On the Structural Link Between Ontologies and Organised Data Sets

ON THE STRUCTURAL LINK BETWEEN ONTOLOGIES AND ORGANISED DATA SETS

BY

ALICIA MARINACHE[1], B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Master of Applied Science

Master of Applied Science (2016) McMaster University

(Software Engineering) Hamilton, Ontario, Canada

TITLE:	On the Structural Link Between Ontologies and Organised Data Sets
AUTHOR:	Alicia Marinache[3] B.Eng. (Software Engineering) University Politechnica, Bucharest, Romania
SUPERVISOR:	Dr. Ridha Khedri
NUMBER OF PAGES:	x, 132

To my family

To my late father, who would have loved to see me pursuing my dreams

Abstract

"Relationships are complicated, sometimes interesting and can often appear unpredictable. This is just the type of situation that mathematics can help you with."

- John D. Barrow, *100 essential things you didn't know you didn't know*

The proposed work focuses on articulating a mathematical framework to capture the structure of an ontology and relate it to organised data sets. In the discussed framework, the ontology structure captures the mereological relationships between concepts. It also uses other relationships relevant to the considered domain of application. The organized dataset component of the framework is represented using diagonal-free cylindric algebra. The proposed framework, called the domain-information structure, enables us to link concepts to data sets through a number of typed data operators. The new framework enhances concurrent reasoning on data for knowledge generation, which is essential for handling big data. We illustrate the advantage of the obtained framework by using it in generating new knowledge from an ontology and a given data set.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Dr. Ridha Khedri for his patience, motivation, and the continuous support of my study and research. His guidance helped me during the time of research and writing of this thesis.

I also thank my fellow colleagues, especially Andrew LeClair and Dr. Jason Jaskolka, for being my sounding board, for the stimulating discussions, and for all the fun we have had during our group meetings.

Last but not the least, I would like to thank my family: my husband and my son for understanding the immensity of my returning to academia, and for fully supporting me every step of this journey; and to all my friends and family for spiritual support throughout writing this thesis and my life in general.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we provide an introduction to knowledge generation from data sets using formal ontologies, and discuss the motivation for the proposed research. In Section 1.1, we present a short introduction into the field of knowledge discovery and conjecture generation, and their importance in today's world. In Section 1.2, we review the specific context in which knowledge generation is currently used, focusing on ontological representations. In Section 1.3, we give the motivation for a mathematical framework for knowledge conjecturing from ontologies and structured data sets. In Section 1.4, we state the proposed research problem, and in section 1.5 we present its main contributions. The structure of the remainder of the thesis is given in Section 1.6.

1.1 General Context

In today's world, data is more available and abundant than ever. With easy access of humans to computers, data is exponentially increasing in size every moment and

information is recorded in various formats, with various grades of reliability. Original data manipulation methods were developed with smaller size of data in mind, more rigorous data validation, and less number of data types or formats. Therefore, there is a gap between the current data availability and its processing methods. Knowledge generation closes the gap by developing new, automated methods for discovering patterns in datasets, and organizing them in smaller and more manageable information.

1.1.1 What is Knowledge Generation?

Scientific discovery is the process of finding new knowledge, such as a new law or theory-exploration, or a new class of objects [DLT07]. Historically, discoveries have been made by individuals: domain experts or novices, stumbling upon empirical observations, or logically deducing new theories from existing ones. In recent years, the rise of modern computers and technologies has made room for new possibilities. Researchers have focused on a specific question: can we use automated systems to do the discoveries for us?

We find in [FPSS96b] that knowledge generation is "the nontrivial process of identifying *valid, novel, useful*, and ultimately *understandable* patterns in data". Thus, knowledge generation can be viewed as the process of searching (usually large) sets of structured or unstructured data to find patterns that allows further inferences to be made.

In our research, we take the notion of knowledge generation further from the 'finding new knowledge in an automated way' definition. We aim to only generate useful

knowledge within a domain. In this way, the focus shifts from simply finding new patterns within data sets to the discovery of new and interesting information within a specific application domain.

1.1.2 What is Knowledge Generation for?

Today, there is an unlimited number of possible applications for knowledge generation in the field of science and engineering. In [Rah14], we find an example of knowledge discovery in the medical field, using the electronic health records for the pharmaceutical industry. By finding patterns and relationships between various diseases and the medications used to treat them, new inferences can be made about how combining two different drugs can increase or decrease certain symptoms, or even produce new symptoms.

In [RV07], we find an example of using knowledge discovery in educational research. From the collected data, a system can predict future learning behaviour, discover new domain models, and discover learning patterns. This new information is used to provide feedback for instructors and to assist with planning, scheduling, grouping students and more.

In the field of astronomy, due to the sheer volume of data surveyed, and the complexity of the data, the current challenge is finding new methods for image analysis, classification, and cataloging of sky objects [FPSS96a]. Several examples of systems developed for knowledge discovery and data mining in astronomy are SKICAT [Ski] and AstroML [Ast].

In manufacturing, knowledge discovery is now used to diagnose, predict and prevent faults, or to support decision systems. Data is analyzed to identify and discover new patterns in various manufacturing processes, such as: production, quality control, maintenance, design, customer relationship management and more [HSSK06].

Knowledge discovery can be used in agriculture for classification, for using pesticides optimally, or to predict and detect diseases. By recording weather-related properties, such as temperature and rain volume, the systems can predict certain plant diseases. In [UPCMA04], we find that by analyzing plant properties, such as sugar, acids, and nitrogen sources, a system can predict how good the fermentation process will be in a specific year. In [LD04], we learn about knowledge discovery tools that are used to classify and grade produce, and to identify defects prior to shipping them. Another example is the detection of disease and prevention of its spread in animals [AJH+04].

Another field where knowledge discovery is widely used is the business and financial domain. In [SSTW01], we find an example of using knowledge discovery in marketing. Due to the advances made in information technology, businesses have access to large volumes of information about their customers. The knowledge hidden in these sets of data is critical; businesses can now use it in direct, targeted marketing for analysing and predicting customer behaviour, to predict future sales trends, to make basket and cohort analysis. In [PLSG05], we find an example of knowledge discovery systems used in fraud detection, by identifying possible fraudulent information on service application, or by detecting fraudulent or possible illegal activities. In the

field of investment, the challenges are similar to the ones in other domains: vast amounts of data are continuously collected. Various knowledge discovery methods and techniques are now being used to analyze and forecast stock data, predict future stock prices, as well as build decision support systems for the financial investment industry [LK08]. In the business performance field, the organizations' ability to rapidly analyze, predict, and improve their performance is vital to the well being of the company. The size of data and the need of 24x7 availability has created another problem for businesses: the costs of storage capabilities are rising. Knowledge discovery in business performance is used to analyse and better understand the organization, as well as to reduce the size of stored data [TSD10].

The gaming industry is another example of an industry taking advantage of the modern techniques employed by knowledge discovery and data mining. There are many ways knowledge discovery can be used, such as: improving game design, preventing fraud, cutting production cost, predicting user preferences and usage trends, increasing the customer renewal [Ken03]. In [SNA], we learn that as one of the most recent techniques in modern sociology, social network analysis is now employed by a large number of sciences, such as *anthropology, biology, communication studies, economics, geography, information science, organizational studies, social psychology, and sociolinguistics*. There is a growing list of knowledge discovery tools and systems used in social network analysis.

1.1.3 How is Knowledge Generation needed?

While similar to pure information extraction, knowledge generation takes the process a step further. It creates abstractions (high-level knowledge) of the low-level data. Often the result of the knowledge generation process is fed back into the process of discovery.

There are other processes concerned with information extraction, such as data mining, pattern recognition and machine learning. Knowledge discovery combines all these previous methods into one complex and complete process. Its focus is on the process of knowledge generation: from preparing the data, to cleaning it, and using data mining techniques to discover new patterns. It is concerned with the scalability and efficiency of algorithms processing large data sets, as well as the result of the data mining: performing visualization and interpretation of the resulting knowledge.

Besides the obvious need for analysing the raw data to discover knew information, a new need has risen in the information systems domain, one related to the security of information. Through knowledge discovery processes we can infer tacit (non-explicit) links between knowledge fragments that put together could give unauthorized access to private information. With the increasing capabilities of knowledge generation and discovery, the information systems must start thinking about protecting themselves against unauthorized and undesirable discovery of knowledge from existing and presumably very secured data [O'L91].

A more recent aspect of the analysis and discovery field is the size of the data. With

the advance of technology in terms of storage capabilities and the relative low cost associated with it, many organizations store record amounts of data. This data eventually needs to be analyzed. In information intensive fields, such as medical, financial, astronomy, and audio-visual, it is common to accumulate terabytes of raw data on a regular basis, thus making storage itself a possible problem. The need to intelligently compress data is becoming imperative. Knowledge generation could help with compression, by recognizing valuable patterns in the raw data, and replacing the actual raw information with specialized theories and relationships.

1.2 Specific Context

1.2.1 Ontologies and knowledge generation

In order to generate knowledge, we need to first define its representation. Albeit a very familiar concept in the automated discovery domain, knowledge representation has only recently been clearly defined. In [Dav93] the author argues there are five aspects of knowledge representation, each with a distinctive role to play. The first aspect describes knowledge representation as a *surrogate* for the knowledge itself, used to reason about concepts and how they describe the world. The second way to describe knowledge representation is as a *set of ontological commitments*, used to answer the question *In what terms should I think about the world?* The third aspect is a *fragmentary theory of intelligent reasoning*, used to express the knowledge as a set of axioms, and, together with a reasoning engine, make new conjectures. The fourth way to describe knowledge representation is as a *medium for pragmatically efficient computation*, used in automating the process of reasoning, and achieved by

organizing the existing knowledge. The fifth and last aspect is described as a *medium of human expression*, used to understand (and later analyze) the natural language in which the knowledge is expressed.

In our thesis, we use a combination of the second and third aspects, in that we view the world through an ontological structure, further represented as a mathematical theory.

Ontologies, defined in [Gru95] as the *explicit specifications of a conceptualization of a domain*, have been used as a mean to knowledge representation in many areas of knowledge systems, such as knowledge acquisition, knowledge reuse and sharing, verification and validation of knowledge based system, as well as for knowledge discovery paired with reasoning engines [BC98, Gru95, HSW97, PB01].

In this thesis the focus is on knowledge generation from structured data sets, with the knowledge represented by an ontology. As with knowledge representation, there are many definitions of what an ontology is, from the classical philosophical term introduced by Aristotle, to the computer science terms in modern age. In [Sow00], Sowa describes ontology as *the study of all existence, of all kinds of entities - abstract and concrete - that make up the world*. He makes the separation between the structure of the ontology (the part that provides knowledge about the world) and the reasoning on the ontology (the part that makes sense of the structure through a framework of abstractions). Throughout the rest of the thesis we use the term *ontology* in the knowledge engineering sense, not in the philosophical sense.

Ontologies have been generally developed as a mean to describe and use domain-specific knowledge. They are viewed as content theories describing a specific domain of knowledge: the concepts found in the domain, their properties, and relationships. Today there are many such domains that make use of ontologies: physics, chemistry, medicine, linguistic, music, etc.

The abstraction level of an ontology varies from general terms, that describe the world across all domains, to domain specific terms, that are restricted to specific knowledge domains [CJB99]. The general-descriptive terms are called *the upper ontology*, and are defined as an ontology which deals with domain-independent concepts and relationships. These ontologies are used as a template to engineer domain-specific ontologies. Currently there are many upper-level ontologies, such as: the Basic Formal Ontology [BFO], CYC [CYC], the Generalized Upper Model [GUM] and more. The Basic Formal Ontology (BFO) consists of a number of sub-ontologies at different levels of granularity. The CYC is a general knowledge base and common-sense reasoning engine, a proprietary system under development since 1986. The Generalized Upper Model (GUM) is a general task and domain independent linguistic ontology, intended for organizing information for expression in natural language.

When we focus on ontologies as content theory, we realize that once a good content theory of the domain is devised, it can be used with different mechanism theories in order to implement effective systems [CJB99]. By clearly defining the knowledge structure of a specific domain, the ontology becomes the foundation of its knowledge representation. By defining both the vocabulary of the domain, as well as the

relationships between its concepts, the ontology allows for natural and unified reasoning processes. Through their vocabulary and structure, ontologies enable knowledge sharing, either within the same domain, or across domains.

Besides representing the world around us and sharing knowledge, ontologies are being used in two areas rich in knowledge: natural language understanding and knowledge generation. For the first purpose, natural language understanding, we generally need both an upper-ontology and a domain-specific ontology (to focus on domain-specific concepts and relationships). For the second purpose, currently most of the work has been focused on domain-specific knowledge, and the discovery of new knowledge from them.

1.2.2 Conjecture Generation

So far we have explored the relationship between knowledge generation and data mining, in which data mining is an important part of the knowledge generation process. When we focus on the domain of mathematics we can similarly review the relationship between knowledge generation and conjecture generation. Conjecture generation is making inferences about mathematical knowledge.

In the context of pure mathematics, the knowledge source is viewed as a set of values along a collection of axioms (definitions, relations and constraints) [FT12]. Conjecture generation is providing the tools to reason explicit new knowledge by finding relations between the axioms and proving (or disproving) them.

While knowledge is now routinely discovered in many science fields, the mathematics field has been approached mainly from the theorem proving side, in which conjectures are already known and only need proving or disproving. So far, most of the discovery work in the mathematics field has been focused on the automated re-discovery of known theories, and very little on the discovery of new theories. In [MBA07], McCasland states that it is possible to automate the discovery of inductive theorems. Similarly, Johansson demonstrated it is possible to automatically generate conjectures and produce most of the theorems already available in the Isabelle libraries [JDB11].

Other works have started to explore the automation of theory discovery, with the explicit goal of making novel discoveries. In his work with the HR system, Colton [CBW99] uses an exhaustive heuristic search to make and prove new conjectures in the field of finite algebras. In [PGS06], Puzis is using a more formal approach to generate new conjectures from a set of axioms. Work has been done in other mathematical domains, such as graph theory, with one example being the GT (Graph Theorist) system [Eps88].

All conjecture generation processes start from a set of known knowledge about the mathematical domain (defined as axioms) and use various strategies to generate new statements that are either true from the start or need to be proved. For the latter, conjecture generation tools are used in conjunction to Automatic Theorem Provers (ATP) to prove or disprove the generated conjectures. Thus *validity* of new information is ensured right after the generation of conjecture. In order to ensure the three remaining properties of new discoveries (as defined in [FPSS96b]), a series of filters

are employed during the process.

Automated discovery uses both data and theories: while knowledge discovery is more geared towards using sets of data, conjecture generation uses existing theories to discover new theorems and theories. Each domain is represented by a set of objects, and classes or categories. However, in order to become a theory, a set of concepts must also contain a set of theorems and proofs [CBW99].

By formalizing an ontology and its structured data set into a mathematical structure, we can use existing conjecture generation methods in order to infer on information system and discover knew knowledge.

1.3 Motivation

Traditionally, knowledge and conjecture generation have been used solely for analysis purposes. We see the possibility to use them in the domain of semantic data compression, by recognizing valuable patterns in the raw data, and replacing the actual raw information with specialized theories and relationships.

There seems to be a disconnect between organizing domain knowledge hierarchically, through ontologies, and using the ontologies as the basis of automated reasoning engines.

We see value in defining an ontology as a mathematical structure and the theory associated with it, thus making it possible to use existing conjecture generation methods

from theory as the reasoning engine on existing data. By separating the domain representation from the reasoning engine, we can focus on describing an information structure theory that can be reasoned upon using existing methods and algorithms. By proposing a new mathematical framework to capture the structure of an ontology and the link to a given organised set of data, we set a solid foundation that can be later used by researchers to (1) engineer ontologies in a formal setting, or (2) employ existing methods and algorithms to generate new knowledge through the reasoning engine.

1.4 Problem Statement

The problem that we focus on in this thesis is the articulation of a mathematical framework to formally describe the structure of an ontology, as well the information system encompassing the data set and its structure and to illustrate the use of the obtained structure for knowledge discovery.

Our first objective is to articulate a mathematical structure that captures an ontology. We observe that several relationships within an ontology could lead to a lattice of concepts. Others would only lead to a family of partial orders. Our aim is to capture both into one mathematical structure that gives an *abstract ontology*.

The second objective is to put together the system that encompasses the ontology structure and data set, on which the knowledge extraction is performed. The relational model of data has been accepted as a clear model for relational databases. We

propose a new mathematical structure, based on Tarski's cylindric algebra, to describe an algebraic data structure. We then link the two structures, using the typed data operators, thus forming a new mathematical structure, which we call the *domain information structure*. The proposed framework can be used further to reason on the information system. Therefore, we need to identify the axioms governing the theory of information structure, as well as identify and prove its theorems.

