A FORMAL APPROACH TO ONTOLOGY MODULARIZATION AND TO THE ASSESSMENT OF ITS RELATED KNOWLEDGE TRANSFORMATION

A FORMAL APPROACH TO ONTOLOGY MODULARIZATION AND TO THE ASSESSMENT OF ITS RELATED KNOWLEDGE TRANSFORMATION

BY

ANDREW LECLAIR, M. A. Sc.

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

DOCTOR OF PHILOSOPHY IN ENGINEERING

Doctor of Philosophy in Engineering (2021) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	A Formal Approach to Ontology Modularization and to the Assessment of its Related Knowledge Transformation
AUTHOR:	Andrew LeClair M. A. Sc., McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Ridha Khedri
NUMBER OF PAGES:	x, 140

Abstract

Knowledge-based systems are composed of multiple agents that each must utilize their understanding of the world to deduce new facts and make intelligent decisions. The understanding of the world that each agent has is formalized using an ontology: a structure which conceptualizes the domain as a set of concepts and their relations. To accommodate the numerous agents, and to minimize the cost of a single monolithic ontology, modularization is used to provide each agent with only the knowledge they require.

This thesis addresses the problems related to defining the transformation of the domain knowledge due to ontology modularization. Existing ontology modularization techniques are evaluated so that they can be assessed with respect to how they transform knowledge, as well as their computational performances. A primary objective is to adapt the existing modularization techniques so that they can be compared based on how they transform domain knowledge.

The objectives of this thesis were addressed systematically by first comprehensively evaluating the literature regarding both ontologies and ontology modularization. Then, we adapted the prominent ontology modularization techniques to the ontology formalism used in this thesis: a Domain Information System. New modularization techniques were then explored which utilized aspects of the domain ontology's

algebra.

This research consolidates the many approaches to ontology modularization into a single set of formal techniques for a domain ontology. As a result, we further the modularization field by unifying the modularization field under a common formalism, creating relations between the modularization techniques, as well as formulating how the knowledge of the ontology is transformed for each technique. Additionally, we introduce a new modularization technique which capitalizes on the Boolean algebra of a domain ontology, and enables the determination of a module based on the desired granularity of the domain knowledge under consideration.

The research of this thesis furthers the ontology modularization field by systematically analyzing current limitations with respect to their ability to determine how knowledge is transformed. We demonstrate that by using the formalism of the domain ontology, we are able to formulate different modularization techniques, each with its own transformations to the domain knowledge. Expressing the knowledge that is transformed via modularization is unique. It is also instrumental in providing the agents of a knowledge-based system with modules that capture the exact domain knowledge they require to make their decisions.

Acknowledgements

To all those who were my soundboard, the ideas would not be formed as much as they are in this thesis without you. And to all those who mentored me, the pride I have in this work is thanks to you.

And to my family who supported me through these five years, nurturing me to complete this thesis despite all circumstances. This would not have been possible without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advancement of modern computer systems and the ease of collecting data, systems that can reason new facts, referred to as knowledge systems, have become a significant area of research. As a result of the cost-effectiveness and improved technology of storage, the amount of data that is a part of these systems is enormous, and further, the data comes from an assortment of domains [98]. Machine learning technology and data processing can be utilized to process the large volumes of data [100]. However, these techniques do not utilize the knowledge of the domains from which the data comes. Knowledge systems instead aim to reason new facts by processing the data in the context of a domain to produce domain knowledge [5]. One difficulty of creating knowledge systems is the necessity of defining a domain so that it is possible to incorporate it to the reasoning process. A domain should provide a context for the data by describing the relationships and concepts that exist in the domain, and how the data relates to these relationships and concepts. The definition of the domain must also be formal so that it can be used in automated reasoning tasks. One method of formally defining a domain is using an ontology.

An *ontology* is used as the structure for knowledge systems and is defined in [49] as "a formal, explicit specification of a shared conceptualization". As previously stated, an ontology provides the formalization necessary to reason and acquire knowledge from a domain. The above definition requires an understanding of what a conceptualization is, what a formal and explicit specification is, and what it means to be shared. A conceptualization is often understood as a relational structure, that is, the world being conceptualized can be represented using concepts and relations. Secondly, the formal and explicit specification is often accomplished using a logic such as Description Logic (DL) [10]. Lastly, the conceptualization may be shared among several agents via ontological commitments. An ontological commitment is an agreement on how the domain (and the concepts and relationships therein) is understood [48]. Therefore, although an agent may not need an entire conceptualization but rather a smaller part, it understands the concepts and relations in the same way as another agent that shares the same ontological commitment to the ontology. This ensures that two agents using the same ontology do not come to different conclusions, resulting in inconsistency or contradictions.

Current ontologies are typically represented as either a taxonomy of concepts (e.g., [102]) or as a graph of concepts (e.g., [31]). They often do not incorporate the data into the hierarchy, and thus, are less suited for the problems associated with the data itself. In particular, a traditional ontology suffers from problems such as information evolution and integration, and data heterogeneity. There is developing research for efficiently utilizing data with an ontology, and most utilize the notion of XML schemas or using the Web Ontology Language (OWL) language [18]. However, a primary issue is that the incorporation of data into the ontology results in bloating

the system which makes reasoning tasks difficult [29]. An alternate approach is to design a system that separates the data from the ontology (e.g., [40]). Such a separation ensures that the ontology is strictly a conceptualization and is not encumbered with data. These systems allow for data storage technologies, such as databases, in conjunction with ontology conceptualization technologies, such as utilizing the Terminological Box (T-Box) of DL. Since the data is separated from the conceptualization, a mapping must be defined to relate the data to the concepts. Examples of systems that separate the data from the concepts are Ontology-based Data Access (OBDA) [130] and Domain Information System (DIS) [84, 85, 86]. These systems both involve a mapping function that relates the data component (where the data is) to the ontological component (where the concepts are).

1.1 Specific Context

In an ontology-based system, there are many agents interacting with a single ontology. The single ontology conceptualizes a domain for the many agents thanks to the agents agreeing on an ontological commitment. Via an ontological commitment, each agent will interpret each concept and relation in the same way. Often, an agent does not need to use the entire ontology because either they are concerned with only a small subset of concepts, or they have the authorization to access only a subset of concepts. The subset of concepts that an agent is provided with is referred to as a module, and a module is produced via a process referred to as modularization. We will be elaborating further on modules and the process of modularization in Chapter 4. It should be the case that it is known a priori what knowledge the agent can learn from the given module. By doing this, we are able to address questions regarding what an

agent can learn. For instance, whether the agent has a module that is appropriate for their decision-making, or whether the agent has a module that allows them to learn facts they do not have the authorization to learn. Therefore, any knowledge the agent needs or is able to learn ought to be within the module. Ideally, from the agent's perspective for a given query, the knowledge they can learn from the module should be no different than what they can learn from the ontology. Governing bodies, such as the European Union Agency for Network and Information Security, articulates needs those needs with titles such as provable privacy and agent autonomy for a knowledge system [28]. They state that it should not be the case that an agent is able to learn facts from a module that it was not intended to know. To ensure that an agent cannot learn unintended facts, it is required that the knowledge within a module be determined a priori in its scope, at the stage of modularization.

If we are to use autonomous agents in a multi-agent system, the problems introduced in [28] must be addressed. These problems include provable privacy in modules [9, 19], agent autonomy over their knowledge [35], and domain evolution [13, 25]. To tackle these problems, we need to determine the knowledge that can be learnt within a module. We must formally define a module, and the associated modularization that produces it. In doing so, the knowledge that can be determined via reasoning can be determined much like how the knowledge is determined in an ontology.

A result of determining what knowledge can be learnt through a module is the ability to measure or quantify the knowledge that is lost or modified due to modularization. Although it was stated that ideally a module should allow for an agent to reason facts as though it were reasoning on the ontology, this is often not the case. A module is a subset of concepts, and so there will be an absence of knowledge

associated with the absence of concepts. Therefore, we should be able to determine what knowledge has been lost due to the modularization process. This is no easy task as the knowledge can be transformed in many ways due to modularization. For instance, knowledge refinement, or the change in the granularity of knowledge, describes how knowledge can be modified rather than strictly lost. The granularity of knowledge is an abstract property of knowledge and is related to understanding what the elementary building blocks of knowledge are [97]. Knowledge loss, on the other hand, refers to the knowledge being absent or indeterminable in the module. The lost knowledge can be determined and then characterized through the property of completeness [20]. However, determining the lost knowledge is not trivial and requires the use of a formal ontology (i.e., based on a logic system), such as one formalized with DL or DIS. The ability to refine knowledge, or determine if a specific subset of knowledge is more refined than another is a complex task and often requires entire systems catered to it [34, 63]. Thus, an approach to modularization that better facilitates the determination of knowledge loss or knowledge refinement is essential.

1.2 Motivation

The formalization of modularization techniques presents many challenges to the ontological field. A primary challenge is the formulation of a modularization technique that is based on established algebra so that the module's properties can be expressed with an algebra's theory. This in turn allows for determining the knowledge in the module using algebraic approaches. The process of modularization leads to the second challenge of quantifying or characterizing the knowledge of a module. There are several different motivations for why one would modularize an ontology, and they all

require the ability to communicate what knowledge can be learnt from the module. However, characterizing the knowledge is a difficult and abstract task. Thus, we focus on characterizing the domain knowledge within a module.

The modularization approaches that can be applied to an ontology depend on how the ontology is formalized or articulated. Since the modularization approach produces a module, the formalism of the ontology restricts what kind of module can be produced. The research in this thesis is focused on modularizing domain ontologies that are in a DIS. In this structure, a domain ontology is composed of a Boolean lattice and a set of graphs, where each graph is rooted to some element of the lattice. The algebraic representation of the domain ontology allows for a formal representation of domain knowledge. A modularization technique, and the resulting module, can also be expressed using the established theory for Boolean lattices and their respective Boolean algebras.

DIS, as a newer formalism for ontology-based systems, does not yet have any modularization approaches developed for it. However, the theory of modularization techniques for other ontology formalisms, such as DL or graph theory, can be applied to the domain ontology of a DIS. The domain ontology presents a unique potential for modularization because it can be represented as both a Boolean lattice and a Boolean algebra. This allows us to apply both logical modularization approaches to the Boolean algebra and graph-based modularization approaches to the Boolean lattice. By doing this, we bridge these two areas of research for ontology modularization that are currently distinct. The research of this thesis is motivated by utilizing the dual nature of logical and graphical modularization approaches to a DIS domain ontology so that the knowledge of a module can be formally characterized. The

characterization of knowledge can lead to further applications in knowledge hiding, privacy, security, personalization, or scoped reasoning.

1.3 Problem Statement

The domain ontology in a DIS is able to be captured using a Boolean Lattice. A Boolean lattice is isomorphic to a Boolean Algebra. Hence, we can use either of these two mathematical structures when discussing the domain ontology of a DIS.The modularization techniques developed for a DIS domain ontology ought to utilize aspects of either the lattice, the algebra, or both. Additionally, the domain knowledge that is preserved or lost due to modularization should be able to be characterized using the language of the used mathematical structures. We first require a formal understanding of what a module is in the context of a DIS domain ontology. Following this, we require how one can modularize a DIS domain ontology to produce such a module. If there are multiple ways to modularize, then the differences–communicated using the defined properties of a module–must be expressed. The proposed research aims to develop modularization techniques that utilize the different aspects of the domain ontology, and to characterize the knowledge that is preserved or lost using each technique.

1.4 Research Questions

In this section, we introduce the research questions that this thesis addresses. We decompose each question into smaller subquestions which are answered by this research. By answering the subquestions, we answer the more broad question that they

are related to.

1. How can we define key concepts needed for modularizing a domain ontology of a DIS
   (a) What is a module, and what are the properties that are useful in describing it?
   (b) What is the process of modularization?
   (c) What is the relationship between a module and the domain ontology, from which it is modularized?
2. Under what conditions can we modularize a domain ontology of a DIS, and what are the consequences?
   (a) What ontology modularization techniques from literature can be applied to a domain ontology?
   (b) How can these modularization methods be formalized using the language of the domain ontology?
   (c) How do the implemented methods compare to the methods found in the literature?
   (d) What properties can be expressed of the modularization method and the produced module(s)?
3. Can we determine how the knowledge of the domain ontology transformed due to modularization?
   (a) How can we define or characterize the knowledge within a module?
   (b) How is the knowledge transformed?

1.5 Objectives

The following objectives are a response to the previously stated research questions. These objectives, and are markers for the accomplishments of the research presented in this thesis. We show the chapter and section that each objective is addressed in in Table 1.1.

Objective	**Chapter & Sections**
Obj. 1	Chapter 4
Obj. 2(a)	Chapter 2
Obj. 2(b)	Section 5.1, 5.3, 6.1, 6.2
Obj. 2(c)	Section 5.4, 6.4
Obj. 2(d)	Section 6.3
Obj. 3	Section 5.2, 6.2

Table 1.1: Correlation between objective and chapters or sections

The objectives that this thesis addresses are as follows:

Obj. 1: Formalizing Key Concepts Related to Ontology Modularization for a Domain Ontology. The following are its subobjectives:

(a) Define a Domain Ontology Module and Modularization

(b) Establish a Relationship Between a Module and a Domain Ontology

Obj. 2: Formalize Ontology Modularization for a Domain Ontology. The following are its subobjectives:

(a) Explore Current Ontology Modularization Techniques

(b) Articulate the Modularization Techniques

(c) Compare the Articulated Techniques to those in Literature

(d) Articulate the Properties of Domain Ontology Modularization and the Associated Module

Obj. 3: Determining how Knowledge is Transformed due to Modularization. The following are its subobjectives:

(a) Articulate what Knowledge is within a Module

(b) Determine how the Knowledge has been Transformed

1.6 Methodology

The objectives were conducted in a sequential and procedural way. The first objective requires the formal definition of what a module is, and what modularization is. Predominant definitions of a module were explored in Chapter 2, and common properties were determined that must also be exhibited by a module of a domain ontology. The definition of a module was provided using the theory of DIS. To express the properties that were determined useful in describing a module, the DIS theory was extended with relations that allowed for module comparison. Finally, modularization itself was defined using this definition of a module.

The second objective requires that the existing modularization techniques for ontologies is examined so that new techniques can be developed for a DIS domain ontology. This objective was accomplished by surveying existing literature following a systematic process outlined by Kitchenham et al. [64]. The literature review was produced as a self-contained paper, which can be found in [79]. The goal of the literature review was to determine what ontology modularization techniques exist, what ontology formalisms they can be used on, and any notable properties of the

modularization technique, such as what components of the ontology is utilized for the modularization process. We completed a quantitative study of the existing literature to extract this data. Ontology modularization techniques that operated on the same ontology formalism and sought similar goals could be compared to one another quantitatively (such as in terms of complexity or tractability) but were unable to be compared to modularization techniques that operated on entirely different ontology formalisms. For instance, the comparison of a technique that operates on a specific DL fragment could not be fairly compared to one that operates on a simple relational graph structure in terms of complexity. Thus, there was a qualitative component to the assessment that fulfilled this comparison which was guided by questions such as the determination of what the motivation for modularization was, and what the desired properties of the module are. Since the literature review that is produced is used to guide the development of modularization techniques in this research, it was essential that the literature review be conducted with the utmost scrutiny and attention to detail.

The remainder of Objective 2 involves the articulation of modularization techniques and elaboration of DIS theory. The first modularization technique articulated utilized lattice theory to express a module. Specifically, it utilized the theory of Boolean lattices, and the theory of ideals. Boolean lattice theory is necessary to properly understand and use the central component of a domain ontology of DIS: a Boolean lattice. The theory of ideals, and specifically of principal ideals, is used to communicate what a module is, and how it can be determined. The second modularization technique expanded the first technique by incorporating the theory of filters, Boolean algebras, and Boolean subalgebras. The relationship between a Boolean

lattice and its respective Boolean algebra is crucial for formulating the second modularization technique. To understand this relationship, it requires an understanding of group and ring theory. The third and final modularization technique utilized the theory of partition sets, and how one is related to the lattice of all Boolean subalgebras of a given Boolean algbera. The modularization techniques in this thesis are defined as functions, and so the theory of binary relations is necessary to be able to characterize these transformations. As this theory is then added to the theory of DIS, the underlying theories of DIS are also required: graph theory, cylndric algebra, and higher-order logic. All of these are used to communicate how a modularization technique produces a new DIS.

The third objective aims to characterize the knowledge within the ontology and, by extension, the modules. To accomplish this, the isomorphism theorems and kernel theory are used. In the first modularization technique proposed, the modularization is defined as a homomorphism between ideals. Using the first isomorphism theorem, this implies the existence of a kernel. Using kernel theory, we use a kernel to characterize the knowledge loss of the defined modularization technique. Additionally, the theory of Boolean algebras and subalgebras is used to communicate knowledge loss through the preservation of the algebra's operator. Finally, using the relationship between the partition lattice (and its respective refinement relation) and the lattice of all Boolean subalgebras of a given Boolean algebra, we define and present results regarding knowledge refinement.

1.7 Main Contribution

The research of this thesis provides the following contributions:

1. The formalization of a module and modularization

The research presented in this thesis introduces, defines, and explores ontology modularizaton techniques that utilize all aspects of a domain ontology. Currently, there is no generally agreed on definition of what a module is. Typically, a module is defined loosely with no mathematical foundations, leading to the inability to compare modules from different techniques. By extension, since there is no formal understanding of what a module is, there is no formalization for the process of modularization. This also makes it difficult to objectively and quantifiably compare modularization techniques to one other. This research resolves this issue by formalizing both a module and modularization. By establishing formal definitions, we can quantitatively compare the modularization techniques and modules which were previously incomparable, or at worst, qualitatively compare.

2. The development of three modularization techniques to a DIS domain ontology

The ontology modularization field is wide and has numerous techniques that all differ based on approach or type of module produced. The most prominent techniques are adapted to a domain ontology, determined by an in depth literature review. Specifically, we adapt view traversal and the logic-based approach. In addition to adapting existing techniques, an entirely new technique which utilizes the Boolean algebra of the domain ontology is introduced. The adaptation of the two existing techniques demonstrates the simplicity in applying existing techniques to DIS. The development of the third technique demonstrates the utility of the underlying Boolean algebra and how it can be used to develop new and innovative approaches to modularization.

3. The unification of graph-based and logic-based modularization approaches

Although this contribution does not directly correlate with an objective, it is important to note that there currently exists a rift between the graph-based modularization techniques and the logic-based. Although much of the work related to using OWL seeks to bridge this, often, OWL techniques isolate and use just the graph part or just the DL. The work in this thesis bridges this rift by demonstrating how a graphical approach on a domain ontology can be related to a logic based approach (and vice versa). Limitations to this bridging are addressed, specifically showing that not all graph-based modules can be extended to a logic-based. However, the ability to relate the two types of modules under a single ontology representation is a novel finding and is the first step to consolidating the ontology modularization field.

4. The characterization and communication of the knowledge within a domain ontology and module

The field of knowledge with respect to knowledge-based systems is increasing in popularity at an extreme rate. The use of knowledge-based systems and producing means to determine the knowledge associated with a set of data is becoming more important. However, all this research is dependant on how knowledge is defined, and what it is. Currently, knowledge is considered nothing more as deducible facts, and there is no sure way of knowing what is not known in a system. The research in this thesis formalizes what knowledge is, and further, what knowledge can exist for a module given the domain ontology it comes from. This is significant when related to topics of provable privacy in an autonomous agent setting. To determine what an agent does (or does not) know, it is essential to be able to determine what it could know.

5. The determination of knowledge loss and knowledge refinement due to modularization

The formalization of what knowledge is allows us to describe how it is transformed. Further, the formalization allows for a more nuanced approach to the transformation of knowledge by differentiating between knowledge loss and knowledge refinement. This research demonstrates how we can measure the knowledge that is lost due to the modularization process, and also how knowledge can be refined by modularization. Further, it allows us to compare the knowledge from one module to another using these properties. For example, whether one module contains more knowledge than another, or whether one module has more refined knowledge than another. The ability to compare the knowledge of one module to another allows for the comparison of agents in a multi-agent system based on the knowledge that they contain.

1.8 Related Publications

We present the publications that are results from the research of this thesis in the following list. We elaborate the impact of each paper in the paragraphs that follow.

1.8.1 Journal Articles

1. A. LeClair, A. Marinache, H. Ghalayini, R. Khedri, and W. MacCaull. A systematic literature review and discussion on ontology modularization techniques. *IEEE Transactions on Knowledge and Data Engineering*, 2020. Review

This survey explored all modularization techniques on ontologies, regardless of how the ontology was formalized. A discussion on open areas of research, specifically on developing a modularization technique that incorporates aspects from both logical and graphical approaches (which are topics discussed in the survey). The results

of this paper were used as motivation for the following papers regarding ontology modularization, and identified techniques and key properties that a modularization technique for a DIS domain ontology must have.

2. A. LeClair and R. Khedri. An algebraic approach to ontology modularization and knowledge refinement. *Journal of Theoretical Computer Science*, 2021. Review

To complement the graphical modularization technique applied to a DIS domain ontology, an algebraic technique was developed to demonstrate the ideas of the logical modularization techniques. In this paper the set of all Boolean subalgebras is utilized to determine specific modules that satisfy the properties of local correctness and local completeness that logical modularization techniques require. The use of the Boolean subalgebras also allows for the definition of knowledge refinement using the partition refinement relation. To the extent of our knowledge, this is one of the first explorations for formally defining knowledge refinement in the context of an ontology.

3. A. LeClair, J. Jaskolka, W. MacCaull, and R. Khedri. Architecture for ontology-supported multi-context reasoning systems. *Data & Knowledge Engineering*, 2021. Review

An architecture for an ontology-supported multi-context reasoning system is proposed. The architecture is independent of ontology formalism, and thus, the modularization approaches proposed in this thesis are not necessarily used in the paper. However, the approach of maintainability promoted by modularization is used as a major motivation for the architecture. It is the impetus for utilizing the Presentation-Abstract-Control style as the basis of the architecture thanks to its natural separation of concerns.

1.8.2 Conference Papers

4. A. LeClair., R. Khedri., and A. Marinache. Toward measuring knowledge loss due to ontology modularization. In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 2: KEOD*. INSTICC, SciTePress, 2019

In this paper, the ideas of graphical modularization techniques are applied to the domain ontology. In particular, utilizing the relational structure of the ontology to extract concepts that constitute a module that is defined and characterized by size and closeness of concepts. The work goes beyond this by formalizing the view traversal technique using the underlying Boolean algebra. This allows for the declaration of lost knowledge due to view traversal on a DIS domain ontology, specifically due to the inability to preserve the complement operator. It is in this work that knowledge loss is first formally defined.

5. A. LeClair, R. Khedri, and A. Marinache. Formalizing graphical modularization approaches for ontologies and the knowledge loss. In J. Dietz, D. Aveiro, and J. Filipe, editors, *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 1297 of *Communications in Computer and Information Science series*, pages 1–25. Springer, 2021. Invited

In this paper, the view traversal modularization technique, comparable to a graphical modularization approach, is extended so that it satisfies the definition of an algebraic module. The introduced module, a principal ideal subalgebra module, demonstrates how the graphical approach for a DIS domain ontology is not separate from the algebraic approach.

6. A. Marinache, R. Khedri, A. LeClair, and W. MacCaull. Dis: A data-centred knowledge representation formalism. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pages 1–8. IEEE, 2021

A case study which demonstrates the construction of a DIS for a movie domain is presented. In this work, it is shown how the domain ontology is constructed from a dataset, and improved with the addition of rooted graphs. Modularization is discussed (although the techniques of this thesis are not applied in this paper) as a means to maintain the domain ontology.

1.9 Thesis Outline

The remainder of this thesis is structured as follows.

In **Chapter 2**, we perform a literature review on ontology formalisms and ontology modularization. We also examine existing ontology-based system formalisms.

In **Chapter 3**, we present the mathematical background necessary for understanding the research in this thesis. We introduce the DIS formalism, and we introduce the motivating example that will be used to illustrate the modularizations of throughout this thesis.

In **Chapter 4**, we formally define the concepts of what a module is, and what modularization is. It is here we explore the properties related to these concepts.

In **Chapter 5**, we present the first set of modularization techniques on a DIS domain ontology based on the lattice structure. In this chapter we also explore the knowledge that is lost due to the defined modularization techniques.

In **Chapter 6**, we present the other modularization technique that is based on the algebra associated to the domain ontology. Notions of knowledge refinement are also introduced in this chapter, and we explore ways in which modules can be created based on what the desired granularity of knowledge.

Finally, in **Chapter 7**, we explore the direction of future research and address remaining open questions. We then conclude the thesis and present any final remarks.

Chapter 2

Literature Review

In this Chapter we evaluate the literature regarding ontology modularization techniques. To evaluate the ontology modularization techniques, we first examine ontology formalisms and methods for representing ontology-based systems. This evaluation allows us to explore the different types of modularization techniques, and the different motivations that exist for them. After each exploration of a topic in the literature, we provide our thoughts on the current state of the literature, and how it relates to the research presented in a discussion. Often, we compare the current literature to DIS, which is further elaborated in Chapter 3.

2.1 Ontology Representations and Formalisms

2.1.1 Description Logic

The first investigated ontology formalism is DL, which is the predominant formalism used. It is the underlying logic for OWL, a language for writing ontologies [87]. DL

is composed of a family of fragments, each with a different level of expressiveness. A primary motivator for using DL as an ontology formalism is that the core reasoning tasks, such as materialization and consistency checking, are decidable for specific fragments [10]. Within DL, there are two components: the T-Box and the Assertional Box (A-Box). The T-Box contains all axioms which define the concepts and relations of the ontology. Examples of T-Box axioms are

$$Mother \sqsubseteq Parent \tag{2.1.1}$$

which states that the concept *Mother* is a subconcept of the concept *Parent*. With this understanding, any data that is a *Mother* is therefore also a *Parent*, but not vice versa. The axiom

$$Person \equiv Human \tag{2.1.2}$$

states that the concept *Person* is equivalent to the concept *Human*. This relation is more strict than the above, because all data that is a *Person* is also a *Human*, and vice versa. Richer assertions can be declared using quantifiers or symbols introduced in specific fragments. For example,

$$Mother \equiv Woman \sqcap \exists \text{hasChild}.Person \tag{2.1.3}$$

states that the concept *Mother* is equivalent to the intersection of those that are a *Woman* and those that are apart of the relationship *hasChild* with another concept that is a *Person*. The use of the existential quantifier increases what can be expressed, however, it can potentially make reasoning more difficult. Depending on which fragment of DL is used, or on the assertion, reasoning can become undecidable [8]. An

example of a fragment is $\mathcal{O}$, which allows for the use of nominals in an assertion. An example of a nominal is

$$Beatle \equiv john \sqcup paul \sqcup george \sqcup ringo \tag{2.1.4}$$

which states that the concept *Beatle* must take on one of the four values of *john*, *paul*, *george* or *ringo*. Other examples of fragments are $\mathcal{H}$ which allows role hierarchy properties (i.e., *subPropertyOf*), and the $\mathcal{U}$ which allows for concept union. For examples of all fragments, we direct the reader to [10].

