Aus dem Departement für Informatik
Universität Freiburg (Schweiz)

Argumentation Systems and Belief Functions

Inaugural–Dissertation

zur Erlangung der Würde eines *Doctor scientiarum informaticarum*
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Freiburg in der Schweiz

vorgelegt von

Norbert Lehmann

aus

Wünnewil (Freiburg)

Nr. 1340
Hausdruckerei, Universität Freiburg
Juni 2001

Von der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Freiburg in der Schweiz angenommen, auf Antrag von Prof. Dr. Jürg Kohlas (Universität Freiburg) und Prof. Dr. Serafin Moral (Universidad de Granada).

Freiburg, im Juni 2001

Der Leiter der Doktorarbeit:	Der Dekan:
Prof. Dr. Jürg Kohlas	Prof. Dr. Alexander von Zelewsky

Acknowledgements

There are many people who helped me finishing this thesis. The following list is therefore by no means exhausting.

First of all, I would like to thank my brother Markus, my sisters Doris and Brigitte, and my parents Agathe and Paul for creating a pleasant environment at home and for supporting me during difficult periods. I am very indebted to my friends of the second soccer team of the SC Düdingen and to my friends of the hockey team St.Ursen for sharing much free time with me. I am also very indebted to my friend Dominicq Riedo whom I know since I was a little boy and who always set an example to me.

I owe a great debt of gratitude to my collaborators and friends at the Department of Informatics of the University of Fribourg, especially Berhard Anrig, Dritan Berzati, Roman Bissig, Rolf Haenni, Patrick Hertelendy, Paul-André Monney and Michael Schumacher for the good cooperation and the interesting and critical discussions. Above all, the cooperation with Rolf Haenni proved to be very fruitful. I am also very grateful to my former colleague Louis Cardona for encouraging me to begin a thesis.

I would also like to thank Sabine Baeriswyl and Myriam Schweingruber for improving the language of my thesis and for correcting many spelling mistakes.

Finally, I would like to thank Prof. Jürg Kohlas, the director of my thesis. His continuous support and our discussions during the last few years inspired me in many aspects. I would also like to thank Prof. Serafin Moral and Prof. Prakash Shenoy who kindly accepted being experts of my thesis.

Abstract

Uncertain knowledge can be represented in the framework of argumentation systems. In this framework, uncertainty is expressed using so-called assumptions. Depending on the setting of the assumptions, a given hypothesis of interest can be proved or falsified. The main goal of assumption-based reasoning is to determine the set of all supporting arguments for a given hypothesis. Such a supporting argument is a particular setting of assumptions.

The assignment of probabilities to assumptions leads to the framework of probabilistic argumentation systems and allows an additional quantitative judgement of a given hypothesis. One possibility to compute the degree of support for a given hypothesis is to compute first the corresponding set of supporting arguments and then to derive the desired result. The problem of this approach is that the set of supporting arguments is sometimes very huge and can't be represented explicitly.

This thesis proposes an alternative way for computing degrees of support which is often superior to the first approach. Instead of computing a symbolic result from which the numerical result is derived, we avoid symbolic computations right away. This can be done due to the fact that degree of support corresponds to the notion of normalized belief in Dempster-Shafer theory. We will show how a probabilistic argumentation system can be transformed into a set of independent mass functions.

For efficient computations, the local computation framework of Shenoy is used. In this framework, computation is based on a message-passing scheme in a join tree. Four different architectures could be used for propagating potentials in the join tree. These architectures correspond to a complete compilation of the knowledge which allows to answer queries fast. In contrast, this thesis proposes a new method which corresponds to a partial compilation of the knowledge. This method is particularly interesting if there are only a few queries. In addition, it can prevent that the join tree has to be reconstructed in order to answer a given query.

Finally, the language ABEL is presented. It allows to express probabilistic argumentations systems in a convenient way. We will show how several examples from different domains can be modeled using ABEL. These examples are also used to point out important aspects of the computational theory presented in the first chapters of this thesis.

Zusammenfassung

Das Konzept der Argumentations-Systeme dient dem Zweck der Darstellung von insicherer oder unpräziser Information. Unsicherheit wird in Argumentations-Systemen durch sogenannte Annahmen dargestellt. Eine gegebene Hypothese kann dann in Abhängigkeit der Annahmen bewiesen oder verworfen werden. Hauptaufgabe des Annahmen-basierten Schliessens ist die Bestimmung von Argumenten welche eine gegebene Hypothese stützen.

Die Zuordnung von Wahrscheinlichkeiten zu den Annahmen führt zum Konzept der probabilistischen Argumentations-Systeme. Eine zusätzliche quantitative Beurteilung einer gegebenen Hypothese wird dadurch möglich. Ein erster Ansatz den Grad der Unterstützung einer Hypothese zu berechnen besteht darin, zuerst die Menge aller stützenden Argumente zu berechnen Das gewünschte numerische Resultat kann dann daraus abgeleitet werden. Häufig ist dieser Ansatz jedoch nicht durchführbar weil die Menge der unterstützenden Argumente zu gross und deshalb nicht explizit darstellbar ist.

In dieser Arbeit stellen wir einen alternativen Ansatz zur Berechnung des Grades der Unterstützung einer Hypothese vor. Dieser alternative Ansatz ist oft effizienter als der ersten Ansatz. Anstatt ein symbolisches Zwischenresultats zu berechnen von welchem dann das numerische Endresultat abgeleitet wird, vermeiden wir symbolisches Rechnen schon ganz zu Beginn. Dies ist möglich weil der Grad der Unterstützung zum Begriff der Glaubwürdigkeit in der Dempster-Shafer Theorie equivalent ist. Wir werden zeigen wie ein gegebenes probabilistisches Argumentations-System in eine Menge von equivalenten Mass Funktionen überführt werden kann.

Als Grundlage für die Berechnungen wird das Konzept der Valuations Netzwerke verwendet. Dadurch wird versucht, die Berechnungen möglichst effizient durchzuführen. Es gibt dabei vier verschiedene Rechenarchitekturen. Diese vier Rechenarchitekturen entsprechen einer vollständigen Kompilation der vorhandenen Informationen. Der Vorteil davon ist dass Abfragen dann sehr schnell beantwortet werden können. Im Gegensatz dazu stellen wir in dieser Arbeit eine neue Methode vor die eher einer partiellen Kompiliation der vorhandenen Informationen entspricht. Diese neue Methode ist vorallem interessant falls nur wenige Abfragen zu beantworten sind. Des weitern kann diese Methode verhindern, dass ein Valuationsnetz zur Beantwortung einer Abfrage neu konstruiert werden muss.

Zum Schluss geben wir eine Einführung in die Modelliersprache ABEL. Diese Sprache erlaubt, probabilistische Argumentations-Systeme auf eine geeignete und komfortable Art und Weise zu formulieren. Wir zeigen wie Beispiele aus verschiedenen Anwendungsgebieten mit ABEL modelliert werden köennen. Diese Beispiele werden auch dazu verwendet, wichtige Aspekte der in den ersten Kapiteln dieser Arbeit dargestellten Rechentheorie zu unterstreichen.

Contents

1 Introduction

Inconsistent, inaccurate, and uncertain information is an elementary part of the world we are living in. For human beings it is not really a problem to deal with that kind of information. Each one of us is able to take reasonable decisions based on inconsistent, inaccurate, and uncertain information. In contrast, computers deal with consistent, precise, and certain information. The integration of real-world information on computers is a main topic of Artificial Intelligence. Several methods and concepts have been developed in this subfield of computer science to represent and to draw inferences which are based on real-world information. In this work, we focus especially on numerical computations for drawing inferences in the field of assumption-based reasoning.

1.1 Motivation and Purpose

The German mathematician and logician Gottfried Wilhelm Leibniz (1646-1716), who invented independently of Sir Isaac Newton the differential and integral calculus, recommended several times that serious attention should be paid to develop a theory of probabilistic reasoning. In his *Ars Conjectandi*, which was published posthumously, the Swiss mathematician Jacob Bernoulli (1654-1705) did the first steps in that direction. He distinguished between *necessary* and *contingent* sentences and studied what conclusions could be derived in that case. The Swiss-German mathematician Johann Heinrich Lambert (1728-1777) extended the work of Bernoulli in several aspects. In 1764, he published *Neues Organon*, which is mainly about **formal logic** and **probability calculus**.

More than two hundred years later, the statisticien Arthur Dempster studied from a pure mathematical point of view upper and lower bounds of probability distributions induced by a multivalued mapping. In (Dempster, 1967), he laid the foundations of the so-called **Dempster–Shafer theory**. Glenn Shafer continued the work of Dempster. Ten years later, he published (Shafer, 1976) a theory of evidence, where inferences are drawn from diverse sources of evidence.

Short time later, he showed in (Shafer, 1979) that the reflections of Bernoulli and Lambert can be considered as predecessor of his theorie of evidence.

Several autors have shown that Dempster-Shafer theory of evidence can be conceived as a theory of *probability of provability* (Pearl, 1988; Laskey & Lehner, 1989) or as a *reliability theory of reasoning with unreliable arguments* (Kohlas, 1981; Kohlas, 1997; Besnard & Kohlas, 1995). By incorporating a symbolic part into the formalism, the **theory of hints** (Kohlas & Monney, 1995) explicitly pointed out the close connection to *Logic*.

The connection to Logic is even much more explicit in the closely related theory of **assumption-based reasoning** (Kohlas & Monney, 1993; Kohlas & Haenni, 1996). There, uncertainty is expressed by so-called assumptions. Depending on the setting of these assumptions, a given hypothesis of interest can be proved or falsified. A particular setting of a collection of assumptions is called an argument. The main goal of assumption-based reasoning is to determine the set of arguments in favor of a given hypothesis of interest. In that way, a qualitative judgment of hypotheses is obtained. The assignment of **probabilities** to assumptions leads to **probabilistic argumentation systems** (Haenni, 1998; Anrig *et al.*, 1999; Haenni *et al.*, 2000). In that way, a quantitative judgment of hypotheses is possible.

The aim of this work is to study the computation of **quantitative judgments** in the field of **probabilistic argumentation systems**. A rather big importance will be attached to computational theories for efficient computations. In particular, we will look at the framework of **valuation networks** (Shenoy, 1989; Shenoy, 1992; Shenoy, 1994), which allows local computation of marginals in a **join tree** on the basis of a message-passing scheme.

1.2 Overview

In Chapter 2, we will introduce **assumption-based reasoning**. The main objective of this theory is to derive arguments, that is, particular settings of collections of assumptions, in favor or against a given hypothesis of interest. The assignment of **probabilities** to assumptions then leads to **probabilistic argumentation systems** and makes a quantitative judgment of hypotheses possible. The notion of **degree of support** will be defined in this context.

The degree of support of a hypothesis can be obtained by first computing symbolic arguments and then calculating the probability of the corresponding formula. However, the set of symbolic arguments may be difficult or even impossible to compute if it is very large. The aim of this work is to propose an alternative way for computing degrees of support. For this, we will introduce in Chapter 3 the **Dempster-Shafer theory** of evidence.

In Chapter 4, we will build a bridge between *probabilistic argumentation systems* and *Dempster-Shafer theory*. In particular, we will show that a probabilistic argumentation system can always be transformed into an equivalent set

of independent potentials. Computing degree of support then corresponds to computing **normalized belief** in Dempster-Shafer theory.

One way to obtain the normalized belief is to compute the joint potential by combining each of these independent potentials. Unfortunately, the joint potential can hardly ever be built explicitly in that way. A better way is given by the framework of **valuation networks** which enables local computation of marginals of a joint valuation. We will see in Chapter 5 that knowledge is represented in this general framework by so-called **valuations** and inferences are drawn using two operators called **combination** and **marginalization**.

The elimination of variables is of central importance in Valuation Networks. By eliminating one variable after another, the marginal for an arbitrary set of variables can be computed. In Chapter 6, we will discuss several **heuristics** for determinating such a **variable elimination sequence**. In addition, we will introduce the notion of $\mathcal{A}$**-disjoint join trees**.

By the elimination of one variable after another, a **join tree** is constructed. Join trees are especially valuable if several marginals have to be computed. Marginals are then computed on the basis of a message-passing scheme. In Chapter 7, we will look at the traditional approach, where an **inward propagation phase** and an **outward propagation phase** is distinguished.

In Chapter 8, we will present an alternative approach, where outward propagation is replaced by a **partial inward propagation**. This alternative approach is especially valuable if only a few queries have to be answered.

For efficient computations, it is very important that appropriate data structures are used. In Chapter 9, we will therefore discuss **implementation aspects** for doing numerical computations.

The **language ABEL** will be presented in Chapter 10. It represents a convenient way to express assumption-based knowledge and to formulate queries. Thus, ABEL is not only a language, it is also the name of a **software package** which includes a **solver part** for doing inference.

To consolidate the knowledge about ABEL, we will look in Chapter 11 at several **applications**. The aim of this chapter is to fill the gap between practical examples and the sometimes complicated mathematical theories.

Finally, the last chapter will give an idea in what direction future research will go. Above all, we think that **approximation methods** are required because exact computation is not feasible for many examples.

1.3 An Introductory Example

The graph shown in Figure 1.1 represents a **communication network**. It consists of nodes a, b, x, y, and z, which are connected by communication wires $w_1, \ldots, w_6$. In order that a node can send a message to another node, there must be at least one directed path from the former node the later. Such a

directed path from one node to another is called a communication path. For example, there are four possible communication paths between node a and node b. These are the following:

$$\langle w_1 - w_5 \rangle \; ; \; \langle w_1 - w_3 - w_6 \rangle \; ; \; \langle w_2 - w_4 - w_6 \rangle \; ; \; \langle w_2 - w_4 - w_3 - w_5 \rangle$$

If one or several communication wires are broken, some point-to-point connections may be impossible. For example, if the communication wire w_1 is broken, node a can still communicate with node b because there are still two communication paths which connect node a with node b. These communication paths are the following:

$$\langle w_2 - w_4 - w_6 \rangle \; ; \; \langle w_2 - w_4 - w_3 - w_5 \rangle$$

However, if in addition also the communication wire w_2 is broken, then a message which is sent from node a does never reach node b.

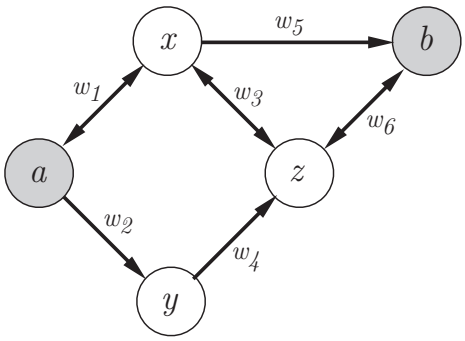

Figure 1.1: A Small Communication Network.

Suppose now that the probability that a wire breaks down is given for each of the communication wires. The problem to solve then is to compute the reliability of a communication, this means that a message which is sent from one node reaches another. One possible approach is to compute first all communication paths between the two nodes. The reliability of the communication can then be derived using the failure probabilities of the communication wires.

The main problem of this approach is that the set of possible communication paths can be huge. For example, consider the communication network shown in Figure 1.2. Its structure is similar to the previous communication network, only that it contains more nodes and more communication wires.

For this bigger communication network, the set of communication paths between the node a and the node b is not as easy to compute as for the previous communication network. It is almost certain that a human being would miss some of the 46 possible communication paths. Nevertheless, a computer can still compute all these communication paths quite efficiently. However, also the electronic brain reaches its limits for even bigger communication networks. In that case, the set of possible communication paths between two nodes can really

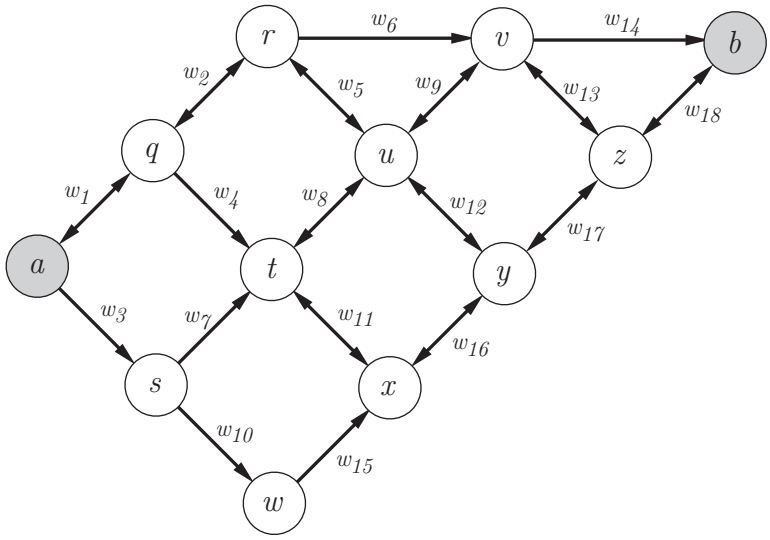

Figure 1.2: A Bigger Communication Network.

be huge. The computation of the reliability of the communication between two nodes then fails for that reason.

The aim of this work is to propose an alternative approach for computing numerical results. Instead of computing a symbolic result (such as the set of communication paths) from which the numerical result is derived, we avoid symbolic computations right away. This alternative approach is often superior to the first approach.

2

Assumption-Based Reasoning

Classical **propositional logic** can be used to encode information or knowledge. In particular, uncertainty can be incorporated by including so-called assumptions which may or may not be true. Whether a given hypothesis can be proved or not depends on the setting of these assumptions. **Assumption-based reasoning**(Kohlas & Monney, 1993; Kohlas & Haenni, 1996) consists then to derive arguments in favor or against the given hypothesis of interest, that is, arguments which allow to prove or to falsify the hypothesis. Therefore, an argument represents a particular setting of a collection of assumptions and can be seen as a chain of possible events that makes the hypothesis true or false. De Kleer's **assumption-based truth maintenance systems** (ATMS) represent a general architecture for determinating such arguments (de Kleer, 1986a; de Kleer, 1986b).

Although propositional logic allows to express a wide range of interesting problems, it is often not very convenient. This inconvenience can be omitted if its generalization given by **set constraint logic** is used instead. The main difference to propositional logic is that variables in the language of set constraint logic are not restricted to only two values.

The assignment of **probabilities** to the assumptions leads to the theory of **probabilistic argumentation systems** (Haenni, 1998; Anrig *et al.*, 1999) and represents in some sense a combination of **logic** and **probability theory**. The incorporation of probability theory can be considered as an added value which allows a quantitative judgment of hypotheses.

In this chapter, the concept of **probabilistic argumentation systems** will be introduced. For this, we will follow the approach given in (Haenni *et al.*, 2000). The main difference is that here, **argumentation systems** as well as **probabilistic argumentation systems** will be based on **set constraint logic** instead of classical propositional logic. The concept of probabilistic argumentation systems will allow us to define the important notions **degree of support** and **degree of possibility**. However, we will first start with a short description of classical **propositional logic** and its generalization given by **set constraint logic**.

2.1 Propositional Logic

The building blocks of propositional logic are atomic statements called **propositions** which can either be true or false. For a finite set of propositions $\mathcal{P} = \{p_1, \ldots, p_n\}$, each $p_i \in \mathcal{P}$ is called an **atom** . The symbol $\bot$ represents the impossible statement and $\top$ represents the statement which is always true. Often, the impossible statement is also called **contradiction** or **falsity**, while **tautology** is usally used for the statement represented by $\top$.

Compound formulas can be build using negation and logical connectors:

(1) atoms, $\bot$, and $\top$ are formulas;

(2) if γ is a formula, then $\neg\gamma$ is a formula;

(3) if γ and δ are formulas, then $(\gamma \wedge \delta)$, $(\gamma \vee \delta)$, $(\gamma \rightarrow \delta)$, and $(\gamma \leftrightarrow \delta)$ are formulas.

Of course, unnecessary parentheses can be omitted by assigning priorities to the logical connectors. The set $\mathcal{L}_\mathcal{P}$ of all formulas generated by the above rules is called **propositional language** over $\mathcal{P}$. A formula $\gamma \in \mathcal{L}_\mathcal{P}$ is called **propositional sentence**.

2.1.1 Semantics

A propositional sentence can be evaluated by assigning truth values 0 (false) or 1 (true) to each proposition. The truth value of an arbitrary formula $\gamma \in \mathcal{L}_\mathcal{P}$ can then be obtained according to Table 2.1.

γ	δ	$\bot$	$\top$	$\neg\gamma$	$\gamma \wedge \delta$	$\gamma \vee \delta$	$\gamma \rightarrow \delta$	$\gamma \leftrightarrow \delta$
0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0
1	0	0	1	0	0	1	0	0
1	1	0	1	0	1	1	1	1

Table 2.1: Truth Values of Compound Formulas.

An assignment of truth values to the elements of $\mathcal{P} = \{p_1, \ldots, p_n\}$ is called **interpretation** relative to $\mathcal{P}$. $N_\mathcal{P} = \{0,1\}^n$ denotes the set of all 2^n different interpretations. Every interpretation $\mathbf{x} \in N_\mathcal{P}$ can be seen as a point $\mathbf{x} = (x_1, \ldots, x_n)$ in the n-dimensional product space $N_\mathcal{P}$. Each component $x_i \in \{0,1\}$ of $\mathbf{x}$ is associated with the corresponding proposition $p_i \in \mathcal{P}$.

An interpretation $\mathbf{x}$ relative to $\mathcal{P}$ is called a **model** of $\gamma \in \mathcal{L}_\mathcal{P}$ if γ evaluates to 1 under the interpretation $\mathbf{x}$. The set of all models of γ is denoted by

$N_{\mathcal{P}}(\gamma)$. Clearly, $N_{\mathcal{P}}(\gamma) \subseteq N_{\mathcal{P}}$ for all formulas $\gamma \in \mathcal{L}_{\mathcal{P}}$. A formula γ is called **unsatisfiable** if $N_{\mathcal{P}}(\gamma) = \emptyset$. Otherwise, γ is called **satisfiable**.

The notion of a model links propositional logic to the algebra of subsets of interpretations:

$$
\begin{aligned}
&(1)\ N_{\mathcal{P}}(\bot) &&= \emptyset,\\
&(2)\ N_{\mathcal{P}}(\top) &&= N_{\mathcal{P}},\\
&(3)\ N_{\mathcal{P}}(\neg\gamma) &&= N_{\mathcal{P}} - N_{\mathcal{P}}(\gamma),\\
&(4)\ N_{\mathcal{P}}(\gamma \wedge \delta) &&= N_{\mathcal{P}}(\gamma) \cap N_{\mathcal{P}}(\delta),\\
&(5)\ N_{\mathcal{P}}(\gamma \vee \delta) &&= N_{\mathcal{P}}(\gamma) \cup N_{\mathcal{P}}(\delta).
\end{aligned}
$$

A formula $\gamma \in \mathcal{L}_{\mathcal{P}}$ **entails** another formula $\delta \in \mathcal{L}_{\mathcal{P}}$ (denoted by $\gamma \models \delta$), if and only if $N_{\mathcal{P}}(\gamma) \subseteq N_{\mathcal{P}}(\delta)$. In this case, δ is a **logical consequence** of γ. Furthermore, γ and δ are **logically equivalent** (denoted by $\gamma \equiv \delta$), if and only if $N_{\mathcal{P}}(\gamma) = N_{\mathcal{P}}(\delta)$. Note that logically equivalent formulas represent exactly the same information.

2.1.2 Normal Forms

In propositional logic, the conjunctive and the disjunctive normal form are two types of formulas which are particularly important. These types of formulas are based on the notions of literals, clauses, and terms:

A **positive literal** is an element of $\mathcal{P} = \{p_1, \ldots, p_n\}$, while the elements of $\neg\mathcal{P} = \{\neg p_1, \ldots, \neg p_n\}$ are **negative literals**. $\mathcal{P}^{\pm} = \mathcal{P} \cup \neg\mathcal{P}$ denotes the set of all literals. A **clause** is a disjunction $\ell_1 \vee \cdots \vee \ell_s$ of literals $\ell_i \in \mathcal{P}^{\pm}$. The empty disjunction is also a clause and corresponds to $\bot$. A clause is called **proper**, if every propositional symbol appears at most once. The symbol $\mathcal{D}_{\mathcal{P}}$ represents the set of all proper clauses. Similarly, a **term** is a conjunction $\ell_1 \wedge \cdots \wedge \ell_s$ of literals $\ell_i \in \mathcal{P}^{\pm}$. Here, the empty conjunction corresponds to $\top$. A term is called **proper**, if every propositional symbol appears at most once. Finally, $\mathcal{C}_{\mathcal{P}}$ represents the set of all proper terms.

A **conjunctive normal form** (CNF for short) is a conjunction $\varphi_1 \wedge \cdots \wedge \varphi_r$ of proper clauses. Similarly, a **disjunctive normal form** (DNF for short) is a disjunction $\psi_1 \vee \cdots \vee \psi_r$ of proper terms. Note that every propositional sentence can be transformed into an equivalent conjunctive or disjunctive normal form (Chang & Lee, 1973). Often, CNF or DNF formulas are considered as sets of clauses and terms. For example, if Γ is the set of clauses for γ and Δ is the set of terms for δ, it is often convenient to write $\Gamma \models \Delta$, $\gamma \models \Delta$, or $\Gamma \models \delta$ instead of $\gamma \models \delta$. Similarly, $\Gamma \equiv \Delta$, $\gamma \equiv \Delta$, or $\Gamma \equiv \delta$ is sometimes used instead of $\gamma \equiv \delta$. Furthermore, $\neg\Gamma$ denotes the corresponding set of negated terms of $\neg\gamma$. Note that the negation of a clause is a term and, similarly, the negation of a term is a clause.

2.2 Set Constraint Logic

Propositional logic is a formal language to describe statements about binary variables. This is sufficient to express a certain class of problems. However, describing the world on the basis of binary variables is sometimes not very convenient. For that reason, propositional logic has been generalized to **set constraint logic** (SCL for short) (Anrig *et al.*, 1997c; Haenni & Lehmann, 1998). The main idea is that each variable can have a finite set of possible values. Constraints about the possible true value of a variable are then the atoms of the language. Such constraints therefore correspond somehow to the notion of literals in propositional logic.

Set constraint logic is closely related to **many-valued logic** (MVL for short) (Hähnle, 1994; Lu, 1996; Lu *et al.*, 1994; Murray & Rosenthal, 1994). The idea behind the MVL approach is that the set of possible truth values is extended from $\{0,1\}$ to an arbitrary set Θ. Depending on properties of Θ (finite, infinite, unordered, partially ordered, totally ordered, etc.), various classes of many-valued logics can be defined (Hähnle & Escalada-Imaz, 1997). The case of a finite and unordered set Θ leads to **signed logic**. The main difference between signed logic and the SCL-framework is that for signed logic, the same set of possible values is used for all variables. From this point of view, SCL is a more general approach, since each variable has its own set of possible values.

The alphabet of set constraint logic is a finite set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$. It is supposed that the true value of a variable $v \in \mathcal{V}$ is exactly one value of a given set of values Θ_v called **frame** of v. An expression $\langle v \in X \rangle$, where X is a subset of Θ_v, is called a **set constraint**. A set constraint $\langle v \in \{\theta_i\} \rangle$, $\theta_i \in \Theta_v$, is an **assignment** and is often abbreviated by $\langle v = \theta_i \rangle$.

SCL-formulas can be build using negation and logical connectors:

(1) set constraints, $\bot$ and $\top$ are SCL-formulas;

(2) if γ is a SCL-formula, then $\neg\gamma$ is a SCL-formula;

(3) if γ and δ are SCL-formulas, then $(\gamma \wedge \delta)$, $(\gamma \vee \delta)$, $(\gamma \rightarrow \delta)$ and $(\gamma \leftrightarrow \delta)$ are SCL-formulas.

Of course, unnecessary parentheses can be omitted by assigning priorities to the logical connectors. The symbol $\mathcal{L}_\mathcal{V}$ denotes the set of all SCL-formulas which can be generated by the above rules.

2.2.1 Semantics

The assignment of a specific value to every variable $v_i \in V$ is called an **interpretation**. The set of all possible interpretations is denoted by $N_\mathcal{V} = \Theta_{v_1} \times \cdots \times \Theta_{v_n}$. An interpretation $\mathbf{x} \in N_\mathcal{V}$ can be seen as a point $\mathbf{x} = (x_1, \ldots, x_n)$ in the n-dimensional product space $N_\mathcal{V}$. For a given interpretation $\mathbf{x}$, the truth value of a set constraint $\langle v_i \in X \rangle$ is 1 (true) whenever $x_i \in X$ and 0 (false)

otherwise. Given the truth values of the set constraints contained in a SCL-formula, the truth value of the SCL-formula itself can be determined in the same way as in propositional logic (see Subsection 2.1.1).

$N_{\mathcal{V}}(\gamma) \subseteq N_{\mathcal{V}}$ denotes the set of all interpretations for which a SCL-formula γ is true. γ **entails** another SCL-formula δ (denoted by $\gamma \models \delta$), if and only if $N_{\mathcal{V}}(\gamma) \subseteq N_{\mathcal{V}}(\delta)$. Furthermore, γ and δ are **equivalent** (denoted by $\gamma \equiv \delta$), if and only if $N_{\mathcal{V}}(\gamma) = N_{\mathcal{V}}(\delta)$. Note that equivalent SCL-formulas represent exactly the same information. The set of variables occuring in a SCL-formula ξ is called the **domain** of ξ and is denoted as $d(\xi)$. Sometimes, we write $N(\gamma)$ for $N_{d(\xi)}(\xi)$.

Some basic properties of SCL-formulas are given by axioms of **set theory** (Gries & Schneider, 1993) and can be used for simplifying SCL-formulas:

(1) $\langle v \in \emptyset \rangle \equiv \bot$,

(2) $\langle v \in \Theta_v \rangle \equiv \top$,

(3) $\neg\langle v \in X \rangle \equiv \langle v \in (\Theta_v - X) \rangle$,

(4) $\langle v \in X_1 \rangle \vee \langle v \in X_2 \rangle \equiv \langle v \in (X_1 \cup X_2) \rangle$,

(5) $\langle v \in X_1 \rangle \wedge \langle v \in X_2 \rangle \equiv \langle v \in (X_1 \cap X_2) \rangle$.

2.2.2 Normal Forms

The conjunctive and the disjunctive normal form are two types of formulas which are particularly important. In set constraint logic, these types of formulas are based on the notions of proper SCL-clauses and proper SCL-terms:

A set constraint $\langle v \in X \rangle$ is called **proper**, if $X \neq \emptyset$ and $X \neq \Theta_v$. A disjunction of proper set constraints $\langle v_1 \in X_1 \rangle \vee \cdots \vee \langle v_q \in X_q \rangle$, where every variable occurs at most once, is called a **SCL-clause**. Similarly, a conjunction of proper set constraints $\langle v_1 \in X_1 \rangle \wedge \cdots \wedge \langle v_q \in X_q \rangle$, where every variable occurs at most once, is called a **SCL-term**. Arbitrary disjunctions or conjunctions of set constraints can be transformed into corresponding SCL-clauses or SCL-terms by applying properties (1) to (5).

We say that a SCL-clause δ_1 **absorbs** another SCL-clause δ_2 whenever $N_{\mathcal{V}}(\delta_1) \subseteq N_{\mathcal{V}}(\delta_2)$. Similarly, a SCL-term γ_1 **absorbs** another SCL-term γ_2 whenever $N_{\mathcal{V}}(\gamma_1) \supseteq N_{\mathcal{V}}(\gamma_2)$. If Γ is a set of SCL-clauses, $\mu(\Gamma)$ denotes the result of removing all absorbed SCL-clauses from Γ. Similarly, $\mu(\Delta)$ is the result of removing absorbed SCL-terms from the set of SCL-terms Δ.

A **conjunctive normal form** is a conjunction $\delta_1 \wedge \cdots \wedge \delta_r$ of proper SCL-clauses. A **disjunctive normal form** is a disjunction $\gamma_1 \wedge \cdots \wedge \gamma_r$ of proper SCL-terms. Every SCL-formula can be transformed into an equivalent conjunctive or disjunctive normal form.

2.3 Argumentation Systems

In the following, the SCL-framework will be used to define **argumentation systems**. Nevertheless, propositional logic is a very important special case of set constraint logic. In addition, it is often much more appropriate to illustrate different concepts. Therefore, we will first define **propositional argumentation systems**:

Definition 2.1 *Let $\mathcal{P}$ and $\mathcal{A}$ be two disjoint sets of propositions. If ξ is a formula in $\mathcal{L}_{\mathcal{A}\cup\mathcal{P}}$, the triple $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$ is called* **propositional argumentation system**. *ξ is called the knowledge base of $\mathcal{AS}_{\mathcal{P}}$.*

Propositional argumentation systems are therefore restricted to binary variables. In contrast, an argumentation system does not have this restriction:

Definition 2.2 *Let $\mathcal{V}$ and $\mathcal{A}$ be two disjoint sets of variables. If ξ is a SCL-formula in $\mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$, the triple $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ is called* **argumentation system**. *ξ is called the knowledge base of $\mathcal{AS}_{\mathcal{V}}$.*

A propositional argumentation system is therefore a special case of an argumentation system. In fact, for a given propositional argumentation system $\mathcal{AS}_{\mathcal{P}}$ it is easy to construct an equivalent argumentation system $\mathcal{AS}_{\mathcal{V}}$.

An argumentation system $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ consists therefore of two disjoint sets $\mathcal{V} = \{v_1, \ldots, v_n\}$ and $\mathcal{A} = \{a_1, \ldots, a_m\}$ of variables and a knowledge base $\xi \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. The knowledge base ξ is often given as a set $\Sigma = \{\xi_1, \ldots, \xi_r\}$ of SCL-clauses $\xi_i \in \mathcal{D}_{\mathcal{A}\cup\mathcal{V}}$. In such a case, the corresponding conjunction $\xi = \xi_1 \wedge \cdots \wedge \xi_r$ can always be used instead.

The elements of $\mathcal{A}$ are called **assumptions** and are essential for expressing uncertain information. They are used to represent uncertain events, unknown circumstances, or possible risks and outcomes. The set $N_{\mathcal{A}}$ of possible interpretations relative to $\mathcal{A}$ is therefore of particular interest. Such an interpretation $\mathbf{s} \in N_{\mathcal{A}}$ is called **scenario**. It represents a possible state of the world and is a fundamental notion in argumentation systems. As an abuse of notation, we will sometimes write $\mathbf{s} \wedge \xi$ for $\mathbf{s} \in N_{\mathcal{A}}$ and $\xi \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. In this case, $\mathbf{s} \in N_{\mathcal{A}}$ is considered as a SCL-formula for which the set of interpretations is equal to $\mathbf{s}$.

2.3.1 Consistent and Inconsistent Scenarios

Some scenarios may become impossible with the given knowledge base. It is therefore necessary to distinguish two different types of scenarios:

Definition 2.3 *Let $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ be an argumentation system. A scenario $\mathbf{s} \in N_{\mathcal{A}}$ is called*

(1) **inconsistent** *relative to* ξ*, if and only if* $\mathbf{s} \wedge \xi \models \bot$*;*

(2) **consistent** *relative to* ξ *otherwise.*

The definition of inconsistent and consistent scenarios leads directly to the definition of the sets $I_{\mathcal{A}}(\xi)$ and $C_{\mathcal{A}}(\xi)$:

$$I_{\mathcal{A}}(\xi) = \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \xi \models \bot\} \tag{2.1}$$

$$C_{\mathcal{A}}(\xi) = \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \xi \not\models \bot\} \tag{2.2}$$

Clearly, $I_{\mathcal{A}}(\xi)$ and $C_{\mathcal{A}}(\xi)$ are complementary sets, that is,

$$C_{\mathcal{A}}(\xi) = N_{\mathcal{A}} - I_{\mathcal{A}}(\xi). \tag{2.3}$$

Example 2.1 Consistent and Inconsistent Scenarios Let the propositional argumentation system $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$ be given by $\mathcal{P} = \{p\}$, $\mathcal{A} = \{a_1, a_2\}$, and $\xi = (a_1 \rightarrow p) \wedge (a_2 \rightarrow \neg p)$. Then, $I_{\mathcal{A}}(\xi) = \{(1,1)\}$ is the set of inconsistent scenarios and $C_{\mathcal{A}}(\xi) = \{(0,0), (0,1), (1,0)\}$ is the set of consistent scenarios. The scenario $(1,1)$ is inconsistent because ξ is unsatisfiable when a_1 and a_2 are simultaneously true. ⊖

The distinction between consistent and inconsistent scenarios is the essence of argumentation systems. It introduces in a natural and convenient way non-monotonicity into set constraint logic. Non-monotonicity is the fundamental property of any formalism for dealing with uncertainty. The question, why and how the distinction between consistent and inconsistent scenarios leads to non-monotonicity, is discussed later in this chapter.

2.3.2 Supporting Scenarios

The situation becomes more interesting if a second SCL-formula $h \in \mathcal{L}_{\mathcal{A} \cup \mathcal{V}}$ called **hypothesis** is given. Hypotheses represent open questions or uncertain statements about some of the variables in $\mathcal{A} \cup \mathcal{V}$. What can be concluded from ξ about the possible truth of h with respect to the given set of assumptions $\mathcal{A}$? Possibly, if the assumptions are set according to some scenarios $\mathbf{s} \in N_{\mathcal{A}}$, then h may be a logical consequence of ξ. In other words, h is **supported** by certain scenarios.

Definition 2.4 *Let* $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ *be an argumentation system and let* $h \in \mathcal{L}_{\mathcal{A} \cup \mathcal{V}}$ *be a hypothesis. A scenario* $\mathbf{s} \in N_{\mathcal{A}}$ *is called*

(1) **quasi-supporting** *scenario for* h *relative to* ξ*, if and only if* $\mathbf{s} \wedge \xi \models h$*;*

(2) **supporting** *scenario for* h *relative to* ξ*, if and only if* $\mathbf{s} \wedge \xi \models h$ *and* $\mathbf{s} \wedge \xi \not\models \bot$*;*

(3) **possibly supporting** *scenario for* h *relative to* ξ*, if and only if* $\mathbf{s} \wedge \xi \not\models \neg h$*.*

These definitions lead directly to the definition of the sets $QS_{\mathcal{A}}(h,\xi)$, $SP_{\mathcal{A}}(h,\xi)$, and $PS_{\mathcal{A}}(h,\xi)$:

$$QS_{\mathcal{A}}(h,\xi) = \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \xi \models h\} \quad (2.4)$$

$$SP_{\mathcal{A}}(h,\xi) = \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \xi \models h,\ \mathbf{s} \wedge \xi \not\models \bot\} \quad (2.5)$$

$$PS_{\mathcal{A}}(h,\xi) = \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \xi \not\models \neg h\} \quad (2.6)$$

The difference between quasi-supporting and supporting scenarios is that quasi-supporting scenarios can be inconsistent. However, inconsistency is usually excluded. Therefore, supporting scenarios are more interesting.

Figure 2.1 illustrates the relation between different subsets of $N_{\mathcal{A}}$ with $I_{\mathcal{A}}(\xi) = \mathrm{A}$, $QS_{\mathcal{A}}(h,\xi) = \mathrm{A} + \mathrm{B}$, $SP_{\mathcal{A}}(h,\xi) = \mathrm{B}$, $PS_{\mathcal{A}}(h,\xi) = \mathrm{B} + \mathrm{C}$, and $C_{\mathcal{A}}(\xi) = \mathrm{B} + \mathrm{C} + \mathrm{D}$.

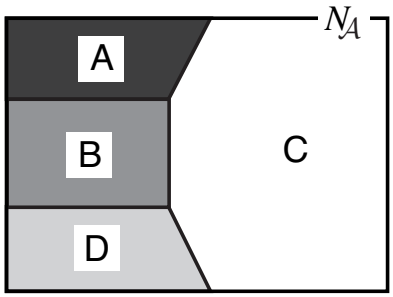

Figure 2.1: Different Subsets of Scenarios.

Note that $I_{\mathcal{A}}(\xi) \subseteq QS_{\mathcal{A}}(h,\xi)$, $SP_{\mathcal{A}}(h,\xi) \subseteq C_{\mathcal{A}}(\xi)$, and $PS_{\mathcal{A}}(h,\xi) \subseteq C_{\mathcal{A}}(\xi)$ for all hypotheses $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. Furthermore, $SP_{\mathcal{A}}(h,\xi) \subseteq QS_{\mathcal{A}}(h,\xi)$ and $SP_{\mathcal{A}}(h,\xi) \subseteq PS_{\mathcal{A}}(h,\xi)$.

Example 2.2 Quasi-Supporting and Supporting Scenarios Again, let the propositional argumentation system $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$ be given by $\mathcal{P} = \{p\}$, $\mathcal{A} = \{a_1, a_2\}$, and $\xi = (a_1 \rightarrow p) \wedge (a_2 \rightarrow \neg p)$. Then, $QS_{\mathcal{A}}(p,\xi) = \{(1,0),(1,1)\}$ and $SP_{\mathcal{A}}(p,\xi) = \{(1,0)\}$. Similarly, $QS_{\mathcal{A}}(\neg p,\xi) = \{(0,1),(1,1)\}$ and $SP_{\mathcal{A}}(\neg p,\xi) = \{(0,1)\}$. $\ominus$

The sets of inconsistent and consistent scenarios can be expressed in terms of quasi-supporting scenarios for $\bot$:

$$I_{\mathcal{A}}(\xi) = QS_{\mathcal{A}}(\bot,\xi) \quad (2.7)$$

$$C_{\mathcal{A}}(\xi) = N_{\mathcal{A}} - QS_{\mathcal{A}}(\bot,\xi) \quad (2.8)$$

Similarly, the set of supporting scenarios for h is determined by sets of quasi-supporting scenarios:

$$SP_{\mathcal{A}}(h,\xi) = QS_{\mathcal{A}}(h,\xi) - QS_{\mathcal{A}}(\bot,\xi) \quad (2.9)$$

Furthermore, the sets of supporting scenarios for $\bot$ and $\top$ are given by:

$$SP_{\mathcal{A}}(\bot, \xi) = \emptyset \tag{2.10}$$
$$SP_{\mathcal{A}}(\top, \xi) = C_{\mathcal{A}}(\xi) \tag{2.11}$$

The problem of computing sets of consistent, inconsistent, and supporting scenarios can therefore be solved by computing sets of quasi-supporting scenarios. The quasi-supporting scenarios are therefore important mainly for technical reasons.

An interesting situation to be considered is the case, where new information is added to the knowledge base ξ. Then, the number of quasi-supporting scenarios is monotonically increasing as shown by the following theorem:

Theorem 2.5 *If $\xi' = \xi \wedge \tilde{\xi} \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$ and $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$, then*

$$QS_{\mathcal{A}}(h, \xi) \subseteq QS_{\mathcal{A}}(h, \xi'). \tag{2.12}$$

Proof See Appendix, page 161. □

If new information is added, the set of supporting scenarios behaves non-monotonically, that is, it may either grow or shrink, both cases are possible. The reason for this is that according to Equation (2.9), $SP_{\mathcal{A}}(h, \xi)$ is the set difference of two monotonically growing sets $QS_{\mathcal{A}}(h, \xi)$ and $QS_{\mathcal{A}}(\bot, \xi)$.

The non-monotonicity of $SP_{\mathcal{A}}(h, \xi)$ is an important property of argumentation systems. It reflects a natural property of how a human's conviction or belief can change when new information is given. Non-monotonicity is therefore a fundamental property of any mathematical formalism for reasoning under uncertainty. Argumentation systems represent a natural and convincing way to achive non-monotonicity.

2.4 Symbolic Arguments

Sets of scenarios $S \subseteq N_{\mathcal{A}}$ (such as $I_{\mathcal{A}}(\xi)$, $C_{\mathcal{A}}(\xi)$, $QS_{\mathcal{A}}(h, \xi)$, and $SP_{\mathcal{A}}(h, \xi)$) tend to grow exponentially with the size of $\mathcal{A}$. An explicit representation as a list of elements $\mathbf{s} \in S$ is therefore not feasible. Therefore, an alternative representation is needed.

2.4.1 Representing Sets of Scenarios

An efficient representation is obtained by considering SCL-terms $\alpha \in \mathcal{C}_{\mathcal{A}}$ for which $N_{\mathcal{A}}(\alpha) \subseteq S$ holds. Let $T(S) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : N_{\mathcal{A}}(\alpha) \subseteq S\}$ be the set of all SCL-terms for which this condition holds. The set $T(S)$ is called **term representation** of S. A term $\alpha \in T(S)$ is called **minimal** in $T(S)$, if there is no other (shorter) term α' in $T(S)$ so that $\alpha \models \alpha'$. The corresponding set

$\mu T(S)$ of minimal terms is called **minimal term representation** of S. Note that

$$S = \bigcup_{\alpha \in T(S)} N_{\mathcal{A}}(\alpha) = \bigcup_{\alpha \in \mu T(S)} N_{\mathcal{A}}(\alpha). \tag{2.13}$$

Example 2.3 Term Representation Suppose that $\mathcal{A} = \{a_1, a_2\}$ is the set of assumptions for a propositional argumentation system. For $S \subseteq N_{\mathcal{A}}$ given by $S = \{(0,0),(1,0),(1,1)\}$, the term representation is $T(S) = \{a_1, \neg a_2, a_1 \wedge a_2, a_1 \wedge \neg a_2, \neg a_1 \wedge \neg a_2\}$. In addition, $\mu T(S) = \{a_1, \neg a_2\}$ is the minimal term representation of S. $\ominus$

The operations for sets of scenarios can be replaced by corresponding operations for minimal term representations (Haenni *et al.*, 2000). Therefore, a set of scenarios S will be represented from now on by its minimal term representation $\mu T(S)$.

2.4.2 Consistent and Inconsistent Arguments

Definition 2.6 *Let $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ be an argumentation system. A SCL-term $\alpha \in \mathcal{C}_{\mathcal{A}}$ is called*

(1) **inconsistent** *relative to ξ, if $N_{\mathcal{A}}(\alpha) \subseteq I_{\mathcal{A}}(\xi)$;*

(2) **consistent** *relative to ξ, if $N_{\mathcal{A}}(\alpha) \subseteq C_{\mathcal{A}}(\xi)$.*

Inconsistent SCL-terms are also called **contradictory**. Note that there are possibly terms $\alpha \in \mathcal{C}_{\mathcal{A}}$ that are neither inconsistent nor consistent relative to ξ. The term representations of $I_{\mathcal{A}}(\xi)$ and $C_{\mathcal{A}}(\xi)$ are

$$I(\xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : N_{\mathcal{A}}(\alpha) \subseteq I_{\mathcal{A}}(\xi)\}, \tag{2.14}$$
$$C(\xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : N_{\mathcal{A}}(\alpha) \subseteq C_{\mathcal{A}}(\xi)\}, \tag{2.15}$$

respectively. The sets $\mu I(\xi)$ and $\mu C(\xi)$ are the corresponding minimal term representations. Some authors do call $I(\xi)$ the **contradiction** of ξ. Similarly, $\mu I(\xi)$ is called **minimal contradiction** of ξ.

The term representations of $I_{\mathcal{A}}(\xi)$ and $C_{\mathcal{A}}(\xi)$ can also be characterized without using the notions of inconsistent and consistent scenarios. If $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ is an argumentation system, then

$$I(\xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : \alpha \wedge \xi \models \bot\}, \tag{2.16}$$
$$C(\xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : \forall \alpha' \supseteq \alpha,\ \alpha' \in \mathcal{C}_{\mathcal{A}},\ \alpha' \wedge \xi \not\models \bot\}. \tag{2.17}$$

2.4.3 Supporting Arguments

The problem of representing sets of scenarios also appears in the case of quasi-supporting and supporting scenarios for a given hypothesis:

Definition 2.7 *Let $\mathcal{AS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A})$ be an argumentation system and let $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$ be a hypothesis. A SCL-term $\alpha \in \mathcal{C}_{\mathcal{A}}$ is called a*

(1) **quasi-supporting argument** *for h relative to ξ, if $N_{\mathcal{A}}(\alpha) \subseteq QS_{\mathcal{A}}(h, \xi)$;*

(2) **supporting argument** *for h relative to ξ, if $N_{\mathcal{A}}(\alpha) \subseteq SP_{\mathcal{A}}(h, \xi)$;*

(3) **possibly supporting argument** *for h relative to ξ, if $N_{\mathcal{A}}(\alpha) \subseteq PS_{\mathcal{A}}(h, \xi)$.*

The term representations of $QS_{\mathcal{A}}(h, \xi)$, $SP_{\mathcal{A}}(h, \xi)$, and $PS_{\mathcal{A}}(h, \xi)$, that is

$$QS(h, \xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : N_{\mathcal{A}}(\alpha) \subseteq QS_{\mathcal{A}}(h, \xi)\}, \tag{2.18}$$
$$SP(h, \xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : N_{\mathcal{A}}(\alpha) \subseteq SP_{\mathcal{A}}(h, \xi)\}, \tag{2.19}$$
$$PS(h, \xi) = \{\alpha \in \mathcal{C}_{\mathcal{A}} : N_{\mathcal{A}}(\alpha) \subseteq PS_{\mathcal{A}}(h, \xi)\}, \tag{2.20}$$

are called **quasi-support**, **support**, and **possibility** for h relative to ξ. Furthermore, the minimal term representations $\mu QS(h, \xi)$, $\mu SP(h, \xi)$, and $\mu PS(h, \xi)$ are called **minimal quasi-support**, **minimal support**, and **minimal possibility** for h relative to ξ.

Of course, the sets $QS(h, \xi)$, $SP(h, \xi)$, and $PS(h, \xi)$ can also be characterized without the notion of scenarios:

$$\begin{aligned} QS(h, \xi) &= \{\alpha \in \mathcal{C}_{\mathcal{A}} : \alpha \wedge \xi \models h\}, \\ SP(h, \xi) &= \{\alpha \in \mathcal{C}_{\mathcal{A}} : \alpha \wedge \xi \models h,\ \forall \alpha' \supseteq \alpha,\ \alpha' \in \mathcal{C}_{\mathcal{A}},\ \alpha' \wedge \xi \not\models \bot\}; \\ PS(h, \xi) &= \{\alpha \in \mathcal{C}_{\mathcal{A}} : \forall \alpha' \supseteq \alpha,\ \alpha' \in \mathcal{C}_{\mathcal{A}},\ \alpha' \wedge \xi \not\models \neg h\}. \end{aligned}$$

2.5 Probabilistic Argumentation Systems

So far, the problem of judging hypotheses has only been considered from a qualitative point of view. An additional judgment of hypotheses can be obtained if a probability distribution is associated to every assumption. If the frame of $a_i \in \mathcal{A}$ is Θ_{a_i} and $\theta_{ij} \in \Theta_{a_i}$, let π_{ij} denote the probability that the true value of a_i is θ_{ij}. As we have imposed that the true value of $a_i \in \mathcal{A}$ must be in Θ_{a_i}, these probabilities have to sum to 1, that is, $\sum_j \pi_{ij} = 1$. The probability distribution π_i can be understand as an estimation that expresses on a scale between 0 and 1 the subjective belief about the true value of a_i. We assume the probabilities π_i to be stochastically independent.

Assigning probabilities to assumptions induces a probabilistic structure upon argumentation systems. Therefore, it is reasonable to define probabilistic argumentation systems:

Definition 2.8 *Let $\mathcal{V}$ and $\mathcal{A}$ be two disjoint sets of variables, ξ a SCL-formula in $\mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$ and $\Pi = \{\pi_1, \ldots, \pi_m\}$ a set of probability distributions on the elements of $\mathcal{A} = \{a_1, \ldots, a_m\}$. Then, the quadruple $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ is called* **probabilistic argumentation system**.

Again, the important special case of propositional argumentation systems can be treated specially. This leads to the quadruple $\mathcal{PAS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A}, \Pi)$, where $\mathcal{P}$ and $\mathcal{A}$ are two disjoint sets of propositions, $\xi \in \mathcal{L}_{\mathcal{A}\cup\mathcal{P}}$ and $\Pi = \{\pi_1, \ldots, \pi_m\}$. The probability distribution π_i associated to $a_i \in \mathcal{A}$ is then given by the number $p(a_i)$, where $0 \leq p(a_i) \leq 1$. This number expresses the subjective belief of a_i being true. The corresponding subjective belief of a_i being false is then implicitly given by the value $1 - p(a_i)$.

2.6 Numerical Arguments

Suppose that a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ is given. As we assume that the probabilities which are assigned to the assumptions are stochastically independent, the probability of a particular scenario $\mathbf{s} = (\theta_{1j}, \ldots, \theta_{mj})$ in $N_{\mathcal{A}}$ is given by

$$p(\mathbf{s}) \;=\; \prod_{i=1}^{m} p(a_i = \theta_{ij}) \;=\; \prod_{i=1}^{m} \pi_{ij}. \tag{2.21}$$

For an arbitrary set of scenarios $S \subseteq N_{\mathcal{A}}$, the probability of S is obtained by summing up the probabilities of the elements of S:

$$p(S) \;=\; \sum_{\mathbf{s}\in S} p(\mathbf{s}). \tag{2.22}$$

In the following, we will be particularly interested in the set of quasi-supporting scenarios $QS_{\mathcal{A}}(h, \xi)$, the set of supporting scenarios $SP_{\mathcal{A}}(h, \xi)$, and the set of possibly supporting scenarios $PS_{\mathcal{A}}(h, \xi)$.

2.6.1 Degree of Quasi-Support

Given a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ and a hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$, the expression

$$dqs(h, \xi) \;=\; p(QS_{\mathcal{A}}(h, \xi)) \tag{2.23}$$

is called **degree of quasi-support** of h relative to ξ. We will show in Chapter 4 that degree of quasi-support corresponds to the notion of **unnormalized belief** in Dempster-Shafer theory (Shafer, 1976).