Our third objective is to illustrate the capturing of the details of the domain in order to enable basic reasoning. We show how the ontology can be enhanced through its specification, and how multiple interpretations can be applied on the same ontology structure and same data set. This further enables the use of existing reasoning method and algorithms to generate new knowledge.

1.5 Main Contributions

The main contributions of my thesis include:

(i) The mathematical formalization of the information system encompassing the ontology structure and the oragnised data set; the new structure can be used to automate knowledge extraction.

(ii) The separation of the information system from the reasoning engine; this allows for applying different reasoning engines for different tasks, without having to modify the information system.

(iii) The ability to concurrently interpret a large set of data through different theories, specific to various sectors and domains; the proposed framework allows for

rapid analysis and knowledge generation.

1.6 Structure of the Research Thesis

The remainder of this research paper is organised as follows:

Chapter 2 introduces the required mathematical background.

Chapter 3 provides an examination of the existing literature on similar mathematical approaches, discusses their limitations.

Chapter 4 introduces the new mathematical structures, along the domain information structure theory.

Chapter 5 illustrates an example, and discusses the strengths and weaknesses of the proposed research.

Chapter 6 draws conclusions and discusses possibilities for future work stemming from the proposed research.

Chapter 2

Mathematical Background

In this chapter we provide the mathematical background needed to make this thesis self-contained. In Section 2.1 we examine homogeneous relations and their representations. In Section 2.2, we detail the needed background on partial orders. In Section 2.3, we present lattices and their algebraic properties. In Section 2.4, we present a variety of algebraic structures, such as: monoid, semigroup, and Boolean algebra. In Section 2.5 we introduce Tarski's Cylindric algebra. In Section2.6 we discuss mathematical structures, followed by the algebraic specifications in Section2.7.

2.1 Homogeneous Relations

The following definitions and results are borrowed from [SS91].

Relations indicate how two or more things connect or relate to each other. When the elements that are connected are of the same set, we call the relation homogeneous. In this work we only consider binary relations.

Let C be a set. A *(homogenous) relation* R on C is defined as $R \subseteq C \times C$.

Relations are sets of tuples, thus we can use all the set operations, such as intersection, union, complement, inclusion etc. For example, the relation *greater than* can be defined on the set of natural numbers as $R = \{(x, y) \mid x, y \in \mathbb{N} : x > y\}$.

Any homogeneous relation can be represented as a directed graph by interpreting the elements of the underlying set as the vertices of the graph and the tuples of R as the edges of the graph.

Definition 2.1.1. ([SS91])
A graph $G = \big(C, R\big)$ consists of a set C of vertices and the associated transition relation $R \subset C \times C$. □

The *empty relation* $\emptyset \subseteq C \times C$ is denoted by O (or O_C when it is not clear what the underlying set is), and the *univeral relation* is $L = C \times C$ (or L_C for clarity). Another distinguished relation on C is the *identity relation*, $\mathbb{I} = \{(x, x) \mid x \in C\}$.

Definition 2.1.2. ([SS91])
Let $R, S \subseteq C \times C$ be relations. Their *product* $R;S \subseteq C \times C$ is given by:
$R;S = \{(x, z) \mid \exists(y \mid y \in C : (x, y) \in R \wedge (y, z) \in S)\}$. □

We call a product of relations the *composition* of relations. The composition is also written as $R;S$ or RS. We write $R^2, R^3, \ldots$ for the powers of R. The composition of

relation is associative and $\mathbb{I}$ is its identity element.

Definition 2.1.3. ([SS91]) Let R be a binary relation on C.

- R is *reflexive* $\iff I \subset R \iff R = R \cup I \iff \forall(x \mid x \in C : (x,x) \in R)$
- R is *symmetric* $\iff \forall(x,y \mid x,y \in C : (x,y) \in R \implies (y,x) \in R)$
- R is *antisymmetric* $\iff \forall(x,y \mid x,y \in C : ((x,y) \in R \wedge (y,x) \in R \implies x = y)$
- R is *transitive* $\iff R^2 \subseteq R \iff \forall(x,y,z \mid x,y,z \in C : (x,y) \in R \wedge (y,z) \in R \implies (x,z) \in R)$ □

Definition 2.1.4. ([SS91])
Let $R \subseteq C \times C$ be a relation. We define its transitive closure as
$R^+ \stackrel{\text{def}}{=} sup_{i \geq 1} R^i = inf\{H \mid R \subseteq H \wedge H \text{ transitive}\}$. □

We can say the following about relations and their transitive closures:

$$R \; transitive \iff R^+ \subset R \iff R^+ = R \tag{2.1}$$

2.2 Posets

Order deals with various notions, such as: precedence, preference, progression. In mathematical terms, an ordering on a set of objects is a *binary relation* that is transitive and antisymmetric. If we consider the order relation as non-strict, we can add

reflexivity. The following definitions and results are borrowed from [DP90].

Definition 2.2.1 ([DP90], pg. 2)**.** Let P be a set. An *order* (or *partial order*) on P is a binary relation on P that is reflexive, antisymmetric, and transitive. □

A set P equipped with an order relation $\leq$ is called an *ordered set* (or a *partial ordered set*, or a *poset*) and it is written as $\big(P, \leq\big)$.

Definition 2.2.2 ([DP90], pg. 7)**.** Let P be an ordered set and let $x, y \in P$. We say x is *covered by* y (or y *covers* x), and write $x \prec y$ or $y \succ x$, if $x < y \wedge \ \forall(z \ \mid \ z \in P \ : \ x \leq z < y \implies z = x\,)$. □

Given an ordered set P we can form a new ordered set P^d (the *dual of* P) by defining $x \leq y$ to hold in P^d iff $y \leq x$ holds in P. Given a statement Φ about ordered sets which is true in all ordered sets, then the dual statement Φ^d is true in all ordered sets, and it is obtained by replacing all occurrences of $\leq$ by $\geq$ and vice versa.

Definition 2.2.3 ([DP90], pg. 14)**.** Given an arbitrary set $Q \subseteq P$ and $x \in P$ we define

- $\uparrow Q = \{y \in P \mid \ \exists(z \ \mid \ z \in Q \ : \ y \geq z\,)\}$, and we call it *up* Q (dual: *down* Q)
- $\uparrow x = \{y \in P \mid y \geq x\}$ (dual: $\downarrow x$) □

Definition 2.2.4 ([DP90], pg. 15)**.** Let P be a poset and $Q \subseteq P$. We call $a \in Q$ a *maximal element* iff $\forall(x \ \mid \ x \in Q \ : \ a \leq x \implies a = x\,)$ (dual: *minimal element*).

We call $a \in Q$ the *greatest* (or *maximum*) element iff $\forall(x \mid x \in Q : x \leq a)$ or a is a maximal element in Q, comparable to all elements in Q (dual: the *least* element). If the *greatest element* of P exists, it is also called *top element* and it is written $\top$ (dual: *bottom element*, $\bot$). □

If P is the power set of a set X, with the $\subseteq$ as the ordering relation, then $\top(\mathcal{P}(X)) = X$ and $\bot(\mathcal{P}(X)) = \emptyset$.

2.3 Lattices

Definition 2.3.1 ([DP90], pg. 27)**.** Let P be a poset and let $S \subseteq P$. An element $x \in P$ is an *upper bound* of S if $\forall(s \mid s \in S : s \leq x)$ (dual: *lower bound*). The set of all upper bounds for S is denoted by S^u (read "S upper") and it is written as $S^u = \{x \in P \mid \forall(s \mid s \in S : s \leq x)\}$ (dual: "S lower", S^l). □

If S^u has a least element, x, then x is called the *least upper bound* or *supremum* of S, $sup(S)$ (dual: *greatest lower bound*, *infimum*, $inf(S)$). When $S = P$, if $\top$ of P exists, $P^u = \{\top\}$ and $sup(P) = \top$. Dually, if $\bot$ of P exists, $P^l = \{\bot\}$ and $inf(P) = \bot$. When $S = \emptyset$, $\emptyset^u = P$ and, if $\bot$ of P exists, $sup(\emptyset) = \bot$. Dually, if $\top$ of P exists, $inf(\emptyset) = \top$.

If $sup\{x, y\}$ exists, we write it as $x \vee y$ or x *join* y. Dually, if $inf\{x, y\}$ exists, we write it as $x \wedge y$ or x *meet* y. Similarly, we write $\bigvee S$ (the *join* of S) for $sup(S)$ and $\bigwedge S$ (the *meet* of S) for $inf(S)$, respectively.

In a poset P, the least upper bound, $x \vee y$ of any two elements, $\{x, y\}$ may fail to exist because x and y have no common upper bound or they have no least upper bound. For example, let $P = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}\}$, with $\subseteq$ the ordering relation. The partial order is represented in Figure 2.1. We see that $\{a\} \wedge \{b\} = \emptyset$, and similarly, $\{a\} \wedge \{c\} = \emptyset$ and $\{b\} \wedge \{c\} = \emptyset$, respectively. Thus, any pair of 2 elements in P has a meet. We see that $\{a\} \vee \{b\} = \{a, b\}$, however, neither the $\{a, c\}$ pair, nor the $\{b, c\}$ pair has a join; there is no set element in P that contains either of these two pairs. Hence P is not a lattice, the meet and join do not exist for any pair of elements of P.

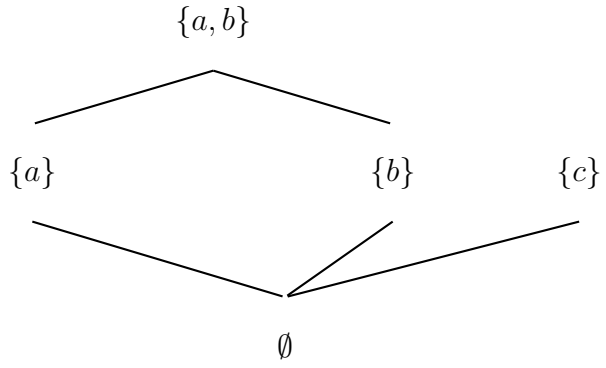


Figure 2.1: Poset

Definition 2.3.2 ([DP90], pg. 29)**.** Let P be a non-empty poset. P is called a *lattice* if $\forall(x, y \mid x, y \in P : (x \vee y)$ *and* $(x \wedge y)$ *exist*$)$. P is called a *complete lattice* if $\forall(S \mid S \subseteq P : \bigvee S$ *and* $\bigwedge S$ *exist*$)$. □

In the lattice definition, S = $\emptyset$ is allowed. Since $\bigvee\emptyset = \bot$ and $\bigwedge\emptyset = \top$, any complete

lattice is bounded, or it has bottom and top elements.

The following definitions and results are borrowed from [DP90].

Definition 2.3.3 ([DP90], pg. 131)**.** Let L be a lattice. L is said to be *distributive* if it satisfies the *distributive law*

$\forall(a, b, c \mid a, b, c \in L : a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c))$. □

Definition 2.3.4 ([DP90], pg. 143)**.** Let L be a lattice with a top $\top$ and a bottom $\bot$. For every $a \in L$, we say $b \in L$ is a *complement* of a if $a \wedge b = \bot$ and $a \vee b = \top$. If a has a unique complement, we denote this complement by a'. □

Definition 2.3.5 ([DP90], pg. 143)**.** A lattice L is called a *Boolean lattice* if

i L is distributive

ii L has a top and a bottom

iii $\forall(a \mid a \in L : !\exists(a' \mid a' \in L : a' \text{ is complement of } a))$ □

Definition 2.3.6 ([DP90], pg. 163)**.** Let L be a lattice with bottom element $\bot$. Then $a \in L$ is called an *atom* if $\bot \prec a$. The set of all atoms in L is denoted by $\mathcal{A}(L)$. □

Corollary 2.3.1 ([DP90], pg. 164)**.** Let L be a finite lattice. Then the following statements are equivalent:

i L is a Boolean lattice

ii L is order-isomorphic to the powerset of its atoms

iii $\forall(a \mid a \in L : !\exists(a' \mid a' \in L : a' \text{ is complement of } a))$ □

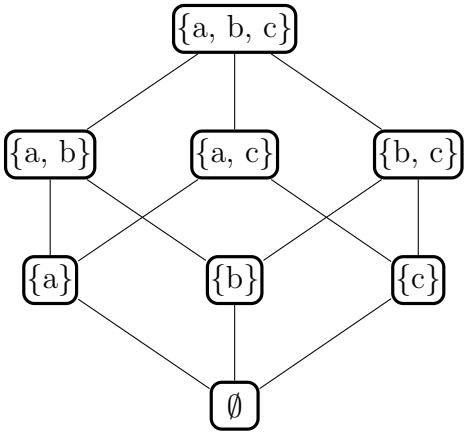


Figure 2.2: Boolean lattice for $\mathcal{P}(\{a, b, c\})$

In Figure 2.3 we show an example of a Boolean lattice isomorphic to the powerset of the set of atoms $\{a, b, c\}$.

2.4 Algebraic structures

The following definitions and results are borrowed from [BM41]. Let B be a set of elements, and let there be a binary operation $\diamond$ on B. The operation $\diamond$ is said to be *idempotent* iff $\forall(x \mid x \in B : x \diamond x = x)$. The operation $\diamond$ is said to be *commutative* iff $\forall(x, y \mid x, y \in B : x \diamond y = y \diamond x)$. The operation $\diamond$ is said to be *associative* iff $\forall(x, y, z \mid x, y, z \in B : x \diamond (y \diamond z) = (x \diamond y) \diamond z)$.

Two binary operations $\diamond$ and $\star$ on B are said to satisfy the *absorption law* iff $\forall(x, y \mid x, y \in B : x \diamond (x \star y) = x \star (x \diamond y) = x)$. They are said to be *mutually distributive* iff $\forall(x, y, z \mid x, y, z \in B : x \diamond (y \star z) = (x \diamond y) \star (x \diamond z) \wedge x \star (y \diamond z) = (x \star y) \diamond (x \star z))$.

On a set B equipped with a binary operation $\diamond$ we call *identity element* a distinguished element, $e \in B$, such that $\forall(x \mid x \in B : x \diamond e = e \diamond x = x)$.

Definition 2.4.1. A *semigroup* is an algebraic structure, $\big(B, \diamond\big)$, where B is a set of elements and $\diamond$ is an associative binary operation. The semigroup is called *commutative* if the operator $\diamond$ is commutative. □

Definition 2.4.2. A *monoid* is an algebraic structure $\big(B, \diamond, e\big)$, where $\big(B, \diamond\big)$ is a semigroup and e is the identity element over $\diamond$. The monoid is called *commutative* if $\diamond$ is commutative. □

Definition 2.4.3. A **Boolean algebra** is an algebraic structure

$\mathcal{B} = \big(B, +, \cdot, -, 0, 1\big)$,

where $+$ and $\cdot$ are binary operations on B, $-$ is a unary operation on B, called complement, 0 and 1 are distinguished elements of B, and the following postulates are satisfied:

(B1) $\big(B, +, 0\big)$ is a commutative monoid;

(B2) $\big(B, \cdot, 1\big)$ is a commutative monoid;

(B3) $+$ and $\cdot$ are mutually distributive;

(B4) $\forall(x \mid x \in B : x + -x = 1 \wedge x \cdot -x = 0)$. □

Additionally, on a Boolean algebra, the two distinguished elements satisfy the following two axioms, called the *annihilator* for $+$ and $\cdot$, respectively:

- Annihilator for $+$: $\forall(x \mid x \in B : x + 1 = 1)$
- Annihilator for $\cdot$: $\forall(x \mid x \in B : x \cdot 0 = 0)$

On a Boolean algebra we can define the ordering relation, and the strict ordering relation, as follows:

$$x \leq y \overset{\text{def}}{\iff} x + y = y \overset{\text{def}}{\iff} x \cdot y = x \tag{2.2}$$

2.5 Cylindric Algebras

The notion of *cylindric algebra* has first been presented by Alfred Tarski in [THM71], following the development of Boolean algebra. While Boolean algebra developed algebraic theories to represent propositional calculus, a new theory was needed to model quantification and equality. By adding the cylindrification operator for quantification, and the diagonal element for equality, Tarski was able to build an algebraic structure for representing first-order logic with equality. In doing so, he created an algebraic theory accessible to mathematicians who do not have a detailed knowledge of the logical apparatus.

In our work we are only interested in the cylindrification operator, hence we focus our mathematical background on diagonal-free cylindric algebras.

