceq_{\oplus} & (b \otimes c) \oslash c & \text{defs of } \otimes \text{ and } \oslash \\ a \otimes c & \preceq_{\oplus} & b \otimes c & \text{by (Y8).} \end{array}$$

(Z8a) $\underline{a \otimes b = b \otimes a.}$

$$\begin{array}{rcll} c & = & a \otimes b & \text{assumption} \\ a & = & c \oslash b & \text{by def of } \otimes \\ b & = & c \oslash a & \text{by (Y12)} \\ b \otimes a & = & c & \text{by (Z1).} \end{array}$$

(Z8b) Follows from the proof of (Z8a).

(Y16) $\underline{(a \oslash b) \oslash c = (a \oslash c) \oslash b.}$

Assume $a \preceq_{\oplus} (b \otimes c) \neq \mathbf{0}$. By (Z5) and (Y13), we have $a \preceq_{\oplus} b$, $a \preceq_{\oplus} c$, $a \oslash c \preceq_{\oplus} b$, and $a \oslash b \preceq_{\oplus} c$. Moreover,

$$\begin{array}{rcll} a \oslash (a \oslash b) & = & (b \otimes c) \oslash c & \text{by (Y11) and (Z2)} \\ a \oslash (b \otimes c) & = & (a \oslash b) \oslash c & \text{by (Y4)} \\ & & & \\ a \oslash (a \oslash c) & = & (c \otimes b) \oslash b & \text{by (Y11) and (Z2)} \\ a \oslash (c \otimes b) & = & (a \oslash c) \oslash b & \text{by (Y4)} \\ & & & \\ (a \oslash b) \oslash c & = & (a \oslash c) \oslash b & \text{by (Z8a).} \end{array}$$

(Z9a) $\underline{a \otimes (b \otimes c) = (a \otimes b) \otimes c}.$

We know that $b \neq \mathbf{0}$ and $c \neq \mathbf{0}$ because one of $a \otimes (b \otimes c)$ and $(a \otimes b) \otimes c$ is defined. If $a = \mathbf{0}$, the proof is trivial. Assume $a \neq \mathbf{0}$. We know that $(b \otimes c) \preceq_{\oplus} c$ by (Z5). We also know $a \otimes (b \otimes c) \preceq_{\oplus} a \otimes c$ by (Z7) and (Z8a).

$$\begin{array}{rcll} X & = & a \otimes (b \otimes c) & \text{assumption} \\ X & = & (b \otimes c) \otimes a & \text{by (Z8a)} \\ X \oslash a & = & b \otimes c & \text{by (Z2)} \\ (X \oslash a) \oslash c & = & b & \text{by (Z2)} \\ (X \oslash c) \oslash a & = & b & \text{by (Y16)} \\ X \oslash c & = & b \otimes a & \text{by (Z1)} \\ X & = & (b \otimes a) \otimes c & \text{by (Z1)} \\ X & = & (a \otimes b) \otimes c & \text{by (Z8a).} \end{array}$$

(Z9b) Follows from the proof of (Z9a).

(Z10) $\underline{a \otimes b = a \text{ iff } a = \mathbf{0} \text{ or } b = \mathbf{1}}$.

$$\begin{array}{rcll} a \otimes b & = & a & \text{assumption} \\ \\ a & \neq & \mathbf{0} & \text{assumption} \\ b \otimes a & = & a & \text{by (Z8a)} \\ b & = & a \oslash a & \text{by (Z2)} \\ b & = & \mathbf{1} & \text{by (Y2)} \\ \\ a & = & \mathbf{0} & \text{assumption} \\ a \otimes b & = & \mathbf{0} & \text{by (Z3)} \\ \\ b & = & \mathbf{1} & \text{assumption} \\ a \otimes b & = & a \otimes \mathbf{1} = a & \text{by (Z4).} \end{array}$$

(Z11) $\underline{(a \otimes b) \oslash c = a \otimes (b \oslash c)}$.

Assume $a \otimes b$ and $b \oslash c$ are defined.

$$\begin{array}{rcll} a \otimes ((b \oslash c) \otimes c) & = & a \otimes b & \text{by (Z1)} \\ (a \otimes (b \oslash c)) \otimes c & = & a \otimes b & \text{by (Z9a)} \\ a \otimes (b \oslash c) & = & (a \otimes b) \oslash c & \text{by (Z2).} \end{array}$$

(Z12) $\underline{(a \otimes c) \oslash (b \otimes c) = a \oslash b.}$

Assume $a \otimes c \preceq_\oplus b \otimes c \neq \mathbf{0}$. It follows that $c \neq \mathbf{0}$ and $b \neq \mathbf{0}$. If $a \otimes c = \mathbf{0}$, then $a = \mathbf{0}$, $(a \otimes c) \oslash (b \otimes c) = \mathbf{0}$, and $a \oslash b = \mathbf{0}$. Suppose that $a \otimes c \neq \mathbf{0}$. It follows, by (Z5), (Z7), (Z8a), and (Y13), that $b \otimes c \preceq_\oplus c$, $a \otimes (b \otimes c) \preceq_\oplus a \otimes c$, and $a \preceq_\oplus (a \otimes c) \oslash (b \otimes c)$.

$$
\begin{array}{rcll}
(a \otimes c) \oslash (b \otimes c) & = & X & \text{assumption} \\
(a \otimes c) \oslash X & = & b \otimes c & \text{by (Y12)} \\
(c \otimes a) \oslash X & = & b \otimes c & \text{by (Z8a)} \\
c \otimes (a \oslash X) & = & b \otimes c & \text{by (Z11)} \\
(a \oslash X) \otimes c & = & b \otimes c & \text{by (Z8a)} \\
a \oslash X & = & b & \text{by def of } \otimes \\
a \oslash b & = & X & \text{by (Y12).} \blacksquare
\end{array}
$$

Proof of Theorem 3.4.7

Suppose $\mathcal{S}$ is finite and has more than two elements. Then for some b such that $b \neq \mathbf{0}$ and $b \neq \mathbf{1}$, the set $\{a : a \preceq_\oplus b\}$ is also finite and its cardinality is less than the cardinality of $\mathcal{S}$ because $\mathbf{1} \notin \{a : a \preceq_\oplus b\}$. Since $\oslash b$ is injective, which follows from (Y3), the cardinality of $\{a \oslash b : a \preceq_\oplus b\}$ equals the cardinality of $\{a : a \preceq_\oplus b\}$. Therefore, the cardinality of $\{a \oslash b : a \preceq_\oplus b\}$ is less than the cardinality of $\mathcal{S}$ and $\mathcal{S} \setminus \{a \oslash b : a \preceq_\oplus b\}$ is not empty. It follows that there is some c for which there is no a such that $a \preceq_\oplus b$ and $a \oslash b = c$, that is, $c \otimes b$ is not defined. $\blacksquare$

Proof of Theorem 3.5.1

Suppose that $A \supset B$ is accepted by Φ. Then $A \cap \overline{B}$ is rejected, $\Phi(A \cap \overline{B}) = \mathbf{0}$, $\Phi(A) = \Phi(A \cap B) \oplus \Phi(A \cap \overline{B}) = \Phi(A \cap B)$. Moreover, we have

$$\begin{aligned} \Phi_B(A) &= \Phi(A \cap B) \oslash \Phi(B) \\ &= \Phi(A) \oslash \Phi(B) \\ &\succeq_\oplus \Phi(A) \qquad \text{by (Y5).} \end{aligned}$$

Note also that $\Phi(A) \oslash \Phi(B) = \Phi(A)$ iff $\Phi(A) = \Phi(A) \otimes \Phi(B)$ iff $\Phi(A) = \mathbf{0}$ or $\Phi(B) = \mathbf{1}$. Therefore, $\Phi_B(A) \succ_\oplus \Phi(A)$ unless $\Phi(A) = \mathbf{0}$ or $\Phi(B) = \mathbf{1}$. ∎

Proof of Theorem 3.5.2

If $A \supset B$ is accepted, then $\overline{B} \supset \overline{A}$ is also accepted. And since $\overline{A}$ is not rejected, then, by Theorem 3.5.1, we have $\Phi_{\overline{A}}(\overline{B}) \succ_\oplus \Phi(\overline{B})$ unless $\Phi(\overline{B}) = \mathbf{0}$ or $\Phi(\overline{A}) = \mathbf{1}$. ∎

Proof of Theorem 3.5.3

If $A \cap B$ is rejected, then $\overline{A} \cup \overline{B}$ is accepted, and so is $B \supset \overline{A}$. And since $\overline{A}$ is not rejected, then, by Theorem 3.5.1, we have $\Phi_{\overline{A}}(B) \succ_\oplus \Phi(B)$ unless $\Phi(B) = \mathbf{0}$ or $\Phi(\overline{A}) = \mathbf{1}$. ∎

Lemma for Theorem 3.5.4

Lemma D.0.4 If $A \cap B \cap C$ is not rejected by Φ, then

$$\Phi_C(A) \otimes \Phi_{A\cap C}(B) = \Phi_C(B) \otimes \Phi_{B\cap C}(A).$$

Proof Assume that $A \cap B \cap C$ is not rejected by Φ. We have that

$$\begin{aligned} \Phi(A \cap B \cap C) &= [\Phi_C(A \cap B)] \otimes \Phi(C) \\ &= [\Phi_{B\cap C}(A) \otimes \Phi_C(B)] \otimes \Phi(C) \\ &= [\Phi_{A\cap C}(B) \otimes \Phi_C(A)] \otimes \Phi(C). \end{aligned}$$

Since $\otimes\Phi(C)$ is a bijection, we have $\Phi_{B\cap C}(A)\otimes\Phi_C(B) = \Phi_{A\cap C}(B)\otimes\Phi_C(A)$. Moreover, by (Z8a), we have

$$\Phi_C(B) \otimes \Phi_{B\cap C}(A) = \Phi_C(A) \otimes \Phi_{A\cap C}(B). \blacksquare$$

Proof of Theorem 3.5.4

Suppose that $A \supset C_i$ is accepted by Φ, for $i = 1, \ldots, n$. Then $\Phi(A \cap \overline{C_i}) = \mathbf{0}$, and we have

$$\begin{aligned}
\Phi_A(\overline{C_1 \cap \ldots \cap C_i}) &= \Phi(A \cap \overline{C_1 \cap \ldots \cap C_i}) \oslash \Phi(A) \\
&= \Phi((A \cap \overline{C_1}) \cup \ldots \cup (A \cap \overline{C_i})) \oslash \Phi(A) \\
&= \left[\Phi(A \cap \overline{C_1}) \oplus \ldots \oplus \Phi(A \cap \overline{C_i})\right] \oslash \Phi(A) \\
&= \mathbf{0}.
\end{aligned}$$

Therefore,

$$\Phi_A(C_1 \cap \ldots \cap C_i) = \mathbf{1},$$

$$\Phi(A \cap C_1 \cap \ldots \cap C_i) = \Phi(A),$$

$$\Phi_{A \cap C_1 \cap \ldots \cap C_{n-1}}(C_n) = \mathbf{1}.$$

Moreover, we have

$$\begin{aligned}
\Phi_{C_1 \cap \ldots \cap C_{n-1}}(A) \otimes \Phi_{A \cap C_1 \cap \ldots \cap C_{n-1}}(C_n) &= \Phi_{C_1 \cap \ldots \cap C_{n-1}}(C_n) \otimes && \text{by Lemma D.0.4} \\
&\quad\ \Phi_{C_1 \cap \ldots \cap C_n}(A) \\
\Phi_{C_1 \cap \ldots \cap C_{n-1}}(A) \otimes \mathbf{1} &= \Phi_{C_1 \cap \ldots \cap C_{n-1}}(C_n) \otimes \\
&\quad\ \Phi_{C_1 \cap \ldots \cap C_n}(A) \\
\Phi_{C_1 \cap \ldots \cap C_{n-1}}(A) &= \Phi_{C_1 \cap \ldots \cap C_{n-1}}(C_n) \otimes && \text{by (Z4)} \\
&\quad\ \Phi_{C_1 \cap \ldots \cap C_n}(A) \\
\Phi_{C_1 \cap \ldots \cap C_{n-1}}(A) &\preceq_{\oplus} \Phi_{C_1 \cap \ldots \cap C_n}(A) && \text{by (Z5)}
\end{aligned}$$

Now, assume that $\Phi_{C_1 \cap \ldots \cap C_{n-1}}(A) = \Phi_{C_1 \cap \ldots \cap C_n}(A)$. Then

$$\begin{aligned}
\Phi_{A \cap C_1 \cap \ldots \cap C_{n-1}}(C_n) &= \Phi_{C_1 \cap \ldots \cap C_{n-1}}(C_n) && \text{by (Y4)} \\
\mathbf{1} &= \Phi_{C_1 \cap \ldots \cap C_{n-1}}(C_n.)
\end{aligned}$$

Therefore, if $\Phi_{C_1 \cap \ldots \cap C_{n-1}}(C_n) \neq \mathbf{1}$, then $\Phi_{C_1 \cap \ldots \cap C_{n-1}}(A) \neq \Phi_{C_1 \cap \ldots \cap C_n}(A)$. ∎

Appendix E

Proofs of Chapter 4

Proof of Theorem 4.3.5

The proof is by induction on the number of nodes in a causal network.