The A-Box contains the axioms which instantiate the concepts with data. Examples of A-Box axioms are

$$Mother(\text{julia}) \tag{2.1.5}$$

which states the data value 'julia' is an instance of the concept *Mother*, and

$$parentOf(\text{julia, john}) \tag{2.1.6}$$

which states that the data value 'julia' is related to the data value 'john' via the *parentOf* relation. Some implementations of a DL-based ontology have an additional third component to the T-Box and A-Box: the Rule Box (R-Box). The R-Box contains axioms pertaining the relations that would normally be found in the T-Box [60]. However, the incorporation of an R-Box is not standard and not found in all DL-based ontologies.

Depending on the authors, an ontology is defined as *only* the T-Box, excluding the A-Box entirely [68]. With the onset of big data, there has been efforts to more formally include the A-Box to produce results that are more respective of the domain

and its data. Examples of extending DL to account for problems such as the veracity and variety of the data are fuzzy DL [114] and contextual DL [115]. Despite these efforts to account for the veracity and variety of data, the sheer volume often requires a basic fragment of DL, such as $\mathcal{ALC}$, to be used [33, 56, 127, 132]. The use of a basic fragment of DL restricts what can be expressed, and therefore, limits novel results from reasoning.

2.1.2 OWL

Ontologies that are formalized with DL are often written and implemented using OWL, such as *GALEN* [101], *FOAF* [41], and the Semantic Web [11]. Much like there are different fragments of DL with varying expressiveness, there are several sublanguages of OWL, each supporting different levels of expressiveness [87]. *OWL Lite* allows for the creation of a hierarchy with simple constraints on it, such as restricted cardinalities, property restrictions, and class intersection. *OWL DL* allows for the maximum expressiveness while still retaining computational completeness and decidability, and corresponds to the $\mathcal{SHOIN}^{(\mathcal{D})}$ fragment of DL [54]. It contains all language constructs but restricts their use, such as not allowing a class to be an instance of another class. *OWL Full* allows for the maximum expressiveness with no computational guarantee. OWL2 is an update to the OWL language and corresponds to the $\mathcal{SROIQ}^{(\mathcal{D})}$ DL fragment [54]. Regardless which OWL sublanguage we refer to, OWL is built using Resource Description Framework (RDF) triples [57]. An RDF triple relates a subject to an object via a predicate. In this way, OWL can be represented as a graph, where the predicate is the edge, and the subject and object are the nodes.

2.1.3 Graph-theory

An ontology can also be formalized using graph-theory. In this way, the ontology is a tuple $\langle D, R \rangle$, where D is the domain, represented as a set of concepts and R is the set of relations on D [47]. The graphs are directed acyclic graphs so that the ontology can be a hierarchy [26]. Unlike DL, the simplistic representation of a graph allows for a quick and intuitive approach to designing the ontology. Additionally, it allows for existing notions from graph theory to be applied to the ontology to perform tasks such as modularization [92]. Since OWL can be represented as a graph thanks to the RDF triples, graph-theory concepts can also be applied to it.

An analogue to increasing the expressivity of a DL by choosing a more expressive fragment in graph-theory is introducing weighted edges. A weighted edge allows for the expressing of one relationship being more important than another by having a higher weight. For instance, in [137], weighted edges are used to assist the clustering process. Weights are calculated using a correlation between concepts, and so the clustering algorithm can then use the weights to determine clusters of similar concepts.

The formality of a graph-based ontology can be further increased by defining it as a lattice-structure using Formal Concept Analysis (FCA). A lattice structure allows for the use of lattice-theory when discussing ontological concepts [93]. FCA conceptualizes a domain where the concepts are described by properties, and the properties determine the hierarchy of concepts. The hierarchy is not fully designed by an ontology engineer, which is different from other graph-based ontologies. With a graph-based ontology, all concepts and relations must be introduced by the ontology engineer, nothing is automatically constructed. By using FCA, it is possible to formalize the focusing on the concepts which exhibit a specific property by utilizing

the theory of sublattices.

2.1.4 Domain Ontology

The final formalism we introduce is the domain ontology found within DIS [85]. The domain ontology is a structure that itself is formed of three substructures: a Boolean lattice, a set of rooted graphs, and a monoid of concepts. Each of these substructures are formally defined using a theory, such as the Boolean lattice formalized using lattice theory, the rooted graphs using graph theory, and the monoid using universal algebra. These three structures work in harmony to conceptualize a domain. The Boolean lattice is the central anchor which conceptualizes the domain and how concepts relate to one another via a *partOf* relation. The rooted graphs are concepts that are not related via the *partOf* relation but instead via some other relation. Each rooted graph must have a root that belongs to the Boolean lattice. Finally, the monoid of concepts, defined as $\mathcal{C} = (C, \oplus, e_c)$, simply represents all possible concepts, represented by the set C, that exist within the domain (i.e., the concepts of the Boolean lattice and the rooted graphs).

The expressivity of a domain ontology comes from the richness of the underlying formalisms. For instance, the Boolean lattice allows for expressing the relationship between concepts via an ordering. However, the Boolean lattice is restricted to only the *partOf* relation. To express more rich relationships between concepts, rooted graphs can be made. Finally, the Boolean lattice can be instead represented as a Boolean algebra, and so the operations associated with that, such as the complement, can be used for expressions. The domain ontology is further detailed in Chapter 3.

2.1.5 Discussion

On the spectrum of expressivity, graph-theory is the least expressive of the three and DL is the most expressive. A graph is able to conceptualize a set of concepts and only one relation. The introduction of additional relations can be introduced to increase expressivity, as well as weighted edges, but it is still limited to using binary relationships. On the other hand, DL allows for several relations as well as quantifiers and–depending on the fragment used–inverse relations, cardinality, nominals, and more. The increase in expressivity comes at the cost of computational complexity for reasoning tasks, and the increase in difficulty in creating the ontology. The difficulty of creating DL ontologies comes from the need to understand the fragment to be used to properly conceptualize the domain, as well as the need for understanding writing DL. OWL can be thought of as a representation which is in between DL and graph-theory for the ontology representation. The reason it is considered in between is because it can be represented as a graph, and although *OWL DL* corresponds to a DL fragment, it cannot go beyond that fragment. That is, you cannot extend *OWL DL* with new DL fragments. Additionally, there exist OWL sublanguages (such as *OWL Full*) which do not have any corresponding DL fragment.

An issue that is prevalent within the DL field is the 'Tower of Babel' problem [52]. As a result of the numerous DL fragments, it has become an issue where two ontologies may not be compatible although both are formalized in DL. This is because one ontology may be formalized with a highly expressive fragment of DL and contains axioms that cannot be written in a less expressive fragment. As an ontology formalized using graph-theory is only a set of concepts and a set of relations on those concepts, the 'Tower of Babel' problem is not as prevalent. However, it does still

exist for ontologies which further restrict the ontology, such as FCA which imposes the lattice structure.

Finally, the inclusion of data to the ontology is something often not discussed. For an OWL ontology, one can easily add data through individuals. Although this is an easy process, each piece of data must be individually added. However, for graph-theory or DL, this is not true. For DL, the data would exist as assertions in the A-Box if the A-Box is considered to be a part of the ontology. For a graph, the data must be included with the set of concepts or not at all.

The domain ontology in a DIS can be compared to each of these ontology representations because of the underlying Boolean lattice and Boolean algebra. The Boolean lattice is formed over the relation *part Of*, and since it is a lattice, it is comparable to FCA. Additionally, it can be compared to a graph if we take the set of concepts L (from the Boolean lattice) and the relation *part Of*. In other words, we interpret the lattice simply as a graph and ignore the order of the relation. Further, the rooted graphs are themselves graphs, and so are directly comparable to graph theory. However, the domain ontology can also be compared to DL because of the Boolean algebra that the Boolean lattice is related to. The Boolean algebra allows for axioms to be written using the operators such as the complement or join. The Boolean algebra is not able to directly express the most expressive fragments of DL, such as the existential operator. However, not only can axioms be made within the domain ontology that do allow for these expressions, but often, the expressivity is limited in DL to ensure decidability. Since we are able to utilize both the Boolean lattice structure and the Boolean algebra of a domain ontology, we are able to compare findings to other ontology formalisms. For example, results regarding the Boolean lattice are relatable

to graph-based ontologies, and results regarding the Boolean algebra are relatable to (or other formal) ontologies.

2.2 Ontology-based Systems

2.2.1 Ontology-based Data Access

OBDA is a technology that seeks to provide access to various types of data sources using semantic technology [130]. An OBDA instance is defined as the pair $(\mathcal{P}, \mathcal{D})$ where $\mathcal{P}$ is an OBDA specification, and $\mathcal{D}$ is a source database. An OBDA specification is defined as $\mathcal{P} = (\mathcal{O}, \mathcal{M}, \mathcal{S})$ where $\mathcal{O}$ is an ontology, $\mathcal{S}$ is a data source schema, and $\mathcal{M}$ is a mapping from $\mathcal{S}$ to $\mathcal{O}$. The $\mathcal{O}$ provides a conceptual view of the data, and allows for queries to be made using the language of the ontology. A goal of OBDA is that by allowing the user to make queries using concepts from the ontology, the queries can be made with a more natural and convenient vocabulary (when compared to data schemas).

In an OBDA system, the ontology is formalized using DL. The queries can be submitted as a SQL query using the language of the ontology. However, the queries are not submitted as SQL, they are instead re-written as a first-order logic query. This way, the data is queried using an FO query. However, re-writing, in the worst case, is of exponential complexity [130]. Therefore, it is not a trivial task to perform the queries on an OBDA system.

Another aspect of concern regarding OBDA is the manual writing of the mappings. Each mapping between the data and the ontology must be written manually, and updated upon each modification to the ontology or data schema [130]. This is often

remedied by version-controlling the ontology so that it is not susceptible to quick changes. Although fixing the issue of needing to constantly re-write mappings, it prevents the ontology from evolving. New concepts cannot be added or existing concepts cannot be modified unless the version is being updated.

2.2.2 DOGMA

DOGMA is an approach for engineering highly reusable and usable ontologies that can be used over several domains and applications [61]. In the DOGMA framework, there is an ontology base which captures the domain axiomatizations, and a set of application axiomatizatons by which different applications commit to the ontology base. The ontology base serves the purpose of conceptualizing the domain, and is formalized as a set of binary relations and a set of concepts. In [61], an example of a bibliography domain is provided. In this example, the ontology base is exemplified as a set of relationships and the concepts they relate. For example, the concept *Book* is related to the concept *Written Material* by the *Has-Type* relation. There are application axiomatizations which provide specific views of this ontology base for specific applications. In one application axiomatization, the concept *Book* is related to the concept *Product*. In another product axiomatization, the concept *Product* does not exist. Despite being different, both of these application axiomatizations have made commitments to the same bibliography ontology base. One or several applications can subscribe to the application axiomatization that fits their needs.

The ability to have multiple applications commit to a single ontology base by choosing the application axiomatization that best fits their needs results in a unique understanding of a context. In a DOGMA system, a context plays a scoping role at

the ontology base level. A term is within a context refers to a concept in the ontology base.

2.2.3 Domain Information System

The domain ontology introduced earlier as an ontology formalism is a part of the Domain Information System [85]. The DIS is composed of a domain ontology, a domain data view, and a mapping function. The domain data view, as introduced, is composed of three substructures: the Boolean lattice, the rooted graphs, and the monoid of concepts. The domain data view is a cylindric algebra which models a structured dataset. Finally, the mapping function is an operator which maps elements of the cylindric algebra to the Boolean lattice. The components are more formally introduced in Chapter 3.

DIS is a formal system that is composed of several different algebras and theories. This formality allows for an expressiveness that is not found in other ontology-based systems. Additionally, it ensures the system is able to handle evolution easily. Similar to OBDA, the separation of the ontological component from the data component ensures that evolution is isolated to only one component. For instance, an update to a concept would only exist within the domain ontology. However, since the mapping function maps specifically from the domain data view to the domain ontology, data evolution is much easier to handle. If the data schema were to be modified (e.g., the addition of a column in a database), the domain ontology can be appropriately updated with minimal work. A concept must be created in the domain ontology that conceptualizes the new column as well as the mapping function needed to relate the two. There is no need to modify SQL queries or a translation function.

2.2.4 Discussion

OBDA and DOGMA both share the assumption that the data component either already exists, or is built independently from the ontology component. This results in the ontology being made for the data component, or for the ontology being fit to the dataset. In both of these systems, the matching of the data schema to the ontology is a non-trivial and inefficient process. Since in a DIS the domain ontology is constructed from the data view, there is no need to fit the domain ontology to the data view. Additionally, mappings are straight forward as there is a 1:1 correspondance between the data schema attributes and the atoms of the domain ontology.

OBDA and DIS are two systems that are composed of three components: an ontology component, a data component, and a mapping component which relates the data to the ontology. However, OBDA uses DL to formalize the ontology, whereas DIS uses a Boolean lattice for the core of the domain ontology, a monoid for the set of all concepts, and rooted graphs for the remainder of the domain ontology. Also, in an OBDA system, the data component and the ontology component are built independently. In a DIS, the atoms of the Boolean lattice come from the attributes of the data component. This relationship also allows for changes resulting from data evolution; data can be changed or modified without influencing the domain ontology. In the case that the schema were to be changed, then the Boolean lattice would need to be recreated. Likewise, any changes in the rooted graphs would not influence the data component as they use concepts that are not part of the Boolean lattice (and thus, not directly related to the dataset).

2.3 Ontology Modularization Techniques

2.3.1 Ontology Modularization Motivations

In the field of ontology modularization, there are four primary motivations that motivate the modularization process: engineering, reasoning, knowledge hiding, and alignment [79]. In several papers, although the motivation(s) of the author(s) may not be explicitly stated as one of the listed four, it can be considered similar enough to one of the four such that it can be compared and related to other papers of that similar motivation. For instance, in the literature, there is similarity between the motivations of *matching* and *alignment*. In particular, although matching seeks to merge two ontologies and alignment seeks to equate concepts between two ontologies, they both aim to produce modules such that the concepts within can be compared to each other. So, in the survey, alignment and matching are considered to be the same motivation.

The papers that are motivated by *engineering* produce a set of modules so that the ontology better follows software engineering principles. Often, the modules increase the usability or maintainability of the ontology, and thus improves the life-cycle. For example, in [129], the ontology is modularized so that it is more easily used as an annotation source by reducing the ontology to a more manageable set of modules which can be manually annotated. In [37], Ghafourian et al. state that their goal is to reduce the size and complexity of ontologies by improving the reusability of ontologies. An improvement to the reusability of an ontology allows for the easy design of a new ontology using modules from previously made ones. Finally, in [116], Stuckenschmidt

and Klein address the need to partition large ontologies (dubbed monolithic ontologies) into entities of meaningful and self-contained modules. This process seeks to help individuals maintain the ontology by instead of reviewing thousands of concepts, they instead review individual modules.

The papers that are motivated by *reasoning* produce a module that is used for reasoning purposes. The reduction of the ontology to the module seeks to improve the efficiency of the reasoning process. The improvement to efficiency is typically achieved by using a smaller set of axioms in the module rather than the entire ontology, or by parallelizing the reasoning algorithm to operate over multiple modules. In [36], Gatens et al. proposes a technique that ultimately extracts a module that is conservatively extended by the ontology. This is similar to the techniques of Del Vescovo et al. in [23] and Grau et al. in [42]. They are all founded on the same idea of utilizing locality-based modules. These, by definition, preserve the knowledge of the ontology in the context of the module, and so can be used for reasoning purposes and not produce incorrect or contradictory results. However, they all operate on some fragment of DL. Modularization techniques that are motivated by reasoning can also be seen by Noy and Musen in [92] and Sen et al. in [111]. In both of these works, DL is not used to formalize the ontology, but rather, it is represented as a graph. Since it is not as formal as DL, the modularization techniques do not seek to produce as rigorously defined modules. Rather than being able to correctly answer *any* query, they seek to extract the concepts and relations necessary to correctly answer the input query.

The papers that are motivated by *knowledge hiding* produce a module that encapsulates a specific view of the ontology. The reason for having a smaller view of the ontology can be for a personalized experience for the user, hiding concepts to increase

the security and privacy, or for abstracting the ontology into a more digestable and user-friendly format. These techniques are typically user-driven, guided by either a query or seed of concepts and relations, to determine what the view is to be about. They are also light-weight in that they are produced when needed. In [128], Wang et al. develop a technique that forgets specific concepts and relations so that the resulting view is a reduced ontology. They also propose three different query-based forgetting operators. This is similar to Koopmann and Schmidt who use the interpolant to define a forgetting process in [73]. In [7], Aroua and Mourad develop a technique for an interactive system that allows for a personalized experience by modularizing the underlying ontology according to the user's selected preferences. The preferences are formed by what the user wants to specifically see from the ontology, with the addition of some additional concepts that the system recognizes as similar.

Finally, the papers that are motivated by *alignment* produce modules that can be compared to the modules, formed by a similar modularization technique, of another ontology. This comparison may be done to determine the similarity of the two ontologies, or determine if one module can be swapped for the other and preserve the conceptualization of the domain. This allows for individual modules to be compared to one another rather than entire ontologies. In [80], Li et al. reduce large scale ontologies to smaller modules, which can then be compared in a matching phase. In [109], Seddiqui and Aono use modularization to reduce two massive, conceptually similar ontologies into modules so that the two ontologies may be aligned. The technique reduces the ordinarily polynomial time process of alignment to logarithmic time.

It is important to point out to the reader that a single technique can strive for multiple motivations; they are not exclusionary. For instance, the technique proposed

by Noy and Musen in [92] can be thought of as one that strives for reasoning, as it seeks to produce a module that answers a specific query. However, the intent of the technique is to produce a customizable, personalizable view for the user so they are interacting with a smaller part of the ontology. Thus, it is also motivated by knowledge hiding. This is because there is often great overlap between the motivations. It is often the case that by modularizing an ontology so it can be compared to another, as one does for alignment, they also are trying to make more reusable and maintainable components, which aligns with engineering.

2.3.2 Ontology Modularization Approaches

In the field of ontology modularization, there are two primary approaches: graphical and logical [79]. A graphical modularization technique determines the concepts that will compose the module using the graph structure of the ontology [116]. The concepts are determined in ways such as traversing the relations of the graph [92], using semantic similarity [37], or clustering techniques [4]. A logical modularization technique determines the concepts that will compose the module using the knowledge and deducible facts of the ontology [43]. Given an input seed, such as an axiom, the module will contain everything necessary to preserve the knowledge of that seed in the module [22]. The survey mentions a third approach, referred to as a hybrid, but there are so few techniques adhering to it, with no uniformity amongst them that they are not further discussed.

Of the two considered approaches, graphical modularization approaches are the most prevalent [79]. From the survey of ontology modularization techniques, nearly two-thirds were graphical approaches. Examples of modularization techniques that

follow the graphical approach are [92] by Noy and Musen, [117] by Stuckenschmidt et al., and [37] by Ghafourian et al.. In [92], the modularization process is initiated by providing a concept. The relations of the graph are traversed, starting from the provided concept, up to a depth that was also provided. All concepts that are part of the traversal are included in the module. In [117], the graph is partitioned by transforming the graph into a dependency graph. Weights are assigned to the relations that can be used to partition the graph into a finite set of islands through an iterative partitioning process. The technique in [37] is similar to that of Stuckenschmidt et al. in [117] because it also partitions the graph, but uses a neighborhood random walk distance rather than a dependency graph. However, it still uses weights of edges to ensure that concepts related to each other with a high weight value are in the same partition. Although the three techniques presented use the graphical structure in different ways to extract a module, they all use the relations, either for traversal or for assigning a weight, to determine the concepts.

As the graphical modularization approach utilizes the ontological structure to acquire the concepts for the module, they are often devised for ontologies represented as a graph. For instance, [4, 14, 17, 27, 30, 32, 38, 39, 59, 80, 81, 90, 92, 96, 99, 105, 106, 109, 110, 111, 116, 117, 121, 136, 131, 133] are all techniques that operate on an ontology defined as a graph. There may be slight variances to the exact definitions used, such as whether the graph is directed or undirected, but all of the ontologies the techniques operate on are composed of a set of concepts, and a set of edges on those concepts. OWL ontologies are also often used for techniques of the graphical approach, such as [3, 7, 37, 62, 89, 108, 107, 120, 123, 129]. These techniques, similar to the techniques that use a traditional graph, use the hierarchical structure that is

formed using the underlying RDF triples.

A logical modularization approach differs from a graphical modularization approach by how the concepts that constitute the module are determined. Examples of modularization techniques that follow the logical approach are [23, 44, 70]. In [23], locality-based modules are used to present an approach that utilizes atoms, which are defined in the paper as "*a set of axioms such that, for any module, it either contains all axioms in the atom or none of them.*". The atoms are said to be induced over an ontology by locality-modularization. Locality-modularization is often referred to as a relaxed form of a conservative extension-based module [43]: whereas a conservative extension-based module requires minimality of the module, a locality-based module does not. In [70], the module is determined via interpolation. A module defined using interpolation is similar to a module defined using conservative extension because they both use the same theory regarding Σ-inseparability. However, whereas conservation extension and techniques derived from it seek to extend the signature of the module to that of the ontology, interpolation techniques seek to reduce (or forget) the ontology's signature to that of the module's. Finally, in [44], Grau et al. present a technique that partitions an ontology such that the partitions are related using connections called ϵ-connections. It seeks to partition the language of the ontology such that each partition (referred to as a sub-domain) is independent from each other. The three presented techniques each utilize the logical side of the ontology in that the objective of modularization is to create a module that preserves the knowledge of the ontology with respect to the language of the module. Each of the techniques use the language of the ontology to do this, whether by extending the language of the module, reducing the language of the ontology, or partitioning the language of the ontology.

As the logical modularization approach seeks to preserve knowledge, it requires a formal representation of the ontology. In [6, 16, 23, 24, 36, 44, 55, 66, 67, 70, 69, 73, 82, 103, 104, 118], we see several logical modularization techniques that all use some fragment of DL. DL is the prominent formal representation of an ontology and is the typical representation used for a logical modularization technique. The axioms and existing reasoning methods can be used to determine whether, for instance, the ontology conservatively extends the module. Similar to graphical modularization techniques, logical modularization techniques can utilize OWL, as shown with [22, 42, 43, 71, 72, 122, 128]. However, it utilizes the respective DL fragment of the OWL ontology, or it uses only the OWL sentences to produce the module rather than the hierarchical structure that is formed like graphical modularization techniques.

2.3.3 Ontology Module Properties

A modularization technique is guided by some property that it is either trying to satisfy or optimize when creating the module. Although many properties exist, several are so similar that they can be categorized together. In [79], these properties are grouped into four prominent categories: conservative extension, structural proximity, semantic similarity, and disjointedness. A modularization technique that seeks to satisfy *conservative extension* is, as the name implies, one that produces a module that is conservatively extended by the ontology, using the definition of conservative extension as defined in Section 3.1.3. Since it produces a module that is conservative extended, it is a binary property: either the module is conservatively extended by the ontology, or it is not. A modularization technique guided by *structural proximity* is one that aims to preserve the ontological structure within the module. This is achieved by

preserving specific relationships, such as nearness of neighbors, adjacency of neighbors, or other structural measures. Often times structural proximity is a property that is optimized (as the full ontology certainly cannot be preserved by the definition of a module), and utilizes measures reminiscent of software engineering principles, such as cohesion and coupling to measure it. A modularization technique that uses *semantic similarity* is one that aims to preserve the context or understanding of the ontology. Whereas a technique optimizing structural proximity uses the ontological structure with little heed to the specific relations or concepts that make it up, a technique optimizing semantic similarity focuses less on the physical structure and more on the relations and importance of concepts. The preservation is determined using an assigned importance of relations to prioritize them for inclusion to the module, determining central concepts based on number of connections or inclusion in specific relationships, and more. Finally, a modularization technique seeking to satisfy *disjointedness* aims to produce modules that are independent from one another. This can be achieved through partitioning techniques, specifically of the signature of the ontology. Similar to the property of conservative extension, disjointedness is a binary property that is either satisfied or not.

The graphical modularization approach, because of its heuristic approach, produces modules that exhibit structural proximity or semantic similarity properties. Techniques such as [3, 12, 27, 39, 59, 62, 90, 92, 96, 105, 106, 108, 109, 116, 117, 120, 123, 136] all seek to optimize structural proximity in one way or another, and [4, 7, 14, 17, 32, 37, 38, 81, 80, 89, 94, 99, 107, 109, 110, 111, 121, 123, 129, 131, 134, 133, 135] all seek to optimize semantic similarity. There were no techniques

that seek to satisfy disjointedness or conservative extension that were also a graphical modularization approach. This aligns with the fact that those two properties require a formal ontology representation so that it may be proven that, for instance, the module is conservatively extended by the ontology. Graphs, which are what the graphical modularization approaches operate on, are not as formal as an ontology written in description logic. Of the techniques that use structural proximity, the prevalent measures found were the f-measure (found in [27]) and the compactness factor (found in [62]). The f-measure uses two metrics called *precision* and *recall* to determine the quality of the module. The precision is the number of relationships found in the module that are also found in the ontology, and recall is the number of relationships in the ontology that are also in the module. The compactness factor is a repertoire of aggregated measurements, such as the interconnectedness of concepts, cohesion, lexical similarity, and euclidean distance. In [39] there is a blurring of the structural proximity and semantic similarity properties. In this work, Ghazvinian et al. use a similarity measurement to ultimately determine the structural proximity of classes. Despite being a mix of the two, it does not share any properties with that of a conservative extension-based technique, or a disjointedness-based one.

Unlike the intuitive-nature of the graphical modularization approach, the logical modularization approach is based on formal logic, and so produces modules that exhibit the properties of conservative extension and/or disjointedness. The techniques of [6, 16, 22, 24, 36, 42, 43, 44, 70, 69, 71, 73, 82, 104, 118, 122, 128] all produce modules that are conservatively extended by the ontology, but may vary in how the module is defined: a minimal module, a locality-based module, or interpolation. The techniques of [23, 55, 66, 72, 103] all produce modules that exhibit the disjointedness

property. The observation that all techniques that produce a module that is conservatively extended by the ontology are categorized as a logical approach is not surprising. The intent of a logical modularization approach is to preserve the knowledge of the ontology with respect to the module, which is realized through conservatively extending it by the ontology. The finding that all disjointedness-based modules are also a logical modularization approach results from the necessity of being able to reason knowledge to ensure a partition contains the necessary axioms to preserve that knowledge.