2.6.2 Degree of Support

For a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$, there is exactly one (unknown) interpretation in $N_{\mathcal{A}}$ that represents the true interpretation. Therefore, if the set of inconsistent scenarios $I_{\mathcal{A}}(\xi)$ is not empty, the prior probability distribution on $N_{\mathcal{A}}$ has to be conditioned on the fact that the true scenario must be in $C_{\mathcal{A}}(\xi)$. Thus, for the knowledge base ξ and a hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$, the expression

$$dsp(h,\xi) \;=\; p(SP_{\mathcal{A}}(h,\xi)|C_{\mathcal{A}}(\xi)) \tag{2.24}$$

is called **degree of support** of h relative to ξ. It is computed as follows:

$$\begin{aligned} dsp(h,\xi) \;&=\; \frac{p(SP_{\mathcal{A}}(h,\xi))}{p(C_{\mathcal{A}}(\xi))} \;=\; \frac{p(QS_{\mathcal{A}}(h,\xi)) - p(QS_{\mathcal{A}}(\bot,\xi))}{1 - p(QS_{\mathcal{A}}(\bot,\xi))} \\ &=\; \frac{dqs(h,\xi) - dqs(\bot,\xi)}{1 - dqs(\bot,\xi)} \end{aligned} \tag{2.25}$$

Therefore, the value of $dsp(h,\xi)$ is completely determined by the two values $dqs(h,\xi)$ and $dqs(\bot,\xi)$. For the special cases, where $h \equiv \bot$ and $h \equiv \top$, we have the following properties:

$$dsp(\bot,\xi) \;=\; 0 \tag{2.26}$$

$$dsp(\top,\xi) \;=\; 1 \tag{2.27}$$

We will show in Chapter 4 that degree of support corresponds to **normalized belief** in Dempster-Shafer theory (Shafer, 1976). An important property of $dsp(h,\xi)$ is that it behaves non-monotonically when new knowledge is added to the knowledge base ξ.

2.6.3 Degree of Possibility

For degree of possibility, the prior probability distribution on $N_{\mathcal{A}}$ is also conditioned by the fact that the true scenario must be in $C_{\mathcal{A}}(\xi)$. Therefore, for a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ and a hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$, the expression

$$dps(h,\xi) \;=\; p(PS_{\mathcal{A}}(h,\xi)|C_{\mathcal{A}}(\xi)) \tag{2.28}$$

is called **degree of possibility** of h relative to ξ. It is computed as follows:

$$\begin{aligned} dps(h,\xi) \;&=\; \frac{p(PS_{\mathcal{A}}(h,\xi))}{p(C_{\mathcal{A}}(\xi))} \;=\; \frac{1 - p(QS_{\mathcal{A}}(\neg h,\xi))}{1 - p(QS_{\mathcal{A}}(\bot,\xi))} \\ &=\; \frac{1 - dqs(\neg h,\xi)}{1 - dqs(\bot,\xi)} \;=\; 1 - dsp(\neg h,\xi) \end{aligned} \tag{2.29}$$

Therefore, $dps(h,\xi)$ can be obtained by computing $dsp(\neg h,\xi)$. For the special cases, where $h \equiv \bot$ and $h \equiv \top$, we have the following properties:

$$dps(\bot,\xi) \;=\; 0 \tag{2.30}$$

$$dps(\top,\xi) \;=\; 1 \tag{2.31}$$

Finally, an important property follows the fact that $SP_{\mathcal{A}}(h,\xi)$ is always a subset of $PS_{\mathcal{A}}(h,\xi)$:

$$dsp(h,\xi) \leq dps(h,\xi) \tag{2.32}$$

Note that the degree of possibility corresponds to the notion of **plausibility** in Dempster-Shafer theory (Shafer, 1976).

2.7 From Symbolic to Numerical Arguments

Suppose that a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ is given. The problem of computing $dsp(h,\xi)$ and/or $dps(h,\xi)$ for a hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$ relative to ξ involves three steps:

(1) compute $dqs(h,\xi)$, respectively $dqs(\neg h,\xi)$;
(2) compute $dqs(\bot,\xi)$;
(3) apply Equation (2.25), respectively Equation (2.29).

The problem to be solved is therefore the computation of $dqs(h,\xi)$ for arbitrary hypotheses $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. For this reason, suppose that the corresponding set of quasi-supporting arguments $QS_{\mathcal{A}}(h,\xi)$ is represented by the set $\mu QS(h,\xi) = \{\alpha_1,\ldots,\alpha_q\}$. Clearly, this set of minimal quasi-supporting arguments defines the DNF $\alpha_1 \vee \cdots \vee \alpha_q$ with

$$QS_{\mathcal{A}}(h,\xi) = N_{\mathcal{A}}(\alpha_1 \vee \cdots \vee \alpha_q) = N_{\mathcal{A}}(\alpha_1) \cup \cdots \cup N_{\mathcal{A}}(\alpha_q). \tag{2.33}$$

The probability $p(QS_{\mathcal{A}}(h,\xi))$ can therefore be seen as a probability of a union of events. This is a classical problem of probability theory and there are several approaches to this problem:

2.7.1 The Inclusion-Exclusion Method

A first and simple approach is given by the so-called **inclusion-exclusion** method (Feller, 1968). In order to illustrate this method, suppose that $q = 2$, thus $\mu QS(h,\xi) = \{\alpha_1,\alpha_2\}$. Then,

$$\begin{aligned} dqs(h,\xi) &= p(N_{\mathcal{A}}(\alpha_1) \cup N_{\mathcal{A}}(\alpha_2)) \\ &= p(N_{\mathcal{A}}(\alpha_1)) + p(N_{\mathcal{A}}(\alpha_2)) - p(N_{\mathcal{A}}(\alpha_1) \cap N_{\mathcal{A}}(\alpha_2)) \\ &= p(\alpha_1) + p(\alpha_2) - p(\alpha_1 \wedge \alpha_2). \end{aligned}$$

The computation of $p(\alpha_1)$, $p(\alpha_2)$, and $p(\alpha_1 \wedge \alpha_2)$ is not difficult and as a consequence, $dqs(h,\xi)$ can be computed easily.

For the general case, suppose that P_k is defined for $1 \leq k \leq q$ as

$$P_k = \sum_{\substack{I \subseteq \{1,\ldots,q\}, \\ 1 \leq |I| \leq k}} (-1)^{|I|+1} \cdot p(\bigcap_{i\in I} N_{\mathcal{A}}(\alpha_i)). \tag{2.34}$$

Then

$$P_2 \leq P_4 \leq \cdots \leq \quad p(N_{\mathcal{A}}(\alpha_1) \cup \cdots \cup N_{\mathcal{A}}(\alpha_q)) \quad \leq \cdots \leq P_3 \leq P_1. \tag{2.35}$$

In addition,

$$p(N_{\mathcal{A}}(\alpha_1) \cup \cdots \cup N_{\mathcal{A}}(\alpha_q)) \quad = \quad P_q. \tag{2.36}$$

However, the computation of P_q involves a summation over $(2^q - 1)$ terms. Therefore, the computational effort needed can quickly become prohibitive.

2.7.2 The Method of Abraham

An alternative method consists in transforming the DNF $\alpha_1 \vee \cdots \vee \alpha_q$ into an equivalent disjunction $\gamma_1 \vee \ldots \vee \gamma_r$ with mutually disjoint formulas $\gamma_i \in \mathcal{L}_{\mathcal{A}}$, that is, $N_{\mathcal{A}}(\gamma_i) \cap N_{\mathcal{A}}(\gamma_j) = \emptyset$ whenever $i \neq j$. The probability $p(QS_{\mathcal{A}}(h, \xi))$ is then simply the sum of the probabilities of the individual formulas γ_i:

$$p(QS_{\mathcal{A}}(h, \xi)) \;=\; p(N_{\mathcal{A}}(\gamma_1) \cup \cdots \cup N_{\mathcal{A}}(\gamma_r)) \;=\; \sum_{i=1}^{r} p(N_{\mathcal{A}}(\gamma_i)). \tag{2.37}$$

The number of terms in such a sum is often much smaller than the number of terms in Equation (2.36). However, the problem of computing such a disjoint representation of $QS_{\mathcal{A}}(h, \xi)$ remains. In addition, the disjoint form should be so that $p(N_{\mathcal{A}}(\gamma_i))$ can be computed easily.

Several methods for this problem have been developed especially in reliability theory. A simple method is presented in (Abraham, 1979) for monotone boolean functions. This method has been generalized in (Kohlas & Monney, 1995) for propositional logic and in (Monney & Anrig, 2000) for set constraint logic. The idea is that the new disjunction $\gamma_1 \vee \ldots \vee \gamma_r$ consists of disjoint terms $\gamma_i \in \mathcal{C}_{\mathcal{A}}$.

In the case of a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$, where the set of assumptions $\mathcal{A}$ consists of propositions only, the probability of a term $\gamma_i = l_1 \wedge \cdots \wedge l_m$ can easily be computed as

$$p(N_{\mathcal{A}}(\gamma_i)) \quad = \quad \prod_{l_k = a_k} p(l_k) \cdot \prod_{l_k = \neg a_k} (1 - p(l_k)). \tag{2.38}$$

In the case, where the set of assumptions $\mathcal{A}$ consists of SCL-variables, the probability of a SCL-term $\gamma_i = X_1(a_1) \wedge \cdots \wedge X_m(a_m)$, where $X_k \subseteq \Theta_{a_k}$, is computed as

$$p(N_{\mathcal{A}}(\gamma_i)) \quad = \quad \prod_{k=1}^{m} \left(\sum_{\theta_{kj} \in X_k} p(a_k = \theta_{kj}) \right). \tag{2.39}$$

However, unfortunately, this method still tends to generate a relatively large number of disjoint terms $\gamma_1 \vee \ldots \vee \gamma_r$.

2.7.3 The Method of Heidtmann

Heidtmann (Heidtmann, 1989) proposed a much better but more complex method for propositional logic. In the case of a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$, where the set of assumptions $\mathcal{A}$ consists of propositions only, every γ_i is represented as a conjunction of the form

$$\gamma_i \quad = \quad \delta_1 \wedge \neg\delta_2 \wedge \cdots \wedge \neg\delta_s$$

so that these factors are independent and each $\delta_j \in C_{\mathcal{A}}$. The probability $p(\gamma_i)$ can then be computed very easily as

$$p(N_{\mathcal{A}}(\gamma_i)) \quad = \quad p(\delta_1) \cdot \prod_{k=2}^{s} (1 - p(\delta_k)). \tag{2.40}$$

The weakness of Heidtmann's method is its restriction to monotone boolean formulas. The generalization of Heidtmann's method to non-monotone boolean formulas can be found in (Bertschy & Monney, 1996). Furthermore, the generalization to set constraint logic is given in (Monney & Anrig, 2000). In any case, Heidtmann's method is often much more efficient than the method of Abraham. In addition, it usually gives a significantly smaller number of terms.

2.8 The "Communication Line" Example

It was shown that numerical arguments can be obtained by first computing symbolic arguments and then using one of the methods described previously. However, this approach is not always feasible because the set of symbolic arguments is sometimes much too big and cannot be represented explicitly. The following simple example will show this.

Example 2.4 Communication Line Suppose that $m + 1$ computers are connected using m connections as shown in Figure 2.2 for the case $m = 3$. When the network was set up, two wires of different quality were used for each connection between two computers. Our main point of interest is whether or not a mail which is sent from the first computer on the left reaches the last computer on the right.

Figure 2.2: A Communication Line with four Computers.

To answer this question, we construct a propositional argumentation system $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$, where the set of propositions $\mathcal{P} = \{x_0, \ldots, x_m\}$ and the set

of assumptions $\mathcal{A} = \{a_1, \ldots, b_m, b_1, \ldots, b_m\}$ are used. Computers are numbered from left to right, starting with the number 0. The meaning of a proposition $x_i \in \mathcal{P}$ is that x_i is true whenever the corresponding computer has received the mail. Accordingly, the assumptions $a_i, b_i \in \mathcal{A}$ represent the two wires which connect the computer $i-1$ to the computer i. The knowledge base ξ consists of $2m$ rules of the form

$$\begin{aligned} (x_{i-1} \wedge a_i) &\rightarrow x_i \\ (x_{i-1} \wedge b_i) &\rightarrow x_i \end{aligned}$$

where $1 \leq i \leq m$. The meaning of a rule $(x_{i-1} \wedge a_i) \rightarrow x_i$ is the following: if computer $i-1$ has received the mail and the wire represented by a_i is working correctly, then computer i receives the mail, too. This situation is visualized in Figure 2.3, where $x_0, \ldots, x_m$ are represented by points and each rule is represented by an arrow labeled with the corresponding assumption.

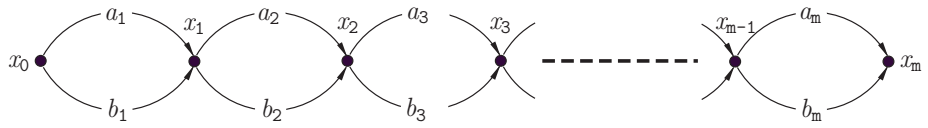

Figure 2.3: Graph for the Communication Line.

It may now be interesting to compute the degree of support that a mail which is send from the first computer on the left receives the last computer on the right. This corresponds to computing $dsp(x_0 \rightarrow x_m, \xi)$.

In order to compute $dsp(x_0 \rightarrow x_m, \xi)$, we have to compute the values $dqs(x_0 \rightarrow x_m, \xi)$ and $dqs(\bot, \xi)$. The corresponding sets of quasi-supporting arguments $QS(x_0 \rightarrow x_m, \xi)$ and $QS(\bot, \xi)$ are given as follows:

$$\begin{aligned} QS(x_0 \rightarrow x_m, \xi) &= \{(a_1 a_2 \ldots a_m), (a_1 a_2 \ldots b_m), \ldots, (b_1 b_2 \ldots b_m)\} \\ QS(\bot, \xi) &= \emptyset \end{aligned}$$

Each argument in $QS(x_0 \rightarrow x_m, \xi)$ corresponds to a possible way to establish a connection between x_0 and x_m (see Figure 2.3). Unfortunately, there are 2^m different ways to establish such a connection. Therefore, it is not feasible to compute $QS(x_0 \rightarrow x_m, \xi)$ explicitly if m is too big. The computation of $dsp(x_0 \rightarrow x_m, \xi)$ then fails for that reason. $\ominus$

The above rather small example shows that computing degree of support by first computing symbolic arguments is often not feasible. The following chapters will show how numerical arguments can be obtained without that symbolic arguments have to be computed first.

3
Dempster-Shafer Theory

The foundations of Dempster-Shafer theory were laid in (Dempster, 1967; Dempster, 1968), where Dempster studied from a purely mathematical point of view upper and lower bounds of probability distributions induced by a multivalued mapping. Shafer continued the work of Dempster and developped in (Shafer, 1976) a theory of evidence, where inferences are drawn from various sources of evidence. He proposed to call set functions having the structure of Dempster's lower probabilities **belief functions** and determined most of the terminology used now. Unfortunately, people occasionally confuse Dempster's upper and lower probabilities with belief functions or use belief functions blindly which can lead to contradictory results (Pearl, 1990; Smets, 1992). More information on Dempster-Shafer theory can be found in (Kong, 1986; Smets, 1988; Thoma, 1991; Kohlas & Monney, 1995).

Nowadays, Dempster-Shafer theory is widely used to represent uncertain knowledge. A piece of evidence can be encoded by a Dempster-Shafer belief function. Given several pieces of evidence encoded by belief functions, the problem to solve is to combine these functions and to compute the strength of belief in one or more given hypotheses.

In this chapter, we will introduce **multivariate Dempster-Shafer belief functions**. First, we will take a look at the different representations which are **mass function**, **belief function**, and **commonality function**. Then, the basic operations **combination**, **marginalization**, **extension**, and **division** will be discussed. Because the efficiency of these operations depends heavily on the representation used, it can be worthwhile to perform a transformation and change the representations. After the discussion of such transformations we will finally look at how Dempster-Shafer belief functions can be stored efficiently.

3.1 Basic Definitions

When we are modeling aspects of the real world we often deal with multivariate situations, where the state space is a product space. Therefore, multivariate

Dempster-Shafer belief functions often turn out to be well suited for the modeling of real world problems.

Variables and Configurations. We define Θ_x as the *state space* of a variable x, i.e. the set of values of x. It is assumed that all variables have finite state spaces. Upper-case italic letters such as D,E,F,... denote sets of variables. Given a set D of variables, let Θ_D denote the Cartesian product $\Theta_D = \times\{\Theta_x : x \in D\}$. Θ_D is called *state space* for D. The elements of Θ_D are *configurations* of D. Upper-case italic letters from the beginning of the alphabet such as A,B,... are used to denote sets of configurations.

Projection of Sets of Configurations. If D and D' are sets of variables, $D' \subseteq D$ and $\mathbf{x}$ is a configuration of D, then $\mathbf{x}^{\downarrow D'}$ denotes the *projection* of $\mathbf{x}$ to D'. If A is a subset of Θ_D, then the projection of A to D', denoted as $A^{\downarrow D'}$, is obtained by projecting each element of A to D', i.e. $A^{\downarrow D'} = \{\mathbf{x}^{\downarrow D'} : \mathbf{x} \in A\}$. Note that $A^{\downarrow D'}$ is a subset of $\Theta_{D'}$.

Extension of Sets of Configurations. If D and D' are sets of variables, $D' \subseteq D$, and B is a subset of $\Theta_{D'}$, then $B^{\uparrow D}$ denotes the *extension* of B to D, i.e. $B^{\uparrow D} = B \times \Theta_{D \setminus D'}$. Note that $B^{\uparrow D}$ is a subset of Θ_D.

Projection and extension of sets of configurations are very important in Dempster-Shafer theory. Therefore, let's look at the following example:

Example 3.1 Projection and Extension of Sets of Configurations Suppose that the set of variables $D = \{x, y, z\}$ is given and suppose in addition that $\Theta_x = \{\mathbf{x}_1, \mathbf{x}_2\}$, $\Theta_y = \{\mathbf{y}_1, \mathbf{y}_2\}$, and $\Theta_z = \{\mathbf{z}_1, \mathbf{z}_2\}$. Then, the state space of D consists of 8 configurations and is given by

$$\Theta_D = \{(\mathbf{x}_1\mathbf{y}_1\mathbf{z}_1), (\mathbf{x}_1\mathbf{y}_1\mathbf{z}_2), (\mathbf{x}_1\mathbf{y}_2\mathbf{z}_1), \ldots, (\mathbf{x}_2\mathbf{y}_2\mathbf{z}_2)\}.$$

There are 256 different sets of configurations which could be build. Let's consider here the set of configurations A given by

$$A = \{(\mathbf{x}_1\mathbf{y}_2\mathbf{z}_2), (\mathbf{x}_2\mathbf{y}_1\mathbf{z}_1), (\mathbf{x}_2\mathbf{y}_2\mathbf{z}_1), (\mathbf{x}_2\mathbf{y}_2\mathbf{z}_2)\}.$$

On the left side of Figure 3.1 the set A is represented in a 3-dimensional cube of which the axes are given by the three variables x, y, and z. For example, the point on the x-axis corresponds to the configuration $\mathbf{x}_2\mathbf{y}_1\mathbf{z}_1$. The projection of A to the set of variables $D' = \{x, y\}$ is denoted as $A^{\downarrow D'}$ and is given by

$$A^{\downarrow D'} = \{(\mathbf{x}_1\mathbf{y}_2), (\mathbf{x}_2\mathbf{y}_1), (\mathbf{x}_2\mathbf{y}_2)\}.$$

In the middle of Figure 3.1 it can be seen that $A^{\downarrow D'}$ is really a subset of $\Theta_{D'}$. If B denotes this set of configurations, that is $B = A^{\downarrow D'}$, then the extension of B to D denoted as $B^{\uparrow D}$ is given by

$$B^{\uparrow D} = \{(\mathbf{x}_1\mathbf{y}_2\mathbf{z}_1), (\mathbf{x}_1\mathbf{y}_2\mathbf{z}_2), (\mathbf{x}_2\mathbf{y}_1\mathbf{z}_1), (\mathbf{x}_2\mathbf{y}_1\mathbf{z}_2), (\mathbf{x}_2\mathbf{y}_2\mathbf{z}_1), (\mathbf{x}_2\mathbf{y}_2\mathbf{z}_2)\}.$$

On the right side of Figure 3.1 it is shown that $B^{\uparrow D}$ is a subset of Θ_D and that it is obtained by a cylindrical extension of B.

$\ominus$

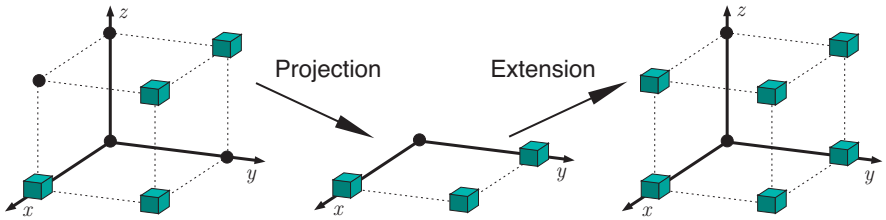

Figure 3.1: Projection and Extension of a Set of Configurations.

3.2 Different Representations

Similar to complex numbers, where $c \in \mathbb{C}$ can be represented in polar or rectangular form, there are also different ways to represent the information contained in a Dempster-Shafer belief function. It can be represented as a *mass function*, as a *belief function*, or as a *commonality function*. The unusual notation $[\varphi]_m$, $[\varphi]_b$, and $[\varphi]_q$ is used instead of m_φ, bel_φ, and q_φ, because it is more convenient. The new notation has among others the following advantages:

- it focuses on the *information* contained in φ, whereas the usual notation focuses on the *representation* of φ.
- it allows to distinguish easily between normalized and unnormalized mass, belief, or commonality functions.

When a set of configurations A is given and we want to refer to the mass, belief or commonality of A, then we write simply $[\varphi(A)]_m$, $[\varphi(A)]_b$, and $[\varphi(A)]_q$. In accordance with Shafer (Shafer, 1991) we speak of *potentials* when no representation is specified. We then write only φ, but without enclosing brackets.

3.2.1 Mass Function

A *mass function* $[\varphi]_m$ on D assigns to every subset A of Θ_D a value in $[0, 1]$, that is $[\varphi]_m : 2^{\Theta_D} \to [0, 1]$. The following condition must be satisfied:

$$\sum_{A \subseteq \Theta_D} [\varphi(A)]_m \quad = \quad 1. \tag{3.1}$$

Intuitively, $[\varphi(A)]_m$ is the belief of evidence for A that has not already been assigned to some proper subset of A. Sometimes, a second condition, $[\varphi(\emptyset)]_m = 0$, is imposed. A mass function for which this additional condition holds is called normalized, otherwise it is called unnormalized.

Given a potential φ on D, D is called the *domain* of φ. The sets $A \subseteq \Theta_D$ for which $[\varphi(A)]_m \neq 0$ are called *focal sets*. We use $FS(\varphi)$ to denote the focal sets of φ. In addition, φ on domain D is called **neutral potential** for D if $[\varphi(\Theta_D)]_m = 1$.

3.2.2 Belief Function

A *belief function* $[\varphi]_b$ on D, $[\varphi]_b : 2^{\Theta_D} \rightarrow [0,1]$, can be obtained in terms of a mass function:

$$[\varphi(A)]_b = \sum_{B:B\subseteq A} [\varphi(B)]_m. \tag{3.2}$$

Again, if $[\varphi(\emptyset)]_b = 0$, then the belief function is called normalized. Note that a normalized mass function always leads to a normalized belief function.

3.2.3 Commonality Function

A *commonality function* $[\varphi]_q$ on D, $[\varphi]_q : 2^{\Theta_D} \rightarrow [0,1]$, can be defined in terms of a mass function:

$$[\varphi(A)]_q = \sum_{B:A\subseteq B} [\varphi(B)]_m. \tag{3.3}$$

It is always $[\varphi(\emptyset)]_q = 1$. Therefore, there is no easy way to see whether a commonality function is normalized or not.

3.3 Normalization

Unnormalized mass, belief, or commonality functions can always be normalized. An advantage of the notation introduced is its ability to distinguish easily between normalized and unnormalized mass, belief, or commonality functions. We write $[\varphi]_M$, $[\varphi]_B$, and $[\varphi]_Q$ when φ is normalized. The transformation is given as follows:

$$[\varphi(A)]_M = \begin{cases} 0 & \text{if } A = \emptyset, \\ \frac{[\varphi(A)]_m}{1-[\varphi(\emptyset)]_m} & \text{otherwise.} \end{cases} \tag{3.4}$$

$$[\varphi(A)]_B = \frac{[\varphi(A)]_b - [\varphi(\emptyset)]_b}{1 - [\varphi(\emptyset)]_b} \tag{3.5}$$

$$[\varphi(A)]_Q = \begin{cases} 1 & \text{if } A = \emptyset, \\ \frac{[\varphi(A)]_q}{1-[\varphi(\emptyset)]_m} & \text{otherwise.} \end{cases} \tag{3.6}$$

Note that above, it is supposed that the given information is not completely contradictory, that is $[\varphi(\emptyset)]_m \neq 1$. The normalizations are very similar since $[\varphi(\emptyset)]_b = [\varphi(\emptyset)]_m$. However, note that the normalization of a commonality function by Equation 3.6 in terms of commonality functions is much more difficult than the normalization of a mass or belief function. This is due to the fact that the computation of $[\varphi(\emptyset)]_m$ is not easy when a commonality function is given.

Example 3.2 Different Representations Suppose that the domain of the potential φ is $D = \{x\}$ and suppose in addition that the state space of x is

$\Theta_x = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$. Therefore, there are exactly 8 sets $A \subseteq \Theta_D$. Let's suppose now that the masses of $[\varphi]_m$ are as shown in Table 3.1. $[\varphi]_m$ is an unnormalized mass function because it is $[\varphi(\emptyset)]_m \neq 0$. The belief function $[\varphi]_b$ and the commonality function $[\varphi]_q$ are obtained by applying Equations 3.2 and 3.3. By applying Equations 3.4, 3.5, and 3.6, the normalized mass function, belief function and commonality function are obtained. The unnormalized functions $[\varphi]_m$, $[\varphi]_b$, and $[\varphi]_q$ as well as their normalized counterparts $[\varphi]_M$, $[\varphi]_B$, and $[\varphi]_Q$ are shown in Table 3.1.

	Unnormalized			Normalized		
$A \subseteq \Theta_D$	$[\varphi]_m$	$[\varphi]_b$	$[\varphi]_q$	$[\varphi]_M$	$[\varphi]_B$	$[\varphi]_Q$
$\emptyset$	0.2	0.2	1.0	0	0	1.000
$\{\mathbf{v}_1\}$	0	0.2	0.7	0	0	0.875
$\{\mathbf{v}_2\}$	0.1	0.3	0.5	0.125	0.125	0.625
$\{\mathbf{v}_1, \mathbf{v}_2\}$	0	0.3	0.4	0	0.125	0.500
$\{\mathbf{v}_3\}$	0	0.2	0.7	0	0	0.875
$\{\mathbf{v}_1, \mathbf{v}_3\}$	0.3	0.5	0.7	0.375	0.375	0.875
$\{\mathbf{v}_2, \mathbf{v}_3\}$	0	0.3	0.4	0	0.125	0.500
$\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$	0.4	1.0	0.4	0.500	1.000	0.500

Table 3.1: Normalization of potentials

$\ominus$

3.4 Basic Operations

The basic operations for potentials are *combination*, *marginalization*, and *extension*. Intuitively, these operations correspond to aggregation, focusing, and refinement, respectively. Another basic operation is *division*. It is used whenever some information has to be removed from a potential.

For every basic operation there are representations which are more appropriate than others and representations which cannot be used directly. For example, the belief function representation is more appropriate for the marginalization than the mass function representation, whereas there is no direct way to marginalize a commonality function.

3.4.1 Combination

Suppose φ and ψ are potentials on D_1 and D_2. The combination of these two potentials produces an (unnormalized) potential on domain $D = D_1 \cup D_2$:

$$[\varphi \otimes \psi(A)]_m = \sum \{[\varphi(B_1)]_m \cdot [\psi(B_2)]_m : B_1^{\uparrow D} \cap B_2^{\uparrow D} = A\}, \quad (3.7)$$

$$[\varphi \otimes \psi(A)]_q = [\varphi(A^{\downarrow D_1})]_q \cdot [\psi(A^{\downarrow D_2})]_q. \tag{3.8}$$

Note that there is no easy way to combine two belief functions.

3.4.2 Marginalization

Suppose φ_1 is a potential on D_1 and suppose $D_2 \subseteq D_1$. The marginalization of φ_1 to D_2 produces a potential on D_2:

$$[\varphi_1^{\downarrow D_2}(B)]_m = \sum_{A:A^{\downarrow D_2}=B} [\varphi_1(A)]_m, \tag{3.9}$$

$$[\varphi_1^{\downarrow D_2}(B)]_b = [\varphi_1(B^{\uparrow D_1})]_b. \tag{3.10}$$

Note that there is no easy way to marginalize a commonality function.

3.4.3 Extension

Suppose φ_2 is a potential on D_2 and suppose $D_2 \subseteq D_1$. The extension of φ_2 to D_1 produces a potential on D_1:

$$[\varphi_2^{\uparrow D_1}(A)]_m = \begin{cases} [\varphi_2(B)]_m & \text{if } A = B^{\uparrow D_1}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.11}$$

$$[\varphi_2^{\uparrow D_1}(A)]_q = [\varphi_2(A^{\downarrow D_2})]_q. \tag{3.12}$$

Note that there is no direct way to extent a belief function.

3.4.4 Division

Suppose ψ and φ are potentials on D_1 and D_2. The division of these two potentials produces a potential on $D = D_1 \cup D_2$ and is formally defined by Equation 3.13. Note that in general the result is not a valid potential, which means that some of the sets $A \subseteq \Theta_D$ can obtain negative masses. In (Thoma, 1991) the class of such functions is called *quasi-belief functions*.

$$\left[\frac{\psi}{\varphi}(A)\right]_q = \begin{cases} \frac{[\psi(A^{\downarrow D_1})]_q}{[\varphi(A^{\downarrow D_2})]_q} & \text{if } [\varphi(A^{\downarrow D_2})]_q \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.13}$$

Note that the division operation is only defined for commonality functions. There is no direct way to divide two mass functions or belief functions.

3.5 Transformations

Transforming a representation into another representation is necessary when the requested basic operation cannot be applied directly. A transformation may

also be valuable when there is another representation which allows to perform the requested basic operation more efficiently.

Given a potential φ on D in one of the three representations it is always possible to derive the others (Thoma, 1991):

$$[\varphi(A)]_m = \sum_{B:A\subseteq B} (-1)^{|B\setminus A|} \cdot [\varphi(B)]_q, \tag{3.14}$$

$$[\varphi(A)]_m = \sum_{B:B\subseteq A} (-1)^{|A\setminus B|} \cdot [\varphi(B)]_b, \tag{3.15}$$

$$[\varphi(A)]_q = \sum_{B:B\subseteq A} (-1)^{|B|} \cdot [\varphi(\overline{B})]_b, \tag{3.16}$$

$$[\varphi(A)]_b = \sum_{B:B\subseteq \overline{A}} (-1)^{|B|} \cdot [\varphi(B)]_q. \tag{3.17}$$

However, the transformations above are usually computationally very expensive. For example, suppose that φ has domain $D = \{x_1, \ldots, x_\ell\}$ and suppose that the state space of a variable x_i is given by $\Theta_{x_i} = \{\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_{k_i}}\}$. Θ_D then has exactly $k_1 k_2 \cdots k_\ell$ configurations. Let's denote the number of configurations of Θ_D by n, that is $n = k_1 k_2 \cdots k_\ell$. If $FS(\varphi)$ contains the maximum of 2^n focal sets, then there would be

$$\sum_{k=0}^{n} \binom{n}{k} \cdot 2^k = 3^n \tag{3.18}$$

terms to sum up for each transformation above. From Table 3.2 it can be seen that this would even for relatively small domains not be feasible, even if we suppose that all variables are binary, that is $\Theta_{x_i} = \{\mathbf{v}_{i_1}, \mathbf{v}_{i_2}\}$ for $1 \leq i \leq \ell$.

$\lvert D\rvert$	n	2^n	$n \cdot 2^n$	3^n
1	2	4	8	9
2	4	16	64	81
3	8	256	2,048	6,561
4	16	65,536	1,048,576	43,046,721
5	32	4,294,967,296	137,438,953,472	1,853,020,188,851,841

Table 3.2: Exponential Growth

3.5.1 Fast Moebius Transformation

An improvement is obtained when *fast Moebius transformations* (Kennes & Smets, 1990; Thoma, 1991; Xu & Kennes, 1994) are used instead. In the following, it is explained how a mass function can be transformed into a commonality function.

Suppose that a mass function $[\varphi]_m$ on domain $D = \{x_1, \ldots, x_\ell\}$ is given and that the state space of a variable x_i is $\Theta_{x_i} = \{\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_{k_i}}\}$. Then,

$$\Theta_D = \{(\mathbf{v}_{1_1}\mathbf{v}_{2_1}\ldots\mathbf{v}_{\ell_1}), (\mathbf{v}_{1_1}\mathbf{v}_{2_1}\ldots\mathbf{v}_{\ell_2}), \ldots, (\mathbf{v}_{1_{k_1}}\mathbf{v}_{2_{k_2}}\ldots\mathbf{v}_{\ell_{k_\ell}})\}$$

has exactly $n := k_1 k_2 \cdots k_\ell$ elements. The commonality function $[\varphi]_q$ is obtained from $[\varphi]_m$ by applying a sequence of n elementary steps. The i-th step will be associated with the i-th element of Θ_D and is given by

$$[\varphi_i(A)] = \begin{cases} [\varphi_{i-1}(A)] + [\varphi_{i-1}(A \cup \{\mathbf{s}_i\})] & \text{if } \mathbf{s}_i \notin A, \\ [\varphi_{i-1}(A)] & \text{otherwise} \end{cases} \tag{3.19}$$

where $\mathbf{s}_i$ is the i-th element of Θ_D. When we start with $[\varphi_0] = [\varphi]_m$ then a sequence $[\varphi_1], [\varphi_2], \ldots, [\varphi_n]$ is generated and finally it is $[\varphi]_q = [\varphi_n]$.

Example 3.3 Fast Moebius Transformation Suppose that the mass function $[\varphi]_m$ is given on domain $D = \{x\}$, where the state space of the variable x is $\Theta_x = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ and suppose in addition that the masses are as shown in Table 3.3. The application of the algorithm generates the intermediate results $[\varphi_1]$, $[\varphi_2]$, and $[\varphi_3]$, respectively. It can be verified that $[\varphi_3]$ is equal to the commonality function $[\varphi]_q$.

	$A \subseteq \Theta_D$	$[\varphi]_m$	$[\varphi_1]$	$[\varphi_2]$	$[\varphi_3]$	$[\varphi]_q$
0	$\emptyset$	0	0.1	0.3	1.0	1.0
1	$\{\mathbf{v}_1\}$	0.1	0.1	0.1	0.8	0.8
2	$\{\mathbf{v}_2\}$	0.2	0.2	0.2	0.6	0.6
3	$\{\mathbf{v}_1, \mathbf{v}_2\}$	0	0	0	0.4	0.4
4	$\{\mathbf{v}_3\}$	0	0.3	0.7	0.7	0.7
5	$\{\mathbf{v}_1, \mathbf{v}_3\}$	0.3	0.3	0.7	0.7	0.7
6	$\{\mathbf{v}_2, \mathbf{v}_3\}$	0	0.4	0.4	0.4	0.4
7	$\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$	0.4	0.4	0.4	0.4	0.4

Table 3.3: Fast Moebius Transformation

$\ominus$

A better understanding of the algorithm is obtained when every set $A \subseteq \Theta_D$ is represented by a point in the n-dimensional boolean cube $\mathcal{B}^n$, where the axes are the elements of Θ_D. In addition, for every set $A \subseteq \Theta_D$ the value $[\varphi(A)]_m$ is attached to the point which corresponds to A. At every step, only the values of half the points of $\mathcal{B}^n$ are changed. At step i, only those values are changed which are attached to the points corresponding to sets which do not contain the i-th element of Θ_D.

Example 3.4 Visualization of the Fast Moebius Transformation Let's look again at the previous example. It is $\Theta_D = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ and the boolean cube $\mathcal{B}^3$ is therefore spread by the three axes $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ respectively. The three steps of the algorithm are shown in Figure 3.2. At each step, only the values of points with an incoming arrow are changed. For example, in the first step, only the values of the points in the $(\mathbf{v}_2, \mathbf{v}_3)$-plane are changed. For example, the arrow from **5** to **4** means that the value attached to $\{\mathbf{v}_1, \mathbf{v}_3\}$ (= **5**) is added to the value attached to $\{\mathbf{v}_3\}$ (= **4**).

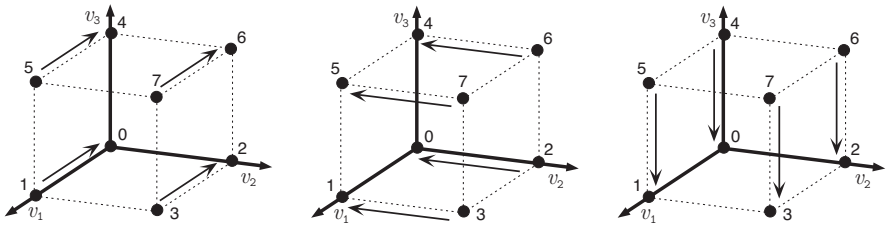

Figure 3.2: Visualization of Fast Moebius.

⊖

Using fast Moebius transformations, the number of terms to sum up in the worst case is reduced from 3^n to $n \cdot 2^n$. Nevertheless, for multivariate Dempster-Shafer belief function, n grows itself exponential and transformations are therefore not feasible in the worst case (see Table 3.2). Fortunately, practical experiences show that the case, where every set $A \subseteq \Theta_D$ is a focal set hardly ever occurs. Instead, very often it can be observed that a potential has only a few focal sets. The following section shows that this observation can be used to store potentials efficiently.

3.6 Storing Potentials

Suppose that a potential φ on domain D is given and suppose that n denotes the number of configurations of Θ_D. Then, there are 2^n different subsets $A \subseteq \Theta_D$. From Table 3.2 it can be seen that it would even for relatively small domains D not be feasible to store the values $[\varphi(A)]_m$ for every $A \subseteq \Theta_D$.

Fortunately, it is very often $[\varphi(A)]_m = 0$ for most of the sets $A \subseteq \Theta_D$. Therefore, it is sufficient to store the values $[\varphi(A)]_m$ only for the sets $A \in FS(\varphi)$. φ is completely determined when for each focal set A the value $[\varphi(A)]_m$ is given. In the same way, a belief function and a commonality function are completely determined whenever the values $[\varphi(A)]_b$, and $[\varphi(A)]_q$ are given for every $A \in FS(\varphi)$.

In the following, two algorithms are presented. The first algorithm transforms a mass function to a commonality function and the second algorithm does the

inverse transformation. Both algorithms take advantage of the sparse representation of mass functions and commonality functions.

Algorithm *mass function to commonality function*
Suppose φ is a potential with domain D and suppose in addition that the values $[\varphi(A_k)]_m$ are known for all $A_k \in FS(\varphi)$. For each of these focal sets the following formula has to be applied:

$$[\varphi(A_k)]_q \quad = \quad \sum_{B:A_k \subseteq B} \{[\varphi(B)]_m : B \in FS(\varphi)\} \tag{3.20}$$

Note that Equation 3.20 can also be used to calculate $[\varphi(A)]_q$ for $A \notin FS(\varphi)$.

Algorithm *commonality function to mass function*
Suppose φ is a potential with domain D and suppose in addition that the values $[\varphi(A_k)]_q$ are known for all $A_k \in FS(\varphi)$. First, an ordering $\{A_1, \ldots, A_n\}$ of the set $FS(\varphi)$ has to be found so that for each pair A_k,$A_l \in FS(\varphi)$, whenever $A_k \supseteq A_l$ we have $k \leq l$. Such an ordering can always be found. The following formula has then to be applied to this ordering:

$$[\varphi(A_k)]_m \quad = \quad [\varphi(A_k)]_q - \sum_{B:A_k \subset B} \{[\varphi(B)]_m : B \in FS(\varphi)\} \tag{3.21}$$

These two algorithms are very simple. They perform well if the set $FS(\varphi)$ is relatively small. In (Dugat & Sandri, 1994) another algorithm is presented which takes advantage of the partially ordered structure of $FS(\varphi)$, due to set-inclusion.

4

Computing Numerical Arguments

For a given probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ and a hypothesis $h \in \mathcal{L}_{\mathcal{V}}$, one possibility to compute $dsp(h, \xi)$ is to compute first the two sets of symbolic arguments $\mu QS(h, \xi)$ and $\mu QS(\bot, \xi)$. Then, the methods described in Section 2.7 can be used to compute $dqs(h, \xi)$ and $dqs(\bot, \xi)$ from which $dsp(h, \xi)$ can finally be derived. However, the example in Section 2.8 showed that it is not always feasible to compute $\mu QS(h, \xi)$ and $\mu QS(\bot, \xi)$.

A more efficient approach is based on the fact that degree of quasi-support corresponds to the notion of **unnormalized belief** in Dempster–Shafer theory (see Chapter 3). Therefore, the method presented in this chapter transforms the given probabilistic argumentation system $(\xi, \mathcal{V}, \mathcal{A}, \Pi)$ into a family of independent potentials $\vartheta_1, \ldots, \vartheta_m$. Then, $dsp(h, \xi)$ can be derived from the joint potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$. The following picture illustrates the two different ways to compute $dsp(h, \xi)$.

The method presented in this chapter constructs for a given probabilistic argumentation system $(\xi, \mathcal{V}, \mathcal{A}, \Pi)$ a potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ on domain $\mathcal{V}$ so that for all hypotheses $h \in \mathcal{L}_{\mathcal{V}}$ it holds that

$$dqs(h, \xi) \quad = \quad [\varphi(H)]_b$$

where $H = N_{\mathcal{V}}(h)$. However, φ is not formed explicitly because it could have too many focal sets. Instead, only its factors $\vartheta_1, \ldots, \vartheta_m$ are computed.

4.1 Constructing Independent Mass Functions

Let $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ be a probabilistic argumentation system. The problem to solve is to find a potential φ on domain $\mathcal{V}$ so that for all hypotheses $h \in \mathcal{L}_{\mathcal{V}}$ it holds that

$$dqs(h, \xi) = [\varphi(H)]_b$$

where $H = N_{\mathcal{V}}(h)$. For this purpose, we will apply the partition algorithm to ξ which yields a symbolic mass function. This symbolic mass function is a set of implications $\vec{\Sigma}''$ which allows finally to derive the potential φ.

4.1.1 Symbolic Mass Functions

First, the knowledge base ξ has to be transformed into an equivalent set of SCL-clauses $\Sigma = \{\xi_1, \ldots, \xi_r\}$. Every SCL-clause $\xi_i \in \Sigma$ can then be written as an implication of the form

$$\xi_i \;=\; \underbrace{l_1 \vee \cdots \vee l_k}_{\in \mathcal{A}} \vee \underbrace{l_{k+1} \vee \cdots \vee l_m}_{\in \mathcal{V}} \;\equiv\; \neg\alpha \vee \beta \;\equiv\; \alpha \to \beta, \tag{4.1}$$

where $\alpha \in \mathcal{C}_{\mathcal{A}}$ is a SCL-term composed of assumptions and $\beta \in \mathcal{D}_{\mathcal{V}}$ is a SCL-clause composed of variables. Therefore, let

$$\vec{\Sigma} \;=\; \{\alpha_1 \to \beta_1, \ldots, \alpha_r \to \beta_r\}$$

denote the corresponding set of implications obtained from Σ. Clearly, Σ and $\vec{\Sigma}$ are logically equivalent.

The Production Rule

Two distinct implications of $\vec{\Sigma}$ can always be replaced by three other implications so that the new set of implications is logically equivalent to $\vec{\Sigma}$. The general form of the production rule is therefore given as follows:

$$\left.\begin{array}{l} \alpha_i \to \beta_i \\ \alpha_j \to \beta_j \end{array}\right\} \;\Longrightarrow\; \left\{\begin{array}{l} (\alpha_i \wedge \neg\alpha_j) \to \beta_i \\ (\alpha_i \wedge \alpha_j) \to (\beta_i \wedge \beta_j). \\ (\neg\alpha_i \wedge \alpha_j) \to \beta_j \end{array}\right. \tag{4.2}$$

Above all, note that $N_{\mathcal{A}}(\alpha_i \wedge \neg\alpha_j)$, $N_{\mathcal{A}}(\alpha_i \wedge \alpha_j)$, and $N_{\mathcal{A}}(\neg\alpha_i \wedge \alpha_j)$ are mutually disjoint. In order to prevent the creation of formulas of the form $\bot \to \beta$, the above production rule is specialized depending on α_i and α_j:

- if $N_{\mathcal{A}}(\alpha_i) \cap N_{\mathcal{A}}(\alpha_j) = \emptyset$:

$$\left.\begin{array}{l} \alpha_i \to \beta_i \\ \alpha_j \to \beta_j \end{array}\right\} \;\Longrightarrow\; \left\{\begin{array}{l} \alpha_i \to \beta_i \\ \alpha_j \to \beta_j \end{array}\right. \tag{4.3}$$

- if $N_{\mathcal{A}}(\alpha_i) = N_{\mathcal{A}}(\alpha_j)$:

$$\left.\begin{array}{l}\alpha_i \to \beta_i \\ \alpha_j \to \beta_j\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}\alpha_i \to (\beta_i \wedge \beta_j)\end{array}\right. \tag{4.4}$$

- if $N_{\mathcal{A}}(\alpha_i) \subseteq N_{\mathcal{A}}(\alpha_j)$:

$$\left.\begin{array}{l}\alpha_i \to \beta_i \\ \alpha_j \to \beta_j\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}\alpha_i \to (\beta_i \wedge \beta_j) \\ (\neg\alpha_i \wedge \alpha_j) \to \beta_j\end{array}\right. \tag{4.5}$$

- if $N_{\mathcal{A}}(\alpha_i) \supseteq N_{\mathcal{A}}(\alpha_j)$:

$$\left.\begin{array}{l}\alpha_i \to \beta_i \\ \alpha_j \to \beta_j\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}\alpha_j \to (\beta_i \wedge \beta_j) \\ (\alpha_i \wedge \neg\alpha_j) \to \beta_i\end{array}\right. \tag{4.6}$$

The first of the four cases above shows that the application of the production rule makes only sense if $N_{\mathcal{A}}(\alpha_i)$ and $N_{\mathcal{A}}(\alpha_j)$ are not already mutually disjoint.

The Simplification Rule

The objective of the following simplification rule is to simplify implications which have the same conclusion. Two such implications can always be simplified as follows:

$$\left.\begin{array}{l}\alpha_i \to \beta \\ \alpha_j \to \beta\end{array}\right\} \Longrightarrow \quad (\alpha_i \vee \alpha_j) \to \beta. \tag{4.7}$$

Two implications with the same conclusion are therefore replaced by only one implication. The new set of implications obtained is logically equivalent to $\vec{\Sigma}$.

Constructing the Symbolic Mass Function

By repeatetly using the production rule until the conditions of all implications are mutually disjoint, a new set of implications $\{\alpha'_1 \to \beta'_1, \ldots, \alpha'_s \to \beta'_s\}$ is created. If an additional implication $\alpha'_0 \to \beta'_0$ of the form $\neg(\vee_{i=1}^{t}\alpha'_i) \to \top$ is added, the resulting set of implications $\vec{\Sigma}'$ given by

$$\vec{\Sigma}' \quad = \quad \{\alpha'_0 \to \beta'_0, \ldots, \alpha'_s \to \beta'_s\}$$

then represents a full decomposition of $N_{\mathcal{A}}$. This means that for each scenario $\mathbf{s} \in N_{\mathcal{A}}$, there is an implication $\alpha'_k \to \beta'_k \in \vec{\Sigma}'$ so that $\mathbf{s} \in N_{\mathcal{A}}(\alpha'_k)$. The set of implications $\vec{\Sigma}'$ satisfies the following three conditions:

(1) $N_{\mathcal{A}}(\alpha'_i) \cap N_{\mathcal{A}}(\alpha'_j) = \emptyset$ for all $i \neq j$,
(2) $\bigcup_{i=0}^{s} N_{\mathcal{A}}(\alpha'_i) = N_{\mathcal{A}}$,
(3) $N_{\mathcal{A}}(\alpha'_i) \neq \emptyset$ for $0 \leq i \leq s$.

It follows from (1) and (2) that $\vec{\Sigma}'$ defines a **partition** of $N_{\mathcal{A}}$, whereas (3) reveals that there are no implications of the form $\bot \rightarrow \beta$. However, there might be implications which have the same conclusion. By repeatetly using the simplification rule until there are no implications which have logical equivalent conclusions, the set of implications

$$\vec{\Sigma}'' \quad = \quad \{\alpha_0'' \rightarrow \beta_0'', \ldots, \alpha_s'' \rightarrow \beta_s''\}$$

obtained satisfies the following additional fourth condition:

(4) $N_{\mathcal{V}}(\beta_i'') \neq N_{\mathcal{V}}(\beta_j'')$ for all $i \neq j$.

We call the set of implications $\vec{\Sigma}''$ a **symbolic mass function**. In the next subsection, we will see that it is easy to derive a mass function from $\vec{\Sigma}''$.

The pseudo code of the partition algorithm is given below. The input of the partition algorithm is the knowledge base ξ and the output is the set of implications $\vec{\Sigma}'' = \{\alpha_0'' \rightarrow \beta_0'', \ldots, \alpha_s'' \rightarrow \beta_s''\}$.

Algorithm *The Partition Algorithm*

Transform ξ **into** $\Sigma = \{\xi_1, \ldots, \xi_r\}$, $\xi_i \in \mathcal{D}_{\mathcal{A}\cup\mathcal{V}}$
Transform Σ **into** $\vec{\Sigma} = \{\alpha_1 \rightarrow \beta_1, \ldots, \alpha_r \rightarrow \beta_r\}$
$\vec{\Sigma}' = \{\top \rightarrow \top\}$
For each $\alpha_i \rightarrow \beta_i$ in $\vec{\Sigma}$ **do**
  $\Gamma = \emptyset$
  For each $\alpha_j \rightarrow \beta_j$ in $\vec{\Sigma}'$ **do**
    If $N_{\mathcal{A}}(\alpha_i) \cap N_{\mathcal{A}}(\alpha_j) \neq 0$ **then**
      If $N_{\mathcal{A}}(\alpha_i) = N_{\mathcal{A}}(\alpha_j)$ **or** $N_{\mathcal{A}}(\alpha_i) \supseteq N_{\mathcal{A}}(\alpha_j)$ **then**
        $\Gamma = \Gamma \cup \{\alpha_j \rightarrow (\beta_i \wedge \beta_j)\}$
      else
        $\Gamma = \Gamma \cup \{(\alpha_j \wedge \neg\alpha_i) \rightarrow \beta_j, (\alpha_j \wedge \alpha_i) \rightarrow (\beta_i \wedge \beta_j)\}$
      EndIf
    EndIf
  Next $\alpha_j \rightarrow \beta_j$
  $\vec{\Sigma}' = \Gamma$
Next $\alpha_i \rightarrow \beta_i$
$\vec{\Sigma}'' = \mathit{Simplify}(\vec{\Sigma}')$

The purpose of *Simplify* is to perform all possible simplifications. In the worst case, $\vec{\Sigma}''$ consists of 2^r implications. Therefore, the partition algorithm can only be applied if the set of clauses Σ obtained from ξ is relatively small.

Example 4.1 Partition Algorithm Suppose that a propositional argumenation system $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$ consists of $\mathcal{P} = \{x, y\}$, $\mathcal{A} = \{a, b\}$ and that the knowledge base ξ is given by the set of clauses $\Sigma = \{\xi_1, \xi_2, \xi_3, \xi_4\}$ as follows:

$\xi_1 = \neg b \vee x$
$\xi_2 = \neg a \vee y$
$\xi_3 = \neg a \vee b \vee \neg x \vee \neg y$
$\xi_4 = a \vee \neg b \vee \neg x \vee \neg y$

First, these clauses have to be written as implications in order to obtain the set of implications $\vec{\Sigma}$ given as follows:

$b \rightarrow x$
$a \rightarrow y$
$(a \wedge \neg b) \rightarrow (\neg x \vee \neg y)$
$(\neg a \wedge b) \rightarrow (\neg x \vee \neg y)$

The partition algorithm as presented above then starts with $\vec{\Sigma}' = \{\top \rightarrow \top\}$ Then, the implications of $\vec{\Sigma}$ are adjoined one after the other to $\vec{\Sigma}'$. At every step of the algorithm, the condition part of all implications of $\vec{\Sigma}'$ are mutually disjoint. After adjoining $b \rightarrow x$, $\vec{\Sigma}'$ is given by

$b \rightarrow x$
$\neg b \rightarrow \top$

After adjoining $a \rightarrow y$, the set $\vec{\Sigma}'$ consists of the following four implications:

$(\neg a \wedge b) \rightarrow x$
$(a \wedge b) \rightarrow (x \wedge y)$
$(\neg a \wedge \neg b) \rightarrow \top$
$(a \wedge \neg b) \rightarrow y$

After adjoining $a \rightarrow y$, the set $\vec{\Sigma}'$ is given by

$(\neg a \wedge b) \rightarrow x$
$(a \wedge b) \rightarrow (x \wedge y)$
$(\neg a \wedge \neg b) \rightarrow \top$
$(a \wedge \neg b) \rightarrow (\neg x \wedge y)$

After adjoining the last implication of $\vec{\Sigma}$, the final result is obtained because there are no simplifications possible. $\vec{\Sigma}''$ is therefore given by

$(\neg a \wedge b) \rightarrow (x \wedge \neg y)$
$(a \wedge b) \rightarrow (x \wedge y)$
$(\neg a \wedge \neg b) \rightarrow \top$
$(a \wedge \neg b) \rightarrow (\neg x \wedge y)$

$\ominus$

4.1.2 Constructing the Mass Function

Let $\vec{\Sigma}'' = \{\alpha_0'' \rightarrow \beta_0'', \ldots, \alpha_s'' \rightarrow \beta_s''\}$ be the set of implications obtained by the partition algorithm from the knowledge base ξ. Then, the corresponding mass function $[\varphi]_m : 2^{N_\mathcal{V}} \rightarrow [0, 1]$ is constructed as follows:

$$[\varphi(B)]_m \;=\; \begin{cases} p(N_\mathcal{A}(\alpha_i'')) & \text{if there is } \alpha_i'' \rightarrow \beta_i'' \in \vec{\Sigma}'' \text{ with } N_\mathcal{V}(\beta_i'') = B, \\ 0 & \text{otherwise.} \end{cases}$$

$[\varphi]_m$ is a valid mass function because $\vec{\Sigma}''$ satisfies conditions (1) to (4) on page 38. The potential φ can be used to answer queries with hypotheses $h \in \mathcal{L}_\mathcal{V}$ because by the above construction, the domain of φ is $d(\varphi) = \mathcal{V}$. We refer to Section 7.6 for the most general case, where $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$.