Definition 2.5.1 ([THM71])**.** A *diagonal-free cylindric algebra of dimension* α is an algebraic structure

$\mathcal{B} = \big(B, +, \cdot, -, 0, 1, c_k\big)_{k<\alpha}$,

where $k, \alpha \in \mathbb{N}$, $+, \cdot$ are binary operations on B, $0, 1$ are distinguished elements of B, $-, c_k$ are unary operations on B, and the following postulates are satisfied for any $x, y \in B$, and any $k, \lambda \in \mathbb{N}$ with $k, \lambda < \alpha$:

(C1) the structure $\big(B, +, \cdot, -, 0, 1\big)$ is a Boolean algebra

(C2) $c_k 0 = 0$

(C3) $x \leq c_k x$

(C4) $c_k(x \cdot c_k y) = c_k x \cdot c_k y$

(C5) $c_k c_\lambda x = c_\lambda c_k x$ □

Axiom (C2) expresses the normality of the c_k operator, and axiom (C3) expresses the generalization aspect of the c_k operator. Axiom (4) is a close analogy of the modular law in lattices. Axiom (5) expresses the commutativity of the c_k operators.

Cylindric algebras can be used to model an information system, by interpreting the elements of B as pieces of information, the $\cdot$ operator as combination of information, and the 0 as "no information". In this context, axiom (C2) ensures that the cylindrification of no information yields no information. Axiom (C3) shows that through cylindrification the information is expanded, not restricted, thus the cylindrification of any piece of data must contain that piece of data.

Proposition 2.5.1 ([THM71])**.**

(1) $c_k x = 0 \iff x = 0$

(2) $c_k 1 = 1$

(3) $c_k c_k y = c_k y$

(4) $x \cdot c_k y = 0 \iff y \cdot c_k x = 0$

(5) $x \leq y \rightarrow c_k x \leq c_k y$

(6) $x \leq c_k y \iff c_k x \leq c_k y$ □

The proofs for the above results can be found in [THM71].

To better understand cylindric algebras, we give its geometrical interpretation as a k-dimensional cylindric algebra that represents the universe of all k-dimensional objects (a set of k-dimensional points) on which the Boolean operators (intersection, union, negation) create new solids [THM71]. The operation of cylindrification on the i-th dimension can be understood as a projection of the solid on the k-1 space that is being extended to the whole "cylinder" along the removed dimension. Figure 2.3 shows the representation of a two-dimensional cylindric algebra. In Figure 2.4 we give the geometrical interpretation of axiom $(C4)$ in a two-dimensional cylindric algebra.

In [IL84], the authors show there exists a natural embedding of the relational algebra described below and the diagonal-free cylindric set algebra. Since its introduction in [Cod70], Codd's relational model of data has been accepted as a clear and succinct model for relational databases.

Let $\mathcal{U}$ be a fixed set of attributes, $A_1, A_2, \cdots, A_n$. We call a set of attributes $J \subseteq \mathcal{U}$ a *type*. With each attribute $A_i \in \mathcal{U}$ there is associated a non-empty *attribute domain*, $D(A_i)$. A *relation of type* J is a set of tuples $R \subseteq \Pi_{A_i \in J} D(A_i)$; an element $t \in R$ is

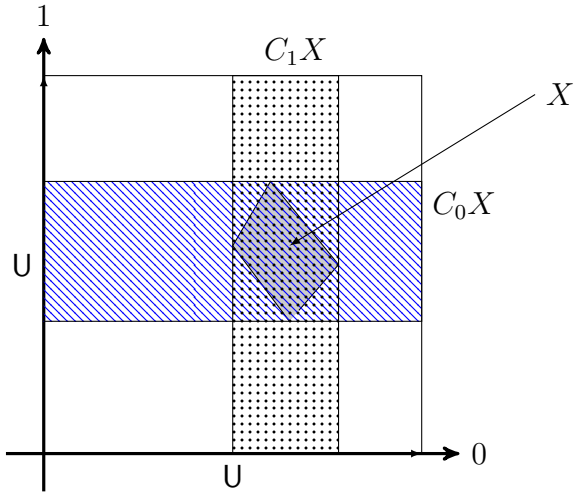


Figure 2.3: Cylindric algebra: geometrical representation [THM71]

called a tuple (of type J). For a tuple t in R, we write $\tau(R) = \tau(t) = J$, and we call the τ operator the type of R (or t, respectively). We say that a tuple t of type J is a mapping, associating a value $t(A_i) \in D(A_i)$ with each attribute $A_i \in J$. A restriction of the mapping to $K \subseteq J$ is written as $t[K]$.

In database theory, a relation of type J is generally assumed to be finite and it is represented as a table with columns representing each attribute in J, and rows corresponding to tuples. The following basic relational operators are defined:

- *Projection* ("vertical" decomposition): $\pi_K(R) = \{t[K] \mid t \in R\}$, where $K \subseteq \tau(R)$
- *Selection* ("horizontal" decomposition): $\sigma_E(R) = \{t \in R \mid E(t) \iff true\}$, where E is the selection condition, usually defined as a logical formula where the atomic conditions are of the form $(A_i = a), a \in D(A_i)$ or $(A_i = A_j), A_i, A_j \in \mathcal{U}$.
- *Union* (the usual set theory union): $R \cup Q$

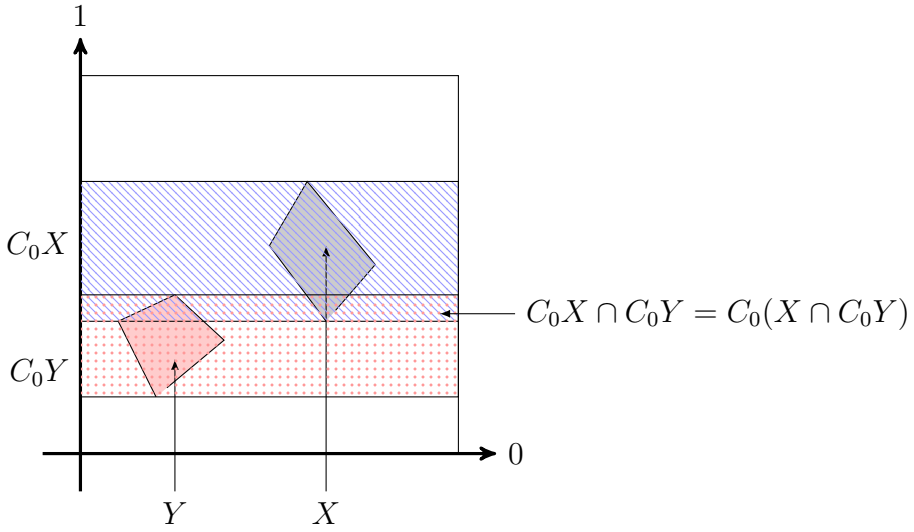


Figure 2.4: Cylindric algebra: geometrical representation of axiom (C4). [THM71]

- *Join* (natural join): $R \bowtie Q = \{t \mid \tau(t) = J \cup K \wedge t[J] \in R \wedge t[K] \in Q\}$

The following results are borrowed from [IL84].

Let R be any relation, let $J = \tau(R)$, and let us define $\mathbb{1} = \Pi_{A_i \in J} D(A_i)$ (the universe of tuples). The mapping

$$h(R) = \{t \in \mathbb{1} \mid t[J] \in R\}$$

is obtained by extending every tuple in R to the entire set of attributes in $\mathcal{U}$ (in all possible ways). It is easy to observe that $h(R)$ can be considered an element of a diagonal-free cylindric set algebra of subsets of $\mathbb{1}$, with the cylindrification operator corresponding to attributes in $\mathcal{U}$.

On a cylindric algebra, we consider generalized cylindrification on a subset of multiple dimensions. Let $J = \{k_1, k_2, \cdots k_n\}$, such that $\forall(i, k_i \mid i \in \mathbb{N}, k_i \in J : i \leq n \wedge k_i <$

α).

$$c_{(J)}x \stackrel{\text{def}}{=} c_{k_1}c_{k_2}\cdots c_{k_n}x$$

The mapping h defines a natural embedding of the relational algebra into the diagonal-free cylindric set algebra of subsets of $\mathbb{1}$.

Theorem 2.5.2 ([IL84])**.**

(a) $h(\pi_K(R)) = c_{(\mathcal{U}-K)}h(R)$

(b) $h(\sigma_E(R)) = \sigma_E(h(R)) = h(R) \cap \sigma_E(\mathbb{1})$

(c) $h(R \cup Q) = h(R) \cup h(Q)(\tau(R) = \tau(Q))$

(d) $h(R \bowtie Q) = h(R) \cap h(Q)$ □

The proof of the above results can be found in [IL84].

Part (a) of the theorem shows that we perform a projection not by shrinking the relation (through removal of one or more attributes), but by expanding it with every possible value of the removed attributes. In [GWW09], the authors observe this method of projection bridges the gap between relational database and ontologies. In traditional relational models, the absence of data is generally treated as negative information, while in knowledge systems it is treated as absence of knowledge. For example, in a database, two address rows (tuples) related to a person, Jane, can be interpreted as 'Jane has exactly 2 homes'. In an ontology, the same information is be

interpreted as 'Jane has at least 2 homes'. Thus, ontologies make no assumption about facts that have not been explicitly described; this open-world behaviour is easily captured by the cylindrification operator, while the projection would have inaccurately restrict the information. The removed attributes describe the part of knowledge we do not explicitly possess, we may assume any value for those attributes.

2.6 Mathematical structures

In [Mar00], we find that a mathematical structure is a set equipped with a collection of functions, relations, and distinguished elements. To describe such a structure we use a language that has symbols for the operations (or functions), relations, and constants (or distinguished elements). The language allows us to write statements and interpret them.

Given a set S, we call $f : S^n \to S$ a function of arrity n, or an n-ary function on S. Similarly, we call $R \subseteq S^n$ an n-ary relation.

Definition 2.6.1. ([Mar00]) We call *signature* and we write
$\Sigma = \big(\mathcal{F},\ \mathcal{R},\ \mathcal{C},\ \{n_f\}_{f\in\mathcal{F}},\ \{n_R\}_{R\in\mathcal{R}}\big)$, where $\mathcal{F}$ is a set of function symbols f with positive integers n_f the arrity of each $f \in \mathcal{F}$, $\mathcal{R}$ is a set of relation symbols R with positive integers n_R the arrity for each $R \in \mathcal{R}$, and $\mathcal{C}$ is a set of constant symbols. □

For example, the signature of rings is defined as $\Sigma_r = \big(\{+,-,\cdot\},\ \emptyset,\ \{0,1\},\ \{2,2,2\},\ \emptyset\big)$, where $+,-,\cdot$ are binary function symbols and $0, 1$ are constants.

Definition 2.6.2 ([Mar00])**.** Let Σ be a signature and let M be a non-empty set, called the universe or domain. We define the *interpretations* of the symbols f, R, c as a set of functions $f^{\mathcal{M}} : M^{n_f} \to M$ for each $f \in \mathcal{F}$, a set of relations $R^{\mathcal{M}} \subset M^{n_R}$ for each $R \in \mathcal{R}$, and a set of elements $c^{\mathcal{M}} \in M$ for each $c \in \mathcal{C}$, respectively.
We call a Σ-*structure* and we write $\mathcal{M} = \big(M, \{f^{\mathcal{M}}\}_{f\in\mathcal{F}}, \{R^{\mathcal{M}}\}_{R\in\mathcal{R}}, \{c^{\mathcal{M}}\}_{c\in\mathcal{C}}\big)$. □

The set of constants $\mathcal{C}$ is a special set of functions, as constants are nullary functions (functions of arrity 0). Any of the sets $\mathcal{F}$, $\mathcal{R}$, $\mathcal{C}$ may be empty.

Definition 2.6.3. A mathematical structure is called an *algebraic structure* if $\mathcal{R} = \emptyset$. A mathematical structure is called a *relational structure* if $\mathcal{F} = \emptyset$. □

In Section 2.1 and 2.4 we gave examples of relational and algebraic structures, respectively.

2.7 Specifications

In Section 2.6 we have given the definition of a signature Σ and an Σ-structure. In order to define a language, we need to describe both its syntax and the rules that show how the language functions. In Section 2.7.1 we detail the syntax of a mathematical language, while in Section 2.7.2 we describe its specification.

2.7.1 Language Syntax

The following definitions and results are borrowed from [HH93, EM85].

Definition 2.7.1 ([HH93])**.** Let $\Sigma = \big(\mathcal{F},\ \mathcal{R},\ \mathcal{C},\ \{n_f\}_{f\in\mathcal{F}},\ \{n_R\}_{R\in\mathcal{R}}\big)$ be a signature and let X be a set of variables.

We call $T_X(\Sigma)$ the *set of X-terms of type* Σ defined inductively as follows:

- $X \cup \mathcal{C} \subseteq T_X(\Sigma)$ (basic terms)
- $\forall(t_1, t_2, \ldots, t_{n_f}, f \ \mid\ t_1, t_2, \ldots, t_n \in T_X(\Sigma) \wedge f \in \mathcal{F} \ :\ f(t_1, t_2, \ldots, t_{n_f})) \in T_X(\Sigma)$ (composite terms)
- There are no other terms in $T_X(\Sigma)$.

We call $\Phi(\Sigma)$ the *set of formulas of type* Σ and define it inductively as follows:

- $\forall(t_1, t_2, \ldots, t_{n_R}, R \ \mid\ t_1, t_2, \ldots, t_{n_R} \in T_X(\Sigma) \wedge R \in \mathcal{R} \ :\ R(t_1, t_2, \ldots, t_{n_R}) \in \Phi(\Sigma))$ (atomic formulas)
- Let $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ be a logical operator and $Q \in \{\forall, \exists\}$ a quantifier.

  $\forall(\phi, \psi, x \ \mid\ \phi, \psi \in \Phi(\Sigma) \wedge x \in X \ :\ \neg\phi \in \Phi(\Sigma) \wedge \phi \square \psi \in \Phi(\Sigma) \wedge\ Q(x \ \mid\ x \in X \ :\ \phi(x)) \in \Phi(\Sigma))$ $\square$

Given a signature Σ and a set of variables X, we call *free variables in a term $t \in T_X(\Sigma)$* the set $FV(t)$ defined inductively by:

- $\forall(x \ \mid\ x \in X \ :\ FV(x) = \{x\})$
- $\forall(c \ \mid\ c \in \mathcal{C} \ :\ FV(x) = \emptyset)$

- $\forall(t_1, \ldots, t_{n_f}, f \mid t_1, \ldots, t_{n_f} \in T_X(\Sigma) \wedge f \in \mathcal{F} : FV(f(t_1, \ldots, t_{n_f})) = \bigcup(i \mid 1 \leq 1 \leq n_f : FV(t_i)))$

A term with no free variables is called *closed term*.

In a formula, any variables that occur in a quantifier are called *bound variables*. In a given formula ϕ any occurrence of a variable not bound by a quantifier is a free variable in ϕ. A variable may have both free and bound occurrences in a formula e.g., $\phi \equiv ((\forall x p(x, y)) \rightarrow q(x))$. In this example x is bound in p, while it is free in q. Variable y is free in p. A formula with no free variables is called *closed formula*.

2.7.2 Algebraic Specifications

Definition 2.7.2 ([EM85])**.** Given a Σ signature and a set of variables X, we call an *equation* w.r.t. Σ the triple $e = (X, L, R)$, where $L, R \in T_X(\Sigma)$. □

If L, R are ground (closed) terms, we call e a ground equation. On a ground equations, the set of variables $X = \emptyset$.

A *specification* consists of a signature Σ and a set E of equations w.r.t. Σ [EM85]. An *algebraic specification* is an algebraic structure of type Σ which satisfies all equations in E.

Extending this definition, we will consider a *specification* as the tuple $SPEC = (\Sigma, E, Fm)$, where E is a set of equations and Φ a set of formulas w.r.t. signature Σ. A Σ-structure $\mathcal{M}$ of specification $SPEC$ must satisfy all equations in E and all formulas in Φ.

Chapter 3

Literature Survey

In this chapter, we survey the literature for similar mathematical approaches on the formalization of ontology structure. In Section 3.1, we look at the existing representations of knowledge, with a focus on ontologies. In Sections 3.2 and 3.3, we examine two approaches to a formal definition for ontologies: as relational structures and as algebraic structures. In Sections 3.2.1 and 3.2.2, we look at existing general, domain-independent relationships in ontologies. In Section 3.3.1, we examine an existing information system, called information algebra. In Section 3.3.2, we examine the ontologies as a model for category theory. In Section 3.5 we discuss a number of other ontology formalisms.