2.3.4 Component of Ontology Utilized

A modularization technique can modularize the ontology using whichever component of the ontology: the data (such as the A-Box in a DL ontology), the concepts (such as the T-Box in a DL ontology), or both. The super majority of techniques utilize only the concepts. They use the relations that relate the concepts and the concepts themselves to create the modules. This is true for graphical approaches that traverse the conceptual structure (i.e., the hierarchy or the graph) to determine the modules, as well as for logical approaches that utilize the axioms that define the concepts. A small minority however, also use data.

In [126], Wandelt and Möller propose a technique that uses the data in the A-Box to guide the modularization process. Partitions (referred to as islands) are created using queries, and the T-Box is then updated to continue the modularization process. The technique in [91] by Nikitina et al. is similar in that the A-Box is partitioned so that the T-Box can be revised to adhere to the A-Box partitions. Pujara et al. propose a technique for an ontology represented as a graph which utilizes both data and concepts in [99]. In this case, the graph nodes can be either concepts or data, so

the partitioning technique, which partitions the graph nodes, will partition on both the concepts and the data. The ontology alignment technique proposed by Ochieng and Kyanda in [94] calculates similarity using the data of the two concepts from the two ontologies so that they can be related to each other.

The absence of modularization techniques that utilize data is not surprising as the issues the amount of data introduces to computation complexity is often considered not worth it. Or, the data is simply not considered part of the ontology and thus not in scope of ontology modularization. However, if an ontology-based system is to be considered in this research, which includes a data view, then data cannot be entirely ignored from the modularization process.

2.3.5 Discussion

The motivations, module properties, and component of the ontology utilized for modularization are dependent on the approach that is used. A logical approach is more associated with reasoning motivations, produce modules that are conservatively extended by the ontology, and utilize only the concepts. In contrast, graphical approaches are typically associated with engineering or alignment motivations, produce modules that exhibit a certain structural proximity or semantic similarity property, and largely utilize the concepts (although some techniques are hybrid). For both approaches, there do not exist techniques that heavily use, or consider, the data for the modularization.

The dependency between the modularization approach and the motivations, module properties, and component of the ontology utilized for modularization is made

more significant because of the separation between graph-theory ontology representations and DL. An ontology formalized with graph-theory is limited to graphical modularization approaches, which is in turn limited to certain motivations, module properties, and component of the ontology utilized. This is also true for an ontology formalized with DL, which is limited to logical modularization techniques.

As it was introduced in Section 2.1.5 that OWL can be thought of as an intermediary between DL and graph-theory ontologies. This is because depending on the sublanguage used, OWL can be interpreted as the related DL fragment, or as a graph formed from the RDF triples. However, not all DL fragments correspond with an OWL sublanguage, and vice versa. For example, OWL Full does not correspond to any single DL fragment. Therefore, for an OWL ontology, it is not necessarily true that any modularization technique that follows a logical approach can be applied to it. Additional to this issue is that although an OWL ontology has both logical and graphical aspects to it, it is not often that a modularization technique utilizes both. For example, modularization techniques for an OWL ontology may utilize just the graph aspect, or just the underlying DL, but not both. In this way, modularizing an OWL ontology is no different than modularizing an ontology formalized as a graph, or with DL. Ideally, there would be a modularization technique that is a combination of both a logical and graphical approach so that it does not limit the motivation, module properties, or component of the ontology utilized.

The limitations of the ontology modularization field–specifically with the separation between graphical and logical modularization approaches–is used as a launch pad for the research of this thesis. The duality of the graphical structure and the algebra of the domain ontology in a DIS has been discussed, and it is the goal that this

duality allows for the bridging between the research between these two approaches of modularization. The bridging of these approaches allows for the light-weight and intuitive approaches associated with graphical approaches to be enhanced with the knowledge-preserving and computationally minimal properties associated with logical approaches.

Chapter 3

Background

In this chapter, we introduce the necessary concepts relevant for the proposed research. First, we introduce the mathematical background, which contains all the theories that are essential for understanding the modularization techniques introduced in this research. Second, we introduce the Domain Information System (DIS), which is the structure used to represent the data and the domain conceptualization.

3.1 Mathematical background

We first introduce lattice theory and Boolean algebra, which are essential for understanding the underlying algebraic structure of the domain ontology that is to be modularized. Second, partition lattices and Boolean subalgebras are introduced as they are the used when defining a module. Finally, we introduce the notion of conservative extension, which is used to formalize logical modularization techniques, and is adapted to the modularization techniques of this thesis.

3.1.1 Lattice Theory and Boolean Algebras

A *lattice* is an abstract structure that can be defined as either a relational or algebraic structure [21]. In our work, we use both the algebraic and the relational definitions of lattices, therefore we provide both of them and we present the connection between them.

Let $(L, \leq)$ be a partially ordered set. For an arbitrary subset $S \subseteq L$, an element $u \in L$ is an *upper bound* of S, if $s \leq u$ for each $s \in S$. Dually, an element $l \in L$ is a *lower bound* of S if $l \leq s$ for each $s \in S$. An upper bound (resp., lower bound) u is defined as a *least upper bound* (resp., a *greatest lower bound*) if $u \leq x$ for each upper bound $x \in S$ (resp., $x \leq l$ for each lower bound $x \in S$). A least upper bound is typically referred to as a *join*, and a greatest lower bound as a *meet*. If every two elements $a, b \in L$ have a join, then the partially ordered set is called a join-semilattice. Similarly, if every two elements $a, b \in L$ have a meet, then the partially ordered set is called a meet-semilattice. As a relational structure, a lattice is a partially ordered set that is both a join- and meet-semilattice.

A lattice can also be defined as the algebraic structure $(L, \oplus, \otimes)$, which consists of a set L and the two binary operators $\oplus$ and $\otimes$ that are commutative, associative, idempotent, and satisfy the absorption law (i.e., $a \oplus (a \otimes b) = a \otimes (a \oplus b) = a$, for $a, b, c \in L$).

The relational and algebraic structures can be connected by the equivalences $a \leq b \iff (a = a \otimes b) \iff (b = a \oplus b)$. The connection between the relational and algebraic definition of a lattice allows us to freely interchange the relational and algebraic aspects in discussion; some concepts are easier to express or explain in one structure over the other.

We also require the notion of a *sublattice*, which is simply defined as a nonempty subset M of a lattice L that satisfies $x \oplus y \in M$ and $x \otimes y \in M$ for all $x, y \in M$.

A *Boolean lattice* [112] is defined as a *complemented distributive* lattice. A complemented lattice is bounded. A bounded lattice contains two distinguished elements, referred to as the *top* $\{1\}$ and the *bottom* $\{0\}$. These elements are comparable to every other element of the Boolean lattice, and follow that for any element $a \in L$, $a \oplus 1 = 1$ and $a \otimes 0 = 0$. A Boolean lattice is also complemented which means that every element a has a *complement*. We have an unary operator $'$ that gives for every $a \in L$ its complement a' such that $a \oplus a' = 1$ and $a \otimes a' = 0$. A distributive lattice is one where the join and meet operators distribute over each other (i.e., a lattice L is distributive if for all $x, y, z \in L$, $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ and $x \oplus (y \otimes z) = (x \oplus y) \otimes (x \oplus z)$).

The algebraic structure for the Boolean lattice is defined as a 6-tuple consisting of a set B, the two binary operators $\otimes$ and $\oplus$ (the meet and join), a unary operator $'$ (the complement), and two elements 0 and 1 in B (the bottom and top). We write this algebraic structure as $\mathcal{B} = (B, \otimes, \oplus, ', 0, 1)$. The complement operator $'$ is defined as above for a complemented lattice. In a finite Boolean algebra, an *atom* is defined as an element $a \in B$ where for any $b \in B$, either $a \otimes b = a$ or $a \otimes b = 0$ [50]. In this work, we consider only finite Boolean algebras. The Boolean lattice and its corresponding Boolean algebra are thus generated from the power set of the atoms [53]. As a result, all Boolean algebras with the same number of atoms are isomorphic to each other.

Other results regarding Boolean lattices that are useful for this research are:

- Any Boolean lattice is isomorphic to a field of sets.
- A Boolean lattice is complete and atomic iff it is isomorphic to the power set

of $\mathcal{P}(E)$ for some set E.

Ideals and Filters

Two distinguished substructures of a Boolean algebra that are used in this research are the *ideal* and *filter*. For a Boolean algebra $\mathcal{B}$ with the carrier set B, $I \subseteq B$ is called an ideal in $\mathcal{B}$ if I is nonempty and if for all $i, j \in I$ and for all $b \in B$ we have $i \otimes b \in I$ and $i \oplus j \in I$. A filter is the dual of an ideal. For a Boolean algebra $\mathcal{B}$ with the carrier set B, $F \subseteq B$ is called a filter in $\mathcal{B}$ if F is nonempty and if for all $i, j \in F$ and for all $b \in B$ we have $i \oplus b \in F$ and $i \otimes j \in F$.

An ideal is called *proper* if $I \neq \{0\}$ or B. It is possible to generate an ideal (filter, respectively) using an element, referred to as the *principal* ideal (principal filter, respectively). Let $\mathcal{B}$ be a Boolean algebra with carrier set B, and $b \in B$. The *principal ideal* (*principal filter*, respectively) generated by b is defined as $L_{\downarrow b} = \{a \in B \mid a \leq b\}$ ($L_{\uparrow b} = \{a \in B \mid a \geq b\}$, respectively). An ideal is *maximal* (or a *prime* ideal) if $I \neq B$ and the only proper ideal containing I is B itself. The dual of a maximal ideal is the *ultrafilter*. It is also established that for any maximal ideal (ultrafilter, respectively) $I \subseteq B$ and any element $x \in B$, I contains exactly one of x and x'.

3.1.2 Partition Lattices and Boolean Subalgebras

To later present algebraic modularization techniques, we use partition lattices to identify the Boolean subalgebras of a Boolean algebra. A partition lattice is the structure that is created by imposing the refinement relation over all the partitions of a set. Let $\mathcal{P}(X)$ be the power set of X. A partition P is a non-empty subset of $\mathcal{P}(X)$ such that $\forall(A, B \mid A, B \in P : A \cap B = \emptyset)$ and $\bigcup_{A \in P} A = X$. Let A and B

be partitions of a set X. The refinement on partitions, denoted by $\leq_P$, is defined as follows:

$$A \leq_P B \iff \forall(s \mid s \in A : \exists(t \mid t \in B : s \subseteq t))^1 \tag{3.1.1}$$

Thus, the top of the partition lattice is the partition with a single element, $\{X\}$, and the bottom of the lattice is the partition formed by the singleton sets.

It is well known that for a finite Boolean algebra, all of its subalgebras can be related to one another as a lattice that is dually isomorphic (i.e., the lattice ordered using the inverse relation) to the corresponding finite partition lattice [15]. In other words, if we have a Boolean algebra $\mathcal{B}$ with carrier set B, then there is a bijection between the lattice of all Boolean subalgebras of $\mathcal{B}$ and the partition lattice of the set of atoms of B. The lattice of Boolean subalgebras is ordered using the inverse of the refinement relation, and so the top element is the *finest* partition from the partition lattice, which is B. The bottom element is the *coarsest* partition, which is the trivial Boolean subalgebra where $0 = 1$. For instance, let $S = \{a, b, c, d\}$ be a set of elements, then $A = \{\{a\}, \{b\}, \{c, d\}\}$ and $B = \{\{a\}, \{b, c, d\}\}$ are certainly two partitions of S, and that $A \leq_P B$ because every element of A is a subset of an element of B. In fact, if we take $C = \{\{a\}, \{b\}, \{c\}, \{d\}\}$ to be the partition formed by all singleton sets, then C is the most fine partition. Each of these partitions correspond to a Boolean algebra formed by using the sets within a partition as the atoms. Thus, as we described, C–the finest partition–corresponds to the original Boolean algebra where a, b, c, and d are the atoms.

[1] We adopt the uniform linear notation provided by Gries and Schneider in [46]. The general form of the notation is $\star(x \mid R : P)$ where $\star$ is the quantifier, x is the dummy or quantified variable, R is predicate representing the range, and P is an expression representing the body of the quantification.

3.1.3 Conservative Extension

Conservative extension is a notion from mathematical logic, and often used in fields such as proof theory. In [88], we find the (proof theoretic) definition of conservative extension that is given by Definition 3.1.1.

Definition 3.1.1. *Let L and L' be logics of the same type (i.e., same signature), and ϕ be a formula in L. We call L' a (proof theoretic) conservative extension of L provided that the following two properties hold:*

1. $L \models \phi \implies L' \models \phi$
2. *For every ϕ in the language of L,* $L' \models \phi \implies L \models \phi$

In addition to the definition provided, there exists a stronger notion of conservative extension called *model theoretic* conservative extension. It is presented as follows:

Definition 3.1.2. *Let T and T' be theories. We call T' a (model theoretic) conservative extension of T if every model of T can be expanded to a model of T'.*

In Chapter 1 of [83], both Definition 3.1.1 and 3.1.2 are used to discuss the properties of an ontological module. In this paper, the T-Box concept assertions are the formulas in referred to in Definition 3.1.1. It is common that the proof for showing that the ontology conservatively extends the module is done in two steps, as exemplified by [45]. The first step, referred to as *local correctness* corresponds to determining the first property: that every formula of the module is a formula of the ontology. The second part, referred to as *local completeness* corresponds to determining the second property: that every formula of the module written in the language of the ontology is a formula of the ontology.

3.2 Domain Information System

A Domain Information System (DIS) [84] consists of three components: A domain ontology, a domain data view, and a function that bridges between the two.

3.2.1 The Domain Ontology

The first component of a Domain Information System is the *Domain Ontology*. The domain ontology is the conceptualization of the domain as a set of concepts and relations. We refer to the knowledge that is captured by these concepts and relations as the *domain knowledge*.

The domain is conceptualized using three structures: a monoid over the set of all concepts, a Boolean lattice formed over the atomic concepts, and a set of rooted graphs with a root concept that belongs to the Boolean lattice. To formally present the domain ontology, we first introduce the monoid structure. Let C be the set of all concepts under consideration, which are all concepts that conceptualize the domain. We define the combination operator $\oplus$ defined on the set of concepts that is commutative and associative, and a distinguished concept, e_C, that represents the empty concept. The empty concept is neutral to the concept composition, that is, for any concept $c \in C$, $c \oplus e_C = c$. The structure $\mathcal{C} = (C, \oplus, e_C)$ is therefore a commutative idempotent monoid. The idempotency allows us to express that if one were to combine a concept with itself, it would return the same concept rather than a new concept (i.e., $c \oplus c = c$). Using the composition operator, we define the partOf relation as follows:

$$k_1 \sqsubseteq_C k_2 \iff k_1 \oplus k_2 = k_2 \tag{3.2.1}$$

We also distinguish a subset of concepts as *atoms*, denoted by *At(C)*, where an atom is defined as:

$$k_1 \text{ is atomic } \iff \forall(k_2 \mid k_2 \in C : (k_2 \sqsubseteq_C k_1) \implies (k_2 = k_1 \text{ or } k_2 = e_C)) \quad (3.2.2)$$

Equation 3.2.2 states that the only concept that is a part of an atom is the empty concept. With this definition, the atoms can be understood as the building blocks of the domain. They are unitary, non-decomposable concepts that can be combined into the other concepts of the domain. This method of combining the atoms is the conceptualization of a Cartesian perspective of the domain. We can then understand the composition operator as determining the concept that is formed from the atoms of the two concepts. This understanding demonstrates the significance of the idempotency property: if two concepts which have the same atoms are combined, the result is a concept that is also formed from the same atoms, and therefore, the same concept.

We now introduce the second structure, the Boolean lattice. The subset of atoms corresponding to the attributes of the data schema is denoted by $T \subseteq At(C)$. The free Boolean lattice generated over T is denoted by $\mathcal{L} = \left(L, \sqsubseteq_C\right)$, where $L = \mathcal{P}(T)$, the power set of T. An algebraic representation of $\mathcal{L}$ is described as $\mathcal{B} = (L, \otimes, \oplus, ', e_C, \top)$. The composition operator $\oplus$ represents the join operator, and its dual $\otimes$, the meet operator. The two operators distribute over each other. For a given concept k, its complement is a concept k' such that $k \oplus k' = \top_{\mathcal{L}}$ and $k \otimes k' = e_C$, where $\top_{\mathcal{L}}$ is the top of the lattice $\mathcal{L}$. Inductively, we define a concept in L as follows: a concept c is either the empty concept, an atomic concept as defined in Equation 3.2.2, or a combination of atoms.

Let $k, k_1, k_2 \in L$:

$$k_1 \oplus k_2 \stackrel{\text{def}}{=} \oplus(c \mid c \in At(L) : c \sqsubseteq_C k_1 \text{ or } c \sqsubseteq_C k_2)$$
$$k_1 \otimes k_2 \stackrel{\text{def}}{=} \oplus(c \mid c \in At(L) : c \sqsubseteq_C k_1 \text{ and } c \sqsubseteq_C k_2)$$
$$k' \stackrel{\text{def}}{=} \oplus(c \mid c \in At(L) : \neg(c \sqsubseteq_C k))$$

We can show that $k_1 \otimes k_2 = (k_1' \oplus k_2')'$, $k_1 \sqsubseteq_C k_2 \iff k_1 \otimes k_2 = k_1$, and $\otimes$ is associative and commutative (and idempotent), as the dual of $\oplus$.

We now introduce the third and final structure needed to formally present the domain ontology, the set of rooted graphs. Let $C_i \subseteq C, R_i \subseteq C_i \times C_i,$ and $t \in C_i$. A rooted graph at t, $G_t = \big(C_i, R_i, t\big)$, is a connected directed graph of concepts with t as a root. We call t the *root* of G_t, and define it as follows:

$$t \in C_i \text{ is } \textit{root} \text{ of } G_t \iff \forall(k \mid k \in C_i : k = t \text{ or } (k,t) \in {R_i}^+).$$

Definition 3.2.1 (Domain Ontology)**.** *Let $\mathcal{C} = \big(C, \oplus, e_{_C}\big)$ 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 $\mathcal{G} = \{G_t \mid G_t$ is a rooted graph at t and $t \in L\}$. A domain ontology is the mathematical structure $\mathcal{O} \stackrel{def}{=} \big(\mathcal{C}, \mathcal{L}, \mathcal{G}\big)$.*

3.2.2 The Domain Data View

The second component of a Domain Information System is the *Domain Data View* that corresponds with an ontology. The domain data view is the structure that is modeled by the data of the domain, such as log data or relational database data. A crucial aspect of the domain data view is that it has the open-world assumption: incomplete or empty data is interpreted as unknown (i.e., it can be anything) rather

than nothing (i.e., no data). This assumption allows us to interpret incomplete data into what it *could be* by allowing the *sort* that has an empty value to be anything, through a process called cylindrification. For instance, for a row in a data table, an empty value in a column can be interpreted as any possible value rather than null. Since cylindrification can only be performed using a column in the dataset, and the columns correspond to an atom of the Boolean lattice of the domain ontology, we say that the domain data view *corresponds with an ontology.*

The domain data view is formalized as a cylindric algebra. The cylindrification operator of the algebra allows us to formalize the process described above. We define the domain data view as follows

Definition 3.2.2 (Domain Data View associated with an ontology)**.** *Let* $\mathcal{O} = \big(\mathcal{C}, \mathcal{L}, \mathcal{G}\big)$ *be a domain ontology. A domain data view associated with* $\mathcal{O}$ *is a diagonal-free cylindric algebra* $\mathcal{A} = \big(A, +, \star, -, 0_A, 1_A, \{\mathbf{c}_\kappa\}_{\kappa \in L}\big)$*, where* L *is the carrier set of* $\mathcal{L}$*.*

For the properties of a cylindric algebra, we refer the reader to [119]; we do not need these properties as our work mainly focuses on and uses the domain ontology of DIS. The elements of A are understood as n-dimensional objects, such as tuples of varying size from 0 to n. For example, let P, Q, R be three sorts of a table, such that $p_1, p_2, ..., p_i \in P$, $q_1, q_2, ..., q_j \in Q$, and $r_1, r_2, ..., r_k \in R$. A tuple such as $\{(p_1, q_1, r_1)\}$ is an element of A, and is a tuple of size 3. The tuple $\{(p_2, q_2)\}$ is also an element of A despite not having an element of the R dimension. The Boolean operators of $\mathcal{A}$ (i.e., $+, \star, -$) create new elements in A [119]. Finally, applying the cylindrification operator on an element $a \in A$, on the i-th dimension, can be understood as a *extension* of a on i. For example, if we let $a = \{(p_1, q_1, r_1)\} \subseteq P \times Q \times R$ be an element of A, then $\mathbf{c}_R(a) = \{(p, q, r) \mid p = p_1 \wedge q = q_1 \wedge r \in R\}$.

The cylindrification on R takes the tuple and extends the R-dimension so that its value can be *anything* that belongs to that sort. It does not affect the P and Q dimensions, which is why the tuple still has bounded variables p_1 and q_1. Therefore, we can have $\{(p_1, q_1, r_1), (p_1, q_1, r_2), ..., (p_1, q_1, r_k)\}$. We are also able to cylindrify on a sort that does not exist in the tuple, but does exist in the data, i.e., we can extend a tuple. If a has instead been defined as $\{(p_1, q_1)\} \subseteq P \times Q$, we would still be able to cylindrify on R. This is the process of extending the tuple from dimension $P \times Q$ to $P \times Q \times R$, where we retain the values p_1 and q_1, but allow the value of r be any value from R. It should be observed that for both definitions used for a, a tuple of dimension $P \times Q \times R$ or a tuple of dimension $P \times Q$, the cylindrification on R will produce the same set of tuples.

On the elements $a \in A$, we define a *focusing*[2] operator that corresponds to the projection operator in Codd's relation algebra. We adopt the notation $a \triangleleft \kappa$ for indicating the focusing of the datum $a \in A$ on the index κ. If κ is a part of $\tau(a)$, to focus a on κ means to find the element $b \in A$ such that $\tau(b) = \kappa$ and b is the projection of a on the concepts given by κ. This can be expressed using the cylindrification operator, by ensuring that cylindrification of both a and b is invariant on κ. Thus, the data view provides us with a language to describe properties on data elements. The focusing operator is used to obtains parts of a datum. If $\kappa \sqsubseteq_C \tau(a)$, the operator is defined as $a \triangleleft \kappa \stackrel{\text{def}}{=} \{b \mid b \in A \wedge \tau(b) = \kappa \wedge \mathbf{c}_{(\tau(a)\setminus\kappa)}a = \mathbf{c}_{(\tau(a)\setminus\kappa)}b\}$. Otherwise, $a \triangleleft \kappa \stackrel{\text{def}}{=} 0_A$.

[2]The term *focusing* is borrowed from information algebra [65].

3.2.3 The Domain Information System

With both the domain ontology and data view introduced, we now can define the *Domain Information System*. Figure 3.1 illustrates an example DIS at a high level. The blue shaded component, which contains the datasets, is an illustration of the domain data view. The red shaded component is a representation of the domain ontology. It is here that we can see a Boolean lattice with the composing concepts denoted by red nodes, and several rooted graphs with concepts denoted by the yellow nodes. The third and final component to define a Domain Information System is the τ operator. The τ operator is a function that maps elements from the domain data view to the Boolean lattice in the domain ontology. In Figure 3.1, an instance of using the operator is shown by the dashed green line which shows that the data element a_{43} is mapped to the concept titled *Attr3*. We define a Domain Information System as follows.

Definition 3.2.3 (Domain Information System)**.** *A Domain Information System (DIS) is the structure $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$ such that:*

- *$\mathcal{O}$ is a Domain Ontology*
- *$\mathcal{A}$ is a Domain Data View associated with $\mathcal{O}$*
- *$\tau : A \to L$ is a function that maps between the Domain Data View and the Domain Ontology and satisfies the following properties for $a, b \in A$:*

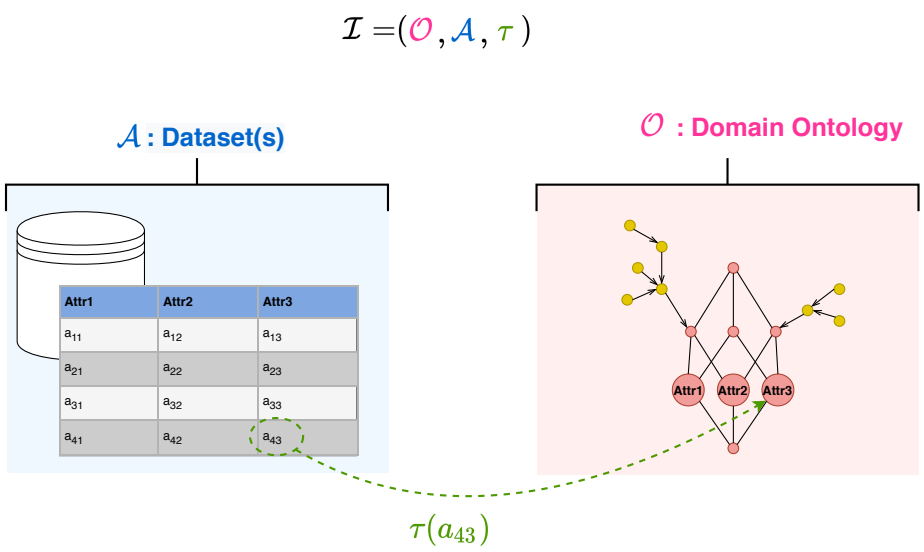


Figure 3.1: High-level representation of a Domain Information System

$$\tau(0_A) = e_C \tag{3.2.3}$$

$$\tau(1_A) = \top \tag{3.2.4}$$

$$\tau(-a) = \sim \tau(a) \tag{3.2.5}$$

$$\tau(a + b) = \tau(a) \oplus \tau(b) \tag{3.2.6}$$

$$\tau(a \star b) = \begin{cases} \tau(a) \otimes \tau(b), & \text{if } a \star b \neq 0_A \\ e_C, & \text{if } a \star b = 0_A \end{cases} \tag{3.2.7}$$

$$\tau(\mathbf{c}_\kappa a) = \begin{cases} \tau(a) \oplus \kappa, & \text{if } a \neq 0_A \\ e_C, & \text{if } a = 0_A \end{cases} \tag{3.2.8}$$

With this understanding of the τ operator, we see how the remaining concepts in the Boolean lattice are mapped from combinations of elements in A using the $+$ and $\star$ operators.