Note that the mass function $[\varphi]_m$ is unequivocally determined for a given set of clauses Σ. Conversely, several sets of clauses may lead to the same mass function.

4.1.3 Computing Degrees of Quasi-Support

Let $h \in \mathcal{L}_\mathcal{V}$ be an arbitrary hypothesis. Since ξ and $\vec{\Sigma}''$ are logically equivalent, it is possible to define the set $QS_\mathcal{A}(h, \xi)$ of quasi-supporting scenarios for h in terms of the implications in $\vec{\Sigma}''$:

$$\begin{aligned} QS_\mathcal{A}(h,\xi) &= \{\mathbf{s} \in N_\mathcal{A} : \mathbf{s} \wedge \xi \models h\} \\ &= \bigcup \{N_\mathcal{A}(\alpha_i'') : \alpha_i'' \to \beta_i'' \in \vec{\Sigma}'',\ \beta_i'' \models h\}. \end{aligned} \tag{4.8}$$

As already mentioned, the sets $N_\mathcal{A}(\alpha_i'')$ are mutually disjoint. The degree of quasi-support can therefore be written as a corresponding sum of probabilities of sets $N_\mathcal{A}(\alpha_i'')$. Furthermore, if φ is the mass function constructed for $\vec{\Sigma}''$, then the equivalence between degree of quasi-support and unnormalized belief can be demonstrated as follows:

$$\begin{aligned} dqs(h,\xi) &= p(QS_\mathcal{A}(h,\xi)) \\ &= \sum \{p(N_\mathcal{A}(\alpha_i'')) : \alpha_i'' \to \beta_i'' \in \vec{\Sigma}'',\ \beta_i'' \models h\} \\ &= \sum_{B \subseteq H} [\varphi(B)]_m \ = \ [\varphi(H)]_b \end{aligned}$$

where $H = N_\mathcal{V}(h)$ represents the hypothesis h. The consequence of this is that

$$dsp(h,\xi) \ = \ [\varphi(H)]_B.$$

Therefore, degree of support corresponds to the notion of **normalized belief** in Dempster–Shafer theory.

4.1.4 Decomposition

As mentioned previously, the partition algorithm should preferably be applied to relatively small sets Σ. If the size of Σ exceeds a certain range, then it is advisable to decompose Σ into several smaller sets $\Sigma_1, \ldots, \Sigma_m$. The decomposition must be so that every assumption $a \in \mathcal{A}$ occurs in at most one of these smaller sets. Therefore, if $d_\mathcal{A}(\Sigma_i)$ denotes the set of assumptions appearing in Σ_i, then there must be

$$d_\mathcal{A}(\Sigma_i) \cap d_\mathcal{A}(\Sigma_j) \ = \ \emptyset \tag{4.9}$$

for all $i \neq j$. This prerequisite is needed because it allows to compute independent mass functions for each of the smaller sets. Independency is a basic requirement for Dempster's rule of combination.

In the following, the simple case, where $\Sigma = \{\xi_1, \ldots, \xi_r\}$ is decomposed into only two parts Σ_1 and Σ_2 is studied first. Therefore, let

$$\begin{aligned} \vec{\Sigma}_1'' &= \{\alpha_0'' \to \beta_0'', \ldots, \alpha_s'' \to \beta_s''\} \\ \vec{\Sigma}_2'' &= \{\gamma_0'' \to \delta_0'', \ldots, \gamma_t'' \to \delta_t''\} \end{aligned}$$

be the corresponding sets of implications obtained from the partition algorithm. A consequence of $d_{\mathcal{A}}(\Sigma_1) \cap d_{\mathcal{A}}(\Sigma_2) = \emptyset$ is that

$$\vec{\Sigma}' \;=\; \{(\alpha_i'' \wedge \gamma_j'') \to (\beta_i'' \wedge \delta_j'') : \; \alpha_i'' \to \beta_i'' \in \vec{\Sigma}_1'', \; \gamma_j'' \to \delta_j'' \in \vec{\Sigma}_2''\}$$

is the resulting set of implications obtained from repeatetly using the production rule for the initial set Σ. The set $QS_{\mathcal{A}}(h, \xi)$ of quasi-supporting scenarios for a hypothesis $h \in \mathcal{L}_{\mathcal{V}}$ can therefore be written as

$$QS_{\mathcal{A}}(h, \xi) \;=\; \bigcup \{N_{\mathcal{A}}(\alpha_i'' \wedge \gamma_j'') : \; (\alpha_i'' \wedge \gamma_j'') \to (\beta_i'' \wedge \delta_j'') \in \vec{\Sigma}', \; \beta_i'' \wedge \delta_j'' \models h\}.$$

Again, the sets $N_{\mathcal{A}}(\alpha_i'' \wedge \gamma_j'')$ are mutually disjoint. The degree of quasi-support is therefore a corresponding sum of probabilities $p(N_{\mathcal{A}}(\alpha_i'' \wedge \gamma_j''))$. $d_{\mathcal{A}}(\Sigma_1) \cap d_{\mathcal{A}}(\Sigma_2) = \emptyset$ implies also that

$$p(N_{\mathcal{A}}(\alpha_i'' \wedge \gamma_j'')) \;=\; p(N_{\mathcal{A}}(\alpha_i'')) \cdot p(N_{\mathcal{A}}(\gamma_j'')).$$

If $[\vartheta_1]_m$ and $[\vartheta_2]_m$ are the mass functions constructed from $\vec{\Sigma}_1''$ and $\vec{\Sigma}_2''$, then $dqs(h, \xi)$ can be expressed in terms of ϑ_1 and ϑ_2. For $H = N_{\mathcal{V}}(h)$ and $\varphi = \vartheta_1 \otimes \vartheta_2$ it is

$$\begin{aligned} dqs(h, \xi) \;&=\; p(QS_{\mathcal{A}}(h, \xi)) \\ &=\; \sum \{p(N_{\mathcal{A}}(\alpha_i'' \wedge \gamma_j'')) : \; (\alpha_i'' \wedge \gamma_j'') \to (\beta_i'' \wedge \delta_j'') \in \vec{\Sigma}', \; \beta_i'' \wedge \delta_j'' \models h\} \\ &=\; \sum \{p(N_{\mathcal{A}}(\alpha_i'')) \cdot p(N_{\mathcal{A}}(\gamma_j'')) : \; (\alpha_i'' \wedge \gamma_j'') \to (\beta_i'' \wedge \delta_j'') \in \vec{\Sigma}', \; \beta_i'' \wedge \delta_j'' \models h\} \\ &=\; \sum_{B_1 \cap B_2 \subseteq H} [\vartheta_1(B_1)]_m \cdot [\vartheta_2(B_2)]_m \;=\; \sum_{B \subseteq H} [(\vartheta_1 \otimes \vartheta_2)(B)]_m \\ &=\; [(\vartheta_1 \otimes \vartheta_2)(H)]_b \;=\; [\varphi(H)]_b. \end{aligned} \tag{4.10}$$

The most general case, where Σ is split up into several subsets $\Sigma_1, \ldots, \Sigma_m$ can be obtained by repeatedly decomposing Σ into only two parts. Therefore, if the knowledge base ξ is transformed into a set of clauses Σ which is then split up into several sets $\Sigma_1, \ldots, \Sigma_m$ so that $d_{\mathcal{A}}(\Sigma_i) \cap d_{\mathcal{A}}(\Sigma_j) = \emptyset$ whenever $i \neq j$, and if next $\vartheta_1, \ldots, \vartheta_m$ are constructed for these sets, then

$$dqs(h, \xi) \;=\; [\varphi(H)]_b$$

for $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ and $H = N_{\mathcal{V}}(h)$.

4.2 The "Communication Line" Example

It might be interesting to look again at the example of Section 2.8. For this example, an argumentation system $\mathcal{AS}_\mathcal{P} = (\xi, \mathcal{P}, \mathcal{A})$ has been constructed. However, we were not able to compute the degree of support of a hypothesis by first computing corresponding sets of symbolic arguments.

In the following, we will first construct a probabilistic argumentation system $\mathcal{PAS}_\mathcal{P} = (\xi, \mathcal{P}, \mathcal{A}, \Pi)$ for the argumentation system $\mathcal{AS}_\mathcal{P}$. For this, we suppose that $p(a_i) = 0.8$ and $p(b_i) = 0.5$. It is often not easy to justify the values which are assigned to assumptions. Here, we try to express the fact that two wires of different quality were used when the network was set up. Finally, we will show how the knowledge base of this example can be transformed into equivalent potentials $\vartheta_1, \ldots, \vartheta_m$.

Example 4.2 Communication Line (continued) In order to construct independent mass functions, the knowledge base ξ consisting of $2m$ rules of the form

$$\begin{aligned}(x_{i-1} \wedge a_i) &\rightarrow x_i \\ (x_{i-1} \wedge b_i) &\rightarrow x_i\end{aligned}$$

has first to be transformed into an equivalent set of clauses Σ. Here, the set Σ consists of $2m$ clauses and is given by

$$\Sigma = \{(\neg x_0 \vee \neg a_1 \vee x_1), (\neg x_0 \vee \neg b_1 \vee x_1), \ldots, (\neg x_{m-1} \vee \neg b_m \vee x_m)\}.$$

Σ can be split up into sets of clauses $\Sigma_1, \ldots, \Sigma_m$. Each of these sets Σ_i contains only two clauses and is given by

$$\Sigma_i = \{(\neg x_{i-1} \vee \neg a_i \vee x_i), (\neg x_{i-1} \vee \neg b_i \vee x_i)\}.$$

Then, these sets of clauses have to be transformed into equivalent sets of implications. Therefore,

$$\vec{\Sigma}_i = \{(a_i \rightarrow (\neg x_{i-1} \vee x_i)), (b_i \rightarrow (\neg x_{i-1} \vee x_i))\}.$$

The results of the partition algorithm applied to each of these sets of implications are then the sets $\vec{\Sigma}'_1, \ldots, \vec{\Sigma}'_m$ given by

$$\vec{\Sigma}''_i = \{((a_i \vee b_i) \rightarrow (\neg x_{i-1} \vee x_i)), (\neg(a_i \vee b_i) \rightarrow \top)\}.$$

Then, mass functions $\vartheta_1, \ldots, \vartheta_m$ with $d(\vartheta_i) = \{x_{i-1}, x_i\}$ can be constructed. If $\Theta_{x_i} = \{\mathbf{f}_i, \mathbf{t}_i\}$, then $\Theta_{\{x_{i-1}, x_i\}}$ is given by

$$\Theta_{\{x_{i-1}, x_i\}} = \{(\mathbf{f}_{i-1}\mathbf{f}_i), (\mathbf{f}_{i-1}\mathbf{t}_i), (\mathbf{t}_{i-1}\mathbf{f}_i), (\mathbf{t}_{i-1}\mathbf{t}_i)\}.$$

Each mass function ϑ_i has two focal sets. The first focal set corresponds to the formula $\neg x_{i-1} \vee x_i$ and the second to the tautology represented by $\top$. Therefore, the first focal set is $\{(\mathbf{f}_{i-1}\mathbf{f}_i), (\mathbf{f}_{i-1}\mathbf{t}_i), (\mathbf{t}_{i-1}\mathbf{t}_i)\}$ and the second is

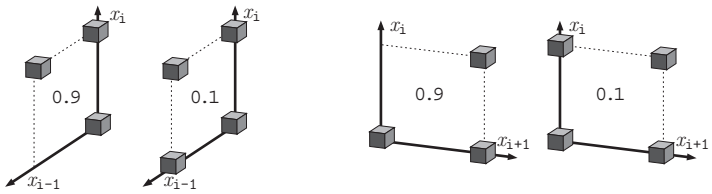

Figure 4.1: The Focal Sets of ϑ_i and ϑ_{i+1}.

$\Theta_{\{x_{i-1},x_i\}}$. Because the probabilities a priori assigned to the assumptions are $p(a_i) = 0.8$ and $p(b_i) = 0.5$, the masses assigned to these focal sets are 0.9 $(= p(a_i \vee b_i))$ and 0.1 $(= p(\overline{a_i \vee b_i}))$. In Figure 4.1, the focal sets of ϑ_i and ϑ_{i+1} are shown.

For the computation of $dsp(x_0 \rightarrow x_m, \xi)$ it would be necessary to compute the joint potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$. However, this is only feasible for relatively small values of m because φ would have 2^m focal sets. For example, the combination $\vartheta_i \otimes \vartheta_{i+1}$ has the $2^2 = 4$ focal sets shown in Figure 4.2.

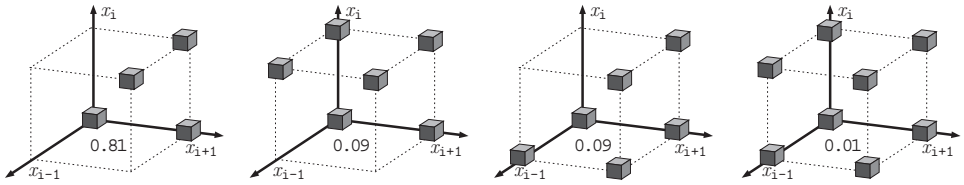

Figure 4.2: The Focal Sets of the Combination $\vartheta_i \otimes \vartheta_{i+1}$.

$\ominus$

The computation of the joint potential is therefore often not feasible. The framework of valuation networks which will be described in the next chapter shows a way how a query of the form $[\varphi(H)]_b$ can be answered without that the joint potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ has to be formed explicitly.

5

Local Computations in Valuation Networks

In the last chapter, it was proved that given a hypothesis h and a knowledge base ξ, $dqs(h, \xi)$ is equal to $[\varphi(H)]_b$ for $H = N_{\mathcal{V}}(h)$ and a potential φ which is equivalent to ξ. Therefore, computing degrees of quasi-support can be reduced to the computation of **unnormalized belief** in Dempster–Shafer theory. However, the last example of the previous chapter has shown that it is often not feasible to compute the joint potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ for given potentials $\vartheta_1, \ldots, \vartheta_m$. Fortunately, in Dempster–Shafer theory it is by Equation 3.10 (see page 30)

$$[\varphi(H)]_b \quad = \quad [\varphi^{\downarrow D_h}(H_h)]_b \tag{5.1}$$

where $D_h = d(h)$ is the set of variables occuring in h and $H_h = N_{D_h}(h)$. Therefore, the problem to solve is to compute the marginal $\varphi^{\downarrow D_h}$ without that the joint potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ has to be formed explicitly.

The solution to this problem is given by the framework of **valuation networks**, where four simple axioms enable **local computation** of marginals of a joint valuation. It is a very general framework and probabilistic argumentation systems as well as Dempster–Shafer theory fit perfectly well into this framework. In the framework of valuation networks, knowledge is represented by so-called **valuations** and inferences are drawn using two operations called **combination** and **marginalization**. The marginal of any subset of variables can be computed by eliminating one variable after another. This is the basic idea of **Shenoy's fusion algorithm**. Thus, the elimination of a single variable is the essential operation of the fusion algorithm.

In this chapter, the framework of valuation networks will be described. First, we will show that Dempster–Shafer theory fits perfectly well into this framework. Then, the four axioms which enable local computation will be given. Afterwards, Shenoy's fusion algorithm will be described. Finally, we will look again at the last example of the previous chapter and we will show that for this example, marginals can be computed efficiently using the framework of valuation networks.

5.1 The Valuation Network Framework

The framework of valuation networks is a very general framework and was first introduced in (Shenoy, 1989). Probabilistic argumentation systems as well as Dempster–Shafer theory fit perfectly well into this framework. Besides these, it can also be used for Bayesian probability theory (Pearl, 1986; Pearl, 1988), Spohn's theory of epistemic beliefs (Spohn, 1987; Shenoy, 1991), and possibility theory (Zadeh, 1987; Dubois & Prade, 1986). In a valuation network, knowledge is represented by **valuations** and inferences are drawn using two operations called **combination** and **marginalization**.

In the following, the framework of valuation networks is presented in a general form. Additional information about this framework can be found in (Shenoy & Shafer, 1990; Shenoy, 1992; Shenoy, 1994).

Variables and Configurations. The symbol Θ_x is used to denote the set of values of a variable x. It is assumed that all variables have finite sets of values. The symbol $\mathcal{V}$ denotes the set of all variables. For $D \subseteq \mathcal{V}$ let Θ_D denote the Cartesian product of the values of variables $x \in D$, that is $\Theta_D = \times\{\Theta_x : x \in D\}$. Θ_D is called the frame for D.

Valuations. Valuations are the basic objects in the framework of valuation networks. Intuitively, a valuation ϕ for a set of variables D represents some knowledge about the variables in D. We write $d(\phi) = D$ in this case and call $d(\phi)$ the *domain* of ϕ.

Combination. The symbol $\otimes$ is used for combination. Combination is a function $(\phi, \psi) \mapsto \phi \otimes \psi$. Intuitively, it corresponds to aggregation of knowledge. If ϕ and ψ are valuations for D_1 and D_2 respectively, then $\phi \otimes \psi$ is a valuation for $D_1 \cup D_2$. Therefore, $d(\phi \otimes \psi) = d(\phi) \cup d(\psi)$.

Marginalization. The symbol $\downarrow$ is used for marginalization. Marginalization is a function $(\phi, D') \mapsto \phi^{\downarrow D'}$, where $D' \subseteq d(\phi)$. Intuitively, it corresponds to coarsening of knowledge. If ϕ is a valuation for D and $D' \subseteq D$, then $\phi^{\downarrow D'}$ is a valuation for D'. Therefore, $d(\phi^{\downarrow D'}) = D'$.

Where Dempster–Shafer theory is concerned, valuations correspond to mass functions, belief functions, or commonality functions. In addition, combination and marginalization were defined in Chapter 3.

5.2 Axioms for Local Computations

In order to be able to compute marginals of a joint valuation efficiently using local computations, the following set of axioms has to be satisfied (Shenoy, 1989; Shenoy & Kohlas, 2000).

Axiom A1 (Neutrality). There is a neutral potential e_D for each domain D. For two arbitrary domains E and F

$$e_E \otimes e_F = e_{E \cup F}. \tag{5.2}$$

Axiom A2 (Commutativity and associativity). Suppose ϕ_1, ϕ_2, and ϕ_3 are valuations for D_1, D_2, and D_3, respectively. Then

$$\phi_1 \otimes \phi_2 = \phi_2 \otimes \phi_1, \tag{5.3}$$
$$\phi_1 \otimes (\phi_2 \otimes \phi_3) = (\phi_1 \otimes \phi_2) \otimes \phi_3. \tag{5.4}$$

Axiom A3 (Transitivity of marginalization). Suppose ϕ is a valuation for D, and suppose $F \subseteq E \subseteq D$. Then

$$\phi^{\downarrow F} = \left(\phi^{\downarrow E}\right)^{\downarrow F}. \tag{5.5}$$

Axiom A4 (Distributivity of marg. over combination). Suppose ϕ_1 and ϕ_2 are valuations for D_1 and D_2 respectively. Then

$$(\phi_1 \otimes \phi_2)^{\downarrow D_1} = \phi_1 \otimes \phi_2^{\downarrow D_1 \cap D_2}. \tag{5.6}$$

Axiom A2 implies that a combination of valuations $\phi_1, \ldots, \phi_m$ can be written in any order and without parentheses. Axiom A3 implies that the order in which variables are eliminated does not matter. Finally, it is Axiom A4 which enables local computations. It implies that it is not necessary to compute the joint valuation when a marginal has to be computed.

The following theorem is similar to Axiom A4 above. It states under which conditions an expression $(\varphi_1 \otimes \varphi_2)^{\downarrow D_3}$ can be written as $\varphi_1^{\downarrow D_3} \otimes \varphi_2$.

Theorem 5.1 *Suppose that ϕ_1 and ϕ_2 are valuations for D_1 and D_2 respectively. If $D_2 \subseteq D_3 \subseteq D_1$ then*

$$(\phi_1 \otimes \phi_2)^{\downarrow D_3} = \phi_1^{\downarrow D_3} \otimes \phi_2. \tag{5.7}$$

Proof See Appendix, page 161. □

Of course, it has to be proved that Dempster–Shafer theory satisfies the axioms of the framework of valuation networks:

Theorem 5.2 *Dempster–Shafer theory satisfies the axioms above.*

Proof See Appendix, page 161. □

5.3 The Fusion Algorithm

The fusion algorithm allows to compute the marginal of any subset of variables by successively eliminating one variable after another. The elimination of a single variable is therefore the essential operation of the fusion algorithm. Let $\Phi = \{\phi_1, \ldots, \phi_m\}$ be the set of given valuations for domains $D_1, \ldots, D_m$ with $D = D_1 \cup \cdots \cup D_m$. For the elimination of a variable $x \in D$, the valuations which contain x are of particular interest. If $\sigma = \otimes\{\phi \in \Phi : \ x \in d(\phi)\}$ denotes the corresponding combined valuation, $S = d(\sigma)$, then

$$\phi_{m+1} \quad = \quad \sigma^{\downarrow S-\{x\}} \tag{5.8}$$

is a new valuation, where x does not occur any more. Then, we can define

$$\mathrm{Fus}_x\{\Phi\} \quad = \quad \{\phi_{m+1}\} \cup \{\phi \in \Phi : \ x \notin d(\phi)\} \tag{5.9}$$

as the set of valuations that remains after the elimination of x. In addition, by Theorem 5.1

$$(\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow D-\{x\}} \quad = \quad \otimes \mathrm{Fus}_x\{\Phi\}. \tag{5.10}$$

By successively eliminating one variable after another, the marginal for any subset can be obtained. Therefore, suppose that $T \subseteq D$ is the set of variables for which the marginal has to be computed and suppose in addition that $\langle x_1, \ldots, x_\ell \rangle$ is the sequence in which the variables in $D - T$ are eliminated. Furthermore, let $\Phi_0 = \Phi = \{\phi_1, \ldots, \phi_m\}$ be the set of valuations at the beginning. The complete process can then be described as follows:

$$\begin{aligned} \Phi_1 &= \mathrm{Fus}_{x_1}(\Phi_0), \\ \Phi_2 &= \mathrm{Fus}_{x_2}(\Phi_1), \\ &\vdots \\ \Phi_\ell &= \mathrm{Fus}_{x_\ell}(\Phi_{\ell-1}). \end{aligned}$$

At each step i of the elimination process, $1 \leq i \leq \ell$, the variable x_i is eliminated and a new valuation ϕ_{m+i} for a corresponding domain D_{m+i} is created. Finally, after the elimination of x_ℓ, it is

$$(\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow T} \quad = \quad \otimes \Phi_\ell.$$

The main operation of the fusion algorithm is the elimination of a single variable. Therefore, it can alternatively be written as a sequence of variable eliminations:

$$\begin{aligned} (\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow T} &= \otimes \mathrm{Fus}_{x_\ell}(\cdots \mathrm{Fus}_{x_2}(\mathrm{Fus}_{x_1}(\{\phi_1, \ldots, \phi_m\})) \cdots) \\ &= (\cdots (\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow D-\{x_1\}} \cdots)^{\downarrow D-\{x_1,\ldots,x_\ell\}} \end{aligned}$$

The fusion algorithm works with every elimination sequence $\langle x_1, \ldots, x_\ell \rangle$. However, the efficiency of the algorithm depends strongly on the choice of the sequence. In the next chapter, we show how a "good" elimination sequence can

be found so that the total time to compute $(\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow T}$ becomes as small as possible.

A method which needs the elimination of all variables will be presented in Chapter 8. For this method, T is therefore equal to the empty set.

5.4 The "Communication Line" Example

In the previous chapter, potentials $\vartheta_1, \ldots, \vartheta_m$ were constructed for the example of Section 2.8. Each potential ϑ_i is defined on domain $d(\vartheta_i) = \{x_{i-1}, x_i\}$. If the state space of the variable x_i is given by $\Theta_{x_i} = \{\mathbf{f}_i, \mathbf{t}_i\}$, then the focal sets of ϑ_i are $\{(\mathbf{f}_{i-1}\mathbf{f}_i), (\mathbf{f}_{i-1}\mathbf{t}_i), (\mathbf{t}_{i-1}\mathbf{t}_i)\}$ and $\Theta_{\{x_{i-1},x_i\}}$. The masses attached to these focal sets are 0.9 and 0.1 respectively. Even though the potentials have only two focal sets, it is not feasible to compute the joint potential $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ if m is too big. This was the reason why it was not possible to compute $dsp(x_0 \rightarrow x_m, \xi)$.

In the following, we will show how $dsp(x_0 \rightarrow x_m, \xi)$ can be computed without that the joint potential has to be formed explicitly. For this, the marginal $\varphi^{\downarrow\{x_0,x_m\}}$ is computed using the fusion algorithm. If the marginal is known, it is easy to compute $dsp(x_0 \rightarrow x_m, \xi)$.

Example 5.1 Communication Line (continued) If $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ denotes the joint potential and if $\langle x_1, x_2, \ldots, x_{m-1}\rangle$ is the elimination sequence, then the fusion algorithm computes

$$\begin{aligned}\varphi^{\downarrow\{x_0,x_m\}} &= \otimes \mathrm{Fus}_{x_{m-1}}(\cdots \mathrm{Fus}_{x_2}(\mathrm{Fus}_{x_1}(\{\vartheta_1, \ldots, \vartheta_m\}))\cdots) \\ &= (\cdots((\vartheta_1 \otimes \vartheta_2)^{\downarrow\{x_0,x_2\}} \otimes \vartheta_3)^{\downarrow\{x_0,x_3\}} \cdots \otimes \vartheta_m)^{\downarrow\{x_0,x_m\}}\end{aligned}$$

Therefore, the elimination of x_1 generates the new potential

$$\vartheta_{m+1} = (\vartheta_1 \otimes \vartheta_2)^{\downarrow\{x_0,x_2\}}$$

on domain $d(\vartheta_{m+1}) = \{x_0, x_2\}$. It can be verified that ϑ_{m+1} has the two focal sets $\{(\mathbf{f}_0\mathbf{f}_2), (\mathbf{f}_0\mathbf{t}_2), (\mathbf{t}_0\mathbf{t}_2)\}$ and $\Theta_{\{x_0,x_2\}}$ with masses 0.81 and 0.19 respectively. In Figure 5.1, the two focal sets of ϑ_{m+1} are shown.

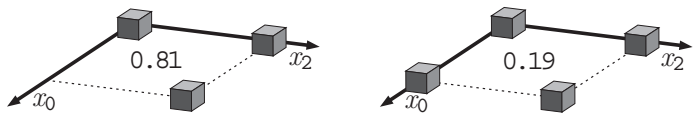

Figure 5.1: The Focal Sets of ϑ_{m+1}.

In general, the elimination of the variable x_i, $1 < i \leq m-1$, generates a new potential ϑ_{m+i} on domain $d(\vartheta_{m+i}) = \{x_0, x_{i+1}\}$ given by

$$\vartheta_{m+i} = (\vartheta_{m+i-1} \otimes \vartheta_{i+1})^{\downarrow\{x_0,x_{i+1}\}}.$$

ϑ_{m+i} has focal sets $\{(\mathbf{f}_0\mathbf{f}_{i+1}), (\mathbf{f}_0\mathbf{t}_{i+1}), (\mathbf{t}_0\mathbf{t}_{i+1})\}$ and $\Theta_{\{x_0,x_{i+1}\}}$ with masses 0.9^{i+1} and $1 - 0.9^{i+1}$ respectively. As an example, Figure 5.2 shows that ϑ_{m+2} differs from ϑ_{m+1} only in the masses assigned to the focal sets.

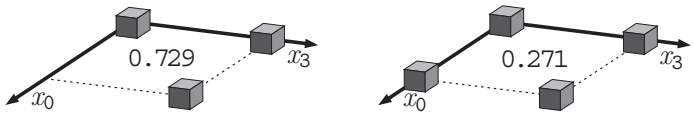

Figure 5.2: The Focal Sets of ϑ_{m+2}.

Finally, the marginal $\varphi^{\downarrow\{x_0,x_m\}}$ is obtained after the elimination of x_{m-1}. Its two focal sets are $\{(\mathbf{f}_0\mathbf{f}_m), (\mathbf{f}_0\mathbf{t}_m), (\mathbf{t}_0\mathbf{t}_m)\}$ and $\Theta_{\{x_0,x_m\}}$ with masses 0.9^m and $1 - 0.9^m$ respectively. Because $dqs(\perp, \xi) = 0$ and because the hypothesis $h = x_0 \rightarrow x_m$ is equivalent to the first focal set of $\varphi^{\downarrow\{x_0,x_m\}}$, it is finally

$$dsp(x_0 \rightarrow x_m, \xi) \quad = \quad 0.9^m$$

$\ominus$

6
Constructing Join Trees

Shenoy's **fusion algorithm**, presented in the previous chapter, computes the marginal of a subset of variables by eliminating the remaining variables one after another. A query of which the domain is the same as the domain of the marginal can then be answered very easily. However, very often there is not only one query but several queries on different domains. In this case, the repeated application of Shenoy's fusion algorithm would be inefficient, because there would be much duplication of effort.

If there are several queries on different domains, it is much more efficient to make use of a **join tree**. A join tree consists of a set of nodes, where each node is connected to one or more neighbor nodes and where the initially given potentials are distributed on the nodes. Marginals are then computed on the basis of a message-passing scheme, where nodes receive and send messages to their neighbor nodes. Therefore, a join tree can be seen as a data structure which allows to **compute marginals efficiently**.

In this chapter, we will see how a join tree can be constructed for the initially given potentials $\vartheta_1, \ldots, \vartheta_m$. For this, we will first show what join trees exactly are. The concept of **$\mathcal{A}$-disjoint join trees** which allows to compute degrees of support by propagating potentials in a join tree will then be introduced. After this, we will show that Shenoy's fusion algorithm leads directly to a join tree. Because the quality of the constructed join tree depends heavily on the elimination sequence used by the fusion algorithm, we will then look at **heuristics** for finding a "good" elimination sequence. We will propose a new heuristic, which is especially suited for the case, where potentials are mass functions. Finally, we will show how the join tree obtained from the application of the fusion algorithm can be simplified.

6.1 Join Trees

A join tree $\mathcal{JT} = (\mathbf{N}, \mathbf{C})$ consists of a set of nodes $\mathbf{N} = \{N_1, \ldots, N_n\}$ and a set of connections $\mathbf{C} = \{(N_i, N_j), \ldots, (N_k, N_\ell)\}$. A connection of the form

(N_i, N_j) means that nodes N_i and N_j are connected together. For two arbitrary nodes N_i and N_j, we use $Path(N_i, N_j)$ to denote the set of nodes on the path between N_i and N_j. Since $\mathcal{JT}$ is a tree, there is a unique path for every pair of nodes. We require that join trees are completely connected, which means that $Path(N_i, N_j) \neq \emptyset$ for all $1 \leq i, j \leq n$.

Every node $N_i \in \mathbf{N}$ of a join tree has a **label** D_i. A join tree has to satisfy the **Markov property**. This property requires that $D_i \cap D_j \subseteq D_k$ for every pair of nodes N_i and N_j and for every node $N_k \in Path(N_i, N_j)$. Therefore, a variable which appears in two nodes appears also in every node on the path between the two nodes.

Example 6.1 The Markov Property On the left side of Figure 6.1, it can be verified that the Markov property is violated by the node $\{b, d\}$. For example, this node is on the path between nodes $\{e, c\}$ and $\{c, d\}$, nevertheless, it does not contain the variable c. In contrast, the Markov property is fullfilled for the join tree on the right side of Figure 6.1. ⊖

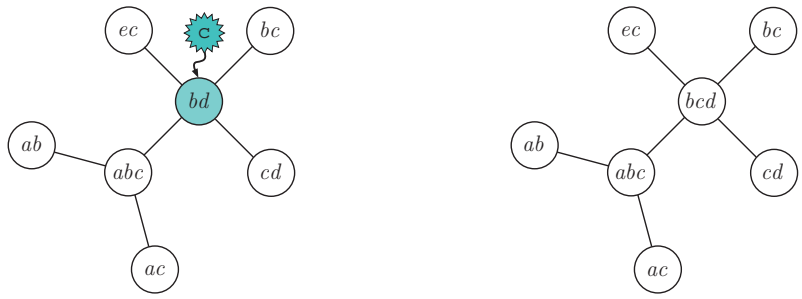

Figure 6.1: Markov Property violated and Markov Property fullfilled.

The Markov property is the reason why join trees are also called **qualitative Markov trees** (Shafer *et al.*, 1987) in literature. Sometimes, also the terms **junction trees** (Jensen *et al.*, 1990b) and **clique trees** (Lauritzen & Spiegelhalter, 1988) are used.

We will see later in this chapter how a join tree $\mathcal{JT} = (\mathbf{N}, \mathbf{C})$ is constructed for the initially given potentials $\vartheta_1, \ldots, \vartheta_m$. Each node $N_i \in \{N_1, \ldots, N_n\}$ will have a **potential** $\varphi_i = \otimes\{\vartheta_k : k \in I_i\}$, where $I_1, \ldots, I_n$ are subsets of $I = \{1, \ldots, m\}$ so that $\bigcup_{k=1}^{n} I_k = I$ and $I_i \cap I_j = \emptyset$ for all $i \neq j$. Of course, some of the sets I_i are empty, which means that the corresponding potential φ_i is equal to the neutral potential. In any case, we can think as if $\vartheta_1, \ldots, \vartheta_m$ were distributed on the nodes of the join tree.

In the following, we will sometimes denote nodes by their label. Although a join tree can contain several nodes which have the same label, it should always be clear which node is concerned.

6.2 $\mathcal{A}$-Disjoint Join Trees

The potentials $\vartheta_1, \ldots, \vartheta_m$ which are used for the construction of a join tree $\mathcal{JT}$ are originally derived from a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = \{\xi, \mathcal{V}, \mathcal{A}, \Pi\}$. Therefore, sets of clauses $\Sigma_1, \ldots, \Sigma_m$ correspond to $\vartheta_1, \ldots, \vartheta_m$ and $\Sigma = \{\Sigma_1, \ldots, \Sigma_m\}$ is equivalent to the knowledge base ξ.

If we suppose that $\Sigma_1, \ldots, \Sigma_m$ are distributed on the nodes of $\mathcal{JT}$ exactly in the same way as $\vartheta_1, \ldots, \vartheta_m$, it means that a set of clauses is attached to each node of the join tree. Therefore, every node of a join tree has a **label**, a **potential**, and a **set of clauses** which is equivalent to the potential.

It has been shown in Chapter 4 that it is important that $d_{\mathcal{A}}(\Sigma_i) \cap d_{\mathcal{A}}(\Sigma_j) = \emptyset$ for $1 \leq i \neq j \leq m$. Each assumption $a \in \mathcal{A}$ occurs therefore in at most one of the sets $\Sigma_1, \ldots, \Sigma_m$ and, consequently, it is not possible that $a \in \mathcal{A}$ occurs in the sets of clauses attached to two different nodes of the join tree.

We call a join tree an **$\mathcal{A}$-disjoint join tree** if every assumption $a \in \mathcal{A}$ occurs in at most one of the sets of clauses attached to its nodes. It is the concept of $\mathcal{A}$-disjoint join trees which allows to compute correctly degrees of support by propagating potentials.

6.3 The Fusion Algorithm

In the following, we will show that the application of Shenoy's fusion algorithm to the initially given potentials $\vartheta_1, \ldots, \vartheta_m$ leads directly to a join tree. However, this join tree often has too many nodes. In Section 6.5 we will therefore demonstrate how a join tree can easily be simplified by removing unnecessary nodes.

6.3.1 Constructing a Join Tree

Suppose that the set of potentials $\Phi_0 = \{\vartheta_1, \ldots, \vartheta_m\}$ on domains $D_1, \ldots, D_m$ is initially given. Suppose in addition that $D = \bigcup_{i=1}^{m} D_i$ is the set of all variables and that $\langle x_1, \ldots, x_\ell \rangle$ is the elimination sequence consisting of variables in $D - T$ for a set $T \subseteq D$. As shown in Section 5.3, the fusion algorithm eliminates at each step i of the elimination process the variable x_i from the current set of potentials Φ_{i-1}. If $\sigma_i = \otimes\{\vartheta \in \Phi_{i-1} : x_i \in d(\vartheta)\}$ denotes the corresponding combined potential, $S_i = d(\sigma_i)$, then

$$\vartheta_{m+i} \quad = \quad \sigma_i^{\downarrow S_i - \{x_i\}} \tag{6.1}$$

is a new potential on domain $D_{m+i} = S_i - \{x_i\}$. Finally,

$$\Phi_i \quad = \quad \{\vartheta_{m+i}\} \cup \{\vartheta \in \Phi_{i-1} : x_i \notin d(\vartheta)\}$$

is the new set of potentials. If $x_i \in D_j$ for $j \leq k$ and $x_i \notin D_j$ otherwise, then step i of the elimination process can be visualized as shown on the left

side of Figure 6.2. Each node $D_1, \ldots, D_k$ contains its corresponding potential $\vartheta_1, \ldots, \vartheta_k$, whereas the nodes S_i and D_{m+i} initially contain the neutral potential. In order to compute ϑ_{m+i}, the nodes $D_1, \ldots, D_k$ send their potential to node S_i. Then, these potentials are combined on node S_i which gives σ_i. Finally, the potential obtained by eliminating x_i from σ_i is sent to node D_{m+i} and corresponds to ϑ_{m+i}.

After the elimination of the last variable x_ℓ, the set Φ_ℓ contains one or more potentials. If Φ_ℓ contains more than one potential, a node with label $S_{\ell+1} = T$ has to be added as shown on the right side of Figure 6.2. Therefore, the nodes

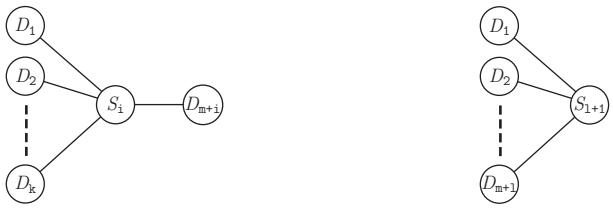

Figure 6.2: Elimination of x_i and after the Elimination of x_ℓ.

are labeled as $D_1, \ldots, D_{m+\ell}, S_1, \ldots, S_{\ell+1}$. It is indeed a join tree, because

(1) the Markov property is satisfied for all nodes,

(2) $Path(N_i, N_j) \neq \emptyset$ for every pair of nodes N_i and N_j.

Initially, the nodes $D_1, \ldots, D_m$ contain the potentials $\vartheta_1, \ldots, \vartheta_m$. The potentials $\sigma_1, \ldots, \sigma_\ell$ and $\vartheta_{m+1}, \ldots, \vartheta_{m+\ell}$ are computed on the nodes $S_1, \ldots, S_\ell$ and $D_{m+1}, \ldots, D_{m+\ell}$. Finally, the potential computed on the root node is equivalent to the marginal $(\vartheta_1 \otimes \cdots \vartheta_m)^{\downarrow T}$.

Example 6.2 Join Tree obtained from the Fusion Algorithm Suppose ϑ_1, ϑ_2 and ϑ_3 have domains $\{a, b\}$, $\{b, c\}$, and $\{c, d\}$. The join tree obtained from the fusion algorithm for $T = \{d\}$ and the elimination sequence $\langle a, b, c\rangle$ is shown in Figure 6.3. Each elimination of a variable has generated a subtree of the join tree. The subtrees resulted by eliminating a and c are emphasized. $\ominus$

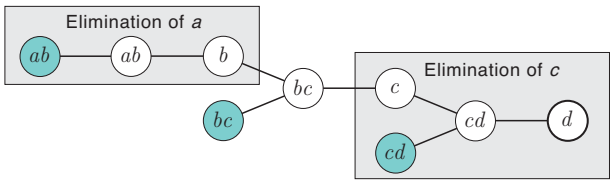

Figure 6.3: Join Tree obtained from the Fusion Algorithm.

6.4 Determinating the Elimination Sequence

The elimination sequence used by the fusion algorithm determines the join tree completely. For example, the two join trees of Figure 6.4 are the results of two different elimination sequences used for the previous example.

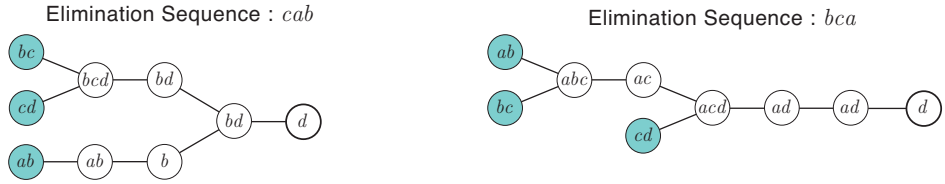

Figure 6.4: Join Trees for different Elimination Sequences.

A join tree can also be considered as a graphical visualization of the computations performed, because a sequence of operations is attached to every node. Often, it can be observed that nodes with a large label slow down computations. For nodes with a very large label they even may not be feasible at all. This is because potentials computed on these nodes often contain much more focal sets than potentials computed on nodes with smaller labels. In addition, it is much more difficult to store potentials with large domains (see Section 9.2).

Therefore, in order to minimize the overall time needed for the computations, the problem to solve is to find an elimination sequence so that the largest label of the constructed join tree is as small as possible. This problem has been studied extensively and it is known to be NP-complete (Arnborg *et al.*, 1987). Several heuristics for finding an elimination sequence producing a "good" join tree have been developed for that reason (Rose, 1970; Bertele & Brioschi, 1972; Yannakakis, 1981; Tarjan & Yannakakis, 1984; Kong, 1986; Kjærulff, 1990; Almond & Kong, 1991; Haenni & Lehmann, 1999). A comparison of different heuristics can be found in (Cano & Moral, 1995).

In this section, we will look at the problem of finding a "good" elimination sequence. The heuristic proposed is especially suited for mass functions. As other heuristics, it tries to minimize the size of the largest label of the constructed join tree, but, in addition, it takes also into consideration that variable elimination is a dynamic process. However, in order to explain this heuristic, it is necessary to talk first about **Hypergraphs** and **Hypertrees**.

6.4.1 Hypergraphs and Hypertrees

A graph $\mathcal{G}$ can be described as a pair $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \{v_1, \ldots, v_n\}$ is a set of vertices and $\mathbf{E} = \{E_1, \ldots, E_k\}$ is a set of edges $E_i \subseteq \mathbf{V}$. If all edges are pairs, then the resulting graph is a **simple graph**. If the edges are arbitrary sets of vertices, then the resulting graph is a **hypergraph** and the edges are called **hyperedges**. Note that we allow the same hyperedge to occur more

than once in a hypergraph. Usually, hypergraphs are drawn using closed curves enclosing the elements of the hyperedges.

Example 6.3 Hypergraph for a Set of Potentials The domains of the initially given set of potentials $\Phi_0 = \{\vartheta_1, \ldots, \vartheta_m\}$ form a hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \bigcup_{i=1}^{m} d(\vartheta_i)$ and $\mathbf{E} = \{d(\vartheta) : \vartheta \in \Phi_0\}$. For example, the hypergraph on the left side of Figure 6.5 corresponds to $\Phi_0 = \{\vartheta_1, \ldots, \vartheta_5\}$, where the potentials $\vartheta_1, \ldots, \vartheta_5$ are defined on domains $\{a, b\}$, $\{b, c\}$, $\{b, d\}$, $\{c, d, e\}$, and $\{e\}$. ⊖

Figure 6.5: Hypergraph and a Covering Hypertree.

A vertice v is called a **leaf** of $\mathcal{H} = (\mathbf{V}, \mathbf{E})$, if it occurs only in one hyperedge $E_i \in \mathbf{E}$. The set of leafs is denoted by *leafs*$(\mathcal{H})$. Two vertices are called **neighbors**, if there is a hyperedge, where both vertices occur. The set of neighbors of a vertice v in $\mathcal{H}$ is denoted by $N(v, \mathcal{H})$. The set consisting of a vertice v and its neighbors is called the **closure** of v in $\mathcal{H}$, thus, $Cl(v, \mathcal{H}) = \{v\} \cup N(v, \mathcal{H})$. The **rank** $r(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is the size of its largest hyperedge. For the hypergraph shown on the left side of Figure 6.5, it is for example *leafs*$(\mathcal{H}) = \{a\}$, $N(b, \mathcal{H}) = \{a, c, d\}$, $Cl(b, \mathcal{H}) = \{a, b, c, d\}$, and $r(\mathcal{H}) = 3$.

A hypergraph $\mathcal{T}$ is a **hypertree** if it is acyclic. However, although it can easily be recognized that the hypergraph on the left side of Figure 6.5 is not acyclic, for example, the formal definition of a cycle in a hypergraph is not as straightforward as for simple graphs. Therefore, a different approach for defining hypertrees will be used here. For that purpose, we need the concepts of twigs and branches. A hyperedge $E_t \in \mathbf{E}$ is called **twig** if there is another hyperedge $E_b \in (\mathbf{E} - E_t)$ so that

$$E_t \cap (\cup(\mathbf{E} - E_t)) \quad = \quad E_t \cap E_b. \tag{6.2}$$

We call any such hyperedge E_b a **branch** for the twig E_t.

A hypergraph is a **hypertree**, if its hyperedges can be ordered, say $\langle E_1, \ldots, E_k \rangle$, so that E_i is a twig in the hypergraph given by the hyperedges $\{E_1, \ldots, E_i\}$, for $i = 2, \ldots, k$. Such an ordering is called a **hypertree construction ordering**. If a hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$ is not a hypertree, it is always possible to find a corresponding **covering hypertree** $\mathcal{T} = (\mathbf{V}, \mathbf{E}')$. $\mathcal{T}$ covers $\mathcal{H}$ if a corresponding $E'_i \in \mathbf{E}'$ is found for every $E_i \in \mathbf{E}$ so that $E_i \subseteq E'_i$. As an example,

the hypergraph on the right side of Figure 6.5 is a covering hypertree of the hypergraph shown on the left.

Finding a covering hypertree for a hypergraph is not difficult. The hypertree $\mathcal{T} = (\mathbf{V}, \{\mathbf{V}\})$ consisting of the hyperedge V only is a covering hypertree for any hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$. However, finding the best covering hypertree, of which the largest hyperedge is as small as possible, is known to be an NP-complete problem (Arnborg *et al.*, 1987). Fortunately, there are many algorithms for finding nearly optimal covering hypertrees. These algorithms determine a **variable elimination sequence** on the basis of a specific heuristic.

Hypertrees are more useful than hypergraphs: a join tree corresponding to a covering hypertree can easily be obtained by going through the reverse hypertree construction ordering. The rank (= size of the largest label) of the join tree is then equal to the rank of the hypertree. Therefore, we are particularly interested in covering hypertrees of which the rank is as small as possible. For example, $\mathcal{T} = (\mathbf{V}, \{\mathbf{V}\})$ is the worst possible covering hypertree for a hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$.

In the following we will see that every elimination sequence which consists of all variables of $\mathcal{H}$, determines a covering hypertree for $\mathcal{H}$.

6.4.2 The Elimination of Variables

From the perspective of hypergraphs, the elimination of a variable x with the fusion algorithm corresponds to a transformation of the hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$ into another hypergraph $\mathcal{H}^{-x} = (\mathbf{V}^{-x}, \mathbf{E}^{-x})$ given by

$$\mathbf{V}^{-x} = \mathbf{V} - \{x\}, \tag{6.3}$$

$$\mathbf{E}^{-x} = (\mathbf{E} - \{E_k \in \mathbf{E} : x \in E_k\}) \cup \{Cl(x, \mathcal{H}) - \{x\}\}. \tag{6.4}$$

The transformation is illustrated on the right side of Figure 6.6, where the elimination of b causes the removal of the hyperedges $\{a, b\}$, $\{b, c\}$, and $\{b, d\}$. In addition, the new hyperedge $\{a, c, d\}$ is added to $\mathcal{H}^{-b}$. The corresponding subtree is shown on the left side of Figure 6.6.

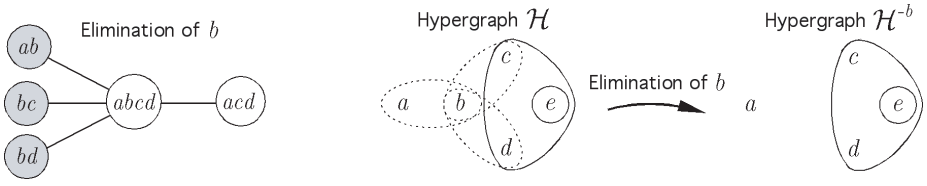

Figure 6.6: Elimination of the Variable b.

The elimination of a variable after another according to an elimination sequence $\langle x_1, \ldots, x_\ell \rangle$ yields the sequence of hypergraphs $\mathcal{H}^{-x_1}, \ldots, \mathcal{H}^{-x_1 \ldots x_\ell}$. The hypergraph $\mathcal{H}^{-x_1}$ is obtained from $\mathcal{H}$ by eliminating x_1. At step i, the elimination of

x_i transforms $\mathcal{H}^{-x_1 \cdots x_{i-1}}$ into $\mathcal{H}^{-x_1 \cdots x_i}$. The sequence of hypergraphs obtained for the previous example and the elimination sequence $\langle a, e, c, d, b \rangle$ is shown in Figure 6.7.

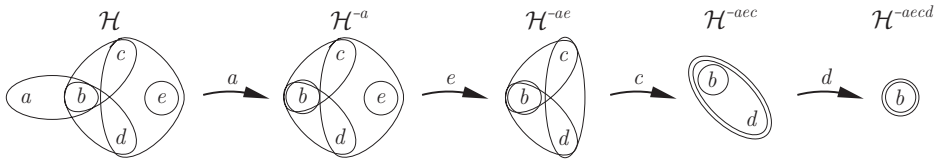

Figure 6.7: Hypergraphs for the Elimination Sequence $\langle a, e, c, d, b \rangle$.

Each time a variable is eliminated, a corresponding subtree is constructed. The join tree finally obtained consists of all these subtrees constructed previously. Therefore, if the variable x_i is eliminated at step i of the elimination process, the largest label of the corresponding subtree is given by $Cl(x_i, \mathcal{H}^{-x_1 \cdots x_{i-1}})$. Therefore, the problem of finding an elimination sequence for which the rank of the constructed join tree is as small as possible, is equivalent to the problem of finding an elimination sequence $\langle x_1, \ldots, x_\ell \rangle$ for which the maximal size of $Cl(x_i, \mathcal{H}^{-x_1 \cdots x_{i-1}})$ for $i \leq \ell$ is as small as possible. Figure 6.8 shows the sequence $Cl(x_i, \mathcal{H}^{-x_1 \cdots x_{i-1}})$ for the previous example and the elimination sequence $\langle a, e, c, d, b \rangle$. Thus, the rank of the corresponding join tree is 3.

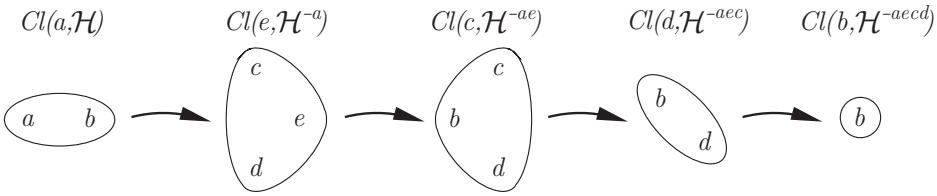

Figure 6.8: $Cl(x_i, \mathcal{H}^{x_1 \cdots x_{i-1}})$ for Elimination Sequence $\langle a, e, c, d, b \rangle$.

A join tree of rank 4 is obtained for the elimination sequence $\langle b, a, e, c, d \rangle$. If the variable b is eliminated first, $Cl(b, \mathcal{H}) = \{a, b, c, d\}$. The elimination sequence $\langle a, e, c, d, b \rangle$ is therefore better than $\langle b, a, e, c, d \rangle$.

If $\langle x_1, \ldots, x_\ell \rangle$ is an elimination sequence consisting of all variables of $\mathcal{H} = (\mathbf{V}, \mathbf{E})$ and if $S_i = Cl(x_i, \mathcal{H}^{-x_1 \cdots x_{i-1}})$, the hypergraph $\mathcal{T}$ of which the hyperedges are given by $\{S_1, \ldots, S_\ell\}$ is a covering hypertree of $\mathcal{H}$. In addition, the sequence of hyperedges $(S_\ell, \ldots, S_1)$ is a hypertree construction ordering for $\mathcal{T}$. Therefore, every elimination sequence consisting of all variables of $\mathcal{H}$ determines a covering hypertree for $\mathcal{H}$.

6.4.3 Heuristic "OSLA – Smallest Clique"

Algorithms for finding nearly optimal covering hypertrees determine a variable elimination sequence $\langle x_1, \ldots, x_\ell \rangle$ for which the maximal size of $Cl(x_i, \mathcal{H}^{-x_1 \ldots x_{i-1}})$ is as small as possible. In any case, the minimal rank of a covering hypertree for a hypergraph $\mathcal{H}$ is at least the size of the largest hyperedge of $\mathcal{H}$. The elimination of a leaf variable corresponds to the replacement of a hyperedge by a smaller hyperedge. It is thus advisable to first eliminate a leaf at every step of the elimination process, if possible.