Base case. The network has only one node. Trivial.

Inductive step. Suppose that the causal network $\langle N, \mathcal{G}, \mathcal{CS} \rangle$ is satisfied by exactly one state of belief Φ. Let $\langle N \cup \{k\}, \mathcal{G}', \mathcal{CS}' \rangle$ be a causal network that results from augmenting $\langle N, \mathcal{G}, \mathcal{CS} \rangle$ by a childless node k such that $\mathcal{CS}'$ is consistent. We want to show that the causal network $\langle N \cup \{k\}, \mathcal{G}', \mathcal{CS}' \rangle$ is satisfied by exactly one state of belief.

I will show this in three steps:

I. Constructing a state of belief Φ'.

II. Showing that Φ' satisfies $\langle N \cup \{k\}, \mathcal{G}', \mathcal{CS}' \rangle$.

III. Showing that Φ' is the only state of belief that satisfies $\langle N \cup \{k\}, \mathcal{G}', \mathcal{CS}' \rangle$.

I. Constructing a state of belief Φ' over propositions $N \cup \{k\}$.

Construct Φ' over propositions $N \cup \{k\}$ as follows:

- $\Phi'(\underline{N} \wedge \underline{k}) = \begin{cases} \mathbf{0}, & \text{if } \Phi(\underline{N}) = \mathbf{0}; \\ \mathcal{CS}_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}), & \text{otherwise.} \end{cases}$

- $\Phi'(A \vee B) = \Phi'(A) \oplus \Phi'(B)$ when $\models \neg(A \wedge B)$.

- $\Phi'(\mathbf{false}) = \mathbf{0}$.

To show that Φ' is a state of belief, it suffices to show that $\Phi'(\mathbf{true}) = \mathbf{1}$:

$$\begin{aligned}
\Phi'(\mathbf{true}) &= \bigoplus_{\underline{N} \wedge \underline{k}} \Phi'(\underline{N} \wedge \underline{k}) \\
&= \bigoplus_{\underline{N} \wedge \underline{k}} \mathcal{CS}_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}), \text{ where } \underline{N} \models \underline{k\diamond} \\
&= \bigoplus_{\underline{N}} \bigoplus_{\underline{k}} \mathcal{CS}_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}) \\
&= \bigoplus_{\underline{N}} \Phi(\underline{N}) \otimes \bigoplus_{\underline{k}} \mathcal{CS}_{\underline{k\diamond}}(\underline{k}) \\
&= \bigoplus_{\underline{N}} \Phi(\underline{N}) \otimes \mathbf{1} \\
&= \bigoplus_{\underline{N}} \Phi(\underline{N}) \\
&= \mathbf{1}.
\end{aligned}$$

Therefore, Φ' is a state of belief over $N \cup \{k\}$. Moreover, it is easy to show that for any sentence A such that $A \not\models \underline{k}$, we have that $\Phi(A) = \Phi'(A)$.

II. Showing that Φ' satisfies the causal network $\langle N \cup \{k\}, \mathcal{G}', \mathcal{CS}' \rangle$. We need to show that

$$IN'_{\Phi}(i, i\diamond, i\lhd \setminus i\diamond) \text{ and } \Phi'_{\underline{i\diamond}}(\underline{i}) = \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \text{ for every } i \text{ in } N \cup \{k\}.$$

Consider the following cases:

Case $i \in N$. Then Condition 12 follows from the induction hypothesis.

Case $i = k$. First, we have

$$\begin{aligned}
\Phi'_{\underline{i\diamond}}(\underline{i}) &= \Phi'(\underline{k} \wedge \underline{k\diamond}) \oslash \Phi'(\underline{k\diamond}) \\
&= \left(\bigoplus_{\underline{N} \models \underline{k\diamond}} \Phi'(\underline{k} \wedge \underline{N}) \right) \oslash \Phi'(\underline{k\diamond}) \\
&= \left(\bigoplus_{\underline{N} \models \underline{k\diamond}} \mathcal{CS}'_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}) \right) \oslash \Phi(\underline{k\diamond}) \\
&= \left(\mathcal{CS}'_{\underline{k\diamond}}(\underline{k}) \otimes \left(\bigoplus_{\underline{N} \models \underline{k\diamond}} \Phi(\underline{N}) \right) \right) \oslash \Phi'(\underline{k\diamond}) \\
&= \left(\mathcal{CS}'_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{k\diamond}) \right) \oslash \Phi(\underline{k\diamond}) \\
&= \mathcal{CS}'_{\underline{k\diamond}}(\underline{k}).
\end{aligned}$$

Moreover, the non–descendants of i are N in this case. Hence,

$$\begin{aligned}
\Phi'_{\underline{i\lhd}}(\underline{i}) &= \Phi'_{\underline{N}}(\underline{k}) \\
&= \Phi'(\underline{k} \wedge \underline{N}) \oslash \Phi'(\underline{N}) \\
&= \left(\mathcal{CS}'_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}) \right) \oslash \Phi(\underline{N}) \\
&= \mathcal{CS}'_{\underline{k\diamond}}(\underline{k}).
\end{aligned}$$

III. Showing that Φ' is the only state of belief that satisfies the causal network $\langle N \cup \{k\}, \mathcal{G}', \mathcal{CS}' \rangle$.

Suppose that Ψ is another state of belief over $N \cup \{k\}$ such that

$$IN_{\Psi}(i, i\diamond, i\triangleleft \setminus i\diamond) \text{ and } \Psi_{\underline{i\diamond}}(\underline{i}) = \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \text{ for every } i \text{ in } N \cup \{k\}.$$

It suffices to show that Φ' and Ψ are equal.

• From the induction hypothesis, we know that Φ is the only state of belief over N that satisfies:

$$IN_{\Phi}(i, i\diamond, i\triangleleft \setminus i\diamond) \text{ and } \Phi_{\underline{i\diamond}}(\underline{i}) = \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \text{ for every } i \text{ in } N.$$

Therefore, Φ' and Ψ cannot disagree on any complete sentence $\underline{N}$.

• To show that Φ' and Φ are equal, we need to show that they cannot disagree on any complete sentence $\underline{N} \wedge \underline{k}$.

The non–descendants of k are N. Hence,

$$\begin{aligned} \Phi'(\underline{N} \wedge \underline{k}) &= \Phi'_{\underline{N}}(\underline{k}) \otimes \Phi'(\underline{N}) \\ &= \mathcal{CS}'_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}). \end{aligned}$$

Moreover,

$$\begin{aligned} \Psi(\underline{N} \wedge \underline{k}) &= \Psi_{\underline{N}}(\underline{k}) \otimes \Psi(\underline{N}) \\ &= \mathcal{CS}'_{\underline{k\diamond}}(\underline{k}) \otimes \Phi(\underline{N}). \end{aligned}$$

Therefore, Φ' and Ψ are equal. ∎

Proof of Theorem 4.4.1

Assume $IN_{\Phi}(I, K, J)$ and let $\underline{I \cup K}$ be such that $\Phi(\underline{I \cup K}) \neq \mathbf{0}$.

Case I. $\Phi(\underline{J \cup K}) = \mathbf{0}$:

$$\begin{array}{rcll} \Phi(\underline{J \cup K}) \oslash \Phi(\underline{K}) & = & \mathbf{0} \oslash \Phi(\underline{K}) & \text{by case} \\ \Phi_{\underline{K}}(\underline{J}) & = & \mathbf{0} & \text{by (Y1) and Definition 3.1.11} \end{array}$$

$$\begin{array}{rcll} \Phi(\underline{I \cup J \cup K}) & = & \mathbf{0} & \text{by case and Lemma D.0.1} \\ \Phi(\underline{I \cup J \cup K}) \oslash \Phi(\underline{I \cup K}) & = & \mathbf{0} \oslash \Phi(\underline{I \cup K}) & \text{by above} \\ \Phi_{\underline{I \cup K}}(\underline{J}) & = & \mathbf{0} & \text{by (Y1) and Definition 3.1.11} \end{array}$$

$$\begin{array}{rcll} \Phi_{\underline{I \cup K}}(\underline{J}) & = & \Phi_{\underline{K}}(\underline{J}) & \text{by above} \end{array}$$

Case II. $\Phi(\underline{J \cup K}) \neq \mathbf{0}$:

$$\begin{array}{rcll} \Phi_{\underline{J \cup K}}(\underline{I}) & = & \Phi_{\underline{K}}(\underline{I}) & \text{assumption} \\ \Phi(\underline{I \cup J \cup K}) \oslash \Phi(\underline{J \cup K}) & = & \Phi(\underline{I \cup K}) \oslash \Phi(\underline{K}) & \text{by Definition 3.1.11} \\ \Phi(\underline{I \cup J \cup K}) \oslash \Phi(\underline{I \cup K}) & = & \Phi(\underline{J \cup K}) \oslash \Phi(\underline{K}) & \text{by (Y4)} \\ \Phi_{\underline{I \cup K}}(\underline{J}) & = & \Phi_{\underline{K}}(\underline{J}) & \text{by Definition 3.1.11} \end{array}$$

Therefore, $IN_{\Phi}(J, K, I)$. ∎

Proof of Theorem 4.4.2

Assume $IN_\Phi(I, \emptyset, J \cup L)$ and let $\underline{J}$ be such that $\Phi(\underline{J}) \neq \mathbf{0}$. Then for some $\underline{L}$, we have $\Phi(\underline{J} \wedge \underline{L}) \neq \mathbf{0}$.

$$\begin{array}{rcll}
\Phi_{\underline{J} \wedge \underline{L}}(\underline{I}) & = & \Phi(\underline{I}) & \text{assumption} \\
\Phi(\underline{I \cup J} \wedge \underline{L}) & = & \Phi(\underline{I}) \otimes \Phi(\underline{J} \wedge \underline{L}) & \text{by Corollary 3.4.3} \\
\bigoplus_{\underline{L}} \Phi(\underline{I \cup J} \wedge \underline{L}) & = & \bigoplus_{\underline{L}} \Phi(\underline{I}) \otimes \Phi(\underline{J} \wedge \underline{L}) & \text{by summing equals} \\
\bigoplus_{\underline{L}} \Phi(\underline{I \cup J} \wedge \underline{L}) & = & \bigoplus_{\underline{L}} \Phi(\underline{J} \wedge \underline{L}) \otimes \Phi(\underline{I}) & \text{by (Z8a)} \\
\bigoplus_{\underline{L}} \Phi(\underline{I \cup J} \wedge \underline{L}) & = & \left[\bigoplus_{\underline{L}} \Phi(\underline{J} \wedge \underline{L})\right] \otimes \Phi(\underline{I}) & \text{by (Z6)} \\
\Phi(\underline{I \cup J}) & = & \Phi(\underline{I}) \otimes \Phi(\underline{J}) & \text{by def of } \Phi \text{ and (Z8a)} \\
\Phi(\underline{I \cup J}) \oslash \Phi(\underline{J}) & = & \Phi(\underline{I}) & \text{by def of } \otimes \\
\Phi_{\underline{J}}(\underline{I}) & = & \Phi(\underline{I}) & \text{by Definition 3.1.11}
\end{array}$$

The applicability of (Z6) requires that

$$\bigoplus_{\underline{L}} \Phi(\underline{J} \wedge \underline{L}) \otimes \Phi(\underline{I}) \preceq_\oplus \Phi(\underline{I}),$$

which follows from

$$\bigoplus_{\underline{L}} \Phi(\underline{J} \wedge \underline{L}) \otimes \Phi(\underline{I}) = \Phi(\underline{I \cup J}) \preceq_\oplus \Phi(\underline{I}).$$

Therefore, $IN(I, \emptyset, J)$.