We are able to express the following relationship between the two orders of the

domain data view and Boolean lattice of a DIS:

Lemma 3.2.1. *Let* $a, b \in A$. *Then* $a \leq b \implies \tau(a) \sqsubseteq_C \tau(b)$

Proof.

$a \leq b$

$\iff$ ⟨ Definition of $\leq$ ⟩

$a + b = b$

$\implies$ ⟨ Apply τ to both sides of equation ⟩

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

$\iff$ ⟨ Property 3.2.6 ⟩

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

$\iff$ ⟨ Definition of $\sqsubseteq_C$ ⟩

$\tau(a) \sqsubseteq_C \tau(b)$ □

3.2.4 Motivating Example

The work presented in this thesis involves different approaches for modularizing a domain ontology in a DIS. In order to contextualize the modularization techniques, as well as facilitate discussion, a motivating example is introduced. The example that is introduced in this section will be carried through the remainder of this paper, and is the reference whenever mentioning 'the' domain ontology.

We remind the reader that a domain ontology is defined as $\mathcal{O} \stackrel{\text{def}}{=} (\mathcal{C}, \mathcal{L}, \mathcal{G})$ where $\mathcal{C}$ is a monoid which is composed of the set of concepts C, $\mathcal{L}$ is a Boolean lattice, and $\mathcal{G}$ is a set of rooted graphs. The atoms of the Boolean lattice relate to the schema of a dataset in $\mathcal{A}$. An example dataset is provided in Table 3.1.

Table 3.1: Park Property Dataset

Longitude	Latitude	Acres	Property Type	Realtor Neighborhood	City
-122.39982015	37.7955308	2.012	Civic Plaza	Financial Distric	San Francisco
-122.49239745	37.7219234	608.486	Regional Park	Lake Shore	San Francisco
-122.41742016	37.79642235	0.1489	Mini Park	Nob Hill	San Francisco
-122.45465821	37.73906281	40.7118	Regional Park	Miraloma Park	San Francisco
-122.40750049	37.78793123	2.5997	Civic Plaza	Downtown	San Francisco
-122.41204637	37.74706105	2.2460	Neighborhood Park	Bernal Heights	San Francisco
⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1 contains a reduced data set taken from the Open Data Sets for the City of San Francisco [1]. In this table there is data which is related to specific parks within the city of San Francisco. The data set has been reduced from 32 attributes to just 6 for both visualization and comprehension purposes. In the table, there are several attributes which are either redundant or not significant with respect to the conceptualization of the domain, such as an attribute for who last edited a park property. Ultimately, each attribute of the data set is composed with one another to produce the concept of a *Park Property*. Therefore, using this dataset, we freely construct the Boolean lattice of the Park Property domain using the attributes of the schema to create the set of atomic concepts. In Figure 3.2 we show the produced Boolean lattice, where each of the atoms corresponds to an attribute of the data set. For legibility purposes, we have shortened some of the attribute names, such as Longitude to simply Long.. The concepts within the Boolean lattice are those that are formed by combining two or more atoms. Concepts denoted by a non-filled circle node have a significance that is recognized by the domain expert. For instance, the concept formed by the combination of *Longitude* and *Latitude* is a non-filled node that is referred to as a *Location*. This concept is the conceptualization of how, with both a longitude and latitude, you have the location of a park. The remaining nodes do not have a recognized significance, but are still conceptualized. For instance, the

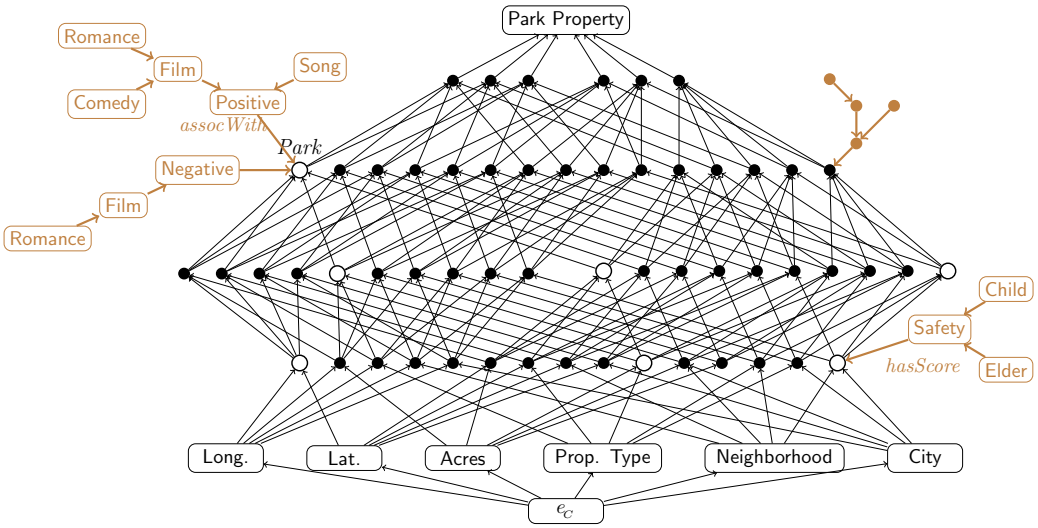


Figure 3.2: The Boolean lattice for the Park Properties

concept formed from *Neighborhood* $\oplus$ *City* may not carry a significance like *Location*, but it still requires a conceptualization. We also direct attention to the empty concept of the monoid, e_C, which is a part of the Boolean lattice. Specifically, it is the bottom element.

Rooted graphs are added to the domain ontology to further enrich the domain with concepts that cannot be defined as a combination of the atoms, as per the guidance of the domain experts. For instance, a park may have different sentimental interpretations based on what type of film or songs it is a part of. A park which is featured in someones favorite movie will be viewed in a positive light than if they had not seen that movie. Likewise, the park can be thought of negatively for similar reasons. Thus, the domain expert creates two rooted graphs, one for positive sentiment and one for negative, which is rooted to the concept *Park*, which is defined as $Lat. \oplus Long. \oplus Acres \oplus Prop.\,Type$. Additionally, a domain expert may recognize that

specific *City Neighborhoods* have specific safety scores, such as how safe it is for children or elders. These can also be represented as a rooted graph, rooted to the concept defined as $Neighborhood \oplus City$. In Figure 3.2, these rooted graphs are denoted by the yellow graphs.

It is important to note that each rooted graph is distinct from one another. In Figure 3.2, although there are two rooted graphs that each have a concept titled *Film*, they are not the same concept. This is due to how the family of rooted graphs is defined in a domain ontology. The result of this is that although the two rooted graphs share concepts with the same name, have the same root, and both are formed using the relation *assocWith*, they can be thought of as two entirely distinct and unique structures. Thus, the concept titled *Film* within the rooted graph of *Positive* is not the same as the concept of the same name within the rooted graph of *Negative*.

3.3 Conclusion

In this chapter, we introduced DIS, the underlying Boolean algebra of a domain ontology, as well as the mathematical theory of lattices, conservative extension, and Boolean subalgebras. This thesis utilizes each of these theories to develop and present modularization techniques for a domain ontology, and also to express the properties of these modularization techniques. We also introduced a motivating example, which is carried throughout the thesis, and is used as the example that each developed modularization technique is applied to. It provides a common starting domain ontology so that each modularization approach can be fairly compared to each another based on how it utilizes the domain ontology as well as by the module that is produced.

Chapter 4

Module and Modularization

In Chapter 2, Section 3.3, we indicated that in the literature, the notion of a module is vaguely introduced. The only characteristic of a module that is agreed upon is that it is a smaller component of the ontology. Since there are no precise definitions, it is not necessarily possible to compare two modules. Since they may be two entirely different structures, it may not be possible (or sound) to say which one is more suitable for a given task. The shortcomings with respect to defining a module cascade to not being able to sufficiently formally define the process of modularization. With no clear and formal definition of what a module is, we cannot reasonably define what the process of making a module is. This research proposes modularization techniques, so it is essential that we articulate a formal definition of what a module is, and by extension, what modularization is.

In this Chapter, we define a module with the characteristics provided by Doran et al. in [27] in mind. According to Doran et al., the module should be a reusable and self-contained component that bears resemblance to the ontology.

4.1 Definition of a Module

Let $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$ be a DIS with a domain ontology $\mathcal{O}$. A *module* is a sub-component of the domain ontology $\mathcal{O}$ such that itself is a domain ontology. We define it as follows:

Definition 4.1.1 (Module)**.** *Given a domain ontology $\mathcal{O} = (\mathcal{C}, \mathcal{L}, \mathcal{G})$, a module M of $\mathcal{O}$ is defined as as a domain ontology $M = (\mathcal{C}_M, \mathcal{L}_M, \mathcal{G}_M)$ satisfying the following conditions:*

- $C_M \subseteq C$
- $\mathcal{L}_M = (L_M, \sqsubseteq_c)$ *such that* $L_M \subseteq C_M$, $\mathcal{L}_M$ *is a Boolean sublattice of* $\mathcal{L}$, *and* $e_c \in L_M$
- $\mathcal{G}_M = \{G_n \mid G_n \in \mathcal{G} \text{ and } t_n \in L_M\}$

where C_M and C are the carrier sets of $\mathcal{C}_M$ and $\mathcal{C}$, respectively. □

The definition we present of an ontological module is compared to a mathematical module in Appendix A.

We present in Figure 4.1 a module of the domain ontology that is given in Figure 3.2. It can be seen that the set of concepts that exist in this module are certainly a subset of the set of concepts of the ontology; no new concepts have been introduced. Additionally, the rooted graphs that exist in the module also exist in the ontology, and the roots of the rooted graphs are unchanged. Finally, the Boolean lattice of the module is a Boolean sublattice of the ontology.

Since a module is defined using sets of concepts and rooted graphs, the question follows as to whether a new module can be determined using set operations on two

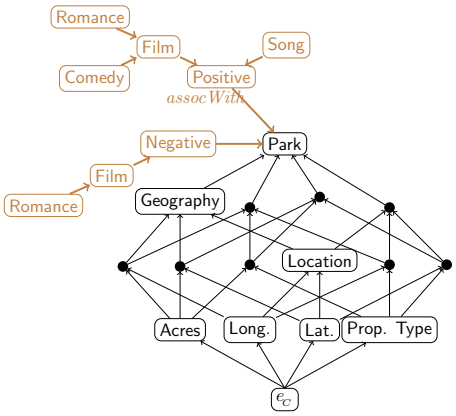


Figure 4.1: The Boolean lattice of the module extracted from the Park Ontology with $c = Park$

modules from the same domain ontology. For the following two claims, we let $\mathcal{O} = (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a domain ontology, and let $\mathcal{O}_1 = (\mathcal{C}_1, \mathcal{L}_1, \mathcal{G}_1)$ and $\mathcal{O}_2 = (\mathcal{C}_2, \mathcal{L}_2, \mathcal{G}_2)$ be two modules of $\mathcal{O}$. We focus on showing that the domain ontology formed via the set intersection or set union of $\mathcal{O}_1$ and $\mathcal{O}_2$ is a module. It is trivial to show that the intersection and union of the two module's set of concepts is a subset of the domain ontology's set of concepts, conforming to Definition 4.1.1. Likewise, the set of rooted graphs that is included to the module are those that have a root in the Boolean lattice. Thus, from the intersection or union of the two module's rooted graphs, we only include those that have root in the Boolean lattice to the module.

Claim 4.1.1. *Let $\mathcal{O}_3 = (\mathcal{C}_3, \mathcal{L}_3, \mathcal{G}_3)$ be the structure where $C_3 = C_1 \cup C_2$, $L_3 = L_1 \cup L_2$, and $\mathcal{G}_3 = \mathcal{G}_1 \cup \mathcal{G}_2$. Then $\mathcal{O}_3$ is a module if and only if $L_1 \subseteq L_2$.*

Proof. We prove this directly.

$L_1 \subseteq L_2$

$\iff$ $\langle$ *Definition of subset* $\rangle$

$L_1 \cup L_2 = L_2$

$\iff \langle L_3 = L_1 \cup L_2 \rangle$

$L_3 = L_2$

$\iff$

$\mathcal{L}_3$ *is exactly* $\mathcal{L}_2$. □

Claim 4.1.2. *Let* $\mathcal{O}_3 = (\mathcal{C}_3, \mathcal{L}_3, \mathcal{G}_3)$ *be the structure where* $C_3 = C_1 \cap C_2$, $L_3 = L_1 \cap L_2$, *and* $\mathcal{G}_3 = \mathcal{G}_1 \cap \mathcal{G}_2$. *Then* $\mathcal{O}_3$ *is a module.*

Proof.

$L_3 = L_1 \cap L_2$

$\iff \langle$ Definition of set intersection $\rangle$

$At(L_3) = At(L_1) \cap At(L_2)$

$\iff \langle$ Definition of concept composition $\rangle$

$\forall (k \mid k \in L_3 : (k = c_1 \oplus c_2 \text{ and } c_1 \in At(L_3) \text{ and } c_2 \in At(L_3)) \implies k \in L_3)$

$\iff \langle$ Definition of Boolean lattice and $\sqsubseteq_C \rangle$

$\mathcal{L}_3 = (L_3, \sqsubseteq_C)$ *form a Boolean lattice* □

In both of the above claims, we can form $\mathcal{O}_3 = (\mathcal{C}_3, \mathcal{L}_3, \mathcal{G}_3)$ as follows. $\mathcal{L}_3$ is formed as the above proofs show. $\mathcal{G}_3$ is the set of all rooted graphs from $\mathcal{G}_1 \cup \mathcal{G}_2$ ($\mathcal{G}_1 \cap \mathcal{G}_2$, respectively) that have root in L_3. $\mathcal{C}_3$ is formed by all concepts in L_3 and of each rooted graph of $\mathcal{G}_3$.

The two claims we have introduced both describe how modules can be determined from two existing modules of the same domain ontology. Although, with the set union operation, it is not possible to determine a *new* module.

To compare modules, we introduce the *module size* relation $\leq_S$ that is a subset of $\mathcal{O} \times \mathcal{O}$. Utilizing the previously introduced domain ontology $\mathcal{O}$ and two modules $\mathcal{O}_1$

and $\mathcal{O}_2$, we can compare the two modules via module size in the following way

$$\mathcal{O}_1 \leq_S \mathcal{O}_2 \iff C_1 \subseteq C_2 \text{ and } L_1 \subseteq L_2 \text{ and } \mathcal{G}_1 \subseteq \mathcal{G}_2.$$

This relation states that $\mathcal{O}_1$ is smaller than $\mathcal{O}_2$. If we let S_0 be the set of all modules of a domain ontology corresponding to a DIS, we can define a partial order using S_0 and $\leq_S$, described in the following Claim.

Claim 4.1.3. *Let $\mathcal{O}$ be a domain ontology, and let S_0 be the set of all modules of $\mathcal{O}$. The $\leq_S$ forms a partial order over S_0, denoted as $(S_0, \leq_S)$.*

Proof. To show that $(S_0, \leq_S)$ is a partial order, we must show that $\leq_S$ is reflexive, anti-symmetric, and transitive.

The $\leq_S$ relation is defined using the subset relation, which is itself reflexive, anti-symmetric, and transitive. Thus, the $\leq_S$ relation inherits each of these properties, and is itself an ordering. □

We also distinguish that this partial order is *bounded*. That is, there is an upper and lower bound. Within S_0, the original domain ontology $\mathcal{O}$ is the upper bound, and the module formed over $e_{\mathcal{C}}$ is the lower bound. This is because for any module $M \in S_0$, it is true that $M \leq_S S_0$. There is no module that is *larger* than the domain ontology it came from. This includes the domain ontology S_0 itself because of the transitivity of $\leq_S$. The lower bound, the module formed over $e_{\mathcal{C}}$, denoted as M_Z, is the smallest module that can be formed. That is, for any module $M \in S_0$, it is true that $M_Z \leq_S M$. This module is formed from stripping away all rooted graphs and Boolean lattice concepts except $e_{\mathcal{C}}$ and $\top$ (and setting them to be equal).

We remind the reader that according to Doran et al. [27], a module should be a self-contained, reusable component that bears resemblance to the ontology. Since

each module is itself a domain ontology, they are certainly self-contained and reusable. Additionally, a module can be related to the ontology it comes from using the $\leq_S$ relation. Although the $\leq_S$ relation describes a module that is no larger than the domain ontology it comes from, it does not necessitate that it is smaller. This is due to the reflexivity of the relation; the module may be of the same size as the domain ontology. Thus, to capture this ability to communicate a module that is strictly smaller than the ontology, we introduce the *strict module size* relation $<_S$, defined as

$$\mathcal{O}_1 <_S \mathcal{O}_2 \iff C_1 \subset C_2 \text{ and } L_1 \subset L_2 \text{ and } \mathcal{G}_1 \subset \mathcal{G}_2$$

This definition is the same as the *module size* relation definition where each instance of the subset relation has been replaced with a proper subset relation.

Claim 4.1.4. *Let $\mathcal{O}$ be a domain ontology, and let S_0 be the set of all modules of $\mathcal{O}$. The $<_S$ forms a strict partial order over S_0, denoted as $(S_0, <_S)$.*

Proof. The proof that $<_S$ is a strict partial order results from its definition. The irreflexivity, asymmetry, and transitivity are all inherited from the $\subset$ relation. □

With the *strict module size* relation, we are able to describe modules that are definitely smaller than the domain ontology they come from, a property often desired in a module.

4.1.1 Definition of Modularization

Since the module is itself a domain ontology, it is also correct to say the module is a sub-ontology for the original domain ontology. Using this, we introduce the notion of *modularization*. Intuitively, modularization is the process of determining a module

from a given domain ontology. As a module is itself a domain ontology, it must be the case that one can modularize a module. Therefore, we introduce $\mathbb{O}$ as a set of domain ontologies. We then declare modularization as the function

$$M : \mathbb{O} \to \mathbb{O}$$

such that $M(\mathcal{O}) = \mathcal{O}_M$ where $\mathcal{O}_M$ is a module of $\mathcal{O}$. Much of the research of this thesis is devoted to defining the modularization function in multiple ways. Each definition of the modularization technique must take in a domain ontology and produce a domain ontology (i.e., a module). The aim is that each modularization technique will be able to be compared to one another as they all take a domain ontology as input and produce a domain ontology as output, but they will be differentiated by their definitions and implementations.

Although the modularization definition provided operates on domain ontologies, to produce new domain ontologies (modules), we can define a DIS around the module. Since the module is formed by taking subsets of the domain ontology–the concepts, the Boolean lattice, and the rooted graphs–the resulting DIS would be formed by restricting the necessary components to only those that occur in the module. For example, the data components in $\mathcal{A}$ that are mapped to concepts in the domain ontology's Boolean lattice and that are *not* in the module's Boolean lattice are no longer necessary in the context of the module. Thus, the mapping function must be restricted as such. This is more formally defined as follows:

Definition 4.1.2 (Data Mapping Function)**.** *Given a DIS $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$, and let $\mathcal{O}_M$ be a module of $\mathcal{O}$. We denote the concept in L that corresponds to the top of the Boolean lattice of $\mathcal{O}_M$ as $\top_M$. We define a data mapping of A into a new dataset,*

A_M, as follows

$$f(a) = a \triangleleft (\tau(a) \otimes \top_M) \tag{4.1.1}$$

where $\tau(a)$ is a concept in L, $\triangleleft$ is the focusing operator on the data, and $\otimes$ is the Boolean lattice join operator.

With the data mapping function, we define the $\mathcal{A}_M$ of the module as follows.

Definition 4.1.3 (DIS Module Data View)**.** *Given a DIS $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$ and a module $\mathcal{O}_M$, we define the respective data view for $\mathcal{O}_M$ as*

$$\mathcal{A}_M = (A_M, +_M, \cdot_M, -_M, 0_M, 1_M, c_{k_M})_{k_M \in L_M}$$

where A_M is defined using the data mapping function, and each operator is the restriction to A_M of its corresponding function in $\mathcal{A}$.

From the definition of the DIS Module Data View and the data mapping function, we note the following:

- $0_M = 0 \triangleleft e_C$
- $1_M = 1 \triangleleft \top_M$

That is, $\tau(0_M) = e_C$ and $\tau(1_M) = \top_M$. With these, we define a DIS Module as follows.

Definition 4.1.4 (DIS Module)**.** *Given a DIS $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$, a DIS-module of $\mathcal{D}$ is the DIS $\mathcal{D}_M = (\mathcal{O}_M, \mathcal{A}_M, \tau_M)$ such that:*

1. *$\mathcal{O}_M = (\mathcal{C}_M, \mathcal{L}_M, \mathcal{G}_M)$ is a module of $\mathcal{O}$*

2. $\mathcal{A}_M = (A_M, +_M, \cdot_M, -_M, 0_M, 1_M, c_{k_M})_{k_M \in L_M}$ *is the module data view respective to* $\mathcal{O}_M$

3. $\tau_M \subseteq \tau \cap (A_M \times L_M)$ *is the restriction of the* τ *operator to only the data in the data view and the concepts of* L_M*.*

where L_M *is the carrier set of the Boolean lattice* $\mathcal{L}_M$*,* A *is the carrier set of* $\mathcal{A}$*, and* A_M *is defined via the data mapping function.*

Since we define a DIS-module from a domain ontology, we elaborate the module size relation to characterize and compare DIS-modules. We let S_0 be the set of all modules of a domain ontology $\mathcal{O}$, and let M_1 and M_2 be two modules that belong to S_0. In the case that $M_1 \leq_S M_2$, then $L_1 \subseteq L_2$. From this, the co-domain of the data mapping function for M_2 will include the co-domain of the data mapping function for M_1 with the additional concepts found in M_2 but not M_1. Thus, we see that the relationship between two domain ontology modules can be extended to their DIS modules as well, defined as follows:

$$\mathcal{D}_1 \leq_D \mathcal{D}_2 \iff \mathcal{O}_1 \leq_S \mathcal{O}_2$$

where $\mathcal{O}_1$ is the domain ontology that corresponds to $\mathcal{D}_1$ and $\mathcal{O}_2$ is the domain ontology that corresponds to $\mathcal{D}_2$.

Conclusion

With this definition of the $\leq_D$ relation, we see how when comparing the size of two domain ontology modules, we are also (perhaps inadvertently) comparing the size of the respective DIS modules. The opposite is also true: when we compare two DIS

modules, we are also comparing their respective domain ontology. The research in this thesis is focused on the development and comparison of modularization techniques for a domain ontology, and the scope of discussion is focused on the domain ontology itself rather than the large DIS level. However, as the above relation states, it should be implicitly understood that when we define a modularization technique and the resulting module that is produced, the results can be used to define and compare DIS modules as described above.

Chapter 5

Lattice-based Modules

We established in Chapter 2.3.2 that the two primary ontology modularization approaches are the graphical and the logical approaches. Graphical modularization approaches are more intuitive and simpler than logical approaches. They often have straightforward approaches which utilize the visual structure of the ontology. This results in modules that do not necessarily guarantee any properties with respect to knowledge preservation. Thus, they are also considered simple. However, their straightforward approach often means they are easy to understand and extremely intuitive. Because of this intuitiveness, it is the first type of modularization approach explored for application to a DIS domain ontology. Since the Boolean lattice is the core of the domain ontology, we refer to graphical modularization techniques as *lattice-based* approaches to reflect that the lattice is what is used for the technique. Appropriately, we refer to the modules produced via a lattice-based approach as a lattice-based module. In this Chapter, we develop and apply lattice-based approaches to a DIS domain ontology, define lattice-based modules, as well as discuss results regarding the knowledge transformation from the aforementioned modularization.

5.1 View Traversal

The first technique we develop for a DIS domain ontology is inspired by the view traversal approach [92]. As introduced, view traversal follows the graphical modularization approach paradigm, and is an extremely intuitive approach to modularizing an ontology.

The view traversal modularization technique introduced by Noy and Musen [92] operates by using a starting concept c to guide a travel directive. The travel directive is what determines which concepts will populate the module. It starts at c and any concept that is related to c is added to the module. From here, it continuously iterates through the concepts that have been added to the module, further adding the related concepts to the module, increasing the set of concepts in future iterations. The number of iterations is specified by the travel directive. Additionally, a travel directive can specify the relation that a concept must be related to c (or any concept in the module) by to be included in the module, and ignore other relations.

If we apply view traversal directly to a domain ontology, we do not necessarily produce a module in accordance with the definition of a domain ontology module found in Section 4. This is because a domain ontology module must include the empty concept, a restriction that is not a part of view traversal. To conform to the definition of a domain ontology module, we impose the requirement that for a view traversal on a starting concept c, the relation of the Boolean lattice must be 'traversed' such that the empty concept is included. Since the empty concept is the bottom element of the lattice, to traverse from c to it requires the travel directive to descend to the bottom of the lattice. If we gather all concepts that are traversed to 'between' c and e_C, we produce the principal ideal generated by c. Therefore, the act

of modularizing a domain ontology using view traversal with travel directive starting at c is equal to the act of determining the principal ideal generated by c.

The principal ideal generated by c, written as $L_{\downarrow c}$, is the Boolean sublattice such that it contains every concept that is *partOf* c. To be a domain ontology module, we must also define the set of rooted graphs that belong to it. However, this can be trivially accomplished by taking only the rooted graphs with a root that belongs to $L_{\downarrow c}$. The view traversal module is defined as follows:

Definition 5.1.1 (View Traversal Module)**.** *Let $\mathcal{O} = (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a domain ontology and let $c \in C$ be a concept that is referred to as a starting concept. A view traversal module of $\mathcal{O}$ starting from a concept c is defined as $\mathcal{M}_c = (\mathcal{C}_c, \mathcal{L}_c, \mathcal{G}_c)$ where the following is true:*

1. $\mathcal{L}_c = (L_{\downarrow c}, \sqsubseteq_C)$ *is a Boolean lattice*
2. $\mathcal{G}_c = \{G_i \mid G_i \in \mathcal{G} \text{ and } t_i \in L_c\}$
3. $C_c = \{k \mid k \in L_{\downarrow c} \vee \exists (G_i \mid G_i \in \mathcal{G}_c : k \in C_i)\}$.

A view traversal module certainly meets the criteria for a module as defined in Definition 4.1.1. A principal ideal is defined as a Boolean sublattice, and so it meets the criteria that the Boolean lattice of the module is a Boolean sublattice of the domain ontology. The only rooted graphs included in the view traversal module are those that have roots in the Boolean sublattice, and so they are a subset of the original set of rooted graphs. Lastly, the set of concepts for the view traversal module is the union of the concepts of the Boolean lattice and all rooted graphs. Since these two components are a part of the domain ontology itself, the union of these concepts must also be a subset of the domain ontology's concepts.