The ideas developed so far lead directly to a widely used heuristic, known as *"One Step Look Ahead – Smallest Clique"* (*OSLA–SC* for short). This heuristic is given by the following pseudo-algorithm:

Pseudo-Algorithm *One Step Look Ahead – Smallest Clique*

1. If there is a leaf variable, eliminate it.
2. If there is no leaf variable, eliminate the variable x for which $|Cl(x, \mathcal{H}^{-x_1 \ldots x_{i-1}})|$ is as small as possible.
3. If there are several such variables, break ties arbitrarily.

If the heuristic is applied to the previous example, the variable a is eliminated first. Then, $Cl(b, \mathcal{H}^{-a}) = \{b, c, d\}$, $Cl(c, \mathcal{H}^{-a}) = Cl(d, \mathcal{H}^{-a}) = \{b, c, d, e\}$, and $Cl(e, \mathcal{H}^{-a}) = \{c, d, e\}$. Therefore, either b or e are eliminated next. Proceeding further, the heuristic generates one of the twelve elimination sequences shown in Figure 6.9. The rank of the corresponding covering hypertree for each of these sequences is 3.

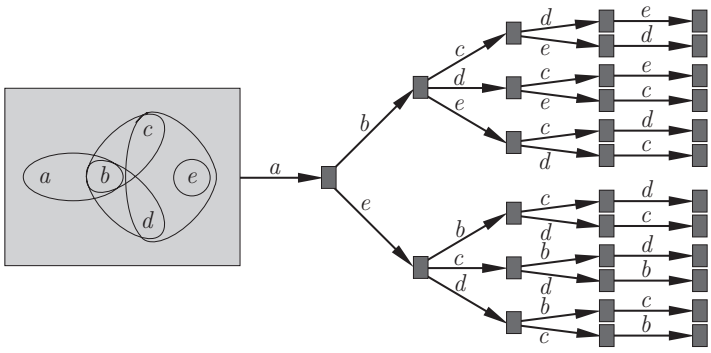

Figure 6.9: Elimination Sequences for "OSLA – Smallest Clique".

6.4.4 Heuristic "OSLA – Fewest Fill-Ins"

Another widely used heuristic is known as *"One Step Look Ahead – Fewest Fill-ins"* (*OSLA–FFI* for short). For this heuristic, it is necessary to construct

first the so-called 2-section graph of the initially given hypergraph $\mathcal{H}$. The elimination of a variable after another according to an elimination sequence $\langle x_1, \ldots, x_\ell \rangle$ generates a sequence of graphs $\mathcal{G}^{-x_1}, \ldots, \mathcal{G}^{-x_1 \ldots x_\ell}$, where $\mathcal{G}^{-x_1 \ldots x_i}$ is the corresponding 2-section graph of $\mathcal{H}^{-x_1 \ldots x_i}$. Therefore, the heuristic selects at each step of the elimination process the variable to eliminate next on the basis of the 2-section graph $\mathcal{G}^{-x_1 \ldots x_{i-1}}$ of the hypergraph $\mathcal{H}^{-x_1 \ldots x_{i-1}}$.

The **2-section graph** $\mathcal{G} = (\mathbf{N}, \mathbf{C})$ of a hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$ is given by $\mathbf{N} = \mathbf{V}$ and $\mathbf{C} = \{(v_i, v_j) : \exists E_k \in \mathbf{E}$ so that $\{v_i, v_j\} \subseteq E_k\}$. Therefore, vertices of hyperedges are connected two by two in the 2-section graph. Figure 6.10 shows the 2-section graph for the previous example.

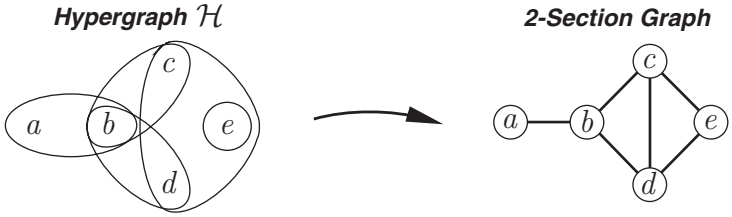

Figure 6.10: Hypergraph with corresponding 2-Section Graph.

For the graph $\mathcal{G} = (\mathbf{N}, \mathbf{C})$, a set $J \subseteq \mathbf{N}$ is completely connected if all pairs of vertices in J are linked together in $\mathcal{G}$. A **clique** is a maximal completely connected subset. For example, the 2-section graph on the right side of Figure 6.10 has the three cliques $\{a, b\}$, $\{b, c, d\}$, and $\{c, d, e\}$.

For the selection of the variable to eliminate next, the heuristic uses the so-called fill-in number for the remaining variables. The **fill-in number** for a variable x to be eliminated from $\mathcal{G}$ is the number of pairs $x_i, x_j \in N(x, \mathcal{G})$ which are not connected. In other words, it is the number of connections which would be filled if we were to eliminate x from $\mathcal{G}$. For example, $|FI(a, \mathcal{G})| = |FI(e, \mathcal{G})| = 0$, $|FI(c, \mathcal{G})| = |FI(d, \mathcal{G})| = 1$, and $|FI(b, \mathcal{G})| = 2$ for the graph shown on the right side of Figure 6.10.

It is shown in (Kong, 1986) that if the fill-in number of a certain variable is zero, it is always optimal to eliminate that variable. It seems therefore natural to eliminate the variable with the smallest fill-in number at each step. In that way, leaf variables are eliminated first because the fill-in number of a leaf variable is always zero. Note however that not only leaf variables may have a fill-in number of zero. For example, the variable e in Figure 6.10 is not a leaf variable, but nevertheless its fill-in number is zero.

The heuristic *"One Step Look Ahead – Fewest Fill-ins"* is given by the following pseudo-algorithm:

Pseudo-Algorithm *One Step Look Ahead – Fewest Fill-ins*

1. Eliminate the variable x for which $|FI(x, \mathcal{G}^{-x_1 \ldots x_{i-1}})|$ is as small as possible.

2. If there are several such variables, break ties arbitrarily.

If the heuristic is applied to the previous example, one of the variables a and e is eliminated first. If a is eliminated first, one of the twelve elimination sequences shown in Figure 6.9 is generated. If e is eliminated first, either a, c or d are eliminated next, because their fill-in number with respect to $\mathcal{G}^{-e}$ is 0. In that way, the heuristic generates one of 26 possible elimination sequences. The rank of the corresponding covering hypertree for each of these sequences is 3.

6.4.5 Heuristic "OSLA – Smallest Clique, Fewest Focal Sets"

The two heuristics presented previously have a weakness. Namely, the fact that sets of values of the variables may be of different size is not taken into consideration. From this point of view, the size of the closure $Cl(x, \mathcal{H}^{-x_1 \ldots x_{i-1}})$ used in *OSLA-SC* is not really appropriate. A better choice is the number of configurations of the closure, thus $|\Theta_{Cl(x,\mathcal{H}^{-x_1 \ldots x_{i-1}})}|$. This number measures more precisely the costs of storing an individual focal set of the potential obtained at step i of the elimination process.

Another weakness is that the selection of the variable to eliminate next is exclusively based on static entities. For example, it is possible to build the join tree without performing any combinations or marginalizations. However, variable elimination is a **dynamic process** and the number of focal sets of potentials involved in the elimination process should also be considered. If there are several variables which could be eliminated next, it seems natural to eliminate the variable x for which the computation of the corresponding potential $\vartheta_{m+i} = \sigma_i^{\downarrow S_i - \{x\}}$ can be performed as fast as possible instead of breaking ties arbitrarily. As the computation of σ_i mainly determines the time needed for the computation of ϑ_{m+i}, it is justified to use the expected number of focal sets of σ_i as an estimate for the time needed. If $\Phi_{i-1}(x)$ denotes the set of potentials available at step i of the elimination process and containing the variable x, an estimate for the number of focal sets of σ_i is given by $\prod_{\vartheta \in \Phi_{i-1}(x)} |FS(\vartheta)|$.

These ideas lead directly to a new heuristic which is based on *OSLA–SC* and given by the following pseudo-algorithm:

Pseudo-Algorithm *OSLA – Smallest Clique, Fewest Focal Sets*

1. If there is a leaf variable, eliminate it.
2. If there is no leaf variable, eliminate the variable x for which $|\Theta_{Cl(x,\mathcal{H}^{-x_1\cdots x_{i-1}})}|$ is as small as possible.
3. If there are several such variables, eliminate the variable x for which $\prod_{\vartheta\in\Phi_{i-1}(x)} |FS(\vartheta)|$ is as small as possible.
4. If there are several such variables, break ties arbitrarily.

The main difference to *OSLA–SC* is that ties are not broken arbitrarily in step 3. If *OSLA–SC* were applied to the previous example, each one of the twelve possible elimination sequences shown in Figure 6.9 could be generated. Depending on the elimination sequence generated, more or less time is needed for the computation of the marginal. In contrast, some of these elimination sequences would never be generated by *"OSLA – Smallest Clique, Fewest Focal Sets"* (*OSLA–SC–FFS* for short).

Therefore, the new heuristic has the following two objectives:

(1) generate an elimination sequence for which the rank of the corresponding join tree is as small as possible.

(2) if there are several elimination sequences possible, select the one which allows to compute the marginal as fast as possible.

Of course, the first objective is much more important than the second one. Nevertheless, we think that it is worthwhile considering the second objective as well. Note that the heuristic *OSLA–FFI* can be modified in the same way as *OSLA–SC*. We focused on the latter heuristic because it is much easier to implement efficiently (see Chapter 9).

6.4.6 Heuristic "OSLA–SC–FFS, Initial Structure"

The potential $\vartheta_{m+i} = \sigma_i^{\downarrow S_i - \{x_i\}}$ which is created in step i by the elimination of the variable x_i has often much more focal sets than each one of the initially given potentials $\vartheta_1, \ldots, \vartheta_m$. Therefore, the heuristic *OSLA–SC–FFS* tends to select variables which occur in the domain of newly created potentials. However, the elimination of a non-leaf variable which occurs only in the domain of initially given potentials is sometimes not a good idea. Such a variable is in some sense located somewhere in the "centre" of the hypergraph given by the domains of the initially given potentials.

It would be better to continue the work done so far by eliminating a variable which is "close" to variables which have already been eliminated. For example, the heuristic *OSLA–FFI* promotes the elimination of such variables: if x_i is eliminated in step i, then the variables in $S_i - \{x_i\}$ are completely connected in

the corresponding 2-section graph $\mathcal{G}^{-x_1 \cdots x_i}$. Therefore, there is a high chance that one of the variables in $S_i - \{x_i\}$ is eliminated in the next step.

Another way to promote the elimination of variables which are "close" to variables already eliminated is not to consider the number of focal sets of the potentials ϑ_{m+i} created during the elimination process. Thus, only the number of focal sets of the initially given potentials $\Phi_0 = \{\vartheta_1, \ldots, \vartheta_m\}$ are used for the selection of the variable to eliminate next. This idea leads to the heuristic *OSLA–SC–FFS–IS* given by the following pseudo algorithm:

Pseudo-Algorithm *OSLA – SC–FFS, Initial Structure*

1. If there is a leaf variable, eliminate it.
2. If there is no leaf variable, eliminate the variable x for which $|\Theta_{Cl(x,\mathcal{H}^{-x_1 \cdots x_{i-1}})}|$ is as small as possible.
3. If there are several such variables, eliminate the variable x for which $\prod_{\vartheta \in (\Phi_0 \cap \Phi_{i-1}(x))} |FS(\vartheta)|$ is as small as possible.
4. If there are several such variables, break ties arbitrarily.

This heuristic is located somewhere between *OSLA–SC* and *OSLA–SC–FFS*. Just as the latter, the number of focal sets is used for selecting the variable to eliminate next. However, here, the join tree could be built in advance without performing any combination or marginalization. Clearly, this is an advantage because it allows to compute marginals more efficiently using a **network of computers**. If each node is associated with a computer, computations can then be performed in parallel on several computers. Thus, each computer receives messages from neighboring computers and computes then messages to its neighbors. Such a **distributed computing** of marginals is not possible for the heuristic *OSLA–SC–FFS*.

In Chapter 11, we will compare the five heuristics for selecting the variable to eliminate next. For each of the examples given there, some heuristics work better than others. Therefore, the conclusion will be that there is no "best" heuristic which gives for all problem instances the best results.

6.5 Simplification of the Join Tree

The join tree obtained from the fusion algorithm often contains more nodes than necessary. For example, the join tree of Figure 6.3 consists of nine nodes, even though initially there were only three potentials given. In the following, five simple simplifications for removing unnecessary nodes are proposed. The first simplification has to be performed prior to the construction of the join tree. In contrast, for the remaining four simplifications we will suppose that the join tree is given right from the beginning because this will allow us to explain these kinds of simplification in an intelligible manner. Note, however,

that these simplifications are actually performed dynamically during the process of eliminating one variable after another.

1. Simplification: Special Handling of Observations

In the previous section, we have seen that the sequence in which variables are eliminated is determined in such a way that the size of the largest label of the resulting join tree is as small as possible. This is also the main objective of the first simplification method. Note, however, that this method has to be applied prior to the construction of the join tree. We even propose to apply it before potentials are constructed for the given probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = \{\xi, \mathcal{V}, \mathcal{A}, \Pi\}$.

The knowledge base ξ can always be transformed into an equivalent set of SCL-clauses. However, ξ is often already given as a set of SCL-clauses $\Sigma = \{\xi_i, \ldots, \xi_r\}$. It can occur that some of these clauses involve only a single variable and correspond to an assignment of a value to this variable. Typically, observations and facts which are added to the knowledge base have often this particular form. Therefore, suppose that $\delta \in \Sigma$ is such a SCL-clause of the form $\langle x_i = \theta_{ij} \rangle$, where $\theta_{ij} \in \Theta_{x_i}$. Σ can then be split up into sets Σ_*, Σ_+, and Σ_- as follows:

$$\begin{aligned} \Sigma_* &= \{\xi_k \in \Sigma : x_i \notin d(\xi_k)\} \\ \Sigma_+ &= \{\xi_k \in \Sigma : \xi_k = \langle x_i \in X_i \rangle \wedge \alpha_k, \theta_{ij} \in X_i\} \\ \Sigma_- &= \{\xi_k \in \Sigma : \xi_k = \langle x_i \in X_i \rangle \wedge \alpha_k, \theta_{ij} \notin X_i\} \end{aligned}$$

The simplification method consists in constructing for Σ an equivalent set of SCL-clauses Σ' containing fewer and simpler SCL-clauses. For example, δ entails each of the SCL-clauses of Σ_+, thus $\delta \models \xi_k$ for all $\xi_k \in \Sigma_+$. Moreover, $\delta \wedge \xi_k \equiv \alpha_k$ for each $\xi_k \in \Sigma_-$ of the form $\langle x_i \in X_i \rangle \wedge \alpha_k$. As a consequence, the set Σ' constructed as

$$\Sigma' = \Sigma_* \cup \{\delta\} \cup \{\alpha_k : \xi_k \in \Sigma_-, \xi_k = \langle x_i \in X_i \rangle \wedge \alpha_k\}$$

is logically equivalent to Σ but contains fewer and simpler SCL-clauses. Usually, the join tree constructed for $\mathcal{PAS}_{\mathcal{V}}{}' = \{\xi', \mathcal{V}, \mathcal{A}, \Pi\}$, where ξ' is given by the set of SCL-clauses Σ' then has smaller labels than the one constructed for $\mathcal{PAS}_{\mathcal{V}} = \{\xi, \mathcal{V}, \mathcal{A}, \Pi\}$.

2. Simplification: Merging Potentials with same Domains

It can happen that two or more potentials of the initially given set of potentials $\Phi_0 = \{\vartheta_1, \ldots, \vartheta_m\}$ have the same domain. In this case, by combining potentials with the same domain, a new set of potentials $\Phi_0' = \{\vartheta_1', \ldots, \vartheta_s'\}$ is obtained, where all potentials have different domains. The fusion algorithm applied to Φ_0' instead of Φ_0 then gives a simpler join tree.

Note that the sequence in which potentials are combined is important. It is better to combine potentials with fewer focal sets first. This kind of simplification is not always advisable. For example, merging potentials with the same domain is sometimes not justified for a method which will be presented later in Chapter 8.

3. Simplification: Direct Marginalization

Suppose that the variable x_i has been elimiminated. The corresponding potentials σ_i and ϑ_{m+i} are then associated with the nodes S_i and D_{m+i} respectively. Similarly, if a variable $x_j \in d(\vartheta_{m+i})$ is eliminated next, then σ_j and ϑ_{m+j} are associated with S_j and D_{m+j} respectively. Of particular interest is the situation, where x_j occurs in none of the remaining potentials. This situation is shown on the left side of Figure 6.11.

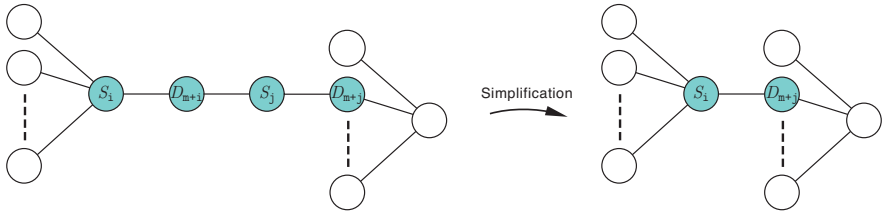

Figure 6.11: Removing the Nodes D_{m+i} and S_j.

If x_j occurs in none of the remaining potentials, two marginalizations are performed one after another. The first one is performed on node S_i and the second one on node S_j. However, this last marginalization would not have been necessary if both variables would have been eliminated at the same time. Figure 6.11 shows that two nodes can then be omitted. Of course, note that it can even happen that $d(\vartheta_{m+i})$ contains several variables which occur in none of the remaining potentials. In this case, even more than two nodes can be omitted.

4. Simplification: Removing Nodes $D_{m+1}, \ldots, D_{m+\ell}$

The fourth simplification concerns the nodes $D_{m+1}, \ldots, D_{m+\ell}$. If D_{m+i} is one of these nodes, it is always between nodes S_i and $S_j \in \{S_{i+1}, \ldots, S_{\ell+1}\}$. Moreover, D_{m+i} has no other neighbor nodes. This situation is shown on the left side of Figure 6.12.

In any case, $D_{m+i} = S_i \cap S_j$. Therefore, node S_i can be directly connected to S_j without violating the Markov property. On the right side of Figure 6.12, D_{m+i} does not occur any more. Thus, the nodes $D_{m+1}, \ldots, D_{m+\ell}$ are not really needed and we propose to remove them from the join tree. It is worth performing this kind of simplification specially if the elimination sequence consists of many variables.

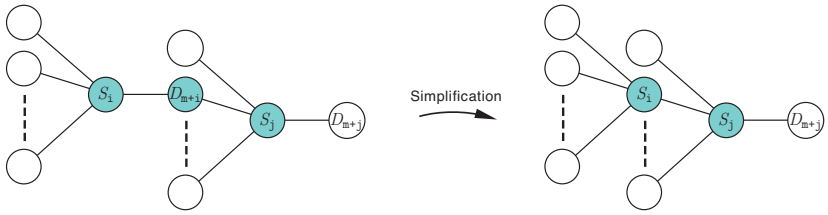

Figure 6.12: Removing the Node D_{m+i}.

5. Simplification: If there is a D_k so that $S_i = D_k$

It can happen that for some $S_i \in \{S_1, \ldots, S_{\ell+1}\}$ there is a corresponding $D_k \in \{D_1, \ldots, D_m\}$ so that $D_k = S_i$. In this case, we propose to replace the node S_i by the corresponding node D_k as shown in Figure 6.13.

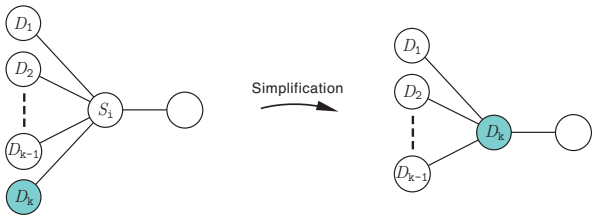

Figure 6.13: Replacing Node S_i if there is a D_k so that $S_i = D_k$.

Example 6.4 Simplification of the Join Tree Let us consider again the join tree of Figure 6.3. It was obtained from the fusion algorithm and consists of nine nodes. Figure 6.14 shows the join trees which are obtained if the proposed simplifications are performed one after another. Eventually, after the application of the fifth simplification, the join tree on the right of Figure 6.14 is obtained. It is composed of only three nodes and is obviously much simpler than the original join tree. ⊖

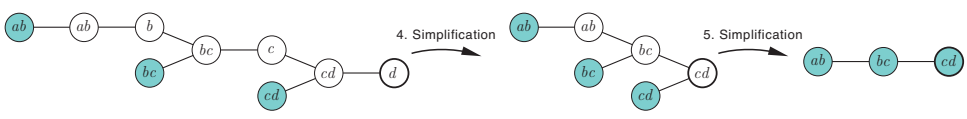

Figure 6.14: Simplification of the Join Tree.

Some Remarks

If we were to use join trees for the purpose of **visualizing computations** or **showing the structure** of a probabilistic argumentation system, it is much better to use the simplified join tree instead of the original join tree. Compare for example the join tree of Figure 6.3 and its simplified counterpart shown on the right side of Figure 6.14. The original join tree consists of nine nodes, whereas the simplified join tree has only three nodes.

A reason against the removal of unnecessary nodes is that a query of which the domain is equal to a label of a removed node can be answered faster. Nevertheless, in our opinion, the gain concerning memory space justifies the removal. In addition, the five different kinds of simplifications can be performed very easily and do not need much computational effort. However, concerning the number of operations (combination/marginalization) which have to be performed, the original and the simplified join tree are often the same. This is due to the fact that only the first and the third simplification change the number of operations.

The third simplification, that is direct marginalization, allows to adapt the heuristics for determinating a variable elimination sequence. If it is performed, then leaf variables can only exist at the start of the elimination process. If one or several leaf variables result from the elimination of a variable, then all these variables are eliminated together in one step. Therefore, once the last leaf variable is eliminated, there is no need to look for leaf variables.

6.6 The "Communication Line" Example

In the previous chapter we have shown how $(\vartheta_1 \otimes \cdots \otimes \vartheta_m)^{\downarrow\{x_0,x_m\}}$ is computed by the fusion algorithm using the elimination sequence $\langle x_1, \ldots, x_{m-1} \rangle$. In the following, we will come back to the computation of this marginal, but we will look at it from the perspective of join trees.

Example 6.5 Communication Line (continued) The potentials $\vartheta_1, \ldots, \vartheta_m$ constructed for the $2m$ rules are defined on domains $\{x_0, x_1\}, \ldots, \{x_{m-1}, x_m\}$, thus $d(\vartheta_i) = \{x_{i-1}, x_i\}$. The corresponding hypergraph $\mathcal{H} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \{x_0, \ldots, x_m\}$ and $\mathbf{E} = \{\{x_0, x_1\}, \ldots, \{x_{m-1}, x_m\}\}$ is shown in Figure 6.15.

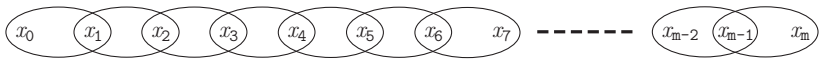

Figure 6.15: Hypergraph for the "Communication Line" Example.

$Cl(x_k, \mathcal{H}) = \{x_{k-1}, x_k, x_{k+1}\}$ and $|FI(x_k, \mathcal{H})| = 1$ for all $x_k \in \{x_1, \ldots, x_{m-1}\}$. Therefore, the two heuristics *"OSLA – Smallest Clique"* and *"OSLA – Fewest Fill-ins"* select an arbitrary variable from $\{x_1, \ldots, x_{m-1}\}$ as the variable to be eliminated first. Whatever variable is eliminated, the size of the closure and

also the fill-in number of the remaining variables do not change. Thus, both heuristics generate one of the $(m-1)!$ possible elimination sequences.

The situation is the same for the heuristic *"OSLA – Smallest Clique, Fewest Focal Sets"*. At the start of the elimination process, $|\Theta_{Cl(x_k,\mathcal{H})}| = 8$ and $\prod_{\vartheta\in\Phi_{i-1}(x_k)} |FS(\vartheta)| = 4$ for all $x_k \in \{x_1,\ldots,x_{m-1}\}$. Whatever variable is eliminated, these reference numbers do not change. In any case, the rank of the join tree corresponding to the generated elimination sequence is 3.

Suppose now that the elimination sequence $\langle x_1,\ldots,x_{m-1}\rangle$ is generated. The corresponding join tree obtained from applying the fusion algorithm is shown in Figure 6.16. $(2m-1)$ of its nodes are of size 2 and $(m-1)$ nodes are of size 3. The potentials $\vartheta_1,\ldots,\vartheta_m$ are distributed on the grey nodes labeled as $\{x_0,x_1\},\ldots,\{x_{m-1},x_m\}$.

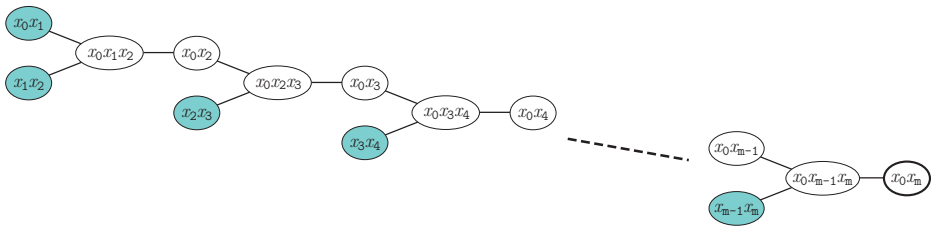

Figure 6.16: Join Tree for the Elimination Sequence $\langle x_1,\ldots,x_{m-1}\rangle$.

If the proposed simplifications are performed, $(m-1)$ nodes of size 2 can be removed. The corresponding simplified join tree is shown in Figure 6.17.

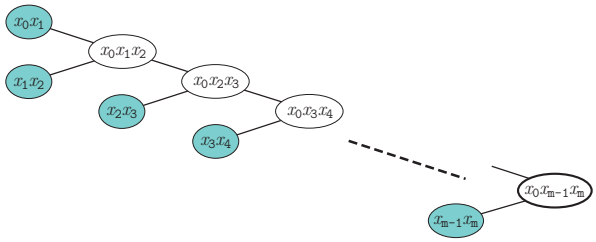

Figure 6.17: Simplified Join Tree.

The computation of $(\vartheta_1\otimes\cdots\otimes\vartheta_m)^{\downarrow\{x_0,x_m\}}$ then corresponds to a propagation towards the root node $\{x_0,x_{m-1},x_m\}$, followed by an additional marginalization. First, $\sigma_1 = \vartheta_1\otimes\vartheta_2$ is computed on the node $\{x_0,x_1,x_2\}$. Then, $\vartheta_{m+1} = \sigma_1^{\downarrow\{x_0,x_2\}}$ is computed and sent to the node $\{x_0,x_2,x_3\}$, where the computation of $\sigma_2 = \vartheta_3\otimes\vartheta_{m+1}$ is done. Proceeding further, the marginal $(\vartheta_1\otimes\cdots\otimes\vartheta_m)^{\downarrow\{x_0,x_{m-1},x_m\}}$ is obtained on the root node. Finally, an additional marginalization is required in order to obtain $(\vartheta_1\otimes\cdots\otimes\vartheta_m)^{\downarrow\{x_0,x_m\}}$. $\ominus$

7
Computing Marginals in Join Trees

The fusion algorithm presented in Chapter 5 eliminates one variable after another. In that way, the marginal for any subset can be computed. The marginal for several subsets of variables could be obtained through repeated application of the fusion algorithm. But this is not very efficient because there would be much unnecessary duplication of effort.

The solution to this problem is given by **join trees**, where nodes can receive and send messages to their neighbor nodes. By storing intermediate results, the computation of several marginals is improved significantly.

The change of viewpoint, from eliminating variables to message passing in a join tree, allows also a much richer understanding. A join tree can easily be visualized graphically and computations can then be represented by arrows between connected nodes.

There are four different architectures which can be used to compute marginals in join trees. For each one of them there is an **inward propagation phase** and an **outward propagation phase**. The main difference of the architectures is mostly the way in which messages are computed during the outward propagation phase.

In this chapter, we will take a closer look at propagation in join trees. Before describing the four architectures we will start with a description of the common ground, consisting of an inward and an outward propagation phase. Finally, we will argue that the **Shenoy-Shafer architecture**, combined with **binary join trees**, is more appropriate for the propagation of multivariate Dempster-Shafer belief functions than the other architectures.

7.1 Computing one Marginal

In the previous section we have shown that a join tree can be obtained from the process of eliminating one variable after another through the fusion algorithm.

Suppose that this join tree is $\mathcal{JT} = (\mathbf{N}, \mathbf{E})$ and is given by the set of nodes $\mathbf{N} = \{N_1, \ldots, N_n\}$ and a set of edges $\mathbf{E}$ between these nodes. One of the nodes is the root node and is denoted as N_r. In addition, every node $N_i \in \mathbf{N}$ contains a potential φ_i on domain $D_i = d(\varphi_i)$.

The process of eliminating one variable after another translates into an inward propagation phase toward the root node. As soon as a node has received a message from every outward neighbor, it is able to compute a message to its inward neighbor. The inward propagation phase is finished when the root node has received a message from each of its outward neighbors. The marginal on the root node can then be computed.

Example 7.1 The Inward Propagation Phase The inward propagation phase for a join tree with root node N_1 is shown in Figure 7.1. In step 1, only leave nodes can send messages. Then, in step 2, nodes N_3, N_4 and N_5 have received the messages from their outward neighbors, so, each of them can compute its message to the root node N_1. Finally, in step 3, the root node has received a message from every neighbor node and can therefore compute its marginal.

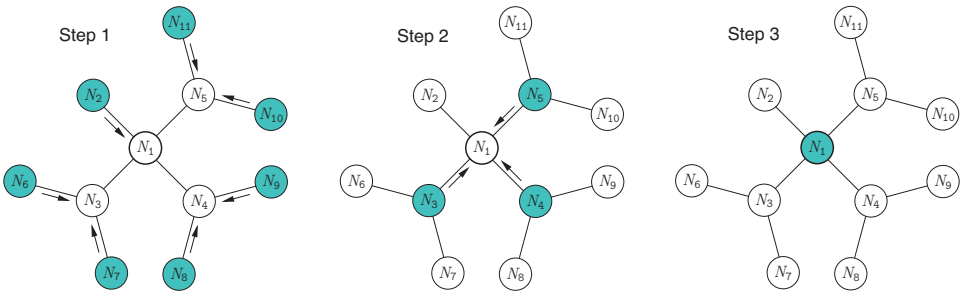

Figure 7.1: The Inward Propagation Phase.

⊖

More formally, suppose that an arbitrary node N_{k_0} has outward neighbors $N_{k_1}, \ldots, N_{k_{m-1}}$ and inward neighbor N_{k_m}. During the inward propagation phase, N_{k_0} receives the messages $\varphi_{k_1 k_0}, \ldots, \varphi_{k_{m-1} k_0}$ from $N_{k_1}, \ldots, N_{k_{m-1}}$. The inward message $\varphi_{k_0 k_m}$ from N_{k_0} to N_{k_m} is then computed as

$$\varphi_{k_0 k_m} = \left(\varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \cdots \otimes \varphi_{k_{m-1} k_0}\right)^{\downarrow D_{k_0} \cap D_{k_m}}. \tag{7.1}$$

If node N_{k_0} is the root node, there is no inward neighbor and, therefore, no inward message is computed.

In order to understand the messages which are sent during the inward propagation phase, let us define for two neighbor nodes N_i and N_j

$$\Phi_{ij} = \{\varphi_k : N_j \notin Path(N_k, N_i)\}, \tag{7.2}$$

$$\phi_{ij} = \otimes \Phi_{ij}. \tag{7.3}$$

$Path(N_k, N_i)$ denotes the set of nodes on the (undirected) path between N_k and N_i. The set Φ_{ij} represents the set of potentials contained in the subtree rooted at N_i. It is always $\varphi_i \in \Phi_{ij}$ and

$$\Phi_{ij} \cup \Phi_{ji} = \{\varphi_1, \ldots, \varphi_n\}, \tag{7.4}$$
$$\Phi_{ij} \cap \Phi_{ji} = \emptyset. \tag{7.5}$$

In addition, if a node N_{k_0} has outward neighbors $N_{k_1}, \ldots, N_{k_{m-1}}$ and inward neighbor N_{k_m}, then

$$\Phi_{k_0 k_m} = \{\varphi_{k_0}\} \cup \Phi_{k_1 k_0} \cup \cdots \cup \Phi_{k_{m-1} k_0}. \tag{7.6}$$

The following theorem can be formulated, using the definition of ϕ_{ij}.

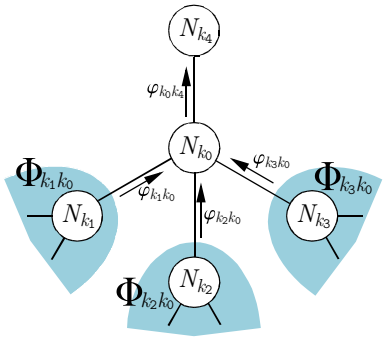

Figure 7.2: Messages during Inward Propagation.

Theorem 7.1 *Suppose a node N_{k_0} has outward neighbors $N_{k_1}, \ldots, N_{k_{m-1}}$ and inward neighbor N_{k_m}. During the inward propagation phase, N_{k_0} combines its potential φ_{k_0} with the incoming messages $\varphi_{k_1 k_0}, \ldots, \varphi_{k_{m-1} k_0}$. Then*

$$\phi_{k_0 k_m}^{\downarrow D_{k_0}} = \varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \cdots \otimes \varphi_{k_{m-1} k_0} \tag{7.7}$$

and the message $\varphi_{k_0 k_m}$ which is sent to N_{k_m} is given by

$$\varphi_{k_0 k_m} = \phi_{k_0 k_m}^{\downarrow D_{k_0} \cap D_{k_m}}. \tag{7.8}$$

Proof See Appendix, page 162. □

The message $\varphi_{k_0 k_m}$ from N_{k_0} to its inward neighbor N_{k_m} is therefore nothing else than the combination of the potentials which are contained in the subtree rooted at N_{k_0} and marginalized to the intersection $D_{k_0} \cap D_{k_m}$.

If N_{k_0} is the root node, it has no inward neighbor and no message is computed. Instead, by combining its potential φ_{k_0} with the incoming messages $\varphi_{k_1 k_0}, \ldots, \varphi_{k_{m-1} k_0}$ the marginal of the root node can be computed.

Theorem 7.2 *Suppose that N_{k_0} is the root node and has outward neighbors $N_{k_1}, \ldots, N_{k_{m-1}}$. During the inward propagation phase, N_{k_0} receives the messages $\varphi_{k_1 k_0}, \ldots, \varphi_{k_{m-1} k_0}$. Then*

$$(\varphi_1 \otimes \cdots \otimes \varphi_n)^{\downarrow D_{k_0}} \quad = \quad \varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \cdots \otimes \varphi_{k_{m-1} k_0}. \tag{7.9}$$

Proof See Appendix, page 163. □

Note that every node of a join tree could be the root node. But in practice, the root node is often chosen in such a way that a given query can be answered by computing the marginal of the root node.

7.2 Computing all Marginals

Suppose that the marginal of the root node has already been obtained after a first inward propagation phase and now the marginals of other nodes have to be computed. This could be done by repeatedly changing the root node followed by an inward propagation phase toward the new root node. But this is not efficient, because it would be unnecessary repetition of effort.

For example, in Figure 7.1 the marginal of N_2 could be obtained after an additional inward propagation phase with N_2 as new root node. However, the only difference between this inward propagation phase and the previous one with root node N_1 is, that N_2 now receives a message from N_1 instead of sending a message to N_1.

In general, two inward propagation phases differ only in the direction of the messages on the path between the two root nodes. To avoid unnecessary repetition of effort, previously performed computations have to be taken into consideration. If all messages are stored during the first inward propagation phase, only the messages on the path between the two root nodes have to be computed. This is clearly much more efficient than performing an additional complete inward propagation phase.

A join tree is an appropriate data structure for computing several marginals. The trick is to perform a first inward propagation phase during which all messages are stored. Then, the marginal of every node can be obtained by an outward propagation phase. First, only the root node can send messages. Then, as soon as a node has received a message from its inward neighbor, it can also compute and send messages to its outward neighbors. The outward propagation phase is finished when all leaf nodes have received a message.

Example 7.2 The Outward Propagation Phase The outward propagation phase for a join tree with root node N_1 is shown in Figure 7.3. In step 1, only the root node can send messages to its outward neighbors N_2, N_3, N_4, and N_5, respectively. Then, in step 2, these four nodes can compute their marginal if necessary. In addition, nodes N_3, N_4 and N_5, respectively, compute a message

to their corresponding outward neighbors. Finally, in step 3, every leaf node has received a message from its inward neighbor and can compute the marginal if necessary.

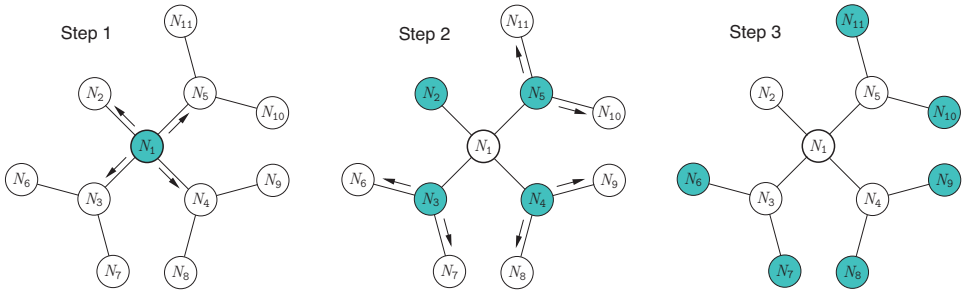

Figure 7.3: The Outward Propagation Phase.

⊖

Note, often it is not advisable to perform a complete outward propagation. Instead, it is better to perform only a partial outward propagation to the set of nodes allowing to answer the given queries.

7.3 Different Architectures

In the following, four different architectures are presented in detail. These architectures are:

- the Shenoy-Shafer architecture
- the Lauritzen-Spiegelhalter architecture
- the Hugin architecture
- the Fast-Division architecture

For each of these architectures we will show what is computed during the inward and the outward propagation phase. They differ from each other mainly by the way in which the outward messages are computed.

In order to describe each architecture we will suppose in the following that a join tree is already given and that one of its nodes is the root node. N_{k_0} will be an arbitrary node with outward neighbors $N_{k_1}, \ldots, N_{k_{m-1}}$ and inward neighbor N_{k_m}. In addition, for two neighbor nodes N_i and N_j with labels D_i and D_j, the set of variables S_{ij} is the intersection $D_i \cap D_j$. Finally, a message from node N_{k_i} to N_{k_j} will be denoted as $\varphi_{k_i k_j}$.

Finally, we will argue that the **Shenoy-Shafer architecture** is most suited for the propagation of multivariate Dempster-Shafer belief functions. If it is

used together with **binary join trees**, it should be the first choice for the propagation of multivariate Dempster-Shafer belief functions.

7.3.1 The Shenoy-Shafer Architecture

The Shenoy-Shafer architecture (SS architecture) was first described in (Shenoy & Shafer, 1990). It is a general architecture, because there is no difference between the inward and the outward propagation phase. The outward messages are computed in the same way as the inward messages. A significant improvement to the SS architecture was the introduction of binary join trees (Shenoy, 1997). The concept of binary join trees will be explained in Section 7.5.

The Inward Propagation Phase
A node N_{k_0}, different from the root node, computes the inward message $\varphi_{k_0k_m}$ to its inward neighbor N_{k_m} as

$$\varphi_{k_0k_m} \quad = \quad (\varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m-1\}))^{\downarrow S_{k_0k_m}} . \tag{7.10}$$

The inward propagation phase is finished when the root node has received a message from each of its outward neighbors.

The Outward Propagation Phase
A node N_{k_0}, which is not a leaf node, can compute the outward message $\varphi_{k_0k_j}$ to its outward neighbor N_{k_j} as

$$\varphi_{k_0k_j} \quad = \quad (\varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m, \ell \neq j\}))^{\downarrow S_{k_0k_j}} . \tag{7.11}$$

The root node can compute and send messages right away. The outward propagation phase is finished when every leaf node has received a message.

Computing Marginals
Every node N_{k_0} can compute the potential φ''_{k_0} as soon as it has received a message from each of its neighbor nodes. It is computed as

$$\varphi''_{k_0} \quad = \quad \varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m\}) . \tag{7.12}$$

The following theorem then holds:

Theorem 7.3 *Suppose that N_{k_0} is an arbitrary node and suppose, in addition, that φ''_{k_0} is computed as above. Then,*

$$\varphi''_{k_0} \quad = \quad \varphi^{\downarrow D_{k_0}} . \tag{7.13}$$

Proof See Appendix, page 164. □

Therefore, φ''_{k_0} is equivalent to $\varphi^{\downarrow D_{k_0}}$, that is, to the joint potential marginalized to D_{k_0}. The marginal of the root node can already be computed after the inward propagation phase. However, note that it is advisable to compute the marginals only for those nodes for which there is a query to answer.

7.3.2 The Lauritzen-Spiegelhalter Architecture

The Lauritzen-Spiegelhalter architecture (LS architecture) was first described in (Lauritzen & Spiegelhalter, 1988) and is very popular in the field of *Bayesian Networks* (Pearl, 1988). It could also be used for multivariate Dempster-Shafer belief functions, but this is not very common. The main difference to the SS architecture is that the division operation is used during the inward propagation phase.

The Inward Propagation Phase
A node N_{k_0}, different from the root node, computes the inward message $\varphi_{k_0k_m}$ to its inward neighbor N_{k_m} in two steps as

$$\varphi'_{k_0} = \varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m-1\}), \tag{7.14}$$

$$\varphi_{k_0k_m} = \varphi'^{\downarrow S_{k_0k_m}}_{k_0}. \tag{7.15}$$

Then, ψ'_{k_0} can be computed as

$$\psi'_{k_0} = \frac{\varphi'_{k_0}}{\varphi_{k_0k_m}}. \tag{7.16}$$

ψ'_{k_0} will be used later during the outward propagation phase. However, note that it is not a valid potential in general. The inward propagation phase is finished when the root node has received a message from every neighbor node. Suppose now that N_{k_0} is the root node. It can then compute

$$\varphi''_{k_0} = \varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell < m\}). \tag{7.17}$$

The Outward Propagation Phase
Suppose that N_{k_0} is the root node. It computes the outward message $\varphi_{k_0k_j}$ to an outward neighbor N_{k_j} as

$$\varphi_{k_0k_j} = \varphi''^{\downarrow S_{k_0k_j}}_{k_0}. \tag{7.18}$$

Every other node N_{k_0} can compute a potential φ''_{k_0} as soon as it has received a message $\varphi_{k_mk_0}$ from its inward neighbor as

$$\varphi''_{k_0} = \psi'_{k_0} \otimes \varphi_{k_mk_0}. \tag{7.19}$$

If N_{k_0} is not a leaf node, it has to compute a message for each of its outward neighbors. The outward message $\varphi_{k_0k_j}$ to an outward neighbor N_{k_j} is then computed as in Equation 7.18 above.

Computing Marginals
The marginals are already computed during the propagation because the potential φ''_{k_0} computed on node N_{k_0} is equal to $\varphi^{\downarrow D_{k_0}}$.

7.3.3 The Hugin Architecture

The Hugin architecture (Jensen *et al.*, 1990a; Jensen *et al.*, 1990b) was obtained by modifying the LS architecture and is very popular in the field of *Bayesian Networks* (Pearl, 1988). In (Lauritzen & Jensen, 1997) it was shown that it can also be applied to more general theories, including the Dempster-Shafer theory.

In the Hugin architecture the division operation is performed during the outward propagation phase on nodes called *separators*. A separator is a special node, which connects two neighbor nodes and of which the label is the intersection of the labels of these two nodes. Figure 7.4 shows a join tree used for the Hugin architecture. The separators are represented as rectangles.

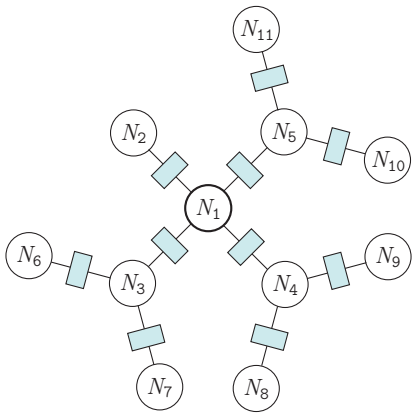

Figure 7.4: A Join Tree with Separators.

The Inward Propagation Phase
A node N_{k_0}, different from the root node, computes the inward message $\varphi_{k_0k_m}$ to its inward neighbor N_{k_m} in two steps as

$$\varphi'_{k_0} = \varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m-1\}), \tag{7.20}$$

$$\varphi_{k_0k_m} = \varphi'^{\downarrow S_{k_0k_m}}_{k_0}. \tag{7.21}$$

φ'_{k_0} is stored on the node N_{k_0} and $\varphi_{k_0k_m}$ is stored on the separator between N_{k_0} and N_{k_m}. The inward propagation phase is finished when the root node has received a message from every neighbor node. Suppose now that N_{k_0} is the root node. It can then compute

$$\varphi''_{k_0} = \varphi_{k_0} \otimes (\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m-1\}). \tag{7.22}$$

The Outward Propagation Phase
Suppose that N_{k_0} is the root node. The outward message $\varphi_{k_0k_j}$ to the separator between N_{k_0} and N_{k_j} is computed as

$$\varphi_{k_0k_j} = \varphi''^{\downarrow S_{k_0k_j}}_{k_0}. \tag{7.23}$$

Then, the message $\tilde{\varphi}_{k_0k_j}$ to N_{k_j} is computed on the separator as

$$\tilde{\varphi}_{k_0k_j} = \frac{\varphi_{k_0k_j}}{\varphi_{k_jk_0}}. \tag{7.24}$$

$\varphi_{k_jk_0}$ was stored on the separator during the inward propagation phase. However, note that $\tilde{\varphi}_{k_0k_j}$ is not a valid potential in general. Every other node N_{k_0} can compute a potential φ''_{k_0} when it has received the message $\tilde{\varphi}_{k_mk_0}$ from the separator between itself and its inward neighbor N_{k_m} as

$$\varphi''_{k_0} = \varphi'_{k_0} \otimes \tilde{\varphi}_{k_mk_0}. \tag{7.25}$$

If N_{k_0} is not a leaf node, it has to compute a message for each of its outward neighbors. The message $\tilde{\varphi}_{k_0k_j}$ to an outward neighbor N_{k_j} is then computed in two steps as shown in Equation 7.23 and Equation 7.24.

Computing Marginals
The marginals are already computed during the propagation because the potential φ''_{k_0} computed on every node N_{k_0} is equal to $\varphi^{\downarrow D_{k_0}}$.

7.3.4 The Fast-Division Architecture

The Fast-Division architecture (FD architecture) was first described in (Bissig *et al.*, 1997) and resulted from a diploma thesis (Bissig, 1996) carried out at the University of Fribourg. At first sight it is similar to the LS and Hugin architectures, but thanks to a useful property it is well suited for the propagation of multivariate Dempster-Shafer belief functions.

The Inward Propagation Phase
A node N_{k_0}, different from the root node, computes the inward message $\varphi_{k_0k_m}$ to its inward neighbor N_{k_m} in two steps as

$$\varphi'_{k_0} = \varphi_{k_0} \otimes \left(\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell \leq m-1\}\right), \tag{7.26}$$

$$\varphi_{k_0k_m} = \varphi'^{\downarrow S_{k_0k_m}}_{k_0}. \tag{7.27}$$

φ'_{k_0} and $\varphi_{k_0k_m}$ have to be stored because they will be used during the outward propagation phase. The inward propagation phase is finished when the root node has received a message from every neighbor node. Suppose now that N_{k_0} is the root node. It can then compute

$$\varphi''_{k_0} = \varphi_{k_0} \otimes \left(\otimes \{\varphi_{k_\ell k_0} : 1 \leq \ell < m\}\right). \tag{7.28}$$

The Outward Propagation Phase
Suppose that N_{k_0} is the root node. It computes the outward message $\varphi_{k_0k_j}$ to an outward neighbor N_{k_j} as

$$\varphi_{k_0k_j} = \varphi''^{\downarrow S_{k_0k_j}}_{k_0}. \tag{7.29}$$

Every other node N_{k_0} can compute a potential φ''_{k_0} as soon as it has received the message $\varphi_{k_m k_0}$ from its inward neighbor as

$$\varphi''_{k_0} = \frac{\varphi'_{k_0} \otimes \varphi_{k_m k_0}}{\varphi_{k_0 k_m}}. \tag{7.30}$$

If N_{k_0} is not a leaf node, it has to compute a message for each of its outward neighbors. The outward message $\varphi_{k_0 k_j}$ to an outward neighbor N_{k_j} is then computed as in Equation 7.29 above.

Speed up Computations
The potential φ''_{k_0} in Equation 7.30 can be computed efficiently thanks to the following property (Bissig, 1996):

$$FS(\varphi''_{k_0}) \subseteq FS(\varphi'_{k_0} \otimes \varphi_{k_m k_0}) \tag{7.31}$$

Therefore, $[\varphi''_{k_0}(A)]_m = 0$ for all $A \notin FS(\varphi'_{k_0} \otimes \varphi_{k_m k_0})$.

Computing Marginals
The marginals are already computed during the propagation because the potential φ''_{k_0} computed on every node N_{k_0} is equal to $\varphi^{\downarrow D_{k_0}}$.

7.4 Comparison and Discussion

In order to compare the four architectures, let us define the potential ϕ_{ij} for two neighbor nodes N_i and N_j like in (7.3), as the combination of the potentials contained in the subtree rooted at N_i. Using this definition it can be verified that for all four architectures

$$\varphi_{k_0 k_m} = \phi_{k_0 k_m}^{\downarrow S_{k_0 k_m}}. \tag{7.32}$$

By way of contrast, the outward message $\varphi_{k_m k_0}$ is not the same in all four architectures. In the SS architecture

$$\varphi_{k_m k_0} = \phi_{k_m k_0}^{\downarrow S_{k_0 k_m}} \tag{7.33}$$

whereas in the LL, Hugin and FD architectures

$$\varphi_{k_m k_0} = \phi_{k_0 k_m}^{\downarrow S_{k_0 k_m}} \otimes \phi_{k_m k_0}^{\downarrow S_{k_0 k_m}}. \tag{7.34}$$

The inward message $\varphi_{k_0 k_m}$ is therefore a factor of the outward message. The LS, Hugin and FD architectures use the division operation to remove $\varphi_{k_0 k_m}$. The marginal $\varphi^{\downarrow D_{k_0}}$ is computed as

$$\left(\frac{\varphi'_{k_0}}{\varphi_{k_0 k_m}}\right) \otimes \varphi_{k_m k_0}, \qquad \text{(Lauritzen-Spiegelhalter)}$$

$$\varphi'_{k_0} \otimes \left(\frac{\varphi_{k_m k_0}}{\varphi_{k_0 k_m}}\right), \qquad \text{(Hugin)}$$

$$\frac{\left(\varphi'_{k_0} \otimes \varphi_{k_m k_0}\right)}{\varphi_{k_0 k_m}}, \qquad \text{(Fast-Division)}$$

which corresponds to the three possibilities to remove $\varphi_{k_0k_m}$. Although the final result of the computations is always a valid potential, this is in general not the case for the intermediate result of the division operation. Similarly, consider the computation of $(\frac{4\cdot3}{6})$ by $(\frac{4}{6})\cdot 3$ or by $4\cdot(\frac{3}{6})$. The result is always the integer 2, but the intermediate results are not integers.

The advantage of the LS, Hugin, and FD architectures is that they require less operations than the SS architecture (Lepar & Shenoy, 1998). For a node with m neighbor nodes, the minimal number of combinations in the SS architecture is $3(m-1)$, when a binary join tree is used (see Section 7.5). By way of contrast, the other architectures use only m combinations and m divisions for the root node and m combinations and $m-1$ divisions for other nodes. Table 7.1 shows that there are less operations used in the LS, Hugin, and FD architectures than in the SS architecture for nodes with 3 or more neighbor nodes.

	LS, Hugin, FD			Shenoy-Shafer	
Neighbors	Combinations	Divisions	Total	Combinations	Total
1	1	0	1	1	1
2	2	1	3	3	3
3	3	2	5	6	6
4	4	3	7	9	9
5	5	4	9	12	12
6	6	5	11	15	15

Table 7.1: The Number of Combinations and Divisions

Nevertheless, the LS and Hugin architectures are not practicable for multivariate Dempster-Shafer belief functions. This is due to the fact that the division of two potentials on domain D involves computing a value for every $A \subseteq \Theta_D$, which is not even possible for relatively small domains D (see Section 3.6). However, practical experiences show that potentials often contain only a few focal sets, even if their state space is very large. The LS and Hugin architectures do not profit from these experiences. The time used by these architectures for computations depends on the *structure* of the join tree. In contrast to this, the time used by the SS and FD architectures depend on the *information* contained in the join tree.

An important, yet undiscussed aspect is the question of which representation (see Section 3.2) to use for the computations. For the SS architecture, it is undoubtedly the best to use the mass function representation, because only combination and marginalization are needed. For the FD architecture, it is not possible to use one representation alone, because division is defined for commonality functions only and marginalization is not defined for commonality functions. Therefore, we propose to use mainly the mass function representation

and the commonality function representation only for the division operation. This is shown in Figure 7.5 below.