The same proof can be carried out with respect to $\Phi_{\underline{K}}$, hence, by Corollary 4.2.4, $IN(I, K, J)$. ∎

Proof of Theorem 4.4.3

Assume $IN_\Phi(J \cup L, K, I)$ and let $\underline{I \cup K \cup L}$ be such that $\Phi(\underline{I \cup K \cup L}) \neq \mathbf{0}$. By the above assumption, Symmetry, and Decomposition, it follows that $IN_\Phi(I, K, J \cup L)$, $IN_\Phi(I, K, L)$, $IN_\Phi(L, K, I)$, and

$$\Phi_{\underline{I \cup K}}(\underline{L}) = \Phi_{\underline{K}}(\underline{L}).$$

Moreover:

$$\begin{array}{rcll}
\Phi(\underline{J \cup L \cup I \cup K}) \oslash \Phi(\underline{I \cup K}) & = & \Phi(\underline{J \cup L \cup K}) \oslash \Phi(\underline{K}) & \text{by Definition 3.1.11} \\
\left[\Phi_{\underline{L \cup I \cup K}}(\underline{J}) \otimes \Phi(\underline{L \cup I \cup K})\right] & = & \left[\Phi_{\underline{L \cup K}}(\underline{J}) \otimes \Phi(\underline{L \cup K})\right] & \text{by Corollary 3.4.3} \\
\oslash \Phi(\underline{I \cup K}) & & \oslash \Phi(\underline{K}) & \\
\Phi_{\underline{L \cup I \cup K}}(\underline{J}) \otimes & = & \Phi_{\underline{L \cup K}}(\underline{J}) \otimes & \text{by (Z11)} \\
[\Phi(\underline{L \cup I \cup K}) \oslash \Phi(\underline{I \cup K})] & & [\Phi(\underline{L \cup K}) \oslash \Phi(\underline{K})] & \\
\Phi_{\underline{L \cup I \cup K}}(\underline{J}) \otimes \Phi_{\underline{I \cup K}}(\underline{L}) & = & \Phi_{\underline{L \cup K}}(\underline{J}) \otimes \Phi_{\underline{K}}(\underline{L}) & \text{by Definition 3.1.11} \\
\Phi_{\underline{L \cup I \cup K}}(\underline{J}) & = & \Phi_{\underline{L \cup K}}(\underline{J}) & \text{by above}
\end{array}$$

Therefore, $IN_\Phi(J, K \cup L, I)$. ∎

Proof of Theorem 4.4.4

Assume $IN_\Phi(I, K, J)$ and $IN_\Phi(I, K \cup J, L)$. Let $\underline{K \cup J \cup L}$ be such that $\Phi(\underline{K \cup J \cup L}) \neq \mathbf{0}$.

$$\begin{array}{rcll}
\Phi_{\underline{K \cup J}}(\underline{I}) & = & \Phi_{\underline{K}}(\underline{I}) & \text{assumption} \\
\Phi_{\underline{K \cup J \cup L}}(\underline{I}) & = & \Phi_{\underline{K \cup J}}(\underline{I}) & \text{assumption} \\
\Phi_{\underline{K \cup J \cup L}}(\underline{I}) & = & \Phi_{\underline{K}}(\underline{I}) & \text{by above}
\end{array}$$

Therefore, $IN_\Phi(I, K, J \cup L)$. ∎

Proof of Theorem 4.4.5

Assume $IN_\Phi(I, L, J)$ and $IN_\Phi(I, J, L)$ and consider $\underline{J} \wedge \underline{L}$ where $\Phi(\underline{J} \wedge \underline{L}) \neq \mathbf{0}$:

$$\begin{array}{rcll}
\Phi_{\underline{J}\wedge\underline{L}}(\underline{I}) & = & \Phi_{\underline{L}}(\underline{I}) & \text{assumption} \\
\Phi_{\underline{J}\wedge\underline{L}}(\underline{I}) & = & \Phi_{\underline{J}}(\underline{I}) & \text{assumption} \\
\Phi_{\underline{L}}(\underline{I}) & = & \Phi_{\underline{J}}(\underline{I}) & \text{by above} \\
\Phi(\underline{I} \wedge \underline{L}) \oslash \Phi(\underline{L}) & = & \Phi_{\underline{J}}(\underline{I}) & \text{by Definition 3.1.11} \\
\Phi(\underline{I} \wedge \underline{L}) \oslash \Phi_{\underline{J}}(\underline{I}) & = & \Phi(\underline{L}) & \text{by (Y12)} \\
\bigoplus_{\underline{L}} \Phi(\underline{I} \wedge \underline{L}) \oslash \Phi_{\underline{J}}(\underline{I}) & = & \bigoplus_{\underline{L}} \Phi(\underline{L}) & \text{by summing equals} \\
\left[\bigoplus_{\underline{L}} \Phi(\underline{I} \wedge \underline{L})\right] \oslash \Phi_{\underline{J}}(\underline{I}) & = & \mathbf{1} & \text{by (Y7)} \\
\Phi(\underline{I}) & = & \Phi_{\underline{J}}(\underline{I}) & \text{by (Y3) and (Y10)} \\
\Phi(\underline{I}) & = & \Phi_{\underline{J}\wedge\underline{L}}(\underline{I}) & \text{by above.}
\end{array}$$

Property (Y12) requires that $\Phi_{\underline{J}}(\underline{I}) \neq \mathbf{0}$, which holds since Φ attributes $\mathbf{0}$ to **false** only.

The same proof can be carried out with respect to $\Phi_{\underline{K}}$, hence, by Corollary 4.2.4, $IN_\Phi(I, K, J \cup L)$. ∎

Proof of Theorem 4.4.6

Verma [Verma, 1986] has shown that if

1. $\mathcal{G}$ is constructed such that $IN_\Phi(i, i\diamond, i\triangleleft \setminus i\diamond)$ for each node i, and if
2. IN_Φ satisfies the graphoid axioms,

then $IN_\mathcal{G}(I, K, J)$ only if $IN_\Phi(I, K, J)$. ∎

Appendix F

Proofs of Chapter 5

Proof of Theorem 5.2.1 Any path between a node in I and another in J must have $\rightarrow i \rightarrow$ as part of it. That is, on any such path, i is a node with linear arrows and i belongs to K. Therefore, K d–separates I from J. ∎

Proof of Theorem 5.2.2 Any path between a node in I and another in J must have $\leftarrow i \rightarrow$ as part of it. That is, on any such path, i is a node with diverging arrows. Therefore, i d–separates I from J. ∎

Proof of Theorem 5.2.3 Any path between a node in J and node i must have either $\rightarrow k \rightarrow i$ or $\leftarrow k \rightarrow i$ as part of it, where k belongs to K. That is, on any such path, k is a node with either linear or diverging arrows and k belongs to K. Therefore, K d–separates J from i. ∎

Proof of Theorem 5.2.4 Any path between a node in I and another in J must have $\rightarrow i \leftarrow$ as part of it. That is, on any such path, i is a node with converging arrows, i does not belong to $\emptyset$, and neither does any of its descendents. Therefore, $\emptyset$ d–separates I from J. ∎

Proof of Theorem 5.3.1

$$\begin{aligned}
\Phi(\underline{i\diamond} \wedge \delta) &= \Phi(\underline{i} \wedge \delta_{i\triangleleft} \wedge \delta_{i\triangleright}) \\
&= \Phi_{\underline{i} \wedge \delta_{i\triangleleft}}(\delta_{i\triangleright}) \otimes \Phi(\underline{i} \wedge \delta_{i\triangleleft}) \\
&= \Phi_{\underline{i}}(\delta_{i\triangleright}) \otimes \Phi(\underline{i} \wedge \delta_{i\triangleleft}), \text{by Theorem 5.2.1} \\
&= \lambda_i(\underline{i}) \otimes \pi_i(\underline{i}). \blacksquare
\end{aligned}$$

Proof of Theorem 5.3.2

$$\begin{aligned}
\Phi_{\underline{i}}(\delta_{i\triangleright}) &= \Phi_{\underline{i}}(\bigwedge_{k \in i\circ} \delta_{i\triangleright k}) \\
&= \bigotimes_{k \in i\circ} \Phi_{\underline{i}}(\delta_{i\triangleright k}), \text{ by Theorem 5.2.2} \\
&= \bigotimes_{k \in i\circ} \lambda_{k.i}(\underline{i}). \blacksquare
\end{aligned}$$

Proof of Theorem 5.3.3

$$\begin{aligned}
\Phi(\underline{i} \wedge \delta_{i\triangleleft}) &= \bigoplus_{\underline{i\diamond}} \Phi(\underline{i} \wedge \delta_{i\triangleleft} \wedge \underline{i\diamond}) \\
&= \bigoplus_{\underline{i\diamond}} \Phi_{\delta_{i\triangleleft} \wedge \underline{i\diamond}}(\underline{i}) \otimes \Phi(\delta_{i\triangleleft} \wedge \underline{i\diamond}) \\
&= \bigoplus_{\underline{i\diamond}} \Phi_{\underline{i\diamond}}(\underline{i}) \otimes \Phi(\delta_{i\triangleleft} \wedge \underline{i\diamond}), \text{ by Theorem 5.2.3} \\
&= \bigoplus_{\underline{i\diamond}} \Phi_{\underline{i\diamond}}(\underline{i}) \otimes \Phi(\bigwedge_{\underline{i\diamond} \models \underline{j}} \delta_{i\triangleleft j} \wedge \underline{j}) \\
&= \bigoplus_{\underline{i\diamond}} \Phi_{\underline{i\diamond}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond} \models \underline{j}} \Phi(\delta_{i\triangleleft j} \wedge \underline{j}), \text{ by Theorem 5.2.4} \\
&= \bigoplus_{\underline{i\diamond}} \mathcal{CS}_{\underline{i\diamond}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond} \models \underline{j}} \pi_{j.i}(\underline{j}). \blacksquare
\end{aligned}$$

Proof of Theorem 5.3.4

In this proof, $\delta_{i\lhd\bar{j}}$ is the observation about nodes connected to node i via incoming arcs, other than arc $j \rightarrow i$.