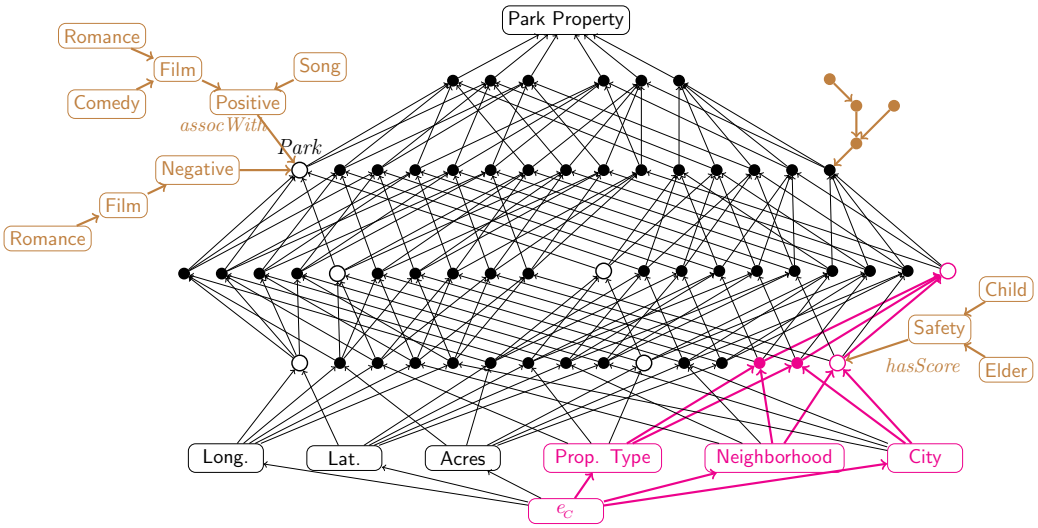


Figure 5.1: The Boolean lattice for the Park Ontology with the view traversal module highlighted for $c =$ *Neighborhood Feature*

We now reintroduce the San Francisco park example from Chapter 3, Section 2.4. Figure 5.1 shows the domain ontology with concepts that will populate the module with starting concept $c =$*Neighborhood Feature*, which is defined as *Prop. Type* $\oplus$ *Neighborhood* $\oplus$ *City*. Figure 5.2 shows the module after it has been extracted. In this module, any concept that is defined as the composition of the parts of *Neighborhood Feature* is included. For instance, *City Neighborhood* is included in the module because it is the composition of *Neighborhood* and *City*. Additionally, we include the empty concept e_C and the rooted graphs from the domain ontology with the root belonging to $L_{\downarrow c}$. In this example, there is the rooted graph that conceptualizes the safety score of city neighborhoods.

From the definition of the view traversal module, it is evident that the principal ideal generated by c is a critical component for a view traversal module. The rooted graphs are determined using the produced principal ideal. Thus, it is important to determine the complexity of the principal ideal's computation. The computation

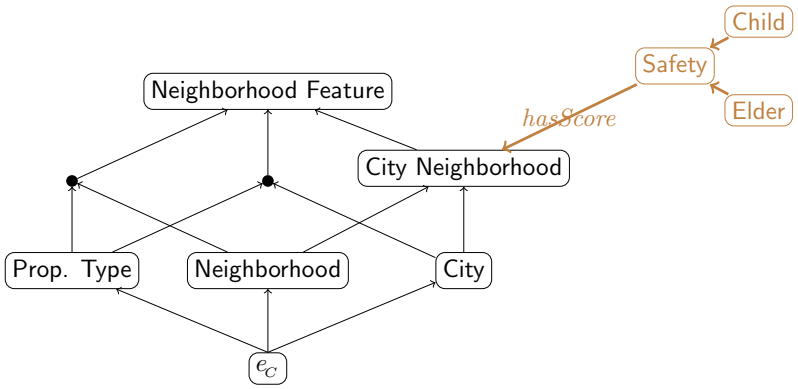


Figure 5.2: The view traversal module using $c =$ *Neighborhood Feature*

involves the determination of the elements in the Boolean lattice that are a part of the principal ideal generated by c. This is a straightforward process in which for all elements $x \in L$, we determine whether $x \oplus c \sqsubseteq_C c$. If so, x is a part of the ideal. Otherwise, it is not. Since $\sqsubseteq_C$ is transitive, we need to iterate through the set L only once. Hence, the complexity of an algorithm for the determination of the principal ideal generated by c is linear to the size of L.

Given what a view traversal module is, the modularization process to get such a module for a domain ontology in a DIS is defined as follows:

Definition 5.1.2 (View Traversal)**.** *Let $\mathcal{O} = (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a domain ontology. We also let $\mathbb{M}_D$ be the set of all possible view traversal modules obtainable from $\mathcal{O}$. View traversal v is a function that is declared as follows*

$$v : \mathbb{M}_D \times C \to \mathbb{M}_D \tag{5.1.1}$$

Where $v(\mathcal{O}, c) = (\mathcal{C}_c, \mathcal{L}_c, \mathcal{G}_c)$ is a view traversal module starting from the given concept c.

According to Definition 5.1.2, given a view traversal module and a starting concept, conducting view traversal will produce a new view traversal module. We note that the original domain ontology belongs to the set of all possible view traversal modules, i.e., $\mathcal{O} \in \mathbb{M}_D$. This is because the entire set L is equal to the trivial principal ideal generated by $\top$. We also distinguish two view traversal modules: the empty ontology module $\mathcal{O}_0$, and the atomic ontology module $\mathcal{O}_a$. The empty ontology module is defined as the view traversal module in which $C = \{e_{_C}\}$. Since it has only the single concept $e_{_C}$, the Boolean lattice is formed of that single element. This Boolean lattice has $2^0 = 1$ concepts, and the unique property that $e_{_C} = \top$. The atomic ontology module is defined as the view traversal module in which $L = \{e_{_C}, a\}$, where a is an atom of L. This Boolean lattice has $2^1 = 2$ concepts.

As the single element in $\mathcal{O}_0$ is $e_{_C}$, there are no rooted graphs in this module, and thus, $C = L$. We refer to this as $\mathcal{O}_0$ characterizing both C and L. This is not true for $\mathcal{O}_a$ since it can potentially have rooted graphs with root a (if any exist in the domain ontology). The concepts in these rooted graphs will populate C, and thus, it is not necessarily true that $C = L$. Therefore, we cannot say $\mathcal{O}_a$ characterizes both C and L. With these distinguished view traversal modules, we provide the following lemma.

Lemma 5.1.1. *Given a domain ontology $\mathcal{O}$ and a concept $a \in At(L)$, then the following are true:*

1. $v(\mathcal{O}, \top) = \mathcal{O}$
2. $v(\mathcal{O}, e_{_C}) = \mathcal{O}_0$, *the empty ontology module*
3. $v(\mathcal{O}, a) = \mathcal{O}_a$, *the atomic ontology module*
4. $\forall(c_1, c_2 \mid c_1, c_2 \in C : (c_1 \not\sqsubseteq_C c_2$ *and* $v(\mathcal{O}, c_2) = (\mathcal{C}_{c_2}, \mathcal{L}_{c_2}, \mathcal{G}_{c_2})) \implies c_1 \notin C_{c_2})$

This lemma describes how the two distinguished view traversal modules, $\mathcal{O}_0$ and $\mathcal{O}_a$, can be determined using the view traversal modularization technique. It also establishes that in the situation that we have two concepts c_1 and c_2, where c_1 is not a part of c_2, then c_1 will not be in the view traversal module determined using $c = c_2$.

The distinguished view traversal modules can also be extended to describe their respective DIS modules. $v(\mathcal{O}, \top)$ will correspond to the DIS module that is the original DIS. Since the domain ontology was not changed, the data will not be restricted, and thus, the DIS is not changed. $\mathcal{O}_0$ only contains the single concept $e_{\mathcal{C}}$ such that $e_{\mathcal{C}} = \top$. Since in this domain ontology module we have no concepts beyond the top and bottom, we have the corresponding data view that has only the 0 and 1. Finally, for the domain ontology module $\mathcal{O}_a$, we have that $a = \top$. When considering the data mapping in Definition 4.1.2, for any $\alpha \in A$, $f(\alpha) = \alpha \triangleleft (\tau(\alpha) \otimes a)$. Using the definition of an atomic concept in Definition 3.2.2, $\tau(\alpha) \otimes a$ must equal either a or $e_{\mathcal{C}}$. Thus, $f(\alpha)$ will only return the data that corresponds to the concept a.

5.1.1 Normalizing the Starting Concept

The principal ideal is generated using c, so it is essential that c is well-defined. The only restriction is that c must be a concept in C, which means that it does not necessarily have to belong to the Boolean lattice–it can be in a rooted graph. Additionally, to better correspond to the view traversal proposed by Noy and Musen in [92] –which allows the travel directive to be defined over a set of concepts– we add that there can be multiple starting concepts. Thus, we have the following three cases for c:

1. c is in the Boolean lattice

2. c is in a rooted graph

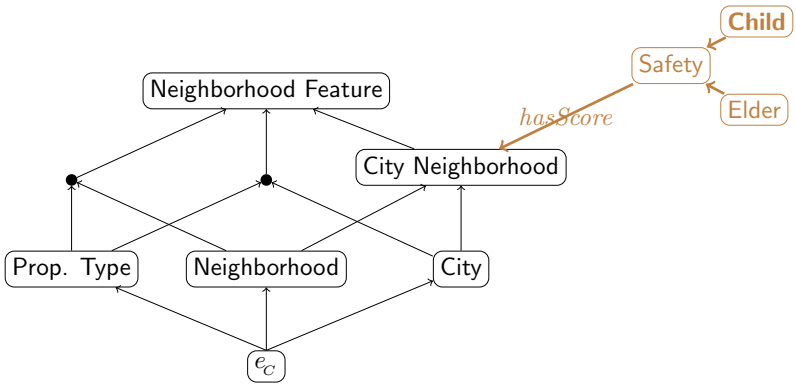


Figure 5.3: Normalizing the starting concept $c = \textit{Child}$

3. There are several starting concepts, $\{c_1, c_2, ..., c_n\}$

The first case is the business-as-usual case, i.e., the principal ideal is determined using c, and the view traversal module is formed using it. In the second case, the principal ideal cannot be determined using c as c is not in the Boolean lattice. For example, let $c = \textit{Child}$, a concept in a rooted graph such as the one shown in Figure 5.3. Since the determination of a principal ideal requires a concept in the Boolean lattice, we use the root of the rooted graph that *Child* belongs to as the starting concept. As the definition of the rooted graph requires the root to belong to the Boolean lattice, it is a safe assumption to use it for view traversal. We refer to this as the *projected concept* and is denoted as c_p. Since *Child* belongs to the rooted graph with root *City Neighborhood*, *City Neighborhood* is the projected concept. Thus, the view traversal with $c = \textit{Child}$ is found by determining the principal ideal generated by $c_p = \textit{City Neighborhood}$. Since the rooted graphs of any concepts in the principal ideal are included in the view traversal module, the rooted graph that *Child* belongs to will be included in the view traversal module created by c_p.

Finally, for the third scenario, since we have more than one starting concept, we

are unable to determine the principal ideal as it uses only a single starting concept. We let C_{V} be the set of concepts that are submitted for view traversal such that $C_{\mathrm{V}} \subseteq C$. Since it is a subset of C, it may be the case some concepts belong to a rooted graph. In this case, for any concept $c \notin L$, we first take the projected concept. We then define the starting concept ultimately used for the view traversal as

$$c = \oplus(c \mid c \in C_{\mathrm{V}} : c). \tag{5.1.2}$$

This states that, when given multiple starting concepts, we compute a single concept as the combination of all submitted concepts (or their projected concept). For example, referring to Figure 3.2, imagine that we use $C_{\mathrm{V}} = \{$*Neighborhood Feature*, *City Neighborhood*, *Child*$\}$. Since *Child* is in a rooted graph, we take the projected concept *City Neighborhood*. Then, we apply Equation 5.1.2 to $C_{\mathrm{V}} = \{$*Neighborhood Feature*, *City Neighborhood*, *City Neighborhood*$\}$. Since *City Neighborhood* $\sqsubseteq_{\mathcal{C}}$ *Neighborhood Feature*, then we can reduce C_{V} by keeping only the join of the two, *Neighborhood Feature*, i.e., $C_{\mathrm{V}} = \{$*Neighborhood Feature*, *Neighborhood Feature*$\}$. Finally, since $\oplus$ is idempotent, we can further reduce C_{V} so that the starting concept used for view traversal is *Neighborhood Feature*. The resulting module will be that in Figure 5.2, i.e., as if we had taken the view traversal with $c =$ *Neighborhood Feature*.

To illustrate why, when given multiple concepts for a view traversal, we use Equation 5.1.2 rather than taking the individual view traversal for each concept, we refer to Figure 5.4. Given two starting concepts, $c_1 = \{$*Long.* $\oplus$ *Acres*$\}$ and $c_2 = \{$*Acres* $\oplus$ *Prop. Type*$\}$, Equation 5.1.2 results in using $c = c_1 \oplus c_2$ for view traversal. The bold lines denote the relationships that would be in the view traversal

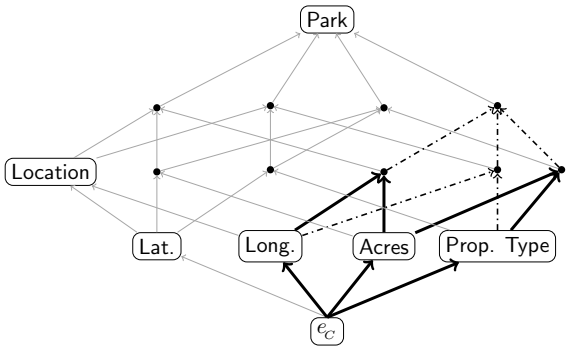


Figure 5.4: Modularization with $c_1 = \textit{Long.} \oplus \textit{Acres}$ and $c_2 = \textit{Acres} \oplus \textit{Prop. Type}$

modules had we taken the two individual view traversals using c_1 and c_2. Comparatively, the view traversal using c results in the module that contains the relationships denoted by the bold lines *and* the dashed lines–the principal ideal generated by c. By taking the view traversal on c, we ensure the two concepts c and $\textit{Long.} \oplus \textit{Prop. Type}$ are in the view traversal module; concepts that are not in either view traversal module found using only c_1 or only c_2. Thus, to ensure the view traversal module contains information about the way atoms can be combined from two different concepts, such as *Long.* and *Prop. Type*, we use Equation 5.1.2 and determine c.

When considering multiple starting concepts, there are two trivial cases: the first is for $c_i, c_j \in C_{\mathrm{V}}$ we have $c_i \sqsubseteq_C c_j$ for any $c_i, c_j \in C_{\mathrm{V}}$, and the second is the composition of concepts in C_{V} produce $c = \top$. The earlier example of when both *City Neighborhood* and *Neighborhood Feature* belonged to C_{V} is an example of the first case because $\textit{City Neighborhood} \sqsubseteq_C \textit{Neighborhood Feature}$. As it was shown, the result was simply *Neighborhood Feature* because $c_i \sqsubseteq_C c_j$ implies $c_i \oplus c_j = c_j$. Thus, in this scenario, c_i can be ignored. In the second case, the view traversal module will be the original domain ontology, i.e., $\mathcal{M}_c = \mathcal{O}$. Although it is counterintuitive to the purpose of modularization as the module will not be smaller than the ontology, this aligns with

the motivation that a view traversal module should contain the concepts necessary to answer queries regarding the starting concept(s). If $C_{\mathrm{V}} = \top$, then it implies that the desired module requires the knowledge of all of the domain ontology's atoms.

For the remainder of this paper, when referring to c, it is assumed that it has been normalized. That is, c is the projected concept if the submitted concept was outside of the Boolean lattice, or Equation 5.1.2 has been applied so that c is a single concept.

5.1.2 Properties of View Traversal

View Traversal Composition and Chaining

Since view traversal is a function, we are able to define the *composition* and *chaining* of view traversals as follows

Claim 5.1.1 (View Traversal Composition)**.** *Let $\mathcal{O} = (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a domain ontology. Let c_1, c_2 be two concepts in the carrier set of $\mathcal{L}$ such that $c_2 \sqsubseteq_C c_1$.*

$$v(v(\mathcal{O}, c_1), c_2) = v(\mathcal{O}, c_2)$$

Proof. A direct proof. Let L_1 be the carrier set of the Boolean lattice of $\mathcal{O}$

$v(\mathcal{O}, c_1)$

$\implies$ $\langle$ *Definition of View Traversal* $\rangle$

$\forall(c \mid L_1 : c \sqsubseteq_C c_1)$

$\implies$ $\langle c_2 \sqsubseteq_C c_1 \rangle$

$c_2 \in L_1$

$\implies$ $\langle$ *Defining the application of View Traversal* $v(v(\mathcal{O}, c_1), c_2) \rangle$

$\forall(c \mid L_1 : c \sqsubseteq_C c_2)$

$\Longrightarrow$ $\langle$ *Definition of View Traversal* $\rangle$

$v(\mathcal{O}, c_2)$ $\square$

Composing view traversals is, in essence, 'modularizing a modularization'. The above claim uses the assumption that $c_2 \sqsubseteq_C c_1$ for if c_2 is not a part of c_1, then the result of the view traversal composition is $\mathcal{O}_0$. If $v(\mathcal{O}, c_1)$ does not contain c_2; this is attempting to perform the modularization of the ontology on a concept that does not exist. Since we are modularizing on a concept that does not exist, it results in 'nothing', i.e., the empty ontology.

The combination of view traversal modules is the act of combining two modules into a single module, and is defined as follows. Let $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$ be a DIS. Let $*$ be an operator with the signature

$$* : \mathbb{M}_D \times \mathbb{M}_D \to \mathbb{M}_D$$

and is defined as $v(\mathcal{O}, c_1) * v(\mathcal{O}, c_2) = v(\mathcal{O}, c_1 \oplus c_2)$.

The view traversal combination function inherits the commutativity and associativity from $\oplus$.

Claim 5.1.2. *Let $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \tau)$ be a DIS. Let c_1 and c_2 be any two concepts from L such that $c_1 \sqsubseteq_C c_2$. Then*

$$v(\mathcal{O}, c_1) * v(\mathcal{O}, c_2) = v(\mathcal{O}, c_2)$$

Proof. We prove this directly.

$v(\mathcal{O}, c_1) * v(\mathcal{O}, c_2)$

$\Longleftrightarrow$ $\langle$ *Definition of view traversal combination* $\rangle$

$v(\mathcal{O}, c_1 \oplus c_2)$

$\iff \langle c_1 \sqsubseteq_C c_2 \Rightarrow c_1 \oplus c_2 = c_2 \rangle$

$v(\mathcal{O}, c_2)$ □

Boolean Algebra of View Traversal Module

The Boolean lattice of a view traversal module is a Boolean sublattice of the domain ontology's Boolean lattice, as defined in Definition 5.1.1. This means that there is a Boolean algebra that corresponds to the Boolean lattice of the view traversal module. However, for any view traversal module formed using $c \neq \top$, although the Boolean lattice of the view traversal module is a Boolean sublattice of the domain ontology's Boolean lattice, the Boolean algebra of the view traversal module is *not* a Boolean subalgebra of the domain ontology's Boolean algebra. This is formally written in the following proposition.

Proposition 5.1.2. *Let $O = (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a domain ontology and $\mathcal{M}_c = (\mathcal{C}_c, \mathcal{L}_c, \mathcal{G}_c)$ be a view traversal module obtained from O formed such that $c \neq \top$. Then:*

1. *The Boolean algebra associated with the view traversal module's ontology does not preserve the complement operator of the Boolean algebra associated with the original ontology.*

2. *The Boolean algebra associated to $\mathcal{L}_c$ is not a Boolean subalgebra of that associated to $\mathcal{L}$.*

3. *The view traversal module is locally complete but not locally correct with respect to the original ontology.*

Proof. The proof of this proposition is shown through example. Let $\mathcal{M}_v$ be any view traversal module formed using $c \neq \top$. In the view traversal module, $e'_C = c$. This

implies that in the domain ontology, $c = \top$ so that $e'_C = \top$ is preserved. However, we stated that $c \neq \top$. Since the complement operator is not preserved, it is not a Boolean subalgebra of the domain ontology's Boolean algebra.

The local completeness of the view traversal module comes from the ideal being closed under the join and meet operations from the ontology. In combination with *all* concepts that are a part of c being in the set L, then any formula written with the join or meet operations using concepts from the view traversal module will be true in the ontology and the module. However, it is not locally correct due to the complement operator not being preserved. □

Although Proposition 5.1.2 states that the view traversal module does not preserve the complement operator from the domain ontology, that does not mean that the significance of the domain knowledge of the module is diminished. For example, as stated, the join and meet operators are preserved. The join and meet operators are the description of how concepts can be combined (the join) to create a new concept, or how certain atoms within a concept can be focused on (the meet). Within the view traversal module, the way you can combine or focus on concepts remains correct with respect to the domain ontology. Thus, any domain knowledge related to the combination (or focusing) of concepts that is learned within the module will remain true in the domain ontology. Since the complement of a concept is the description of determining the concept that is composed of all the atoms that are not a part of the concept being complemented, it makes sense that when the scope of the domain is reduced via modularization, the complement operator returns a concept within this restricted scope.

5.2 View Traversal Knowledge Loss

It was established that the view traversal modularization technique produces a module that does not preserve the complement operator. Due to this, domain knowledge pertaining the complement of a concept is lost or changed. For instance, for any $a, b \in L$ and $a \in L_M$ but $b \notin L_M$, if in the domain ontology $a' = b$ then we will lose the knowledge that $a' = b$ in the module. The concept a will have a complement in the module, but it will not be b, thus the domain knowledge has changed. Since view traversal is defined as a function, we are able to quantify the domain knowledge that is lost (or changed) using the kernel of the function.

The Boolean lattice of the view traversal module is determined as the principal ideal generated by c. The first theorem of isomorphism [124] states that the ideal (i.e., the module) is the kernel of a homomorphism from the domain ontology to some *other* Boolean algebra (i.e., some other module). We denote the homomorphism that maps one Boolean algebra to another as $f : \mathcal{B}_1 \rightarrow \mathcal{B}_2$. In Figure 5.5, we show the relationship between f, the view traversal module, and the kernel. In this figure, we depict the Boolean algebras of the homomorphism by their respective Boolean lattices, a representation more familiar when discussing ontologies and modules. In the leftmost circle, labelled B_1, is the original domain ontology that is being modularized. The rightmost Boolean lattice, in the circle labelled B_2, is the module mapped to by the function f. Finally, the third Boolean lattice at the bottom of the figure, labelled $ker(f)$, is the byproduct of the homomorphism and corresponds to the first theorem of isomorphism, as described. The kernel is populated by all elements from B_1 that are mapped to the identity element $e_{_C}$ of B_2 by f.

For example, we define an instance of f as the f_{p_0} for a given $p_0 \in B_1$ and for

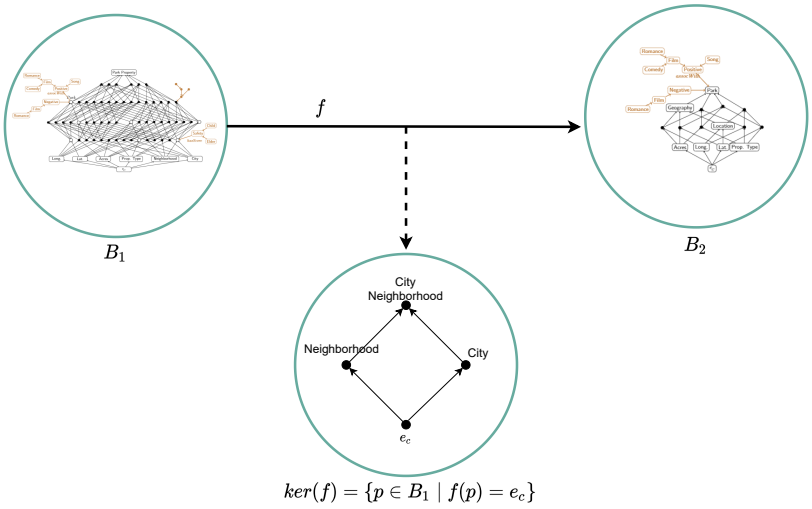


Figure 5.5: The relationship between the kernel and the homomorphism f

every $p \in B_1$:

$$f_{p_0}(p) = p \otimes p_0 \tag{5.2.1}$$

The kernel of f_{p_0} is defined as the set of all elements which map to the identity element by f_{p_0}:

$$ker(f_{p_0}) = \{p \in B_1 \mid f_{p_0}(p) = e_C\} \tag{5.2.2}$$

With this definition of f_{p_0}, we set $p_0 = Park$, where $Park = Lat \oplus Long \oplus Acres \oplus Prop\ Type$. We then use Equation 5.2.1 to determine the concepts that will be in the second carrier set of the second Boolean algebra. For example,

$$f_{p_0}(Lat) = Lat \otimes Park = Lat,$$
$$f_{p_0}(Park\ Property) = Park\ Property \otimes Park = Park$$
$$f_{p_0}(Lat \oplus City) = (Lat \oplus City) \otimes Park = Lat$$
$$f_{p_0}(City) = City \otimes Park = e_C$$

$$f_{p_0}(\mathit{City} \oplus \mathit{Neighborhood}) = (\mathit{City} \oplus \mathit{Neighborhood}) \otimes \mathit{Park} = e_{_C}$$

The function f_{p_0} maps any concept that is a part of *Park*–such as *Lat*–to itself, e.g., $f_{p_0}(\mathit{Lat}) = \mathit{Lat}$. Any concept with a part that is also a part of *Park* is mapped to the concept that is a part of *Park*, such as the example with $f_{p_0}(\mathit{Lat} \oplus \mathit{City})$. The remaining concepts that do not meet either of these two conditions are mapped to the empty concept, which populate the kernel of f. We recognize that these remaining concepts are those that do not share any parts with *Park*. Thus, with the starting concept $p_0 = \mathit{Park}$, the kernel of the homomorphism f_{p_0} is composed of the concept $p_0' = \mathit{City} \oplus \mathit{Neighborhood}$ and all of its parts:

$$\mathit{ker}(f_{p_0}) = \{c \in L \mid c \sqsubseteq_{_C} p_0'\} \tag{5.2.3}$$

Equations 5.2.1 and 5.2.2 together state that any concept $p \in B_1$ where $p \otimes p_0 = e_{_C}$ belongs to the set of $\mathit{ker}(f_{p_0})$. Using the definition of the concept complement and Boolean lattice meet, we get $p_0' \otimes p = e_{_C}$. Therefore, $p_0' \in \mathit{ker}(f_{p_0})$. For any $c \sqsubseteq_{_C} p_0'$, according to the definition of *partOf*, $c \sqsubseteq_{_C} p_0' \iff c \otimes p_0' = c$. With this definition of *partOf* and Equation 5.2.1, we get $f_{p_0}(c) = (c \otimes p_0') \otimes p_0$. Using the associativity of $\otimes$ operator, we get $f_{p_0}(c) = c \otimes (p_0' \otimes p_0) = c \otimes e_{_C} = e_{_C}$. Therefore, any element $c \sqsubseteq_{_C} p_0'$ also belongs to the kernel of f_{p_0}, which explains Equation 5.2.3.