3.1 Knowledge representation

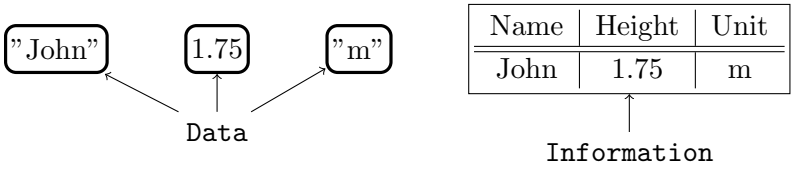
3.1.1 Introduction

Before we start analysing and formalizing any new theories, we need to clearly understand and define the key concepts in this thesis. What is data, information, and

knowledge? What are the differences between them? What is knowledge representation? Why do we need them? These are a few questions we need to answer in order to proceed with the presentations of our proposed mathematical theory and its applications.

In [Zin07] we find a comprehensive discussion on data vs information vs knowledge. *Data* is facts, observations, unorganized and unstructured. *Information* is data in context: structured, categorized, given meaning. *Knowledge* is information with understanding, information being evaluated, organized. For example, "John", "1.75", "m" are all pieces of data. In the context of a medical record system, we categorize "John" as a Name, "1.75" as Height, and "m" as Unit for height; these are now information. In the context of defining "Average Height" as between 1.65 m and 1.80 m, we can deduct that John is of Average Height; this is knowledge. In the context of computer science, we see data as symbols, with no meaning; information as structured, organized data, with explicit relationships between them; and knowledge as implicit relationships that can be inductively or deductively generated from existing information. If it is persisted in the system, knowledge becomes information. Figure 3.1.1 illustrates this example.

Representation is a relationships between two domains, usually one being more abstract and the other more concrete [BL04]. Knowledge representation is mainly a way of using symbols to represent a collection of statements or concepts. In [Dav93] the author describes five roles that knowledge representation can take; we have detailed



AverageHeight:= 1.60m $\leq$ Height $\leq$ 1.85m

Knowledge: John has AverageHeight

Figure 3.1: Data, Information, Knowledge

them in Section 1.2. The first role is being a surrogate for the concrete 'thing' it represents, in order to enable reasoning about the knowledge, rather than acting only on the information. Thus knowledge representation becomes a crucial step in reasoning on data.

Knowledge can be shared, and researchers need a representation that enables easy sharing and merging. For our thesis, knowledge is used to reason upon; the representation enables various methods and algorithms to be used to reason on existing data and information. In the following sections we discuss a few approaches to knowledge and information formal representation.

3.1.2 Knowledge representation through Ontology

The idea to represent knowledge in hierarchical structures, in order to enable automated reasoning on it, has been studied for over 35 years, with little results [GPL04]. The knowledge-base building domain has faced many challenges during its relatively short history. First, researchers have been arguing on which aspect of knowledge-base

engineering to be addressed first: representation or reasoning? Some application domains, such as the medical field, focused on the engineering of a unified system and succeeded (UMLS, the medical ontologies, etc.). Other fields have been less successful in developing a formal knowledge representation and expert system.

From reviewing the literature, it seems that each field, each domain of application, sometimes even each project, starts by engineering their own version of ontology, instead of reusing or extending existing ones. Thus, the research on knowledge representation through ontologies has shifted from engineering one formal system that can be universally applied or extended, to merging, sharing, and aligning existing ontologies. In other words, there is no common formal definition of what an ontology is.

In [DJBF$^+$08], the authors focus on the knowledge representation aspect, motivated by the fact that the field is dynamic and knowledge is an evolving, living subject, thus ontologies must be constantly be updated. In order to keep the updates as objective as possible, little to no human interaction is desired, and the focus switches onto building tools to automate the process of adding, deleting, and replacing knowledge. In order to do that, an ontology is defined as a formal mathematical structure, used to build neighbour sets, or concepts that are closely related. These sets are in turn used to build equivalence classes for terms and concepts, and automatically update the ontology from existing documents.

In the context of the semantic web, in order to automatically extract knowledge from

a document, both the ontology and the knowledge base are formally defined [Add10]. With the new formal structures, the authors define a data language, used to automate pattern extraction, and ultimately generate new knowledge.

3.2 Ontology as a Relational Structure

The philosophical definition of ontology is the study of that which is (from the Greek *onto-* i.e. being and *-logia* i.e. study) [Gru95, Oxf]. Ontologies have been described as catalogs of the "things" that exist in a certain domain of interest, and the relationships between the things [Spe06]. In the context of knowledge engineering, Gruber is one of the first to formally describe an ontology as "an explicit specification of a conceptualization" [Gru95]. With this definition, the things are called concepts and their realization is known as entities or individuals. Between the concepts of any domain there are relations, which themselves can be categorized using relation types.

In our research, we focus on the general, structural relationships between the concepts of an ontology. These are domain-independent relationships, which can be found in any type of ontology. The following is a short, non-exhaustive list of relationships found in a variety of existing ontologies.

3.2.1 Taxonomical relationships

Taxonomically, ontologies are pure classification systems, thus having two types of relationships: `isA` and `hasA`. The `isA` relationship itself comes in two flavours. The first is *supertype-subtype* and is used to classify classes of concept; e.g., *Dog* `isA` *Mammal*,

where *Dog* and *Mammal* are both classes of concepts. The second flavour is *type-instance* and is used to relate individuals to their type (or class, or concept); e.g., *Max* `isA` *Dog*, where *Max* is an entity, the instance of the concept *Dog*. In a similar way, the `hasA` has two mereological aspects. The first is the *component-integral object* type of relationship, also known as parthood; e.g., *Car* `hasA` *Wheel*, where both *Car* and *Wheel* are concepts. The second aspect is the *member-collection* type of relationship, that links a concept to a group of concepts; e.g., *Team* `hasA` *Player*.

In the domain of natural language, we find a variety of semantic relationships, such as: synonyms, antonyms, morphological, subsumption, and mereological relations. At the conceptual level, synonyms can be seen as the equivalence relationship, as described in [Gil02]. The subsumption relation can be seen as the supertype-subtype `isA` relationship [Hea92], [NKCN08], [Bra83]. Mereological relationships, or the relationships of parthood, are discussed below in more details. Antonyms are a strictly semantical relationship, and as such they pose no interest to our thesis. Similarly, morphological relationships, known as the transformation of words, are domain specific, not structural, and they are out of the scope of this work.

3.2.2 Mereological relationships

Mereology is the study of parthood relations, from the Greek $\mu\varepsilon\varrho o\varsigma$, "part" [Var15]. Mereology has been studied since antiquity and by many scholars. We present details found in [Var15], [GAM03], [WCH87]. In natural language, *parthood* relationship can have multiple meanings, such as: composition, aggregation, or containment. The *composition* and *aggregation* describe the relationship between a holonym (the whole)

and a meronym (the part). The difference between the two is in the life cycle or purpose of the meronym. In aggregation, the part exists individually outside the relationship; in composition, the part cannot exist without the whole. For example, *Person* `partOf` *Company* is an aggregation relationship; if the company ceases to exists, the people continue to have meaning or purpose. *Account* `partOf` *Company* is a composition relationship; when the company does not exist, its accounts have no more meaning. We observe that `hasA` and `partOf` describe the same relationship, in opposite directions: `hasA` is directed from the whole or collection to the part or member, while `partOf` is directed from the part to the whole. We say that `hasA` is the inverse of `partOf`, and vice versa.

Intuitively, the `isA` relationship, as subsumption (not instantiation) is transitive: $(A \texttt{ isA } B \land B \texttt{ isA } C) \implies A \texttt{ isA } C$. Similarly, `partOf` relationship is transitive: $(A \texttt{ partOf } B \land B \texttt{ partOf } C) \implies A \texttt{ partOf } C$.

In [Var15], we find the parthood relationship expressed mathematically as a partial order, written as $\leq$. Intuitively the parthood is reflexive, as everything is considered part of itself. The parthood relationship is transitive; any part of a part of a whole is itself a part of the whole. The parthood relationship is asymmetric as well, as two distinct things cannot be part of each other. Other mereological concepts can be expressed with parthood. For example, *equality* is written as $=$, and defined by $\forall(x, y \mid x, y \in P : x = y \overset{\text{def}}{\iff} x \leq y \land y \leq x)$. The *proper parthood* relationship is defined by $\forall(x, y \mid: x < y \overset{\text{def}}{\iff} x \leq y \land \neg(x = y))$.

There are many philosophical questions that arise regarding how mereological relationships change relative to time, space, etc. From the taxonomical and ontological point of view, the relativity of mereological relationships to real life parameters can be safely ignored. Our scope is to articulate the mathematical structure for an ontology, thus it suffices to describe the mereological axioms 'in void'. An exhaustive discussion of various philosophical issues can be found in [Var15].

3.2.3 Other domain-independent relationships

In studying a variety of domain specific ontologies, as well as core ontologies, we discovered a number of other domain-independent relationships [Gil02, LG05, Mar, Dub, FOA, GO], etc. We discuss below each of these relationships.

The `memberOf` relationships is just a binary relation; it is not reflexive, anti-symmetric or transitive. Formally, the `memberOf` is represented by set inclusion, A `memberOf` $B \overset{\text{def}}{\iff} A \in B$.

The `linkedTo` and `associateTo` relationships are both reflexive and transitive. Semantically, they are being represented as a non-directional link between two concepts, thus they are symmetric, A `linkedTo` $B \implies B$ `linkedTo` A.

The `dependentOn` relationship is similar to the `linkedTo` and `associateTo`, in that it is reflexive and transitive. A `dependentOn` $B \wedge B$ `dependentOn` $A \implies$ the relation is cyclic, thus it is not anti-symmetric.

The `synonym` relationship can be expressed as the `equality` relationship, which, in turn, can be expressed using the `partOf` relation, as we have seen in Section 3.2.2.

In [LG05] the authors discuss a variety of spatial and temporal relationships. A few of them are generic, domain independent, such as: `before` and `after` as temporal relators, `onTopOf` and `below` etc. as spatial relators. The temporal relators are partial orders, and so are the spatial relators (in a linear space). Other spatial relators, such as `locatedIn` and `coveredBy` can be expressed as `partOf`.

3.3 Ontology as an Algebraic Structure

Ontology experts have been faced with the task of representing ontologies in a format that can be fed into and understood by computers. Below we discuss some approaches on formalizing an ontology using algebraic structures.

3.3.1 Information Algebra

The connection between different representations of information has been studied previously by Kohlas and Stark [KS98]. They have proposed a new approach to modeling semantic information, called information algebra, and showed that relational databases are a model of this theory. The information algebra is comprised of a mathematical structure, called the organized data set, $\big(\Phi, D\big)$, where Φ is the set of information and D is a lattice, together with a set of ten axioms.

In [KS98] the authors define the theoretical mathematical structure describing information processing, called 'information algebra'. Information is considered as a set of elements, or pieces of information. There are two main operations on information:

composition (or *combination*) and *focusing* (or *reduction*). Composition of information refers to getting two or more pieces of information combined into a new piece; or seeing that an information may be composed from different elements. If Φ is the set of pieces of information, the combination operator is denoted as multiplication: $\cdot : \Phi \times \Phi \to \Phi$. Intuitively, the combination operator is associative and commutative (the order the information is combined doesn't matter). Thus, the structure $\big(\Phi, \cdot\big)$ is a commutative semigroup. Focusing of information describes the part of information that answers a specific question (the frame or domain). A partial order is defined on the set of frames, D with the meaning: if x and y are frames in D (representing two questions), $x \leq y$ means that y is more precise than x, or x is coarser than y. D is assumed to be a lattice, with $x \wedge y$ being the infimum, the finest frame that is coarser than both x and y (dual definition for supremum, $x \vee y$).

Each information $\phi \in \Phi$ concerns a certain domain $d(\phi) \in D$, where $d(\phi)$ is called the label or mark of ϕ. The combination operator satisfies the labeling axiom: $d(\phi\psi) = d(\phi) \vee d(\psi)$.

Φ contains *unit elements* e_x for each frame $x \in D$ such that $\forall(\phi \in \Phi \mid d(\phi) = x \to \phi e_x = \phi)$. The unit element e_x represents the empty information over frame x.

If $x \in D$ and $\phi \in \Phi$ such that $x \leq d(\phi)$, then $\phi^{\downarrow x}$ denotes the part of information in ϕ which concerns frame (question) x. This operation is called focusing, $^{\downarrow x} : \Phi \times D \to \Phi$, and it satisfies a number of axioms, such as: transitivity, combination, idempotency, and stability [KS98].

Information algebra is a generalization of relational algebra, thus a mathematical framework to support any information system. The axioms defined in the information algebra have a direct correspondence to properties of relational algebra. In particular,

the combination and focusing operators solve some of the computational problems of query processing.

3.3.2 Ontology as a graph in Category Theory

Category theory is a powerful mathematical branch that abstracts a mathematical structure into a collection of its objects and morphisms between the objects. It also provides rules for combining existing structures into new ones or decomposing a given structure into more elementary ones.

Many properties of mathematical systems can be graphically represented in a simple diagram, which depicts the objects of a collection, C, and how they relate in the context C (the arrows between the objects). In figure 3.2 we show the representation of a category with objects X, Y, Z, morphisms f, g, g ∘ f, and one of the identity morphisms, 1_X.

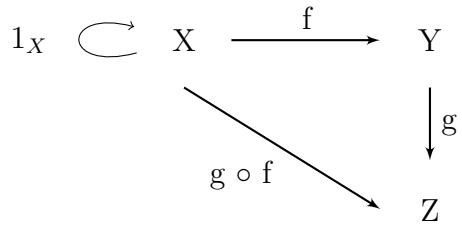


Figure 3.2: Category theory: graphic representation

While in the ontological domain the experts usually represent ontologies as a graph or tree, the need for a more formal approach rose, and the result was that category

theory became a likely formalism for ontological structure representation. There are many examples of such approaches in the literature and we discuss a few below.

In [JD01], we find such an example from e-commerce. The vertices of the diagram represent the entities (types, or the objects of the domain), the arrows represent various relationships between entities (the morphisms). By viewing ontologies as trees or directed graphs, the edges on these structures represent relations that can be composed. When considering real-life applications, the commuting diagrams usually represent business rules. Applying the mathematical system of category theory to an ontology enables the discovery of implicit pathways, and may represent new knowledge for the domain expert.

In [KHES05] the authors are faced with the task of distributing and integrating ontologies. As with any type of information, ontologies may be shared and combined. Consider that a relationship from ontology A to ontology B can be defined, representing a transformation or translation of concepts and relations in one ontology into the terminology of the other. If we also can define a relationship from ontology B to ontology C, the immediate question is: "how can we transform A into C?". It is a very natural step to recognize the situation lends itself to category theory, where the ontologies are now the objects and the relationships between them morphisms. By representing the ontological aspects through category theory, we can use additional results, such as ontology merging, that extends the product elements of category theory. Future research can focus on applying the abstract and very general framework

of category theory to the more concrete discipline of ontologies, to discover new algorithms and representations, as well as to provide a formal, unified framework for ontology engineering.

In [HC06], the category theory is proposed as a formal language for defining ontologies, by making the same observation: ontologies are collections of concepts at different levels of abstraction and specialization, with relationships between the concepts, where relationships are as important as the concepts themselves. Providing this mathematical theory as a model for ontologies and knowledge representation allows analysts and domain experts to have a common foundation for understanding and sharing their systems, complete with functionality, components, and calculations. The mathematical system enables one to prove or disprove that the system exhibits certain properties and behaviours. Properties of the system can be engineered with precision, they synthesize the system in a proper, logic manner.

3.4 Other Ontology Formalisms

Other ontology formalisms are: Frame Logic, Description Logic, and Formal Concept Analysis. Frame Logic treats classes, entities, and their relationships as objects [dBH04]. For example, in Frame Logic we express "John is a Person" as $John : Person$ and "John is Mary's Child" as $Mary[has-Child \rightarrow John]$. John, Mary, has-Child, and Person are all terms.

Description Logic is a Frame Logic extension in which classes are treated as unary predicates, and their relationships are binary predicates. The entities are terms. In

Description Logic, the two statements above are expressed as $Person(John)$ and $hasChild(Mary, John)$, where only the entities John and Mary are terms, and Person and hasChild are predicates. Description Logic is separated into two parts: the T-box and A-Box. The T-box (or the terminological box) is the collection of definitions (or terminology) for the application domain, the predicates that relate only classes (or concepts). The A-box (or the assertional box) is a collection of facts (or assertions) about a specific interpretation of the domain, it contains the predicates that relate entities to classes.

