Distributed Anytime Generic Inference in Valuation Algebras

Abhishek Dasgupta

Exeter College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2018

Dedicated to the memory of
my grandparents,
Jyotirmoy and Ava Dasgupta,
and my uncles,
Anup and Amit Dasgupta

Abstract

In this thesis, we construct a general theoretical framework for anytime inference which automatically gives us instantiations of inference in various important domains, such as Bayesian networks, relational algebras, disjunctive normal forms and set potentials in Dempster-Shafer theory. Our framework is based on local computation schemes for inference, which perform inference by message-passing on junction trees. We also undertake an analysis of distributed inference. Our theoretical framework is implemented as a software library to illustrate examples of anytime inference using the theoretical framework.

Exact inference in general is a #P-hard problem. Due to the prohibitive computational complexity, approximate inference is well-studied. In many applications a coarse approximation is sufficient. Ideally we would like such an approximation process to be anytime. In an anytime algorithm, the approximation improves monotonically with time, and the process can be interrupted and resumed without restarting. This is especially useful in domains where time may be limited such as continuous learning and robotics. While there are several domain-specific anytime algorithms, little work has been done on generalisation. It is advantageous to develop a theoretical framework which can work for several domains, including statistical and logical inference. We use the theory of generic inference, a unification of prior local computation schemes, as a starting point for our framework. In generic inference, information is represented as elements of an algebra called a valuation algebra, with certain axioms for approximate and exact inference. We introduce axioms and operations on valuation algebras to support anytime inference. We illustrate anytime inference with applications to several domains.

Computational overhead can also be reduced by making the algorithm distributed. For our algorithm, we analyse the tradeoffs between computational, communication and synchronisation costs. We give bounds on the number of processors for maximum concurrency and describe a processor assignment algorithm for optimisation of communication costs.

The overall aim of the thesis is to contribute to the study of anytime inference, and to do so in a manner that enables some measure of control over the degree of approximation and easy applicability to a wide range of domains.

Acknowledgements

I would like to thank my supervisor Samson Abramsky, without whose guidance this thesis would not have been possible, for the encouraging conversations, which never failed to counter my pessimism when things were not going according to plan, and even when they were. I also thank Bob Coecke and everyone in the Quantum Group for being such a nice, friendly group of people, and for the memories of open-air talks and barbecues.

I am grateful to the Clarendon Fund, Exeter College and the University of Oxford for funding my study here.

I also thank Kohei Kishida, Ninad Rajgopal and Mahima Mitra for their helpful comments on the thesis.

For making my time at Oxford memorable (other than the thesis, which was, of course, extremely fun), I thank my friends. There are too many that I would like to thank and I'm sure I will miss some of you, so I shall not attempt it, but I will cherish the memories I made in Oxford for as long as I live. I will remember the delicious potlucks, impromptu baking sessions, walks along the river, the random advice, poetry sessions, table tennis matches, punting and most important of all, the comforting camaraderie.

Last, but not the least, I am grateful to my parents for their unconditional love and support throughout the vicissitudes of my time here.

Contents

Chapter 1

Introduction

> "When you have eliminated the impossible, whatever remains, however improbable, must be the truth."
>
> – Arthur Conan Doyle, *Sherlock Holmes, The Sign of the Four*

Arriving at a conclusion from known premises is one of the most fundamental cognitive processes; we use it every time we pick up an umbrella when it is cloudy (prediction of rain), and adjust our route when we hear of a traffic jam on the radio (prediction of delay). This process is that of inference, and in its formalised version is one of the most important areas of research in computer science. Inference in the form of statistical inference is a fundamental component in machine learning [Bishop, 2006], which itself finds application in solving problems in the areas of computer vision and speech recognition. Logical inference [Smith, 2003] in deriving conclusions or entailments from true premises, finds application in constraint satisfaction problems [Russell et al., 2003], relational databases [Date, 1975], theorem provers and formal verification. Inference is generally computationally expensive, which has led to significant work in heuristic approaches and in approximate inference [Dechter, 1998, Koller et al., 1999, Heskes et al., 2002]

This thesis explores approximate inference using the framework of generic inference which unifies the statistical and logical aspects of inference. We look at the inference problem from an algebraic, rather than statistical point of view, using the language of valuation algebras. Valuation algebras abstract the notion of inference across several domains like belief functions in Dempster-Shafer theory, statistical inference in probabilistic graphical models, resolution in disjunctive normal forms and querying in relational databases, among

others [Pouly and Kohlas, 2011]. The notion of a valuation is that it is a piece of information, which can take the form of a conditional probability distribution or a disjunctive normal form.

Generally, inference problems are in the complexity class #P-hard [Russell et al., 2003], where #P is the set of counting problems associated with the problems in NP. We use the approximation method of ordered valuation algebras introduced in [Haenni, 2004], which uses a partial order to formulate information approximation. This framework is extended to support anytime inference, where the exact solution of the inference problem is monotonically approached with increasing time allocated to the algorithm. Our framework preserves the genericity, allowing its application to various useful instances of valuation algebras. We provide constructive proofs of convergence of our anytime inference algorithm and consider the problem from a software architecture point of view, providing a proof-of-concept implementation as well as an analysis of the tradeoffs involved in a distributed version of our algorithm.

This chapter is organised as follows: Section 1.1 gives a general background to the thesis, introducing the reader to the generalised distributive law which is one of the key predecessors of the valuation algebra framework. This section shows the significance of the distributed law in achieving efficiency gains. Section 1.2 gives a more detailed introduction to the two strands of our thesis: namely anytime, approximate inference in valuation algebras and an analysis of the tradeoffs involved in distributed inference. Section 1.3 gives the organisation of the thesis, with the key contributions highlighted.

1.1 Background

The broad area of our thesis is, as mentioned before, in the area of inference algorithms, coupled with the theory of generic inference, which we review in chapter 2. In this section we give a brief history of the research area and its general objective.

For computer scientists, programmers and mathematicians, one of the most fundamental notions is that of generalisation of mathematical structures, algorithms and code. General-

isation helps in the elucidation of hitherto latent structure, and saves time by unifying various formalisms, algorithms or mathematical theories into one framework. It embodies the write it once principle in programming, which is utilised at all levels, from loops and subroutines at the low level to object oriented programming and types in functional programming. The examples of generalisation in mathematics and theoretical computer science are too numerous to list; a few notable ones in theoretical computer science and programming languages are the formulation of category theory as a generalisation of monoids and homomorphisms, generalised algebraic data types in functional programming languages like Haskell, and generic programming constructs like templates in C++ or generics in Java.

In this thesis, we focus on a particular kind of genericity that encompasses:

Logical systems These include systems of disjunctive normal forms and problems like satisfiability which can be expressed in terms of a more generic theory.

Statistical inference in probabilistic graphical models Inference in probabilistic graphical models typically involves focusing on the probability distribution of a particular set of variables, given certain evidence, by marginalising out portions of the full underlying joint probability distribution. The underlying graph in the probabilistic graphical model can be either directed (representing Bayesian networks) or undirected (representing Markov networks).

Fourier and Hadamard transforms Hadamard transforms (used in quantum computation and data encryption) and Fourier transforms (extensively used in signal processing) can be formulated in terms of this generic framework.

Belief functions in Dempster-Shafer theory. Belief functions are a generalisation of classical probability theory, modelling the degree of belief in a proposition as a real number assigned to a set of events rather than to a single event. This allows one to model ignorance as well as ambiguity. Belief functions are used in sensor networks and data fusion. Inference in belief functions is also expressible in this framework as shown in [Shenoy and Shafer, 1990].

Each of these systems has been studied extensively but

the commonality in the algorithms relating to these varied structures was first mentioned by Shenoy-Shafer in their unification of inference in belief functions and inference in probabilistic systems as in the Lauritzen-Spiegelhalter architecture [Lauritzen and Spiegelhalter, 1990]. An extensive unification was performed by Aji and McEliece in their article on the generalised distributive law [Aji and McEliece, 2000] which is key to making the algorithms relating to these problems more efficient. In fact, the mathematical structure of valuation algebras (reviewed in Chapter 2) can be viewed as a direct descendant of the early version of the framework in [Shenoy and Shafer, 1990, Shenoy, 1997], the application of local computation to probabilistic graphical models in [Lauritzen and Spiegelhalter, 1990] and the generalised distributive law of [Aji and McEliece, 2000].

In the following sections, we show the utility of the distributive law (section 1.1.1), followed by a review of the notion of a generalised distributive law (section 1.1.2) as discussed in [Aji and McEliece, 2000].

1.1.1 Utility of the distributive law

We are familiar with the (left-hand) distributive law in arithmetic

$$a \cdot (b + c) = a \cdot b + a \cdot c \tag{1.1}$$

where $a, b, c \in \mathbb{R}$. There is an corresponding right-hand distributive law:

$$(b + c) \cdot a = b \cdot a + c \cdot a \tag{1.2}$$

These two are equivalent because of the commutativity of $\cdot$. When, as in this case, both left and right-hand distributive laws hold, we say $\cdot$ is distributive over $+$.

If we consider the number of arithmetic operations, the left hand side has two operations, whereas the right has three. This can be considered a trivial application of the distributive law to improve efficiency by reducing the number of operations. However we can get greater improvements in efficiency in larger problems as we shall see in a couple of examples.

► **Example 1.1.** Consider the functions $\phi : A \times A \times A \times A \to \mathbb{R}$ and $\gamma : A \times A \to \mathbb{R}$, where A is a set with $|A| = m$. Tuples

in the domain of ϕ are denoted by (w, x, y, z), whereas those in the domain of γ are denoted (r, x). Suppose we have to compute $\alpha : A \times A \to \mathbb{R}$ and $\beta : A \to \mathbb{R}$:

$$\begin{aligned} \alpha(w, x) &= \sum_{y,z,r \in A} \phi(w, x, y, z)\gamma(r, x) \\ \beta(y) &= \sum_{x,z,w,r \in A} \phi(w, x, y, z)\gamma(r, x) \end{aligned}$$

Summing out the variables in the product of ϕ and γ is also known as marginalisation. For example, in the first equation we are obtaining $\alpha(w, x)$ by marginalising out the variables y, z, r. Similarly $\beta(y)$ is obtained by marginalising out the variables x, z, w, r. Let's look at the number of arithmetic operations required for computation of these functions, without recourse to the distributive law. Then in the calculation of $\alpha(w, x)$, for each of the m^2 terms in the domain of α, there are m^3 terms (corresponding to the marginalised variables y, z, r). Each of these terms requires a multiplication and summation, so a total of $2m^5$ operations are involved. Similarly for $\beta(y)$, we marginalise x, z, w, r, and for each of the m terms, we have corresponding m^4 terms on the right hand side, making a total of $2m^5$ operations again. Thus, if we do not take advantage of the distributive law, we need a total of $4m^5$ operations.

On the other hand, using the distributive law, we get

$$\alpha(w, x) = \left(\sum_{y,z \in A} \phi(w, x, y, z) \right) \left(\sum_{r \in A} \gamma(r, x) \right) \tag{1.3}$$

This can be used to simplify the computation. We can rewrite the above as $\alpha(w, x) = \alpha_1(w, x)\alpha_2(x)$ where

$$\alpha_1(w, x) = \sum_{y,z \in A} \phi(w, x, y, z) \tag{1.4}$$

$$\alpha_2(x) = \sum_{r \in A} \gamma(r, x) \tag{1.5}$$

For α_1, the computation now only requires m^4 operations (m^2 entries in the domain of α_1 and m^2 entries corresponding to the marginalised variables y, z). Similarly $\alpha_2(x)$ requires m^2 operations, making a total of $m^4 + m^2$ operations. If we then compute the m^2 values of $\alpha(w, x) = \alpha_1(w, x)\alpha_2(x)$, that requires m^2 multiplications, making a total of $m^4 + 2m^2$ which is a significant reduction from the previous inefficient method which required $2m^5$ operations.

Similarly we can write $\beta(y)$ using the distributive law as

$$\begin{aligned}\beta(y) &= \sum_{x,z,w\in A} \phi(w,x,y,z)\left(\sum_{r\in A}\gamma(r,x)\right) \\ &= \sum_{x,w\in A} \phi(w,x,y,z)\alpha_2(x) \qquad (1.6)\end{aligned}$$

If we calculate the values of $\alpha_2(x)$, which takes m^2 operations and then use equation 1.6 ($2m^4$ operations), we get a total of $2m^4 + m^2$ operations, compared to $2m^5$ for the direct method.

Also, if we wanted to compute both $\alpha(w,x)$ and $\beta(y)$ then we could cache the output of $\alpha_2(x)$, thus needing to compute it only once. Then the computation of both $\alpha(w,x)$ and $\beta(y)$ would only take $3m^4 + 2m^2$ compared to $4m^5$ for the direct method.

This was a simple example. In the following section we shall review the notion of the generalised distributive law and its applications to more general examples.

1.1.2 Generalised distributive law

The generalised distributive law is the generalisation of the distributive law in arithmetic to commutative semirings. As we shall extensively use commutative semirings and other structures such as monoids and semigroups in later chapters, we present the definitions below before proceeding to discuss the generalised distributive law.

A commutative semiring can be defined in terms of a monoid, which in turn can be defined in terms of a semigroup. The definitions of semigroup, monoid, commutative semigroup, subsemigroup, commutative monoid are as in [Rosen, 1999, section 5.1.2]. The definition of semiring is from [Golan, 2013, chapter 1].

▶ **Definition 1.1.** SEMIGROUP. A semigroup is a set S together with a binary operation $\cdot : S \times S \to S$ that satisfies the associative property:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \forall a, b, c \in S$$

▶ **Definition 1.2.** MONOID. A monoid is a semigroup $(S, \cdot)$ with the identity element: there exists $e \in S$ such that for all $a \in S$, $e \cdot a = a \cdot e = a$.

▶ **Definition 1.3.** COMMUTATIVE SEMIGROUP. A commutative semigroup is a semigroup $(S, \cdot)$ where $\cdot$ is commutative, i.e. $a \cdot b = b \cdot a$ for all $a, b \in S$.

▶ **Definition 1.4.** SUBSEMIGROUP. A nonempty subset T of a semigroup $(S, \cdot)$ is a subsemigroup of S if T is closed under $\cdot$.

▶ **Definition 1.5.** COMMUTATIVE MONOID. A commutative monoid is a monoid $(S, \cdot)$ where $\cdot$ is commutative, i.e. $a \cdot b = b \cdot a$ for all $a, b \in S$.

▶ **Definition 1.6.** SEMIRING. A semiring is a set A with two binary operations $+ : A \times A \to A$ (addition) and $\cdot : A \times A \to A$ (multiplication), with two elements $0, 1 \in A$, such that:

1. $(A, +)$ is a commutative monoid with identity element 0.
2. $(A, \cdot)$ is a monoid with identity element 1.
3. Multiplication left and right distributes over addition; for all $a, b, c \in A$:
   (a) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   (b) $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$
4. Multiplication by 0 results in 0: for all $a \in A$, $a \cdot 0 = 0 \cdot a = 0$.

A semiring is denoted by $(A, +, \cdot, 0, 1)$. We also require $1 \neq 0$, as otherwise we would get the trivial semiring $A = \{0\}$.

▶ **Definition 1.7.** COMMUTATIVE SEMIRING. A semiring $(A, +, \cdot, 0, 1)$ is commutative iff $\cdot$ is a commutative operation, i.e. for all $a, b \in A$: $a \cdot b = b \cdot a$.

The generalised distributive law is the distributive law (point 3 of def. 1.6) in a commutative semiring $(A, +, \cdot, 0, 1)$. As $\cdot$ is commutative in a commutative semiring, this collapses the left and right-hand distributive laws in (def. 1.6) to one.

The distributive law in arithmetic is then a special case of this generalised distributive law, with the commutative semiring as $(\mathbb{R}^+, +, \cdot, 0, 1)$; the semiring operations being arithmetic addition and multiplication respectively.

1.1.3 Marginalisation of Product Form

In this section we review a problem called the marginalisation of product form (MPF). Several well-known problems can be formulated as instances of the MPF, like the Hadamard

transform and inference in Bayesian networks. This section is intended to be a brief introduction to the utility of the distributive law. A detailed discussion of the algorithm and complexity analysis can be found in [Aji and McEliece, 2000, Section III].

► **Notation.** We first define some useful notation in the interest of brevity and continuity with similar notation used in later chapters:

N1 Variables: Variables are denoted by lowercase letters, possibly with subscripts, from a *universe* $V = \{x_1, x_2, \ldots, y_1, y_2, \ldots, y, z, \ldots\}$. Sets of variables are denoted by capital letters, with optional subscripts.

N2 Frame: The set of possible values a variable x can take called the *frame* of x and is denoted by Ω_x. We also define the frame of a set of variables X as the Cartesian product of the frames of its component variables: $\Omega_X = \prod_{x \in X} \Omega_x$. This can be thought of as the set of all functions $\mathbf{x} : X \to \bigcup_{x \in X} \Omega_x$.
If $|\Omega_x| = 2$, then we say x is a *binary variable*.

N3 Tuples: A tuple with variables from $X \subseteq V$ is a function $\mathbf{x} : X \to \bigcup_{x \in X} \Omega_x$ with the condition that $\mathbf{x}(x) \in \Omega_x$ for all $x \in X$. This is thus an assignment of values to variables in X. We also interchangeably use $\mathbf{x}$ to denote such a function's unique representation in the tuple notation. The standard tuple notation allows us to write $\mathbf{x} \in \Omega_X$, and we call $\mathbf{x}$ an X-tuple.

N4 Tuple projection: The projection of an X-tuple $\mathbf{x}$ to a Y-tuple, where $Y \subseteq X$ is the tuple as a function restricted to the subdomain Y: $\mathbf{x} \upharpoonright_Y$. The projection in the tuple notation is denoted by $\mathbf{x}^{\downarrow Y}$.

N5 Tuple partition: We can represent the partition of $\mathbf{x}$, an X-tuple into tuples with disjoint domains, but whose union is X. For example an X-tuple can be split into a S-tuple for $S \subseteq X$ and an $X \setminus S$-tuple. This is denoted as $\mathbf{x} = (\mathbf{x}^{\downarrow S}, \mathbf{x}^{\downarrow X \setminus S})$.

N6 Functions: In this section we shall consider a particular class of functions denoted by $\Phi = \{\alpha_1, \alpha_2, \ldots, \beta, \gamma, \ldots\}$, where the functions are denoted by lowercase Greek letters. Each function in Φ has the type $\alpha : \prod_{x \in X} \Omega_x \to A$ for some

$X \subseteq V$ and $(A, +, \cdot, 0, 1)$ is a commutative semiring. We also indicate the type as $\alpha : \Omega_X \to A$ where $\Omega_X = \prod_{x \in X} \Omega_x$. As the domains of all functions are always Ω_X for some $X \subseteq V$, we refer to X as the *domain label* for a function $\alpha : \Omega_X \to A$. If the semiring A is implied or specified, we further abbreviate the type notation of a function $\alpha : \Omega_X \to A$ as just $\alpha : X$.

We can now proceed to the definition of the marginalisation of product form problem.

▶ **Definition 1.8.** PRODUCT FORM. We consider a set of functions $F = \{\phi_1, \phi_2, \ldots, \phi_n\}$, with $F \subseteq \Phi$, where Φ is the class of functions with the form $\alpha : \Omega_X \to A$. Each member function ϕ_i has domain label X_i ($\phi_i : \Omega_{X_i} \to A$). Then the product form π for the set of functions F is defined as:

$$\pi : \Omega_X \to A; \quad \pi(\mathbf{x}) = \phi_1(\mathbf{x}_1) \cdot \phi_2(\mathbf{x}_2) \cdot \ldots \cdot \phi_n(\mathbf{x}_n) \tag{1.7}$$

where $\mathbf{x}$ is an X-tuple, where $X = \cup_{i=1}^{n} X_i$ and $\mathbf{x}_i = \mathbf{x}^{\downarrow X_i}$.

▶ **Definition 1.9.** MARGINALISATION. Marginalisation of a function $\alpha : \Omega_X \to A$ to a function $\beta : \Omega_Y \to A$ is defined for $Y \subseteq X$ as

$$\beta(\mathbf{y}) = \sum_{\mathbf{z} \in \Omega_X,\ \mathbf{z}^{\downarrow T} = \mathbf{y}} \alpha(\mathbf{z}) \tag{1.8}$$

▶ **Definition 1.10.** MARGINALISATION OF PRODUCT FORM (MPF). Let π be the product form of $F = \{\phi_1, \phi_2, \ldots, \phi_n\}$. Then the marginalisation of product form problem is the computation of a set of functions $K = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ where each $\alpha_i \in K$ is a marginalisation over one or more variables of the product form π.

Thus an MPF problem can be represented as a 4-tuple $\langle V, A, F, K \rangle$, with V the universe of variables, $(A, +, \cdot, 0, 1)$ as the semiring, F being the set of functions which constitute the product form π, and K. Each function also has an associated domain label as described in the notation previously.

We also call $\phi_i \in F$ as *local kernels* and their domain labels X_i *local domain labels*. The product form π is termed the *global kernel*. Functions in K which have to be computed are called *objective functions*.

Now that we have defined the MPF, we can consider a few examples, which show how various problems can be mapped to the MPF. We start with the problem considered in example 1.1.

▶ **Example 1.2.** Here we have $V = \{r, w, x, y, z\}$, the semiring as the real semiring $(\mathbb{R}^+, +, \cdot, 0, 1)$, $F = \{\phi : \{w, x, y, z\}, \gamma : \{r, x\}\}$, and the set of objective functions $K = \{\alpha : \{w, x\}, \beta : \{y\}\}$, with

$$\begin{aligned} \alpha(w, x) &= \sum_{y,z,r \in A} \phi(w, x, y, z)\gamma(r, x) \\ \beta(y) &= \sum_{x,z,w,r \in A} \phi(w, x, y, z)\gamma(r, x) \end{aligned}$$

This naturally defines a MPF according to the definition above.

▶ **Example 1.3.** HADAMARD TRANSFORM. Let there be binary variables $x_1, x_2, x_3, y_1, y_2, y_3$, and $f(y_1, y_2, y_3)$ be a real-valued function. Consider the following MPF problem under the commutative semiring of $\mathbb{R}$

D1	$\{y_1, y_2, y_3\}$	$f(y_1, y_2, y_3)$
D2	$\{x_1, y_1\}$	$(-1)^{x_1 y_1}$
D3	$\{x_2, y_2\}$	$(-1)^{x_2 y_2}$
D4	$\{x_3, y_3\}$	$(-1)^{x_3 y_3}$
D5	$\{x_1, x_2, x_3\}$	1

Then the global kernel π and the objective function at local domain $H(x_1, x_2, x_3)$ are

$$\begin{aligned} \pi(x_1, x_2, x_3, y_1, y_2, y_3) &= f(y_1, y_2, y_3)(-1)^{x_1 y_1 + x_2 y_2 + x_3 y_3} \\ H(x_1, x_2, x_3) &= \sum_{y_1, y_2, y_3} f(y_1, y_2, y_3)(-1)^{x_1 y_1 + x_2 y_2 + x_3 y_3} \end{aligned}$$

Thus $H(x_1, x_2, x_3)$ is the Hadamard transform of $F(y_1, y_2, y_3)$ and can be expressed in the MPF form. It can also be shown that the Fourier transforms over any finite abelian group can be represented in an MPF form [Pouly and Kohlas, 2011, Section 3.12].

▶ **Example 1.4.** BAYESIAN NETWORK. A Bayesian network usually depicts a causal relationship between variables and represents a joint probability distribution. Consider the following example (fig. 1.1) of a causal network of a burglary alarm being triggered [Pearl, 1988, p49, section 2.2.4]. Mr. Holmes has a burglary alarm installed in his house. The nodes in this Bayesian network are binary variables:

- burglary: indicates whether a burglary has occurred.

- earthquake: indicates whether earthquake occurred.
- alarm: indicates whether alarm rang.
- radio-report: indicates whether there was a report on the radio (this would mean it is most probably an earthquake).
- watson-call: indicates whether Dr. Watson, Mr. Holmes' neighbour called to report a burglary.

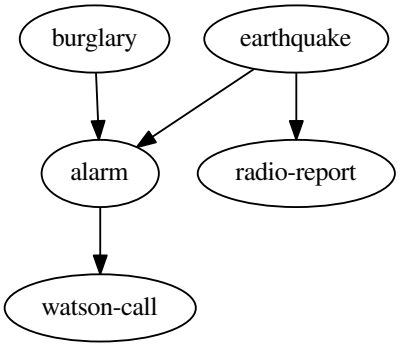


Figure 1.1: The alarm Bayesian network.

The alarm is triggered by a burglary (burglary $\rightarrow$ alarm) , but there is also a chance that an earthquake can trigger it (earthquake $\rightarrow$ alarm). However, an earthquake would presumably be reported on the radio (earthquake $\rightarrow$ radio-report). An alarm would cause Mr. Holmes' neighbour, Dr. Watson to call him (alarm $\rightarrow$ watson-call). Mr. Watson is a prankster, so it is possible that there was no alarm as well. The degree of uncertainty in each of these links is represented by a conditional probability distribution, which is the probability distribution of the variable conditioned on its parents. So for example, the conditional probability distribution of the alarm being triggered is $P(\text{alarm}|\text{burglary}, \text{earthquake})$. We discuss Bayesian networks in greater detail in chapter 4, where we discuss applications of our anytime inference framework. A similar instance is also discussed as an example for logical inference in chapter 4.

If we want to find a particular probability distribution, for example probability of alarm $P(\text{alarm})$, we have to compute the joint probability distribution $\pi(\text{watson-call}, \text{alarm}, \text{radio-report}, \text{burglary}, \text{earthquake})$

$$\begin{aligned}\pi(\text{watson-call}, \text{alarm}, \text{radio-report}, \text{burglary}, \text{earthquake}) = \\ P(\text{burglary}) \times P(\text{earthquake}) \times P(\text{alarm} \mid \text{burglary}, \text{earthquake}) \\ \times P(\text{radio-report} \mid \text{earthquake}) \times P(\text{watson-call} \mid \text{alarm})\end{aligned}$$

and then marginalise out necessary variables:

$$P(\text{alarm}) = \sum_{v \in W} \pi(\text{watson-call}, \text{alarm}, \text{radio-report}, \text{burglary}, \text{earthquake}) \tag{1.9}$$

where $W = \{\text{burglary}, \text{earthquake}, \text{radio-report}, \text{watson-call}\}$.

As we see, this problem maps quite naturally to a marginalisation of product form problem $\langle V, R, F, K \rangle$, if we consider:

- $V = \{\text{alarm}, \text{burglary}, \text{earthquake}, \text{radio-report}, \text{watson-call}\}$.
- The commutative semiring as the real semiring $(\mathbb{R}^+, +, \cdot, 0, 1)$.
- The set of local kernels are just the conditional probability distributions represented by the edges: $F = \{\phi_1 :$

$\{\text{burglary}\}$, $\phi_2 : \{\text{earthquake}\}$,
$\phi_3 : \{\text{alarm} \mid \text{burglary}, \text{earthquake}\}$, $\phi_4 : \{\text{radio-report} \mid \text{earthquake}\}$,
$\phi_5 : \{\text{watson-call} \mid \text{alarm}\}\}$

- The set of objective functions as the probability distributions we want to obtain from the product form (the joint probability distribution): $K = \{\alpha : \{\text{alarm}\}\}$.

In this section we have shown the applicability of the generalised distributive law to a couple of instances, namely Hadamard transforms and Bayesian networks. In the next chapter, we shall consider a generalisation of the ideas discussed in this chapter and introduce the framework of valuation algebras which is the key theoretical framework for this thesis.

1.2 Themes of the thesis

We have two main themes in our thesis - that of generic anytime inference in valuation algebras and an analysis of distributed inference.

1.2.1 Anytime Inference in Valuation Algebras

The core mathematical framework of our thesis is that of valuation algebras in the theory of generic inference. The framework of valuation algebras builds upon the generalised distributive law specified earlier, and abstracts the notion of information, taking specific forms like probability potentials, belief functions, disjunctive normal forms, Gaussian potentials, among others [Pouly and Kohlas, 2011]. In this theme we consider the anytime inference problem in the context of valuation algebras.

The research area of anytime algorithms evolved from the need to specify algorithms which could approximate in an elegant manner. Instead of an algorithm terminating after an unspecified amount of time, we are able to tune the accuracy via a parameter (usually time) passed to the algorithm. The algorithm could also be designed to be interruptible, gradually improving its accuracy until terminated by the user. Such algorithms are important in online learning where new data is being streamed in [Ueno et al., 2006], in intelligent systems, decision making under uncertainty [Horsch and Poole, 1998] and robotics [Zilberstein, 1996] where due to the limitation

of interacting in real-time there may not be sufficient time to compute an exact answer.

The core framework of valuation algebras, which arose out of unification [Shenoy and Shafer, 1990] of similar message-passing schemes for inference in belief functions and Bayesian networks is limited to exact inference. Due to the complexity considerations of exact inference, there is a natural interest in approximate inference schemes. Approximate inference schemes for specific instances of valuation algebras are abundant. A few examples of such work are approximate inference in probabilistic graphical models [van de Ven and Ramos, 2012, Ramos and Cozman, 2005] and belief functions [Haenni and Lehmann, 2002, Harmanec, 1999]. These frameworks often take advantage of instance-specific features, but, in doing so, do not offer much insight into features of the inference algorithm that can be generalised or lifted to a higher level of abstraction.

In contrast, approximation inference schemes based on valuation algebras offer the advantage of genericity, with an axiomatic framework which allows algorithms to be proved for correctness. In this area, we mention two articles in particular which can be considered as predecessors to the work described in the thesis. The first of these is the work on semiring induced valuations by [Kohlas and Wilson, 2008] which introduces a class of valuation algebras which generalises several well-known valuation algebra instances that are covered in the thesis, such as arithmetic potentials, disjunctive normal forms and relational algebras. They also implement approximate inference, in the form of deriving upper and lower bounds of the solution to the inference problem. We do not utilise this method as our goal is to get a sequence of better approximations for the solution to the inference problem.

The second, ordered valuation algebras [Haenni, 2004] discusses a approximate inference algorithm for valuation algebras. The approximate inference framework is however not anytime in the sense that the inference algorithm can be interrupted and resumed to give a better approximation.

The contribution in this strand of the thesis is to develop a framework which is sufficiently generic to allow us to capture a wide variety of instances. We improve upon the state of the art in this area by introducing a framework that supports interruptible anytime algorithms, augmenting the valuation

algebra axioms to support anytime inference. We also provide detailed proofs of correctness of the algorithm. The advantage of such an anytime inference framework is that it offers the user of the framework a degree of control over the approximation, while having a single algorithm support a wide variety of instances.

1.2.2 Analysis of Distributed Inference

In recent years, we have seen the rise of multi-core processors and an increasing emphasis on parallel computation as the sequential speed of single processors plateaus. The existence of large datasets has made it possible to perform statistical inference on them, leading to insights especially in the fields of complex systems and bioinformatics.

With the introduction of MapReduce [Dean and Ghemawat, 2008] there has been an increasing interest in the development of parallel frameworks. One of the problems of MapReduce is that it does not work well for problems which have anything but the most trivial dependencies between their data. In the past few years, new frameworks [Low et al., 2010, Haller and Miller, 2011, Malewicz et al., 2010] have been developed which try to address this issue. Most of these are based on representing the data in the form of a graph which represents the dependencies among the data. However, comparatively fewer efforts [Pace, 2012, Feldman et al., 2010, Karloff et al., 2010, Goodrich et al., 2011] have focused on analysing the communication costs or the tradeoffs involved in distributed computation in relation to programming frameworks like MapReduce or GraphLab.

Our focus in this part of the thesis is to analyse distributed algorithms in local computation using the framework introduced in [Solomonik et al., 2014], which generalises the well-studied Bulk Synchronous Parallel [Valiant, 1990] model. In distributed computation, there are three kinds of costs: communication (between processors), computation (within a processor), and synchronisation (between processors). Understanding the tradeoffs between these costs is vital to any implementation of distributed computation. For example, if we have an algorithm in which each step depends upon the previous step, there would be no point in making this algorithm distributed as each processor would have to wait for

the previous processor to finish, making the implementation effectively sequential.

We derive these tradeoffs and show that there is a maximum number of processors for a distributed computation of the inference problem in valuation algebras. In doing so, we present an optimal processor assignment algorithm and compare it with similar prior work [Pouly and Kohlas, 2005], noting the advantages of our approach.

1.3 Overview of the thesis

This section gives an overview of the thesis by chapter, along with the contributions of the thesis.

1.3.1 Overview of prior work

Chapter 1 is this introduction, which sets out the themes of the thesis and reviews the generalised distributive law, which is one of the key forerunners of the generic algebraic framework discussed in chapter 2.

Chapter 2 reviews the theoretical frameworks that form the basis of the contributions of this thesis.

As mentioned before, we shall be working in the framework of valuation algebras which abstract the notion of information. We review the axioms and show examples of application to arithmetic potentials, disjunctive normal forms, relational algebras and set potentials. Three of the instances discussed (arithmetic potentials, disjunctive normal forms and relational algebras) can also be expressed as instances of a particular class of valuation algebras: that which are induced by a semiring.

We look at how the inference problem is defined in such a framework and describe the inference algorithm. We then consider the literature on approximate inference, in particular, the framework of ordered valuation algebras, and look at approximate inference algorithms.

Chapter 6 considers the second theme of the thesis, that of distributed computation. We review an existing framework of communication, computation and synchronisation cost tradeoffs to look at the local computation problem in a distributed setting.

1.3.2 Contributions of the thesis

Chapter 3 is the first contribution of the thesis, which develops a framework of anytime inference as described in section 1.2.1.

We extend valuation algebras to support anytime inference, followed by an anytime inference algorithm which is guaranteed to converge to the solution of the exact inference algorithm. We prove that the convergence is monotonic (soundness) as well as guaranteed to converge to the exact solution (completeness). We also show that the class of semiring induced valuation algebras, which comprise arithmetic potentials, disjunctive normal forms and relational algebras, among others, is amenable to anytime inference. We also show that the valuation algebra of set potentials supports anytime inference. Further, we show that subject to an additional condition, anytime ordered valuation algebras are also ordered valuation algebras.

The wide variety of instances of anytime ordered valuation algebras illustrates the utility of working in a generic framework as we get several instantiations of our anytime inference algorithm for free.

Chapter 4 considers the application of our framework to the instances discussed in chapters 2 and 3, namely arithmetic potentials, disjunctive normal forms, relational algebras and set potentials. We also discuss interpretation of anytime inference with examples.

Chapter 5 describes a proof-of-concept implementation of the anytime inference algorithm introduced in Chapter 3, and describes the results of its application to a few instances of anytime ordered valuation algebras.

Chapter 6 derives tradeoffs for distributed local computation and obtains a bound on maximal concurrency. As part of this derivation we describe a processor assignment algorithm for optimisation of communication costs and prove optimality. The algorithm is compared to prior work, and the advantages of our approach noted.

Chapter 7 concludes the thesis, with a remark on future work.

Chapter 2

Generic Inference

The core mathematical framework of our thesis is that of valuation algebras in the theory of *generic inference*. Genericity is an important concept in mathematics and computer science – the capability of abstracting out the common features in algorithms and computations into a generic theory is a common pattern in the literature, often leading to an improvement in understanding and characterisation of computational processes and mathematical theories. In programming languages, we are familiar with concepts which illustrate this paradigm, like *generic programming*, where procedures are specified in terms of placeholder types (which are instantiated for concrete types like integers, floats and characters as needed), thus reducing code duplication, to concepts like *generalised algebraic data types* in functional programming languages like Haskell.

The utility of generic inference can be understood from a simpler example: sorting. Sorting algorithms can be defined on any totally ordered structure with an order relation $\leq$. All a sorting algorithm needs is the specification of the order relation; it does not matter whether the items being sorted are integers, real numbers or character strings. Generic inference generalises inference algorithms by abstracting the essential components of information in an algebraic structure. We obtain generic algorithms by lifting algorithms from specific instantiations of a structure (example: for sorting, a sorting algorithm that operates on character strings) to a generic, abstract version. In the particular case of generic inference, the valuation algebra structure was obtained from a similar lifting process applied to inference algorithms in the Lauritzen-Spiegelhalter architecture [Lauritzen and Spiegelhalter, 1990]

which defined an algorithm for solving the inference problem on Bayesian networks, using a technique called *local computation*. Shenoy and Shafer noted that the same algorithm could be used to solve the inference problem on belief functions, and proposed a sufficient set of axioms for an algebraic framework that is sufficient for the generic inference algorithm [Shenoy and Shafer, 1990]. This was extended by Kohlas into a theory of valuation algebras, and a computer implementation of the valuation algebra along with concrete instantiations was covered in [Pouly, 2010].

Following the aforementioned work, many instances of valuation algebras have been found in areas of mathematics and computer science, ranging from areas like Fourier transforms, logic and relational databases to systems of linear equations. Some of these instances will be described later in the chapter. In all these instances, the computationally interesting problem can be expressed in terms of the generic inference structure, which allows us to use the generic inference algorithm. The unification of these various, apparently disparate mathematical structures leads to a simpler and more coherent description of the inference problem. A comprehensive account is given in [Pouly and Kohlas, 2011].

The essence of generic inference and the algebra of valuations is considering information, which comes in pieces, and refers to questions from a particular domain; which can either be *combined* (to form, for example a joint probability distribution) or *projected* to focus on a particular aspect (variables corresponding to the query in an inference problem) of the information. Thus the valuation algebra has generic operations corresponding to the structure of knowledge and its transformation.

This chapter is organised into the following sections. We begin by reviewing the axioms of a valuation algebra in section 2.1. Section 2.2 and section 2.3 discuss instantiations of valuation algebras, such as arithmetic potentials (for Bayesian networks), disjunctive normal forms, set potentials and relational algebras.

Section 2.4 describes the inference problem in the context of valuation algebras. The algorithm for exact solution of the inference problem is described in section 2.7 using join trees.

Sections 2.8 through 2.11 review the ordered valuation algebra framework for approximate inference [Haenni, 2004].

Section 2.8 presents the axioms of ordered valuation algebras, with instances in sections 2.9 and 2.10. The approximate inference algorithm is described in section 2.11.

Section 2.12 summarises the chapter and concludes the discussion on generic inference.

2.1 Valuation Algebras

Valuation algebras have elements which represent information pieces or *valuations* in an abstract manner. All operations in the valuation algebra are defined on these elements. Before proceeding to the formal definition, we give an overview of the key components of the valuation algebra. A valuation algebra comprises (i) a *universe of valuations* where valuations refer to pieces of information about a set of variables, which are drawn from (ii) a *universe of variables*. We also have (iii) a *domain* operation which gives us the set of variables pertaining to a particular valuation, (iv) a *combination* operation which combines the information in two valuations, and (v) a *projection* (also known as *focusing* or *marginalisation*) operation which focuses a valuation on a subset of variables in a valuation domain.

▶ **Definition 2.1.** VALUATION ALGEBRA. A valuation algebra is a tuple $\langle \Phi, V, d, \otimes, \downarrow \rangle$ with the axioms *A1* – *A6* as shown below. The tuple components are as follows:

Universe of valuations (Φ) is a set.[1]

Universe of variables (V) is a set.[2]

Labelling (d) The labelling (domain) operation is $d : \Phi \to \mathcal{D}$.

Combination ($\otimes$) The combination operation is $\otimes : \Phi \times \Phi \to \Phi$.

Projection ($\downarrow$) The projection operation $\downarrow$: $\Phi \times \mathcal{D} \twoheadrightarrow \Phi$ is a partial function.[3]

The following axioms are then introduced on $\langle \Phi, V, d, \otimes, \downarrow \rangle$:

A1 Commutative semigroup: Φ is associative and commutative under $\otimes$.

A2 Labeling: For $\phi, \psi \in \Phi$,

$$d(\phi \otimes \psi) = d(\phi) \cup d(\psi) \tag{2.1}$$

[1] Members are denoted by lowercase Greek letters: $\phi, \psi, \ldots$

[2] Members are denoted by Roman lowercase letters (with possible subscripts): $x, y, \ldots$. Sets of variables are denoted by uppercase letters: $S, T, \ldots$. The powerset of V is denoted by $\mathcal{D}$.

[3] This projects or focuses a valuation ϕ to a domain $X \subseteq d(\phi)$. For $(\phi, X) \in \Phi \times \mathcal{D}$, we denote the projection operation as $\phi^{\downarrow X}$.

A3 Projection: For $\phi \in \Phi, X \in \mathcal{D}$ and $X \subseteq d(\phi)$

$$d(\phi^{\downarrow X}) = X \tag{2.2}$$

A4 Transitivity: For $\phi \in \Phi$ and $X \subseteq Y \subseteq d(\phi)$,

$$(\phi^{\downarrow Y})^{\downarrow X} = \phi^{\downarrow X} \tag{2.3}$$

A5 Combination: For $\phi, \psi \in \Phi$ with $d(\phi) = X$, $d(\psi) = Y$ and $Z \in D$ such that $X \subseteq Z \subseteq X \cup Y$,

$$(\phi \otimes \psi)^{\downarrow Z} = \phi \otimes \psi^{\downarrow Z \cap Y} \tag{2.4}$$

As $\otimes$ is commutative, we do not have left and right versions of this axiom.