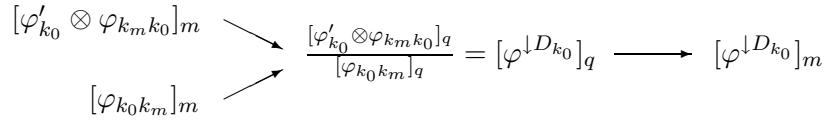

Figure 7.5: Changing the Representation in the FD Architecture.

Note that we propose to use the algorithms of Section 3.6 for the transformation from mass to commonality functions and from commonality to mass functions. However, transformations are often computationally expensive. This is the reason why the SS architecture is more efficient than the FD architecture. In addition, it is simpler and easier to understand. Therefore, the Shenoy-Shafer architecture, together with binary join trees (see Section 7.5), is best suited for the propagation of multivariate Dempster-Shafer belief functions.

A comparison of the LS, the Hugin, and the SS architectures for probability distributions can be found in (Lepar & Shenoy, 1998). The authors conclude that for probability distributions, the SS architecture is on an average more efficient than the LS and Hugin architectures.

7.5 Binary Join Trees

The Shenoy-Shafer architecture as it is presented in Subsection 7.3.1 is not very efficient when multiple marginals have to be computed. The problem is that many redundant combinations would be recomputed. Therefore, Shenoy proposes binary join trees (BJT) (Shenoy, 1997) to eliminate this drawback.

To illustrate the problem, let us consider first the join tree on the left side of Figure 7.6. The node N_{k_0} has four neighbor nodes $N_{k_1}, \ldots, N_{k_4}$. In the SS architecture, the messages $\varphi_{k_0 k_1}, \ldots \varphi_{k_0 k_4}$ would be computed as

$$\begin{aligned}
\varphi_{k_0 k_1} &= (\varphi_{k_0} \otimes \varphi_{k_2 k_0} \otimes \varphi_{k_3 k_0} \otimes \varphi_{k_4 k_0})^{\downarrow S_{k_0 k_1}} \\
\varphi_{k_0 k_2} &= (\varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \varphi_{k_3 k_0} \otimes \varphi_{k_4 k_0})^{\downarrow S_{k_0 k_2}} \\
\varphi_{k_0 k_3} &= (\varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \varphi_{k_2 k_0} \otimes \varphi_{k_4 k_0})^{\downarrow S_{k_0 k_3}} \\
\varphi_{k_0 k_4} &= (\varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \varphi_{k_2 k_0} \otimes \varphi_{k_3 k_0})^{\downarrow S_{k_0 k_4}}
\end{aligned}$$

involving twelve combinations. In addition, the marginal $\varphi^{\downarrow D_{k_0}}$ would be computed as

$$\varphi^{\downarrow D_{k_0}} = \varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \varphi_{k_2 k_0} \otimes \varphi_{k_3 k_0} \otimes \varphi_{k_4 k_0}$$

involving four additional combinations. In that way, a node with m neighbor nodes would need $m(m-1)$ combinations for the computation of the messages and m combinations for the marginal. Therefore, m^2 combinations would be needed in total.

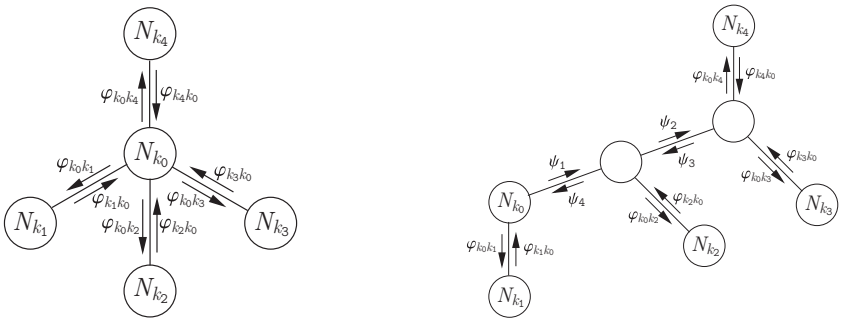

Figure 7.6: Ordinary Join Tree and corresponding Binary Join Tree.

Now, let us consider the join tree on the right side of Figure 7.6. It was constructed from the join tree on the left through addition of two nodes. Every node of the join tree has now three neighbor nodes at most. A join tree with this property is called a **binary join tree**. The messages $\psi_1, \ldots, \psi_4$ are computed by

$$\begin{aligned}
\psi_1 &= \varphi_{k_0} \otimes \varphi_{k_1k_0} \\
\psi_2 &= \psi_1 \otimes \varphi_{k_2k_0} \\
\psi_3 &= \varphi_{k_3k_0} \otimes \varphi_{k_4k_0} \\
\psi_4 &= \psi_3 \otimes \varphi_{k_2k_0}
\end{aligned}$$

involving four combinations. The messages $\varphi_{k_0k_1}, \ldots, \varphi_{k_0k_4}$ can then be obtained as

$$\begin{aligned}
\varphi_{k_0k_1} &= (\varphi_{k_0} \otimes \psi_4)^{\downarrow S_{k_0k_1}} \\
\varphi_{k_0k_2} &= (\psi_1 \otimes \psi_3)^{\downarrow S_{k_0k_2}} \\
\varphi_{k_0k_3} &= (\psi_2 \otimes \varphi_{k_4k0})^{\downarrow S_{k_0k_3}} \\
\varphi_{k_0k_4} &= (\psi_2 \otimes \varphi_{k_3k0})^{\downarrow S_{k_0k_4}}
\end{aligned}$$

involving four additional combinations. Finally, the marginal $\varphi^{\downarrow D_{k_0}}$ can be obtained using only one combination. It is

$$\varphi^{\downarrow D_{k_0}} = \psi_1 \otimes \psi_4 = \psi_2 \otimes \psi_3.$$

Every join tree can be transformed into a binary join tree through addition of additional nodes. Then, $3m - 4$ combinations are needed for the computation of all messages for a node with m neighbor nodes. Further, one additional combination is required for the computation of the marginal. Therefore, $3(m - 1)$ combinations are needed in total.

The number of combinations needed for the computation of the messages and the marginal is shown in Table 7.2. Without the use of a binary join tree, a lot of unnecessary combinations would be performed for nodes which have many neighbor nodes.

	SS without BJT			SS with BJT		
Neighbors	Messages	Marginal	Total	Messages	Marginal	Total
1	0	1	1	0	1	1
2	2	2	4	2	1	3
3	6	3	9	5	1	6
4	12	4	16	8	1	9
5	20	5	25	11	1	12
6	30	6	36	14	1	15

Table 7.2: The number of combinations needed

The main idea of binary join trees is to store intermediate results for minimizing the number of combinations. Binary join trees are a significant improvement to the SS architecture. Nevertheless, it is important to realize that binary join trees represent a tradeoff between computing time and memory space. By storing intermediate results, less computing time is needed by the sake of using more memory space. Whereas memory is usually not a problem for small examples, it is often the main problem for large examples.

Special care has to be taken for nodes having a large number of neighbor nodes. For such nodes it can happen that the combination of incoming messages generate huge mass functions even if the incoming messages were relatively small. Nevertheless, it is still possible that every outgoing message is relatively small because a marginalization was performed. In a binary join tree, these huge mass functions representing intermediate results would have been stored and therefore much memory space would have been lost.

7.6 Answering Queries

We have already seen in Chapter 2 that

$$dsp(h, \xi) \quad = \quad \frac{dqs(h, \xi) - dqs(\bot, \xi)}{1 - dqs(\bot, \xi)}. \tag{7.35}$$

Therefore, it is sufficient to consider only queries of the form $dqs(h, \xi)$. The formula h is called the **hypothesis** of the query and ξ represents the knowledge base. In addition, we say that the set of variables $d(h)$, that is the variables which appear in h, is the **domain** of the query. In the following, we will distinguish the cases, where $h \in \mathcal{L}_{\mathcal{V}}$ and where $h \in \mathcal{L}_{\mathcal{A} \cup \mathcal{V}}$. Finally, we will show what to do if a query cannot be answered on a given join tree.

7.6.1 Query with hypothesis $h \in \mathcal{L}_{\mathcal{V}}$

If $h \in \mathcal{L}_{\mathcal{V}}$ and $d(h) \subseteq D_{k_0}$, then $dqs(h, \xi)$ can be computed as

$$dqs(h, \xi) \quad = \quad [\varphi^{\downarrow D_{k_0}}(H)]_b \tag{7.36}$$

for $H = N_{D_{k_0}}(h)$, where the potential φ is constructed from ξ. Therefore, a query can be answered on a given join tree if it contains at least one node of which the label is a superset of the domain of the query.

The Shenoy-Shafer architecture computes $\varphi^{\downarrow D_{k_0}}$ by propagating potentials in a join tree. The picture on the left side of Figure 7.7 shows that the node N_{k_0} receives a message from each of its neighbor nodes. If $N_{k_1}, \ldots, N_{k_m}$ are the neighbor nodes of N_{k_0},

$$\varphi^{\downarrow D_{k_0}} \quad = \quad \varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \cdots \otimes \varphi_{k_m k_0}. \tag{7.37}$$

If a query $dqs(h, \xi)$ is already known before a join tree is constructed, then the construction should be organized in such a way that there will be at least one node of which the label is a superset of $d(h)$. One possibility to do this is the use of an elimination sequence consisting of variables in $d(\xi) - d(h)$. The root node of the join tree resulting from Shenoy's fusion algorithm will then have the label $d(h)$. Another possibility is to add a neutral potential ι_h of which the domain is $d(h)$. Every join tree constructed from a set of potentials containing ι_h will then have a node of which the label is a superset of $d(h)$. Therefore, an inward propagation toward this node is then needed.

Of course, the same procedure is also valid if more than one query is known prior to the construction of a join tree. However, sometimes it is not advisable to construct a join tree which allows to answer all these queries because the join tree constructed gets worse with every additional neutral potential added. Therefore, we propose to split up the queries according to a certain strategie into several groups and to build a join tree for each group if there are many queries.

7.6.2 Query with hypothesis $h \in \mathcal{L}_{\mathcal{A} \cup \mathcal{V}}$

The computation of $dqs(h, \xi)$ for $h \in \mathcal{L}_{\mathcal{A} \cup \mathcal{V}}$ is quite similar to the computation of $dqs(h, \xi)$ for $h \in \mathcal{L}_{\mathcal{V}}$ discussed previously. However, in addition to the potential φ_i, it is now required to store on every node N_i of the join tree also the symbolic mass function $\vec{\Sigma}'_i$ from which φ_i was derived.

Suppose in the following that $d_{\mathcal{A}}(h) \subseteq \mathcal{A}$ denotes the set of assumptions occuring in h and $d(h) \subseteq \mathcal{V}$ the set of variables. A query, where $h \in \mathcal{L}_{\mathcal{A} \cup \mathcal{V}}$ can only be answered if there is a node N_{k_0} so that

- $d_{\mathcal{A}}(h) \subseteq d_{\mathcal{A}}(\vec{\Sigma}'_{k_0})$,
- $d(h) \subseteq D_{k_0}$.

If such a node N_{k_0} does not exist, it is always possible to construct a join tree which meets the two above requirements. Therefore, we suppose in the following that such a node N_{k_0} exists. In any case, note that it is not possible that more than one node of a join tree satisfies the two above requirements because we construct join trees in such a way that every assumption $a \in \mathcal{A}$ occurs only in one node. A join tree with this property was called an $\mathcal{A}$-disjoint join tree (see Chapter 4).

The hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$ can always be written as

$$h \quad = \quad h_1 \vee h_2$$

where $h_1 \in \mathcal{L}_{\mathcal{V}}$ and $h_2 \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. Almost always, there are even several possibilities to decompose h in such a way. It is relatively easy to see that

$$dqs(h_1 \vee h_2, \xi) \quad = \quad dqs(h_1, \neg h_2 \wedge \xi) \tag{7.38}$$

because

$$\begin{aligned} QS_{\mathcal{A}}(h_1 \vee h_2, \xi) &= \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \xi \models h_1 \vee h_2\} \\ &= \{\mathbf{s} \in N_{\mathcal{A}} : \mathbf{s} \wedge \neg h_2 \wedge \xi \models h_1\} \;=\; QS_{\mathcal{A}}(h_1, \neg h_2 \wedge \xi). \end{aligned}$$

This allows to compute $dqs(h, \xi)$ as

$$dqs(h, \xi) \quad = \quad [\tilde{\varphi}^{\downarrow D_{k_0}}(H_1)]_b \tag{7.39}$$

for $H_1 = N(h_1)^{\uparrow D_{k_0}}$ and where the potential $\tilde{\varphi}$ corresponds to $\neg h_2 \wedge \xi$. We supposed that $d_{\mathcal{A}}(h) \subseteq d_{\mathcal{A}}(\vec{\Sigma}'_{k_0})$ because $\tilde{\varphi}^{\downarrow D_{k_0}}$ can then be computed similar to $\varphi^{\downarrow D_{k_0}}$. The picture on the right side of Figure 7.7 shows that

$$\tilde{\varphi}^{\downarrow D_{k_0}} \quad = \quad \tilde{\varphi}_{k_0} \otimes \varphi_{k_1 k_0} \otimes \cdots \otimes \varphi_{k_m k_0}. \tag{7.40}$$

Therefore, the only difference is that φ_{k_0} is replaced by the potential $\tilde{\varphi}_{k_0}$, which is constructed from $\neg h_2$ and from the symbolic mass function $\vec{\Sigma}'_{k_0}$.

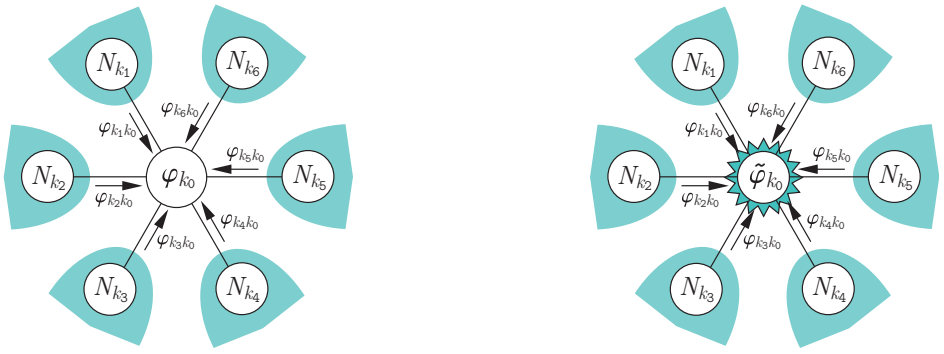

Figure 7.7: Hypothesis $h \in \mathcal{L}_{\mathcal{V}}$ and Hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$.

Finally, let us summarize the steps which have to be taken in order to answer a query of the form $dqs(h, \xi)$, where $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$:

1. Find a node N_{k_0} with label D_{k_0} so that
   - $d(h) \subseteq D_{k_0}$
   - $d_{\mathcal{A}}(h) \subseteq d_{\mathcal{A}}(\vec{\Sigma}'_{k_0})$ for the symbolic mass function $\vec{\Sigma}'_{k_0}$ of φ_{k_0}
2. Transform h into $h = h_1 \vee h_2$, where $h_1 \in \mathcal{L}_{\mathcal{V}}$ and $h_2 \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$
3. Build the potential $\tilde{\varphi}_{k_0}$ from $\vec{\Sigma}'_{k_0}$ and $\neg h_2$
4. Compute $\tilde{\varphi}^{\downarrow D_{k_0}} = \tilde{\varphi}_{k_0} \otimes \varphi_{k_1 k_0} \otimes \cdots \otimes \varphi_{k_m k_0}$
5. Return $dqs(h, \xi) = [\tilde{\varphi}^{\downarrow D_{k_0}}(H_1)]_b$, where $H_1 = N(h_1)^{\uparrow D_{k_0}}$

7.6.3 Modifying the Join Tree

The construction of a join tree is computationally very expensive. Therefore, if a join tree is already given and there is a query which cannot be answered, it is not always a good idea to construct a new join tree. Another possibility is to modify the existing join tree in such a way that the query can be answered after the modifications. Above all, this approach is very interesting if the modifications involve only a small part of the join tree.

In (Xu, 1995), an algorithm is proposed which modifies a given join tree in such a way that it contains a new node afterwards which allows to answer the given query. Therefore, the input values of Xu's algorithm are a join tree $\mathcal{JT} = (\mathbf{N}, \mathbf{E})$ and the domain $d(h)$ of the query. The result of the algorithm is a new join tree containing a node of which the label is a superset of $d(h)$. The pseudo code of the algorithm is as follows:

Pseudo-Algorithm *Modification Method proposed by Xu*

(1) Find a set $b = \{N_{t_1}, \ldots, N_{t_k}\} \subseteq \mathbf{N}$ so that $D_{t_i} \cap d(h) \neq \emptyset$ and $d(h) \subseteq \bigcup_{i=1}^{k} D_{t_i}$

(2) Build the (smallest) set a that contains b and all the nodes on the path among the nodes in b

(3) If $|a| = 1$, then stop

(4) Select $N_i \in a$ so that it has only one neighbor N_j in a

$$\mathcal{N}_j = \{N_k \in a : (N_k, N_j) \in \mathbf{E}\}$$
$$\mathcal{S}_j = \bigcup\{(D_j \cap D_k) : N_k \in a\}$$

(5) Create node N' with label $D' = (D_i \cup D_j) \cap (\mathcal{S}_j \cup d(h))$

(6) $\mathbf{E} = \mathbf{E} \cup \{(N_k, N') : N_k \in \mathcal{N}_j\} \cup \{(N_j, N')\} - \{(N_k, N_j) : N_k \in \mathcal{N}_j\}$
$\mathbf{N} = \mathbf{N} \cup \{N'\}$
$a = a \cup \{N'\} \setminus \{N_i, N_j\}$

(7) Go back to step (3)

Note that the set of nodes a represents a subtree of the original join tree. The modifications affect only this subtree, while the rest of the join tree is not changed. At each pass of the algorithm, there is one node less in a. The set of nodes a represents always a tree and it is therefore always possible to select a node $N_i \in a$ in step (4) which has only one neighbor node N_j in a. Step (5) guarantees that the Markov property is satisfied and that there will be a node of which the label is a superset of $d(h)$. At step (6), the set of nodes of the original join tree is a subset of the set of nodes of the modified join tree.

To illustrate Xu's algorithm, let us look at the join tree on the left side of Figure 7.8. A query with domain $d(h) = \{a, g\}$ cannot be answered directly, because there is no node of which the label is a superset of $d(h)$. Therefore, let us look at how Xu's algorithm solves the problem.

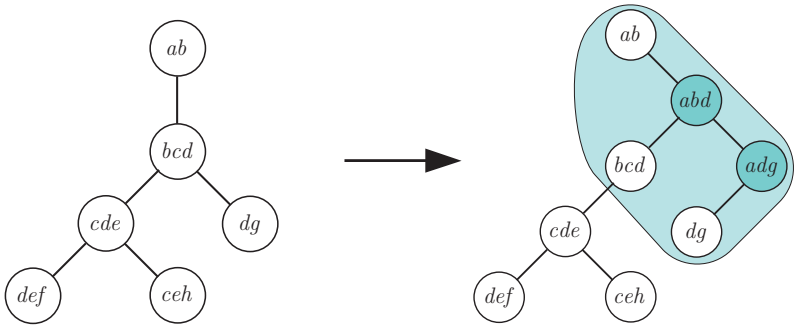

Figure 7.8: The original Join Tree and the modified Join Tree.

Example 7.3 The Modification Method proposed by Xu Let us look again at the join tree on the left side of Figure 7.8. If the domain of the query is $d(h) = \{a, g\}$, then (we denote nodes by their label)

$$\begin{aligned} b &= \{\{a, b\}, \{d, g\}\} \\ a &= \{\{a, b\}, \{b, c, d\}, \{d, g\}\} \end{aligned}$$

are obtained in steps (1) and (2). Because $|a|$ is not equal to 1, we continue with step (4). Here, nodes $\{a, b\}$ as well as $\{d, g\}$ could be chosen. If we choose $N_i = \{a, b\}$, $N_j = \{b, c, d\}$ and, additionally,

$$\begin{aligned} \mathcal{N}_j &= \{\{a, b\}, \{d, g\}\}, \\ \mathcal{S}_j &= \{b, d\}. \end{aligned}$$

The newly created node N' has the label $\{a, b, d\}$. It is connected to the nodes $\{a, b\}$ and $\{b, c, d\}$, whereas the edges from $\{b, c, d\}$ to $\{a, b\}$ and $\{d, g\}$ are removed. Finally, set a consists of two nodes and is given by

$$a = \{\{a, b, d\}, \{d, g\}\}.$$

Repeating steps (3) to (7) with the new set of nodes a and choosing $N_i = \{a, b, d\}$ and $N_j = \{d, g\}$ then gives

$$\begin{aligned} \mathcal{N}_j &= \{\{a, b, d\}\}, \\ \mathcal{S}_j &= \{d\}. \end{aligned}$$

A new node with label $\{a, d, g\}$ is then created. It is connected to $\{a, b, d\}$ and $\{d, g\}$, whereas the edges between $\{a, b, d\}$ and $\{d, g\}$ is removed. Finally, the set a is given by

$$a = \{\{a, d, g\}\}$$

so that the algorithm terminates.

The corresponding join tree obtained by Xu's algorithm is shown on the right side of Figure 7.8. The query with domain $d(h) = \{a, g\}$ can now be answered because one of the two new nodes has the label $\{a, d, g\}$. To answer the query, an inward propagation phase toward the node $\{a, d, g\}$ has to be initiated. However, if the messages were stored in the original join tree, only a partial inward propagation would be needed, because the changes involve only a small part of the original join tree. ⊖

8

An Alternative to Outward Propagation

An inward propagation phase followed by an outward propagation phase corresponds in a way to a **complete compilation** of the given knowledge allowing then to answer queries very quickly. This approach is especially advantageous when there are a lot of queries distributed over all the nodes of a join tree.

In the following, we propose a new method which is very appropriate when there are relatively few queries. It corresponds to a **partial compilation** of the given knowledge. This partial compilation results from an inward propagation phase, where intermediate results are stored. Later, instead of performing a complete outward propagation phase, a partial inward propagation is performed for each query. A description of this new method can also be found in (Lehmann & Haenni, 1999).

In this chapter, we will first explain the basic idea on which the new method is based. Then, we will take a closer look at the new method which consists of a first inward propagation phase followed by a partial inward propagation phase. Finally, we will show how the new method can be improved further.

8.1 The Basic Idea

The new method is based on an alternative way for computing degrees of quasi-support. Suppose that $\mathcal{PAS}_{\mathcal{V}} = \{\xi, \mathcal{V}, \mathcal{A}, \Pi\}$ represents a probabilistic argumentation system. For the knowledge base ξ and every $h \in \mathcal{L}_{\mathcal{V}}$

$$\xi \models h \qquad \Longleftrightarrow \qquad \xi \wedge \neg h \models \bot. \tag{8.1}$$

Using this equivalence, it can be concluded that

$$QS_{\mathcal{A}}(h, \xi) \;=\; QS_{\mathcal{A}}(\bot, \xi \wedge \neg h). \tag{8.2}$$

and therefore it is also

$$dqs(h, \xi) \;=\; dqs(\bot, \xi \wedge \neg h). \tag{8.3}$$

Because computing degree of quasi-support is equivalent to computing unnormalized belief in Dempster–Shafer theory (see Chapter 4)

$$dqs(h,\xi) \quad = \quad [\varphi^{\downarrow D_h}(H)]_b \tag{8.4}$$

for a joint potential $\varphi = \varphi_1 \otimes \cdots \otimes \varphi_n$ which is equivalent to ξ, $D_h = d(h)$ and $H = N_{D_h}(h)$. However, by using Equation 8.3, $dqs(h,\xi)$ can alternatively be computed as

$$dqs(h,\xi) \quad = \quad [(\varphi \otimes \upsilon)(\emptyset)]_b \tag{8.5}$$

where the potential υ corresponds to $\neg h$ and is given by $[\upsilon(H^c)]_m = 1$.

This alternative way of computing $dqs(h,\xi)$ can also be used for the computation of $dsp(h,\xi)$. Traditional methods compute $dsp(h,\xi)$ as

$$dsp(h,\xi) \quad = \quad [\varphi^{\downarrow D_h}(H)]_B \tag{8.6}$$

$$\quad = \quad \frac{[\varphi^{\downarrow D_h}(H)]_b - [\varphi^{\downarrow D_h}(\emptyset)]_b}{1 - [\varphi^{\downarrow D_h}(\emptyset)]_b} \tag{8.7}$$

by computing the marginal $\varphi^{\downarrow D_h}$ on one of the nodes of the join tree. If D_h is not a subset of the label of the root node, an inward and an outward propagation phase is needed for this (see Chapter 7). In contrast, the new method computes $dsp(h,\xi)$ as

$$dsp(h,\xi) \quad = \quad \frac{[(\varphi \otimes \upsilon)^{\downarrow D_r}(\emptyset)]_b - [\varphi^{\downarrow D_r}(\emptyset)]_b}{1 - [\varphi^{\downarrow D_r}(\emptyset)]_b} \tag{8.8}$$

where D_r is the label of the root node.

8.2 The New Method

The new method computes $dsp(h,\xi)$ for a probabilistic argumentation system $\mathcal{PAS}_\mathcal{V} = \{\xi, \mathcal{V}, \mathcal{A}, \Pi\}$ by performing an inward propagation phase followed by a second partial inward propagation.

8.2.1 The Inward Propagation Phase

The inward propagation phase is used to compute the value $c_1 = [\varphi(\emptyset)]_m$ and has to be performed only once even if there are many queries. It is exactly the same as for traditional methods described in Section 7.1. For the reason of completeness it is nevertheless described again in the following.

First, a root node N_r has to be selected. Then, an inward propagation toward N_r is started. Every node N_{k_0} with neighbor nodes $N_{k_1}, \ldots, N_{k_m}$, where N_{k_m} denotes the inward neighbor of N_{k_0} computes

$$\varphi'_{k_0} \quad = \quad \varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \ldots \otimes \varphi_{k_{m-1} k_0} \tag{8.9}$$

as soon as it has received the messages $\varphi_{k_1k_0}, \ldots, \varphi_{k_{m-1}k_0}$ from its outward neighbors. The message $\varphi_{k_0k_m}$ to the inward neighbor N_{k_m} is then computed by an additional marginalization as

$$\varphi_{k_0k_m} \quad = \quad \varphi'^{\downarrow D_{k_0} \cap D_{k_m}}_{k_0}. \tag{8.10}$$

The inward propagation phase is finished as soon as the root node N_r has combined its potential with every incoming message yielding φ'_r which is equal to $\varphi^{\downarrow D_r}$. The value $[\varphi^{\downarrow D_r}(\emptyset)]_m$ is then equal to $[\varphi(\emptyset)]_m$. This value is of particular interest because afterwards it is used for normalization.

8.2.2 The Partial Inward Propagation

For each query, an additional inward propagation phase is required to compute $dqs(h, \xi)$ (which is equal to $[\varphi^{\downarrow D_h}(H)]_b$). In the following, first only the very basic things needed to understand the method are explained. Then, two improvements are given and discussed in detail.

The Basic Approach
First, a potential υ which corresponds to $\neg h$ is constructed in such a way that $d(\upsilon) = d(h)$ and $[\upsilon(H^c)]_m = 1$ for $H = N(h)$. The potential υ is then added to the join tree. For this purpose, there must be at least one node in the join tree of which the label is a superset of $d(\upsilon)$. Suppose that this node is denoted as N_υ. Now, an inward propagation phase can be started. Every node N_{k_0} different from N_υ computes as usually

$$\tilde{\varphi}'_{k_0} \quad = \quad \varphi_{k_0} \otimes \tilde{\varphi}_{k_1k_0} \otimes \ldots \otimes \tilde{\varphi}_{k_{m-1}k_0} \tag{8.11}$$

as soon as it has received the messages $\tilde{\varphi}_{k_1k_0}, \ldots, \tilde{\varphi}_{k_{m-1}k_0}$ from its outward neighbors. In contrast, if N_{k_0} is equal to N_υ, then it computes

$$\tilde{\varphi}'_{k_0} \quad = \quad \varphi_{k_0} \otimes \tilde{\varphi}_{k_1k_0} \otimes \ldots \otimes \tilde{\varphi}_{k_{m-1}k_0} \otimes \upsilon. \tag{8.12}$$

In any case, the inward message from N_{k_0} to N_{k_m} is obtained by an additional marginalization as

$$\tilde{\varphi}_{k_0k_m} \quad = \quad \tilde{\varphi}'^{\downarrow D_{k_0} \cap D_{k_m}}_{k_0}. \tag{8.13}$$

The inward propagation phase is finished as soon as the root node N_r has received all messages and has computed the potential $\tilde{\varphi}''_r$ by combining every incoming message with its potential. It is then

$$\tilde{\varphi}''_r \quad = \quad (\varphi \otimes \upsilon)^{\downarrow D_r}. \tag{8.14}$$

The value $c_2 = [(\varphi \otimes \upsilon)^{\downarrow D_r}(\emptyset)]_m$ is of particular interest. If $c_1 = [\varphi^{\downarrow D_r}(\emptyset)]_m$ is the value obtained after the first inward propagation phase, then the following theorem holds:

Theorem 8.1 *Suppose that $H = N(h)$ and $D_h = d(h)$. In addition, suppose that υ corresponds to $\neg h$. If c_1 and c_2 are defined as above, then*

$$[\varphi^{\downarrow D_h}(H)]_B = \frac{c_2 - c_1}{1 - c_1}. \tag{8.15}$$

Proof See Appendix, page 164. □

This theorem states that only the two values c_1 and c_2 are needed to compute the degree of support of a hypothesis $h \in \mathcal{L}_\mathcal{V}$. The value c_1 is obtained after the first inward propagation phase, whereas c_2 is obtained after the additional inward propagation phase.

1. Improvement: Storing Intermediate Results
By comparing the two inward propagation phases described on the previous pages, it can be seen that unnecessary computations are performed during the second inward propagation phase. For all nodes which are not on the path from N_υ to N_r it is always $\varphi'_{k_0} = \tilde{\varphi}'_{k_0}$. For these nodes, the computation of $\tilde{\varphi}'_{k_0}$ can be avoided if intermediate results are stored during the first inward propagation phase. Only a partial inward propagation from N_υ to the root node N_r is then necessary. This is of course much more efficient than performing a complete inward propagation phase.

2. Improvement: Hypothesis represented by several Potentials
The second improvement allows to answer a larger class of queries. It is based on the fact that the hypothesis $h \in \mathcal{L}_\mathcal{V}$ can always be represented as a disjunction $h = h_1 \vee \cdots \vee h_k$, where for every formula h_i it is $d(h_i) \subseteq d(h)$. Then $\neg h = \neg h_1 \wedge \cdots \wedge \neg h_k$ represents the negated hypothesis. For each h_i a potential υ_i is constructed so that $d(\upsilon_i) = d(h_i)$ and $[\upsilon_i({H_i}^c)]_m = 1$ for $H_i = N(h_i)$. In such a way, υ_i is equivalent to $\neg h_i$ and in addition, $\upsilon = \upsilon_1 \otimes \cdots \otimes \upsilon_k$ is equivalent to $\neg h$. Each of the potentials $\upsilon_1, \ldots, \upsilon_k$ is then added to the join tree. Therefore, for each υ_i there must be at least one node of which the label is a superset of $d(\upsilon_i)$.

Example 8.1 Adding Potentials to the Join Tree On the left side of Figure 8.1, the potentials υ_1,υ_2, and υ_3 are added to three different nodes of the join tree. However, note that it may also happen that several potentials are added to the same node.

Then, a partial inward propagation toward the root node can be started. Again, it is not necessary to perform a complete inward propagation phase if intermediate results are stored during the first inward propagation phase. On the right side of Figure 8.1 it can be seen that actually only the nodes N_1, N_3, N_4, and N_7 have to perform computations. ⊖

Note that the transformation of h into a disjunction is only interesting when the domain of every formula h_i is a strict subset of $d(h)$, that is $d(h_i) \subset d(h)$ for $1 \leq i \leq k$. In this case, it can happen that a query can be answered even if there is no node in the join tree of which the label is a superset of $d(h)$.

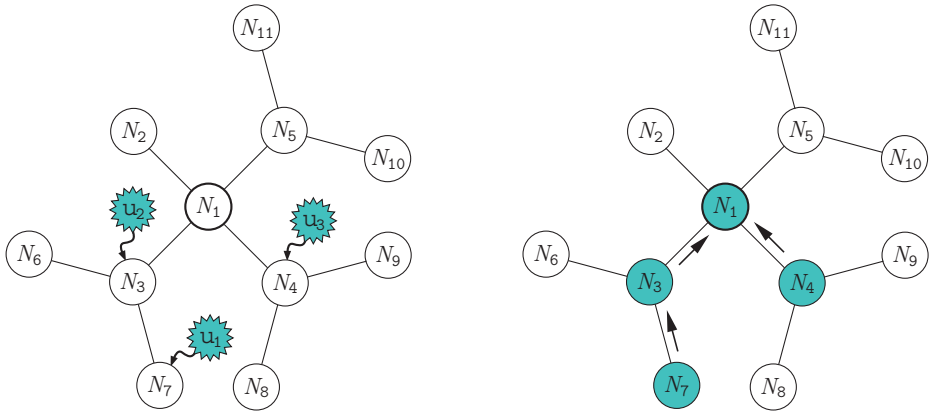

Figure 8.1: Partial Inward Propagation.

Therefore, a larger class of queries can be answered if h is transformed into a disjunction $h = h_1 \vee \cdots \vee h_k$.

Note also that the transformation of h into a disjunction $h = h_1 \vee \cdots \vee h_k$ is not unique. One possibility is for example to compute the *disjunctive normal form* of h. Usually, this can be done efficiently because the hypothesis h is hardly ever a complicated formula.

Example 8.2 Implications as Queries Suppose that the query h is given by $h = p \rightarrow q$. The new method first transforms h into the equivalent disjunction $h = \neg p \vee q$. Then, $h = h_1 \vee h_2$ with $h_1 = \neg p$ and $h_2 = q$, and therefore $d(h_1) = \{p\}$ and $d(h_2) = \{q\}$. Traditional methods cannot answer the query if there is not at least one node in the join tree of which the label is a superset of $d(h) = \{p, q\}$. However, the query can be answered by the new method because even if there is no node of which the label is a superset of $\{p, q\}$ there is always a node of which the label contains p and another node of which the label contains q. ⊖

8.2.3 The Algorithm

Suppose that a join tree with nodes $N_1, \ldots, N_n$ was constructed for a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = \{\xi, \mathcal{V}, \mathcal{A}, \Pi\}$. Every node N_i of the join tree contains a potential φ_i and the joint potential $\varphi = \varphi_1 \otimes \ldots \otimes \varphi_n$ corresponds to the knowledge base ξ. The following algorithm computes $dsp(h, \xi)$ for every hypothesis $h \in \{h_1, \ldots, h_m\}$.

Algorithm *The New Method*

Select a node N_r as root node
Inward Propagation : $c_1 = [\varphi^{\downarrow D_r}(\emptyset)]_m$
If $c_1 = 1$ **then Exit**
For each h_i **in** $\{h_1, \ldots, h_m\}$ **do**
  Transform h_i **into** $h_i = h_{i_1} \vee \cdots \vee h_{i_k}$
  For $j = 1$ **to** k **do**
    $H_{i_j} = N(h_{i_j})$
    Build υ_{i_j} **so that** $d(\upsilon_{i_j}) = d(h_{i_j})$ **and** $[\upsilon_{i_j}({H_{i_j}}^c)]_m = 1$
    Select node N_υ **so that** $d(\upsilon_{i_j}) \subseteq D_\upsilon$
    Add υ_{i_j} to node N_υ
  Next j
  Partial Inward Propagation : $c_2 := [(\varphi \otimes \upsilon)^{\downarrow D_r}(\emptyset)]_m$
  Output $dsp(h_i, \xi) = \frac{c_2 - c_1}{1 - c_1}$
Next h_i

If $c_1 = 1$ then the given knowledge is completely contradictory. In this case, c_2 would also be equal to 1 for every hypothesis. Otherwise, each query is answered by a partial inward propagation.

8.3 The "Communication Line" Example

In the following, the example of Section 2.8 will be used to demonstrate the new method. The query $dsp(x_0 \rightarrow x_m, \xi)$ is answered by a first complete inward propagation phase and an additional partial inward propagation phase. Although here, all nodes are involved in the partial inward propagation phase, it can nevertheless be seen that the new method requires a much simpler join tree than the one used for traditional methods (see Figure 6.17).

Example 8.3 Communication Line (continued) First, the hypothesis has to be transformed into an equivalent disjunction. Here, the hypothesis $h = x_0 \rightarrow x_m$ is equivalent to $h = h_1 \vee h_2$, where $h_1 = \neg x_0$ and $h_2 = x_m$. Therefore, the query can always be answered on whatever join tree is used. The join tree constructed by the application of Shenoy's fusion algorithm using the elimination sequence $\langle x_0, x_1, \ldots, x_m \rangle$ is shown in Figure 8.2. In addition, by using the above elimination sequence, the node labeled $\{x_{m-1}, x_m\}$ will be the root node denoted by N_r.

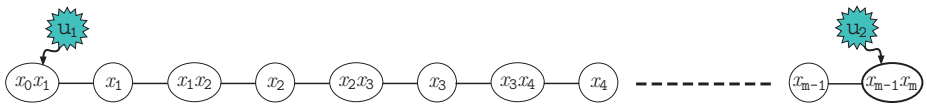

Figure 8.2: Join Tree for the "Communication Line" Example.

The potentials $\vartheta_1, \ldots, \vartheta_m$ obtained for the knowledge base ξ are distributed on this join tree. For every potential ϑ_i with domain $\{x_{i-1}, x_i\}$ there is a corresponding node with the same label. On the other hand, the nodes with label $\{x_1\}, \ldots, \{x_{m-1}\}$ contain the neutral potential. If $\varphi = \vartheta_1 \otimes \cdots \otimes \vartheta_m$ denotes the joint potential, then the inward propagation gives

$$c_1 \quad = \quad [\varphi^{\downarrow D_r}(\emptyset)]_m \quad = \quad 0.$$

Then, two potentials υ_1 and υ_2 equivalent to $\neg h_1$ and $\neg h_2$ have to be constructed. It can be verified that $d(\upsilon_1) = \{x_0\}$ and $d(\upsilon_2) = \{x_m\}$ and that each of the two potentials has only one focal set given by $\{\mathbf{f}_0\}$ and $\{\mathbf{t}_m\}$. The partial inward propagation toward the root node is then equivalent to the computation of

$$((\cdots((\upsilon_1 \otimes \vartheta_1)^{\downarrow\{x_1\}} \otimes \vartheta_2)^{\downarrow\{x_2\}} \otimes \cdots \otimes \vartheta_{m-1})^{\downarrow\{x_{m-1}\}} \otimes \upsilon_2 \otimes \vartheta_m)^{\downarrow\emptyset}.$$

It can be verified that every intermediate result of the above formula is a potential which has exactly two focal sets. Therefore, the message sent from a node with label $\{x_{i-1}, x_i\}$ to its neighbor node labeled $\{x_i\}$ is a potential with focal sets $\{\mathbf{f}_i\}$ and Θ_{x_i}. The masses assigned to these focal sets are 0.9^i and $1 - 0.9^i$ respectively. Finally, the combination with υ_2 determines the result of the partial inward propagation given by

$$c_2 \quad = \quad [(\varphi \otimes \upsilon)^{\downarrow D_r}(\emptyset)]_m \quad = \quad 0.9^m.$$

The values c_1 and c_2 are sufficent to compute $dsp(x_0 \rightarrow x_m, \xi) = 0.9^m$. $\ominus$

8.4 Differences to Traditional Methods

The main difference between traditional methods and the new method is that traditional methods compute for a given hypothesis h first the marginal $[\varphi^{\downarrow D_h}]_m$, where $D_h = d(h)$. The marginal is then used to compute the value $[\varphi^{\downarrow D_h}(H)]_b$ for $H = N(h)$. In contrast, the new method computes directly the value $[\varphi^{\downarrow D_h}(H)]_b$. Therefore, traditional methods correspond in a certain way to a complete compilation of the knowledge, whereas the new method corresponds only to a partial compilation.

There is also a difference concerning the join tree required by the two sort of methods. For a given probabilistic argumentation system $\mathcal{PAS}_\mathcal{V} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$, the new method requires often a simpler join tree. This is due to the fact that traditional methods require the join tree to contain at least one node of which the label is a superset of the domain of the query. By comparing Figure 6.17 and Figure 8.2 it can be seen that this requirement can lead to a worse join tree.

Finally, for a given join tree a larger class of queries can be answered by the new method. If for example the query $dsp(x_1 \rightarrow x_m)$ has to be answered, then

traditional methods cannot use the join tree of Figure 6.17 directly because there is no node of which the label is a superset of $\{x_1, x_m\}$. Therefore, a new join tree has to be constructed or the join tree has to be modified as proposed in Section 7.6. In contrast, by using the new method and by transforming the query into an equivalent disjunction, the join tree of Figure 8.2 can also be used to compute $dsp(x_1 \rightarrow x_m)$.

Traditional methods are more efficient when there are many queries and when the domains of these queries do not imply that the join tree used has to be modified several times.

8.5 Improvements

In the previous section it is shown that an inward propagation phase followed by a partial inward propagation is sufficient to answer a query. In order to answer queries as fast as possible it is therefore important that inward propagation is as fast as possible. For this reason, we present in the following two improvements

8.5.1 Arranging the Combinations

Suppose that N_{k_0} has outward neighbors $N_{k_1}, \ldots, N_{k_{m-1}}$. During the first complete inward propagation phase, N_{k_0} has to compute

$$\varphi'_{k_0} \quad = \quad \varphi_{k_0} \otimes \varphi_{k_1 k_0} \otimes \ldots \otimes \varphi_{k_{m-1} k_0}.$$

Because of associativity and commutativity there are many ways to compute this. Although the final result is always the same there may be big differences in the time needed. This is shown by the following example:

Example 8.4 Arranging the Combinations Suppose that the potentials φ_1, φ_2, and φ_3 are given. Then,

$$(\varphi_1 \otimes \varphi_2) \otimes \varphi_3 \quad = \quad (\varphi_1 \otimes \varphi_3) \otimes \varphi_2 \quad = \quad (\varphi_2 \otimes \varphi_3) \otimes \varphi_1$$

are three possible ways to compute the combination of these potentials. Suppose now that φ_1 and φ_2 both have 2 focal sets, whereas φ_3 has 100 focal sets. If φ_1 is first combined with φ_2 and then with φ_3, then one 2×2-combination and one 4×100-combination is needed in the worst case. In contrast, if φ_1 is first combined with φ_3 and then with φ_2, then one 2×100-combination and one 2×200-combination is needed in the worst case. The first case is obviously much more efficient. $\ominus$

Therefore, as the time needed to combine two potentials is correlated to the number of focal sets it seems natural to combine first potentials possessing fewer focal sets. This heuristic is used in the following algorithm:

Algorithm *Arranging the Combinations*

$\Psi := \{\varphi_{k_0}\} \cup \{\varphi_{k_i k_0} : 1 \leq i < m\}$
Repeat
  Choose $\vartheta_k \in \Psi$ and $\vartheta_l \in \Psi$ **so that**
    $|FS(\vartheta_k)| \leq |FS(\vartheta)|$ for all $\vartheta \in \Psi$ **and**
    $|FS(\vartheta_l)| \leq |FS(\vartheta)|$ for all $\vartheta \in \Psi \setminus \{\vartheta_k\}$
  $\Psi = \Psi \cup \{\vartheta_k \otimes \vartheta_l\} \setminus \{\vartheta_k, \vartheta_l\}$
Until $|\Psi| = 1$

Every node uses this algorithm to combine its incoming messages with its potential. At the end, the set Ψ contains only one potential which is then equal to φ'_{k_0}. If N_{k_0} is the root node N_r, then φ'_{k_0} is equal to $\varphi^{\downarrow D_r}$. Otherwise, the inward message $\varphi_{k_0 k_m}$ to N_{k_m} can easily be computed using Equation 8.10 by an additional marginalization.

The algorithm can be visualized when for every combination a new node is created. If N_{k_0} has m neighbor nodes, then there are exactly $m-1$ new nodes with labels D_{k_0} created. Each of these newly created nodes serves to store the corresponding intermediate result. As an example, in Figure 8.3 two trees are shown for a given node with 4 neighbor nodes. The tree on the left corresponds to the computation of $((\varphi_{k_0} \otimes \varphi_{k_1 k_0}) \otimes \varphi_{k_2 k_0}) \otimes \varphi_{k_3 k_0}$, whereas the one on the right corresponds to $(\varphi_{k_0} \otimes \varphi_{k_1 k_0}) \otimes (\varphi_{k_2 k_0} \otimes \varphi_{k_3 k_0})$.

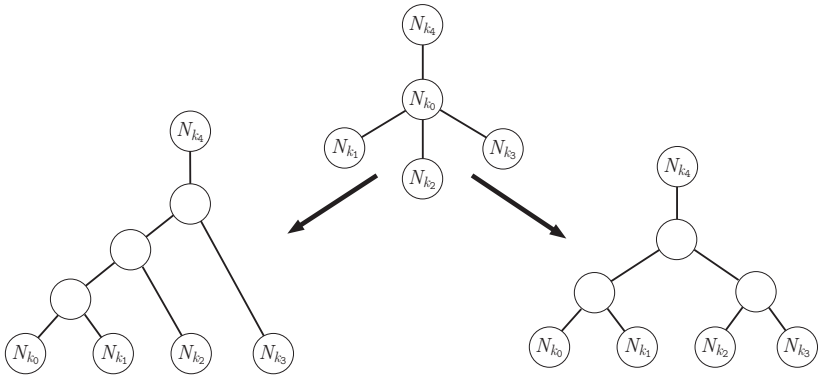

Figure 8.3: Visualizing the Algorithm.

The algorithm presented here is related to the technique of **binary join trees** (see Section 7.5) because it also minimizes the number of combinations needed by storing intermediate results. However, here the nodes of the join tree are originally not restricted to at most 3 neighbor nodes. But because combination is a binary operation the visualization of the algorithm leads always to a binary join tree.

In (Xu, 1991) another method is presented which minimizes the number of combinations by saving intermediate results. The messages from outward neighbors

of N_{k_0} are combined accumulatively with the potential of N_{k_0}. The binary join tree on the left side of Figure 8.3 corresponds to this method. If N_{k_0} has m neighbor nodes, then there are $\prod_{k=2}^{m} \binom{k}{2} = \frac{m! \cdot (m-1)!}{2^{(m-1)}}$ possible sequences of combinations. Only one of them corresponds to the method presented in (Xu, 1991), whereas our method tries to select a good sequence depending on φ_{k_0} and the incoming messages $\varphi_{k_1 k_0}, \ldots, \varphi_{k_{m-1} k_0}$.

8.5.2 No Need to Store Intermediate Results

For nodes with a lot of neighbor nodes, it can happen that the combination of incoming messages generates huge mass functions even if the incoming messages were relatively small and even if the heuristic of the last Subsection is used. Nevertheless, outgoing messages can be relatively small in the case, where a marginalization is performed. Storing these intermediate results can therefore necessitate a huge amount of memory. In addition, it is advantageous mainly for the outward propagation phase and it is not that much important in this approach since the outward propagation is replaced by a partial inward propagation.

8.5.3 Selection of Nodes

In the algorithm presented in Subsection 8.2.3 a node N_v has to be selected to which a newly constructed potential v is then added. As already mentioned, there may be many nodes which could be used for this purpose. However, it is best to select the one which is closest to the root node. In such a way, the path to the root node is then as short as possible.

9

Implementation Aspects

In the previous chapters it was shown that computing **degrees of support** is equivalent to computing **normalized belief** in Dempster-Shafer theory. An efficient way for computing normalized belief is to use **Shenoy's fusion algorithm** which allows to compute the marginal for a given query. If there are several queries, then it is advisable to use a **join tree**. The computation of the marginals is then based on a message-passing scheme in the join tree. The **Shenoy-Shafer architecture** describes such a message-passing scheme and is best suited for Dempster-Shafer theory. The initially given potentials and the computed messages are then represented as **mass functions**.

In the following, we will show how the computation of marginals in a join tree can be **implemented efficiently** on a computer. For this, no particular knowledge about programming languages is needed even though we used the programming language LISP to implement the data structures and algorithms presented later in this chapter. More precisely, we worked on computers of **Apple Power Macintosh** type using the excellent software package **Macintosh Common Lisp** from Digitool.

We start with a discussion about storing **mass functions**. The main part of this discussion will be about storing an individual **focal set** of a mass function. The representation of individual focal sets is important because it determines heavily the amount of memory and the time used for combination, projection and extension. However, the representation of mass functions is at least as important as the representation of an individual focal set. Especially, the amount of time used for combining mass functions depends strongly on how mass functions are represented.

Finally, a data structure called **variable link list** will be presented. This data structure is very appropriate for the **fusion algorithm**.

9.1 Representing Focal Sets

As shown in Chapter 3, a mass function ϑ on domain $D = \{x_1, \ldots, x_n\}$ is completely determined by its focal sets and the masses of these focal sets.

If each variable x_j has the set of values $\Theta_{x_j} = \{1, \ldots, k_j\}$, then the set of configurations Θ_D consists of $\mathrm{K} = k_1 \cdot \ldots \cdot k_n$ elements. We suppose in the following that the configuration $\mathbf{c}_r$, where r is given by

$$r = \sum_{j=1}^{n} ((r_j - 1) \cdot \prod_{i=j+1}^{n} k_i). \tag{9.1}$$

corresponds to the assignment of values $r_1, \ldots, r_n$ to the variables $x_1, \ldots, x_n$, where $1 \leq r_j \leq k_j$ for $1 \leq j \leq n$. This way of enumerating the configurations is shown on the left side of Figure 9.1.

In the following, different ways of representing a set $A \subseteq \Theta_D$ will be presented. We are in particular interested in representations which allow to compute $A \cap B$, $A^{\downarrow D'}$, and $A^{\uparrow D''}$ for $B \subseteq \Theta_D$ and $D' \subseteq D \subseteq D''$ as fast as possible. This is due to the fact that the basic operations for mass functions make heavy use of these kind of computations. In addition, it is also important that the coding of a set $A \subseteq \Theta_D$ does not need too much memory space. We will see later that this goal is not easy to achieve.

In order to explain different ways of coding sets, the 3 sets A_1, A_2, and A_3 shown on the right side of Figure 9.1 will be used. The sets A_1 and A_3 represent two extrem cases, where a set consists of only one element and of all elements of Θ_D, respectively.

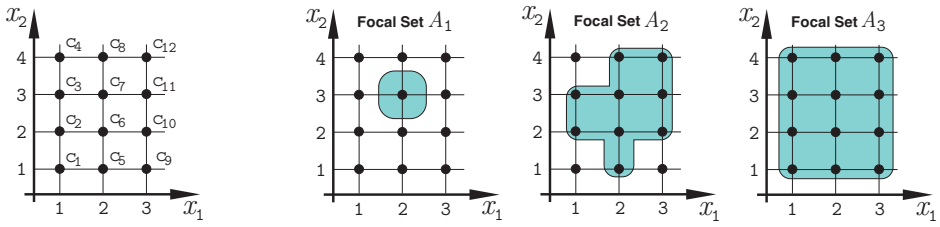

Figure 9.1: Representing Sets $A \subseteq \Theta_D$.

In the following, we will discuss six different representations. In order not to mix up a set $A \subseteq \Theta_D$ with its representation, we use $r_1(A), \ldots, r_6(A)$ to denote the corresponding representation of A.

9.1.1 List of Configurations

A focal set A of a mass function ϑ defined on domain D is first of all a subset of Θ_D. Therefore, a focal set A can be represented as a list of its elements. As an example, the 3 sets on the right side of Figure 9.1 are then represented as follows:

Example 9.1 Representing the Sets of Figure 9.1 $r_1(A_1) = \{\mathbf{c}_7\}$
$r_1(A_2) = \{\mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7, \mathbf{c}_8, \mathbf{c}_{10}, \mathbf{c}_{11}, \mathbf{c}_{12}\}$
$r_1(A_3) = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7, \mathbf{c}_8, \mathbf{c}_9, \mathbf{c}_{10}, \mathbf{c}_{11}, \mathbf{c}_{12}\}$ ⊖

The above example shows that focal sets with only a few number of elements can be stored efficiently. If a configuration $\mathbf{c}_i$ is represented by the number $(i-1)$, then $\lceil \log_2(\mathrm{K}-1) \rceil$ bits are needed for each configuration. A focal set with h elements then requires $h \cdot \lceil \log_2(\mathrm{K}-1) \rceil$ bits. Thus, the focal set given by Θ_D represents the worst case and requires $\mathrm{K} \cdot \lceil \log_2(\mathrm{K}-1) \rceil$ bits.

Where the computation of $A \cap B$, $A^{\downarrow D'}$, and $A^{\uparrow D''}$ for $A, B \subseteq \Theta_D$ and $D' \subseteq D \subseteq D''$ is concerned, it can be verified that the representation of a focal set as a set of its elements is not that much appropriate. For example, $A \cap B$ corresponds to the computation of the intersection of both sets. If A and B have only a few number of elements, then the intersection can be computed more or less efficiently. However, the computation of $A \cap B$ needs much more time if A and B have many elements.