The cardinality of the kernel is associated with the injectivity of the homomorphism. That is, the larger the kernel is, the less injective the homomorphism is. This is a result of the homomorphism mapping concepts from the first domain ontology, $\mathcal{B}_\infty$, to the second, $\mathcal{B}_\in$. If B_2 is much smaller in size than B_1, then more concepts from B_1 will be mapped to $e_{_C}$, and thus, populate the kernel. As an exercise, consider $p_0 = \top$, then $B_2 = B_1$ and $\mathit{ker}(f) = \{e_{_C}\}$. This, although trivial, states that

we are modularizing around the top concept, and so we should lose no knowledge–as we established when defining view traversal modules, we produce the original domain ontology when modularizing on the $\top$ concept. Since we know that every concept that is a part of p_0 maps to itself, if $p_0 = \top$, then we will have every element be mapped to itself. Thus, the kernel will only contain one element: the empty concept. With this, we make the connection between the domain knowledge lost due to modularization and the kernel of f. Since the kernel is populated by the concepts that are being lost due to the modularization, it is the embodiment of the domain knowledge that is lost. This is why the kernel is only the empty concept when modularizing on the top concept: we are losing no domain knowledge.

We now relate the results regarding the kernel to the view traversal module. Referring to Figure 5.5, the homomorphism f is equivalent to determining the view traversal on c. The kernel is used to generate the view traversal on c', shown in Equation 5.2.3, where $p_0 = c$. Therefore, it is possible to deterministically quantify the domain knowledge lost when extracting a module using view traversal by determining the view traversal on c'. Not only is the domain knowledge lost quantifiable as a set (the kernel), but it can be reasoned on or further modularized as it itself is a module.

5.3 Principal Ideal Subalgebra Module

Proposition 5.1.2 states that the Boolean algebra of the view traversal module is not a Boolean subalgebra of the domain ontology it is modularized from. We use the results of Harding in [51] to provide a means of transforming a view traversal module such that it preserves the complement operator, i.e., it is a Boolean subalgebra of the domain ontology.

The view traversal module is formed using the principal ideal generated by c. The concepts that are a complement of those in the principal ideal generated by c are absent. Since the principal filter generated by c' is the dual of the principal ideal generated by c, for every concept in the principal ideal, its complement will be in the principal filter. Thus, we union the principal ideal generated by c with the principal filter generated by c' to form a Boolean subalgebra.

We introduce the following notation so that a definition can be made. Let $\mathcal{L}$ and $\mathcal{L}_c$ be Boolean lattices such that $\mathcal{L}_c$ is a Boolean sublattice of $\mathcal{L}$. Let $\mathcal{B}$ be the Boolean algebra associated with $\mathcal{L}$, let $c \in L$ and let c' be the complement of c. Let $L_{\downarrow c}$ be the set of concepts in the principal ideal generated by c (i.e., the view traversal of c), and let $L_{\uparrow c'}$ be the set of concepts in the principal filter generated by c'. Let $L_{\updownarrow c} \stackrel{\text{def}}{=} L_{\downarrow c} \cup L_{\uparrow c'}$ be the carrier set for a Boolean algebra $\mathcal{B}_c$. We call the module that is formed using this set as the *principal ideal subalgebra module* and it is defined as follows.

Definition 5.3.1 (Principal Ideal Subalgebra Module)**.** *Let $\mathcal{O} \stackrel{def}{=} (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a given domain ontology. Let c be a concept in the carrier set L of $\mathcal{L}$. Let $L_{\updownarrow c} \stackrel{def}{=} L_{\downarrow c} \cup L_{\uparrow c'}$. The principal ideal subalgebra module $\mathcal{M}_c = (\mathcal{C}_c, \mathcal{L}_c, \mathcal{G}_c)$ is defined as:*

1. $\mathcal{G}_c = \{G_i \mid G_i \in \mathcal{G}$ *and* $G_i = (C_i, R_i, t_i)$ *and* $t_i \in L_{\updownarrow c}\}$
2. $C_c = \{c \mid c \in L_{\updownarrow c}$ *or* $\exists (G_i \mid G_i \in \mathcal{G}_c : c \in C_i)\}$ *is the carrier set for* $\mathcal{C}_c$
3. $\mathcal{L}_c = (L_{\updownarrow c}, \sqsubseteq_V)$, *where* $\sqsubseteq_V$ *is the restriction relation of* $\sqsubseteq_C$ *to* $L_{\updownarrow c}$.

We present an example of the principal ideal subalgebra module in Figures 5.6 and 5.7. Similar to how the view traversal module's rooted graphs are determined, the principal ideal subalgebra module's rooted graphs are those that are in the domain

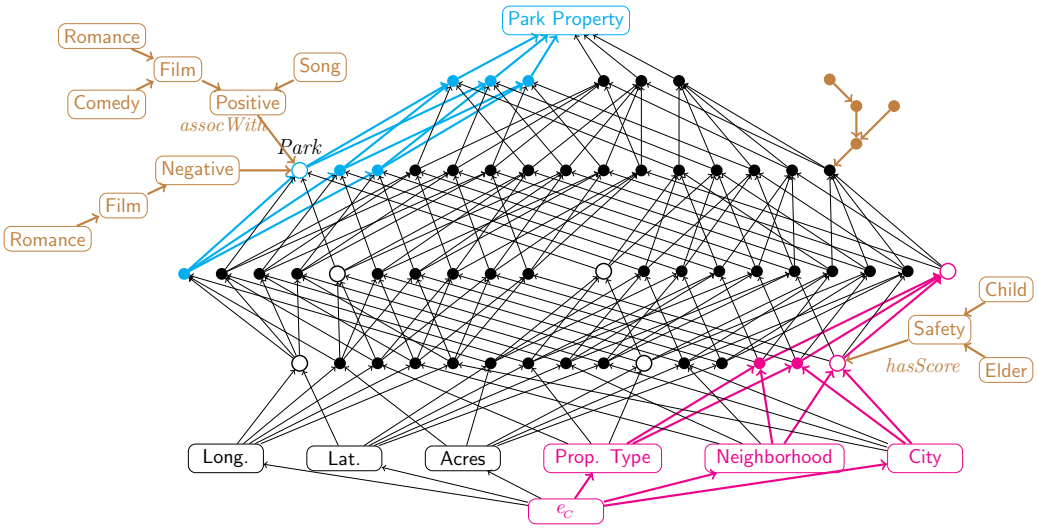


Figure 5.6: The Boolean lattice for the Park Ontology with the principal ideal subalgebra module highlighted for $c =$ *Neighborhood Feature*

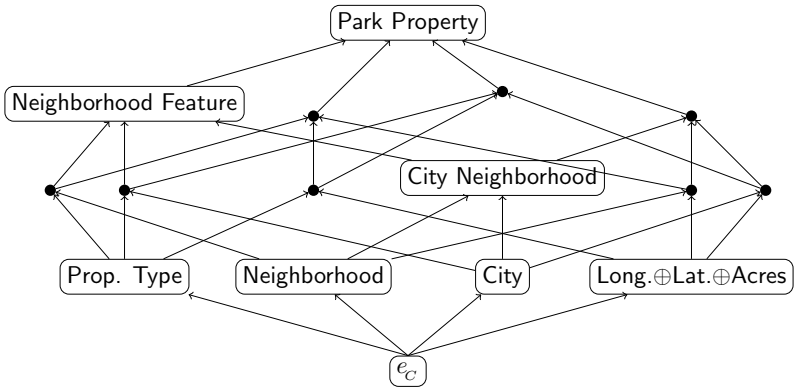


Figure 5.7: The Boolean lattice of the principal ideal subalgebra module extracted from the Park Ontology with $c =$ *Neighborhood Feature*

ontology with root belonging to the set $L_{\updownarrow c}$. In Figure 5.6 we show the domain ontology that is being modularized. The magenta highlighted concepts and relations are those that belong to the view traversal module with $c = \textit{Neighborhood Feature}$. The principal filter generated by c' are the concepts and relations highlighted by cyan. Since c is *Neighborhood Feature*–the concept composed of the atoms *Prop.Type*, *Neighborhood*, and *City*–the concept c' is the one composed of *Long*, *Lat*, and *Acres*. It is important to observe that the sets for the principal ideal and the principal filter are disjoint. In Figure 5.7, the Boolean lattice that is formed from $L_{\updownarrow c}$ is shown.

The set of atoms for the Boolean lattice in Figure 5.7 is formed from the atoms of the principal ideal with the addition of the bottom element of the principal filter, c'. This set of atoms is not a subset of the atoms of the ontology; $\textit{Long} \oplus \textit{Lat} \oplus \textit{Acres}$ is not an atom in the ontology but it is an atom in the module. Conceptually, the domain is modelled by the combinations of the atoms. They are the unitary, non-decomposable elements. To have an atom in one domain that is a composite concept in another (i.e., it is not an atom) implies that we are losing the domain knowledge associated with the individual parts. Afterall, in the module, we cannot decompose $\textit{Long} \oplus \textit{Lat} \oplus \textit{Acres}$ into its sub-components of *Long*, *Lat*, *Acres*, or any combination of two; the concept is now non-decomposable. This is referred to as the *coarsening* of the domain, and that the domain knowledge that can be derived from the module is *coarser* than the domain knowledge that can be derived from the original ontology. For all intents and purposes, in this module, the concepts of *Long*, *Lat*, or *Acres* simply do not exist.

Although the domain knowledge within the principal ideal subalgebra module is coarser, it preserves the complement operator from the ontology. Thus, we have the following claim.

Claim 5.3.1. *Let $\mathcal{M}_c$ be the module obtained by view traversal using concept c. Let c' be the complement of c. When we extend $\mathcal{M}_c$ with the principal filter generated by c', we produce a module that is both locally correct and locally complete.*

Proof. The above claim holds by construction. The principal ideal subalgebra module is formed using a Boolean subalgebra that is conservatively extended by the domain ontology's Boolean algebra. □

According to Claim 5.3.1, every satisfiable formula in the module is satisfiable in the domain ontology, and every satisfiable formula in the domain ontology written in the language of the module is satisfiable in the module. For example, noting that e_C is in the language of the module, the formula $e'_C = \mathit{Park\ Property}$ is true in both the module and domain ontology. The fact that this formula is true is what differentiates the principal ideal subalgebra module from the view traversal module. The formula $\mathit{Long} \sqsubseteq_C \mathit{Long} \oplus \mathit{Lat} \oplus \mathit{Acres}$ is true in the domain ontology but not in the module because the concept *Long* does not belong to the language of the module.

We refer to Figure 5.6 to relate the view traversal module and the corresponding principal ideal subalgebra module. The view traversal module would contain only the magenta concepts–the principal ideal generated by c. The principal ideal subalgebra module contains these concepts *in addition to* the concepts in the cyan Boolean sublattice. Therefore, we can *extend* a view traversal module to a principal ideal subalgebra module by extending the carrier set of the Boolean lattice of the view traversal module with the carrier set of the principal filter generated by c' (and the respective rooted graphs). Conversely, a principal ideal subalgebra module can be *truncated* to a view traversal module by removing any element belonging to the principal filter generated by c' (and the respective rooted graphs) from the carrier set

of the Boolean lattice.

A primary motivation for modularization is to acquire a smaller component that is more usable. Since a principal ideal subalgebra module extends a view traversal module, it will be greater in the number of concepts it contains than the respective view traversal module. However, this is the cost in ensuring all domain knowledge is retained when modularizing. Although the principal ideal subalgebra module will be larger than the respective view traversal module, we present the following claims which describe how we can easily determine if the requested principal ideal subalgebra module will be smaller than the domain ontology. If it is smaller than the domain ontology–even by a small amount–then it is still adhering to the motivation that a module is more usable.

Claim 5.3.2 (Maximal Principal Ideal Subalgebra Module)**.** *Let c be a concept in an ontology's Boolean lattice L. If the principal ideal generated by c is a maximal ideal, then the resulting principal ideal subalgebra module is equal to the original ontology.*

Proof. This follows from the definitions of a maximal ideal and ultrafilter. More specifically, for any $x \in L$, a maximal ideal I contains exactly one of x or x'. As the corresponding ultrafilter, F, is the dual of I, then for every $x \in I$, x' must belong to F. Therefore, every element from L is in either I or F. □

Claim 5.3.3. *Let c be a concept in the Boolean lattice L. If the principal ideal generated by c is not a maximal ideal, then the resulting principal ideal subalgebra is not isomorphic to the original Boolean algebra.*

Proof. We prove this by contradiction. We assume that the principal ideal generated by c is not maximal. We also assume that every element $x \in L$ belongs to either the

principal ideal I, or the corresponding filter F. This implies every atom a belongs to either I or F. For the principal ideal to be proper, it cannot include every atom in its set. Similarly, for the filter to be proper, it can at most contain one atom. Therefore, I must contain every atom save for one that must belong to F. An ideal that contains all atoms save one is a maximal ideal (and a filter generated using an atom is an ultrafilter), contradicting the assumption they are not maximal. □

As a result of the final claim, a principal ideal subalgebra module generated using a concept that is directly related to the top concept with the $\sqsubseteq_C$ relation (i.e., a concept that is the combination of all but one atom) is guaranteed to be smaller in size than the original domain ontology. Thus, we see a trade-off between the view traversal module and principal ideal subalgebra module: the view traversal module is smaller than the principal ideal subalgebra module, but it does not preserve the domain knowledge regarding the complement operator.

5.4 Summary and Conclusion

In this Chapter, we introduced and explored lattice-based modularization techniques for a domain ontology. These techniques are comparable to the graphical modularization techniques thanks to their utilization of the lattice relation to determine which concepts are added to the module. The techniques introduced in this Chapter produce the view traversal module and the principal ideal subalgebra module. The view traversal module is shown to be light and easy to compute, but does not preserve the complement operator, and thus, the knowledge within is not locally correct. The principal ideal subalgebra module remedies this at the cost of including more

concepts, and thus is larger.

The relationship between the module's Boolean lattice and the domain ontology's Boolean lattice, as well as their Boolean algebras, is explored as well. Results regarding which knowledge is lost and the ability to quantify it is made. Specifically, utilizing the concept of the kernel. These results are novel and to the best of our knowledge, the first related to quantifying how knowledge is lost and transformed due to the modularization process.

As mentioned, the techniques in this Chapter align with the graphical modularization techniques. However, we are able to relate the view traversal module, a module comparable to a graphical-based module, to the principal ideal subalgebra module, one more comparable to a logic-based module. This is a unique finding because it demonstrates how the domain ontology can be evaluated in a holistic way without compromise. The relationship between the Boolean lattice –the 'graph-based' component– and the Boolean algebra –the 'logic-based' component– is what enables this perspective. We can communicate meaningful properties related to both modules and modularization in both graphical and logical terms. Ultimately, with a domain ontology, and the DIS it is a part of, we can unify the divide between graphical and logical modularization approaches introduced in Chapter 2.3.2.

Chapter 6

Algebraic Modules

6.1 The Algebraic Module

In the previous chapter, a principal ideal subalgebra module was introduced as a module formed using a Boolean subalgebra of the domain ontology's Boolean algebra. The Boolean algebra of the principal ideal subalgebra module is formed by taking the union of a principal ideal formed over c and the principal filter formed over c'. However, in this Chapter we show that we can use other Boolean subalgebras to form a module. Thus, in the following we generalize a principal ideal subalgebra module to an algebraic module to account for these additional Boolean subalgebras.

Definition 6.1.1 (Algebraic Module)**.** *Let $\mathcal{O} \stackrel{def}{=} (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a given domain ontology. An algebraic module $\mathcal{M}_a = (\mathcal{C}_a, \mathcal{L}_a, \mathcal{G}_a)$ is defined as:*

1. $\mathcal{G}_a = \{G_i \mid G_i \in \mathcal{G}$ *and* $G_i = (C_i, R_i, t_i)$ *and* $t_i \in L_a\}$
2. $C_a = \{c \mid c \in L_a$ *or* $\exists (G_i \mid G_i \in \mathcal{G}_a : c \in C_i)\}$ *is the carrier set for* $\mathcal{C}_a$

3. *$\mathcal{L}_a = (L_a, \sqsubseteq_a)$, where $\sqsubseteq_a$ is the restriction relation of $\sqsubseteq_C$ to L_a and the Boolean algebra associated with $\mathcal{L}_a$ is a Boolean subalgebra of the Boolean algebra associated with $\mathcal{L}$.*

From Definition 5.3.1, it is clear that all principal ideal subalgebra modules are also algebraic modules. However, we will show that the inverse is not true: all algebraic modules are also principal ideal subalgebra modules. This implies, as all principal ideal subalgebra modules can be found by extending a view traversal module, there must be algebraic modules that cannot be found from extending a view traversal module.

To formally conclude this, we introduce a function f that transforms a view traversal module to a principal ideal subalgebra module. Let $\mathcal{O} \stackrel{\text{def}}{=} (\mathcal{C}, \mathcal{L}, \mathcal{G})$ be a domain ontology, with $\mathcal{L} \stackrel{\text{def}}{=} (L, \sqsubseteq_C)$. The lattice $\mathcal{L}$ corresponds to the Boolean algebra $\mathcal{B} \stackrel{\text{def}}{=} (L, \otimes, \oplus, e_C, \top, ')$.

Claim 6.1.1. *Let P be the set of all principal ideals of $\mathcal{B}$ and let B be the set of all Boolean subalgebras of $\mathcal{B}$. For every $p = (L_{\downarrow c}, \sqsubseteq_C) \in P$, we denote by $(L_{\uparrow c'}, \sqsubseteq_C)$ the principal filter generated by c'. Let $f : P \longrightarrow B$, where $f(p) = (L_{\downarrow c} \cup L_{\uparrow c'}, \otimes, \oplus, e_C, \top, ')$.*

The function f is total, injective, and non-surjective.

Proof. We prove this claim in three steps. First, the proof of the totality of the function follows from the existence of a principal filter for any principal filter in a Boolean algebra. For every element x in a principal ideal, its complement, x', belongs to a principal filter. The Boolean algebra formed using the respective principal ideal and principal filter is a Boolean subalgebra.

Second, the proof of the injectivity follows from the uniqueness of a principal ideal to the concept that generates it. The corresponding principal filter is also be unique

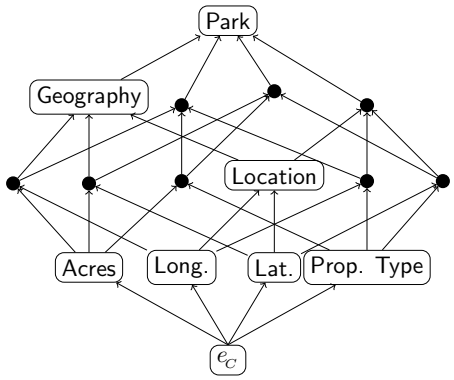


Figure 6.1: The Boolean lattice of the module extracted from the Park Ontology with $c = Park$

because it is the dual of the principal ideal. Therefore, the Boolean subalgebra formed from the principal ideal and principal filter is also unique.

Finally, the non-surjectivity is shown through an example. Let $\mathcal{B}$ be the Boolean algebra that corresponds to Figure 3.2 and let B be the Boolean algebra with the atoms $B_1 = \{Acres \oplus Lat, Long \oplus Prop\ \ Type\}$. B is certainly a Boolean subalgebra of $\mathcal{B}$, but it does not contain a principal ideal. Therefore, B_1 is a Boolean subalgebra that does not have an image by the inverse of f because no principal ideal can be extended to form B_1. □

Since we can utilize the isomorphism between the Boolean algebra and respective Boolean lattice, we call $f(p)$ the *view traversal extension* of p.

The non-surjectivity of f implies that there exist Boolean subalgebras that cannot be made by extending a principal ideal (i.e., a view traversal module). As algebraic modules are formed from Boolean subalgebras, this means that the set of principal ideal subalgebra modules is a proper subset of the set of algebraic modules. We utilize the lattice of all Boolean subalgebras of a finite Boolean algebra, introduced in Chapter 3.1, to characterize the remaining algebraic modules.

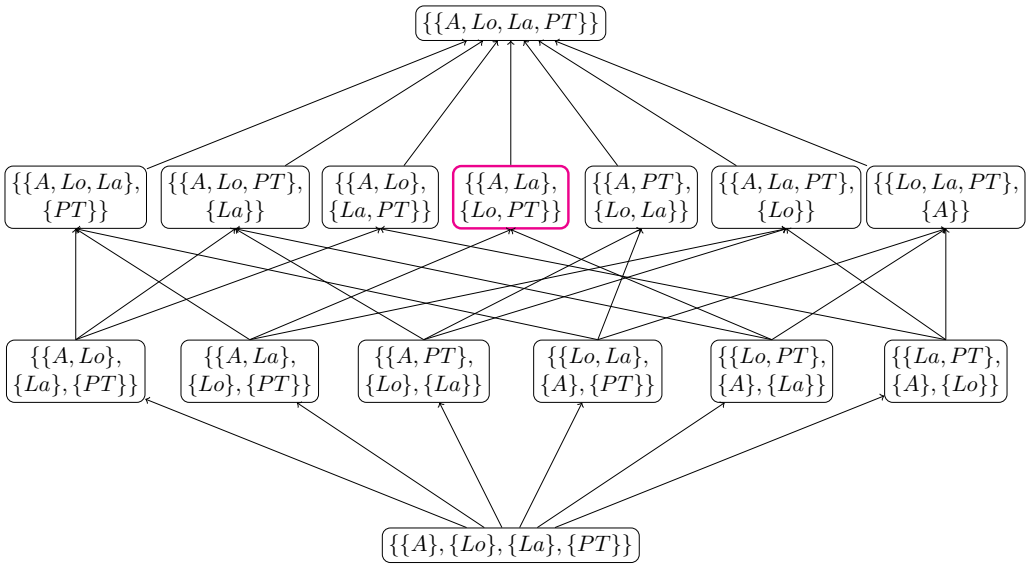


Figure 6.2: The partition lattice for *Park*

6.2 The Lattice of Algebraic Modules

6.2.1 Algebraic Module Refinement

The Boolean subalgebras of a domain ontology can be ordered as a lattice structure, which implies that there is an ordering relation over the set of Boolean subalgebras. We present three figures to illustrate the ordering of all Boolean subalgebras of a finite Boolean algebra. We present Figure 6.1 as the Boolean lattice we are investigating for all algebraic modules. The top concept is *Park* and is a combination of the four atoms *Acres*, *Long.*, *Lat.*, and *Prop. Type*. Figure 6.2 shows the partition lattice of the set formed by the four atoms. Each partition can be made into a Boolean lattice in the following way: let P be a partition of a set of concepts X, we derive the set of atoms A_P corresponding to P as:

$$A_P = \{B \mid B \in P : \oplus(x \mid x \in B : x)\} \tag{6.2.1}$$

For example, in Figure 6.2, the element highlighted in magenta corresponds to the partition $P = \{\{A, La\}, \{Lo, PT\}\}$. Applying Equation 6.2.1 to P, we determine $A_P = \{A \oplus La, Lo \oplus PT\}$.

The partition can be related via the partition refinement relation. We remind the reader that the partition refinement is defined as follows. Let $\mathcal{P}(X)$ be the power set of X. A partition P is a non-empty subset of $\mathcal{P}(X)$ such that $\forall(A, B \mid A, B \in P : A \cap B = \emptyset)$ and $\cup_{A \in P} = X$. Let A and B be partitions of a set X. The refinement on partitions, denoted by $\leq_P$, is defined as:

$$A \leq_p B \iff \forall(s \mid s \in A : \exists(t \mid t \in B : s \subseteq t)) \tag{6.2.2}$$

Thus, the top of the partition lattice is the partition with a single element, the set $\{X\}$, and the bottom of the lattice is the partition formed by the singleton sets.

In the same way that partitions can be related to one another in Figure 6.2 using the partition refinement relation, the Boolean lattices created using Equation 6.2.1 can also be related using a module refinement relation. This is thanks to the face that the lattice of Boolean subalgebras is order isomorphic to the dual of the partition lattice. Figure 6.3 is the lattice that is formed after transforming each partition into a respective Boolean lattice and relating the lattices. The magenta node in this lattice corresponds to the Boolean lattice formed from the partition P, the magenta node in Figure 6.2.

The partition lattice is ordered using the partition refinement relation. We define the algebraic module coarsening/refinement refinement relation used to order the lattice of Boolean subalgebras as follows:

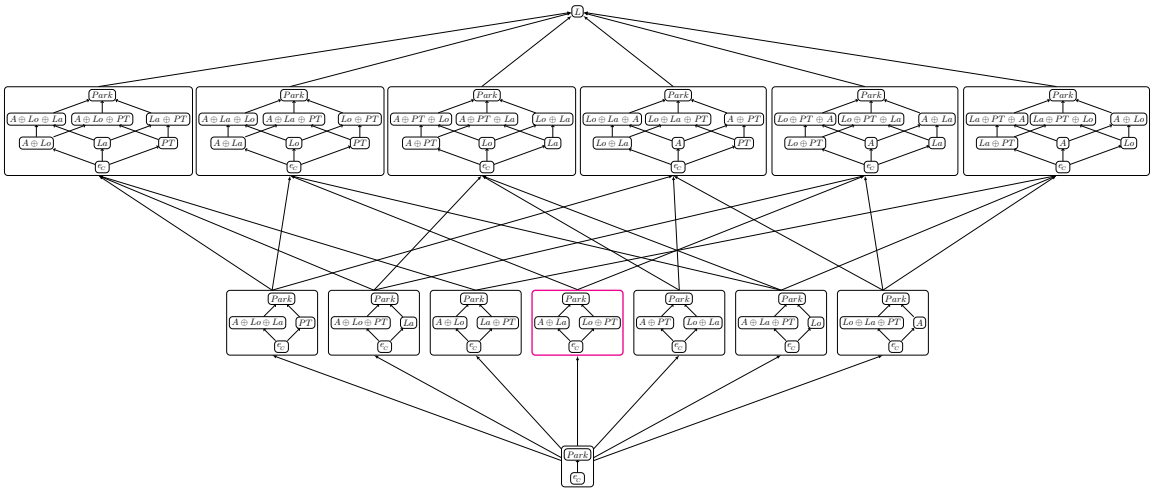


Figure 6.3: The lattice of Boolean subalgebras for $Park$

Definition 6.2.1 (Module Coarsening/Refinement)**.** *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two algebraic modules that are built from the partitions P_1 and P_2 of a set of concepts, respectively. We say that $\mathcal{M}_1$ is a coarsening of $\mathcal{M}_2$, or $\mathcal{M}_2$ is a refinement of $\mathcal{M}_1$, iff $P_2 \leq_p P_1$, and is denoted by $\mathcal{M}_1 \leq_M \mathcal{M}_2$*

The module in the magenta box of Figure 6.3, hereby referred to as At_1, is a refinement of the module at the bottom of the lattice which contains only $Park$ and e_C. The module highlighted in magenta is one of several ways to refine the bottom algebraic module. Every module in the same row as At_1 is a refinement of the bottom algebraic module. Additionally, At_1 can be *refined* into one of two modules: one with a Boolean lattice formed by the atoms $\{A \oplus La, Lo, PT\}$ (referred to as At_2) or one with a Boolean lattice formed by the atoms $\{A, La, Lo \oplus PT\}$ (referred to as At_3). This can be seen in Figure 6.3 as the two Boolean lattices that are related to At_1.