Formal Concept Analysis (FAC) is a mathematical theory of data analysis using formal contexts and concept lattices. Data is presented in a very basic format, as a lattice of concepts and their attributes, called formal context. Formal concepts are built from concepts sharing the same attributes.

3.5 Discussion

Most of the existing algebraic formalisms define an ontology as a strictly hierarchical structure. Information algebra and Formal Concept Analysis consider strictly relational datasets, where concepts are related by the `isA` or `hasA` relationships, and other relations are ignored. A lattice of concept is formed, based on taxonomical relations. As such, the meaning of the relationships is not formally described or understood. Description Logic brings the other relations into the picture, however they do so in a static, monolitic way, where the definition of the ontology contains both T-box and A-box equations. The organised data sets are usually formalised through relational algebras, not taking in account the need to capture the open world aspect of ontologies.

As we have seen in the previous sections, there exists many approaches to formally describing an ontology. However, most of the efforts deal with a single aspect of ontologies, that of knowledge representation and management, by focusing on the creation, sharing, and reuse of knowledge. Also, based on our review, we have not found any formal approach to describing the link between an ontology and its data structure. The literature contains examples of building an ontology from a data structure, or creating a data structure from an ontology, however no research on formally mapping a data structure to an existing ontology.

We look at capturing the ontology structure by using the `partOf` relation instead of the taxonomical ones. We also capture the other relationships in an ontology as a family of graphs with certain properties, which will be discussed in the next chapter. By building the lattice of concepts through the `partOf` relation we give the relation a clear understanding, and we enable the link between the organised data sets and the ontology structure (the concepts of the lattice are indices in the data structure). Finally, by using Tarski's cylindric algebra to capture the organised data set, we can now describe the open world of ontologies.

In our research we aim to formalize both the ontology and a given organised set of data in such a way that a mapping between the two is possible. The mathematical system obtained can be used with existing reasoning systems, or new reasoning systems can be developed. Thus, the recently developed ontology-based data-access technology [CGL$^+$12] becomes a model for our theory.

Most of the formalisms presented take a static approach, data is usually incorporated in the ontology definition, and if something changes in either the structure or the data, the entire definition of an ontology must be modified. We aim for a dynamic approach, that allows a modular approach, in the sense that when the data changes, only the domain information structure needs to be modified. In our research, we further separate the DL into specific mathematic structures, making use of the separation of concerns paradigm. In doing so, we open the door to allow multiple interpretations on the same ontology definition, as well as using multiple data sources on one definition.

Chapter 4

Domain Information Structure

In this chapter, we introduce the notion of a (domain) information structure based on ontologies and we give its mathematical structure. We start by observing that the information system can be separated into three distinct components. The first component is the structure of information that it deals with, which is given in Section 4.1 as an abstract ontology represented by a set of concepts and the relationships between them. The second component, which is described in Section 4.2, is the data stored in the information system, given by a set of values for the concepts, also called realizations of concepts or entities. In Section 4.2, we propose a formal mathematical theory that describes the domain information structure, by linking the two structures together. In Section 4.3, we describe the third components, which is the algebraic specification that gives more links between the two previous elements of the information system.

4.1 Abstract Ontology

As discussed in Section 1.2, ontologies are widely used today to classify and analyze information. They capture the set of concepts in a domain, their attributes, and the relationships between the concepts. The process of concept classification is known as *concept analysis*. In our work we focus only on the general relationships between concepts, such as the relations `isA`, `memberOf`, `hasA`. We call such an ontology the *abstract ontology*, in the sense that it describes the most general, abstract view of an application domain.

On an abstract ontology, let C be a set of its concepts. On C we define a binary operator $\times$, which represents the combination of two concepts. Semantically, the order of the concepts in a combination does not matter, a concept $c = c_1 \times c_2$ is isomorphic to a concept $c' = c_2 \times c_1$. Thus, we require that $\times$ is commutative (strictly speaking up to an isomorphism). In addition, we require that the operator $\times$ is idempotent.

Let $\langle C, \times, e_{_C} \rangle$ be a commutative idempotent monoid of concepts. Intuitively, the $e_{_C}$ concept is a kind of pseudo-concept that is neutral to the composition of concepts.

Definition 4.1.1. For every $c_1, c_2 \in C$, we say that c_1 is a *partOf* c_2, that we denote by

$$c_1 \sqsubseteq_{_C} c_2 \iff^{\text{def}} \exists(c \mid c \in C : c_1 \times c = c_2) \tag{4.1}$$

□

As shown in Section 3.2.2, the `partOf` relation is a partial order, satisfying the three axioms of posets. Considering the mereological relationship just a partial order is a simplification, as the relation is much more than that (a full discussion on other principles of mereological relationships is found in [Var15]). The poset properties are its core properties and they allow us to build the structure of the ontology, as well as to link it to organised data sets, as described below.

Starting from a main domain concept, we consider a subset $L \subseteq C$ of concepts, on which the `partOf` forms a Boolean lattice of concepts, $\mathcal{L} = \left(L, \sqsubseteq_C\right)$. The subset L contains the main concept, along with all its parts and subparts until its atoms. Therefore, e_C is included in the subset L; the e_C concept is the bottom concept: even if two concepts are structurally unrelated, they do have one concept in common, the pseudo-concept e_C. The lattice represents the structure of the main domain concept, which we call the *top of the lattice*, and we write it $\top_{\mathcal{L}}$. If we consider the set of all atoms in the lattice, $\mathcal{A}(L)$, we see that the lattice is isomorphic to the lattice of $\mathcal{P}(\mathcal{A}(L))$, which is a Boolean lattice. Hence $\mathcal{L}$ is a Boolean lattice, as shown in Section 2.3.

We start with a finite set of concepts, as the real life application domains contain a finite set of concepts. The combination operator $\times$ is commutative (up to an isomorphism) and idempotent, and as such the lattice of concepts is finite. We show an example of such a lattice in Figure 4.1. Note that the lattice can be large and not all concepts have a meaning in the application domain, however they must be present in the lattice, to ensure it is a Boolean lattice.

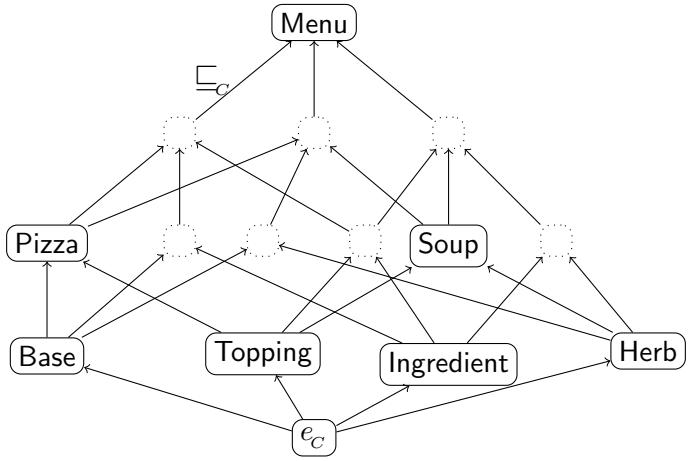


Figure 4.1: $\mathcal{L} = \big(L, \sqsubseteq_C\big), L \subseteq C$

The concepts in the lattice L may be related to other concepts in C through one or more relationships. We call the concept that belongs to the lattice the *top element* of the relationship. We consider only subsets of concepts that can be connected to the lattice: from any concept in the relational structure we can reach the top element, by relation composition. The relationships can be represented as rooted graphs, where the root is always part of the lattice.

Given a (possibly empty) family $\mathcal{G} = \{G_i\}_{i \in L}$ of rooted graphs on subsets of C, with each G_i we associate a relation R_i. The relations R_i do not need to be orders, but they need to be connected to the top element. Given an element $a \in C$, we can use its relational representation, R_a defined as $\{(x, x) \mid x = a\}$.

Definition 4.1.2 ([SS91])**.** Let $C_i \subseteq C$, and R_i a relation on C_i. We call a *rooted graph* the graph $G_i = \big(C_i, R_i\big)$ iff $\exists!(t_i \mid t_i \in C_i : \mathbb{L}; R_{t_i} \subset {R_i}^*)$. We denote a

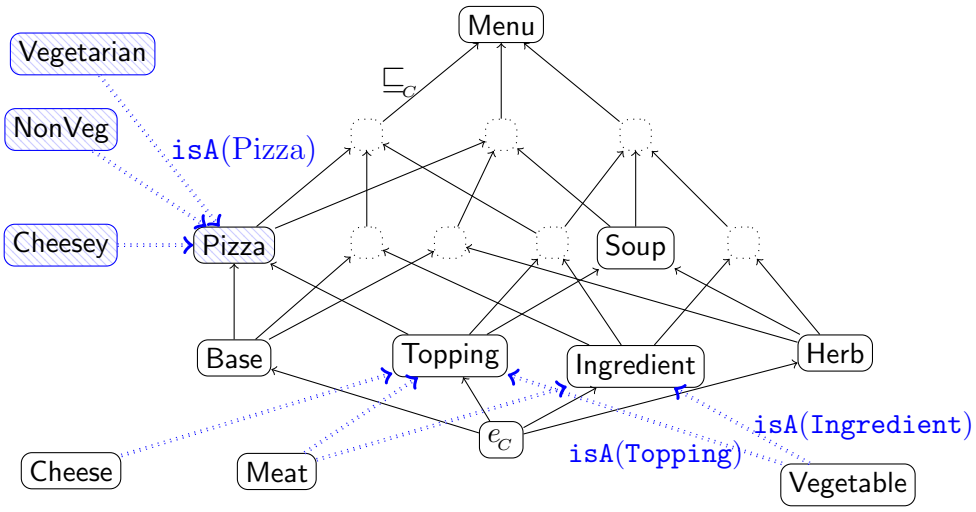


Figure 4.2: Family of rooted graphs

rooted graph as $G_i = \big(C_i, R_i, t_i\big)$, and we call t_i the *root* of G_i or its *top element*. □

The above can be written as follows

$$t_i \in C_i \text{ is } \textit{root} \text{ iff } \forall(c \mid c \in C_i : c = t_i \vee (c, t_i) \in {R_i}^+) \tag{4.2}$$

An example of a rooted graph is illustrated in Figure 4.2, where *Pizza* is the root of the graph.

The carrier sets of two rooted graphs $G_i, G_j \in \mathcal{G}$, for $i \neq j$, do not have to be disjoint (i.e., a concept can be part of two relations). For example, the concept *Vegetable* is part of the *Topping* rooted graph $G_{Top} = \big(C_{Top}, R_{Top}, Topping\big)$, with a possible structure shown in Figure 4.2, where R_{Top} is a hierarchical `isA` relation. The same concept *Vegetable* is part of a *Ingredient* rooted graph $G_{Ing} = \big(C_{Ing}, R_{Ing}, Ingredient\big)$, where R_{Ing} is another `isA` relation.

We can now formally define the abstract ontology as a mathematical structure:

Definition 4.1.3. Let $\big(C, \times, e_{_C}\big)$ be a commutative idempotent monoid. Let $\mathcal{L} = \big(L, \sqsubseteq_{_C}\big)$ be a Boolean lattice, with $L \subseteq C$, such that $e_{_C} \in L$. Let $\mathcal{G} = \{G_{t_i}\}_{t_i \in L}$ be a family of rooted graphs at t_i.
We call an *abstract ontology* the mathematical structure $\mathcal{O} = \big(C, \mathcal{L}, \mathcal{G}\big)$. □

The definition indicates that each relation has a top element, which is the root of the corresponding rooted graph, and that the root on any rooted graph in $\mathcal{G}$ ends up in the lattice. It also ensures that all the concepts in each relation are connected to the main lattice structure. If $G = \big(\{t_i\}, \mathbb{I}, t_i\big)$, then t_i must be part of the lattice.

The ontology structure also ensures that there are no disconnected concepts. A concept is either part of the lattice or part of one of the rooted graphs that are rooted in the lattice. With this understanding, we write
$C = L \cup \bigcup (G_i \mid G_i \in \mathcal{G} \wedge G_i = \big(C_i, R_i, t_i\big) : C_i)$.

Proposition 4.1.1. Let $\mathcal{O} = \big(C, \mathcal{L}, \mathcal{G}\big)$ be an abstract ontology.
$\forall (c_1, c_2 \mid c_1, c_2 \in C : c_1 \sqsubseteq_{_C} c_2 \iff c_1 \times c_2 = c_2)$

Proof.

$c_1 \sqsubseteq_{_C} c_2$

$\iff$ ⟨ Definition 4.1.1 ⟩

$$
\begin{array}{ll}
 & \exists(c \mid c \in C : c \times c_1 = c_2) \\
\Longrightarrow & \langle\ \forall(a,b,c \mid a,b,c \in C : a = b \implies a \times c = b \times c)\ \rangle \\
 & \exists(c \mid c \in C : c_1 \times c \times c_1 = c_1 \times c_2) \\
\Longleftrightarrow & \langle\ \times \text{ is commutative}\ \rangle \\
 & \exists(c \mid c \in C : c \times c_1 \times c_1 = c_1 \times c_2) \\
\Longleftrightarrow & \langle\ \times \text{ is idempotent, } c_1 \times c_1 = c_1\ \rangle \\
 & \exists(c \mid c \in C : c \times c_1 = c_1 \times c_2) \\
\Longleftrightarrow & \langle\ c \times c_1 = c_2\ \rangle \\
 & c_2 = c_1 \times c_2
\end{array}
$$

□

The $\Longleftarrow$ part is immediate:

Proof.

$$
\begin{array}{ll}
 & c_1 \times c_2 = c_2 \\
\Longrightarrow & \langle\ \text{Definition 4.1.1, } \exists c = c_2\ \rangle \\
 & c_1 \sqsubseteq_C c_2
\end{array}
$$

□

4.2 Domain Information Structure

In Section 2.5, we have discussed the link between relational algebras and cylindric algebras, and the reason we use cylindric algebras to define the information system, instead of the widely used relational algebras.

Using Codd's relational data model, we consider a structured dataset with a collection of indices. An entity $a \in A$ is written as a tuple of values for the given set of indices, or an element of a Cartesian product. The `partOf` relation is defined in this context, and viewed as a relationship between projections on a subset of index values in the Cartesian product.

The structure of the data being described by the abstract ontology, the set of indices is represented by the carrier set of the lattice of concepts: the top element and all its parts and their subparts, down to atoms, including the pseudo-concept e_C. We express the data set structure through the cylindric algebra: $\mathcal{A} = \big(A, +, \cdot, -, 0, 1, c_k\big)_{k \in L}$, where c_k is the cylindrification operator on all the subparts of concept $k \in L$, defined as follows.

$$\forall(k, k_i \mid k, k_i \in L \wedge k = \times(i \mid 1 \leq i \leq n : k_i) : c_k x \stackrel{\text{def}}{=} c_{k_1} c_{k_2} \cdots c_{k_n} x) \quad (4.3)$$

What we ultimately need to do is to link the data to the ontology structure. We do that by defining three functions, as follows.

Definition 4.2.1. Let $\mathcal{O}$ be an abstract ontology as defined in Definition 4.1.3 and

let $\mathcal{A} = \big(A, +, \cdot, -, 0, 1, c_k\big)_{k \in L}$ be a cylindric algebra. We call *associated concept* the operator $\tau' : A \to C$ which associates an entity to the concept it instantiates. For a given $G_i = \big(C_i, R_i, t_i\big) \in \mathcal{G}$, and $c \in C_i$ we call the *associated type* the operator $\rho_i : C_i \to L$ defined as $\rho_i(c) = R_i^+(c) = t_i$. We can define $\rho = \bigcup(G_i \mid G_i \in \mathcal{G} : \rho_i)$. We call *type* the operator $\tau : A \to L$ defined as $\tau = \rho \circ \tau'$. □

Therefore, an entity can have multiple types depending on the relation considered for its associated concept.

Example 4.2.1.

Consider the lattice of concepts for a restaurant menu, described in Figure 4.3. The concepts *Menu*, *Pizza*, *Soup*, *Base*, *Topping*, *Ingredient*, *Herb*, and e_C form the lattice of concepts. Concepts *Cheese*, *Meat*, *Vegetables* are part of rooted graphs, one with root *Topping*, and one with root *Ingredient*. *Mozzarella*, *Parmesan*, *Ham*, etc. are entities.

Consider the entity *ThinAndCrips*, a realization of concept *Base*. The associated concept for it is given by $\tau'(ThinAndCrips) = Base$. With *Base* being in the lattice, the type of *ThinAndCrisp* is given by $\tau(ThinAndCrisp) = Base$. In this case, the associated concept of an entity and its type correspond to the same concept.