A6 Domain: For $\phi \in \Phi$ with $d(\phi) = X$,

$$\phi^{\downarrow X} = \phi \tag{2.5}$$

A1 – A4 can be understood as following from the intuition of information in pieces being represented as valuations. *A1*, the commutative semigroup axiom expresses the notion that the order of combining information should not matter. *A2* refers to the fact that when we combine information, the questions that they refer to (the variables in their domains) should be aggregated too, in the natural definition of a set union. *A3* expresses the stability of the domain operation under projection. *A4*, transitivity, says that successive projections can be combined.

A5, the combination axiom states how inference is affected when a new piece of information arrives. Either we combine the new information and project it to the required domain, or we first focus the new information appropriately before combination. This axiom is key to the improvements in efficiency offered by valuation algebras. We can see that the axiom states that we do not need to combine two valuations first to find the projection, instead we can eliminate the part that we do not need and then combine which leads to less computation. This axiom is connected to the distributive law; the connection becomes clear in the proofs of the axioms for valuation algebra instances in the next section.

A6 expresses stability with respect to projection of a valuation to its own domain.

In the following sections (sections 2.2 and 2.3), we give some examples of valuation algebras. The material in these sections is mostly from [Pouly and Kohlas, 2011, chapter 1]. We shall consider these valuation algebra instances: arithmetic potentials, disjunctive normal forms, relational algebras and set potentials. The fact that a single framework encompasses such seemingly disparate instances is the primary reason for focusing on valuation algebras in this thesis.

In particular, there is a subclass of valuation algebras which encompasses arithmetic potentials, disjunctive normal forms, relational databases, among others. These are the valuation algebras induced by a semiring. Instances of semiring induced valuation algebras differ only in the semiring that induces them. Thus by studying a single class of valuation algebras, we get several instantiations for free. Out of the valuation algebra instances discussed in the thesis, all but one, that of set potentials are instances of semiring induced valuation algebras.

2.2 Semiring induced valuation algebras

Semiring induced valuation algebras are a subclass of valuation algebras with several useful instances like arithmetic potentials and disjunctive normal forms. We use the definition of semiring induced valuation algebras from [Kohlas and Wilson, 2008].

▶ **Definition 2.2.** SEMIRING INDUCED VALUATION ALGEBRA. A semiring induced valuation algebra is induced by a commutative semiring (def. 1.7) denoted by $\mathcal{A} = (A, +, \cdot, 0, 1)$ and the set of frames associated with variables $x \in V$, $\{\Omega_x : x \in V\}$, where V is the universe of variables. Here *frame* is the same definition as in *N2* from section 1.1.3. We also re-use the notation of *configuration space* Ω_D of a set of variables D and the notation of *tuples* from the notation in section 1.1.3.

Universe of valuations An valuation ϕ is a function $\phi : \Omega_D \to A$ for some $D \subseteq V$.[4] Then the universe of valuations is

$$\Phi = \bigcup_{D \subseteq V} \{\phi : \Omega_D \to A\} \tag{2.6}$$

with a fixed A for all functions ϕ.

[4] The set of all valuations with a domain D is denoted by Φ_D

Universe of variables The universe of variables is V.

Domain We define $d : \Phi \to \mathcal{D}$ as $d(\phi) = D$ iff $\phi : \Omega_D \to A$.

Combination The combination operator $\otimes : \Phi \times \Phi \to \Phi$ is defined as $\phi_1 \otimes \phi_2 : \Omega_{D_1 \cup D_2} \to A$ where $\phi_1 : \Omega_{D_1} \to A$, $\phi_2 : \Omega_{D_2} \to A$, where the function $\phi_1 \otimes \phi_2 : \Omega_{D_1 \cup D_2} \to A$ is:

$$(\phi_1 \otimes \phi_2)(\mathbf{x}) = \phi_1(\mathbf{x}^{\downarrow D_1}) \cdot \phi_2(\mathbf{x}^{\downarrow D_2}) \tag{2.7}$$

Projection The projection operator $\downarrow$: $\Phi \times D \twoheadrightarrow \Phi$ defined as $\phi^{\downarrow D'} : \Omega_{D'} \to A$ where $\phi : \Omega_D \to A$ and $D' \subseteq D$. The function $\phi^{\downarrow D'} : \Omega_{D'} \to A$ is defined[5] as

[5] The sum is well-defined as D, and thus Ω_D, is finite.

$$\phi^{\downarrow T}(\mathbf{x}) = \sum_{\mathbf{z} \in \Omega_D :\ \mathbf{z}^{\downarrow T} = \mathbf{x}} \phi(\mathbf{z}) \tag{2.8}$$

Then $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a semiring induced valuation algebra induced by the semiring $(A, +, \cdot, 0, 1)$.

We proceed to show that this tuple $\langle \Phi, V, d, \otimes, \downarrow \rangle$ as defined above satisfies the axioms of a valuation algebra.

▶ **Theorem 2.1.** *[Pouly and Kohlas, 2011, theorem 5.2] The structure $\langle \Phi, V, d, \otimes, \downarrow \rangle$ defined in def. 2.2 is a valuation algebra.*

Proof. We show that axioms *A1 – A6* are satisfied by $\langle \Phi, V, d, \otimes, \downarrow \rangle$.

A1 Commutative semigroup: We have to show that Φ is associative and commutative under $\otimes$. Commutativity under $\otimes$ naturally follows from the commutativity of the underlying semiring multiplicative operation $(\cdot)$. For associativity, consider valuations ϕ, ψ, ξ with domains X, Y, Z respectively. Then we have to show

$$\phi \otimes (\psi \otimes \xi) = (\phi \otimes \psi) \otimes \xi \tag{2.9}$$

Let's consider the LHS of (2.9). Then, for $\mathbf{x} \in \Omega_{X \cup Y \cup Z}$,

$$\begin{aligned}(\phi \otimes (\psi \otimes \xi))(\mathbf{x}) &= \phi(\mathbf{x}^{\downarrow X}) \cdot (\psi \otimes \xi)(\mathbf{x}^{\downarrow Y \cup Z}) \\ &= \phi(\mathbf{x}^{\downarrow X}) \cdot (\psi((\mathbf{x}^{\downarrow Y \cup Z})^{\downarrow Y}) \cdot \xi((\mathbf{x}^{\downarrow Y \cup Z})^{\downarrow Z})) \\ &= \phi(\mathbf{x}^{\downarrow X}) \cdot (\psi(\mathbf{x}^{\downarrow Y}) \cdot \xi(\mathbf{x}^{\downarrow Z}))\end{aligned}$$

A similar treatment for the RHS will give the same, due to associativity of $\cdot$ in the underlying semiring.

A2 Labeling: This follows directly from the definition of combination: $d(\phi \otimes \psi) = d(\phi) \cup d(\psi)$

A3 Projection: This follows directly from the definition of projection operation in semiring induced valuation algebras: $d(\phi^{\downarrow X}) = X$

A4 Transitivity: For $\phi \in \Phi$ and $X \subseteq Y \subseteq d(\phi)$, we have to show

$$(\phi^{\downarrow Y})^{\downarrow X} = \phi^{\downarrow X} \tag{2.10}$$

Let's denote $Z = d(\phi)$, then consider the LHS:

We have $(\mathbf{z}^{\downarrow Y})^{\downarrow X} = \mathbf{z}^{\downarrow X}$ which follows from a requirement of projection that $X \subseteq Y \subseteq Z$

$$\begin{aligned}
(\phi^{\downarrow Y})^{\downarrow X}(\mathbf{x}) &= \sum_{\mathbf{y}\in\Omega_Y:\ \mathbf{y}^{\downarrow X}=\mathbf{x}} \phi^{\downarrow Y}(\mathbf{y}) \\
&= \sum_{\mathbf{y}\in\Omega_Y:\ \mathbf{y}^{\downarrow X}=\mathbf{x}} \left(\sum_{\mathbf{z}\in\Omega_Z:\ \mathbf{z}^{\downarrow Y}=\mathbf{y}} \phi(\mathbf{z}) \right) \\
&= \sum_{\mathbf{y}\in\Omega_Y, \mathbf{z}\in\Omega_Z:\ \mathbf{y}^{\downarrow X}=\mathbf{x},\ \mathbf{z}^{\downarrow Y}=\mathbf{y}} \phi(\mathbf{z}) \\
&= \sum_{\mathbf{z}\in\Omega_Z:\ \mathbf{z}^{\downarrow X}=\mathbf{x}} \phi(\mathbf{z}) = \phi^{\downarrow X}(\mathbf{x}) = \text{RHS}
\end{aligned}$$

A5 Combination: For $\phi_1, \phi_2 \in \Phi$ with $d(\phi_1) = X$, $d(\phi_2) = Y$ and $Z \in \mathcal{D}$ such that $X \subseteq Z \subseteq X \cup Y$, we have to show that

$$(\phi_1 \otimes \phi_2)^{\downarrow Z} = \phi_1 \otimes \phi_2^{\downarrow Z \cap Y} \tag{2.11}$$

Expanding this:

$$\begin{aligned}
(\phi_1 \otimes \phi_2)^{\downarrow Z}(\mathbf{x}) &= \sum_{\mathbf{k}^{\downarrow Z}=\mathbf{x}} (\phi_1 \otimes \phi_2)(\mathbf{k}) \\
&= \sum_{\mathbf{k}^{\downarrow Z}=\mathbf{x}} \phi_1(\mathbf{k}^{\downarrow X}) \cdot \phi_2(\mathbf{k}^{\downarrow Y}) \\
&= \phi_1(\mathbf{x}^{\downarrow X}) \cdot \sum_{\mathbf{k}^{\downarrow Z}=\mathbf{x}} \phi_2(\mathbf{k}^{\downarrow Y}) \quad \text{using } \mathbf{k}^{\downarrow X} = (\mathbf{k}^{\downarrow Z})^{\downarrow X} = \mathbf{x}^{\downarrow X} \\
&= \phi_1(\mathbf{x}^{\downarrow X}) \cdot \sum_{\mathbf{y}\in\Omega_{Y-Z}} \phi_2(\mathbf{x}^{\downarrow Z\cap Y}, \mathbf{y})
\end{aligned}$$

On the other hand, the right hand side:

$$\begin{aligned}
(\phi_1 \otimes \phi_2{}^{\downarrow Z\cap Y})(\mathbf{x}) &= \phi_1(\mathbf{x}^{\downarrow X}) \cdot \phi_2{}^{\downarrow Z\cap Y}(\mathbf{x}^{\downarrow Z\cap Y}) \\
&= \phi_1(\mathbf{x}^{\downarrow X}) \cdot \sum_{\mathbf{k}^{\downarrow Z\cap Y}=\mathbf{x}^{\downarrow Z\cap Y}} \phi_2(\mathbf{k}) \\
&= \phi_1(\mathbf{x}^{\downarrow X}) \cdot \sum_{\mathbf{y}\in\Omega_{Y-Z}} \phi_2(\mathbf{x}^{\downarrow Z\cap Y}, \mathbf{y})
\end{aligned}$$

A6 Domain: This follows directly from the definition of the

projection operation: $\phi^{\downarrow X} = \phi$.

$$\phi^{\downarrow X} = \phi \tag{2.12}$$

This shows that a semiring induced valuation algebra follows the axioms and is indeed a valuation algebra. □

Having established the concept of semiring induced valuation algebras, we proceed to discuss three instances of it: arithmetic potentials, disjunctive normal forms and relational algebras. There are many other examples of semiring induced valuation algebras, a detailed introduction to which can be found in [Kohlas and Wilson, 2008]. In certain cases, the valuation algebra induced by the semiring has the idempotent property, i.e. $\phi \otimes \phi = \phi$; then we may use more efficient architectures for local computation such as the Lauritzen-Spiegelhalter architecture [Kohlas, 2003].

2.2.1 Arithmetic potentials

The valuation algebra of probability potentials was one of the first instances to be expressed in the framework. Probability potentials represent (conditional) probability distributions in Bayesian networks, and factors in Markov networks. In this section, we consider a simple, unnormalised form of the probability potential, termed *arithmetic potentials* here, defined as mappings from a configuration space to real values. We do not lose much by this simplification as arithmetic potentials can represent Bayesian networks, and normalisation can be performed at the end of the inference process.

Arithmetic potentials are a semiring induced valuation algebra, with the underlying semiring being $(\mathbb{R}^+, +, \cdot, 0, 1)$, where $\mathbb{R}^+$ is the set of non-negative reals. A arithmetic potential π on domain $d(\pi) = D$ is a function $\pi : \Omega_D \to \mathbb{R}^+$. Thus π is a mapping of the configuration space onto real numbers, where the numbers are proportional to the probability of that particular configuration.

► **Example 2.1.** [Pouly and Kohlas, 2011, section 1.3] As an example of computation with arithmetic potentials, we introduce a domain of three variables $R = \{a, b, c\}$ and the corresponding frames being all binary: $\Omega_a = \Omega_b = \Omega_c = \{0, 1\}$.

We show an example of the combination ($p_3 = p_1 \otimes p_2$) and projection ($p_3^{\downarrow\{a',c'\}}$) of arithmetic potentials in table 2.1.

a	b	p_1
0	0	0.6
0	1	0.4
1	0	0.3
1	1	0.7

b	c	p_2
0	0	0.2
0	1	0.8
1	0	0.9
1	1	0.1

a	b	c	$p_1 \otimes p_2$
0	0	0	0.12
0	0	1	0.48
0	1	0	0.36
0	1	1	0.04
1	0	0	0.06
1	0	1	0.24
1	1	0	0.63
1	1	0	0.07

a	c	$p_3^{\downarrow\{a',c'\}}$
0	0	0.48
0	1	0.52
1	0	0.69
1	1	0.31

Table 2.1: Combination and projection of arithmetic potentials.

2.2.2 Disjunctive normal forms

Disjunctive normal forms in propositional logic are disjunctions of conjunctive clauses. Here are the key terms in order to understand DNFs:

Literals Literals $x, \neg x$ represent the values $x = 1, x = 0$, where $x \in V$. respectively. If $D \subseteq V$ then $\Lambda_D = \{x, \neg x : x \in D\}$ denotes the set of all literals in D.

Term A term is a non-repetitive conjunction $\ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_n$ of literals $\ell_i \in \Lambda_D, 1 \leq i \leq n$; we can also consider terms as the equivalent sets $\{\ell_1, \ell_2, \ldots, \ell_n\} \subseteq \Lambda_D$ of literals.

DNF Formula We denote $\mathcal{T}_D = 2^{\Lambda_D}$ as the set of all possible terms. Then a DNF formula is a non-repetitive disjunction $\tau_1 \vee \cdots \vee \tau_m$ of terms $\tau_i \in \mathcal{T}_D$, $1 \leq i \leq m$. Similarly to literals, we can consider the equivalent sets of terms $\{\tau_1, \ldots, \tau_m\}$, as a representation of the DNF formula. We use $\mathcal{D}_D = 2^{\mathcal{T}_D}$ to denote the set of all possible formulae corresponding to D.

Full DNF formula A full DNF formula is one in which in each term each variable appears once as a literal, as itself or its negation.

Model An assignment to variables that makes the formula true is called a satisfying assignment. The set of satisfying assignments for a formula is called the model.

Disjunctive normal forms are a semiring induced valuation algebra, with the underlying semiring being the boolean semiring $(\{0,1\}, +, \cdot, 0, 1)$. This means valuations with a

domain $D \subseteq V$ are of the form $\delta : \Omega_D \to \{0,1\}$, where V is the universe of variables and Ω_D is the Cartesian product of the variable frames, which are all binary: $\Omega_D = \{0,1\}^{|D|}$

Alternatively, we can represent the function δ as the subset $M_D(\delta) \subseteq \Omega_D$ for which the image of δ is 1, i.e. $M_D(\delta)$ is the preimage of 1. Then $M_D(\delta)$ is the model of δ. In a disjunctive normal form, if any term is satisfied then the entire formula is satisfied. Going with the usual interpretation of 0 as $\bot$ (false) and 1 as $\top$ (true), each $\mathbf{x} \in \Omega_D$ that maps to 1 represents a term which contains all literals. Thus we can have a *bijective* mapping between the *model* and the *full disjunctive normal form* of a particular formula. The *truth table* is a list of the mapping of the configuration space Ω_D to $\{0,1\}$ which shows whether the formula is true (evaluates to 1) for a particular assignment to variables.

▶ **Example 2.2.** Let's look at a simple example of a valuation $\delta = ab + a\bar{c}$ (replacing $\vee$ with $+$, $\neg x$ as $\bar{x}$) with the domain $D = \{a, b, c\}$. Then the configuration space (all possible assignments to variables) is shown in table 2.2 along with the value of δ.

Thus the model $M_D(\delta) = \{(1,0,0), (1,1,0), (1,1,1)\}$, the subset of the configuration space which maps to 1. The model corresponds to the full DNF formula: $\delta' = a\bar{b}\bar{c} + ab\bar{c} + abc$, which is logically equivalent to $\delta = ab + a\bar{c}$:

a	b	c	δ
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	**1**
1	1	1	**1**

Table 2.2: Truth table for the formula $\delta = ab + a\bar{c}$

$$
\begin{aligned}
\delta' &= a\bar{b}\bar{c} + ab\bar{c} + abc \\
&= a\bar{b}\bar{c} + ab\bar{c} + ab\bar{c} + abc && \text{as } x + x = x \\
&= a\bar{c}(\bar{b} + b) + ab(\bar{c} + c) && \text{using distributivity} \\
&= a\bar{c} + ab && \text{as } x + \bar{x} = 1
\end{aligned}
$$

▶ **Example 2.3.** Now let's look at combination and projection of disjunctive normal forms. We have valuations corresponding to the formulae $\xi_1 = a \to b, \xi_2 = b \to c, \xi_3 = \xi_1 \otimes \xi_2$ and $\xi_3^{\downarrow\{a,c\}}$.

2.2.3 Relational algebras

Relational algebra is used in the theory of databases, to model relational databases. They form the basis of database query languages such as SQL.

Relational algebras are an instance of semiring induced valuation algebra, with the underlying semiring being the

a	b	$a \to b$
0	0	1
0	1	1
1	0	0
1	1	1

b	c	$b \to c$
0	0	1
0	1	1
1	0	0
1	1	1

a	b	c	ξ_3
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

a	c	$\xi_3^{\downarrow\{a,c\}}$
0	0	1
0	1	1
1	0	0
1	1	1

Table 2.3: Configuration spaces of the disjunctive normal forms $\xi_1 = a \to b$, $\xi_2 = b \to c$, $\xi_3 = \xi_1 \otimes \xi_2$, $\xi_3^{\downarrow\{a,c\}}$

boolean semiring $(\{0,1\}, +, \cdot, 0, 1)$. This means valuations with a domain $D \subseteq V$ are of the form $\phi : \Omega_D \to \{0,1\}$, where V is the universe of variables. Unlike in the previous instance of disjunctive normal forms, in relational algebras the frames Ω_x for $x \in V$ need not be binary.

Alternatively, we can represent the function ϕ as the subset $R \subseteq \Omega_D$ for which the image of ϕ is 1, i.e. R is the preimage of 1. Here, R is a relation, comprising tuples $\mathbf{x} \in \Omega_D$. Using the language of the theory of databases, the variables $x, y, z, \ldots$ are called *attributes* and sets of tuples are called *relations*. The set of valuations is then the set of all possible tuples over all possible subsets of variables:

$$\Phi = \bigcup_{S \subseteq V} 2^{\Omega_s} \tag{2.13}$$

where 2^{Ω_S} is the powerset of Ω_S. In this instance, the valuation algebra operations are well known operations in relational algebras, so we introduce them:

Combination Combination is defined by the *natural join* ($\bowtie$): If R_1, R_2 are relations with domains S, T respectively, then

$$R_1 \otimes R_2 = R_1 \bowtie R_2 = \{\mathbf{x} \in \Omega_{S \cup T} : \mathbf{x}^{\downarrow S} \in R_1, \mathbf{x}^{\downarrow T} \in R_2\} \tag{2.14}$$

Projection Projection is defined for a relation R with domain S and $T \subseteq S$ as

$$R^{\downarrow T} = \{\mathbf{x}^{\downarrow T} : \mathbf{x} \in R\} \tag{2.15}$$

Alternatively, in the language of relational algebras, the notation $\pi_T(R) = R^{\downarrow T}$ is used, where π is the projection operator.

► **Example 2.4.** An example of the combination and projection operations in relational algebra follows. If there are two relations R_1, R_2, with $d(R_1) = \{\text{fruit}, \text{colour}\}$ and $d(R_2) = \{\text{fruit}, \text{calories per 100g}\}$.

fruit	colour
banana	green
orange	orange
apple	red
pomegranate	red
blueberry	indigo
strawberry	red

Table 2.4: R_2

fruit	calories per 100g
banana	89
apple	100
blueberry	57
strawberry	37

fruit	colour	calories per 100g
banana	green	80
apple	red	100
blueberry	indigo	57
strawberry	red	37

Table 2.5: $\pi_{\{\text{colour}\}}(R_3) = R_3^{\{\text{colour}\}}$

colour
green
red
indigo

2.3 Set potential valuation algebra

Set potentials are a general case of belief potentials introduced in Dempster-Shafer's theory of evidence [Shafer et al., 1976]. They are unnormalised versions of belief potentials. The advantage of belief potentials over standard probability theory is in their ability to express ignorance or partial information in a manner not possible in probability theory. This is the reason for the extensive use of belief functions in the sensor network literature, which involves fusion of information from various sources, not all them equally reliable [Denoeux, 2000, Yu and Frincke, 2005, Sentz and Ferson, 2002, Murphy, 1998].

► **Definition 2.3.** SET POTENTIAL VALUATION ALGEBRA. [Pouly and Kohlas, 2011, section 5.7] As in the previous case, a variable $x \in V$ has a finite frame Ω_x. Also the Cartesian product Ω_D is the set of configurations corresponding to the domain $D \subseteq V$. We also recall the operations of join and projection from relational algebras, defined in section 2.2.3:

If $R_1 \subseteq \Omega_S, R_2 \subseteq \Omega_T$, the join operation is

$$R_1 \bowtie R_2 := \{\mathbf{x} \in \Omega_{S \cup T} : \mathbf{x}^{\downarrow S} \in R_1, \mathbf{x}^{\downarrow T} \in R_2\}$$

and the projection is, where $T \subseteq S$ here:

$$\pi_T(R) := R^{\downarrow T} := \{\mathbf{x}^{\downarrow T} : \mathbf{x} \in R\}$$

The elements of the valuation algebra here are set potentials. A *set potential* on a domain D is defined by a function $\phi : 2^{\Omega_D} \to \mathbb{R}^+$. The subsets $A \subseteq \Omega_D$ for which $\varphi(A) \neq 0$ are known as *focal sets*. Then $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a valuation algebra where the components of the valuation algebra are as follows

Universe of valuations The universe of valuations is

$$\Phi = \bigcup_{D \subseteq V} \{\phi : 2^{\Omega_D} \to \mathbb{R}^+\} \tag{2.16}$$

Universe of variables The universe of variables is V.

Domain For a set potential $\phi : 2^{\Omega_D} \to \mathbb{R}^+$, $d(\varphi) = D$.

Combination The combination of two set potentials φ_1 and φ_2 on $D_1, D_2 \subseteq V$ respectively is a simplified version Dempster's rule of combination[Dempster, 1968, Jøsang and Pope, 2012]:

$$m_1 \otimes m_2(A) := \sum_{B \bowtie C = A, B \subseteq \Omega_{D_1}, C \subseteq \Omega_{D_2}} m_1(B) \cdot m_2(C) \quad \text{for all } A \subseteq \Omega_D,\ D = D_1 \cup D_2 \tag{2.17}$$

Projection Projection of the set potential φ from D to $D' \subseteq D$ is defined by

$$\varphi^{\downarrow D'}(A) := \sum_{\pi_{D'}(B) = A, B \subseteq \Omega_D} \varphi(B) \quad \text{for all } A \subseteq \Omega_{D'} \tag{2.18}$$

► **Example 2.5.** Consider two variables $\{a, b\}$ with binary frames $\Omega_a = \Omega_b = \{0, 1\}$. Now consider two set potentials φ_1, φ_2 with domains $\{a, b\}, \{a\}$ respectively:

focal set	φ_1
$\{(0,0)\}$	0.7
$\{(1,0),(0,1)\}$	0.1
$\{(0,0),(1,1)\}$	0.2

focal set	φ_2
$\{(1)\}$	0.6
$\{(0)\}$	0.4

We construct the following table as an intermediate step.

The first column contains φ_1 and the top row contains φ_2. Both of them have been extended to the union domain $d(\varphi_1) \cup d(\varphi_2) = \{a, b\}$. Then the intersection between the

corresponding tuple sets with the product is in the internal cells.

	$\{(1,1),(1,0)\},0.6$	$\{(0,1),(0,0)\},0.4$
$\{(0,0)\},0.7$	$\emptyset,0.42$	$\{(0,0)\},0.28$
$\{(1,0),(0,1)\},0.1$	$\{(1,0)\},0.06$	$\{(0,1)\},0.04$
$\{(0,0),(1,1)\},0.2$	$\{(1,1)\},0.12$	$\{(0,0)\},0.08$

To complete the combination, we then add the values of all internal cells with equal tuple set. The result is projected afterwards.

focal set	$\varphi_1 \otimes \varphi_2$
$\emptyset$	0.42
$\{(0,0)\}$	0.36
$\{(0,1)\}$	0.04
$\{(1,0)\}$	0.06
$\{(1,1)\}$	0.12

focal set	$(\varphi_1 \otimes \varphi_2)^{\downarrow\{a\}}$
$\emptyset$	0.42
$\{(1)\}$	0.18
$\{(0)\}$	0.40

► **Theorem 2.2.** *[Pouly and Kohlas, 2011, theorem 5.5] The structure $\langle \Phi, V, d, \otimes, \downarrow \rangle$ defined in def. 2.3 is a valuation algebra.*

Proof. To show that $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a valuation algebra, we have to show that *A1 – A6* hold.

A1 Commutative semigroup: We have to show that Φ is associative and commutative under $\otimes$. Commutativity follows from the commutativity of semiring multiplication and natural join. To show associativity, we assume $\varphi \in \Phi_S, \psi \in \Phi_T, \nu \in \Phi_U$ and $A \subseteq \Omega_{S \cup T \cup U}$

$$\begin{aligned}(\varphi \otimes (\psi \otimes \nu))(A) &= \sum_{B \bowtie E = A} \varphi(B) \cdot (\psi \otimes \nu)(E) \\ &= \sum_{B \bowtie E = A} \varphi(B) \cdot \sum_{C \bowtie D = E} \psi(C) \cdot \nu(D) \\ &= \sum_{B \bowtie C \bowtie D = A} \varphi(B) \cdot \psi(C) \cdot \nu(D)\end{aligned}$$

We get the same result, if we start with $((\varphi \otimes \psi) \otimes \nu)(A)$, proving associativity.

A2 Labeling: This follows directly from the definition of combination: $d(\phi \otimes \psi) = d(\phi) \cup d(\psi)$

A3 Projection: This follows directly from the definition of projection.

A4 Transitivity: For $\varphi \in \Phi$ with $S \subseteq T \subseteq d(\varphi)$, we have

$$\begin{aligned}(\varphi^{\downarrow T})^{\downarrow S}(A) &= \sum_{\pi_S(B)=A} \varphi^{\downarrow T}(B) = \sum_{\pi_S(B)=A} \sum_{\pi_T(C)=B} \varphi(C) \\ &= \sum_{\pi_S(\pi_T(C))=A} \varphi(C) = \sum_{\pi_S(C)=A} \varphi(C) = \varphi^{\downarrow S}(A) \quad \text{using transitivity of } \pi\end{aligned}$$

A5 Combination: Suppose that $\varphi \in \Phi_S, \psi \in \Phi_T$ and $A \subseteq \Omega_Z$, where $S \subseteq Z \subseteq S \cup T$. Using the combination property of relational algebras, we have

$$\begin{aligned}(\varphi \otimes \psi)^{\downarrow Z}(A) &= \sum_{\pi_Z(B)=A} (\varphi \otimes \psi)(B) = \sum_{\pi_Z(B)=A} \sum_{C \bowtie D=B} \varphi(C) \cdot \psi(D) \\ &= \sum_{\pi_Z(C \bowtie D)=A} \varphi(C) \cdot \psi(D) = \sum_{C \bowtie \pi_{T \cap Z}(D)=A} \varphi(C) \cdot \psi(D) \\ &= \sum_{C \bowtie E=A} \varphi(C) \cdot \sum_{\pi_{T \cap Z}(D)=E} \psi(D) \\ &= \sum_{C \bowtie E=A} \varphi(C) \cdot \psi^{\downarrow T \cap Z}(E) = \varphi \otimes \psi^{\downarrow T \cap Z}(A)\end{aligned}$$

A6 Domain: For $\varphi \in \Phi$ and $X = d(\varphi)$, we have

$$\varphi^{\downarrow X}(A) = \sum_{B=A} \varphi(B) = \varphi(A)$$

This shows that the valuation algebra of set potentials follows the axioms and is indeed a valuation algebra. □

2.4 Inference Problem

The previous section introduced the theoretical framework of valuation algebras and the axioms of the framework followed by some instances of valuation algebras. Now that we have an abstract generic framework for representing information, we can perform inference. Inference in this case refers to the aggregation of the various pieces of information in the knowledgebase, followed by focusing/projection on the relevant domain of interest. This computational problem is called the *inference problem* for valuation algebras and is of interest to the fields of statistics and machine learning as well as logical inference, constraint satisfaction problems, theorem proving and formal verification. We start with a formal definition of inference problems and show how it is expressed in some of the valuation algebra instances described earlier. In section 2.7, we discuss the algorithms for efficient computation

of the inference problem.

▶ **Definition 2.4.** INFERENCE PROBLEM. [Pouly and Kohlas, 2011, def. 2.1] The *inference problem* is the task of computing

$$(\phi_1 \otimes \cdots \otimes \phi_n)^{\downarrow X_i},\ i \in \{1 \dots k\} \tag{2.19}$$

for a given *knowledgebase* or collection of valuations $\{\phi_1, \dots, \phi_n\} \subseteq \Phi$ and domains $X = \{X_1, \dots, X_k\}$ where $X_i \subseteq d(\phi_1 \otimes \cdots \otimes \phi_n)$. The domains X_i are called *queries*.

In this thesis, we shall restrict ourselves to the simplest case, that of the single query inference problem: $|X| = 1$.

2.5 Inference in semiring induced valuation algebras

2.5.1 Arithmetic potentials

The valuation algebra of arithmetic potentials can express Bayesian networks. In this section we re-cast the earlier example from example 1.4 in the introduction in the language of valuation algebras.

As we recall, the joint probability distribution for a Bayesian network can be factored into its constituent conditional probability distributions. Thus the *knowledgebase* here is the set of conditional probability distributions which are variables conditioned on their parents in the directed acyclic graph representation. We also wish to find out the probability for a particular subset of variables, thus these subsets are the *queries*.

▶ **Example 2.6.** In the specific case of example 1.4, if we want to find out P(alarm), the knowledgebase is the set of local kernels

$$\begin{aligned} F = \{&\phi_1 : \{\text{burglary}\}, \phi_2 : \{\text{earthquake}\}, \phi_3 : \{\text{alarm}, \text{burglary}, \text{earthquake}\}, \\ &\phi_4 : \{\text{radio-report}, \text{earthquake}\}, \phi_5 : \{\text{watson-call}, \text{alarm}\}\} \end{aligned} \tag{2.20}$$

where the domains of the valuations ϕ_i are indicated next to them, and the query set is $K = \{\text{alarm}\}$.

This instantiates the inference problem definition. We will consider Bayesian networks in more detail in chapter 4.

2.5.2 Disjunctive normal forms

We consider the following example of a simple inference problem in the valuation algebra of disjunctive normal forms which illustrates logical inference in the formalism of the inference problem definition in (def. 2.4).

► **Example 2.7.** Consider the set of variables $V = \{a, b, c\}$ and the formulae $\xi_1 = a \to b$, $\xi_2 = b \to c$. Using standard inference rules, we get $a \to c$.

We can also obtain this inference via the methods of combination and marginalisation as explained in section 2.2.2. We can see the equivalence between the full disjunctive normal form and the model by combining the DNFs and projecting: combining ξ_1 and ξ_2 gives us

$$\begin{aligned} \xi_1 \otimes \xi_2 &= (\neg a \vee b) \wedge (\neg b \vee c) \\ &= (\neg a \wedge \neg b) \vee (\neg a \wedge c) \vee (b \wedge \neg b) \vee (b \wedge c) \\ &= (\neg a \wedge \neg b) \vee (\neg a \wedge c) \vee (b \wedge c) \end{aligned}$$

If we marginalise out b, then we get

$$(\xi_1 \otimes \xi_2)^{\downarrow\{a,c\}} = \neg a \vee (\neg a \wedge c) \vee c = \neg a \vee c \tag{2.21}$$

which is the same as $a \to c$.

Alternatively we can consider the model equivalent in table 2.3, which gives the same result.

2.5.3 Relational algebras

Query answering in relational databases is another instance of the inference problem. The knowledgebase is a set of database tables which belong to the valuation algebra as described in section 2.2.3. The inference problem computes the natural join of all the elements in the knowledgebase and projects the answer to the attributes of interest. However the *selection operator*, an important part of relational databases, does not have a corresponding operation in the valuation algebra. One way of achieving an integration of selections into the valuation algebra framework is to transform them to relations which can then be added to the knowledgebase. However transforming selections into relations can cause production of relations with infinite cardinality if queries over attributes with infinite frames are involved. In such

cases, we can perform selection enabled local computation [Schneuwly, 2007]. The following example is taken from [Pouly and Kohlas, 2011], which transforms selections into relations that augment the knowledgebase.

► **Example 2.8.** [Pouly and Kohlas, 2011, section 2.2] We consider four tables (relations) with information about students, their grades, courses and their professors:

SID	Student
1	Ann
2	John
3	Laura

SID	Grade	LID
1	C	901
2	A	901
2	B	903

LID	Lecture	Semester
901	Chemistry	S1898
902	Mathematics	S1801
903	Physics	A1905

PID	LID
1	903
2	902
3	901

PID	Professor
1	Einstein
2	Gauss
3	Curie

Thus we have a knowledgebase of five relations $\{r_1 \ldots, r_5\}$. If we have the database query

Find all students with grade A or B in a lecture of professor Einstein

then we have a single query for the inference problem: {Student}. We augment the knowledgebase with the selections, s_1 (Grade) $= \{A, B\}$ and s_2 (Professor) = {Einstein}. Then we can formulate the inference problem as

$$(r_1 \otimes r_2 \otimes r_3 \otimes r_4 \otimes r_5 \otimes s_1 \otimes s_2)^{\downarrow\{\text{Student}\}} \tag{2.22}$$

2.6 Inference in set potentials

For set potentials, we consider the following example given by Shafer [Shafer, 1982]. We neglect normalisation and thus can use set potentials instead of belief potentials.

► **Example 2.9.** Consider a disorder called *ploxoma* which has two variants, θ_1, called *virulent ploxoma*, which is fatal, and θ_2 called *ordinary ploxoma* which varies in severity and is treatable. Virulent ploxoma can be identified without doubt

at the time of a victim's death, but to distinguish between the two variants in the early stages one has to perform a blood test with possible outcomes x_1, x_2, x_3. The following is known:

1. Blood tests of a large number of patients dying of virulent ploxoma showed the outcomes x_1, x_2 and x_3 occurring 20, 20 and 60 percent of the time respectively.
2. A study of patients with prolonged ploxoma (and thus almost certainly ordinary ploxoma) showed outcome x_1 occurring 85 percent of the time and the outcomes x_2, x_3 occurring 15 percent of the time (this was before methods were perfected to distinguish between x_2 and x_3). There is some doubt as to whether the patients in the study are a fair sample of ordinary ploxoma victims, but experts are 75 percent sure that the criteria for patient selection would not affect the distribution of test outcomes.
3. Most people seeking medical help for ploxoma are seeking it for ordinary ploxoma. While there have been no careful statistical studies, physicians are convinced that only 5–15 percent of ploxoma patients suffer from virulent ploxoma.

This introduces the variable d for the disease, with values $\{\theta_1, \theta_2\}$ and t for the test result with values $\{x_1, x_2, x_3\}$. There is further discussion in [Shafer, 1982] about how to translate the evidence into the knowledgebase. Here we proceed directly to enumerating the valuations in the knowledgebase; and focus on formulating the reasoning with set potentials as an inference problem.

From the first statement, we get the following mass function m_1:

$$\begin{array}{ll} \{(\theta_1, x_1), (\theta_2, x_1), (\theta_2, x_2), (\theta_2, x_3)\} & 0.2 \\ \{(\theta_1, x_2), (\theta_2, x_1), (\theta_2, x_2), (\theta_2, x_3)\} & 0.2 \\ \{(\theta_1, x_3), (\theta_2, x_1), (\theta_2, x_2), (\theta_2, x_3)\} & 0.6 \end{array}$$

Thus we have a belief of 0.6 for the proposition that the blood test returns x_3 for a patient with virulent ploxoma. This mass function does not give any evidence about the test outcome with disease θ_2. The second factor m_2 is

$$\begin{array}{rr} \{(\theta_2, x_1), (\theta_1, x_1), (\theta_1, x_2), (\theta_1, x_3)\} & 0.85 \cdot 0.75 \\ \{(\theta_2, x_2), (\theta_2, x_3), (\theta_1, x_1), (\theta_1, x_2), (\theta_1, x_3)\} & 0.15 \cdot 0.75 \\ \{(\theta_2, x_1), (\theta_2, x_2), (\theta_2, x_3), (\theta_1, x_1), (\theta_1, x_2), (\theta_1, x_3)\} & 0.25 \end{array}$$

The second mass function is telling us that in ordinary ploxoma θ_2, the test outcome is x_1 in 85 percent of the cases, which we scale by 0.75 to reflect the confidence in the outcome by experts. There are 25 percent cases which give us no new information. The last mass function m_3 is

$$\begin{array}{cc} \{(\theta_1)\} & 0.05 \\ \{(\theta_2)\} & 0.85 \\ \{(\theta_1),(\theta_2)\} & 0.10 \end{array}$$

This signifies that 85 percent of the time, the patient is suffering from ordinary ploxoma. The ambiguity in the remaining percentage is represented in the set potential by assigning 5 percent to virulent ploxoma and 10 percent possibility to both cases.

Now suppose we want to calculate the belief held for θ_1 and θ_2 given that the test result is, say x_1. Similarly to Bayesian networks, we add an observation set potential expressing that the test result is x_1: $m_o : \{\{(x_1)\}, 1.0\}$. Then we can compute the belief for presence of virulent or ordinary ploxoma by solving

$$(m_1 \otimes m_2 \otimes m_3 \otimes m_0)^{\downarrow\{d\}} \tag{2.23}$$

We shall look at the application of our anytime inference framework to this example in section 4.2.

2.7 The Inference Algorithm

2.7.1 Fusion, Variable Elimination and Join Trees

In this section, we shall discuss the algorithm which solves the inference problem stated in section 2.4. The inference algorithm described here is one among others, such as the *fusion* algorithm [Shenoy, 1992] and *bucket elimination* algorithm [Dechter, 1998], which are referred to in the literature as *local computation* schemes. These local computation schemes reduce the computational complexity considerably from the naïve approach of combining all the valuations and then projecting to the desired query.

We discuss the fusion algorithm which uses variable elimination instead of marginalization. This technique is generalised in a message-passing algorithm. We use the algorithms and notation from [Haenni, 2004].