If i is not observed, then

$$
\begin{aligned}
&\Phi_{\underline{j}}(\delta_{j\rhd i}) \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{j}}(\delta_{j\rhd i} \wedge \underline{i} \wedge \underline{i\diamond j}) \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{j}}(\delta_{i\rhd} \wedge \delta_{i\lhd\bar{j}} \wedge \underline{i} \wedge \underline{i\diamond j}), \text{ because } \delta_{j\rhd i} \equiv \delta_{i\rhd} \wedge \delta_{i\lhd\bar{j}} \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{j}\wedge\delta_{i\lhd\bar{j}}\wedge\underline{i}\wedge\underline{i\diamond j}}(\delta_{i\rhd}) \otimes \Phi_{\underline{j}\wedge\delta_{i\lhd\bar{j}}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \Phi_{\underline{j}}(\delta_{i\lhd\bar{j}} \wedge \underline{i\diamond j}) \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{i}}(\delta_{i\rhd}) \otimes \Phi_{\underline{j}\wedge\delta_{i\lhd\bar{j}}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \Phi_{\underline{j}}(\delta_{i\lhd\bar{j}} \wedge \underline{i\diamond j}), \text{ by Theorem 5.2.1} \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{i}}(\delta_{i\rhd}) \otimes \Phi_{\underline{j}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \Phi_{\underline{j}}(\delta_{i\lhd\bar{j}} \wedge \underline{i\diamond j}), \text{ by Theorem 5.2.3} \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{i}}(\delta_{i\rhd}) \otimes \Phi_{\underline{j}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \Phi(\delta_{i\lhd\bar{j}} \wedge \underline{i\diamond j}), \text{ by Theorem 5.2.4} \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{i}}(\delta_{i\rhd}) \otimes \Phi_{\underline{j}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \Phi\Big(\bigwedge_{\underline{i\diamond j}\models\underline{l}} \delta_{i\lhd l} \wedge \underline{l}\Big) \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \Phi_{\underline{i}}(\delta_{i\rhd}) \otimes \Phi_{\underline{j}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond j}\models\underline{l}} \Phi(\delta_{i\lhd l} \wedge \underline{l}), \text{ by Theorem 5.2.4} \\
&= \bigoplus_{\underline{i}} \bigoplus_{\underline{i\diamond j}} \lambda_i(\underline{i}) \otimes \mathcal{CS}_{\underline{j}\wedge\underline{i\diamond j}}(\underline{i}) \otimes \bigotimes_{\underline{i\diamond j}\models\underline{l}} \pi_{l.i}(\underline{l}).
\end{aligned}
$$

If i is an observed node, then $\delta_{j\rhd i}$ is either i or $\neg i$. Therefore, $\Phi_j(\delta_{j\rhd i}) = \mathbf{1}$ and $\Phi_{\neg j}(\delta_{j\rhd i}) = \mathbf{0}$ when $\delta_{j\rhd i} \equiv i$. Moreover, $\Phi_j(\delta_{j\rhd i}) = \mathbf{0}$ and $\Phi_{\neg j}(\neg\delta_{j\rhd i}) = \mathbf{1}$ when $\delta_{j\rhd i} \equiv \neg i$. This follows from the way auxiliary nodes are created. Therefore,

$$
\lambda_{i.j}(\underline{j}) = \begin{cases} \langle \mathbf{1}, \mathbf{0} \rangle, & \text{if } \delta \models i; \\ \langle \mathbf{0}, \mathbf{1} \rangle, & \text{if } \delta \models \neg i. \end{cases} \quad \blacksquare
$$

Proof of Theorem 5.3.5

The observations δ can be decomposed into two observations:

- $\delta_{i \triangleright k}$, the observation about nodes connected to i via arc $i \rightarrow k$.
- $\delta_{k \triangleleft i}$, the observation about nodes connected to k via arc $k \leftarrow i$.

Moreover, the updated support BL_i is closely related to parental support:

$$\begin{aligned} BL_i &= \langle \Phi(i \wedge \delta), \Phi(\neg i \wedge \delta) \rangle \\ &= \langle \Phi(i \wedge \delta_{i \triangleright k} \wedge \delta_{k \triangleleft i}), \Phi(\neg i \wedge \delta_{i \triangleright k} \wedge \delta_{k \triangleleft i}) \rangle \end{aligned}$$

$$\pi_{i.k} = \langle \Phi(i \wedge \delta_{k \triangleleft i}), \Phi(\neg i \wedge \delta_{k \triangleleft i}) \rangle.$$

Therefore, $\pi_{i.k}$ equals BL_i when $\delta_{i \triangleright k} \equiv$ **true**. Hence,

$$\pi_{i.k} = \pi_i \otimes \bigotimes_{l \in i o k} \lambda_{l.i}. \blacksquare$$

Proof Theorem 5.4.2

I will show that if $\pi_{o.p} \in S_n$ or $\lambda_{o.q} \in S_n$, then o is connected to i via an undirected path of length n. This implies that the number of non–empty states in a backward propagation cannot be more than the maximal length of a path between node i and some other node. The proof is by induction on n.

Case $n = 1$:

$$S_1 = \{\pi_{j.i} : j \to i\} \cup \{\lambda_{k.i} : k \leftarrow i\}.$$

It is clear that each of j and k is connected to i via a path of length 1.

Case $n > 1$: Suppose that if $\pi_{o.p} \in S_{n-1}$ or $\lambda_{o.q} \in S_{n-1}$, then o is connected to i via an undirected path of length $n - 1$. Consider the members of S_n:

$$\begin{aligned} S_n \;=\; & \{\pi_{j.k} : j \to k, \exists p : \pi_{k.p} \in S_{n-1} \text{ or } \exists q \neq j : \lambda_{k.q} \in S_{n-1}\} \cup \\ & \{\lambda_{k.j} : k \leftarrow j, \exists p : \lambda_{j.p} \in S_{n-1} \text{ or } \exists q \neq k : \pi_{j.q} \in S_{n-1}\}. \end{aligned}$$

Case $\pi_{j.k} \in S_n$: Then $j \to k$ and either $\pi_{k.p} \in S_{n-1}$ or $\lambda_{k.q} \in S_{n-1}$. Therefore, k is connected to i via a path of length $n - 1$, hence, j is connected to i via a path of length n.

Case $\lambda_{k.j} \in S_n$: Then $k \leftarrow j$ and either $\lambda_{j.p} \in S_{n-1}$ or $\pi_{j.q} \in S_{n-1}$. Therefore, j is connected to i via a path of length $n - 1$, hence, k is connected to i via a path of length n.

Therefore, if $\pi_{o.p} \in S_n$ or $\lambda_{o.q} \in S_n$, then o is connected to i via an undirected path of length n. ∎

Proof Theorem 5.4.4

I will show that if $\pi_{p.o} \in S_n$ or $\lambda_{q.o} \in S_n$, then o is connected to some boundary node via an undirected path of length n. This implies that the number of non–empty states in a forward propagation cannot be more than the maximal length of a path in the causal network. The proof is by induction on n.

Case $n = 1$**:**

$$\begin{aligned} S_1 &= \{\pi_{j.k} : j \text{ is a root node with only one child } k\} \cup \\ &\quad \{\lambda_{k.j} : k \text{ is a leaf node with only one parent } j\}. \end{aligned}$$

If $\pi_{j.k} \in S_1$, then k is connected to a boundary node j via an undirected path of length 1. Furthermore, if $\lambda_{k.j} \in S_1$, then j is connected to a boundary node k via an undirected path of length 1.

Case $n > 1$**:** Suppose that if $\pi_{p.o} \in S_{n-1}$ or $\lambda_{q.o} \in S_{n-1}$, then o is connected to some boundary node via an undirected path of length $n-1$. Consider the members of S_n:

$$\begin{aligned} S_n &= \{\pi_{j.k} : \pi_{j.k} \notin S_{<n}, j \to k, \forall p \to j : \pi_{p.j} \in S_{<n}, \forall k \neq q \leftarrow j : \lambda_{q.j} \in S_{<n}\} \cup \\ &\quad \{\lambda_{k.j} : \lambda_{k.j} \notin S_{<n}, k \leftarrow j, \forall p \leftarrow k : \lambda_{p.k} \in S_{<n}, \forall j \neq q \to k : \pi_{q.k} \in S_{<n}\}. \end{aligned}$$

Case $\pi_{j.k} \in S_n$**:** Then $\pi_{j.k} \notin S_{<n}$, $j \to k$, for all $p \to j : \pi_{p.j} \in S_{<n}$, and for all $k \neq q \leftarrow j : \lambda_{q.j} \in S_{<n}$. Therefore, some $\pi_{p.j}$ or some $\lambda_{q.j}$ belongs to S_{n-1}; otherwise, $\pi_{j.k} \in S_{<n}$. Hence, j is connected to a boundary node via a path of length $n-1$ and k is connected to a boundary node via a path of length n.

Case $\lambda_{k.j} \in S_n$**:** Then $\lambda_{k.j} \notin S_{<n}$, $k \leftarrow j$, for all $p \leftarrow k : \lambda_{p.k} \in S_{<n}$, and for all $j \neq q \to k : \pi_{q.k} \in S_{<n}$. Therefore, some $\lambda_{p.k}$ or some $\pi_{q.k}$ belongs to S_{n-1}; otherwise, $\lambda_{k.j} \in S_{<n}$. Hence, k is connected to a boundary node via a path of length $n-1$ and j is connected to a boundary node via a path of length n.

Therefore, if $\pi_{p.o} \in S_n$ or $\lambda_{q.o} \in S_n$, then o is connected to a boundary node via an undirected path of length n. ∎

Appendix G

Proofs of Chapter 7

Proof of Theorem 7.3.1

Logical conjunction is commutative and associative.

If $(a \wedge b) \wedge c = a$,
then $a \models b \wedge c$ and $a \models b$.
Hence, $a \wedge b = a$.

$a \preceq_{\oplus} b$ precisely when there is c such that $a \wedge c = b$.
Hence, $a \preceq_{\oplus} b$ precisely when $b \models a$.

For all $a \in \mathcal{O}$,
true is the only member of $\mathcal{O}$ satisfying $a \wedge \mathbf{true} = a$, and
false is the only member of $\mathcal{O}$ satisfying $a \wedge \mathbf{false} = \mathbf{false}$. ∎

If we represent sentences using their models, then objection summation would be set intersection, objection order would be $\supseteq$; the zero objection would be the set of all models; and the full objection would be the empty set. These observations simplify the following proofs.

Proof of Theorem 7.4.1

We know that $\langle \mathcal{O}, \wedge \rangle$ is a partial support structure. Therefore, we need to show that $\angle$ is a support scaling function.

If we represent objections using their models, then objection scaling would be set difference when the first argument is not the full set $\mathbf{0}$. In this and the following proof, I assume that objections are represented using their models.

(Y1) $\underline{\mathbf{0} \angle a = \mathbf{0} \text{ when } a \neq \mathbf{0}.}$

Follows directly from definition of $\angle$.

(Y2) $\underline{a \angle \mathbf{1} = a.}$

Follows directly from definition of $\angle$.

(Y3) $\underline{a \angle c = b \angle c \text{ only if } a = b \text{ when } a, b \preceq_{\oplus} c \neq \mathbf{0}.}$

Assume $a \angle c = b \angle c$ only if $a = b$.
If $a = \mathbf{0}$, then
$a \angle c = \mathbf{0}$,
$b \angle c = \mathbf{0}$, and
b must be $\mathbf{0}$.
And vice versa if $b = \mathbf{0}$.
If neither a nor b are $\mathbf{0}$, then
$\angle$ is set difference,
which satisfies $a \setminus c = b \setminus c$ only if $a = b$ when $a, b \supseteq c$.