9.1.2 Binary Representation

A representation which was already proposed in (Xu & Kennes, 1994) is to use bitstrings for representing subsets. If

$$b_i \quad = \quad \begin{cases} 1 & \text{if } \mathbf{c}_i \in A, \\ 0 & \text{otherwise} \end{cases}$$

then each set $A \subseteq \Theta_D$ is unequivocally determined by the bitstring $\%b_{\mathrm{K}} \ldots b_1$. Storing a bitstring is easy because for each bitstring $\%b_{\mathrm{K}} \ldots b_1$ there is a corresponding number $r_2(A)$ given by

$$r_2(A) \quad = \quad \sum_{i=1}^{K} b_i \cdot 2^{i-1}. \tag{9.2}$$

Therefore, $A \subseteq \Theta_D$ is represented by a number $r_2(A) \in \{0, \ldots, 2^{\mathrm{K}} - 1\}$. The corresponding bitstring $\%b_{\mathrm{K}} \ldots b_1$ is the binary representation of the (decimal) number $r_2(A)$. The i-th bit of a bitstring is assigned to the configuration $\mathbf{c}_i$. For example, the 4th bit is assigned to $\mathbf{c}_4$ because $\mathbf{c}_4$ is represented by the decimal number $2^3 = 8$ which is equivalent to the bitstring %1000.

Example 9.2 Representing the Sets of Figure 9.1 $r_2(A_1) = 2^6 = 64$
$\Rightarrow$ %000001000000 = 64
$r_2(A_2) = 2^1 + 2^2 + 2^4 + 2^5 + 2^6 + 2^7 + 2^9 + 2^{10} + 2^{11} = 3830$
$\Rightarrow$ %111011110110 = 3830
$r_2(A_3) = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 + 2^8 + 2^9 + 2^{10} + 2^{11} = 4095$
$\Rightarrow$ %111111111111 = 4095 $\ominus$

Storing $r_2(A)$ for $A \subseteq \Theta_D$ requires at most K bits. The sets which require exactly K bits are those which contain the configuration $\mathbf{c}_{\mathrm{K}}$. These sets are identified with a number $r_2(A) \in \{2^{\mathrm{K}-1}, \ldots, 2^{\mathrm{K}} - 1\}$. Therefore, it is $\mathbf{c}_{\mathrm{K}} \in A$ for half of the sets $A \subseteq \Theta_D$.

The main advantage of the binary representation is that $A \cap B$, $A^{\downarrow D'}$, and $A^{\uparrow D''}$ for $A, B \subseteq \Theta_D$ and $D' \subseteq D \subseteq D''$ can be computed very fast. For example, $A \cap B$ is computed as (AND $r_2(A)$ $r_2(B)$), where AND represents the operation which performs the bit-wise logical "and" of two binary numbers. The main processor of a computer has built-in commands which allow to perform this kind of operation extremely fast.

9.1.3 Normal Forms

Every set $A \subseteq \Theta_D$ can be represented as a union of sets which are intersections of sets $S_j \subseteq \Theta_{x_j}$ for $1 \leq j \leq n$. Therefore,

$$r_3(A) \quad = \quad \bigcup_{i=1}^{h_1} C^i \quad \text{where } C^i = \bigcap_{j=1}^{n} S_j^i,\ S_j^i \subseteq \Theta_{x_j} \tag{9.3}$$

corresponds to the *disjunctive normal form* of propositional logic. Another possibility for representing $A \subseteq \Theta_D$ is as an intersection of sets which are unions of sets $S_j \subseteq \Theta_{x_j}$. This signifies that

$$r_4(A) \quad = \quad \bigcap_{i=1}^{h_2} D^i \quad \text{where } D^i = \bigcup_{j=1}^{n} S_j^i,\ S_j^i \subseteq \Theta_{x_j} \tag{9.4}$$

corresponds to the *conjunctive normal form* of propositional logic. There are almost always different possibilities of representing a set in such a way.

Example 9.3 Representing the Sets of Figure 9.1 $r_3(A_1) = ((x_1 \in \{2\}) \cap (x_2 \in \{3\}))$
$r_4(A_1) = ((x_1 \in \{2\}) \cup (x_2 \in)) \cap ((x_1 \in) \cup (x_2 \in \{3\}))$

$r_3(A_2) = ((x_1 \in \{1,2,3\}) \cap (x_2 \in \{2,3\})) \cup ((x_1 \in \{2\}) \cap (x_2 \in \{1,2,3,4\})) \cup$
$\quad ((x_1 \in \{2,3\}) \cap (x_2 \in \{2,3,4\}))$
$r_4(A_2) = ((x_1 \in \{2\}) \cup (x_2 \in \{2,3,4\})) \cap ((x_1 \in \{2,3\}) \cup (x_2 \in \{1,2,3\}))$

$r_3(A_3) = ((x_1 \in \{1,2,3\}) \cap (x_2 \in \{1,2,3,4\}))$
$r_4(A_3) = ((x_1 \in \{1,2,3\}) \cup (x_2 \in))$ $\ominus$

We are particularly interested in short representations. In the above example, $r_3(A_1)$ is shorter than $r_4(A_1)$. On the other hand, $r_4(A_2)$ is shorter than $r_3(A_2)$. Therefore, there are sets $A \subseteq \Theta_D$, where $r_3(A)$ is shorter and sets $A \subseteq \Theta_D$, where $r_4(A)$ is shorter. In the worst case, both representations require at least $k_1 \cdot \ldots \cdot k_{n-1}$ terms if we suppose that $k_n \geq k_j$ for all $j < n$. However, the construction of such a set representing the worst case is quite complicated. Much more insteresting is the average case. Therefore, what is the average number of terms in a real world problem ?

In order to answer this question, we computed $|r_3(A)|$ for a collection of subsets of Θ_D obtained by sampling every set $A \subseteq \Theta_D$ with equal probabilities. a

uniform law. To simplify things, only sets D composed of binary variables were considered. Figure 9.2 shows the results of selecting 100,000 sets for $|D| = 5$ and $|D| = 6$. It can be verified that around $2^{|D|-1}$ terms were used for most of the sets.

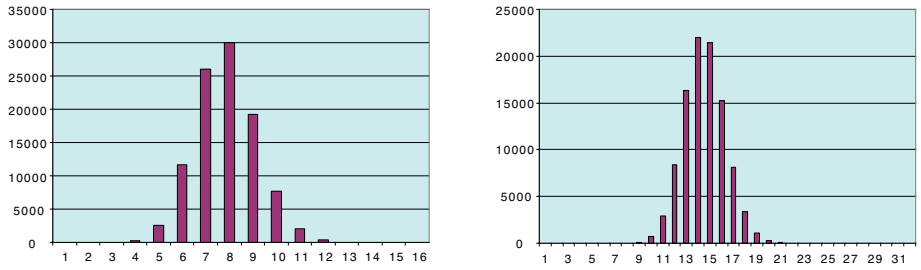

Figure 9.2: Estimated Number of Terms, $|D| = 5$ and $|D| = 6$.

Very often, the situation is even better than shown in Figure 9.2. For example, Figure 9.3 shows the results of computing $|r_3(A)|$ for the focal sets of messages and potentials of the example "Grid-8". It can be seen that almost all of these focal sets can be represented using only a few terms.

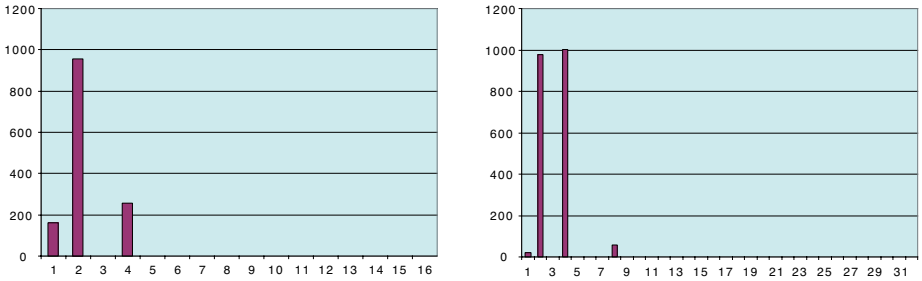

Figure 9.3: Number of Terms for "Grid-8", $|D| = 5$ and $|D| = 6$.

For estimating the memory space needed by these two representations, we suppose in the following that $k_j \leq k_n$ for all $j \leq n$. In this case, a set $S_j \subseteq \Theta_{x_j}$ can be represented using k_n bits (see Subsection 9.1.2) and $n \cdot k_n$ bits are needed for C^i and D^i respectively. Finally, the representation of $A \subseteq \Theta_D$ as proposed in Equations 9.3 and 9.4 requires $h_1 \cdot n \cdot k_n$ and $h_2 \cdot n \cdot k_n$ bits respectively. Thus, short representations require less memory space.

No matter how, sets S_j^i which are equal to Θ_{x_j} are not really needed in Equation 9.3. Thus, the term C^i can sometimes be represented as an intersection of less than n subsets S_j^i. Therefore, $A \subseteq \Theta_D$ can be represented as

$$r_5(A) \quad = \quad \bigcup_{i=1}^{h_1} C^i \quad \text{where } C^i = \bigcap_{j \in I_i} S_j^i,\ S_j^i \subseteq \Theta_{x_j} \tag{9.5}$$

where $I_1, \ldots, I_{h_1} \subseteq \{1, \ldots, n\}$. The same idea is also valid for the representation proposed in Equation 9.4 because those sets S_j^i which are equal to the empty set can be omited. Therefore, $A \subseteq \Theta_D$ can be represented as

$$r_6(A) \quad = \quad \bigcap_{i=1}^{h_2} D^i \quad \text{where } D^i = \bigcup_{j \in I_i} S_j^i,\ S_j^i \subseteq \Theta_{x_j} \tag{9.6}$$

where $I_1, \ldots, I_{h_2} \subseteq \{1, \ldots, n\}$. Sometimes, this idea can help to decrease memory requirements.

Concerning the computation of $A \cap B$ and $A^{\downarrow D'}$ for $A, B \subseteq \Theta_D$ and $D' \subseteq D$, it can be verified that $A \cap B$ is quite difficult to compute for r_3 and r_5. On the other hand, $A^{\downarrow D'}$ is difficult for r_4 and r_6. To illustrate this, let us look at the computation of $A \cap B$. First, suppose that

$$\begin{aligned} r_4(A) &= A_1 \cap \cdots \cap A_n \\ r_4(B) &= B_1 \cap \cdots \cap B_m \end{aligned}$$

where each A_i and each B_j is a union of subsets of Θ_{x_k} for $x_k \in D$. In this case, the computation of $r_4(A \cap B)$ is not very difficult because

$$r_4(A \cap B) \quad = \quad A_1 \cap \cdots \cap A_n \cap B_1 \cap \cdots \cap B_m.$$

Of course, not all these terms are really needed. A term A_i can be removed if there is another term B_j so that $B_j \subseteq A_i$. In the same way, B_j can be removed if there is a term A_i so that $A_i \subseteq B_j$. In addition, $r_4(A \cap B)$ may contain intersections of the form $A_i \cap B_j$ which could be replaced by a single term. In any case, these simplifications are computationaly not very expensive.

Now, let us suppose that

$$\begin{aligned} r_3(A) &= A_1 \cup \cdots \cup A_n \\ r_3(B) &= B_1 \cup \cdots \cup B_m \end{aligned}$$

where each A_i and each B_j is an intersection of subsets of Θ_{x_k} for $x_k \in D$. Then, $r_3(A \cap B)$ might be quite difficult to compute as

$$r_3(A \cap B) \quad = \quad \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} (A_i \cap B_j) \right).$$

Often, there are terms which are not necessary. A term $A_i \cap B_j$ can be removed if there is another term $A_k \cap B_\ell$ so that $A_i \cap B_j \subseteq A_k \cap B_\ell$. In addition, $r_3(A \cap B)$ may contain unions $(A_i \cap B_j) \cup (A_k \cap B_\ell)$ which could be replaced by a single term. This kind of simplification is computationaly expensive if n and m are large.

9.1.4 Comparison

It was already mentioned that a representation has the following two main objectives:

- it should not use too much memory space to store a focal set $A \subseteq \Theta_D$,
- it should be able to compute $A \cap B$, $A^{\downarrow D'}$, and $A^{\uparrow D''}$ for $A, B \subseteq \Theta_D$ and $D' \subseteq D \subseteq D''$ as fast as possible.

The binary representation is the best choice if we consider only operations. Above all, if the state space of the domain of a mass function is small, then $A \cap B$, $A^{\downarrow D'}$, and $A^{\uparrow D''}$ can be computed very fast and only a small amount of memory space is required to store a single focal set. However, the binary representation can be a bad choice where memory space is concerned. Suppose for example that the potential ϑ is defined on domain $D = \{x_1, \ldots, x_{12}\}$, where $x_1, \ldots, x_{12}$ are binary variables. Then, Θ_D consists of 4096 configurations. Therefore, each focal set of ϑ requires 512 bytes. If ϑ has 100,000 focal sets, then almost 50 Mbytes are needed for these focal sets. As a consequence, it could be advisable to use another representation in the case, where the domain of the potentials are big.

The current implementation of the propagation framework only disposes of the binary representation. A **hybrid approach**, where different representations are used could help to decrease the amount of memory space needed. For example, the binary representation could be used to perform the operations, whereas the disjunctive normal form is used for potentials having a large domain. However, note that a hybrid approach involves a bigger computational effort.

It could be argued that potentials usually have less than 100,000 focal sets. Of course, this is true for the initially given potentials $\vartheta_1, \ldots, \vartheta_m$. However, in Chapter 11 we will see some relative simple examples, where potentials computed during the propagation phase can have a huge number of focal sets. No matter how, there are examples, where **exact computation** is not feasible. Therefore, **approximation methods** are needed which avoid the generation of potentials having a huge number of focal sets.

9.2 Representing Mass Functions

As explained in Chapter 3, a mass function ϑ defined on domain D consists of a set of pairs $(A, [\vartheta(A)]_m)$, where $A \subseteq \Theta_D$ and $0 \leq [\vartheta(A)]_m \leq 1$. In any case, it is impossible to store such a pair for all sets $A \subseteq \Theta_D$ because there would be too many. Instead, only the corresponding pairs of **focal sets** of ϑ are stored, what means, the pairs $(A, [\vartheta(A)]_m)$, where $[\vartheta(A)]_m > 0$.

In the current implementation of the propagation framework, the **binary representation** presented in Subsection 9.1.2 is used for the focal sets of ϑ. This

means that $|\Theta_D|$ bits are required for each focal set. The consequence of enumerating the configurations of Θ_D as proposed in Equation 9.1 is that the order of the variables of $d(\vartheta)$ does matter. Thus, we have to think of $d(\vartheta)$ as an ordered sequence of variables and not as an unordered set.

We do not only suppose that the domain of each mass function is an ordered sequence of variables. Additionally, we require also that the domain of each mass function has to respect a **global order** of the variables. The reason for this preliminary condition is that the basic operations can be performed much more efficiently. To illustrate this, imagine that two mass functions ϑ and φ have the same domain. It is easy to verify that $\vartheta \otimes \varphi$ can only be computed efficiently if the order of the variables in $d(\vartheta)$ and $d(\varphi)$ is the same.

The values $[\vartheta(A)]_m$ are stored in the **IEEE-754 floating-point format**. This standard defines conventions for 32- and 64-bit arithmetic. We use **double precision** (64 bit) instead of single precision (32 bit) because Macintosh Common Lisp supports only double precision. However, there are also other reasons why it is preferable to use double precision:

- the main processor of today's computer executes operations for double precision almost as fast as for single precision,
- rounding errors are smaller when double precision is used.

9.2.1 Binary Tree for Combination and Marginalization

The complexity of combination and marginalization depends heavily on the computer representation of mass functions. Just imagine for example that two mass function φ and ϑ have to be combined and that both mass functions have a huge number of focal sets. Then, $A \cap B$ has to be computed for every focal set $A \in FS(\varphi)$ and $B \in FS(\vartheta)$. Usually, lots of these intersections are equal and must be merged together. However, this means that for each newly computed intersection we have to look whether the same set was already obtained previously.

A binary tree can be used to minimize the amount of time needed for this lookup. If the binary representation is used to represent the focal sets, then the branching of the tree is determined in a natural way by the bitstrings of the focal sets. Every branch of the binary tree has a bitstring for label. The concatenation of all bitstrings on the path from the root to a leaf is the binary representation of a focal set. Therefore, it is sufficient to store the corresponding mass in the leaf. Figure 9.4 shows the insertion of four focal sets represented by %0010, %1110, %0011 and %1111 with masses of 0.48, 0.12, 0.32 and 0.08.

A binary tree is valuable for combination and marginalization, but it is not useful for extension. Therefore, if the last element has been inserted into the binary tree, then we have to go through the tree and store the elements in an array. This means that binary trees are only used temporarily. In the rest of the time arrays are used to store the elements of mass functions.

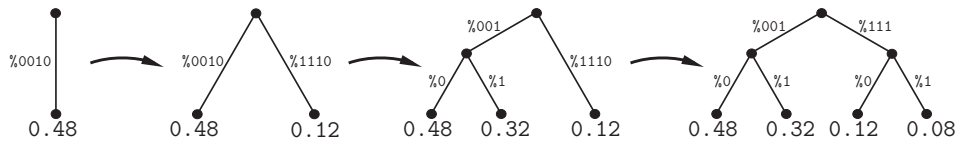

Figure 9.4: Binary Tree for Combination and Marginalization.

9.2.2 Hash Table for Combination and Marginalization

Another, very promising idea is to use a hash table when combining two mass functions and when marginalizing a mass function to a smaller domain. A hash table is a data structure which is used to store for given keys $key_1, \ldots, key_n$ the corresponding values $value_1, \ldots, value_n$. For this purpose, a **hash function** computes for the key the **hash value** which is then used as an offset in the hash table. Finally, some of the entries of the hash table contain a linked list with all values where the hash value of the corresponding keys were the same. Figure 9.5 shows an example of a hash table. Of course, hash functions should produce a uniform distribution of the values in the hash table, which means that all linked lists are of more or less the same size. Note that it may be necessary to rebuild a larger hash table if there are already too many values in the hash table. In this case, a new hash value has to be computed for each of the entries of the former hash table.

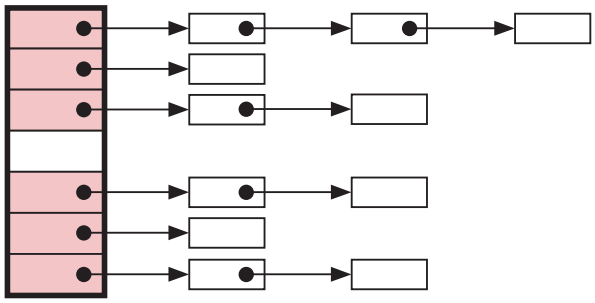

Figure 9.5: An Example of a Hash Table.

Now, suppose that the combination $\vartheta = \vartheta_1 \otimes \vartheta_2$ of two mass functions has to be computed. Therefore, for each pair $(A_i, [\vartheta_1(A_i)]_m)$ of ϑ_1 and each pair $(B_j, [\vartheta_2(B_j)]_m)$ of ϑ_2, the intersection $C_k = A_i \cap B_j$ and the corresponding product $c_k = [\vartheta_1(A_i)]_m \cdot [\vartheta_2(B_j)]_m$ have to be computed. If the binary representation (see Subsection 9.1.2) is used, then C_k is a (large) binary number. Using a hash table, the value c_k can be stored using C_k as corresponding key. The main advantage of a hash table is that it represents a very efficent way to test whether the same binary number C_k was already obtained previously. If C_k was already obtained, the corresponding mass c_k has to be added to the

already existing mass, otherwise, c_k is added for the key C_k to the hash table. If $n = |FS(\vartheta_1)| \cdot |FS(\vartheta_2)|$ denotes the number of intersections to be computed and s the size of the hash table, the average length of the linked lists is then at most n/s. By using a very big hash table (s very large), the computation of $\vartheta_1 \otimes \vartheta_2$ can be done quite fast. Using a hash table, the time complexity of combination is still $O(n^2)$. In contrast, using binary trees as described in the previous subsection, the time complexity of combination is only $O(n \cdot log_2(n))$. Nevertheless, for mass functions which are not too big, hash tables are often much more efficient than binary trees.

A hash table is also very appropriate for marginalization because it allows to test efficiently whether a given focal set was already obtained or not.

9.3 Bitmasks for Marginalization and Extension

It was already mentioned that the binary representation is very appropriate for combining mass functions. In the following, we show that it is also appropriate for marginalization and extension.

9.3.1 Bitmasks

In the following, suppose that $D = \{x_1, \ldots, x_n\}$ and that $E \subseteq D$. Suppose in addition that $\Theta_{x_j} = \{1, \ldots, k_j\}$ for $1 \leq j \leq n$. By enumerating the configurations as proposed in Equation 9.1 we obtain

$$\Theta_D = \{\mathbf{c}_1, \ldots, \mathbf{c}_\mathrm{K}\} \text{ where } \mathrm{K} = \prod_{x_j \in D} k_j, \tag{9.7}$$

$$\Theta_E = \{\mathbf{c}'_1, \ldots, \mathbf{c}'_\mathrm{L}\} \text{ where } \mathrm{L} = \prod_{x_j \in E} k_j. \tag{9.8}$$

First, let us look at the operation **marginalization**. Suppose that the mass function ϑ is defined on domain D. In order to build $\vartheta^{\downarrow E}$, we have to compute $A^{\downarrow E}$ for all $A \in FS(\vartheta)$. For this purpose, we will use the sequence of sets $C_E^D = \langle C_1, \ldots, C_\mathrm{L} \rangle$, where $C_i \subseteq \Theta_D$ is given by

$$C_i = \{\mathbf{c} \in \Theta_D : \mathbf{c}^{\downarrow E} = \mathbf{c}'_i\} = \mathbf{c}'_i \times \Theta_{D-E}. \tag{9.9}$$

$C_E^D = \langle C_1, \ldots, C_\mathrm{L} \rangle$ allows to compute $A^{\downarrow E}$ for an arbitrary set $A \subseteq \Theta_D$ as

$$A^{\downarrow E} = \{\mathbf{c}'_i \in \Theta_E : A \cap C_i \neq, C_i \in C_E^D\}. \tag{9.10}$$

To illustrate the computation of $A^{\downarrow E}$, let us look at the example shown in Figure 9.6, where $D = \{x_1, x_2\}$. The case $E = \{x_1\}$ is shown on the left and $E = \{x_2\}$ is on the right. The corresponding sequences C_E^D are given by

$$C_{\{x_1\}}^{\{x_1,x_2\}} = \langle \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4\}, \{\mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7, \mathbf{c}_8\}, \{\mathbf{c}_9, \mathbf{c}_{10}, \mathbf{c}_{11}, \mathbf{c}_{11}\} \rangle,$$
$$C_{\{x_2\}}^{\{x_1,x_2\}} = \langle \{\mathbf{c}_1, \mathbf{c}_5, \mathbf{c}_9\}, \{\mathbf{c}_2, \mathbf{c}_6, \mathbf{c}_{10}\}, \{\mathbf{c}_3, \mathbf{c}_7, \mathbf{c}_{11}\}, \{\mathbf{c}_4, \mathbf{c}_8, \mathbf{c}_{12}\} \rangle.$$

Now, suppose that $A = \{\mathbf{c}_7, \mathbf{c}_{10}, \mathbf{c}_{11}\}$. $A^{\downarrow E}$ is computed as the set of all $\mathbf{c}'_i \in \Theta_E$, where the intersection of the corresponding set $C_i \in C_E^D$ with A is not empty. In this way, $A^{\downarrow\{x_1\}} = \{\mathbf{c}'_2, \mathbf{c}'_3\}$ and $A^{\downarrow\{x_2\}} = \{\mathbf{c}'_2, \mathbf{c}'_3\}$ is obtained.

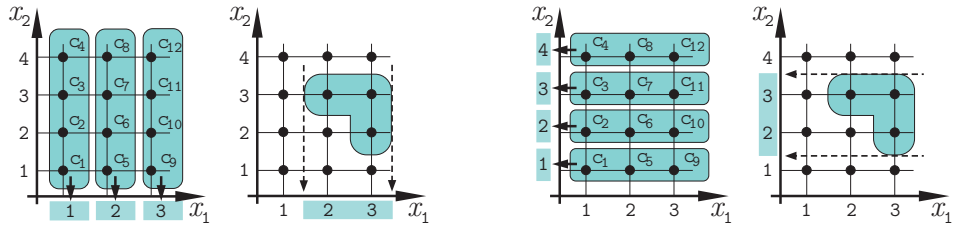

Figure 9.6: The Computation of $A^{\downarrow E}$ using the Sequence C_E^D.

Now, let us turn to the operation **extension**. Suppose that $d(\varphi) = E$. In order to build $\varphi^{\uparrow D}$, we compute $B^{\uparrow D}$ for all sets $B \in FS(\varphi)$ as

$$B^{\uparrow D} = \bigcup_{\mathbf{c}'_i \in B} \{C_i \in C_E^D\}. \tag{9.11}$$

The binary representation is used for each $C_i \in C_E^D$. $M_E^D = \langle M_1, \ldots, M_{\mathrm{L}} \rangle$, where $M_i = r_2(C_i)$ is therefore an ordered sequence of bitstrings. For example, the sequences M_E^D for the previous example are given by

$$M_{\{x_1\}}^{\{x_1,x_2\}} = \langle \%000000001111, \%000011110000, \%111100000000 \rangle,$$
$$M_{\{x_2\}}^{\{x_1,x_2\}} = \langle \%000100010001, \%001000100010, \%010001000100, \%100010001000 \rangle.$$

The term **bitmask** is used to denote such a sequence M_E^D. Note that M_E^D is completely determined by the sets D and E. In addition, M_E^D is all what is needed to compute $A^{\downarrow E}$ and $B^{\uparrow D}$ for $A \subseteq \Theta_D$, $B \subseteq \Theta_E$, and $E \subseteq D$.

9.3.2 Operations for Binary Numbers

The main processor of today's computer has built-in commands which are executed extremely fast and which play an important role in the construction of bitmasks. The bit-wise logical "and", the bit-wise logical "or" and the logical shift left are three such built-in commands.

Syntax: (AND num_1 num_2) ; (OR num_1 num_2)
num_1, num_2 as well as the result of both commands are binary numbers. As an example, Figure 9.7 shows the results of applying these two commands to the operands %0001101101110011 and %0111000101010001.

Syntax: (LSL num pos)
The binary number num is shifted pos positions to the left. This corresponds to the multiplication of num with the value 2^{pos}. As an example, Figure 9.8 shows the result of shifting %0001101101110011 one position to the left.

	AND:		OR:
	%0001101101110011		%0001101101110011
	%0111000101010001		%0111000101010001
	%0001000101010001		%0111101101110011

Figure 9.7: Bit-Wise Logical "And" and Bit-Wise Logical "Or".

LSL:	%0001101101110011
	%0011011011100110

Figure 9.8: Logical Shift Left.

9.3.3 Constructing Bitmasks

In the following, we show how a bitmask M_E^D can be constructed for given sets of variables D and $E \subseteq D$. To this purpose, let us first look at the *special case* where $E = D - \{x_j\}$ for an arbitrary variable $x_j \in D$. The set $D = \{x_1, \ldots, x_n\}$ can then be divided into three parts:

$$D = \{\underbrace{x_1, \ldots, x_{j-1}}_{\textsf{left}}, \underbrace{x_j}_{\textsf{middle}}, \underbrace{x_{j+1}, \ldots, x_n}_{\textsf{right}}\}$$

The size of the state space of these sets will be denoted by the numbers u_l, u_j and u_r, respectively. Therefore, these numbers are defined as follows:

$$u_l = |\Theta_{\{x_1, \ldots, x_{j-1}\}}|, \quad (9.12)$$

$$u_j = |\Theta_{\{x_j\}}|, \quad (9.13)$$

$$u_r = |\Theta_{\{x_{j+1}, \ldots, x_n\}}|. \quad (9.14)$$

Figure 9.9 shows that $M_1 \in M_E^D$ is completely determined by these numbers. In particular, M_1 consists of u_j identical blocks containing u_r bits. In each of these blocks, only the first bit is set to 1.

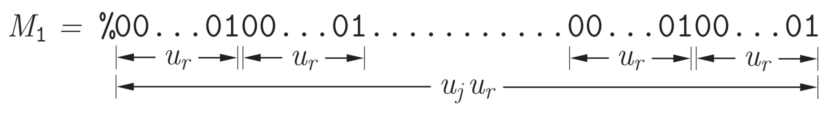

Figure 9.9: The Structure of $M_1 \in M_E^D$.

The sequence M_E^D consists of exactly $\mathrm{L} = u_l \cdot u_r$ bitstrings. Each of these bitstrings can be constructed from M_1. For $1 \leq i < \mathrm{L}$, this is done by

$$M_{i+1} = \begin{cases} \mathsf{LSL}(M_i, 1) & \text{if } i \neq k \cdot u_r, \\ \mathsf{LSL}(M_i, (u_j - 1) \cdot u_r + 1) & \text{otherwise.} \end{cases} \tag{9.15}$$

Now, let us look at the *general case*. Suppose that $D - E = \{x_{r_1}, \ldots, x_{r_\ell}\}$. The construction of M_E^D can be described as a serie of transformations

$$M_D^D \Rightarrow M_{D-\{x_{r_1}\}}^D \Rightarrow M_{D-\{x_{r_1}, x_{r_2}\}}^D \Rightarrow \cdots\cdots \Rightarrow M_E^D$$

where M_D^D corresponds to the sequence of configurations of Θ_D, thus $M_i \in M_D^D$ is given by $r_2(\mathbf{c}_i)$ for $\mathbf{c}_i \in \Theta_D$. Therefore, it is sufficient to study the transformation of $M_{E'}^D$ into $M_{E''}^D$ for $E'' = E' - \{x_{k_j}\}$ and $E' \subseteq D$. We will see that this transformation is similar to the special case discussed above.

So the sequence $M_{E'}^D$ is given for a set $E' = \{x_{k_1}, \ldots, x_{k_h}\} \subseteq D$. If $x_{k_j} \in E'$, then E' can be divided into three parts:

$$E' = \{\underbrace{x_{k_1}, \ldots, x_{k_{j-1}}}_{\mathsf{left}}, \underbrace{x_{k_j}}_{\mathsf{middle}}, \underbrace{x_{k_{j+1}}, \ldots, x_{k_\ell}}_{\mathsf{right}}\}$$

As previously, u_l', u_j' and u_r' denote the size of the state space of these sets. $M_{E'}^D = \langle M_1', \ldots, M_{\mathrm{K}}' \rangle$ consists of $\mathrm{K} = u_l' u_j' u_r'$ bitstrings, whereas $M_{E''}^D = \langle M_1'', \ldots, M_{\mathrm{L}}'' \rangle$ will have a length of $\mathrm{L} = u_l' u_r'$. It is always possible to write the index i of $M_i'' \in M_{E''}^D$ unequivocally as $i = i_1 u_r' + i_2$, where $0 \leq i_1 < u_l'$ and $0 < i_2 \leq u_r'$. $M_i'' \in M_{E''}^D$ is then given by

$$M_i'' = \mathsf{OR}\{M_{s+ku_r'}' \in M_{E'}^D : 0 \leq k < u_j'\} \tag{9.16}$$

where $s = i_1 u_j' u_r' + i_2$. Therefore, $M_i'' \in M_{E''}^D$ is the bit-wise logical "or" of u_j' bitstrings of $M_{E'}^D$. In any case, it is more efficient to treat the transformation $M_D^D \Rightarrow M_{D-\{x_{r_1}\}}^D$ as proposed at the beginning of this subsection.

9.3.4 Optimizations

There are two optimizations which are worth performing: the first optimization prevents that the same bitmask is constructed several times, whereas the second speeds up the construction of bitmasks.

Using a Cache for the Bitmasks

It is a waste of time if the same bitmask is constructed several times. This can be prevented if a cache is used. Prior to the construction of a bitmask M_E^D, we search the cache for bitmasks M_F^D, where $E \subseteq F$. If several such bitmasks are found, then the one, where $|\Theta_F|$ is smallest, is used as starting point for the

construction of M_E^D. Thus, if M_E^D is already in the cache, then there is nothing to do.

It is worthwhile to handle the special case where all variables are **binary** variables. In this case, the cache is a hierarchical data structure indexed by

- the state space of D, thus $|\Theta_D|$,
- the bitstring which represents $E \subseteq D$.

In the general case of variables with an arbitrary finite state space, we have in addition a list containing the state space of every variable of D.

Speeding up the Construction of Bitmasks

Suppose that $\{x_{j_1}, \ldots, x_{j_2}\} \subseteq D - E$ and that D can be written as

$$D = \{\underbrace{x_1, \ldots, x_{j_1-1}}_{\text{left}}, \underbrace{x_{j_1}, \ldots, x_{j_2}}_{\text{middle}}, \underbrace{x_{j_2+1}, \ldots, x_n}_{\text{right}}\}.$$

The construction of M_E^D is improved significantly if this block of variables is replaced by a new variable of which the state space has the size $|\Theta_{\{x_{j_1},\ldots,x_{j_2}\}}|$. Proceeding further by replacing such blocks of variables, a new set D' is obtained from D, where every pair of variables of $D' - E$ is separated by at least one variable of E. Most important is the fact that $M_E^{D'}$ is equal to M_E^D and that $M_E^{D'}$ can be constructed much faster than M_E^D.

9.4 Implementing the Fusion Algorithm

The fusion algorithm presented in Chapter 5 and the different heuristics for selecting the variable to eliminated next (see Section 6.4) can be implemented efficiently if the appropriate data structure is used. Practical experience (see Chapter 11) proved that a particular data structure which we called **variable-potential link list** (VPLL) is well suited for this purpose.

9.4.1 Initial State of the VPLL

Initially, the set of potentials $\Phi_0 = \{\vartheta_1, \ldots, \vartheta_m\}$ is given. The fusion algorithm then eliminates at each step i of the elimination process a variable from the current set of potentials Φ_{i-1}. For the selection of the variable to eliminate next, it is important that the set of potentials $\Phi_{i-1}(x) = \{\vartheta \in \Phi_{i-1} : x \in d(\vartheta)\}$ containing x in their domain, can be constructed rapidly for every remaining variable x. The VPLL is especially suited for this task because it contains for every variable x a list of pointers to potentials $\vartheta \in \Phi_{i-1}(x)$. Similarly, every $\vartheta \in \Phi_{i-1}$ has access to the variables in $d(\vartheta)$ through a list of pointers.

Example 9.4 Initial State of the VPLL Let $\Phi_0 = \{\vartheta_1, \vartheta_2, \vartheta_3\}$ be the initially given set of potentials. If we suppose that $d(\vartheta_1) = \{a, c\}$, $d(\vartheta_2) = \{b, c\}$, and $d(\vartheta_3) = \{a, c, d\}$, then the initial state of the VPLL is shown in Figure 9.10. ⊖

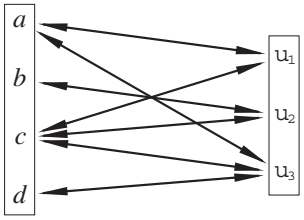

Figure 9.10: Initial State of the VPLL.

9.4.2 Reorganizing the VPLL

If the variable x_i is eliminated at step i of the elimination process, then the variable x_i, all potentials in $\Phi_{i-1}(x_i)$ and all links to these potentials have to be removed. In addition, a new potential ϑ_{m+i} has to be added. The following example shows these operations for the previous example.

Example 9.5 Eliminating a Variable from the VPLL Suppose that the variable d has to be eliminated. Because $\Phi_0(d) = \{\vartheta_3\}$, the potential ϑ_3 and links pointing to ϑ_3 have to be removed from the VPLL together with the variable d. This operations are shown on the left side of Figure 9.11. After the removal, a new potential ϑ_4 with $d(\vartheta_4) = \{a, c\}$ has to be added. Therefore, links from a and c to ϑ_4 must be inserted. This operation is shown on the right side of Figure 9.11. ⊖

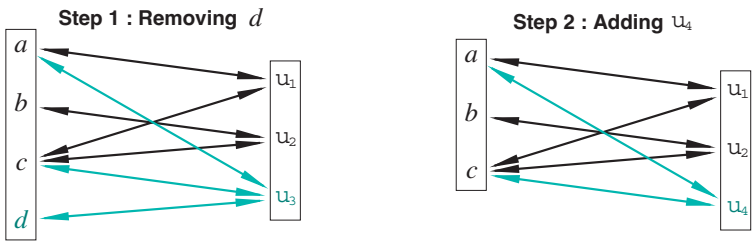

Figure 9.11: Reorganizing the VPLL.

9.4.3 Computing the Criterions

The heuristics presented in Section 6.4 use at step i of the elimination process the following criterions to select the variable to eliminate next:

$$CR_1(x) = |FI(x, \mathcal{G}^{-x_1 \ldots x_{i-1}})| \tag{9.17}$$
$$CR_2(x) = |Cl(x, \mathcal{H}^{-x_1 \ldots x_{i-1}})| \tag{9.18}$$
$$CR_3(x) = |\Theta_{Cl(x, \mathcal{H}^{-x_1 \ldots x_{i-1}})}| \tag{9.19}$$
$$CR_4(x) = \prod_{\vartheta \in \Phi_{i-1}(x)} |FS(\vartheta)| \tag{9.20}$$

The elimination of a variable changes the values $CR_1(x), \ldots, CR_4(x)$ only for a small subset of variables. Therefore, it is advantageous to compute these values for all variables once at the beginning of the elimination process. When the variable x_i has been eliminated, then $CR_2(x)$ to $CR_4(x)$ have to be recomputed only for the set of variables $Cl(x_i, \mathcal{H}^{-x_1 \ldots x_{i-1}}) - \{x_i\}$.

The computation of $CR_1(x)$ is not as easy because the set of variables for which $CR_1(x)$ has to be recomputed after the elimination of the variable x_i is not as small as for the other criterions. The recomputation is necessary for the set of variables $\{z \in Cl(y, \mathcal{H}^{-x_1 \ldots x_{i-1}}), y \in Cl(x_i, \mathcal{H}^{-x_1 \ldots x_{i-1}})\} - \{x_i\}$. This is illustrated by the following example.

Example 9.6 Recomputing the Criterions Suppose that ϑ_1, ϑ_2, ϑ_3, and ϑ_4 are potentials with domains $\{a, b\}$, $\{a, c\}$, $\{b, d\}$, and $\{c, d\}$. The hypergraph $\mathcal{H}$ given by these domains and the corresponding 2-section graph $\mathcal{G}$ are shown in Figure 9.12.

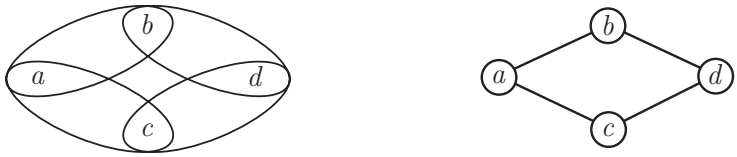

Figure 9.12: Hypergraph $\mathcal{H}$ and corresponding 2-Section Graph $\mathcal{G}$.

The value of the criterions $CR_2(x)$ to $CR_4(x)$ is determined by the closure of the variables a, b, c, and d. Initially, we have

$$Cl(a, \mathcal{H}) = \{a, b, c\} \qquad Cl(c, \mathcal{H}) = \{a, c, d\}$$
$$Cl(b, \mathcal{H}) = \{a, b, d\} \qquad Cl(d, \mathcal{H}) = \{b, c, d\}$$

The elimination of the variable a changes some of these values. It is then

$$Cl(b, \mathcal{H}^{-a}) = \{b, c, d\} \quad ; \quad Cl(c, \mathcal{H}^{-a}) = \{b, c, d\} \quad ; \quad Cl(d, \mathcal{H}^{-a}) = \{b, c, d\}$$

Above all, note that $Cl(d, \mathcal{H}) = Cl(d, \mathcal{H}^{-a})$. It is therefore not necessary to recompute $CR_2(d)$ to $CR_4(d)$ after the elimination of the variable a.

Now, let us look at the first criterion $CR_1(x)$. Initially, we have

$$\begin{array}{ll} |FI(a,\mathcal{G})| = 1 & |FI(c,\mathcal{G})| = 1 \\ |FI(b,\mathcal{G})| = 1 & |FI(d,\mathcal{G})| = 1 \end{array}$$

After the elimination of the variable a, we obtain

$$|FI(b,\mathcal{G}^{-a})| = 0 \quad ; \quad |FI(c,\mathcal{G}^{-a})| = 0 \quad ; \quad |FI(d,\mathcal{G}^{-a})| = 0$$

Therefore, the value of $CR_1(d)$ has changed from 1 to 0. In contrast, there was no change for the corresponding value of the other three criterions. $\ominus$

10
The Software Package ABEL

The predecessor of ABEL was a language called **ABL** (Lehmann, 1994) and was based on the idea of **assumption-based reasoning** (Kohlas & Monney, 1993; Kohlas & Monney, 1995). As ABL was restricted to propositional logic it was not powerfull enough to express a wide range of interesting problems. This lack of expressive power leads to the more general language **ABEL** which is the shorthand expression for **Assumption-Based Evidential Language**. We refer to (Anrig *et al.*, 1997a; Anrig *et al.*, 1997b) and especially to (Anrig *et al.*, 1999) for more information about ABEL.

ABEL is not only a language, it is also the name of a software package which includes a **solver part** for doing inference. At the time of writting these lines, the current version of the software package ABEL is version 2.2.

The continuous development is the reason why we recommend to check the **homepage of ABEL** at

`http://www-iiuf.unifr.ch/tcs/abel`

There is the main source of up-to-date information about ABEL. Among other things, there is a **tutorial** which is especially helpfull for people which are not yet familiar with ABEL. There is also a **collection of examples** which show that ABEL is a **general language** and can be used to treat problems from different areas. Finally, the **source code** of ABEL as well as **stand-alone applications** can be downloaded for several operating systems including Apple Macintosh, Windows 95/98/NT and UNIX.

In this chapter, we will give a short description of the language ABEL. Finally, the "Communication Line" example will show that the task of modeling is not as easy as it may be thought at first sight.

10.1 ABEL - the Language

The language ABEL is based on three other languages: from **Common LISP** it adopts prefix notation, from **Pulcinella** it uses the idea of the commands

`tell`, `ask`, and `empty`, and from **ABL** it inherits the concept of modules and the syntax of queries.

Working with ABEL usually involves three sequential steps. First, the given information is expressed using the command `tell`. The resulting model is called **basic knowledge base**. It describes the part of the available information that is relatively constant and static in course of time. Second, actual facts or observations about the concrete, actual situation are specified using the command `observe`. The basic knowledge base completed by observations is called **actual knowledge base**. Finally, queries about the actual knowledge base are expressed using the command `ask`.

10.1.1 Modeling Information

The command `tell` is used to build the basic knowledge base. Its syntax is as follows:

```
(tell [:key]
   <instruction-1>
   <instruction-2>
         ...
   <instruction-n>)
```

The sequence of instructions after the keyword `tell` is interpreted conjunctively and in parallel. Each instruction is one of the following:

(1) a definition of a type
(2) a definition of variables or assumptions
(3) a definition of a module
(4) a statement
(5) an instance of a module

Optionally, a key can be assigned to a `tell` command. This key is used by the command `empty`, which will be explained later in Subsection 10.1.4.

The **type** (or the **domain**) of variables and assumptions is defined using the command `type`. Of course, there are already pre-defined types, namely `integer`, `real`, and `binary`. The following examples show that it is also possible to define subsets of pre-defined types:

```
(type test (passed failed))
(type colors (red green blue yellow))
(type month (1 2 3 4 5 6 7 8 9 10 11 12))
(type year integer)
(type month (integer 1 12))
(type pos-integer (integer 0 *))
(type neg-real (real * 0))
```

Variables are defined using the command `var`. For every variable a type has to be declared. The syntax of the command `var` is as follows:

```
(var <var-1> <var-2> ... <var-n> <type>)
```

The following examples show that the type of a variable is either a pre-defined type, a user-defined type or a new type specification:

```
(var c1 c2 colors)
(var a b n integer)
(var language (french german spanish english))
(var p q r binary)
```

Assumptions are defined in a similar way using the command `ass`. In contrast to variables, probabilities can optionally be specified. The syntax of the command `ass` is as follows:

```
(ass <ass-1> <ass-2> ... <ass-n> <type> [<probabilities>])
```

Probabilities are given as a list of values between 0 and 1 summing to 1. The order in which the probabilities appear in the list corresponds to the order in which the values for the given type are defined. If there are no probabilities specified, then ABEL assumes equal probabilities for the values. Examples of defining assumptions are:

```
(ass test1 test2 test (0.8 0.2))
(ass weather (sun clouds rain) (0.5 1/3 1/6))
(ass c1 c2 c3 colors)
(ass ok? binary 0.75)
```

The last example shows that if the type of an assumption is `binary`, it is sufficient to specify the probability p of the positive literal. The probability of the negative literal is given implicitly by $1 - p$.

In the current version of ABEL, assumptions must have finite domains. In any case, the methodes presented in this thesis are only applyable if all variables and assumptions have finite domains. In the following, we therefore don't discuss models where variables are of type `integer`, `real` or a subtype of these two pre-defined types. Instead, we concentrate on models where variables are of type `binary` or have a finite set of values.

Constraints always compare two expressions and are used to restrict the possible values of variables and assumptions. The syntax is as follows:

```
(<operator> <expression-1> <expression-2>)
```

ABEL provides the operators `=`, `<>`, `<=`, `>`, `>=`, and `in`. For variables with a finite set of values, only the operators `=`, `<>`, and `in` can be used. In the case of the operator `in`, the first expression must be a variable or an assumption and the second expression must be a subdomain of the corresponding domain. The following examples show a few constraints:

```
(= c1 blue)
(in language (german spanish))
ok?
```

The last example shows that variables and assumptions of type `binary` are considered as (atomic) constraints. In addition, there are two pre-defined constraints `tautology` and `contradiction` which have the constant logical values $\top$ and $\bot$, respectively.

Statements are build of constraints using the logical connectors `and`, `or`, `not`, `->`, `<->`, and `xor`. A constraint itself is considered as an (atomic) statement. Examples of statements are:

```
(and (= c1 red) (= c2 blue))
(-> ok? (in language (german spanish)))
ok?
```

A **module** represents a step of abstraction in the modeling process. The basic idea is that similar parts of the given information are modeled only once. Each module has a name and consists of a set of parameters and a body. Parameters are variables or assumptions, the body of a module is a sequence of ABEL instructions. The syntax for a module definition is:

```
(module <name> (<param-1> <param-2> ... <param-n>)
   <instruction-1>
   <instruction-2>
         ...
   <instruction-n>)
```

Note that a type has to be specified for each of the parameters. The parameter list is therefore a sequence of variable and assumption definitions. An example of a definition of a module is as follows:

```
(module AND-GATE ((var in1 in2 out binary))
  (ass ok binary 0.99)
  (-> ok (<-> out (and in1 in2))))
```

An **instance of a module** is obtained by using the following syntax:

```
(<name> [:key] <arg-1> <arg-2> ... <arg-n>)
```

For example, the following command creates an instance of the above module:

```
(AND-GATE :A1 in1 in2 v3)
```

The types of the arguments are implicitly given by the parameter specification of the module definition. It is therefore not necessary to specify the types of the arguments outside the module. In addition, note that a key can be assigned to an instance of a module.

10.1.2 Modeling Observations

Usually, observations are added to the basic knowledge base in order to obtain the actual knowledge base. Observations describe the actual situation and my change in course of time. Therefore, it is important to separate them from the

basic knowledge base which is constructed using the command `tell`. ABEL provides the command `observe` to specify observations. Its syntax is as follows:

```
(observe [:key]
   <statement-1>
   <statement-2>
        ...
   <statement-n>)
```

The sequence of ABEL statements after the keyword `observe` is interpreted conjunctively. Examples of observations are:

```
(observe (= c1 blue))
(observe (in language (german spanish)))
(observe ok?)
```

Similar to the command `tell`, a key can be assigned to an `observe` command. This key is used by the command `empty`, which will be explained later in Subsection 10.1.4.

10.1.3 Formulating Queries

Queries about the actual knowledge base, that is the information modeled by `tell` and `observe` commands, are expressed with the command `ask`. In that way, symbolic or numerical arguments in favor or against a hypothesis can be obtained. The syntax of the `ask` command is as follows:

```
(ask <query-1>
     <query-2>
        ...
     <query-n>)
```

For a given hypothesis, different kinds of arguments may be of interest: **support**, **quasi-support**, **plausibility**, or **doubt**. The corresponding queries are:

```
(sp <statement>)
(qs <statement>)
(pl <statement>)
(db <statement>)
```

The result of such a query is a sequence of arguments, that is, a sequence of conjunctions of normal or negated ABEL constraints over assumptions.

It is also possible to ask for numerical arguments like **degree of support**, **degree of quasi-support**, **degree of possibility**, or **degree of doubt**. The corresponding queries are:

```
(dsp <statement>)
(dqs <statement>)
(dpl <statement>)
(ddb <statement>)
```

The result of such a query is a number which is between 0 and 1. This number represents the probability of the result obtained from the corresponding symbolic query. It is computed using the a priori probabilities which were assigned to the assumptions.

10.1.4 Further Facilities

Sometimes, it is necessary to delete the actual knowledge base or parts of it. For this purpose, ABEL provides the command `empty`. If `empty` is called without arguments, the actual knowledge base is deleted. If the keyword `observe` is supplied, only the observations are deleted. Moreover, by supplying one or several keywords used within `tell` or `observe` commands, only the corresponding parts of the actual knowledge base are deleted.

```
(empty)
(empty observe)
(empty :part1 :part2)
(empty :first-measure)
(empty :second-measure)
```

To obtain more readable ABEL code, **comments** can be introduced into the code. There are two different ways to write comments:

```
#| This is a comment over one
   or several lines |#
; This is another comment
```

10.2 The "Communication Line" Example

In this section, we will take a look at the "Communication Line" example, which has already been used in previous chapters. The main goal is to show how this example can be modeled in ABEL.

We will consider the case of four computers connected as shown in Figure 10.1. The point of interest will be whether a mail which is send from the first computer on the left reaches the last computer on the right.

Figure 10.1: A Communication Line with 4 Computers.

The first step is to determine the variables and assumptions. Following the train of thought of Section 2.8, four variables and six assumptions are required. The binary variables `x0`, ..., `x3` represent the information whether or not the

corresponding computer has received the mail. For example, `x2` is true represents the situation, where the third computer from the left has received the mail. In contrast, `x2` is false represents the situation, where it has not received the mail.

We know that two wires of different quality were used when setting up the connections and that these wires are not completely reliable. To express this uncertainty, a binary assumption is needed for each of the six wires. `a1`, `a2`, and `a3` are used for the wires which are of better quality, whereas `b1`, `b2`, and `b3` are used for the wires which are of lower quality. The definition of these variables and assumptions in ABEL is as follows:

```
(tell
  (var x0 x1 x2 x3 binary)
  (ass a1 a2 a3 binary 0.8)
  (ass b1 b2 b3 binary 0.5))
```

It is often not easy to justify the a priori probabilities which are assigned to the assumptions. Here, however, we don't want to concentrate on this subject. Therefore, just keep in mind that these values express the fact that two wires of different quality were used.

Now that we have defined the variables and assumptions, we can turn towards the connections between the four computers. In order that a computer receives the mail, note that at least one of the two incoming wires should be working correctly. Furthermore, the computer on its left must also have received the mail. These rules are expressed in ABEL as follows:

```
(tell
  (-> (and x0 a1) x1)
  (-> (and x1 a2) x2)
  (-> (and x2 a3) x3)

  (-> (and x0 b1) x1)
  (-> (and x1 b2) x2)
  (-> (and x2 b3) x3))
```

For example, `(-> (and x1 a2) x2)` means the following: if the second computer from the left has receive the mail and the wire represented by `a2` is working correctly, then also the third computer receives the mail.

By adding observations, the basic knowledge base could now be completed to reflect the concrete situation. For example, we could have observed that a particular wire is broken or that a particular computer has received the mail. Here, however, we do not perform any observations. Therefore, the actual knowledge base is equal to the basic knowledge base.

We have already mentioned that the main point of interest is whether or not the mail from the first computer on the left reaches the last computer. For this, the set of supporting arguments for the hypothesis `(-> x0 x3)` has to be computed. The corresponding query is the following:

```
(ask (sp (-> x0 x3)))
```

The result consists of the following eight arguments:

```
QUERY: (SP (-> X0 X3))
  23.3% : A1 A2 A3
  14.6% : A2 A3 B1
  14.6% : A1 A3 B2
  14.6% : A1 A2 B3
   9.1% : A1 B2 B3
   9.1% : A3 B1 B2
   9.1% : A2 B1 B3
   5.7% : B1 B2 B3
```

Each one of these arguments describes a situation where the mail from the first computer reaches the last computer. For example, this is the case if the wires represented by `a1`, `a2`, and `a3` are working correctly.

It may also be interesting to compute the probability that the mail reaches the last computer. For this, the following query is used:

```
(ask (dsp (-> x0 x3)))
```

The resulting value 0.729 can be interpreted as the strength of belief that the mail reaches the last computer.

```
QUERY: (DSP (-> X0 X3))
  0.729
```

Sometimes, modeling is not as easy as it may be thought at first sight. We can also be surprised by the result of a query if we expected a different result. For example, let us consider the hypothesis `(-> x3 x0)` instead of `(-> x0 x3)`. This hypothesis corresponds to a situation where a mail is send from right to left instead from left to right.

```
(ask (dsp (-> x3 x0)))
```

Intuitively, we may have expected that the result will be the same as the previous one. However, this is not at all the case:

```
QUERY: (DSP (-> X3 X0))
  0.000
```

Therefore, the mail which is send from the computer on the right never reaches the first computer on the left. This is not what we might have expected. Therefore, what is wrong ?

Of course, the result is correct and it is not a fault of ABEL. If we look carefully at how we have modeled a connection, we notice that it was modeled as a directed connection from left to right. As an example, the implication `(-> (and x1 a2) x2)` says something about mails which are send from left to right, but nothing about mails which are send from right to left.