Fusion algorithm

The fusion algorithm was first described by Shenoy [Shenoy, 1992, Shenoy, 1997]. Later, the essential idea was rediscovered by Dechter in the *bucket elimination* framework [Dechter, 1998]. Let $\Psi = \{\phi_1, \ldots, \phi_r\} \subseteq \Phi$ be a given set of valuations and $D \subseteq V$ be the domain of interest where $V \supseteq d(\phi_1) \cup \cdots \cup d(\phi_n)$. The fundamental operation of the fusion algorithm is the elimination of a single variable $x \in \Delta$ where $\Delta = V \setminus D$ is the set of variables to be eliminated. We introduce the following notation:

$$\Psi_x := \{\phi \in \Psi : x \in d(\phi)\} \quad \text{and} \quad \Psi_x^* := \{\phi \in \Psi : x \notin d(\phi)\} \tag{2.24}$$

where Ψ_x and Ψ_x^* denote the corresponding subsets of valuations which have the variable x and its complement. In Dechter's framework, Ψ_x is called the bucket of x [Dechter, 1998]. Only valuations in Ψ_x are changed by the elimination of x. The remaining set of valuations after eliminating x from Ψ is

$$\mathrm{Fus}_x(\Psi) := \{(\otimes\Psi_x)^{-x}\} \cup \Psi_x^* \tag{2.25}$$

Thus the marginal of the joint valuation can be computed by successively eliminating the variables in $\Delta = \{x_1, \ldots, x_s\}$. If $\langle x_1, \ldots, x_s \rangle$ is an arbitrary sequence in which variables are eliminated, then

$$\mathrm{Fus}_{\{x_1,\ldots,x_s\}}(\Psi) := \mathrm{Fus}_{x_s} \odot \ldots \odot \mathrm{Fus}_{x_1}(\Psi) \tag{2.26}$$

is the set of remaining valuations after eliminating all the variables in Δ. The entire process is called the *fusion algorithm* or *bucket elimination*. Finally:

$$(\otimes\Psi)^{\downarrow D} = (\otimes\Psi)^{-\Delta} = \odot\mathrm{Fus}_\Delta(\Psi) \tag{2.27}$$

The efficiency of the algorithm is strongly dependent on the elimination order; several heuristic methods have been developed to find good elimination sequences [Almond, 1995, Almond and Kong, 1991, Cano and Moral, 1995, Haenni and Lehmann, 1999].

Binary join trees

The fusion algorithm is a simple algorithm for solving the inference problem. However it is not a good solution in the

case when multiple projections are required from the joint valuation. It can be shown that fusion corresponds to inward propagation [Shenoy, 1997, sections 4,5] of valuations (with marginalisation) from the leaves to the root of a binary tree with nodes as valuations. Inward propagation gives us the ability to reuse the computations of the inward phase when several marginals are requested.

▶ **Definition 2.5.** JOIN TREE. Also known as *junction trees*, *clique trees*, *hypertrees*, *cluster trees* and *bucket trees*, a join tree is a *tree* $G = (N, E)$ of nodes N and edges E, with a label called a *domain*, $\lambda : N \to 2^V$ associated with each node and satisfying the *running intersection property*. V is the universe of variables.

Running intersection property. For a tree $G = (N, E)$ with nodes $\{i, j, k, \ldots\} \in N$, there is an unique path $P_G(i, j)$ which is represented as the set of nodes on the path from i to j in G. Then the running intersection property states that

$$\lambda(i) \cup \lambda(j) \subseteq \lambda(k), k \in P_G(i, j) \tag{2.28}$$

This implies, that if a variable is in the domain of two distinct nodes, then it is in the domain of each node on the unique path between these two nodes.

▶ **Definition 2.6.** ROOTED JOIN TREE. A rooted join tree is a join tree $G = (N, E)$ with directed edges; the unique root of the tree is denoted by $root(N)$.

▶ **Definition 2.7.** BINARY JOIN TREE. A *binary join tree* is a *rooted join tree* in which every node has an outgoing edge (to the parent) and either zero (if it is a leaf), or two incoming edges.

▶ **Notation.** We use the following notation for denoting elements of the binary join tree (abbreviated as BJT):

- $L(n)$ denotes the left child of a node n, or *nil* if n is a leaf.
- $R(n)$ denotes the right child of a node n, or *nil* if n is a leaf.
- $P(n)$ denotes the parent of node n or *nil* if $n = root(N)$.
- $S(n)$ denotes the sibling of node n or *nil* if $n = root(N)$. Thus $S(n) = R(P(n))$ if $n = L(P(n))$, or $S(n) = L(P(n))$ otherwise.

Next we consider how to construct a BJT for a given set $\Psi = \{\phi_1, \ldots, \phi_r\} \subseteq \Phi$ of valuations and a domain $D \subseteq V$

of interest. Here Ψ is the knowledgebase and D is the query. The process of constructing the join tree is similar to that of the fusion algorithm, in which variables are eliminated one after the other. Let $N_\Psi = \{n_1, \ldots, n_r\}$ be the initial set of nodes n_i with $d(n_i) = d(\phi_i)$, $L(n_i) = nil$, $R(n_i) = nil$ and $P(n_i) = nil$. The set of variables to be eliminated is $\Delta = V \setminus D = \{x_1, \ldots, x_s\}$. For s variables to be eliminated, the binary join tree can be constructed in $s + 1$ steps. At each step a set of nodes N_x is selected. During the first s steps, N_x comprises the nodes that contain the current variable x. At each of the $s + 1$ steps, pairs of nodes are selected from N_x and linked to a new node until only one node remains. If Δ^c is used to keep track of the eliminated variables, then the complete procedure goes as follows:

---

function CONSTRUCT-BINARY-JOIN-TREE(N_Ψ, Δ)
  $N \leftarrow \emptyset; \Delta^c \leftarrow \emptyset; root(N) = nil$
  repeat
    if $\Delta = \emptyset$ **then**
      $N_x = N_\Psi$
    else
      select $x \in \Delta$ using some heuristic
      $N_x \leftarrow \{n \in N_\Psi : x \in d(n)\}$;
    end if
    while $|N_x| > 1$ **do**
      generate new node n with $P(n) = nil$;
      select distinct $n_1, n_2 \in N_x$ using some heuristic.
      $P(n_1) \leftarrow n; P(n_2) \leftarrow n$;
      $L(n) \leftarrow n_1; R(n) \leftarrow n_2$;
      $d(n) \leftarrow (d(n_1) \cup d(n_2)) \setminus \Delta^c$;
      $N_x \leftarrow (N_x \setminus \{n_1, n_2\}) \cup \{n\}$;
      $N \leftarrow N \cup \{n_1, n_2\}$
    end while
    select n from $N_x = \{n\}$;
    if $\Delta = \emptyset$ **then**
      $root(N) = n$;
    else
      $\Delta \leftarrow \Delta \setminus \{x\}; \Delta^c \leftarrow \Delta^c \cup \{x\}$
      $N_\Psi \leftarrow \{n \in N_\Psi : x \notin d(n)\} \cup d(n)$
    end if

until $root(N) \neq nil$
$N \leftarrow N \cup \{n\}$; **return** N;
end function

The return value $N = \{n_1, \ldots, n_{2r-1}\}$ is a BJT where the domains of the leaves correspond to the domains of the valuations in Ψ such that $D \subseteq d(root(N))$. There is a strong correspondence between this procedure for constructing join trees and the fusion algorithm discussed earlier. If we implemented the combination procedure in fusion as a tree structure, we would produce a join tree which would not be binary. Here we restrict combination to two valuations at a time to form the binary join tree.

▶ **Example 2.10.** [Haenni, 2004, Example 2] Let $\Psi = \{\phi_1, \ldots, \phi_n\}$ be the initial set of valuations assigned to corresponding nodes in $N_\Psi = \{n_1, \ldots, n_9\}$ with domains

$$d(n_1) = \{a, x_1\}, \quad d(n_2) = \{a, c, x_1\}, \quad d(n_3) = \{b, x_1, x_2\},$$
$$d(n_4) = \{b, c, x_2\}, \quad d(n_5) = \{c, x_2\}, \quad d(n_6) = \{a, x_3, x_4\},$$
$$d(n_7) = \{d, x_3, x_4\}, \quad d(n_8) = \{c, x_5\}, \quad d(n_9) = \{a, b\}.$$

The query is $D = \{a, b, c, d\}$. Thus the set of variables to be eliminated is $\Delta = \{x_1, \ldots, x_5\}$. We eliminate the variables in increasing order of index, that is x_i is eliminated before x_j iff $i < j$. Similarly, when we are choosing the nodes to be combined from N_x, we choose the nodes with the smallest indices. The table below shows a trace of an invocation of the algorithm:

Table 2.6: Trace of binary join tree construction for example 2.10.

Δ	N_Ψ	N_x	n	$d(n)$
$\{\underline{x_1}, \ldots, x_5\}$	$\{n_1, \ldots, n_9\}$	$\{\underline{n_1}, \underline{n_2}, n_3\}$	n_{10}	$\{a, c, x_1\}$
		$\{\underline{n_3}, \underline{n_{10}}\}$	n_{11}	$\{a, b, c, x_1, x_2\}$
$\{\underline{x_2}, \ldots, x_5\}$	$\{n_4, \ldots, n_9, n_{11}\}$	$\{\underline{n_4}, \underline{n_5}, n_{11}\}$	n_{12}	$\{b, c, x_2\}$
		$\{\underline{n_{11}}, \underline{n_{12}}\}$	n_{13}	$\{a, b, c, x_2\}$
$\{\underline{x_3}, x_4, x_5\}$	$\{n_6, \ldots, n_9, n_{13}\}$	$\{\underline{n_6}, \underline{n_7}\}$	n_{14}	$\{a, d, x_3, x_4\}$
$\{\underline{x_4}, x_5\}$	$\{n_8, n_9, n_{13}, n_{14}\}$	$\{n_{14}\}$	n_{14}	$\{a, d, x_3, x_4\}$
$\{\underline{x_5}\}$	$\{n_8, n_9, n_{13}, n_{14}\}$	$\{n_8\}$	n_8	$\{c, x_5\}$
$\{\}$	$\{n_8, n_9, n_{13}, n_{14}\}$	$\{\underline{n_8}, \underline{n_9}, n_{13}, n_{14}\}$	n_{15}	$\{a, b, c\}$
		$\{\underline{n_{13}}, \underline{n_{14}}, n_{15}\}$	n_{16}	$\{a, b, c, d\}$
		$\{\underline{n_{15}}, \underline{n_{16}}\}$	n_{17}	$\{a, b, c, d\}$

The execution trace of the algorithm is similar to that of the fusion algorithm. x_1 is the first variable to be eliminated; it

appears in the domains of nodes $N_x = \{n_1, n_2, n_3\}$; two nodes n_{10}, n_{11} are generated to connect the nodes in the binary join tree. If we had not restricted ourselves to binary join trees, then there would be one parent node connecting each of n_1, n_2, n_3; instead we first connect n_1, n_2 to the parent node n_{10} and then connect n_3 to n_{10} to form n_{11}. Thus we get $d(n_{10}) = \{a, c, x_1\}$ and $d(n_{11}) = \{a, b, c, x_1, x_2\}$. Similarly we proceed with the elimination of $x_2, \ldots, x_5$. After the elimination, four nodes remain $N_\Psi = \{n_8, n_9, n_{13}, n_{14}\}$. We need three more nodes n_{15}, n_{16}, n_{17} to connect them. The result is a binary join tree with nodes $N = \{n_1, \ldots, n_{17}\}$ and $root(N) = n_{17}$. The query $D = \{a, b, c, d\}$ is the domain of $d(root(N))$. The constructed binary join tree is shown in figure 2.1.

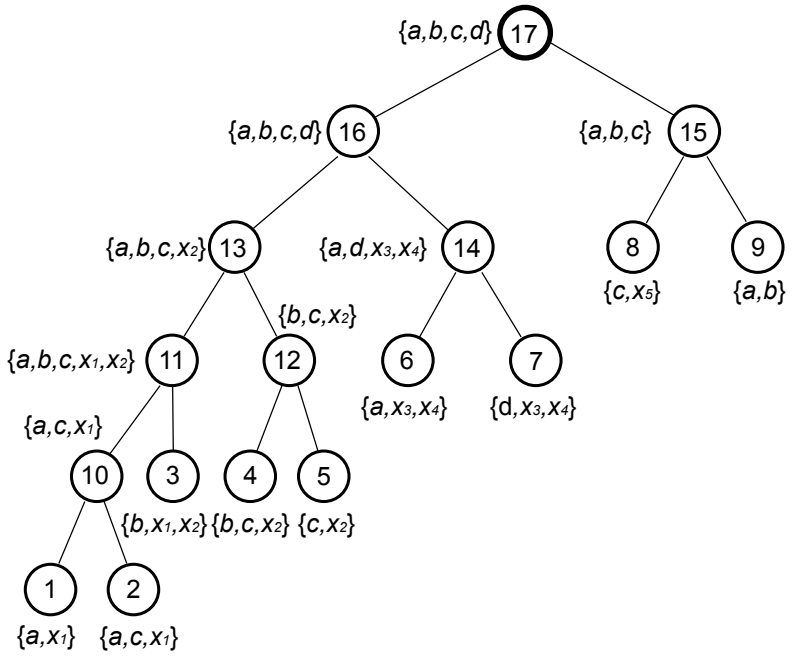


Figure 2.1: The binary join tree from example 2.10.

► **Remark.** In this and the following chapters 3 through 5, we do not concern ourselves with whether the binary junction trees are balanced. The inference algorithm presented in this chapter, and the anytime inference algorithm presented in chapter 3 will still be correct regardless.

We give references to methods that can be used to construct balanced binary join trees in chapter 6 which requires balanced binary junction trees for some results.

2.7.2 Inward propagation

The first phase of the algorithm is inward propagation of messages sent from the leaves of the BJT towards the root.

▶ **Notation.** Let $\Psi = \{\phi_1, \ldots, \phi_r\}$ be a set of valuations, D be the domain of interest and $N = \{n_1, \ldots, n_{2r-1}\}$ be the nodes of the constructed BJT.

We denote the r leaves of the BJT by $leaves(N)$. Then for each valuation $\phi \in \Psi$, we select the corresponding leaf $n \in leaves(N)$ with $d(n) = d(\phi)$ and assign ϕ to n; the valuation at node n is denoted by $\phi(n)$. The message sent from a leaf node n to the parent node $P(n)$ is denoted by $\phi_s(n)$.

We also denote the *queue* of nodes to be processed by $next(N)$ which denotes the subset of nodes that have received messages from both its children and is thus ready to compute its own valuation.

$$next(N) := \{n \in N : \phi_s(n) = nil, \phi_s(L(n)) \neq nil, \phi_s(R(n)) \neq nil\} \tag{2.29}$$

For all nodes which are not leaves, we set $\phi(n) = nil$. The content of the message is $\phi(n)$ marginalised to the common variables between the node and its parent $d(n) \cap d(P(n))$. Thus for any n which is not the root, $\phi_s(n) = \phi(n)^{-\Delta(n)}$ where $\Delta(n) = d(n) \setminus d(P(n))$. Here the notation used for $\phi(n)^{-\Delta(n)}$ is the same as in section 2.7.1, where φ^{-X} for $X \subseteq V$ denotes elimination of variables in X from φ. Thus $\phi(n)^{-\Delta(n)} = \phi(n)^{d(n) \cap d(P(n))}$.

After the parent node n has received messages from both its child nodes $L(n), R(n)$, its own valuation $\phi(n)$ is calculated by combining the messages:

$$\phi(n) = \phi_s(L(n)) \otimes \phi_s(R(n)) \tag{2.30}$$

Initially we set $\phi_s(n) = nil$ for all $n \in N$. We also define $\Delta(root(N)) := d(root(N)) \setminus D$ as the set of variables to be eliminated at the end. Then the inward propagation algorithm can be written as

---

function INWARD(N)
  for all $n \in leaves(N)$ do $\phi_s(n) \leftarrow \phi(n)^{-\Delta(n)}$
  while $next(N) \neq \emptyset$ **do**
    select n from $next(N)$

$\phi(n) \leftarrow \phi_s(L(n)) \otimes \phi_s(R(n))$
$\phi_s(n) \leftarrow \phi(n)^{-\Delta(n)}$
end while
return $\phi_s(root(N))$
end function

At the end of the inward propagation procedure, we get the marginal $(\otimes\Phi)^{\downarrow D}$ at the root node, $root(N)$.

▶ **Remark.** *Time complexity of inward propagation.* From the inward propagation algorithm we can see that there will be $n-1$ combinations which is the more time-consuming operation generally compared to projection. Thus the time complexity of inward propagation is $O(nk)$ where $O(k)$ is the maximum time taken for combination at a node. This is usually a function of the size of the join tree label (the number of variables assigned). It follows that the largest join tree label will determine the complexity:

▶ **Definition 2.8.** TREEWIDTH. The treewidth of a join tree $G = (N, E)$ with labels $\lambda : N \to 2^V$ where $\lambda(n)$ denotes the label of n, is the size of the largest join tree label:

$$\omega(N) = \max\{|\lambda(n)| : n \in N\}$$

The treewidth is an important parameter that determines complexity. The goals of most heuristics in this research area is to reduce the join tree width, and thus the time complexity.

2.7.3 Outward propagation

The outward propagation calculates the marginals for each node in the join tree. We reuse the intermediate results (messages passed from nodes to parents, and the valuation at each node) to construct the marginals at each node.

▶ **Notation.** We denote $\phi_r(n)$ to be the message that a parent node $P(n)$ sends to its child node n. The *marginal* at n, which is the solution to the inference problem with query $d(n)$, is denoted by $\phi'(n) = (\otimes\Phi)^{\downarrow d(n)}$. The *sibling* (other child of the parent node) of n is denoted by $S(n)$. The *queue* of nodes to be processed as in the inward part of the algorithm is then

$$next'(N) := \{n \in N : \phi'(n) = nil, \phi'(P(n)) \neq nil\} \qquad (2.31)$$

The above is the set of nodes that are ready to receive a message from the parent node $P(n)$.

$\phi_r(n)$ carries all the information about non-descendants of n. The marginal $\phi'(n) = (\otimes\Phi)^{\downarrow d(n)}$ at node n is obtained using $\phi'(n) = \phi(n) \otimes \phi_r(n)$, where $\phi(n) = \phi_s(L(n)) \otimes \phi_s(R(n))$ is the valuation computed at node n in the inward phase.

The message $\phi_r(n)$ is determined by $\phi_r(P(n))$ and $\phi_s(S(n))$. If we denote $\Delta'(n) = d(P(n)) \setminus d(n)$ to denote a set of variables in $d(P(n))$ but not in $d(n)$, then

$$\phi_r(n) = (\phi_r(P(n)) \otimes \phi_s(S(n)))^{-\Delta'(n)} \tag{2.32}$$

For the root node, $\phi_r(root(N)) = e_D$ and $\phi'(root(N) = \phi(root(N))$. Also, Then the outward phase can be described as follows:

---

function OUTWARD(N)
  $\phi_r(root(N)) \leftarrow e_D; \phi'(root(N)) \leftarrow \phi(root(N))$
  while $next'(N) \neq \emptyset$ **do**
    select n from $next'(N)$
    $\phi_r(n) \leftarrow (\phi_r(P(n)) \otimes \phi_s(S(n)))^{-\Delta'(n)}$
    $\phi'(n) \leftarrow \phi(n) \otimes \phi_r(n)$
  end while
  return $\{\phi'(n) : n \in N\}$
end function

OUTWARD(N) returns the marginal $\phi'(n) = (\otimes\Phi)^{\downarrow d(n)}$ for each $n \in N$. Nodes with identical domains get identical results. The algorithm stops after $2 \cdot (r-1)$ steps, requiring $4 \cdot (r-1)$ combinations.

For this we need an identity valuation for each domain e_D for each domain $D \subseteq V$. This is naturally present in some instances, such as valuation algebras introduced by a boolean semiring: $e_D(\mathbf{x}) = 1$ for all $\mathbf{x} \in \Omega_D$. Otherwise, we need to use architectures such as the outward phase in the Shenoy-Shafer architecture [Pouly and Kohlas, 2011, section 4.1.1]. We shall not be requiring outward propagation in the thesis, so this matter is not elaborated further.

2.7.4 Local Computation Architectures

In this section we briefly discuss some of the local computation architectures prevalent in the literature. All the architectures utilise a covering join tree for the valuations in the knowledge base. A covering join tree for an inference problem is a join tree that has at least one node n (with label $\lambda(n)$) for every valuation ϕ_i such that $d(\phi_i) \subseteq \lambda(n)$. The local computation architectures differ in the implementation of inward and outward (collect and distribute) phases of the inference algorithm.

Shenoy-Shafer In the Shenoy-Shafer architecture, we store the messages sent between nodes for later re-use. The inward and outward propagation algorithms in this architecture are thus similar to those presented in the previous sections. Unlike the previous section which only considered binary join trees, the Shenoy-Shafer architecture is applicable to arbitrary join trees. However binary join trees generally perform better in the Shenoy-Shafer architecture [Shenoy, 1997]. A general introduction can also be found at [Pouly and Kohlas, 2011, section 4.1].

Lauritzen-Spiegelhalter This architecture was introduced to perform local computation in Bayesian networks [Lauritzen and Spiegelhalter, 1990]. This differs from the Shenoy-Shafer architecture in using a division operation during the inward propagation phase. Thus the Lauritzen-Spiegelhalter architecture is less general than the Shenoy-Shafer architecture as it only supports valuation algebras with division.

The advantage of the Lauritzen-Spiegelhalter architecture is that messages need not be stored. In the Shenoy-Shafer architecture it is necessary to store the inward propagation messages, as otherwise a particular node would get back the message it sent in the outward phase, counting the information twice. In Lauritzen-Spiegelhalter, the nodes can factor our the message from its own valuation after the message has been sent.

HUGIN The HUGIN architecture was introduced in [Jensen et al., 1990] as a modification of the Lauritzen-Spiegelhalter architecture. Here, the division is postponed to the outward propagation phase. The division takes place in a separate class of nodes called *separators* (s_{ij}) which lie between neighbouring

nodes i and j with a label $\lambda(s_{ij}) = \lambda(i) \cap \lambda(j)$. The advantage over Lauritzen-Spiegelhalter is that division is less costly due to the smaller label sizes of the separators.

2.8 Ordered Valuation Algebras

As exact inference is a #P-hard problem, we need frameworks for approximate inference. The framework reviewed in this section, and introduced by [Haenni, 2004], extends the generic inference framework to accommodate approximate inference.

Before proceeding to the definition, we need to consider two notions which are essential to any such approximate inference framework, an approximation relation and a method for construction of approximate valuations:

Approximation relation The first notion to consider when building a framework for approximate inference is to have a *relation for approximation*; we would need to know when a valuation is more approximate than the other; and ideally would also be able to quantify the degree of approximation. For this, we introduce a *completeness relation* $\succeq$. If ϕ, ϕ' are two valuations, then $\phi \succeq \phi'$ means that ϕ is *more complete* than ϕ'. Intuitively, the information contained in ϕ is more informative and a better approximation than the information encoded by ϕ'. Further, we assume a *partial order*:

- *Reflexivity*: $\phi \succeq \phi$ for all $\phi \in \Phi$.
- *Anti-symmetry*: $\phi \succeq \phi'$ and $\phi' \succeq \phi$ implies $\phi = \phi'$ for all $\phi, \phi' \in \Phi$.
- *Transitivity*: $\phi \succeq \phi'$ and $\phi' \succeq \phi''$ implies $\phi \succeq \phi''$ for all $\phi, \phi', \phi'' \in \Phi$.

Also it is reasonable to assume that comparison of valuations is only defined when the domains match. Thus $\phi \succeq \phi'$ implies $d(\phi) = d(\phi')$ for all $\phi, \phi' \in \Phi$ and $\succeq$ actually defines separate completeness relations $\succeq_D$ for each subsemigroup[6] Φ_D.

[6] defined at def. 1.4

Approximate valuation construction In addition to an approximation relation such as partial order, we also need a method or operation to construct approximate valuations.

This is accomplished in [Haenni, 2004] through the introduction of a time-bound combination operation which constructs approximate valuations. Combination is chosen to be the operation here since it is usually the more computationally intensive one, as compared to marginalisation. Variations of the inward and outward propagation algorithm are considered which restrict the time required for computation, guaranteeing computation of an approximate valuation within a specified time bound.

▶ **Remark.** Our treatment of ordered valuation algebras varies slightly from [Haenni, 2004], which does not include the time-bound combination operation in the ordered valuation algebra definition, instead considering it as an additional component. We have opted to include the time-bounded combination operation in the main definition for simplicity, along with its associated axioms, here *R5 – R9*, originally *R1 – R5* in [Haenni, 2004].

▶ **Definition 2.9.** Ordered Valuation Algebra. An ordered valuation algebra is a tuple $\langle \Phi, V, d, \otimes, \downarrow, \succeq, \otimes_t \rangle$ satisfying *R1–R9* where $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a valuation algebra. The additional components are the partial order $\succeq$: $\Phi \times \Phi$ and the time-bound combination operation $\otimes_t : \Phi \times \Phi \to \Phi$

R1 Partial order: There is a partial order $\succeq$ on Φ such that $\phi \succeq \phi'$ implies $d(\phi) = d(\phi')$ for all $\phi, \phi' \in \Phi$. Also the infimum $\inf(\Psi)$ exists for each subset $\Psi \subseteq \Phi_D$ and for all $D \subseteq V$.

R2 Null element: The least complete valuations for a particular domain D are the null valuations for that domain n_D. Thus $n_D \otimes \phi = \phi \otimes n_D = n_D$ for all $\phi \in \Phi_D$ and for all $D \subseteq V$. Since null elements for a particular domain are unique, $n_{D_1} \otimes n_{D_2} = n_{D_1 \cup D_2}$ for $D_1, D_2 \subseteq V$ and $n_D^{\downarrow D'} = n_{D'}$ for all $D' \subseteq D$.

R3 Combination preserves partial order: If $\phi_1, \phi_1', \phi_2, \phi_2' \in \Phi$ are valuations such that $\phi_1 \succeq \phi_1'$ and $\phi_2 \succeq \phi_2'$, then $\phi_1 \otimes \phi_2 \succeq \phi_1' \otimes \phi_2'$.

R4 Marginalisation preserves partial order: If $\phi, \phi' \in \Phi$ are valuations such that $\phi \succeq \phi'$, then $\phi^{\downarrow D} \succeq \phi'^{\downarrow D}$ for all $D \subseteq d(\phi) = d(\phi')$.

The time bound combination operation $\otimes_t : \Phi \times \Phi \to \Phi$ satisfies the following axioms, for all $\phi_1, \phi_2 \in \Phi$ and $t \in \mathbb{R}^+ \cup \{\infty\}$.

R5 Computation time: The effective time to compute $\phi_1 \otimes_t \phi_2$ is less than t units.

R6 Approximate valuation: The exact combination of two potentials $\phi_1 \otimes \phi_2$ is approximated by the time-bound combination; $\phi_1 \otimes \phi_2 \succeq \phi_1 \otimes_t \phi_2$.

R7 Monotone: The approximation gets better with time; $\phi_1 \otimes_{t'} \phi_2 \succeq \phi_1 \otimes_t \phi_2$ for all $t' \geq t$. In other words, operation $\otimes_t$ is *monotonic* with respect to t.

R8 Null valuation at $t = 0$: $\phi_1 \otimes_0 \phi_2 = n_D$ where n_D is the null element of the valuation algebra corresponding to the domain $D = d(\phi_1) \cup d(\phi_2)$.

R9 Complete: Given sufficient time, the time-bound combination operation returns the same result as the exact combination; $\phi_1 \otimes_\infty \phi_2 = \phi_1 \otimes \phi_2$

Using these axioms, we can arrive at the result that if a set of valuations $\Psi = \{\phi_1, \ldots, \phi_r\}$ is approximated by a corresponding set of less complete valuations $\Psi' = \{\phi'_1, \ldots, \phi'_r\}$, then $(\otimes\Psi')^{\downarrow D}$ is an approximation of the exact marginal of the joint valuation $(\otimes\Psi)^{\downarrow D}$.

The time-bound combination operator $\otimes_t$ is not commutative or associative as different sequences of combinations can lead to different outcomes. This is not surprising as the implementation of $\otimes_t$ contributes to the result; since the result is not exact, but approximate, it is natural to expect slightly different results depending on the exact sequence of combination. Since we are trying to find an approximate solution, it does not matter which one of the many possible approximate solutions we arrive at. The INWARD(N) algorithm is modified to incorporate the resource-bound combination operator $\otimes_t$. There is also a corresponding outward propagation algorithm which we do not discuss here.

Now we present some examples of ordered valuation algebras to give an intuitive sense of how the axioms work.

In the following two sections we extend the valuation algebra instances discussed previously. We do not provide proofs of correctness, however we shall show in chapter 3

that our framework of anytime ordered valuation algebras subsumes ordered valuation algebras, subject to an additional condition. We do discuss instances of ordered valuation algebras: arithmetic potentials, disjunctive normal forms and relational algebras, as well as set potentials. We shall conclude the discussion of ordered valuation algebras with set potentials which are not semiring induced.

2.9 Semiring induced ordered valuation algebras

We shall use the same definition of a semiring induced valuation algebra as in def. 2.2, with the additional constraint that the semiring be positive and partially ordered (defined below). This is similar to the treatment of approximations in [Haenni, 2004] and constitutes a generalization of the instances in that article. We shall use the notation for configuration space Ω_D of a set of variables D and the associated notation for tuples from definition 2.2 (semiring induced valuation algebras).

▶ **Definition 2.10.** PARTIALLY ORDERED SEMIRING. [Golan, 2013, chapter 20] A semiring $(A, +, \cdot, 0, 1)$ is partially-ordered if and only if there exists a partial order relation $\preceq_A$ on A satisfying the following conditions for elements r, r', and r'' of R:

(1) If $r \preceq_A r'$ then $r + r'' \preceq_A r' + r''$;

(2) If $r \preceq_A r'$ and $0 \preceq_A r''$ then $r \cdot r'' \preceq_A r' \cdot r''$ and $r'' \cdot r \preceq_A r'' \cdot r'$,

▶ **Definition 2.11.** POSITIVE SEMIRING. [Golan, 2013, chapter 20] A partially ordered semiring $(A, +, \cdot, 0, 1)$ is positive if and only if $0 \preceq r$ for all $r \in A$.

We derive two further properties that will be used in Chapter 3.

▶ **Lemma 2.3.** *For a partially ordered semiring $(A, +, \cdot, 0, 1)$, with the partial order denoted as $\preceq_A$, we have:*

(1) If $r \preceq_A r'$ and $s \preceq_A s'$ then $r + s \preceq_A r' + s'$;

(2) If $r \preceq_A r'$ and $s \preceq_A s'$ then $r \cdot s \preceq_A r' \cdot s'$

Proof. For (1), we note that $r + s \preceq_A r' + s$ using (1) of the partially ordered semiring definition. Similarly, we have

$r' + s \preceq_A r' + s'$. From transitivity of partial order, we get $r + s \preceq_A r' + s'$.

For (2), we note that $r \cdot s \preceq_A r' \cdot s$ using (2) of the partially ordered semiring definition. Similarly, we have $r' \cdot s \preceq_A r' \cdot s'$. From transitivity of partial order, we get $r \cdot s \preceq_A r' \cdot s'$. □

▶ **Remark.** We proceed to the definition of semiring induced ordered valuation algebras. As we shall not be using the ordered valuation algebra in the thesis, we do not show validity of the ordered valuation algebra axioms *R1 – R9*.

▶ **Definition 2.12.** SEMIRING INDUCED ORDERED VALUATION ALGEBRAS**.** A semiring induced ordered valuation algebra is a tuple $\langle \Phi, V, d, \otimes, \downarrow, \succeq, \otimes_t \rangle$, where $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a semiring induced valuation algebra (def. 2.2) induced by a positive, partially ordered, commutative semiring $(A, +, \cdot, 0, 1)$. The partial order in the semiring is denoted by order $\preceq_A$. We also have the partial order relation for the valuation algebra and the time-bound combination operation:

Partial order Valuations having the same domain D form a partially ordered set $\Phi_D \subseteq \Phi$. For two valuations $\phi, \phi' \in \Phi_D$, $\phi \preceq \phi'$ iff $\phi(\mathbf{x}) \preceq_A \phi'(\mathbf{x})$ for all $\mathbf{x} \in \Omega_D$.

Time bound combination operation The time bound combination operation is defined as follows: For $\phi_1 : \Omega_{D_1} \to A$ and $\phi_2 : \Omega_{D_2} \to A$, the time bound combination $\phi_1 \otimes_t \phi_2$ is a function $\phi_1 \otimes \phi_2 : \Omega_{D_1 \cup D_2} \to A$. Thus the time bound combination has the same type as the combination operation $\otimes$. It is implemented as the COMBINE(ϕ_1, ϕ_2, t) algorithm.

Null element The null element of a valuation algebra corresponding to a particular domain D is $n_D : \Omega_D \to A$, with $n_D(\mathbf{x}) = 0$ for all $\mathbf{x} \in \Omega_D$.

We can consider a semiring induced valuation ϕ with domain D, i.e. $\phi : \Omega_D \to A$ as being represented by a list $L_\phi = \langle (\mathbf{x}_1, \phi(\mathbf{x}_1)) \dots (\mathbf{x}_n, \phi(\mathbf{x}_n)) \rangle$, where $\{\mathbf{x}_1 \dots \mathbf{x}_n\} \subseteq \Omega_D$ is the subset that does not map to 0.

We then consider two valuations ϕ_1 on domain D_1 and ϕ_2 on domain D_2 with $L_{\phi_1} = \langle (\mathbf{x}_1, \phi_1(\mathbf{x}_1)) \dots (\mathbf{x}_{n_1}, \phi_1(\mathbf{x}_{n_1})) \rangle$ and $L_{\phi_2} = \langle (\mathbf{y}_1, \phi_2(\mathbf{y}_1)) \dots (\mathbf{y}_{n_2}, \phi_2(\mathbf{y}_{n_2})) \rangle$ If $\mathbf{x} \in \Omega_{D_1}$ and $\mathbf{y} \in \Omega_{D_2}$ with $\mathbf{x}^{\downarrow D_1 \cap D_2} = \mathbf{y}^{\downarrow D_1 \cap D_2}$ then we denote by $\mathbf{xy}$ the configuration in $\Omega_{D_1 \cup D_2}$ for which $\mathbf{xy}^{\downarrow D_1} = \mathbf{x}$ and $\mathbf{xy}^{\downarrow D_2} = \mathbf{y}$.

function COMBINE(ϕ_1, ϕ_2, t)
  $L \leftarrow \langle\rangle; i \leftarrow 1; j \leftarrow 1$
  initialise timer to t units
  while *timer*() > 0 and ($i \leq n_1$ or $j \leq n_2$) **do**
    if $j > n_2$ or ($i \leq n_1$ and $\pi_1(\mathbf{x}_i) \succeq \pi_2(\mathbf{y}_j)$) **then**
      while *timer*() > 0 and for r from 1 to $j-1$ **do**
        if $\mathbf{x}_i^{\downarrow D_1 \cap D_2} = \mathbf{y}_r^{\downarrow D_1 \cap D_2}$ **then**
          insert $[\mathbf{x}_i\mathbf{y}_r, \phi_1(\mathbf{x}_1) \cdot \phi_2(\mathbf{y}_r)]$ into L.
        end if
      end while
      $i \leftarrow i + 1$;
    else
      while *timer*() > 0 and for r from 1 to $i-1$ **do**
        if $\mathbf{y}_j^{\downarrow D_1 \cap D_2} = \mathbf{x}_r^{\downarrow D_1 \cap D_2}$ **then**
          insert $[\mathbf{x}_r\mathbf{y}_j, \phi_1(\mathbf{x}_r) \cdot \phi_2(\mathbf{y}_j)]$ into L.
        end if
      end while
      $j \leftarrow j + 1$
    end if
  end while
  return valuation corresponding to L
end function

2.9.1 Arithmetic potentials

To extend the valuation algebra instance of arithmetic potentials from section 2.2.1, we need to define the partial order. The time-bound combination operation follows from the time-bound combination for semiring induced ordered valuation algebra definition.

Partial order For two arithmetic potentials π, π' ($\pi, \pi' : \Omega_D \to \mathbb{R}^+$):

$$\pi \succeq \pi' \text{ equivalent to } d(\pi) = d(\pi') \text{ and } \pi(\mathbf{x}) \geq \pi'(\mathbf{x}) \text{ for all } \mathbf{x} \in \Omega_D \tag{2.33}$$

The above defines a completeness relation. The infimum for a set of potentials Π on D is

$$\inf(\Pi)(\mathbf{x}) = \min_{\pi \in \Pi} \pi(\mathbf{x}). \tag{2.34}$$

Null elements The least complete arithmetic potentials are those for which $\pi(\mathbf{x}) = 0$ for all $\mathbf{x} \in \Omega_D$, which are also the null elements of the valuation algebra.

2.9.2 Disjunctive normal forms

▶ **Remark.** This instance of disjunctive normal forms considers it at the formula level as this is the approach in [Haenni, 2004]. We note that the valuation algebra of formulae and the model are identical upto logical equivalence as we discussed in section 2.2.2, and as also discussed in [Kohlas et al., 1999].

For disjunctive normal forms, we consider a formula to be more informative or a better approximation than another if its terms are subterms of the other. This leads to a larger model, which is the basis for approximation. The elements of the ordered valuation algebra in this case are:

Partial order If we consider DNF potentials $[\delta, D]$ and $[\delta', D']$ where D, D' are the respective domains and let

$$[\delta, D] \succeq [\delta', D'] \text{ equivalent to } D = D' \text{ and } \forall \tau' \in \delta' \text{ there is } \tau \in \delta \text{ such that } \tau \subseteq \tau' \tag{2.35}$$

Set inclusion $\subseteq$ forms a lattice. Thus $\succeq$ forms a partial order and satisfies *R1*. For Δ, a set of DNF potentials on domain D, then infimum $\inf(\Delta)$ is the combination $\otimes\Delta$ of the potentials in Δ.

Null element $[\emptyset, D]$ is the null element of the combination.

2.9.3 Relational algebras

We recall that a valuation in a relational algebra is a relation, comprising *tuples* over certain variables, called *attributes*. Thus a relation R over a set of variables S is a subset of the configuration space of S, i.e. $R \subseteq \Omega_S$.

Partial order The completeness relation is defined as such, for R_1, R_2 over the same domain $d(R_1) = d(R_2) = D$:

$$R_1 \succeq R_2 := R_1 \supseteq R_2 \tag{2.36}$$

This satisfies *R1* as relations on different domains are not compared, and $\supseteq$ gives a partial order on sets, and the infimum relation on any domain D is the empty relation $\emptyset$ on D.

Null element The null element n_D for a domain D is the empty relation $\emptyset$ on the domain D.

2.10 Set potential ordered valuation algebra

▶ **Remark.** We use set potentials in keeping with the rest of the thesis; [Haenni, 2004] considers belief potentials which are normalised. As in the case of semiring induced ordered valuation algebras, we do not show validity of *R1 – R9* since we do not use ordered valuation algebras in the thesis.

We consider the set potentials defined as $m : 2^{\Omega_D} \to \mathbb{R}^+$. The set of such set potentials forms a valuation algebra. The extension of the valuation algebra to an ordered valuation algebra is as follows:

Partial order For two such belief potentials m, m', we define the completeness relation as such:

$$m \succeq m' := d(m) = d(m'),\ m(A) \geq m'(A) \quad \text{for all } A \subseteq \Omega_D \tag{2.37}$$

Thus set potentials with different domains cannot be compared, $\geq$ is a total order, and there is the natural infimum belief potential for every domain D which is the belief potential with $m(A) = 0$ for all A.

Null element The null element is also the infimum belief potential for a domain D with $m(A) = 0$ for all A.

Time-bound combination The time bound combination operation is defined as follows: For $m_1 : 2^{\Omega_{D_1}} \to A$ and $m_2 : 2^{\Omega_{D_2}} \to A$, the time bound combination $m_1 \otimes_t m_2$ is a function $m_1 \otimes m_2 : 2^{\Omega_{D_1 \cup D_2}} \to A$. Thus the time bound combination has the same type as the combination operation $\otimes$. It is implemented as the COMBINE(m_1, m_2, t) algorithm below.

We represent a mass function $m : 2^{\Omega_D} \to [0, 1]$ as an ordered list $L_m = \langle [A_1, m(A_1), \ldots, [A_n, m(A_n)] \rangle$, where $\{A_1, \ldots, A_n\} \subseteq 2^{\Omega_D}$ is the set of focal elements of m with $m(A_i) \geq m(A_j)$ for $1 \leq i \leq j \leq n$. If m_1 on D_1 and m_2 on D_2 are represented by corresponding sets of focal sets $\{A_1, \ldots, A_{n_1}\}$ and $\{B_1, \ldots, B_{n_2}\}$, then the resource-bounded combination can be carried out thus:

```
function COMBINE(m_1, m_2, t)
    L ← ⟨⟩; i ← 1; j ← 1
    initialise timer to t units
    while timer() > 0 and (i ≤ n_1 or j ≤ n_2) do
        if j > n_2 or (i ≤ n_1 and m_1(A_i) ≥ m_2(B_j)) then
            while timer() > 0 and for r from 1 to j − 1 do
                C ← A_i ⋈ B_r
                m ← m_1(A_i) · m_2(B_r)
                if [C, m'] ∈ L then
                    m' ← m' + m
                else
                    insert [C, m] into L.
                end if
            end while
            i ← i + 1;
        else
            while timer() > 0 and for r from 1 to i − 1 do
                C ← A_r ⋈ B_j
                m ← m_1(A_r) · m_2(B_j)
                if [C, m'] ∈ L then
                    m' ← m' + m
                else
                    insert [C, m] into L.
                end if
            end while
            j ← j + 1
        end if
    end while
    return L
end function
```

2.11 Approximate inference algorithm

Like the exact inference algorithm in section 2.7, the approximate inference algorithm comprises two steps: inward and outward propagation.