Consider the entity *Tomato*, a realization of concept *Vegetable*. Its associated type is given by $\tau'(Tomato) = Vegetable$. However, its type depends on the relationship considered. If we consider *Tomato* as a soup ingredient, then its type is $\tau(Tomato) = \rho(\tau'(Tomato))$, where ρ represents the relationship in the rooted graph at *Ingredient*. Thus, $\tau(Tomato) = Ingredient$. If we consider *Tomato* as a pizza topping, its type

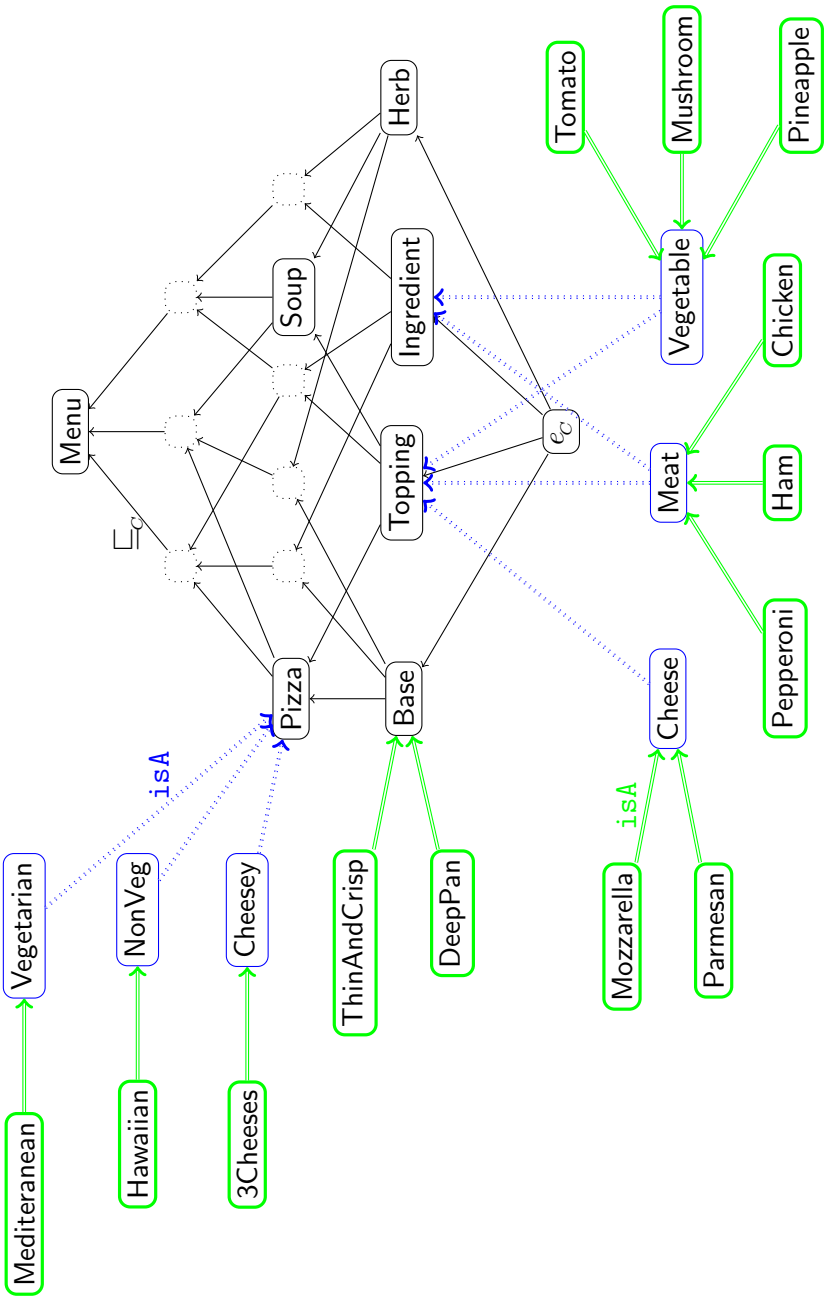


Figure 4.3: Menu Restaurant ontology

is given by $\tau(Tomato) = Topping$.

We can now link the two structures into one mathematical system.

Definition 4.2.2. We call *domain information structure* a mathematical structure $\mathcal{I} = \big(\mathcal{O}, \mathcal{A}, \tau\big)$, where $\mathcal{O} = \big(C, \mathcal{L}, \mathcal{G}\big)$ is an abstract ontology as presented in Definition 4.1.3, $\mathcal{A} = \big(A, +, \cdot, -, 0, 1, c_k\big)_{k \in L}$ is a cylindric algebra, and $\tau = \tau' \circ \rho$ is the type operator, such that the following postulates are satisfied for all $a, b \in L$ and every $c \in C$:

(1) $c \in C_i \wedge \rho(c) = t_i \vee c \in L \wedge \rho(c) = c$

(2) $\tau(a \cdot b) = \tau(a) \times \tau(b)$

(3) $\tau(0) = \top_{\mathcal{L}}$

(4) $\tau(1) = e_{_C}$ □

In the context of C being the set of concepts and A the set of entities, intuitively, Axiom (1) expresses the fact that the type of an entity, or the concept it represents, is part of the lattice $\mathcal{L}$. In other words, a concept is expressed as a tuple of its subparts. Axiom (1) describes the structure as a loosely typed structure, by ensuring that when the associated concept of an entity is in a rooted graph, its type is represented by the root of the graph. A loosely typed structure presents the advantage of generalization, by treating all concepts in a given connected relation as the top element, from the point of view of its type or structure. If complete structural details are required, then strongly typed structure should be considered instead. In our structure all concepts

are either part of the lattice, or part of a rooted graph, hence connected to the lattice. As such, all entities will have an associated concept that is either part of the lattice or, when the associated concept is part of the rooted graph, they can be represented by the root through the given relation. Axiom (2) says that when we combine two entities, the type of the resulting entity is the same as the combination of the types each entity. Axioms (3) and (4) ensure the validity of Axiom (2).

For every $a, b \in A$:

$$\tau(a \cdot b) = \tau(a) \times \tau(b)$$

$$\Longrightarrow \qquad \langle \text{ Substitute b with } 0 \rangle$$

$$\tau(a \cdot 0) = \tau(a) \times \tau(0)$$

$$\Longleftrightarrow \qquad \langle \text{ Annihilator for } \cdot \rangle$$

$$\tau(0) = \tau(a) \times \tau(0)$$

This must be satisfied for every $a \in A$, thus it holds only for $\tau(0) = \top_{\mathcal{L}}$.

Similarly,

$$\tau(a \cdot b) = \tau(a) \times \tau(b)$$

$$\Longrightarrow \qquad \langle \text{ Substitute b with } 1 \rangle$$

$$\tau(a \cdot 1) = \tau(a) \times \tau(1)$$

$$\Longleftrightarrow \qquad \langle \text{ Identity element over } \cdot \rangle$$

$$\tau(a) = \tau(a) \times \tau(1)$$

This must be satisfied for every $a \in A$, thus it holds only for $\tau(1) = e_{\mathcal{C}}$.

The indices on an entity are concepts in the Boolean lattice. When we cylindrify an entity $a \in A$ on a concept $k \in L$, we might extend the type of a with the indices of k. We have

$$\forall(a, k \mid a \in A \wedge k \in L : \tau(c_k a) = k \times \tau(a)) \tag{4.4}$$

For every $t_i, t_j \in L$, we denote by $c_{(t_j \setminus t_i)}$ the operator c_k such that:

$$c_{(t_j \setminus t_i)} \stackrel{\text{def}}{=} c_k, \text{ where } k = \times(c \mid c \in \mathcal{A}(L) \wedge c \sqsubseteq_C t_j \wedge \neg(c \sqsubseteq_C t_i) : c) \tag{4.5}$$

Example 4.2.2.

Consider the lattice of *Order* concept described in Figure 4.4, where *Order* has five subparts, *OID*, *Customer*, *Date* and *Address*. Each of these concepts, except *OID* has more subparts; the bottom of the lattice is the pseudoconcept e_C. (Note: some of the concepts in the Boolean lattice are not shown, to ensure readability.)

An entity $a \in A$ is a tuple of values for its atomic subparts. For example, $a = (156, 'Michael', 'Smith')$ is a possible instantiation of the *Customer* concept, and we say a is of type *Customer*.

We write $\tau(c) = Customer$ and we can interpret it as $\{CID, FirstName, LastName\}$.

On $\mathcal{A}$ we define a new relator, (instance) `partOf`, and we write it as $\sqsubseteq$:

$$a \sqsubseteq b \iff b \leq c_{(\tau(b) \setminus \tau(a))} a \tag{4.6}$$

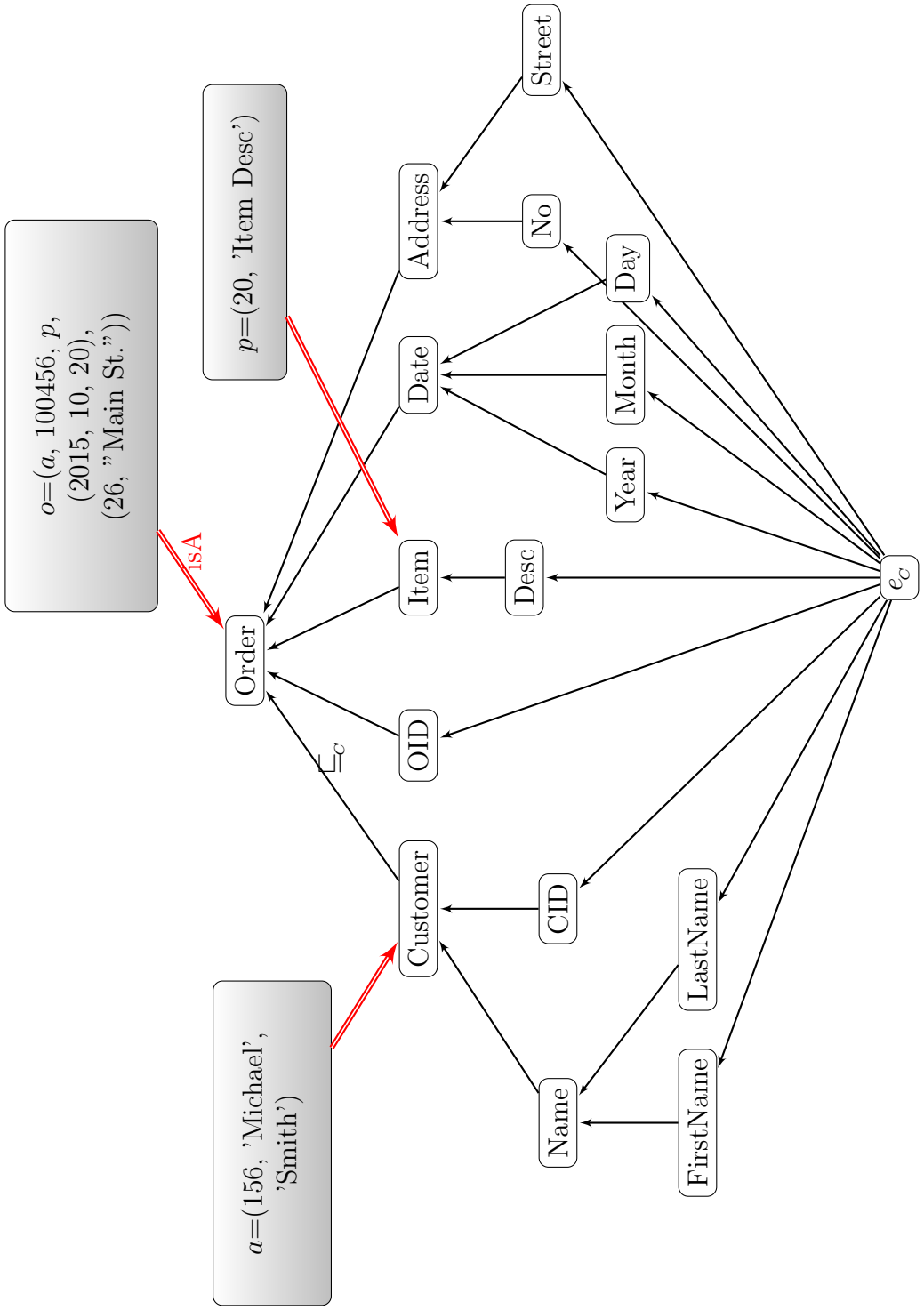


Figure 4.4: Information system: A partial view of the lattice of concepts

This definition expresses the fact that the entity b must be contained in the cylindrification of its subpart a, with the cylindrification executed on the indices that are found in b, and not found in the subpart a.

In Example 4.2.2, we have $o = (a, 100456, p, 2015, 10, 20, 26, "MainSt.")$ and $a = (156, 'Michael', 'Smith')$. Moreover, $Customer \sqsubseteq_C Order$, with $\tau(a) = Customer$ and $\tau(o) = Order$. The cylindrification of entity a on the concept that represents parts that are in *Order*, but not in *Customer* is the set of all entities of type *Order* that contain the subpart a, in other words the set of all the orders for customer a.

We write it as $c_{(Order \setminus Customer)}a = \{o \mid o \in A \wedge c \sqsubseteq o\}$. We can now use all the axioms of the information structure to reason on the ontology and organised data sets, to generate new knowledge.

Proposition 4.2.1.

1. $\forall(a, b \mid a, b \in A : \tau(c_{(\tau(b) \setminus \tau(a))}b) = \tau(b))$
2. $\forall(a, b \mid a, b \in A : \tau(c_{(\tau(b) \setminus \tau(a))}a) = \tau(a) \times \tau(b))$

Proof. For any $a, b \in A$, let $c_a = \times(c \mid c \in \mathcal{A}(L) \wedge c \sqsubseteq_C \tau(a) \wedge \neg(c \sqsubseteq_C \tau(b)) : c)$, $c_b = \times(c \mid c \in \mathcal{A}(L) \wedge c \sqsubseteq_C \tau(b) \wedge \neg(c \sqsubseteq_C \tau(a)) : c)$, and $c_{ab} = \times(c \mid c \in \mathcal{A}(L) \wedge c \sqsubseteq_C \tau(b) \wedge c \sqsubseteq_C \tau(a) : c)$. With these definitions, $\tau(a) = c_a \times c_{ab}$ and $\tau(b) = c_b \times c_{ab}$.

1. $\tau(c_{(\tau(b) \setminus \tau(a))}b)$

   $=$ $\langle$ Definition 4.4, 4.5 & $t_j = \tau(b)$ & $t_i = \tau(a)$ $\rangle$

$$\times(c \mid c \in \mathcal{A}(L) \wedge c \sqsubseteq_C \tau(b) \wedge \neg(c \sqsubseteq_C \tau(a)) \ : \ c\,) \times \tau(b)$$

$=$ $\langle$ Definition of c_b above $\rangle$

$$c_b \times \tau(b)$$

$=$ $\langle$ Proposition 4.1.1, Definition 4.6 & $\tau(b) = c_b \times c_{ab} \implies c_b \sqsubseteq_C \tau(b)$ $\rangle$

$$\tau(b)$$

2. $\tau(c_{(\tau(b)\setminus\tau(a))}a)$

$=$ $\langle$ Definition 4.4, 4.5 & $t_j = \tau(b)$ & $t_i = \tau(a)$ $\rangle$

$$\times(c \mid c \in \mathcal{A}(L) \wedge c \sqsubseteq_C \tau(b) \wedge \neg(c \sqsubseteq_C \tau(a)) \ : \ c\,) \times \tau(a)$$

$=$ $\langle$ Definition of c_b above $\rangle$

$$c_b \times \tau(a)$$

$=$ $\langle$ $\tau(a) = c_a \times c_{ab}$, Definition above $\rangle$

$$c_b \times c_a \times c_{ab}$$

$=$ $\langle$ $\times$ is idempotent, $c_{ab} = c_{ab} \times c_{ab}$ $\rangle$

$$c_b \times c_a \times c_{ab} \times c_{ab}$$

$=$ $\langle$ $\times$ is commutative $\rangle$

$$(c_a \times c_{ab}) \times (c_b \times c_{ab})$$

$=$ $\langle$ $\tau(b) = c_b \times_{ab}$ & $\tau(a) = c_a \times c_{ab}$, Definitions above $\rangle$

$$\tau(a) \times \tau(b)$$

□

Proposition 4.2.2.

1. In a domain information structure, all the entities of a rooted graph $G_i \in \mathcal{G}$ represent the same type

2. $\forall(a, b \mid a, b \in A : a \sqsubseteq b \implies \tau(a) \sqsubseteq_C \tau(b))$

Proof.