6.2.2 Algebraic Module Composition

As all of the Boolean subalgebras of a domain ontology are ordered using the algebraic module refinement relation, it is possible to utilize the operators of the resulting lattice to compute algebraic modules based on input parameters. Let $\mathcal{M}_A$ and $\mathcal{M}_B$ be two algebraic modules of the same domain ontology. The query of $\mathcal{M}_A \vee \mathcal{M}_B$, where $\vee$ is the join operation of the lattice of Boolean subalgebras, is the request for an algebraic module that is the refinement of *both* $\mathcal{M}_A$ and $\mathcal{M}_B$. Module refinement is defined using the underlying partition sets, and so the query is the act of determining the algebraic module formed from the partition set P such that $P \leq_p P_A$ and $P \leq_p P_B$, where P_A and P_B are the partition sets for $\mathcal{M}_A$ and $\mathcal{M}_B$ respectively. Therefore, we understand the process of module refinement as the process of forming an algebraic module from a partition set where every element in that partition set is a subset of an element in the partition set used to form the algebraic module that is being refined. The atoms of the refined algebraic module are formed by 'breaking up' one or more of the atoms of the algebraic module it refines. Therefore, the join operator is determining an algebraic module that is 'breaking up' element(s) from both P_A *and* P_B.

Whereas $\mathcal{M}_A \vee \mathcal{M}_B$ determines an algebraic module that is a refinement of both $\mathcal{M}_A$ and $\mathcal{M}_B$, $\mathcal{M}_A \wedge \mathcal{M}_B$ results in a module that is the coarsening of both $\mathcal{M}_A$ and $\mathcal{M}_B$. If the join operator is the process of determining an algebraic module that breaks up an atom in the two algebraic modules, then the meet operator is the process of determining an algebraic module that combines atoms in the two algebraic modules. This is the process of losing the parts of a concept and instead making it unitary, i.e., an atom. This is achieved by making a concept that may not be an atom

in the ontology into an atom in the algebraic module. The partition sets of $\mathcal{M}_A$ and $\mathcal{M}_B$, P_A and P_B respectively, are both a refinement of P.

As the refinement of an algebraic module is defined using the partition refinement relation, it follows that the more refined a module is, the larger it is in cardinality of the carrier set of the lattice. This is demonstrated in Figure 6.3 by the bottom of the lattice containing the Boolean subalgebra with only two concepts; the minimal Boolean subalgebra composed of only the constants. As you ascend the lattice of Boolean subalgebras, the number of atoms within the Boolean subalgebra at a given node of the lattice increases until you reach the top of the lattice that contains the original Boolean algebra. Thus, if one thinks of using the join and meet operations as a means to traverse the lattice, one can utilize the join and meet operations to determine larger (or smaller) algebraic modules. For instance, given two algebraic modules $\mathcal{M}_A$ and $\mathcal{M}_B$, the operation of $\mathcal{M}_A \vee \mathcal{M}_B$ returns the module that refines both $\mathcal{M}_A$ and $\mathcal{M}_B$: $\mathcal{M}_C$. As a more refined algebraic module was described as having more concepts, it follows that $\mathcal{M}_C$ is larger than both $\mathcal{M}_A$ and $\mathcal{M}_B$ so long as $\mathcal{M}_A \not\leq_M \mathcal{M}_B$ or $\mathcal{M}_B \not\leq_M \mathcal{M}_A$.

In the case that $\mathcal{M}_A \leq_M \mathcal{M}_B$, then $\mathcal{M}_A \vee \mathcal{M}_B = \mathcal{M}_B$ and $\mathcal{M}_A \wedge \mathcal{M}_B = \mathcal{M}_A$. This results in the algebraic module that satisfies the operation is not a third module $\mathcal{M}_C$, but is either $\mathcal{M}_A$ or $\mathcal{M}_B$. These relationships are communicated via the module refinement relation as follows:

$$\mathcal{M}_A \leq_M \mathcal{M}_B \iff \mathcal{M}_A \vee \mathcal{M}_B = \mathcal{M}_B \iff \mathcal{M}_A \wedge \mathcal{M}_B = \mathcal{M}_A \tag{6.2.3}$$

6.2.3 Knowledge Refinement of Algebraic Modules

In the example we had three Boolean subalgebras with atoms $\mathrm{At}_1 = \{A \oplus La, Lo \oplus PT\}$, $\mathrm{At}_2 = \{A \oplus La, Lo, PT\}$, and $\mathrm{At}_3 = \{A, La, Lo \oplus PT\}$. Both At_2 and At_3 are refinements of At_1, and so not only is $\mathrm{At}_1 \wedge \mathrm{At}_2 = \mathrm{At}_1$ and $\mathrm{At}_1 \wedge \mathrm{At}_3 = \mathrm{At}_1$ but also $\mathrm{At}_2 \wedge \mathrm{At}_3 = \mathrm{At}_1$. These relationships allow for the explicit communication that At_1 shares concepts with both At_2 *and* At_3. However, since At_1 is coarser than both, it is losing knowledge that At_2 has and knowledge that At_3 has. This loss of knowledge comes with the trade-off that At_1 corresponds with a smaller module than either At_2 or At_3.

The knowledge that is lost in the coarsening process can be found, if the correct algebraic modules are available. The join operation can be used to re-construct the knowledge that was lost during the coarsening process. For instance, the coarsening of At_2 into At_1 implies there must exist some Boolean subalgebra that when joined with At_1, results in At_2. This Boolean algebra, referred to as At_4, contains the knowledge that was lost in the coarsening of At_2. Referring to Figure 6.3, we see there are in fact two Boolean subalgebras that satisfy the requirement of At_4. The two Boolean subalgebras are the ones formed over the sets $\{A \oplus Lo \oplus La, PT\}$ and $\{A \oplus Lo \oplus PT, La\}$. If we let At_4 equal either of those two sets, then $\mathrm{At}_1 \vee \mathrm{At}_4 = \mathrm{At}_2$ is true. Thus, we see that although knowledge is lost in the modularization process – as demonstrated in the coarsening of a module – it can be reconstructed. Additionally, the reconstruction process demonstrates that the knowledge that is lost is embodied in other module(s). In the example, this is demonstrated by the existence of a At_4. The existence of more than one module that satisfied the requirements for At_4 implies that the knowledge needed to reconstruct At_2 from At_1 exists in more than one module.

6.3 The Search Space of Algebraic Modules

The ability to create new modules using the operators of the lattice becomes more important when considering the magnitude of how many modules can exist for a domain ontology. As a partition lattice is formed using the atoms of the Boolean lattice of the domain ontology, the number of partitions that exist for a given Boolean lattice generated from n atoms is given by the Bell number, defined recursively:

$$B_n = \begin{cases} 1, & \text{if } n = 0. \\ \sum_{k=0}^{n} \binom{n}{k} B_k, & \text{if } n \geq 1. \end{cases} \tag{6.3.1}$$

It should be noted that the Bell number grows at a rate with n that makes manual determination of modules infeasible, for instance, for $n = 8$, the Bell number B is equal to 4140. This means that there are 4140 partitions, and thus, 4140 algebraic modules. However, this is not a concern because in [125] it is shown that the majority of database tables have schema with less than ten attributes. Table 6.1 gives a table taken from [125] showing that a small percentage of investigated tables have schema with more than ten attributes. Additionally, it is envisioned that the user who is modularizing has a set of goals with respect to the module they seek. It would not be the case that they want to investigate *all* possible modules, but rather, modules of specific size or with specific concepts. These restrictions on the module further restrict the size of candidate modules. Therefore, with both the realistic size of database schemas and the restriction of candidate modules, the growth of the Bell number is not an expected limitation.

Table 6.1: Percent of Tables with Number of Attributes (borrowed from [125])

	> 5	$5 - 10$	< 10
atlas	10.23%	68.18%	21.59%
biosql	75.56%	24.44%	0.00%
coppermine	52.17%	30.43%	17.39%
ensembl	54.84%	38.06%	7.10%
mediawiki	61.97%	19.72%	18.31%
phpbb	40.00%	44.29%	15.71%
typo3	21.88%	31.25%	46.88%
opencart	57.20%	33.05%	9.75%
Average	46.73%	36.18%	17.09%

The other consideration is the calculation and determination of all of the partitions. However, the determination of all set partitions is a trivial task in terms of complexity, as shown in [95]. In fact, in practice, the computation is constant time.

6.4 Summary and Conclusion

In this Chapter, we introduced and explored algebraic-based modularization techniques for a domain ontology. These techniques are comparable to the logical modularization techniques thanks to their utilization of the Boolean algebra to determine which concepts are added to the module. The algebraic modules of this Chapter were inspired by determining that there must be other algebraic modules other than those of the principal ideal subalgebra modules. What we determined is that not only are there other modules, but that they can be related to one another via the partition lattice. The partition lattice is an indirect representation of all the algebraic modules for a given domain ontology, and how the modules are related to one another via a refinement relation. The refinement relation is used to introduce the notion of knowledge refinement. The atoms of an algebraic module does not necessarily need to be

a subset of the atoms of the domain ontology. In such a case, the algebraic module has a less refined (or coarser) set of knowledge than the domain ontology.

The total set of algebraic modules and the process of determining them aligns with the notion of logic-based modularization techniques. A primary focus of these techniques is preserving the axioms of the domain ontology in the module. However, the notion of knowledge refinement introduces a unique scenario where there is certainly knowledge of the atoms missing. This finding leads to an interesting notion of knowledge refinement rather than loss. The knowledge of the atoms is not absent in the same way that knowledge is lost due to view traversal, but rather, knowledge of the lost atoms is absorbed into a new atom. Findings such as this are unique to the modularization of a domain ontology.

Chapter 7

Future Work and Conclusion

7.1 Future Work

In this thesis, we propose two different types of modules that can be produced from modularizing a domain ontology: the lattice-based modules and the algebraic modules. These two types of modules correspond, respectively, to the graphical-based modules and logic-based modules found in the literature.

The theory of these modules is explored, but we did not further explore the automation of the respective modularization techniques. Time complexities of determining such modules, such as the principal ideal subalgebra module requiring linear time to compute, was established. However, the implementation and optimization of the modularization should be further explored to demonstrate both application and feasibility of these techniques. Algorithmic analysis should also be explored to ensure the implementation aligns with the hypothesized results. For example, it was stated that the number of partitions of a domain ontology's carrier set, and thus number of algebraic modules, can be determined using the Bell number. This number grows

exponentially with the size of the carrier set. However, we claim that the set of algebraic modules that one would need to "search" is greatly reduced based on desired properties, such as size or concept granularity. This claim should be explored and verified by automating the process of generating algebraic modules, and allowing for one to reduce the set of algebraic modules based on these example properties.

The modularization techniques explored in this paper are implementations of only two techniques found in literature: view traversal and conservative extension. These were manifested as the lattice-based and algebraic modules, respectively. However, the field of research for modularization techniques of an ontology is vast and diverse, and the exploration and implementation of other techniques could prove fruitful. For example, in the field of logical approaches, there are interpolation techniques in addition to the traditional conservative extension techniques. These interpolation techniques focus on hiding parts of the signature while ensuring that the module is still locally correct and locally complete. We hypothesize that although the implementation of other techniques may not result in new types of modules (i.e., a module different from lattice-based or algebraic) it will result in new motivations or ways of measuring or characterizing the existing modules.

Additionally, the research of this thesis was focused on the introduction and elaboration of the theory for modularization of a domain ontology. Future research should explore case-studies and applications for modularization. Several claims in this research point to potential areas of optimization or different use cases for modularization, that should be demonstrated via a case study. For instance, the difference between a view traversal module and a principal ideal subalgebra module is found in the size and preservation of the complement operator. We presented that a view

traversal module is a small, light-weight module that is determined via a starting concept. Similarly, a principal ideal subalgebra module is determined via a starting concept, but is not as small since it contains additional concepts so the complement operator is preserved. Future work that establishes scenarios where a view traversal module is appropriate versus a principal ideal subalgebra module (and vice versa) should be explored to better formulate use-cases, and in turn, engineering practices for modularizing a DIS.

Finally, this work introduces a formal approach to understanding and characterizing knowledge in a domain ontology. We introduce two different types of knowledge, and methods in preserving them. When discussing view traversal, we introduced the preservation or loss of domain knowledge via the kernel. With this, we explored the quantification of knowledge via the absence (or presence) of specific concepts. With the introduction of algebraic modules, we explored a more nuanced understanding of knowledge via knowledge refinement. With knowledge refinement, a specific concept may not be absent, but rather, indistinguishable from a composite concept. The topic of knowledge and the various ways in characterizing it is an extremely rich field of research, and deserves to be further explored. The underlying theories of DIS allows for the distinguishing of different types of knowledge, but with case-studies and a further exploration of modularization techniques, new types of knowledge can be characterized. Much like how the exploration of case-studies can establish use-cases and engineering practices for modularization of a DIS, it can also establish use-cases for which types of knowledge should (or can) be preserved. The modularization technique can be chosen based on the requirements regarding knowledge preservation.

This work is an exploratory body of research which established the theory for the

modularization of a domain ontology in a DIS. There are several significant results, such as the unification of graph-based and logic-based modularization approaches, that this research discusses. Ontology modularization is an extremely rich field of research with several different applications. This work establishes a solid foundation of theory which can be used to further explore applications and case-studies of modularizing a domain ontology. The theory can also be further explored to discuss innovative findings regarding topics such as engineering practices or knowledge.

7.2 Conclusion

Ontology modularization is a field of research which addresses the complications that result from ontologies growing to an unmanageable size, or to mitigate the effects of the evolution of the domain. The size of the ontology, whether it be measured by the number of concepts or relations, affects how practical tasks such as reasoning can be performed. This is magnified when only a small component of the ontology is needed for the reasoning. Additionally, as the domain evolves, it is often the case that existing concepts need to be modified, new concepts be added, or old concepts be removed. This can result in a need to recompile the entire ontology. DIS is a formalism that was introduced to address the needs related to incorporating data to an ontology. This research explores how the domain ontology component of a DIS can be modularized to address the listed problems. In particular, how a module can be extracted so a smaller component of the ontology is used for reasoning, rather than the entire domain ontology. This research focuses on establishing a formal foundation for modularizing a domain ontology by utilizing the Boolean lattice and Boolean algebra theories associated with the domain ontology.

The graphical approaches for modularization in literature were applied to the lattice structure of the domain ontology. Specifically, the view traversal modularization technique was applied. The unique duality of representing the domain ontology as a lattice and also as an algebra allowed us to go beyond measuring the module formed from view traversal using strictly graphical properties. For instance, we could measure the knowledge preservation of the view traversal module by determining if it was a Boolean subalgebra, and thus, preserving all operators. Since the view traversal module is not a Boolean subalgebra, it can be proven that knowledge related to the complement operator is lost. Specifically, the lost knowledge is quantified using the mathematical structure of the kernel. The principal ideal subalgebra module was introduced as an extension of the view traversal module which does preserve the complement operator, and thus, preserves the knowledge related to the operators since the associated Boolean algebra is a Boolean subalgebra of the domain ontology. Finally, we introduce the set of algebraic modules respective to a domain ontology, and how they parallel the logical approaches of modularization found in the literature. The result of how the Boolean subalgebras are related to the partition lattice allow us to compare the algebraic modules based on the refinement of knowledge–a way of formally understanding knowledge that is unique to the use of a domain ontology.

This research establishes a formal foundation of research for the modularization of a domain ontology. However, we discussed venues of future work for demonstrating the applicability of these techniques. For instance, the automation of the proposed techniques should be explored to demonstrate how one might modularize a domain ontology. Additionally, the exploration of case-studies will high-light different use-cases for modularizing a domain ontology. This in turn may exemplify the scenarios

in which one would use one technique, such as view traversal, over another, such as determining a principal ideal subalgbera module. Additionally, there are other modularization techniques or approaches that can be applied to a domain ontology, such as interpolation.

We conclude this research by saying that this research set out to explore the feasibility of modularizing a domain ontology in a DIS, and to unify the currently disparate fields of graphical and logical modularization approaches. The simplicity in communicating the results of modularizing using the underlying Boolean algebra demonstrates the usefulness of formalizing in DIS, and the algebra allows for the simple determination of the modules. We imagine with the future implementation that uses the theory of this research, one could easily and simply extract modules based on the knowledge they wish to preserve or refine, and to combine the modules to create a new domain ontology. This in turn would allow for several co-operating agents in a single domain, each with their own sets of knowledge. The field of ontology modularization is a rich field, and one that has many different use-cases. It is shown how utilizing DIS and its underlying Boolean algebra allows for a flexibility to approaches, whether it be a graphical or logical approach, where each have their own distinct usefulness.

Appendix A

Correspondance of an Ontological Module to a Mathematical Module

In this appendix, we compare the definition of a domain ontology module to that of a mathematical module. A mathematical module is the generalization of a vector space over a field, and is defined as follows.

Definition A.0.1 (Mathematical Module [74])**.** *Let R be a ring and 1_R is its multiplicative identity. A (left R-) module $\mathcal{M}$ consists of an abelian group $(M, +)$ and an operation $\cdot : R \times M \to M$ such that for all $r, s \in R$ and for all $x, y \in M$ we have:*

1. $r \cdot (x + y) = r \cdot x + r \cdot y$
2. $(r + s) \cdot x = r \cdot x + s \cdot x$
3. $(rs) \cdot x = r \cdot (s \cdot x)$
4. $1_R \cdot x = x$

A *right R-module* is defined similarly except that the ring acts on the right of the $\cdot$ operator. If R is commutative, then *left R-modules* are the same as *right R-modules* and are simply called *R-modules*.

Within a module, it is common to refer to the carrier set of the ring as the *scalars*, and the $\cdot$ operator as *scalar multiplication*. Intuitively, a module allows for the transformation of the elements within the group by scaling it by some value. For instance, within a vector space, the scalar values allow one to scale a vector's length or direction.

We present a structure that is a mathematical module, and will later compare it to the definition of a module we have earlier presented. We first introduce the ring, the structure that will make up the scalar values that can be multiplied with the group. It was established that a Boolean lattice is interchangeable with a Boolean algebra, and as shown in [58, 113], determining a Boolean ring from a Boolean algebra is a straightforward process. To do so, the Boolean ring is formed using the same carrier set and multiplicative operator as the Boolean algebra, which in this case is L and $\otimes$ respectively, and the additive operator is instead *xor*, denoted $\odot$. The *xor* operator is defined in the following way. Let x, y be any two elements in L, then we define *xor* as

$$x \odot y = (x \oplus y) \otimes \neg(x \otimes y) \tag{A.0.1}$$

where $\neg$ is the complement operator, and $\oplus$ is the additive operator used in the Boolean algebra (the lattice join operator). The Boolean ring borrows the multiplicative and complement operator from the Boolean algebra, as well as the constants. Therefore, the Boolean ring is defined as $R = (L, \odot, \otimes, e_{\subset}, \top, \neg)$, where $e_{\subset}$ is the identity of the additive operator *xor*, $\top$ is the identity of the multiplicative operator $\otimes$, and the complement operator is defined the same as the Boolean lattice. Therefore, in this module, the scalars are simply elements from the set L, which compose the Boolean lattice.

The second component necessary for defining a mathematical module is the abelian group M. The elements of the group are what are transformed by the scalars. We introduce the set of all principal ideals of a Boolean lattice, and denote it P. For this module, we let P be the set of all principal ideals of the Boolean lattice $\mathcal{L}$. For any principal ideal $p \in P$, we let $\top_p$ be the element that generates it, i.e., the top. Before we present the definition of the + operator, we introduce two notions: a concept c, and the principal ideal generated by a concept c, denoted by $L_{\downarrow c}$. For two principal ideals, $p, q \in P$, we define the concept c as

$$c = (\top_p \oplus_L \top_q) \otimes_L (\top_p \otimes_L \top_q)'_L \tag{A.0.2}$$

where $\oplus_L$, $\otimes_L$, and $'_L$ are the join, meet, and complement operators from the Boolean lattice $\mathcal{L}$. With the concept c, we define $p + q$ as

$$m + n = L_{\downarrow c}. \tag{A.0.3}$$

Thus, the + operator takes two principal ideals, and returns a principal ideal. This principal ideal is formed from the concept that is the result of*xor*'ing the tops of the two submitted principal ideals. With the set P and the + operator, we form the abelian group $M = (P, +)$. It is known that *xor* is associative, commutative, and is closed under its carrier set [2]. Additionally, the identity element is $e_{_C}$, and any element is self-inversable, i.e., for any $p \in P$, $p + p = e_{_C}$.

The final component necessary for defining the mathematical module is the scalar multiplication operator. With the currently defined Boolean ring as our scalars, and the set of principal ideals forming our group, we introduce an operator which

scales a principal ideal by some element of the Boolean algebra. We present the $\cdot$ operator which takes a principal ideal and element from the Boolean ring, and returns a principal ideal. Let $p \in P$ and $r \in R$, we then define a concept c_2 such that

$$c_2 = \top_m \otimes r. \tag{A.0.4}$$

We then define $\cdot$ as

$$m \cdot r = L_{\downarrow c_2}. \tag{A.0.5}$$

In this, the $\cdot$ operator functions similar to $+$ in that it operates on the top of a principal ideal to produce a new concept that is used to generate a principal ideal. We claim the structure $Z = ((M, +), \cdot)$ is a module.

Claim A.0.1. *Let $\mathcal{O}$ be a domain ontology with Boolean lattice $\mathcal{L}$, and let R be the Boolean ring that corresponds to it. Let $(P, +)$ be the Abelian group formed over the set of all principal ideals of $\mathcal{L}$, denoted P, and the xor over principal ideals operation, denoted $+$. Finally, let $\cdot$ be the scalar multiplication of a principal ideal by an element in R.*

Then the structure $Z = ((P, +), \cdot)$ is an R-module.

Proof. To show that the structure Z is a module, it must satisfy the four properties in Definition A.0.1. Since our operations are defined using the tops of the principal ideals, we can reduce the proof of the properties to the lattice operations that they are built from. For example, let $r, s \in R$ and $x, y \in P$. Then $r \cdot (x + y)$ reduces to $r \otimes (\top_x \odot \top_y)$. In this way, it functions the same as a ring. The left and right distributivity is of course satisfied. Likewise, the property $(rs) \cdot x = r \cdot (s \cdot x)$ can be rewritten as $(r \otimes s) \otimes \top_x = r \otimes (s \otimes \top_x)$. Trivially, the multiplication operator

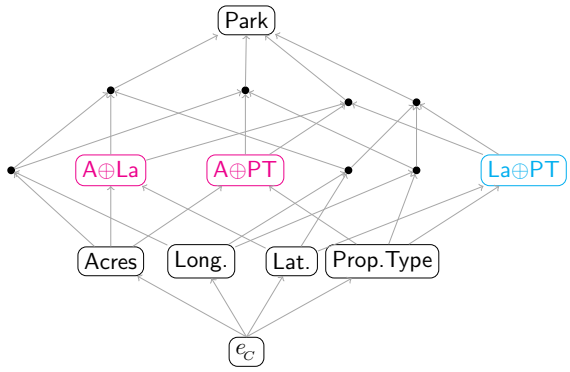
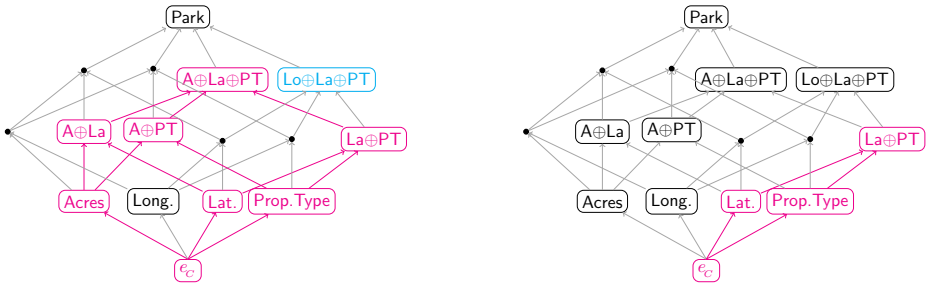


Figure A.1: Depiction of performing the *xor* operator on the two magenta concepts to produce the cyan concept.

is associative by definition. Finally, the property $1_R \cdot x = x$ is demonstrated by recognizing $1_R = \top$ and rewriting the property as $\top \otimes \top_x = \top_x$, which is true by $\top$ being the identity of the $\otimes$ operator. □

The definition of the module Z in this way utilizes two forms of the *xor* operator. The first, $\odot$, belongs to the Boolean ring and operates on two concepts in L. Intuitively, given two concepts c_1 and c_2, $c_1 \odot c_2$ is the query for the concept c_3 that is composed of parts that exist in either c_1 *or* c_2 *but not both*. For example, in Figure A.1, we take the query of $(A \oplus La) \odot (A \oplus PT)$. In this example, *Acres* is a part of both submitted concepts, so it will not be a part of the resulting concept. The concepts *Latitude* and *Property Type* are a part of one of the input concepts but are not shared between both, thus, the resulting concept will be the composition of these two concepts. This is shown as the cyan concept $La \oplus PT$ which is the combination of *Latitude* and *Property Type*.

The second *xor* operator, the + operator, operates on two principal ideals. It determines the principal ideal that is formed from the concept that results from the *xor*'ing of the tops of the two principal ideals.