2.11.1 Inward propagation

Let's consider the inward propagation algorithm in INWARD(N) and replace the combination operator with $\otimes_t$. As the total number of combinations in the inward phase is $r-1$ (one at each step of the algorithm for every non-leaf node in the BJT); a trivial way to ensure a T unit time-bound algorithm is to allocate $T/(r-1)$ units for each of the $r-1$ combination steps. However some combinations may take more or less time, so this is not a particularly efficient approach.

If T is the remaining time, and s the number of remaining steps, then $t = T/s$ is the approximate time which we can allocate for the next combination $\phi_s(L(n)) \otimes \phi_s(R(n))$ at node $n \in next(N)$; also

▶ **Notation.** EFFECTIVE TIME FOR COMBINATION. let $T_{\text{eff}}(\phi_1, \phi_2)$ denote the effective time required for the exact combination $\phi_1 \otimes \phi_2$. such that

1. for $t \geq T_{\text{eff}}(\phi_1, \phi_2)$, $\phi_1 \otimes \phi_2 = \phi_1 \otimes_t \phi_2$
2. for $t < T_{\text{eff}}(\phi_1, \phi_2)$, $\phi_1 \otimes \phi_2 \succ \phi_1 \otimes_t \phi_2$

The effective time for the combination $\phi_1 \otimes \phi_2$ at node n is

$$\mathcal{T}(n) = T_{\text{eff}}(\phi_s(L(n)), \phi_s(R(n))) \tag{2.38}$$

For the remaining $s-1$ steps of the algorithm, $(T - \min\{\mathcal{T}(n), T/s\})/(s-1)$ units are left. Thus the actual allocation of available time $t = T/s$ increases monotonically during the process. To maximise increase at the beginning, it is important, at each step of the algorithm to select the node $n \in next(N)$ such that $\mathcal{T}(n)$ is as small as possible. As $\mathcal{T}(n)$ is unknown, we use some $\mathcal{T}'(n)$ to estimate the time required for exact combination at each node n and minimise $\mathcal{T}'(n)$ instead of $\mathcal{T}(n)$. A possible $\mathcal{T}'(n)$ could be some function of $d(n)$ for example.

We construct a resource-bounded version of the propagation algorithm with input BJT having nodes $N = \{n_1, \ldots, n_{2r-1}\}$ and total available time T. We use $steps(N) := r-1$ to represent the number of necessary combinations; calling the function $timer()$ allows one to determine the number of remaining units. The definitions of $L(n), R(n), P(n), S(n)$ are the same as in def. 2.7.1.

function INWARD(N, T)
  $s \leftarrow steps(N)$;
  initialise timer to T units.
  for all $n \in leaves(N)$ do $\phi_s(n) \leftarrow \phi(n)^{-\Delta(n)}$
  while $next(N) \neq \emptyset$ **do**
    select n from $next(N)$ such that $\mathcal{T}'(n)$ is minimal;
    $\phi(n) \leftarrow \phi_s(L(n)) \otimes_{T/s} \phi_s(R(n))$;
    $\phi_s(n) \leftarrow \phi(n)^{-\Delta(n)}$
    $s \leftarrow s - 1$
    $T \leftarrow timer()$
  end while
  return $\phi_s(root(N))$
end function

This process terminates after a maximum of T units. The result is an incomplete valuation $\phi \in \Phi$, such that $(\otimes\Psi)^{\downarrow D} \succeq \phi$ as a direct consequence of (R3), (R4) and (R6). If ϕ, ϕ' are the outputs of INWARD(N, T) and INWARD(N, T') with $T' \geq T$, then (R6) guarantees that $\phi' \succeq \phi$. As a consequence of (R8) and (R9), INWARD$(N, 0)$ returns the least complete valuation n_D and INWARD(N, ∞) produces the exact solution $(\otimes\Psi)^{\downarrow D}$.

2.12 Conclusion

In this chapter, we reviewed the framework of generic inference, the associated theory of valuation algebras, and presented examples of generic inference as an unifying abstraction across various representations of information, from relational algebras to Bayesian networks. Then we described the binary join tree structure and the inward and outward propagation algorithms for exact inference.

Next, we reviewed the framework of ordered valuation algebras, which supports approximate (but not anytime) inference using a time-bound combination operator. We showed the applicability of ordered valuation algebras to the instances of arithmetic potentials, disjunctive normal forms, relational algebras and set potentials. In the next chapter we introduce our work on anytime inference in valuation algebras which builds upon the formalisms presented in this chapter.

Chapter 3

Anytime Generic Inference

As we noted in the introduction, one of the contributions of the thesis is in the area of anytime inference. Anytime inference refers to solving the inference problem approximately, while allowing for the approximation to be improved with time and making use of intermediate results. This can be useful in applications with limited space, and for efficiency reasons, such as in continuous learning and robotics. In this chapter we introduce the framework for constructing an anytime inference algorithm based on the valuation algebra framework reviewed in chapter 2.

The chapter is organised as follows: Section 3.1 introduces the anytime ordered valuation algebra framework. We show in section 3.2 that the class of semiring induced valuation algebras are amenable to anytime inference, and in section 3.3 show that set potential valuation algebras support anytime inference. In section 3.4, we discuss outward propagation. Section 3.5 concludes.

We also present proofs of soundness (thm 3.4) and completeness (thm 3.5) of anytime inference. Soundness of anytime inference refers to the monotonicity of anytime inference with time on the obtained solutions, and convergence to the exact valuation given sufficient time. Completeness of anytime inference asserts the existence of an upper bound on the time for computing the exact valuation.

3.1 Anytime Ordered Valuation Algebras

In this section, we extend valuation algebras in a structure we refer to as *anytime ordered valuation algebras*. We introduce the extension, and in the following section give examples of anytime ordered valuation algebras. The primary purpose of introducing anytime ordered valuation algebras is to develop an anytime inference algorithm within the framework of generic inference. Such extensions preserve the genericity inherent in valuation algebras, but add axioms to simplify or add features to the inference algorithm. For example, valuation algebras were extended to weighted valuation algebras to study communication complexity [Pouly and Kohlas, 2005].

We shall informally discuss the additional components in an anytime ordered valuation algebra before proceeding to the formal definition. In addition to the operations of a valuation algebra, we also have (i) a *partial order* operation which relates valuations with the same domain[1] (ii) a *composition* operator, which composes two valuations of the same domain into a valuation more complete (containing more information) than either[2], (iii) a *truncation* operator which truncates a valuation to a less informative valuation and a complementary *inverse truncation* operation, and finally, (iv) a *time bound update* operation which is designed to extend (add more information via composition) a combination of two valuations.

[1] This is the same operation as in ordered valuation algebras.

[2] A version of this operation and truncation specific to belief function computations was considered in [Haenni and Lehmann, 2002]

▶ **Definition 3.1.** ANYTIME ORDERED VALUATION ALGEBRA. An *anytime ordered valuation algebra* is a tuple $\langle \Phi, V, d, \otimes, \downarrow, \succeq, K, \oplus, \rho \rangle$ which satisfies axioms $A7$ – $A21$ presented in this definition, and where the sub-tuple $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a valuation algebra. The additional operations compared to a valuation algebra are:

In the following we use $\mathbb{N}$ to denote the set of natural numbers which contain zero.

Partial order There is a partial order $\succeq : \Phi \times \Phi$ on Φ.

Composition The *composition operator* $\oplus : \Phi \times \Phi \to \Phi$ is a family of operations

$$\oplus_D : \Phi_D \times \Phi_D \to \Phi_D \quad D \subseteq V \tag{3.1}$$

with each $\oplus_D$ operating on a type labelled by a domain D.

Truncation The *truncation function* $\rho : \Phi \times \mathbb{N} \to \Phi$ is a family

of operations

$$\rho_D : \Phi_D \times \mathbb{N} \to \Phi_D \tag{3.2}$$

with each ρ_D operating on a type labelled by a domain D. domain D and is referred to as ρ collectively.

The inverse truncation is the complement of the truncated valuation $\rho(\phi, k)$ such that $\bar{\rho}(\phi, k) \oplus \rho(\phi, k) = \phi$

We also have a **inverse truncation** function $\bar{\rho}_D : \Phi_D \times \mathbb{N} \to \Phi_D$.

We define the associated notion of **valuation size** $|\cdot| : \Phi \to \mathbb{N}$ of a valuation ϕ as $|\phi| := \min\{k : \rho(\phi, k) = \phi\}$

Thus we limit ourselves to finite valuations.

Time bound update The time bound update is a function, with the integer input parameter as time:

$$K : \Phi \times \Phi \times \Phi \times \Phi \times \mathbb{N} \to \Phi \times \mathbb{N} \times \mathbb{N}$$

This operation is denoted by K in the formal framework and UPDATE in the algorithm.

Like with the previous operations, the time bound update is indexed by the type of the valuation and is referred to as K collectively:

$$K_{A,B} : \Phi_A \times \Phi_A \times \Phi_B \times \Phi_B \times \mathbb{N} \to \Phi_{A\cup B} \times \mathbb{N} \times \mathbb{N}$$

$$K(\phi_A, \phi'_A, \phi_B, \phi'_B, t) = \langle \phi, k_A, k_B \rangle, \text{ where}$$

$$\phi := (\phi_A \otimes \rho(\phi'_B, k_B)) \oplus (\rho(\phi'_A, k_A) \otimes \phi_B) \oplus (\rho(\phi'_A, k_A) \otimes \rho(\phi'_B, k_B))$$

This structure $\langle \Phi, V, d, \otimes, \downarrow, \succeq, K, \oplus, \rho \rangle$ is an *anytime ordered valuation algebra* if $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a valuation algebra and the following axioms *A7 – A21* are satisfied:

A7 – A11 are axioms relating to the approximation relation (partial order $\preceq$), out of which *A7 – A10* are similar to *R1 – R4* of ordered valuation algebras; any differences are noted.

A7 Partial Order[3]: There is a partial order $\succeq$ on Φ such that $\phi \succeq \phi'$ implies $d(\phi) = d(\phi')$ for all $\phi, \phi' \in \Phi$.

[3] In *R4*, there is an additional condition that the infimum $\inf(\Psi)$ exists for all $\Psi \subseteq \Phi_D$. It is not clear why this condition is required in ordered valuation algebras; it is also not used in our framework so we have not kept it.

A8 Null element[4] : The least complete valuations for a particular domain D are the null valuations for that domain D. We require the property that $n_D \otimes \phi = n_{D\cup A}$ for all $\phi \in \Phi_A$ and for all $A, D \subseteq V$, which implies that n_D is the null element for the domain D. Since null elements for a particular domain are unique, we also have $n_D^{\downarrow D'} = n_{D'}$ for all $D' \subseteq D$.

[4] This is a stronger condition than that in *R2* which states that $n_D \otimes \phi = n_D$. Our condition implies the condition in *R2*, if we set $A = D$.

A9 Combination preserves partial order[5]: If $\phi_1, \phi'_1, \phi_2, \phi'_2 \in \Phi$ are valuations such that $\phi_1 \succeq \phi'_1$ and $\phi_2 \succeq \phi'_2$, then $\phi_1 \otimes \phi_2 \succeq \phi'_1 \otimes \phi'_2$.

[5] same as *R3*

A10 Projection preserves partial order[6]: If $\phi, \phi' \in \Phi$ are valuations such that $\phi \succeq \phi'$, then $\phi^{\downarrow D} \succeq \phi'^{\downarrow D}$ for all $D \subseteq d(\phi) = d(\phi')$.

[6] same as *R4*

A11 Composition preserves partial order: If $\psi, \phi, \phi' \in \Phi$ are valuations such that $\phi \succeq \phi'$, then $\psi \oplus \phi \succeq \psi \oplus \phi'$.

A12 Composition forms a monoid: The composition operator $\oplus$ forms a commutative monoid with the valuation sub-semigroup $(\Phi_D, \oplus)$, with the null valuation n_D as the identity element.

A12 – A14 pertain to the composition operation $\oplus$

A13 Combination $\otimes$ distributes over composition $\oplus$:

$$(\phi_1 \oplus \phi_1') \otimes \phi = (\phi_1 \otimes \phi) \oplus (\phi_1' \otimes \phi) \tag{3.3}$$

A14 Projection $\downarrow$ distributes over composition $\oplus$:

$$(\phi' \oplus \phi'')^{\downarrow D} = \phi'^{\downarrow D} \oplus \phi''^{\downarrow D}, D \subseteq d(\phi). \tag{3.4}$$

A15 Size of truncated valuations: For $k \leq |\phi|$, $|\rho(\phi, k)| = k$ and $|\bar{\rho}(\phi, k)| = |\phi| - k$.

A15 – A17 pertain to the truncation operation $\downarrow$.

A16 Truncation monotonically increasing: Truncation is monotonically increasing with the non-negative integer parameter. For $k, k' \leq |\phi|$, $k' > k$ implies $\rho(\phi, k') \succ \rho(\phi, k)$ and vice versa for the complementary valuation $\bar{\rho}(\phi, k') \prec \bar{\rho}(\phi, k)$.

A17 Zero truncation: Truncation with $k = 0$ gives the neutral valuation for that domain: $\rho(\phi, 0) = n_{d(\phi)}$ where $d(\phi)$ is the domain of ϕ.

In the following, $\langle \phi, k_1, k_2 \rangle$ is the result of $K(\phi_1, \phi_1', \phi_2, \phi_2', t)$:

A18 Time bounded update: The time for computation of $K(\phi_1, \phi_1', \phi_2, \phi_2', t)$ does not exceed t units. This is satisfied operationally through the specification of the function as an algorithm which performs an approximate combination within the specified time.

A18 – A21 pertain to the time-bound truncated update K.

A19 Monotonic time bounded update: The condition $t' \geq t$ implies $k_1' \geq k_1,\ k_2' \geq k_2$ where $\langle \phi', k_1', k_2' \rangle = K(\phi_1, \phi_1', \phi_2, \phi_2', t')$.

A20 Zero time update: For $t = 0$, $k_1 = k_2 = 0$

A21 Maximum time for update: For all $\phi_1', \phi_2' \in \Phi$ there exists a $T_{\phi_1', \phi_2'}$ such that for $t \geq T_{\phi_1', \phi_2'}$, $k_1 = |\phi_1'|$, $k_2 = |\phi_2'|$.

▶ **Remark.** The time bound update operation is designed to add to an existing combination, $\phi_A \otimes \phi_B$, making it more

complete. This can be seen if we add $\phi_A \otimes \phi_B$ to ϕ, where $\langle \phi, k_A, k_B \rangle = K(\phi_A, \phi'_A, \phi_B, \phi'_B, t)$:

$$
\begin{aligned}
&\phi_A \otimes \phi_B \oplus \phi \\
&= (\phi_A \otimes \phi_B) \oplus (\phi_A \otimes \rho(\phi'_B, k_B)) \oplus (\rho(\phi'_A, k_A) \otimes \phi_B) \oplus \\
&\quad (\rho(\phi'_A, k_A) \otimes \rho(\phi'_B, k_B)) \\
&= (\phi_A \oplus \rho(\phi'_A, k_A)) \otimes (\phi_B \oplus \rho(\phi'_B, k_B)) \quad \text{using distributivity from } A13
\end{aligned}
$$

▶ **Definition 3.2.** TIME BOUNDED COMBINATION. We also define another operation, the *time bounded combination* K' which can be expressed in terms of K:

$$
\begin{aligned}
K'(\phi'_A, \phi'_B, t) &:= K(n_A, \phi'_A, n_B, \phi'_B, t) \\
&= \langle (n_A \otimes \rho(\phi'_B, k_B)) \oplus (\rho(\phi'_A, k_A) \otimes n_B) \oplus (\rho(\phi'_A, k_A) \otimes \rho(\phi'_B, k_B)), k_A, k_B \rangle \\
&= \langle n_{A \cup B} \oplus n_{A \cup B} \oplus (\rho(\phi'_A, k_A) \otimes \rho(\phi'_B, k_B)), k_A, k_B \rangle && \text{using } A8 \\
&= \langle \rho(\phi'_A, k_A) \otimes \rho(\phi'_B, k_B), k_A, k_B \rangle && \text{using } A12
\end{aligned}
$$

where n_A, n_B are the null valuations for the corresponding domains A and B.

We can also show that anytime ordered valuation algebras are also ordered valuation algebras, with a single additional condition that the infimum $\inf(\Psi)$ exists for all $\Psi \subseteq \Phi_D$, where $\Phi_D \subseteq \Phi$ is the sub-semigroup of the valuations with domain D. Thus any instance of anytime ordered valuation algebras which also satisfy this additional condition, satisfy the axioms of an ordered valuation algebra.

▶ **Theorem 3.1.** *An anytime ordered valuation algebra* $\langle \Phi, V, d, \otimes, \downarrow, \succeq, K, \oplus, \rho \rangle$ *is also an ordered valuation algebra* $\langle \Phi, V, d, \otimes, \downarrow, \succeq, \otimes_t \rangle$ *with*

$$
\phi_1 \otimes_t \phi_2 := \rho(\phi_1, k_1) \otimes \rho(\phi_2, k_2) \quad \textit{where}
$$
$$
\langle \rho(\phi_1, k_1) \otimes \rho(\phi_2, k_2), k_1, k_2 \rangle = K'(\phi_1, \phi_2, t)
$$

and the additional condition that the infimum $\inf(\Psi)$ *exists for all* $\Psi \subseteq \Phi_D$*, where* $\Phi_D \subseteq \Phi$ *is the sub-semigroup of the valuations with domain* D*.*

Proof. The core valuation algebra component $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is the same in both the cases. Here, let the domains of ϕ_1, ϕ_2 be D_1, D_2 respectively. We prove the statement of the theorem by showing that if any instance satisfies *A7 – A21*, it shall also satisfy *R1 – R9*, subject to the additional condition being

satisfied. We proceed to the enumeration of the ordered valuation algebra axioms with the accompanying proofs.

R1 Partial Order: The partial order axiom in ordered valuation algebras has the additional condition that the infimum $\inf(\Psi)$ exists for all $\Psi \subseteq \Phi_D$. It is not clear why this is required in the framework, so we have omitted it, and made it an additional requirement in the theorem.

R2 Null element: In comparison to the ordered valuation algebra condition

$$\phi \otimes n_D = n_D \otimes \phi = n_D, \ \forall \phi \in \Omega_D$$

our condition (*A8*) is stronger

$$\phi \otimes n_D = n_D \otimes \phi = n_{D \cup A}, \ \forall \phi \in \Omega_A, A \subseteq V$$

Our condition is more general, and the condition for ordered valuation follows when we set $A = D$.

R3 Combination preserves partial order These are the same in both, corresponding to *A9* in our framework.

R4 Projection preserves partial order These are the same in both, corresponding to *A10* in our framework.

R5 Computation time: We have to show that effective time to compute $\phi_1 \otimes_t \phi_2$ is less than t units. This is guaranteed by *A18* which limits the execution time of K, and thus K'.

R6 Approximate valuation: We have to show that the exact combination of two potentials $\phi_1 \otimes \phi_2$ is approximated by the time-bound combination; $\phi_1 \otimes \phi_2 \succeq \phi_1 \otimes_t \phi_2$. We know that $\phi_1 \otimes_t \phi_2 = \rho(\phi_1, k_1) \otimes \rho(\phi_2, k_2)$. From the definition of ρ, we have $\rho(\phi_1, k_1) \preceq \phi_1$ (likewise for ϕ_2). As combination $\otimes$ preserves the partial order (*A9*), we have $\rho(\phi_1, k_1) \otimes \rho(\phi_2, k_2) \preceq \phi_1 \otimes \phi_2$.

R7 Monotone: We have to show that the approximation gets better with time; $\phi_1 \otimes_{t'} \phi_2 \succeq \phi_1 \otimes_t \phi_2$ for all $t' \geq t$. This is guaranteed by the monotonicity of the time bound update K, and thus K, in *A19*.

R8 Null valuation at $t = 0$: We have to show that $\phi_1 \otimes_0 \phi_2 = n_D$ where n_D is the null element of the valuation algebra

corresponding to the domain $D = D_1 \cup D_2$. From *A20*, we have $k_1 = k_2 = 0$ for $t = 0$. This gives us

$$\begin{aligned}\phi_1 \otimes_0 \phi_2 &= \rho(\phi_1, 0) \otimes \rho(\phi_2, 0) \\ &= n_{D_1} \otimes n_{D_2} && \text{using } A17 \\ &= n_{D_1 \cup D_2} && \text{using } A8\end{aligned}$$

R9 Complete: We have to show that given sufficient time, the time-bound combination operation returns the same result as the exact combination; $\phi_1 \otimes_\infty \phi_2 = \phi_1 \otimes \phi_2$. While we do not have infinite time in our framework, we do have an temporal upper bound which is equivalent because we have finite valuations. From *A21*, we have for all $\phi_1, \phi_2 \in \Phi$ there exists a T_{ϕ_1,ϕ_2} such that for $t \geq T_{\phi_1,\phi_2}$, $k_1 = |\phi_1|$, $k_2 = |\phi_2|$. Thus for $t \geq T_{\phi_1,\phi_2}$, we have

$$\begin{aligned}\phi_1 \otimes_t \phi_2 &= \rho(\phi_1, |\phi_1|) \otimes \rho(\phi_2, |\phi_2|) \\ &= \phi_1 \otimes \phi_2 && \text{from definition of size}\end{aligned}$$

□

3.1.1 Anytime inference algorithm

We shall show that we can construct a anytime inference algorithm in the anytime ordered valuation algebra framework. Here we shall focus on developing a refinement algorithm which can improve upon approximate inference results.

To develop a refinement algorithm which can incrementally obtain more complete valuations, we need to store the partial valuations at each step. We use a modified version of the propagation algorithm [Shenoy, 1997, Haenni, 2004]. This method is similar to that adopted by [Haenni and Lehmann, 2002] though the referenced method only applies for belief function computations, and lacks proofs of correctness.

Intuitive understanding of the algorithm. We recall that in the inward propagation algorithm INWARD(N), propagation of information as messages proceeded from the leaves to the root. The leaf nodes contain exact valuations as they constitute the input to the inference problem. For any non-leaf node n, there are two children, $L(n)$ and $R(n)$ which are the left and right children of n. In exact inference, the child nodes compute the portion of the valuation relevant to

the parent node $\phi^{\downarrow d(n)\cap d(P(n))}$ and send it as a message (ϕ_s) to the parent. In anytime inference, we compute a partial valuation based on combination of truncations of the child valuations. Our algorithm comprises two parts: an initial inward propagation INWARD(N,t) followed by successive refinements REFINE(N,t).

Initial inward propagation. For every node n, we have a portion of the valuation at the child node $(L(n), R(n))$ which has been taken into consideration for constructing the parent valuation. This portion is stored and denoted by τ $(\tau(L(n)), \tau(R(n)))$. We also have the complementary valuation, the portion which was not used to construct the parent due to lack of time. This portion is also stored and is denoted by κ $(\kappa(L(n)), \kappa(R(n)))$. The initial assignment of τ, κ in a propagation from the leaves to the root constitutes the INWARD(N,t) algorithm.

Successive refinements. For refinement using the REFINE(N,t) algorithm, we extend the non-leaf valuations by combining portions of the child valuations which have not been combined (κ). In the following procedures, $\Delta(n) = d(n) \setminus d(P(n))$[7] is the set of variables to be eliminated as we propagate messages from the children nodes to the parent node. To get the solution to the inference problem at the final step, we also define $\Delta(root(N)) = d(root(N)) \setminus X$ where X is the query. There are r valuations in the knowledgebase resulting in $r-1$ combination steps in the BJT.

[7] This definition of $\Delta(n)$ gives us $\phi^{d(n)\cap d(P(n))} = \phi^{-\Delta(n)}$

Traversal sequence for propagation. The propagation proceeds from the leaves to the root. Thus it is helpful to consider the BJT as a directed acyclic graph, with the edges reversed. Instead of the usual arrows from $P(n)$ to n, we have arrows from n to $P(n)$ which reflects the underlying dependency graph of the computation. Then the sequence of nodes to be traversed is the topological sort of the BJT G' where G' is the subgraph of G without the leaves of G. This traversal sequence which shall remain the same in both INWARD and REFINE is denoted by TOPOLOGICAL-SORT(G').

We hereby the present the algorithms.

1: **procedure** INWARD(N,t)
2: $s \leftarrow r-1$
3: initialise timer to t units.
4: for all $n \in leaves(G)$ do $\phi_s(n) \leftarrow \phi(n)^{-\Delta(n)}$

5: **foreach** $n \in$ TOPOLOGICAL-SORT(G') **do**
6: $(\phi(n), k_1, k_2) \leftarrow K(\phi_s(L(n)), \phi_s(R(n)), t/s)$
7: $\tau(L(n)) \leftarrow \rho(\phi_s(L(n)), k_1)$
8: $\tau(R(n)) \leftarrow \rho(\phi_s(R(n)), k_2)$
9: $\kappa(L(n)) \leftarrow \bar{\rho}(\phi_s(L(n)), k_1)$
10: $\kappa(R(n)) \leftarrow \bar{\rho}(\phi_s(R(n)), k_2)$
11: $\phi_s(n) \leftarrow \phi(n)^{-\Delta(n)}$
12: $s \leftarrow s - 1$
13: $t \leftarrow timer()$
14: **end foreach**
15: **return** $\phi_s(root(N))$
16: **end procedure**

1: **procedure** REFINE(N, t)
2: **initialise** timer to t units
3: $s \leftarrow r - 1$
4: **foreach** $n \in$ TOPOLOGICAL-SORT(G') **do**
5: $(\nu(n), k_1, k_2) \leftarrow K'(\tau(L(n)), \kappa(L(n)), \tau(R(n)), \kappa(R(n)), t/s)$
6: $t \leftarrow timer()$
7: $\tau(L(n)) \leftarrow \tau(L(n)) \oplus \rho(\kappa(L(n)), k_1)$
8: $\tau(R(n)) \leftarrow \tau(R(n)) \oplus \rho(\kappa(R(n)), k_2)$
9: $\kappa(L(n)) \leftarrow \bar{\rho}(\kappa(L(n)), k_1)$
10: $\kappa(R(n)) \leftarrow \bar{\rho}(\kappa(R(n)), k_2)$
11: $\phi(n) \leftarrow \phi(n) \oplus \nu(n)$
12: $\kappa(n) \leftarrow \kappa(n) \oplus \nu(n)^{-\Delta(n)}$
13: $s \leftarrow s - 1$
14: **end foreach**
15: **return** $\phi_s(root(N))$
16: **end procedure**

This procedure refines the existing valuations in the binary join tree G, taking at most time t units. We ensure that the algorithm is interruptible in lines 9–12 using appropriate caching of partial valuations.

3.1.2 Properties of anytime inference

In this section we shall prove the correctness of the algorithm in the previous section by proving two properties of the algorithm:

Soundness Soundness of anytime inference refers to the assertion that the valuations obtained in the refinement process

are approximations of the exact valuation, and that further refinements are better approximations of the exact valuation.

Completeness Completeness of anytime inference asserts the existence of an upper bound on the time for computing the exact valuation.

Before embarking on the proof of soundness, we introduce a couple of definitions, introduce the required notation for the proof, and prove a couple of lemmas which will be used in the main proof.

▶ **Definition 3.3.** LEVEL. The level of a node n is defined recursively as

$$\text{LEVEL}(n) = \begin{cases} 0 & \text{if } n \text{ is a leaf} \\ 1 + \max(\text{LEVEL}(L(n)), \text{LEVEL}(R(n))) \end{cases} \tag{3.5}$$

▶ **Notation.** In both the soundness and completeness proofs, there is an iterative improvement in terms of better approximation of a valuation at each step. For valuations in the BJT G with nodeset N, we shall follow the convention of affixing a superscript to denote which step of the refinement process it is at. Thus $\phi^i(n)$ is the valuation at node n after step i. We affix superscripts to the caches accordingly. For the values (k_1, k_2) obtained in line 6 of INWARD and line 5 of REFINE, we affix a tuple of (n, i) to denote the node n at which the combination is happening and the step i. So we have the following notation, in which i indicates the value of the quantity after step i of the refinement process, where step 0 is the initial propagation:

- valuation at node n: $\phi^i(n)$
- the portion of truncated valuation at node n which has been propagated: $\tau^i(n)$
- for κ, we need a tuple to denote the state as it is updated twice, for nodes at level > 0. $\kappa^{i,1}(n)$ represents the state after line 12 and $\kappa^{i,2}(n)$ represents the state after line 9 has been executed from the parent node.

  For the leaf nodes, $\kappa(n)$ is updated once as leaves are not part of TOPOLOGICAL-SORT(G'), thus here we can write $\kappa^i(n)$ without ambiguity.

  Thus κ (unlike τ) is updated twice in each iteration: once at

node n and once at $P(n)$ except for $n \in leaves(N)$, when κ is updated once. This is summarised in the diagram.

- truncation parameters at node n: $k_1^{n,i}, k_2^{n,i}$

This is the formalisation of the soundness property for step i of soundness at level d.

▶ **Definition 3.4.** SOUNDNESS(d, i)**.**

$$\text{SOUNDNESS}(d,i) := \phi^i(n) \preceq \phi(n);\ \phi^i(n) = \tau^i(L(n)) \otimes \tau^i(R(n))\ \forall\ n \in D_d \tag{3.6}$$

where D_d is the set of nodes at level d and $\phi(n)$ is the valuation obtained from exact inference at node n. Intuitively, SOUNDNESS(d, i) is the statement of the soundness property $\phi^i(n) \preceq \phi(n)$ holds for level d for step i. The second portion $\phi^i(n) = \tau^i(L(n)) \otimes \tau^i(R(n))$ is introduced as that shall be used in the induction hypothesis.

We also denote the soundness at a particular level d for all i collectively as:

$$\text{SOUNDNESS}(d) := \bigwedge_{k=0}^{j} \{\text{SOUNDNESS}(d,k)\} \tag{3.7}$$

▶ **Lemma 3.2.** *For $n \in leaves(N)$, $\tau^i(n) \oplus \kappa^i(n) = \phi_s(n)$ for all $i = \{0, \ldots, j\}$.*

Proof. Intuitively this proof can be easily seen as the valuations at the leaves are exact. So we combine more chunks of the valuation in successive refinements, but the composition of the valuation that has been combined (τ) and the part which has not been (κ) is always the exact valuation. Note that this shall not hold for non-leaf nodes as the part which has not been computed may itself be incomplete.

For the base case, we look at INWARD(N, t). For each leaf node, only one of lines 7 and 8 and one of lines 9 and 10 will be executed as it is either the left or the right node of the parent node. As they have the same form, we can write:

$$\begin{aligned} \tau^0(n) \oplus \kappa^0(n) &= \rho(\phi_s(n), k) \oplus \bar{\rho}(\phi_s(n), k) \\ &= \phi_s(n) \quad \text{by definition of } \rho \text{ and } \bar{\rho} \end{aligned}$$

This establishes the base case. For the induction step, let us assume $\phi^i(n) \oplus \kappa^i(n) = \phi_s(n)$. Then we have to show $\phi^{i+1}(n) \oplus \kappa^{i+1}(n) = \phi_s(n)$. For this we note that the only time τ and κ are updated in REFINE is in lines 7–10. Similar to the base case, whether the node n is a left leaf node or a right leaf

node does not matter. Then:

$$\begin{aligned}\tau^{i+1}(n) \oplus \kappa^{i+1}(n) &= (\tau^i(n) \oplus \rho(\kappa^i(n), k)) \oplus \bar{\rho}(\kappa^i(n), k) \\ &= \tau^i(n) \oplus \kappa^i(n) \quad \text{using } \oplus \text{ associativity and } \rho, \bar{\rho} \text{ complementarity} \\ &= \phi_s(n)\end{aligned}$$

This establishes the statement of the lemma. □

▶ **Lemma 3.3.** *For $n \in N - leaves(N)$, and c being one of $L(n), R(n)$, and at* REFINE *step i, the following identity holds:* $\tau^i(c) \oplus \kappa^{i+1,1}(c) = (\phi^{i+1}(c))^{-\Delta(c)}$

Proof. We now establish an identity from lines 7–10 of REFINE for non-leaf nodes. Using the symmetry of $L(n)$ and $R(n)$ we can write an identity, where c is one of $L(n), R(n)$, and k is k_1, k_2 respectively:

$$\begin{aligned}\tau^{i+1}(c) &= \tau^i(c) \oplus \rho(\kappa^{i+1,1}(c), k^{n,i+1}) && \text{line 7,8 REFINE} \\ \kappa^{i+1,2}(c) &= \bar{\rho}(\kappa^{i+1,1}(c), k^{n,i+1}) && \text{line 9,10 REFINE} \\ \tau^{i+1}(c) \oplus \kappa^{i+1,2}(c) &= \tau^i(c) \oplus \kappa^{i+1,1}(c) && \text{summing above, using } \rho, \bar{\rho} \text{ complementarity} \\ &= \tau^i(c) \oplus \kappa^{i,2}(c) \oplus \nu^{i+1}(c)^{-\Delta(c)} && \text{line 12 of REFINE to expand } \kappa^{i+1,1}(c)\end{aligned}$$

In the case of $c \in leaves(N)$, we know that there is only one κ update (in line 9 or 10 of REFINE depending on whether c is a left or right child). To avoid encumbering the notation at this point, we still consider two κ updates, with the second one being a null operation. This is achieved by setting $\nu^r(c) = 0$ for all $r \in \{0 \cdots j\}, c \in leaves(N)$.

We can expand the recursive form

$$\tau^{i+1}(c) \oplus \kappa^{i+1,2}(c) = \tau^i(c) \oplus \kappa^{i,2}(c) \oplus \nu^{i+1}(c)^{-\Delta(c)}$$

to get (with the limit at 1 as REFINE starts at iteration 1, following INWARD at 0):

$$\tau^i(c) \oplus \kappa^{i+1,1}(c) = \tau^{i+1}(c) \oplus \kappa^{i+1,2}(c) = \tau^1(c) \oplus \kappa^{1,2}(c) \oplus \nu^2(c)^{-\Delta(c)} \oplus \cdots \oplus \nu^{i+1}(c)^{-\Delta(c)}$$

We connect τ, κ from INWARD and REFINE via the following

identities:

$$\begin{aligned}
\tau^1(c) \oplus \kappa^{1,2}(c) &= \tau^0(c) \oplus \kappa^{1,1}(c) \\
&= \tau^0(c) \oplus \kappa^0(c) \oplus \nu^1(c)^{-\Delta(c)} \\
&= \phi_s^0(c) \oplus v^1(c)^{-\Delta(c)} && \text{using } \tau^0(c) \oplus \kappa^0(c) = \phi_s^0(c), \text{ lines 7–10 of INWARD} \\
&= (\phi^0(c) \oplus \nu^1(c))^{-\Delta(c)} && \text{using } A14, \text{ Projection } \downarrow \text{ distributes over composition } \oplus
\end{aligned}$$

Substituting into the previous equation, we get, via repeated application of $A14$:

$$\begin{aligned}
\tau^i(c) \oplus \kappa^{i+1,1}(c) &= \tau^1(c) \oplus \kappa^{1,2}(c) \oplus \nu^2(c)^{-\Delta(c)} \oplus \cdots \oplus \nu^{i+1}(c)^{-\Delta(c)} \\
&= (\phi^0(c) \oplus \nu^1(c))^{-\Delta(c)} \oplus \nu^2(c)^{-\Delta(c)} \oplus \cdots \oplus \nu^{i+1}(c)^{-\Delta(c)} \\
&= (\phi^1(c) \oplus \nu^2(c))^{-\Delta(c)} \oplus \nu^2(c)^{-\Delta(c)} \oplus \cdots \oplus \nu^{i+1}(c)^{-\Delta(c)} && \text{use } \phi^{i+1}(c) = \phi^i(c) \oplus \nu^{i+1}(c) \\
&= (\phi^i(c) \oplus \nu^{i+1}(c))^{-\Delta(c)} \\
&= \phi^{i+1}(c)^{-\Delta(c)}
\end{aligned}$$

which gives us the identity, for $n \in N - \mathit{leaves}(N)$ and c being one of $L(n), R(n)$:

$$\tau^i(c) \oplus \kappa^{i+1,1}(c) = \phi^{i+1}(c)^{-\Delta(c)}$$

□

► **Theorem 3.4. Soundness of anytime inference.** *Consider the following length $j+1$ sequence of operations for $j > 0$:* $\textsc{inward}(N_0, t_0 > 0)$, $\textsc{refine}(N_1, t_1), \ldots, \textsc{refine}(N_j, t_j)$ *where N_{k+1} is the node set of the BJT after step k, where step o is the initial inward propagation. If we denote $\phi_j := \textsc{refine}(N_j, t_j)$ for $j > 0$ then soundness asserts $\phi_0 \preceq \phi_1 \preceq \cdots \preceq \phi_j \preceq \phi$ where ϕ is the valuation that would been returned by exact inference and $\phi_0 := \textsc{inward}(N_0, t_0)$.*

Proof. We shall structure the proof in three stages. The first stage (S1) is an induction on the number of steps for level 1, the second stage is a induction on the level.

S1 The first stage[8] proves soundness for the nodes in G which are the immediate parents of the leaves, that is the nodes at level 1 of the tree. Thus we show that $\textsc{soundness}(1)$ is true. We use induction for this, showing $\textsc{soundness}(1, 0)$ as the base case, and proving the induction step by showing that $\textsc{soundness}(1, i+1)$ follows from $\textsc{soundness}(1, i)$, the induction hypothesis.

[8] Proof stage S1:

$$\textsc{soundness}(1, 0)$$
$$\textsc{soundness}(1, i) \rightarrow \textsc{soundness}(1, i+1)$$

S2 The second stage[9] shows that if soundness is true for nodes at level d and lesser then it also holds for the nodes at level $d+1$. We show this in two stages: we first show that the initial valuation ($i = 0$) at n for level $d+1$ satisfies soundness ($\text{SOUNDNESS}(d+1, 0)$), by using the fact that soundness holds for level d and below ($\bigwedge_{m=1}^{d} \{\text{SOUNDNESS}(m)\}$). Then we use $\text{SOUNDNESS}(d+1, 0)$ as a base case and use the induction hypothesis $\text{SOUNDNESS}(d+1, i)$ to show $\text{SOUNDNESS}(d+1, i+1)$. Then we apply induction to get $\text{SOUNDNESS}(d+1)$ if $\text{SOUNDNESS}(d)$ holds.

The process of using $\text{SOUNDNESS}(d+1, 0)$ as a base case and then the induction hypothesis $\text{SOUNDNESS}(d+1, i)$ to show $\text{SOUNDNESS}(d+1, i+1)$ is similar to S1 but not exactly the same. Otherwise we could have used nested induction, where we induct on i within the larger induction on the level d.

[9] Proof stage S2:

$$\bigwedge_{m=1}^{d} \{\text{SOUNDNESS}(m)\} \rightarrow \text{SOUNDNESS}(d+1, 0)$$
$$\text{SOUNDNESS}(d+1, i) \rightarrow \text{SOUNDNESS}(d+1, i+1)$$

S3 From *S1* and *S2*, we show that soundness is true at the root of the BJT[10] by applying induction using the base case $\text{SOUNDNESS}(1)$ and the induction step, $\text{SOUNDNESS}(d)$ implies $\text{SOUNDNESS}(d+1)$, which was shown in S2. From applying the induction, we get $\text{SOUNDNESS}(d_{max})$ is true, where d_{max} is the maximum value of d in the the BJT G. . Since the only node at d_{max} is the root node, and the valuation returned from the INWARD and REFINE algorithm is a (possibly marginalised) valuation at the root, this proves the statement of the soundness theorem.

[10] Proof stage S3: $\text{SOUNDNESS}(d_{max})$

For both *S1* and *S2*, it is trivial to see that we shall always have an non-decreasing sequence $\phi^0(n) \preceq \phi^1(n) \preceq \cdots \preceq \phi^j(n)$ from line 11 of $\text{REFINE}(N, t)$ as by definition $\phi \preceq \phi \oplus \phi'$.

S1. Here we shall prove $\text{SOUNDNESS}(1)$, soundness for nodes $n \in D_1$. Here D_1 is the set of nodes at level 1, i.e. those nodes whose children are both leaves. We shall give an inductive proof on the steps i, showing $\text{SOUNDNESS}(1, 0)$ as the base case, and proving the induction step by showing that $\text{SOUNDNESS}(1, i+1)$ follows from $\text{SOUNDNESS}(1, i)$, the induction hypothesis.

$\text{SOUNDNESS}(1)$

S1 Base. The base case for *S1* is to show $\phi^0(n) \preceq \phi(n)$ and $\phi^0(n) = \tau^0(L(n)) \otimes \tau^0(R(n))$. As this is the zeroth step, we are looking at the valuation at n after the INWARD invocation.