1. $\tau'(a) = c_a \wedge \tau'(b) = c_b \wedge c_a, c_b \in C_i$

   $\implies$ $\langle$ Definition 4.2.1 & $\tau = \rho \circ \tau'$ & $x = y \implies f(x) = f(y)$ $\rangle$

   $\tau(a) = \rho(\tau'(a)) = \rho(c_a) \wedge \tau(b) = \rho(\tau'(b)) = \rho(b)$

   $\implies$ $\langle$ Definiton 4.2.2, Axiom 1 $\rangle$

   $\tau(a) = t_i \wedge \tau(b) = t_i$

   $\implies$ $\langle$ Transitivity of $=$ $\rangle$

   $\tau(a) = \tau(b)$

2. $a \sqsubseteq b$

   $\iff$ $\langle$ Definition 4.6, notation $c_k = c_{(\tau(b) \setminus \tau(a))}$ $\rangle$

   $b \leq c_k a$

   $\iff$ $\langle$ A is a Boolean algebra & Definition 2.2 $\rangle$

$b \cdot c_k a = b$

$\Longrightarrow$ $\langle\ x = y \implies c_k x = c_k y\ \rangle$

$c_k(b \cdot c_k a) = c_k b$

$\Longleftrightarrow$ $\langle$ Definition 2.5.1 (4) $\rangle$

$c_k b \cdot c_k a = c_k b$

$\Longrightarrow$ $\langle\ x = y \implies \tau(x) = \tau(y)\ \rangle$

$\tau(c_k b \cdot c_k a) = \tau(c_k b)$

$\Longleftrightarrow$ $\langle$ Definition 4.2.2 (2) $\rangle$

$\tau(c_k b) \times \tau(c_k a) = \tau(c_k b)$

$\Longleftrightarrow$ $\langle$ Proposition 4.2.1, with notation $c_k = c_{(\tau(c_j)\backslash\tau(c_i))}$:

$\tau(c_k b) = \tau(b)\ \&\ \tau(c_k a) = \tau(a) \times \tau(b)\ \rangle$

$\tau(b) \times \tau(a) \times \tau(b) = \tau(b)$

$\Longleftrightarrow$ $\langle\ \times$ is comutative $\rangle$

$\tau(a) \times \tau(b) \times \tau(b) = \tau(b)$

$\Longleftrightarrow$ $\langle\ \times$ is idempotent $\rangle$

$\tau(a) \times \tau(b) = \tau(b)$

$\Longleftrightarrow$ $\langle$ Definition 4.1.1, $\exists c = \tau(b)\ \rangle$

$\tau(a) \sqsubseteq_C \tau(b)$

□

4.2.1 Extending relationships

On the abstract ontology, the relationships R_i can be extended using the lattice structure, thus extending the ontology. From a relation R_i we can infer other relationships between elements of C_i and elements of L.

Definition 4.2.3. Let $\mathcal{O} = \big(C, \mathcal{L}, \mathcal{G}\big)$ be an abstract ontology and let $G_i = \big(C_i, R_i, t_i\big) \in \mathcal{G}$ be a rooted graph, with $t_i \in C_i$ its root.

We call the *lattice extension* of R_i, the relation ${R_i}^{\uparrow} = R_i \cup R'_i$, where

$R'_i = \{(c,t) \mid c \in C_i \wedge t \in L \wedge (c,t_i) \in R_i \wedge t_i \sqsubseteq_C t\}$. □

Proposition 4.2.3. Let $G_i = \big(C_i, R_i, t_i\big) \in \mathcal{G}$ be a rooted graph, where R_i is transitive. The lattice extension of R_i is given by $R'_i = \{(c,t) \mid c \in C_i \wedge t \in L \wedge t_i \sqsubseteq_C t\}$.

Proof.

$$
\begin{aligned}
& R'_i \\
= & \quad \langle \text{ Definition 4.2.3 } \rangle \\
& \{(c,t) \mid c \in C_i \wedge t \in L \wedge (c,t_i) \in R_i \wedge t_i \sqsubseteq_C t\} \\
= & \quad \langle \ R_i \text{ is transitive } \iff R_i = R_i^{+} \ \rangle \\
& \{(c,t) \mid c \in C_i \wedge t \in L \wedge (c,t_i) \in {R_i}^{+} \wedge t_i \sqsubseteq_C t\} \\
= & \quad \langle \ t_i \text{ is root and Definition 4.2 } \rangle \\
& \{(c,t) \mid c \in C_i \wedge t \in L \wedge \mathit{true} \wedge t_i \sqsubseteq_C t\} \\
= & \quad \langle \text{ Identity of } \wedge \ \rangle \\
& \{(c,t) \mid c \in C_i \wedge t \in L \wedge t_i \sqsubseteq_C t\}
\end{aligned}
$$

□

Up until now, we have only considered one type of relations, the ones defined between concepts. In order to complete the ontology specification, we need to also consider relations between individuals (realizations of concepts) and a special type of heterogeneous relations between an individual and a concept that is not its type.

We can extend relationships on concepts to heterogeneous relationships on entities and concepts. Considering the example in Figure 4.5 *Giraffe* `eats` *Leaf* is a relation between two concepts: *Giraffe* and *Leaf*. *Berta* and *Gabby* are two individuals of associated concept *Giraffe*, thus they carry all the properties of their class. We always have *Berta* `eats` *Leaf* and *Gabby* `eats` *Leaf*.

Mathematically we express this by the following:
$\forall(a, c, c', G_i \mid a \in A \wedge G_i \in \mathcal{G} \wedge c, c' \in C_i :$
$\tau'(a) = c \wedge (c, c') \in R_i \implies \exists(R_{(A,C_i)} \mid R_{(A,C_i)} \subseteq A \times C_i : (a, c') \in R_{(A,C_i)}))$,
where $R_{(A,C_i)}$ is a heterogeneous relation corresponding to R_i.

In a similar manner we can extend the relationships on concepts to their corresponding relationships on entities. Knowing that *Giraffe* `eats` *Leaf*, we can infer there exists a realization of concept *Leaf*, the entity *MyLeaf* s.t. *Gabby* `eats` *MyLeaf*.

We say that for every rooted graph $G_i \in \mathcal{G}$, its relationship R_i has a correspondent relationship R_{i_A} s.t.

$$\forall(c, c', G_i \mid G_i \in \mathcal{G} \wedge c, c' \in C_i : (c, c') \in R_i \implies$$
$$\exists(a, a', R_{i_A} \mid a, a' \in A \wedge R_{i_A} \subseteq A \times A : \tau'(a) = c \wedge \tau'(a') = c' \wedge (a, a') \in R_{i_A})).$$

4.3 Domain Non-Structural Specification

So far we have only discussed the axioms pertinent to the structural parts of the information system: the ontology structure $\mathcal{O}$ and the data structure $\mathcal{A}$. However, there are other domain-specific knowledge that ought to be articulated. In this Section we discuss an example of this knowledge and we argue for the need of additional domain axioms. We will call this part of the specification of an ontology *domain non-structural specification*.

In [Bec03], we find a reasoning example using an ontology, pictured in Figure 4.5. In this example, concepts *Leaf*, *Plant*, e_C are part of the lattice, with Leaf $\sqsubseteq_C$ Plant and $e_C \sqsubseteq_C$ Leaf. In the literature, both statements *Giraffe* `eats` *Leaf* and *Vegetarian* `eats` *Plant* are treated as relationships.

We see that the first statement is a true relationship, while the second statement is the definition of the abstract concept *Vegetarian*. Such a definition cannot be captured only within the information structure framework, thus we need to provide the ontology specification to complement its structure.

As discussed in Section 2.7, given an information structure $\mathcal{I}$, we define its specification as the tuple $S_O = \big(\mathcal{I}, E, \Phi\big)$, where E is a set of valid equations w.r.t. $\mathcal{I}$ and Φ a set of axioms in $\mathcal{I}$.

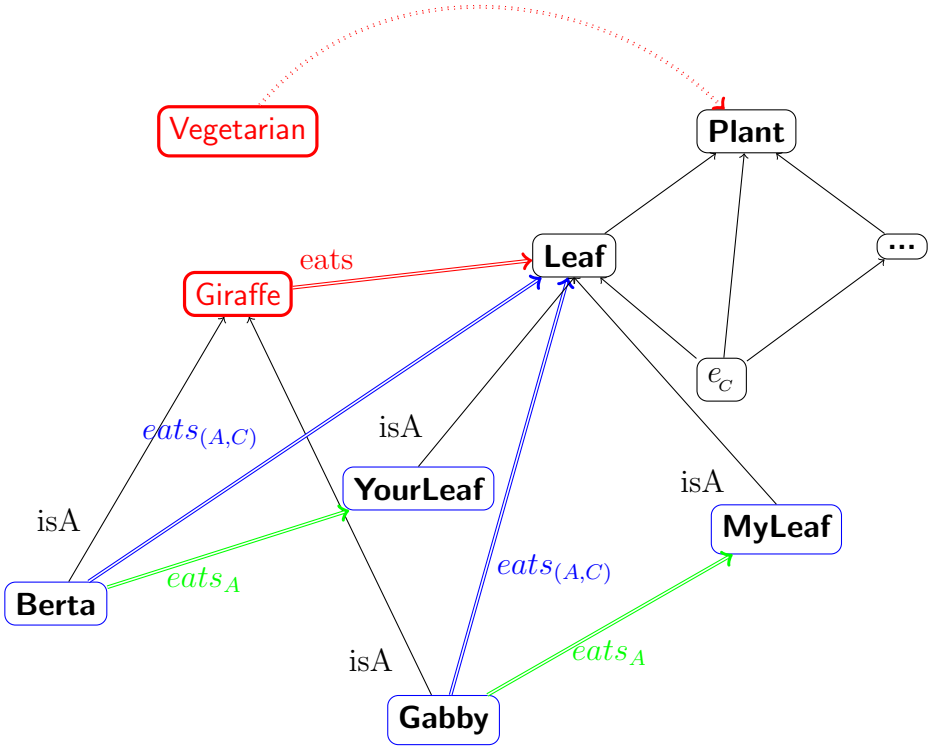


Figure 4.5: Living things ontology

We consider the example in Figure 4.5. From the relations *Giraffe* `eats` *Leaf* and $Leaf \sqsubseteq_C Plant$, we infer that *Giraffe* $\texttt{eats}^{\uparrow}$ *Plant* (1).

Using the ontology specification, the vegetarian definition equation can be written as $Vegetarian = \{c \mid c \in C \wedge c\ \texttt{eats}^{\uparrow}\ Plant\}$ (2).

From (1) and (2) we infer that $Giraffe \in Vegetarian$. This non-structural knowledge about the domain is given by formulas on the language of the ontology $\mathcal{O}$.
The ontology specification is given by the set of axioms defined in Sections 4.1 and 4.2, plus all the domain specific axioms, such as definitions of abstract concepts.

Chapter 5

Discussion

In this chapter, we discuss various aspects of the Information System. In Section 5.1, we discuss some possible application domains for which the use of the information system presented in Chapter 4 is suitable. We also discuss the importance of such techniques and applications. In Section 5.2, we assess the strengths and weaknesses of the main contributions.

5.1 Discussion

Ontologies have generally been viewed as a hierarchical structure based on the `isA` relation. We have seen in the literature survey that the `isA` relation can have many interpretations and allows the creation of statements such as *Girrafe* `isA` *Vegetarian* and *Berta* `isA` *Giraffe*. From the automation point of view, the two statements are completely different, and trying to translate them into a language to be used by a computer (for automated reasoning) is a daunting task. Our research aims to solve this problem by clearly defining relationships and their domain and co-domain, in

such a way that a translation from English (using an ontology) to a specification language is an easy task. For this reason, our hierarchy is based not on the `isA` relation, but on the strictly structural `partOf` relation. All other relationships in the ontology are treated as a family, with certain properties.

We have based our work on the Description Logic idea, however, we have gone one step further in applying the separation of concerns. In our research, we define the ontological structure as one part of the T-box: the set of concepts and their relationships. The second part of the T-box, the actual concept definitions, are contained in the ontology specification. In DL, the A-box contains the entities and their relationships to the associated concepts. With our data set structure we describe the theory that governs data sets. By separating the information system into all these modules, we allow for multiple interpretations (the ontology specification) to be applied on one ontology definition ($\mathcal{O}$). In a similar way, we allow for multiple data set structures ($\mathcal{A}$) to be linked to an ontology. Also, the reasoning engine is now completely separated from the information system. Because we have expressed the information system as a mathematical structure, any existing reasoners that deal with mathematical theories can be used in order to discover or generate knowledge. We illustrate the differences between the Information System and Description Logic with Figure 5.1.

The dynamic aspect of data is captured using our formal definition of ontology w.r.t. the organised data set. Assume the ontology has been defined at a point in time where a certain set of data existed. When a new set of data is acquired, the ontology can be extended with new structural information or new relationships, based on the new

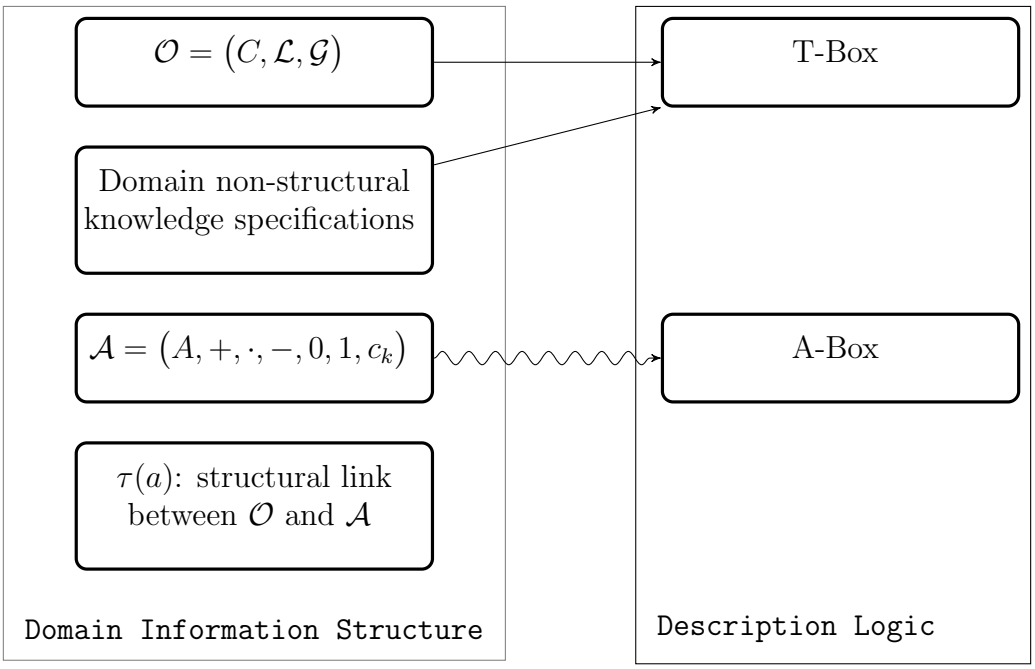


Figure 5.1: Information Structure vs Description Logic

data set. Based on the parthood of entities in the dataset, we can either extend the lattice with new concepts, or new lattices could be constructed for new main concepts.

The concepts outside the lattice can themselves have parts on their own. When we look at the ontology, we see a pattern formed by the lattice and its family of rooted graphs. This pattern can be applied every time we consider a concept in the ontology to be the main concept (the top of the lattice). This allows us to "slice" the ontology and analyse this new view, based on which concept is considered to be the main one. The data set is the one that drives the choice of the main concept, as the ontological structure (the lattice) describes the structure of the organised data set.

Regarding the `partOf` relationship, they way we have simplified the parthood might

not be enough in certain conditions. In our work we consider that two parts combine into one unique concept. That might not be true in real life, when we can have same parts making up different concepts, depending on the order of combination or other considerations. For example, two different concepts might be made up of the exact same parts; e.g., a table and a chair will both have legs and a top. Our work does not deal with such situations.

We find very few examples of ontologies linked to data. Usually researchers either start from a data set and build the ontology, or start with an ontology and crowd-source the creation of data sets. Below we present a case study, mainly employing the second method.

5.1.1 Restaurant Menu - Illustrative Example

We have started from the Pizza Ontology as defined in [Piz] and we have it modified to a Restaurant Menu ontology, as shown in Figure 4.3.