(Y4) $\underline{a \angle b = c \angle d \text{ only if } a \angle c = b \angle d \text{ when } a \preceq_{\oplus} b, c \preceq_{\oplus} d \text{ and } b, c, d \neq \mathbf{0}.}$

Since $b, c, d \neq \mathbf{0}$, it follows that $a \neq \mathbf{0}$.
Therefore, $\angle$ is set difference,
which satisfies $a \setminus b = c \setminus d$ only if $a \setminus c = b \setminus d$ when $d \subseteq b, c \subseteq a$.

(Y5) $\underline{a \angle b \succeq_{\oplus} a \ \text{ when } a \preceq_{\oplus} b \neq \mathbf{0}.}$

Case $a = \mathbf{0}$:

$a = a \angle b.$

Case $a \neq \mathbf{0}$:

$\angle$ is set difference,

which satisfies $a \setminus b \subseteq a$.

(Y6a) $\underline{a \wedge b \preceq_{\oplus} c \text{ only if } (a \angle c) \wedge (b \angle c) \text{ is defined } \ \text{when } c \neq \mathbf{0}.}$

This is trivial since $\wedge$ is defined everywhere in objection calculus.

(Y7) $\underline{(a \wedge b) \angle c = (a \angle c) \wedge (b \angle c) \ \text{ when } a \wedge b \preceq_{\oplus} c \neq \mathbf{0}.}$

Trivial if either $a = \mathbf{0}$ or $b = \mathbf{0}$.

If neither is $\mathbf{0}$, then

$\angle$ is set difference,

which satisfies $(a \cup b) \setminus c = (a \cup c) \setminus (b \cup c)$ when $c \subseteq a \cup b$.

(Y8) $\underline{a \angle c \preceq_{\oplus} b \angle c \text{ only if } a \preceq_{\oplus} b \ \text{ when } a, b \preceq_{\oplus} c \neq \mathbf{0}.}$

Trivial if either $a = \mathbf{0}$ or $b = \mathbf{0}$.

If neither is $\mathbf{0}$, then

$\angle$ is set difference,

which satisfies $a \setminus c \supseteq b \setminus c$ only if $a \supseteq b$ when $a, b \supseteq c$.

(Y6b) $\underline{a \preceq_{\oplus} c,\ b \preceq_{\oplus} c, \text{ and } (a \oslash c) \oplus (b \oslash c) \text{ is defined only if } a \oplus b \preceq_{\oplus} c \ \text{ when } c \neq \mathbf{0}.}$

Since $a \supset c$ and $b \supset c$, we have $a \cap b \subset c$.

Since the set of supports $\mathcal{O}$ is finite, it follows by Theorem 3.4.7 that the support structure is not bijective. ∎

Proof of Theorem 7.4.4

We want to show that $a \sqcup b = c$ precisely when $c \angle b = a$ and $c \preceq_{\oplus} b \neq \mathbf{0}$. Suppose that objections are represented by their models. We want to show that

$$a \cup b = c \text{ and } [a \cap b = \emptyset \text{ or } a = \mathbf{0}] \text{ and } b \neq \mathbf{0}$$

precisely when

$$c \angle b = a \text{ and } c \supseteq b \neq \mathbf{0}.$$

$\Longrightarrow$ Assume $c \angle b = a$ and $c \supseteq b \neq \mathbf{0}$.

Case $c = \mathbf{0}$:

$a = \mathbf{0}$.
Hence, $\mathbf{0} \cup b = \mathbf{0} = c$,
$a = \mathbf{0}$ and $b \neq \mathbf{0}$.

Case $c \neq \mathbf{0}$:

$c \angle b = c \setminus b = a$.
Hence, $a \cup b = c$,
$a \cap b = \emptyset$, and $b \neq \mathbf{0}$.

$\Longleftarrow$ Assume $a \cup b = c$ and $[a \cap b = \emptyset$ or $a = \mathbf{0}]$ and $b \neq \mathbf{0}$.

Case $a = \mathbf{0}$:

$c = \mathbf{0}$.
Hence, $c \angle b = \mathbf{0} = a$, and
$c \supseteq b$ and $b \neq \mathbf{0}$.

Case $a \neq \mathbf{0}$:

$a \cup b = c$, and
$a \cap b = \emptyset$.
Hence, $c \angle b = c \setminus b = a$,
$c \supseteq b$ and $b \neq \mathbf{0}$. ∎

Proof of Theorem 7.5.4

$\Longrightarrow$ Assume $WIN_{\Phi}(I, K, J)$:

$$\Phi(\underline{I} \wedge \underline{K}) = \alpha \vee \Phi(\underline{K}) \text{ only if } \Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \alpha \vee \Phi(\underline{J} \wedge \underline{K}).$$

Since $\Phi(\underline{I} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{K})$,
$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K})$.

$\Longleftarrow$ Assume

$$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K}).$$

Suppose $\Phi(\underline{I} \wedge \underline{K}) = \alpha \vee \Phi(\underline{K})$.
We need to show that $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \alpha \vee \Phi(\underline{J} \wedge \underline{K})$.
This is equivalent to $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \alpha \vee \Phi(\underline{K}) \vee \Phi(\underline{J} \wedge \underline{K})$,
which is equivalent to $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K})$,
which is true by supposition. ∎

Proof of Theorem 7.5.5

Assume $WIN_{\Phi}(I, K, J)$:

$$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K}).$$

From $\Phi(\underline{I} \wedge \underline{K}) = \breve{\Phi}_{\underline{K}}(\underline{I}) \vee \Phi(\underline{K})$, and
$\Phi(\underline{J} \wedge \underline{K}) = \breve{\Phi}_{\underline{K}}(\underline{J}) \vee \Phi(\underline{K})$, we conclude
$\Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K}) = \breve{\Phi}_{\underline{K}}(\underline{I}) \vee \breve{\Phi}_{\underline{K}}(\underline{J}) \vee \Phi(\underline{K})$.
Hence, $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \breve{\Phi}_{\underline{K}}(\underline{I}) \vee \breve{\Phi}_{\underline{K}}(\underline{J}) \vee \Phi(\underline{K})$, and
$\breve{\Phi}_{\underline{K}}(\underline{I} \wedge \underline{J}) := \breve{\Phi}_{\underline{K}}(\underline{I}) \vee \breve{\Phi}_{\underline{K}}(\underline{J})$. ∎

Proof of Theorem 7.5.7

Assume $WIN_\Phi(I, K, J \cup L)$:

$$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K}).$$

Then $\bigwedge_{\underline{L}} \Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \bigwedge_{\underline{L}} \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$,
$\bigwedge_{\underline{L}} \Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \bigwedge_{\underline{L}} \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$, and
$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K})$.
Hence, $WIN_\Phi(I, K, J)$. ∎

Proof of Theorem 7.5.8

Assume $WIN_\Phi(I, K, J \cup L)$:

$$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K}).$$

Since $\Phi(\underline{J} \wedge \underline{K}) \models \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$,
$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$.
By Decomposition, $WIN_\Phi(I, K, J)$:
$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K})$.
Hence, $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$, and
$WIN_\Phi(I, K \cup J, L)$. ∎

Proof of Theorem 7.5.9

Assume $WIN_\Phi(I, K, J)$ and $WIN_\Phi(I, K \cup J, L)$:

$$\begin{aligned} \Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) &= \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K}) \\ \Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) &= \Phi(\underline{I} \wedge \underline{J} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K}). \end{aligned}$$

Substituting for $\Phi(\underline{I} \wedge \underline{J} \wedge \underline{K})$ in the second equation,
$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$.
Since $\Phi(\underline{J} \wedge \underline{K}) \models \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$,
$\Phi(\underline{I} \wedge \underline{J} \wedge \underline{L} \wedge \underline{K}) = \Phi(\underline{I} \wedge \underline{K}) \vee \Phi(\underline{J} \wedge \underline{L} \wedge \underline{K})$.
Hence, $WIN_\Phi(I, K, J \cup L)$. ∎

Proof of Theorem 7.6.5

The proof is by induction on the number of nodes in a causal network.

Base case. The network has one node. Trivial.

Inductive step. The network has more than one node. Suppose that

1. $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is obtained by adding a childless node k to another causal network $\langle \mathcal{L}', \mathcal{O}', \mathcal{G}', \mathcal{SO}' \rangle$.

2. The network $\langle \mathcal{L}', \mathcal{O}', \mathcal{G}', \mathcal{SO}' \rangle$ is satisfied by exactly one state of belief Φ'.

I will show that the network $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is satisfied by exactly one state of belief.

Showing that $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is satisfied by some state of belief.

Let N and N' be the primitive propositions in languages $\mathcal{L}$ and $\mathcal{L}'$, respectively. Construct a mapping Φ from $\mathcal{L}$ to $\mathcal{O}$ such that

1. $\Phi(\underline{N'} \wedge \underline{k}) = \Phi'(\underline{N'}) \vee \mathcal{SO}_{\underline{k\diamond}}(\underline{k})$, where $\underline{N'} \models \underline{k\diamond}$.

2. $\Phi(A \vee B) = \Phi(A) \wedge \Phi(B)$ when $\models \neg(A \wedge B)$.

3. $\Phi(\mathbf{false}) = \mathbf{true}$.

I first show that Φ is a state of belief and then show that it satisfies $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$. To show that Φ is a state of belief, all I need to show is that $\Phi(\mathbf{true}) = \mathbf{false}$.

$$
\begin{aligned}
\Phi(\mathbf{true}) &= \bigwedge_{\underline{N}} \Phi(\underline{N}) \\
&= \bigwedge_{\underline{N'} \wedge \underline{k}} \Phi(\underline{N'} \wedge \underline{k}) \\
&= \bigwedge_{\underline{N'} \wedge \underline{k} \models \underline{k\diamond}} \Phi'(\underline{N'}) \vee \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \\
&= \bigwedge_{\underline{N'} \models \underline{k\diamond}} [\Phi'(\underline{N'}) \vee \mathcal{SO}_{\underline{k\diamond}}(k)] \wedge [\Phi'(\underline{N'}) \vee \mathcal{SO}_{\underline{k\diamond}}(\neg k)] \\
&= \bigwedge_{\underline{N'} \models \underline{k\diamond}} \Phi'(\underline{N'}) \vee [\mathcal{SO}_{\underline{k\diamond}}(k) \wedge \mathcal{SO}_{\underline{k\diamond}}(\neg k)]
\end{aligned}
$$

$$\begin{aligned} &= \bigwedge_{\underline{N'}} \Phi'(\underline{N'}) \\ &= \mathbf{false}. \end{aligned}$$

We can also show that Φ and Φ' agree on any sentence that does not entail $\underline{k}$.

To show that Φ satisfies $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$, we need to show

$$WIN_{\Phi}(i, i\diamond, i\triangleleft \setminus i\diamond) \text{ and } \Phi(\underline{i} \wedge \underline{i\diamond}) = \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\underline{i\diamond}). \tag{12}$$

Consider the following cases:

Case $i \neq k$:

Then $i \in N'$ and Condition 12 is equivalent to

$$WIN_{\Phi'}(i, i\diamond, i\triangleleft \setminus i\diamond) \text{ and } \Phi'(\underline{i} \wedge \underline{i\diamond}) = \mathcal{SO}'_{\underline{i\diamond}}(\underline{i}) \vee \Phi'(\underline{i\diamond}),$$

which follows from the induction hypothesis.