(a) The principal ideal and the scalar. (b) The resulting principal ideal

Figure A.2: Depiction of scaling the principal ideal formed over $A{\oplus}La{\oplus}PT$ by $Lo{\oplus}La{\oplus}PT$

The final component to the module Z is the scalar multiplication $\cdot$ which takes a principal ideal of $\mathcal{L}$ and a concept from L, and returns a new principal ideal. The concept from L is referred to as the scalar value that multiplies the principal ideal. Figure A.2 is an example of scalar multiplication, where the magenta principal ideal is formed from the concept $A{\oplus}La{\oplus}PT$, and the scalar constant that is to be multiplied with it is $Lo{\oplus}La{\oplus}PT$. As the scalar multiplication is guided by the join of the scalar and the top of the principal ideal, we are determining the principal ideal that is formed from the concept $(A{\oplus}La{\oplus}PT) \otimes (Lo{\oplus}La{\oplus}PT)$, which is $La{\oplus}PT$. Thus, the result is the principal ideal formed over $La{\oplus}PT$.

The abelian group of the module is formed using the set of all principal ideals P, and a type of *xor* operator. For a principal ideal $p \in P$, a subgroup can be formed by letting Q be the set of all principal ideals that are a part of p, written as

$$Q = \{L_{\downarrow c} \mid c \in L \wedge c \sqsubseteq_C \top_p\}. \tag{A.0.6}$$

This equation states that Q is populated by all principal ideals formed from a concept in the Boolean lattice that is a part of the principal ideal p and no others. Thanks

to the requirement that the generating concept c must be in L, and thus, associated with a principal ideal in P, we have $Q \subseteq P$. Thus, $(Q, +)$ is a subgroup of $(P, +)$, and further, $Z_Q = ((Q, +), \cdot)$ is a *R-submodule.*

As Q is a subset of P, the submodule Z_Q is smaller than Z. As the concept that generates Q must be *partOf* $\top_p$, we can order the submodules. For instance, let $Z_1 = ((P_1, +), \cdot)$ and $Z_2 = ((P_2, +), \cdot)$ be two R-submodules of the same R-module. Then

$$Z_1 \leq Z_2 \iff P_1 \subseteq P_2.$$

With this relation, we will always eventually bottom out to the submodule with the group formed over the set $\{e_{_C}\}$ after a finite number of steps. This notion of decreasing size and bottoming out is established in the following claim.

Claim A.0.2. *Let $Z = ((P, +), \cdot)$ be a module. Z is an Artinian module.*

Proof. We prove this by contradiction. Let $Z = ((P, +), \cdot)$ be a R-module, and let Z_1 be a submodule of Z. If Z is not artinian, then there exists a submodule Z_2 such that $Z_2 < Z_1$, i.e., we can always find a smaller submodule. If we let $Z_1 = ((\{e_{_C}\}, +), \cdot))$, then Z_2 must be formed over a subgroup of $(\{e_{_C}\}, +)$ that is not equal to the group itself. Since $e_{_C}$ is the smallest element in the Boolean lattice, it is impossible to acquire a smaller group. Therefore, Z_1 is the smallest submodule, and Z must be Artinian.

For any R-module $Z = ((P, +), \cdot)$, $e_{_C} \sqsubseteq_{_C} \top_p$. Then $Z_t = ((\{e_{_C}\}, +), \cdot)$ is a R-submodule of Z. □

The point to distinguish in this claim is that for any R-module, we will always *eventually* bottom-out to the R-submodule $Z_t = ((\{e_{_C}\}, +), \cdot)$. This is due to the

empty concept being a part of every concept in the Boolean lattice, i.e.,

$$\forall(c \mid c \in L : e_C \sqsubseteq_C c).$$

Thus, the Z_t module is a R-submodule of every module, and is where the descending chain of submodules stabilizes.

The mathematical modules are a different structure than the ontologicals module discussed in this paper. An ontological module is defined as a domain ontology with restrictions such as the Boolean lattice is a Boolean sublattice of the domain ontology it is modularized from. On the other hand, a mathematical module is a structure formed over a set of ontological modules and an *xor* operator. Thus, when referring to a mathematical module, one is not referring to a single ontological module but rather a set of ontological modules. Additionally, the mathematical module describes how one can scale a module with a concept. In conclusion, there is a relationship between the ontological module and the mathematical module, but they are not interchangeable.

Bibliography

[1] Recreation and parks properties. https://data.sfgov.org/Culture-and-Recreation/Recreation-and-Parks-Properties/gtr9-ntp6. Accessed: 2021-07-18.

[2] Accu. All about xor, Jun 2012.

[3] S. S. Ahmed, M. Malki, and S. M. Benslimane. Ontology partitioning: Clustering based approach. *International Journal of Information Technology and Computer Science (IJITCS)*, 2015.

[4] A. Algergawy, S. Babalou, M. J. Kargar, and S. H. Davarpanah. Seecont: A new seeding-based clustering approach for ontology matching. In *East European Conference on Advances in Databases and Information Systems*, pages 245–258. Springer, 2015.

[5] S. S. Anand, D. A. Bell, and J. G. Hughes. The role of domain knowledge in data mining. In *Proceedings of the fourth international conference on Information and knowledge management*, pages 37–43, 1995.

[6] A. Armas Romero, M. Kaminski, B. Cuenca Grau, and I. Horrocks. Module

extraction in expressive ontology languages via datalog reasoning. *Journal of Artificial Intelligence Research*, 55:499–564, 2016.

[7] E. Aroua and A. Mourad. An ontology-based framework for enhancing personalized content and retrieval information. In *Research Challenges in Information Science (RCIS), 2017 11th International Conference on*, pages 276–285. IEEE, 2017.

[8] F. Baader. Terminological cycles in a description logic with existential restrictions. In *IJCAI*, volume 3, pages 325–330, 2003.

[9] F. Baader, D. Borchmann, and A. Nuradiansyah. The identity problem in description logic ontologies and its application to view-based information hiding. In *Joint International Semantic Technology Conference*, pages 102–117. Springer, 2017.

[10] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, D. Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[11] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In *Mechanizing mathematical reasoning*, pages 228–248. Springer, 2005.

[12] S. Babalou, A. Algergawy, and B. König-Ries. An ontology-based scientific data integration workflow. In *29th GI-Workshop Grundlagen von Datenbanken*, pages 30–35, 2017.

[13] S. Benomrane, Z. Sellami, and M. B. Ayed. An ontologist feedback driven ontology evolution with an adaptive multi-agent system. *Advanced Engineering Informatics*, 30(3):337–353, 2016.

[14] L. Bi, X.-q. Di, and Y. Zhang. Ontology modularization method based on the k-pso algorithm. In *Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), International Congress on*, pages 2009–2013. IEEE, 2016.

[15] G. Birkhoff. *Lattice theory*, volume 25. American Mathematical Soc., 1940.

[16] J. Chen, G. Alghamdi, R. A. Schmidt, D. Walther, and Y. Gao. Ontology extraction for large ontologies via modularity and forgetting. In *Proceedings of the 10th International Conference on Knowledge Capture*, pages 45–52, 2019.

[17] D. Cirella and H. Gu. Generating abstraction networks using semantic similarity measure of ontology concepts. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 840–843. IEEE, 2017.

[18] I. F. Cruz, H. Xiao, et al. The role of ontologies in data integration. *Engineering intelligent systems for electrical engineering and communications*, 13(4):245, 2005.

[19] B. Cuenca Grau. Privacy in ontology-based information systems: A pending matter. *Semantic Web*, 1(1, 2):137–141, 2010.

[20] M. d'Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou. Criteria and evaluation for ontology modularization techniques. *Modular ontologies*, pages 67–89, 2009.

[21] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[22] C. Del Vescovo, D. D. Gessler, P. Klinov, B. Parsia, U. Sattler, T. Schneider, and A. Winget. Decomposition and modular structure of bioportal ontologies. In *International Semantic Web Conference*, pages 130–145. Springer, 2011.

[23] C. Del Vescovo, B. Parsia, U. Sattler, and T. Schneider. The modular structure of an ontology: Atomic decomposition and module count. In *WoMO*, pages 25–39, 2011.

[24] C. Del Vescovo and R. Penaloza. Dealing with ontologies using cods. CEUR, 2014.

[25] J. L. Dietz. *What is Enterprise Ontology?* Springer, 2006.

[26] Z. Ding and Y. Peng. A probabilistic extension to ontology language owl. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10–pp. IEEE, 2004.

[27] P. Doran, V. Tamma, and L. Iannone. Ontology module extraction for ontology reuse: an ontology engineering perspective. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 61–70. ACM, 2007.

[28] P. Drogkaris and A. Bourka. Towards a framework for policy development in cybersecurity - security and privacy considerations in autonomous agents. Technical report, European Union Agency For Network and Information Security, March 2019.

[29] J. Du, K. Wang, and Y.-D. Shen. A tractable approach to abox abduction over description logic ontologies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[30] K. Etminani, A. R. Delui, and M. Naghibzadeh. Overlapped ontology partitioning based on semantic similarity measures. In *Telecommunications (IST), 2010 5th International Symposium on*, pages 1013–1018. IEEE, 2010.

[31] W. Fang, L. Ma, P. E. Love, H. Luo, L. Ding, and A. Zhou. Knowledge graph for identifying hazards on construction sites: Integrating computer vision with ontology. *Automation in Construction*, 119:103310, 2020.

[32] G. Figueiredo, A. Duchardt, M. M. Hedblom, and G. Guizzardi. Breaking into pieces: An ontological approach to conceptual model complexity management. In *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2018.

[33] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary abox: Cutting ontologies down to size. In *International Semantic Web Conference*, pages 343–356. Springer, 2006.

[34] F. Fonseca, M. Egenhofer, C. Davis, and G. Câmara. Semantic granularity in ontology-driven geographic information systems. *Annals of mathematics and artificial intelligence*, 36(1-2):121–151, 2002.

[35] A. Freitas, A. R. Panisson, L. Hilgert, F. Meneguzzi, R. Vieira, and R. H. Bordini. Applying ontologies to the development and execution of multi-agent systems. In *Web Intelligence*, volume 15, pages 291–302. IOS Press, 2017.

[36] W. Gatens, B. Konev, and F. Wolter. Lower and upper approximations for depleting modules of description logic ontologies. In *ECAI*, pages 345–350, 2014.

[37] S. Ghafourian, A. Rezaeian, and M. Naghibzadeh. Graph-based partitioning of ontology with semantic similarity. In *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*, pages 80–85. IEEE, 2013.

[38] S. Ghafourian, A. Rezaeian, and M. Naghibzadeh. Modularization of graph-structured ontology with semantic similarity. In *Workshop on Modular Ontologies (WoMO) 2013*, page 25, 2013.

[39] A. Ghazvinian, N. F. Noy, and M. A. Musen. From mappings to modules: using mappings to identify domain-specific modules in large ontologies. In *Proceedings of the sixth international conference on Knowledge capture*, pages 33–40. ACM, 2011.

[40] A. Giovannini, A. Aubry, H. Panetto, M. Dassisti, and H. El Haouzi. Ontology-based system for supporting manufacturing sustainability. *Annual Reviews in Control*, 36(2):309–317, 2012.

[41] J. Golbeck and M. Rothstein. Linking social networks on the web with foaf: A semantic web case study. In *AAAI*, volume 8, pages 1138–1143, 2008.

[42] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: extracting modules from ontologies. In *Proceedings of the 16th international conference on World Wide Web*, pages 717–726. ACM, 2007.

[43] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. A logical framework for modularity of ontologies. In *IJCAI*, volume 2007, pages 298–303, 2007.

[44] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Extracting modules from ontologies: A logic-based approach. In *Modular Ontologies*, pages 159–186. Springer, 2009.

[45] B. C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. In *KR*, pages 198–209, 2006.

[46] D. Gries and F. Schenider. *A Logical Approach to Discrete Math*. Springer Texts And Monographs In Computer Science. Springer-Verlag, New York, 1993.

[47] N. Guarino. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, volume 46. IOS press, 1998.

[48] N. Guarino, M. Carrara, and P. Giaretta. Formalizing ontological commitment. In *AAAI*, volume 94, pages 560–567, 1994.

[49] N. Guarino, D. Oberle, and S. Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

[50] P. R. Halmos. *Lectures on Boolean algebras*. Courier Dover Publications, 2018.

[51] J. Harding, C. Heunen, B. Lindenhovius, and M. Navara. Boolean subalgebras of orthoalgebras. *Order*, pages 1–47, 2017.

[52] M. J. Healy and T. P. Caudell. Ontologies and worlds in category theory: implications for neural systems. *Axiomathes*, 16(1-2):165–214, 2006.

[53] R. Hirsch and I. Hodkinson. *Relation algebras by games*. Elsevier, 2002.

[54] P. Hitzler, M. Krotzsch, and S. Rudolph. *Foundations of semantic web technologies*. CRC press, 2009.

[55] M. Horridge, J. M. Mortensen, B. Parsia, U. Sattler, and M. A. Musen. A study on the atomic decomposition of ontologies. In *International Semantic Web Conference*, pages 65–80. Springer, 2014.

[56] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The instance store: Dl reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.

[57] I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of web semantics*, 1(1):7–26, 2003.

[58] J. Hsiang and G.-S. Huang. Some fundamental properties of boolean ring normal forms. In *Satisfiability Problem: Theory and Applications*, pages 587–602, 1996.

[59] W. Hu, Y. Zhao, and Y. Qu. Partition-based block matching of large class hierarchies. In *Asian Semantic Web Conference*, pages 72–83. Springer, 2006.

[60] U. Hustadt, B. Motik, and U. Sattler. Reducing shiq-description logic to disjunctive datalog programs. *KR*, 4:152–162, 2004.

[61] M. Jarrar and R. Meersman. Ontology engineering–the dogma approach. In *Advances in Web Semantics I*, pages 7–34. Springer, 2008.

[62] M. Kachroudi, S. Zghal, and S. Ben Yahia. Ontopart: at the cross-roads of ontology partitioning and scalable ontology alignment systems. *International Journal of Metadata, Semantics and Ontologies*, 8(3):215–225, 2013.

[63] C. M. Keet. A formal theory of granularity. *Free University of Bozen-Bolzano*, 2008.

[64] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

[65] J. Kohlas and R. F. Stärk. Information algebras and consequence operators. *Logica Universalis*, 1(1):139–165, 2007.

[66] B. Konev, C. Lutz, D. K. Ponomaryov, and F. Wolter. Decomposing description logic ontologies. In *KR*, 2010.

[67] B. Konev, C. Lutz, D. Walther, and F. Wolter. Logical difference and module extraction with cex and mex. In *Description Logics*, 2008.

[68] B. Konev, C. Lutz, D. Walther, and F. Wolter. Formal properties of modularisation. In *Modular Ontologies*, pages 25–66. Springer, 2009.

[69] B. Konev, C. Lutz, D. Walther, and F. Wolter. Model-theoretic inseparability and modularity of description logic ontologies. *Artificial Intelligence*, 203:66–103, 2013.

[70] B. Konev, D. Walther, and F. Wolter. Forgetting and uniform interpolation in large-scale description logic terminologies. In *IJCAI*, pages 830–835, 2009.

[71] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, and M. Zakharyaschev. Minimal module extraction from dl-lite ontologies using qbf solvers. In *IJCAI*, volume 9, pages 836–841, 2009.

[72] R. Kontchakov, F. Wolter, and M. Zakharyaschev. Logic-based ontology comparison and module extraction, with an application to dl-lite. *Artificial Intelligence*, 174(15):1093–1141, 2010.

[73] P. Koopmann and R. A. Schmidt. Count and forget: Uniform interpolation of shq-ontologies. Technical report, Tech. Rep., The University of Manchester, 2014.

[74] J. Lambek. *Lectures on rings and modules*, volume 283. American Mathematical Soc., 2009.

[75] A. LeClair, J. Jaskolka, W. MacCaull, and R. Khedri. Architecture for ontology-supported multi-context reasoning systems. *Data & Knowledge Engineering*, 2021. Review.

[76] A. LeClair and R. Khedri. An algebraic approach to ontology modularization and knowledge refinement. *Journal of Theoretical Computer Science*, 2021. Review.

[77] A. LeClair., R. Khedri., and A. Marinache. Toward measuring knowledge loss due to ontology modularization. In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 2: KEOD*. INSTICC, SciTePress, 2019.

[78] A. LeClair, R. Khedri, and A. Marinache. Formalizing graphical modularization approaches for ontologies and the knowledge loss. In J. Dietz, D. Aveiro, and J. Filipe, editors, *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 1297 of *Communications in Computer and Information Science series*, pages 1–25. Springer, 2021. Invited.

[79] A. LeClair, A. Marinache, H. Ghalayini, R. Khedri, and W. MacCaull. A systematic literature review and discussion on ontology modularization techniques. *IEEE Transactions on Knowledge and Data Engineering*, 2020. Review.

[80] Y. Li, Z. Jianhui, J. Liu, and Y. Hou. Matching large scale ontologies based on filter and verification. *Mathematical Problems in Engineering*, 2020, 2020.

[81] J. Lozano, J. Carbonera, M. Abel, and M. Pimenta. Ontology view extraction: an approach based on ontological meta-properties. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, pages 122–129. IEEE, 2014.

[82] M. Ludwig and B. Konev. Practical uniform interpolation and forgetting for alc tboxes with applications to logical difference. In *Proc. Int. Workshop Tempor. Represent. Reason.*, pages 318–327, 2014.

[83] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *IJCAI*, volume 7, pages 453–458, 2007.

[84] A. Marinache. On the structural link between ontologies and organised data sets. Master's thesis, 2016.

[85] A. Marinache, R. Khedri, A. LeClair, and W. MacCaull. Dis: A data-centred knowledge representation formalism. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pages 1–8. IEEE, 2021.

[86] A. Marinache, R. Khedri, and W. MacCaull. A data-centered framework for domain knowledge representation. Technical Report CAS-19-09-RK, McMaster University, 2019.

[87] D. L. McGuinness, F. Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[88] R. K. Meyer. Conservative extension in relevant implication. *Studia Logica*, 31(1):39–46, 1973.

[89] T. Mittra and M. M. Ali. Parallelized and distributed task based ontology matching in clustering environment with semantic verification. *CSI Transactions on ICT*, 5(3):265–279, 2017.

[90] M. A. Movaghati and A. A. Barforoush. Modular-based measuring semantic quality of ontology. In *Computer and Knowledge Engineering (ICCKE), 2016 6th International Conference on*, pages 13–18. IEEE, 2016.

[91] N. Nikitina, B. Glimm, and S. Rudolph. Wheat and chaff–practically feasible interactive ontology revision. In *International Semantic Web Conference*, pages 487–503. Springer, 2011.

[92] N. Noy and M. Musen. Traversing ontologies to extract views. *Modular Ontologies*, pages 245–260, 2009.

[93] M. Obitko, V. Snasel, J. Smid, and V. Snasel. Ontology design with formal concept analysis. In *CLA*, volume 128, pages 1377–1390, 2004.

[94] P. Ochieng and S. Kyanda. A statistically-based ontology matching tool. *Distributed and Parallel Databases*, 36(1):195–217, 2018.

[95] M. Orlov. Efficient generation of set partitions. *Engineering and Computer Sciences, University of Ulm, Tech. Rep*, 2002.

[96] S. K. Pati, S. Mallick, A. Chakraborty, and A. Das. Informative gene selection using clustering and gene ontology. In *Emerging Technologies in Data Mining and Information Security*, pages 417–427. Springer, 2019.

[97] Z. Pawlak. Granularity of knowledge, indiscernibility and rough sets. In *1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36228)*, volume 1, pages 106–110. IEEE, 1998.

[98] A. Pomp, A. Paulus, A. Kirmse, V. Kraus, and T. Meisen. Applying semantics to reduce the time to analytics within complex heterogeneous infrastructures. *Technologies*, 6(3):86, 2018.

[99] J. Pujara, H. Miao, L. Getoor, and W. W. Cohen. Ontology-aware partitioning for knowledge graph identification. In *Proceedings of the 2013 workshop on Automated knowledge base construction*, pages 19–24. ACM, 2013.

[100] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):1–16, 2016.

[101] A. Rector, J. Rogers, and P. Pole. The galen high level ontology. 1996.

[102] A. A. Salatino, T. Thanapalasingam, A. Mannocci, A. Birukou, F. Osborne, and E. Motta. The computer science ontology: A comprehensive automatically-generated taxonomy of research areas. *Data Intelligence*, 2(3):379–416, 2020.

[103] G. Santipantakis and G. A. Vouros. Modularizing ontologies for the construction of e-shiq distributed knowledge bases. In *Hellenic Conference on Artificial Intelligence*, pages 192–206. Springer, 2014.

[104] G. M. Santipantakis and G. A. Vouros. Modularizing owl ontologies using ehq+ddl shiq. In *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, volume 1, pages 411–418. IEEE, 2012.

[105] S. Sarkar and A. Dong. Characterizing modularity, hierarchy and module interfacing in complex design systems. In *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 375–384. American Society of Mechanical Engineers, 2011.

[106] K. Saruladha, G. Aghila, and B. Sathiya. Neighbour based structural proximity measures for ontology matching systems. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 1079–1085. ACM, 2012.

[107] K. Saruladha, G. Aghila, and B. Sathiya. A partitioning algorithm for large scale ontologies. In *Recent Trends In Information Technology (ICRTIT), 2012 International Conference on*, pages 526–530. IEEE, 2012.

[108] B. Sathiya, T. Geetha, and K. Saruladha. Psom 2—partitioning-based scalable ontology matching using mapreduce. *Sādhanā*, 42(12):2009–2024, 2017.

[109] M. H. Seddiqui and M. Aono. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *Journal of web semantics*, 7(4):344–356, 2009.

[110] J. Seidenberg and A. Rector. Web ontology segmentation: analysis, classification and use. In *Proceedings of the 15th international conference on World Wide Web*, pages 13–22, 2006.

[111] J. Sen, A. R. Mittal, D. Saha, and K. Sankaranarayanan. Functional partitioning of ontologies for natural language query completion in question answering systems. In *IJCAI*, pages 4331–4337, 2018.

[112] R. Sikorski, R. Sikorski, R. Sikorski, P. Mathématicien, R. Sikorski, and P. Mathematician. *Boolean algebras*, volume 2. Springer, 1969.

[113] M. H. Stone. The representation of boolean algebras. *Bulletin of the American Mathematical Society*, 44(12):807–816, 1938.

[114] U. Straccia. Towards a fuzzy description logic for the semantic web (preliminary report). In *European Semantic Web Conference*, pages 167–181. Springer, 2005.

[115] T. Strang, C. Linnhoff-Popien, and K. Frank. Cool: A context ontology language to enable contextual interoperability. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 236–247. Springer, 2003.

[116] H. Stuckenschmidt and M. Klein. Structure-based partitioning of large concept hierarchies. In *International semantic web conference*, volume 3298, pages 289–303. Springer, 2004.

[117] H. Stuckenschmidt and A. Schlicht. Structure-based partitioning of large ontologies. In *Modular Ontologies*, pages 187–210. Springer, 2009.

[118] B. Suntisrivaraporn, G. Qi, Q. Ji, and P. Haase. A modularization-based approach to finding all justifications for owl dl entailments. In *Asian Semantic Web Conference*, pages 1–15. Springer, 2008.

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

[120] A. Tiwari and A. Kumar. Comparative analysis of optimized algorithms for ontology clustering. In *2018 5th IEEE Uttar Pradesh Section International Conference on Electric al, Electronics and Computer Engineering (UPCON)*, pages 1–7. IEEE, 2018.

[121] D.-T. Tran, D.-H. Ngo, and P.-T. Do. An information content based partitioning method for the anatomical ontology matching task. In *Proceedings of the Third Symposium on Information and Communication Technology*, pages 272–281. ACM, 2012.

[122] D. Tsarkov. Improved algorithms for module extraction and atomic decomposition. In *25th International Workshop on Description Logics*, page 345. Citeseer, 2012.

[123] P. van Damme, M. Quesada-Martínez, R. Cornet, and J. T. Fernández-Breis. From lexical regularities to axiomatic patterns for the quality assurance of

biomedical terminologies and ontologies. *Journal of biomedical informatics*, 84:59–74, 2018.

[124] B. L. Van der Waerden, E. Artin, and E. Noether. *Moderne algebra*, volume 31950. Springer, 1950.

[125] P. Vassiliadis, A. V. Zarras, and I. Skoulis. How is life for a table in an evolving relational schema? birth, death and everything in between. In *International Conference on Conceptual Modeling*, pages 453–466. Springer, 2015.

[126] S. Wandelt and R. Möller. Islands and query answering for alchi-ontologies. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*, pages 224–236. Springer, 2009.

[127] S. Wandelt and R. Möller. Towards abox modularization of semi-expressive description logics. *Applied Ontology*, 7(2):133–167, 2012.

[128] Z. Wang, K. Wang, R. Topor, and J. Z. Pan. Forgetting for knowledge bases in dl-lite. *Annals of Mathematics and Artificial Intelligence*, 58(1-2):117–151, 2010.

[129] P. Wennerberg, K. Schulz, and P. Buitelaar. Ontology modularization to improve semantic medical image annotation. *Journal of biomedical informatics*, 44(1):155–162, 2011.

[130] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyaschev. Ontology-based data access: A survey. IJCAI Organization, 2018.

[131] X. Xu, Y. Wu, J. Chen, and J. Shen. Sub-ontology mapping based web services discovery framework. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 3, pages V3–363. IEEE, 2010.

[132] J. Xua, P. Shironoshitaa, U. Visserb, N. Johna, and M. Kabukaa. Module extraction for efficient object query over ontologies with large aboxes. 2014.

[133] X. Xue, J. Lu, and J. Chen. Using nsga-iii for optimising biomedical ontology alignment. *CAAI Transactions on Intelligence Technology*, 4(3):135–141, 2019.

[134] X. Xue and J.-S. Pan. A segment-based approach for large-scale ontology matching. *Knowledge and Information Systems*, 52(2):467–484, 2017.

[135] X. Xue and A. Ren. A large scale multi-objective ontology matching framework. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 250–255. Springer, 2017.

[136] X. Xue and Z. Tang. An evolutionary algorithm based ontology matching system. *Journal of Information Hiding and Multimedia Signal Processing*, 8(3):551 – 556, 2017. High heterogeneity;Matcher combination;Matching system;Ontology matching;Optimal model;Recall and precision;Semantic correspondence;State of the art;.

[137] L. Zhang and Z. Wang. Ontology-based clustering algorithm with feature weights. *Journal of Computational Information Systems*, 6(9):2959–2966, 2010.