The model has to be reformultated if we want to consider also situations, where mails are send from right to left. The new model still consists of the same variables and assumptions, however, the connections are modeled differently. Therefore, we should first delete the old knowledge base:

```
(empty)
```

The ABEL code for the new model is as follows:

```
(tell
  (var x0 x1 x2 x3 binary)
  (ass a1 a2 a3 binary 0.8)
  (ass b1 b2 b3 binary 0.5)

  (-> a1 (<-> x0 x1))
  (-> a2 (<-> x1 x2))
  (-> a3 (<-> x2 x3))

  (-> b1 (<-> x0 x1))
  (-> b2 (<-> x1 x2))
  (-> b3 (<-> x2 x3)))
```

For example, `(-> a2 (<-> x1 x2))` can be interpreted as follows: if the wire represented by `a2` is working correctly, the second and the third computer share the same information. Thus, each computer passes mails which it has received to a connected computer if the corresponding wire is working correctly.

Let us now compute the same queries again:

```
(ask (dsp (-> x0 x3))
     (dsp (-> x3 x0)))
```

The result of these queries is as we have expected:

```
QUERY: (DSP (-> X0 X3))
  0.729
QUERY: (DSP (-> X3 X0))
  0.729
```

11
Applications

In the previous chapter, the modeling language ABEL was presented. In the following, we will consolidate and extent the knowledge about ABEL by considering examples in the domain of **Medical Diagnostic**, **Digital Circuits**, **Communication Networks**, **Public-key Cryptography**, and **Information Retrieval** respectively. For example, we will show that large digital circuits can be expressed by using of the **module** concept.

The main goal of this chapter is to build a bridge between the modeling language ABEL and the computational theory presented in previous chapters. For this reason, we will look at the four **heuristics** to determinate the **variable elimination sequence** which have been discussed in Section 6.4. We will see that there is no "best" heuristic. Nevertheless, there may be huge differences in the needed time by each one of the heuristics. For some examples, computations are even not feasible for some of the four heuristics. By the application of such a heuristic, a corresponding **join tree** is constructed. We will show how unnecessary nodes can be eliminated by using the **simplifications** proposed in Section 6.5. Furthermore, we will look at the method described in Chapter 8 which consists of an inward propagation phase followed by a second partial inward propagation.

The algorithms were implemented in the programming language Lisp using the software package *Macintosh Common Lisp*. The times required for computations were measured on computer of Apple Power Macintosh G3 type running with 400 MHz and having 128 MByte built-in memory.

In the following, we will start with a fictitious example which was first introduced in (Lauritzen & Spiegelhalter, 1988) in order to illustrate the use of Bayesian networks. This example will be treated by assumption-based reasoning. The main objective will be to show that assumption-based reasoning is **non-monotone**, in the sense that additional information can decrease the belief of a hypothesis.

11.1 Medical Diagnostic

A doctor has to decide whether a patient, who complains about shortness-of-breath, suffers from bronchitis, lung cancer, or tuberculosis. The doctor's medical knowledge is summarized as follows:

> "Shortness-of-breath (dyspnoea) may be due to tuberculosis, lung cancer, or bronchitis, or none of them, or more than one of them. A recent visit to an under-developed country increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis; nor does the presence or absence of dyspnoea."

In Figure 11.1, this small piece of medical knowledge is shown as a causal network. Causal relations are described by uncertain implications and direction of causality is from top to bottom. Note that some effects have more than one cause and that some causes produce more than one effect.

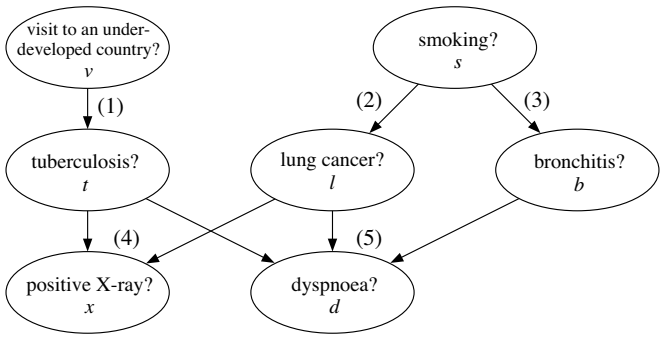

Figure 11.1: Causal Network for the "Chest Clinic" Example.

All causal relations of Figure 11.1 are uncertain. Therefore, for each of the eight causal relations an assumption is needed. In addition, another five assumptions are needed because it is furthermore assumed that for all effects there are also other (unknown) causes. The fictitious medical knowledge is modeled with the thirteen assumptions $a_1, \ldots, a_{13}$ as follows:

$$
\begin{array}{ll}
(1) & v \wedge a_1 \rightarrow t, \;\; a_2 \rightarrow t, \;\; t \rightarrow (v \wedge a_1) \vee a_2; \\
(2) & s \wedge a_3 \rightarrow \ell, \;\; a_4 \rightarrow \ell, \;\; \ell \rightarrow (s \wedge a_3) \vee a_4; \\
(3) & s \wedge a_5 \rightarrow b, \;\; a_6 \rightarrow b, \;\; b \rightarrow (s \wedge a_5) \vee a_6; \\
(4) & t \wedge a_7 \rightarrow x, \;\; \ell \wedge a_8 \rightarrow x, \;\; a_9 \rightarrow x, \\
 & x \rightarrow (t \wedge a_7) \vee (\ell \wedge a_8) \vee a_9; \\
(5) & t \wedge a_{10} \rightarrow d, \;\; \ell \wedge a_{11} \rightarrow d, \;\; b \wedge a_{12} \rightarrow d, \;\; a_{13} \rightarrow d \\
 & d \rightarrow (t \wedge a_{10}) \vee (\ell \wedge a_{11}) \vee (b \wedge a_{12}) \vee a_{13}.
\end{array}
$$

If the probabilities

$$p(a_1) = 0.1,\ \ p(a_2) = 0.01,\ \ p(a_3) = 0.2,\ \ p(a_4) = 0.01,\ \ p(a_5) = 0.3,$$
$$p(a_6) = 0.1,\ \ p(a_7) = 0.9,\ \ p(a_8) = 0.8,\ \ p(a_9) = 0.1,\ \ p(a_{10}) = 0.9,$$
$$p(a_{11}) = 0.8,\ \ p(a_{12}) = 0.7,\ \ p(a_{13}) = 0.1,$$

are supposed, then the knowledge base can be written as follows:

```
(tell
  (var visit tuberculosis x-ray lung-cancer
       smoker bronchitis dyspnoea binary)
  (ass a1 a6 a9 a13 binary 0.1)
  (ass a2 a4 binary 0.01)
  (ass a3 binary 0.2)
  (ass a5 binary 0.3)
  (ass a7 a10 binary 0.9)
  (ass a8 a11 binary 0.8)
  (ass a12 binary 0.7)

  (-> (and visit a1) tuberculosis)
  (-> a2 tuberculosis)
  (-> tuberculosis (or (and visit a1) a2))

  (-> (and smoker a3) lung-cancer)
  (-> a4 lung-cancer)
  (-> lung-cancer (or (and smoker a3) a4))

  (-> (and smoker a5) bronchitis)
  (-> a6 bronchitis)
  (-> bronchitis (or (and smoker a5) a6))

  (-> (and lung-cancer a7) x-ray)
  (-> (and tuberculosis a8) x-ray)
  (-> a9 x-ray)
  (-> x-ray (or (and lung-cancer a7) (and tuberculosis a8) a9))

  (-> (and tuberculosis a10) dyspnoea)
  (-> (and lung-cancer a11) dyspnoea)
  (-> (and bronchitis a12) dyspnoea)
  (-> a13 dyspnoea)
  (-> dyspnoea (or (and tuberculosis a10)
                   (and lung-cancer a11)
                   (and bronchitis a12) a13)))
```

The fact that the patient is suffering from dyspnoea can be introduced using the command `observe`:

```
? (observe dyspnoea)
```

Degrees of support and degrees of plausibility for tuberculosis, lung cancer, and bronchitis can then be computed:

```
? (ask (dsp tuberculosis) (dsp lung-cancer) (dsp bronchitis))
       (dpl tuberculosis) (dpl lung-cancer) (dpl bronchitis))
QUERY: (DSP TUBERCULOSIS)
  0.118
QUERY: (DSP LUNG-CANCER)
  0.270
QUERY: (DSP BRONCHITIS)
  0.486
QUERY: (DPL TUBERCULOSIS)
  0.206
QUERY: (DPL LUNG-CANCER)
  0.367
QUERY: (DPL BRONCHITIS)
  0.591
```

The differences between degrees of support and degrees of plausibility are relatively large. Also, none of the three hypotheses are strongly supported or very plausible. This is a sign that additional observations are required. From a discussion with the patient the doctor then learns that the patient has recently visited an under-developed country.

```
? (observe visit)
```

This additional fact increases the degree of support for tuberculosis:

```
? (ask (dsp tuberculosis) (dsp lung-cancer) (dsp bronchitis)
       (dpl tuberculosis) (dpl lung-cancer) (dpl bronchitis))
QUERY: (DSP TUBERCULOSIS)
  0.206
QUERY: (DSP LUNG-CANCER)
  0.270
QUERY: (DSP BRONCHITIS)
  0.486
QUERY: (DPL TUBERCULOSIS)
  0.206
QUERY: (DPL LUNG-CANCER)
  0.367
QUERY: (DPL BRONCHITIS)
  0.591
```

It is still not sufficient for the doctor to make a decision. After he observes that the patient is a smoker he obtains more support for lung cancer and bronchitis:

```
? (observe smoker)

? (ask (dsp tuberculosis) (dsp lung-cancer) (dsp bronchitis)
       (dpl tuberculosis) (dpl lung-cancer) (dpl bronchitis))
QUERY: (DSP TUBERCULOSIS)
  0.206
QUERY: (DSP LUNG-CANCER)
  0.367
QUERY: (DSP BRONCHITIS)
  0.591
```

```
QUERY: (DPL TUBERCULOSIS)
  0.206
QUERY: (DPL LUNG-CANCER)
  0.367
QUERY: (DPL BRONCHITIS)
  0.591
```

Degrees of support and degrees of plausibility are now equal. But the doctor is not yet satisfied as he is not able to discriminate between lung cancer and bronchitis. So, he orders a chest X-ray test which turns out to be negative:

```
? (observe (not x-ray))

? (ask (dsp tuberculosis) (dsp lung-cancer) (dsp bronchitis)
       (dpl tuberculosis) (dpl lung-cancer) (dpl bronchitis))
QUERY: (DSP TUBERCULOSIS)
  0.062
QUERY: (DSP LUNG-CANCER)
  0.062
QUERY: (DSP BRONCHITIS)
  0.760
QUERY: (DPL TUBERCULOSIS)
  0.062
QUERY: (DPL LUNG-CANCER)
  0.062
QUERY: (DPL BRONCHITIS)
  0.760
```

Now, the doctor is satisfied because he is able to state a very clear diagnosis. Bronchitis has a quite strong degree of support, whereas lung cancer and tuberculosis are very unplausible. Figure 11.2 summarizes the process of finding the diagnosis.

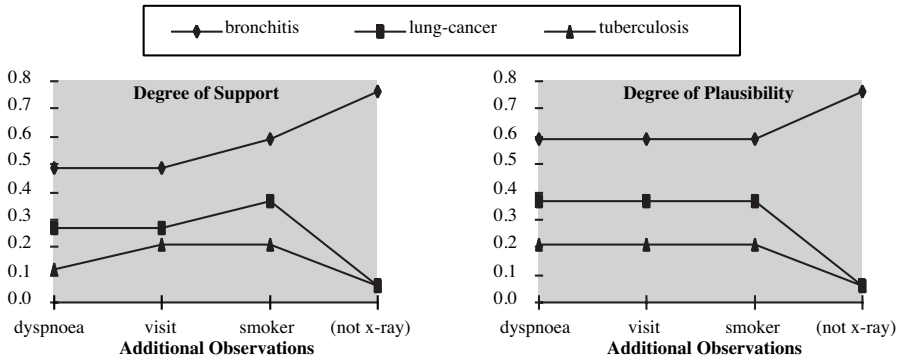

Figure 11.2: Degree of Support and Degree of Plausibility.

Note that the last observation decreases the degrees of support and the degrees of plausibility of lung-cancer and tuberculosis significantly. This shows that assumption-based reasoning is **non-monotone**, in the sense that additional information can decrease the belief of hypotheses.

Concerning computations, this example is interesting in the sense that observations are added one after another to the basic knowledge base given by the doctor's medical knowledge. Usually, such **observations** allow to construct better join trees. Therefore, we proposed in Section 6.5 a special treatment of observations. However, a join tree obtained in that way does not necessarily have less nodes. For example, the first join tree on the left side of Figure 11.3 corresponds to the doctor's medical knowledge. It has less nodes than the other join trees which were obtained by such a special treatment of observations. No matter how, the quality of a join tree is mainly determined by the size of its nodes and not by the number of nodes. From this point of view, the first join tree on the left side of Figure 11.3 is worse than the others.

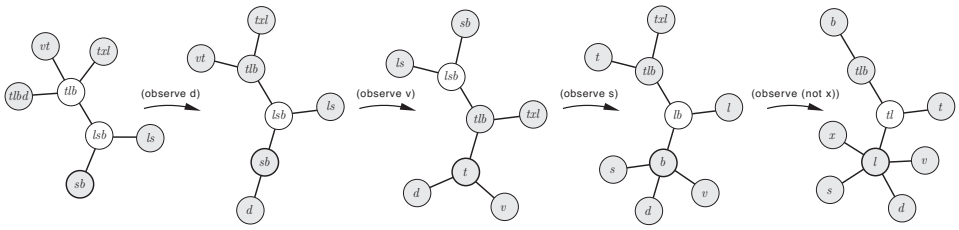

Figure 11.3: Join Trees for the "Chest Clinic" Example.

Looking at the join trees of Figure 11.3, the question could be asked whether it is a good idea to reconstruct the join tree if additional observations become available. Of course, here for this small example, it does not really matter because the reconstruction of the join tree does not need much time. However, just imagine a situation, where several thousand variables were used to express a certain medical knowledge. In this case, the reconstruction of the join tree is probably not possible because this would need too much time. Therefore, it is much more efficient to perform then only a so-called **updating** of the join tree by adding the corresponding observation to one of the nodes and by performing an outward propagation phase afterwards.

11.2 Diagnostics of Digital Circuits

Nowadays, digital circuits are used in a wide range of products. For example, digital circuits are used in cars amoung others for the air-conditioning system, the brake system and for measuring the current speed. Moreover, a television set is today almost a computer and can be considered in some sense as a huge digital circuit.

In the following, we will see that digital circuits can be formulated as a propositional argumentation system $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$. We will see that ABEL's concept of **modules** is very appropriate in this case because it allows to describe complex digital circuits. Further, we will look at the different **heuristics** for de-

terminating the elimination sequence which have been discussed in Section 6.4. Finally, the five **simplifications** proposed in Section 6.5 are discussed.

A digital circuit is build by connecting very simple gates together. Three examples of such gates are shown in Figure 11.4. Each of these gates computes the value of its output *out* as a function of its two inputs in_1 and in_2. For example, the *and-gate* on the left side of Figure 11.4 computes the value 1 only if its two inputs are 1 as well. Otherwise, if either in_1 or in_2 is 0, then the value of *out* is 0 as well. If we identify 0 with false and 1 with true, then this corresponds to the computation of the **logical and** of in_1 and in_2. Similarly, the *or-gate* computes the **logical or** and the *xor-gate* the **logical exclusive or** of the corresponding two inputs.

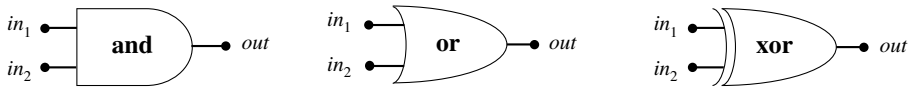

Figure 11.4: Three simple Gates.

It can happen that a simple gate is defective. In this case, we assume that anything could be obtained for the corresponding output *out*. However, note that simple gates are rarely defective. We may even be able to a priori assign corresponding probabilities to the three simple gates. These probabilities may be obtained for example from statistical investigations about failure rates of these gates. In the following, the probabilities 0.99, 0.98, and 0.95, respectively, will be assigned to simple gates. In ABEL, the definition of the simple gates is as follows:

```
(tell
  (module AND-GATE ((var in1 in2 out binary))
    (ass ok binary 0.99)
    (-> ok (<-> out (and in1 in2))))

  (module OR-GATE ((var in1 in2 out binary))
    (ass ok binary 0.98)
    (-> ok (<-> out (or in1 in2))))

  (module XOR-GATE ((var in1 in2 out binary))
    (ass ok binary 0.95)
    (-> ok (<-> out (xor in1 in2)))))
```

The parameter list of each module contains for both inputs in_1 and in_2 as well as for the output *out* a corresponding binary variable. In addition, a local assumption *ok* with the corresponding probability is created each time a module is initiated. Finally, note that the behaviour of a gate is only specified in the case, where it is working correctly.

By connecting simple gates together, more complex digital circuits are obtained. Usually, such a digital circuit implements a certain input-output re-

lation. Therefore, it computes the values of its outputs as a function of its inputs. For example, consider the digital circuit on the left side of Figure 11.5. It is a so-called *1-BIT-ADDER* and has three input wires in_1, in_2, and in_c, respectively, and two output wires *out* and out_c, respectively.

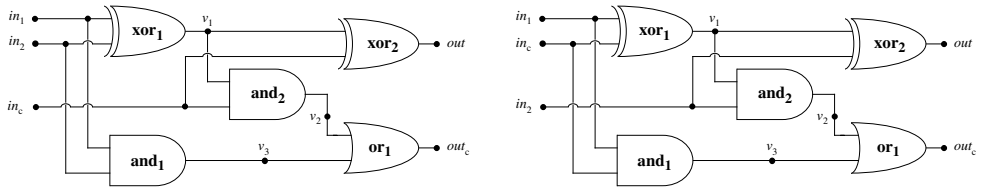

Figure 11.5: Two Digital Circuits for Binary Addition.

The digital circuit on the left side of Figure 11.5 can be modeled in ABEL as follows:

```
(tell
  (XOR-GATE :X1 in1 in2 v1)
  (XOR-GATE :X2 inc v1 out)
  (AND-GATE :A1 in1 in2 v3)
  (AND-GATE :A2 inc v1 v2)
  (OR-GATE  :O1 v2 v3 outc))
```

It is not necessary to define the variables explicitly because the definition is already given implicitly by the parameter list of the modules. In addition, note that a key is assigned to each instance of a module. In that way, we will be able to access later the local assumption *ok* of the corresponding module.

The input-output relation implemented by the digital circuit on the left side of Figure 11.5 is given in Table 11.1. For each possible configuration of the inputs, the corresponding values of the outputs are given. No matter how, different digital circuits can implement the same input-output relation. For example, the digital circuit on the right side of Figure 11.5 implements also the input-output relation of Table 11.1. Both digital circuits of Figure 11.5 are very similar. The only difference is the renaming of the inputs.

Suppose now that $in_1 = 0$, $in_2 = 0$, and $in_c = 0$. Normally, it should then be $out = 0$ and $out_c = 0$. However, suppose that $out = 1$ and $out_c = 0$ are measured instead. In this case, it is not possible that all five gates are working correctly. Instead, one or several gates must be defective. The main goal in diagnostics of digital circuits is to find the set of defective gates.

To find the set of defective gates, let us first introduce the observations:

```
? (observe (not in1) (not in2) (not inc) out (not outc))
```

Given these observations, it is now interesting to compute the degree of belief that each of the five simple gates is not working correctly. This is done as follows:

in_1	in_2	in_c	out	out_c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 11.1: Binary Addition

```
? (ask (dsp (not X1.ok)) (dsp (not X2.ok)) (dsp (not A1.ok))
       (dsp (not A2.ok)) (dsp (not O1.ok)))
QUERY: (DSP (NOT X1.OK))
  0.513
QUERY: (DSP (NOT X2.OK))
  0.513
QUERY: (DSP (NOT A1.OK))
  0.010
QUERY: (DSP (NOT A2.OK))
  0.010
QUERY: (DSP (NOT O1.OK))
  0.020
```

Therefore, the two xor-gates have quite a high support that they are not working correctly. In contrast, the other three gates have a very low support (but it is still possible that one or even all of them are defective). In order to discriminate between the xor-gates, a second measurement yields $v_1 = 1$:

```
? (observe v1)
```

We obtain then the following results:

```
? (ask (dsp (not X1.ok)) (dsp (not X2.ok)) (dsp (not A1.ok))
       (dsp (not A2.ok)) (dsp (not O1.ok)))
QUERY: (DSP (NOT X1.OK))
  1.000
QUERY: (DSP (NOT X2.OK))
  0.050
QUERY: (DSP (NOT A1.OK))
  0.010
QUERY: (DSP (NOT A2.OK))
  0.010
QUERY: (DSP (NOT O1.OK))
  0.020
```

The first xor-gate must be defective. However, again, note that it can still be that one or several of the other gates are defective.

The above example has illustrated the process of finding defective gates. In the following, we will take a look at larger digital circuits. The problem of finding sets of defective gates is then not that easy anymore. The *1-BIT-ADDER* will be used as building block for constructing larger circuits. For example, consider the digital circuit on the left side of Figure 11.6. It was obtained by connecting two *1-BIT-ADDER* together and represents thus a *2-BIT-ADDER* which is used to compute the sum of two binary numbers. These numbers are given by (in_1, in_3) and (in_2, in_4), respectively.

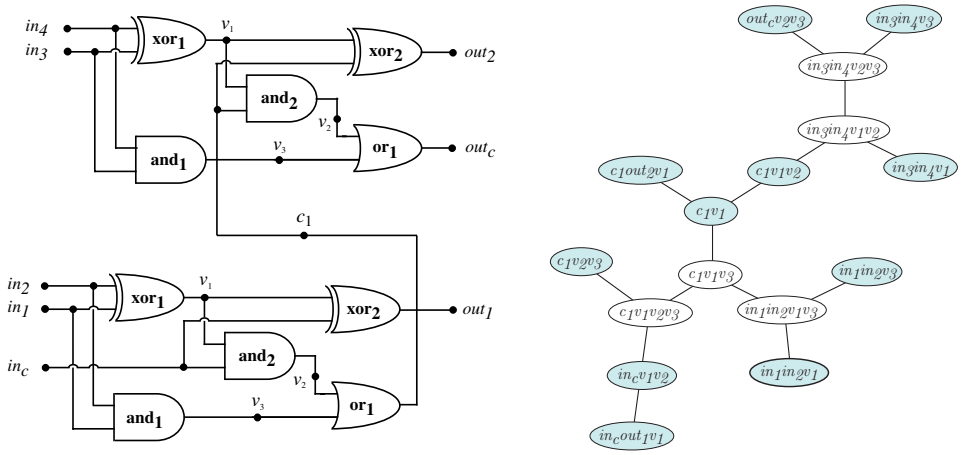

Figure 11.6: Digital Circuit for Binary Addition.

In order to use the *1-BIT-ADDER* as building block for larger digital circuits, we have to formulate it as a module. This is done as follows:

```
(tell
  (module 1-BIT-ADDER ((var in1 in2 inc out outc binary))
    (var v1 v2 v3 binary)
    (XOR-GATE :X1 in1 in2 v1)
    (XOR-GATE :X2 inc v1 out)
    (AND-GATE :A1 in1 in2 v3)
    (AND-GATE :A2 inc v1 v2)
    (OR-GATE  :O1 v2 v3 outc)))
```

Thus, it has five input-output variables and three additional local variables. The initiation of a *2-BIT-ADDER* would then be as follows:

```
(tell
  (1-BIT-ADDER :A1 in1 in2 inc out1 c1)
  (1-BIT-ADDER :A2 in3 in4 c1 out2 outc))
```

In the following, we will consider a *N-BIT-ADDER* for $N = 64$, 128, 192, and 256, respectively. We will even consider two different types of *N-BIT-ADDER*:

the first type is constructed using *1-BIT-ADDER* shown on the left side of Figure 11.5, whereas *1-BIT-ADDER* shown on the right side of Figure 11.5 are used for the second type. To each one of these digital circuits, the four **heuristics** for determinating the variable elimination sequence discussed in Section 6.4 will be applied. Furthermore, the five **simplifications** proposed in Section 6.5 will be discussed.

We will consider the particular situation, where $0 + 0$ gives 2^{N-1} as result. Thus, the highest bit represented by out_N has an incorrect value. For each digital circuit, a propositional argumentation system $\mathcal{AS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A})$ is constructed. Table 11.2 shows that $\mathcal{AS}_{\mathcal{P}}$ involves many variables and assumptions. Furtermore, the set of clauses Σ which is obtained for the knowledge base ξ is quite big.

Example	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\Sigma\rvert$
64-BIT-ADDER	449	320	1282
128-BIT-ADDER	897	640	2562
192-BIT-ADDER	1345	960	3842
256-BIT-ADDER	1793	1280	5122

Table 11.2: Characteristics of the Examples, (both types)

The four **heuristics** for determinating the variable elimination sequence have been applied to these examples. The corresponding time needed for the inward propagation is shown in Table 11.3 for the first type of *N-BIT-ADDER* and in Table 11.4 for the second type of *N-BIT-ADDER*. Note that binary trees as described in Subsection 9.2.1 were used for computations. The use of hash tables (see Subsection 9.2.2) would result in even shorter response times.

	Heuristic (One Step Look Ahead)			
	SC	SC-FFS	SC-FFS-IS	FFI
64-BIT-ADDER	0.256	0.267	0.266	0.526
128-BIT-ADDER	0.618	0.664	0.644	1.834
192-BIT-ADDER	1.069	1.147	1.159	4.287
256-BIT-ADDER	1.587	1.731	1.717	8.318

Table 11.3: Time [sec] used for Inward Propagation, (1st type)

Thus, more or less the same time is spent for both types of *N-BIT-ADDER*. Almost always, the least amount of time is needed for the heuristic *OSLA - Smallest Clique*, whereas *OSLA - Fewest Fill-ins* is much worse than the other

	Heuristic (One Step Look Ahead)			
	SC	SC-FFS	SC-FFS-IS	FFI
64-BIT-ADDER	0.250	0.247	0.256	0.492
128-BIT-ADDER	0.589	0.611	0.619	1.734
192-BIT-ADDER	0.981	1.076	1.039	4.052
256-BIT-ADDER	1.476	1.601	1.563	7.879

Table 11.4: Time [sec] for Inward Propagation, (2nd type)

heuristics. The reason is that the elimination of a variable necessitates a recomputation for a certain subset of the variables. This recomputation is performed fastest for the former heuristic whereas the latter heuristic needs much more time. Here, it was not advantageous that *OSLA - Fewest Fill-ins* spends much more time to determine the variable elimination sequence. The variable elimination sequences obtained by the other heuristics are not really that much worse. However, it is often valuable to spend more time for determinating a better variable elimination sequence because the variable elimination sequence determinates heavily the time needed for the inward propagation. The recomputation is also the reason why the remaining three heuristics are a little bit slower than *OSLA - Smallest Clique*.

In Section 6.5, we proposed five **simplifications** to eliminate unnecessary nodes of the join tree. Figure 11.7 shows the results obtained for the two types of *256-BIT-ADDER* and the four heuristics. There are always six columns which go together. The first column represents the number of nodes if no simplification is performed. Similarly, the i-th column represents the number of nodes if the first $i-1$ simplifications are performed. Thus, the last column corresponds to the situation, where all five simplifications are performed. Here, there are always 5635 nodes if no simplification is performed. If only the first simplification is performed, the join tree has 1540 nodes less. Finally, the join tree consists of about 2560 nodes if all simplifications are performed. The least number of nodes obtained is 2304 for the heuristic *OSLA - Smallest Clique, Fewest Focal Sets, Initial Structure*.

Figure 11.8 shows the size of nodes of the join tree obtained for the two types of *256-BIT-ADDER* in the case where all five simplifications are performed. Especially, note that the largest nodes are of size three.

A size of three for the largest nodes is pretty small. The small size is one of the reasons why the inward propagation phase took not much time. Another important reason is the linear structure of a *N-BIT-ADDER*. Such a linear structure is also recognizable in the corresponding join tree (see Figure 11.6). In addition, each node of the join tree has only a few neighbor nodes and incoming messages have only a few focal sets. Therefore, each node of the join tree can compute the message to its inward neighbor quite fast.

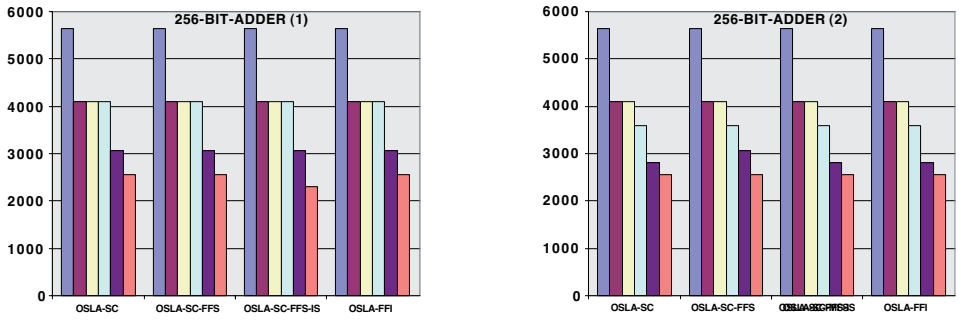

Figure 11.7: Five Simplifications of the Join Tree.

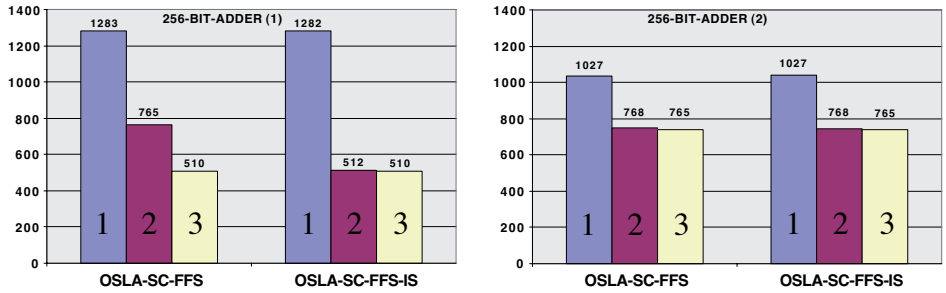

Figure 11.8: Size of the Nodes of the Join Tree.

11.3 Communication Networks

A communication network consists of **nodes** which are connected by **communication wires**. If one or several nodes or communication wires are not working correctly, then some point-to-point connections may be impossible. For the purpose of simplification, we will suppose that only communication wires can break down. If the **probability** that a wire breaks down is given for each of the communication wires, the problem is then to compute the reliability of the communication between two different nodes.

Communication networks can easily be described by **directed graphs**. For example, consider the communication network shown on the left side of Figure 11.9. It consists of the nodes u, v, w, x, and y. These nodes are connected by communication wires $w_1, \ldots, w_6$. Note that the communication wires w_1, w_3, and w_6, respectively, are bi-directed, whereas w_2, w_5, and w_5, respectively, are directed.

The problem of this example is to compute the reliability of the communication between the nodes u and y. This corresponds to compute all possible **communication paths** between these two nodes and then derive the reliability

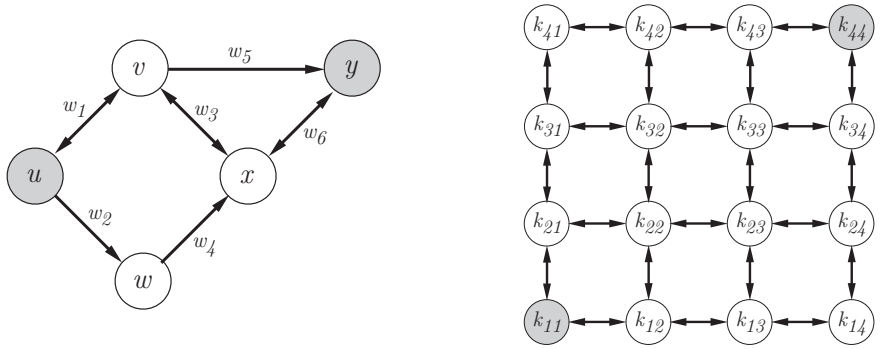

Figure 11.9: Size of the Nodes of the Join Tree.

by using the given failure probabilities. Here, the four possible communication paths between nodes u and y are the following:

$$\langle w_1 - w_5 \rangle \; ; \; \langle w_1 - w_3 - w_6 \rangle \; ; \; \langle w_2 - w_4 - w_6 \rangle \; ; \; \langle w_2 - w_4 - w_3 - w_5 \rangle$$

Similarly, $\langle w_6 - w_3 - w_1 \rangle$ is the only possible communication path for the inverse communication, thus between y and u.

A communication network can easily be modeled in ABEL. For example, the communication network shown on the left side of Figure 11.9 can be formulated in ABEL by defining binary variables for the nodes and by defining binary assumptions for the communication wires:

```
(tell
  (var u v w x y binary)
  (ass ok1 binary 0.8)
  (ass ok2 binary 0.7)
  (ass ok3 binary 0.9)
  (ass ok4 binary 0.6)
  (ass ok5 binary 0.8)
  (ass ok6 binary 0.9))
```

Assumptions `ok1`, ..., `ok6` represent the communication wires $w_1, \ldots, w_6$. Note that above we have assumed that independent probabilities are given for the six communication wires. Finally, the network can also be modeled very easily. For each directed communication wire an implication of the form $ok_i \rightarrow (n_1 \rightarrow n_2)$ is needed. Similarly, an implication of the form $ok_i \rightarrow (n_1 \leftrightarrow n_2)$ is required for each bi-directed communication wire:

```
(tell
  (-> ok1 (<-> u v))
  (-> ok2 (-> u w))
  (-> ok3 (<-> v x))
  (-> ok4 (-> w x))
  (-> ok5 (-> v y))
```

```
  (-> ok6 (<-> x y)))
```

The reliability of the communication between the nodes u and y can now be obtained by the following query:

```
? (ask (dsp (-> u y)))
QUERY: (DSP (-> U Y))
  0.857
```

This result can be interpreted as the probability that a message which is sent from node u reaches the node y assuming that the communication wires break down according to the given failure probabilities. In a similar way, the reliability of the inverse communication can be computed:

```
? (ask (dsp (-> y u)))
QUERY: (DSP (-> Y U))
  0.648
```

The communication between y and u is therefore less reliable then the communication between u and y. This is also what we have expected since there is only one possible communication path between y and u.

In the following, we will consider communication networks as shown on the right side of Figure 11.9. The nodes are arranged as a grid so that each node is connected with its neighbor nodes by bi-directed communication wires. In particular, we will look at communication networks where each line and each column consists of $n = 4$, 5, 6, 7, and 8 nodes. Such a communication network has n^2 nodes and $2n(n-1)$ commmunication wires. The modeling in ABEL for $n = 2$ is for example as follows:

```
(tell
  (var k11 k12 k21 k22 binary)
  (ass ok1 ok2 ok3 ok4 binary 0.8)

  (-> ok1 (<-> k11 k12))
  (-> ok2 (<-> k11 k21))
  (-> ok3 (<-> k12 k22))
  (-> ok4 (<-> k21 k22)))
```

Thus, the reliability of each communication wire is assumed to be 0.8. Above all, we will be interested in the reliability of a communication between the two nodes in opposite corners. For $n = 2$, the query is as follows:

```
? (ask (dsp (-> k11 k22)))
```

In order to answer the query, first, a probabilistic argumentation system $\mathcal{PAS}_{\mathcal{P}} = (\xi, \mathcal{P}, \mathcal{A}, \Pi)$ is constructed. The method presented in Chapter 8 is then used. Thus, an inward propagation phase is performed during which a join tree is constructed. In that way, the value $dqs(\bot, \xi)$ is obtained on the root node. Next, the negated hypothesis is added to appropriate nodes and a partial inward propagation is performed afterwards. The resulting value $dqs(h, \xi)$ allows then to compute $dsp(h, \xi)$.

Table 11.5 shows for each of the four **heuristics** presented in Section 6.4 the time needed for the (complete) inward propagation. Table 11.6 contains the time used for the partial inward propagation. For the computations, binary trees as described in Subsection 9.2.1 were used. Notice especially the strong increase of the amount of time. In addition, there are huge differences between the heuristics. Apparently, *OSLA-SC-FFS* is not well suited for this example here. Clearly, all heuristics eliminate first the variables which represent the four nodes in the corners. Then, variables representing nodes on the border are elimininated. The main difference in the generated variable elimination sequence is if the size of the domain for all remaining variables is five. Obviously, the choice of the variable to eliminate next determines heavily the time needed for the propagation.

	Heuristic (One Step Look Ahead)			
	SC	SC-FFS	SC-FFS-IS	FFI
GRID-4	0.023	0.026	0.025	0.045
GRID-5	0.117	0.289	0.117	0.124
GRID-6	0.954	2.371	0.932	0.990
GRID-7	15.495	36.576	58.823	32.043
GRID-8	892.008	5163.437	103.169	886.401

Table 11.5: Time [sec] used for (Complete) Inward Propagation

	Heuristic (One Step Look Ahead)			
	SC	SC-FFS	SC-FFS-IS	FFI
GRID-4	0.057	0.076	0.104	0.081
GRID-5	0.732	1.454	0.738	0.729
GRID-6	11.218	26.701	11.043	4.408
GRID-7	128.889	563.246	1184.306	750.416
GRID-8	-------	-------	1222.213	-------

Table 11.6: Time [sec] used for Partial Inward Propagation

We might conclude that *OSLA - Smallest Clique* is well suited for this example. However, note that the corresponding numbers in Table 11.5 are in some sense misleading: *OSLA-SC-FFS* is a special case of *OSLA - Smallest Clique*. As a consequence, the latter heuristic could also generate one of the two (bad) elimination sequences obtained by the former heuristic. Therefore, *OSLA - Smallest Clique* has a wide margin concerning the elimination sequence generated.

Table 11.10 shows the **size of the nodes** of the join tree which is obtained for each of the four heuristics for the example `GRID-7`. Table 11.11 contains the corresponding chart for the example `GRID-8`. The best join tree for `GRID-7` was obtained by *OSLA-SC* and *OSLA-FFI*. The two corresponding join trees contain only one node which has a label of size nine. Consistently, these two heuristics are faster than the others. However, for the example `GRID-8`, there is no correlation between the size of the nodes and the time used for the propagation: here, *OSLA-SC-FFS-IS* generates the best join trees. Surprisingly, the former heuristic is much slower than each one of the other heuristics (see Table 11.5).

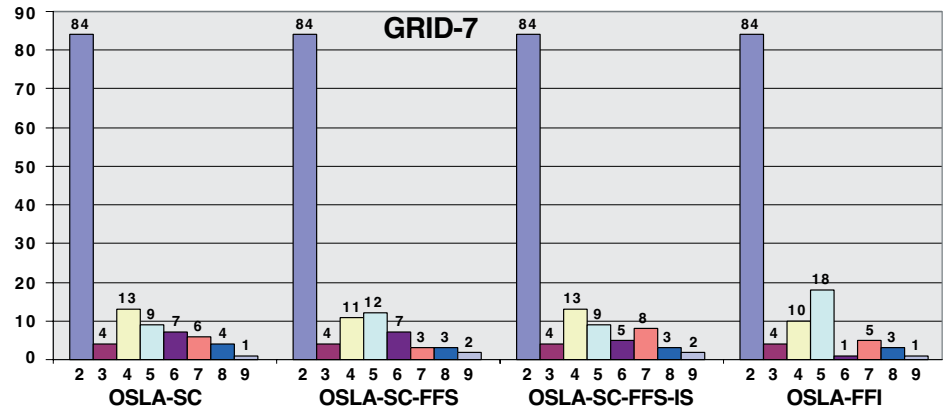

Figure 11.10: Size of the Nodes of the Join Tree.

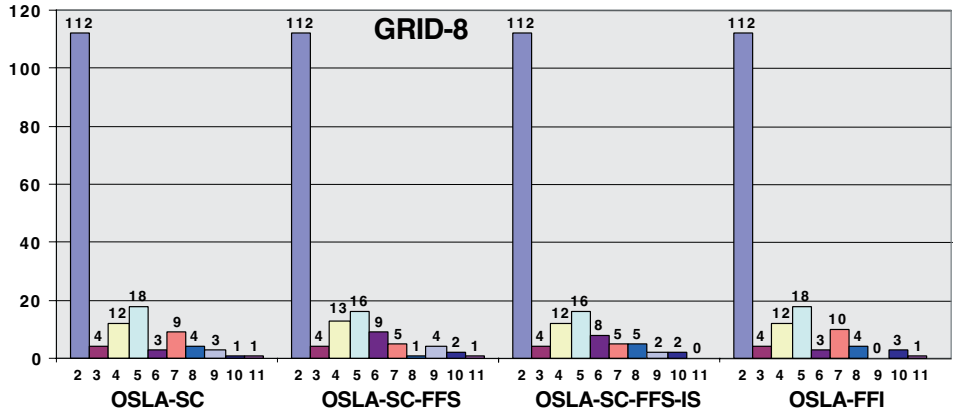

Figure 11.11: Size of the Nodes of the Join Tree.

The partial inward propagation requires much more time than the (complete) inward propagation. At first sight, this seems contradictory as some of the previously computed messages were reused and only a part of the join tree is involved in the partial inward propagation. The reason why propagation takes

so much time is that messages have many focal sets. For example, Figure 11.12 shows the number of focal sets of messages which are sent in the join tree obtained by *OSLA-SC-FFS-IS* for the example `GRID-8`. On the left side, the (complete) inward propagation is shown, whereas the partial inward propagation is shown on the right side. The number in each node represents the size of the corresponding label. Above all, note that most of the time is spend for combining messages and that huge mass functions are created in that way. Nevertheless, the message which is sent to an inward neighbor contains much less focal sets because an additional marginalization is performed.

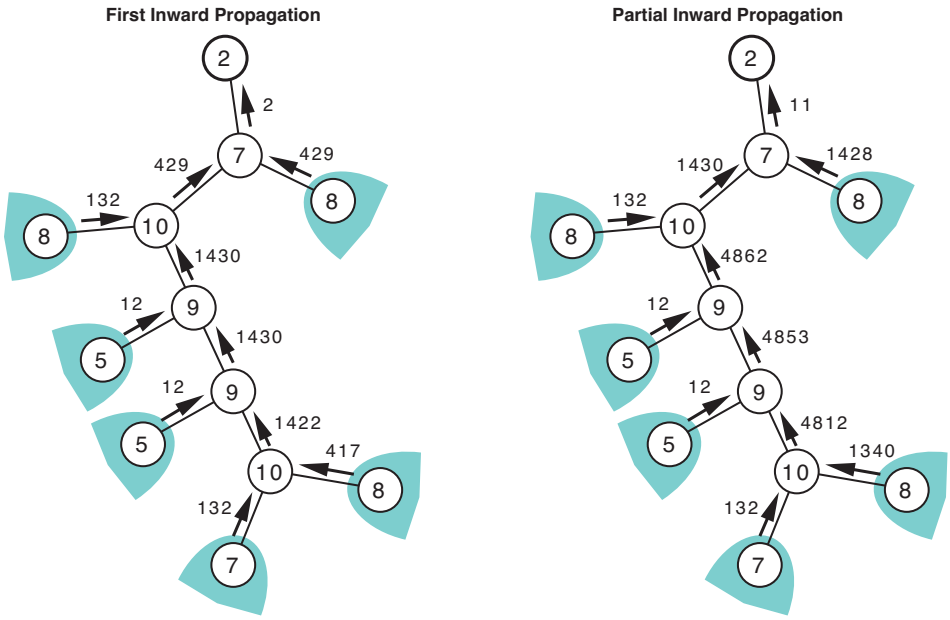

Figure 11.12: The Number of Focal Sets of the Messages.

Finally, this example points out that the **binary representation** which has been proposed in Section 9.1 is not really appropriate if the label of a node is large. In that case, much memory space is needed to store a mass function. For the example `GRID-8`, the partial inward propagation failed for almost all four heuristics because there was not enough memory available.

11.4 Web of Trust

The increasing importance of computer networks requires secure communication techniques. For that purpose, messages are encrypted. One way of encoding messages is to use **public key** algorithms like RSA (Rivest *et al.*, 1978). The idea behind public key systems is shown in Figure 11.13: a pair of keys is defined where one key is public and the other is secret. Everyone can encrypt

a message with the public key. However, the encrypted message can only be decrypted with the secret key.

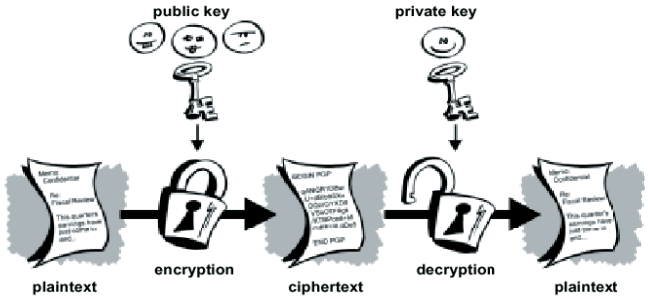

Figure 11.13: Public Key Cryptography.

The problem using these systems is to get the public key of the receiver. If another key than the real public key of the receiver is used to encrypt the message, then the owner of the wrong public key can read the message. Therefore, the sender is interested in the authenticity of the public key.

A solution is the **Public Key Infrastructure** (PKI for short) presented in (Maurer, 1996) and (Bütikofer, 2000). The problem is as mentioned above: the sender (called Alice in the rest of the text) has a public key of the receiver (called Bob), but she does not know, whether the public key is authentic. Alice also has some public keys of other people, but she is not sure, whether they are all authentic or not. Furthermore, she trusts only some of the people. Suppose that Alice also gets certificates and recommendations of the other people she knows. Certificates concern the authenticity of other public keys, and recommendations influences the trust in other people. Based of this information she wants to decide, whether the public key of Bob is authentic or not.

The authenticity of a public key is always relative to a possible sender. It is noted as a predicate $Aut_{A,X}$, where X is the expected owner of the public key, and A is the point of view, from which the public key is authentic or not. Another predicate $Cert_{X,Y}$ says that X confirms the authenticity of Y. With certificates, another problem arises: why should the receiver A of the certificate trust X? For that purpose, a predicate $Trust_{A,X}$ is introduced into the PKI. Finally, the predicate $Rec_{X,Y}$ represents a request of X to trust Y.

The following two rules express the relation between authenticity, certificate, trust, and recommendation. The first rule tells us, that a certificate from X for Y implies the authenticity of Y, if (1) X is authentic, and (2) A trusts X. The second rule is analogous for the relation between recommendation and trust.

$$(Aut_{A,X} \wedge Trust_{A,X}) \rightarrow (Cert_{X,Y} \rightarrow Aut_{A,Y}),$$
$$(Aut_{A,X} \wedge Trust_{A,X}) \rightarrow (Rec_{X,Y} \rightarrow Trust_{A,Y}).$$

Because trust is not a question of yes or no, it should be graded as a value in $[0, 1]$. The graduation of the trust influences all the other values. The problem

then is to calculate the authenticity of Bob's public key, which is also a value in $[0, 1]$.

The problem of computing authenticity is illustrated for the simple situation shown in Figure 11.14. Alice wants to send a confidential message to Bob, but she is not sure about the authenticity of Bob's public key. However, she trusts Dan and Conny whose public keys are known (but possibly not authentic). Dan and Conny give her both the same public key for Bob. Furthermore, Conny certificates the public key of Dan.

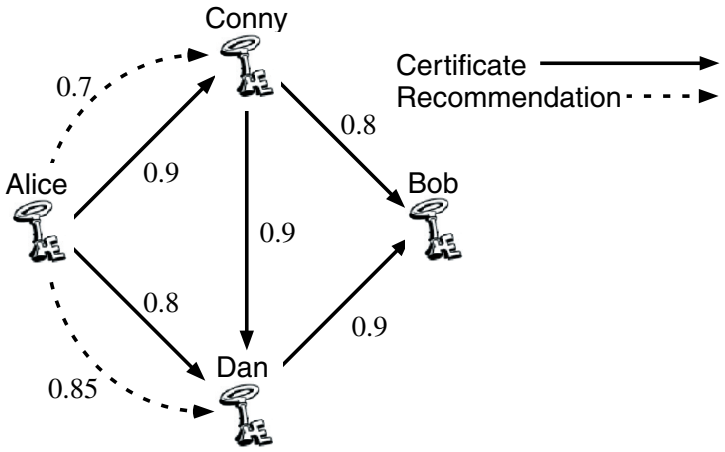

Figure 11.14: A Trust and Authenticity Network.

First, the variables for the trust and authenticity of each person including Alice, are defined.

```
(tell
  (var Aa Ta Ac Tc Ad Td Ab Tb binary))
```

Second, the recommendations and certificates are defined as binary assumptions.

```
(tell
  (ass Cac binary 0.9)
  (ass Rac binary 0.7)
  (ass Cad binary 0.8)
  (ass Rad binary 0.85)
  (ass Ccd binary 0.9)
  (ass Ccb binary 0.8)
  (ass Cdb binary 0.9))
```

The relations between the variables can then be written as follows:

```
(tell
  (-> (and Aa Ta Cac) Ac)
```

```
  (-> (and Aa Ta Rac) Tc)
  (-> (and Aa Ta Cad) Ad)
  (-> (and Aa Ta Rad) Td)
  (-> (and Ac Tc Ccd) Ad)
  (-> (and Ac Tc Ccb) Ab)
  (-> (and Ad Td Cdb) Ab))
```

Because Alice knows and trusts herself, the two variables `Aa` and `Ta` are always true:

```
(observe Aa Ta)
```

Now, Alice's view of the situation is defined and the computation of the support for the authenticity of Bob's public key can begin. To get a quantitative judgment, the following numerical query has to be used:

```
? (ask (dsp Ab))
QUERY: (DSP AB)
  0.825
```

11.5 Information Retrieval

The information retrieval problem consists in selecting from a collection of documents the relevant documents according to a given query. The selection of the documents should

(1) contain all relevant documents, and

(2) contain no irrelevant documents.

Satisfying just one of the above criteria is very simple. The first criterion, for example, can be completely satisfied by selecting the entire collection of documents. Similarly, the second criterion is completely satisfied when no documents are returned. Therefore, the information retrieval problem consists in satisfying both criteria at the same time.

As an example consider the collection of documents shown in Figure 11.15. The collection consists of books about geography and biology. The lower part of the picture shows the relations between the books, the middle part the relations between the terms appearing in the books, and the upper part represents possible queries. For each query, the most relevant books must be selected.

The information retrieval problem can be solved by a corresponding ABEL model (Picard & Haenni, 1998; Picard, 2000). First, the nodes appearing in the graph of the database are defined as binary variables.

```
(tell
  (var query1 query2 binary)
  (var valley lake mountain plankton fish binary)
  (var book1 book2 book3 book4 book5 binary))
```

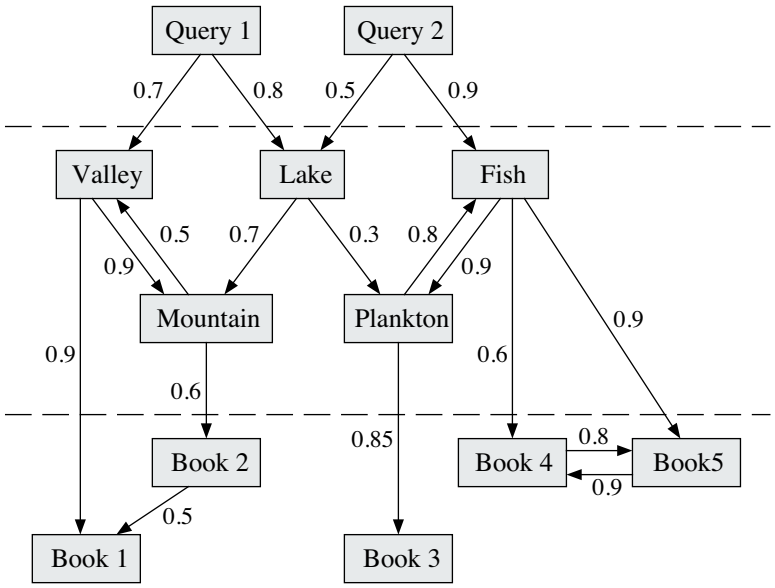

Figure 11.15: A simple Literature Database.

The links between the different nodes of the graph in Figure 11.15 are of different strength. From the point of view of probabilistic argumentation systems we say that the links are uncertain. Therefore, an assumption with a probability as indicated in Figure 11.15 is defined for every link:

```
(tell
  (ass query1-valley binary 0.7)
  (ass query1-lake binary 0.8)
  (ass query2-lake binary 0.5)
  (ass query2-fish binary 0.9)

  (ass valley-mountain binary 0.9)
  (ass lake-mountain binary 0.7)
  (ass lake-plankton binary 0.3)
  (ass mountain-valley binary 0.5)
  (ass plankton-fish binary 0.8)
  (ass fish-plankton binary 0.9)

  (ass valley-book1 binary 0.9)
  (ass mountain-book2 binary 0.6)
  (ass plankton-book3 binary 0.85)
  (ass fish-book4 binary 0.6)
  (ass fish-book5 binary 0.9)

  (ass book2-book1 binary 0.5)
  (ass book4-book5 binary 0.8)
  (ass book5-book4 binary 0.9))
```

The uncertain relations between the nodes of the graph can now be expressed by corresponding ABEL rules.

```
(tell
  (-> (and query1 query1-valley) valley)
  (-> (and query1 query1-lake) lake)

  (-> (and query2 query2-lake) lake)
  (-> (and query2 query2-fish) fish)

  (-> (and valley valley-mountain) mountain)
  (-> (and lake lake-mountain) mountain)
  (-> (and lake lake-plankton) plankton)
  (-> (and mountain mountain-valley) valley)
  (-> (and plankton plankton-fish) fish)
  (-> (and fish fish-plankton) plankton)

  (-> (and valley valley-book1) book1)
  (-> (and mountain mountain-book2) book2)
  (-> (and plankton plankton-book3) book3)
  (-> (and fish fish-book4) book4)
  (-> (and fish fish-book5) book5)

  (-> (and book2 book2-book1) book1)
  (-> (and book4 book4-book5) book5)
  (-> (and book5 book5-book4) book4))
```

To obtain a measure for selecting the most relevant books of the first query, let us compute the following degrees of support:

```
? (ask (dsp (-> query1 book1)) (dsp (-> query1 book2))
       (dsp (-> query1 book3)) (dsp (-> query1 book4))
       (dsp (-> query1 book5)))
QUERY: (DSP (-> QUERY1 BOOK1))
  0.753
QUERY: (DSP (-> QUERY1 BOOK2))
  0.502
QUERY: (DSP (-> QUERY1 BOOK3))
  0.204
QUERY: (DSP (-> QUERY1 BOOK4))
  0.177
QUERY: (DSP (-> QUERY1 BOOK5))
  0.182
```

The first two books are obviously the most relevant documents in the collection. In a similar way we can compute the results for the second query:

```
? (ask (dsp (-> query2 book1)) (dsp (-> query2 book2))
       (dsp (-> query2 book3)) (dsp (-> query2 book4))
       (dsp (-> query2 book5)))
QUERY: (DSP (-> QUERY2 BOOK1))
  0.215
QUERY: (DSP (-> QUERY2 BOOK2))
```

```
  0.210
QUERY: (DSP (-> QUERY2 BOOK3))
  0.713
QUERY: (DSP (-> QUERY2 BOOK4))
  0.843
QUERY: (DSP (-> QUERY2 BOOK5))
  0.865
```

Now, the situation has changed: the first two books are rather irrelevant, while the other three books seem to fit well for the second query.