Case $i = k$:

Then $i\triangleleft = N'$ and Condition 12 is equivalent to

$$WIN_{\Phi}(k, k\diamond, N' \setminus k\diamond) \text{ and } \Phi(\underline{k} \wedge \underline{k\diamond}) = \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi(\underline{k\diamond}).$$

I first show that $\Phi(\underline{k} \wedge \underline{k\diamond}) = \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi(\underline{k\diamond})$:

$$\begin{aligned} \Phi(\underline{k} \wedge \underline{k\diamond}) &= \bigwedge_{N' \models \underline{k\diamond}} \Phi(\underline{k} \wedge N') \\ &= \bigwedge_{N' \models \underline{k\diamond}} \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi'(N') \\ &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \bigwedge_{N' \models \underline{k\diamond}} \Phi'(N') \\ &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi'(\underline{k\diamond}) \\ &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi(\underline{k\diamond}). \end{aligned}$$

To show $WIN_{\Phi}(k, k\diamond, N' \setminus k\diamond)$, it suffices to show that

$$\Phi(\underline{k} \wedge \underline{k\diamond} \wedge \underline{N'}) = \Phi(\underline{k} \wedge \underline{k\diamond}) \vee \Phi(\underline{N'}),$$

when $\underline{N'} \models \underline{k\diamond}$.

$$\begin{aligned}\Phi(\underline{k} \wedge \underline{k\diamond} \wedge \underline{N'}) &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi'(\underline{N'}) \\ &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi(\underline{k\diamond}) \vee \Phi(\underline{N'}) \\ &= \Phi(\underline{k} \wedge \underline{k\diamond}) \vee \Phi(\underline{N'}).\end{aligned}$$

Showing that $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is satisfied by exactly one state of belief.

Suppose that a state of belief Ψ over N satisfies $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$. Then Ψ must agree with Φ and Φ' on any sentence that does not entail $\underline{k}$. This follows because

1. Φ' is the only state of belief that satisfies the portion of $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ not including node k.

2. Ψ must agree with Φ' on sentences that do not entail $\underline{k}$.

3. Φ agrees with Φ' on sentences that do not entail $\underline{k}$.

To show that Ψ completely agrees with Φ, we must show that

$$\Psi(\underline{N'} \wedge \underline{k}) = \Phi(\underline{N'} \wedge \underline{k}).$$

$$\begin{aligned}\Phi(\underline{N'} \wedge \underline{k}) &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Phi'(\underline{N'}), \text{ where } \underline{N'} \models \underline{k\diamond} \\ &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Psi(\underline{N'}) \\ &= \mathcal{SO}_{\underline{k\diamond}}(\underline{k}) \vee \Psi(\underline{k\diamond}) \vee \Psi(\underline{N'}) \\ &= \Psi(\underline{k} \wedge \underline{k\diamond}) \vee \Psi(\underline{N'}) \\ &= \Psi(\underline{k} \wedge \underline{N'}).\end{aligned}$$

Therefore, Ψ and Φ are equal. ∎

Proof of Theorem 7.6.6

Let $\Phi : \mathcal{L} \to \mathcal{O}$ be an objection–based state of belief, where N are the primitive propositions of $\mathcal{L}$. Let $1, \ldots, n$ be a total order of the propositions in N. Construct a wob causal network $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ such that

- The parents of node i in $\mathcal{G}$ are nodes $1, \ldots, i-1$.
- $\mathcal{SO}_{\underline{i\diamond}}(\underline{i}) = \begin{cases} \Phi_{\underline{i\diamond}}(\underline{i}), & \text{if } \Phi(\underline{i\diamond}) \neq \mathbf{true}; \\ \mathbf{true}, & \text{otherwise.} \end{cases}$

It is clear that $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$ is a wob causal network. Therefore, it is satisfied by exactly one state of belief. I will now show that Φ satisfies $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$. First, by definition of $\mathcal{SO}$, we have

$$\Phi(\underline{i\diamond} \wedge \underline{i}) = \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\underline{i\diamond}).$$

Second, by definition of $\mathcal{G}$, the non–descendents of a node are also its parents, $i\diamond = i\triangleleft$. Therefore,

$$\begin{aligned} \Phi(\underline{i} \wedge \underline{i\triangleleft}) &= \Phi(\underline{i} \wedge \underline{i\triangleleft}) \vee \Phi(\underline{i\triangleleft}) \\ &= \Phi(\underline{i} \wedge \underline{i\diamond}) \vee \Phi(\underline{i\triangleleft}), \text{ where } \underline{i\diamond} = \underline{i\triangleleft}. \end{aligned}$$

Hence, $WIN_{\Phi}(i, i\diamond, i\triangleleft \setminus i\diamond)$, and Φ satisfies $\langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$. ∎

Proof of Theorem 7.6.7

Verma [Verma, 1986] has shown that if

1. $\mathcal{G}$ is constructed such that $WIN_{\Phi}(i, i\diamond, i\triangleleft)$ for each node i, and if
2. WIN_{Φ} satisfies the graphoid axioms,

then $IN_{\mathcal{G}}(I, K, J)$ only if $WIN_{\Phi}(I, K, J)$. ∎

Proof of Theorem 7.7.1

This proof is based on the following observations:

1. The observation $\delta_{i\lhd}$ can be decomposed into n observations when i has n parents. The observation associated with parent j, denoted by $\delta_{i\lhd j}$, is about nodes connected to i via arc $i \leftarrow j$.

2. According to Theorem 5.2.3, once the parents of node i are observed, node i becomes weakly independent from nodes connected to it via incoming arcs.

3. According to Theorem 5.2.4, nodes connected to i via one incoming arc are weakly independent from nodes connected to i via another incoming arc.

$$
\begin{aligned}
\Phi(\underline{i} \wedge \delta_{i\lhd}) &= \bigwedge_{\underline{i\diamond}} \Phi(\underline{i} \wedge \delta_{i\lhd} \wedge \underline{i\diamond}) \\
&= \bigwedge_{\underline{i\diamond}} \breve{\Phi}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\delta_{i\lhd} \wedge \underline{i\diamond}), \text{ by Theorem 5.2.3} \\
&= \bigwedge_{\underline{i\diamond}} \breve{\Phi}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\bigwedge_{\underline{i\diamond}\models\underline{j}} \delta_{i\lhd j} \wedge \underline{j}) \\
&= \bigwedge_{\underline{i\diamond}} \breve{\Phi}_{\underline{i\diamond}}(\underline{i}) \vee \bigvee_{\underline{i\diamond}\models\underline{j}} \Phi(\delta_{i\lhd j} \wedge \underline{j}), \text{ by Theorem 5.2.4} \\
&= \bigwedge_{\underline{i\diamond}} \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \bigvee_{\underline{i\diamond}\models\underline{j}} \mu_{j.i}(\underline{j}). \blacksquare
\end{aligned}
$$

Proof of Theorem 7.7.2

$\Phi(\underline{j} \wedge \delta)$ equals $\Phi(\underline{j} \wedge \delta_{i \triangleleft j} \wedge \delta_{j \triangleright i})$. Moreover, if $\delta_{j \triangleright i} = \mathbf{true}$, then $\Phi(\underline{j} \wedge \delta)$ equals $\Phi(\underline{j} \wedge \delta_{i \triangleleft j})$. Therefore, $\mu_{j.i}$ equals BL_j when $\delta_{j \triangleright i}$ is suppressed:

$$\mu_{j.i} = \mu_j \vee \bigvee_{l \in j \circ i} \nu_{l.j}.$$

BL_j is proved later to be

$$\mu_j \vee \bigvee_{l \in j \circ} \nu_{l.j},$$

but the proof does not depend on this one. ∎

Proof of Theorem 7.7.3

This proof is based on the following observations:

1. The observation δ can be decomposed into m observations when i has m children. The observation associated with child k, denoted by $\delta_{i \triangleright k}$, is about nodes connected to i via the arc $i \rightarrow k$.

2. According to Theorem 5.2.2, once node i is observed, nodes connected to i via one outgoing arc become weakly independent from nodes connected to i via another outgoing arc.

$$\begin{aligned}
\breve{\Phi}_{\underline{i}}(\delta_{i \triangleright}) &= \breve{\Phi}_{\underline{i}}(\bigwedge_{k \in i \circ} \delta_{i \triangleright k}) \\
&= \bigvee_{k \in i \circ} \breve{\Phi}_{\underline{i}}(\delta_{i \triangleright k}), \text{ by Theorem 7.5.5} \\
&= \bigvee_{k \in i \circ} \nu_{k.i}.
\end{aligned}$$

Proof of Theorem 7.7.4

This proof is based on the following observations:

1. When node k is not observed the observation $\delta_{i \triangleright k}$ can be decomposed into two observations. The first observation, denoted by $\delta_{k \triangleright}$, is about nodes below k. The second observation, denoted by $\delta_{k \triangleleft \bar{i}}$, is about nodes connected to k via incoming arcs other than $i \to k$.

2. When node k is not observed, the observation $\delta_{k \triangleleft \bar{i}}$ can be decomposed into $n-1$ observations when k has n parents. The observation associated with parent l, denoted by $\delta_{k \triangleleft l}$, is about nodes connected to k via arc $k \leftarrow l$.

3. According to Theorem 5.2.1, once node k is observed, nodes above k become weakly independent from nodes below k.

4. According to Theorem 5.2.3, once the parents of node k are observed, node k weakly becomes independent from nodes connected to k via incoming arcs.

5. According to Theorem 5.2.4, nodes connected to k via one incoming arc are weakly independent of nodes connected to k via another incoming arc.

If node k is not observed, then

$$\begin{aligned}
&\Phi(\underline{i} \wedge \delta_{i \triangleright k}) \\
&= \bigwedge_{\underline{k}} \Phi(\underline{k} \wedge \underline{i} \wedge \delta_{i \triangleright k}) \\
&= \bigwedge_{\underline{k}} \Phi(\underline{k} \wedge \underline{i} \wedge \delta_{k \triangleright} \wedge \delta_{k \triangleleft \bar{i}}) \\
&= \bigwedge_{\underline{k}} \breve{\Phi}_{\underline{k}}(\delta_{k \triangleright}) \vee \Phi(\underline{k} \wedge \underline{i} \wedge \delta_{k \triangleleft \bar{i}}),
\end{aligned}$$

by Theorem 5.2.1

$$\begin{aligned}
&= \bigwedge_{\underline{k}} \breve{\Phi}_{\underline{k}}(\delta_{k \triangleright}) \vee \bigwedge_{\underline{k \diamond i}} \Phi(\underline{k} \wedge \underline{k \diamond i} \wedge \underline{i} \wedge \delta_{k \triangleleft \bar{i}}) \\
&= \bigwedge_{\underline{k}} \breve{\Phi}_{\underline{k}}(\delta_{k \triangleright}) \vee \bigwedge_{\underline{k \diamond i}} \breve{\Phi}_{\underline{k \diamond i} \wedge \underline{i}}(\underline{k}) \vee \Phi(\underline{k \diamond i} \wedge \underline{i} \wedge \delta_{k \triangleleft \bar{i}}),
\end{aligned}$$

by Theorem 5.2.3

$$= \bigwedge_{\underline{k}} \breve{\Phi}_{\underline{k}}(\delta_{k \triangleright}) \vee \bigwedge_{\underline{k \diamond i}} \breve{\Phi}_{\underline{k \diamond i} \wedge \underline{i}}(\underline{k}) \vee \Phi(\underline{k \diamond i} \wedge \delta_{k \triangleleft \bar{i}}) \vee \Phi(\underline{i}),$$

by Theorem 5.2.4

$$\begin{aligned}
&= \bigwedge_{\underline{k}} \breve{\Phi}_{\underline{k}}(\delta_{k \triangleright}) \vee \bigwedge_{\underline{k \diamond i}} \breve{\Phi}_{\underline{k \diamond i} \wedge \underline{i}}(\underline{k}) \vee \Phi\Big(\bigwedge_{\underline{k \diamond i} \models \underline{l}} \underline{l} \wedge \delta_{k \triangleleft l}\Big) \vee \Phi(\underline{i}) \\
&= \bigwedge_{\underline{k}} \breve{\Phi}_{\underline{k}}(\delta_{k \triangleright}) \vee \bigwedge_{\underline{k \diamond i}} \breve{\Phi}_{\underline{k \diamond i} \wedge \underline{i}}(\underline{k}) \vee \bigvee_{\underline{k \diamond i} \models \underline{l}} \Phi(\underline{l} \wedge \delta_{k \triangleleft l}) \vee \Phi(\underline{i}),
\end{aligned}$$

by Theorem 5.2.4

$$= \left(\bigwedge_{\underline{k}} \nu_i(\underline{k}) \vee \bigwedge_{\underline{k \diamond i}} \mathcal{SO}_{\underline{k \diamond j} \wedge \underline{i}}(\underline{k}) \vee \bigvee_{\underline{k \diamond i} \models \underline{l}} \mu_{l.k}(\underline{l}) \right) \vee \Phi(\underline{i}).$$