$\text{SOUNDNESS}(1, 0)$

We see that $\phi^0(n)$ is assigned once, in line 6 of INWARD:

$$\begin{aligned}(\phi(n), k_1, k_2) \leftarrow K(\phi_s(L(n)), \phi_s(R(n)), t/s) &\qquad \text{line 6, INWARD}\\ \phi^0(n) = \rho(\phi_s(L(n)), k_1^{n,0}) \otimes \rho(\phi_s(R(n)), k_2^{n,0}) &\qquad K \text{ definition}\\ \phi^0(n) = \tau^0(L(n)) \otimes \tau^0(R(n)) &\qquad \text{lines 7–8, INWARD}\end{aligned}$$

As $\rho(\phi, k) \preceq \phi$ by definition, we get $\rho(\phi_s(L(n)), k_1^{n,0}) \preceq \phi_s(L(n))$, and similarly for $R(n)$. As partial order is preserved under combination (*A9*), we have $\phi^0(n) \preceq \phi_s(L(n)) \otimes \phi_s(R(n)) = \phi(n)$.

S1 Induction. We have to show that given the induction hypothesis $\phi^i(n) \preceq \phi(n)$ and $\phi^i(n) = \tau^i(L(n)) \otimes \tau^i(R(n))$, we have $\phi^{i+1}(n) \preceq \phi(n)$ and $\phi^{i+1}(n) = \tau^{i+1}(L(n)) \otimes \tau^{i+1}(R(n))$.

$\text{SOUNDNESS}(1, i) \rightarrow \text{SOUNDNESS}(1, i+1)$

We note that the valuation $\phi(n)$ is only updated once in REFINE, in line 11, which is extended by $\nu^{i+1}(n)$ as given by line 5:

$$\begin{aligned}\nu^{i+1}(n) &= K'(\tau^i(L(n)), \kappa^i(L(n)), \tau(R(n)), \kappa(R(n), t/s)\\ &= (\tau^i(L(n)) \otimes \rho(\kappa^i(R(n)), k_2^{i+1})) \oplus (\rho(\kappa^i(L(n)), k_1^{i+1}) \otimes \tau^i(R(n))) \oplus\\ &\quad (\rho(\kappa^i(L(n), k_1^{i+1}) \otimes \rho(\kappa^i(R(n)), k_2^{i+1}))\end{aligned}$$

Here we note that since n is at level 1, it only depends on τ and κ from the previous step, i, instead of the current step. Then

$$\begin{aligned}\phi^{i+1}(n) &= \phi^i(n) \oplus \nu^{i+1}(n) && \text{line 11, REFINE}\\ &= (\tau^i(L(n)) \otimes \tau^i(R(n))) \oplus \nu^{i+1}(n) && \text{using induction hypothesis for } \phi^i(n)\\ &= (\tau^i(L(n)) \oplus \rho(\kappa^i(L(n)), k_1^{i+1})) && \text{substituting } \nu^{i+1}(n), \text{ and}\\ &\quad \otimes (\tau^i(R(n)) \oplus \rho(\kappa^i(R(n)), k_2^{i+1})) && \text{using } \otimes \text{ distributes over } \oplus \ (A13)\end{aligned}$$

Note that at this point, using lines 7, 8 of REFINE, we get $\tau^{i+1}(L(n)) = \tau^i(L(n)) \oplus \rho(\kappa^i(L(n)), k_1^{i+1})$ and similarly for $R(n)$, so we already get the second part of what we wanted to prove: $\phi^{i+1}(n) = \tau^{i+1}(L(n)) \otimes \tau^{i+1}(R(n))$. For the first part:

$$\begin{aligned}\phi^{i+1}(n) &\preceq (\tau^i(L(n)) \oplus \kappa^i(L(n))) \otimes (\tau^i(R(n)) \oplus \kappa^i(R(n))) && \text{using definition of } \rho \text{ and } A9\\ &= (\phi_s(L(n)) \otimes \phi_s(R(n)) = \phi(n) && \text{using lemma 3.2}\end{aligned}$$

This shows that $\phi^{i+1}(n) \preceq \phi(n)$.

S2. The second stage shows that if $\bigwedge_{m=1}^{d}\{\text{SOUNDNESS}(m)\}$ holds, that is soundness is true for nodes at level d and lesser,

$\bigwedge_{m=1}^{d}\{\text{SOUNDNESS}(m)\} \rightarrow \text{SOUNDNESS}(d+1)$

then SOUNDNESS$(d+1)$ is true, that is it holds for the nodes at level $d+1$. For this we shall show SOUNDNESS$(d+1,0)$ that is the valuation at nodes in level $d+1$ are approximations to the exact valuation at the first step (0), and induct to show that as refinement progresses, we get a successive sequence of approximations to the exact valuation at the node in level $d+1$.

S2 Base. We have to show $\phi^0(n) \preceq \phi(n)$ and $\phi^0(n) = \tau^0(L(n)) \otimes \tau^0(R(n))$ for $n \in D_{d+1}$ if the induction hypothesis holds.

$\bigwedge_{m=1}^{d}\{\text{SOUNDNESS}(m)\} \rightarrow \text{SOUNDNESS}(d+1,0)$

As this is the initial inward propagation step, we get, from line 6–8 in INWARD[11]: we get

[11] INWARD
6: $(\phi(n), k_1, k_2) \leftarrow K(\phi_s(L(n)), \phi_s(R(n)), t/s)$
7: $\tau(L(n)) \leftarrow \rho(\phi_s(L(n)), k_1)$
8: $\tau(R(n)) \leftarrow \rho(\phi_s(R(n)), k_2)$

$$\begin{aligned}\phi^0(n) &= \rho(\phi_s^0(L(n)), k_1) \otimes \rho(\phi_s^0(R(n)), k_2) \quad \text{from definition of } K \\ &= \tau^0(L(n)) \otimes \tau^0(R(n))\end{aligned}$$

This shows the second part of the proof. For the first part:

$$\begin{aligned}\phi^0(n) &= \rho(\phi_s^0(L(n)), k_1) \otimes \rho(\phi_s^0(R(n)), k_2) \\ &\preceq \phi_s^0(L(n)) \otimes \phi_s^0(R(n)) \qquad \text{using } \rho \text{ and } A9\end{aligned}$$

Now we use the induction hypothesis $\phi^0(L(n)) \preceq \phi(L(n))$ (likewise for $R(n)$). This is true as $L(n), R(n)$ are at depth d or lesser. Then we can apply *A14* (Projection $\downarrow$ distributes over composition $\oplus$), to get $\phi_s^0(L(n)) \preceq \phi_s(L(n))$ as $\phi_s(L(n))$ is a projection of $\phi(L(n))$ (likewise for $R(n)$) using line 11 of INWARD. Then we have

$$\begin{aligned}\phi^0(n) &\preceq \phi_s^0(L(n)) \otimes \phi_s^0(R(n)) \\ &\preceq \phi_s(L(n)) \otimes \phi_s(R(n)) \qquad \text{using } A9 \\ &= \phi(n)\end{aligned}$$

S2 Induction. In the S2 base case, we showed that SOUNDNESS$(d+1,0)$ holds. Now we shall show the induction step, that is if SOUNDNESS$(d+1,i)$ is true, then SOUNDNESS$(d+1,i+1)$ is true.

SOUNDNESS$(d+1,i) \rightarrow$ SOUNDNESS$(d+1,i+1)$

This proceeds similarly to the S1 induction case, with one important difference. Here we have $n \in D_{d+1}$, where $d \geq 1$. In this case, line 12 in REFINE extends the portion of the valuation which has not been computed for the left and right nodes of n. Thus we use the τ cache at the earlier timestep i, but the

κ cache from the *current* timestep $i+1$, as the left and right nodes of n would have been visited before n in TOPOLOGICAL-SORT. Then from line 5 of REFINE:

$$\begin{aligned}\nu^{i+1}(n) &= K'(\tau^i(L(n)), \kappa^{i+1,1}(L(n)), \tau^i(R(n)), \kappa^{i+1,1}(R(n), t/s) \\ &= (\tau^i(L(n)) \otimes \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1})) \oplus (\rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1}) \otimes \tau^i(R(n))) \oplus \\ &\quad (\rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1}) \otimes \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1}))\end{aligned}$$

Here κ has the index $(i+1,1)$ as this is *after* the first update to κ but before the second on line 9–10 of REFINE.

As $\phi^i(n) = \tau^i(L(n)) \otimes \tau^i(R(n))$ using the induction hypothesis SOUNDNESS$(d+1, i)$, we get

REFINE
7: $\tau(L(n)) \leftarrow \tau(L(n)) \oplus \rho(\kappa(L(n)), k_1)$
8: $\tau(R(n)) \leftarrow \tau(R(n)) \oplus \rho(\kappa(R(n)), k_2)$
9: $\kappa(L(n)) \leftarrow \bar{\rho}(\kappa(L(n)), k_1)$
10: $\kappa(R(n)) \leftarrow \bar{\rho}(\kappa(R(n)), k_2)$
11: $\phi(n) \leftarrow \phi(n) \oplus \nu(n)$
12: $\kappa(n) \leftarrow \kappa(n) \oplus \nu(n)^{-\Delta(n)}$

$$\begin{aligned}\phi^{i+1}(n) &= \phi^i(n) \oplus \nu^{i+1}(n) \\ &= (\tau^i(L(n)) \otimes \tau^i(R(n))) \oplus \nu^{i+1}(n) \\ &= (\tau^i(L(n)) \otimes \tau^i(R(n))) \\ &\quad \oplus (\tau^i(L(n)) \otimes \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1})) \\ &\quad \oplus (\rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1}) \otimes \tau^i(R(n))) \\ &\quad \oplus (\rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1}) \otimes \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1})) \\ &= (\tau^i(L(n)) \oplus \rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1}) \otimes (\tau^i(R(n)) \oplus \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1}) && \text{using } A13 \\ &= \tau^{i+1}(L(n)) \otimes \tau^{i+1}(R(n)) && \text{lines 7,8 of REFINE}\end{aligned}$$

The last line establishes the second part of this proof, now we have to show $\phi^{i+1}(n) \preceq \phi(n)$:

$$\begin{aligned}\phi^{i+1}(n) &= (\tau^i(L(n)) \oplus \rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1}) \otimes (\tau^i(R(n)) \oplus \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1}) \\ &\preceq (\tau^i(L(n)) \oplus \kappa^{i+1,1}(L(n))) \otimes (\tau^i(R(n)) \oplus \kappa^{i+1,1}(R(n))) \quad \text{using } A11 \\ &= (\phi^{i+1}(L(n)))^{-\Delta(L(n))} \otimes (\phi^{i+1}(R(n)))^{-\Delta(R(n))} \quad \text{using lemma 3.3, } A9\end{aligned}$$

Now we can use the fact that $\bigwedge_{m=1}^{d}\{\text{SOUNDNESS}(m)\}$ holds to assert $\phi^{i+1}(L(n)) \preceq \phi(L(n))$ (similarly for $R(n)$). As partial order $\preceq$ is preserved under projection $\downarrow$ ($A10$), we also get $(\phi^{i+1}(L(n)))^{-\Delta(L(n))} \preceq \phi_s(L(n))$ where $\phi_s(L(n)) := \phi(L(n))^{-\Delta(L(n))}$. Substituting these into the above gives

$$\phi^{i+1}(n) \preceq \phi_s(L(n)) \otimes \phi_s(R(n))$$

From the inward propagation for exact inferencewe know that $\phi(n) = \phi_s(L(n)) \otimes \phi_s(R(n))$. This shows $\phi^{i+1}(n) \preceq \phi(n)$ and $\phi^{i+1}(n) = \tau^{i+1}(L(n)) \otimes \tau^{i+1}(R(n))$ is shown previously. This establishes SOUNDNESS$(d+1, i+1)$ from

SOUNDNESS$(d+1, i)$ and $\bigwedge_{m=1}^{d}\{\text{SOUNDNESS}(m)\}$.

S3. This is as described in the proof structure section: From *S1* and *S2*, we show that soundness is true at the root of the BJT by applying induction using the base case SOUNDNESS(1) and the induction step, SOUNDNESS(d) implies SOUNDNESS$(d+1)$, which was shown in S2. From applying the induction, we get SOUNDNESS(d_{max}) is true, where d_{max} is the maximum value of d in the the BJT G. . Since the only node at d_{max} is the root node, and the valuation returned from the INWARD and REFINE algorithm is a (possibly marginalised) valuation at the root, this proves the statement of the soundness theorem.

□

► **Theorem 3.5. Completeness of anytime inference.** *Consider the following length $j+1$ sequence of operations for $j > 0$,* INWARD$(N_0, t_0 > 0)$, REFINE$(N_1, t_1), \ldots,$ REFINE(N_j, t_j) *with t_i such that the sequence of valuations ϕ_i obtained is strictly increasing:*

$$\phi_0 \prec \phi_1 \prec \cdots \prec \phi_j \prec \phi$$

Here N_{k+1} is the node set of the BJT after step k, where step o is the initial inward propagation. The valuations at each step are defined as $\phi_0 :=$ INWARD(N_0, t_0) *and $\phi_i :=$* REFINE(N_i, t_i)*. The valuation obtained from exact inference is ϕ. Then* completeness *asserts that*

$$\exists k > 0 \text{ s.t. } \phi_k = \phi$$

Proof. Note that we are implicitly using *A19*, as we are assuming that there is a time t_i for which some progress has been made in the refinement step i at the root node (valuation has been updated). This is only possible because time bound truncated combination is monotonic with time.

Like the soundness proof, we shall structure this proof into three parts: C1 proves that completeness holds at level 1 (leaves are the nodes). C2 is the inductive step that shows completeness holds for level m if it holds for all levels $n < m$. C3 is the application of the induction to give us the completeness result at the root node.

C1. We use the result from lemma 3.2 as applied to the leaf nodes $c = \{L(n), R(n)\}$:

$$\tau(c) \oplus \kappa(c) = \phi(c)^{-\Delta(c)} = \phi_s(c) \tag{3.8}$$

$\phi_s(c) = \phi(c)^{-\Delta(c)} = \phi(c)^{\downarrow(d(c)\cap d(P(c)))}$

Now we consider the valuation update at line 11 of REFINE for step $i+1$. Since n is at level 1, it only depends on τ and κ from the previous step i.

From soundness proof stage S1 we get:

$$\phi^{i+1}(n) = (\tau^i(L(n)) \oplus \rho(\kappa^i(L(n)), k_1^{n,i+1})) \otimes (\tau^i(R(n)) \oplus \rho(\kappa^i(R(n)), k_1^{n,i+1})))$$

From lines 7 to 10 of REFINE, we get:

$$\begin{aligned}
\tau^{i+1}(L(n)) &= \tau^i(L(n)) \oplus \rho(\kappa^i(L(n)), k_1^{n,i+1}) \\
\tau^{i+1}(R(n)) &= \tau^i(R(n)) \oplus \rho(\kappa^i(R(n)), k_2^{n,i+1}) \\
\kappa^{i+1}(L(n)) &= \bar{\rho}(\kappa^i(L(n)), k_1^{n,i+1}) \\
\kappa^{i+1}(R(n)) &= \bar{\rho}(\kappa^i(R(n)), k_2^{n,i+1}) \qquad (3.9)
\end{aligned}$$

From *A16* we know that $\bar{\rho}(\phi, k) \prec \phi$. We also know from *A15* that for $k < |\phi|$ (the case here), $|\bar{\rho}(\phi, k)| = |\phi| - k$.

Thus from the last two equations above we get a decreasing sequence for $m \in \{L(n), R(n)\}$:

$$\kappa^0(m) \succ \kappa^1(m) \succ \cdots \succ \kappa^i(m) \succ \kappa^{i+1}(m) \qquad (3.10)$$

which, implies a corresponding decreasing sequence of their respective sizes

$$|\kappa^0(m)| > |\kappa^1(m)| > \cdots > |\kappa^i(m)| > |\kappa^{i+1}(m)| \qquad (3.11)$$

As valuations have finite size, the above sequence is bounded below by 0. Thus we shall eventually reach the null valuation. Let this κ sequence terminate for some r_1, r_2 for $L(n)$ and $R(n)$ respectively, that is $\kappa^{r_1}(L(n)) = \emptyset$ and $\kappa^{r_2}(R(n)) = \emptyset$.

Without loss of generality, take $r_1 \geq r_2$. Then τ at $R(n)$ will stop updating from steps $r_2 + 1$ to r_1. This can be seen at line 8 of REFINE at step r_2:

$$\begin{aligned}
\tau^{r_2+1}(R(n)) &= \tau^{r_2}(R(n)) \oplus \rho(\kappa^{r_2}(R(n)), k_2^{n,r_2+1}) \\
&= \tau^{r_2}(R(n)) \oplus \emptyset \qquad \text{using } \kappa^{r_2}(R(n)) = \emptyset \\
&= \tau^{r_2}(R(n))
\end{aligned}$$

Thus $\tau^{r_1}(R(n)) = \tau^{r_2}(R(n))$.

Then at step $i \geq r_1$ we have $c \in \{L(n), R(n)\}$:

$$\begin{aligned} \tau^i(c) \oplus \kappa^i(c) &= \phi_s(c) && \text{using (3.8)} \\ \tau^i(c) \oplus \kappa^i(c) &= \tau^i(c) = \phi_s(c) && \text{using } \kappa^i(c) = \emptyset \end{aligned}$$

Then we have

$$\begin{aligned} \phi^{r_1+1}(n) &= (\tau^{r_1}(L(n)) \oplus \rho(\kappa^{r_1}(L(n)), k_1^{n,r_1+1})) \otimes (\tau^{r_1}(R(n)) \oplus \rho(\kappa^{r_1}(R(n)), k_1^{n,r_1+1}))) \\ \phi^{r_1+1}(n) &= \tau^{r_1}(L(n)) \otimes \tau^{r_1}(R(n)) \quad \text{using } \kappa^{r_2}(L(n)) = \emptyset \text{ and } \kappa^{r_2}(R(n)) = \emptyset \\ &= \phi_s(L(n)) \otimes \phi_s(R(n)) \\ &= \phi(n) \end{aligned}$$

This shows completeness for level 1 we have a $k = r_1 + 1$ for which $\phi^k(n) = \phi(n)$.

C2 This is the inductive step that shows completeness holds for level $d+1$ if it holds for all levels $\leq d$.

For the second part of the proof we consider the equation for valuation update $\phi^{i+1}(n)$ from S2 induction step.

$$\phi^{i+1}(n) = (\tau^i(L(n)) \oplus \rho(\kappa^{i+1,1}(L(n)), k_1^{n,i+1})) \otimes (\tau^i(R(n)) \oplus \rho(\kappa^{i+1,1}(R(n)), k_2^{n,i+1})))$$

As in the case for soundness, for level > 1, κ is updated twice in each refinement. This is reflected in the indices of κ which gain a superscript of 1 or 2 denoting the update sequence.

As in S2 induction, in the case of $c \in \mathit{leaves}(N)$; where $c \in \{L(n), R(n)\}$, we know that there is only one κ update (in line 9 or 10 of REFINE depending on whether c is a left or right child). To avoid encumbering the notation at this point, we still consider two κ updates, with the second one being a null operation. This is achieved by setting $v^r(c) = 0$ for all $r \in \{0 \cdots j\}, c \in \mathit{leaves}(N)$.

The intuition here is that once the κ update in line 12 switches off for $L(n)$ and $R(n)$, the problem becomes equivalent to C1 where the completeness at the first level was dependent on the children being exact. We shall then use the same argument as before.

As the $L(n), R(n)$ are at level $\leq d$, we can use the induction hypothesis, i.e. they satisfy the completeness criterion. From completeness, we have $\phi^{r_1}(L(n)) = \phi(L(n))$ and $\phi^{r_2}(R(n)) = \phi(R(n))$ for some $r_1, r_2 \in \mathbb{N}$. Again as before, without loss of generality we assume $r_1 \geq r_2$ and $r = \max(r_1, r_2)$. Since the

valuation at $L(n)$ and $R(n)$ have reached completion (with the one at $R(n)$ reaching completion earlier), for step $r+1$, we can write $\nu^{r+1}(c) = \emptyset$ for $c \in \{L(n), R(n)\}$; this must be the case, as otherwise we would have had the valuation $\phi^r(c)$ updated in line 11 of REFINE. This switches off the κ update in line 12 for $L(n)$ and $R(n)$.

Using line 12 of REFINE to write, for $c \in \{L(n), R(n)\}$:

$$\begin{aligned} \kappa^{r+1,1}(c) &= \kappa^{r,2}(c) \oplus \nu^{r+1}(c)^{-\Delta(c)} \\ &= \kappa^{r,2}(c) \quad (\nu^{r+1}(c) = \emptyset) \end{aligned}$$

Thus we can drop the additional superscript on κ of 1 or 2 (as κ updates only once).

From lemma 3.3, at step r:

$$\tau^r(c) \oplus \kappa^{r+1,1}(c) = (\phi^{r+1}(c))^{-\Delta(c)}, \quad c \in \{L(n), R(n)\}$$

Dropping the extra subscript on κ:

$$\tau^r(c) \oplus \kappa^r(c) = (\phi^{r+1}(c))^{-\Delta(c)} = (\phi(c))^{-\Delta(c)}$$

as $\phi^{r+1}(m) = \phi(m)$ via completeness.

We can then write:

$$\tau^i(m) \oplus \kappa^i(m) = (\phi(m))^{-\Delta(m)} \quad i > r$$

At node n, we then have the following τ, κ updates:

$$\begin{aligned} \tau^{i+1}(L(n)) &= \tau^i(L(n)) \oplus \rho(\kappa^i(L(n)), k_1^{n,i+1}) \\ \tau^{i+1}(R(n)) &= \tau^i(R(n)) \oplus \rho(\kappa^i(R(n)), k_2^{n,i+1}) \\ \kappa^{i+1}(L(n)) &= \bar{\rho}(\kappa^i(L(n)), k_1^{n,i+1}) \\ \kappa^{i+1}(R(n)) &= \bar{\rho}(\kappa^i(R(n)), k_2^{n,i+1}) \end{aligned} \tag{3.12}$$

With the above equations we end up with the same situation as C1, specifically (3.8) and (3.9). Then we get, using similar reasoning:

$$\kappa^0(m) \succ \kappa^1(m) \succ \cdots \succ \kappa^i(m) \succ \kappa^{i+1}(m) \tag{3.13}$$

The decreasing sequence of κ eventually culminates in the null valuation.

The rest of the argument proceeds in the same manner as C1. Thus we have shown completeness holds for level $d+1$ if

it holds for all levels $\leq d$.

C3 To obtain the completeness result at the root node, we apply induction on the level using C1 as the base case and C2 as the induction step. □

3.2 Semiring induced anytime ordered valuation algebras

In the following section we discuss the class of anytime ordered valuation algebras that are induced by a positive (def. 2.11), partially ordered (def. 2.10), commutative semiring. We shall use the notation for configuration space Ω_D of a set of variables D and the associated notation for tuples from definition 2.2 (semiring induced valuation algebras).

As in the case of semiring induced ordered valuation algebras, several instances such as arithmetic potentials, disjunctive normal forms and relational algebras are part of this general class. In chapter 4, we shall consider these semiring induced anytime ordered valuation algebras in greater detail.

► **Definition 3.5.** SEMIRING INDUCED ANYTIME ORDERED VALUATION ALGEBRAS**.** A semiring induced anytime ordered valuation algebra is a tuple $\langle \Phi, V, d, \otimes, \downarrow, \succeq, K, \oplus, \rho \rangle$, where $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a semiring induced valuation algebra (def. 2.2) induced by a positive, partially ordered, commutative semiring $(A, +, \cdot, 0, 1)$ with a partial order $\preceq_A$ on elements of the semiring.

We can consider a semiring induced valuation ϕ with domain D, i.e. $\phi : \Omega_D \to A$ as being represented by a list $L_\phi = \langle (\mathbf{x}_1, \phi(\mathbf{x}_1)) \dots (\mathbf{x}_n, \phi(\mathbf{x}_n)) \rangle \in 2^{\Omega_D \times A}$, where $\{\mathbf{x}_1 \dots \mathbf{x}_n\} \subseteq \Omega_D$ is the subset that does not map to 0. For convenience, we shall use the notation $S(D) := 2^{\Omega_D \times A}$. If the inducing semiring has a total order, then we order the tuples of L_ϕ by the semiring values $\varphi(\mathbf{x}_i)$, from highest to lowest.[12] Then we have the following operations:

[12] We do this to ensure that configuration with the highest "mass" get combined first. We comment on this further in chapter 4.

Partial order Valuations having the same domain D form a partially ordered set $\Phi_D \subseteq \Phi$. For two valuations $\phi, \phi' \in \Phi_D$,

$$\phi \preceq \phi' \text{ iff } \phi(\mathbf{x}) \preceq_A \phi'(\mathbf{x}) \quad \forall\, \mathbf{x} \in \Omega_D \tag{3.14}$$

Composition The composition operation ($\oplus : \Phi_D \times \Phi_D \to \Phi_D$)

is defined as follows:

$$(\phi \oplus \psi)(\mathbf{x}) = \phi(\mathbf{x}) + \psi(\mathbf{x}), \quad \forall\, \mathbf{x} \in \Omega_D \tag{3.15}$$

Truncation The truncation function $\rho : \Phi_D \times \mathbb{N} \to \Phi_D$ is defined as follows: $\rho(\phi, k)$ is such that

$$L_{\rho(\phi,k)} = \langle (\mathbf{x}_1, \phi(\mathbf{x}_1)) \dots (\mathbf{x}_m, \phi(\mathbf{x}_m)) \rangle \quad m = \min(k, n),\ n = |L_\phi| \tag{3.16}$$

Thus the truncated valuation corresponds the first k elements of L_φ. We also get the size of a valuation $|\phi| := |L_\phi| = n$. The inverse truncation is similarly defined as

$$L_{\bar{\rho}(\phi,k)} = \langle (\mathbf{x}_{m+1}, \phi(\mathbf{x}_{m+1})) \dots (\mathbf{x}_n, \phi(\mathbf{x}_n)) \rangle \tag{3.17}$$

Time bound update The time bound update is defined as an algorithm: $K(\phi_1, \phi_1', \phi_2, \phi_2', t) := \textsc{update}(\phi_1, \phi_1', \phi_2, \phi_2', t)$. We also define the secondary methods:

- $\textsc{insert}(\phi_1 : \Phi_A,\ \phi_2 : \Phi_B,\ i, j : \mathbb{N},\ L : S(A \cup B))$.
  This method inserts a combination into the configuration space.
- $\textsc{combine-extend}(\phi_1 : \Phi_A,\ \phi_2 : \Phi_B, \langle i, j : \mathbb{N},\ L : S(A \cup B) \rangle, i', j' : \mathbb{N}) \to \mathbb{N} \times \mathbb{N} \times (S(A \cup B))$.
  This method incrementally adds combinations into the configuration and updates the state, going from the state $\rho(\phi_1, i) \otimes \rho(\phi_2, j)$ to $\rho(\phi_1, i + i') \otimes \rho(\phi_2, j + j')$.
- $\textsc{update-extend}(\phi_1, \phi_1' : \Phi_A,\ \phi_2, \phi_2' : \Phi_B, \langle i, j : \mathbb{N},\ L_1, L_2, L_3 : S(A \cup B) \rangle, i', j' : \mathbb{N}) \to \mathbb{N} \times \mathbb{N} \times S(A \cup B) \times S(A \cup B) \times S(A \cup B)$.
  This method updates the state, effectively going from

$$(\rho(\phi_1', i) \otimes \phi_2) \oplus (\phi_1 \otimes \rho(\phi_2', j)) \oplus (\rho(\phi_1', i) \otimes \rho(\phi_2', j)) \quad \text{initial state, to}$$
$$(\rho(\phi_1', i + i') \otimes \phi_2) \oplus (\phi_1 \otimes \rho(\phi_2', j + j')) \oplus (\rho(\phi_1', i + i') \otimes \rho(\phi_2', j + j')) \quad \text{final state}$$

  where L_1, L_2, L_3 are the lists corresponding to the valuations being composed by $\oplus$.
- $\textsc{update}(\phi_1, \phi_1' : \Phi_A,\ \phi_2, \phi_2' : \Phi_B, t : \mathbb{N}) \to \phi_{A \cup B} \times \mathbb{N} \times \mathbb{N}$ is the implementation of the time bound update operation K.

We then consider two valuations ϕ_1 on domain D_1 and ϕ_2 on domain D_2 with $L_{\phi_1} = \langle (\mathbf{x}_1, \phi_1(\mathbf{x}_1)) \dots (\mathbf{x}_{n_1}, \phi_1(\mathbf{x}_{n_1})) \rangle$ and $L_{\phi_2} = \langle (\mathbf{y}_1, \phi_2(\mathbf{y}_1)) \dots (\mathbf{y}_{n_2}, \phi_2(\mathbf{y}_{n_2})) \rangle$ If $\mathbf{x} \in \Omega_{D_1}$ and

$\mathbf{y} \in \Omega_{D_2}$ with $\mathbf{x}^{\downarrow D_1 \cap D_2} = \mathbf{y}^{\downarrow D_1 \cap D_2}$ then we denote by $\mathbf{xy}$ the configuration in $\Omega_{D_1 \cup D_2}$ for which $\mathbf{xy}^{\downarrow D_1} = \mathbf{x}$ and $\mathbf{xy}^{\downarrow D_2} = \mathbf{y}$. Then we can define the algorithms:

1: **function** INSERT(ϕ_1, ϕ_2, i, j, L)
2: **if** $\mathbf{x}_i^{\downarrow D_1 \cap D_2} = \mathbf{y}_j^{\downarrow D_1 \cap D_2}$ **then**[13]
3: insert $[\mathbf{x}_i\mathbf{y}_j, \phi_1(\mathbf{x}_i) \cdot \phi_2(\mathbf{y}_j)]$ into L.
4: **end if**
5: **end function**

[13] Here $\mathbf{x}_i, \mathbf{y}_j$ refer to the tuples in $L_{\varphi_1}, L_{\varphi_1}$ respectively

---

1: **function** UPDATE-EXTEND($\phi_1, \phi_1', \phi_2, \phi_2', \langle i, j, L_1, L_2, L_3 \rangle, i', j'$)
2: **for** $k \leftarrow 1$ to $|\phi_1|$ **do**
3: **for** $m \leftarrow j + 1$ to $j + j'$ **do**
4: INSERT($\phi_1, \phi_2', k, m, L_1$)
5: **end for**
6: **end for**
7: **for** $k \leftarrow 1$ to $|\phi_2|$ **do**
8: **for** $m \leftarrow i + 1$ to $i + i'$ **do**
9: INSERT($\phi_2, \phi_1', k, m, L_2$)
10: **end for**
11: **end for**
12: $\langle i, j, L_3 \rangle =$ COMBINE-EXTEND($\phi_1', \phi_2', \langle i, j, L_3 \rangle, i', j'$)
13: **return** $\langle i, j, L_1, L_2, L_3 \rangle$
14: **end function**

1: **function** COMBINE-EXTEND($\phi_1, \phi_2, \langle i, j, L \rangle, i', j'$)
2: **for** $k \leftarrow 1$ to $i + i'$ **do**
3: **for** $m \leftarrow j + 1$ to $j + j'$ **do**
4: INSERT(ϕ_1, ϕ_2, k, m, L)
5: **end for**
6: **end for**
7: **for** $k \leftarrow i + 1$ to $i + i'$ **do**
8: **for** $m \leftarrow 1$ to $j + j'$ **do**
9: INSERT(ϕ_1, ϕ_2, k, m, L)
10: **end for**
11: **end for**
12: **return** $\langle i + i', j + j', L \rangle$
13: **end function**

1: **function** UPDATE($\phi_1, \phi_1', \phi_2, \phi_2', t$)

2: $\langle i, j, L_1, L_2, L_3 \rangle \leftarrow \langle 0, 0, \langle\rangle, \langle\rangle, \langle\rangle \rangle$
3: $n_1 \leftarrow |\phi_1'|; n_2 \leftarrow |\phi_2'|$
4: initialise timer to t units
5: **while** $timer() > 0$ and $i < n_1$ and $j < n_2$ **do**
6: $\langle i, j, L_1, L_2, L_3 \rangle \leftarrow$ UPDATE-EXTEND$(\phi_1, \phi_1', \phi_2, \phi_2', \langle i, j, L_1, L_2, L_3 \rangle, 0, 1)$
7: **if** not $timer() > 0$ **then**
8: **break**
9: **end if**
10: $\langle i, j, L_1, L_2, L_3 \rangle \leftarrow$ UPDATE-EXTEND$(\phi_1, \phi_1', \phi_2, \phi_2', \langle i, j, L_1, L_2, L_3 \rangle, 1, 0)$
11: **end while**
12: **if** $i \geq n_1$ **then**
13: **while** $timer() > 0$ and $j < n_2$ **do**
14: $\langle i, j, L_1, L_2, L_3 \rangle \leftarrow$ UPDATE-EXTEND$(\phi_1, \phi_1', \phi_2, \phi_2', \langle i, j, L_1, L_2, L_3 \rangle, 0, 1)$
15: **end while**
16: **else**
17: **while** $timer() > 0$ and $i < n_1$ **do**
18: $\langle i, j, L_1, L_2, L_3 \rangle \leftarrow$ UPDATE-EXTEND$(\phi_1, \phi_1', \phi_2, \phi_2', \langle i, j, L_1, L_2, L_3 \rangle, 1, 0)$
19: **end while**
20: **end if**
21: **return** $\langle$TO-VALUATION$(L_1) \oplus$ TO-VALUATION$(L_2) \oplus$ TO-VALUATION$(L_3), i, j \rangle$
22: **end function**

TO-VALUATION takes the list representation of a valuation and converts it into the valuation.

▶ **Theorem 3.6.** *The structure introduced in def. 3.5 satisfies the axioms of an anytime ordered valuation algebra.*

Proof. To show that def. 3.5 is an anytime ordered valuation algebra, we need to show that *A7 – A21* hold.

A7 Partial Order: This is as in the definition.

A8 Null element: The null element for a domain D is defined as the unique function n_D which maps all elements to the semiring zero; $n_D : \Omega_D \to \{0\}$. That this is the least element follows from the positivity of the semiring.

Then we have to show

1. $n_D \otimes \phi = n_{D \cup A}$, where $\phi \in \Omega_A$. For all $\mathbf{x} \in \Omega_{D \cup A}$:

$$(n_D \otimes \phi)(\mathbf{x}) = n_D(\mathbf{x}^{\downarrow D}) \cdot \phi(\mathbf{x}^{\downarrow A}) = 0$$

Thus $n_D \otimes \phi = n_{D \cup A}$ as it maps all $\mathbf{x}$ to 0.

2. $n_D^{\downarrow D'} = n_{D'}$. For all $\mathbf{x} \in \Omega_{D'}$

$$n_D^{\downarrow D'}(\mathbf{x}) = \sum_{z \in \Omega_D : \mathbf{z}^{\downarrow T} = \mathbf{x}} n_D(\mathbf{z}) = 0$$

Thus $n_D^{\downarrow D'} = n_{D'}$.

A9 Combination preserves partial order: For $\phi_1 \succeq \phi_1', \phi_2 \succeq \phi_2'$, we have to show $\phi_1 \otimes \phi_2 \succeq \phi_1' \otimes \phi_2'$. For all $\mathbf{x} \in \Omega_{D_1 \cup D_2}$, we have

$$\begin{aligned}(\phi_1 \otimes \phi_2)(\mathbf{x}) &= \phi_1(\mathbf{x}^{\downarrow D_1}) \cdot \phi_2(\mathbf{x}^{\downarrow D_2}) \\ &\succeq_A \phi_1'(\mathbf{x}^{\downarrow D_1}) \cdot \phi_2'(\mathbf{x}^{\downarrow D_2}) = (\phi_1' \otimes \phi_2')(\mathbf{x}) \quad \text{using lemma 2.3}\end{aligned}$$

This follows from distributivity of $\succeq_A$ over $\cdot$, and from $\phi_i(\mathbf{x}^{\downarrow D_i}) \succeq_A \phi_i'(\mathbf{x}^{\downarrow D_i})$ for $i = \{1, 2\}$.

A10 Projection preserves partial order: For $\phi \succeq \phi'$, we have to show $\phi^{\downarrow D'} \succeq \phi'^{\downarrow D'}$ for $D' \subseteq D$. For all $\mathbf{x} \in \Omega_{D'}$, we have

$$\phi^{\downarrow D'}(\mathbf{x}) = \left(\sum_{\mathbf{z} \in \Omega_D :\ \mathbf{z}^{\downarrow D'} = \mathbf{x}} \phi(\mathbf{z}) \right) \succeq_A \left(\sum_{\mathbf{z} \in \Omega_D :\ \mathbf{z}^{\downarrow D'} = \mathbf{x}} \phi'(\mathbf{z}) \right) = \phi'^{\downarrow D'}(\mathbf{x}) \quad \text{using lemma 2.3}$$

This follows from distributivity of $\succeq_A$ over $+$, and from $\phi(\mathbf{z}) \succeq_A \phi'(\mathbf{z})$ for all $\mathbf{z} \in \Omega_D$.

A11 Composition preserves partial order: We have to show that if $\psi, \phi, \phi' \in \Phi$ are valuations such that $\phi \succeq \phi'$, then $\psi \oplus \phi \succeq \psi \oplus \phi'$. Let $d(\phi) = d(\phi') = d(\psi) = D$, as is required by *A7*. Then from $\phi \succeq \phi'$, we have $\phi(\mathbf{x}) \succeq_A \phi'(\mathbf{x})$ for all $\mathbf{x} \in \Omega_D$. From the monotone property of partial order in a semiring (def. 2.10), we get $\psi(\mathbf{x}) + \phi(\mathbf{x}) \succeq_A \psi(\mathbf{x}) + \phi'(\mathbf{x})$. As this is a point-wise expression that is true for all $\mathbf{x} \in \Omega_D$, we have $\psi \oplus \phi \succeq \psi \oplus \phi'$.

A12 Composition forms a monoid: We recall that the composition operation ($\oplus : \Phi_D \times \Phi_D \to \Phi_D$) is defined as follows:

$$(\phi \oplus \psi)(\mathbf{x}) = \phi(\mathbf{x}) + \psi(\mathbf{x}), \quad \forall\ \mathbf{x} \in \Omega_D \tag{3.18}$$

Here + is semiring addition which is associative and commutative, causing $\oplus$ to be so as well. The identity for $\oplus$ is the neutral element of the domain D, n_D which follows from o being the semiring additive identity.

A13 Combination $\otimes$ distributes over composition $\oplus$: If $\phi_1 = \phi_1' \oplus \phi_1''$ and $\phi_2 = \phi_2' \oplus \phi_2''$ then we have to show that:

$\phi_1 \otimes \phi_2 = (\phi_1' \otimes \phi_2') \oplus (\phi_1' \otimes \phi_2'') \oplus (\phi_1'' \otimes \phi_2') \oplus (\phi_1'' \otimes \phi_2'')$.
LHS applied to $\mathbf{x}$ is $\phi_1(\mathbf{x}^{\downarrow S}) \times \phi_2(\mathbf{x}^{\downarrow T})$, where $d(\phi_1) = S$ and $d(\phi_2) = T$.

$$\begin{aligned}\text{RHS is } &(\phi_1'(\mathbf{x}^{\downarrow S}) \cdot \phi_2'(\mathbf{x}^{\downarrow T})) + (\phi_1'(\mathbf{x}^{\downarrow S}) \cdot \phi_2''(\mathbf{x}^{\downarrow T})) + \\ &(\phi_1''(\mathbf{x}^{\downarrow S}) \cdot \phi_2'(\mathbf{x}^{\downarrow T})) + (\phi_1''(\mathbf{x}^{\downarrow S}) \cdot \phi_2''(\mathbf{x}^{\downarrow T})) \\ = &(\phi_1'(\mathbf{x}^{\downarrow S}) + \phi_1''(\mathbf{x}^{\downarrow S})) \cdot (\phi_2'(\mathbf{x}^{\downarrow S}) + \phi_2''(\mathbf{x}^{\downarrow T}) = \text{LHS} \\ &\text{using distributivity of } \cdot \text{ over } +.\end{aligned}$$

A14 Projection $\downarrow$ distributes over composition $\oplus$: We have to show that if $\phi = \phi' \oplus \phi''$ that $\phi^{\downarrow D} = \phi'^{\downarrow D} \oplus \phi''^{\downarrow D}$, where $D \subseteq d(\phi)$. The LHS applied to $\mathbf{x}$ is $\phi^{\downarrow D}(\mathbf{x}) = \sum_{\mathbf{z}^{\downarrow D} = \mathbf{x}} \phi(\mathbf{z}) = \sum_{\mathbf{z}^{\downarrow D} = \mathbf{x}} (\phi' \oplus \phi'')(\mathbf{z})$, and the RHS is

$$\begin{aligned}(\phi'^{\downarrow D} \oplus \phi''^{\downarrow D})(\mathbf{x}) &= \phi'^{\downarrow D}(\mathbf{x}) + \phi''^{\downarrow D}(\mathbf{x}) \\ = \sum_{\mathbf{z}^{\downarrow D} = \mathbf{x}} \phi'(\mathbf{z}) + \sum_{\mathbf{z}^{\downarrow D} = \mathbf{x}} \phi''(\mathbf{z}) &= \sum_{\mathbf{z}^{\downarrow D} = \mathbf{x}} (\phi' \oplus \phi'')(\mathbf{z})\end{aligned}$$

where we use the associativity and commutativity of $+$.