12
Future Work

Research is an ongoing process. In the following, we will present some ideas which show the direction future research could take. Above all, we think that it is most important to look at **approximation methods** because exact numerical computation is not possible for some examples. In any case, an approximated numerical result which is close to the exact value may often be sufficent for a user. We will present such an approximation method. One of the advantages of this method is that it computes besides the approximated result also an interval as a bound of the exact value.

A topic which has not been addressed in this work is the usability of numerical computations for obtaining symbolic results. Especially, it can happen that it is impossible to compute the set of all arguments in favor of a hypothesis of interest. In that case, numerical computations may still be possible and could help finding *important* arguments. The sketch of such a method will be given later in this chapter.

12.1 Approximation for Dempster-Shafer Theory

The previous chapter has shown that exact computation is sometimes not possible. For example, consider the type of communication network which has been discussed in Section 11.3. The nodes of such a communication network are arranged in a grid as shown on the right of Figure 11.9. It has been shown that there is a strong increase in the time needed for propagation. For the example `GRID-8` and the heuristic *OSLA-SC-FFS-IS*, the first (complete) inward propagation took almost two minutes, the partial inward propagation even more than twenty minutes.

The reason for this is shown in Figure 11.12. Above all, this figure points out that some of the messages which are sent in the join tree have a large number of focal sets. As a consequence, much time is required for the combination of these messages.

In the following, we will take a closer look at messages which are sent in the join tree obtained from applying the heuristic *OSLA-SC-FFS-IS* to the example

GRID-8. In particular, we will consider the messages $m_1, \ldots, m_6$ and $m'_1, \ldots, m'_6$ shown in Figure 12.1.

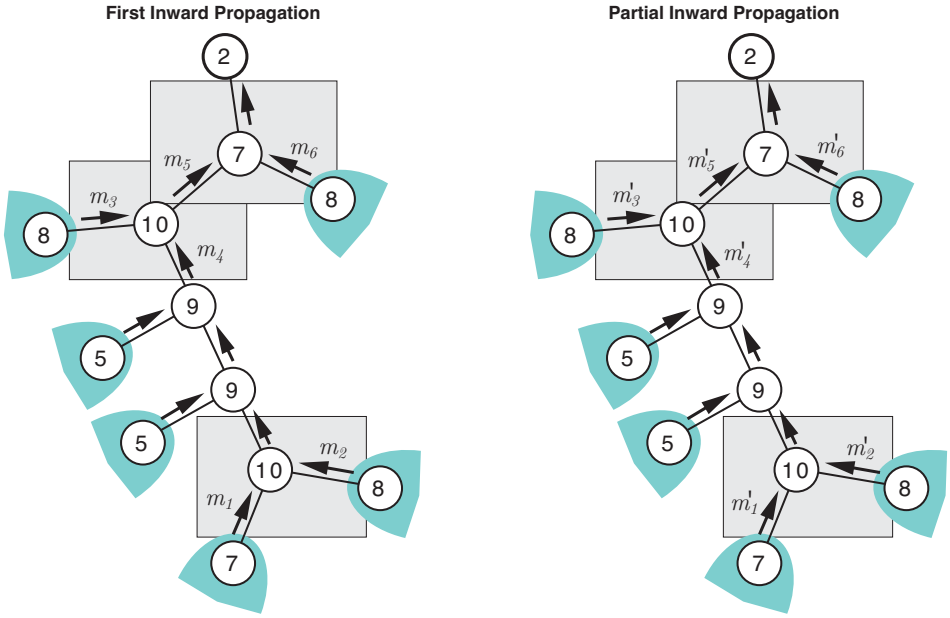

Figure 12.1: Messages $m_1, \ldots, m_6$ and $m'_1, \ldots, m'_6$.

The messages $m_1, \ldots, m_6$ are sent during the first (complete) inward propagation. From the Figure 11.12 it can be seen that each one of these messages contains a large number of focal sets. In Figure 12.2, a pie chart represents the masses for each one of the messages $m_1, \ldots, m_6$.

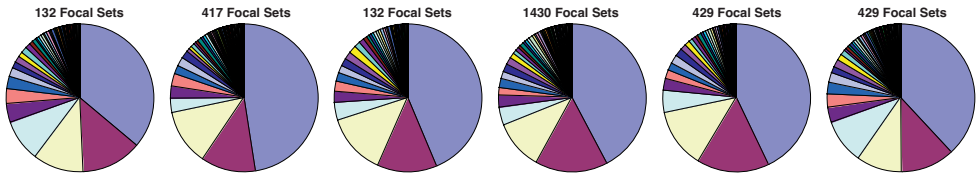

Figure 12.2: Masses of the Messages $m_1, \ldots, m_6$.

The sum of the masses is equal to 1 for each of the messages $m_1, \ldots, m_6$. Nevertheless, Figure 12.2 shows clearly that there are masses which are much bigger than others. In other words, masses assigned to only a few focal sets represent almost the total of all masses. This is also shown in Table 12.1. For example, the collection consisting of the 16 largest masses of the message m_1 represents more than ninety percent of the total of all masses.

The situation is similar for the messages which are sent during the partial inward propagation. In Figure 12.3, each one of the messages m'_2, m'_4, m'_5, and

	0.900	**0.990**	**0.999**	**1.000**
m_1	16 Sets 12.12 %	61 Sets 46.21 %	97 Sets 73.48 %	132 Sets 100.00 %
m_2	16 Sets 3.83 %	112 Sets 26.85 %	251 Sets 60.19 %	417 Sets 100.00 %
m_3	14 Sets 10.60 %	50 Sets 37.87 %	80 Sets 60.60 %	132 Sets 100.00 %
m_4	19 Sets 1.32 %	142 Sets 9.93 %	449 Sets 31.39 %	1430 Sets 100.00 %
m_5	13 Sets 3.03 %	62 Sets 14.45 %	149 Sets 34.73 %	429 Sets 100.00 %
m_6	20 Sets 4.66 %	114 Sets 26.57 %	244 Sets 56.87 %	429 Sets 100.00 %

Table 12.1: Masses of the Messages $m_1, \ldots, m_6$.

m'_6 is represented by a corresponding pie chart. The messages m'_1 and m'_3 are not recomputed during the partial inward propagation. Therefore, m'_1 and m'_3 are equal to m_1 and m_3 respecively.

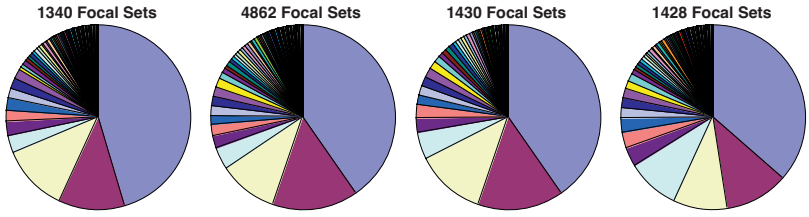

Figure 12.3: Masses of m'_2,m'_4,m'_5, and m'_6.

Again, the masses which are assigned to a few focal sets represent almost the total of all masses. For example, Table 12.2 shows that the 24 largest masses of m'_2 represent more than ninety percent of all masses of m'_2.

Where the messages $m_1, \ldots, m_6$ is concerned, $m_1 \otimes m_2$, $m_3 \otimes m_4$, and $m_5 \otimes m_6$ is computed during the first inward propagation phase. Similarly, $m'_1 \otimes m'_2$, $m'_3 \otimes m'_4$, and $m'_5 \otimes m'_6$ is computed afterwards during the partial inward propagation. The time needed to combine two mass functions (messages are represented as mass functions) is determinated by the number of focal sets of the mass functions. In order to speed up computations, it is a natural approach to consider only the most important focal sets for each combination.

The easiest approach for this is to consider only the k focal sets with the largest masses for a fixed k. From Figure 12.4 it can be seen that in that way less focal

	0.900	**0.990**	**0.999**	**1.000**
m'_2	24 Sets 1.79 %	214 Sets 15.97 %	619 Sets 46.19 %	1340 Sets 100.00 %
m'_4	27 Sets 0.55 %	251 Sets 5.16 %	947 Sets 19.47 %	4862 Sets 100.00 %
m'_5	18 Sets 1.25 %	101 Sets 7.06 %	304 Sets 21.25 %	1430 Sets 100.00 %
m'_6	29 Sets 2.03 %	219 Sets 15.33 %	588 Sets 41.17 %	1428 Sets 100.00 %

Table 12.2: Masses of the Messages m'_2, m'_4, m'_5, and m'_6.

sets are taken into account for combination. It was obtained for the example `GRID-8` and $k = 200$. Therefore, whenever two potentials were combined only the 200 focal sets with the largest masses were considered. The result of this is that messages send in the join tree have much less focal sets compared to case where no approximation is performed.

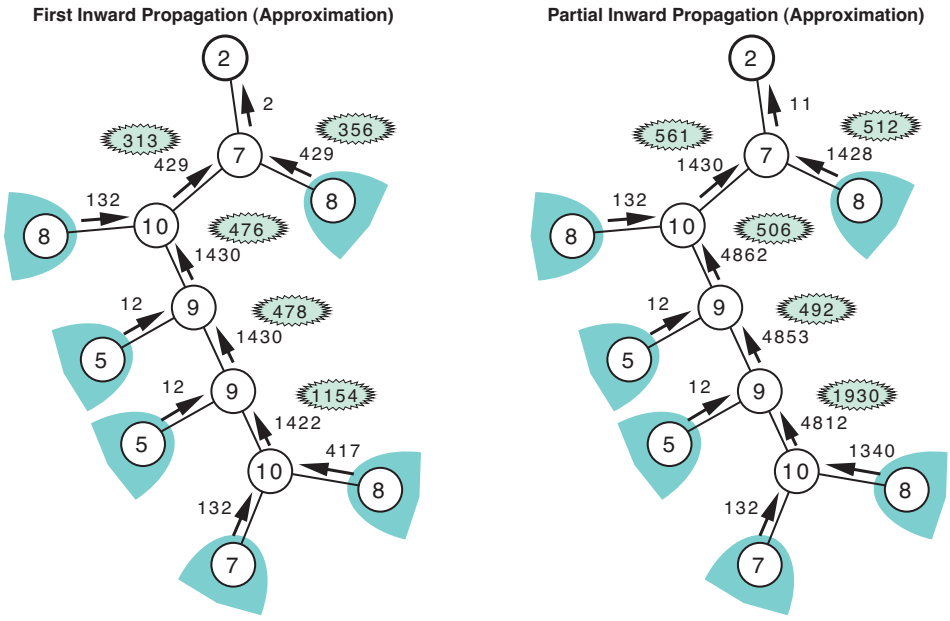

Figure 12.4: Number of Focal Sets for `GRID-8`, $k = 200$.

As a consequence, propagation is performed much faster than previously: about 25 seconds are required for the first inward propagation and less than 45 seconds for the partial inward propagation. Nevertheless, an important question to

answer is whether or not the obtained numerical result is a good approximation for the exact value.

12.1.1 Approximating Degree of Quasi-Support

For exact computation, the root node contains after the first inward propagation a potential φ. If for combination only the k most important focal sets are taken into account, the root node then contains instead of φ an approximation $\tilde{\varphi}$. Usually, the sum of the masses of $\tilde{\varphi}$ is less than 1. We therefore define ϵ as follows:

$$\epsilon \;=\; 1 - \sum_{A \subseteq \Theta_{D_r}} [\tilde{\varphi}(A)]_m$$

There is a strong correlation between φ and $\tilde{\varphi}$ because

$$[\tilde{\varphi}(A)]_m \;\leq\; [\varphi(A)]_m$$

for all sets $A \subseteq \Theta_{D_r}$, $D_r = d(\tilde{\varphi})$. As a consequence, $FS(\tilde{\varphi}) \subseteq FS(\varphi)$. Most important is the fact that the exact value $[\varphi(A)]_b$ can be bound by an interval. For all sets $A \subseteq \Theta_{D_r}$, the following condition holds:

$$[\tilde{\varphi}(A)]_b \;\leq\; [\varphi(A)]_b \;\leq\; [\tilde{\varphi}(A)]_b + \epsilon$$

This allows to approximate $dqs(h, \xi)$ because unnormalized belief corresponds to degree of quasi-support. If $dqs_a(h, \xi)$ corresponds to $[\tilde{\varphi}(H)]_b$ and denotes the approximation for $dqs(h, \xi)$, then

$$dqs_a(h, \xi) \;\leq\; dqs(h, \xi) \;\leq\; dqs_a(h, \xi) + \epsilon.$$

12.1.2 Approximating Degree of Support

The first inward propagation yields the value $[\tilde{\varphi}()]_m$ and the error ϵ_1. Similarly, the value $[\tilde{\varphi}(H)]_m$ and the error ϵ_2 are obtained from the partial inward propagation. Therefore, we have the following two inequalities:

$$\begin{array}{rcccl} [\tilde{\varphi}()]_b & \leq & [\varphi()]_b & \leq & [\tilde{\varphi}()]_b + \epsilon_1 \\ [\tilde{\varphi}(H)]_b & \leq & [\varphi(H)]_b & \leq & [\tilde{\varphi}(H)]_b + \epsilon_2 \end{array}$$

The (exact) normalized belief in the hypothesis is computed as

$$[\varphi(H)]_B \;=\; \frac{[\varphi(H)]_b - [\varphi()]_b}{1 - [\varphi()]_b}.$$

Using the above inequalities, an approximation for $[\varphi(H)]_B$ can be given. For this, the following lower and upper bounds will be used:

$$\begin{aligned} LBound &= \frac{[\tilde{\varphi}(H)]_b - ([\tilde{\varphi}()]_b + \epsilon_1)}{1 - ([\tilde{\varphi}()]_b + \epsilon_1)} \\ UBound &= \frac{([\tilde{\varphi}(H)]_b + \epsilon_2) - [\tilde{\varphi}()]_b}{1 - [\tilde{\varphi}()]_b} \end{aligned}$$

The approximation of the (exact) normalized belief is then given by

$$LBound \;\leq\; [\varphi(H)]_B \;\leq\; UBound$$

This allows to approximate $dsp(h,\xi)$ because normalized belief corresponds to degree of support. The following inequality is valid:

$$LBound \;\leq\; dsp(h,\xi) \;\leq\; UBound$$

Example 12.1 Approximation for `GRID-8` and $k = 200$ The first inward propagation yields the value $dqs_a(\bot,\xi) = 0$ and the error $\epsilon_1 = 0.019994$. Similarly, $dqs_a(x_{11} \rightarrow x_{88},\xi) = 0.838848$ and $\epsilon_2 = 0.064518$ result from the partial inward propagation. Therefore,

$$\begin{array}{rcrcl} 0 & \leq & dqs(\bot,\xi) & \leq & 0.019994 \\ 0.838848 & \leq & dqs(x_{11} \rightarrow x_{88},\xi) & \leq & 0.903367 \end{array}$$

are the corresponding two inequalities to approximate degrees of quasi-support. The approximation for degree of support is then given as follows:

$$0.835560 \;\leq\; dsp(x_{11} \rightarrow x_{88},\xi) \;\leq\; 0.903366$$

$\ominus$

This kind of approximation is quite similar to other approximation methods (Tessem, 1993; Bauer, 1996). Nevertheless, there are important differences. Above all, we think that an approximation method should not only compute an approximated value. Instead, it should also give bounds for the exact value.

The presented approximation method does not work in the case where the masses of focal sets are uniformely distributed. In addition, the interval used as a bound for the exact value is not that usable if the knowledge base is almost contradictory. Note that Monto-Carlo methods (Wilson, 1991; Kreinovich *et al.*, 1994) suffer from the same problem.

12.2 Numerical Precomputations

The set of symbolic arguments in favor of a certain hypothesis of interest is sometimes huge and cannot be computed explicitly. However, in such a situation, we are often not interested in the complete set of arguments. Instead, the *most important arguments* may be sufficient. In the following, we will sketch a method for computing *important* arguments. The main idea is to construct from the initially given probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}} = (\xi, \mathcal{V}, \mathcal{A}, \Pi)$ another argumentation system. For that purpose, a numerical precomputation is performed as if we were to answer a quantitative query for $\mathcal{PAS}_{\mathcal{V}}$. However, the numerical precomputation is then used for the construction of the (smaller) argumentation system.

Suppose that the set of (symbolic) arguments $QS(h,\xi)$ has to be computed for a hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. Note that $QS(h,\xi)$ is equal to $QS(\bot, \xi \wedge \neg h)$. Therefore, we construct a join tree which corresponds to $\xi \wedge \neg h$ and perform a numerical inward propagation followed by an outward propagation. A node N_{k_0} of the join tree has then received the messages $N_{k_1k_0}, \ldots, N_{k_mk_0}$ from its neighbor nodes $N_1, \ldots, N_m$ and could compute its marginal $\varphi^{\downarrow D_{k_0}}$ as explained in Chapter 7 by

$$\varphi^{\downarrow D_{k_0}} \quad = \quad \varphi_{k_0} \otimes \varphi_{k_1k_0} \otimes \cdots \otimes \varphi_{k_mk_0}, \tag{12.1}$$

where φ_{k_0} is the potential which is stored on the node N_{k_0}.

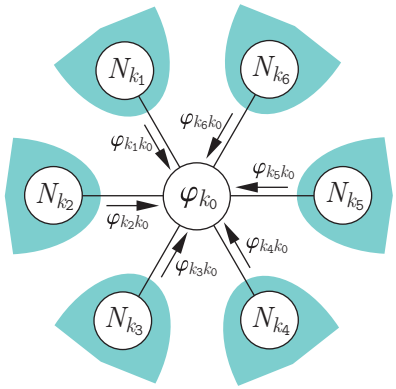

Figure 12.5: Messages for the Node N_{k_0}.

The value $[\varphi^{\downarrow D_{k_0}}()]_m$ corresponds to $dqs(h,\xi)$ and represents the probability of the set $QS(h,\xi)$. It can be obtained on every node of the join tree if a numerical inward propagation followed by an outward propagation is performed. It might be interesting to compute for each node N_{k_0} the importance of the corresponding potential φ_{k_0}. For that purpose, we compute for each node N_{k_0}

$$\tilde{\varphi}^{\downarrow D_{k_0}} \quad = \quad \varphi_{k_1k_0} \otimes \cdots \otimes \varphi_{k_mk_0}, \tag{12.2}$$

as usually, but without considering the potential φ_{k_0}. If $[\tilde{\varphi}^{\downarrow D_{k_0}}()]_m$ is small, then φ_{k_0} is more important. In contrast, if it is close to $[\varphi^{\downarrow D_{k_0}}()]_m$, then φ_{k_0} has only a marginal influence.

The next step is to consider only those potentials φ_{k_0} for which $[\tilde{\varphi}^{\downarrow D_{k_0}}()]_m$ is small. This corresponds to selecting a subset of all potentials which are contained in the join tree. In that way, also a set of variables $\mathcal{V}' \subseteq \mathcal{V}$ and a set of assumptions $\mathcal{A}' \subseteq \mathcal{A}$ are determinated. Note that the set of all potentials is equivalent to $\xi \wedge \neg h$. Similarly, the selected subset of potentials is equivalent to a certain formula ξ'_h, where $\xi \wedge \neg h \models \xi'_h$. The probabilistic argumentation system $\mathcal{PAS}_{\mathcal{V}}{}' = (\xi'_h, \mathcal{V}', \mathcal{A}', \Pi')$ is often much smaller than the original argumentation system. In addition, $\mathcal{PAS}_{\mathcal{V}}{}'$ contains only information which is important for the given hypothesis $h \in \mathcal{L}_{\mathcal{A}\cup\mathcal{V}}$. Perhaps, $\mathcal{PAS}_{\mathcal{V}}{}'$ is small enough in order to perform

symbolic computations. In that way, only a subset of $QS(h, \xi)$ is obtained. However, there is a big chance that this subset contains the most important arguments.

12.3 Modular Join Tree Construction

Large digital circuits are usually composed of several smaller digital circuits. Each one of these smaller digital circuits consists itself of even smaller digital circuits. To express such a digital circuit in ABEL, the *module* concept can be used. A a digital circuits then contains calls to other modules representing smaller digital circuits. Therefore, the **calls to modules** representing digital circuits define a **hierarchical structure**. This hierarchical structure can be considered as an added value and constructing the join tree without taking the hierarchical structure into account is of course not optimal.

The construction of a join tree can be improved in such a situation if an appropriate method is used. We have performed some tests for such a method. This method goes through the hierarchical structure from top to down and constructs at each step a corresponding join tree. Therefore, each call to a module results in a join tree constructed from the contents of the module. Of course, it is important that the **Markov property** (see Chapter 6) is not violated. For that reason, the set of *seperators* of a call to a module is taken into account when the join tree for the corresponding call to the module is constructed later. These seperators are also used to connect the different join trees together such that only one single join tree is finally obtained.

A
Abbreviations

Abbreviation	Meaning
ABEL	Assumption-Based Evidential Language
BJT	Binary Join Tree
CNF	Conjunctive Normal Form
DNF	Disjunctive Normal Form
FD architecture	Fast–Division architecture
JT	Join Tree
LS architecture	Lauritzen–Spiegelhalter architecture
MVL	Many-valued Logic
OSLA-FFI	One Step Look Ahead - Fewest Fill-ins
OSLA-SC	One Step Look Ahead - Smallest Clique
OSLA-SC-FFS	OSLA - Smallest Clique, Fewest Focal Sets
OSLA-SC-FFS-IS	OSLA - Smallest Clique, Fewest Focal Sets, Initial Structure
PKI	Public Key Infrastructure
SCL	Set Constraint Logic
SS architecture	Shenoy–Shafer architecture
VPLL	Variable-Potential Link List
VN	Valuation Network

B
Proof of Theorems

Proof of Theorem 2.5

$$
\begin{aligned}
QS_{\mathcal{A}}(h,\xi') &= QS_{\mathcal{A}}(h,\xi\wedge\tilde{\xi}) = \{\mathbf{s}\in N_{\mathcal{A}}\colon \mathbf{s}\wedge\xi\wedge\tilde{\xi}\models h\}\\
&= \{\mathbf{s}\in N_{\mathcal{A}}\colon \mathbf{s}^{\uparrow A\cup P}\cap N_{\mathcal{A}\cup\mathcal{V}}(\xi\wedge\tilde{\xi})\subseteq N_{\mathcal{A}\cup\mathcal{V}}(h)\}\\
&= \{\mathbf{s}\in N_{\mathcal{A}}\colon \mathbf{s}^{\uparrow A\cup P}\cap N_{\mathcal{A}\cup\mathcal{V}}(\xi)\cap N_{\mathcal{A}\cup\mathcal{V}}(\tilde{\xi})\subseteq N_{\mathcal{A}\cup\mathcal{V}}(h)\}\\
&\supseteq \{\mathbf{s}\in N_{\mathcal{A}}\colon \mathbf{s}^{\uparrow A\cup P}\cap N_{\mathcal{A}\cup\mathcal{V}}(\xi)\subseteq N_{\mathcal{A}\cup\mathcal{V}}(h)\} = QS_{\mathcal{A}}(h,\xi) \qquad \square
\end{aligned}
$$

Proof of Theorem 5.1

$$(\phi_1\otimes\phi_2)^{\downarrow D_3} = (\phi_1\otimes\phi_2^{\uparrow D_3})^{\downarrow D_3} = \phi_1^{\downarrow D_1\cap D_3}\otimes\phi_2^{\uparrow D_3} = \phi_1^{\downarrow D_3}\otimes\phi_2. \qquad \square$$

Proof of Theorem 5.2

Axiom A1 (Neutrality)
The mass function φ on D with $[\varphi(\Theta_D)]_m = 1$ corresponds to the neutral element.

Axiom A2 (Commutativity)
Suppose that $d(\varphi_1) = D_1$ and $d(\varphi_2) = D_2$. Then, $d(\varphi_1\otimes\varphi_2) = D_1\cup D_2$. For every $A\subseteq\Theta_{D_1\cup D_2}$ it is

$$
\begin{aligned}
[\varphi_1\otimes\varphi_2(A)]_q &= [\varphi_1(A^{\downarrow D_1})]_q\cdot[\varphi_2(A^{\downarrow D_2})]_q\\
&= [\varphi_2(A^{\downarrow D_2})]_q\cdot[\varphi_1(A^{\downarrow D_1})]_q = [\varphi_2\otimes\varphi_1(A)]_q
\end{aligned}
$$

Axiom A2 (Associativity)
Suppose that $d(\varphi_1) = D_1$, $d(\varphi_2) = D_2$, and $d(\varphi_3) = D_3$. If D is defined as $D = D_1\cup D_2\cup D_3$ then $d(\varphi_1\otimes\varphi_2\otimes\varphi_3) = D$. For every $A\subseteq D$ it is

$$[\varphi_1\otimes(\varphi_2\otimes\varphi_3)(A)]_q = [\varphi_1(A^{\downarrow D_1})]_q\cdot[\varphi_2\otimes\varphi_3(A^{\downarrow D_2\cup D_3})]_q$$

$$\begin{aligned} &= [\varphi_1(A^{\downarrow D_1})]_q \cdot ([\varphi_2(A^{\downarrow D_2})]_q \cdot [\varphi_3(A^{\downarrow D_3})]_q) \\ &= ([\varphi_1(A^{\downarrow D_1})]_q \cdot [\varphi_2(A^{\downarrow D_2})]_q) \cdot [\varphi_3(A^{\downarrow D_3})]_q \\ &= ([\varphi_1 \otimes \varphi_2(A^{\downarrow D_1 \cup D_2})]_q) \cdot [\varphi_3(A^{\downarrow D_3})]_q \\ &= [(\varphi_1 \otimes \varphi_2) \otimes \varphi_3)(A)]_q \end{aligned}$$

Axiom A3 (Transitivity of marginalization)
Suppose that $d(\varphi) = D$ and $F \subseteq E \subseteq D$. For every $A \subseteq \Theta_F$

$$[\varphi^{\downarrow F}(A)]_b \;=\; [\varphi(A^{\uparrow D})]_b$$

because of Equation 3.10. On the other hand,

$$[(\varphi^{\downarrow E})^{\downarrow F}(A)]_b \;=\; [\varphi^{\downarrow E}(A^{\uparrow E})]_b \;=\; [\varphi((A^{\uparrow E})^{\uparrow D})]_b \;=\; [\varphi(A^{\uparrow D})]_b$$

since $(A^{\uparrow E})^{\uparrow D} = A^{\uparrow D}$. Therefore, finally it is

$$[\varphi^{\downarrow F}(A)]_b \;=\; [(\varphi^{\downarrow E})^{\downarrow F}(A)]_b.$$

Axiom A4 (Distributivity of marginalization over combination)
Suppose $d(\varphi_1) = D_1$ and $d(\varphi_2) = D_2$. In addition, suppose $D = D_1 \cup D_2$. In the following, let $A_1 \in FS(\varphi_1)$ and $A_2 \in FS(\varphi_2)$. Then, it is sufficient to prove that

$$(A_1^{\uparrow D} \cap A_2^{\uparrow D})^{\downarrow D_1} \;=\; A_1 \cap (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}.$$

• First, we show that $(A_1^{\uparrow D} \cap A_2^{\uparrow D})^{\downarrow D_1} \subseteq A_1 \cap (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$:
Let $\mathbf{x} \in (A_1^{\uparrow D} \cap A_2^{\uparrow D})^{\downarrow D_1}$. Then, there exists $\mathbf{y} = (\mathbf{x}, \mathbf{r})$ so that $\mathbf{y} \in A_1^{\uparrow D}$ and $\mathbf{y} \in A_2^{\uparrow D}$. First, $\mathbf{y} \in A_1^{\uparrow D}$ implies $\mathbf{x} \in A_1$. On the other hand, $\mathbf{y} \in A_2^{\uparrow D}$ implies $(\mathbf{x}^{\downarrow D_1 \cap D_2}, \mathbf{r}) \in A_2$. Therefore, it is $\mathbf{x}^{\downarrow D_1 \cap D_2} \in A_2^{\downarrow D_1 \cap D_2}$ and also $\mathbf{x} \in (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$. Finally, we have $\mathbf{x} \in A_1 \cap (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$.

• Second, we show that $(A_1^{\uparrow D} \cap A_2^{\uparrow D})^{\downarrow D_1} \supseteq A_1 \cap (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$:
Let $\mathbf{x} \in A_1 \cap (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$. Therefore, it is $\mathbf{x} \in A_1$ and $\mathbf{x} \in (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$. First, $\mathbf{x} \in (A_2^{\downarrow D_1 \cap D_2})^{\uparrow D_1}$ implies $\mathbf{x}^{\downarrow D_1 \cap D_2} \in A_2^{\downarrow D_1 \cap D_2}$. Given this, there exists $\mathbf{y} = (\mathbf{x}^{\downarrow D_1 \cap D_2}, \mathbf{r})$ so that $\mathbf{y} \in A_2$. Then, it is $(\mathbf{x}, \mathbf{r}) \in A_2^{\uparrow D}$. On the other hand, $\mathbf{x} \in A_1$ implies $(\mathbf{x}, \mathbf{r}) \in A_1^{\uparrow D}$. Thus, $(\mathbf{x}, \mathbf{r}) \in (A_1^{\uparrow D} \cap A_2^{\uparrow D})$ and as consequence, we have $\mathbf{x} \in (A_1^{\uparrow D} \cap A_2^{\uparrow D})^{\downarrow D_1}$. □

Proof of Theorem 7.1

The theorem is proved by induction over the join tree. First, if N_{k_0} is a leaf node, then $\phi_{k_0 k_m} = \varphi_{k_0}$ and it is trivially

$$\phi_{k_0 k_m}^{\downarrow D_{k_0}} \;=\; \varphi_{k_0}$$

and

$$\varphi_{k_0k_m} \;=\; \left(\phi_{k_0k_m}^{\downarrow D_{k_0}}\right)^{\downarrow D_{k_0}\cap D_{k_m}} \;=\; \phi_{k_0k_m}^{\downarrow D_{k_0}\cap D_{k_m}}.$$

Otherwise, it is $\phi_{k_0k_m} = \varphi_{k_0} \otimes \phi_{k_1k_0} \otimes \cdots \otimes \phi_{k_{m-1}k_0}$ and therefore

$$\phi_{k_0k_m}^{\downarrow D_{k_0}} \;=\; (\varphi_{k_0} \otimes \phi_{k_1k_0} \otimes \cdots \otimes \phi_{k_{m-1}k_0})^{\downarrow D_{k_0}}.$$

With $S_i = d(\phi_{k_ik_0})$ and $U_\ell = D_{k_0} \cup \bigcup\{S_i : 1 \leq i \leq \ell\}$ it is $U_0 \subseteq \cdots \subseteq U_{m-2}$ and

$$\phi_{k_0k_m}^{\downarrow D_{k_0}} \;=\; \left(\left(\left(\varphi_{k_0} \otimes \phi_{k_1k_0} \otimes \cdots \otimes \phi_{k_{m-1}k_0}\right)^{\downarrow U_{m-2}}\right)^{\downarrow U_{m-3}} \cdots\right)^{\downarrow U_0}.$$

Applying the third axiom of the valuation network framework to the innermost marginal gives

$$\phi_{k_0k_m}^{\downarrow D_{k_0}} \;=\; \left(\left(\left(\varphi_{k_0} \otimes \cdots \otimes \phi_{k_{m-2}k_0}\right) \otimes \phi_{k_{m-1}k_0}^{\downarrow S_{m-1}\cap U_{m-2}}\right)^{\downarrow U_{m-3}} \cdots\right)^{\downarrow U_0}.$$

At this point of the proof the **Markov property** of the join tree is used. Whenever for a variable $x \in S_{m-1}$ it is also $x \in U_{m-2}$ then by the Markov property it is also $x \in D_{k_0}$ and $x \in D_{k_{m-1}}$. Conversely, for every variable x with $x \in D_{k_0}$ and $x \in D_{k_{m-1}}$ it is also $x \in U_{m-2}$ and $x \in S_{m-1}$. Therefore, $S_{m-1} \cap U_{m-2} = D_{k_0} \cap D_{k_{m-1}}$. In addition, the result of Theorem 5.1 can then be applied so that

$$\phi_{k_0k_m}^{\downarrow D_{k_0}} \;=\; \left(\left(\left(\varphi_{k_0} \otimes \cdots \otimes \phi_{k_{m-2}k_0}\right)^{\downarrow U_{m-3}}\right)^{\downarrow U_{m-4}} \cdots\right)^{\downarrow U_0} \otimes \phi_{k_{m-1}k_0}^{\downarrow D_{k_0}\cap D_{k_{m-1}}}.$$

Proceeding further this way we finally obtain

$$\phi_{k_0k_m}^{\downarrow D_{k_0}} \;=\; \varphi_{k_0} \otimes \phi_{k_1k_0}^{\downarrow D_{k_0}\cap D_{k_1}} \otimes \cdots \otimes \phi_{k_{m-1}k_0}^{\downarrow D_{k_0}\cap D_{k_{m-1}}}.$$

From the initial assertion of induction we finally can conclude that

$$\phi_{k_0k_m}^{\downarrow D_{k_0}} \;=\; \varphi_{k_0} \otimes \varphi_{k_1k_0} \otimes \cdots \otimes \varphi_{k_mk_0}.$$

The message $\varphi_{k_0k_m}$ is then given by

$$\varphi_{k_0k_m} \;=\; \left(\phi_{k_0k_m}^{\downarrow D_{k_0}}\right)^{\downarrow D_{k_0}\cap D_{k_m}} \;=\; \phi_{k_0k_m}^{\downarrow D_{k_0}\cap D_{k_m}}.$$

□

Proof of Theorem 7.2

Although N_{k_0} has no inward neighbor we think of N_{k_0} as if it has an inward neighbor N_{k_m}. Then, the statement follows from

$$\Phi_{k_0k_m} \;=\; \{\varphi_1, \ldots, \varphi_n\} \tag{B.1}$$

and from Theorem 7.1. □

Proof of Theorem 7.3

If N_{k_0} is the root node, then by Theorem 7.2 proved above

$$\varphi''_{k_0} \quad = \quad \varphi^{\downarrow D_{k_0}}.$$

Otherwise, it is easy to see that in the Shenoy–Shafer architecture a node N_{k_0} receives the same messages regardless which node is set as root node. Therefore, N_{k_0} is set as root node and then Theorem 7.2 can be applied again. □

Proof of Theorem 8.1

$[\varphi^{\downarrow D_H}(H)]_B$ is given by

$$[\varphi^{\downarrow D_H}(H)]_B \quad = \quad \frac{[\varphi^{\downarrow D_H}(H)]_b - [\varphi^{\downarrow D_H}()]_b}{1 - [\varphi^{\downarrow D_H}()]_b}. \tag{B.2}$$

If υ is so that $[\upsilon(H^c)]_m = 1$ and $d(\upsilon) = D_H$, then

$$[\varphi^{\downarrow D_H}(H)]_b \quad = \quad \sum_{A \subseteq H} [\varphi^{\downarrow D_H}(A)]_m = \sum_{A \cap H^c =} [\varphi^{\downarrow D_H}(A)]_m \tag{B.3}$$

$$= \quad \sum_{A \cap B =} ([\varphi^{\downarrow D_H}(A)]_m \cdot [\upsilon(B)]_m) = [(\varphi^{\downarrow D_H} \otimes \upsilon)()]_m. \tag{B.4}$$

Therefore,

$$[\varphi^{\downarrow D_H}(H)]_B \quad = \quad \frac{[(\varphi^{\downarrow D_H} \otimes \upsilon)()]_m - [\varphi^{\downarrow D_H}()]_m}{1 - [\varphi^{\downarrow D_H}()]_m} \tag{B.5}$$

$$= \quad \frac{[(\varphi \otimes \upsilon)^{\downarrow D_H}()]_m - [\varphi^{\downarrow D_H}()]_m}{1 - [\varphi^{\downarrow D_H}()]_m} \tag{B.6}$$

$$= \quad \frac{[(\varphi \otimes \upsilon)()]_m - [\varphi()]_m}{1 - [\varphi()]_m} \tag{B.7}$$

$$= \quad \frac{[(\varphi \otimes \upsilon)^{\downarrow D_r}()]_m - [\varphi^{\downarrow D_r}()]_m}{1 - [\varphi^{\downarrow D_r}()]_m} \tag{B.8}$$

$$= \quad \frac{c_2 - c_1}{1 - c_1} \tag{B.9}$$

□

References

Abraham, J.A. 1979. An Improved Algorithm for Network Reliability. *IEEE Transactions on Reliability*, **28**, 58–61.

Almond, R.G., & Kong, A. 1991 (November). *Optimality Issues in Constructing a Markov Tree from Graphical Models*. Research Report A-3. Department of Statistics, Harvard University.

Anrig, B., Haenni, R., & Lehmann, N. 1997a. *ABEL – A New Language for Assumption-Based Evidential Reasoning under Uncertainty*. Tech. Rep. 97–01. University of Fribourg, Institute of Informatics.

Anrig, B., Haenni, R., Kohlas, J., & Lehmann, N. 1997b. Assumption-Based Modeling Using ABEL. *Pages 171–182 of:* Gabbay, Dov M., Kruse, Rudolf, Nonnengart, Andreas, & Ohlbach, Hans Jürgen (eds), *Proceedings of the First International Joint Conference on Qualitative and Quantitative Practical Reasoning*. LNAI, vol. 1244. Berlin: Springer.

Anrig, B., Lehmann, N., & Haenni, R. 1997c. *Reasoning with Finite Set Constraints*. Working Paper 97–11. Institute of Informatics, University of Fribourg.

Anrig, B., Bissig, R., Haenni, R., Kohlas, J., & Lehmann, N. 1999. *Probabilistic Argumentation Systems: Introduction to Assumption-Based Modeling with ABEL*. Tech. Rep. 99-1. Institute of Informatics, University of Fribourg.

Arnborg, S., Corneil, D., & Proskurowski, A. 1987. Complexity of Finding Embedings in a k-Tree. *SIAM Journal of Algebraic and Discrete Methods*, **38**, 277–284.

Bauer, M. 1996. Approximations for Decision Making in the Dempster-Shafer Theory of Evidence. *Pages 73–80 of:* Horvitz, Eric, & Jensen, Finn (eds), *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*. San Francisco: Morgan Kaufmann Publishers.

Bertele, U., & Brioschi, F. 1972. *Nonserial Dynamic Programming*. Academic Press.

Bertschy, R., & Monney, P.A. 1996. A Generalization of the Algorithm of Heidtmann to Non-Monotone Formulas. *Journal of Computational and Applied Mathematics*, **76**, 55–76.

Besnard, P., & Kohlas, J. 1995. Evidence Theory Based on General Consequence Relations. *Int. J. of Foundations of Computer Science*, **6**(2), 119–135.

Bissig, R. 1996. *Eine schnelle Divisionsarchitektur für Belief-Netzwerke.* M.Phil. thesis, Institute of Informatics, University of Fribourg.

Bissig, R., Kohlas, J., & Lehmann, N. 1997. Fast-Division Architecture for Dempster-Shafer Belief Functions. *Pages 198–209 of:* M.Gabbay, Dov, Kruse, R., Nonnengart, A., & Ohlbach, H.J. (eds), *First International Joint Conference on Qualitative and Quantitative Practical Reasoning ECSQARU-FAPR'97, Bad Honnef.* Springer-Verlag.

Bütikofer, M. 2000. *A new trust management for PGP.* M.Phil. thesis, Institute for Theoretical Computer Science, ETH Zürich.

Cano, A., & Moral, S. 1995. Heuristic Algorithms for the Triangulation of Graphs. *Pages 166–171 of:* Bouchon-Meunier, B., Yager, R.R., & Zadeh, L.A. (eds), *Proceedings of the Fifth IPMU Conference.* Springer.

Chang, C.L., & Lee, R.C.T. 1973. *Symbolic Logic and Mechanical Theorem Proving.* Boston: Academic Press.

de Kleer, J. 1986a. An Assumption-based TMS. *Artificial Intelligence*, **28**, 127–162.

de Kleer, J. 1986b. Extending the ATMS. *Artificial Intelligence*, **28**, 163–196.

Dempster, A. P. 1967. Upper and Lower Probabilities Induced by a Multivalued Mapping. *Annals of Mathematical Statistics*, **38**, 325–339.

Dempster, A. P. 1968. A generalization of Bayesian inference. *Journal of the Royal Statistical Society*, **30 (Series B)**, 205–247.

Dubois, D., & Prade, H. 1986. *Possibility Theory: An Approach to Computerized Processing of Uncertainty.* New York: Plenum Press.

Dugat, V., & Sandri, S. 1994. Complexity of Hierarchical Trees in Evidence Theory. *ORSA Journal on Computing*, **6**, 37–49.

Feller, W. 1968. *An Introduction to Probability Theory and its Applications.* Third edn. Vol. 1. John Wiley and Sons.

Gries, D., & Schneider, F.B. 1993. *A Logical Approach to Discrete Math.* Springer-Verlag.

Haenni, R. 1998. Modeling Uncertainty with Propositional Assumption-Based Systems. *Pages 446–470 of:* Parson, S., & Hunter, A. (eds), *Applications of Uncertainty Formalisms*. Lecture Notes in Artifical Intelligence 1455. Springer-Verlag.

Haenni, R., & Lehmann, N. 1998. Reasoning with Finite Set Constraints. *Pages 1–6 of: ECAI'98, Workshop W17: Many-valued logic for AI application*.

Haenni, R., & Lehmann, N. 1999. *Efficient Hypertree Construction*. Tech. Rep. 99–03. University of Fribourg, Institute of Informatics.

Haenni, R., Kohlas, J., & Lehmann, N. 2000. Probabilistic Argumentation Systems. *In:* Kohlas, J., & Moral, S. (eds), *Defeasible Reasoning and Uncertainty Management Systems: Algorithms*. Kluwer.

Hähnle, R. 1994. Short Conjunctive Normal Forms in Finitely-Valued Logics. *Journal of Logic and Computation*, **4**(6), 905–927.

Hähnle, R., & Escalada-Imaz, G. 1997. Deduction in Many-Valued Logics: a Survey. *Mathware & Soft Computing*, **IV**(2), 69–97.

Heidtmann, K.D. 1989. Smaller Sums of Disjoint Products by Subproduct Inversion. *IEEE Transactions on Reliability*, **38**(3), 305–311.

Jensen, F.V., Olesen, K.G., & Andersen, S.K. 1990a. An Algebra of Bayesian Belief Universes for Knowledge-Based Systems. *Networks*, **20**(5), 637–659.

Jensen, F.V., Lauritzen, S.L., & Olesen, K.G. 1990b. Bayesian Updating in Causal Probabilistic Networks by Local Computations. *Computational Statistics Quarterly*, **4**, 269–282.

Kennes, R., & Smets, P. 1990. Computational Aspects of the Möbius Transform. *Pages 344–351 of: Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence*.

Kjærulff, U. 1990. *Triangulation of Graphs – Algorithms Giving Total State Space*. Tech. Rep. R 90–09. Department of Mathematics and Computer Science, Aalborg University.

Kohlas, J. 1981. The Reliability of Reasoning with Unreliable Arguments. *Annals of Operations Research*, **32**, 67–113.

Kohlas, J. 1997. Allocation of Arguments and Evidence Theory. *Theoretical Computer Science*, **171**, 221–246.

Kohlas, J., & Haenni, R. 1996. *Assumption-Based Reasoning and Probabilistic Argumentation Systems*. Tech. Rep. 96–07. Institute of Informatics, University of Fribourg.

Kohlas, J., & Monney, P.A. 1993. Probabilistic Assumption-Based Reasoning. *In:* Heckerman, & Mamdani (eds), *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*. Kaufmann, Morgan Publ.

Kohlas, J., & Monney, P.A. 1995. *A Mathematical Theory of Hints. An Approach to the Dempster-Shafer Theory of Evidence.* Lecture Notes in Economics and Mathematical Systems, vol. 425. Springer-Verlag.

Kong, A. 1986. *Multivariate Belief Functions and Graphical Models.* Ph.D. thesis, Department of Statistics, Harvard University.

Kreinovich, V.Y., Bernat, A., Borrett, W., Mariscal, Y., & Villa, E. 1994. Monte-Carlo methods make Dempster-Shafer formalism feasible. *Pages 175–191 of:* Yager, R.R., Kacprzyk, J., & Fedrizzi, M. (eds), *Advances The Dempster-Shafer Theory of Evidence.* New York: John Wiley and Sons.

Laskey, K.B., & Lehner, P.E. 1989. Assumptions, Beliefs and Probabilities. *Artificial Intelligence*, **41**, 65–77.

Lauritzen, S.L., & Jensen, F.V. 1997. Local Computation with Valuations from a Commutative Semigroup. *Annals of Mathematics and Artificial Intelligence*, **21**, 51–69.

Lauritzen, S.L., & Spiegelhalter, D.J. 1988. Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems. *Journal of Royal Statistical Society*, **50**(2), 157–224.

Lehmann, N. 1994. *Entwurf und Implementation einer annahmenbasierten Sprache.* Diplomarbeit. Institute of Informatics, University of Fribourg.

Lehmann, N., & Haenni, R. 1999. An alternative to outward propagation for Dempster-Shafer belief functions. *Pages 256–267 of:* Hunter, Anthony, & Parsons, Simon (eds), *Proceedings of the 5th European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU-99).* LNAI, vol. 1638. Berlin: Springer.

Lepar, V., & Shenoy, P. P. 1998. A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions. *Pages 328–337 of:* Cooper, Gregory F., & Moral, Serafín (eds), *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98).* San Francisco: Morgan Kaufmann.

Lu, J. 1996. Logic Programming with Signs and Annotations. *Journal of Logic and Computation*, **6**(6), 755–778.

Lu, J., Murray, N., & Rosenthal, E. 1994. *A Framework for Automated Reasoning in Multiple-Valued Logics.* Technical Report 94-04. State University of New York at Albany.

Maurer, U.M. 1996. Modeling a Public-Key Infrastructure. *In: Proc. of the European Symposium on Research in Computer Security ESORICS'96.* Lecture Notes in Computer Science.

Monney, P.-A., & Anrig, B. 2000. Computing the Probability of Formulas Representing Events in Product Spaces. *Pages 197–208 of:* Bouchon-Meunier, B., Yager, R.R., & Zadeh, L.A. (eds), *Information, Uncertainty and Fusion*. Kluwer Academic Publishers.

Murray, N.V., & Rosenthal, E. 1994. Adapting Classical Inference Techniques to Multiple-Valued Logics using Signed Formulas. *Fundamenta Informaticae*, **21**(3), 237–253.

Pearl, J. 1986. Fusion, Propagation and Structuring in Belief Networks. *Artificial Intelligence*, **29**, 241–288.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publ. Inc.

Pearl, J. 1990. Reasoning with belief functions: an analysis of compatibility. *International Journal of Approximate Reasoning*, **4**, 363–389.

Picard, J. 2000. *Probabilistic Argumentation Systems applied to Information Retrieval*. M.Phil. thesis, Institut Interfacultaire d'Informatique, University of Neuchatel.

Picard, J., & Haenni, R. 1998. Modeling Information Retrieval with Probabilistic Argumentation Systems. *In: Proceedings of the 20th BCS-IRSG Annual Colloquium*.

Rivest, L. R., Shamir, A., & Adlemann, M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, **21**(2), 120–126.

Rose, D.J. 1970. Triangulated Graphs and the Elimination Process. *Journal of Mathematical Analysis and Applications*, **32**, 597–609.

Shafer, G. 1976. *A Mathematical Theory of Evidence*. Princeton: Princeton University Press.

Shafer, G. 1979. Non-Additive Probabilities in the Work of Bernoulli and Lambert. *Archive for History of Exact Sciences*, **19**, 309–370.

Shafer, G. 1991. *An Axiomatic Study of Computation in Hypertrees*. Working Paper 232. School of Business, The University of Kansas.

Shafer, G., Shenoy, P. P., & Mellouli, K. 1987. Propagating belief functions in qualitative Markov trees. *International Journal of Approximate Reasoning*, **1**(4), 349–400.

Shenoy, P. P. 1989. A Valuation-Based Language for Expert Systems. *International Journal of Approximate Reasoning*, **3**(5), 383–411.

Shenoy, P. P. 1991. On Spohn's Theory of Epistemic Beliefs. *Lecture Notes in Computer Science*, **521**, 2–??

Shenoy, P. P. 1992. Valuation Based Systems: A Framework for Managing Uncertainty in Expert Systems. *Pages 83–104 of:* Zadeh, L.A., & Kacprzyk, J. (eds), *Fuzzy Logic for the Management of Uncertainty*. John Wiley and Sons.

Shenoy, P. P. 1994. Conditional Independence in Valuation-Based Systems. *International Journal of Approximate Reasoning*, **10**(3), 203–234.

Shenoy, P. P. 1997. Binary Join Trees for Computing Marginals in the Shenoy-Shafer Architecture. *International Journal of Approximate Reasoning*, **17**(2–3), 239–263.

Shenoy, P. P., & Kohlas, J. 2000. Computation in Valuation Algebras. *In:* Kohlas, J., & Moral, S. (eds), *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 5: Algorithms for Uncertainty and Defeasible Reasoning*. Kluwer, Dordrecht.

Shenoy, P. P., & Shafer, G. 1990. Axioms for Probability and Belief Function Propagation. *Pages 169–198 of:* Shachter, R.D., & al. (eds), *UAI*. North-Holland, Amsterdam.

Smets, Ph. 1988. Belief functions. *Pages 253–286 of:* Smets, Ph., Mamdani, A., Dubois, D., & Prade, H. (eds), *Non-Standard Logics for Automated Reasoning*. London: Academic Press.

Smets, Ph. 1992. Resolving misunderstandings about belief functions. *International Journal of Approximate Reasoning*, **6**, 321–344.

Spohn, W. 1987. Ordinal Conditional Functions: A Dynamic Theory of Epistemic States. *Pages 105–134 of:* Harper, W., & Skyrms, B. (eds), *Causation in Decision, Belief Change and Statistics*, vol. 2. Dordrecht: D. Reidel.

Tarjan, R.E., & Yannakakis, M. 1984. Simple Linear Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal of Computing*, **13**, 566–579.

Tessem, B. 1993. Approximations for Efficient Computation in the Theory of Evidence. *Artificial Intelligence*, **61**(2), 315–329.

Thoma, H. M. 1991. Belief Function Computations. *Pages 269–307 of:* I.R. Goodman, M.M. Gupta, H.T. Nguyen, & Rogers, G.S. (eds), *Conditional Logic in Expert Systems*. Elsevier Science.

Wilson, N. 1991. A Monte-Carlo Algorithm for Dempster-Shafer Belief. *Pages 414–417 of:* D'Ambrosio, Bruce D., Smets, Philippe, & Bonissone, Piero P. (eds), *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*. San Mateo, CA, USA: Morgan Kaufmann Publishers.

Xu, H. 1991. *An Efficient Implementation of Belief Function Propagation*. Tech. Rep. 91–4. IRIDIA, Université Libre de Bruxelles.

Xu, H. 1995. Computing marginals for arbitrary subsets from marginal representation in Markov trees. *Artificial Intelligence*, **74**, 177–189.

Xu, H., & Kennes, R. 1994. Steps Toward Efficient Implementation of Dempster-Shafer Theory. *Pages 153–174 of:* R.R. Yager, J. Kacprzyk, & M. Fedrizzi (eds), *Advances in the Dempster-Shafer Theory of Evidence.* John Wiley and Sons, New York.

Yannakakis, M. 1981. Computing the Minimum Fill-in is NP-complete. *SIAM Journal of Algebraic and Discrete Methods*, **2**, 77–79.

Zadeh, L. A. 1987. A Theory of Approximate Reasoning. *Pages 367–412 of:* Yager, R. R., Ovchinnikov, S., Tong, R. M., & Nguyen, H. T. (eds), *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh.* New York: John Wiley & Sons, Inc.

Index

Curriculum Vitae

Personalien

Vorname	:	Norbert
Name	:	Lehmann
Adresse	:	St.Wolfgang 2
Wohnort	:	3186 Düdingen
Geburtsdatum	:	8. April 1971
Geburtsort	:	Freiburg
Telefon	:	026 / 493 23 65
Heimatort	:	Wünnewil
Zivilstand	:	ledig

Schulbildung

1977 - 1983	:	Primarschule in Düdingen
1983 - 1986	:	Sekundarschule in Düdingen
1986 - 1990	:	Matura Typus C, Kollegium St.Michael in Freiburg
1990 - 1994	:	Informatikstudium mit Nebenfach Mathematik an der Universität Freiburg

Sprachen

- Muttersprache Deutsch
- Gute Kenntnisse in Französisch
- Gute Kenntnisse in Englisch