Hence,

$$\breve{\Phi}_{\underline{i}}(\delta_{i \triangleright k}) := \bigwedge_{\underline{k}} \nu_i(\underline{k}) \vee \bigwedge_{\underline{k \diamond j}} \mathcal{SO}_{\underline{k \diamond i} \wedge \underline{i}}(\underline{k}) \vee \bigvee_{\underline{k \diamond j} \models \underline{l}} \mu_{l.k}(\underline{l}).$$

If node k is observed, then $\delta_{i \triangleright k}$ is either k or $\neg k$. Therefore,

$$\breve{\Phi}_{\underline{i}}(\delta_{i \triangleright k}) := \Phi_{\underline{i}}(\delta_{i \triangleright k}) = \begin{cases} \Phi_{\underline{i}}(k) = \mathbf{false}, & \text{if } \delta \models k; \\ \Phi_{\underline{i}}(\neg k) = \mathbf{true}, & \text{if } \delta \models \neg k. \end{cases}$$ ∎

Proof of Theorem 7.7.5

This proof is based on two observations:

1. The observation δ can be decomposed into two observations. The first observation, denoted by $\delta_{i\triangleleft}$, is about nodes above i, while the second observation, denoted by $\delta_{i\triangleright}$, is about nodes below i. This decomposition is possible because node i is not observed.
2. According to Theorem 5.2.1, once the state of node i is observed, nodes above i become weakly independent of nodes below i.

$$
\begin{aligned}
\Phi(\underline{i} \wedge \delta) &= \Phi(\underline{i} \wedge \delta_{i\triangleright} \wedge \delta_{i\triangleleft}) \\
&= \breve{\Phi}_{\underline{i}}(\delta_{i\triangleright}) \vee \Phi(\underline{i} \wedge \delta_{i\triangleleft}), \text{ by Theorem 7.5.5. } \blacksquare
\end{aligned}
$$

Appendix H

Proofs of Chapter 8

Notational conventions

The languages $\mathcal{L}$ and $\mathcal{O}$ are constructed from disjoint sets of propositional symbols: Σ and Ω. Members of Σ are denoted by $\sigma, \sigma_1, \sigma_2, \ldots$, while members of Ω are denoted by $\omega, \omega_1, \omega_2, \ldots$ Sentences in $\mathcal{L}$ are denoted by A, B, and C, while sentences in $\mathcal{O}$ are denoted by α, β, and γ. Δ denotes a sentences in a propositional language over $\Sigma \cup \Omega$. Σ and Ω induce two sets of truth assignments: T_Σ and T_Ω. Members of T_Σ are denoted by $LT, LT_1, LT_2, \ldots$, while members of T_Ω are denoted by OT, OT_1, $OT_2, \ldots$ A composite truth assignment is a pair (LT_i, OT_j). Finally, I overload the symbols LT and OT: LT denotes a sentence in $\mathcal{L}$ that is satisfied only by LT, and OT denotes a sentence in $\mathcal{O}$ that is satisfied only by OT.

Proof of Theorem 8.2.8

$\Longrightarrow$ Assume that I is a prime implicant for $Label(A, \Delta, \mathcal{O})$ and is consistent with Δ:

1. $I \models Label(A, \Delta, \mathcal{O})$.
2. $I' \models Label(A, \Delta, \mathcal{O})$ only if $I \equiv I'$ or $I \models I'$.
3. $\Delta \not\models \neg I$.

Then

1. Suppose $I \notin \mathcal{O}$, and let I' be the result of removing from I those literals that are not in $\mathcal{O}$. Then $I' \models Label(A, \Delta, \mathcal{O})$ and $I \models I'$, which is a contradiction with the assumption. Hence, $I \in \mathcal{O}$ and $\neg I \in \mathcal{O}$.
2. Since $I \models Label(A, \Delta, \mathcal{O})$, then $\Delta \wedge I \models A$. Hence, $\Delta \models I \supset A$.
3. Let I' be a conjunctive clause such that $\Delta \models I' \supset A$, $I' \not\equiv I$, and $\neg I' \models \neg I$. Then $\Delta \wedge I' \models A$ and $I \models I'$. Moreover, $I' \in \mathcal{O}$ because $I \in \mathcal{O}$. Hence, $I' \models Label(A, \Delta, \mathcal{O})$, which is a contradiction with the assumption. Hence, there is no such I'.
4. $\Delta \not\models \neg I$.

Therefore, $\neg I$ belongs to $\mathcal{O}$ and is a minimal support for A with respect to Δ.

$\Longleftarrow$ Assume that $\neg I$ belongs to $\mathcal{O}$ and is a minimal support for A with respect to Δ:

1. $\neg I \in \mathcal{O}$.
2. $\Delta \models I \supset A$.
3. $\Delta \models I' \supset A$ only if $\neg I' \equiv \neg I$ or $\neg I' \models \neg I$.
4. $\Delta \not\models \neg I$.

Then

1. $I \in \mathcal{O}$.
2. Since, $\Delta \models I \supset A$, then $\Delta \wedge I \models A$. Hence, $I \models Label(A, \Delta, \mathcal{O})$, which follows from the definition of $Label(A, \Delta, \mathcal{O})$.
3. Let I' be a conjunctive clause I' such that $I' \models Label(A, \Delta, \mathcal{O})$, $I \not\equiv I'$, and $I \models I'$. Then $\Delta \wedge I' \models A$, $\neq I' \not\equiv \neg I$, and $\neg I' \models \neg I$, which is a contradiction with the assumption. Hence, there is no such I'.
4. $\Delta \not\models \neg I$.

Therefore, I is a prime implicant for $Label(A, \Delta, \mathcal{O})$ and is consistent with Δ. ∎

Lemma H.0.5 For any satisfiable A, there exists LT such that

1. $LT \models A$.
2. $\Phi(\neg A) \wedge \neg\Phi(LT)$ is satisfiable.

Proof of Lemma H.0.5

Suppose otherwise:
For all LT such that $LT \models A$, we have $\Phi(\neg A) \models \Phi(LT)$.
Then $\Phi(\neg A) \models \bigwedge_{LT \models A} \Phi(LT) \equiv \Phi(A)$.
A contradiction with the definition of Φ: $\Phi(\neg A) \wedge \Phi(A) \equiv \mathbf{false}$. ∎

Lemma H.0.6 $(LT, OT) \models \Delta^{\Phi}$ precisely when $OT \models \neg\Phi(LT)$.

Proof of Lemma H.0.6

$\Longrightarrow$ Suppose $(LT, OT) \models \Delta^{\Phi}$.
Since $\Phi(LT) \supset \neg LT \in \Delta^{\Phi}$, we have
$(LT, OT) \models \Phi(LT) \supset \neg LT$.
That is, $(LT, OT) \models \neg\Phi(LT) \vee \neg LT$.
Hence, $OT \models \neg\Phi(LT)$.

$\Longleftarrow$ Suppose $(LT, OT) \models \neg\Phi(LT)$.
Must show that $(LT, OT) \models \Phi(A) \supset \neg A$
for all $\Phi(A) \supset \neg A \in \Delta^{\Phi}$.

Case $LT \models \neg A$:

Then $(LT, OT) \models \Phi(A) \supset \neg A$.

Case $LT \models A$:

By definition of Φ,
$\Phi(A) \models \Phi(LT)$, and
$\neg\Phi(LT) \models \neg\Phi(A)$.
Since $OT \models \neg\Phi(LT)$, we have $OT \models \neg\Phi(A)$, and
$(LT, OT) \models \Phi(A) \supset \neg A$. ∎

Lemma H.0.7 If $OT \models \neg\Phi(A)$, then there exists LT such that

1. $LT \models A$, and
2. $OT \models \neg\Phi(LT)$.

Proof of Lemma H.0.7
Suppose that $OT \models \neg\Phi(A)$.
Then A is satisfiable.
If A is unsatisfiable, then by definition of Φ,
$\Phi(A) = \mathbf{true}$, and
$\neg\Phi(A) = \mathbf{false}$.
A contradiction with supposition.

Suppose that for all LT such that $LT \models A$, we have $OT \models \Phi(LT)$.
Then $OT \models \bigwedge_{LT \models A} \Phi(LT) \equiv \Phi(A)$.
A contradiction with premise of lemma: $OT \models \neg\Phi(A)$. ∎

Proof of Theorem 8.3.3

By definition of Δ^Φ,
$\Delta^\Phi \models \Phi(\neg A) \supset A$, and
$\Delta^\Phi \wedge \Phi(\neg A) \models A$.

Suppose $\Delta^\Phi \wedge \alpha \models A$.
Must show that $\alpha \models \Phi(\neg A)$.
Suppose $\alpha \not\models \Phi(\neg A)$.
Must establish a contradiction.

Case $\alpha =$ **false**:

$\alpha \models \Phi(\neg A)$.
A contradiction.

Case $\alpha \neq$ **false**:

By supposition, $\alpha \wedge \neg\Phi(\neg A)$ is satisfiable:
There exists OT such that $OT \models \alpha \wedge \neg\Phi(\neg A)$.
Given Lemma H.0.7 and $OT \models \neg\Phi(\neg A)$,
there must exist LT such that $LT \models \neg A$ and $OT \models \neg\Phi(LT)$.
Given Lemma H.0.6 and $OT \models \neg\Phi(LT)$,
$(LT, OT) \models \Delta^\Phi$.
Hence, $(LT, OT) \models \Delta^\Phi \wedge \alpha \wedge \neg A$, and
$\Delta^\Phi \wedge \alpha \not\models A$.
A contradiction. ∎

Proof of Theorem 8.3.2

It suffices to show that if $\alpha \not\equiv \mathbf{false}$ and $\alpha \in \mathcal{O}$, then $\Delta^{\Phi} \wedge \alpha$ is satisfiable.

Suppose $\alpha \not\equiv \mathbf{false}$ and $\alpha \in \mathcal{O}$.

There must exist some LT such that $\neg\Phi(LT) \wedge \alpha$ is satisfiable.

To see why, suppose that $\alpha \models \Phi(LT)$ for all LT.

Then $\alpha \models \bigwedge_{LT} \Phi(LT) = \mathbf{false}$.

A contradiction.

Let OT be a model of $\neg\Phi(LT) \wedge \alpha$.

By Lemma H.0.6, $(LT, OT) \models \Delta^{\Phi}$.

Hence, $(LT, OT) \models \Delta^{\Phi} \wedge \alpha$. ∎

Proof of Theorem 8.3.4

Must show the following:

1. $\mathit{Label}(\mathbf{true}, \Delta, \mathcal{O}) = \mathbf{true}$.

   The weakest sentence α in $\mathcal{O}$ such that $\Delta \wedge \alpha \models \mathbf{true}$ is clearly **true**.

2. $\mathit{Label}(\mathbf{false}, \Delta, \mathcal{O}) = \mathbf{false}$.

   Note that Δ is non–committal about $\mathcal{O}$. That is, the only sentences in $\mathcal{O}$ that are entailed by Δ are valid. Hence, the only sentences in $\mathcal{O}$ that are contradictory with Δ are unsatisfiable. Therefore, the weakest sentence α in $\mathcal{O}$ such that $\Delta \wedge \alpha \models \mathbf{false}$ is clearly **false**.