A15 Size of truncated valuations: Follows from definition.

A16 Truncation monotonically increasing: We have to show that $\rho(\phi, k) \succ \rho(\phi, k')$ for all $k > k'$, and $k, k' \leq |\varphi|$. For the semiring induced valuation $\phi : \Omega_D \to A$ we defined $\rho(\phi, k)$ as the valuation corresponding to

$$L_{\rho(\phi,k)} = \langle (\mathbf{x}_1, \phi(\mathbf{x})_1) \dots (\mathbf{x}_m, \phi(\mathbf{x})_m) \rangle \quad m = \min(k, |L_\varphi|) \tag{3.19}$$

As $k, k' \leq |\varphi| = |L_\varphi|$, we have $L_{\rho(\phi,k')} \subset L_{\rho(\phi,k)} \subseteq L_\varphi$. We can see that $\rho(\phi, k) \succ \rho(\varphi, k')$ as

$$\begin{aligned}&\rho(\varphi, k)(\mathbf{x}) \succ \rho(\varphi, k')(\mathbf{x}) = 0 \qquad \mathbf{x} \in \{\mathbf{x}_i : k' < i \leq k\} \\ &\rho(\varphi, k)(\mathbf{x}) = \rho(\varphi, k')(\mathbf{x}) \qquad \text{otherwise}\end{aligned}$$

A17 Zero truncation: Follows from definition.

A18 Time bounded update: Time taken from $K(\phi_1, \phi_1', \phi_2, \phi_2', t)$ does not exceed t units. As we referred to in the axiom definition, this axiom is satisfied operationally by the implementation of the time bound truncated combination $\textsc{update}(\phi_1, \phi_1', \phi_2, \phi_2', t)$.

A19 Monotonic time bounded update: For t, t':

$$\langle \phi, k_1, k_2 \rangle = K(\phi_1, \phi_1', \phi_2, \phi_2', t) \tag{3.20}$$
$$\langle \phi', k_1', k_2' \rangle = K(\phi_1, \phi_1', \phi_2, \phi_2', t') \tag{3.21}$$

First, we note that we refer to k_1, k_2 in the UPDATE algorithm as i, j respectively.

We have to show that the condition $t' \geq t$ implies $k_1' \geq k_1$ and $k_2' \geq k_2$. We note that UPDATE calls UPDATE-EXTEND. We know that any call to UPDATE-EXTEND$(\phi_1, \phi_2, \langle i, j, L_1, L_2, L_3 \rangle, i', j')$ will return $\langle i'', j'', L_1', L_2', L_3' \rangle$ where $i'' \geq i, j'' \geq j$. In UPDATE-EXTEND, the updating of i, j actually happens in line 12 with the call to COMBINE-EXTEND. Starting from the same initial state, if we have $t' \geq t$, the loops in line 5, 13 and 17 of UPDATE, will have possibly more invocations of UPDATE-EXTEND, thus leading to a higher i, j for t' as compared to t. Thus given that $t' \geq t$, we get $k_1' \geq k_1, k_2' \geq k_2$.

A20 Zero time update: For $t = 0$, no loop is entered in the UPDATE algorithm and the initial state is returned, thus $k_1 = k_2 = 0$.

A21 Maximum time for update: If we have no time limit, $timer() > 0$ is always true and can be effectively removed. As valuations in our framework are finite, we can see that UPDATE will take a finite number of steps, giving us a finite time. From the termination conditions of the loops, we get $k_1 = n_1 = |\phi_1'|$ and $k_2 = n_2 = |\phi_2'|$.

□

3.2.1 Instances of semiring induced anytime ordered valuation algebras

As stated earlier, several common instances of valuation algebras are semiring induced: arithmetic potentials, disjunctive normal forms and relational algebras. To show that these instances also support anytime inference, we have to additionally show that the respective inducing semirings are positive, and that we can define a partial order on them.

Before embarking on the proofs, we mention that given existence of a zero element in the semiring $(A, +, \cdot, 0, 1)$, we can always define a *preorder*:

$$a \preceq b \text{ iff } \exists\ c, \text{ s.t. } a + c = b$$

A preorder has two conditions: (i) reflexivity: $a \preceq a$, and (ii) transitivity: $a \preceq b, b \preceq c$ implies $a \preceq c$. The first follows if we set $c = 0$. For the second, we have from the preorder condition $a + d = b, b + e = c$. Substituting, we get $a + (d + e) = c$, and thus $a \preceq c$.

We however need a *partial order*, which imposes the additional condition of antisymmetry: $a \preceq b, b \preceq a$ implies $a = b$. If we define a preorder on the semiring in the manner described above, all we need to do is to show that the preorder is actually a partial order.

► **Remark.** According to def. 3.5, any positive, partially ordered, commutative semiring $(A, +, \cdot, 0, 1)$, along with a set of variables V, and variable frames $\{\Omega_x : x \in V\}$, induces a valuation algebra[14]. This valuation algebra was shown to be an anytime ordered valuation algebra in theorem 3.6. For theorems 3.7 and 3.8, where we show various instances of def. 3.5 are anytime ordered valuation algebras, we then just show that the inducing semiring is positive, partially ordered and commutative. The variables V and the frames are independent of the choice of semiring; each choice of V and frames will give a different semiring induced anytime ordered valuation algebra.

[14] This follows from the fact that a commutative semiring, with V and frames, induce a valuation algebra, shown in theorem 2.1

► **Theorem 3.7.** *Arithmetic potentials when induced by* $(\mathbb{R}^+, +, \cdot, 0, 1)$ *with the* $\preceq$ *preorder relation defined above are an anytime ordered valuation algebra.*

Proof. All we have to show is that $(\mathbb{R}^+, +, \cdot, 0, 1)$ is (i) positive, and (ii) has a partial order. The first immediately follows as $0 \preceq x$, for all $x \in \mathbb{R}^+$. For the second, consider $a \preceq b$ and $b \preceq a$. This gives $a + c = b$ and $b + d = a$, from which we get $a + c + d = a$. As we are working in $\mathbb{R}^+$, we have cancellativity, which we use $c + d = 0$. As $\mathbb{R}^+$ is positive, this gives $c = d = 0$, and we have the partial order. □

► **Theorem 3.8.** *Disjunctive normal forms and relational algebras when induced by* $(\{0, 1\}, +, \cdot, 0, 1)$ *with the* $\preceq$ *preorder relation defined above are an anytime ordered valuation algebra.*

Proof. Disjunctive normal forms and relational algebras are induced by the same semiring $(\{0, 1\}, +, \cdot, 0, 1)$ with $+ = \vee$ and $\cdot = \wedge$ for disjunctive normal forms. We have to show that $(\{0, 1\}, +, \cdot, 0, 1)$ is (i) positive, and (ii) has a partial order. The first immediately follows as $0 \preceq x$, for all $x \in \{0, 1\}$. For the

second, consider $a \preceq b$ and $b \preceq a$. This gives $a + c = b$ and $b + d = a$, from which we get $a + c + d = a$. We can check that the only assignment to c, d that makes it true for all a is $c = d = 0$, and we have the partial order. □

All anytime ordered valuation algebras are also ordered valuation algebras (theorem 3.1), if they have an infimum $\inf(\Psi)$ for all $\Psi \subseteq \Phi_D$. If the inducing semiring has an infimum for all subsets then we can see that

$$\inf(\Psi)(\mathbf{x}) = \inf(\{\varphi(\mathbf{x}) : \varphi \in \Psi\})$$

As the infimum for every subset exists for reals and booleans, the instances of semiring induced anytime ordered valuation algebras discussed here are also ordered valuation algebras.

We shall present a detailed exposition of the application of our framework to these instances in chapter 4.

3.3 Set potential anytime ordered valuation algebra

We discussed the valuation algebra instance of set potentials in section 2.3, and the corresponding ordered valuation algebra instance in section 2.10. Set potentials also utilise the notation of configuration space of a set of variables D, Ω_D, except the valuations are mappings from subsets of the configuration space to the non-negative reals. We shall use the notation for configuration space Ω_D of a set of variables D and the associated notation for tuples from definition 2.2 (semiring induced valuation algebras). In this section we shall define the additional operations that will constitute the set potential anytime ordered valuation algebra. Before the definition, we recall the definition of the natural join $\bowtie$, and the projection operator π from relational algebras, which we used to define combination $\otimes$ and projection operation $\downarrow$ for set potentials back in definition 2.3.

If $R_1 \subseteq \Omega_S, R_2 \subseteq \Omega_T$, the join operation is

$$R_1 \bowtie R_2 := \{\mathbf{x} \in \Omega_{S \cup T} : \mathbf{x}^{\downarrow S} \in R_1, \mathbf{x}^{\downarrow T} \in R_2\}$$

and the projection is, where $T \subseteq S$ here:

$$\pi_T(R) := R^{\downarrow T} := \{\mathbf{x}^{\downarrow T} : \mathbf{x} \in R\}$$

▶ **Definition 3.6.** SET POTENTIAL ANYTIME ORDERED VALUATION ALGEBRAS**.** A set potential anytime ordered valuation algebra is a tuple $\langle \Phi, V, d, \otimes, \downarrow, \succeq, K, \oplus, \rho \rangle$, where $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a set potential valuation algebra (def. 2.3)

We can consider a set potential valuation ϕ with domain D, i.e. $\phi : 2^{\Omega_D} \to \mathbb{R}^+$ as being represented by a list $L_\phi = \langle (A_1, \phi(A_1)) \dots (A_n, \phi(A_n)) \rangle$, where $A_i \subseteq \Omega_D$ and we only keep the sets which do not map to 0. For convenience, we shall use the notation $S(D) := 2^{2^{\Omega_D} \times \mathbb{R}^+}$. These sets, which have a non-zero mapping are known as *focal sets*. We order the tuples of L_ϕ by the values $\varphi(A_i)$, from highest to lowest. Then we have the following:

Partial order Valuations having the same domain D form a partially ordered set $\Phi_D \subseteq \Phi$. For two valuations $\phi, \phi' \in \Phi_D$,

$$\phi \preceq \phi' \text{ iff } \phi(A) \leq \phi'(A) \quad \forall\ A \subseteq \Omega_D \tag{3.22}$$

Composition The composition operation ($\oplus : \Phi_D \times \Phi_D \to \Phi_D$) is defined as follows:

$$(\phi \oplus \psi)(\mathbf{x}) = \phi(A) + \psi(A), \quad \forall\ A \subseteq \Omega_D \tag{3.23}$$

Truncation The truncation function $\rho : \Phi_D \times \mathbb{N} \to \Phi_D$ is defined as follows:

$$L_{\rho(\phi,k)} = \langle (A_1, \phi(A_1)) \dots (A_m, \phi(A_m)) \rangle \quad m = \min(k, n),\ n = |L_\phi| \tag{3.24}$$

Thus the truncated valuation corresponds the first k elements of L_φ. We also get the size of a valuation $|\phi| := |L_\phi| = n$. The inverse truncation is similarly defined as

$$L_{\bar{\rho}(\phi,k)} = \langle (\mathbf{x}_{m+1}, \phi(A_{m+1})) \dots (\mathbf{x}_n, \phi(A_n)) \rangle \tag{3.25}$$

Time-bound truncated update The time-bounded truncated update operation is nearly identical to that of semiring induced anytime ordered valuation algebras (def 3.5). The time bound truncated update is defined as an algorithm: $K(\phi_1, \phi_1', \phi_2, \phi_2', t) := \textsc{update}(\phi_1, \phi_1', \phi_2, \phi_2', t)$. We also define the secondary methods:

- $\textsc{insert}(\phi_1 : \Phi_A,\ \phi_2 : \Phi_B,\ i, j : \mathbb{N},\ L : S(A \cup B))$. This method inserts a combination into the list representation of the set potential.

- COMBINE-EXTEND$(\phi_1 : \Phi_A,\ \phi_2 : \Phi_B, \langle i, j : \mathbb{N},\ L : S(A \cup B)\rangle, i', j' : \mathbb{N}) \to \mathbb{N} \times \mathbb{N} \times (S(A \cup B))$.
  This method incrementally adds combinations into the list representation and updates the state, going from the state $\rho(\phi_1, i) \otimes \rho(\phi_2, j)$ to $\rho(\phi_1, i + i') \otimes \rho(\phi_2, j + j')$.
- UPDATE-EXTEND$(\phi_1, \phi_1' : \Phi_A,\ \phi_2, \phi_2' : \Phi_B, \langle i, j : \mathbb{N},\ L_1, L_2, L_3 : S(A \cup B)\rangle, i', j' : \mathbb{N}) \to \mathbb{N} \times \mathbb{N} \times S(A \cup B) \times S(A \cup B) \times S(A \cup B)$.
  This method updates the state, effectively going from

$$(\rho(\phi_1', i) \otimes \phi_2) \oplus (\phi_1 \otimes \rho(\phi_2', j)) \oplus (\rho(\phi_1', i) \otimes \rho(\phi_2', j)) \quad \text{initial state, to}$$
$$(\rho(\phi_1', i + i') \otimes \phi_2) \oplus (\phi_1 \otimes \rho(\phi_2', j + j')) \oplus (\rho(\phi_1', i + i') \otimes \rho(\phi_2', j + j')) \quad \text{final state}$$

  where L_1, L_2, L_3 are the lists corresponding to the valuations being composed by $\oplus$.
- UPDATE$(\phi_1, \phi_1' : \Phi_A,\ \phi_2, \phi_2' : \Phi_B, t : \mathbb{N}) \to \phi_{A \cup B} \times \mathbb{N} \times \mathbb{N}$ is the implementation of the time bound update operation K.

Out of the algorithms mentioned above, only INSERT needs to be changed for set potentials; the rest are identical to those in definition 3.5.

1: **function** INSERT$(\phi_1, \phi_2, i, j, L)$
2: insert $[A_i \bowtie A_j, \phi_1(A_i) \cdot \phi_2(A_j)]$ into L.
3: **end function**

▶ **Theorem 3.9.** *The structure introduced in def. 3.6 satisfies the axioms of an anytime ordered valuation algebra.*

Proof. To show that def. 3.6 is an anytime ordered valuation algebra, we have to show *A7 – A21* hold.

A7 Partial Order As in the definition.

A8 Null element For a domain D, the null element is the set potential with sets all subsets of the configuration space Ω_D to 0: $n_D(A) = 0\ \forall\ A \subseteq \Omega_D$. It can be easily seen that this is also the least element in Ω_D.

Then we have to show

1. $n_D \otimes \phi = n_{D \cup A}$, where $\phi \in \Omega_A$. For all $A \subseteq \Omega_{D \cup A}$:

$$(n_D \otimes \phi)(A) = \sum_{B \bowtie C = A} n_D(B) \cdot \phi(C) = 0$$

Thus $n_D \otimes \phi = n_{D \cup A}$

2. $n_D^{\downarrow D'} = n_{D'}$. For all $A \subseteq \Omega_{D'}$

$$n_D^{\downarrow D'}(A) = \sum_{\pi_{D'}(B)=A} n_D(B) = 0$$

Thus $n_D^{\downarrow D'} = n_{D'}$.

A9 Combination preserves partial order Let $d(\phi) = d(\phi') = S$ and $d(\psi) = d(\psi') = T$. We have to show that if $\phi \preceq \phi', \psi \preceq \psi'$, then $\phi \otimes \psi \preceq \phi' \otimes \psi'$.

For $A \subseteq \Omega_{S \cup T}$

$$\begin{aligned}(\phi \otimes \psi)(A) &= \sum_{B \bowtie C = A} \phi(B) \cdot \psi(C) \\ &\leq \sum_{B \bowtie C = A} \phi'(B) \cdot \psi'(C) \\ &= (\phi' \otimes \psi')(A)\end{aligned}$$

From the definition of $\preceq$, we get $\phi \otimes \psi \preceq \phi' \otimes \psi'$.

A10 Projection preserves partial order Let $d(\phi) = d(\phi') = S$ and $D \subseteq S$. We have to show that if $\phi \preceq \phi'$, then $\phi^{\downarrow D} \preceq \phi'^{\downarrow D}$. This follows simply:

$$\phi^{\downarrow D}(A) = \sum_{\pi_D(B)=A} \phi(B) \leq \sum_{\pi_D(B)=A} \phi'(B) = \phi'^{\downarrow D}(A)$$

A11 Composition preserves partial order: We have to show that if $\psi, \phi, \phi' \in \Phi$ are valuations such that $\phi \succeq \phi'$, then $\psi \oplus \phi \succeq \psi \oplus \phi'$. Let $d(\phi) = d(\phi') = d(\psi) = D$, as is required by *A7*. Then from $\phi \succeq \phi'$, we have $\phi(Z) \geq \phi'(Z)$ for all $Z \subseteq \Omega_D$. As $\geq$ is monotone with respect to addition, we have $\psi(Z) + \phi(Z) \geq \psi(Z) + \phi'(Z)$. As this is a point-wise expression that is true for all $Z \subseteq \Omega_D$, we have $\psi \oplus \phi \succeq \psi \oplus \phi'$.

A12 Composition forms a monoid: We recall that the composition operation ($\oplus : \Phi_D \times \Phi_D \to \Phi_D$) is defined as follows:

$$(\phi \oplus \psi)(A) = \phi(A) + \psi(A), \quad \forall\ A \subseteq \Omega_D \tag{3.26}$$

Here + is arithmetic addition which is associative and commutative, causing $\oplus$ to be so as well. The identity for $\oplus$ is the neutral element of the domain D, n_D.

A13 Combination $\otimes$ distributes over composition $\oplus$: We have to show that

$$(\phi \oplus \phi') \otimes \psi = (\phi \otimes \psi) \oplus (\phi' \otimes \psi) \tag{3.27}$$

Let $d(\phi) = d(\phi') = S$ and $d(\psi) = T$. From the definition of $\oplus$, we get $d(\phi \oplus \phi') = S$. Then, we have for $A \subseteq \Omega_{S \cup T}$, where $\Omega_{S \cup T}$ is the configuration space associated with $S \cup T$:

$$\begin{aligned}
((\phi \oplus \phi') \otimes \psi)(A) &= \sum_{B \bowtie C = A} (\phi \oplus \phi')(B) \cdot \psi(C) \\
&= \sum_{B \bowtie C = A} (\phi(B) + \phi'(B)) \cdot \psi(C) \\
&= \sum_{B \bowtie C = A} \phi(B) \cdot \psi(C) + \sum_{B \bowtie C = A} \phi'(B) \cdot \psi(C) \\
&= (\phi \otimes \psi)(A) \oplus (\phi' \otimes \psi)(A) \\
&= ((\phi \otimes \psi) \oplus (\phi' \otimes \psi))(A)
\end{aligned}$$

which shows the equality.

A14 Projection $\downarrow$ distributes over composition $\oplus$: We have to show that, for $d(\phi) = d(\phi') = S$ and $D \subseteq d(\phi)$:

$$(\phi \oplus \phi')^{\downarrow D} = \phi^{\downarrow D} \oplus \phi'^{\downarrow D} \tag{3.28}$$

Then we have

$$\begin{aligned}
((\phi \oplus \phi')^{\downarrow D})(A) &= \sum_{\pi_D(B) = A} (\phi \oplus \phi')(B) \\
&= \sum_{\pi_D(B) = A} \phi(B) + \sum_{\pi_D(B) = A} \phi'(B) \\
&= \phi^{\downarrow D}(A) + \phi'^{\downarrow D}(A) \\
&= (\phi^{\downarrow D} \oplus \phi'^{\downarrow D})(A)
\end{aligned}$$

which shows the equality.

A16 Truncation monotonically increasing: We have to show that $\rho(\phi, k) \succ \rho(\phi, k')$ for all $k > k'$, and $k, k' \leq |\varphi|$. For the set potential $\phi : 2^{\Omega}_{D} \to \mathbb{R}^{+}$ we defined $\rho(\phi, k)$ as the valuation corresponding to

$$L_{\rho(\phi,k)} = \langle (A_1, \phi(A_1)) \dots (A_m, \phi(A_m)) \rangle \quad m = \min(k, |L_\varphi|)$$

As $k, k' \leq |\varphi| = |L_\varphi|$, we have $L_{\rho(\phi,k')} \subset L_{\rho(\phi,k)} \subseteq L_\varphi$. We can

see that $\rho(\phi, k) \succ \rho(\varphi, k')$ as

$$\begin{aligned} &\rho(\varphi,k)(A) > \rho(\varphi,k')(A) = 0 \qquad A \in \{A_i : k' < i \leq k\} \\ &\rho(\varphi,k)(A) = \rho(\varphi,k')(A) \qquad\qquad \text{otherwise} \end{aligned}$$

A17 Zero truncation: Follows from definition.

The axioms *A17* – *A21* have the same proof as that for semiring induced valuation algebras, except for the operation INSERT which was defined previously.

□

3.4 Outward propagation

In the earlier sections we considered only inward propagation, which produces the marginal of the combined joint valuation at the root node. A refinement algorithm was also introduced which would improve the accuracy of the marginal at the root node incrementally, converging to the exact inference solution after a finite number of steps. We did not consider the outward propagation process which generally follows the inward propagation process to produce the marginals at all the nodes of the join tree.

The outward propagation algorithm involves sending messages from the root towards the leaves. Each node combines the messages from its parent and its siblings (all nodes except its own children) and combines them to get the marginal at the node. For binary join trees, the OUTWARD propagation is then identical to the outward propagation algorithm presented in section 2.7.3. The only difference in the anytime inference case is that the messages ϕ_r from the parent nodes and ϕ_s from the sibling nodes are approximations to the exact valuations thus giving approximate marginals at the interior nodes. An outward propagation following a approximate inward propagation is thus automatically approximate as well. There can be several such inward-outward propagations before convergence to the exact marginals.

3.5 Conclusion

In this chapter we have introduced the framework of anytime ordered valuation algebras. The significance of anytime

ordered valuation algebras is that it allows us to perform approximate inference in the valuation algebra framework while having the ability to improve upon the approximation if required. This process of refinement can be repeated by the user, and the approximation is guaranteed to improve with time. This thus allows fine-grained control over the progress of the inference process.

On the other hand, the genericity of the framework allows its immediate application to a wide variety of instances, including the important class of semiring induced valuation algebras as well as set potentials. We also prove soundness and completeness of the anytime inference algorithm. In the next chapter, we shall consider instances of anytime ordered valuation algebras in more detail, with emphasis on concrete examples.

Chapter 4

Instances of Anytime Inference in Valuation Algebras

In this chapter we describe some instances of the anytime ordered valuation algebra introduced in chapter 3. For each instance, we give a general introduction and map it to the anytime ordered valuation algebra, followed by application to a concrete example which illustrates anytime inference.

The instances considered are: Bayesian networks (section 4.1.1), including Bayesian and Markov networks, disjunctive normal forms (section 4.1.2), relational algebras (section 4.1.3) and set potentials (section 4.2). Section 4.3 concludes with a summary of the instances discussed.

4.1 Semiring induced anytime ordered valuation algebras

4.1.1 Arithmetic potentials – Bayesian networks

Probabilistic graphical models express the dependence of variables in probability distributions. Various common statistical models like Hidden Markov Models and Ising models can be described as probabilistic graphical models [Koller et al., 2007]. They are one of the most useful valuation algebra instances with a wide range of applications, ranging from engineering, biomedical sciences, image processing and natural language processing. Probabilistic graphical models can be broadly divided into undirected and directed graphical models. Directed graphical models, also known

as Bayesian networks are suited for probability distributions which can be decomposed into constituent (conditional) probability distributions with a causal relationship between them; whereas undirected graphical models, also known as Markov networks are suited for data which are not causally, perhaps spatially related, as in image processing algorithms. In this section we shall focus on Bayesian networks.

▶ **Definition 4.1.** PROBABILISTIC GRAPHICAL MODEL. A probabilistic graphical model is a graph $G = (\mathcal{V}, \mathcal{E})$ representation of a joint probability distribution over random variables $v \in \mathcal{V}$, where edges $(v, w) \in \mathcal{E}$ represent dependencies between variables v and w in the probability distribution.

▶ **Definition 4.2.** BAYESIAN NETWORK. Bayesian networks are probabilistic graphical models where G is a directed acyclic graph (DAG). The edges in the DAG represent conditional independence criteria.

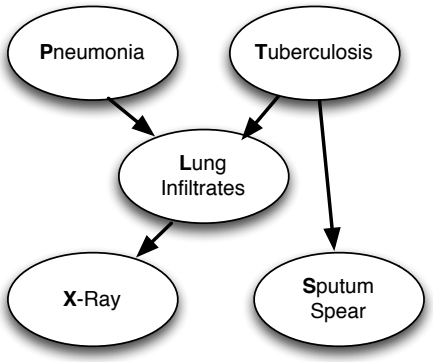


Figure 4.1: Bayesian network showing causes and effects of lung infiltrates

A way of intuitively understanding Bayesian networks is that of effects and causes. The probability that a particular effect $E = e$ was caused by a particular cause $C = c$ can be represented as $P(E = e|C = c)$. We represent the relation between effect and cause by a directed arrow from the cause to the effect.

Another way of viewing Bayesian networks is as a representation of the factorized joint probability distribution over all the variables depicted in the graph. In a graph over the variables $X_1 \dots X_n$, each variable X_i has a conditional probability distribution of $P(X_i|\mathbf{Pa}_{X_i})$, where $\mathbf{Pa}_{X_i}$ are the parents of X_i in the graph.

An example of a Bayesian network is shown in figure 4.1, depicting the causes and effects of lung infiltrates. The arrows

indicate causal direction: lung infiltrates (L) can be caused by either or both of pneumonia (P) and tuberculosis (T), which will show up in an X-ray (X). If it is indeed tuberculosis (T) that is the cause of lung infiltrates there is a probability that it will show up in a sputum spear (S). The entire joint probability distribution can be factorized as

$$P(P, T, L, X, S) = P(P)P(T)P(L \mid P, T)P(X \mid L)P(S \mid T)$$

Thus a Bayesian network is one where we can factorise the joint probability distribution as

$$P(X_1, \dots X_n) = \prod_{i=1}^{n} P(X_i | \mathbf{Pa}_{X_i}) \tag{4.1}$$

Inference in Bayesian networks involves finding the solution to the inference problem, which is the projection of the joint probability distribution to a query of interest. First, we need to create a binary join tree. We start by creating a valuation for each conditional probability distribution and use the join tree construction process from section 2.7.1. The inward propagation algorithm (section 3.1.1) then gives the solution to the inference problem.

► **Example 4.1.** We consider the ASIA network (fig. 4.2) from [Lauritzen and Spiegelhalter, 1990] which depicts a causal network which could be used to diagnose patients in a clinic. All the variables are binary; if the condition represented by the variable is true, it is set to 1. The variables are as follows

A	visited Asia	S	smokes
T	tuberculosis	L	lung cancer
E	either tuberculosis or cancer	B	bronchitis
X	abnormal X-ray result	D	dysponea

The network shows the causal relationships, such as smoking can cause both lung cancer and bronchitis. Bronchitis usually won't show up in an X-ray result but tuberculosis or lung cancer will, and this is represented in the connections between variables.

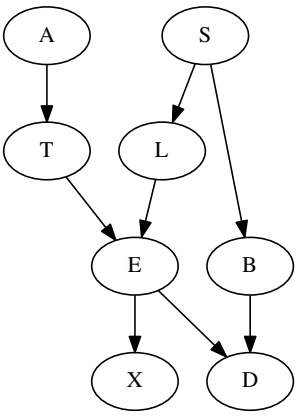


Figure 4.2: The *asia* Bayesian network

We choose a query of $\{A, B, D\}$ which represent the variables of whether a patient has (i) visited Asia (ii) bronchitis and (iii) dysponea. Once we have constructed the binary join tree for the inference problem (fig. 4.4), we can perform anytime inference using the algorithms discussed in chapter 3. The valuations of the inference problem in this case directly

correspond to the conditional probability distributions (c.p.d.) which form the factors of the joint probability distribution $P(A, T, S, L, B, E, X, D)$. Due to the direct correspondence between valuations and the conditional probability distributions, we denote the valuation corresponding to a c.p.d. by the notation $\phi_{N_1,N_2,\ldots|M_1.M_2,\ldots}$ where $P(N_1, N_2, \ldots|M_1, M_2, \ldots)$ is the corresponding c.p.d. Then the inference problem is

$$(\phi_A \otimes \phi_{T|A} \otimes \phi_s \otimes \phi_{L|S} \otimes \phi_{B|S} \otimes \phi_{E|L,T} \otimes \phi_{X|E} \otimes \phi_{D|B,E})^{\downarrow\{A,B,D\}}$$

The valuations (probability distributions) are shown in table 4.1.

The inward propagation process starts at the leaves of the join tree. If we consider the leaf valuations $\phi_{T|A}$ and $\phi_{E|L,T}$ with insufficient time allocated for exact combination, we get a subset of the full configuration space. The progress of anytime inference is shown in table 4.2 by comparing the valuation at the root node (which is the solution to the inference problem) under successive refinements in table 4.2. As expected, incompleteness $\epsilon(t)$ decreases with time (fig. 4.3), where $D = d(\varphi)$ and φ_t denotes the valuation at the root node at time t:

$$\epsilon(t) = 1 - \sum_{\mathbf{x}\in\Omega_D} \varphi_t(\mathbf{x}) / \sum_{\mathbf{x}\in\Omega_D} \varphi(\mathbf{x}) \tag{4.2}$$

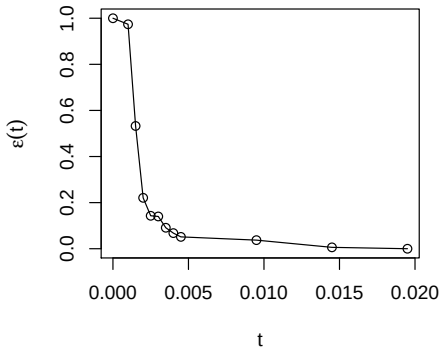


Figure 4.3: Degree of incompleteness of the inference solution with time, in the *asia* Bayesian network

► **Remark.** When we were defining semiring induced anytime ordered valuation algebras (def. 2.12), we stated

> If the inducing semiring also has a total order, then we order the tuples of L_ϕ by the semiring values $\varphi(\mathbf{x}_i)$, from highest to lowest. We do this to ensure that configuration with the highest "mass" get combined first.

Here L_φ is, as we recall, the list representation of φ. If we look at fig. 4.3, we see that the rate of progress of anytime inference is greatest at the beginning, and plateaus after a point. This is not an accident. Combining the configurations with the highest weight first ensures that the total mass of the arithmetic potential is as large as early as possible, leading more quickly to a lower degree of incompleteness.

A	$p(A)$
1	0.99
0	0.01

S	$p(S)$
0	0.5
1	0.5

A	T	$p(A,T)$
1	1	0.99
0	1	0.95
0	0	0.05
1	0	0.01

S	L	$p(S,L)$
1	1	0.99
0	1	0.9
0	0	0.1
1	0	0.01

S	B	$p(S,B)$
1	1	0.7
0	0	0.6
0	1	0.4
1	0	0.3

E	X	$p(E,X)$
0	0	0.98
1	1	0.95
1	0	0.05
0	1	0.02

L	T	E	$p(L,T,E)$
0	0	0	1.0
0	1	0	1.0
1	0	0	1.0
1	1	1	1.0
0	0	1	0.0
0	1	1	0.0
1	0	1	0.0
1	1	0	0.0

B	E	D	$p(B,E,D)$
0	0	0	0.9
1	1	1	0.9
0	1	0	0.8
1	0	0	0.7
1	0	1	0.3
0	1	1	0.2
0	0	1	0.1
1	1	0	0.1

Table 4.1: Probability potentials for the *asia* network.

Table 4.2: Anytime inference in the *asia* Bayesian network, showing the incompleteness and the valuation at the root node, which is the solution to the inference problem.

t	0.0015	0.0025	0.0035	0.0045	0.0195
incompleteness	0.5326	0.1429	0.0910	0.0513	0.0000
$\{A,B,D\}$					
1 1 1	0.4412	0.4412	0.4412	0.4644	0.4730
1 0 0	0.0259	0.3377	0.3377	0.3540	0.3599
1 0 1		0.0779	0.0808	0.0808	0.0855
1 1 0			0.0490	0.0490	0.0715
0 1 1					0.0046
0 0 0	0.0002	0.0002	0.0002	0.0002	0.0036
0 1 0					0.0008
0 0 1					0.0008

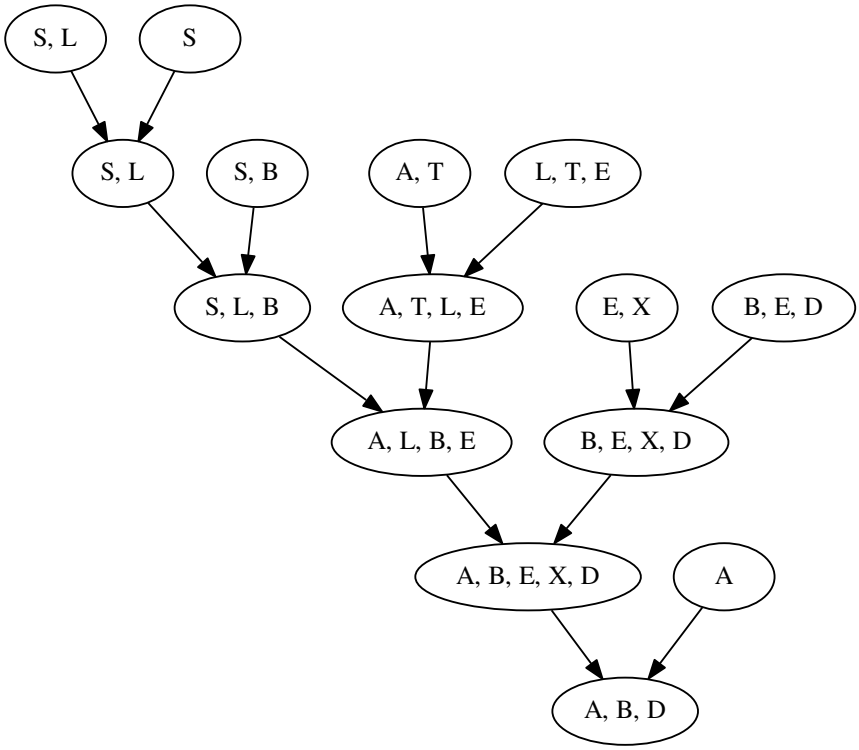


Figure 4.4: Binary join tree for the *asia* Bayesian network

4.1.2 Disjunctive normal forms

We gave a brief introduction to the application of valuation algebras in propositional logic in section 2.2.2. Here we shall consider them in more detail, particularly the valuation algebra of disjunctive normal forms which is an anytime ordered semiring induced valuation algebra (theorem 3.8).

We shall use the same notation in this section as that in section 2.2.2. Briefly, the language of propositional logic contains literals, which can be either variables ($x, y, \ldots$) or their negation $\neg x, \neg y, \ldots$. Any combination of such literals with the binary connectives of disjunction/logical-or ($\vee$) or conjunction/logical-and ($\vee$) is a valid propositional formula. In propositional logic, all functions are boolean, taking values true ($\top$) or false ($\bot$). Conventionally these are also indicated by 1 and 0 respectively. The valuation corresponding to a formula δ is $[\delta, D]$ where D is the domain of the formula.

For disjunctive normal forms, we may choose to perform inference on the model or on the formula itself. Both representations can be expressed in the valuation algebra framework [Pouly and Kohlas, 2011, Chapter 7]. We always work with full disjunctive normal forms (every literal or its negation appears in every term), which have a bijective correspondence to the model. When representing, we show the minimised, logically equivalent version.

We now consider what anytime inference means for propositional logic. In the previous instance of Bayesian networks, anytime inference was depicted as computing chunks of the (unnormalised) probability distribution, with the inference algorithm converging to the exact inference answer eventually. In the case of probability distributions, anytime inference is particularly useful since we may only be interested in the region of the probability distribution with the greatest area under curve, which is always computed first.

In the case of propositional logic, the notion of anytime inference is less immediately intuitive. Our method for anytime, approximate inference in disjunctive normal forms can be interpreted as an anytime inference on the model, where the anytime inference algorithm converges on the exact model over time, gradually computing the truth assignments of the model. As each model has an associated formula, this becomes an anytime inference algorithm on formulae as well,

converging to the exact formula eventually. We can think of each term in the disjunctive normal form as a state or a choice that makes the statement true. Then with anytime inference we are gradually uncovering more states or choices.

Now we consider anytime inference in a well-known example from the literature, that of the *alarm* network described in [Pearl, 1988]. We review the example and show how anytime inference proceeds.

► **Example 4.2.** The inference problem for the alarm system uses clauses to describe the various pathways by which an alarm in a house can be activated. The interpretations of the propositional symbols are given below, followed by the clauses.

a: the alarm system in the house of Mr Holmes is ringing,
b: there is a burglary,
e: an earthquake has occurred,
c: there is confirmation of the earthquake on the radio,
w: the neighbour of Mr Holmes, Mr Watson phones Mr Holmes,
g: the neighbour, Mrs Gibson phones Mr Holmes,
d: the daughter of Mr Holmes phones.

$$\xi_1 = b \wedge a_1 \rightarrow a \qquad \xi_{10} = a_4 \rightarrow \neg g$$
$$\xi_2 = a \wedge a_1 \rightarrow a \qquad \xi_{11} = \neg a \rightarrow \neg g$$
$$\xi_3 = a_2 \rightarrow a \qquad \xi_{12} = a \wedge a_5 \rightarrow d$$
$$\xi_4 = \neg b \wedge \neg e \wedge \neg a_2 \rightarrow \neg a \qquad \xi_{13} = \neg a_5 \rightarrow \neg d$$
$$\xi_5 = \neg a \wedge \neg a_2 \rightarrow \neg a \qquad \xi_{14} = \neg a \rightarrow \neg d$$
$$\xi_6 = a \rightarrow w \qquad \xi_{15} = e \wedge a_6 \rightarrow c$$
$$\xi_7 = a_3 \rightarrow w \qquad \xi_{16} = \neg e \rightarrow \neg c$$
$$\xi_8 = \neg a \wedge \neg a_3 \rightarrow \neg w \qquad \xi_{17} = \neg a_6 \rightarrow \neg c$$
$$\xi_9 = a \wedge \neg a_4 \rightarrow g$$

The variables in the knowledgebase are

$$\{a, b, c, d, e, g, w, a_1, a_2, a_3, a_4, a_5, a_6\}$$

The rules tell us the various causes ($a_1 - a_6$) for an alarm to ring (a). They denote the following:

$\xi_1 - \xi_5$ A burglary generates alarm in Mr. Holmes' house, if the alarm is functioning (a_1). The alarm could also be caused by an earthquake, or other causes (a_2). These are the only ways an alarm can arise (ξ_4, ξ_5).

$\xi_6 - \xi_8$ If there is an alarm, Mr. Watson, who is Mr. Holmes' neighbour phones him, though he may also phone as a joke (a_3).

$\xi_9 - \xi_{11}$ Thankfully, Mr. Holmes has another neighbour, Mrs. Gibson. However she can only phone him if there is an alarm and she is able to hear it ($\bar{a}_4$).

$\xi_{15} - \xi_{17}$ If Mr. Holmes' daughter is at home then she phones if there is an alarm (a_5). Also, if there was an earthquake there would be confirmation on the radio if the earthquake was recorded (a_6).

$\xi_{18} - \xi_{20}$ We may also incorporate some evidence, such as (i) Mr. Watson phones ($\xi_{18} = w$), (ii) Mrs. Gibson does not phone ($\xi_{19} = \neg g$) and (iii) there is no confirmation of an earthquake on the radio ($\xi_{20} = \neg c$).

We now consider certain variables we are interested in: namely, whether the alarm was raised (a), whether there was a burglary (b) or earthquake (e). We may also be interested in the various reporting methods for an alarm ($a_1 \dots a_6$) giving us a query of $x = \{a, b, e, a_1, a_2, a_3, a_4, a_5, a_6\}$. A binary join tree representing the inference problem is shown in fig. 4.6. The labels (domains) for the leaf nodes (the valuations $\xi_1 \dots \xi_{20}$) are shown alongside the nodes.