The ontology structure, $\mathcal{O}$ is based on the set of concepts, given by

$C = \{Menu, Pizza, Soup, Base, Topping, Ingredient, Broth, Herb, Cheese, Meat,$

$Vegetable, Vegetarian, NonVeg, Cheesey, e_C\}$. The lattice $\mathcal{L}$, is given by $L \subseteq C$,

$L = \{Menu, Pizza, Soup, Base, Topping, Ingredient, Broth, Herb, e_C\}$. The family of rooted graphs, $\mathcal{G}$ is given by $\mathcal{G} = \{G_1, G_2, G_3\}$, where

$G_1 = \big(C_1, R_1, Pizza\big), C_1 = \{Pizza, Vegetarian, NonVeg, Cheesey\}, R_1 = \texttt{isA}$;

$G_2 = \big(C_2, R_2, Topping\big), C_2 = \{Topping, Cheese, Meat, Vegetable\}, R_2 = \texttt{isA}$;

$G_3 = \big(C_3, R_3, Ingerdient\big), C_2 = \{Ingerdient, Meat, Vegetable\}, R_3 = \texttt{isA}$.

For the $\mathcal{A}$ structure, we use the following (simplified) data:

Category

ID	Name	PID	ID	Name	PID
1	Pizza		7	Vegetarian	1
2	Soup		8	NonVeg	1
3	Base	1	9	Cheesey	1
4	Topping	1	10	Cheese	4
5	Ingredient	2	11	Meat	4, 5
6	Herb	2	12	Vegetable	4, 5

Basic

ID	CategID	Name	Price
1	10	Mozzarella	7.95
2	10	Parmesan	15.95
3	11	Pepperoni	2.95
4	11	Chicken	10.95
5	12	Tomato	3.99
6	3	ThinAndCrisp	0.95
7	3	DeepPan	1.25

Menu Item

ID	CategID	Name	Price
1	7	Mediteranean	12.95
2	8	Hawaiian	15.95
3	9	3Cheeses	12.95
4	2	Chicken Noodle	4.95
5	2	Minestrone	3.99

Besides the two structures, the information system defines the type functions:

$\tau' : A \to C$

$\tau'(Tomato) = Vegetable$

$\tau'(Chicken) = Meat$

The type of an entity is given either by associated concept, when the concept is in

the lattice, or by a chosen relationship otherwise.

$\tau(DeepPan) = Base$

$\tau(Tomato) = Topping$

$\tau(Tomato) = Ingredient$

The domain non-structural specification is given by a set of axioms, such as the recipe formula:

$Meat \notin Vegetarian$

$Meat \notin Minestrone$

5.2 Assessment of the Contributions

In this section, we discuss the strengths and weaknesses of the main contributions presented in this thesis. It is important to highlight both the strengths and weaknesses of the theory developed so that we are able to further refine a solution to the problem of structural reasoning on organised data sets using ontologies.

5.2.1 Strengths of the Contributions

By formalizing the information system encompassing both the ontology structure and organised set of data, the new structure can be used to automate knowledge extraction. We have seen in Section 1.2.2 that there exists automated tools to generate conjectures from mathematical theory. These tools can be now integrated with our domain information system and used to generate conjectures and prove (or disprove) them.

By separating the information system from the reasoning engine, we can allow multiple reasoners to be applied for different tasks, with no modifications needed in the information system. In a similar way, separating the ontology structure from its domain non-structural specification, we allow the application of different specifications or interpretations on the same ontology definition. This is similar to the contextual-ontologies [BAF+06]. For example, if we have the concept *Vegetarian* in the ontology, in one specification we can define it as "An animal who eats only non-animal products" while in another we can define it as "A human who eats vegetables, fruits, and fish". The concept will stay the same, we do not need to change the ontology and the existing relationships between concepts, we only change the axiomatic definition of concepts in the domain non-structural specification.

The separation of concerns in the information system allows for concurrent interpretation of large sets of data. By taking a set of data and applying different ontologies and interpretations, we can generate knowledge specific to various sectors of domains, without having to manipulate the entire data set. Only the concepts, entities, and individuals specific to the domain will be looked at. This allows for rapid analysis and knowledge generation.

5.2.2 Weaknesses of the Contributions

The proposed structure for a domain information system is formed by layers of mathematical structures that bring some structural complexity with it. It also brings a

steep learning and usability curve with it. Moreover, at this stage of our work, we only conjecture the usefulness of this structure without demonstrating it.

5.3 Conclusion

The framework presented aims to standardize a domain that has seen little formalization so far. Aside from describing a new mathematical theory that can be used by existing reasoning engines without any modifications done to the reasoners, our research allows for multiple interpretations of the same set of data and ontology to be applied in parallel. This enables analysis of large data sets in a more efficient way.

Chapter 6

Conclusion and Future Work

In this thesis, we described a new approach to knowledge representation through formal ontologies and structured data sets. We looked at existing formalizations for knowledge representation, both through ontologies and through other mathematical structures, and we have examined the literature for a link between ontologies and data sets. Based on this survey we have presented a unified view on ontologies and data structures and we have bridged the formal field of mathematics to that of the ontological application domains. We have articulated a new mathematical structure that captures an ontology for which we have shown that the relation `partOf` forms a Boolean lattice on a subset of concepts, while other relations form a family of rooted graphs, whose roots are in the lattice. We have also articulated a new mathematical theory that governs the data, a theory based on Tarski's cylindric algebra. We have described the structural link between the two mathematical structures, using the typed data operator, thus forming a new mathematical structure, which we call the domain information structure. We have shown how the typed data operators enable multiple interpretations of the same data set. We have also described the domain

non-structural specification, which allows us to capture the application domain in more detail. We have described a few ways to extend the ontological relationships and show that they enable basic reasoning on the domain information structure.

6.1 Future Work

In this section we look at future work in the area of knowledge discovery through ontologies and structured data sets. One aspect of knowledge discovery we have not examined yet is the reasoning engine. In future work, we will investigate how we can improve reasoning on a domain information structure, as well as build automated tools for reasoning.

Another aspect to investigate is parallel reasoning: either by applying multiple datasets on the same abstract ontology, or applying multiple domain ontologies to the same data set. This way we will allow the analysis of the same domain through data sets collected from various sources, as well as the analysis of the same data from multiple points of view, from different domains.

We also see knowledge generation being used in semantic compression. By describing the theory that governs the data in even more detail, we can replace large sets of data by the theory associated to it.

6.2 Closing Remarks

The quest to represent philosophical and natural language structures into a format that could be fed to computers is one that humans have been struggled with for decades. While the problem might never be fully eliminated, by bridging the formal field of mathematics to the more informal field of ontologies, we hope to bring the experts from the two fields together, and offer them a common structure they could work with.

Bibliography

[Add10] Mehdi Adda. A pattern language for knowledge discovery in a semantic web context. *International Journal of Information Technology and Web Engineering (IJITWE)*, 5(2), 2010.

[AJH+04] J.M. Aerts, P. Jans, D. Halloy, P. Gustin, and D. Berckmans. Labeling of cough data from pigsfor on-line disease monitoring by sound analysis. *American Society of Agricultural and Biological Engineers*, 48(1):351–354, 2004.

[Ast] Astroml: Machine learning and data mining for astronomy. `http://www.astroml.org`.

[BAF+06] Djamal Benslimane, Ahmed Arara, Gilles Falquet, Zakaria Maamar, Philippe Thiran, and Faiez Gargouri. Contextual ontologies. In Tatyana Yakhno and ErichJ. Neuhold, editors, *Advances in Information Systems*, volume 4243 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2006.

[BC98] Trevor Bench-Capon. The role of ontologies in the verification and

validation of knowledge based systems. *Database and Expert Systems Applications*, 1998.

[Bec03] Sean Bechhofer. Owl reasoning examples. `http://owl.man.ac.uk/2003/why/latest/`, 2003.

[BFO] Bfo (the basic formal ontology). `http://ifomis.uni-saarland.de/bfo/`.

[BL04] Ronald J. Brachman and Hector J. Lavesque. *Knowledge Representation and Reasoning*. Elsevier, 2004.

[BM41] Garrett Birkhoff and Saunders MacLane. *A Survey of Modern Algebra*. Macmillan, New York, 1941.

[Bra83] Ronald J. Brachman. What is-a is and isn't. an analysis of taxonomic links in semantic networks. *IEEE Computer*, 1983.

[CBW99] Simon Colton, Alan Bundy, and Toby Walsh. Automatic concept formation in pure mathematics. *Machine learning*, pages 786 – 791, 1999.

[CGL+12] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Marco Montali, and Ario Santoso. *Web Reasoning and Rule Systems: 6th International Conference, RR 2012, Vienna, Austria, September 10-12, 2012. Proceedings*. Springer, 2012.

[CJB99] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are ontologies, and why do we need them? 1999.

[Cod70] E.F. Codd. A relational model of data for large shared databanks. *ACM*, 13:377–387, 1970.

[CYC] Cyc. `http://www.cyc.com`.

[Dav93] Randall Davis. What is a knowledge representation? *AI Magazine*, (14):17–33, 1993.

[dBH04] Jos de Bruijn and Stijn Heymans. On the relationship between description logic-based and f-logic-based ontologies. (2), 2004.

[DJBF+08] Lisa Di-Jorio, Sandra Bringay, CÃ©line Fiot, Anne Laurent, and Maguelonne Teisseire. Sequential patterns for maintaining ontologies over time. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5332 of *Lecture Notes in Computer Science*, pages 1385–1403. Springer, 2008.

[DLT07] S. Dzeroski, P. Langley, and L. Todorovski. *Computational Discovery*, chapter Computational Discovery of Scientific Knowledge, pages 1–14. Springer, 2007.

[DP90] B.A. Davey and H.A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[Dub] Dublin core. `http://dublincore.org/`.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specifications*. Springer, 1985.

[Eps88] S. Epstein. Learning and discovery: one system's search for mathematical knowledge. *Computational Intelligence*, 4:42–53, 1988.

[FOA] Foaf - friend of a friend. `http://xmlns.com/foaf/spec/`.

[FPSS96a] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AAAI Magazine*, pages 37–54, 1996.

[FPSS96b] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Advances in knowledge discovery and data mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[FT12] B. Furletti and F. Turini. Knowledge discovery in ontologies. *Intelligent Data Analysis*, 16:513–534, 2012.

[GAM03] R. Girju, A.Badulescu, and D. Moldovan. Learning semantic constraints for the automatic discovery of part-whole relations. 2003.

[Gil02] Alan Gilchrist. Thesauri, taxonomies and ontologies ± an etymological notthesauri, taxonomies and thesauri, taxonomies and ontologies - an etymological note. 2002.

[GO] Go: Gene ontology. `http://geneontology.org/page/ontology-relations`.

[GPL04] James Geller, Yehoshua Perl, and Jintae Lee. Ontology challenges: A thumbnail historical perspective. *Knowledge and Information Systems*, 6:375–379, 2004.

[Gru95] Thomas Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, 1995.

[GUM] Gum (the generalized upper model). `http://www.ontospace.uni-bremen.de/ontology/gum.html`.

[GWW09] Krzysztof Goczyla, Aleksander Waloszek, and Wojciech Waloszek. Algebra of ontology modules for semantic agents. In NgocThanh Nguyen, Ryszard Kowalczyk, and Shyi-Ming Chen, editors, *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, volume 5796 of *Lecture Notes in Computer Science*, pages 492–503. Springer, 2009.

[HC06] MichaelJohn Healy and ThomasPreston Caudell. Ontologies and worlds in category theory: Implications for neural systems. *Axiomathes*, 16(1-2):165–214, 2006.

[Hea92] Marti Hearst. Automatic acquisition of hyponyms from large text corpora. 1992.

[HH93] William Hatcher and Michel Hebert. *Model Theory*, 1993.

[HSSK06] J.A. Hardin, M. Shahbaz, Srinivas, and A. Kusiak. Data mining in manufacturing: A review. *Journal of Manufacturing Science and Engineering*, 128:969–976, November 2006.

[HSW97] G. Van Heijst, A. Th. Schreiber, and B. J. Wielinga. Using explicit ontologies in kbs development. 1997.

[IL84] Tomasz Imielinski and Witold Lipski. The relational model of data and cylindric algebras. *Journal of Computer and System Sciences*, 28:80–102, 1984.

[JD01] Michael Johnson and C.N.G. Dampney. On category theory as a (meta) ontology for information systems research. *FOIS (Formal Ontology in Information Systems)*, 2001.

[JDB11] Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47-3:251–289, Oct 2011.

[Ken03] David Kennerly. Better game design through data mining. `http://www.gamasutra.com/view/feature/131225/better_game_design_through_data_.php`, 2003.

[KHES05] Markus Krotzsch, Pascal Hitzler, Marc Ehrig, and York Sure. Category theory in ontology research: Concrete gain from an abstract approach. Technical report, U of Karlsruhe, 2005.

[KS98] Jurg Kohlas and Robert F. Stark. Information algebras and consequence operators. 1998.

[LD04] V. Leemans and M. F Destain. A real timegrading method of apples based on features extracted from defects. *Journal of Food Engineering*, 61:83–89, 2004.

[LG05] Eric Little and GalinaL.Rogova. Ontology meta-model for building a situational picture of catastrophic events. 2005.

[LK08] Sheng-Tun Li and Shu-Ching Kuo. Knowledge discovery in financial investment for forecasting and trading strategy through wavelet-based {SOM} networks. *Expert Systems with Applications*, 34(2):935 – 951, 2008.

[Mar] Marinetlo: a top-level ontology for the marine domain. `http://www.ics.forth.gr/isl/MarineTLO/`.

[Mar00] David Marker. *Model Theory: An Introduction*. Springer, 2000.

[MBA07] Roy McCasland, Alan Bundy, and Serge Autexier. Automated discovery of inductive theorems. *Special Issue of Studies in Logic, Grammar and Rhetoric on Computer Reconstruction of the Body of Mathematics: From Insight to Proof*, 10-23(A14):135–149, 2007.

[NKCN08] P. Nguyen, Ken Kaneuwa, Dan Corbett, and Minh-Quang Nguyen. An ontology formalization of relation type hierarchy in conceptual structure theory. 2008.

[O'L91] Daniel O'Leary. Knowledge discovery as a threat to database security. 1991.

[Oxf] Oxford dictionaries: Ontology. `http://www.oxforddictionaries.com/definition/english/ontology`.

[PB01] Joseph Phillips and Bruce Buchanan. Ontology-guided knowledge discovery in databases. *K-CAP'01*, 2001.

[PGS06] Yury Puzis, Yi Gao, and Geoff Sutcliffe. Automated generation of interesting theorems. In *Proceedings of the 19th International FLAIRS Conference*, volume A22, pages 49–54, 2006.

[Piz] Pizza ontology. `http://owl.cs.manchester.ac.uk/research/co-ode/`.

[PLSG05] C. Phua, V. Lee, K. Smith, and R. Gayler. A comprehensive survey of data mining-based fraud detection research. *Artificial Intelligence Review*, pages 1–14, 2005.

[Rah14] Bijan Raheemi. Data mining and knowledge discovery in healthcare and medicine. *IEEE Ottawa*, 2014.

[RV07] C. Romero and S. Ventura. Educational data mining: A survey from 1995 to 2005. *Expert Systems with Applications*, 33:135–146, 2007.

[Ski] The skicat system for processing and analyzing digital imaging sky surveys. `http://adsabs.harvard.edu/abs/1995PASP..107.1243W`.

[SNA] Social network analysis theory and applications. `http://train.ed.psu.edu/WFED-543/SocNet_TheoryApp.pdf`.

[Sow00] John Sowa. *Knowledge Representation: logical, philosophical, and computational foundations*. 2000.

[Spe06] Andrew Spear. *Ontology for the Twenty First Century: An Introduction with Recommendations (BFO)*, 2006.

[SS91] Gunther Schmidt and Thomas Strohlein. *Relations and Graphs*. Springer, 1991.

[SSTW01] M. Shaw, C. Subramaniam, G.W. Tan, and M. Welge. Knowledge management and data mining for marketing. *Decision Support Systems*, 31:127–137, 2001.

[THM71] A. Tarski, L. Henkin, and J.D. Monk. *Cylindric Algebras*. North-Holland, 1971.

[TSD10] E. Turban, R. Sharda, and D. Delen. *Business Intelligene*. Prentince Hall, 2010.

[UPCMA04] A. Urtubia, J. R. Perez-Correa, M. Meurens, and E. Agosin. Monitoring large scale wine fermentations with infrared spectroscopy. *Talanta*, 64(3):778–784, 2004.

[Var15] Achille Varzi. Mereology. `http://plato.stanford.edu/entries/mereology`, 2015.

[WCH87] Morton Winston, Roger Chaffin, and Douglas Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11:417–444, 1987.

[Zin07] Chaim Zins. Conceptual approaches for defining data, information, and knowledge. *Journal of the American Society for Information Science and Technology*, 58(4):479–493, 2007.