3. $\mathit{Label}(A \wedge B, \Delta, \mathcal{O}) = \mathit{Label}(A, \Delta, \mathcal{O}) \wedge \mathit{Label}(B, \Delta, \mathcal{O})$.

   Let α and β be the weakest sentences in $\mathcal{O}$ such that

   $$\Delta \wedge \alpha \models A \text{ and } \Delta \wedge \beta \models B.$$

   Must show that $\alpha \wedge \beta$ is the weakest sentence in $\mathcal{O}$ such that

   $$\Delta \wedge \alpha \wedge \beta \models A \wedge B.$$

   It suffices to show that if

   $$\Delta \wedge \gamma \models A \wedge B,$$

   then $\gamma \models \alpha \wedge \beta$. Suppose

   $$\Delta \wedge \gamma \models A \wedge B.$$

   Then

   $$\Delta \wedge \gamma \models A \text{ and } \Delta \wedge \gamma \models B.$$

   Therefore, $\gamma \models \alpha$, $\gamma \models \beta$, and $\gamma \models \alpha \wedge \beta$.

4. $\mathit{Label}(A, \Delta, \mathcal{O}) = \mathit{Label}(B, \Delta, \mathcal{O})$ when $\models A \equiv B$. Follows because the definition of $\mathit{Label}(A, \Delta, \mathcal{O})$ does not depend on the syntax of A. ∎

Proof of Theorem 8.4.3

Suppose that α is the conjunction of all disjunctive clauses I, such that I belongs to $\mathcal{O}$ and I is a prime implicate of Δ. We must show that

1. $\Delta \models \alpha$.

2. If $\Delta \models \alpha \wedge \beta$ and $\beta \in \mathcal{O}$, then $\alpha \models \beta$.

$\Delta \models \alpha$ follows easily from the definition of α. Now suppose that $\Delta \models \alpha \wedge \beta$ and $\beta \in \mathcal{O}$. Let us express β in terms of its prime implicates,

$$\beta = \bigwedge J, \text{ where } J \text{ is a prime implicate of } \beta.$$

Since $\Delta \models \beta$, each J is an implicate of Δ. Therefore, for each J, there is some I such that

1. I is a prime implicate of Δ.

2. I belongs to $\mathcal{O}$.

3. $I \models J$.

Therefore, α entails each J and also entails β. ∎

Proof of Theorem 8.5.1

Must show that $\neg\Phi(A)$ is the strongest sentence in $\mathcal{O}$ such that

$$\Delta^{\Phi} \wedge A \models \neg\Phi(A).$$

By Theorem 8.3.3, the sentence $\Phi(A)$ is logically the weakest in $\mathcal{O}$ such that

$$\Delta^{\Phi} \wedge \Phi(A) \models \neg A.$$

Therefore, $\neg\Phi(A)$ is logically the strongest in $\mathcal{O}$ such that

$$\Delta^{\Phi} \wedge A \models \neg\Phi(A). \blacksquare$$

Proof of Theorem 8.3.6

Let Φ be the state of belief satisfying $CN = \langle \mathcal{L}, \mathcal{O}, \mathcal{G}, \mathcal{SO} \rangle$. We must show that

$$\Delta^{\Phi} = \bigwedge_{A \in \mathcal{L}} \Phi(A) \supset \neg A$$

is equivalent to

$$\Delta^{CN} = \bigwedge_{i \in N} \underline{i\diamond} \wedge \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \supset \neg \underline{i}.$$

It suffices to show the following:

$$\begin{aligned} \Delta^{\Phi} &\models \underline{i\diamond} \wedge \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \supset \neg \underline{i} \\ \Delta^{CN} &\models \Phi(A) \supset \neg A. \end{aligned}$$

By definition of Φ, $\Phi(\underline{i\diamond} \wedge \underline{i}) = \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\underline{i\diamond})$. Therefore,

$$\begin{aligned} \Delta^{\Phi} &\models \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\underline{i\diamond}) \supset \neg(\underline{i} \wedge \underline{i\diamond}) \\ &\models \underline{i\diamond} \wedge (\mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \vee \Phi(\underline{i\diamond})) \supset \neg \underline{i} \\ &\models \underline{i\diamond} \wedge \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \supset \neg \underline{i}. \end{aligned}$$

Suppose that A is a complete sentence. Then

$$A \equiv \bigwedge_{A \models \underline{i\diamond} \wedge \underline{i}} \underline{i\diamond} \wedge \underline{i}.$$

And by definition of Φ,

$$\Phi(A) = \bigvee_{A \models \underline{i\diamond} \wedge \underline{i}} \mathcal{SO}_{\underline{i\diamond}}(\underline{i}).$$

Moreover,

$$\begin{aligned} \Delta^{CN} &\models \underline{i\diamond} \wedge \underline{i} \supset \neg \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \\ &\models \bigwedge_{A \models \underline{i\diamond} \wedge \underline{i}} \underline{i\diamond} \wedge \underline{i} \supset \bigwedge_{A \models \underline{i\diamond} \wedge \underline{i}} \neg \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \\ &\models \bigwedge_{A \models \underline{i\diamond} \wedge \underline{i}} \underline{i\diamond} \wedge \underline{i} \supset \neg \bigvee_{A \models \underline{i\diamond} \wedge \underline{i}} \mathcal{SO}_{\underline{i\diamond}}(\underline{i}) \\ &\models A \supset \neg \Phi(A) \\ &\models \Phi(A) \supset \neg A. \end{aligned}$$

Suppose that A is not a complete sentence. There must exist complete sentences $\{A_i\}$ such that A is equivalent to $\bigvee_i A_i$. Then

$$\begin{aligned}
\Delta^{CN} &\models \Phi(A_i) \supset \neg A_i \\
&\models \bigwedge_i \Phi(A_i) \supset \bigwedge_i \neg A_i \\
&\models \Phi(\bigvee_i A_i) \supset \neg \bigvee_i A_i \\
&\models \Phi(A) \supset \neg A. \blacksquare
\end{aligned}$$

Bibliography

[Aleliunas, 1988] Aleliunas, Romas 1988. A new normative theory of probabilistic logic. In *Proceedings of CSCSI–88*. CSCSI. 67–74.

[Bonissone, 1987] Bonissone, Piero 1987. Summarizing and propagating uncertain information with triangular norms. *International Journal of Approximate Reasoning* 1:71–101.

[Cooper, 1990] Cooper, F. G. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence* 42(2-3):393–405.

[Darwiche, 1993] Darwiche, Adnan Y. 1993. Fuzzy probability calculus. (Submitted to IEEE–FUZZ–93).

[de Kleer *et al.*, 1992] de Kleer, Johan; Mackworth, Alan K.; and Reiter, Raymond 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2-3):197–222.

[Doyle, 1990] Doyle, Jon 1990. Methodological simplicity in expert system construction: The case of judgements and reasoned assumptions. In Shafer, Glenn and Pearl, Judea, editors 1990, *Readings in Uncertain Reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, California. 689–693.

[Dubois and Prade, 1988] Dubois, Didier and Prade, Henri 1988. *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Plenum Press, New York.

[Fagin and Halpern, 1989] Fagin, Ronald and Halpern, Joseph Y. 1989. Uncertainty, belief, and probability. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI)*. 1161–1167.

[Fine, 1973] Fine, Terrence L. 1973. *Theories of Probability.* Academic Press, Inc.

[Ginsberg, 1988] Ginsberg, Matthew L. 1988. Multivalued logics: a uniform approach for reasoning in artificial intelligence. *Computational Intelligence* 4:265–316.

[Halpern and Fagin, 1990] Halpern, Joseph Y. and Fagin, Ronald 1990. Two views of belief: Belief as generalized probabilities and belief as evidence. In *Proceedings of AAAI.* AAAI. 112–119.

[Horvitz *et al.*, 1989] Horvitz, Eric J.; Cooper, Gregory F.; and Suerdmont, H. Jacques 1989. Bounded conditioning: Flexible inference for decisions under scarce resources. Technical Report KSL-89-42, Knowledge Systems Laboratory, Stanford University.

[Hunter, 1991] Hunter, David 1991. Non–monotonic reasoning and the reversibility of belief change. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence.* 159–164.

[Koopman, 1940] Koopman, Bernard O. 1940. The axioms and algebra of intuitive probability. *Annals of Mathematics* 41(2):269–292.

[Krantz *et al.*, 1971] Krantz, D.; Luce, R. D.; Suppes, P.; and Tversky, A. 1971. *Foundation of measurement*, volume I. Academic Press, New York.

[Kraus *et al.*, 1990] Kraus, S.; Lehmann, D.; and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 24(1-2):167–207.

[Pearl, 1988] Pearl, Judea 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., San Mateo, California.

[Peot and Shachter, 1991] Peot, Mark A. and Shachter, Ross D. 1991. Fusion and propagation with multiple observations in belief networks. *Artificial Intelligence* 48(3):299–318.

[Polya, 1954] Polya, George 1954. *Patterns of Plausible Inference.* Princeton University Press, Princeton, NJ.

[Reiter and de Kleer, 1987] Reiter, Ray and de Kleer, Johan 1987. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proceedings of AAAI.* AAAI. 183–188.

[Rescher, 1969] Rescher, Nicholas 1969. *Many–Valued Logic.* McGraw–Hill, Inc.

[Rosser and Turquette, 1952] Rosser, J. B. and Turquette, A. R. 1952. *Many-Valued Logic.* North-Holland, Amsterdam.

[Shafer, 1976] Shafer, Glenn 1976. *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, NJ.

[Shenoy and Shafer, 1990] Shenoy, Parkash P. and Shafer, Glenn 1990. Axioms for probability and belief–function propagation. *Uncertainty in Artificial Intelligence; R.D. Shachter, T.S. Levitt, L.N. Kanal and J.F. Lemmer, eds.* 4.

[Shenoy, 1989] Shenoy, Parkash P. 1989. A valuation–based language for expert systems. *International Journal of Approximate Reasoning* 5(3):383–411.

[Spohn, 1987] Spohn, Wolfgang 1987. Ordinal conditional functions: A dynamic theory of epistemic states. *Causation in Decision, Belief Change, and Statistics; W. L. Harper and B. Skyrms, eds.* 2:105–134.

[Spohn, 1990] Spohn, Wolfgang 1990. A general non–probabilistic theory of inductive reasoning. In Kanal, L.; Shachter, R.; Levitt, T.; and Lemmer, J., editors 1990, *Uncertainty in Artificial Intelligence 4.* Elsevier Science Publishers. 149–158.

[Srinivas and Breese, 1992] Srinivas, Sampath and Breese, Jack 1992. IDEAL: Influence Diagram Evaluation and Analysis in Lisp documentation and user guide. Technical Report 23, Rockwell International Science Center, Palo Alto Laboratory.

[Suermondt and Cooper, 1988] Suermondt, H.J. and Cooper, G.F. 1988. Updating probabilities in multiply–connected belief networks. In *Fourth Workshop on Uncertainty in Artificial Intelligence.* 335–343.

[Verma, 1986] Verma, T. S. 1986. Causal networks: Semantic and expressiveness. Technical Report R-65, Cognetive Systems Laboratory, UCLA.

[Walley, 1973] Walley, Peter 1973. *Varieties of Modal and Comparative Probability.* University Microfilm International.

[Zadeh, 1975] Zadeh, L. A. 1975. The concept of a linguistic variable and its application to approximate reasoning—I&II. *Information Sciences* 8:199–249.

[Zadeh, 1976] Zadeh, L. A. 1976. The concept of a linguistic variable and its application to approximate reasoning—III. *Information Sciences* 9:43–80.