Performing the exact inference with the query x yields:

$$a_3\bar{b}\bar{e}\bar{a_2}\bar{a} + \bar{a_1}\bar{e}\bar{a_2}a_3\bar{a} + \bar{a_1}\bar{a_6}\bar{a_2}a_3\bar{a} + ba_1aa_4\bar{e} + ba_1a\bar{a_6}a_4 + a_1a\bar{a_6}ea_4 + a_2aa_4\bar{e} + a_2a\bar{a_6}a_4$$

which is equivalent to the CNF form [Kohlas et al., 1999, Section 2.1]:

$$(\bar{b} + \bar{a_1} + a)(\bar{e} + \bar{a_1} + a)(\bar{a_2} + a)(b + e + a_2 + \bar{a})(a + a_2 + \bar{a})(a + a_3)(\bar{a} + a_4)(\bar{e} + \bar{a_6}) \tag{4.3}$$

By substituting further observables in eq. 4.3, we can see which conditions must be true. The conditions are precisely the set of assignments to the remaining variables which render the formula true, i.e. the model. For example with $a = 1, a_5 = 1$ (the alarm rang and the alarm system is working), we get

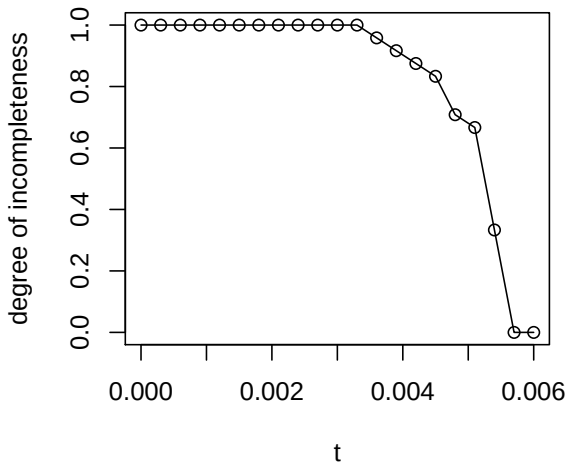
$$a_4((a_2 + b)(\bar{e} + \bar{a_6}) + \bar{a_6}e) \tag{4.4}$$

The model of the above formula can be interpreted as follows:

Mrs. Gibson hasn't heard the alarm (a_4), since she would have called if she had. Also, the alarm could be caused by (i) no earthquake ($\bar{e}$) or no confirmation (a_6), and either a burglary happened (b) or other causes triggered the alarm (a_2) (ii) an earthquake did happen (e), but without confirmation ($\bar{a}_6$). When we perform anytime inference, we progressively obtain the full model. For example, for $t = 0.0051$, we obtain $a_4(b(\bar{e} + \bar{a_6}) + \bar{a_6}e)$, which does not consider the case where the alarm could have been caused from a_2.

To show progress of anytime inference in this instance (fig. 4.5), we can define a degree of incompleteness $\epsilon(\delta') = 1 - |M_D(\delta')|/|M_D(\delta)|$, where δ' is an approximation of δ ($\delta \preceq \delta'$).

Figure 4.5: Progress of anytime inference for the *alarm* instance.



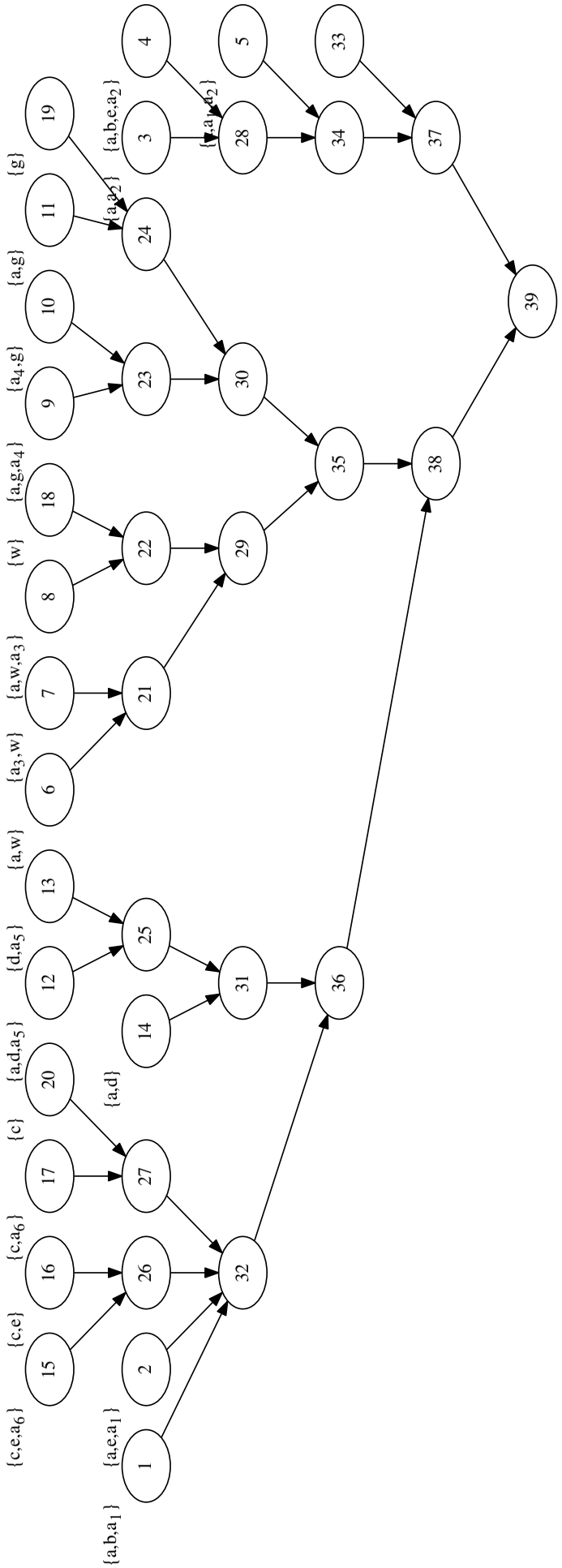


Figure 4.6: Binary join tree for the *alarm* instance

4.1.3 Relational algebras

We reviewed the application of the valuation algebra framework to the instance of relational algebras in sections 2.2.3 and 2.5.3. As relational algebras are also a semiring induced valuation algebra, with the semiring $(\{0,1\},+,\cdot)$, they form an anytime ordered valuation algebra automatically (theorem 3.8). Relational algebras can be considered as a semiring induced valuation algebra as follows:

$$R = \{\mathbf{x} | \phi_R(\mathbf{x}) = 1,\ \mathbf{x} \in \Omega_{d(\phi_R)}\} \tag{4.5}$$

Thus the relation R is the set of tuples which are mapped by the corresponding semiring induced valuation ϕ_R to 1.

We note that relational algebras are induced by the same boolean semiring as disjunctive normal forms. Just like in the case of disjunctive normal forms, we uncovered more of the model as the inference progressed, here we uncover more tuples in the relation as inference progresses.

Thus the anytime inference algorithm in the case of relational algebras produces a partial view of the database table for the selected query (which is the domain of the root node in the constructed binary join tree as in the other instances). As the uncombined portions of the relations are cached, the anytime inference algorithm can incrementally produce new records of the query on demand. The interpretation of anytime inference here is thus trivial and we do not consider a specific example.

4.2 Set potential anytime ordered valuation algebra

As we recall from section 2.3, set potentials are unnormalised belief potentials, which are a generalisation of probability potentials to subsets of the configuration space in Dempster-Shafer's theory of evidence [Shafer et al., 1976]. We discussed the valuation algebra instance of set potentials with the respective combination and projection rules in section 2.3. Set potentials considered as an anytime ordered valuation algebra was discussed in section 3.3.

► **Example 4.3.** We reconsider the example of ploxoma considered in section 2.6 for illustration of anytime inference.

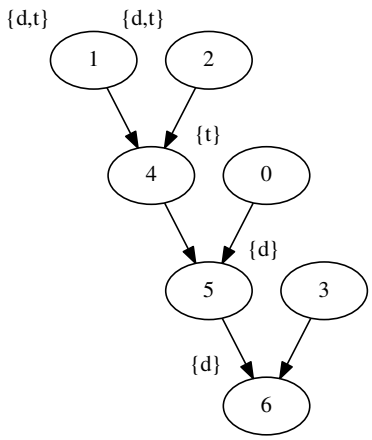


Figure 4.7: Binary join tree for the *ploxoma* inference problem

To briefly summarise, there are two forms of a disease, ordinary ploxoma (θ_2) and virulent ploxoma (θ_1). Ploxoma can be detected by one of three tests, represented by x_1, x_2, x_3. Thus the frame of discernment (the set of possible outcomes in the configuration space) is $d \times t$ where $d = \{\theta_1, \theta_2\}$ and $t = \{x_1, x_2, x_3\}$. The following set potentials are represented in the problem (reproduced from section 2.6 for ease of reference)

	domain	focal sets	mass
m_1	$\{d,t\}$	$m_{10} = \{(\theta_1,x_1),(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.2
		$m_{11} = \{(\theta_1,x_2),(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.2
		$m_{12} = \{(\theta_1,x_3),(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.6
m_2	$\{d,t\}$	$m_{20} = \{(\theta_2,x_1),(\theta_1,x_1),(\theta_1,x_2),(\theta_1,x_3)\}$	$0.85 \cdot 0.75$
		$m_{21} = \{(\theta_2,x_2),(\theta_2,x_3),(\theta_1,x_1),(\theta_1,x_2),(\theta_1,x_3)\}$	$0.15 \cdot 0.75$
		$m_{22} = \{(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3),(\theta_1,x_1),(\theta_1,x_2),(\theta_1,x_3)\}$	0.25
m_3	$\{d\}$	$m_{30} = \{\theta_1\}$	0.05
		$m_{31} = \{\theta_2\}$	0.85
		$m_{32} = \{\theta_1,\theta_2\}$	0.10
m_0	$\{t\}$	$m_{00} = \{x_1\}$	1.0

Table 4.3: Belief functions in the *ploxoma* inference problem

We can now consider anytime inference by considering the join tree (figure 4.7) for the inference problem

$$(m_0 \otimes m_1 \otimes m_2 \otimes m_3)^{\downarrow\{d\}} \tag{4.6}$$

Owing to the simplicity of the example (4 set potentials, 2 variables), we can give details of the inference process. The first step is computing $m_4 = m_1 \otimes_t m_2$, with $d(m_4) = \{d,t\}$. We show the table for the exact computation; we shall consider truncated versions of m_4 to illustrate anytime inference.

Table 4.4: $m_4 = m_1 \otimes m_2$

	focal set	mass
$m_{12} \cap m_{20}$	$\{(\theta_1,x_3),(\theta_2,x_1)\}$	0.3825
$m_{11} \cap m_{20}, m_{10} \cap m_{20}$	$\{(\theta_1,x_1),(\theta_2,x_1)\}$	0.255
$m_{12} \cap m_{22}$	$\{(\theta_1,x_3),(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.15
$m_{12} \cap m_{21}$	$\{(\theta_1,x_3),(\theta_2,x_2),(\theta_2,x_3)\}$	0.0675
$m_{11} \cap m_{22}$	$\{(\theta_1,x_2),(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.05
$m_{10} \cap m_{22}$	$\{(\theta_1,x_1),(\theta_2,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.05
$m_{11} \cap m_{21}$	$\{(\theta_1,x_2),(\theta_2,x_2),(\theta_2,x_3)\}$	0.0225
$m_{10} \cap m_{21}$	$\{(\theta_1,x_1),(\theta_2,x_2),(\theta_2,x_3)\}$	0.0225

Next we combine m_0 which is the test result (x_1), giving us m_5. We also show $m_5^{\downarrow\{d\}}$ which will be combined with m_3, the

incidence of virulent and ordinary ploxoma in the population (somewhat like a Bayesian prior).

focal set (m_5)	focal set ($m_5^{\downarrow\{d\}}$)	mass
$\emptyset$	$\emptyset$	0.09
$\{(\theta_1, x_1)\}$	$\{\theta_1\}$	0.0225
$\{(\theta_2, x_1)\}$	$\{\theta_2\}$	0.5825
$\{(\theta_1, x_1), (\theta_2, x_1)\}$	$\{\theta_1, \theta_2\}$	0.305

Table 4.5: $m_5 = m_0 \otimes m_4$

Finally we reach the root node, the solution to the inference problem $m_6 = (m_0 \otimes m_1 \otimes m_2 \otimes m_3)^{\downarrow\{d\}} = m_5^{\downarrow\{d\}} \otimes m_3$:

focal set	mass (m_5)	mass (m_3)	mass (m_6)
$\emptyset$	0.09	0.0	0.138 25
$\{\theta_1\}$	0.0215	0.05	0.018 625
$\{\theta_2\}$	0.5825	0.85	0.812 625
$\{\theta_1, \theta_2\}$	0.305	0.10	0.0305

Table 4.6: Solution to the *ploxoma* inference problem

By considering truncations of the m_5 valuation (table 4.4), we can obtain approximations to the exact inference problem at the root node m_6. As the other intermediate valuation m_5 has a single binary variable, in most cases there will be sufficient time for exact combination of m_5 and m_3. The caching of the uncombined focal sets in $m_4 \otimes m_0$ lets us perform anytime inference. We combine the focal sets with greatest weight first so that we get diminishing returns with time. The progress of anytime inference is shown in table 4.7. We consider the cases when we take $k = 2, 4, 6, 8$ focal sets respectively from m_4, with the degree of incompleteness $\epsilon(\phi) = 1 - \sum_A [\phi(A)]_m$ shown in fig. 4.8.

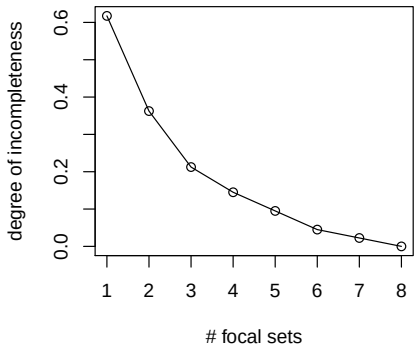


Figure 4.8: Degree of incompleteness of the *ploxoma* inference solution. As we include more focal sets, the degree of incompleteness converges to zero for the exact solution.

focal set	$k = 2$	$k = 4$	$k = 6$	$k = 8$ (exact)
$\emptyset$	0.019125	0.094125	0.096625	0.13825
$\{\theta_1\}$	0.01275	0.01275	0.01525	0.018625
$\{\theta_2\}$	0.580125	0.722625	0.812625	0.812625
$\{\theta_1, \theta_2\}$	0.0255	0.0255	0.0305	0.0305

Table 4.7: Approximations to the exact inference value (last column) for *ploxoma*

4.3 Conclusion

In this chapter we discussed several important instances of anytime ordered valuation algebras, specifically Bayesian networks, disjunctive normal forms in propositional logic, relational algebras and set potentials. Out of these all but set potentials are special cases of semiring induced anytime

ordered valuation algebras which we showed to be a class of valuation algebras that permit anytime inference in chapter 3. While the first three instances are semiring induced, the choice of the semiring and the applications influence the interpretation and significance of anytime inference.

Chapter 5

Implementation

In this chapter, we describe our implementation of the theoretical framework of anytime ordered valuation algebras presented in the preceding chapters. In contrast to existing software implementations of local computation [Pouly, 2010] which only consider exact inference, we implement an anytime, approximate inference framework as covered in chapter 3. We use Python as the implementation language because of its relative ease of use and popularity within the scientific computing community in general.

The chapter is organised as follows: Section 5.1 gives an overview of the library with the core functions. Section 5.2 gives examples of usage for Bayesian networks and disjunctive normal forms. We conclude with a brief note on further work in section 5.3. Examples of code from a Python REPL (read-eval-print-loop) session are identified with lines starting with a triple greater-than sign (⋙).

5.1 Overview

All inference algorithms in local computation are implemented as message-passing algorithms. In particular, we develop an implementation based on binary join trees. Initially, the binary join tree data structure contains the domains of the valuations only, so it is uninitialised with the actual valuations. The inward propagation algorithm propagates the valuations towards the root node, combining valuations from children nodes and projecting them to the domain of the parent node. By the nature of the construction of the binary join tree structure, the domain of the root node is the query for the inference problem, so this solves the inference problem

for that query. For illustration of anytime inference, we have initially implemented and focused on the anytime inward propagation algorithm (section 3.1). The following section describes the structure of the library, followed by section 5.1.2 where we discuss the valuation algebra operations.

5.1.1 Structure

The library is organised into submodules:

geninf Core valuation algebra class (`Valuation`) and operations, including parsers for common data formats and the anytime inference algorithm (`geninf.inference`)

geninf.instances Instance-specific operations such as combination ($\otimes$), time-bound combination ($\otimes_t$)[1] and projection ($\downarrow$). All instances derive from the abstract `Valuation` base class.

geninf.data Data files used for tests and examples.

geninf.experiments Code for the examples used in chapter 4, shown in section 5.2.

geninf.tests Test suite.

[1] In the formal framework, $\otimes_t$ is implemented in terms of the time bound update K. We have kept a separate $\otimes_t$ in the implementation for efficiency purposes.

5.1.2 Valuation algebra operations

We recall the structure of the anytime ordered valuation algebra as $\langle \Phi, V, d, \otimes, \downarrow, \succeq, K, \oplus, \rho \rangle$ (def. 3.1). These operations are implemented as the following methods in the instances, which derive from the abstract base class `Valuation`:

combine Two valuations v_1 and v_2 can be combined by v_1`.combine`(v_2). There is also an equivalent infix notation ($*$).

project The projection operation for a valuation v can be performed by v`.project`(domain)

domain For a valuation v, v`.domain()` returns the domain of v.

combine_t Two valuations v_1 and v_2 are combined with a time limit t by v_1`.combine_t`(v_2, t)

$\geq, \leq$ The comparison operators can be used to compare valuations in an instance with a partial order.

trunc, itrunc Returns the truncation and inverse truncation $(\rho, \bar{\rho})$ of valuations. For example, v`.trunc`(k) gives $\rho(v, k)$.

extend The composition operation for two valuations, with an equivalent infix notation of (+).

update_t The time bound update operation K.

Some instances also provide additional methods such as `tex` for typesetting and completeness measures.

The core inference module is implemented in `geninf.inference` with the `inward_t` and `refine` methods. Setting up an inference problem has the following steps: (i) read data into valuations (ii) given a query, construct a binary join tree from the valuations and (iii) run the inward propagation and refinement algorithms. This is implemented in the `Inference` class shown below, which encapsulates these operations. On an `Inference` object, the `run` method starts the inward propagation process, calling the `refine` method for further refinements. History of the intermediate valuations and join trees is maintained to facilitate inspection.

```
class Inference():
    _N = None # join tree state
    _elimination_order = []
    _valuations = []
    _query = []
    _refinecount = 0
    _bjts = []
    _s = 0

    def __init__(self, valuations, query, elimination_order):
        self._valuations = valuations
        self._query = query
        self._elimination_order = elimination_order

        self._N = construct_bjt(self._valuations, self._elimination_order)
        self._bjts = [(0, self._N)]
        self._s = len(self._valuations)

    def assign_leaf_nodes(self):
        valuations_to_leaf_nodes(self._N, self._valuations)

    def run(self, t):
        N_last = self._bjts[-1][1]
        if self._refinecount == 0:
            solution, Nn = inward_t(N_last, self._query, t, self._s)
        else:
            solution, Nn = refine(N_last, self._query, t, self._s)
        self._refinecount += 1
```

```
        self._bjts.append((t, Nn))
        return solution, Nn

    def reset(self):
        # forget history
        self._bjts = [self._bjts[0]]
        self._refinecount = 0
```

Initially, we have not focused on optimisation, instead aiming to build a library of the approximate and anytime generic inference framework.

5.2 Usage

5.2.1 Arithmetic potentials - Bayesian networks

We implement arithmetic potentials in this instance. Arithmetic potentials are represented in the implementation as a list of tuples of configuration points and their associated weight. Variables are encoded as integers.

To be of practical use, an inference framework should be able to read in and construct networks from external sources. There are a few standardised repositories for Bayesian networks[2] [3] – we shall be using the BNLEARN repository which provides DSC files (among other formats), which our library can parse.

[2] http://www.cs.huji.ac.il/site/labs/compbio/Repository/
[3] http://www.bnlearn.com

We shall give an example of using probability potentials; followed by showing how partial combination works. Then we shall consider the ASIA network shown in [Lauritzen and Spiegelhalter, 1990], and show partial time-bound inference.

► **Example 5.1.** If we consider the example given in the section on probability potentials (example 2.1), we can represent it as follows where the variables a, b, c are mapped to $1, 2, 3$ respectively. The `ProbabilityValuation` constructor takes the following arguments (i) a list of variables, (ii) a hash with the keys as the variables and the values as the frames of the variables and (iii) the configuration space with associated probabilities.

```
>>> from geninf.instances.probability import ProbabilityValuation
>>> p1 = ProbabilityValuation([1,2],{1:[0,1],2:[0,1]},
...     [({1:0,2:0},0.6),({1:0,2:1},0.4),({1:1,2:0},0.3),({1:1,2:1},0.7)])
>>> p2 = ProbabilityValuation([2,3],{2:[0,1],3:[0,1]},
...     [({2:0,3:0},0.2),({2:0,3:1},0.8),({2:1,3:0},0.9),({2:1,3:1},0.1)])
```

```
>>> p3 = p1 * p2
>>> print(p3)
[({1: 1, 2: 1, 3: 0}, 0.63), ({1: 0, 2: 0, 3: 1}, 0.48),
    ({1: 0, 2: 1, 3: 0}, 0.36000000000000004),
({1: 1, 2: 0, 3: 1}, 0.24), ({1: 0, 2: 0, 3: 0}, 0.12),
({1: 1, 2: 1, 3: 1}, 0.06999999999999999),
{1: 1, 2: 0, 3: 0}, 0.06), ({1: 0, 2: 1, 3: 1}, 0.04000000000000001)]
```

The output p_3 is expressed in the form a tuple with the first tuple in the form {variable: value} where the integer is the binary 0 or 1 (as this is an arithmetic potential). Thus the entry in the output `({1:0, 2:0, 3:0}, 0.12)` refers to the probability of the configuration $a = b = c = 0$ being 0.12. If we translate the output into tabular form we get table 5.1 with a representing $a = 0$ and $\bar{a}$ representing $a = 1$ and similarly for b, c. This is as expected the same as that given in example 2.1.

A	B	C	$p_1 \otimes p_2$
a	b	c	0.12
a	b	$\bar{c}$	0.48
a	$\bar{b}$	c	0.36
a	$\bar{b}$	$\bar{c}$	0.04
$\bar{a}$	b	c	0.06
$\bar{a}$	b	$\bar{c}$	0.24
$\bar{a}$	$\bar{b}$	c	0.63
$\bar{a}$	$\bar{b}$	$\bar{c}$	0.07

Table 5.1: Configuration space for p_3

► **Example 5.2.** Next we consider partial time-bound combination for the above example. Time bound combination works by combining only the configurations with the highest mass first, and continuing if time is left. This can be done by replacing a call to `combine` with `combine_t`.

```
>>> import operator
>>> from geninf.instances.probability import ProbabilityValuation
>>> p1 = ProbabilityValuation([1,2],{1:[0,1],2:[0,1]},
...    [({1:0,2:0},0.6),({1:0,2:1},0.4),({1:1,2:0},0.3),({1:1,2:1},0.7)])
>>> p2 = ProbabilityValuation([2,3],{2:[0,1],3:[0,1]},
...    [({2:0,3:0},0.2),({2:0,3:1},0.8),({2:1,3:0},0.9),({2:1,3:1},0.1)])
>>> p3 = p1.combine_t(p2, 0.0002)
>>> print(p3)
[({1: 1, 2: 1, 3: 0}, 0.63), ({1: 0, 2: 0, 3: 1}, 0.48)]
```

As we can see, calling `combine_t` with a restriction of $t = 0.0002$ has resulted in a partial configuration; the rest of the configuration is implicitly set to zero. Increasing the time allocated for the combination returns further elements from the configuration space. In figure 5.1, we plot a graph of the *error* of the valuation ϕ (`p3`), $\epsilon_t(\phi) = \sum_{\mathbf{x}} \phi(\mathbf{x}) - \sum_{\mathbf{x}} \phi_t(\mathbf{x})$ where $\phi_t(\mathbf{x})$ is the configuration mass at $\mathbf{x}$, when the combination has been allowed to run for time t, and ϕ is the corresponding exact valuation.

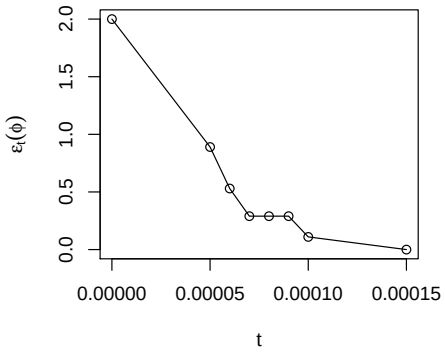


Figure 5.1: Graph of error in partial combination with time

► **Example 5.3.** Anytime inference for the *asia* network was

considered in example 4.1, with progress of anytime inference shown in fig. 4.3 and table 4.2 for the binary join tree in fig. 4.4. We used a query of {asia, bronchitis, dysponea} or the variables $\{A, B, D\}$, represented in the code as `[0,4,7]`. We perform an initial inward propagation for $t = 0.001$, with successive refinements. The inference converges to the exact solution at about $t = 0.02$. The code for the example is shown below.

```
if __name__ == "__main__":
    asianet = io.StringIO(geninf.data.networks.asia_net)
    d = DSC_Parser(asianet)
    query = [0,4,7]
    valuations = d.get_valuations()
    to_be_eliminated = list(set(range(len(d.nodes))) - set(query))
    asia = Inference(valuations, query, to_be_eliminated)
    asia.assign_leaf_nodes()

    solution_incompleteness = [(0.0, 1)]
    solutions = [(0.0, null(query))]
    times = [0.001, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005,
             0.0005, 0.0005, 0.005, 0.005, 0.005]
    t = 0
    for duration in times:
        sol, _ = asia.run(duration)
        t += duration
        solution_incompleteness.append((t, sol.incompleteness()))
        solutions.append((t, sol))

    with open("asia-incompleteness.txt", "w") as f:
        for t,i in solution_incompleteness:
            print(t,i,file=f)
```

5.2.2 Disjunctive Normal Forms

We implement disjunctive normal forms at the language level. They are represented in the implementation as a set of terms, where a term is a set of literals. Literals are a variable or its negation, for example `a,a'` represent $a, \bar{a}$ respectively. Conjunction and disjunction are denoted by (.) and (+) respectively.

► **Example 5.4.** Here we consider the inference in example 2.7, where we illustrated the equivalence of inference at the level of language and model, with $((a \to b).(b \to c))^{\downarrow\{a,c\}} = a \to c$.

```
>>> from geninf.instances.logic import *
>>> v1 = LogicValuation.expr("a' + b")
>>> v2 = LogicValuation.expr("b' + c")
>>> (v1 * v2).project(['a','c'])
a' + c
```

▶ **Example 5.5.** We considered the instance of disjunctive normal forms in the alarm example (example 4.2). Progress of anytime inference was shown in fig. 4.5 with the corresponding binary join tree shown in fig. 4.6. The query comprised the variables representing alarm, burglary, earthquake and the various causes of alarms: $\{a, b, e, a_1, \ldots, a_6\}$. The method `sat_count` returns the size of the model. The code for the example is shown below.

```
if __name__ == "__main__":
    alarm = Inference(geninf.data.alarm.vals, geninf.data.alarm.query,
                      ['w','g','d','c'])
    solutions = [(0,null(geninf.data.alarm.query))]
    sat = lambda v: v.sat_count(geninf.data.alarm.query)
    t = 0
    delta = 0.0003
    for i in range(20):
        s, _ = alarm.run(delta)
        ls = solutions[-1][1]
        t += delta
        solutions.append((t, s))
    f = open("alarm-sat-counts.txt", 'w')
    for t,s in solutions:
        print("%.4f\t%d" % (t,sat(s)), file=f)
    f.close()
```

5.3 Conclusion

In this chapter we have presented a library for approximate anytime inference. We have given an overview of the library structure with the core functions and presented examples for the instances of probability potentials and disjunctive normal forms. Due to the generic nature of the framework, we only need to specify certain operations for the instances instead of re-implementing the anytime inference algorithm for each instance. The progress of anytime inference for some of the examples was also presented in chapter 4. Future versions of the library will have support for more instances of the anytime ordered valuation algebra.

Chapter 6

Analysis of Distributed Inference

As mentioned in chapter 1, the increasing size of datasets involved in statistical inference and machine learning has led to the development of frameworks to structure algorithms, such as the MapReduce framework which has been applied to diverse areas such as genome sequencing [McKenna et al., 2010], machine learning [Chu et al., 2007] and astrophysics [Mackey et al., 2008]. With the emergence of multicore machines and the general acceptance that traditional methods of increasing core processor speeds are yielding diminishing returns, the theoretical study of these models as well as their application to the development of new parallel frameworks have become important. To parallelise algorithms operating on large graphs, frameworks such as Pregel [Malewicz et al., 2010] and GraphLab [Low et al., 2010] have been developed. Even though these frameworks have proved useful in practical applications, there is often a lack of theoretical analysis of communication costs and the tradeoff between computation and communication cost. Recently there have been efforts to describe frameworks such as MapReduce from a theoretical perspective, linking it to well-established theories of parallel algorithms such as BSP and PRAM [Pace, 2012, Feldman et al., 2010, Karloff et al., 2010, Goodrich et al., 2011].

Our contribution in this chapter is to undertake such a theoretical analysis of communication cost and the communication, computation and synchronisation tradeoffs involved in local computation. Here local computation refers to the general family of message passing schemes which are used to perform inference in valuation algebras.

We give an algorithm for optimisation of communication and computation costs, and perform tradeoff analysis for the case of balanced binary join trees. We compare our optimisation algorithm with prior work on optimisation of communication costs in weighted valuation algebras and note the advantages of our approach.

The chapter is organised as follows:

1. Section 6.1 reviews the communication, computation and synchronisation tradeoff framework introduced in [Solomonik et al., 2014]. This is the framework that we shall be using for our analysis and is a generalisation of the BSP [Valiant, 1990] framework.
2. Section 6.2 introduces the processor assignment algorithm and shows that it optimises communication costs. This is followed by the tradeoffs involved in local computation using Solomonik's framework. We give bounds on the number of processors for maximum concurrency and pairwise tradeoffs between communication, computational and synchronisation costs.
3. Section 6.3 reviews prior work on optimisation of communication costs in weighted valuation algebras [Pouly and Kohlas, 2005], with comparisons to our analysis and processor assignment algorithm described in section 6.2.
4. Section 6.4 concludes.

6.1 Communication, Computation and Synchronisation Tradeoffs in Distributed Computation

In section 6.1.2 we review the framework proposed in [Solomonik et al., 2014] which derives tradeoffs between communication, synchronisation and computational costs in a distributed algorithm, and gives bounds on the total execution time. In section 6.1.1, we review essential concepts in parallel computation.

6.1.1 Dependency graph, parallel schedule, and critical path

▶ **Definition 6.1.** DEPENDENCY GRAPH. The dependency graph of a program is a directed acyclic graph $G = (V, E)$ representing the dependencies in the data and the flow of execution in the algorithm.

▶ **Definition 6.2.** MAXIMUM CONCURRENCY. Maximum concurrency is defined as the maximum number of concurrent tasks at any point of the execution

The vertices of the dependency graph $V = I \cup Z \cup O$ are comprised of the *input* to the algorithm, I (vertices of in-degree 0), the *intermediary* vertices representing the computations, Z and the *output* vertices O (vertices of out-degree 0). The edges representing data dependencies are $E \subset V \times (Z \cup O)$. The data dependencies constrain the maximum concurrency possible; for example, if the dependency graph is a line graph with $V = (v_1, \ldots, v_n)$ and $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$ then the algorithm is effectively constrained to be sequential (maximum concurrency of 1); the computational cost of an algorithm having a line graph as its dependency graph would thus be identical to its sequential counterpart. A quantification of the degree of data dependency is the critical path length:

▶ **Definition 6.3.** CRITICAL PATH LENGTH. The critical path length is the length of the longest directed path between input node and output node. The corresponding path is called the *critical path*.

Colouring or processor assignment. In any distributed algorithm we need to assign processors to the nodes (this is also called, interchangeably, colouring a node, where we think of the processors as various colours) in the dependency graph. A *parallelisation* corresponds to a colouring of the dependency graph $G = (V, E)$, or a partitioning of the vertices into p disjoint sets C_i ($i = \{1 \ldots p\}$), where processor i computes $C_i \cap (Z \cup O)$. A vertex v of colour (processor) i, i.e. $v \in C_i$ has to be communicated to a different processor j if there is an edge $(v, k) \in E$ such that $k \in C_j$.

▶ **Definition 6.4.** PARALLEL SCHEDULE. The parallel schedule of a distributed algorithm with a dependency graph

$G = (V, E)$ is another DAG $\bar{G} = (\bar{V}, \bar{E})$, which has p edge-disjoint paths, corresponding to p processors; each path corresponding to the tasks executed by a certain processor, such as communication, computation and synchronisation.

In addition to the computation vertices in the underlying dependency graph (V, E), we also add other types of vertices which represent communication and synchronisation in the distributed system; this gives an explicit representation of these tasks in the parallel schedule; thus the vertices $v \in \bar{V}$ can belong to either:

- $v \in \bar{V}_c$, computation node, denoted by round nodes ○
- $v \in \bar{V}_s$, synchronisation point, denoted by diamond nodes ◇
- $v \in \bar{V}_o$, sending (outbox) message point, denoted by square nodes □
- $v \in \bar{V}_i$, receiving (inbox) message point, denoted by square nodes □

Each vertex $v \in \bar{V}_c \cup \bar{V}_s \cup \bar{V}_r$ should be adjacent to at most one incoming and outgoing edge. Only the synchronisation vertices can have $k \geq 1$ incoming and k outgoing vertices corresponding to a k-way synchronisation among processors.

Each message in this model has a single originating and destination processor. However messages need not be directly transferred to the destination and can be routed through intermediary nodes. Multiple messages may be transmitted through the same synchronisation vertex. A schedule is **point-to-point** if a synchronisation vertex has at most two incoming and outgoing paths.

► **Example 6.1.** CONSTRUCTION OF A PARALLEL SCHEDULE. We shall consider the following coloured dependency graph and convert it to a parallel schedule. The dependency graph is coloured with 3 processors, indicated by shaded, clear and dotted outlines respectively.

Converting this to a parallel schedule results in subtrees allocated to the three processors being converted to paths representing execution flow in the parallel schedule (fig. 6.2).

In this case, we get 2 synchronisation points and 4 send/receive nodes.

6.1.2 Model Description

The framework has three primary parameters (coefficients) corresponding to communication, computation and synchronisation costs. There are also the associated algorithmic costs. The parameters and costs are summarised in table below.

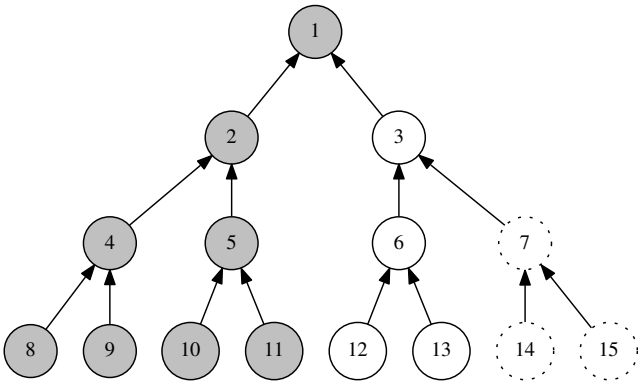


Figure 6.1: Coloured dependency graph

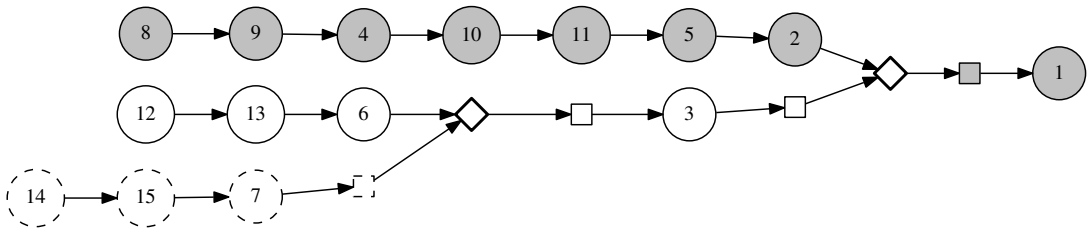


Figure 6.2: Parallel schedule corresponding to fig. 6.1

coefficient	cost	cost contribution
α, network latency, or time for synchronisation between two or more processors	S, number of synchronisations or network latency cost.	$\alpha \cdot S$
β, time to send or receive a word of data from the network.	W, number of words transferred or communication cost.	$\beta \cdot W$
γ, time to perform a (floating point) operation on local data.	F, number of local floating point operations performed or computational cost.	$\gamma \cdot F$

Each of the quantities S, W, F is considered along individual execution paths in the parallel schedule. The sequence of operations performed by any processor locally is thus a lower bound for these quantities. The execution time of the parallel schedule is related to these quantities by:

$$\max(\alpha \cdot S, \beta \cdot W, \gamma \cdot F) \leq \text{execution time} \leq \alpha \cdot S + \beta \cdot W + \gamma \cdot F \tag{6.1}$$

The analysis is asymptotic. Overlaps between communication and computation are not considered, but instead considered separately. In order to perform the asymptotic analysis, we need to consider the dependency graph of the computation.

The parallel schedule runtime can then be defined as the maximum total weight of any path in the schedule.

$$T(\bar{G}) = \max_{\pi \in \Pi} \sum_{v \in \pi} \hat{t}(v) \tag{6.2}$$

where Π is the set of p edge disjoint paths representing the sequence of computation, communication and synchronisation performed by each processor.

Here $\hat{t}(v)$ is the cost associated with a particular task, which varies according to whether it is computation, communication or synchronisation:

$$\hat{t}(v) = \begin{cases} \gamma & v \in \bar{V}_c \\ \alpha & v \in \bar{V}_s \\ \beta & v \in \bar{V}_o \\ \beta & v \in \bar{V}_i \end{cases} \tag{6.3}$$

We depart slightly from [Solomonik et al., 2014] in that their framework allows each vertex in the parallel schedule to represent multiple vertices of the dependency graph. We do not need this, and it also simplifies discussion by having a direct correspondence between the parallel schedule and the dependency graph.

6.2 Tradeoffs for Local Computation

In this section we discuss the tradeoffs involved in distributed local computation using the framework described in the previous section. In contrast to prior work on weighted

valuation algebras reviewed in the section 6.3, this work looks at the communication, computation and synchronisation costs rather than just the communication cost. We derive bounds on the communication and single processor computational cost and discuss how it scales with the number of processors. We also give bounds on the number of processors for maximum concurrency. Beyond this bound, increasing the number of processors does not lead to faster overall execution time, instead leading to unnecessary additional communication and synchronisation overhead. For our analysis, we shall focus on inward propagation. Irrespective of whether we use the the exact inward propagation (section 2.7.2) or the anytime inference algorithm (section 3.1.1), we have the same binary join tree structure and thus the same dependency graph.

We shall use these definitions in the next section:

▶ **Definition 6.5.** DEPTH OF A NODE. Depth of a node in a tree is the length of the unique path between the node to the root node. The root node is at depth 0.

▶ **Definition 6.6.** DEPTH OF A TREE. Depth of a tree is the maximum depth of a node in the tree.

6.2.1 Processor assignment algorithm

We note that the dependency graph for the anytime inward propagation algorithm is the binary join tree itself $N = (V, E)$; our aim will be to construct a corresponding parallel schedule $\bar{N} = (\bar{V}, \bar{E})$ which shall give us our cost estimates.

▶ **Example 6.2.** We give a simple example for illustration before discussing the processor assignment algorithm. The colouring shall proceed recursively. Consider the base case (tree of depth $d = 1$, one root node, two leaf nodes, figure 6.3). This has a maximum concurrency of 2, as the leaf nodes can independently project their own valuations and send them to the root node. For the case $d = 2$ we consider the decomposition of the tree into subtrees T_1 and T_2 where T_1, T_2 are trees of depth 1; the binary join tree now has a maximum concurrency of 4 as shown. Since each processor is responsible for a single connected subtree of the binary join tree, it only needs to communicate the results once from the root of the subtree it is allocated.

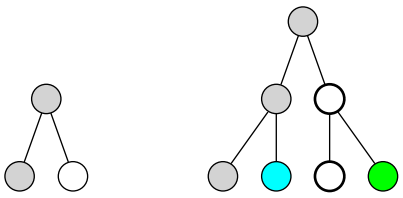


Figure 6.3: The first two cases $d = 1, 2$ in the PROCESSOR-ASSIGN algorithm, left: $d = 1$, right: $d = 2$

We formalise the processor assignment algorithm in

PROCESSOR-ASSIGN, where the set of processors is an ordered set M and $c(i)$ denotes the colour (processor) of a node i. We note that the critical path length of a tree is the depth of the tree. The depth of a tree with the root node n is denoted by $\text{DEPTH}(n)$.

The processor assignment is invoked at the root node.

1: **procedure** PROCESSOR-ASSIGN(n, M)
2: **if** $|M| = 1$ **then**
3: COLOUR-SUBTREE(n, M_0) [1]
4: **return**
5: **end if**
6: $c(n) \leftarrow M_0$
7: split M in half to M_A, M_B s.t. $M_0 = M_{A,0}$ [2]
8: $M_L \leftarrow M_A$ **if** $\text{DEPTH}(L(n)) \geq \text{DEPTH}(R(n))$ **else** M_B
9: $M_R \leftarrow M_B$ **if** $\text{DEPTH}(L(n)) < \text{DEPTH}(R(n))$ **else** M_A
10: PROCESSOR-ASSIGN($L(n), M_L$)
11: PROCESSOR-ASSIGN($R(n), M_R$)
12: **end procedure**

[1] M_0 is the first element of M

[2] It does not matter which set has more processors. We can decide which set to give more processors to (M_A or M_B) to make the algorithm reproducible. Both M_L and M_R are ordered sets.

1: **procedure** COLOUR-SUBTREE(n, m)
2: $c(n) \leftarrow m$
3: **if** $n \in \mathit{leaves}(N)$ **then**
4: **return**
5: **end if**
6: COLOUR-SUBTREE($L(n), m$)
7: COLOUR-SUBTREE($R(n), m$)
8: **end procedure**

Before proving optimality, let's consider an example. We already looked at this dependency graph in section 6.1.1 and showed the conversion to a parallel schedule. This is the colouring that is obtained if we apply PROCESSOR-ASSIGN to the underlying balanced BJT with $|M| = 3$.

Now we shall show that this algorithm is optimal with respect to communication cost, while utilising the maximum concurrency available. To optimise communication cost, we show that we assign all the nodes in the critical path to the same processor. Assigning nodes in the critical path to different processors would only increase communication costs, while not decreasing computation time.

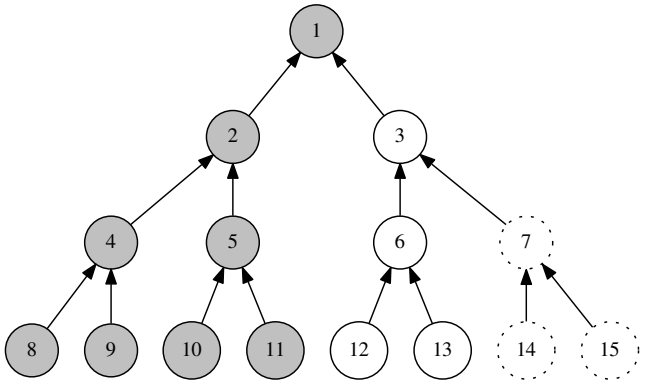


Figure 6.4: Processor assignment for $d = 3, p = 3$, 15 nodes, 8 valuations, number of messages = 2 ($3 \to 1$, $7 \to 3$)

▶ **Theorem 6.1.** PROCESSOR-ASSIGN *assigns all nodes of the critical path length the same colour.*

Proof. The root node is assigned M_0. We ensure that the critical path is assigned M_0 by lines 8–9. To see this, we use the notion that the tree depth is the same as the critical path length. We also use the inductive notion for depth:

$$\text{DEPTH}(n) = 1 + \max(\text{DEPTH}(L(n)), \text{DEPTH}(R(n))) \tag{6.4}$$

Once the root node has been assigned M_0, we have a choice between the left and right subtree roots to colour M_0. Since nodes in the critical path must lie in the subtree with the higher depth, we colour the corresponding subtree root the same as M_0. This continues till we have $|M| = 1$, when we colour the entire subtree, which would include the critical path. □

▶ **Theorem 6.2.** PROCESSOR-ASSIGN *generates at most* $p - 1$ *communication steps for* p *processors.*

Proof. This easily follows from the fact that nodes of the same colour belong to a connected subtree of the same colour. Thus the tree is partitioned into subtrees which communicate only across the edge connecting one subtree to another.

To see why nodes of the same colour from a connected subtree, consider two nodes m and n of the same colour. There are three cases (i) m is on the path from n to the root (ii) when n is on the path from m to the root (iii) both m and n belong to a subtree rooted at r.

The cases (i) and (ii) are interchangeable with m being swapped with n. We consider case (i) first and note that

the $L(m)$ and $R(m)$ subtrees will get assigned a disjoint pool of processors. Without loss of generality, take $n \in \text{SUBTREE}(L(m))$, where $\text{SUBTREE}(n)$ is the subtree rooted at n. The fact that $c(n) = c(m)$ implies that the processor pool assigned to $L(m)$ must have had $c(m)$. In that case, we would have $c(L(m)) = c(m)$ as $c(m)$ would have been ordered first in the pool assigned to $L(m)$. We can now recursively apply this procedure to $L(m)$ and n which are the same colour to get a $c(m)$ coloured path from m to n.

For case (iii), we have both $m, n \in \text{SUBTREE}(r)$ for some r. For this, we again use the fact that the left and right subtrees of the root r, $L(r)$ and $R(r)$ subtrees get assigned a disjoint pool of processors. As m and n belong to $L(r)$ and $R(r)$ (or the other way round, we pick one without loss of generality), the only way they get the same processor is via COLOR-SUBTREE, in which case the the entire subtree is coloured $c(r)$, which will make $c(m) = c(n) = c(r)$.

As we have at most p coloured, connected subtrees, we will have at most $p - 1$ communication steps. □

An important simplification in our analysis is that all valuations have the *same weight*. From a communication cost perspective this can be taken as the maximum weight of the valuations that can be computed at any join tree node and is similar to consideration of treewidth as a measure of computational complexity in inference. This allows us to generate an optimal processor distribution without relying on an initial colouring or assignment of processors to the leaf valuations as is done in [Pouly and Kohlas, 2005], reviewed in section 6.3.

The PROCESSOR-ASSIGN algorithm gives a colouring or partition of the dependency graph $N = (V, E)$. Now we can construct a parallel schedule corresponding to the (coloured) dependency graph:

- For edges (v, w) with $c(v) \neq c(w)$, the processor $c(v)$ would need to send a message to $c(w)$. For this, we add a send/receive node pair, denoted by □. As the framework requires synchronisation before data can change control, we have to connect these via a synchronisation point, denoted by $\diamond$.
- Nodes in the same partition, which are the responsibility of a particular processor, are stacked to reflect the execution flow for the corresponding processor.

Before we state the tradeoffs, we estimate the costs for the following: computation (F_L, theorem 6.3), total communication (Q_L, theorem 6.4), communication (W_L, theorem 6.5) and synchronisation (S_L, theorem 6.6). The subscript L denotes local computation.

Each of these quantities is measured along a path in the parallel schedule. It is useful to recall that each path in the parallel schedule represents a parallel processor, and thus these quantities, except for Q_L, are taken as the maximum along any particular execution path.

► **Remark.** In the following, we take the cost of combination at a particular join tree node to be $O(k)$. Here k is a function that denotes the maximum complexity of combination at a join tree node; it is generally a function of the treewidth of the join tree (size of the largest join tree label). This function is naturally dependent on the particular instance of valuation algebra being discussed. The particular functional form of k does not affect the following proof and discussion.

► **Remark.** For the tradeoff analysis, we impose the condition that the BJT be balanced. While a BJT in general will not be balanced, we outline two approaches to construct balanced binary join trees for specific instances of valuation algebras.

The first approach is for arithmetic potentials (Bayesian networks). [Darwiche, 2001] introduces *dtrees* which permits exact inference for Bayesian networks while offering a smooth tradeoff between time and space. The introduced structure is similar to BJTs, a relationship which is formalised in [Darwiche, 2001, section 6.1], which gives a mapping from dtrees to join trees, while preserving the underlying tree structure of a dtree. The mapping in fact gives a BJT, as the conditional probability tables (valuations) are assigned to the leaves and every node has at most three neighbours.

In section 6.3, Darwiche presents a method to balance dtrees, essentially by combining nodes till the tree is balanced. Since a balanced dtree can be converted to a balanced BJT, this method can be used to construct balanced BJTs.

The second method can work for instances which have an identity element. Then we can add nodes containing identity to make the tree balanced. As the time complexity is linear in the number of nodes, this will only work if the number of nodes added to a join tree is linear in the number of existing

nodes.

▶ **Theorem 6.3.** *The number of floating point operations F_L in the parallel schedule across any path in the parallel schedule is $O(nk/p)$*

Proof. First, we note that the balanced binary join tree will have 2^k nodes at level k. Thus if n is the number of nodes then

$$n = 2^0 + \cdots + 2^d \tag{6.5}$$

where we add up the nodes for each level, where d is the depth of the tree, which can be expressed in terms of n, $d = \lg(n+1) - 1$.

We make a further assumption that $p = 2^k$, for $k < m$. If this is not so, we round p down to the nearest power of 2, as we shall see, this only changes F_L by at most a factor of 2.

We note that in the parallel schedule, each path corresponds to a different processor. Therefore the length of the longest path will determine the complexity. We can see that in a balanced tree, M_0 gets assigned the most number of nodes, including the root node. Thus, the number of nodes assigned to M_0 will determine the floating point complexity. As we have assumed equal sizes for (transmitted) valuations, the complexity of combination is $O(k)$ at every non-leaf node.

Let's first get the number of nodes assigned to M_0. In a balanced tree, all paths from the root to the leaves have equal length, so we always have $M_{L,0} = M_0$. Thus the left branch of the tree starting from the root node $(n, L(n), L(L(n)), \ldots)$ is always assigned M_0, till the processor pool is exhausted ($|M| = 1$ in line 2 of PROCESSOR-ASSIGN). At that point, the entire subtree at n is assigned to processor M_0. So to get the number of nodes assigned to M_0, there are two steps: (i) find at which depth we have $|M| = 1$; M_0 will be assigned the left branch till this point, and (ii) the number of nodes in the subtree when $|M| = 1$ ($M = \{M_0\}$) as these get assigned to M_0.

For step (i), as we start with 2^k processors, we will have k steps till we get $|M| = 1$. M_0 is assigned the left branch till this point, i.e k nodes.

For step (ii), we colour (assign) the subtree n at depth k to M_0. The subtree rooted at node n at depth k will have $2^0 + 2^0 + \cdots + 2^{d-k}$ where d is the depth of the tree. As no combination happens at the leaves (nodes at highest depth),

we shall be computing combinations for C nodes:

$$C := k + 2^0 + 2^1 + \cdots + 2^{d-k-1} \tag{6.6}$$

As we explained at the beginning of the proof, the number of nodes assigned to M_0 determines the floating point complexity. If we leave the leaves of the tree, then absent the $O(k)$ factor, F_L is the same as C:

$$\begin{aligned} C &= k + (2^0 + 2^0 + \cdots + 2^{d-k-1}) \\ &= k + (2^{d-k} - 1) \\ &= k - 1 + \frac{n+1}{2p} \end{aligned}$$

As $k = O(\lg n)$, we can neglect that term. Thus we get $C = O(n/p)$ and $F_L = O(nk/p)$. If we have to round p down to the nearest power of 2, we get $p' = p/2$ as a minimum, which keeps the same $O(nk/p)$ bound. □

▶ **Theorem 6.4.** *$Q_L = O(pk)$, where p is the number of processors and k is the upper bound on weights of transmitted valuations, analogous to the k parameter defined in the anytime inference algorithm for the truncation function ρ.*

Proof. As each processor i is responsible for a single connected subtree only (theorem 6.2), we can add a synchronisation and send/receive vertices to the single edge $(v, w) \in E$ such that $c(v) \neq c(w)$ with $c(v) = i$. As the communication cost (bytes transferred) of the valuation is proportional to the truncated portion, i.e. $O(k)$ we get the worst-case communication cost across a single path as $O(pk)$. □

▶ **Example 6.3.** We look at an example to illustrate the argument made in theorem 6.4, in particular where we calculate the number of nodes assigned to the first processor. Consider a BJT with 16 valuations (and thus $2 \cdot 16 - 1 = 31$ nodes).

Let's start with two processors: we perform the obvious assignment, the root and the left subtree are assigned processor M_0. For this example, we indicate the assignment to processor M_0 with shaded nodes, while the unshaded nodes represent assignments to any other processor. We show the assignments to processor M_0 for $p = 4, 8, 16$ in the same figure.

We do not show the case $p = 3$, as the processor assignment would happen on the right subtree, keeping the left

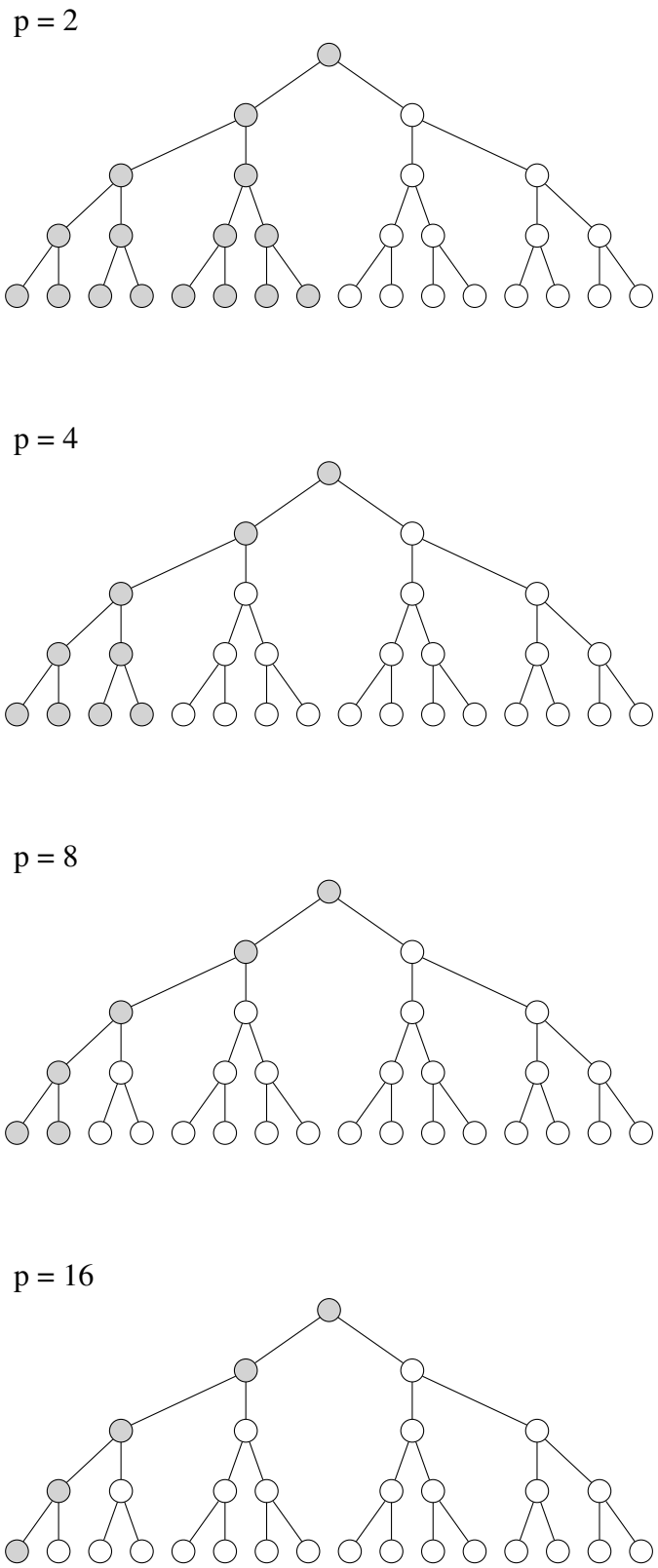


Figure 6.5: Processor assignment in a BJT constructed from 16 valuations for $p = 2, 4, 8, 16$. We show the assignments for the first processor only, to illustrate the argument in theorem 6.4

subtree unchanged. As we shall see, only for $p = 2^i$, does the allocation to processor m_1 change. For instance, for $p = 4$, we assign the right subtree of the left subtree (i.e. the subtree rooted at $R(L(root(N)))$) to the fourth processor according to PROCESSOR-ASSIGN.

Beyond $p = 16$, we do not achieve an increase in concurrency by adding processors, which only serves to increase communication costs. This is due to the nature of the dependency graph (a tree), where each successive level (from the bottom up, with top being towards the root) has fewer nodes and depends upon the results from the level below it.

▶ **Corollary. Maximum concurrency in local computation.** *For the case of a balanced BJT, the maximum concurrency for a knowledgebase with n valuations is achieved with n processors.*

Proof. For $p = n$, the number of local floating point operations $F_L = O(nk/p) = O(k)$ which is the lower bound for a single processor. Intuitively, the tree structure of the dependency graph constrains parallelism and the greatest concurrency is at the leaves, which are bounded by n, due to the nature of the BJT construction algorithm. □

▶ **Theorem 6.5.** *The maximum communication cost W_L across any single path in the parallel schedule is $O(k \lg p)$*

Proof. We know that each processor i is responsible for a single connected subtree only (theorem 6.2). Like in theorem 6.3, we assume $p = 2^m$, and let's consider the processor M_0. It is easy to see that M_0 will have only incoming links, as it is the processor assigned to the root node. As the left most branch of the tree is always assigned to M_0, the number of incoming links to M_0 is the upper bound for communication along any single path. M_0 will be assigned the left branch, but *not* the right branch, until $|M| = 1$. These right branches will have to send messages to M_0, and there will be atmost m messages as we have m steps till we get $|M| = 1$. Thus we get $W_L = O(mk) = O(k \lg p)$. □

▶ **Theorem 6.6.** *The maximum synchronisation cost S_L across any single path in the parallel schedule is $O(\lg p)$.*

Proof. Each synchronisation is one-to-one in our case, there is no synchronisation of more than two processors. Thus each synchronisation corresponds to a communication step, and we get the same cost, except the factor of k. □

▶ **Corollary.** *The tradeoff between* total *communication cost and computational cost measured over a single execution path, can be expressed as* $Q_L \cdot F_L = O(nk^2)$*, where* n *is the number of valuations, and* k *is the upper bound on the* size *of transmitted valuations.*

▶ **Corollary. Bound on total execution time.** *For a local computation parallel schedule, with* p *processors and* n *valuations, the execution time is bounded by* $\alpha O(\lg p) + \beta O(k \lg p) + \gamma O(nk/p)$*, thus effectively* $O(k \lg p) + O(nk/p)$*.*

Proof. This result is obtained by application of eq. (6.1) and results from theorems 6.3, 6.5, 6.6. □

6.3 Communication costs in Weighted Valuation Algebras

In this section we review the work on optimisation of communication costs in weighted valuation algebras [Pouly and Kohlas, 2005], followed by a comparison to our analysis in section 6.3.2. We focus on the discussion of the algorithm for optimisation of communication costs and refer the reader to the article for a detailed discussion of their framework.

Weighted valuation algebras are an extension of valuation algebras with an additional weight function which allows us to calculate the communication cost. Assigning weights to valuations reflect the real-world requirement of executing local computation on large join trees, distributed across multiple processors. Then, nodes in the underlying join tree structure of local computation are identified with real processors, following from the work of [Kohlas, 2003] which treated join tree nodes as virtual processors, exchanging messages between them.

▶ **Definition 6.7.** The weight function for a valuation algebra is defined on the valuations. If $\langle \Phi, V, d, \otimes, \downarrow \rangle$ is a valuation algebra, then a function $\omega : \Phi \to \mathbb{N}$ is called a weight function if for all $\phi \in \Phi$ and $x \subseteq d(\phi)$ we have $\omega(\phi) \geq \omega(\phi^{\downarrow x})$.

This definition can be extended to valuation algebras with an identity e by assigning a constant to $\omega(e)$. Generally, the identity element has no information ($\omega(e) = 0$).

Some examples of weight functions are:

▶ **Example 6.4.** For *semiring induced valuation algebras*, defined as $\phi : \Omega_S \to A$ where Ω_S is the frame of S, or the set of all possible configurations of a set of variables S and A is the semiring whose values are assigned. Then a general weight function for such valuations is

$$\omega(\phi) = \prod_{x \in S} |\Omega_x| \tag{6.7}$$

▶ **Example 6.5.** A possible weight function for belief potentials with domain s is

$$\omega(\phi) = 2^{|\Omega_s|} \tag{6.8}$$

Another representation of belief potentials is counting the number of focal sets, as these shrink when projected to a subdomain.

6.3.1 Optimisation of communication costs

First, we define a penalty function:

▶ **Definition 6.8.** On a graph $G = (V, E)$, where G is the join tree, a penalty function is a map $p : V \to (\mathbb{N} \cup \{\infty\})^{|P|}$ where P is the set of processors. The map $p_i(v)$ denotes the total weight of a subtree of a node v, if processor i has been assigned to v, denoted by $m(i) = v$.

There are two phases in this processor assignment algorithm: an inward phase, from the leaves to the root which computes the penalty function for all interior nodes, and an outward phase which assigns a processor to the interior nodes based on the minimum penalty.

Inward (Phase 1) For each leaf v,

$$p_i(v) = \begin{cases} 0 & \text{if } m(v) = i \\ \infty & \text{otherwise} \end{cases} \tag{6.9}$$

Then, recursively for interior nodes:

$$p_i(v) = \sum_{u \in \{L(v), R(v)\}} \min_{j \in P} \{f_u . d_{ij} + p_j(u)\} \tag{6.10}$$

where d_{ij} is the distance between processors i and j and f_u is the weight of the valuation at u.

Outward (Phase II) The complete processor mapping $m : V \to P$ is determined such that the total communication costs are minimised. First, the root node r is assigned i such that $p_i(r) \leq p_j(r)$ for all $j \in P$. The interior nodes are then assigned recursively, with $m(v) = i$ if $f_v \cdot d_{i,m(ch(v))} + p_i(v) \leq f_v \cdot d_{j,m(ch(v))} + p_j(v)$ for all $j \in P$. The procedure is well-defined since the children of a node are assigned processors first.

6.3.2 Comparison with our analysis

A key difference between the analysis presented in [Pouly and Kohlas, 2005] and the one presented in section 6.2 is that our work is based on established theories of parallel computation, like [Solomonik et al., 2014], a generalisation of the BSP model. This enables us, in contrast to prior work, to give an asymptotic analysis of communication costs, as well as the tradeoff with computational and synchronisation costs.

We shall now compare processor assignment algorithms.

▶ **Example 6.6.** If we apply the optimisation algorithm discussed in the previous section to example 6.1, keeping the processor assignments at the leaves only (as we require an initial assignment for the inward propagation of penalties), we get the assignment as shown in fig 6.6. The corresponding processor distribution using PROCESSOR-ASSIGN is shown in fig 6.4.

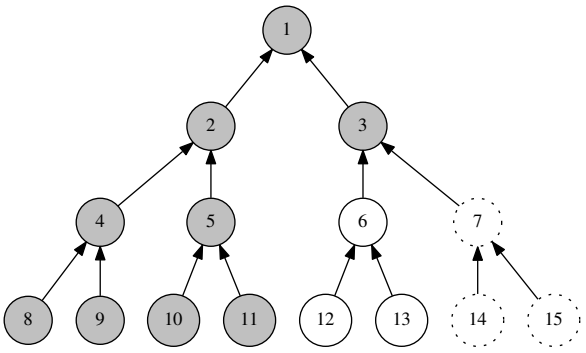


Figure 6.6: Processor distribution for example 6.1 with $d = 3, p = 3$ using the optimisation algorithm described in section 6.3.1. Number of messages = 2 ($6 \to 3$, $7 \to 3$).

Thus the communication cost in both optimisation methods is same, if we impose the condition of valuations having equal weight. If we consider the inward propagation of penalties in our example, we can see that any node with both

parents (or subtrees) having a colour i will get coloured i to minimize the penalty. To see this, we look at the penalty assignment during inward propagation

$$p_i(v) = \sum_{u \in \{L(v), R(v)\}} \min_{j \in P} \{\delta_{ij} + p_j(u)\}$$

where we set $f_u = 1$ as all our valuations have the same weight and $d_{ij} = \delta_{ij}$ where δ_{ij} is the Kronecker delta which is 1 if $i = j$, 0 otherwise. If both leaves are the same processor, say j, then we can see that $p_j(v) = 0$ if $p_j(L(v)) = p_j(R(v)) = 0$. Thus this is the processor that will get assigned in the outward phase. In effect, this will also give rise to a j-coloured subtree.

In effect, PROCESSOR-ASSIGN generalises the optimisation algorithm in section 6.3.1, when ignoring valuation weight. However it has the advantage of finding the optimal communication cost given the number of processors without depending on some arbitrary initial assignment of valuations.

6.4 Conclusion

In this chapter, we analysed the local computation algorithm for anytime inference in a distributed setting. In any distributed computation, communication and synchronisation costs become important in addition to computational costs. Using the theory of [Solomonik et al., 2014] which generalises the widely used BSP model, we give results for tradeoffs in synchronisation, communication and computational costs, including a processor assignment algorithm which is optimal with respect to computation and communication costs. We also consider the tradeoffs for when the binary join tree is balanced. While this work in the context of local computation in valuation algebras was performed independently, we note that similar results are known in the literature on prefix sums [Blelloch, 1990], due to the similarity in dependency graph structure.

The tradeoff result shows that increasing the number of processors reduces the computational overhead per processor as expected, but increases total communication volume proportionally ($Q_L \cdot F_L = O(nk^2)$). This also gives us a bound on the number of processors for distributed inference, beyond which adding processors will not contribute to a reduced run-

time of the algorithm. We compare our processor assignment algorithm with previous work and note the advantage of our approach, which is independent of an initial assignment of valuations to processors.

Chapter 7

Conclusion

In this thesis we explored aspects of inference in a setting that is sufficiently general to include a wide gamut of applications, which range from propositional logic and relational algebras to Bayesian networks and set potentials in Dempster Shafer theory. Our work is rooted in the extensive literature on local computation [Lauritzen and Spiegelhalter, 1990, Shenoy, 1992, Shenoy and Shafer, 1990], message-passing algorithms [Shenoy, 1997, Pearl, 1988] and the framework of generic inference [Pouly and Kohlas, 2011, Haenni, 2004]. We make contributions to anytime inference and analysis of distributed local computation, which are summarised below.

Anytime inference We introduced a framework to perform anytime inference in valuation algebras (Chapter 3; also as [Dasgupta and Abramsky, 2016, preprint]). Anytime inference has the advantage of giving partial results, or an approximation to the exact inference solution which can be improved with time without repeating unnecessary computations. In contrast to previous work on anytime inference, the proposed theoretical framework of anytime ordered valuation algebras is generic, with guarantees of soundness and completeness under certain axioms. This genericity allows us to capture the essential features of valuation algebra instances which permit anytime inference.

The introduced framework is also applicable to various instances, discussed in chapter 4. The class of semiring induced valuation algebras, which includes instances such as arithmetic potentials and disjunctive normal forms, among others is also shown to be an anytime ordered valuation algebra. This makes a large class of valuation algebras immediately amenable to anytime inference.

In chapter 4, we explored a selected subset of these, namely, Bayesian networks, disjunctive normal forms in propositional logic, relational algebras and set potentials. As exact inference is #P-hard, approximate anytime inference can be a useful tool to explore the inference problem, and in certain cases, have a reasonable interpretation even as a partial solution.

Distributed inference With the advent of large datasets and increasing availability of multicore machines, it has become not only important, but also practically useful to consider algorithms for distributed computation. In chapter 6, we performed an analysis of distributed local computation with the inward propagation algorithm. We derived tradeoffs between computational, synchronisation and communication costs, which are important considerations for a distributed algorithm. This also gave a bound on number of processors for maximum concurrency. Our analysis is based on [Solomonik et al., 2014], which generalises the well-established Bulk Synchronous Parallel model [Valiant, 1990]. A processor assignment algorithm is also presented, which offers some advantage as compared to prior work on optimisation of communication costs in local computation [Pouly and Kohlas, 2005].

Implementation In chapter 5 we described the implementation of our anytime inference framework as a software library in Python, with examples of usage. The library is modular and supports additional instances. Due to the genericity of the framework, specification of the basic operations of the anytime ordered valuation algebra is sufficient for the anytime algorithm.

Future Work Throughout the thesis, our focus has been to preserve the genericity of the local computation framework. Working on a generic framework allows immediate applicability to a large class of problems in a unified manner. The applications of generic inference and local computation are numerous [Pouly and Kohlas, 2011]; aside from the common instances reviewed in chapter 2, it can formulate Fourier and Hadamard transforms, linear programming and constraint satisfaction problems. We considered a small subset of these as an application of anytime inference; exploration of other instances is left for future work. In particular, we note that it should be relatively straightforward to generalise the instance of set potential anytime ordered valuation algebras (where

the valuations map to the reals) to arbitrary semiring induced set potential anytime ordered valuation algebras.

The work covered in this thesis used the the most general of local computation architectures, partly to ensure that we wouldn't have to consider special cases for specific instances of valuation algebras. However there are specialised message-passing architectures, such as the Lauritzen-Spiegelhalter and HUGIN architectures, briefly reviewed in section 2.7.4. These architectures improve upon certain aspects by considering additional restrictions on the valuation algebra, such as the division operation in Lauritzen-Spiegelhalter. Consideration of such architectures would simplify some of the algorithms, as it does in the case for exact inference.

Other areas of interest as a natural extension of this thesis are relating this work to other distributed anytime inference algorithms, such as for MAP inference in Bayesian networks [van de Ven and Ramos, 2012, Williams et al., 1997], as well as other approximate inference techniques such as the mini-bucket scheme [Dechter and Rish, 2003].

In Perspective From a broader perspective, this work contributes to the well-established literature on local computation and message-passing algorithms. Due to the very general nature of the valuation algebra framework, the notions of the generalised distributive law it captures, and the ubiquity of the join tree structure, local computation schemes appear and are discovered in various contexts. As an example, recent work has shown applications of local computation in the topological approach to contextuality [Abramsky, 2013, Abramsky et al., 2015, Kishida, 2016], extending applications of local computation to the quantum realm. Here contextuality refers to the aspect of quantum mechanics where a measurement of a quantum observable depends upon the context, such as the experimental setup and the set of commuting observables. The message-passing architecture also makes it easier to transition to a distributed computation environment and local computation has found application in sensor networks [Paskin et al., 2005] to perform inference.

Appendix A

Appendix

A.1 Axioms of Anytime Ordered Valuation Algebras

For convenience, we have listed the axioms of anytime ordered valuation algebras that were defined in def. 3.1, with their use within the framework on the following page.

Axiom		Use
A1 – *A6*		Core valuation algebra
A7	Partial Order	used to define the approximation relation
A8	Null element	used to define the time-bound combination K' in terms of the time-bound update K'; also used to derive that we get the null valuation at $t = 0$
A9	Combination preserves partial order	used extensively in the soundness proof
A10	Projection preserves partial order	
A11	Composition preserves partial order	
A12	Composition forms a monoid	
A13	Combination $\otimes$ distributes over composition $\oplus$	used in soundness when extending cached valuations from the last step
A14	Projection $\downarrow$ distributes over composition $\oplus$	
A15	Size of truncated valuations	used in completeness, to show that a decreasing sequence of valuations converge to the null valuation
A16	Truncation monotonically increasing	used in completeness, to show a sequence of valuations is decreasing
A17	Zero truncation	monotonicity of anytime inference and compatibility with *R5* – *R9* of ordered valuation algebras
A18	Time bounded update	
A19	Monotonic time bounded update	
A20	Zero time update	
A21	Maximum time for update	

List of Figures

Bibliography

[Abramsky, 2013] Abramsky, S. (2013). *Relational Databases and Bell's Theorem*, pages 13–35. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Abramsky et al., 2015] Abramsky, S., Barbosa, R. S., Kishida, K., Lal, R., and Mansfield, S. (2015). Contextuality, Cohomology and Paradox. In Kreutzer, S., editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 211–228, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Aji and McEliece, 2000] Aji, S. M. and McEliece, R. J. (2000). The generalized distributive law. *IEEE transactions on Information Theory*, 46(2):325–343.

[Almond and Kong, 1991] Almond, R. and Kong, A. (1991). Optimality issues in constructing a Markov tree from graphical models. *Res. Rep. A-3, Harvard Univ., Dept. of Stat.*

[Almond, 1995] Almond, R. G. (1995). *Graphical belief modeling*. CRC Press.

[Bishop, 2006] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer.

[Blelloch, 1990] Blelloch, G. E. (1990). Prefix sums and their applications.

[Cano and Moral, 1995] Cano, A. and Moral, S. (1995). Heuristic algorithms for the triangulation of graphs. In *Advances in Intelligent Computing—IPMU'94*, pages 98–107. Springer.

[Chu et al., 2007] Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281.

[Darwiche, 2001] Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41.

[Dasgupta and Abramsky, 2016] Dasgupta, A. and Abramsky, S. (2016). Anytime inference in valuation algebras. *CoRR*, abs/1605.04218.

[Date, 1975] Date, C. (1975). *An introduction to database systems*. Addison-Wesley publ.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

[Dechter, 1998] Dechter, R. (1998). Bucket elimination: A unifying framework for probabilistic inference. In *Learning in graphical models*, pages 75–104. Springer.

[Dechter and Rish, 2003] Dechter, R. and Rish, I. (2003). Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153.

[Dempster, 1968] Dempster, A. P. (1968). A generalization of Bayesian inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 205–247.

[Denoeux, 2000] Denoeux, T. (2000). A neural network classifier based on Dempster-Shafer theory. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(2):131–150.

[Feldman et al., 2010] Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., and Svitkina, Z. (2010). On distributing symmetric streaming computations. *ACM Transactions on Algorithms (TALG)*, 6(4):66.

[Golan, 2013] Golan, J. S. (2013). *Semirings and their Applications*. Springer Science & Business Media.

[Goodrich et al., 2011] Goodrich, M. T., Sitchinava, N., and Zhang, Q. (2011). Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation*, pages 374–383. Springer.

[Haenni, 2004] Haenni, R. (2004). Ordered valuation algebras: a generic framework for approximating inference. *International Journal of Approximate Reasoning*, 37(1):1 – 41.

[Haenni and Lehmann, 1999] Haenni, R. and Lehmann, N. (1999). *Efficient hypertree construction*. Citeseer.

[Haenni and Lehmann, 2002] Haenni, R. and Lehmann, N. (2002). Resource bounded and anytime approximation of belief function computations. *International Journal of Approximate Reasoning*, 31(1–2):103 – 154.

[Haller and Miller, 2011] Haller, P. and Miller, H. (2011). Parallelizing Machine Learning– Functionally: A Framework and Abstractions for Parallel Graph Processing. In *Scala Workshop 2011*.

[Harmanec, 1999] Harmanec, D. (1999). Faithful approximations of belief functions. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 271–278. Morgan Kaufmann Publishers Inc.

[Heskes et al., 2002] Heskes, T., Albers, K., and Kappen, B. (2002). Approximate inference and constrained optimization. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 313–320. Morgan Kaufmann Publishers Inc.

[Horsch and Poole, 1998] Horsch, M. C. and Poole, D. (1998). An anytime algorithm for decision making under uncertainty. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 246–255. Morgan Kaufmann Publishers Inc.

[Jensen et al., 1990] Jensen, F. V., Lauritzen, S. L., and Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational statistics quarterly*.

[Jøsang and Pope, 2012] Jøsang, A. and Pope, S. (2012). Dempster's rule as seen by little colored balls. *Computational Intelligence*, 28(4):453–474.

[Karloff et al., 2010] Karloff, H., Suri, S., and Vassilvitskii, S. (2010). A model of computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics.

[Kishida, 2016] Kishida, K. (2016). Logic of Local Inference for Contextuality in Quantum Physics and Beyond. In

Ioannis Chatzigiannakis, Michael Mitzenmacher, Y. R. and Sangiorgi, D., editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113:1–113:14, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Kohlas, 2003] Kohlas, J. (2003). *Information algebras: Generic structures for inference*. Springer Science & Business Media.

[Kohlas et al., 1999] Kohlas, J., Haenni, R., and Moral, S. (1999). Propositional information systems. *Journal of Logic and Computation*, 9(5):651–681.

[Kohlas and Wilson, 2008] Kohlas, J. and Wilson, N. (2008). Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artificial Intelligence*, 172(11):1360–1399.

[Koller et al., 2007] Koller, D., Friedman, N., Getoor, L., and Taskar, B. (2007). Graphical Models in a Nutshell. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*. MIT Press.

[Koller et al., 1999] Koller, D., Lerner, U., and Angelov, D. (1999). A general algorithm for approximate inference and its application to hybrid bayes nets. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 324–333. Morgan Kaufmann Publishers Inc.

[Lauritzen and Spiegelhalter, 1990] Lauritzen, S. L. and Spiegelhalter, D. J. (1990). Readings in uncertain reasoning. chapter Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems, pages 415–448. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Low et al., 2010] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). GraphLab: A New Parallel Framework for Machine Learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California.

[Mackey et al., 2008] Mackey, G., Sehrish, S., Bent, J., Lopez, J., Habib, S., and Wang, J. (2008). Introducing map-reduce to high end computing. In *Petascale Data Storage Workshop, 2008. PDSW'08. 3rd*, pages 1–6. IEEE.

[Malewicz et al., 2010] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 135–146, New York, NY, USA. ACM.

[McKenna et al., 2010] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., et al. (2010). The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9):1297–1303.

[Murphy, 1998] Murphy, R. R. (1998). Dempster-Shafer theory for sensor fusion in autonomous mobile robots. *Robotics and Automation, IEEE Transactions on*, 14(2):197–206.

[Pace, 2012] Pace, M. F. (2012). {BSP} vs mapreduce. *Procedia Computer Science*, 9(0):246 – 255. Proceedings of the International Conference on Computational Science, {ICCS} 2012.

[Paskin et al., 2005] Paskin, M., Guestrin, C., and McFadden, J. (2005). A robust architecture for distributed inference in sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 55–62. IEEE.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

[Pouly, 2010] Pouly, M. (2010). NENOK – A software architecture for generic inference. *International Journal on Artificial Intelligence Tools*, 19(01):65–99.

[Pouly and Kohlas, 2005] Pouly, M. and Kohlas, J. (2005). Minimizing communication costs of distributed local computation. Technical report.

[Pouly and Kohlas, 2011] Pouly, M. and Kohlas, J. (2011). *Generic Inference: A Unifying Theory for Automated Reasoning*. Wiley-Blackwell.

[Ramos and Cozman, 2005] Ramos, F. T. and Cozman, F. G. (2005). Anytime anyspace probabilistic inference. *International Journal of Approximate Reasoning*, 38(1):53–80.

[Rosen, 1999] Rosen, K. H. (1999). *Handbook of discrete and combinatorial mathematics*. CRC press.

[Russell et al., 2003] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (2003). *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River.

[Schneuwly, 2007] Schneuwly, C. (2007). *Computing in valuation algebras*. PhD thesis, Ph. D. thesis, Department of Informatics, University of Fribourg.

[Sentz and Ferson, 2002] Sentz, K. and Ferson, S. (2002). *Combination of evidence in Dempster-Shafer theory*, volume 4015. Citeseer.

[Shafer, 1982] Shafer, G. (1982). Belief functions and parametric models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 322–352.

[Shafer et al., 1976] Shafer, G. et al. (1976). *A mathematical theory of evidence*, volume 1. Princeton university press Princeton.

[Shenoy, 1992] Shenoy, P. P. (1992). Valuation-based systems: A framework for managing uncertainty in expert systems. In *Fuzzy logic for the management of uncertainty*, pages 83–104. John Wiley & Sons, Inc.

[Shenoy, 1997] Shenoy, P. P. (1997). Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of approximate reasoning*, 17(2):239–263.

[Shenoy and Shafer, 1990] Shenoy, P. P. and Shafer, G. (1990). Axioms for probability and belief-function propagation. *Machine Intelligence and Pattern Recognition*, 9:169 – 198. Uncertainty in Artificial Intelligence.

[Smith, 2003] Smith, P. (2003). *An introduction to formal logic*. Cambridge University Press.

[Solomonik et al., 2014] Solomonik, E., Carson, E., Knight, N., and Demmel, J. (2014). Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 307–318. ACM.

[Ueno et al., 2006] Ueno, K., Xi, X., Keogh, E., and Lee, D.-J. (2006). Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 623–632. IEEE.

[Valiant, 1990] Valiant, L. G. (1990). A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111.

[van de Ven and Ramos, 2012] van de Ven, J. and Ramos, F. (2012). Distributed Anytime MAP Inference. *arXiv preprint arXiv:1202.3767*.

[Williams et al., 1997] Williams, E., Santos, E., and Shimony, S. E. (1997). Experiments with distributed anytime inferencing: Working with cooperative algorithms. In *In Proceedings of the AAAI Workshop on Building ResourceBounded Reasoning Systems, 86–91*. Citeseer.

[Yu and Frincke, 2005] Yu, D. and Frincke, D. (2005). Alert confidence fusion in intrusion detection systems with extended Dempster-Shafer theory. In *Proceedings of the 43rd annual Southeast regional conference-Volume 2*, pages 142–147. ACM.

[Zilberstein, 1996] Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73.